DesignWare DW8051 MacroCell Solution Databook

Version 3.2a, June 2000

Document Order Number: 26613-000 IB
DesignWare DW8051 MacroCell Solution Databook, v3.2a

Contents

Figures

Tables

Examples

About This Manual

The DW8051 MacroCell is a synthesizable, technology-independent microcontroller core that is object code compatible with the industry standard 8051 microcontroller. The DW8051 MacroCell Solution includes the DW8051 MacroCell, an automated test suite, and an example DW8051 design. The DW8051 MacroCell Solution is engineered for use with Synopsys coreConsultant. coreConsultant, in turn, provides automatic installation, configuration, test suite operation, and synthesis for the DW8051 MacroCell.

There are two versions of the DW8051 MacroCell Solution available:

- The DW8051 version includes encrypted source code for the DW8051 MacroCell (DW8051_core)
- The DW8051-Source version includes unencrypted source code for the DW8051 MacroCell (DW8051_core)

Encryption of the source code for the DW8051 MacroCell is the only difference between the DW8051 and DW8051-Source versions of the DW8051 MacroCell Solution. In both versions, the verification environment and design example are unencrypted. Both versions are available in VHDL and Verilog.

This databook provides:

- DW8051 MacroCell reference data
- Procedures to install, configure, verify, and synthesize the DW8051 MacroCell
- Information needed to integrate the DW8051 MacroCell into your design

This preface includes the following sections:

- Audience
- Manual Contents
- Related Publications
- Other DesignWare Products
- SOLV-IT! Online Help
- Customer Support
- Ordering Publications
- Conventions

Audience

This manual is intended for designers who are using, or are considering using, the DW8051 MacroCell with Synopsys synthesis tools.

Manual Contents

This databook is organized as follows:

Chapter 1: Introduction

Introduces the DW8051 MacroCell Solution, lists product features, performance characteristics, 803x/805x compatibility, and supported HDL synthesis and simulation tools.

Chapter 2: DW8051 Architectural Overview

Describes the DW8051 hardware architecture, memory organization, instruction set, and instruction timing.

Chapter 3: DW8051 Hardware Description

Describes the operation of the DW8051 internal hardware modules.

Chapter 4: DW8051 User Guide

Describes how to install, configure, simulate, and synthesize the DW8051 MacroCell using coreConsultant, and provides information you need to integrate the DW8051 into your application design.

Chapter 5: DW8051 Test Suite

Describes the architecture of the DW8051 test suite and provides procedures to modify and operate the test suite directly instead of through coreConsultant.

Appendix A: Opcode Tests

Provides lists of DW8051 test suite opcode tests, sorted by instruction type, test number, and opcode.

Appendix B: DW8051/DW80C320 Differences

Describes the differences between the DW8051 MacroCell and the Dallas Semiconductor DS80C320.

Related Publications

For additional information about the DesignWare Foundation Library, see Synopsys Online Documentation (SOLD), which is included with the software, or Documentation on the Web, which is available through SolvNET on the Synopsys Web page at

```
http://www.synopsys.com
```

These Synopsys manuals supply additional information about DesignWare Components:

- *DesignWare Foundation Library Databook, Volumes 1-3*
- *DesignWare User Guide*
- *DesignWare Developer Guide*
- *DesignWare DWPCI MacroCell Solution Databook*

These Synopsys manuals supply additional information about Synopsys synthesis, simulation, and design reuse tools:

- *coreConsultant User Guide*
- *Design Compiler Reference Manual*
- *Design Compiler Command-Line Interface Guide*
- *Design Compiler User Guide*
- *VHDL Compiler Reference Manual*

- *HDL Compiler for Verilog Reference Manual*
- *Test Compiler User Guide*
- *Test Compiler Reference Manual*

For additional information about DesignWare products, or to subscribe to the DesignWare Technical Bulletin, send e-mail to

```
designware@synopsys.com
```

or visit the DesignWare Web site at

```
http://www.synopsys.com/designware
```

Other DesignWare Products

The DesignWare family of products includes pre-verified, reusable designs and software tools that enable you to reuse existing designs. In addition to the DW8051 MacroCell Solution, Synopsys offers the following DesignWare products:

- DesignWare Foundation Library – A library of pre-designed, pre-verified components that are tightly integrated with Synopsys synthesis tools. The Foundation Library contains over 100 components grouped into the following categories: Math, Logic, Memory, and Application-Specific.
- DesignWare DWPCI MacroCell Solution – A complete solution for PCI design. The DWPCI MacroCell Solution is available in both VHDL and Verilog.
- DesignWare Developer – A software tool that enables you to encapsulate your design data so that you can you can reuse your existing designs that same way that you use DesignWare components.

SOLV-IT! Online Help

SOLV-IT! is the Synopsys electronic knowledge base. It contains information about Synopsys and its tools and is updated daily. For more information about SOLV-IT!, send e-mail to

`solvitfb@synopsys.com`

or go to the Synopsys Web page at

`http://www.synopsys.com`

and click SolvNET.

Customer Support

If you have problems, questions, or suggestions regarding the DW8051 or other DesignWare MacroCells, please call the DesignWare MacroCell support hotline at 1.877.4.BEST.IP (1.877.423.7847).

If you have problems, questions, or suggestions regarding the DesignWare Foundation Library or other Synopsys products, contact the Synopsys Technical Support Center in one of the following ways:

- Send e-mail to

  support_center@synopsys.com

- Call (650) 584-4200 outside the continental United States or call (800) 245-8005 inside the continental United States, from 7 AM to 5:30 PM Pacific Standard Time, Monday through Friday.

Ordering Publications

To order additional databooks or other Synopsys publications, please contact your local Synopsys sales office or send e-mail to: `designware@synopsys.com`.

Conventions

The following conventions are used in Synopsys documentation.

Table 1 Text Conventions

Convention	**Description**
`Courier`	Indicates command syntax. In command syntax and examples, shows system prompts, text from files, error messages, and reports printed by the system.
italic	Indicates a user specification, such as *`object_name`*
bold	In interactive dialogs, indicates user input (text you type).
`[ ]`	Denotes optional parameters, such as *`pin1`* `[`*`pin2 ... pinN`*`]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`

Table 1 Text Conventions

Convention	Description
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

1

Introduction

The DW8051 MacroCell is a technology-independent, synthesizable, microcontroller core that is instruction set compatible with the industry standard 8051 and configurable to match any of the common 803x/805x variants. This chapter provides an overview of the DW8051 MacroCell Solution. The topics are:

- DW8051 MacroCell Solution
- DW8051 MacroCell Features
- DW8051 MacroCell Performance Overview
- Application Software Compatibility
- Configurable Features
- Synopsys coreConsultant
- Synthesis Tool Compatibility

- Simulation Tool Compatibility
- DW8051/DW8051-Source Version Differences
- Required Resources

DW8051 MacroCell Solution

To provide a complete solution for implementing an embedded 8051-compatible microcontroller, the DW8051 MacroCell Solution includes:

- DW8051 MacroCell – A silicon-proven, technology-independent, configurable core that is object code compatible with the standard 8051 family of microprocessors. The DW8051 is available in either encrypted source code format (DW8051) or unencrypted VHDL or Verilog source code format (DW8051-Source). If you are interested in purchasing the source code version (DW8051-Source), call 1.877.4.BEST.IP (1.877.423.7847).
- Synopsys coreConsultant for automatic installation, configuration, simulation, and synthesis of the DW8051 MacroCell
- Extensive test suite that includes:
  - HDL testbench that instantiates DW8051_core, plus models for internal/external ROM and RAM and other external peripherals
  - Processes that trace the program counter and write accesses to external RAM
  - A collection of 8051 assembler programs that test all of the instruction set opcodes, plus miscellaneous tests for internal hardware.

- Simulation reference files
- Automatic test suite configuration and operation through the coreConsultant graphical user interface (GUI).

• An example 8032-compatible microcontroller design that uses DW8051_core. This example design also illustrates how to build and connect 8051-compatible port modules for designs where it is preferable to use standard 8051 port modules instead of the DW8051 enhanced memory interface.

  The example 8032 design is non-synthesizable, and is included for illustrative purposes only.

• 0.35-micron Cell Based Array (CBA) technology library to be used as an example library for synthesis

• A copy of this databook in Portable Document Format (PDF)

DW8051 MacroCell Features

The DW8051 MacroCell provides the following design features and enhancements to the standard 8051 microcontroller:

• Compatible with industry standard 803x/805x:
  - Standard 8051 instruction set
  - Optional full-duplex serial ports selectable through parameter
  - Optional third timer selectable through parameter
  - Control signals for standard 803x/805x I/O ports

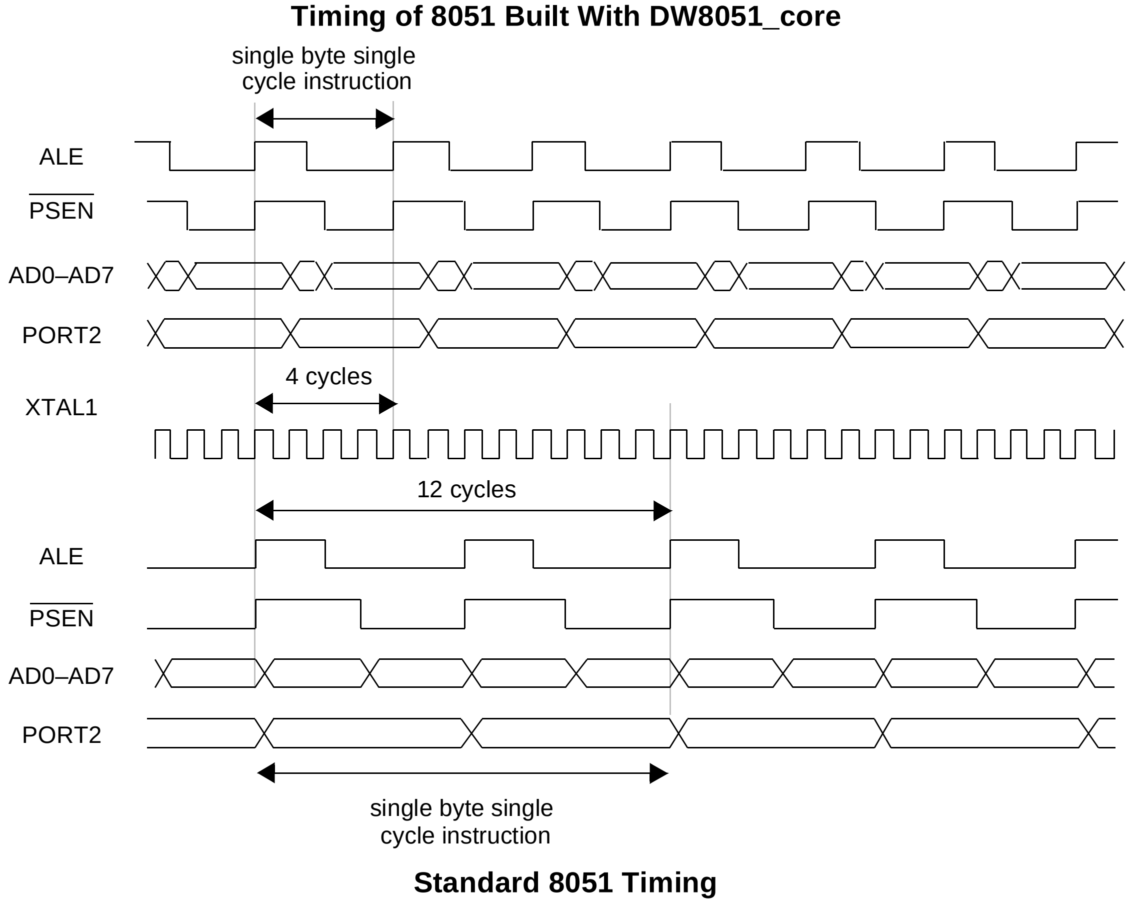
• High speed architecture:
  - 4 clocks/instruction cycle

  - 2.5X average improvement in instruction execution time over the standard 8051
  - Runs DC to greater than 120-MHz clock (Synopsys recommends a target technology of 0.25-micron or less for operation at frequencies greater than 100 MHz)
  - Wasted bus cycles eliminated
  - Dual data pointers
- Parameterizable internal RAM address range
- Parameterizable internal ROM address range
- Simple integration of user-defined peripherals through external Special Function Register (SFR) interface
- Enhanced memory interface with 16-bit address bus
- Variable length `MOVX` to access fast/slow RAM peripherals
- Fully static synchronous design
- Supports industry standard compilers, assemblers, emulators, and ROM monitors
- Supports FPGA Compiler II

DW8051 MacroCell Performance Overview

The DW8051 processor core provides increased performance by executing instructions in a 4-clock bus cycle, as opposed to the 12-clock bus cycle in the standard 8051 (see Figure 1-1). The shortened bus timing improves the instruction execution rate for most instructions by a factor of three over the standard 8051 architectures.

Figure 1-1   Comparative Timing of DW8051 and Industry Standard 8051


Timing of 8051 Built With DW8051_core
single byte single cycle instruction
ALE
PSEN
AD0–AD7
PORT2
4 cycles
XTAL1
12 cycles
ALE
PSEN
AD0–AD7
PORT2
single byte single cycle instruction
Standard 8051 Timing

The average speed improvement for the entire instruction set is approximately 2.5X, calculated as follows:

Number of Opcodes	**Speed Improvement**
150	3.0X
51	1.5X
43	2.0X
2	2.4X
Total: 255	Average: 2.5X
Note: Comparison is for DW8051 and standard 8051 running at the same clock frequency.	

There is not an exact 3X improvement in speed because some instructions require a different number of instruction cycles on the DW8051 than they do on the standard 8051. In the standard 8051, all instructions except for `MUL` and `DIV` take one or two instruction cycles to complete. In the DW8051 architecture, instructions can take between one and five instruction cycles to complete. However, because of the 3X faster instruction cycle time, the average speed improvement for all instructions is 2.5X.

Application Software Compatibility

The DW8051 is object code compatible with the industry standard 8051 microcontroller. That is, object code compiled with an industry standard 8051 compiler or assembler will execute on the DW8051 and will be functionally equivalent. However, because the DW8051 uses a different instruction timing than the standard 8051, code with timing loops may require modification.

See “Instruction Set” on page 2-16 for a list of the number of instruction cycles required to perform each instruction on the DW8051. The DW8051 instruction cycle timing and number of instruction cycles required for each instruction are compatible with the Dallas Semiconductor DS80C320.

Configurable Features

You can easily configure the DW8051 hardware to be functionally compatible with a variety of 803x/805x configurations. For example, you can implement one serial port for compatibility with the Intel 8051, or you can implement two serial ports for compatibility with the Dallas Semiconductor DS80C320.

The configurable features of the DW8051 are:

- Standard (6-source) or extended (13-source) interrupt unit
- Interface to either 128 or 256 bytes of internal RAM
- Interface for up to 64 KB of internal ROM
- Two, one, or zero serial ports
- Optional third timer (Timer 2)

Table 1-1 provides a feature-by-feature comparison of the DW8051 MacroCell and several common 803x/805x configurations. Similar hardware feature comparisons appear in the detailed hardware descriptions throughout this databook.

The DW8051 MacroCell is similar to the DS80C320 in terms of hardware features and instruction cycle timing. However, there are some important implementation differences between the DW8051 and the DS80C320, as detailed in Appendix B.

Table 1-1 Feature Summary of DW8051 and Common 803x/805x Configurations

Feature	**Intel 8031**	**Intel 8031**	**Intel 80C32**	**Intel 80C52**	**Dallas DS80C320**	**DW8051**
Clocks per instruction cycle	12	12	12	12	4	4
Internal ROM(1)	–	4 KB	–	8 KB	–	up to 64 KB
Internal RAM(1)	128 bytes	128 bytes	256 bytes	256 bytes	256 bytes	128 bytes or 256 bytes
Data Pointers	1	1	1	1	2	2
Serial Ports	1	1	1	1	2	0, 1, or 2
16-bit Timers	2	2	3	3	3	2 or 3
Interrupt sources (total of internal and external)	5	5	6	6	13	6 or 13
Stretch memory cycles	no	no	no	no	yes	yes
Internal watchdog timer	no	no	no	no	yes	no
Internal power fail detection	no	no	no	no	yes	no
Timed access protection	no	no	no	no	yes	no
(1) Internal RAM and ROM are external to DW8051_core to simplify simulation and implementation of technology-specific RAM/ROM.						

Synopsys coreConsultant

The DW8051 MacroCell Solution requires the Synopsys coreConsultant tool, which Synopsys provides with the DW8051 MacroCell Solution. coreConsultant is a software tool that greatly enhances the ease-of-use of DesignWare MacroCells and other cores that have been developed and packaged for use with coreConsultant.

Packaging a core for use with coreConsultant captures the core developer’s knowledge of the design and its expected target environment. coreConsultant, in turn, uses that knowledge to provide automatic configuration and high-quality synthesis of the core.

coreConsultant provides the following services to enable you to rapidly generate a highly-optimized gate-level netlist for the DW8051 MacroCell in your selected target technology:

- Installation – coreConsultant automatically installs the DW8051 MacroCell Solution into your selected target directory and verifies that the installation was successful.
- User interface – coreConsultant is your user interface to the DW8051 MacroCell. The coreConsultant GUI guides you through the DW8051 MacroCell design flow activities in the required order. You can also execute the design flow through a batch mode interface.
- MacroCell configuration – You select your DW8051 MacroCell configuration options through the coreConsultant GUI and coreConsultant automatically writes out customized VHDL or Verilog source code for your selected configuration.
- MacroCell simulation – coreConsultant is your interface to the DW8051 MacroCell test suite for verification of the DW8051 as a standalone block. You select which test programs you want to execute and coreConsultant automatically invokes your selected VHDL or Verilog simulation tool to run the simulation. When the simulation is complete, coreConsultant reports the test results.

- Automatic, high-quality synthesis – coreConsultant prompts you for context-specific synthesis specifications, allows you to select a synthesis strategy, then drives Design Compiler to synthesize a highly optimized gate-level version of your custom DW8051 configuration.
- Synthesis results analysis – coreConsultant invokes a series of Design Compiler design checks and analyzes the extracted synthesis reports, then presents the data to you in HTML format through your selected web browser. The analyzed synthesis results include summary reports with links to progressively more detailed data.

For more information about coreConsultant features and user procedures, refer to the *coreConsultant User Guide*.

Synthesis Tool Compatibility

The DW8051 MacroCell Solution is compatible with Synopsys synthesis tools.

Simulation Tool Compatibility

The VHDL version of the DW8051 MacroCell Solution is compatible with the following VHDL simulators:

- Synopsys VSS
- Synopsys Scirocco
- Cadence Leapfrog
- MTI ModelSim

The Verilog version of the DW8051 MacroCell Solution is compatible with the following Verilog simulators:

- Synopsys VCS
- Cadence Verilog-XL
- Cadence NC-Verilog
- MTI ModelSim

DW8051/DW8051-Source Version Differences

There are two versions of the DW8051 MacroCell Solution available:

- The DW8051 version includes encrypted source code for the DW8051 MacroCell (DW8051_core). All other files included with the DW8051 MacroCell Solution, such as testbench and design example files, are unencrypted and identical to the files included in the DW8051-Source version.
- The DW8051-Source version includes unencrypted source code for the DW8051 MacroCell.

The DW8051 and DW8051-Source versions of the DW8051 MacroCell Solution are identical except that the DW8051 version contains encrypted source code for the DW8051 MacroCell.

The DW8051-Source (unencrypted) version simulates faster than the DW8051 (encrypted) version because the RTL code can be directly simulated by any of the supported VHDL or Verilog simulators. For the DW8051 (encrypted) version, you must generate a GTECH simulation model of the DW8051, which requires longer simulation runtimes than simulating the RTL source code directly.

Required Resources

The DW8051 MacroCell Solution requires the following resources:

- SPARC compatible workstation configured as follows:
  - Operating system – SparcOS5 (Solaris)
  - 50 MB available disk space for installation (80 MB for compile)
  - 110 MB available swap space
  - 50 MB physical memory
  - CD-ROM drive

  Note:

  SparcOS4 with OpenWindows 3.0 also requires Sun OpenWindows Patches #100444-74, #100492-12.
- Licensed Synopsys software: Design Compiler, (V)HDL Compiler, coreConsultant
- Synopsys IP licenses (see Table 1-2)
- Optional Synopsys license: Test Compiler (for scan insertion synthesis), PrimeTime (for timing model extraction)
- A licensed simulation tool:

  Supported VHDL simulators include: Synopsys VSS, Cadence LeapFrog, and MTI ModelSim

  Supported Verilog simulators include: Synopsys VCS, Cadence Verilog-XL, and MTI ModelSim

- Technology library files from your ASIC manufacturer. (Synopsys recommends a 0.25-micron technology library for operation at frequencies higher than 100 MHz.)
- For the Verilog version of the DW8051 MacroCell Solution, you also need the Perl utility installed on your system. The Perl utility is available at *www.perl.com*.

Table 1-2 License Features Required for DW8051

Purpose	**Encrypted Package**	**Source Package**
Unpacking the MacroCell	DesignWare-Foundation DW-IP-Consultant	DesignWare-8051-Source DW-IP-Consultant
Writing Out the RTL Source	N/A	DesignWare-8051-Source DW-IP-Consultant
Simulation	DesignWare-Foundation DW-IP-Consultant Simulator License	DesignWare-8051-Source DW-IP-Consultant Simulator License
Synthesis	DesignWare-Foundation DW-IP-Consultant DC-Expert	DesignWare-8051-Source DW-IP-Consultant DC-Expert

2

DW8051 Architectural Overview

This chapter provides a technical overview and description of the DW8051 MacroCell architecture. The topics are:

- Input/Output Signals
- User-Modifiable Parameters
- DW8051 Architecture

Input/Output Signals

Figure 2-1 illustrates the DW8051 interface signals. Table 2-1 describes the function of each signal.

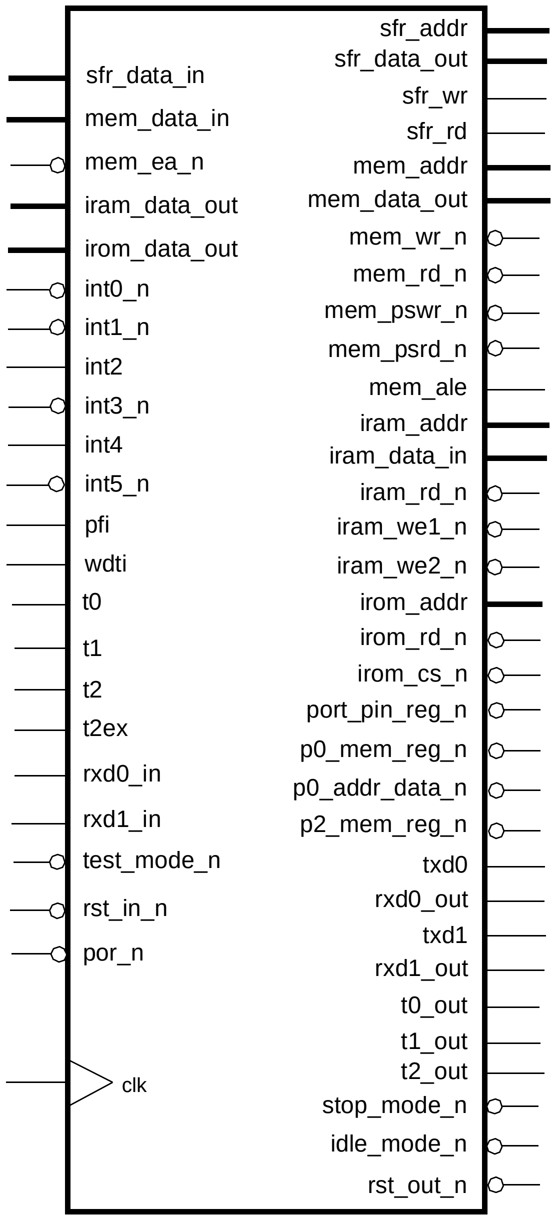


Figure 2-1 DW8051 Input/Output Signals

Table 2-1 Signal Descriptions

Pin Name	Size	Type	Function
clk	1	Input	Main system clock. All internal registers, except one register used to generate the mem_ale output, are positive edge triggered.
por_n	1	Input	Power-on reset, active low. This signal is mandatory for initialization and must be active for at least 2 clock cycles. The rising edge of this input is synchronized internally to the rising edge of clk.
rst_in_n	1	Input	Standard 8051 reset input, active low, synchronized internally to the end of the next bus cycle; must be active for at least 8 clock cycles.
rst_out_n	1	Output	Reset output, active low. Logical AND of por_n and internally synchronized rst_in_n. This signal is used internally to reset all modules and may be used to reset externally connected hardware.
stop_mode_n	1	Output	Indicates that the DW8051 core has entered stop mode, active low. The only way to exit stop mode is to apply reset.
idle_mode_n	1	Output	Indicates that the DW8051 core has entered idle mode, active low. The DW8051 exits idle mode on reset or when an enabled interrupt occurs.
test_mode_n	1	Input	Test mode input for scan test, active low. Must be kept high during normal operation.
sfr_addr	8	Output	Address bus for external peripherals/ports.
sfr_data_out	8	Output	Output data to internal and external SFR registers. Data is valid only when sfr_wr is asserted.
sfr_data_in	8	Input	Input data from external SFR peripherals, sampled on next rising edge of clk after sfr_rd is asserted.
sfr_wr	1	Output	Load signal for external SFR register/ports, active for one clock cycle.

Table 2-1 Signal Descriptions

Pin Name	Size	Type	Function
sfr_rd	1	Output	Read signal for external SFR register/ports, active for one clock cycle.
mem_addr	16	Output	Address lines to external ROM and RAM. Valid half a clock cycle before the falling edge of mem_ale until half a clock cycle after the falling edge of mem_ale.
mem_data_out	8	Output	Output data to external RAM. Valid one clock cycle before rising edge of mem_wr_n/mem_pswr_n until one clock cycle after.
mem_data_in	8	Input	Muxed input data from external ROM/RAM. Sampled on rising edge of mem_rd_n/mem_psrd_n.
mem_wr_n	1	Output	Write strobe for external RAM. Data on mem_data_out should be latched on the rising edge of mem_wr_n.
mem_rd_n	1	Output	Read enable for external RAM. Data provided on mem_data_in are sampled on the rising edge of mem_rd_n.
mem_pswr_n	1	Output	Write strobe for external ROM. This enables users to download programs to external ROM space. Data on mem_data_out should be latched on the rising edge of mem_pswr_n. Data can be transferred to external ROM by MOVX instructions, the same way that data is transferred to external RAM. mem_pswr_n is activated instead of mem_wr_n when the WRS control bit is set in the SPC_FNC SFR. This is a special feature that is provided to enable reprogramming of flash EPROMs.
mem_psrd_n	1	Output	Read enable for external ROM. Data provided on mem_data_in are sampled on the rising edge of mem_psrd_n.
mem_ale	1	Output	Address latch enable signal. Addresses muxed on an external addr/data bus should be latched on the falling edge of mem_ale.

Table 2-1 Signal Descriptions

Pin Name	**Size**	**Type**	**Function**
mem_ea_n	1	Input	External program memory enable.When mem_ea_n is held high, the DW8051 CPU executes out of internal program memory (if available and unless the program counter exceeds the parameterized internal ROM space). When mem_ea_n is held low, the CPU executes only out of external program memory (through mem bus).
iram_addr	8	Output	Address to internal RAM. Stable for one clk cycle during internal RAM read access and for two clk cycles during internal RAM write accesses.
iram_data_in	8	Output	Data to be written to internal RAM, valid in the second half (second clk cycle) of the write cycle.
iram_data_out	8	Input	Data to be read from internal RAM. Read data must be valid on the rising edge of clk at the end of the C3 cycle.
iram_rd_n	1	Output	Read strobe to internal RAM, active (low) in C2 for all instructions. iram_rd_n is also active in C3 for all instructions that require indirect internal RAM read accesses and all instructions that require direct read access to internal RAM registers R0–R7.
iram_we1_n	1	Output	Write enable 1 to internal RAM, active (low) for two clk cycles (whole write cycle). Indicates that the write address (iram_addr) is valid.
iram_we2_n	1	Output	Write enable 2, active (low) in the second half (second clk cycle) of the write cycle. Indicates that write data (iram_data_in) is valid.
irom_addr	16	Output	Address to internal ROM. Valid at the end of cycle C1.
irom_data_out	8	Input	Data to be read from the internal ROM, latched at the end of C4.
irom_rd_n	1	Output	Read strobe to internal ROM, active (low) from the end of C2 through the end of C4. irom_rd_n is optional and is not needed for address-only driven ROM implementations.

Table 2-1 Signal Descriptions

Pin Name	Size	Type	Function
irom_cs_n	1	Output	Select, active (low) for each read cycle. irom_cs_n is optional and is not needed for address-only driven ROM implementations.
port_pin_reg_n	1	Output	Signal to external standard 8051 port modules to select between read of output register and pin. Pin selected if high, register selected if low.
p0_mem_reg_n	1	Output	Signal to external standard 8051 port 0 module to select between output of address/data and output port register data. Addr/data selected if high, port register selected if low.
p0_addr_data_n	1	Output	Signal to standard 8051 port 0 module to select between output of address and data. Address selected if high, data selected if low.
p2_mem_reg_n	1	Output	Signal to standard 8051 port 2 module to select between output of address/data and output port register data. Address/data selected if high, port register selected if low.
int0_n	1	Input	External interrupt line 0, active low, configurable as edge-sensitive or level-sensitive.
int1_n	1	Input	External interrupt line 1, active low, configurable as edge-sensitive or level-sensitive.
int2	1	Input	External interrupt line 2, edge-sensitive, active high. This pin has no function when the extd_intr parameter is set to 0.
int3_n	1	Input	External interrupt line 3, edge-sensitive, active low. This pin has no function when the extd_intr parameter is set to 0.
int4	1	Input	External interrupt line 4, edge-sensitive, active high. This pin has no function when the extd_intr parameter is set to 0.

Table 2-1 Signal Descriptions

Pin Name	**Size**	**Type**	**Function**
int5_n	1	Input	External interrupt line 5, edge-sensitive, active low. This pin has no function when the extd_intr parameter is set to 0.
pfi	1	Input	Power-fail interrupt input, edge-sensitive, active high. This pin has no function when the extd_intr parameter is set to 0.
wdti	1	Input	Watchdog-timer interrupt input, edge-sensitive, active high. This pin has no function when the extd_intr parameter is set to 0.
t0	1	Input	Timer/Counter 0 external input.
t1	1	Input	Timer/Counter 1 external input.
t2	1	Input	Timer/Counter 2 external input. This pin has no function when the timer2 parameter is set to 0.
t2ex	1	Input	Timer/Counter 2 capture/reload trigger. This pin has no function when the timer2 parameter is set to 0.
t0_out	1	Output	Timer/Counter 0 output, active (high) for 1 clock cycle when timer/counter 0 overflows. If Timer 0 is operated in mode 3 (two separate 8-bit timers/counters), this pin is active when the low byte timer/counter overflows.
t1_out	1	Output	Timer/Counter 1 output, active (high) for 1 clock cycle when timer/counter 1 overflows.
t2_out	1	Output	Timer/Counter 2 output, active (high) for 1 clock cycle when timer/counter 2 overflows. This pin has no function (output is high) when the timer2 parameter is set to 0.
rxd0_in	1	Input	Serial Port 0 input. This pin has no function when the serial parameter is set to 0.
rxd0_out	1	Output	Serial Port 0 data output for mode 0, logic '1' for modes 1, 2, and 3. This pin has no function (output is high) when the serial parameter is set to 0.

Table 2-1 Signal Descriptions

Pin Name	Size	Type	Function
txd0	1	Output	Serial Port 0 clock output for mode 0, data output for modes 1, 2, and 3. This pin has no function (output is high) when the serial parameter is set to 0.
rxd1_in	1	Input	Serial Port 1 input. This pin has no function when the serial parameter is set to 0 or 1.
rxd1_out	1	Output	Serial Port 1 data output for mode 0, logic ’1’ for modes 1, 2, and 3. This pin has no function (output is high) when the serial parameter is set to 0 or 1.
txd1	1	Output	Serial Port 1 clock output for mode 0, data output for modes 1, 2, and 3. This pin has no function (output is high) when the serial parameter is set to 0 or 1.

User-Modifiable Parameters

Table 2-2 lists the DW8051 user-modifiable parameters. coreConsultant automatically sets the parameter values when you specify your configuration of the DW8051.

Table 2-2 User-Modifiable Parameters

Parameter	Function	Legal Range
ram_256	Internal RAM size, determines addressability of internal RAM (0 = 128 bytes, 1 = 256 bytes)	0 or 1
timer2	Timer 2 present (0 = Not present, 1 = Present)	0 or 1
rom_addr_size	Determines how many of the 16 internal ROM address bits (irom_addr) are used (0 = no internal ROM present); unused irom_addr pins are tied to logic 0	0 – 16

Table 2-2 User-Modifiable Parameters

Parameter	**Function**	**Legal Range**
serial	Number of serial ports (0 = No serial port present, 1 = Serial Port 0 present, 2 = Serial Ports 0 and 1 present). If you select serial = 2, you must also select the extended interrupt unit (extd_intr = 1) to receive interrupts from Serial Port 1. If you select serial = 2 and extd_intr = 0, you can still operate Serial Port 1 in polling mode.	0, 1, or 2
extd_intr	Use extended interrupt unit with 13 sources (extd_intr = 1) or standard interrupt unit with 6 sources (extd_intr = 0)	0 or 1

DW8051 Architecture

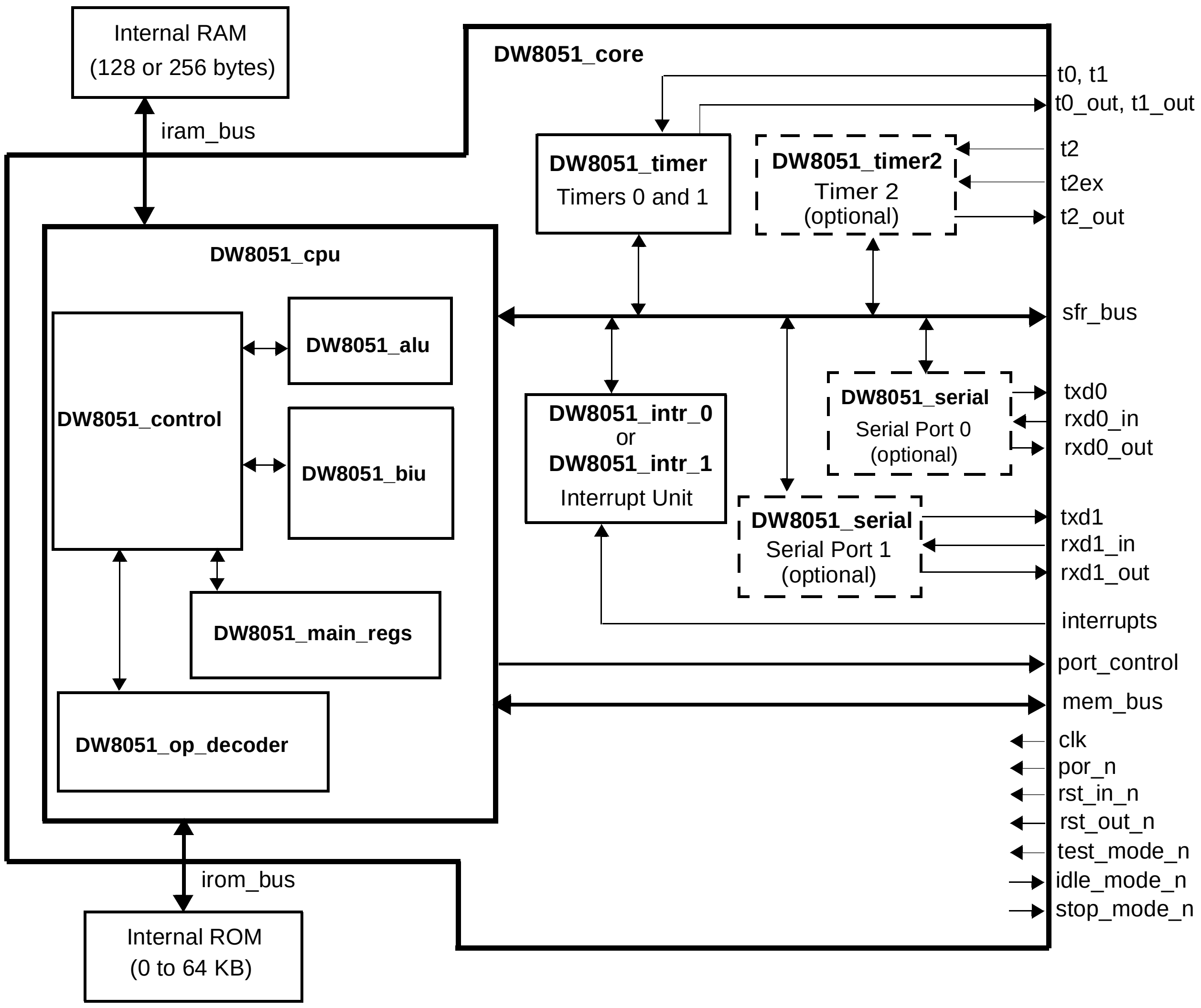
Figure 2-2 illustrates the hardware architecture of the DW8051 core. The name of the top-level module is DW8051_core. The following submodules and interfaces are optional and selectable through parameter settings:

- The optional upper 128-byte internal RAM is accessible only when *ram_256* = 1. The *iram_addr* bus is always 8 bits. However, the DW8051 will not use addresses 80h–FFh when *ram_256* = 0.
- The internal ROM address range (number of *irom_addr* pins used) is determined by the *rom_addr_size* parameter.
- Timer 2 (DW8051_timer2) is present only when *timer2* = 1. (Timers 0 and 1 are always present.)
- Serial Port 0 (DW8051_serial) is present when *serial* = 1 or 2.
- Serial Port 1 (DW8051_serial) is present only when *serial* = 2.

- The Interrupt Unit is either DW8051_intr_0 (6-source) when *extd_intr* = 0 or DW8051_intr_1 (13-source) when *extd_intr* = 1.

DW8051_core provides interfaces to internal RAM and ROM. The actual internal RAM and ROM modules reside outside of DW8051_core and must be implemented by the user. The DW8051 testbench includes simulation models for internal ROM and RAM.

Figure 2-2 DW8051 Block Diagram

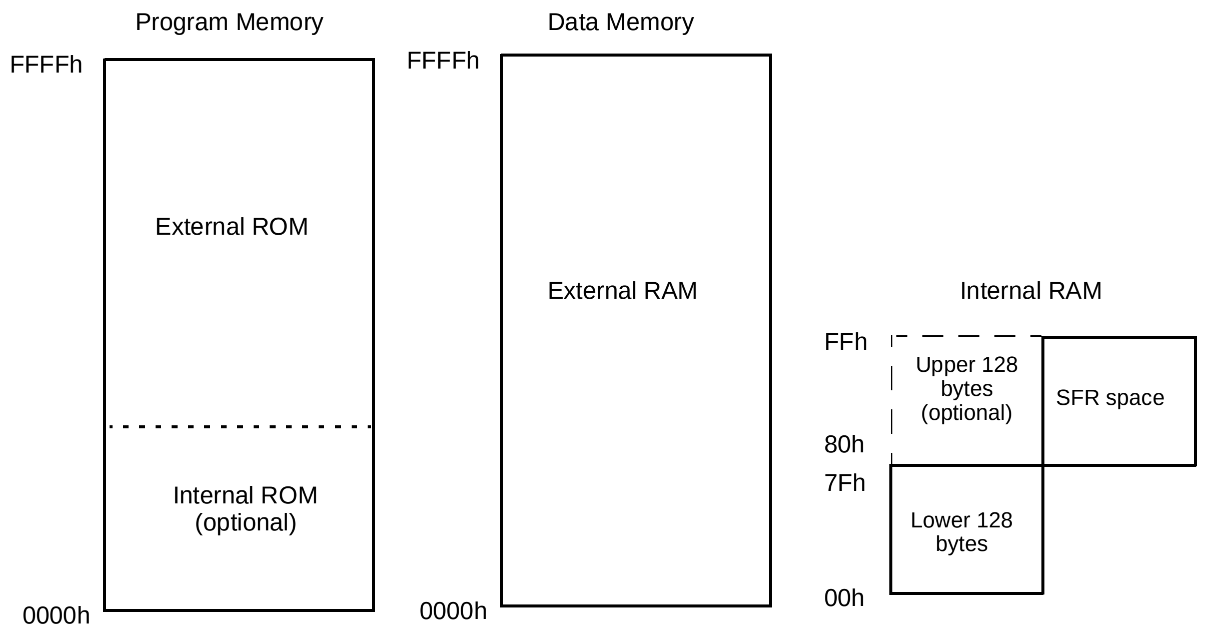


Memory Organization

Memory organization in the DW8051 is similar to that of the industry standard 8051. There are three distinct memory areas: program memory (ROM), data memory (external RAM), and registers (internal RAM).

Figure 2-3 shows the DW8051 memory map. Program memory and data memory can be up to 64 KB each and share the same address range (0000h–FFFFh), but are accessed differently. The internal RAM addresses overlap the lower external RAM addresses, but are accessed through different instruction types.

Figure 2-3 Memory Map



Program Memory

The DW8051 can address up to 64 KB of program memory at addresses 0000h–FFFFh. Program memory can be implemented as internal ROM, external ROM, or a combination of internal and external ROM.

External ROM

External ROM is normally read-only, but can be written to for program downloading by activating the *mem_pswr_n* signal. To activate *mem_pswr_n*, set bit 0 (WRS) in the SPC_FNC register (SFR address 8Fh). The WRS bit switches between the activation of the *mem_wr_n* output (WRS = 0, default after reset) and the *mem_pswr_n* output (WRS = 1) for `MOVX` opcodes.

This feature enables writing to external Flash EPROMs (or ROM replaced by RAM for software development) with normal `MOVX @DPTR` or `MOVX @Ri` opcodes.

The standard 8051 architectures access external ROM through the P0, P2, and PSEN signals. The DW8051 accesses external ROM through the *mem_addr*, *mem_data_in*, and *mem_data_out* buses, and the *mem_psrd_n* and *mem_pswr_n* control signals. For designs in which it is preferable to do so, the DW8051 supports connection of the port modules to provide the standard 8051 functionality.

Internal ROM

The *rom_addr_size* parameter determines address range of the internal ROM. The number of bits of the 16-bit internal ROM address bus (*irom_addr*) that are used is equal to *rom_addr_size*. The remaining *irom_addr* bits are tied to logic 0. If *rom_addr_size* = 0, there is no internal ROM, in which case the entire program memory is readable and writable using the *mem_addr*, *mem_data_in*, and *mem_data_out* buses, and the *mem_psrd_n* and *mem_pswr_n* control signals.

The DW8051 fetches from program memory automatically, beginning at the reset address (0000h). If *mem_ea_n* = 1, the DW8051 executes out of internal ROM until the program address exceeds (2 *rom_addr_size* – 1), then proceeds to execute out of external ROM. If *mem_ea_n* = 0, the DW8051 always executes out of external ROM.

The DW8051 provides only the interface to the internal ROM. The actual implementation of internal ROM is up to the user. In the DW8051 testbench, the internal ROM is modeled as a separate process.

External RAM

The DW8051 can access up to 64 KB of external RAM, at addresses 0000h–FFFFh, using `MOVX` instructions. The DW8051 accesses external RAM through the *mem_addr*, *mem_data_in*, and *mem_data_out* buses, and the *mem_rd_n* and *mem_wr_n* control signals.

Because the DW8051 provides all 16 address bits on *mem_addr*, it is not necessary to multiplex the lower 8 address bytes through an 8-bit port module. For designs that take advantage of this feature and do not use the port modules, the port module control signals (*port_pin_reg_n, p0_mem_reg_n, p0_addr_data_n, p2_mem_reg_n, and mem_ale*) are not needed.

For designs in which it is preferable to use the standard 8051 port modules, the DW8051 supports connection of the port modules as shown in the example design (in the dw8051/example directory).

To replace the function of the Port 2 latch in designs that do not use a Port 2 module, the DW8051 provides an additional special function register, MPAGE, at SFR address 92h. During `MOVX A, @Ri` and `MOVX @Ri, A` instructions, the DW8051 places the contents of the MPAGE register on the upper 8 address bits (*mem_addr[15:8]*). This provides the paging function that is normally provided by the Port 2 latch. The MPAGE register has no function when a Port 2 module is used to connect external RAM.

If the external RAM is connected to the mem_addr outputs, you may need to adapt existing software that was written for the standard memory interface. If the software uses the paging function of the instructions `MOVX A, @Ri` and `MOVX @Ri, A`, change the address of the register holding the memory page from A0h (Port 2) to 92h (MPAGE).

Using SFR address 92h instead of the PORT2 register at SFR address A0h leaves the bit-addressable A0h SFR address available for other operations.

Internal RAM

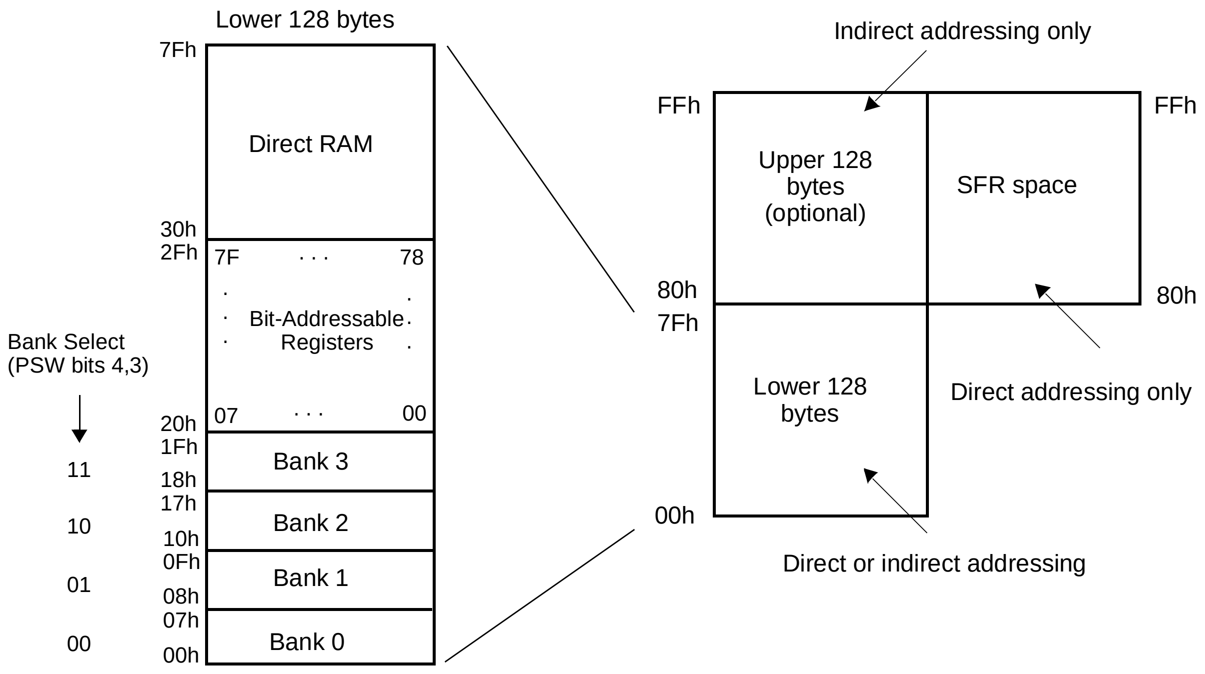
The internal RAM (Figure 2-4) consists of:

- 128 bytes of registers and scratchpad memory accessible through direct or indirect addressing (*iram_addr* addresses 00h–7Fh)
- Optional upper 128 bytes of scratchpad memory accessible through indirect addressing (*iram_addr* addresses 80h–FFh). The optional upper 128 bytes of RAM is addressable only when the parameter *ram_256* = 1.
- 128 special function registers (SFRs) accessible through direct addressing (*sfr_addr* addresses 80h–FFh)

The DW8051 provides only the interface to the internal RAM. The actual implementation of the 128 or 256-byte internal RAM is up to the user. For simulation, the DW8051 testbench includes an internal RAM simulation model. The SFRs (with the exception of user-defined SFR peripherals) are built into DW8051_core.

The lower 128 bytes are organized as shown in Figure 2-4. The lower 32 bytes form four banks of eight registers (R0–R7). Two bits on the program status word (PSW) select which bank is in use. The next 16 bytes form a block of bit-addressable memory space at bit addresses 00h–7Fh. All of the bytes in the lower 128 bytes are accessible through direct or indirect addressing on the iram_bus.

Figure 2-4   Internal RAM Organization



The SFRs and the optional upper 128 bytes of RAM share the same address range (80h–FFh). However, the actual address space is separate and is differentiated by the type of addressing. Direct addressing accesses the SFRs on the sfr_bus, indirect addressing accesses the optional upper 128 bytes of RAM on the iram_bus.

Most SFRs are reserved for specific functions as described in “Special Function Registers” on page 2-27. Unused SFR addresses are available for connecting on-chip peripherals. SFR addresses ending in 0h or 8h are bit-addressable.

Instruction Set

All DW8051 instructions are binary code compatible and perform the same functions that they do in the industry standard 8051. The effects of these instructions on bits, flags, and other status functions is identical to the industry standard 8051. However, the timing of the instructions is different, both in terms of number of clock cycles per instruction cycle and timing within the instruction cycle.

Table 2-4 lists the DW8051 instruction set and the number of instruction cycles required to complete each instruction. Table 2-3 defines the symbols and mnemonics used in Table 2-4.

Table 2-3 Legend for Instruction Set Table

Symbol	Function
A	Accumulator
Rn	Register R0–R7
direct	Internal register address
@Ri	Internal register pointed to by R0 or R1 (except MOVX)
rel	Two’s complement offset byte
bit	Direct bit address
#data	8-bit constant
#data 16	16-bit constant
addr 16	16-bit destination address
addr 11	11-bit destination address

Table 2-4 DW8051 Instruction Set

Mnemonic	Description	Byte	Instr. Cycles	Hex Code
	Arithmetic			
ADD A, Rn	Add register to A	1	1	28–2F
ADD A, direct	Add direct byte to A	2	2	25
ADD A, @Ri	Add data memory to A	1	1	26–27
ADD A, #data	Add immediate to A	2	2	24
ADDC A, Rn	Add register to A with carry	1	1	38–3F
ADDC A, direct	Add direct byte to A with carry	2	2	35
ADDC A, @Ri	Add data memory to A with carry	1	1	36–37

Table 2-4 DW8051 Instruction Set (continued)

Mnemonic	**Description**	**Byte**	**Instr. Cycles**	**Hex Code**
ADDC A, #data	Add immediate to A with carry	2	2	34
SUBB A, Rn	Subtract register from A with borrow	1	1	98–9F
SUBB A, direct	Subtract direct byte from A with borrow	2	2	95
SUBB A, @Ri	Subtract data memory from A with borrow	1	1	96–97
SUBB A, #data	Subtract immediate from A with borrow	2	2	94
INC A	Increment A	1	1	04
INC Rn	Increment register	1	1	08–0F
INC direct	Increment direct byte	2	2	05
INC @Ri	Increment data memory	1	1	06–07
DEC A	Decrement A	1	1	14
DEC Rn	Decrement register	1	1	18–1F
DEC direct	Decrement direct byte	2	2	15
DEC @Ri	Decrement data memory	1	1	16–17
INC DPTR	Increment data pointer	1	3	A3
MUL AB	Multiply A by B	1	5	A4
DIV AB	Divide A by B	1	5	84
DA A	Decimal adjust A	1	1	D4
	Logical			
ANL A, Rn	AND register to A	1	1	58–5F

Table 2-4 DW8051 Instruction Set (continued)

Mnemonic	**Description**	**Byte**	**Instr. Cycles**	**Hex Code**
ANL A, direct	AND direct byte to A	2	2	55
ANL A, @Ri	AND data memory to A	1	1	56–57
ANL A, #data	AND immediate to A	2	2	54
ANL direct, A	AND A to direct byte	2	2	52
ANL direct, #data	AND immediate data to direct byte	3	3	53
ORL A, Rn	OR register to A	1	1	48–4F
ORL A, direct	OR direct byte to A	2	2	45
ORL A, @Ri	OR data memory to A	1	1	46–47
ORL A, #data	OR immediate to A	2	2	44
ORL direct, A	OR A to direct byte	2	2	42
ORL direct, #data	OR immediate data to direct byte	3	3	43
XRL A, Rn	Exclusive-OR register to A	1	1	68–6F
XRL A, direct	Exclusive-OR direct byte to A	2	2	65
XRL A, @Ri	Exclusive-OR data memory to A	1	1	66–67
XRL A, #data	Exclusive-OR immediate to A	2	2	64
XRL direct, A	Exclusive-OR A to direct byte	2	2	62
XRL direct, #data	Exclusive-OR immediate to direct byte	3	3	63
CLR A	Clear A	1	1	E4
CPL A	Complement A	1	1	F4
SWAP A	Swap nibbles of A	1	1	C4
RL A	Rotate A left	1	1	23

Table 2-4 DW8051 Instruction Set (continued)

Mnemonic	**Description**	**Byte**	**Instr. Cycles**	**Hex Code**
RLC A	Rotate A left through carry	1	1	33
RR A	Rotate A right	1	1	03
RRC A	Rotate A right through carry	1	1	13
	Data Transfer			
MOV A, Rn	Move register to A	1	1	E8–EF
MOV A, direct	Move direct byte to A	2	2	E5
MOV A, @Ri	Move data memory to A	1	1	E6–E7
MOV A, #data	Move immediate to A	2	2	74
MOV Rn, A	Move A to register	1	1	F8–FF
MOV Rn, direct	Move direct byte to register	2	2	A8–AF
MOV Rn, #data	Move immediate to register	2	2	78–7F
MOV direct, A	Move A to direct byte	2	2	F5
MOV direct, Rn	Move register to direct byte	2	2	88–8F
MOV direct, direct	Move direct byte to direct byte	3	3	85
MOV direct, @Ri	Move data memory to direct byte	2	2	86–87
MOV direct, #data	Move immediate to direct byte	3	3	75
MOV @Ri, A	MOV A to data memory	1	1	F6–F7
MOV @Ri, direct	Move direct byte to data memory	2	2	A6–A7
MOV @Ri, #data	Move immediate to data memory	2	2	76–77
MOV DPTR, #data	Move immediate to data pointer	3	3	90
MOVC A, @A+DPTR	Move code byte relative DPTR to A	1	3	93

Table 2-4 DW8051 Instruction Set (continued)

Mnemonic	**Description**	**Byte**	**Instr. Cycles**	**Hex Code**
MOVC A, @A+PC	Move code byte relative PC to A	1	3	83
MOVX A, @Ri	Move external data (A8) to A	1	2–9*	E2–E3
MOVX A, @DPTR	Move external data (A16) to A	1	2–9*	E0
MOVX @Ri, A	Move A to external data (A8)	1	2–9*	F2–F3
MOVX @DPTR, A	Move A to external data (A16)	1	2–9*	F0
PUSH direct	Push direct byte onto stack	2	2	C0
POP direct	Pop direct byte from stack	2	2	D0
XCH A, Rn	Exchange A and register	1	1	C8–CF
XCH A, direct	Exchange A and direct byte	2	2	C5
XCH A, @Ri	Exchange A and data memory	1	1	C6–C7
XCHD A, @Ri	Exchange A and data memory nibble	1	1	D6–D7
* Number of cycles is user-selectable. See Stretch Memory Cycles in this chapter. .				
	Boolean			
CLR C	Clear carry	1	1	C3
CLR bit	Clear direct bit	2	2	C2
SETB C	Set carry	1	1	D3
SETB bit	Set direct bit	2	2	D2
CPL C	Complement carry	1	1	B3
CPL bit	Complement direct bit	2	2	B2
ANL C, bit	AND direct bit to carry	2	2	82
ANL C, /bit	AND direct bit inverse to carry	2	2	B0

Table 2-4 DW8051 Instruction Set (continued)

Mnemonic	**Description**	**Byte**	**Instr. Cycles**	**Hex Code**
ORL C, bit	OR direct bit to carry	2	2	72
ORL C, /bit	OR direct bit inverse to carry	2	2	A0
MOV C, bit	Move direct bit to carry	2	2	A2
MOV bit, C	Move carry to direct bit	2	2	92
	Branching			
ACALL addr 11	Absolute call to subroutine	2	3	11–F1
LCALL addr 16	Long call to subroutine	3	4	12
RET	Return from subroutine	1	4	22
RETI	Return from interrupt	1	4	32
AJMP addr 11	Absolute jump unconditional	2	3	01–E1
LJMP addr 16	Long jump unconditional	3	4	02
SJMP rel	Short jump (relative address)	2	3	80
JC rel	Jump on carry = 1	2	3	40
JNC rel	Jump on carry = 0	2	3	50
JB bit, rel	Jump on direct bit = 1	3	4	20
JNB bit, rel	Jump on direct bit = 0	3	4	30
JBC bit, rel	Jump on direct bit = 1 and clear	3	4	10
JMP @A+DPTR	Jump indirect relative DPTR	1	3	73
JZ rel	Jump on accumulator = 0	2	3	60
JNZ rel	Jump on accumulator /= 0	2	3	70
CJNE A, direct, rel	Compare A, direct JNE relative	3	4	B5

Table 2-4 DW8051 Instruction Set (continued)

Mnemonic	**Description**	**Byte**	**Instr. Cycles**	**Hex Code**
CJNE A, #d, rel	Compare A, immediate JNE relative	3	4	B4
CJNE Rn, #d, rel	Compare reg, immediate JNE relative	3	4	B8–BF
CJNE @Ri, #d, rel	Compare ind, immediate JNE relative	3	4	B6–B7
DJNZ Rn, rel	Decrement register, JNZ relative	2	3	D8–DF
DJNZ direct, rel	Decrement direct byte, JNZ relative	3	4	D5
	Miscellaneous			
NOP	No operation	1	1	00
There is an additional reserved opcode (A5) that performs the same function as NOP. All mnemonics are copyright © Intel Corporation 1980.				

Instruction Timing

Instruction cycles in the DW8051 are 4 clock cycles in length, as opposed to the 12 clock cycles per instruction cycle in the standard 8051. This translates to a 3X improvement in execution time for most instructions.

However, some instructions require a different number of instruction cycles on the DW8051 than they do on the standard 8051. In the standard 8051, all instructions except for `MUL` and `DIV` take one or two instruction cycles to complete. In the DW8051 architecture, instructions can take between one and five instruction cycles to complete.

For example, in the standard 8051, the instructions `MOVX A, @DPTR` and `MOV direct, direct` each take 2 instruction cycles (24 clock cycles) to execute. In the DW8051 architecture, `MOVX A, @DPTR` takes two instruction cycles (8 clock cycles) and `MOV direct, direct` takes three instruction cycles (12 clock cycles). Both instructions execute faster on the DW8051 than they do on the standard 8051, but require different numbers of clock cycles.

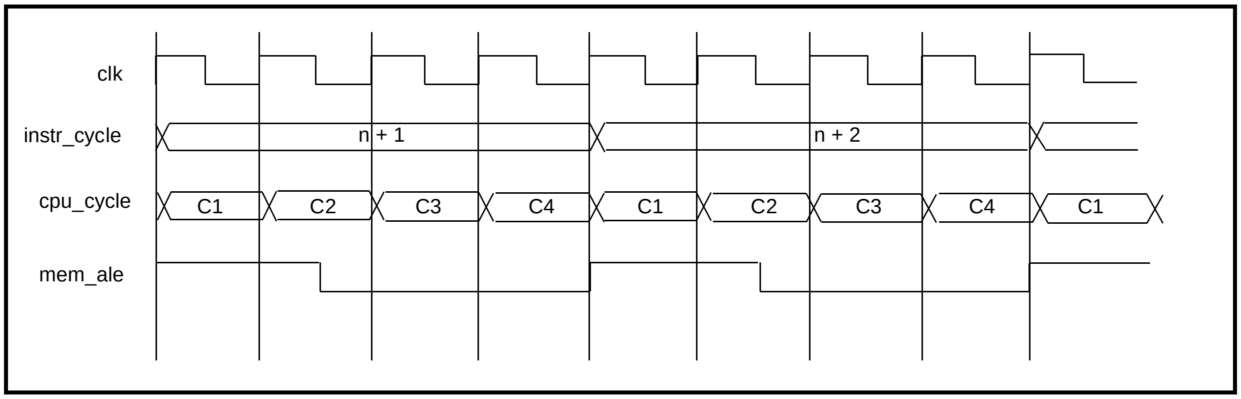
For timing of real-time events, use the numbers of instruction cycles from Table 2-4 to calculate the timing of software loops. The bytes column of Table 2-4 indicates the number of memory accesses (bytes) needed to execute the instruction. In most cases, the number of bytes is equal to the number of instruction cycles required to complete the instruction. However, as indicated in Table 2-4, there are some instructions (for example, `DIV` and `MUL`) that require a greater number of instruction cycles than memory accesses.

By default, the DW8051 timer/counters run at 12 clock cycles per increment so that timer-based events have the same timing as with the standard 8051. The timers can be configured to run at 4 clock cycles per increment to take advantage of the higher speed of the DW8051.

CPU Timing

As previously stated, a DW8051 instruction cycle consists of *4 clk* cycles. Each *clk* cycle forms a CPU cycle. Therefore, an instruction cycle consists of 4 CPU cycles: C1, C2, C3, and C4, as illustrated in Figure 2-5. Various events occur in each CPU cycle, depending on the type of instruction being executed. Throughout this databook, the labels C1, C2, C3, and C4 in timing descriptions refer to the 4 CPU cycles within a particular instruction cycle.

Figure 2-5 CPU Timing for Single-Cycle Instruction



Stretch Memory Cycles

The stretch memory cycle feature enables application software to adjust the speed of data memory access. The DW8051 can execute the `MOVX` instruction in as little as 2 instruction cycles. However, it is sometimes desirable to stretch this value; for example, to access slow memory or slow memory-mapped peripherals such as UARTs or LCDs.

The three LSBs of the Clock Control Register (at SFR location 8Eh) control the stretch value. You can use stretch values between zero and seven. A stretch value of zero adds zero instruction cycles, resulting in `MOVX` instructions executing in two instruction cycles. A stretch value of seven adds seven instruction cycles, resulting in `MOVX` instructions executing in nine instruction cycles. The stretch value can be changed dynamically under program control.

By default, the stretch value resets to one (three cycle `MOVX`). For full-speed data memory access, the software must set the stretch value to zero. The stretch value affects only data memory access. The only way to reduce the speed of program memory (ROM) access is to use a slower clock.

The stretch value affects the width of the read/write strobe and all related timing. Using a higher stretch value results in a wider read/write strobe, which allows the memory or peripheral more time to respond.

Table 2-5 lists the data memory access speeds for stretch values zero through seven. MD2–0 are the three LSBs of the Clock Control Register (CKCON.2–0). For timing diagrams, see “External RAM Timing” on page 4-50.

Table 2-5 Data Memory Stretch Values

MD2	**MD1**	**MD0**	**Memory Cycles**	**Read/Write Strobe Width (Clocks)**	**Strobe Width Time @25 MHz**
0	0	0	2	2	80 ns
0	0	1	3 (default)	4	160 ns
0	1	0	4	8	320 ns
0	1	1	5	12	480 ns
1	0	0	6	16	640 ns
1	0	1	7	20	800 ns
1	1	0	8	24	960 ns
1	1	1	9	28	1120 ns

Dual Data Pointers

The DW8051 employs dual data pointers to accelerate data memory block moves. The standard 8051 data pointer (DPTR) is a 16-bit value used to address external data RAM or peripherals. The DW8051 maintains the standard data pointer as DPTR0 at SFR locations 82h and 83h. It is not necessary to modify code to use DPTR0.

The DW8051 adds a second data pointer (DPTR1) at SFR locations 84h and 85h. The SEL bit in the DPTR Select register, DPS (SFR 86h), selects the active pointer. When SEL = 0, instructions that use the DPTR will use DPL0 and DPH0. When SEL = 1, instructions that use the DPTR will use DPL1 and DPH1. SEL is the bit 0 of SFR location 86h. No other bits of SFR location 86h are used.

All DPTR-related instructions use the currently selected data pointer. To switch the active pointer, toggle the SEL bit. The fastest way to do so is to use the increment instruction (`INC DPS`). This requires only one instruction to switch from a source address to a destination address, saving application code from having to save source and destination addresses when doing a block move.

Using dual data pointers provides significantly increased efficiency when moving large blocks of data.

The SFR locations related to the dual data pointers are:

82h	DPL0	DPTR0 low byte
83h	DPH0	DPTR0 high byte
84h	DPL1	DPTR1 low byte
85h	DPH1	DPTR1 high byte
86h	DPS	DPTR Select (LSB)

Special Function Registers

The Special Function Registers (SFRs) control several of the features of the DW8051. Most of the DW8051 SFRs are identical to the standard 8051 SFRs. However, there are additional SFRs that control features that are not available in the standard 8051.

Table 2-6 lists the DW8051 SFRs and indicates which SFRs are not included in the standard 8051 SFR space. When writing software for the DW8051, use equate statements to define the SFRs that are specific to the DW8051 and custom peripherals.

In Table 2-6, SFR bit positions that contain a 0 or a 1 cannot be written to and, when read, always return the value shown (0 or 1). SFR bit positions that contain "–" are available but not used. Table 2-7 lists the reset values for the SFRs.

Table 2-6 Special Function Registers

Register	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Addr**
SP	–	–	–	–	–	–	–	–	81h
DPL0	–	–	–	–	–	–	–	–	82h
DPH0	–	–	–	–	–	–	–	–	83h
DPL1(1)	–	–	–	–	–	–	–	–	84h
DPH1(1)	–	–	–	–	–	–	–	–	85h
DPS(1)	0	0	0	0	0	0	0	SEL	86h
PCON	SM OD 0	–	1	1	GF1	GF0	STO P	IDLE	87h
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88h
TMOD	GAT E	C/T	M1	M0	GAT E	C/T	M1	M0	89h
TL0	–	–	–	–	–	–	–	–	8Ah
TL1	–	–	–	–	–	–	–	–	8Bh
TH0	–	–	–	–	–	–	–	–	8Ch
TH1	–	–	–	–	–	–	–	–	8Dh

Table 2-6 Special Function Registers (continued)

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Addr
CKCON(1,7)	–	–	T2M	T1M	T0M	MD2	MD1	MD0	8Eh
SPC_FNC(1)	0	0	0	0	0	0	0	WRS	8Fh
EXIF(1,4)	IE5	IE4	IE3	IE2	1	0	0	0	91h
MPAGE(1)	–	–	–	–	–	–	–	–	92h
SCON0(3)	SM0_0	SM1_0	SM2_0	REN_0	TB8_0	RB8_0	TI_0	RI_0	98h
SBUF0(3)	–	–	–	–	–	–	–	–	99h
IE(6)	EA	ES1	ET2	ES0	ET1	EX1	ET0	EX0	A8h
IP(6)	1	PS1	PT2	PS0	PT1	PX1	PT0	PX0	B8h
SCON1(1,5)	SM0_1	SM1_1	SM2_1	REN_1	TB8_1	RB8_1	TI_1	RI_1	C0h
SBUF1(1,5)	–	–	–	–	–	–	–	–	C1h
T2CON(2)	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2	C8h
RCAP2L(2)	–	–	–	–	–	–	–	–	CAh
RCAP2H(2)	–	–	–	–	–	–	–	–	CBh
TL2(2)	–	–	–	–	–	–	–	–	CCh
TH2(2)	–	–	–	–	–	–	–	–	CDh

Table 2-6 Special Function Registers (continued)

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Addr
PSW	CY	AC	F0	RS1	RS0	OV	F1	P	D0h
EICON(1 ,6)	SM OD1	1	EPFI	PFI	WDTI	0	0	0	D8h
ACC	–	–	–	–	–	–	–	–	E0h
EIE(1,4)	1	1	1	EWD I	EX5	EX4	EX3	EX2	E8h
B	–	–	–	–	–	–	–	–	F0h
EIP(1,4)	1	1	1	PWD I	PX5	PX4	PX3	PX2	F8h

(1) Not part of standard 8051 architecture.
(2) Present only when Timer 2 is implemented (timer2 = 1).
(3) Present only when Serial Port 0 is implemented (serial > 0).
(4) Present only when the extended interrupt unit is implemented (extd_intr = 1).
(5) Present only when Serial Port 1 is implemented (serial = 2).
(6) Bits ES1, PS1, EPFI, PFI, and WDTI are present only when the extended interrupt unit is implemented (extd_intr = 1). Otherwise, these bits are always read as '0'.
(7) The TM2 bit in the CKCON register is available, but not used, when Timer 2 is not implemented (timer2 = 0).

Table 2-7 Special Function Register Reset Values

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Addr
SP	0	0	0	0	0	1	1	1	81h
DPL0	0	0	0	0	0	0	0	0	82h
DPH0	0	0	0	0	0	0	0	0	83h
DPL1(1)	0	0	0	0	0	0	0	0	84h

Table 2-7 Special Function Register Reset Values(continued)

Register	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Addr**
DPH1(1)	0	0	0	0	0	0	0	0	85h
DPS(1)	0	0	0	0	0	0	0	0	86h
PCON	0	0	1	1	0	0	0	0	87h
TCON	0	0	0	0	0	0	0	0	88h
TMOD	0	0	0	0	0	0	0	0	89h
TL0	0	0	0	0	0	0	0	0	8Ah
TL1	0	0	0	0	0	0	0	0	8Bh
TH0	0	0	0	0	0	0	0	0	8Ch
TH1	0	0	0	0	0	0	0	0	8Dh
CKCON(1)	0	0	0	0	0	0	0	1	8Eh
SPC_FNC(1)	0	0	0	0	0	0	0	0	8Fh
EXIF(1,4)	0	0	0	0	1	0	0	0	91h
MPAGE(1)	0	0	0	0	0	0	0	0	92h
SCON0(3)	0	0	0	0	0	0	0	0	98h
SBUF0(3)	0	0	0	0	0	0	0	0	99h
IE	0	0	0	0	0	0	0	0	A8h
IP	1	0	0	0	0	0	0	0	B8h
SCON1(1,5)	0	0	0	0	0	0	0	0	C0h
SBUF1(1,5)	0	0	0	0	0	0	0	0	C1h

Table 2-7 Special Function Register Reset Values(continued)

Register	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Addr**
T2CON(2)	0	0	0	0	0	0	0	0	C8h
RCAP2L(2)	0	0	0	0	0	0	0	0	CAh
RCAP2H(2)	0	0	0	0	0	0	0	0	CBh
TL2(2)	0	0	0	0	0	0	0	0	CCh
TH2(2)	0	0	0	0	0	0	0	0	CDh
PSW	0	0	0	0	0	0	0	0	D0h
EICON(1)	0	1	0	0	0	0	0	0	D8h
ACC	0	0	0	0	0	0	0	0	E0h
EIE(1,4)	1	1	1	0	0	0	0	0	E8h
B	0	0	0	0	0	0	0	0	F0h
EIP(1,4)	1	1	1	0	0	0	0	0	F8h

(1) Not part of standard 8051 architecture.
(2) Present only when Timer 2 is implemented (timer2 = 1).
(3) Present only when Serial Port 0 is implemented (serial > 0).
(4) Present only when the extended interrupt unit is implemented (extd_intr = 1).
(5) Present only when Serial Port 1 is implemented (serial = 2).

The following SFRs are related to CPU operation and program execution:

81h	SP	Stack Pointer
D0h	PSW	Program Status Word (Table 2-8)

E0h	ACC	Accumulator Register
F0h	B	B Register

Table 2-8 lists the functions of the bits in the PSW SFR. Detailed descriptions of the remaining SFRs appear with the associated hardware descriptions in Chapter 3 of this databook.

Table 2-8 PSW Register – SFR D0h

Bit	**Function**
PSW.7	CY – Carry flag. Set to 1 when the last arithmetic operation resulted in a carry (during addition) or borrow (during subtraction), otherwise cleared to 0 by all arithmetic operations.
PSW.6	AC – Auxiliary carry flag. Set to 1 when the last arithmetic operation resulted in a carry into (during addition) or borrow from (during subtraction) the high order nibble, otherwise cleared to 0 by all arithmetic operations.
PSW.5	F0 – User flag 0. Bit-addressable, general purpose flag for software control.
PSW.4	RS1 – Register bank select bit 1. Used with RS0 to select a register blank in internal RAM.
PSW.3	RS0 – Register bank select bit 0, decoded as: RS1 RS0 Bank Selected 0 0 Register bank 0, addresses 00h–07h 0 1 Register bank 1, addresses 08h–0Fh 1 0 Register bank 2, addresses 10h–17h 1 1 Register bank 3, addresses 18h–1Fh
PSW.2	OV – Overflow flag. Set to 1 when the last arithmetic operation resulted in a carry (addition), borrow (subtraction), or overflow (multiply or divide). Otherwise, the bit cleared to 0 by all arithmetic operations.
PSW.1	F1 – User flag 1. Bit-addressable, general purpose flag for software control.
PSW.0	P – Parity flag. Set to 1 when the modulo-2 sum of the 8 bits in the accumulator is 1 (odd parity), cleared to 0 on even parity.

3

DW8051 Hardware Description

This chapter provides technical data about the DW8051 MacroCell hardware operation and timing. The topics are:

- Timers/Counters
- Serial Interface
- SFR Bus Peripheral Interface
- Interrupts
- Reset
- Power Saving Modes

Timers/Counters

The DW8051 includes two timer/counters (Timer 0 and Timer 1) and an optional third timer/counter (Timer 2). Timer 2 is present when the parameter *timer2* = 1. Each timer/counter can operate as either a timer with a clock rate based on the *clk* pin, or as an event counter clocked by the *t0* pin (Timer 0), *t1* pin (Timer 1), or the *t2* pin (Timer 2).

Each timer/counter consists of a 16-bit register that is accessible to software as two SFRs:

- Timer 0 – TL0 and TH0
- Timer 1 – TL1 and TH1
- Timer 2 – TL2 and TH2

803x/805x Compatibility

In general, when *timer2* = 1, the implementation of the timers/counters is similar to that of the Dallas Semiconductor DS80C320. Table 3-1 summarizes the differences in timer/counter implementation between the Intel 8051, the Dallas Semiconductor DS80C320, and the DesignWare DW8051 MacroCell.

Table 3-1 Timer/Counter Implementation Comparison

Feature	**Intel 8051**	**Dallas DS80C320**	**DW8051**
Number of timers	2	3	2 or 3
Timer 0/1 overflow available as output signals	not implemented	not implemented	t0_out, t1_out (one clk pulse)
Timer 2 output enable	n/a	implemented	not implemented
Timer 2 downcount enable	n/a	implemented	not implemented

Table 3-1 Timer/Counter Implementation Comparison

Feature	Intel 8051	Dallas DS80C320	DW8051
Timer 2 overflow available as output signal	n/a	implemented	t2_out (one clk pulse)

Timers 0 and 1

Timers 0 and 1 each operate in four modes, as controlled through the TMOD SFR (Table 3-2) and the TCON SFR (Table 3-3). The four modes are:

- 13-bit timer/counter (mode 0)
- 16-bit timer/counter (mode 1)
- 8-bit counter with auto-reload (mode 2)
- Two 8-bit counters (mode 3, Timer 0 only)

Table 3-2 TMOD Register – SFR 89h

Bit	Function
TMOD.7	GATE – Timer 1 gate control. When GATE = 1, Timer 1 will clock only when int1_n = 1 and TR1 (TCON.6) = 1. When GATE = 0, Timer 1 will clock only when TR1 = 1, regardless of the state of int1_n.
TMOD.6	C/T – Counter/Timer select. When C/T = 0, Timer 1 is clocked by clk/4 or clk/12, depending on the state of T1M (CKCON.4). When C/T = 1, Timer 1 is clocked by the t1 pin.
TMOD.5	M1 – Timer 1 mode select bit 1.
TMOD.4	M0 – Timer 1 mode select bit 0, decoded as: M1M0Mode 0 0Mode 0 : 13-bit counter 0 1Mode 1 : 16-bit counter 1 0Mode 2 : 8-bit counter with auto-reload 1 1Mode 3 : Two 8-bit counters

Table 3-2 TMOD Register – SFR 89h

Bit	Function
TMOD.3	GATE – Timer 0 gate control. When GATE = 1, Timer 0 will clock only when int0_n = 1 and TR0 (TCON.4) = 1. When GATE = 0, Timer 0 will clock only when TR0 = 1, regardless of the state of int0_n.
TMOD.2	C/T – Counter/Timer select. When C/T = 0, Timer 0 is clocked by clk/4 or clk/12, depending on the state of T0M (CKCON.3). When C/T = 1, Timer 0 is clocked by the t0 pin.
TMOD.1	M1 – Timer 0 mode select bit 1.
TMOD.0	M0 – Timer 0 mode select bit 0, decoded as: M1M0Mode 0 0Mode 0 : 13-bit counter 0 1Mode 1 : 16-bit counter 1 0Mode 2 : 8-bit counter with auto-reload 1 1Mode 3 : Two 8-bit counters

Table 3-3 TCON Register – SFR 88h

Bit	Function
TCON.7	TF1 – Timer 1 overflow flag. Set to 1 when the Timer 1 count overflows and cleared when the CPU vectors to the interrupt service routine.
TCON.6	TR1 – Timer 1 run control. Set to 1 to enable counting on Timer 1.
TCON.5	TF0 – Timer 0 overflow flag. Set to 1 when the Timer 0 count overflows and cleared when the CPU vectors to the interrupt service routine.
TCON.4	TR0 – Timer 0 run control. Set to 1 to enable counting on Timer 0.

Table 3-3 TCON Register – SFR 88h

Bit	Function
TCON.3	IE1 – Interrupt 1 edge detect. If external interrupt 1 is configured to be edge-sensitive (IT1 = 1), IE1 is set by hardware when a negative edge is detected on the int1_n pin and is automatically cleared when the CPU vectors to the corresponding interrupt service routine. In edge-sensitive mode, IE1 can also be cleared by software. If external interrupt 1 is configured to be level-sensitive (IT1 = 0), IE1 is set when the int1_n pin is low and cleared when the int1_n pin is high. In level-sensitive mode, software cannot write to IE1.
TCON.2	IT1 – Interrupt 1 type select. When IT1 = 1, the DW8051 detects int1_n on the falling edge (edge-sensitive). When IT1 = 0, the DW8051 detects int1_n as a low level (level-sensitive).
TCON.1	IE0 – Interrupt 0 edge detect. If external interrupt 0 is configured to be edge-sensitive (IT0 = 1), IE0 is set by hardware when a negative edge is detected on the int0_n pin and is automatically cleared when the CPU vectors to the corresponding interrupt service routine. In edge-sensitive mode, IE0 can also be cleared by software. If external interrupt 0 is configured to be level-sensitive (IT0 = 0), IE0 is set when the int0_n pin is low and cleared when the int0_n pin is high. In level-sensitive mode, software cannot write to IE0.
TCON.0	IT0 – Interrupt 0 type select. When IT1 = 1, the DW8051 detects int0_n on the falling edge (edge-sensitive). When IT1 = 0, the DW8051 detects int0_n as a low level (level-sensitive).

Mode 0

Mode 0 operation, illustrated in Figure 3-1, is the same for Timer 0 and Timer 1. In mode 0, the timer is configured as a 13-bit counter that uses bits 0–4 of TL0 (or TL1) and all 8 bits of TH0 (or TH1). The timer enable bit (TR0/TR1) in the TCON SFR starts the timer. The C/T bit selects the timer/counter clock source, *clk* or *t0/t1*.

The timer counts transitions from the selected source as long as the GATE bit is 0, or the GATE bit is 1 and the corresponding interrupt pin (*int0_n* or *int1_n*) is deasserted.

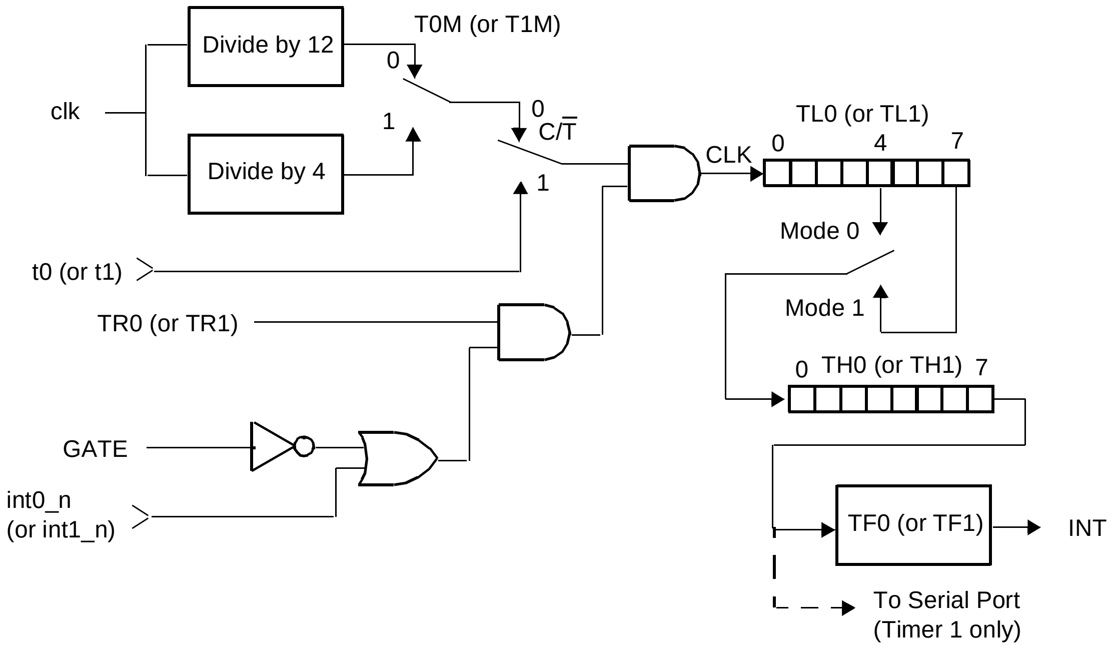
When the 13-bit count increments from 1FFFh (all ones), the counter rolls over to all zeros, the TF0 (or TF1) bit is set in the TCON SFR, and the *t0_out* (or *t1_out*) pin goes high for one clock cycle.

The upper 3 bits of TL0 (or TL1) are indeterminate in mode 0 and must be masked when the software evaluates the register.

Mode 1

Mode 1 operation is the same for Timer 0 and Timer 1. In mode 1, the timer is configured as a 16-bit counter. As illustrated in Figure 3-1, all 8 bits of the LSB register (TL0 or TL1) are used. The counter rolls over to all zeros when the count increments from FFFFh. Otherwise, mode 1 operation is the same as mode 0.

Figure 3-1 Timer 0/1 – Modes 0 and 1

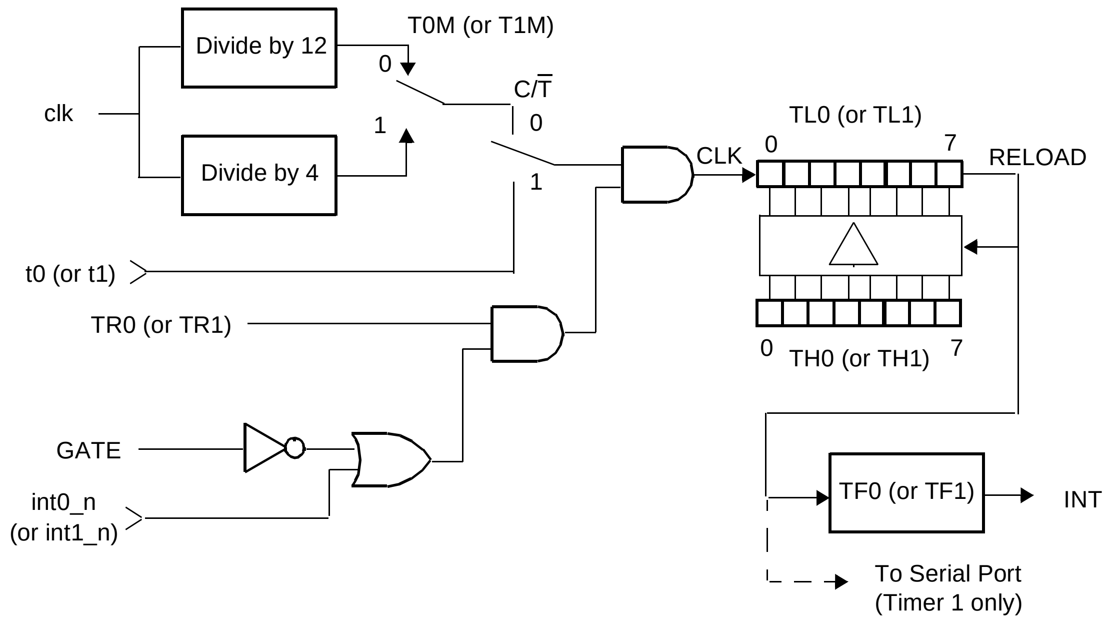


Mode 2

Mode 2 operation is the same for Timer 0 and Timer 1. In mode 2, the timer is configured as an 8-bit counter, with automatic reload of the start value. The LSB register (TL0 or TL1) is the counter and the MSB register (TH0 or TH1) stores the reload value.

As illustrated in Figure 3-2, mode 2 counter control is the same as for mode 0 and mode 1. However, in mode 2, when TL*n* increments from FFh, the value stored in TH*n* is reloaded into TL*n*.

Figure 3-2 Timer 0/1 – Mode 2



Mode 3

In mode 3, Timer 0 operates as two 8-bit counters and Timer 1 stops counting and holds its value.

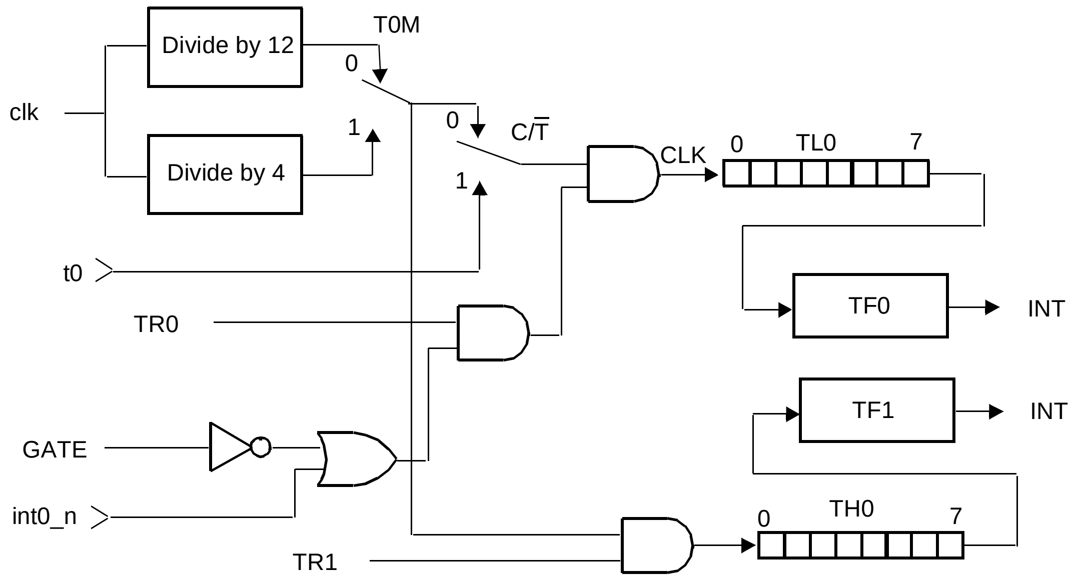
As shown in Figure 3-3, TL0 is configured as an 8-bit counter controlled by the normal Timer 0 control bits. TL0 can either count *clk* cycles (divided by 4 or by 12) or high-to-low transitions on *t0*, as determined by the C/T bit. The GATE function can be used to give counter enable control to the *int0_n* signal.

TH0 functions as an independent 8-bit counter. However, TH0 can only count *clk* cycles (divided by 4 or by 12). The Timer 1 control and flag bits (TR1 and TF1) are used as the control and flag bits for TH0.

When Timer 0 is in mode 3, Timer 1 has limited usage because Timer 0 uses the Timer 1 control bit (TR1) and interrupt flag (TF1). Timer 1 can still be used for baud rate generation and the Timer 1 count values are still available in the TL1 and TH1 registers.

Control of Timer 1 when Timer 0 is in mode 3 is through the Timer 1 mode bits. To turn Timer 1 on, set Timer 1 to mode 0, 1, or 2. To turn Timer 1 off, set it to mode 3. The Timer 1 C/T bit and T1M bit are still available to Timer 1. Therefore, Timer 1 can count *clk*/4, *clk*/12, or high-to-low transitions on the *t1* pin. The Timer 1 GATE function is also available when Timer 0 is in mode 3.

Figure 3-3 Timer 0 – Mode 3



Timer Rate Control

The default timer clock scheme for the DW8051 timers is 12 *clk* cycles per increment, the same as in the standard 8051. However, in the DW8051, the instruction cycle is 4 *clk* cycles.

Using the default rate (12 clocks per timer increment) allows existing application code with real-time dependencies, such as baud rate, to operate properly. However, applications that require fast timing can set the timers to increment every 4 *clk* cycles by setting bits in the Clock Control register (CKCON) at SFR location 8Eh (see Table 3-4).

The CKCON bits that control the timer clock rates are:

CKCON bit	Counter/Timer
5	Timer 2
4	Timer 1

When a CKCON register bit is set to 1, the associated counter increments at 4-*clk* intervals. When a CKCON bit is cleared, the associated counter increments at 12-*clk* intervals. The timer controls are independent of each other. The default setting for all three timers is 0 (12-*clk* intervals). These bits have no effect in counter mode.

Table 3-4 CKCON Register – SFR 8Eh

Bit	**Function**
CKCON.7,6	Reserved
CKCON.5	T2M – Timer 2 clock select. When T2M = 0, Timer 2 uses clk/12 (for compatibility with 80C32); when T2M = 1, Timer 2 uses clk/4. This bit has no effect when Timer 2 is configured for baud rate generation.
CKCON.4	T1M – Timer 1 clock select. When T1M = 0, Timer 1 uses clk/12 (for compatibility with 80C32); when T1M = 1, Timer 1 uses clk/4.
CKCON.3	T0M – Timer 0 clock select. When T0M = 0, Timer 0 uses clk/12 (for compatibility with 80C32); when T0M = 1, Timer 0 uses clk/4.
CKCON.2–0	MD2, MD1, MD0 – Control the number of cycles to be used for external MOVX instructions. See “Stretch Memory Cycles” in Chapter 2 for details.

Timer 2

Timer 2 is present only when the *timer2* parameter is set to 1. When present, Timer 2 runs only in 16-bit mode and offers several capabilities not available with Timers 0 and 1. The modes available with Timer 2 are:

- 16-bit timer/counter
- 16-bit timer with capture

- 16-bit auto-reload timer/counter
- Baud rate generator

The SFRs associated with Timer 2 are:

- T2CON – SFR C8h (Table 3-5)
- RCAP2L – SFR CAh – Used to capture the TL2 value when Timer 2 is configured for capture mode, or as the LSB of the 16-bit reload value when Timer 2 is configured for auto-reload mode.
- RCAP2H – SFR CBh – Used to capture the TH2 value when Timer 2 is configured for capture mode, or as the MSB of the 16-bit reload value when Timer 2 is configured for auto-reload mode.
- TL2 – SFR CCh – Lower 8 bits of the 16-bit count.
- TH2 – SFR CDh – Upper 8 bits of the 16-bit count.

Table 3-5 T2CON Register – SFR C8h

Bit	**Function**
T2CON.7	TF2 – Timer 2 overflow flag. Hardware will set TF2 when Timer 2 overflows from FFFFh. TF2 must be cleared to 0 by the software. TF2 will only be set to a 1 if RCLK and TCLK are both cleared to 0. Writing a 1 to TF2 forces a Timer 2 interrupt if enabled.
T2CON.6	EXF2 – Timer 2 external flag. Hardware will set EXF2 when a reload or capture is caused by a high-to-low transition on the t2ex pin, and EXEN2 is set. EXF2 must be cleared to 0 by the software. Writing a 1 to EXF2 forces a Timer 2 interrupt if enabled.
T2CON.5	RCLK – Receive clock flag. Determines whether Timer 1 or Timer 2 is used for Serial Port 0 timing of received data in serial mode 1 or 3. RCLK =1 selects Timer 2 overflow as the receive clock. RCLK = 0 selects Timer 1 overflow as the receive clock.

Table 3-5 T2CON Register – SFR C8h

Bit	**Function**
T2CON.4	TCLK – Transmit clock flag. Determines whether Timer 1 or Timer 2 is used for Serial Port 0 timing of transmit data in serial mode 1 or 3. TCLK =1 selects Timer 2 overflow as the transmit clock. TCLK = 0 selects Timer 1 overflow as the transmit clock.
T2CON.3	EXEN2 – Timer 2 external enable. EXEN2 = 1 enables capture or reload to occur as a result of a high-to-low transition on t2ex, if Timer 2 is not generating baud rates for the serial port. EXEN2 = 0 causes Timer 2 to ignore all external events at t2ex.
T2CON.2	TR2 – Timer 2 run control flag. TR2 = 1 starts Timer 2. TR2 = 0 stops Timer 2.
T2CON.1	C/T2 – Counter/timer select. C/T2 = 0 selects a timer function for Timer 2. C/T2 = 1 selects a counter of falling transitions on the t2 pin. When used as a timer, Timer 2 runs at 4 clocks per increment or 12 clocks per increment as programmed by CKCON.5, in all modes except baud rate generator mode. When used in baud rate generator mode, Timer 2 runs at 2 clocks per increment, independent of the state of CKCON.5.
T2CON.0	CP/RL2 – Capture/reload flag. When CP/RL2 = 1, Timer 2 captures occur on high-to-low transitions of t2ex, if EXEN2 = 1. When CP/RL2 = 0, auto-reloads occur when Timer 2 overflows or when high-to-low transitions occur on t2ex, if EXEN2 = 1. If either RCLK or TCLK is set to 1, CP/RL2 will not function and Timer 2 will operate in auto-reload mode following each overflow.

Timer 2 Mode Control

Table 3-6 summarizes how the SFR bits determine the Timer 2 mode.

Table 3-6 Timer 2 Mode Control Summary

RCLK	**TCLK**	**CP/RL2**	**TR2**	**Mode**
0	0	1	1	16-bit timer/counter with capture
0	0	0	1	16-bit timer/counter with auto-reload
1	X	X	1	Baud rate generator

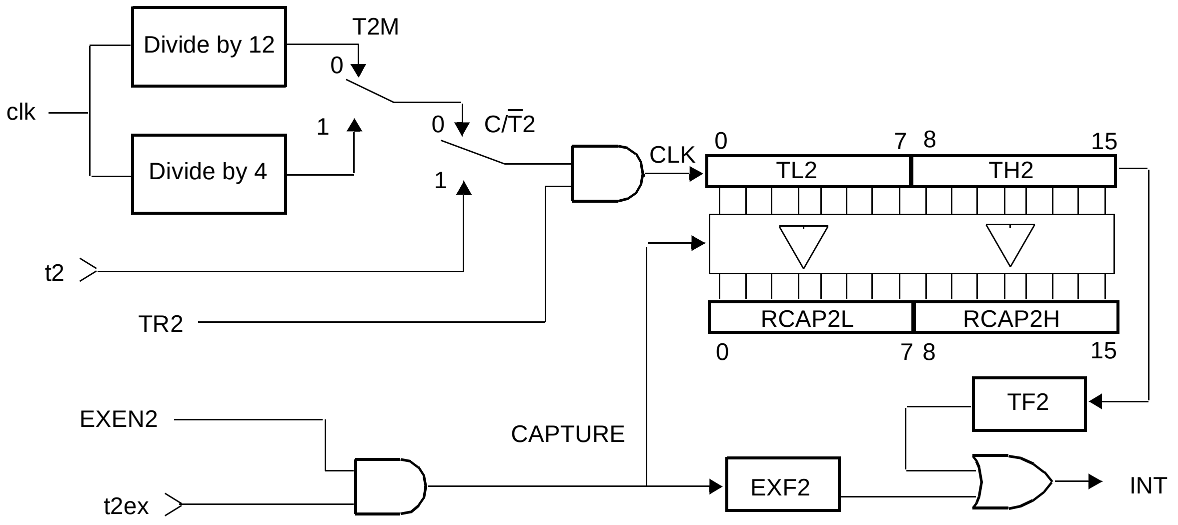
Table 3-6 Timer 2 Mode Control Summary

RCLK	**TCLK**	**CP/RL2**	**TR2**	**Mode**
X	1	X	1	Baud rate generator
X	X	X	0	Off
X = Don’t care.				

16-Bit Timer/Counter Mode

Figure 3-4 illustrates how Timer 2 operates in timer/counter mode with the optional capture feature. The C/T2 bit determines whether the 16-bit counter counts *clk* cycles (divided by 4 or 12), or high-to-low transitions on the *t2* pin. The TR2 bit enables the counter. When the count increments from FFFFh, the TF2 flag is set, and *t2_out* goes high for one *clk* cycle.

Figure 3-4 Timer 2 – Timer/Counter with Capture



16-Bit Timer/Counter Mode with Capture

The Timer 2 capture mode (Figure 3-4) is the same as the 16-bit timer/ counter mode, with the addition of the capture registers and control signals.

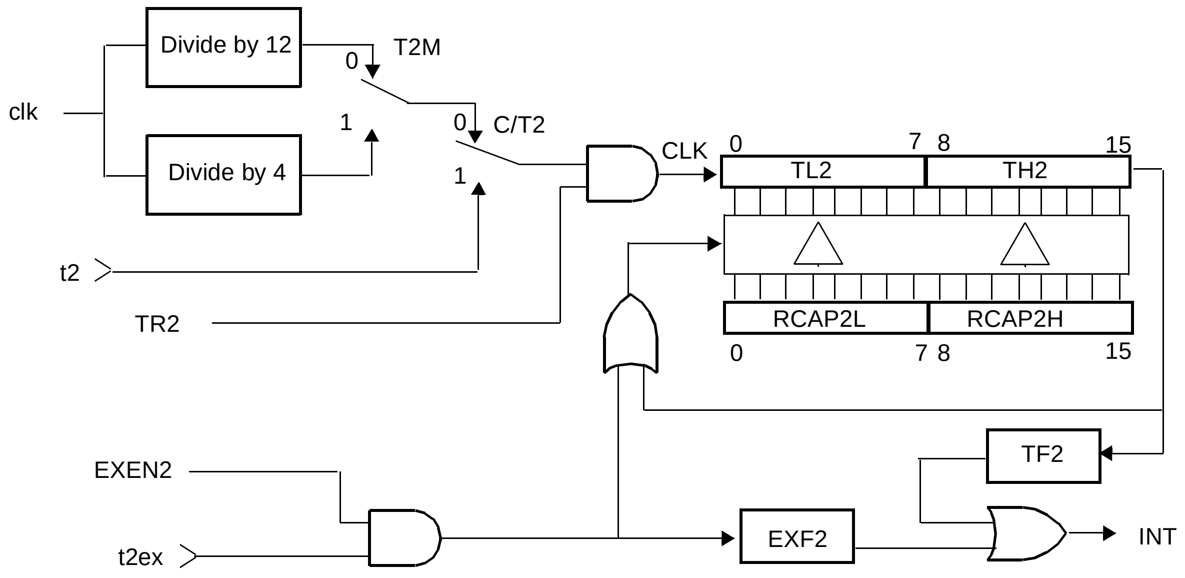
The CP/RL2 bit in the T2CON SFR enables the capture feature. When CP/RL2 = 1, a high-to-low transition on *t2ex* when EXEN2 = 1 causes the Timer 2 value to be loaded into the capture registers (RCAP2L and RCAP2H).

16-Bit Timer/Counter Mode with Auto-Reload

When CP/RL2 = 0, Timer 2 is configured for the auto-reload mode illustrated in Figure 3-5. Control of counter input is the same as for the other 16-bit counter modes. When the count increments from FFFFh, Timer 2 sets the TF2 flag and the starting value is reloaded into TL2 and TH2. The software must preload the starting value into the RCAP2L and RCAP2H registers.

When Timer 2 is in auto-reload mode, a reload can be forced by a high-to-low transition on the *t2ex* pin, if enabled by EXEN2 = 1.

Figure 3-5   Timer 2 – Timer/Counter with Auto-Reload



Baud Rate Generator Mode

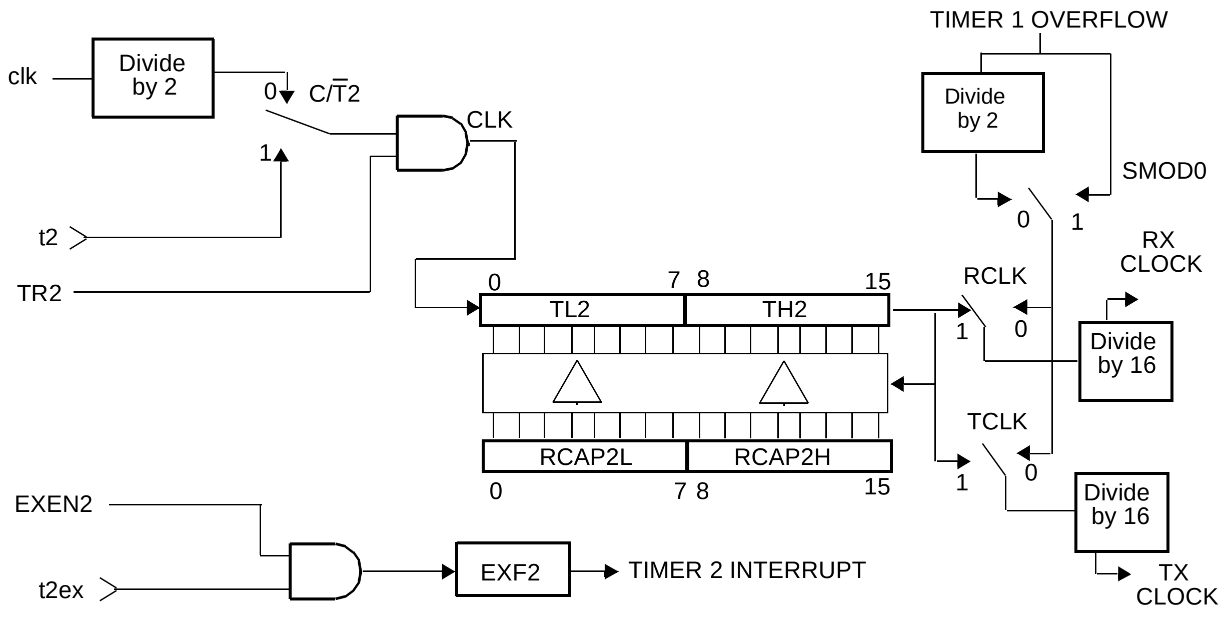
Setting either RCLK or TCLK to 1 configures Timer 2 to generate baud rates for Serial Port 0 in serial mode 1 or 3. In baud rate generator mode, Timer 2 functions in auto-reload mode. However, instead of setting the TF2 flag, the counter overflow generates a shift clock for the serial port function. As in normal auto-reload mode, the overflow also causes the preloaded start value in the RCAP2L and RCAP2H registers to be reloaded into the TL2 and TH2 registers.

When either TCLK = 1 or RCLK = 1, Timer 2 is forced into auto-reload operation, regardless of the state of the CP/RL2 bit.

When operating as a baud rate generator, Timer 2 does not set the TF2 bit. In this mode, a Timer 2 interrupt can only be generated by a high-to-low transition on the *t2ex* pin setting the EXF2 bit, and only if enabled by EXEN2 = 1.

The counter time base in baud rate generator mode is *clk*/2. To use an external clock source, set C/T2 to 1 and apply the desired clock source to the *t2* pin.

Figure 3-6 Timer 2 – Baud Rate Generator Mode



Serial Interface

The DW8051 provides two optional serial ports. Serial Port 0 is identical in operation to the standard 8051 serial port. Serial Port 1 is identical to Serial Port 0, except that Timer 2 cannot be used as the baud rate generator for Serial Port 1.

The *serial* parameter determines how many serial ports are present (0, 1, or 2). The extended interrupt unit is required to handle interrupt requests from Serial Port 1. However, if the standard interrupt unit is used, Serial Port 1 can still be used by software polling of the RI_1 and TI_1 flags.

Each serial port can operate in synchronous or asynchronous mode. In synchronous mode, the DW8051 generates the serial clock and the serial port operates in half-duplex mode. In asynchronous mode, the serial port operates in full-duplex mode. In all modes, the DW8051 buffers received data in a holding register, enabling the UART to receive an incoming word before the software has read the previous value.

Each serial port can operate in one of four modes, as outlined in Table 3-7.

Table 3-7 Serial Port Modes

Mode	**Sync/Async**	**Baud Clock**	**Data Bits**	**Start/Stop**	**9th Bit Function**
0	Sync	clk/4 or clk/12	8	None	None
1	Async	Timer 1 or Timer 2(1)	8	1 start, 1 stop	None
2	Async	clk/32 or clk/64	9	1 start, 1 stop	0, 1, parity
3	Async	Timer 1 or Timer 2(1)	9	1 start, 1 stop	0, 1, parity
(1) Timer 2 available for Serial Port 0 only.					

The SFRs associated with the serial ports are:

- SCON0 – SFR 98h – Serial Port 0 control (Table 3-8).
- SBUF0 – SFR 99h – Serial Port 0 buffer.
- SCON1 – SFR C0h – Serial Port 1 control (Table 3-9).
- SBUF1 – SFR C1h – Serial Port 1 buffer.

Table 3-8 SCON0 Register – SFR 98h

Bit	**Function**
SCON0.7	SM0_0 – Serial Port 0 mode bit 0.
SCON0.6	SM1_0 – Serial Port 0 mode bit 1, decoded as: SM0_0SM1_0Mode 0 0 0 0 1 1 1 0 2 1 1 3
SCON0.5	SM2_0 – Multiprocessor communication enable. In modes 2 and 3, SM2_0 enables the multiprocessor communication feature. If SM2_0 = 1 in mode 2 or 3, RI_0 will not be activated if the received 9th bit is 0. If SM2_0 = 1 in mode 1, RI_0 will only be activated if a valid stop is received. In mode 0, SM2_0 establishes the baud rate: when SM2_0 = 0, the baud rate is clk/12; when SM2_0 = 1, the baud rate is clk/4.
SCON0.4	REN_0 – Receive enable. When REN_0 = 1, reception is enabled.
SCON0.3	TB8_0 – Defines the state of the 9th data bit transmitted in modes 2 and 3.
SCON0.2	RB8_0 – In modes 2 and 3, RB8_0 indicates the state of the 9th bit received. In mode 1, RB8_0 indicates the state of the received stop bit. In mode 0, RB8_0 is not used.
SCON0.1	TI_0 – Transmit interrupt flag. Indicates that the transmit data word has been shifted out. In mode 0, TI_0 is set at the end of the 8th data bit. In all other modes, TI_0 is set when the stop bit is placed on the txd0 pin. TI_0 must be cleared by the software.

Table 3-9   SCON1 Register – SFR C0h

Bit	**Function**
SCON1.7	SM0_1 – Serial Port 1 mode bit 0.
SCON1.6	SM1_1 – Serial Port 1 mode bit 1, decoded as: SM0_1SM1_1Mode 0 0 0 0 1 1 1 0 2 1 1 3
SCON1.5	SM2_1 – Multiprocessor communication enable. In modes 2 and 3, SM2_1 enables the multiprocessor communication feature. If SM2_1 = 1 in mode 2 or 3, RI_1 will not be activated if the received 9th bit is 0. If SM2_1 = 1 in mode 1, RI_1 will only be activated if a valid stop is received. In mode 0, SM2_1 establishes the baud rate: when SM2_1 = 0, the baud rate is clk/12; when SM2_1 = 1, the baud rate is clk/4.
SCON1.4	REN_1 – Receive enable. When REN_1 = 1, reception is enabled.
SCON1.3	TB8_1 – Defines the state of the 9th data bit transmitted in modes 2 and 3.
SCON1.2	RB8_1 – In modes 2 and 3, RB8_1 indicates the state of the 9th bit received. In mode 1, RB8_1 indicates the state of the received stop bit. In mode 0, RB8_1 is not used.
SCON1.1	TI_1 – Transmit interrupt flag. Indicates that the transmit data word has been shifted out. In mode 0, TI_1 is set at the end of the 8th data bit. In all other modes, TI_1 is set when the stop bit is placed on the txd1 pin. TI_1 must be cleared by the software.
SCON1.0	RI_1 – Receive interrupt flag. Indicates that a serial data word has been received. In mode 0, RI_1 is set at the end of the 8th data bit. In mode 1, RI_1 is set after the last sample of the incoming stop bit, subject to the state of SM2_1. In modes 2 and 3, RI_1 is set at the end of the last sample of RB8_1. RI_1 must be cleared by the software.

803x/805x Compatibility

In general, when *serial* = 2, the implementation of the serial interface is similar to that of the Dallas Semiconductor DS80C320. Table 3-10 summarizes the differences in serial interface implementation between the Intel 8051, the Dallas Semiconductor DS80C320, and the DesignWare DW8051 MacroCell.

Table 3-10 Serial Interface Implementation Comparison

Feature	**Intel 8051**	**Dallas DS80C320**	**DW8051**
Number of serial ports	1	2	0, 1, or 2
Framing error detection	not implemented	implemented	not implemented
Slave address comparison for multiprocessor communication	not implemented	implemented	not implemented

Mode 0

Serial mode 0 provides synchronous, half-duplex serial communication. For Serial Port 0, serial data output occurs on *rxd0_out*, serial data is received on *rxd0_in*, and *txd0* provides the shift clock for both transmit and receive. For Serial Port 1, the corresponding pins are *rxd1_out*, *rxd1_in*, and *txd1*.

The serial mode 0 baud rate is either *clk*/12 or *clk*/4, depending on the state of the SM2_0 bit (or SM2_1 for Serial Port 1). When SM2_0 = 0, the baud rate is *clk*/12; when SM2_0 = 1, the baud rate is *clk*/4.

Mode 0 operation is identical to the standard 8051. Data transmission begins when an instruction writes to the SBUF0 (or SBUF1) SFR. The UART shifts the data out, LSB first, at the selected baud rate, until the 8-bit value has been shifted out.

Mode 0 data reception begins when the REN_0 (or REN_1) bit is set and the RI_0 (or RI_1) bit is cleared in the corresponding SCON SFR. The shift clock is activated and the UART shifts data in on each rising edge of the shift clock until 8 bits have been received. One machine cycle after the 8th bit is shifted in, the RI_0 (or RI_1) bit is set and reception stops until the software clears the RI bit.

Figure 3-7 through Figure 3-10 illustrate Serial Port Mode 0 transmit and receive timing for both low-speed (*clk*/12) and high-speed (*clk*/4) operation.

Figure 3-7   Serial Port Mode 0 Receive Timing – Low Speed Operation

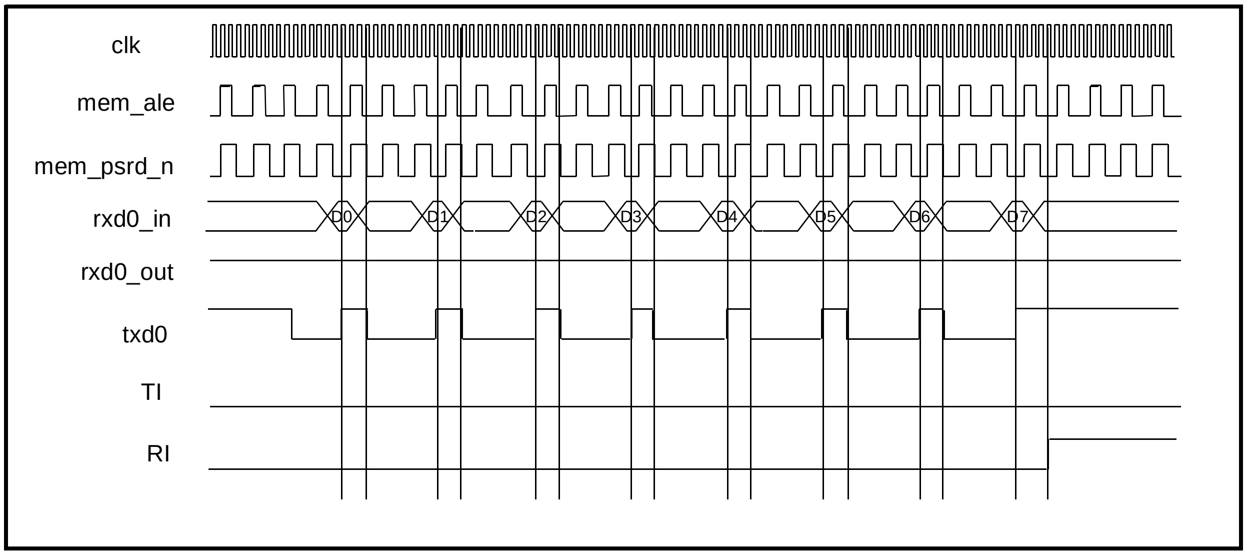


Figure 3-8   Serial Port Mode 0 Receive Timing – High Speed Operation

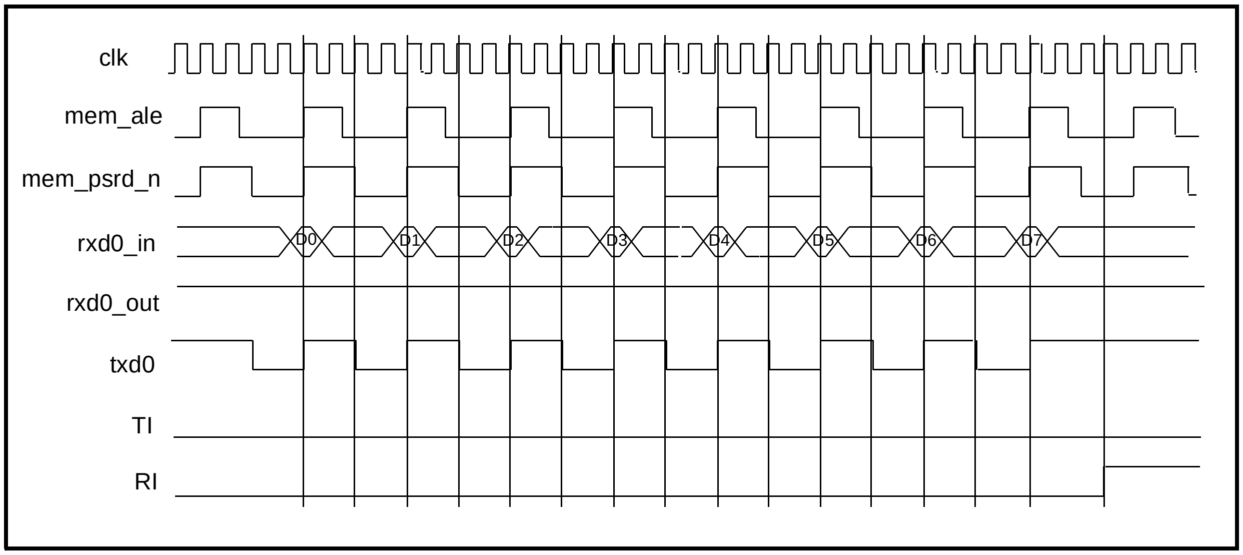


Figure 3-9 Serial Port Mode 0 Transmit Timing – Low Speed Operation

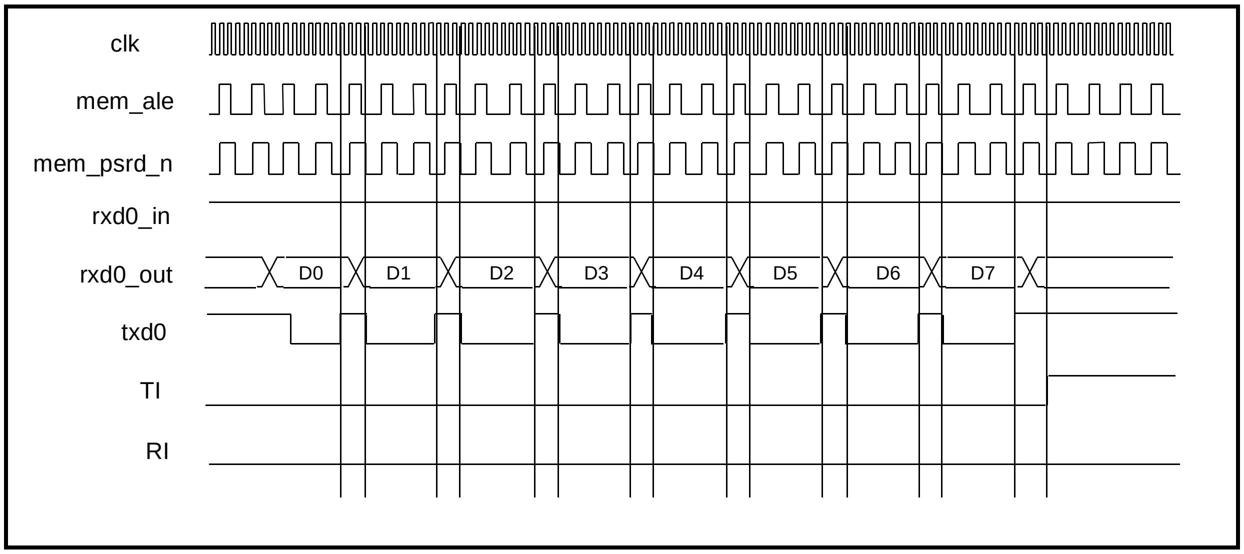
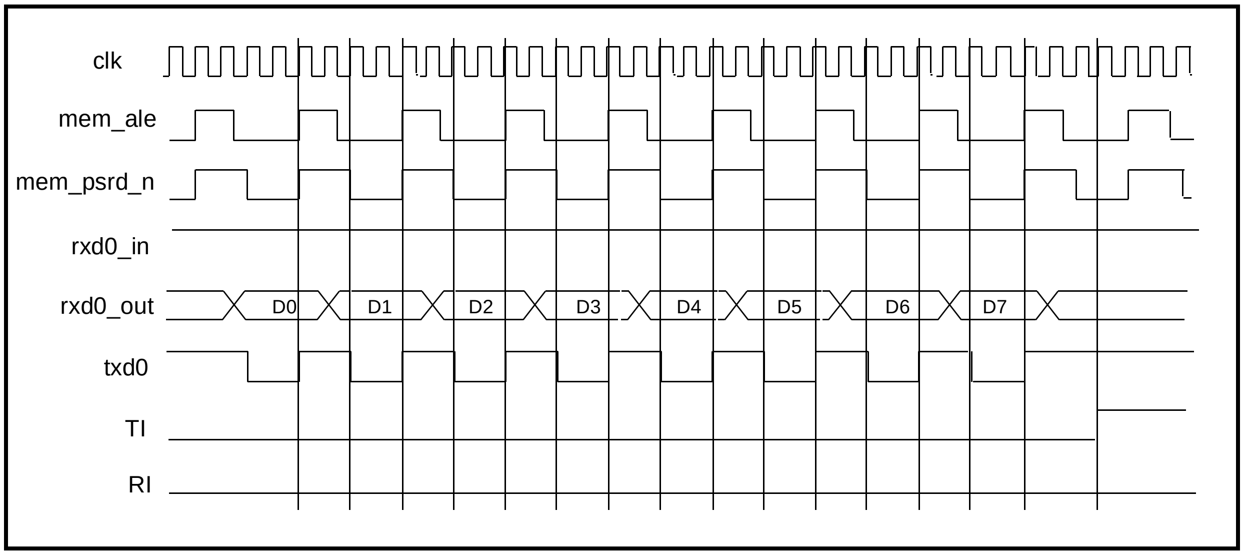


Figure 3-10 Serial Port Mode 0 Transmit Timing – High Speed Operation



Mode 1

Mode 1 provides standard asynchronous, full-duplex communication, using a total of 10 bits: 1 start bit, 8 data bits, and 1 stop bit. For receive operations, the stop bit is stored in RB8_0 (or RB8_1). Data bits are received and transmitted LSB first.

Mode 1 Baud Rate

The mode 1 baud rate is a function of timer overflow. Serial Port 0 can use either Timer 1 or Timer 2 to generate baud rates. Serial Port 1 can only use Timer 1. The two serial ports can run at the same baud rate if they both use Timer 1, or different baud rates if Serial Port 0 uses Timer 2 and Serial Port 1 uses Timer 1.

Each time the timer increments from its maximum count (FFh for Timer 1 or FFFFh for Timer 2), a clock is sent to the baud rate circuit. The clock is then divided by 16 to generate the baud rate.

When using Timer 1, the SMOD0 (or SMOD1) bit selects whether or not to divide the Timer 1 rollover rate by 2. Therefore, when using Timer 1, the baud rate is determined by the equation:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}}}{32} \times \text{Timer 1 Overflow}$$

SMOD0 is SFR bit PCON.7; SMOD1 is SFR bit EICON.7.

When using Timer 2, the baud rate is determined by the equation:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow}}{16}$$

To use Timer 1 as the baud rate generator, it is best to use Timer 1 mode 2 (8-bit counter with auto-reload), although any counter mode can be used. The Timer 1 reload value is stored in the TH1 register, which makes the complete formula for Timer 1:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}}}{32} \times \frac{\text{clk}}{12 \times (256 - \text{TH1})}$$

The 12 in the denominator in the above equation can be changed to 4 by setting the T1M bit in the CKCON SFR. To derive the required TH1 value from a known baud rate (when TM1 = 0), use the equation:

$$\text{TH1} = 256 - \frac{2^{\text{SMODx}} \times \text{clk}}{384 \times \text{Baud Rate}}$$

You can also achieve very low serial port baud rates from Timer 1 by enabling the Timer 1 interrupt, configuring Timer 1 to mode 1, and using the Timer 1 interrupt to initiate a 16-bit software reload. Table 3-11 lists sample reload values for a variety of common serial port baud rates.

Table 3-11 Timer 1 Reload Values for Common Serial Port Mode 1 Baud Rates

Desired Baud Rate	**SMODx**	**C/T**	**Timer1 Mode**	**TH1 Value for 33-MHz clk**	**TH1 Value for 25-MHz clk**	**TH1 Value for 11.0592-MHz clk**
57.6 Kb/s	1	0	2	FDh	FEh	FFh
19.2 Kb/s	1	0	2	F7h	F9h	FDh
9.6 Kb/s	1	0	2	EEh	F2h	FAh
4.8 Kb/s	1	0	2	DCh	E5h	F4h
2.4 Kb/s	1	0	2	B8h	CAh	E8h
1.2 Kb/s	1	0	2	71h	93h	D0h

To use Timer 2 as the baud rate generator, configure Timer 2 in auto-reload mode and set the TCLK and/or RCLK bits in the T2CON SFR. TCLK selects Timer 2 as the baud rate generator for the transmitter; RCLK selects Timer 2 as the baud rate generator for the receiver. The 16-bit reload value for Timer 2 is stored in the RCAP2L and RCA2H SFRs, which makes the equation for the Timer 2 baud rate:

$$\text{Baud Rate} = \frac{\text{clk}}{32 \times (65536 - \text{RCAP2H,RCAP2L})}$$

where RCAP2H,RCAP2L is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned number.

The 32 in the denominator is the result of the *clk* being divided by 2 and the Timer 2 overflow being divided by 16. Setting TCLK or RCLK to 1 automatically causes the *clk* to be divided by 2, as shown in Figure 3-6, instead of the 4 or 12 determined by the T2M bit in the CKCON SFR.

To derive the required RCAP2H and RCAP2L values from a known baud rate, use the equation:

$$\text{RCAP2H,RCAP2L} = 65536 - \frac{\text{clk}}{32 \times \text{Baud Rate}}$$

Table 3-12 lists sample values of RCAP2L and RCAP2H for a variety of desired baud rates.

Table 3-12 Timer 2 Reload Values for Common Serial Port Mode 1 Baud Rates

Baud Rate	**C/T2**	**33-MHz clk RCAP2H**	**33-MHz clk RCAP2L**	**25-MHz clk RCAP2H**	**25-MHz clk RCAP2L**	**11.0592-MHz clk RCAP2H**	**11.0592-MHz clk RCAP2L**
57.6 Kb/s	0	FFh	EEh	FFh	F2h	FFh	FAh
19.2 Kb/s	0	FFh	CAh	FFh	D7h	FFh	EEh
9.6 Kb/s	0	FFh	95h	FFh	AFh	FFh	DCh
4.8 Kb/s	0	FFh	29h	FFh	5Dh	FFh	B8h
2.4 Kb/s	0	FEh	52h	FEh	BBh	FFh	70h
1.2 Kb/s	0	FCh	A5h	FDh	75h	FEh	E0h

When either RCLK or TCLK is set, the TF2 flag will not be set on a Timer 2 rollover, and the *t2ex* reload trigger is disabled.

Mode 1 Transmit

Figure 3-11 illustrates the mode 1 transmit timing. In mode 1, the UART begins transmitting after the first rollover of the divide-by-16 counter after the software writes to the SBUF0 (or SBUF1) register. The UART transmits data on the *txd0* (or *txd1*) pin in the following order: start bit, 8 data bits (LSB first), stop bit. The TI_0 (or TI_1) bit is set 2 *clk* cycles after the stop bit is transmitted.

Mode 1 Receive

Figure 3-12 illustrates the mode 1 receive timing. Reception begins at the falling edge of a start bit received on *rxd0_in* (or *rxd1_in*), when enabled by the REN_0 (or REN_1) bit. For this purpose, *rxd0_in* (or *rxd1_in*) is sampled 16 times per bit for any baud rate. When a falling edge of a start bit is detected, the divide-by-16 counter used to generate the receive clock is reset to align the counter rollover to the bit boundaries.

For noise rejection, the serial port establishes the content of each received bit by a majority decision of 3 consecutive samples in the middle of each bit time. This is especially true for the start bit. If the falling edge on *rxd0_in* (or *rxd1_in*) is not verified by a majority decision of 3 consecutive samples (low), then the serial port stops reception and waits for another falling edge on *rxd0_in* (or *rxd1_in*).

At the middle of the stop bit time, the serial port checks for the following conditions:

- RI_0 (or RI_1) = 0, and
- If SM2_0 (or SM2_1) = 1, the state of the stop bit is 1.
  (If SM2_0 (or SM2_1) = 0, the state of the stop bit doesn’t matter.)

If the above conditions are met, the serial port then writes the received byte to the SBUF0 (or SBUF1) register, loads the stop bit into RB8_0 (or RB8_1), and sets the RI_0 (or RI_1) bit. If the above conditions are not met, the received data is lost, the SBUF register and RB8 bit are not loaded, and the RI bit is not set.

After the middle of the stop bit time, the serial port waits for another high-to-low transition on the (*rxd0_in* or *rxd1_in*) pin.

Mode 1 operation is identical to that of the standard 8051 when Timers 1 and 2 use *clk*/12 (the default).

Figure 3-11 Serial Port 0 Mode 1 Transmit Timing

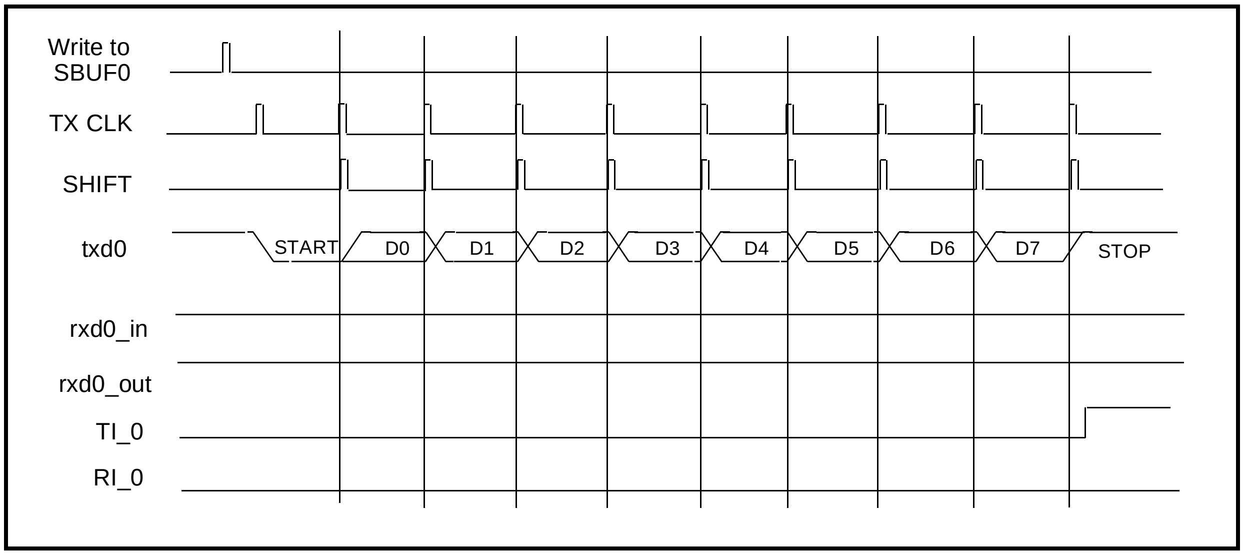
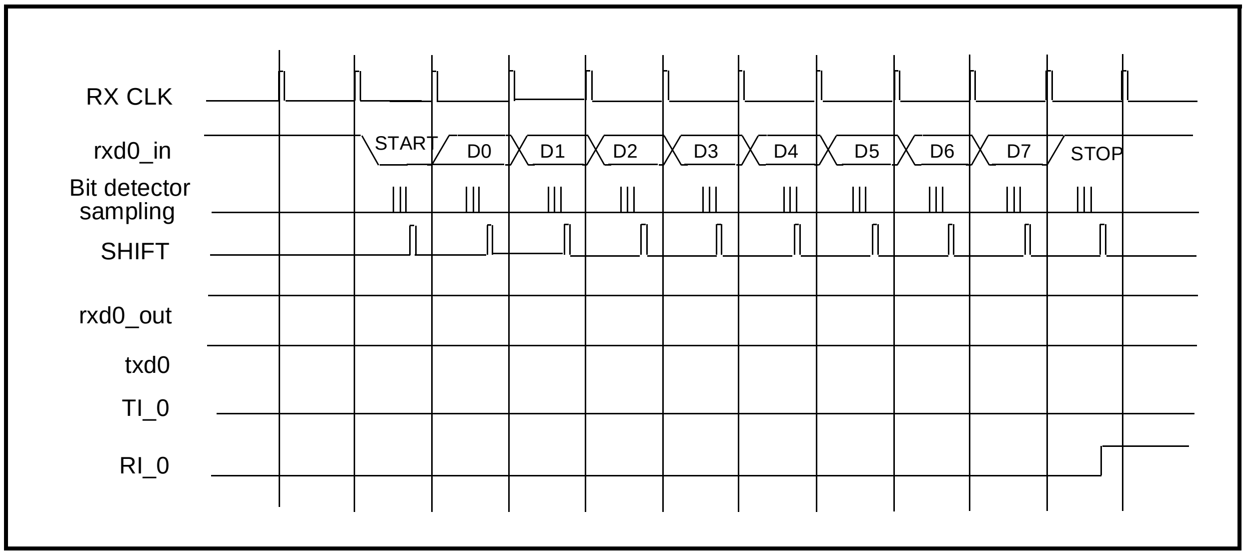


Figure 3-12 Serial Port 0 Mode 1 Receive Timing



Mode 2

Mode 2 provides asynchronous, full-duplex communication, using a total of 11 bits: 1 start bit, 8 data bits, a programmable 9th bit, and 1 stop bit. The data bits are transmitted and received LSB first. For transmission, the 9th bit is determined by the value in TB8_0 (or TB8_1). To use the 9th bit as a parity bit, move the value of the P bit (SFR PSW.0) to TB8_0 (or TB8_1).

The mode 2 baud rate is either *clk*/32 or *clk*/64, as determined by the SMOD0 (or SMOD1) bit. The formula for the mode 2 baud rate is:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}} \times \text{clk}}{64}$$

Mode 2 operation is identical to the standard 8051.

Mode 2 Transmit

Figure 3-13 illustrates the mode 2 transmit timing. Transmission begins after the first rollover of the divide-by-16 counter following a software write to SBUF0 (or SBUF1). The UART shifts data out on the *txd0* (or *txd1*) pin in the following order: start bit, data bits (LSB first), 9th bit, stop bit. The TI_0 (or TI_1) bit is set when the stop bit is placed on the *txd0* (or *txd1*) pin.

Mode 2 Receive

Figure 3-14 illustrates the mode 2 receive timing. Reception begins at the falling edge of a start bit received on *rxd0_in* (or *rxd1_in*), when enabled by the REN_0 (or REN_1) bit. For this purpose, *rxd0_in* (or *rxd1_in*) is sampled 16 times per bit for any baud rate. When a falling edge of a start bit is detected, the divide-by-16 counter used to generate the receive clock is reset to align the counter rollover to the bit boundaries.

For noise rejection, the serial port establishes the content of each received bit by a majority decision of 3 consecutive samples in the middle of each bit time. This is especially true for the start bit. If the falling edge on *rxd0_in* (or *rxd1_in*) is not verified by a majority decision of 3 consecutive samples (low), then the serial port stops reception and waits for another falling edge on *rxd0_in* (or *rxd1_in*).

At the middle of the stop bit time, the serial port checks for the following conditions:

- RI_0 (or RI1) = 0, and
- If SM2_0 (or SM2_1) = 1, the state of the stop bit is 1.
  (If SM2_0 (or SM2_1) = 0, the state of the stop bit doesn't matter.)

If the above conditions are met, the serial port then writes the received byte to the SBUF0 (or SBUF1) register, loads the 9th received bit into RB8_0 (or RB8_1), and sets the RI_0 (or RI_1) bit. If the above conditions are not met, the received data is lost, the SBUF register and RB8 bit are not loaded, and the RI bit is not set. After the middle of the stop bit time, the serial port waits for another high-to-low transition on the (*rxd0_in* or *rxd1_in*) pin.

Figure 3-13 Serial Port 0 Mode 2 Transmit Timing

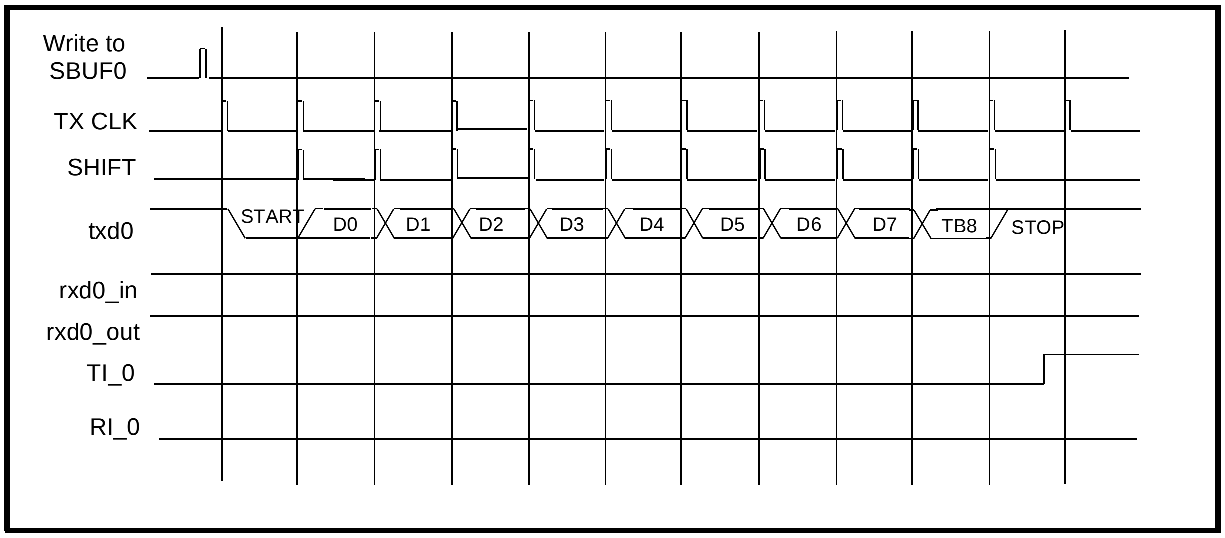
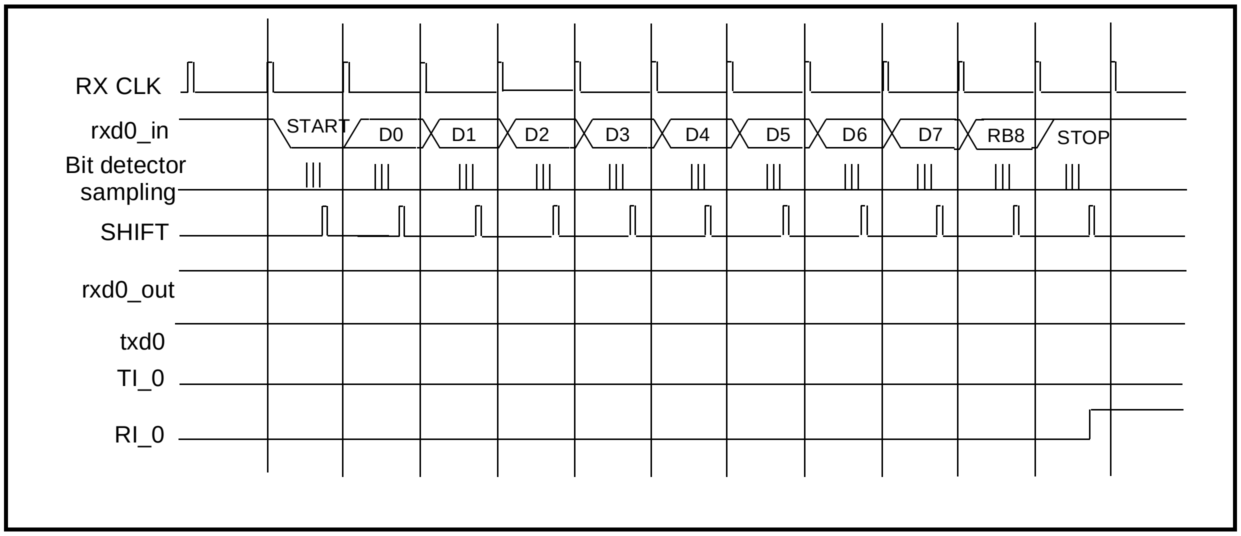


Figure 3-14 Serial Port 0 Mode 2 Receive Timing



Mode 3

Mode 3 provides asynchronous, full-duplex communication, using a total of 11 bits: 1 start bit, 8 data bits, a programmable 9th bit, and 1 stop bit. The data bits are transmitted and received LSB first.

The mode 3 transmit and operations are identical to mode 2. The mode 3 baud rate generation is identical to mode 1. That is, mode 3 is a combination of mode 2 protocol and mode 1 baud rate. Figure 3-15 illustrates the mode 3 transmit timing. Figure 3-16 illustrates the mode 3 receive timing.

Mode 3 operation is identical to that of the standard 8051 when Timers 1 and 2 use *clk*/12 (the default).

Figure 3-15   Serial Port 0 Mode 3 Transmit Timing

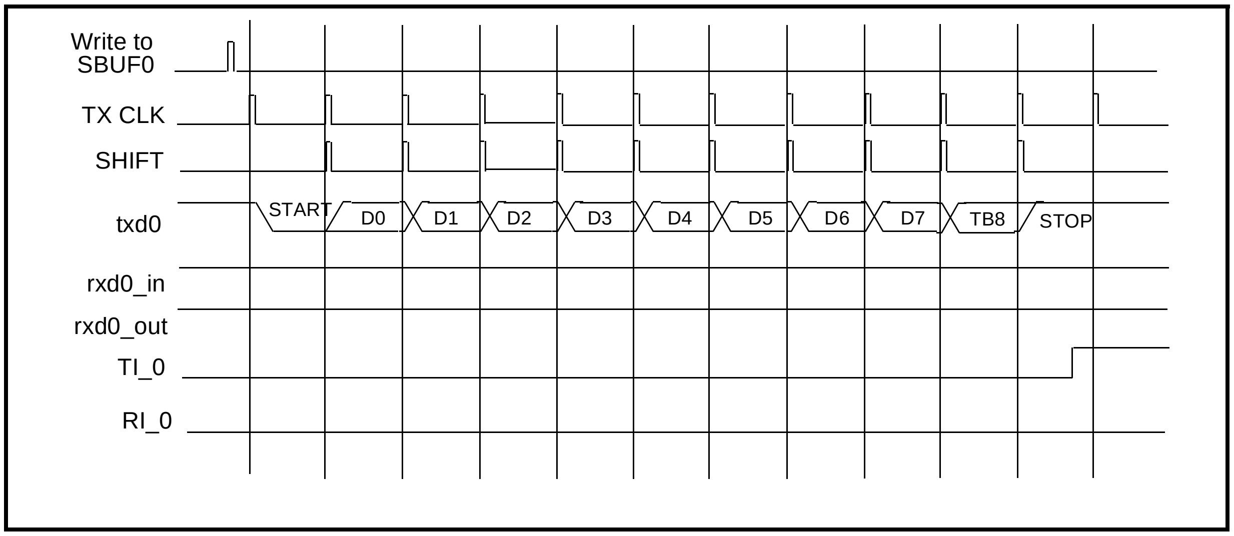
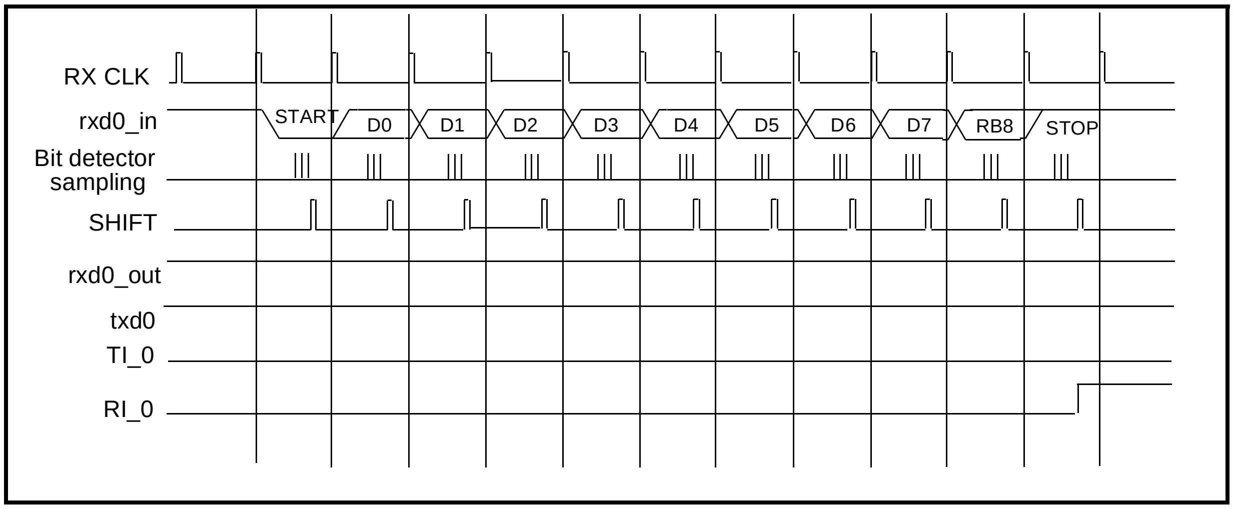


Figure 3-16 Serial Port 0 Mode 3 Receive Timing



Multiprocessor Communications

The multiprocessor communication feature is enabled in modes 2 and 3 when the SM2 bit is set in the SCON SFR for a serial port (SM2_0 for Serial Port 0, SM2_1 for Serial Port 1). In multiprocessor communication mode, the 9th bit received is stored in RB8_0 (or RB8_1) and, after the stop bit is received, the serial port interrupt is activated only if RB8_0 (or RB8_1) = 1.

A typical use for the multiprocessor communication feature is when a master wants to send a block of data to one of several slaves. The master first transmits an address byte that identifies the target slave. When transmitting an address byte, the master sets the 9th bit to 1; for data bytes, the 9th bit is 0.

When SM2_0 (or SM2_1) = 1, no slave will be interrupted by a data byte. However, an address byte interrupts all slaves so that each slave can examine the received address byte to determine whether that slave is being addressed. Address decoding must be done by software during the interrupt service routine. The addressed slave clears its SM2_0 (or SM2_1) bit and prepares to receive the data bytes. The slaves that are not being addressed leave the SM2_0 (or SM2_1) bit set and ignore the incoming data bytes.

SFR Bus Peripheral Interface

The DW8051 accesses internal and external peripherals by using Special Function Registers (SFRs). The internal peripherals include the timer/counters, SFR RAM, interrupt unit, and the optional serial ports.

External SFR Bus

You can directly attach custom designed peripherals to the DW8051 using the SFR interface signals *sfr_data_in[7:0]*, *sfr_data_out[7:0]*, *sfr_addr[7:0]*, *sfr_rd*, and *sfr_wr*. SFR addresses that are not used for DW8051 internal SFRs are available for connecting external SFR peripherals. See Chapter 2 for a list of DW8051 internal SFRs.

The SFR bus is a synchronous 8-bit bus. All registers in the peripherals attached to this bus are memory mapped into the DW8051 SFR address space and can be accessed by DW8051_core in the same manner as any internal SFR.

The SFR bus can only be used for on-chip peripheral blocks. The SFR bus timing does not permit the attachment of off-chip peripheral blocks. The number of peripherals that can be attached to the SFR interface is limited only by the number of free SFR addresses and electrical timing considerations.

There are several benefits when you connect peripherals directly to the SFR bus. Peripheral registers will be mapped into the SFR address space and can take advantage of the DW8051 direct addressing modes. However, DW8051_core only reads *sfr_data_in* when *sfr_addr* does not match the address of any internal SFR. Otherwise, *sfr_data_in* is ignored.

The addresses of standard SFRs that are not built into DW8051_core because the associated internal peripheral has been excluded through parameter settings can be used for external SFR peripherals. However, this can cause code compatibility problems and is therefore not recommended. For example, the SBUF0 SFR is not built when the *serial* parameter = 0, and can therefore be used for an external SFR peripheral. However, software that depends on the SBUF0 SFR at address 99h will not function properly if SFR address 99h is used for an external device.

See Chapter 4 for an example of custom SFR peripheral integration and associated timing diagrams.

Bit Addressing

Bit addressing in SFR space is available at all SFR addresses that end with 0 or 8. For example, the PSW SFR at address D0h is bit-addressable. Table 3-13 lists the bit-addressable SFRs and their usage.

Table 3-13 Bit–Addressable SFRs

SFR Address	Usage
80h	Not present in DW8051_core, available for external SFR peripheral
88h	TCON

Table 3-13 Bit–Addressable SFRs

SFR Address	Usage
90h	Not present in DW8051_core, available for external SFR peripheral
98h	SCON0 when the serial parameter is 1 or 2, not present when serial = 0
A0h	Not present in DW8051_core, available for external SFR peripheral
A8h	IE
B0h	Not present in DW8051_core, available for external SFR peripheral
B8h	IP
C0h	SCON1 when serial = 2, not present when serial = 0 or 1
C8h	T2CON when timer2 = 1, not present when timer2 = 0
D0h	PSW
D8h	EICON
E0h	ACC
E8h	EIE when extd_intr = 1, not present when extd_intr = 0
F0h	B
F8h	EIP when extd_intr = 1, not present when extd_intr = 0

Interrupts

The setting of the user-modifiable parameter *extd_intr* determines whether the DW8051 is built with the 6-source standard interrupt unit or the 13-source extended interrupt unit.

The DW8051 standard interrupt unit (selected when *extd_intr* = 0) supports following interrupt sources:

- *int0_n* – External interrupt, active low, configurable as edge-sensitive or level-sensitive
- *int1_n* – External interrupt, active low, configurable as edge-sensitive or level-sensitive
- TF0 – Timer 0 interrupt
- TF1 – Timer 1 interrupt
- TF2 or EXF2 – Timer 2 interrupt
- TI_0 or RI_0 – Internal receive/transmit interrupt from Serial Port 0

The DW8051 extended interrupt unit (selected when *extd_intr* = 1) supports the 6 standard interrupts plus 7 additional interrupt sources:

- *pfi* – External power-fail interrupt, level-sensitive, active high
- *int2* – External interrupt, edge-sensitive, active high
- *int3_n* – External interrupt, edge-sensitive, active low
- *int4* – External interrupt, edge-sensitive, active high
- *int5_n* – External interrupt, edge-sensitive, active low
- *wdti* – External watchdog-timer interrupt, edge-sensitive, active high
- TI_1 or RI_1 – Internal receive/transmit interrupt from optional Serial Port 1

Interrupts from the serial ports and Timer 2 cannot be forced by software if the corresponding blocks are excluded from DW8051_core through parameter settings. Interrupts from excluded blocks will never occur. The corresponding interrupt enable and priority bits are available but not used.

803x/805x Compatibility

In general, when *extd_intr* = 1, the DW8051 interrupt structure is similar to that of the Dallas Semiconductor DS80C320. Table 3-14 summarizes the differences in interrupt structure between the Intel 8051, the Dallas Semiconductor DS80C320, and the DesignWare DW8051 MacroCell.

Table 3-14 Interrupt Compatibility Summary for Extended Interrupt Unit

Feature	**Intel 8051**	**Dallas DS80C320**	**DW8051**
Power fail interrupt	not implemented	internally generated	externally generated
External interrupt 0	implemented	implemented	implemented
Timer 0 interrupt	implemented	implemented	implemented
External interrupt 1	implemented	implemented	implemented
Timer 1 interrupt	implemented	implemented	implemented
Serial port 0 interrupt	implemented	implemented	implemented when serial > 0
Timer 2 interrupt	not implemented	implemented	implemented when timer2 = 1
Serial port 1 interrupt	not implemented	implemented	implemented when serial = 2
External interrupt 2	not implemented	implemented	implemented
External interrupt 3	not implemented	implemented	implemented

Table 3-14 Interrupt Compatibility Summary for Extended Interrupt Unit

Feature	**Intel 8051**	**Dallas DS80C320**	**DW8051**
External interrupt 4	not implemented	implemented	implemented
External interrupt 5	not implemented	implemented	implemented
Watchdog timer interrupt	not implemented	internally generated	externally generated
Real-time clock interrupt	not implemented	implemented	not implemented

Interrupt SFRs

The following SFRs are associated with interrupt control:

- IE – SFR A8h (Table 3-15)
- IP – SFR B8h (Table 3-16)
- EXIF – SFR 91h (Table 3-17)
- EICON – SFR D8h (Table 3-18)
- EIE – SFR E8h (Table 3-19)
- EIP – SFR F8h (Table 3-20)

The IE and IP SFRs provide interrupt enable and priority control for the standard interrupt unit, as with the standard 8051. Additionally, these SFRs provide control bits for the Serial Port 1 interrupt. These bits (ES1 and PS1) are available only when the extended interrupt unit is implemented (*extd_intr* = 1). Otherwise, they are read as 0.

Bits ES0, ES1, ET2, PS0, PS1, and PT2 are present, but not used, when the corresponding module is not implemented.

The EXIF, EICON, EIE, and EIP registers provide flags, enable control, and priority control for the optional extended interrupt unit.

Table 3-15 IE Register – SFR A8h

Bit	**Function**
IE.7	EA – Global interrupt enable. Controls masking of all interrupts except power-fail interrupt (*pfi*). EA = 0 disables all interrupts (EA overrides individual interrupt enable bits). When EA = 1, each interrupt is enabled or masked by its individual enable bit.
IE.6	ES1 – Enable Serial Port 1 interrupt. ES1 = 0 disables Serial Port 1 interrupts (TI_1 and RI_1). ES1 = 1 enables interrupts generated by the TI_1 or RI_1 flag. ES1 is available only when the extended interrupt unit is implemented (*extd_intr* = 1). Otherwise, it is read as 0. If the extended interrupt unit is implemented and Serial Port 1 is not implemented (*serial* < 2), ES1 is present but not used.
IE.5	ET2 – Enable Timer 2 interrupt. ET2 = 0 disables Timer 2 interrupt (TF2). ET2 = 1 enables interrupts generated by the TF2 or EXF2 flag. If Timer 2 is not implemented (timer2 = 0), ET2 is present but not used.
IE.4	ES0 – Enable Serial Port 0 interrupt. ES0 = 0 disables Serial Port 0 interrupts (TI_0 and RI_0). ES0 = 1 enables interrupts generated by the TI_0 or RI_0 flag. If Serial Port 0 is not implemented (serial = 0), ES0 is present but not used.
IE.3	ET1 – Enable Timer 1 interrupt. ET1 = 0 disables Timer 1 interrupt (TF1). ET1 = 1 enables interrupts generated by the TF1 flag.
IE.2	EX1 – Enable external interrupt 1. EX1 = 0 disables external interrupt 1 (int1_n). EX1 = 1 enables interrupts generated by the int1_n pin.
IE.1	ET0 – Enable Timer 0 interrupt. ET0 = 0 disables Timer 0 interrupt (TF0). ET0 = 1 enables interrupts generated by the TF0 flag.
IE.0	EX0 – Enable external interrupt 0. EX0 = 0 disables external interrupt 0 (int0_n). EX0 = 1 enables interrupts generated by the int0_n pin.

Table 3-16 IP Register – SFR B8h

Bit	Function
IP.7	Reserved. Read as 1.
IP.6	PS1 – Serial Port 1 interrupt priority control. PS1 = 0 sets Serial Port 1 interrupt (TI_1 or RI_1) to low priority. PS1 = 1 sets Serial Port 1 interrupt to high priority. PS1 is available only when the extended interrupt unit is implemented (extd_intr = 1). Otherwise, it is read as 0. If the extended interrupt unit is implemented and Serial Port 1 is not implemented (serial < 2), PS1 is present but not used.
IP.5	PT2 – Timer 2 interrupt priority control. PT2 = 0 sets Timer 2 interrupt (TF2) to low priority. PT2 = 1 sets Timer 2 interrupt to high priority. If Timer 2 is not implemented (timer2 = 0), PT2 is present but not used.
IP.4	PS0 – Serial Port 0 interrupt priority control. PS0 = 0 sets Serial Port 0 interrupt (TI_0 or RI_0) to low priority. PS0 = 1 sets Serial Port 0 interrupt to high priority. If Serial Port 0 is not implemented (serial = 0), PS0 is present but not used.
IP.3	PT1 – Timer 1 interrupt priority control. PT1 = 0 sets Timer 1 interrupt (TF1) to low priority. PT1 = 1 sets Timer 1 interrupt to high priority.
IP.2	PX1 – External interrupt 1 priority control. PX1 = 0 sets external interrupt 1 (int1_n) to low priority. PT1 = 1 sets external interrupt 1 to high priority.
IP.1	PT0 – Timer 0 interrupt priority control. PT0 = 0 sets Timer 0 interrupt (TF0) to low priority. PT0 = 1 sets Timer 0 interrupt to high priority.
IP.0	PX0 – External interrupt 0 priority control. PX0 = 0 sets external interrupt 0 (int0_n) to low priority. PT0 = 1 sets external interrupt 0 to high priority.

Table 3-17 EXIF Register – SFR 91h

Bit	Function
EXIF.7	IE5 – External interrupt 5 flag. IE5 = 1 indicates that a falling edge was detected at the int5_n pin. IE5 must be cleared by software. Setting IE5 in software generates an interrupt, if enabled.

Table 3-17 EXIF Register – SFR 91h

Bit	Function
EXIF.6	IE4 – External interrupt 4 flag. IE4 = 1 indicates that a rising edge was detected at the int4 pin. IE4 must be cleared by software. Setting IE4 in software generates an interrupt, if enabled.
EXIF.5	IE3 – External interrupt 3 flag. IE3 = 1 indicates that a falling edge was detected at the int3_n pin. IE3 must be cleared by software. Setting IE3 in software generates an interrupt, if enabled.
EXIF.4	IE2 – External interrupt 2 flag. IE2 = 1 indicates that a rising edge was detected at the int2 pin. IE2 must be cleared by software. Setting IE2 in software generates an interrupt, if enabled.
EXIF.3	Reserved. Read as 1.
EXIF.2–0	Reserved. Read as 0.

Table 3-18 EICON Register – SFR D8h

Bit	Function
EICON.7	SMOD1 – Serial Port 1 baud rate doubler enable. When SMOD1 = 1, the baud rate for Serial Port 1 is doubled.
EICON.6	Reserved. Read as 1.
EICON.5	EPFI – Enable power-fail interrupt. EPFI = 0 disables power-fail interrupt (pfi). EPFI = 1 enables interrupts generated by the pfi pin.
EICON.4	PFI – Power-fail interrupt flag. PFI = 1 indicates a power-fail interrupt was detected at the pfi pin. PFI must be cleared by software before exiting the interrupt service routine. Otherwise, the interrupt occurs again. Setting PFI in software generates a power-fail interrupt, if enabled.
EICON.3	WDTI – Watchdog timer interrupt flag. WDTI = 1 indicates a watchdog timer interrupt was detected at the wdti pin. WDTI must be cleared by software before exiting the interrupt service routine. Otherwise, the interrupt occurs again. Setting WDTI in software generates a watchdog timer interrupt, if enabled.

Table 3-18 EICON Register – SFR D8h

Bit	**Function**
EICON.2–0	Reserved. Read as 0.

Table 3-19 EIE Register – SFR E8h

Bit	**Function**
EIE.7–5	Reserved. Read as 1.
EIE.4	EWDI – Enable watchdog timer interrupt. EWDI = 0 disables watchdog timer interrupt (wdti). EWDI = 1 enables interrupts generated by wdti pin.
EIE.3	EX5 – Enable external interrupt 5. EX5 = 0 disables external interrupt 5 (int5_n). EX5 = 1 enables interrupts generated by the int5_n pin.
EIE.2	EX4 – Enable external interrupt 4. EX4 = 0 disables external interrupt 4 (int4). EX4 = 1 enables interrupts generated by the int4 pin.
EIE.1	EX3 – Enable external interrupt 3. EX3 = 0 disables external interrupt 3 (int3_n). EX3 = 1 enables interrupts generated by the int3_n pin.
EIE.0	EX2 – Enable external interrupt 2. EX2 = 0 disables external interrupt 2 (int2). EX2 = 1 enables interrupts generated by the int2 pin.

Table 3-20 EIP Register – SFR F8h

Bit	**Function**
EIP.7–5	Reserved. Read as 1.
EIP.4	PWDI – Watchdog timer interrupt priority control. WDPI = 0 sets watchdog timer interrupt (wdti) to low priority. PS0 = 1 sets watchdog timer interrupt to high priority.

Table 3-20 EIP Register – SFR F8h

Bit	**Function**
EIP.3	PX5 – External interrupt 5 priority control. PX5 = 0 sets external interrupt 5 (int5_n) to low priority. PX5 = 1 sets external interrupt 5 to high priority.
EIP.2	PX4 – External interrupt 4 priority control. PX4 = 0 sets external interrupt 4 (int4) to low priority. PX4 = 1 sets external interrupt 4 to high priority.
EIP.1	PX3 – External interrupt 3 priority control. PX3 = 0 sets external interrupt 3 (int3_n) to low priority. PX3 = 1 sets external interrupt 3 to high priority.
EIP.0	PX2 – External interrupt 2 priority control. PX2 = 0 sets external interrupt 2 (int2) to low priority. PX2 = 1 sets external interrupt 2 to high priority.

Interrupt Processing

When an enabled interrupt occurs, the CPU vectors to the address of the interrupt service routine (ISR) associated with that interrupt, as listed in Table 3-21. The CPU executes the ISR to completion unless another interrupt of higher priority occurs. Each ISR ends with a `RETI` (return from interrupt) instruction. After executing the `RETI`, the CPU returns to the next instruction that would have been executed if the interrupt had not occurred.

An ISR can only be interrupted by a higher priority interrupt. That is, an ISR for a low-level interrupt can only be interrupted by high-level interrupt. An ISR for a high-level interrupt can only be interrupted by the power-fail interrupt (extended interrupt unit only).

The DW8051 always completes the instruction in progress before servicing an interrupt. If the instruction in progress is `RETI`, or a write access to any of the IP, IE, EIP, or EIE SFRs, the DW8051 completes one additional instruction before servicing the interrupt.

Interrupt Masking

The EA bit in the IE SFR (IE.7) is a global enable for all interrupts except the power-fail interrupt. When EA = 1, each interrupt is enabled/masked by its individual enable bit. When EA = 0, all interrupts are masked. The only exception is the power-fail interrupt, which is not affected by the EA bit. When EPFI = 1, the power-fail interrupt is enabled, regardless of the state of the EA bit.

Table 3-22 provides a summary of interrupt sources, flags, enables, and priorities.

Table 3-21 Interrupt Natural Vectors and Priorities

Interrupt	**Description**	**Natural Priority**	**Interrupt Vector**
pfi	Power fail interrupt	0	33h
int0_n	External interrupt 0	1	03h
TF0	Timer 0 interrupt	2	0Bh
int1_n	External interrupt 1	3	13h
TF1	Timer 1 interrupt	4	1Bh
TI_0 or RI_0	Serial Port 0 transmit or receive	5	23h
TF2 or EXF2	Timer 2 interrupt	6	2Bh
TI_1 or RI_1	Serial Port 1 transmit or receive	7	3Bh
int2	External interrupt 2	8	43h
int3_n	External interrupt 3	9	4Bh
int4	External interrupt 4	10	53h
int5_n	External interrupt 5	11	5Bh
wdti	Watchdog timer interrupt	12	63h

Interrupt Priorities

There are two stages of interrupt priority assignment: interrupt level and natural priority. The interrupt level (highest, high, or low) takes precedence over natural priority. The power-fail interrupt, if enabled, always has highest priority and is the only interrupt that can have highest priority. All other interrupts can be assigned either high or low priority.

In addition to an assigned priority level (high or low), each interrupt has a natural priority, as listed in Table 3-21. Simultaneous interrupts with the same priority level (for example, both high) are resolved according to their natural priority. For example, if *int0_n* and *int2* are both programmed as high priority, *int0_n* takes precedence.

Once an interrupt is being serviced, only an interrupt of higher priority level can interrupt the service routine of the interrupt currently being serviced.

Table 3-22 Interrupt Flags, Enables, and Priority Control

Interrupt	**Description**	**Flag**	**Enable**	**Priority Control**
pfi	Power fail interrupt	EICON.4	EICON.5	N/A
int0_n	External interrupt 0	TCON.1	IE.0	IP.0
TF0	Timer 0 interrupt	TCON.5	IE.1	IP.1
int1_n	External interrupt 1	TCON.3	IE.2	IP.2
TF1	Timer 1 interrupt	TCON.7	IE.3	IP.3
TI_0 or RI_0	Serial Port 0 transmit or receive	SCON0.0 (RI_0), SCON0.1 (TI_0)	IE.4	IP.4
TF2 or EXF2	Timer 2 interrupt	T2CON.7 (TF2), T2CON.6 (EXF2)	IE.5	IP.5

Table 3-22 Interrupt Flags, Enables, and Priority Control

Interrupt	**Description**	**Flag**	**Enable**	**Priority Control**
TI_1 or RI_1	Serial Port 1 transmit or receive	SCON1.0 (RI_1), SCON1.1 (TI_1)	IE.6	IP.6
int2	External interrupt 2	EXIF.4	EIE.0	EIP.0
int3_n	External interrupt 3	EXIF.5	EIE.1	EIP.1
int4	External interrupt 4	EXIF.6	EIE.2	EIP.2
int5_n	External interrupt 5	EXIF.7	EIE.3	EIP.3
wdti	Watchdog timer interrupt	EICON.3	EIE.4	EIP.4

Interrupt Sampling

The internal timers and serial ports generate interrupts by setting their respective SFR interrupt flag bits. The DW8051 samples external interrupts once per instruction cycle, at the rising edge of *clk* at the end of cycle C4.

int0_n and int1_n are both active low and can be programmed to be either edge-sensitive or level-sensitive, through the IT0 and IT1 bits in the TCON SFR. For example, when IT0 = 0, *int0_n* is level-sensitive and the DW8051 sets the IE0 flag when the *int0_n* pin is sampled low. When IT0 = 1, *int0_n* is edge-sensitive and the DW8051 sets the IE0 flag when the *int0_n* pin is sampled high then low on consecutive samples.

The remaining four external interrupts are edge-sensitive only. *int2* and *int4* are active high, *int3_n* and *int5_n* are active low.

The power-fail (*pfi*) and watchdog timer (*wdti*) interrupts are active high and sampled once per instruction cycle. The power-fail interrupt is level-sensitive, and the watchdog timer interrupt is edge-sensitive.

To ensure that edge-sensitive interrupts are detected, the corresponding ports should be held high for 4 *clk* cycles and then low for 4 *clk* cycles. Level-sensitive interrupts are not latched and must remain active until serviced.

Interrupt Latency

Interrupt response time depends on the current state of the DW8051. The fastest response time is 5 instruction cycles: 1 to detect the interrupt, and 4 to perform the *LCALL* to the ISR.

The maximum latency (13 instruction cycles) occurs when the DW8051 is currently executing a `RETI` instruction followed by a `MUL` or `DIV` instruction. The 13 instruction cycles in this case are: 1 to detect the interrupt, 3 to complete the `RETI`, 5 to execute the `DIV` or `MUL`, and 4 to execute the `LCALL` to the ISR. For the maximum latency case, the response time is 13 x 4 = 52 *clk* cycles.

Single-Step Operation

The DW8051 interrupt structure provides a way to perform single-step program execution. When exiting an ISR with an `RETI` instruction, the DW8051 will always execute at least one instruction of the task program. Therefore, once an ISR is entered, it cannot be re-entered until at least one program instruction is executed.

To perform single-step execution, program one of the external interrupts (for example, *int0_n*) to be level-sensitive and write an ISR for that interrupt that terminates as follows:

```
JNB  TCON.1,$   ; wait for high on int0_n
JB   TCON.1,$   ; wait for low on int0_n
RETI            ; return for ISR
```

The CPU enters the ISR when *int0_n* goes low, then waits for a pulse on *int0_n*. Each time *int0_n* is pulsed, the CPU exits the ISR, executes one program instruction, then re-enters the ISR.

Reset

The DW8051 provides two reset inputs, *por_n* and *rst_in_n*. *por_n* is the power-on reset input. *rst_in_n* provides the functionality of the standard 8051 RST input.

For either external reset source, the DW8051 remains in the reset state until the external reset signal is removed. Both sources of reset initialize the SFRs to their reset values, as listed in Table 2-7. The internal RAM is not affected by either *por_n* or *rst_in_n*. When the activated reset signal is removed, the DW8051 exits the reset state and begins program execution at the standard reset vector address 0000h.

Power On Reset

The por_n input must be driven low for at least two clk cycles to ensure proper initialization. There is no need to externally synchronize the *por_n* signal to *clk*. An internal 2-stage synchronizer synchronizes *por_n* to clk.

Asserting *por_n* immediately aborts the current operation, forces all internal logic into the reset state, and activates the *rst_out_n* signal. The *rst_out_n* signal is used internally to reset all modules and can be used to reset externally connected hardware.

The DW8051 exits the internal reset state and deasserts the *rst_out_n* signal at least 1 *clk* cycle after the release of *por_n* and always on the rising edge of *clk*. CPU operation starts 1.5 *clk* cycles after the release of *rst_out_n*. Figure 3-17 illustrates the power-on reset timing.

Standard Reset

rst_in_n provides the same functionality as the standard 8051 RST input, with inverse polarity. The DW8051 always samples *rst_in_n* in cycle 4 (C4) of the 4-*clk* instruction cycle, and goes into the reset state if 2 consecutive samples of *rst_in_n* are low. Therefore, *rst_in_n* must be asserted (active low) for at least 2 instruction cycles (8 clk cycles). Shorter pulses on *rst_in_n* may be ignored.

The DW8051 activates *rst_out_n* and resets internal hardware at the end of the C4 cycle in which *rst_in_n* is sampled low for the second time. Figure 3-18 illustrates the *rst_in_n* assertion timing.

The DW8051 releases *rst_out_n* at the end of the second C4 cycle in which *rst_in_n* is sampled high. The CPU begins execution at address 0000h, 1.5 *clk* cycles after the release of *rst_out_n*. Figure 3-19 illustrates the *rst_in_n* deassertion timing.

Figure 3-17 Timing of por_n Reset

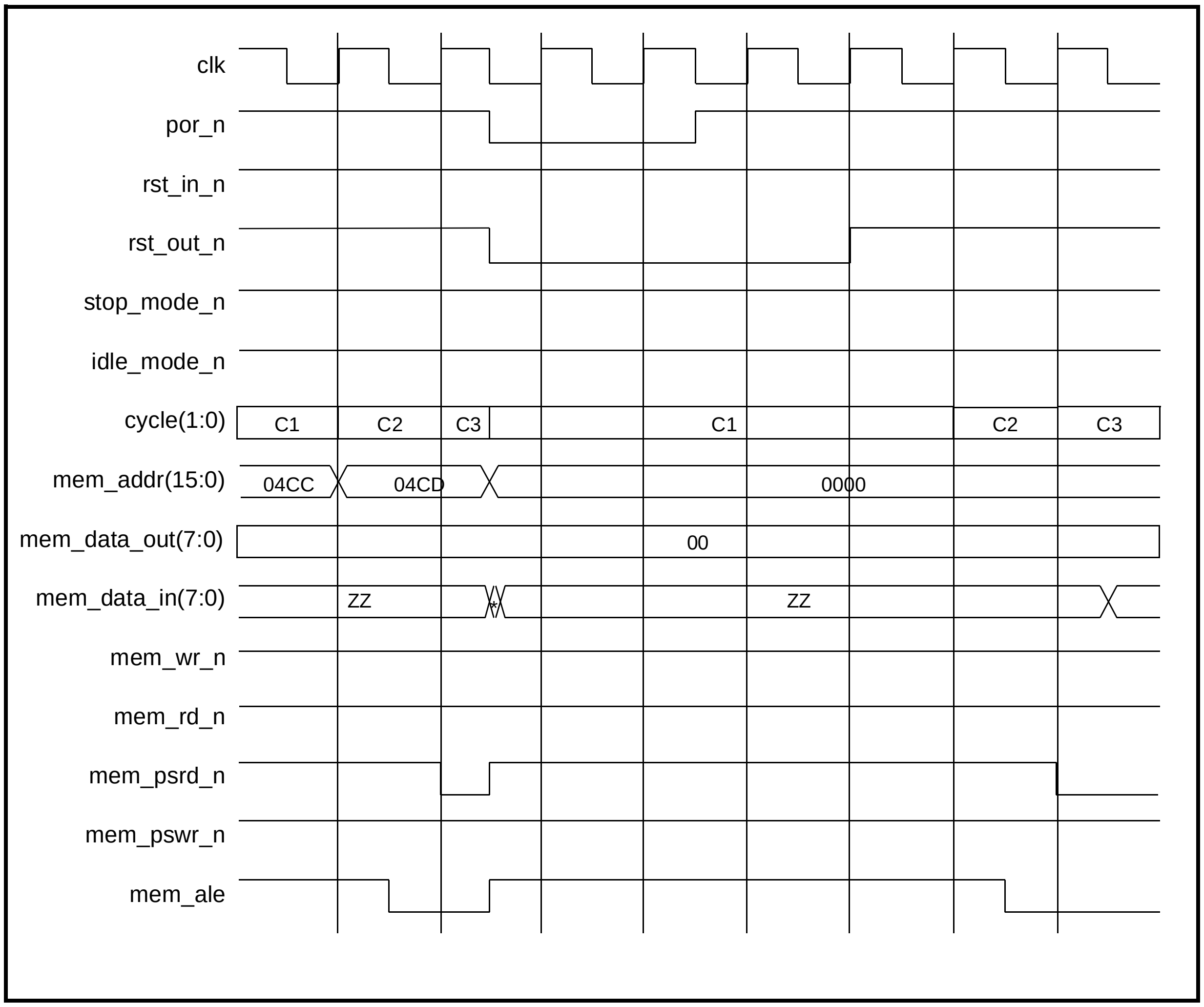


Figure 3-18   Assertion of rst_in_n

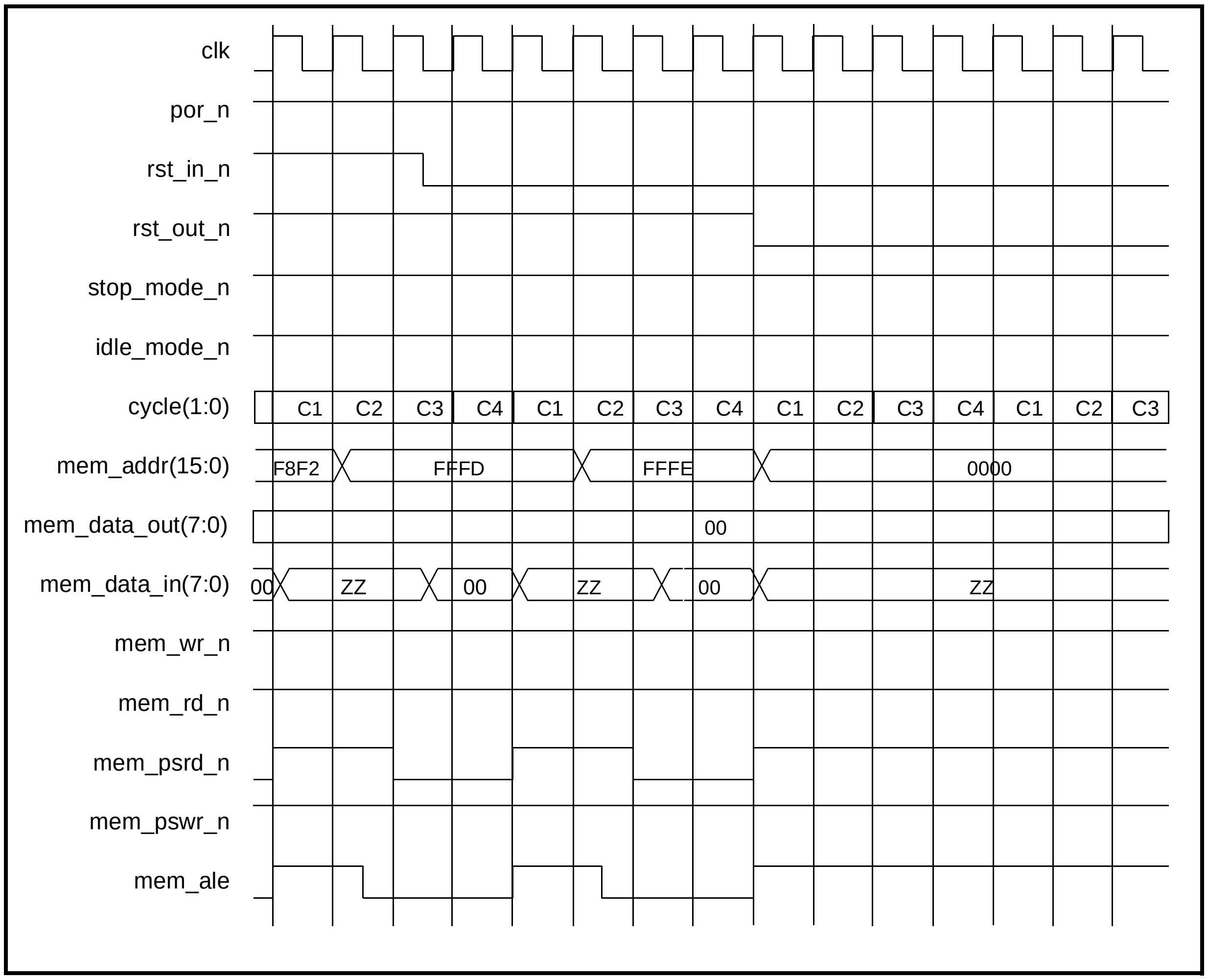
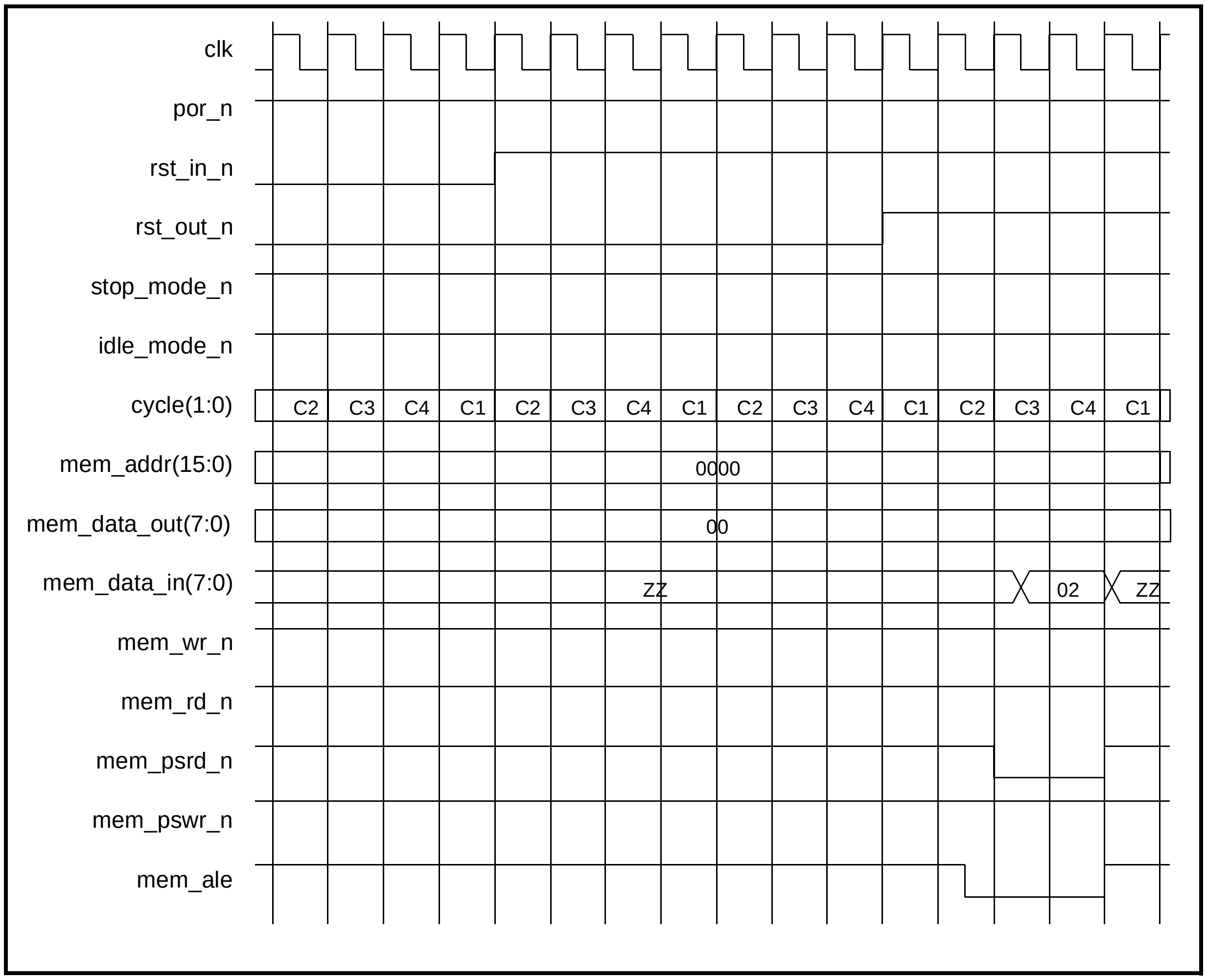


Figure 3-19   Deassertion of rst_in_n



Power Saving Modes

The DW8051 provides two power saving modes: idle mode and stop mode. Table 3-23 summarizes the differences in power saving features between the Intel 8051, the Dallas Semiconductor DS80C320, and the DesignWare DW8051 MacroCell.

The bits that control entry into idle and stop modes are in the PCON register at SFR address 87h (see Table 3-24).

Table 3-23 Power Saving Modes Compatibility Summary

Feature	**Intel 8051**	**Dallas DS80C320**	**DW8051**
Idle mode	Clock gated internally. Exit idle mode by interrupt or reset.	Clock gated internally. Exit idle mode by interrupt or reset.	Clock not gated internally. Exit idle mode by interrupt or reset.
Stop mode	Clock gated internally. Exit stop mode by interrupt or reset.	Clock gated internally. Exit stop mode by interrupt or reset.	Clock not gated internally. Exit stop mode by power-on reset only.
Power management modes	not implemented	implemented	not implemented
Ring oscillator select	not implemented	implemented	not implemented
Band Gap select	not implemented	implemented	not implemented

Idle Mode

An instruction that sets the IDLE bit (PCON.0) causes the DW8051 to enter idle mode when that instruction completes. In idle mode, CPU processing is suspended, internal registers maintain their current data, and the *idle_mode_n* output is activated. However, unlike the standard 8051, the *clk* is not disabled internally.

Figure 3-20 illustrates the timing relationships of DW8051_core signals when entering idle mode.

There are three ways to exit idle mode: activate any enabled interrupt, *por_n*, or *rst_in_n*. Activation of any enabled interrupt causes the hardware to clear the IDLE bit and terminate idle mode. The CPU executes the ISR associated with the received interrupt. The `RETI` instruction at the end of the of ISR returns the CPU to the instruction following the one that put the DW8051 into idle mode.

Figure 3-21 illustrates the timing relationships of DW8051_core signals when exiting idle mode due to an interrupt.

Activating either *por_n* or *rst_in_n* causes the DW8051 to exit idle mode, reset internal modules, and begin program execution at the standard reset vector address 0000h. See the reset timing diagrams for timing information.

Table 3-24 PCON Register – SFR 87h

Bit	**Function**
PCON.7	SMOD0 – Serial Port 0 baud rate doubler enable. When SMOD0 = 1, the baud rate for Serial Port 0 is doubled.
PCON.6–4	Reserved.
PCON.3	GF1 – General purpose flag 1. Bit-addressable, general purpose flag for software control.
PCON.2	GF0 – General purpose flag 0. Bit-addressable, general purpose flag for software control.
PCON.1	STOP – Stop mode select. Setting the STOP bit places the DW8051 in stop mode.
PCON.0	IDLE – Idle mode select. Setting the IDLE bit places the DW8051 in idle mode.

Stop Mode

An instruction that sets the STOP bit (PCON.1, see Table 3-24) causes the DW8051 to enter stop mode when that instruction completes. In stop mode, CPU processing is suspended, internal registers maintain their current data, and the *stop_mode_n* output is activated. However, unlike the standard 8051, the *clk* is not disabled internally. The *clk* source must be removed externally.

In stop mode, the internal cycle counter is reset. Because most internal operations are controlled by the cycle counter, internal flip-flops do not change state in stop mode and, therefore, there is a significant reduction in power consumption.

The only way to exit stop mode is by asserting *por_n*. The DW8051 executes its reset sequence and begins program execution at the standard reset vector address 0000h.

Figure 3-22 illustrates the DW8051 stop mode timing.

Figure 3-20 Idle Mode Entry Timing

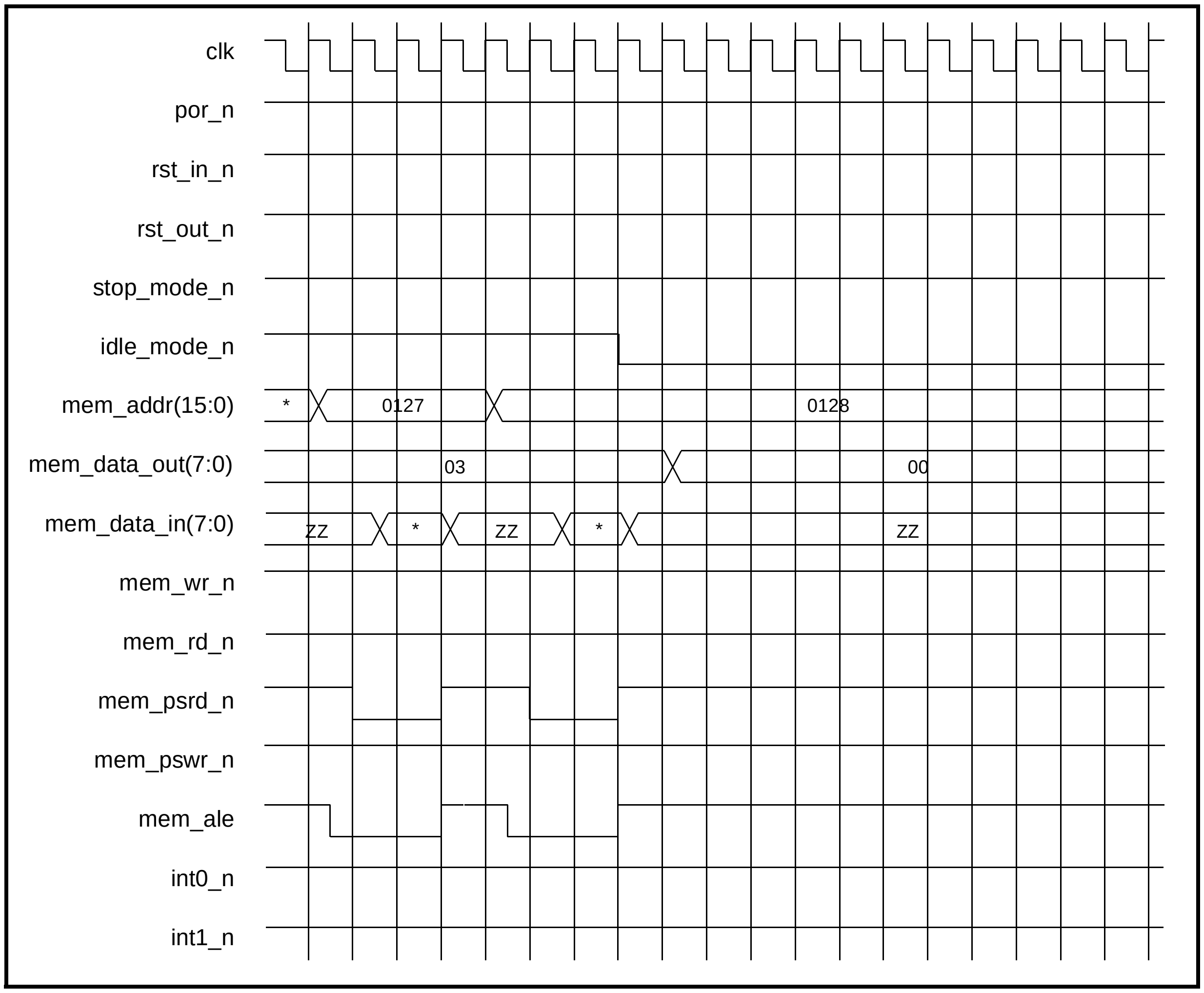


Figure 3-21   Idle Mode Exit Timing

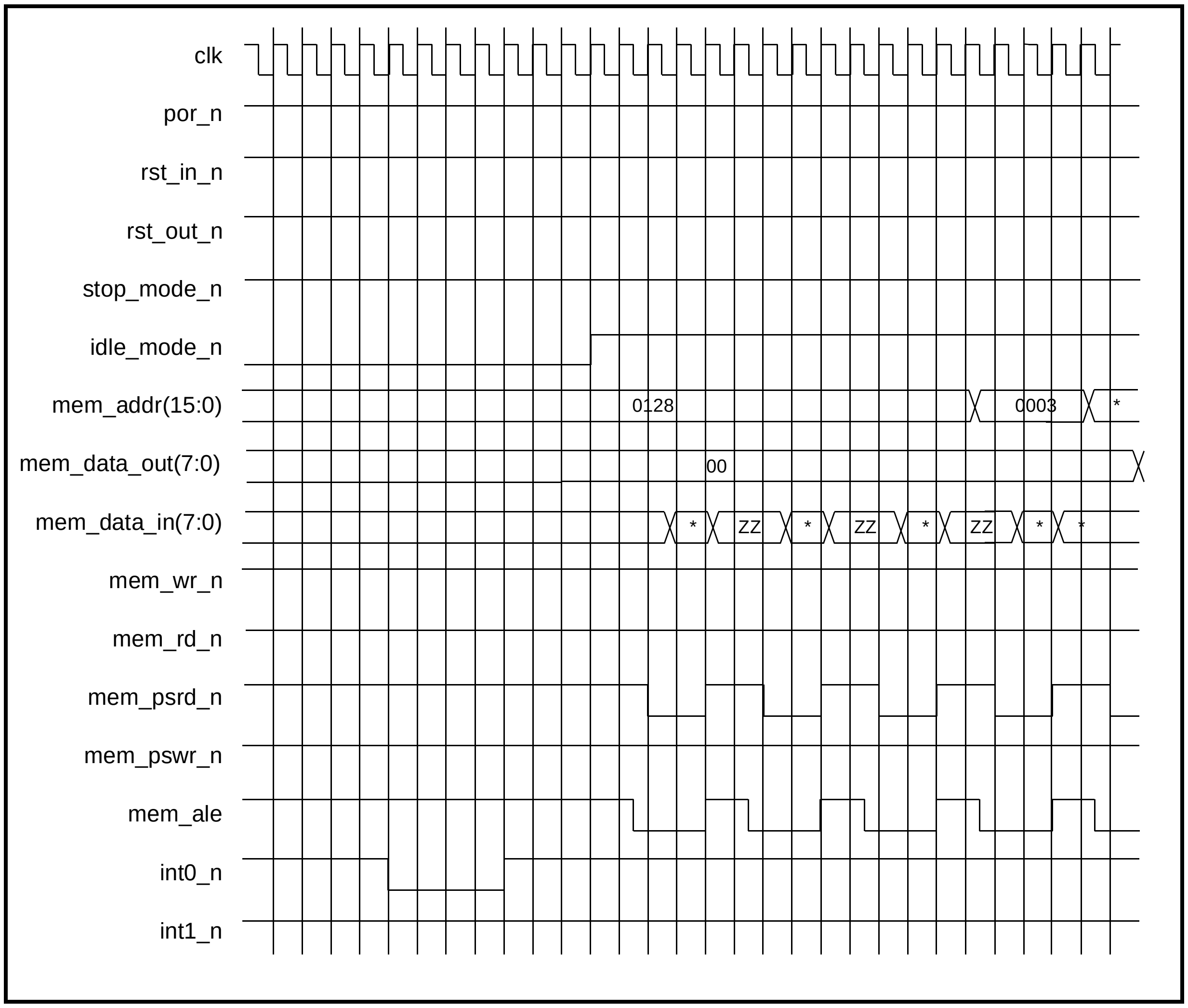
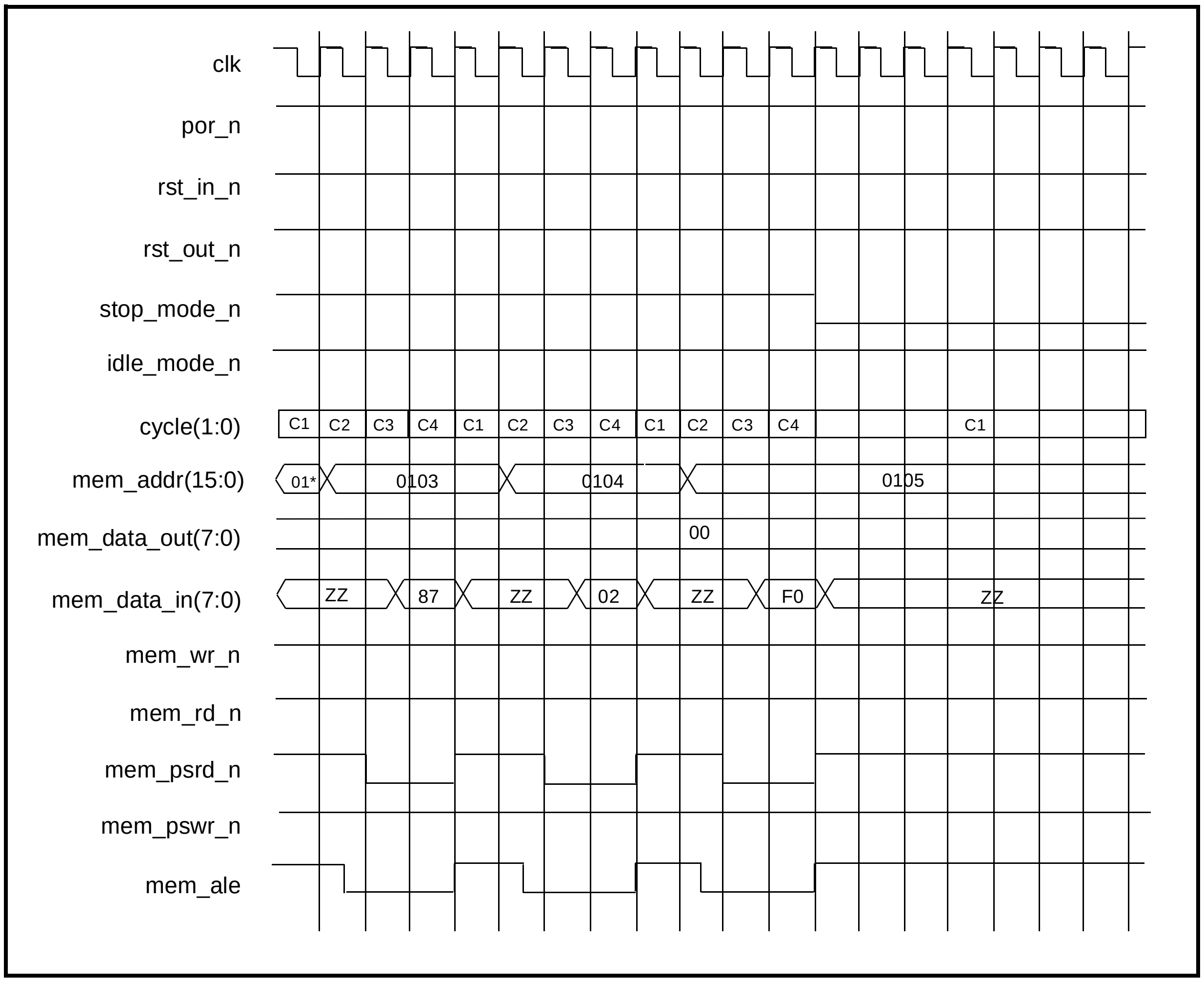


Figure 3-22   Stop Mode Timing



4

DW8051 User Guide

The DW8051 MacroCell Solution includes a complete development environment that includes the DW8501 MacroCell, test suite, and example design. The DW8051 MacroCell Solution also includes coreConsultant for automated installation, configuration, simulation, and synthesis. Implementing the DW8051 in your application design involves the following tasks:

- Installing the DW8051 Core Kit
- Creating a Workspace
- Specifying Your Configuration
- Simulating the DW8051 MacroCell
- Synthesizing the DW8051 MacroCell
- Confirming Your Gate-Level Design
- Integrating the DW8051 into Your ASIC Design

- Reading Designs Back in After Layout
- Manufacturing Test
- Software Development and Debugging

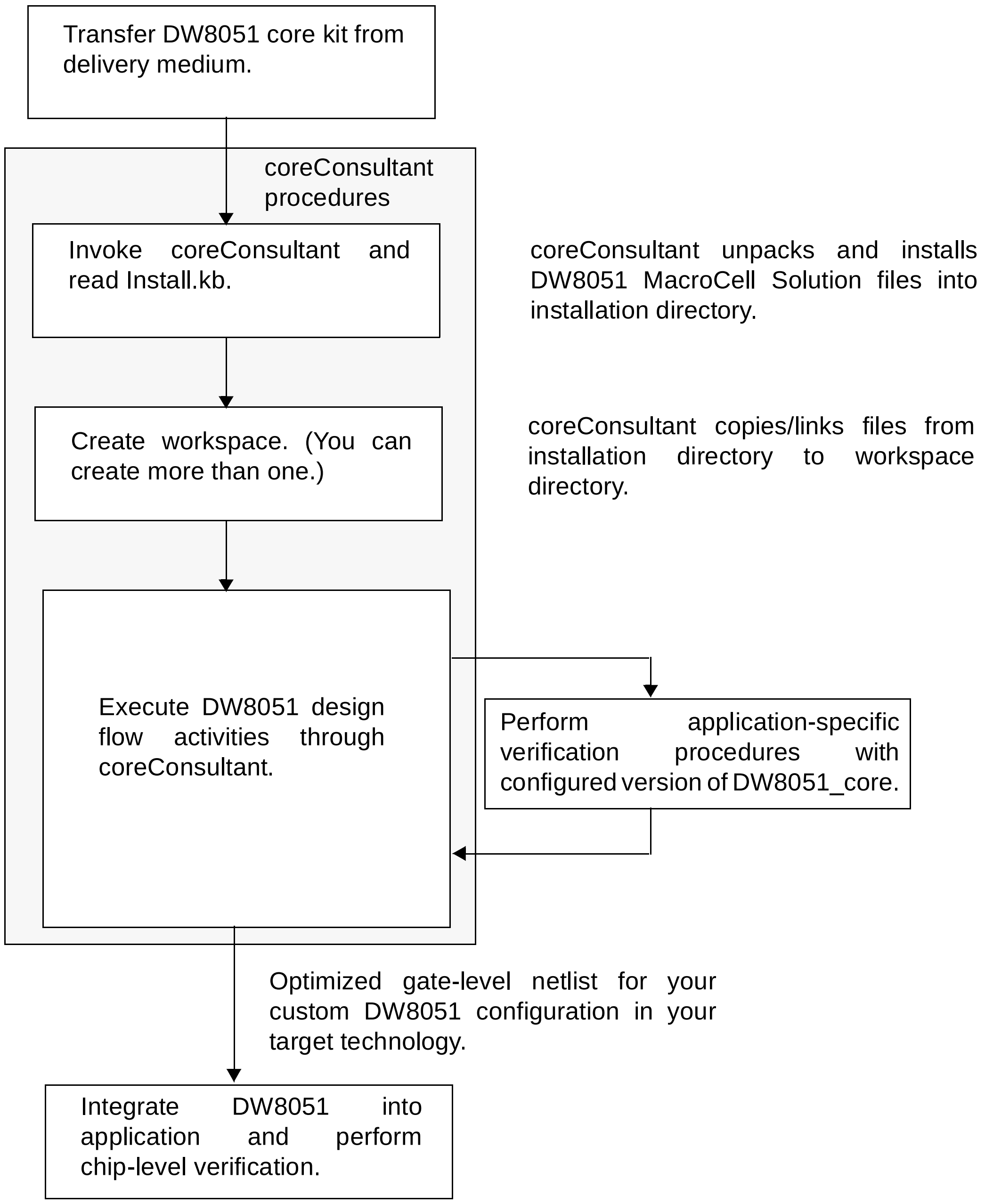
Basic Design Flow

coreConsultant is your interface to the DW8051 MacroCell, as illustrated in Figure 4-1. You use coreConsultant to install the DW8051 MacroCell Solution core kit. Then, you follow the coreConsultant design flow to configure, simulate, and synthesize your custom implementation(s) of the DW8051 MacroCell.

The input to the coreConsultant design flow is the DW8051 MacroCell Solution core kit. The final output from the coreConsultant design flow is an optimized gate-level netlist for your custom DW8051 MacroCell configuration, in your target technology.

The coreConsultant GUI guides you through the DW8051 MacroCell design flow in your workspace. The DW8051 MacroCell design flow includes the default set of coreConsultant design activities described in the *coreConsultant User Guide*, plus additional activities that are specific to DW8051 simulation.

Figure 4-1 DW8051 MacroCell Design Flow


Transfer DW8051 core kit from delivery medium.
coreConsultant procedures
Invoke coreConsultant and read Install.kb.
coreConsultant unpacks and installs DW8051 MacroCell Solution files into installation directory.
Create workspace. (You can create more than one.)
coreConsultant copies/links files from installation directory to workspace directory.
Execute DW8051 design flow activities through coreConsultant.
Perform application-specific verification procedures with configured version of DW8051_core.
Optimized gate-level netlist for your custom DW8051 configuration in your target technology.
Integrate DW8051 into application and perform chip-level verification.

You execute the entire DW8051 design flow through coreConsultant, as shown in Figure 4-1, except for the following application-specific activities:

- Integrating the DW8051 MacroCell into your application design.
- Verifying the DW8051 in the context of your application design. The DW8051 design flow includes steps to verify your custom configured implementation of the DW8051 as a standalone block, using the DW8051 test suite. You should also verify the DW8051 in the context of your application design, according to your design verification strategy.
- Creating custom test programs and operating the DW8051 test suite directly, if you choose to do so. coreConsultant enables you to select and execute any or all of the supplied test programs. If you want to execute custom test programs that you create, you must operate the test suite directly, outside of the coreConsultant environment, as described in Chapter 5.

This user guide describes how to perform design flow activities that are specific to the DW8051 MacroCell. For details about the default coreConsultant design flow activities, refer to the *coreConsultant User Guide*.

Installing the DW8051 Core Kit

The DW8051 MacroCell Solution is delivered as a core kit, which is a set of files in the Synopsys coreConsultant knowledge database (.kb) format. Installing the DW8051 MacroCell Solution core kit is a two-step process :

1. Transfer the DW8051 core kit from the delivery medium to your local disk. The delivery medium is either CD-ROM or electronic software transfer (EST). In either case, follow the supplied instructions to transfer the core kit to your local disk.
2. Invoke coreConsultant and read the file Install.kb from the DW8051 MacroCell Solution core kit.

   Install.kb contains the knowledge that coreConsultant needs to properly unpack and install the DW8051 MacroCell Solution into your selected installation directory.

Installation Directories and Files – VHDL Version

When coreConsultant completes the installation of the VHDL version of the DW8051 MacroCell Solution, your selected installation directory contains the directories and files described in Table 4-1 and Table 4-2.

Figure 4-2 shows the directory structure for the VHDL (DW8051 and DW8051-Source) versions of the DW8051 MacroCell Solution. In Figure 4-2, the selected installation directory name is dw8051.

Table 4-1 DW8051 MacroCell Directory Structure – VHDL Version

Directory	Contents
dw8051/asm_tests	Test programs

Table 4-1 DW8051 MacroCell Directory Structure – VHDL Version

Directory	**Contents**
dw8051/cba	Cell Based Array (CBA) technology library
dw8051/doc	A PDF version of this databook. If you do not already have the Adobe Acrobat PDF reader installed, you can download it free from www.adobe.com.
dw8051/example	An example 8032 implementation using the DW8051
dw8051/kb	coreConsultant knowledge database (KB) files
dw8051/packages	VHDL package files needed for DW8051_core
dw8051/quickmap	Script to generate a GTECH simulation model for DW8051_core
dw8051 /sim_reference	Golden reference for output trace, strobe, and RAM contents files from simulation of the DS80C320 on a hardware modeler
dw8051/src	VHDL source files. In the DW8051 version, the VHDL source files for DW8051_core and its submodules are encrypted. In the DW8051-Source version, the VHDL source files for DW8051_core and its submodules are not encrypted.
dw8051/ scan_chain_example	Example Test Compiler scan chain insertion script for DW8051_core
dw8051/testbench	VHDL source files for the DW8051 testbench, plus the testbench command file and simulation scripts

Figure 4-2 Directory Structure – VHDL Version

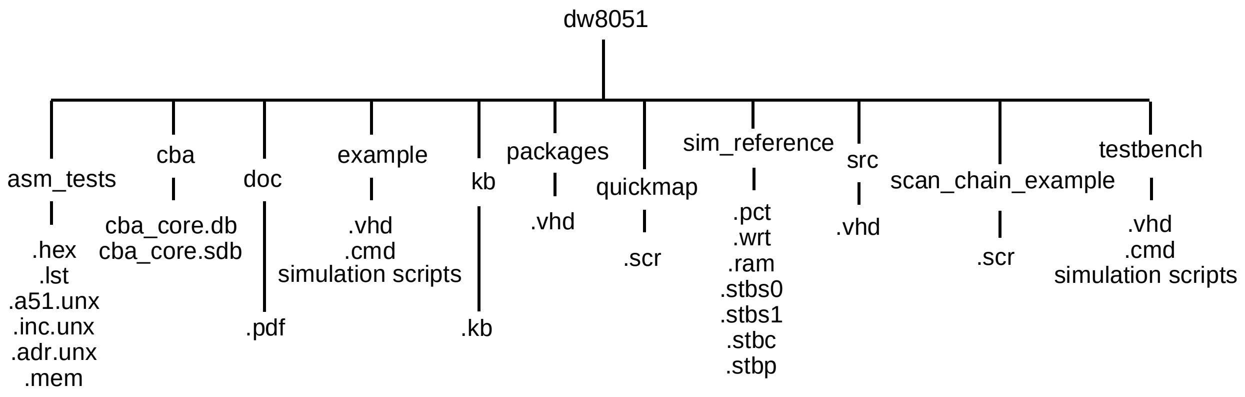


Table 4-2 Files and Directories in DW8051 – VHDL Version

Extension(1)	**Location**	**Description**
.a51.unx	./asm_tests	Assembler source code in UNIX format
.inc.unx	./asm_tests	Include file for assembler programs
.adr.unx	./asm_tests	Include file with address definitions for assembler programs
.lst	./asm_tests	List file of test program
.hex	./asm_tests	Intel Hex code of test program
.pct	./sim_reference	Program counter trace file
.wrt	./sim_reference	External RAM write trace file
.ram	./sim_reference	External RAM contents, either 32 or 128 bytes
.stbs0	./sim_reference	Strobe file for rxd/txd of Serial Port 0
.stbs1	./sim_reference	Strobe file for rxd/txd of Serial Port 1
.stbc	./sim_reference	Strobe file for DW8051_core output signals

Table 4-2 Files and Directories in DW8051 – VHDL Version (continued)

Extension(1)	Location	Description
.stbp	./sim_reference	Strobe file for s8032 output signals
.vhd	./src ./example	VHDL source files. The source files for DW8051_core in the src directory will be encrypted if you install the core kit without a DesignWare-8051-Source license. The source files in the example directory are always unencrypted.
.cmd	./testbench ./example	Command file for testbench execution, configured for actual simulation
.kb	./kb	coreConsultant knowledge database files
.scr	./quickmap	Script to generate a GTECH simulation model for DW8051_core, required for DW8051 (encrypted) version
.scr	./scan_chain _example	Example Test Compiler scan chain insertion script for DW8051_core
.pdf	./doc	PDF documentation files
.db	./cba	CBA library database file
.sdb	./cba	CBA library symbols database file
(1) Some of the files in ./sim_reference have the additional extension .core or .s8032 to indicate that the reference file could not be obtained by simulation of DS80C320 on the hardware modeler. These files were instead generated by simulation of DW8051_core (.core) or the example s8032 design (.s8032).		

Installation Directories and Files – Verilog Version

When coreConsultant completes the installation of the Verilog version of the DW8051 MacroCell Solution, your selected installation directory contains the directories and files described in Table 4-3 and Table 4-4.

Figure 4-3 shows the directory structure for the Verilog (DW8051 and DW8051-Source) versions of the DW8051 MacroCell Solution. In Figure 4-3, the selected installation directory name is dw8051.

Table 4-3 DW8051 MacroCell Directory Structure – Verilog Version

Directory	**Contents**
dw8051/asm_tests	Test programs
dw8051/cba	Cell Based Array (CBA) technology library
dw8051/doc	A PDF version of this databook. If you do not already have the Adobe Acrobat PDF reader installed, you can download it free from www.adobe.com.
dw8051/example	An example 8032 implementation using the DW8051
dw8051/kb	coreConsultant knowledge database (KB) files
dw8051/quickmap	Script to generate a GTECH simulation model for DW8051_core
dw8051/sim_reference	Golden reference for output trace, strobe, and RAM contents files from simulation of the DS80C320 on a hardware modeler
dw8051/src	Verilog source files. In the DW8051 version, the Verilog source files for DW8051_core and its submodules are encrypted. In the DW8051-Source version, the Verilog source files for DW8051_core and its submodules are not encrypted.
dw8051/ scan_chain_example	Example Test Compiler scan chain insertion script for DW8051_core
dw8051/testbench	Verilog source files for the DW8051 testbench, plus the testbench command file and simulation scripts

Figure 4-3 Directory Structure – Verilog Version

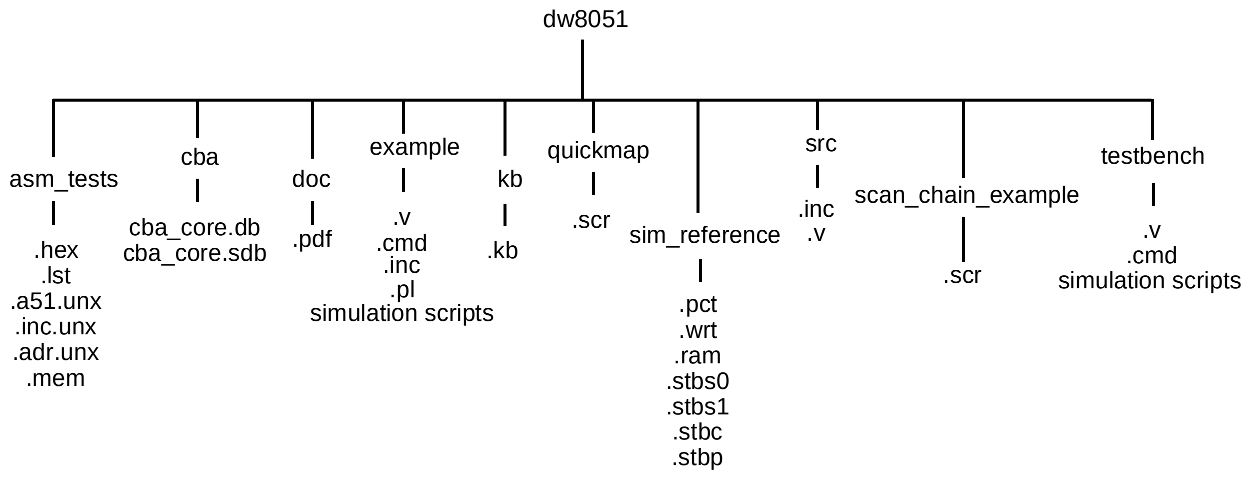


Table 4-4 Files and Directories in DW8051 – Verilog Version

Extension(1)	**Location**	**Description**
.a51.unx	./asm_tests	Assembler source code in UNIX format
.inc.unx	./asm_tests	Include file for assembler programs
.adr.unx	./asm_tests	Include file with address definitions for assembler programs
.lst	./asm_tests	List file of test program
.hex	./asm_tests	Intel Hex code of test program
.mem	./asm_tests	Test program after conversion from Intel Hex code to Verilog readable format by the ihex2mem.pl script
.pct	./sim_reference	Program counter trace file
.wrt	./sim_reference	External RAM write trace file
.ram	./sim_reference	External RAM contents, either 32 or 128 bytes
.stbs0	./sim_reference	Strobe file for rxd/txd of Serial Port 0

Table 4-4 Files and Directories in DW8051 – Verilog Version (continued)

Extension(1)	**Location**	**Description**
.stbs1	./sim_reference	Strobe file for rxd/txd of Serial Port 1
.stbc	./sim_reference	Strobe file for DW8051_core output signals
.stbp	./sim_reference	Strobe file for s8032 output signals
.v	./src ./example	Verilog source files. The source files for DW8051_core in the src directory will be encrypted if you install the core kit without a DesignWare-8051-Source license. The source files in the example directory are always unencrypted.
.inc	./src ./example	Verilog include file for ‘define statements
.cmd	./testbench ./example	Command file for testbench execution, configured for actual simulation
.task	./testbench	Command file for testbench execution after conversion to Verilog task by the cmd2task_c.pl script
.kb	./kb	coreConsultant knowledge database files
.pl	./asm_tests ./testbench	Perl scripts that convert test programs and testbench command file into Verilog-readable format
.pdf	./doc	PDF documentation files
.scr	./quickmap	Script to generate a GTECH simulation model for DW8051_core, required for DW8051 (encrypted) version
.scr	./ scan_chain_exampl e	Example Test Compiler scan chain insertion script for DW8051_core
.db	./cba	CBA library database file
.sdb	./cba	CBA library symbols database file

Note for Table 4-4.
(1) Some of the files in `./sim_reference` have the additional extension `.core` or `.s8032` to indicate that the reference file could not be obtained by simulation of DS80C320 on the hardware modeler. These files were instead generated by simulation of DW8051_core (`.core`) or the example s8032 design (`.s8032`).

Creating a Workspace

Follow the procedure in the *coreConsultant User Guide* to create a workspace for the DW8051. When you create your workspace, coreConsultant copies and/or links directories and files from the core kit installation directory to your selected workspace directory.

When coreConsultant finishes creating your workspace, it displays an activity list that contains the default coreConsultant design flow activities, plus simulation activities that are specific to the DW8051 MacroCell. To continue your work, click on the first activity in the list and follow the instructions to complete the activity. Refer to the *coreConsultant User Guide* and online help detailed instructions. The remainder of this chapter provides the information that you need to execute the DW8051-specific activities and integrate the DW8051 into your application design.

Specifying Your Configuration

When you perform the coreConsultant Specify Configuration activity, coreConsultant displays a dialog where you select values for the DW8051 user-configurable parameters. The default configuration is:

- 13-source interrupt unit (*extd_intr* = 1)
- 256-byte internal RAM (*ram_256* = 1)
- 8-KB internal ROM (*rom_addr_size* = 13)
- 2 serial ports (*serial* = 2)
- Three 16-bit timers (*timer2* = 1)

Specify your configuration by setting the configuration options (parameter values) as described in Table 4-5. When you complete the Specify Configuration activity, coreConsultant writes your selected parameter values to the file DW8051_parameter.v(hd) in your workspace src directory.

Table 4-5 Configuration Options

Configurable Feature	**Options**
Interrupt Unit Type	Extended – Selects extended interrupt unit with 13 sources (extd_intr = 1)
	Standard – Selects standard interrupt unit with 6 sources (extd_intr = 0)
Size of Internal RAM	256 bytes – Implements addressability to 256 bytes of internal RAM (ram256 = 1)
	128 bytes – Implements addressability to 128 bytes of internal RAM (ram256 = 0)

Table 4-5 Configuration Options

Configurable Feature	**Options**
Internal ROM Address Bus Width	Determines how many of the 16 internal ROM address bits (irom_addr) are used. Legal values are 0–16 (0 = no internal ROM present). Unused irom_addr pins are tied to logic 0. Associated HDL parameter is rom_addr_size.
Number of Serial Ports	None – No serial port present (serial = 0)
	One – Serial Port 0 present (serial = 1)
	Two – Serial Ports 0 and 1 present (serial = 2). If you choose to implement both serial ports, then you must select the extended interrupt unit (extd_intr = 1) to handle interrupt requests from Serial Port 1. If you select the standard interrupt unit (extd_intr = 0), you can still operate Serial Port 1 in polling mode.
Timer 2	Included – Timer 2 present (timer2 = 1)
	Removed – Timer 2 not present (timer2 = 0)

Simulating the DW8051 MacroCell

The DW8051 MacroCell Solution includes a comprehensive test suite that tests all DW8051 opcodes, internal peripherals, and special functions. The test suite includes:

- A testbench that instantiates DW8051_core, plus models and/or processes that emulate internal and external ROM and RAM, serial port devices, an external SFR device, and a test pattern generator to test the interrupt ports
- A set of test programs

- A testbench command file that specifies which test programs the testbench will execute
- A set of simulation reference (“golden”) files

In addition, there are special programs available as examples of how to test internal RAM and ROM.

coreConsultant automatically configures the testbench, invokes your simulator to run the simulation, and compares the simulation results against the supplied reference files.

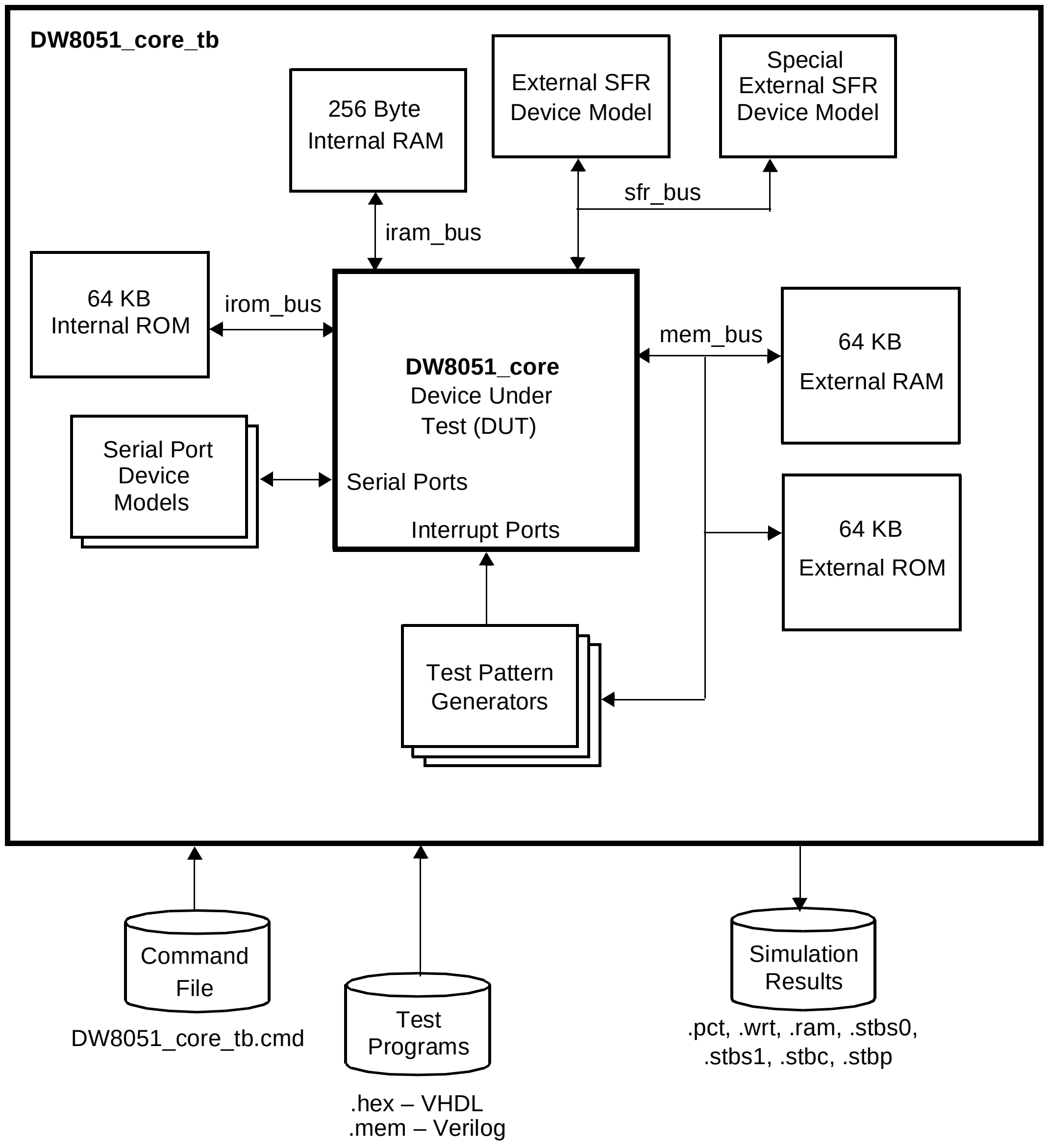
DW8051 Testbench Architecture

Figure 4-4 shows a block diagram of the DW8051 testbench (DW8051_core_tb). The testbench instantiates DW8051_core as the device under test, plus several models to support testing of internal peripherals and functions:

- Processes that emulate 64 KB of external ROM and 64 KB of external RAM connected to the external memory bus (mem_bus)
- 256-byte internal RAM model connected to the internal RAM bus (iram_bus)
- Processes that emulate up to 64 KB of internal ROM connected to the internal ROM bus (irom_bus)
- Multiple instances of a test pattern generator that is used for the interrupt tests
- Two instances of a serial port device model to support the serial port tests
- An external SFR bus device model to support testing the external SFR bus

- A special external SFR device model to support the test program sfr_iram_test, which verifies that external SFR bus devices are not affected by internal RAM accesses

Figure 4-4 DW8051 Testbench Architecture



Simulation Methods

There are two interfaces to the DW8051 MacroCell test suite:

- Configure and operate the test suite through coreConsultant.

  The DW8051 coreConsultant design flow includes DW8051-specific simulation activities that enable you to configure and operate the test suite through the coreConsultant GUI. The coreConsultant simulation activities enable you to execute any or all of the supplied test programs on your custom configuration of the DW8051 and check the results.

- Configure and operate the test suite directly from the UNIX command line using the supplied test suite configuration utilities and simulation scripts. You need to configure and operate the test suite directly if you want to create and execute any custom test programs.

The following sections describe how to configure and operate the test suite through coreConsultant, by executing the coreConsultant DW8051 simulation-specific activities. This is the recommended method for verifying your custom DW8051 configuration.

For a comprehensive description of the DW8051 test suite and procedures to configure and operate the test suite directly, refer to Chapter 5.

Verification Activities

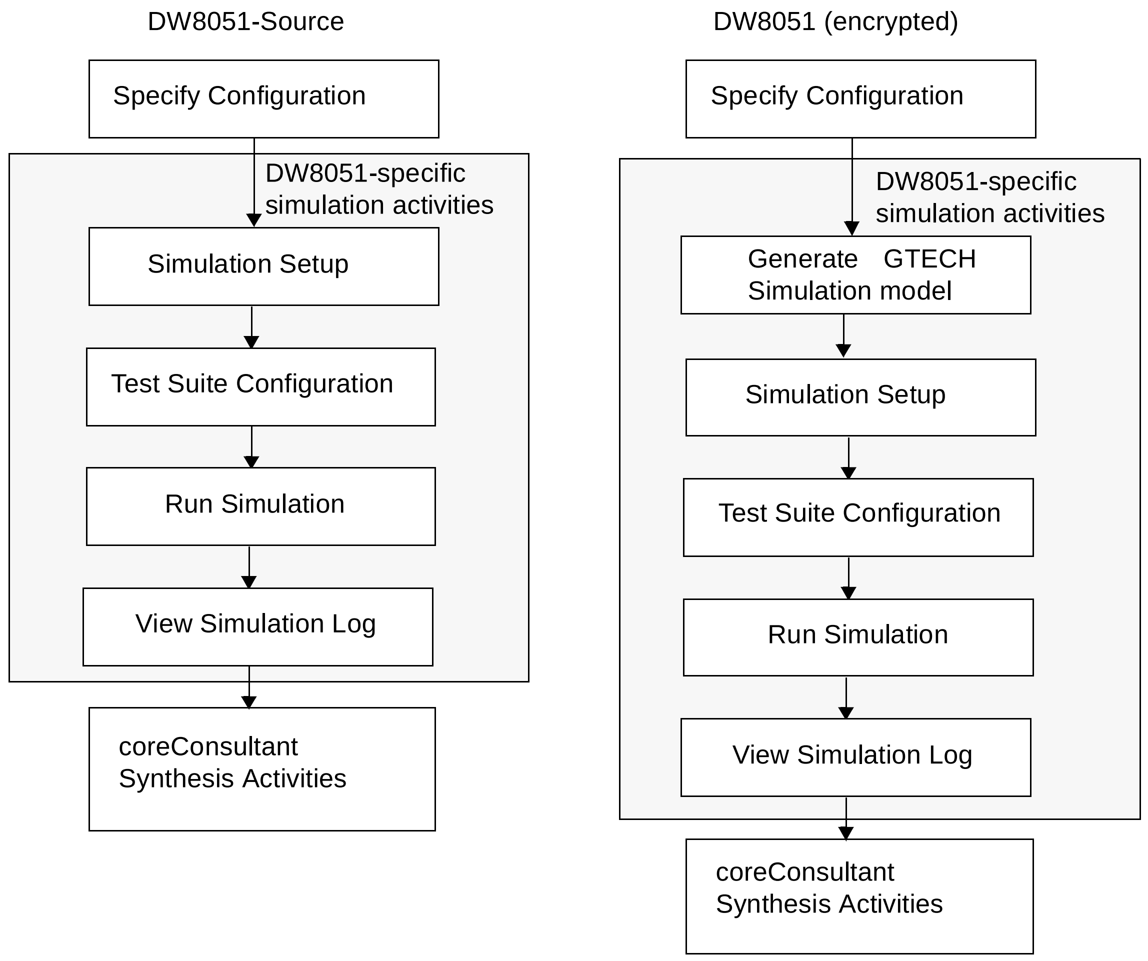
Simulation procedures differ slightly for the DW8051-Source and DW8051 (encrypted) versions of the DW8051 MacroCell Solution, as illustrated in Figure 4-5:

- For the DW8051 (encrypted) version, you must generate a GTECH simulation model for your custom DW8051 configuration. coreConsultant automatically uses the GTECH simulation model for DW8051_core when you run the simulation.
- For the DW8051-Source version, you do not need to generate a GTECH simulation model because you can simulate the RTL directly. Therefore, you can skip the Generate GTECH Simulation Model activity.

The coreConsultant simulation activities for the DW8051 MacroCell are:

1. Generate GTECH Simulation Model
2. Simulation Setup
3. Test Suite Configuration
4. Run Simulation
5. View Simulation Log

Figure 4-5 Simulation Procedures



Generating a GTECH Simulation Model

For the DW8051 (encrypted) version of the DW8051 MacroCell Solution, you must create a GTECH simulation model of your custom DW8051 configuration because the simulator cannot read the encrypted source files. The only exception is if you are using Synopsys VSS or Scirocco for VHDL simulation. VSS and Scirocco can read the encrypted source files.

Note:

If you are not using VSS or Scirocco and therefore need to create a GTECH simulation model, you will need a DesignWare-Foundation license to complete the simulation model generation process.

To create a GTECH simulation model, execute the DW8051-specific coreConsultant Generate GTECH Simulation Model activity. coreConsultant displays a dialog where you specify values for the parameters listed in Table 4-6.

Specify your parameters, then click OK. coreConsultant invokes Design Compiler to perform a low-effort compile (quick map) of your custom DW8051_core configuration using the Synopsys technology-independent GTECH library. The output file that contains your simulation model is <*workspace*>/quickmap/ DW8051_core_sim.v(hd).

Table 4-6 Generate GTECH Simulation Model Parameters

Parameter	**Description**
Run remotely	Selects whether to run the quick map on the specified host machine instead of the local workstation. For example, you may want to run the quick map on a high-capacity network server for faster execution.
Send Email	Selects whether to send email to the specified user when the quick map synthesis is complete. When you receive the email notification, the GTECH simulation model is ready and you can perform the remaining simulation activities.

Simulation Setup

The Simulation Setup activity creates a new directory named sim_res_core in your workspace to contain the output files from simulation of the test programs.

After creating the sim_res_core directory, the Simulation Setup activity runs one of the DW8051 test programs on your selected VHDL or Verilog simulator and checks the result to ensure that your simulation environment is properly configured.

To perform the Simulation Setup activity:

1. Make sure that one of the following simulators is properly installed on your system and all its required environment variables are set:

   - VHDL
     Synopsys VSS, Synopsys Scirocco, Cadence Leapfrog, or MTI ModelSim

   - Verilog
     Synopsys VCS, Cadence Verilog-XL, Cadence NC-Verilog, or MTI ModelSim

2. Click Simulation Setup in the coreConsultant activity list.

3. Select your VHDL or Verilog simulator and click OK.

When the test program completes successfully (no simulation errors encountered and the test result files in sim_res_core match the corresponding reference files in sim_reference), coreConsultant enables the Test Suite Configuration activity.

If the test program fails to complete successfully, coreConsultant posts an error message indicating the cause of the problem so that you can correct the problem and repeat Simulation Setup.

Test Suite Configuration

The Test Suite Configuration activity configures the testbench command file to execute selected test programs for your custom DW8051 configuration. By default, most of the test programs applicable to your custom DW8051 configuration are enabled in the Test Suite Configuration spreadsheet. Some of the more time-consuming tests are disabled by default so that you can quickly simulate the majority of the test programs, then selectively simulate the few tests that require the most simulation time.

In the Test Suite Configuration spreadsheet, enable or disable individual tests by setting the value in the Test Enabled column to 1 (enabled) or 0 (disabled). To determine the function of the various tests listed in the Test Suite Configuration dialog, use the coreConsultant "What’s This" help feature. For more information about the testbench command file and test programs, refer to the test suite description in Chapter 5.

When you click OK, coreConsultant writes (or overwrites) the configured testbench command file to DW8051_core_tb.cmd in your workspace testbench directory. The DW8051 testbench is now configured to automatically execute the selected tests on your configured version of DW8051_core.

Run Simulation

The Run Simulation activity generates a script to invoke your selected VHDL or Verilog simulator to simulate the testbench. You can also select to invoke the script to execute the simulation directly from the coreConsultant Run Simulation dialog. The simulation runs all the test programs that you selected in the Test Suite Configuration activity.

There are user-selectable parameters to the Run Simulation activity, as listed in Table 4-7. Select and/or enter values for the Run Simulation parameters in the Run Simulation dialog, then click OK to complete the Run Simulation activity.

Table 4-7 Run Simulation Parameters

Run Simulation Parameter	**Description**
Simulation Script File	The name of the generated simulation run script. The default is SimScript.csh.
Simulation Log File	The name of the simulation log file. The default is Simulation.log.
Generate Scripts Only	Selects whether to execute the simulation or just generate the simulation run script. If checked (the default), coreConsultant generates, but does not execute, the simulation run script. You can execute the script at a later time by invoking the run script (SimScript.csh) directly from the UNIX command line.
Send Email	Selects whether to send email to the specified user when the simulation is complete.
Run remotely	Selects whether to run the simulation on the specified host machine instead of the local workstation. For example, you may want to run the simulation on a high-capacity network server for faster execution.

coreConsultant writes the simulation run script to your workspace testbench directory. If you selected to run the script, coreConsultant executes the simulation run script.

To execute or re-execute the simulation script at a later time, go to the testbench directory in your workspace and execute the simulation run script. For example, if you accepted the default name for the simulation run script, use the following commands:

```
% cd <workspace>/testbench
% ./SimScript.csh
```

The simulation run script performs the following functions:

- Runs the simulation on your selected simulator and directs the output to the simulation log file in your workspace testbench directory. The default simulation log file name is Simulation.log.
- Compares the simulation results for each test in sim_res_core to the corresponding reference file in sim_reference and appends the results of the comparison to the simulation log file.

You can then use the View Simulation Log activity to check the results of the comparisons.

View Simulation Log

The View Simulation Log activity displays the simulation log file in your selected web browser. The name of the log file is <*workspace*>/testbench/Simulation.log unless you selected a different name. At the end of the log file, there is a test summary indicating the number of passes and failures. There should not be any test failures.

If any of the supplied test programs fail, contact your Synopsys representative.

If any of your custom test programs fail, re-analyze the test program execution using a waveform viewer. For more information about creating and executing custom test programs, refer to “Creating and Executing Custom Tests” on page 5-17.

Application Specific Simulation

In addition to verifying your custom DW8051 configuration as a standalone block using the supplied test suite, you should also verify the DW8051 in the context of your application design according to your verification strategy. To do so, integrate the DW8051 into your application design as described in Integrating the DW8051 into Your ASIC Design in this chapter and simulate the design according to your verification strategy.

If you are using DW8051-Source, you can simulate the RTL version of your custom DW8051 configuration. If you are using DW8051 (encrypted), you must simulate the GTECH simulation model for your custom DW8051 configuration.

Simulating Internal RAM

DW8051_core includes an interface for internal RAM, but not the RAM itself. When you integrate the DW8051 into your application for application-specific testing, you must also implement a model for internal RAM and connect the model to the internal RAM interface, as described in Interfacing to Internal RAM in this chapter.

The DW8051 testbench (DW8051_core_tb) includes a simulation model for a 256-byte internal RAM (DW8051_ram_256) that is connected to the internal RAM interface. You can use the testbench simulation model for internal RAM as an example of how to implement external RAM for application-specific testing.

Simulating Internal ROM

DW8051_core includes an interface for internal ROM, but not the ROM itself. When you integrate the DW8051 into your application for application-specific testing, you must also implement a model for internal ROM and connect the model to the internal ROM interface, as described in Interfacing to Internal ROM in this chapter.

The DW8051 testbench (DW8051_core_tb) includes processes that emulate internal and external ROM. You can use these processes as examples of how to model internal ROM for application-specific testing.

For VHDL, there are separate process for internal and external ROM. The testbench automatically invokes the correct process to perform transactions on either the internal ROM bus or the external memory bus, depending on the value of *rom_addr_size*.

For Verilog, a single process monitors the internal ROM interface to determine when the internal ROM is being accessed. When the testbench detects an internal ROM access, it places the corresponding data on the internal ROM data bus. When the testbench detects an external ROM access, it places the corresponding data on the external memory data bus.

Synthesizing the DW8051 MacroCell

To synthesize your custom DW8051 configuration, perform the coreConsultant set of synthesis activities: context/intent specification, strategy selection, synthesis, and results analysis.

The default set of synthesis attributes on DW8051_core should provide acceptable synthesis results in most libraries, using the coreConsultant DC_opto_strategy. The only synthesis attribute values you need to specify are the clock and design context attributes.

When performing the synthesis activities with DW8051, be aware of the following default synthesis intent specifications. You can override these and any other synthesis intent specifications:

- The default OptimizationPriorities for DW8051_core is timing_area. If your design goals differ, you need to change the value of OptimizationPriorities.
- The input/output constraint attributes are applied only to the top-level ports of DW8051_core. There are no constraint attributes explicitly set on subblocks. The synthesis strategy is a top-down compile of DW8051_core. If you want to alter the strategy to compile subblocks individually, you need to specify port intent on the subblocks. One way extract budgets for subblocks is to use the coreConsultant DC_design_budgeting_strategy, as described in the *coreConsultant User Guide*.
- Min and Max values for the delay constraints on the top-level ports are set to equal values.
- PreserveBoundaries is false on DW8051_core, which means that the DW8051_core and its subblocks are subject to Design Compiler boundary optimization.
- TestReadyCompile is true, which means that Design Compiler will insert scan flip-flops on the first compile (compile -scan).

Confirming Your Gate-Level Design

One of the coreConsultant synthesis strategy selection parameters that you select is the output format for the gate-level netlist of DW8051_core. The default format is either VHDL or Verilog, depending on which version of the DW8051 MacroCell you are using.

If you select VHDL or Verilog, you can confirm the synthesized design by simulating the gate-level netlists with the same testbench used to verify the RTL design functionality.

coreConsultant stores the gate-level netlist output from Design Compiler in the directories <*workspace*>/syn/<*phase*>, where <*phase*> is the name of a compile phase (for example, incr1). The number of synthesis phases executed depends on the value that you select for the synthesis strategy selection QoR Effort parameter.

Confirm the synthesized design by simulating the gate-level netlist with the same testbench used to verify the RTL design functionality.

Integrating the DW8051 into Your ASIC Design

After you are satisfied with the area, timing, and functionality of the synthesized design, integrate your DW8051 implementation into your design. To do so, instantiate DW8051_core and connect its ports as needed for the design implementation. The following sections describe how to:

- Instantiate DW8051_core
- Interface to internal memory devices
- Interface to external memory devices
- Interface to devices on the external SFR bus

Instantiating DW8051_core (VHDL)

Example 4-1 shows how to instantiate DW8051_core in VHDL.

Example 4-1   Instantiating DW8051_core in VHDL

```
library IEEE,DW8051;
use IEEE.std_logic_1164.all;
use DW8051.std_logic_misc.all;
use DW8051.std_logic_arith.all;
use DW8051.std_logic_unsigned.all;
use DW8051.DW8051_components.all;
use DW8051.DW8051_packages.all;

entity my_8051 is
end my_8051;

architecture str of my_8051 is

signal clk, por_n, rst_in_n, rst_out_n : std_logic;
signal test_mode_n                     : std_logic;
signal stop_mode_n, idle_mode_n        : std_logic;
signal sfr_addr                        : std_logic_vector (7 downto 0);
signal sfr_data_out, sfr_data_in       : std_logic_vector (7 downto 0);
signal sfr_wr, sfr_rd                  : std_logic;
signal mem_addr                        : std_logic_vector (15 downto 0);
signal mem_data_out, mem_data_in       : std_logic_vector (7 downto 0);
signal mem_wr_n, mem_rd_n              : std_logic;
signal mem_pswr_n, mem_psrd_n          : std_logic;
signal mem_ale, mem_ea_n               : std_logic;
signal int0_n, int1_n, int2, int3_n    : std_logic;
signal int4, int5_n, pfi, wdti         : std_logic;
signal rxd0_in, rxd0_out, txd0         : std_logic;
signal rxd1_in, rxd1_out, txd1         : std_logic;
signal t0, t1, t2, t2ex                : std_logic;
signal t0_out, t1_out, t2_out          : std_logic;
signal port_pin_reg_n                  : std_logic;
signal p0_mem_reg_n, p0_addr_data_n    : std_logic;
signal p2_mem_reg_n                    : std_logic;
signal iram_addr                       : std_logic_vector (7 downto 0);
signal iram_data_out                   : std_logic_vector (7 downto 0);
signal iram_data_in                    : std_logic_vector (7 downto 0);
signal iram_rd_n                       : std_logic;
signal iram_we1_n                      : std_logic;
signal iram_we2_n                      : std_logic;
signal irom_addr                      : std_logic_vector (15 downto 0)

signal irom_data_out                   : std_logic_vector (7 downto 0);
signal irom_rd_n                       : std_logic;
signal irom_cs_n                       : std_logic;


begin
  -- component instantiation:
  U0: DW8051_core
```

```
port map (clk            => clk,
          por_n          => por_n,
          rst_in_n       => rst_in_n,
          rst_out_n      => rst_out_n,
          test_mode_n    => test_mode_n,
          stop_mode_n    => stop_mode_n,
          idle_mode_n    => idle_mode_n,
          sfr_addr       => sfr_addr,
          sfr_data_out   => sfr_data_out,
          sfr_data_in    => sfr_data_in,
          sfr_wr         => sfr_wr,
          sfr_rd         => sfr_rd,
          mem_addr       => mem_addr,
          mem_data_out   => mem_data_out,
          mem_data_in    => mem_data_in,
          mem_wr_n       => mem_wr_n,
          mem_rd_n       => mem_rd_n,
          mem_pswr_n     => mem_pswr_n,
          mem_psrd_n     => mem_psrd_n,
          mem_ale        => mem_ale,
          mem_ea_n       => mem_ea_n,
          int0_n         => int0_n,
          int1_n         => int1_n,
          int2           => int2,
          int3_n         => int3_n,
          int4           => int4,
          int5_n         => int5_n,
          pfi            => pfi,
          wdti           => wdti,
          rxd0_in        => rxd0_in,
          rxd0_out       => rxd0_out,
          txd0           => txd0,
          rxd1_in        => rxd1_in,
          rxd1_out       => rxd1_out,
          txd1           => txd1,
          t0             => t0,
          t1             => t1,
          t2             => t2,
          t2ex           => t2ex,
          t0_out         => t0_out,
          t1_out         => t1_out,
          t2_out      => t2_out,
          port_pin_reg_n  => port_pin_reg_n,
          p0_mem_reg_n   => p0_mem_reg_n,
          p0_addr_data_n  => p0_addr_data_n,
          p2_mem_reg_n   => p2_mem_reg_n,
                iram_addr        => iram_addr,
                iram_data_out    => iram_data_out,
                iram_data_in     => iram_data_in,
                iram_rd_n        => iram_rd_n,
                iram_we1_n       => iram_we1_n,
                iram_we2_n       => iram_we2_n,

                irom_addr        => irom_addr,
                irom_data_out    => irom_data_out,
                irom_rd_n        => irom_rd_n,
                irom_cs_n        => irom_cs_n
               );
end str;
-------------------------------------------------------------
-- pragma translate_off
configuration my_8051_cfg of my_8051 is
  for str
    for U0:
      DW8051_core use configuration DW8051.DW8051_core_cfg_rtl;
    end for;
  end for;
end my_8051_cfg;
-- pragma translate on
```

Instantiating DW8051_core (Verilog)

Example 4-2 shows how to instantiate DW8051_core in Verilog.

Example 4-2   Instantiating DW8051_core in Verilog

```
module my_8051;
  reg  clk;
  reg  por_n;
  reg  rst_in_n;
  wire rst_out_n;
  reg  test_mode_n;
  wire stop_mode_n;
  wire idle_mode_n;
  wire [7:0] sfr_addr;
  wire [7:0] sfr_data_out;
  wire [7:0] sfr_data_in;
  wire sfr_wr;
  wire sfr_rd
  wire [15:0] mem_addr;
  wire [7:0] mem_data_out;
  reg  [7:0] mem_data_in;
  wire mem_wr_r;
  wire mem_rd_n;
  wire mem_pswr_n;
  wire mem_psrd_n;
  wire mem_ale;
  reg  mem_ea_n;
  wire int0_n;
  wire int1_n;
  wire int2;
  wire int3_n;
  wire int4;
  wire int5_n;
  wire pfi;
  wire wdti;
  wire rxd0_in;
  wire rxd0_out, txd0;
  wire rxd1_in;
  wire rxd1_out, txd1;
  wire t0;
  wire t1;
  wire t2;
  wire t2ex;
  wire t0_out, t1_out, t2_out;
  wire port_pin_reg_n, p0_mem_reg_n, p0_addr_data_n, p2_mem_reg_n;
  wire [7:0] iram_addr, iram_data_out, iram_data_in;
  wire iram_rd_n, iram_we1_n, iram_we2_n;
  wire [15:0] irom_addr;
  reg [7:0] irom_data_out;
  wire irom_rd_n, irom_cs_n;
  //-----------------------------------------------------------------
  // DW8051 instantiation:
  //-----------------------------------------------------------------
  DW8051_core U0 (
          .clk                  (clk),
          .por_n                (por_n),
          .rst_in_n             (rst_in_n),
          .rst_out_n            (rst_out_n),
          .test_mode_n          (test_mode_n),

          .stop_mode_n          (stop_mode_n),
          .idle_mode_n          (idle_mode_n),

          .sfr_addr             (sfr_addr),
          .sfr_data_out         (sfr_data_out),
          .sfr_data_in          (sfr_data_in),
          .sfr_wr               (sfr_wr),
          .sfr_rd               (sfr_rd),
          .mem_addr              (mem_addr),
          .mem_data_out          (mem_data_out),
          .mem_data_in           (mem_data_in),
          .mem_wr_n              (mem_wr_n),
          .mem_rd_n              (mem_rd_n),
          .mem_pswr_n            (mem_pswr_n),
          .mem_psrd_n            (mem_psrd_n),
          .mem_ale               (mem_ale),
          .mem_ea_n              (mem_ea_n),

          .int0_n                (int0_n),
          .int1_n                (int1_n),
          .int2                  (int2),
          .int3_n                (int3_n),
          .int4                  (int4),
          .int5_n                (int5_n),

          .pfi                   (pfi),
          .wdti                  (wdti),

          .rxd0_in               (rxd0_in),
          .rxd0_out              (rxd0_out),
          .txd0                  (txd0),
          .rxd1_in               (rxd1_in),
          .rxd1_out              (rxd1_out),
          .txd1                  (txd1),

          .t0                    (t0),
          .t1                    (t1),
          .t2                    (t2),
          .t2ex                  (t2ex),

          .t0_out                (t0_out),
          .t1_out                (t1_out),
          .t2_out                (t2_out),

          .port_pin_reg_n        (port_pin_reg_n),
          .p0_mem_reg_n          (p0_mem_reg_n),

          .p0_addr_data_n        (p0_addr_data_n),
          .p2_mem_reg_n          (p2_mem_reg_n),


          .iram_addr             (iram_addr),
          .iram_data_out         (iram_data_out),
          .iram_data_in          (iram_data_in),
          .iram_rd_n             (iram_rd_n),
          .iram_we1_n            (iram_we1_n),
          .iram_we2_n            (iram_we2_n),

          .irom_addr             (irom_addr),
          .irom_data_out         (irom_data_out),
          .irom_rd_n           (irom_rd_n),
          .irom_cs_n           (irom_cs_n)
          );
endmodule //my_8051
```

Interfacing to Internal RAM

The DW8051 MacroCell provides an interface for either 128 or 256 bytes of internal RAM, as determined by the *ram_256* parameter. *ram_256* = 0 selects 128 bytes; *ram_256* = 1 selects 256 bytes. The default is 256 bytes (*ram_256* = 1).

The internal RAM address bus (*iram_addr*) is always 8 bits wide. The *ram_256* parameter determines whether or not DW8051_core will address the upper 128 bytes.

Internal RAM Read Interface

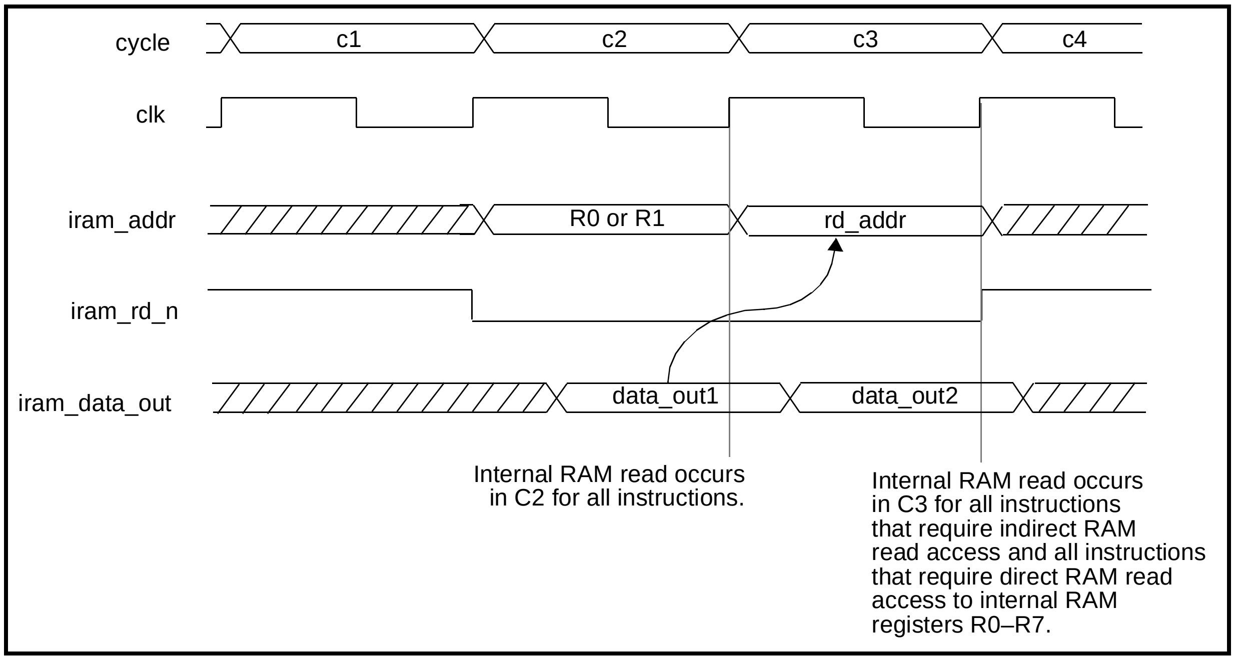
The internal RAM read interface signals are *iram_addr*, *iram_data_out*, and *iram_rd_n*. All internal RAM read accesses occur within one *clk* cycle.

For all instructions, the DW8051 performs an internal RAM read to register R0 or R1 in C2 of the first instruction cycle. The DW8051 performs an additional internal RAM read during C3 for instructions that involve either of the following types of internal RAM accesses:

- Indirect internal RAM read access (for example, `MOV A, @Ri` where `i` = 0 or 1)
- Direct internal RAM read access to registers R0–R7 (for example, `INC Rn`, where n = 0–7)

Figure 4-6 shows the internal RAM read timing for an instruction that requires an indirect internal RAM read. *iram_rd_n* remains asserted for two clock cycles to perform the two internal RAM read accesses during C2 and C3.

Figure 4-6 Internal RAM Read Signals and Timing

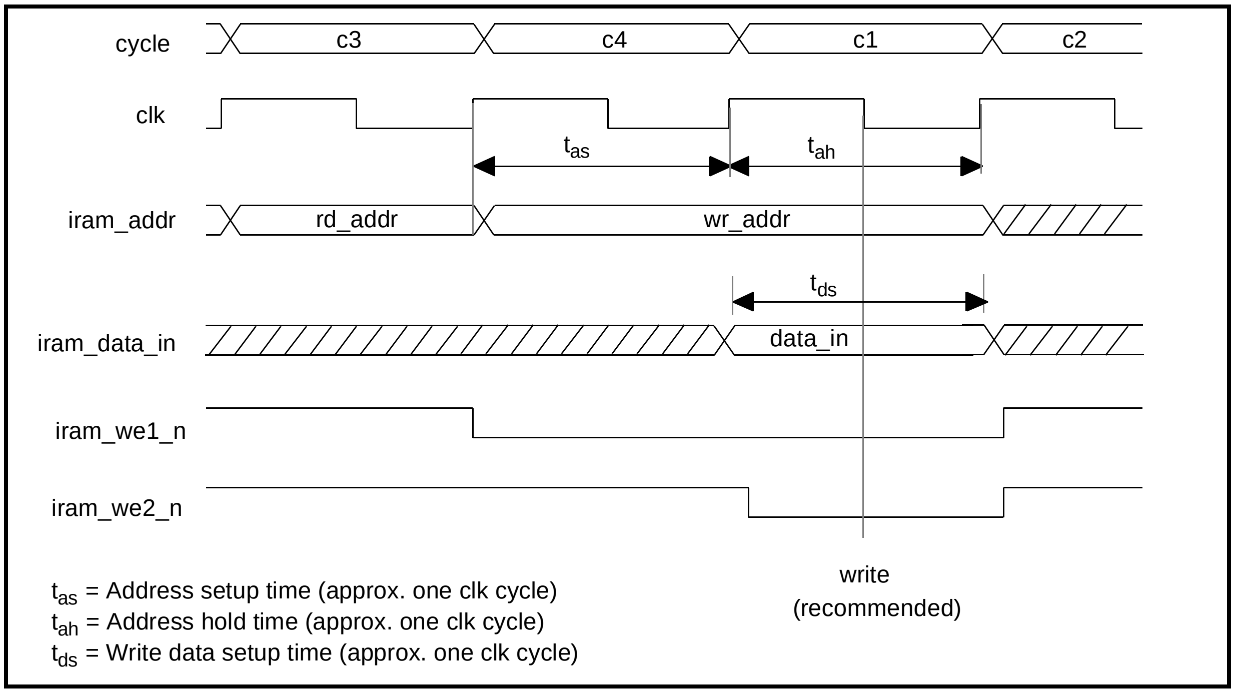

cycle
c1
c2
c3
c4
clk
iram_addr
R0 or R1
rd_addr
iram_rd_n
iram_data_out
data_out1
data_out2
Internal RAM read occurs in C2 for all instructions.
Internal RAM read occurs in C3 for all instructions that require indirect RAM read access and all instructions that require direct RAM read access to internal RAM registers R0–R7.

Internal RAM Write Interface

The internal RAM write interface signals are *iram_addr*, *iram_data_in*, *iram_we1_n* and *iram_we2_n*. All internal RAM write accesses occur within two *clk cycles*, as illustrated in Figure 4-7. *iram_we1_n* goes active in C4 to indicate that there is a valid internal RAM address on *iram_addr. iram_we2_n* goes active in C1 to indicate that there is valid data on *iram_data_in*.

Figure 4-8 and Figure 4-10 show the recommended internal RAM implementations for asynchronous and synchronous RAMs.

Figure 4-7 Internal RAM Write Signals and Timing



cycle
c3
c4
c1
c2
clk
t_{as}
t_{ah}
iram_addr
rd_addr
wr_addr
t_{ds}
iram_data_in
data_in
iram_we1_n
iram_we2_n
write (recommended)
t_{as} = Address setup time (approx. one clk cycle)
t_{ah} = Address hold time (approx. one clk cycle)
t_{ds} = Write data setup time (approx. one clk cycle)

Implementing Internal RAM

To implement internal RAM, connect a technology-specific, gate-level RAM netlist, supplied by your ASIC vendor, to the internal RAM bus. In addition, you may need to implement glue logic so that your technology-specific RAM module can meet the signal and timing requirements of the DW8051 internal RAM interface.

Interfacing to Asynchronous RAM

Figure 4-8 illustrates an example asynchronous internal RAM implementation using the DW8051 internal RAM interface. The implementation in Figure 4-8 uses glue logic to cause the RAM write enable input to go active on the falling edge of *clk* in the middle of C1, when *iram_we1_n* and *iram_we2_n* are both active. Figure 4-9 illustrates the asynchronous RAM interface timing.

Figure 4-8   Example Asynchronous Internal RAM Implementation

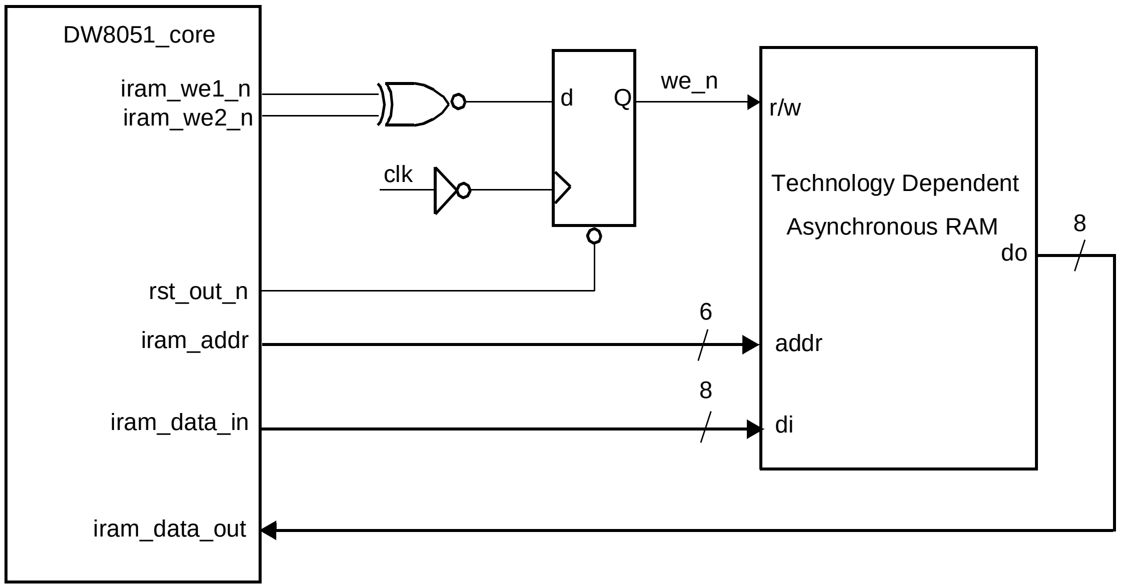
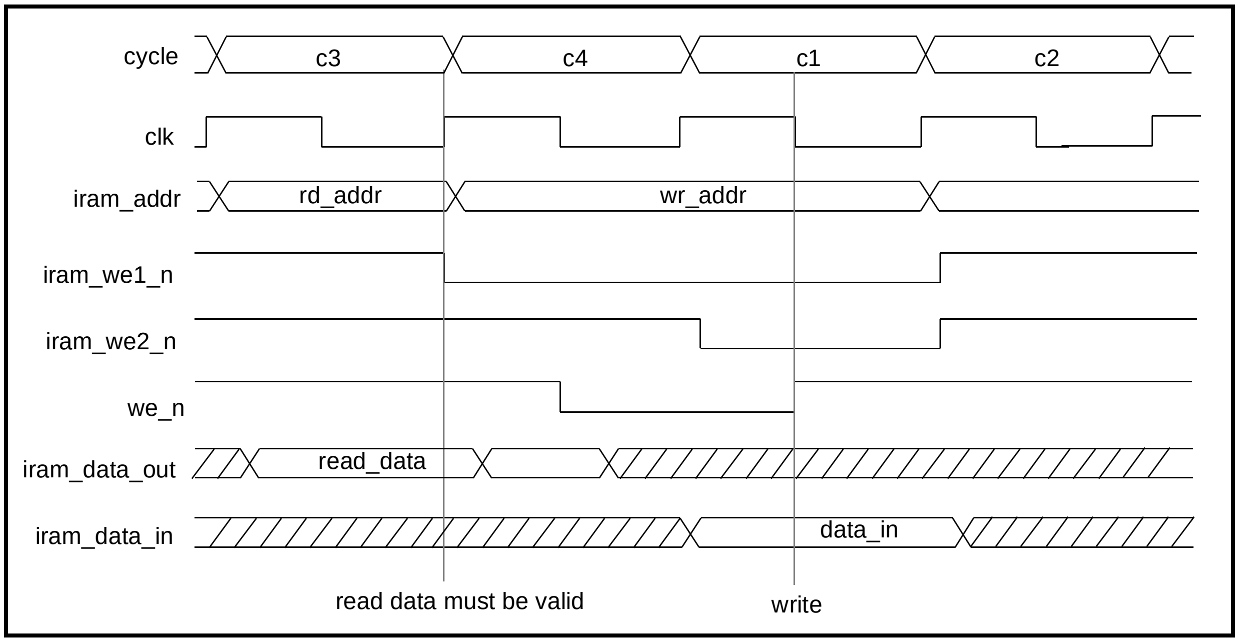


Figure 4-9   Asynchronous RAM Interface Timing



Read Cycle Timing Parameters

The read access time for the configuration in Figure 4-8 is approximately one *clk* cycle minus some delays. The formula for read access time is:

Tacc_required = Tcy – Taddr – Tff_setup

where:

Tacc_required = Read access time required for asynchronous RAM
Tcy = Clock period
Taddr = Clock to output delay for *iram_addr* flip-flops
Tff_setup = Setup time of flip-flop in DW8051_cpu

For a clock frequency of 25 MHz (clock period 40 ns), this leads to an access time of approximately 25–30 ns for most target technologies.

Write Cycle Timing Parameters

For write access of the example asynchronous RAM implementation shown in Figure 4-8, there are 4 timing parameters of interest:

Tawb_setup = Setup time for *iram_addr* to falling edge of *we_n*
Tdiwb_setup = Setup time for *iram_data_in* to rising edge of *we_n*
Tdiwb_hold = Hold time for *iram_data_in* from rising edge of *we_n*
Twb_width = Width of *we_n* pulse

The values of these parameters are:

Tawb_setup = approximately 1/2 clock cycle
Tdiwb_setup = approximately 1/2 clock cycle
Tdiwb_hold = approximately 1/2 clock cycle
Twb_width = approximately 1 clock cycle

For a clock frequency of 25 MHz (clock period 40 ns), all write parameters are approximately 20 or 40 ns, which are values that meet the requirements of most target technologies.

Interfacing to Synchronous RAM

Figure 4-10 illustrates an example synchronous internal RAM implementation using the DW8051 internal RAM interface. Figure 4-11 illustrates the synchronous RAM interface timing.

Read Cycle Timing Parameters

For synchronous RAM read access, there are four parameters of interest:

Traddr_setup = Setup time of *iram_addr* to rising edge of *clk*
Traddr_hold = Hold time of *iram_addr* to rising edge of *clk*
Trdata_setup = Setup time *rd_data* to rising edge of *clk*
Trdata_hold = Hold time *rd_data* to rising edge of *clk*

For the configuration shown in Figure 4-10, all read parameters are approximately 1/2 clock cycle. For a clock frequency of 25 MHz (clock period 40 ns), these parameters are approximately 20 ns, which is acceptable for most semiconductor RAMs.

Figure 4-10   Example Synchronous Internal RAM Implementation

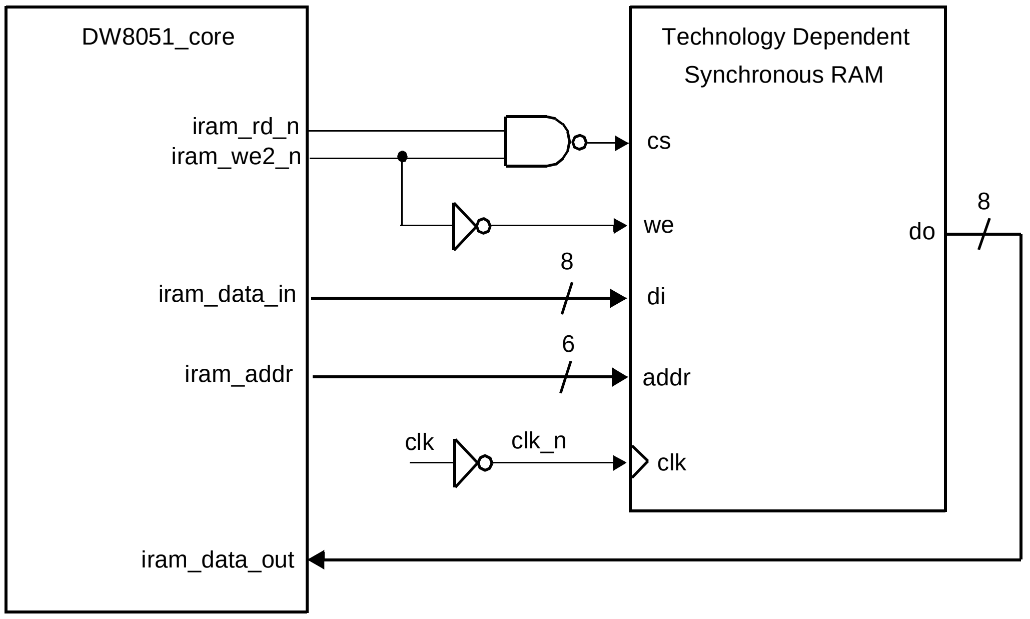
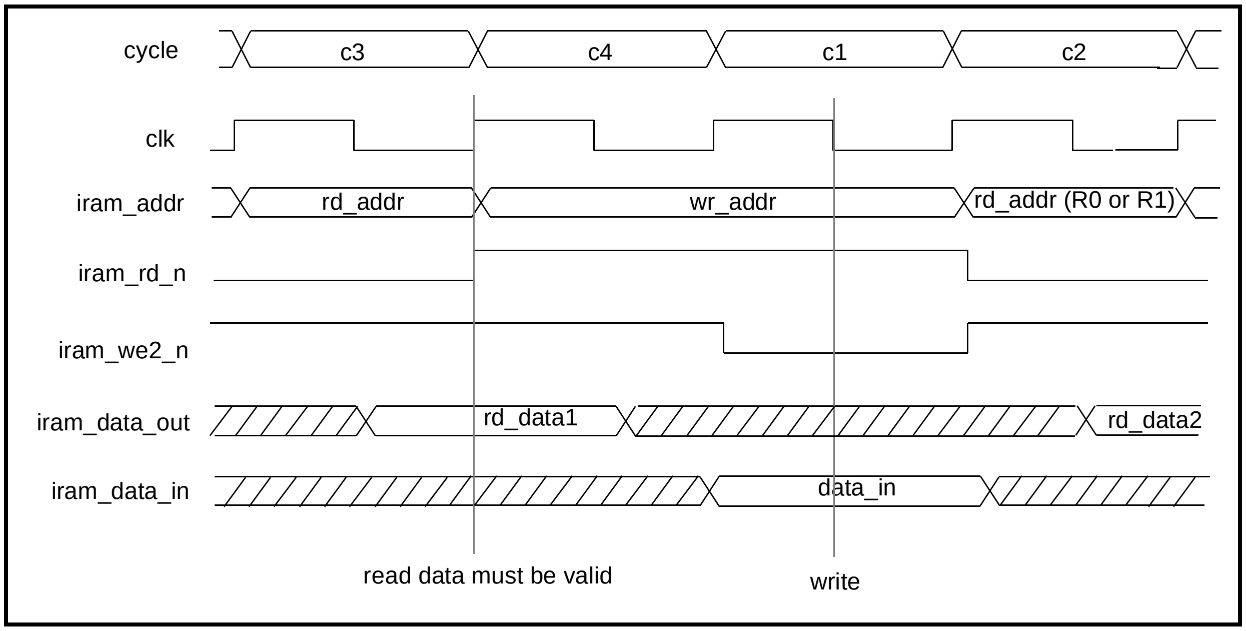


Figure 4-11   Synchronous RAM Interface Timing



The overall read access time for the configuration in Figure 4-10 is approximately 1/2 clock cycle (*iram_addr* latched on falling edge of *clk*) minus some delays. The formula for read access time is:

Tacc_required = 0.5**Tcy* – *Tff_setup*

where:

Tacc_required = Read access time required for synchronous RAM
Tcy = Clock period
Tff_setup = Setup time of flip-flop in DW8051_cpu

For a clock frequency of 25 MHz (clock period 40 ns), this leads to an access time of approximately 10–15 ns for most target technologies. This is a value that must be regarded as critical. The timing for this path must be evaluated in detail for a given target frequency.

Write Cycle Timing Parameters

For the write access, there are four parameters of interest:

Twaddr_setup = Setup time for *iram_addr* from rising edge of *clk*
Twaddr_hold = Hold time for *iram_addr* to rising edge of *clk*
Twdata_setup = Setup time of *iram_data_in* from rising edge of *clk*
Twdata_hold = Hold time for *iram_data_in* to rising edge of *clk*

For the configuration shown in Figure 4-10, the parameters values are:

Twaddr_setup = approximately 1/2 clock cycle
Twaddr_hold = approximately 1/2 clock cycle
Twdata_setup = approximately 1.5 clock cycle
Twdata_hold = approximately 1/2 clock cycle

For a clock frequency of 25 MHz (clock period 40 ns), all write parameters are approximately 20 or 60 ns, which are values that meet the requirements of most target technologies.

Interfacing to Internal ROM

The DW8051 MacroCell provides an interface for up to 64 KB of internal ROM, as determined by the *rom_addr_size* parameter. The default value is *rom_addr_size* = 13 (8-KB internal ROM).

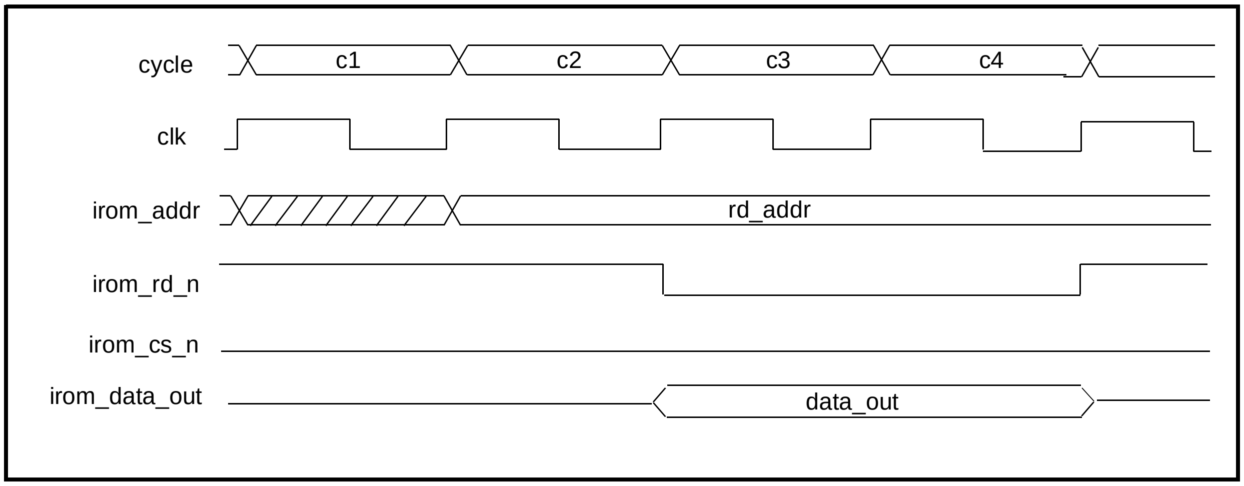
The *irom_addr* bus is always 16 bits wide. The *rom_addr_size* parameter determines how many of the 16 *irom_addr* bits are used; unused *irom_addr* bits are tied to logic 0.

Internal ROM Interface Signals

The internal ROM interface signals are *irom_addr*, *irom_rd_n*, *irom_cs_n*, and *irom_data_out*. Figure 4-12 shows the internal ROM interface timing. The access time of internal ROM is exactly 2 *clk* cycles from assertion of *irom_rd_n*.

The internal ROM read access can also be driven by address only, by holding *irom_rd_n* and *irom_cs_n* low (always active). The access time for address-only internal ROM reads is 3 *clk* cycles from a valid address on *irom_addr*.

Figure 4-12 Internal ROM Interface Timing



Implementing Internal ROM

To implement internal ROM, connect a technology-specific, gate-level ROM netlist, supplied by your ASIC vendor, to the internal ROM bus. In addition, you may need to implement glue logic so that your technology-specific ROM module can meet the signal and timing requirements of the DW8051 internal ROM interface.

Interfacing to External Memory Devices

The DW8051 provides the signals needed to interface with external ROM and RAM either through standard 8051 port modules (with Port 0 used as a multiplexed address/data bus), or through a more efficient memory interface with a 16-bit address bus.

Standard 8051 Port Modules

The example design (s8032) in the dw8051/example directory includes examples of how to build and control standard 8051 port modules (P0, P1, P2, and P3). Figure 4-13 illustrates the interconnections between DW8051_core and the standard port modules in the example design. Figure 4-14 illustrates the interfaces between an 8032-compatible microcontroller built from DW8051_core and external ROM and RAM, using the standard 8051 ports.

In Figure 4-13, the *port_pin_reg_n* signal switches between read of the port latch and the input pads for instructions that use the read-modify-write feature. *p0_mem_reg_n* and *p2_mem_reg_n* control whether port modules P0 and P2 provide memory or port latch data at the output. *p0_addr_data_n* determines whether port P0 drives the lower 8 bits of memory address or memory output data. The *mem_ale* signal is used to externally latch the lower 8 address bits provided at port P0.

The alternate function inputs and outputs of P1 and P3 do not require individual enables to qualify the alternate functions. The DW8051_core outputs *txd0*, *rxd0*, *txd1*, *rxd1*, *t2_out*, *mem_rd_n*, and *mem_wr_n* are tied high when the associated peripherals are not enabled.

Figure 4-13   DW8051_core to Standard Port Module Connections

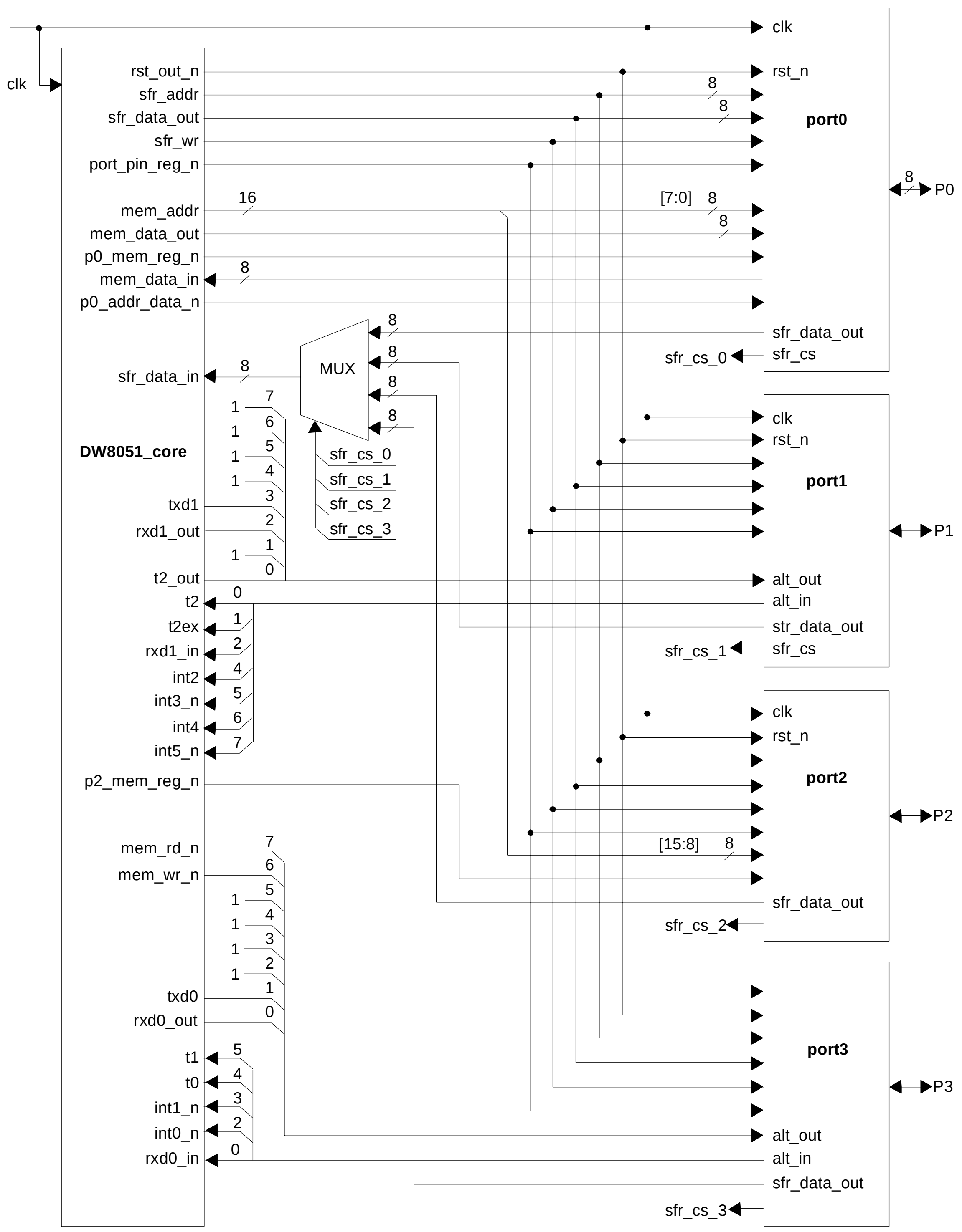


Figure 4-14 ROM/RAM Interface for 8032 Built from DW8051_core

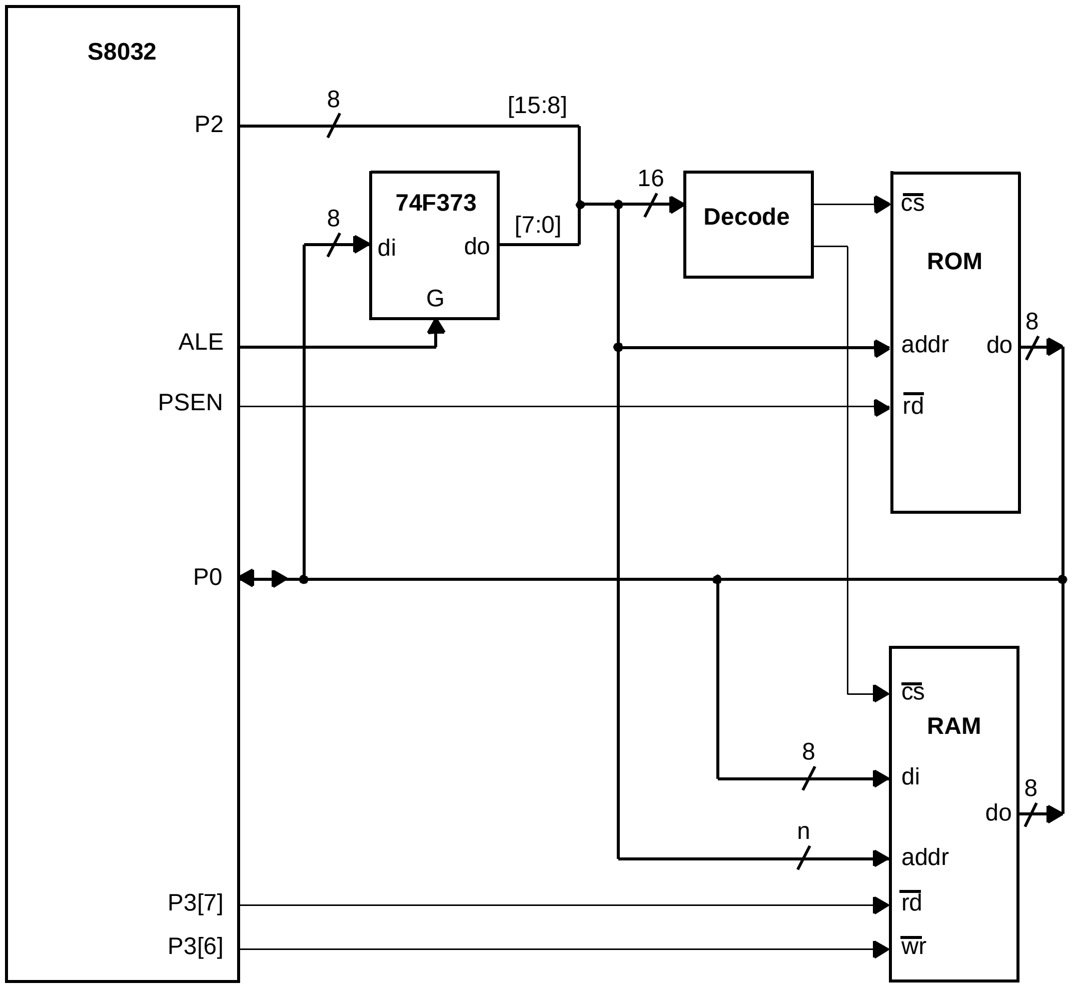
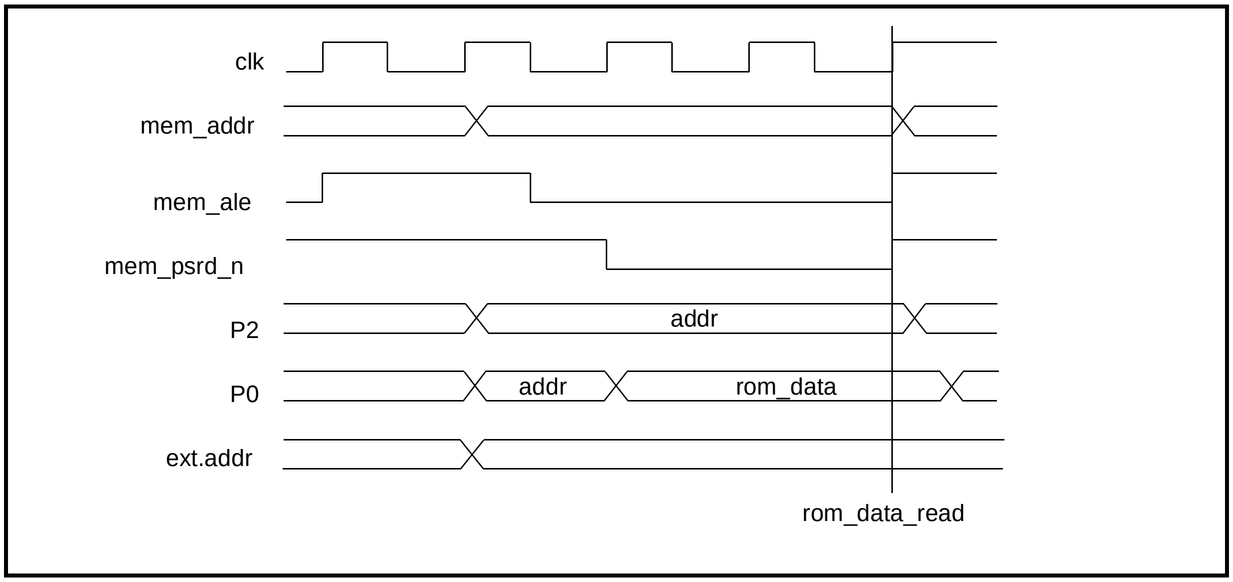


Figure 4-15 illustrates the timing for external ROM read access. The limiting factor is usually the access time of the (E)PROM from address change (access time from *rd* active is shorter). The required (E)PROM access time for this configuration is about 2.7 to 2.9 clock cycles (depending on pad/external delays). This equates to approximately 110 ns for a 25-MHz clock.

Using a standard (E)PROM with 120-ns access time with some margins yields a maximum clock frequency of approximately 20 MHz.

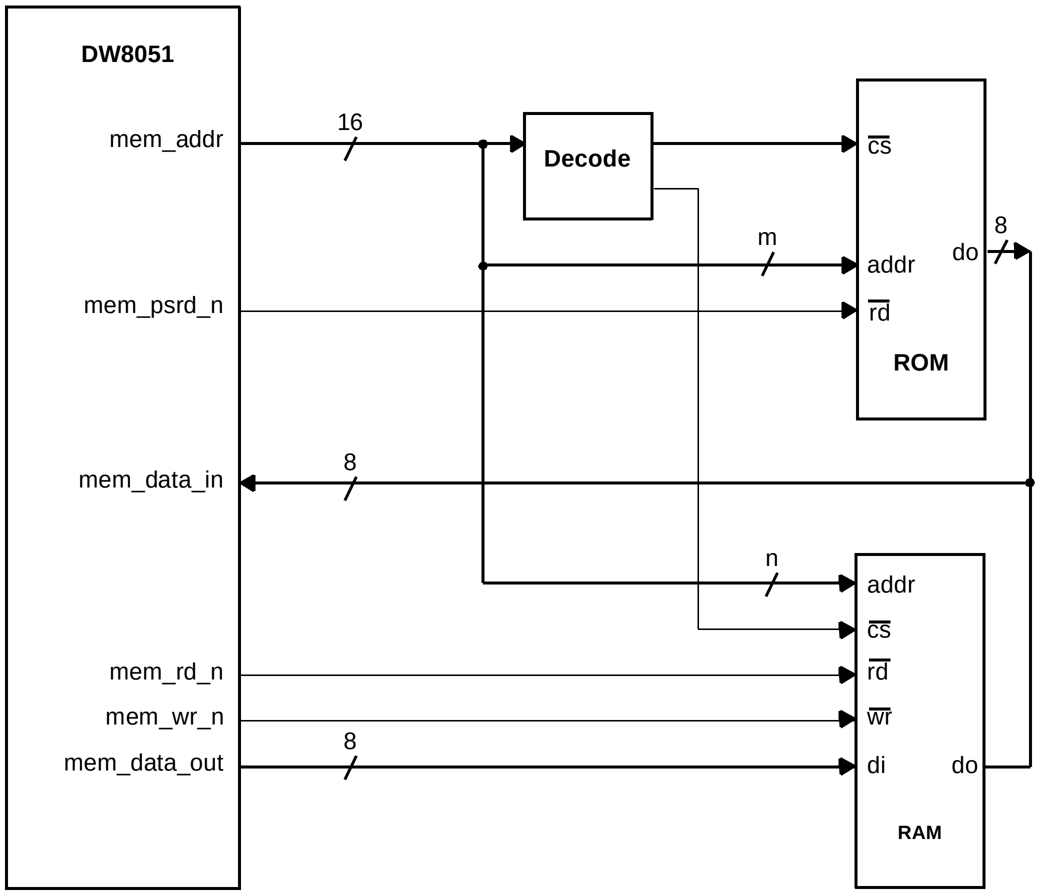
Figure 4-15 External ROM Timing



16-Bit Address Memory Interface

To implement the 16-bit address bus memory interface, provide all 16 *mem_addr* lines as primary output ports, as illustrated in Figure 4-16. This eliminates the external latch for the lower address bits and makes the *mem_ale* line obsolete.

Figure 4-16 16-Bit Address Bus Memory Interface Connections



External ROM Timing

Figure 4-17 illustrates the timing of DW8051_core signals for external ROM read access. Figure 4-18 illustrates the external ROM read timing for an 8032-compatible microcontroller built from DW8051_core.

Figure 4-17 DW8051_core Signals for Program Memory Read Cycle

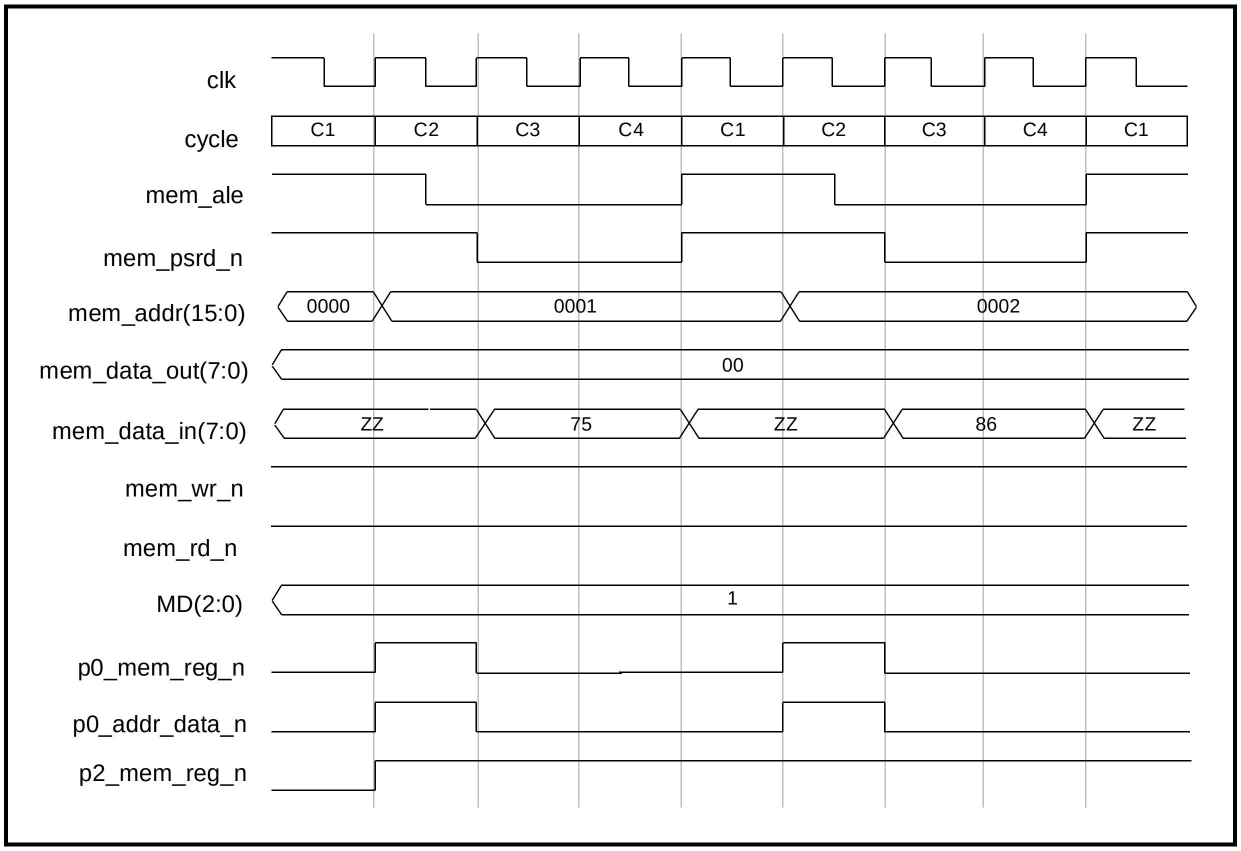
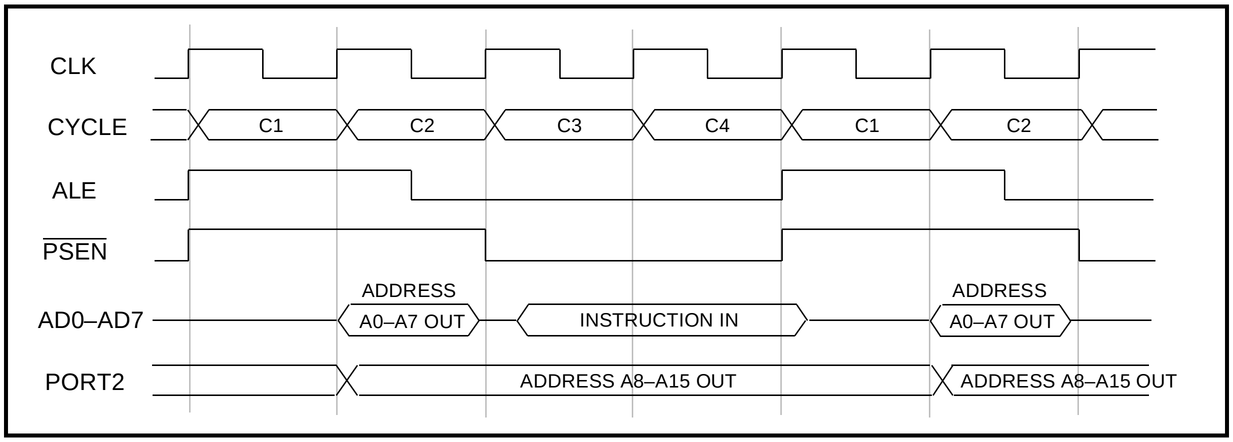


Figure 4-18 8032-Compatible I/O Signals for Program Memory Read Cycle



External RAM Timing

The timing for external RAM read and write operations depends on the settings of stretch memory cycle control bits (CKCON.2–0). The following figures illustrate the DW8051_core signals for a variety of stretch memory cycle control settings:

Operation	**DW8051_core Timing**	**8032-Compatible Timing**
RAM read, Stretch = 0	Figure 4-19	Figure 4-20
RAM write, Stretch = 0	Figure 4-21	Figure 4-22
RAM read, Stretch = 1	Figure 4-23	Figure 4-24
RAM write, Stretch = 1	Figure 4-25	Figure 4-26
RAM write, Stretch = 2	Figure 4-27	Figure 4-28

Figure 4-19   DW8051_core Signals for Data Memory Read with Stretch=0

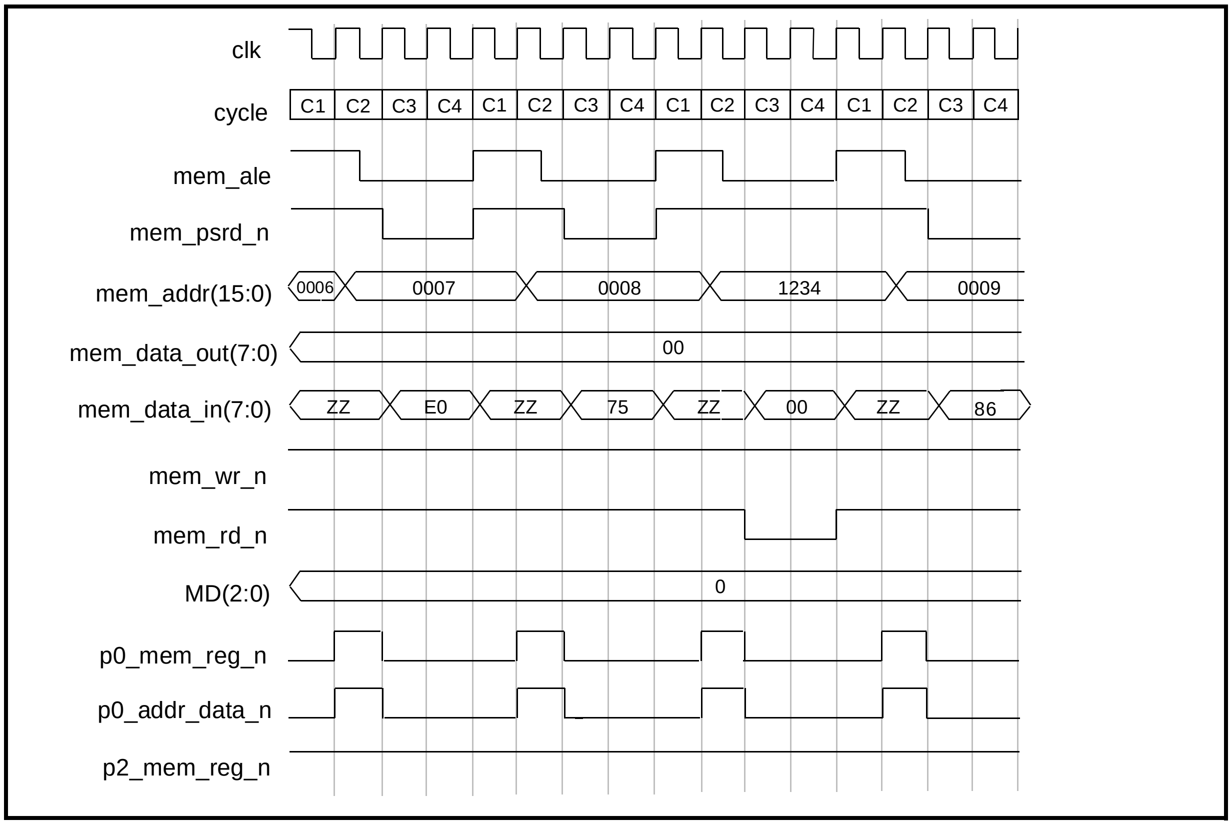


Figure 4-20 8032-Compatible I/O Signals for Data Memory Read with Stretch=0

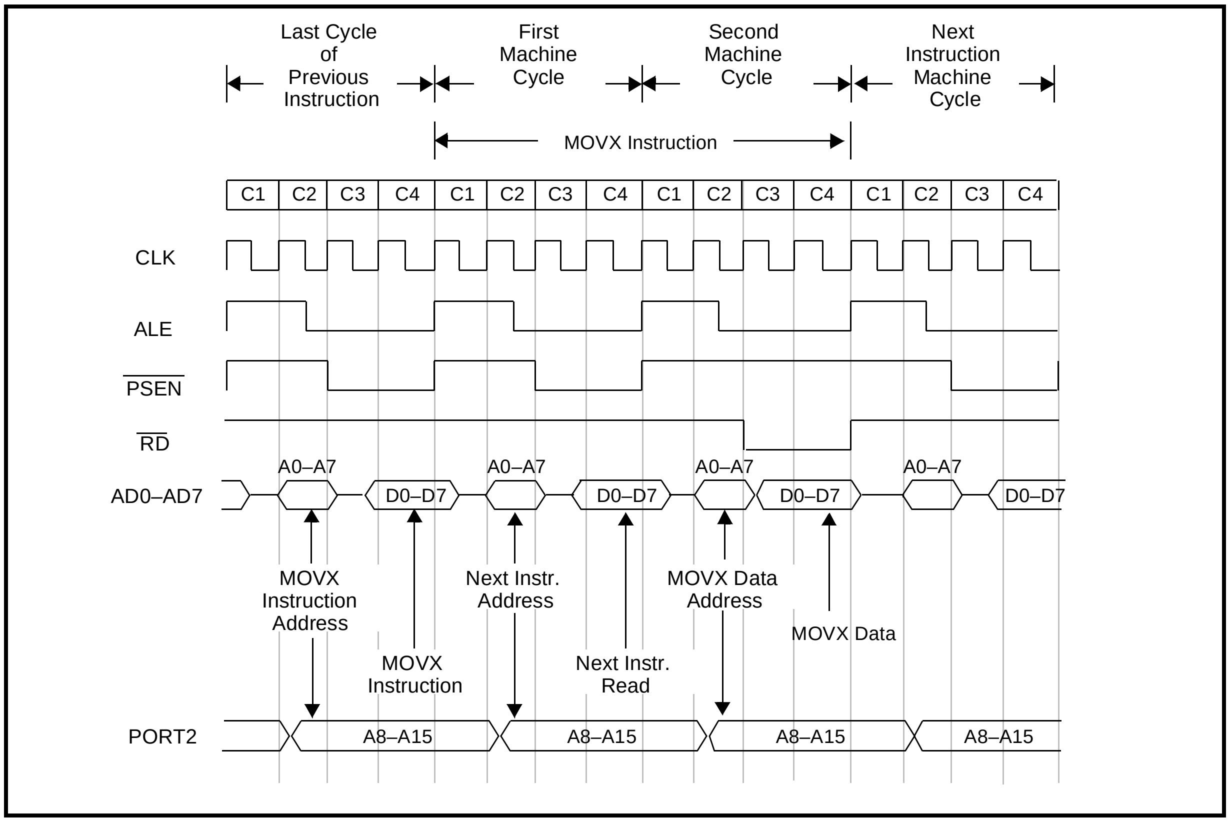


Figure 4-21 DW8051_core Signals for Data Memory Write with Stretch=0

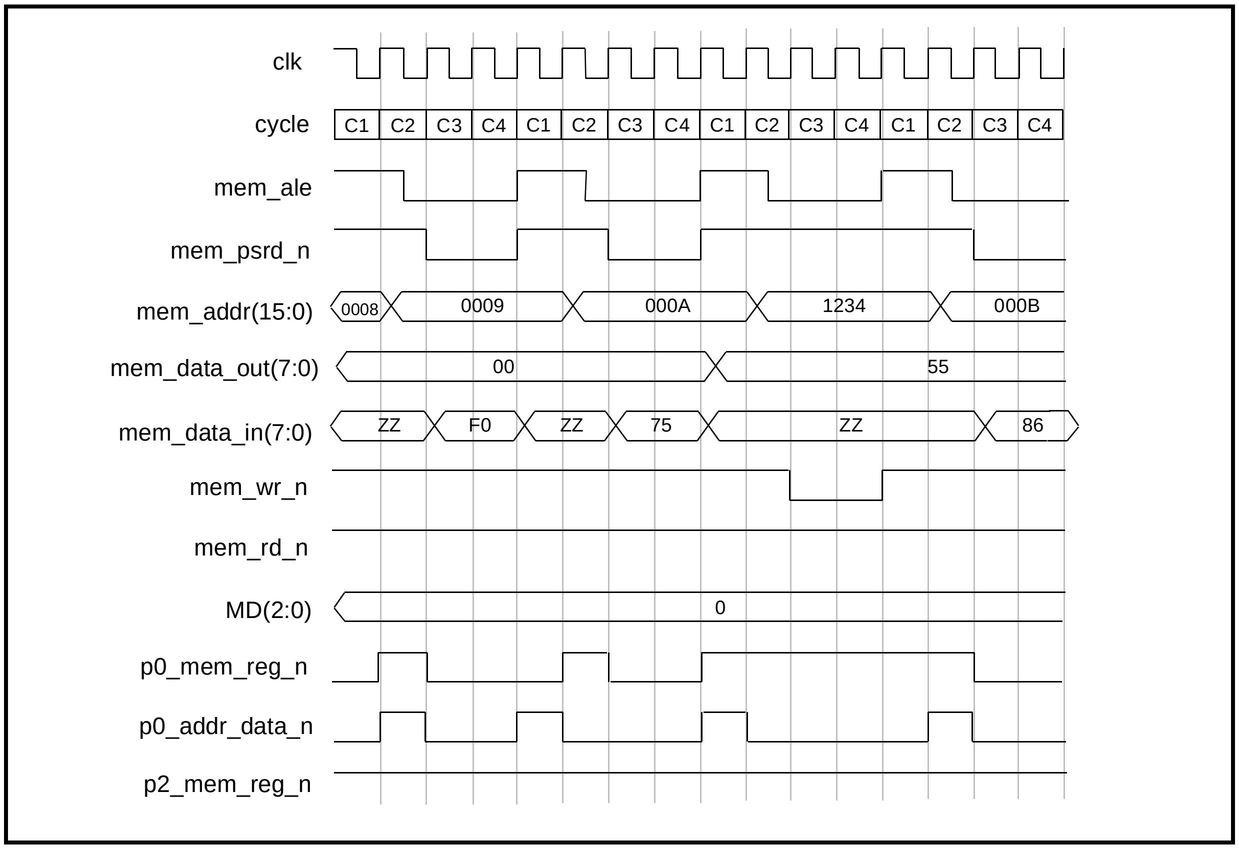


Figure 4-22 8032-Compatible I/O Signals for Data Memory Write with Stretch=0

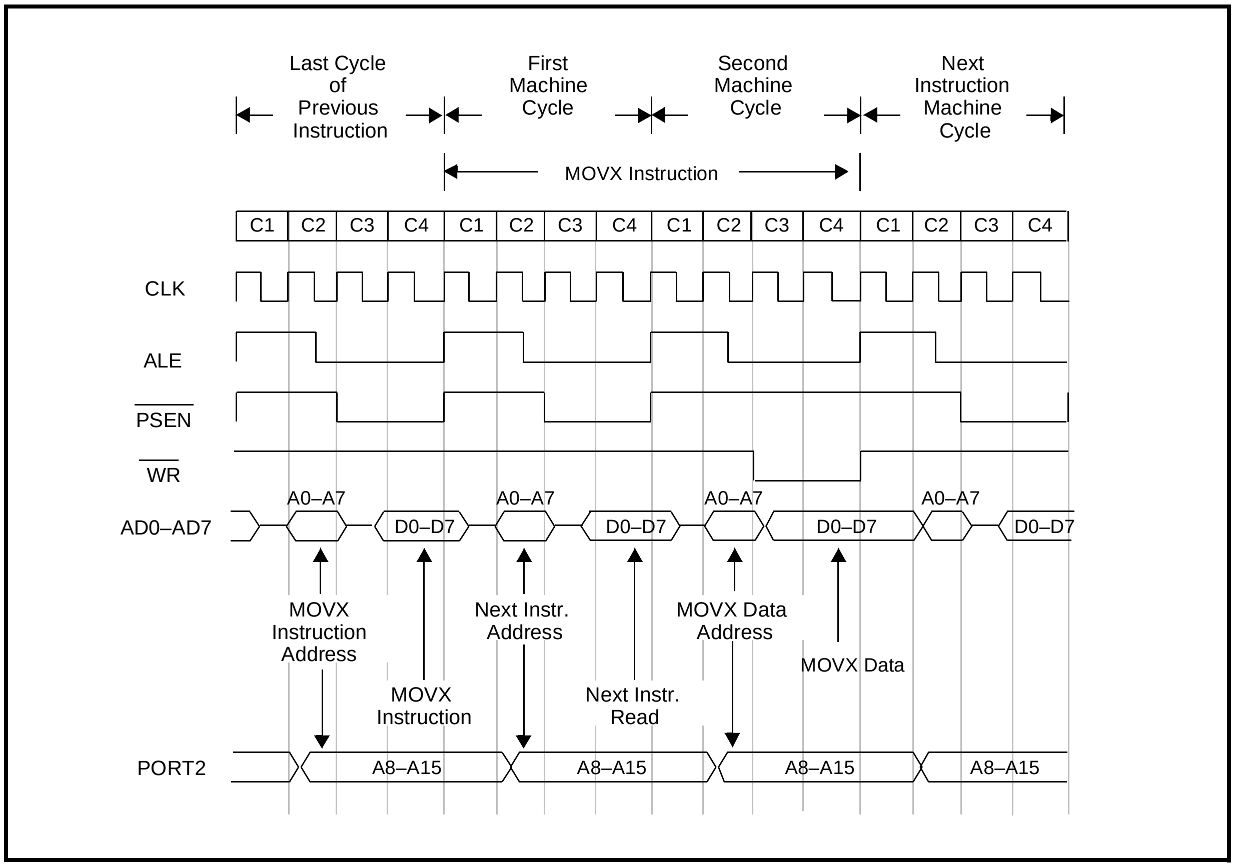


Figure 4-23 DW8051_core Signals for Data Memory Read with Stretch=1

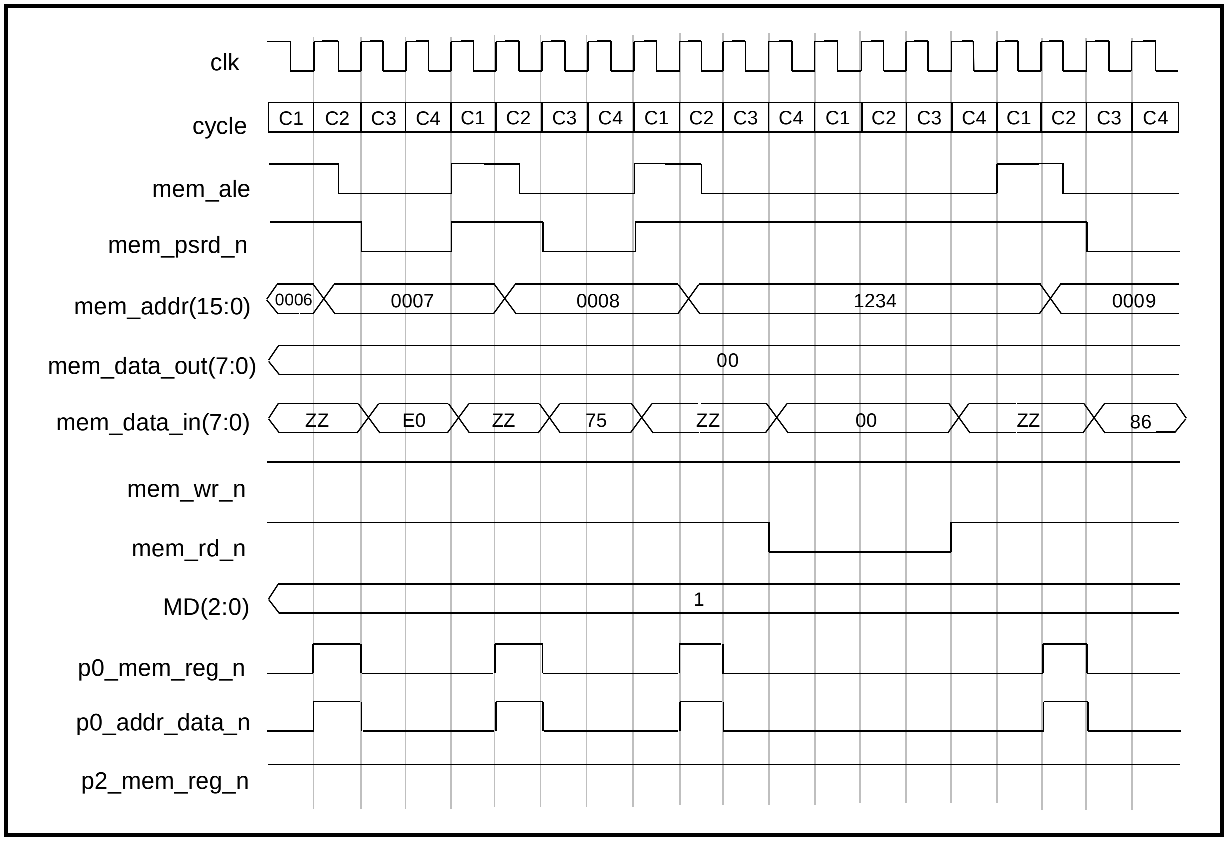


Figure 4-24 8032-Compatible I/O Signals for Data Memory Read with Stretch=1

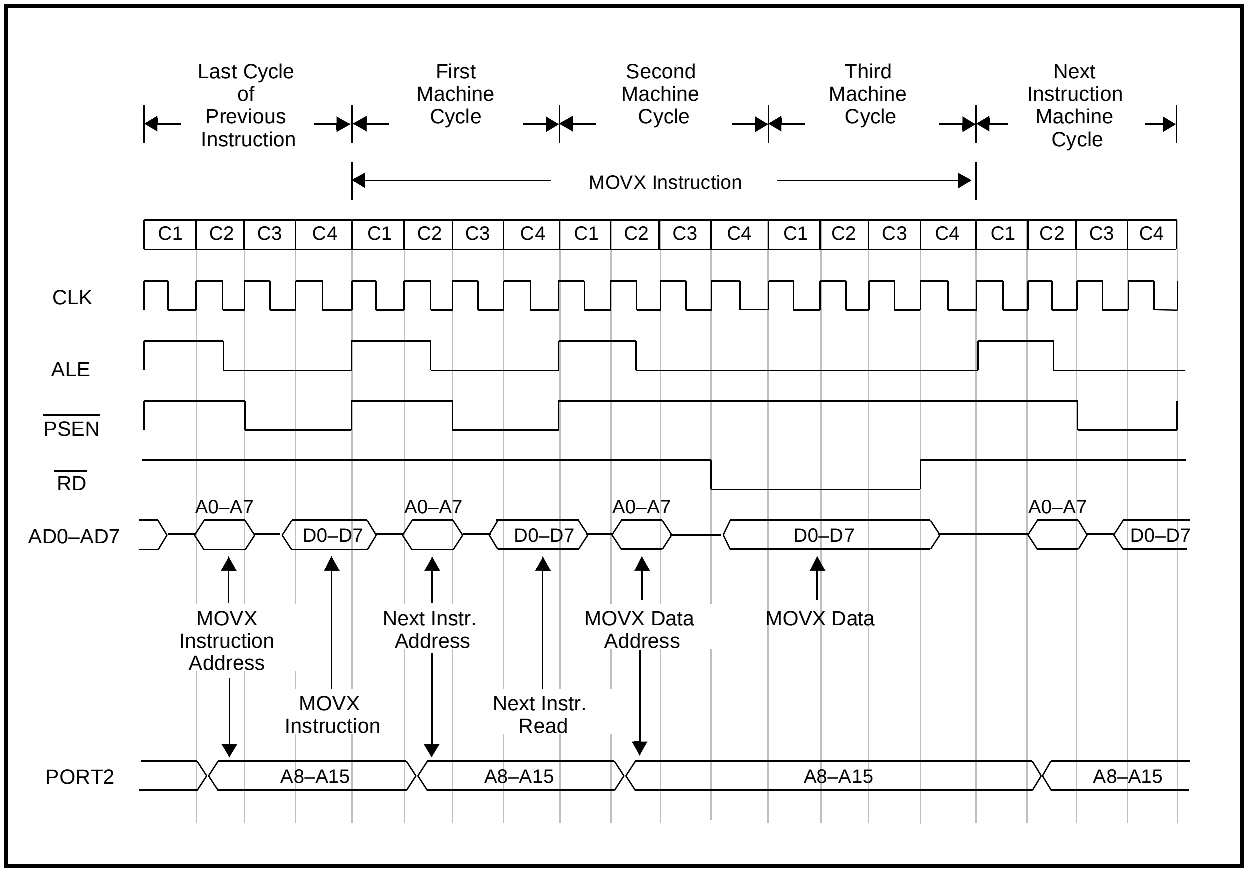


Figure 4-25 DW8051_core Signals for Data Memory Write with Stretch=1

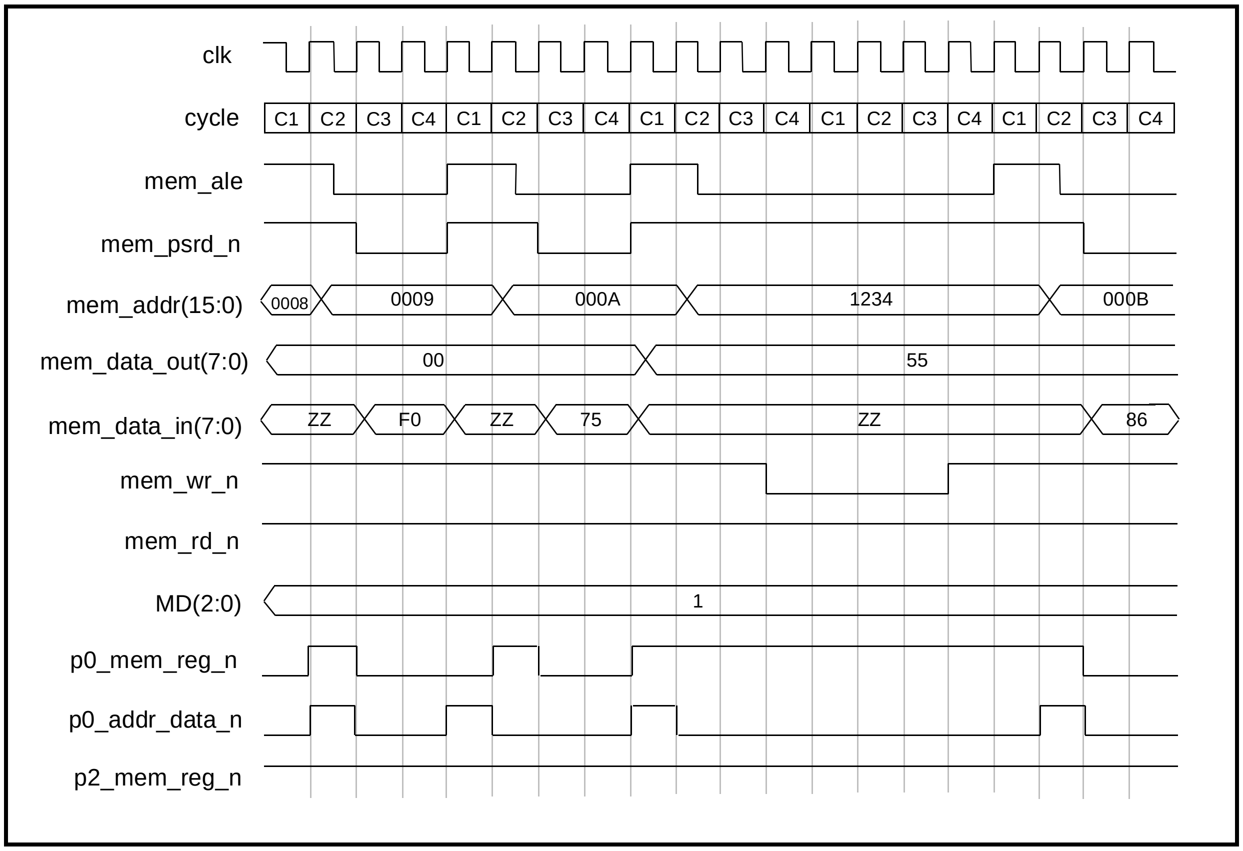


Figure 4-26   8032-Compatible I/O Signals for Data Memory Write with Stretch=1

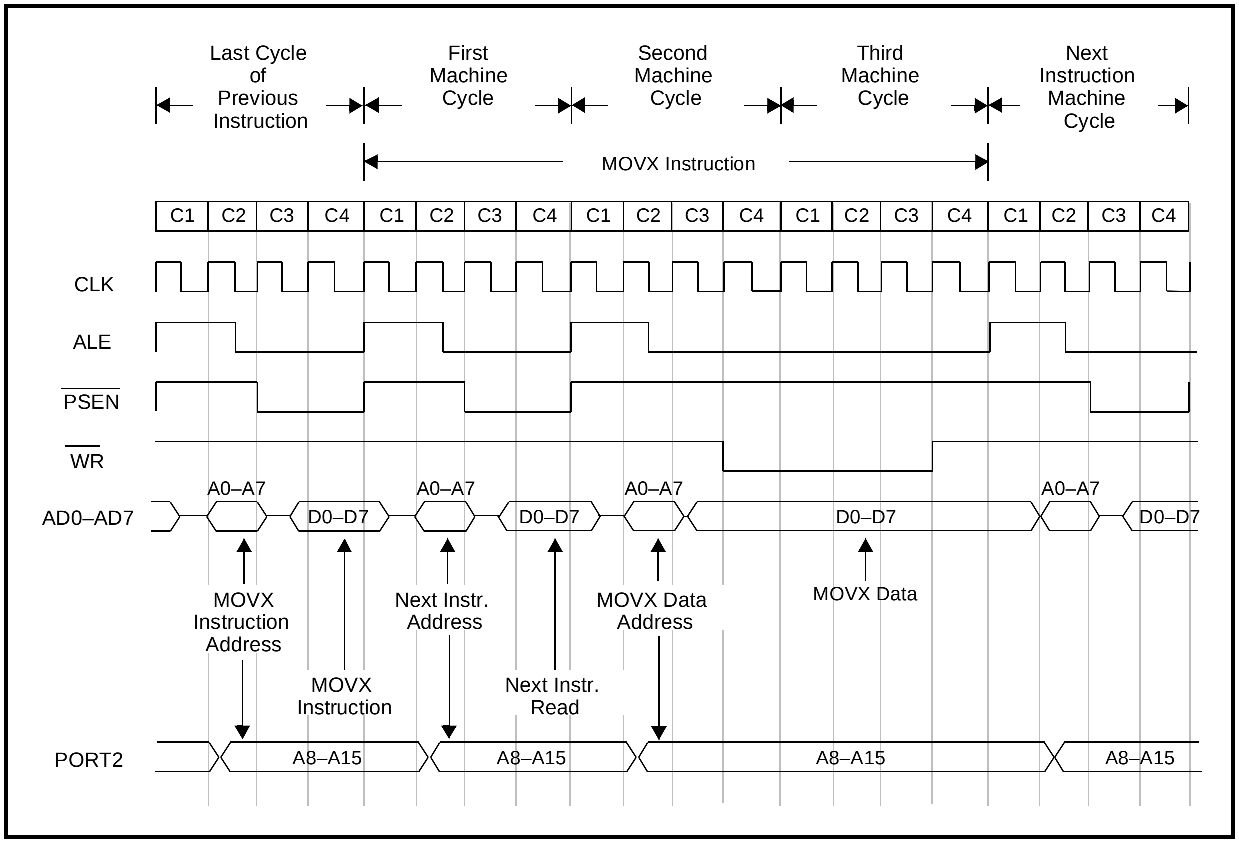


Figure 4-27 DW8051_core Signals for Data Memory Write with Stretch=2

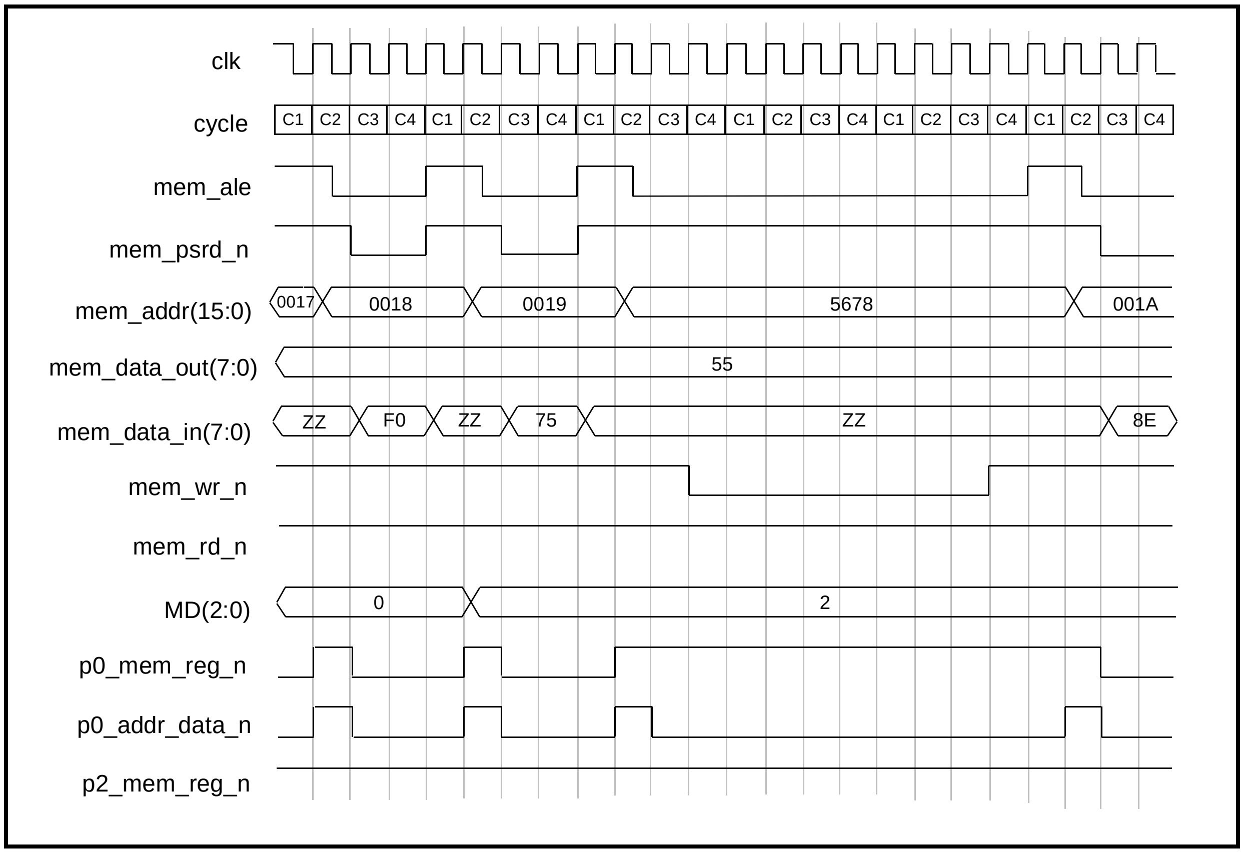
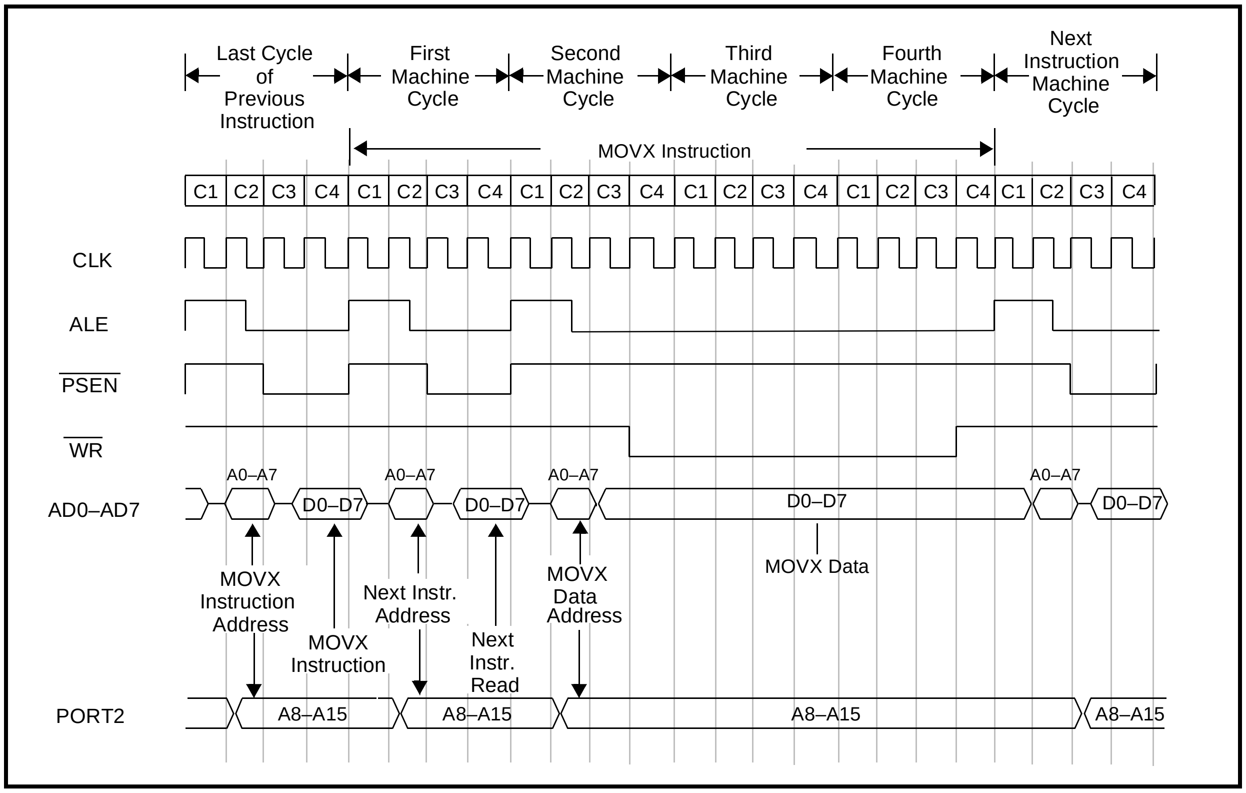


Figure 4-28 8032-Compatible I/O Signals for Data Memory Write with Stretch=2



Custom SFR Peripheral Integration

You can connect external peripherals to the external SFR bus and assign each peripheral any of the unused SFR addresses listed in Table 4-8.

Table 4-8 Available SFR Addresses for External Peripherals

80h(1)	90h(1)	93h–97h	9Ah–9Fh	A0h(1)–A7h
A9h–AFh	B0h(1)–B7h	B9h–BFh	C2h–C7h	C9h
CEh	CFh	D1h–D7h	D9h	DAh–DFh
E1h–E7h	E9h	EAh–EFh	F1h–F7h	F9h–FFh

Table 4-8 Available SFR Addresses for External Peripherals

(1) Addresses 80h, 90h, A0h, and B0h are only available if you are not using the associated standard 8051 port module (P0, P1, P2, or P3).

Figure 4-29 shows the interface between a custom peripheral and the DW8051 external SFR bus. The peripheral is assigned a free address in the SFR memory map (9Ah in this example).

The output data line *sfr_data_9A* from this peripheral is multiplexed with output data from other peripherals before being fed into the DW8051 *sfr_data_in* port. The *sfr_cs_9A* signal indicates that the DW8051 is performing a write or read access to the peripheral at address 9A.

The SFR bus is fully synchronous. The DW8051 performs all SFR bus read and write accesses within one clk cycle.

SFR Bus Write Timing

Figure 4-30 illustrates the DW8051 SFR write timing. All SFR write accesses occur in C1 of the 4-cycle instruction cycle. That is, data on *sfr_data_out* must be latched with the rising edge of *clk* at the end of C1. *sfr_wr* is active only during C1 and should therefore be used as a load enable signal.

Figure 4-29   SFR Peripheral Interfacing

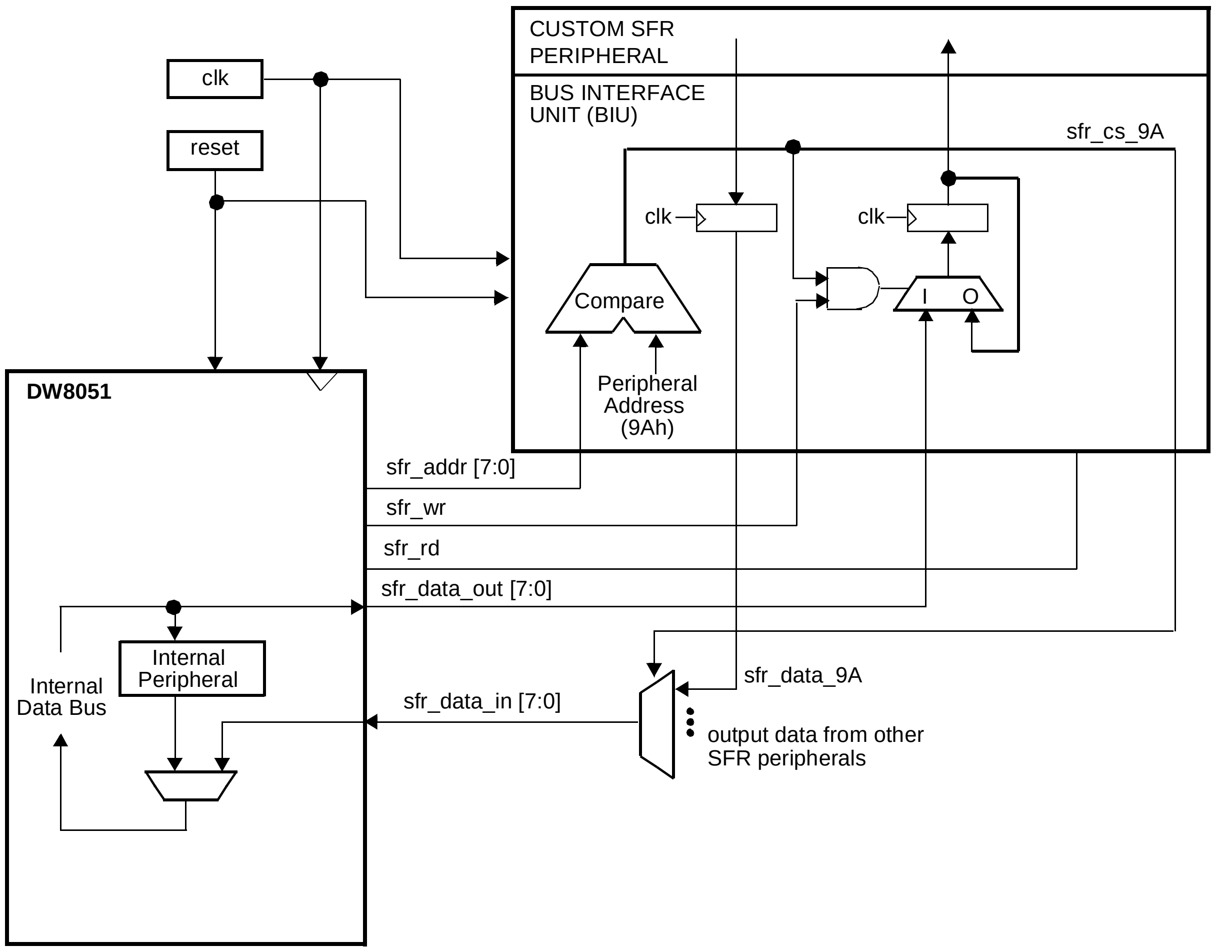
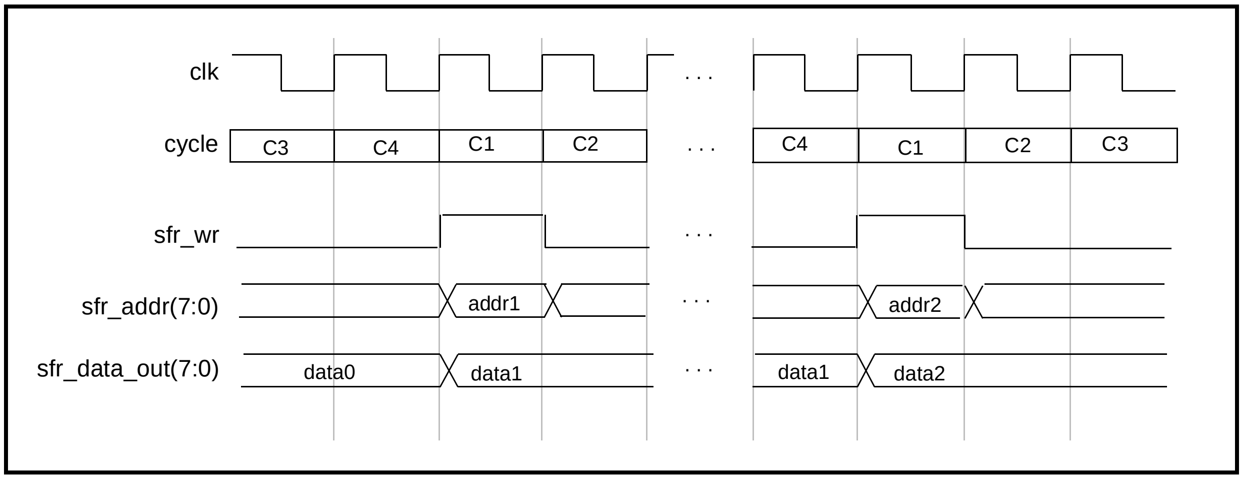


Figure 4-30 SFR Bus Write Timing



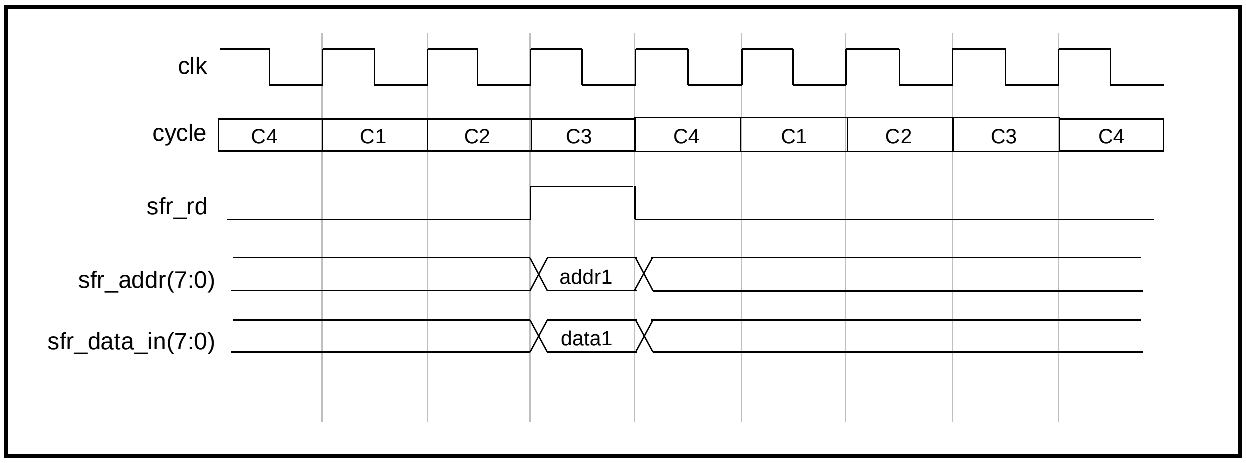
SFR Bus Read Timing

Figure 4-31 illustrates the DW8051 SFR read timing. All normal SFR read accesses occur in C3. *sfr_rd* is active during C3 to indicate a read access. However, the *sfr_rd* signal is not necessary for proper operation. Data from several external sources can be multiplexed to *sfr_data_in*, according to the decoded *sfr_addr*. DW8051_core stores the data on *sfr_data_in* at the end of C3. DW8051_core ignores all data on *sfr_data_in* except for read accesses to free SFRs.

One possible usage of *sfr_rd* is to freeze output data of a transparent latch during DW8051_core read. This provides an efficient way to implement input ports.

During execution of instructions with indirect source (for example, `MOV A, @Ri` or `MOV @Ri, A`), and for the `POP, RET`, and `RETI` instructions, *sfr_rd* is also active in C2. However, external SFRs are always read at the end of C3.

Figure 4-31 SFR Bus Read Timing



Reading Designs Back in After Layout

After floorplanning and/or layout, you back-annotate the delays based on actual wire and gate loading capacitance into Design Compiler in order to:

- Perform static timing analysis
- Incrementally compile your design in synthesis to resolve hold or setup time violations, or constraint violations.

This can be accomplished if your layout tool, or that used by your vendor, writes SDF (Standard Delay Format) files or writes out a dc_shell script file that sets loads on all nets in the design.

If SDF is to be imported, execute the `read_timing` command using the appropriate arguments. Note that the interface to the layout tool is critical for optimal design results. Refer to the *Design Compiler Reference Manual* for more information on `read_timing`.

Manufacturing Test

Test synthesis using Synopsys Test Compiler showed that the size of the design increased only by approximately 10% for most technologies while inserting a scanpath, with the methodologies full_scan and multiplexed_flip_flop. The achieved fault coverage was greater than 98%.

Any effort spent in adding logic for a full functional test would have lead to a considerable increase in gate count. However, full functional tests rarely achieve fault coverages over 90%, especially in control oriented designs such as the DW8051. Thus, the selected and currently supported manufacturing test approach is the scanpath test.

The TestReadyCompile attribute is true by default on DW8051_core, which means that Design Compiler performs a test-ready compile when you execute the coreConsultant Synthesize activity.

Scanpath insertion is typically a chip-level function. To help you with scanpath insertion for DW8051_core, Synopsys supplies an example Test Compiler script (<Workspace>/scan_chain_example/dw8051_scan_chain.scr). For scanpath testing, be aware of the following facts:

- The dw8051_scan_chain.scr script defines the *sfr_data_in*[7] port as the scanpath input (*scan_in*), and the *sfr_data_out*[7] port as the scanpath output (*scan_out*).
- The *test_mode_n* port is a mandatory primary port for scantest. *test_mode_n* is used for the following purposes:
  - Controllability of the core's reset function:
    Because the reset inputs are double-synchronized internally and then used as asynchronous resets for all modules of the

core, Test Compiler considers this to be uncontrollable (i.e., the asynchronous pin is driven by ungated seqential logic). The *test_mode_n* pin is used to gate the synchronized reset before it is applied to other modules. This ensures that Test Compiler considers the internal net to be controllable by an input port or a combination of input ports.

- Bypassing the internal RAM from the scan-chain

Because of these two functions, *test_mode_n* cannot be used to double as the *test_se* (test-scan-enable) pin. The *test_se* pin is automatically inserted by Test Compiler when the `insert_scan` command is executed.

- One flip-flop used to generate the *mem_ale* signal is running on the inverted *clk* and is, therefore, excluded from scantest.

The internal RAM and, if present, internal ROM are not tested by the scanpath test. These modules have to be tested by a functional test.

Testing Internal RAM

You can test internal RAM by executing a normal 8051 program. The assembler programs asm_tests/ram0_chk.a51.unx (for 128-byte internal RAM) and asm_tests/ram1_chk.a51.unx (for 256-byte internal RAM) provide basic tests to fully test the RAM. You can use these files as templates to create more advanced tests if needed.

Consult with your ASIC vendor for other internal RAM test methods.

Testing Internal ROM

You can also test internal ROM by executing a normal 8051 program. There are two ROM test assembler programs in the asm_tests directory: rom_test.a51.unx and rom_dump.a51.unx that you can use as examples to create your manufacturing test programs. Refer to "Internal ROM Tests" on page 5-9 for descriptions of these tests.

Consult with your ASIC vendor for other internal ROM test methods.

Software Development and Debugging

The complete timing for the DW8051 has been designed to be compatible with the Dallas DS80C320 chip. Thus, for software development/debugging, in principle, any ICE (in circuit emulator) that supports the Dallas DS80C320 can be used with the DW8051. However, because the pinout for your design normally will be different from the standard 8032 pinout, you must use an ICE that supports emulation out of external ROM.

Another strategy is to use development software that communicates to a ROM monitor.

For a list of third-party tools that have been used successfully for DW8051 software development and debugging, contact your Synopsys representative.

5

DW8051 Test Suite

The DW8051 MacroCell Solution includes an extensive test suite that you can use to verify your customized DW8051 MacroCell implementation. The coreConsultant design flow for the DW8051 MacroCell includes DW8051-specific activities that configure the test suite for your custom DW8051 configuration and execute the test programs on your selected VHDL or Verilog simulator.

For most designs, all you need to do for RTL simulation is execute the full set of test programs through the coreConsultant GUI. However, you can create your own test programs and execute them on the DW8051 testbench. To do so, you must operate the test suite directly instead of through the coreConsultant GUI.

This chapter provides background information that you need to understand how the DW8051 test suite works and how to create and execute custom test programs. The topics are:

- Understanding the DW8051 MacroCell Test Suite
- Simulation Procedures
- Creating and Executing Custom Tests

Understanding the DW8051 MacroCell Test Suite

The DW8051 MacroCell Solution includes a comprehensive test suite that tests all DW8051 opcodes, internal peripherals, and special functions. The test suite includes:

- A testbench that instantiates DW8051_core, plus models and/or processes that emulate internal and external ROM and RAM, serial port devices, an external SFR device, and a test pattern generator to test the interrupt ports
- A set of test programs
- A testbench command file that specifies which test programs the testbench will execute
- Scripts that automatically configure the testbench and run the simulation
- A set of simulation reference (“golden”) files
- A script that compares your simulation results against the simulation reference files

In addition, there are special programs available as examples of how to test internal RAM and ROM.

In order to use the DW8051 test suite to verify your DW8051 implementation, you need to understand the following topics:

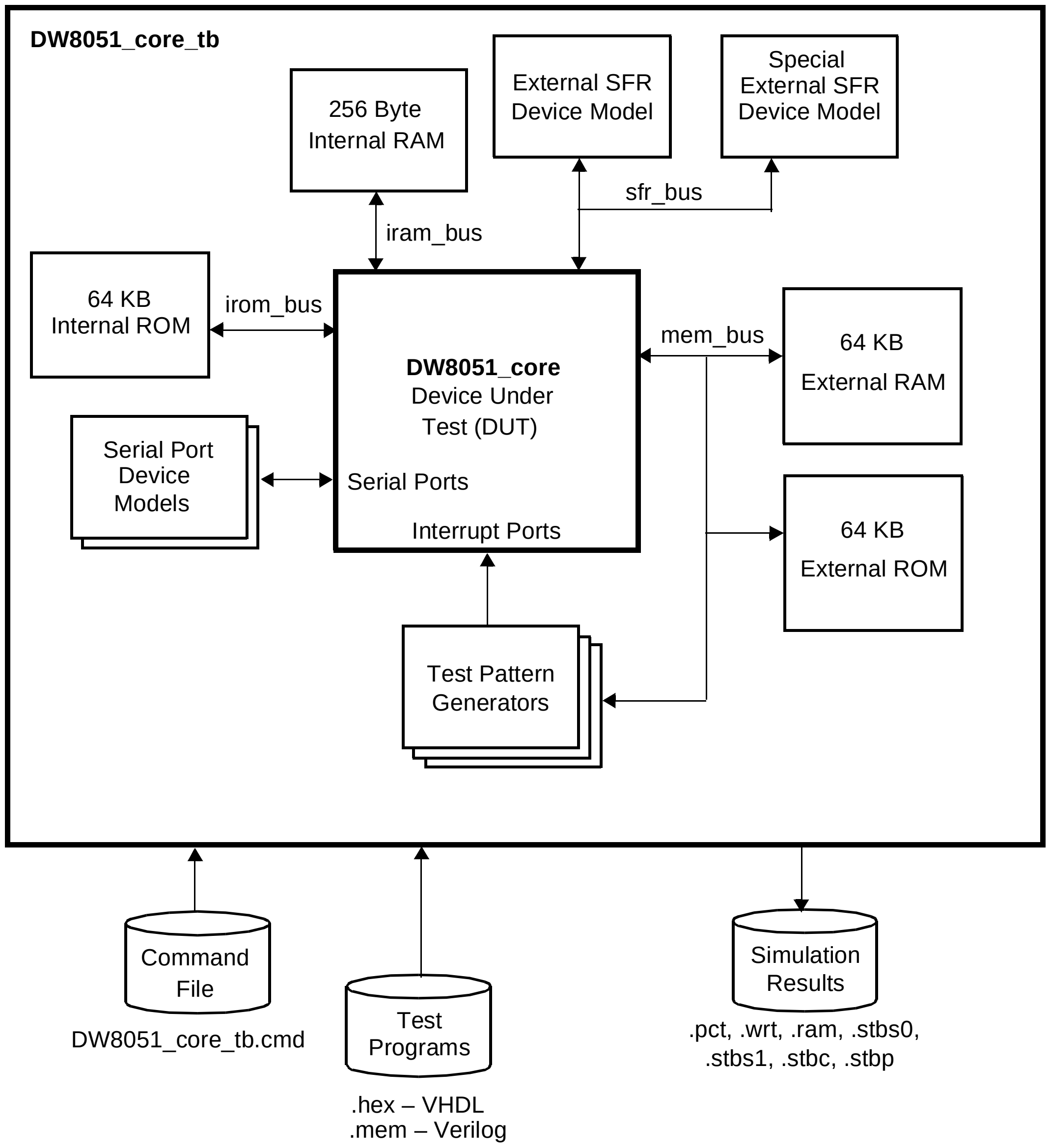
- DW8051 testbench architecture
- Test programs
- Testbench command file

DW8051 Testbench Architecture

Figure 5-1 shows a block diagram of the DW8051 testbench (DW8051_core_tb). The testbench instantiates DW8051_core as the device under test, plus several models to support testing of internal peripherals and functions:

- Processes that emulate 64 KB of external ROM and 64 KB of external RAM connected to the external memory bus (mem_bus)
- 256-byte internal RAM model connected to the internal RAM bus (iram_bus)
- Processes that emulate up to 64 KB of internal ROM connected to the internal ROM bus (irom_bus)
- Multiple instances of a test pattern generator that is used for the interrupt tests
- Two instances of a serial port device model to support the serial port tests
- An external SFR bus device model to support testing the external SFR bus
- A special external SFR device model to support the test program *sfr_iram_test*, which verifies that external SFR bus devices are not affected by internal RAM accesses

Figure 5-1   DW8051 Testbench Architecture



During simulation, DW8051_core_tb reads the testbench command file (DW8051_core_tb.cmd) and executes the test programs listed in the command file. As the tests are executed, DW8051_core_tb writes out simulation results files that contain traces of the program counter and write access to external RAM, and strobes of various DW8051_core output signals. Simulation results files are written to the directory <*workspace*>/sim_res_core.

For VHDL, the testbench source files are DW8051_core_tb.vhd (entity and architecture) and DW8051_core_tb_cfg.vhd (VHDL configuration Conf_DW8051_core_tb).

For Verilog, the testbench source file is DW8051_core_tb.v.

You can easily transfer the functionality of DW8051_core_tb to the testbench of your design. For an example of how to do so, see s8032_tb.vhd (VHDL) or s8032_tb.v (Verilog) in the example design directory (<*workspace*>/example).

Test Programs

The DW8051_core testbench (DW8051_core_tb) executes standard 8051 programs that are provided in Intel hex or in a simple hex format (one opcode per line). There are tests for all opcodes, internal peripherals, and other special functions.

Most instructions of the 8051 architecture operate on internal registers and RAM locations. Therefore, the result of an operation is not directly observable at the DW8051_core ports. To make the results observable, the DW8051 testbench writes all test results to external RAM space by use of the `MOVX @DPTR,A` instruction.

The test programs, located in the directory <*workspace*>/asm_tests, are provided in both assembler source code and in Intel hex format.

For Verilog, the simulation scripts automatically convert the test program hex files into formats that can be read by the Verilog `$readmemh` command.

Naming Conventions

The test program filenames follow the naming convention *test_name.ext*. Possible extensions (*.ext*) are:

.a51	Assembler source code (DOS file)
.a51.unx	Assembler source code (UNIX file)
.lst	List file generated by assembler
.obj	Object file generated by assembler
.hex	Intel Hex file generated by Obj-Hex converter

Opcode Tests

There are 58 tests for all possible opcodes, with the naming convention op_*opcode.ext*. There are 14 additional opcode tests that are used for testing configurations where there is only 128 bytes of RAM. These additional tests use the naming convention *oopcode_s.ext*.

Appendix A lists the opcode tests by test name, function tested, and opcode.

All opcode tests use, in addition to the opcode in test, a dedicated set of instructions that have been tested by detailed manual inspection of simulation results. These instructions are:

```
02       LJMP
12       LCALL
22       RET
90       MOV  DPTR,  #data16
90       MOV  DPTR,  #data16
A3       INC  DPTR
F0       MOVX @DPTR,  A
E5       MOV  A,  direct
```

```
E6, E7   MOV  A,  @Ri
75       MOV  direct,  #data
```

Miscellaneous Tests

In addition to the opcode tests, there are several miscellaneous test programs that verify the operation of the DW8051 internal peripherals and special functions. Table 5-1 lists the miscellaneous tests.

Table 5-1 Miscellaneous Tests

Test Name	**Function**
ram0_chk	Test for small internal RAM (128 bytes, ram_256 = 0); template for production test
ram1_chk	Test for large internal RAM (256 bytes, ram_256 = 1); template for production test
rom_dump	Dumps internal ROM contents to external RAM; template for production test
rom_test	Tests transition between internal ROM and external ROM for rom_addr_size = 13
div_0, div_1, div_2, and div_3	Extensive tests for DIV AB with all operands
idle_tst	Test for idle mode feature
int0_xx	Tests for DW8051_core with standard interrupt unit
int1_xx	Tests for DW8051_core with extended interrupt unit
int_flag	Interrupt test for DW8051 configured with extd_intr = 1 and timer2 = 1
int_tst	Tests that interrupts are recognized correctly in all instruction cycles and that the return addresses are generated properly
md_0_7	Test for DW8051 memory stretch mode feature
mpage	Test for MPAGE register

Table 5-1 Miscellaneous Tests

Test Name	Function
mul_0, mul_1, mul_2, and mul_3	Extensive tests for MUL AB with all operands
rel_jmp	Test for relative jump over $0000H boundary (SJMP)
s0_xx	Tests for DW8051_core Serial Port 0 module
s1_xx	Tests for DW8051_core Serial Port 1 module
ext_sfr	Test for external SFR bus
sfr_rst	Test for SFR reset values
sfr_iram_test	Test to verify that SFR and internal RAM accesses do not overlap
t01_out	Test for Timer 0 and Timer 1 output
t2_out	Test for Timer 2 output
t2_out_s	t2_out test for s8032
tim0_xx	Tests for Timer 0
tim1_xx	Tests for Timer 1
tim2_xx	Tests for Timer 2 module
tim_f	Tests a full 16-bit counter period for both Timer 0 and Timer 1
ext_ajmp	Special test for AJMP instruction
int_ack	Test for interrupt acknowledge timing

Internal RAM Tests

The test programs *ram0_chk* (for 128-byte internal RAM) and *ram1_chk* (for 256-byte internal RAM) provide basic tests to fully test the internal RAM simulation model. In the testbench command file, enable either *ram0_chk* or *ram1_chk*, as needed for your selected internal RAM size. The internal RAM tests are provided in assembler source code that you can modify for your implementation or to use as templates for manufacturing test programs.

Internal ROM Tests

There are two test programs for internal ROM: *rom_test* and *rom_dump*. The *rom_test* program tests the ROM address space capability of an 8-KB ROM. The *rom_dump* program reads the contents of internal ROM and dumps the contents to external RAM.

Both of the ROM test programs are written for 8-KB ROM (*rom_addr_size* = 13). If you are using a different internal ROM size, you must modify the test programs. Like all of the test programs, the internal ROM tests are provided in assembler source code that you can modify for your implementation or to use as templates for manufacturing test programs.

rom_test

The rom_test.a51.unx program tests the ROM address space capability of an 8-KB ROM (*rom_addr_size* = 13). The *rom_test* program requires a ROM contents file that the testbench automatically loads into internal and external ROM:

- For the VHDL version of the DW8051 MacroCell, the testbench automatically loads a ROM contents file (<workspace>/asm_tests/rom_test.hex) into the memory arrays that emulate internal and external ROM.
- For the Verilog version of the DW8051 MacroCell, the testbench automatically loads a ROM contents file (<workspace>/asm_tests/rom_test.mem) into a single program memory array that emulates both internal and external ROM.

If you are specifying a value other than 13 for *rom_addr_size*, you must modify and recompile the test program rom_test.a51.unx. When you recompile rom_test.a51.unx, the associated ROM contents file is automatically regenerated for the selected *rom_addr_size* value.

rom_dump

The rom_dump.a51.unx program reads the contents of internal ROM and dumps the contents to external RAM. Like the *rom_test* program, *rom_dump* is designed for an 8-KB internal ROM and must be modified to support any other internal ROM size. The *rom_dump* test is disabled by default in testbench command file (DW8051_core_tb.cmd.all).

The *rom_dump* also requires a ROM contents file that the testbench automatically loads into internal ROM:

- For the VHDL version of the DW8051 MacroCell, the testbench automatically loads a ROM contents file (<workspace>/asm_tests/rom_dump.hex) into the memory array that emulates internal ROM.
- For the Verilog version of the DW8051 MacroCell, the testbench automatically loads a ROM contents file (<workspace>/asm_tests/rom_dump.mem) into a single program memory array that emulates both internal and external ROM.

If you are specifying a value other than 13 for *rom_addr_size*, you must modify and recompile the test program rom_dump.a51.unx. When you recompile rom_dump.a51.unx, the associated ROM contents file is automatically regenerated for the selected *rom_addr_size* value.

The rom_dump.a51.unx program is based on the `MOVC A,@A+DPTR` instruction and is executed out of external code memory. To use the rom_dump.a51.unx test, the *mem_ea_n* port has to be a primary port of your design.

At start of the test, the *mem_ea_n* port has to be held low so that the test program is executed out of external code memory. Each time a `MOVC A,@A+DPTR` instruction has been read from external memory, the *mem_ea_n* port must be held high for 11 clock cycles (the `MOVC A,@A+DPTR` instruction executes in 3 instruction cycles) to fetch the code from internal ROM instead of the external ROM. Read ROM contents are then written to external RAM space and can be observed at the *mem_data_out* port.

DW8051_core_tb provides a function for automatic toggle of the *mem_ea_n* port. If enabled (*:romchk_on*), DW8051_core_tb detects a read from external ROM when the `MOVC A,@A+DPTR` opcode (93h) is used and automatically holds the *mem_ea_n* port high for 11 clock cycles.

Note:

> The ROM check program should contain no other code with data 93h.

Testbench Command File

The testbench command file DW8051_core_tb.cmd controls the operation of the testbench. The command file contains several commands and names of test programs to be executed.

coreConsultant automatically configures the testbench command file when you execute the Test Suite Configuration activity. You can also manually edit the testbench command file to add the names of any custom test programs that you want to execute.

Verilog simulators also require the command file (DW8051_core_tb.cmd) to be converted into a Verilog task (DW8051_core_tb.task). The testbench simulation script that coreConsultant generates invokes a set of supplied Perl scripts to perform this conversion automatically.

Command File Format

In the DW8051_core_tb.cmd file, comment lines can start with a hash (#), semicolon (;), or blank. Commands always start with a colon (:). Table 5-2 lists the commands that are currently accepted. All other lines are treated as names of test programs.

Command File Example

Example 5-1 shows an example testbench command file (DW8051_core_tb.cmd) that invokes the test for opcode 84h (`DIV AB`). When you run the simulation, the testbench executes the op_84 test and produces .pct, .wrt, .stbc, and .ram files.

Example 5-1 Testbench Command File for Opcode 84h

```
:op_test_ih    ; opcode test with Intel hex file input
:pct_on        ; PC trace on
:wrt_on        ; RAM write trace on
:stbc_on       ; Core signal strobe on
op_84          ; Test for DIV AB
:pct_off        ; PC trace off
:wrt_off        ; RAM write trace off
:stbc_on       ; Core signal strobe off
:eot           ; done
```

Table 5-2 Testbench Command File Commands

Command	**Function**
:pct_on	Turns on dump of PC trace for following test programs. PC traces are written to ../sim_res_core/test_name.pct.
:pct_off	Turns off dump of PC trace.
:wrt_on	Turns on dump of RAM write trace for following test programs. RAM write traces are written to ../sim_res_core/test_name.wrt.
:wrt_off	Turns off dump of RAM write trace.
:stbc_on	Turns on strobe of DW8051_core signals. Strobe files are written to ../sim_res_core/test_name.stbc.
:stbc_off	Turns off strobe of DW8051_core signals.
:stbs0_on	Turns on strobe of Serial Port 0 signals (rxd0 and txd0). (rxd0 is a combination of rxd0_in and rxd0_out).
:stbs0_off	Turns off strobe of Serial Port 0 signals.
:stbs1_on	Turns on strobe of Serial Port 1 signals (rxd1 and txd1). (rxd1 is a combination of rxd1_in and rxd1_out).
:stbs1_off	Turns off strobe of Serial Port 1 signals.
:romchk_on	Turns on toggle of mem_ea_n for ROM check.
:romchk_off	Turns off toggle of mem_ea_n for ROM check.
:op_test_sh	Defines following test_names as opcode tests with simple hex file input. Files are read from ../asm_tests/test_name.shex. shex files are designated for very simple opcode tests only. Hex opcodes (first 2 characters of a line) are read line-by-line into consecutive ROM locations, starting at 0000h. Any characters behind the first 2 characters of a line are treated as comments. Comment lines must start with a semicolon (;).

Table 5-2 Testbench Command File Commands

Command	Function
:op_test_ih	Defines following test_names as opcode tests with standard Intel hex file input. Files are read from ../asm_tests/test_name.hex. Intel hex files are accepted either in UNIX or DOS format. Thus, all programming tools for machines running UNIX or DOS can be used to develop test programs. Development of the DW8051 test suite employed a standard DOS A51 assembler, L51 linker/ locator and OBJHEX object/hex code converter.

Modifying the Testbench Command File Manually

The coreConsultant DW8051-specific Test Suite Configuration activity automatically generates a testbench command file that includes the set of test programs that you specify. However, you can only select from the list of supplied test programs. If you want to add your own custom test programs to the testbench command file, then you must edit the testbench command file manually. To do so, perform the following steps:

1. Open the file <*workspace*>/testbench/DW8051_core_tb.cmd for editing.
2. Refer to Table 5-1 and the comments in DW8051_core_tb.cmd to determine which tests you want to disable.
3. Place a comment symbol (#) at the beginning of the line of each test program you want to disable.

   The DW8051_core_tb.cmd.all file lists the opcode tests for a 256-byte internal RAM implementation of the DW8051. However, for certain opcodes, the opcode tests are different for the 128-byte internal RAM implementation. These opcode tests, listed separately in Appendix A have _s appended to the test name. If you are implementing the 128-byte internal RAM (*ram256* = 0), disable the 256-byte version of these tests and enable the

128-byte versions. The DW8051_core_tb.cmd.all file contains comments to help you select the correct opcode tests for your selected internal RAM size.

4. Save and close DW8051_core_tb.cmd.

Testbench File Conversion for Verilog

Because of the limited file input operations in Verilog, the DW8051 test programs and testbench command file must be converted into formats that can be used by the Verilog simulator. There are two Perl scripts available that perform the required conversions:

`<`*`workspace`*`>/asm_tests/ihex2mem.pl`

Converts test programs from Intel hex format to a format that can be read by the Verilog `$readmemh` command.

`<`*`workspace`*`>/testbench/cmd2task_c.pl`

Converts testbench command file into a Verilog task.

The simulation run script (Simulation.log) that coreConsultant generates when you execute the Run Simulation activity automatically invokes cmd2task_c.pl with the -c option to convert the testbench command file and all enabled test programs to the required formats before beginning the simulation. The following sections describe how the conversion scripts work for informational purposes.

Converting the Test Programs with ihex2mem.pl

Because of the limited file input operations in Verilog, Intel hex files generated from assembler, C, or PL/M sources must be converted into a format that can be read by the Verilog `$readmemh` command.

In the Verilog versions of the DW8051 MacroCell Solution, there is a Perl script, ihex2mem.pl, in the asm_tests directory that performs the required conversion for you. For example, to convert the files op94.hex and op95.hex to op94.mem and op95.mem, execute the command:

```
% perl ihex2mem.pl op94.hex op95.hex
```

Converting the Testbench Command File with cmd2task.pl

Due to the limited file read operations of Verilog, the DW8051_core_tb.cmd file must be converted into a Verilog task, DW8051_core_tb.task. The Verilog task is then included by the testbench.

In the Verilog versions of the DW8051 MacroCell Solution, there is a Perl script, cmd2task_c.pl, in the src directory that performs the required conversion for you. The cmd2task_c.pl script reads the (modified) DW8051_core_tb.cmd file and converts it to DW8051_core_tb.task. Because the .cmd file contains a list of programs to be read and executed, there is a -c (convert) option to the cmd2task_c.pl script that automatically converts the corresponding .hex files to .mem files.

For example, after you modify the DW8051_core_tb.cmd file, execute the following command to convert DW8051_core_tb.cmd to DW8051_core_tb.task and automatically convert all of the enabled test programs from .hex files to .mem files:

```
% perl cmd2task_c.pl -c
```

Simulation Procedures

coreConsultant automatically configures the testbench and testbench command file to match your selected DW8051 configuration when you execute the testbench configuration and simulation procedures through the coreConsultant GUI.

After your perform the coreConsultant DW8051-specific simulation activities at least once, you can re-execute the test suite at any time by going to your workspace testbench directory and executing the simulation run script. The name of the simulation run script is SimScript.csh unless you selected a different name when you executed the Run Simulation activity. For example, to run the default simulation run script, execute the following commands:

```
% cd <workspace>/testbench
% cd ./SimScript.csh
```

If you want to execute your own custom test programs, either instead of or in addition to the supplied test programs, then you must also manually edit the testbench command file to include your custom test programs before you run the simulation. Refer to "Creating and Executing Custom Tests" for details.

Creating and Executing Custom Tests

In addition to the test programs provided with the DW8051 test suite, you can create and execute your own custom test programs, using the DW8051 testbench. For example, you can extend the testbench to test user-defined peripherals connected to DW8051_core (for example, through the SFR bus interface) by incorporating the peripherals into the testbench and writing test programs for the peripherals.

The coreConsultant interface to the DW8051 test suite does not provide for inclusion of custom test programs. To execute your custom programs in the DW8051 test suite, you must manually modify the testbench command file and manually operate the testbench using the supplied simulation scripts.

Writing Test Programs

To create test programs, use any of the supplied assembly test programs as an example to create a new test program (my_test.a51.unx). The testbench also provides facilities for initialization and/or dump of registers/memory and error handling in test programs.

Initialization/Dump Facilities

Some of the tests use include files for initialization and/or dump of selected internal registers and memory locations. You can also use these include files in your custom test programs. The provided include files are:

init.inc.unx

Initializes internal RAM locations 00h–37h and 70h–8Fh with their own addresses.

big_dump.inc.unx

Dumps selected registers/internal memory locations to external memory (with the `MOVX @DPTR,A` instruction). The dump is 128 bytes long and matches the length of the dump of the external memory contents to the file ../sim_res_core/*test_name.ram* that occurs if the program terminates at address 0FFFCh/0FFFFh. (This function is provided by DW8051_core_tb). Table 5-3 lists the memory dump contents for big_dump.inc.unx.

bdump_s.inc.unx

Modified version of big_dump.inc.unx that does not dump contents of internal memory locations 80h–8Fh. This file is for configurations with 128 bytes of internal RAM. Table 5-4 lists the memory dump contents for bdump_s.inc.unx.

Table 5-3 Memory Dump Contents for big_dump.inc.unx

Address Range	**Contents**
00h–1Fh	Internal RAM, direct/indirect (00h–1Fh)
20h–2Fh	Bit addressable segment (20h–2Fh)
30h–37h	Scratchpad memory area 30h–37h
38h–5Fh	Selected SFRs, as listed in Table 5-5
60h–7Fh	Internal RAM, indirect (70h–8Fh)

Table 5-4 Memory Dump Contents for bdump_s.inc.unx

Address Range	**Contents**
00h–1Fh	Internal RAM, direct/indirect (00h–1Fh)
20h–2Fh	Bit addressable segment (20h–2Fh)
30h–37h	Scratchpad memory area 30h–37h
38h–5Fh	Selected SFRs, as listed in Table 5-5
60h–6Fh	Internal RAM, indirect (70h–7Fh)

Error Handling Facilities

To determine the end of a (partial) test, there must be a jump to a location in the range 0FFFAh–0FFFFh at the end of the test. DW8051_core_tb keeps track on opcode fetches from one of these addresses and stops execution, with additional actions depending on the detected address as listed in Table 5-6.

By jumping to different exit addresses a test program can report if an error has been detected and perform external RAM dumps of different sizes.

Table 5-5 Selected SFRs in Memory Dump Addresses 38h–5Fh

Addr	**Content**	**Addr**	**Content**	**Addr**	**Content**
38h	80h	46h	CKCON (@8Eh)	54h	RCAP2H (@CBh)
39h	SP (@81h)	47h	90h	55h	TL2 (@CCh)
3Ah	DPL0 (@82h)	48h	EXIF (@91h)	56h	TH2 (@CDh)
3Bh	DPH0 (@83h)	49h	SCON0 (@98h)	57h	PSW (@D0h)
3Ch	DPL1 (@84h)	4Ah	99h	58h	D8h
3Dh	DPH1 (@85h)	4Bh	A0h	59h	ACC (@E0h)
3Eh	DPS (@86h)	4Ch	IE (@A8h)	5Ah	EIE (@E8h)
3Fh	PCON (@87h)	4Dh	B0h	5Bh	B (@F0h)
40h	TCON (@88h)	4Eh	IP (@B8h)	5Ch	EIP (@F8h)
41h	TMOD (@89h)	4Fh	SCON1 (@C0h)	5Dh	–
42h	TL0 (@8Ah)	50h	C1h	5Eh	–
43h	TL1 (@8Bh)	51h	T2CON (@C8h)	5Fh	–
44h	TH0 (@8Ch)	52h	C9h	-	-

Table 5-5 Selected SFRs in Memory Dump Addresses 38h–5Fh

Addr	Content	Addr	Content	Addr	Content
45h	TH1 (@8Dh)	53h	RCAP2L (@CAh)	-	-

Table 5-6 Testbench Exit Addresses and Actions

Exit Address	Action
0FFFAh	Stop execution of test, print ”--- Error in test detected !”, no RAM dump.
0FFFBh	Stop execution of test, print ”--- Error in test detected !”, dump first 32 bytes of simulated external RAM to file ../sim_res_core/test_name.ram.
0FFFCh	Stop execution of test, print ”--- Error in test detected !”, dump first 128 bytes of simulated external RAM to file ../sim_res_core/test_name.ram.
0FFFDh	Stop execution of test, no error message, no RAM dump.
0FFFEh	Stop execution of test, no error message, dump first 32 bytes of simulated external RAM to file ../sim_res_core/test_name.ram.
0FFFFh	Stop execution of test, no error message, dump first 128 bytes of simulated external RAM to file ../sim_res_core/test_name.ram.

Assembling and Executing Custom Test Programs

After you write your custom test program, perform the following steps to assemble the test program and execute the test program in the DW8051 test suite:

1. Configure and operate the test suite at least once through coreConsultant by performing the DW8051-specific simulation activities. This step is necessary to set up the test suite.
2. Place your test program in the asm_tests directory of your workspace, for example:

```
% cd <workspace>/asm_tests
% ls my_test.*
my_test.a51.unx
```

3. Assemble the program using a standard 8051 assembler. There are a variety of shareware standard 8051 assemblers available.
4. Verify that the assembler created the correct output files:

   ```
   % ls my_test.*
   my_test.a51.unx
   my_test.hex
   my_test.lst
   ```

5. Modify the testbench command file to include your new test program. (See “Modifying the Testbench Command File Manually” on page 5-14 for details.)
6. Run the simulation and examine the results on a simulation waveform viewer.

   After you determine the test results are valid, you can use the simulation output files that the testbench writes to the sim_res_core directory as a reference files for future simulations.

7. Move the validated simulation output files to the sim_reference directory:

   ```
   % cd <workspace>/sim_res_core
   % mv my_test.* ../sim_reference
   ```

For future simulations of the new test (my_test), perform the following steps:

1. Go to the <*workspace*>/testbench directory:

   ```
   % cd <workspace>/testbench
   ```

2. Make sure that DW8051_core_tb.cmd is configured to include your custom test.

3. Execute the simulation run script (the default name for the simulation run script is SimScript.csh):

   ```
   % ./SimScript.csh
   ```

4. View the simulation log file by executing the coreConsultant View Simulation Log activity.

A

Opcode Tests

The following tables list the opcode tests in dw8051/asm_tests. Table A-1 lists the tests in order of instruction type. Table A-2 lists the tests in order of test name. Table A-3 lists the tests in order of hex opcode.

In addition, there are modified versions of 13 of the opcode tests. These tests, listed in Table A-4, use the naming convention oopcode_s and are used for configurations where there are 128 bytes of internal RAM (*ram_256* = 0).

Table A-1 Opcode Tests Sorted by Instruction Type

Instruction	**Test Name**		**Instruction**	**Test Name**
ARITHMETIC OPERATIONS			ARITHMETIC OPERATIONS	

Table A-1 Opcode Tests Sorted by Instruction Type (continued)

Instruction	**Test Name**		**Instruction**	**Test Name**
ADD A, Rn	op_28_3f		INC A	op_04_0f
ADD A, direct	op_25_35		INC Rn	op_04_0f
ADD A, @Ri	op_26_37		INC direct	op_04_0f
ADD A, #data	op_24_34		INC @Ri	op_04_0f
ADDC A, Rn	op_28_3f		DEC A	op_14_1f
ADDC A, direct	op_25_35		DEC Rn	op_14_1f
ADDC A, @Ri	op_26_37		DEC direct	op_14_1f
ADDC A, #data	op_24_34		DEC @Ri	op_14_1f
SUBB A, Rn	op_98_9f		INC DPTR	op_a3
SUBB A, direct	op_95		MUL AB	op_a4
SUBB A, @Ri	op_96_97		DIV AB	op_84
SUBB A, #data	op_94		DA A	op_d4
LOGICAL OPERATIONS			LOGICAL OPERATIONS	
ANL A, Rn	op_52_5f		XRL A, direct	op_62_6f
ANL A, direct	op_52_5f		XRL A, @Ri	op_62_6f
ANL A, @Ri	op_52_5f		XRL A, #data	op_62_6f
ANL A, #data	op_52_5f		XRL direct, A	op_62_6f
ANL direct, A	op_52_5f		XRL direct, #data	op_62_6f
ANL direct, #data	op_52_5f		CLR A	op_x3_x4
ORL A, Rn	op_42_4f		CPL A	op_x3_x4
ORL A, direct	op_42_4f		RL A	op_x3_x4

Table A-1 Opcode Tests Sorted by Instruction Type (continued)

Instruction	Test Name	Instruction	Test Name
ORL A, @Ri	op_42_4f	RLC A	op_x3_x4
ORL A, #data	op_42_4f	RR A	op_x3_x4
ORL direct, A	op_42_4f	RRC A	op_x3_x4
ORL direct, #data	op_42_4f	SWAP A	op_x3_x4
XRL A, Rn	op_62_6f	-	-
DATA TRANSFER		DATA TRANSFER	
MOV A, Rn	op_e8_ef	MOV @Ri, #data	op_76_77
MOV A, direct	op_e5	MOV DPTR, #data16	op_90_f0
MOV A, @Ri	op_e6_e7	MOVC A, @A+DPTR	op_93
MOV A, #data	op_74	MOVC A, @A+PC	op_83
MOV Rn, A	op_f8_ff	MOVX A, @Ri	op_e2_e3
MOV Rn, direct	op_a8_af	MOVX A, @DPTR	op_e0
MOV Rn, #data	op_78_7f	MOVX @Ri, A	op_7a_7b
MOV direct, A	op_f5	MOVX @DPTR, A	op_90_f0
MOV direct, Rn	op_88_8f	PUSH direct	op_c0
MOV direct, direct	op_85	POP direct	op_d0
MOV direct, @Ri	op_86_87	XCH A, Rn	op_c8_cf
MOV direct, #data	op_75	XCH A, direct	op_c5
MOV @Ri, A	op_f6_f7	XCH A, @Ri	op_c6_c7
MOV @Ri, direct	op_a6_a7	XCHD A, @Ri	op_d6_d7
BIT MANIPULATION		BIT MANIPULATION	

Table A-1 Opcode Tests Sorted by Instruction Type (continued)

Instruction	**Test Name**		**Instruction**	**Test Name**
CLR C	op_x2_x3		ORL C, /bit	op_x2_x3
CLR bit	op_x2_x3		MOV C, bit	op_x2_x3
SETB C	op_x2_x3		MOV bit, C	op_x2_x3
SETB bit	op_x2_x3		JC rel	op_40_50
CPL C	op_x2_x3		JNC rel	op_40_50
CPL bit	op_x2_x3		JB bit, rel	op_10_30
ANL C, bit	op_x2_x3		JNB bit, rel	op_10_30
ANL C, /bit	op_x2_x3		JBC bit, rel	op_10_30
ORL C, bit	op_x2_x3		-	-
PROGRAM BRANCHING			PROGRAM BRANCHING	
ACALL addr11	op_11_f1		JNZ rel	op_60_70
LCALL addr16	op_12		CJNE A, direct, rel	op_b4_bf
RET	op_22		CJNE A, #data, rel	op_b4_bf
RETI	op_32		CJNE Rn, #data, rel	op_b4_bf
AJMP addr11	op_01_e1		CJNE @Ri, #data, rel	op_b4_bf
LJMP addr16	op_02		DJNZ Rn, rel	op_d5_df
SJMP rel	op_80		DJNZ direct, rel	op_d5_df
JMP @A+DPTR	op_73		NOP	op_00_a5
JZ rel	op_60_70		-	-

Table A-2 Opcode Tests Sorted by Test Name

Test Name	Opcode(s) Tested	Instruction
op_00_a5	00, A5	NOP
op_01_e1	01, 21, 41, 61, 81, A1, C1, E1	AJMP
op_02	02	LJMP
op_04_0f	04	INC A
	05	INC direct
	06,07	INC @Ri
	08–0F	INC Rn
op_10_30	10	JBC
	20	JB
	30	JNB
op_11_f1	11, 31, 51, 71, 91, B1, D1, F1	ACALL
op_12	12	LCALL
op_14_1f	14	DEC A
	15	DEC direct
	16, 17	DEC @Ri
	18–1F	DEC Rn
op_22	22	RET
op_24_34	24	ADD A, #data
	34	ADDC A, #data
op_25_35	25	ADD A, direct

Table A-2 Opcode Tests Sorted by Test Name (continued)

Test Name	**Opcode(s) Tested**	**Instruction**
	35	ADDC A, direct
op_26_37	26, 27	ADD A, @Ri
	36, 37	ADDC A, @Ri
op_28_3f	28–2F	ADD A, Rn
	38–3F	ADDC A, Rn
op_32	32	RETI
op_40_50	40	JC
	50	JNC
op_42_4f	42	ORL direct, A
	43	ORL direct, #data
	44	ORL A, #data
	45	ORL A, direct
	46, 47	ORL A, @Ri
	48–4F	ORL A, Rn
op_52_5f	52	ANL direct, A
	53	ANL direct, #data
	54	ANL A, #data
	55	ANL A, direct
	56, 57	ANL A, @Ri
	58–5F	ANL A, Rn
op_60_70	60	JZ

Table A-2   Opcode Tests Sorted by Test Name (continued)

Test Name	**Opcode(s) Tested**	**Instruction**
	70	JNZ
op_62_6f	62	XRL direct, A
	63	XRL direct, #data
	64	XRL A, #data
	65	XRL A, direct
	66, 67	XRL A, @Ri
	68–6F	XRL A, Rn
op_73	73	JMP @A+DPTR
op_74	74	MOV A, #data
op_75	75	MOV direct, #data
op_76_77	76, 77	MOV @Ri, #data
op_78_7f	78–7F	MOV Rn, #data
op_80	80	SJMP
op_83	83	MOVC A, @A+PC
op_84	84	DIV AB
op_85	85	MOV direct, direct
op_86_87	86, 87	MOV direct, @Ri
op_88_8f	88–8F	MOV direct, Rn
op_90_f0	90	MOV DPTR, #data
	F0	MOVX @DPTR, A
op_93	93	MOVC A, @A+DPTR

Table A-2 Opcode Tests Sorted by Test Name (continued)

Test Name	**Opcode(s) Tested**	**Instruction**
op_94	94	SUBB A, #data
op_95	95	SUBB A, direct
op_96_97	96, 97	SUBB A, @Ri
op_98_9f	98–9F	SUBB A, Rn
op_a3	A3	INC DPTR
op_a4	A4	MUL AB
op_a6_a7	A6, A7	MOV @Ri, direct
op_a8_af	A8–AF	MOV Rn, direct
op_b4_bf	B4	CJNE A, #data
	B5	CJNE A, direct
	B6, B7	CJNE @Ri, #data
	B8–BF	CJNE Rn, #data
op_c0_d0	C0	PUSH direct
	D0	POP direct
op_c5	C5	XCH A, direct
op_c6_c7	C6, C7	XCH A, @Ri
op_c8_cf	C8–CF	XCH A, Rn
op_d4	D4	DA A
op_d5_df	D5	DJNZ direct
	D8–DF	DJNZ Rn
op_d6_d7	D6, D7	XCHD A, @Ri

Table A-2 Opcode Tests Sorted by Test Name (continued)

Test Name	**Opcode(s) Tested**	**Instruction**
op_e0	E0	MOVX A, @DPTR
op_e2_f3	E2, E3	MOVX A, @Ri
	F2, F3	MOVX @Ri, A
op_e5	E5	MOV A, direct
op_e6_f7	E6, E7	MOV A, @Ri
op_e8_ef	E8–EF	MOV A, Rn
op_f5	F5	MOV direct, A
op_f6_f7	F6, F7	MOV @Ri, A
op_f8_ff	F8–FF	MOV Rn, A
op_x2_x3	A2	MOV C, bit
	92	MOV bit, C
	82	ANL C, bit
	B0	ANL C, /bit
	72	ORL C, bit
	A0	ORL C, /bit
	C3	CLR C
	C2	CLR bit
	B3	CPL C
	B2	CPL bit
	D3	SETB C
	D2	SETB bit

Table A-2 Opcode Tests Sorted by Test Name (continued)

Test Name	Opcode(s) Tested	Instruction
op_x3_x4	E4	CLR A
	F4	CPL A
	23	RL A
	33	RLC A
	03	RR A
	13	RRC A
	C4	SWAP A

Table A-3 Opcode Tests Sorted by Opcode

Opcode	Mnemonic	Test Name	Opcode	Mnemonic	Test Name
00	NOP	op_00_a5	80	SJMP	op_80
01	AJMP	op_01_e1	81	AJMP	op_01_e1
02	LJMP	op_02	82	ANL C,bit	op_x2_x3
03	RR A	op_x3_x4	83	MOVC A,@A+PC	op_83
04	INC A	op_04_0f	84	DIV AB	op_84
05	INC direct	op_04_0f	85	MOV dir,dir	op_85
06	INC @R0	op_04_0f	86	MOV dir,@R0	op_86_87
07	INC @R1	op_04_0f	87	MOV dir,@R1	op_86_87
08	INC R0	op_04_0f	88	MOV dir,R0	op_88_8f
09	INC R1	op_04_0f	89	MOV dir,R1	op_88_8f

Table A-3 Opcode Tests Sorted by Opcode (continued)

Opcode	Mnemonic	Test Name	Opcode	Mnemonic	Test Name
0A	INC R2	op_04_0f	8A	MOV dir,R2	op_88_8f
0B	INC R3	op_04_0f	8B	MOV dir,R3	op_88_8f
0C	INC R4	op_04_0f	8C	MOV dir,R4	op_88_8f
0D	INC R5	op_04_0f	8D	MOV dir,R5	op_88_8f
0E	INC R6	op_04_0f	8E	MOV dir,R6	op_88_8f
0F	INC R7	op_04_0f	8F	MOV dir,R7	op_88_8f
10	JBC	op_10_30	90	MOV DPTR,#data	op_90_f0
11	ACALL	op_11_f1	91	ACALL	op_11_f1
12	LCALL	op_12	92	MOV bit,C	op_x2_x3
13	RRC A	op_x3_x4	93	MOVC A,@A+DPTR	op_93
14	DEC A	op_14_1f	94	SUBB A,#data	op_94
15	DEC direct	op_14_1f	95	SUBB A,dir	op_95
16	DEC @R0	op_14_1f	96	SUBB A,@R0	op_96_97
17	DEC @R1	op_14_1f	97	SUBB A,@R1	op_96_97
18	DEC R0	op_14_1f	98	SUBB A,R0	op_98_9f
19	DEC R1	op_14_1f	99	SUBB A,R1	op_98_9f
1A	DEC R2	op_14_1f	9A	SUBB A,R2	op_98_9f
1B	DEC R3	op_14_1f	9B	SUBB A,R3	op_98_9f
1C	DEC R4	op_14_1f	9C	SUBB A,R4	op_98_9f
1D	DEC R5	op_14_1f	9D	SUBB A,R5	op_98_9f

Table A-3 Opcode Tests Sorted by Opcode (continued)

Opcode	Mnemonic	Test Name	Opcode	Mnemonic	Test Name
1E	DEC R6	op_14_1f	9E	SUBB A,R6	op_98_9f
1F	DEC R7	op_14_1f	9F	SUBB A,R7	op_98_9f
20	JB	op_10_30	A0	ORL C,/bit	op_x2_x3
21	AJMP	op_01_e1	A1	AJMP	op_01_e1
22	RET	op_22	A2	MOV C,bit	op_x2_x3
23	RL A	op_x3_x4	A3	INC DPTR	op_a3
24	ADD A,#data	op_24_34	A4	MUL AB	op_a4
25	ADD A,dir	op_25_35	A5	reserved	op_00_a5
26	ADD A,@R0	op_26_37	A6	MOV @R0,dir	op_a6_a7
27	ADD A,@R1	op_26_37	A7	MOV @R1,dir	op_a6_a7
28	ADD A,R0	op_28_3f	A8	MOV R0,dir	op_a8_af
29	ADD A,R1	op_28_3f	A9	MOV R1,dir	op_a8_af
2A	ADD A,R2	op_28_3f	AA	MOV R2,dir	op_a8_af
2B	ADD A,R3	op_28_3f	AB	MOV R3,dir	op_a8_af
2C	ADD A,R4	op_28_3f	AC	MOV R4,dir	op_a8_af
2D	ADD A,R5	op_28_3f	AD	MOV R5,dir	op_a8_af
2E	ADD A,R6	op_28_3f	AE	MOV R6,dir	op_a8_af
2F	ADD A,R7	op_28_3f	AF	MOV R7,dir	op_a8_af
30	JNB	op_10_30	B0	ANL C,/bit	op_x2_x3
31	ACALL	op_11_f1	B1	ACALL	op_11_f1
32	RETI	op_32	B2	CPL bit	op_x2_x3

Table A-3   Opcode Tests Sorted by Opcode (continued)

Opcode	Mnemonic	Test Name	Opcode	Mnemonic	Test Name
33	RLC A	op_x3_x4	B3	CPL C	op_x2_x3
34	ADDC A,#data	op_24_34	B4	CJNE A,#data	op_b4_bf
35	ADDC A,dir	op_25_35	B5	CJNE A,dir	op_b4_bf
36	ADDC A,@R0	op_26_37	B6	CJNE @R0,#data	op_b4_bf
37	ADDC A,@R1	op_26_37	B7	CJNE @R1,#data	op_b4_bf
38	ADDC A,R0	op_28_3f	B8	CJNE R0,#data	op_b4_bf
39	ADDC A,R1	op_28_3f	B9	CJNE R1,#data	op_b4_bf
3A	ADDC A,R2	op_28_3f	BA	CJNE R2,#data	op_b4_bf
3B	ADDC A,R3	op_28_3f	BB	CJNE R3,#data	op_b4_bf
3C	ADDC A,R4	op_28_3f	BC	CJNE R4,#data	op_b4_bf
3D	ADDC A,R5	op_28_3f	BD	CJNE R5,#data	op_b4_bf
3E	ADDC A,R6	op_28_3f	BE	CJNE R6,#data	op_b4_bf
3F	ADDC A,R7	op_28_3f	BF	CJNE R7,#data	op_b4_bf
40	JC	op_40_50	C0	PUSH direct	op_c0_d0
41	AJMP	op_01_e1	C1	AJMP	op_01_e1
42	ORL dir,A	op_42_4f	C2	CLR bit	op_x2_x3
43	ORL dir,#data	op_42_4f	C3	CLR C	op_x2_x3
44	ORL A,#data	op_42_4f	C4	SWAP A	op_x3_x4
45	ORL A,dir	op_42_4f	C5	XCH A,dir	op_c5
46	ORL A,@R0	op_42_4f	C6	XCH A,@R0	op_c6_c7
47	ORL A,@R0	op_42_4f	C7	XCH A,@R1	op_c6_c7

Table A-3 Opcode Tests Sorted by Opcode (continued)

Opcode	Mnemonic	Test Name	Opcode	Mnemonic	Test Name
48	ORL A,R0	op_42_4f	C8	XCH A,R0	op_c8_cf
49	ORL A,R1	op_42_4f	C9	XCH A,R1	op_c8_cf
4A	ORL A,R2	op_42_4f	CA	XCH A,R2	op_c8_cf
4B	ORL A,R3	op_42_4f	CB	XCH A,R3	op_c8_cf
4C	ORL A,R4	op_42_4f	CC	XCH A,R4	op_c8_cf
4D	ORL A,R5	op_42_4f	CD	XCH A,R5	op_c8_cf
4E	ORL A,R6	op_42_4f	CE	XCH A,R6	op_c8_cf
4F	ORL A,R7	op_42_4f	CF	XCH A,R7	op_c8_cf
50	JNC	op_40_50	D0	POP direct	op_c0_d0
51	ACALL	op_11_f1	D1	ACALL	op_11_f1
52	ANL dir,A	op_52_5f	D2	SETB bit	op_x2_x3
53	ANL dir,#data	op_52_5f	D3	SETB C	op_x2_x3
54	ANL A,#data	op_52_5f	D4	DA A	op_d4
55	ANL A,dir	op_52_5f	D5	DJNZ direct	op_d5_df
56	ANL A,@R0	op_52_5f	D6	XCHD A,@R0	op_d6_d7
57	ANL A,@R1	op_52_5f	D7	XCHD A,@R1	op_d6_d7
58	ANL A,R0	op_52_5f	D8	DJNZ R0	op_d5_df
59	ANL A,R1	op_52_5f	D9	DJNZ R1	op_d5_df
5A	ANL A,R2	op_52_5f	DA	DJNZ R2	op_d5_df
5B	ANL A,R3	op_52_5f	DB	DJNZ R3	op_d5_df
5C	ANL A,R4	op_52_5f	DC	DJNZ R4	op_d5_df

Table A-3 Opcode Tests Sorted by Opcode (continued)

Opcode	**Mnemonic**	**Test Name**	**Opcode**	**Mnemonic**	**Test Name**
5D	ANL A,R5	op_52_5f	DD	DJNZ R5	op_d5_df
5E	ANL A,R6	op_52_5f	DE	DJNZ R6	op_d5_df
5F	ANL A,R7	op_52_5f	DF	DJNZ R7	op_d5_df
60	JZ	op_60_70	E0	MOVX A,@DPTR	op_e0
61	AJMP	op_01_e1	E1	AJMP	op_01_e1
62	XRL dir,A	op_62_6f	E2	MOVX A,@R0	op_e2_f3
63	XRL dir,#data	op_62_6f	E3	MOVX A,@R1	op_e2_f3
64	XRL A,#data	op_62_6f	E4	CLR A	op_x3_x4
65	XRL A,dir	op_62_6f	E5	MOV A,dir	op_e5
66	XRL A,@R0	op_62_6f	E6	MOV A,@R0	op_e6_e7
67	XRL A,@R1	op_62_6f	E7	MOV A,@R1	op_e6_e7
68	XRL A,R0	op_62_6f	E8	MOV A,R0	op_e8_ef
69	XRL A,R1	op_62_6f	E9	MOV A,R1	op_e8_ef
6A	XRL A,R2	op_62_6f	EA	MOV A,R2	op_e8_ef
6B	XRL A,R3	op_62_6f	EB	MOV A,R3	op_e8_ef
6C	XRL A,R4	op_62_6f	EC	MOV A,R4	op_e8_ef
6D	XRL A,R5	op_62_6f	ED	MOV A,R5	op_e8_ef
6E	XRL A,R6	op_62_6f	EE	MOV A,R6	op_e8_ef
6F	XRL A,R7	op_62_6f	EF	MOV A,R7	op_e8_ef
70	JNZ	op_60_70	F0	MOVX @DPTR,A	op_90_f0
71	ACALL	op_11_f1	F1	ACALL	op_11_f1

Table A-3 Opcode Tests Sorted by Opcode (continued)

Opcode	Mnemonic	Test Name	Opcode	Mnemonic	Test Name
72	ORL C,bit	op_x2_x3	F2	MOVX @R0,A	op_e2_f3
73	JMP @A+DPTR	op_73	F3	MOVX @R1,A	op_e2_f3
74	MOV A,#data	op_74	F4	CPL A	op_x3_x4
75	MOV dir,#data	op_75	F5	MOV dir,A	op_f5
76	MOV @R0,#data	op_76_77	F6	MOV @R0,A	op_f6_f7
77	MOV @R1,#data	op_76_77	F7	MOV @R1,A	op_f6_f7
78	MOV R0,#data	op_78_7f	F8	MOV R0,A	op_f8_ff
79	MOV R1,#data	op_78_7f	F9	MOV R1,A	op_f8_ff
7A	MOV R2,#data	op_78_7f	FA	MOV R2,A	op_f8_ff
7B	MOV R3,#data	op_78_7f	FB	MOV R3,A	op_f8_ff
7C	MOV R4,#data	op_78_7f	FC	MOV R4,A	op_f8_ff
7D	MOV R5,#data	op_78_7f	FD	MOV R5,A	op_f8_ff
7E	MOV R6,#data	op_78_7f	FE	MOV R6,A	op_f8_ff
7F	MOV R7,#data	op_78_7f	FF	MOV R7,A	op_f8_ff

Table A-4 Opcode Tests for 128-Byte Internal RAM Configurations

Test Name	Opcode(s) Tested	Instruction
o75_s	75	MOV direct, #data
o76_77_s	76, 77	MOV @Ri, #data

Table A-4 Opcode Tests for 128-Byte Internal RAM Configurations

Test Name	**Opcode(s) Tested**	**Instruction**
o85_s	85	MOV direct, direct
o86_87_s	86, 87	MOV direct, @Ri
o88_8f_s	88–8F	MOV direct, Rn
oa6_a7_s	A6, A7	MOV @Ri, direct
oa8_af_s	A8–AF	MOV Rn, direct
oc0_d0_s	C0	PUSH direct
	D0	POP direct
oc6_c7_s	C6, C7	XCH A, @Ri
od6_d7_s	D6, D7	XCHD A, @Ri
oe6_e7_s	E6, E7	MOV A, @Ri
of5_s	F5	MOV direct, A
of6_f7_s	F6, F7	MOV @Ri, A
of8_ff_s	F8–FF	MOV Rn, A

B

DW8051/DS80C320 Differences

The DW8051 MacroCell is similar to the DS80C320 in terms of hardware features and instruction cycle timing. However, there are some important implementation differences between the DW8051 and the DS80C320 in the following features:

- Serial Ports
- Timer 2
- Watchdog Timer
- Power Fail Detector
- Stop Mode
- Timed Access Protection
- Parallel Ports

Serial Ports

The DW8051 does not implement serial port framing error detection and does not implement slave address comparison for multiprocessor communications. Therefore, the DW8051 also does not implement the following SFRs: SADDR0, SADDR1, SADEN0, and SADEN1.

Timer 2

The DW8051 does not implement Timer 2 downcounting mode or the downcount enable bit (T2MOD, bit 0). Also, the DW8051 does not implement Timer 2 output enable (T2OE) bit (TMOD2, bit 1). Therefore, the T2MOD SFR is also not implemented in the DW8051 core. However, for applications that require T2OE functionality (for example, applications that require a standard 8051 P1 port module), you can build a T2MOD SFR and connect it to the SFR bus.

Also, the DW8051 Timer 2 overflow output is active for one clock cycle. In the DS80C320, the Timer 2 overflow output is a square wave with a 50% duty cycle.

Watchdog Timer

The DW8051 does not include an internal watchdog timer. However, the DW8051 does have a dedicated input port (*wdti*) for an interrupt from an external watchdog timer. The DW8051 also implements the watchdog timer interrupt flag, enable, and priority control bits for the external watchdog timer interrupt, but does not implement automatic watchdog timer reset. Because the watchdog timer control bits (WRTF, EWT, and RWT) are not implemented, the DS80C320

WDCON SFR (at SFR address D8h) is named EICON in the DW8051, and is implemented only when the extended interrupt unit is implemented.

Power Fail Detector

The DW8051 does not include an internal power fail detector. However, the DW8051 does have a dedicated input port (*pfi*) for an interrupt from an external power fail detector. The DW8051 also implements the power fail interrupt flag and enable bits for the external power fail interrupt.

Stop Mode

The clock is not gated in the DW8051 as it is in the DS80C320. However, the DW8051 internal cycle counter is reset in stop mode and, because most internal operations are controlled by the cycle counter, no internal flip-flops change state in stop mode and there is a significant reduction in power consumption.

In addition, the DW8051 exits stop mode only when reset, whereas the DS80C320 also exits stop mode through external interrupts or power fail interrupt. Therefore, the DW8051 does not implement a ring oscillator and does not implement the RGMD, RGSL, and BGS bits of the EXIF SFR (at SFR address 91h).

Timed Access Protection

The DW8051 does not implement timed access protection and therefore, does not implement the TA SFR.

Parallel Ports

The DW8051 does not implement the DS80C320 multiplexed parallel ports (P0–P3). Instead, the DW8051 provides dedicated ports for memory interface, interrupts, serial interface, and other I/O functions. However, the DW8051 provides all the control signals required to operate parallel port modules. The example design provided with the DW8051 MacroCell includes source code for the parallel port modules and illustrates how to connect and control the parallel port modules to provide a DS80C320 compatible interface.

Index