Chipsmall
深圳市津利帝科技有限公司
Geelead Technology Limited
采购部

Core8051

DirectCore

Product Summary

Intended Use

- Embedded System Control
- Communication System Control
- I/O Control

Key Features

- 100% ASM51 (8051/80C31/80C51) Compatible Instruction Set[1]
- Control Unit
  - 8-Bit Instruction Decoder
  - Reduced Instruction Time of up to 12 Cycles
- Arithmetic Logic Unit
  - 8-Bit Arithmetic and Logical Operations
  - Boolean Manipulations
  - 8 by 8-Bit Multiplication and 8 by 8-Bit Division
- 32-Bit I/O Ports
  - Four 8-Bit I/O Ports
  - Alternate Port Functions, such as External Interrupts, Provide Extra Port Pins when Compared with the Standard 8051
- Serial Port
  - Simultaneous Transmit and Receive
  - Synchronous Mode, Fixed Baud Rate
  - 8-Bit UART Mode, Variable Baud Rate
  - 9-Bit UART Mode, Fixed Baud Rate
  - 9-Bit UART Mode, Variable Baud Rate
  - Multiprocessor Communication
- Two 16-Bit Timer/Counters
- Interrupt Controller
  - Four Priority Levels with 13 Interrupt Sources
- Internal Data Memory Interface
  - Can Address up to 256B of Data Memory Space
- External Memory Interface
  - Can Address up to 64kB of External Program Memory
  - Can Address up to 64kB of External Data Memory
  - Demultiplexed Address/Data Bus Enables Easy Connection to Memory
  - Variable Length MOVX to Access Fast/Slow RAM or Peripherals
  - Wait Cycles to Access Fast/Slow ROM
  - Dual Data Pointer to Fast Data Block Transfer
- Special Function Register (SFR) Interface
  - Services up to 101 External SFRs
- Optional On-Chip Instrumentation (OCI) Debug Logic
- Supports all Major Actel Device Families
- Optional Power-Saving Modes

Supported Families

- Fusion
- ProASIC3/E
- ProASIC$^{\underline{PLUS}}$
- Axcelerator
- RTAX-S
- SX-A
- RTSX-S

Core Deliverables

- Evaluation Version
  - Compiled RTL Simulation Model Fully Supported in the Actel Libero® Integrated Design Environment (IDE)
- Netlist Version
  - Structural Verilog and VHDL Netlists (with and without I/O Pads) Compatible with the Actel Designer Software Place-and-Route Tool
  - Compiled RTL Simulation Model Fully Supported in Actel Libero IDE
- RTL Version
  - Verilog and VHDL Core Source Code
  - Core Synthesis Scripts
- Testbench (Verilog and VHDL)

Synthesis and Simulation Support

- Synthesis
  - Synplicity®
  - Synopsys® (Design Compiler™, FPGA Compiler™, FPGA Express™)
  - Exemplar™
- Simulation
  - OVI - Compliant Verilog Simulators
  - Vital - Compliant VHDL Simulators

1. *For more information, see the Core8051 Instruction Set Details User's Guide*

Core Verification

- Comprehensive VHDL and Verilog Testbenches
- Users Can Easily Add Custom Tests by Modifying the User Testbench Using the Existing Format

Contents

General Description

The Core8051 macro is a high-performance, single-chip, 8-bit microcontroller. It is a fully functional eight-bit embedded controller that executes all ASM51 instructions and has the same instruction set as the 80C31. Core8051 provides software and hardware interrupts, a serial port, and two timers.

The Core8051 architecture eliminates redundant bus states and implements parallel execution of fetch and execution phases. Since a cycle is aligned with memory fetch when possible, most of the one-byte instructions are performed in a single cycle. Core8051 uses one clock per cycle. This leads to an average performance improvement rate of 8.0 (in terms of MIPS) with respect to the Intel device working with the same clock frequency.

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

Table 1 shows the speed advantage of Core8051 over the standard 8051. A speed advantage of 12 in the first column means that Core8051 performs the same instruction 12 times faster than the standard 8051. The second column in Table 1 lists the number of types of instructions that have the given speed advantage. The third column lists the total number of instructions that have the given speed advantage. The third column can be thought of as a subcategory of the second column. For example, there are two types of instructions that have a three-time speed advantage over the classic 8051, for which there are nine explicit instructions.

Table 1 • **Core8051 Speed Advantage Summary**

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

The average speed advantage is 8.0. However, the real speed improvement seen in any system will depend on the instruction mix.

Core8051 consists of the following primary blocks:

- Memory Control Block – Logic that Controls Program and Data Memory
- Control Processor Block – Main Controller Logic
- RAM and SFR Control Block
- ALU – Arithmetic Logic Unit
- Reset Control Block – Provides Reset Condition Circuitry
- Clock Control Block
- Timer 0 and 1 Block
- ISR – Interrupt Service Routine Block
- Serial Port Block
- Port Registers Block
- PMU – Power Management Unit Block
- OCI block – On-Chip Instrumentation Logic for Debug Capabilities

Figure 1 shows the primary blocks of Core8051.

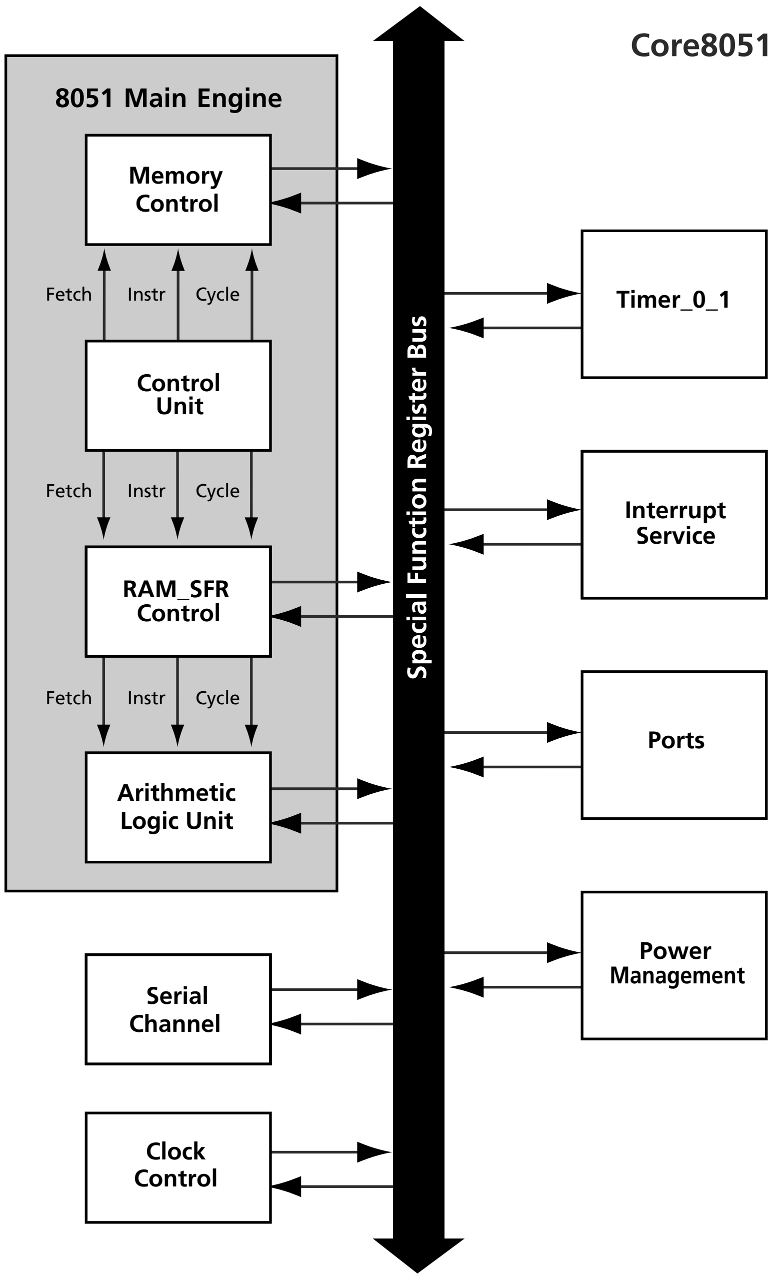

Figure 1 • **Core8051 Block Diagram**

Core8051 Device Requirements

Core8051 has been implemented in several of the Actel device families. A summary of the implementation data is listed in Table 2 through Table 4. Table 2 lists implementation data without OCI logic.

Table 2 • **Core8051 Device Utilization and Performance - No OCI**

Family	Cells or Tiles				Utilization		Performance
	Sequential	Combinatorial	Total	RAM Blocks	Device	Total	
Fusion	528	3629	4157	1	AFS600	30%	36 MHz
ProASIC3/E	528	3629	4157	1	A3PE600-2	30%	36 MHz
ProASIC<u>PLUS</u>	528	3909	4437	1	APA150-STD	72%	24 MHz
Axcelerator	619	2344	2963	1	AX250-3	70%	52 MHz
RTAX-S	619	2344	2963	1	RTAX1000S-1	16%	29 MHz
SX-A	646	2780	3426	-	A54SX72A-3	57%	33 MHz
RTSX-S	646	2780	3426	-	RT54SX72S-1	57%	19 MHz

Note: Data in this table was achieved using typical synthesis and layout settings. Performance was achieved using the Core8051 macro alone.

Table 3 lists implementation data with OCI logic (no trace memory and no hardware triggers).

Table 3 • **Core8051 Device Utilization and Performance - OCI without Trace Memory and Hardware Trigger**

Family	Cells or Tiles				Utilization		Performance
	Sequential	Combinatorial	Total	RAM Blocks	Device	Total	
Fusion	621	3923	4544	1	AFS600	33%	33 MHz
ProASIC3/E	621	3923	4544	1	A3PE600-2	33%	33 MHz
ProASIC<u>PLUS</u>	621	4249	4870	1	APA150-STD	79%	20 MHz
Axcelerator	739	2646	3385	1	AX500-3	42%	44 MHz
RTAX-S	739	2646	3385	1	RTAX1000S-1	19%	25 MHz
SX-A	765	2914	3679	-	A54SX72A-3	61%	29 MHz
RTSX-S	765	2914	3679	-	RT54SX72S-1	61%	19 MHz

Note: Data in this table was achieved using typical synthesis and layout settings. Performance was achieved using the Core8051 macro alone.

Table 4 lists implementation data with OCI logic (256-word trace memory and one hardware trigger).

Table 4 • **Core8051 Device Utilization and Performance - OCI with 256-Word Trace Memory and One Hardware Trigger**

Family	Cells or Tiles				Utilization		Performance
	Sequential	Combinatorial	Total	RAM Blocks	Device	Total	
Fusion	718	4323	5041	3	AFS600	37%	33 MHz
ProASIC3/E	718	4323	5041	3	A3PE600-2	37%	33 MHz
ProASIC<u>PLUS</u>	717	4709	5426	4	APA150-STD	88%	20 MHz
Axcelerator	843	3023	3866	3	AX500-3	48%	40 MHz
RTAX-S	843	3023	3866	3	RTAX1000S-1	21%	24 MHz

Note: Data in this table was achieved using typical synthesis and layout settings. Performance was achieved using the Core8051 macro alone.

Core8051 Verification

The comprehensive verification simulation testbench (included with the Netlist and RTL versions of the core) verifies correct operation of the Core8051 macro. The verification testbench applies several tests to the Core8051 macro, including:

- Operation Code Tests
- Peripheral Tests
- Miscellaneous Tests

Using the supplied user testbench as a guide, the user can easily customize the verification of the core by adding or removing tests.

I/O Signal Descriptions

The port signals for the Core8051 macro are defined in Table 5 on page 6 and illustrated in Figure 2. Core8051 has 239 I/O signals that are described in Table 5 on page 6.

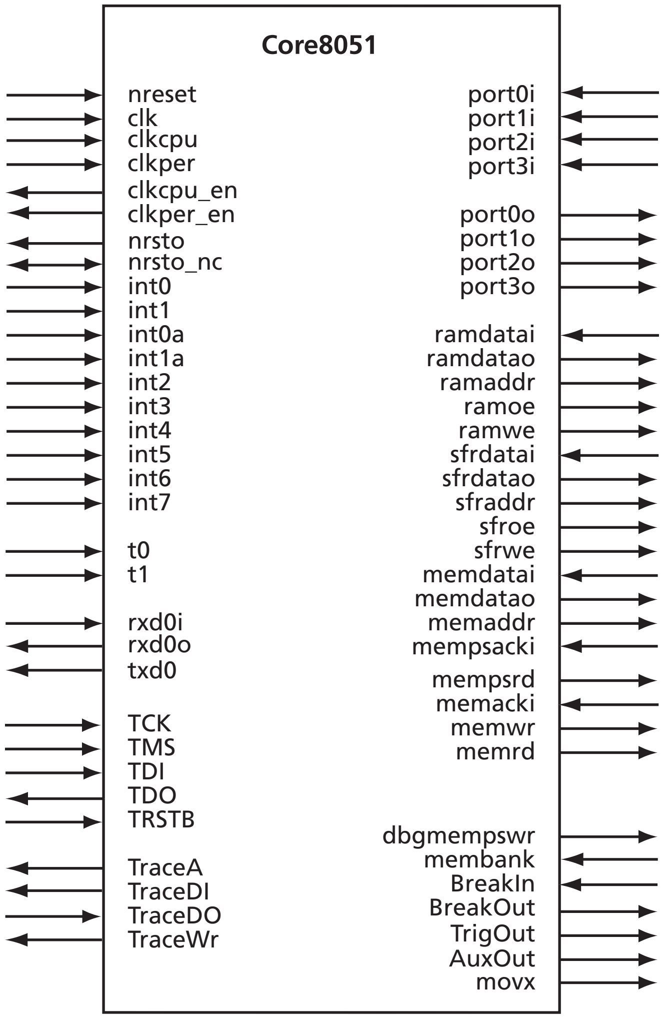

Figure 2 • **Core8051 I/O Signal Diagram**

Table 5 • **Core8051 Pin Description**

Name	Type	Polarity/Bus Size	Description
port0i port0o	Input Output	8 8	**Port 0** 8-bit bidirectional I/O port with separated inputs and outputs
port1i port1o	Input Output	8 8	**Port 1** 8-bit bidirectional I/O port with separated inputs and outputs
port2i port2o	Input Output	8 8	**Port 2** 8-bit bidirectional I/O port with separated inputs and outputs
port3i port3o	Input Output	8 8	**Port 3** 8-bit bidirectional I/O port with separated inputs and outputs
clk	Input	Rise	**Clock** input for internal logic
clkcpu	Input	Rise	**CPU Clock** input for internal controller logic (must either be the same as the clk input or a gated version of the clk input)
clkper	Input	Rise	**Peripheral Clock** input for internal peripheral logic (must either be the same as the clk input or a gated version of the clk input)
clkcpu_en	Output	High	**CPU Clock Enable** This output may be used to optionally create a gated version of the clk input signal for connection to the clkcpu input (see "Power Management Implementation" section on page 36).
clkper_en	Output	High	**Peripheral Clock Enable** This output may be used to optionally create a gated version of the clk input signal for connection to the clkper input (see "Power Management Implementation" section on page 36).
nreset	Input	Low	**Hardware Reset Input** A logic 0 on this pin for two clock cycles while the oscillator is running resets the device.
nrsto	Output	Low	**Peripheral Reset Output** This globally buffered signal can be connected to logic outside Core8051 to provide an active-low asynchronous reset to peripherals.
nrsto_nc	Bidirectional (no-connect)	Low	**Peripheral Reset No-Connect** This signal is connected to nrsto internally and is only used by the SX-A/RTSX-S implementations, in which case it must be brought up to a top-level package pin and left unconnected at the board-level. This signal should not be used (connected) for any other device families.
movx	Output	High	**Movx** instruction executing
 TCK TMS TDI TDO nTRST dbgmempswr	 Input Input Input Output Input Output	 Rise High High High Low High	**On-Chip Debug Interface (Optional)** JTAG test clock. If OCI is not used, connect to logic 1. JTAG test mode select. If OCI is not used, connect to logic 0. JTAG test data in. If OCI is not used, connect to logic 0. JTAG test data out JTAG test reset. If OCI is not used, connect to logic 1. Optional debug program storage write

Table 5 • **Core8051 Pin Description (Continued)**

Name	**Type**	**Polarity/Bus Size**	**Description**
membank	Input	4	Optional code memory bank selection. If not used, connect to logic 0 values.
BreakIn	Input	High	Break bus input. When sampled high, a breakpoint is generated. If not used, connect to logic 0.
BreakOut	Output	High	Break bus output. This will be driven high when Core8051 stops emulation. This can be connected to an open-drain Break bus that connects to multiple processors, so that when any CPU stops, all others on the bus are stopped within a few clock cycles.
TrigOut	Output	High	Trigger output. This signal can be optionally connected to external test equipment to cross-trigger with internal Core8051 activity.
AuxOut	Output	High	Auxiliary output. This signal is an optional general-purpose output that can be controlled via the OCI debugger software.
TraceA	Output	8	Trace address outputs. This bus should be connected to external RAM address pins for trace debug memory.
TraceDI	Output	20	Trace data to external synchronous RAM data input pins for trace debug memory.
TraceDO	Input	20	Trace data from external synchronous RAM data output pins for trace debug memory. If OCI is not used, connect to logic 0 values.
TraceWr	Output	High	Trace write signal to external synchronous RAM write enable for trace debug memory.
			External Interrupt Inputs
int0	Input	Low/Fall	External interrupt 0
int1	Input	Low/Fall	External interrupt 1
int0a	Input	High	External interrupt 0a
int1a	Input	High	External interrupt 1a
int2	Input	High	External interrupt 2
int3	Input	High	External interrupt 3
int4	Input	High	External interrupt 4
int5	Input	High	External interrupt 5
int6	Input	High	External interrupt 6
int7	Input	High	External interrupt 7
			Serial Port Interface
rxdi	Input	–	Serial port receive data
rxdo	Output	–	Serial port transmit data in mode 0
txd	Output	–	Serial port transmit data or data clock in mode 0
			Timer Inputs
t0	Input	Fall	Timer 0 external input
t1	Input	Fall	Timer 1 external input
			External Memory Interface
mempsacki	Input	High	Program memory read acknowledge

Table 5 • **Core8051 Pin Description (Continued)**

Name	Type	Polarity/Bus Size	Description
memacki	Input	High	Data memory acknowledge
memdatai	Input	8	Memory data input
memdatao	Output	8	Memory data output
memaddr	Output	16	Memory address
mempsrd	Output	High	Program store read enable
memwr	Output	High	Data memory write enable
memrd	Output	High	Data memory read enable
			Internal Data Memory Interface
ramdatai	Input	8	Data bus input
ramdatao	Output	8	Data bus output
ramaddr	Output	8	Data file address
ramwe	Output	High	Data file write enable
ramoe	Output	High	Data file output enable
			External Special Function Registers Interface
sfrdatai	Input	8	SFR data bus input
sfrdatao	Output	8	SFR data bus output
sfraddr	Output	7	SFR address
sfrwe	Output	High	SFR write enable
sfroe	Output	High	SFR output enable

Memory Organization

The Core8051 microcontroller utilizes the Harvard architecture, with separate code and data spaces.

Memory organization in Core8051 is similar to that of the industry standard 8051. There are three memory areas, as shown in Figure 3:

- Program Memory (Internal RAM, External RAM, or External ROM)
- External Data Memory (External RAM)
- Internal Data Memory (Internal RAM)

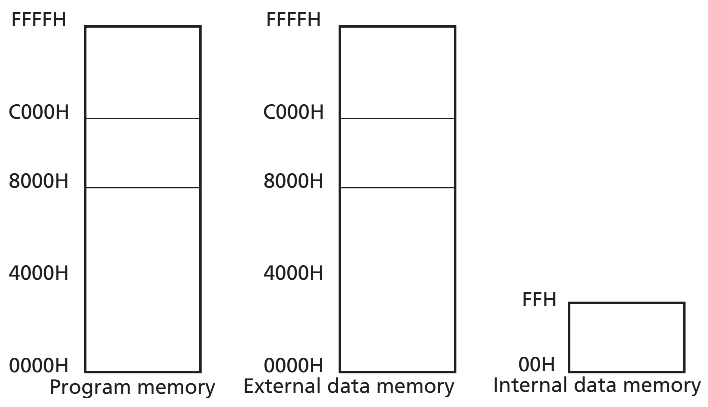

Figure 3 • **Core8051 Memory Map**

Program Memory

Core8051 can address up to 64kB of program memory space, from 0000H to FFFFH. The External Bus Interface services program memory when the mempsrd signal is active. Program memory is read when the CPU performs fetching instructions or MOVC.

After reset, the CPU starts program execution from location 0000H. The lower part of the program memory includes interrupt and reset vectors. The interrupt vectors are spaced at eight-byte intervals, starting from 0003H.

Program memory can be implemented as Internal RAM, External RAM, External ROM, or a combination of all three.

External Data Memory

Core8051 can address up to 64kB of external data memory space, from 0000H to FFFFH. The External Bus Interface services data memory when the memrd signal is active. Writing to external program memory is only supported in debug mode using the OCI logic block and external debugger hardware and software. Core8051 writes into external data memory when the CPU executes MOVX @Ri,A or MOVX @DPTR,A instructions. The external data memory is read when the CPU executes MOVX A,@Ri or MOVX A,@DPTR instructions.

There is improved variable length of the MOVX instructions to access fast or slow external RAM and external peripherals. The three low-ordered bits of the ckcon register control stretch memory cycles. Setting ckcon stretch bits to logic 1 values enables access to very slow external RAM or external peripherals.

Table 6 shows how the External Memory Interface signals change when stretch values are set from zero to seven. The widths of the signals are counted in clk cycles. The reset state of the ckcon register has a stretch value equal to one (001), which enables MOVX instructions to be performed with a single stretch clock cycle inserted.

Table 6 • **Stretch Memory Cycle Width**

ckcon Register				**Read Signal Width**		**Write Signal Width**	
ckcon.2	**ckcon.1**	**ckcon.0**	**Stretch Value**	**memaddr**	**memrd**	**memaddr**	**memwr**
0	0	0	0	1	1	2	1
0	0	1	1	2	2	3	1
0	1	0	2	3	3	4	2
0	1	1	3	4	4	5	3
1	0	0	4	5	5	6	4
1	0	1	5	6	6	7	5
1	1	0	6	7	7	8	6
1	1	1	7	8	8	9	7

There are two types of instructions; one provides an 8-bit address to the external data RAM, the other a 16-bit indirect address to the external data RAM.

In the first instruction type, the contents of R0 or R1 in the current register bank provide an 8-bit address. The eight high ordered bits of address are stuck at zero. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins are controlled by an output instruction preceding the MOVX.

In the second type of MOVX instructions, the data pointer generates a 16-bit address. This form is faster and more efficient when accessing very large data arrays (up to 64kB), since no additional instructions are needed to set up the output ports.

In some situations, it is possible to mix the two MOVX types. A large RAM array, with its high-order address lines, can be addressed via the data pointer or with code to output high-order address bits to any port followed by a MOVX instruction using R0 or R1.

Internal Data Memory

The internal data memory interface services up to 256 bytes of off-core data memory. The internal data memory address is always one byte wide. The memory space is 256 bytes large (00H to FFH) and can be accessed by direct or indirect addressing. The SFRs occupy the upper 128 bytes. This SFR area is available only by direct addressing. Indirect addressing accesses the upper 128 bytes of internal RAM.

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

The internal data memory is not instantiated in Core8051. The user may use internal memory resources if the ProASIC^PLUS or Axcelerator families are used. The SX-A and RTSXS-S families have no internal memory resources, thus the user would need to either create and instantiate a distributed RAM (comprised of FPGA combinatorial and sequential cells) or use an external memory device.

Special Function Registers

Internal Special Function Registers

A map of the internal Special Function Registers is shown in Table 7. Only a few addresses are occupied; the others are not implemented. Read access to unimplemented addresses will return undefined data, while write access will have no effect.

Table 7 • **Internal Special Function Register Memory Map**

Hex	Bin								Hex
	X000	X001	X010	X011	X100	X101	X110	X111	
F8	–	–	–	–	–	–	–	–	FF
F0	b	–	–	–	–	–	–	–	F7
E8	–	–	–	–	–	–	–	–	EF
E0	acc	–	–	–	–	–	–	–	E7
D8	–	–	–	–	–	–	–	–	DF
D0	psw	–	–	–	–	–	–	–	D7
C8	–	–	–	–	–	–	–	–	CF
C0	–	–	–	–	–	–	–	–	C7
B8	ien1	ip1	–	–	–	–	–	–	BF
B0	p3	–	–	–	–	–	–	–	B7
A8	ien0	ip0	–	–	–	–	–	–	AF
A0	p2	–	–	–	–	–	–	–	A7
98	scon	sbuf	–	–	–	–	–	–	9F
90	p1	–	dps	–	–	–	–	–	97
88	tcon	tmod	tl0	tl1	th0	th1	ckcon	–	8F
80	p0	sp	dpl	dph	dpl1	dph1	–	pcon	87

The reset value for of each of the predefined special function registers is listed in Table 8.

Table 8 • **Special Function Register Reset Values**

Register	Location	Reset value	Description
p0	80h	FFh	Port 0
sp	81h	07h	Stack Pointer
dpl	82h	00h	Data Pointer Low 0
dph	83h	00h	Data Pointer High 0
dpl1	84h	00h	Dual Data Pointer Low 1

Table 8 • **Special Function Register Reset Values (Continued)**

dph1	85h	00h	Dual Data Pointer High 1
pcon	87h	00h	Power Control
tcon	88h	00h	Timer/Counter Control
tmod	89h	00h	Timer Mode Control
tl0	8Ah	00h	Timer 0, low byte
tl1	8Bh	00h	Timer 1, high byte
th0	8Ch	00h	Timer 0, low byte
th1	8Dh	00h	Timer 1, high byte
ckcon	8Eh	01h	Clock Control (Stretch=1)
p1	90h	FFh	Port 1
dps	92h	00h	Data Pointer Select Register
scon	98h	00h	Serial Port 0, Control Register
sbuf	99h	00h	Serial Port 0, Data Buffer
p2	A0h	FFh	Port 2
ien0	A8h	00h	Interrupt Enable Register 0
ien1	B8h	00h	Interrupt Enable Register 1
p3	B0h	FFh	Port 3
ip0	A9h	00h	Interrupt Enable Register 0
ip1	B9h	00h	Interrupt Enable Register 1
psw	D0h	00h	Program Status Word

External Special Function Registers

The external SFR interface services up to 101 off-core special function registers. The off-core peripherals can use all addresses from the SFR address space range 80H to FFH except for those that are already implemented inside the core.

When a read instruction occurs with a SFR address that has been implemented both inside and outside the core, the read will return the contents of the internal SFR.

When a write instruction occurs with a SFR that has been implemented both inside and outside the core, the value of the external SFR is overwritten.

Instruction Set

All Core8051 instructions are binary code compatible and perform the same functions as they do with the industry standard 8051. Table 9 on page 12 and Table 10 on page 12 contain notes for mnemonics used in the various Instruction Set tables. In Table 11 on page 12 through Table 15 on page 15, the instructions are ordered in functional groups. In Table 16 on page 16, the instructions are ordered in the hexadecimal order of the operation code. For more detailed information about the Core8051 instruction set, refer to the *Core8051 Instruction Set Details User's Guide*.

Table 9 • **Notes on Data Addressing Modules**

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

Table 10 • **Notes on Program Addressing Modes**

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

Functional Ordered Instructions

Table 11 through Table 15 on page 15 lists the Core8051 instructions, grouped according to function.

Table 11 • **Arithmetic Operations**

Mnemonic	Description	Byte	Cycle
ADD A,Rn	Adds the register to the accumulator	1	1
ADD A,direct	Adds the direct byte to the accumulator	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1	2
ADD A,#data	Adds the immediate data to the accumulator	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1	1
ADDC A,direct	Adds the direct byte to A with a carry flag	2	2
ADDC A,@Ri	Adds the indirect RAM to A with a carry flag	1	2
ADDC A,#data	Adds the immediate data to A with carry a flag	2	2
SUBB A,Rn	Subtracts the register from A with a borrow	1	1
SUBB A,direct	Subtracts the direct byte from A with a borrow	2	2
SUBB A,@Ri	Subtracts the indirect RAM from A with a borrow	1	2
SUBB A,#data	Subtracts the immediate data from A with a borrow	2	2
INC A	Increments the accumulator	1	1
INC Rn	Increments the register	1	2
INC direct	Increments the direct byte	2	3
INC @Ri	Increments the indirect RAM	1	3
DEC A	Decrements the accumulator	1	1
DEC Rn	Decrements the register	1	1
DEC direct	Decrements the direct byte	1	2
DEC @Ri	Decrements the indirect RAM	2	3

Table 11 • **Arithmetic Operations (Continued)**

Mnemonic	**Description**	**Byte**	**Cycle**
INC DPTR	Increments the data pointer	1	3
MUL A,B	Multiplies A and B	1	5
DIV A,B	Divides A by B	1	5
DA A	Decimal adjust accumulator	1	1

Table 12 • **Logic Operations**

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

Table 13 • **Data Transfer Operations**

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

Table 14 • **Boolean Manipulation Operations**

Mnemonic	**Description**	**Byte**	**Cycle**
CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3

Table 14 • **Boolean Manipulation Operations (Continued)**

ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,bit	AND complements of direct bit to the carry	2	2
ORL C,bit	OR direct bit to the carry flag	2	2
ORL C,bit	OR complements of direct bit to the carry	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

Table 15 • **Program Branch Operations**

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

Hexadecimal Ordered Instructions

The Core8051 instructions are listed in order of hexidecimal opcode (operation code) in Table 16.

Table 16 • **Core8051 Instruction Set in Hexadecimal Order**

Opcode	Mnemonic	Opcode	Mnemonic
00H	NOP	10H	JBC bit,rel
01H	AJMP addr11	11H	ACALL addr11
02H	LJMP addr16	12H	LCALL addr16
03H	RR A	13H	RRC A
04H	INC A	14H	DEC A
05H	INC direct	15H	DEC direct
06H	INC @R0	16H	DEC @R0
07H	INC @R1	17H	DEC @R1
08H	INC R0	18H	DEC R0
09H	INC R1	19H	DEC R1
0AH	INC R2	1AH	DEC R2
0BH	INC R3	1BH	DEC R3
0CH	INC R4	1CH	DEC R4
0DH	INC R5	1DH	DEC R5
0EH	INC R6	1EH	DEC R6
0FH	INC R7	1FH	DEC R7
20H	JB bit,rel	30H	JNB bit,rel
21H	AJMP addr11	31H	ACALL addr11
22H	RET	32H	RETI
23H	RL A	33H	RLC A
24H	ADD A,#data	34H	ADDC A,#data
25H	ADD A,direct	35H	ADDC A,direct
26H	ADD A,@R0	36H	ADDC A,@R0
27H	ADD A,@R1	37H	ADDC A,@R1
28H	ADD A,R0	38H	ADDC A,R0
29H	ADD A,R1	39H	ADDC A,R1
2AH	ADD A,R2	3AH	ADDC A,R2
2BH	ADD A,R3	3BH	ADDC A,R3
2CH	ADD A,R4	3CH	ADDC A,R4
2DH	ADD A,R5	3DH	ADDC A,R5
2EH	ADD A,R6	3EH	ADDC A,R6
2FH	ADD A,R7	3FH	ADDC A,R7
40H	JC rel	50H	JNC rel
41H	AJMP addr11	51H	ACALL addr11

Table 16 • **Core8051 Instruction Set in Hexadecimal Order (Continued)**

Opcode	Mnemonic	Opcode	Mnemonic
42H	ORL direct,A	52H	ANL direct,A
43H	ORL direct,#data	53H	ANL direct,#data
44H	ORL A,#data	54H	ANL A,#data
45H	ORL A,direct	55H	ANL A,direct
46H	ORL A,@R0	56H	ANL A,@R0
47H	ORL A,@R1	57H	ANL A,@R1
48H	ORL A,R0	58H	ANL A,R0
49H	ORL A,R1	59H	ANL A,R1
4AH	ORL A,R2	5AH	ANL A,R2
4BH	ORL A,R3	5BH	ANL A,R3
4CH	ORL A,R4	5CH	ANL A,R4
4DH	ORL A,R5	5DH	ANL A,R5
4EH	ORL A,R6	5EH	ANL A,R6
4FH	ORL A,R7	5FH	ANL A,R7
60H	JZ rel	70H	JNZ rel
61H	AJMP addr11	71H	ACALL addr11
62H	XRL direct,A	72H	ORL C,bit
63H	XRL direct,#data	73H	JMP @A+DPTR
64H	XRL A,#data	74H	MOV A,#data
65H	XRL A,direct	75H	MOV direct,#data
66H	XRL A,@R0	76H	MOV @R0,#data
67H	XRL A,@R1	77H	MOV @R1,#data
68H	XRL A,R0	78H	MOV R0,#data
69H	XRL A,R1	79H	MOV R1,#data
6AH	XRL A,R2	7AH	MOV R2,#data
6BH	XRL A,R3	7BH	MOV R3,#data
6CH	XRL A,R4	7CH	MOV R4,#data
6DH	XRL A,R5	7DH	MOV R5,#data
6EH	XRL A,R6	7EH	MOV R6,#data
6FH	XRL A,R7	7FH	MOV R7,#data
80H	SJMP rel	90H	MOV DPTR,#data16
81H	AJMP addr11	91H	ACALL addr11
82H	ANL C,bit	92H	MOV bit,C
83H	MOVC A,@A+PC	93H	MOVC A,@A+DPTR
84H	DIV AB	94H	SUBB A,#data
85H	MOV direct,direct	95H	SUBB A,direct

Table 16 • **Core8051 Instruction Set in Hexadecimal Order (Continued)**

Opcode	Mnemonic	Opcode	Mnemonic
86H	MOV direct,@R0	96H	SUBB A,@R0
87H	MOV direct,@R1	97H	SUBB A,@R1
88H	MOV direct,R0	98H	SUBB A,R0
89H	MOV direct,R1	99H	SUBB A,R1
8AH	MOV direct,R2	9AH	SUBB A,R2
8BH	MOV direct,R3	9BH	SUBB A,R3
8CH	MOV direct,R4	9CH	SUBB A,R4
8DH	MOV direct,R5	9DH	SUBB A,R5
8EH	MOV direct,R6	9EH	SUBB A,R6
8FH	MOV direct,R7	9FH	SUBB A,R7
A0H	ORL C,~bit	B0H	ANL C,~bit
A1H	AJMP addr11	B1H	ACALL addr11
A2H	MOV C,bit	B2H	CPL bit
A3H	INC DPTR	B3H	CPL C
A4H	MUL AB	B4H	CJNE A,#data,rel
A5H[1]	–	B5H	CJNE A,direct,rel
A6H	MOV @R0,direct	B6H	CJNE @R0,#data,rel
A7H	MOV @R1,direct	B7H	CJNE @R1,#data,rel
A8H	MOV R0,direct	B8H	CJNE R0,#data,rel
A9H	MOV R1,direct	B9H	CJNE R1,#data,rel
AAH	MOV R2,direct	BAH	CJNE R2,#data,rel
ABH	MOV R3,direct	BBH	CJNE R3,#data,rel
ACH	MOV R4,direct	BCH	CJNE R4,#data,rel
ADH	MOV R5,direct	BDH	CJNE R5,#data,rel
AEH	MOV R6,direct	BEH	CJNE R6,#data,rel
AFH	MOV R7,direct	BFH	CJNE R7,#data,rel
C0H	PUSH direct	D0H	POP direct
C1H	AJMP addr11	D1H	ACALL addr11
C2H	CLR bit	D2H	SETB bit
C3H	CLR C	D3H	SETB C
C4H	SWAP A	D4H	DA A
C5H	XCH A,direct	D5H	DJNZ direct,rel
C6H	XCH A,@R0	D6H	XCHD A,@R0
C7H	XCH A,@R1	D7H	XCHD A,@R1
C8H	XCH A,R0	D8H	DJNZ R0,rel
C9H	XCH A,R1	D9H	DJNZ R1,rel

Table 16 • **Core8051 Instruction Set in Hexadecimal Order (Continued)**

Opcode	Mnemonic	Opcode	Mnemonic
CAH	XCH A,R2	DAH	DJNZ R2,rel
CBH	XCH A,R3	DBH	DJNZ R3,rel
CCH	XCH A,R4	DCH	DJNZ R4,rel
CDH	XCH A,R5	DDH	DJNZ R5,rel
CEH	XCH A,R6	DEH	DJNZ R6,rel
CFH	XCH A,R7	DFH	DJNZ R7,rel
E0H	MOVX A,@DPTR	F0H	MOVX @DPTR,A
E1H	AJMP addr11	F1H	ACALL addr11
E2H	MOVX A,@R0	F2H	MOVX @R0,A
E3H	MOVX A,@R1	F3H	MOVX @R1,A
E4H	CLR A	F4H	CPL A
E5H	MOV A,direct	F5H	MOV direct,A
E6H	MOV A,@R0	F6H	MOV @R0,A
E7H	MOV A,@R1	F7H	MOV @R1,A
E8H	MOV A,R0	F8H	MOV R0,A
E9H	MOV A,R1	F9H	MOV R1,A
EAH	MOV A,R2	FAH	MOV R2,A
EBH	MOV A,R3	FBH	MOV R3,A
ECH	MOV A,R4	FCH	MOV R4,A
EDH	MOV A,R5	FDH	MOV R5,A
EEH	MOV A,R6	FEH	MOV R6,A
EFH	MOV A,R7	FFH	MOV R7,A

1. The A5H opcode is not used by the original set of ASM51 instructions. In Core8051, this opcode is used to implement a trap instruction for the OCI debugger logic.

Instruction Definitions

All Core8051 core instructions can be condensed to 53 basic operations, alphabetically ordered according to the operation mnemonic section, as shown in Table 17.

Table 17 • **PSW Flag Modification (CY, OV, AC)**

	Flag				Flag		
Instruction	**CY**	**OV**	**AC**	**Instruction**	**CY**	**OV**	**AC**
ADD	X	X	X	SETB C	1	–	–
ADDC	X	X	X	CLR C	0	–	–
SUBB	X	X	X	CPL C	X	–	–
MUL	0	X	–	ANL C,bit	X	–	–

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

Table 17 • **PSW Flag Modification (CY, OV, AC) (Continued)**

	Flag				Flag		
Instruction	**CY**	**OV**	**AC**	**Instruction**	**CY**	**OV**	**AC**
DIV	0	X	–	ANL C,~bit	X	–	–
DA	X	–	–	ORL C,bit	X	–	–
RRC	X	–	–	ORL C,~bit	X	–	–
RLC	X	–	–	MOV C,bit	X	–	–
CJNE	X	–	–				

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

Instruction Timing

Program Memory Bus Cycle

The execution for instruction N is performed during the fetch of instruction N+1. A program memory fetch cycle without wait states is shown in Figure 4. A program memory fetch cycle with wait states is shown in Figure 5 on page 21. A program memory read cycle without wait states is shown in Figure 6 on page 21. A program memory read cycle with wait states is shown in Figure 7 on page 22.

The following conventions are used in Figure 4 to Figure 19 on page 27:

Table 18 • **Conventions used in Figure 4 to Figure 19**

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

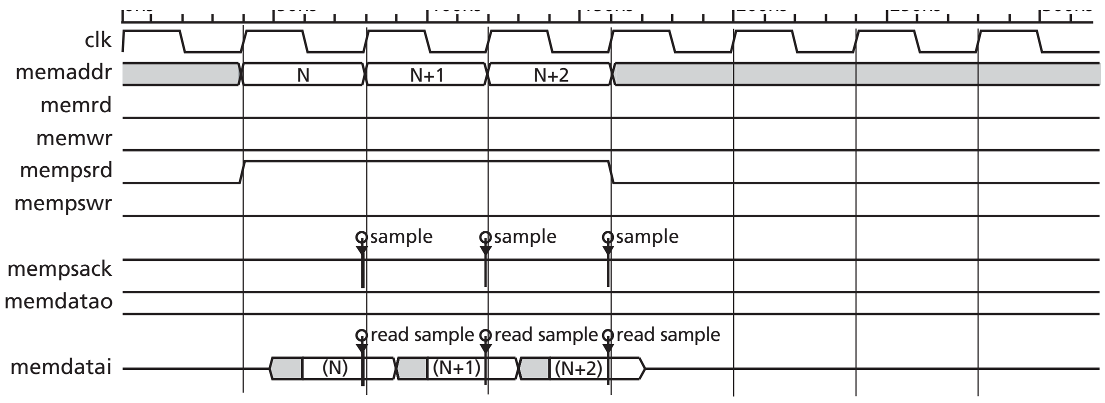

Figure 4 • **Program Memory Fetch Cycle without Wait States**

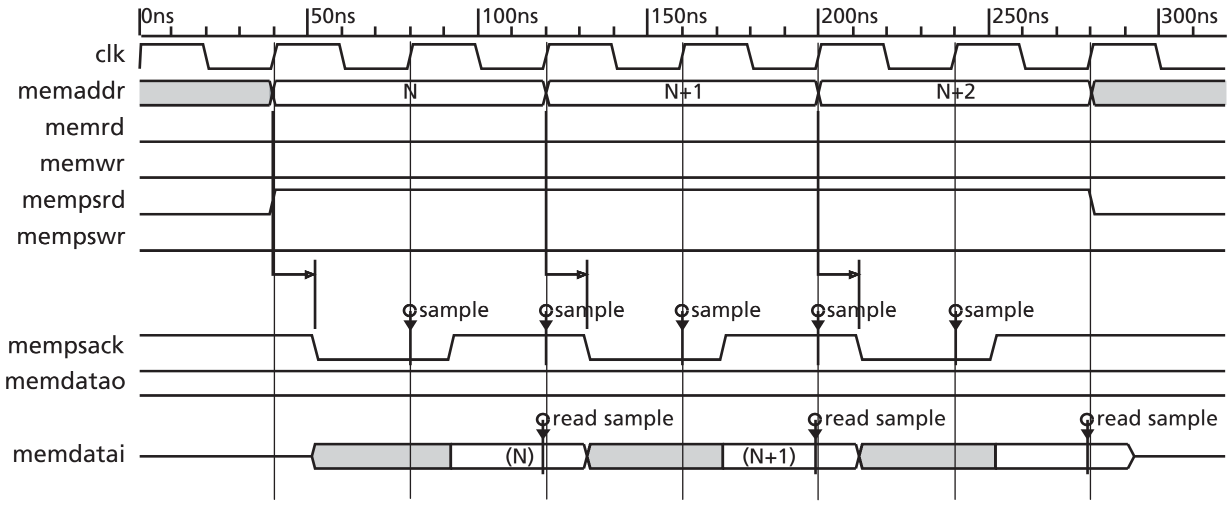

Figure 5 • **Program Memory Fetch with Wait States**

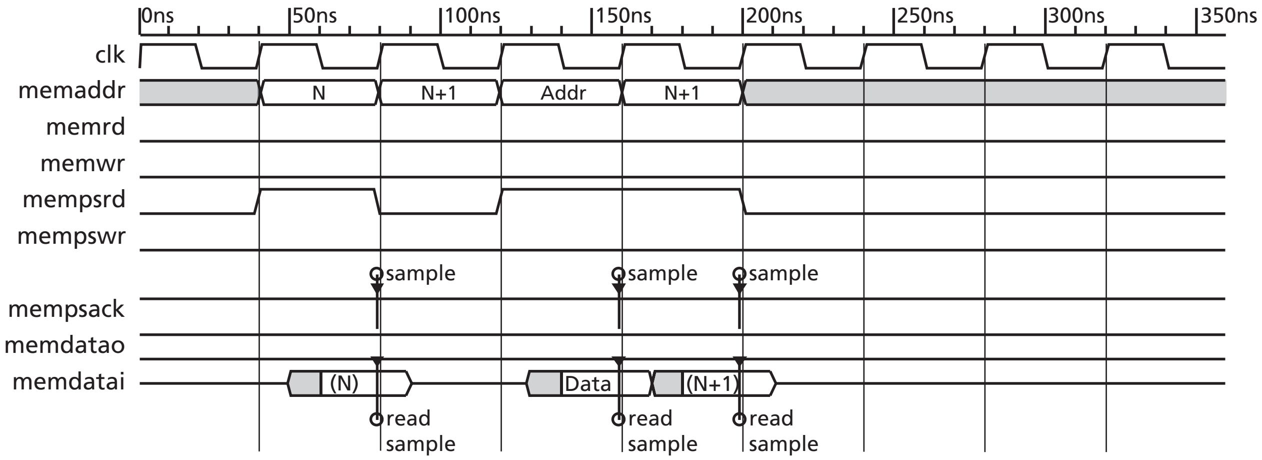

Figure 6 • **Program Memory Read Cycle without Wait States**

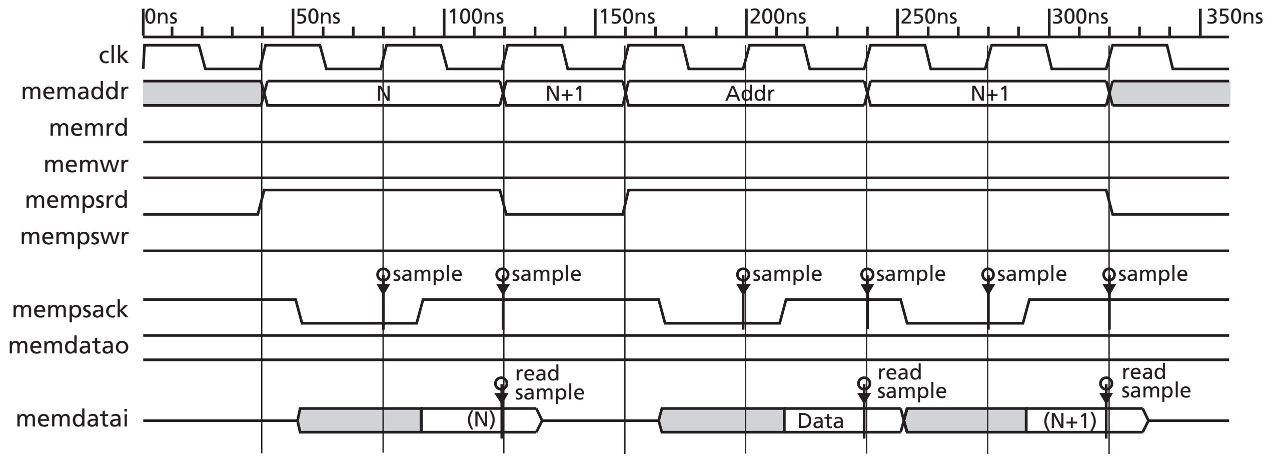

Figure 7 • **Program Memory Read Cycle with Wait States**

External Data Memory Bus Cycle

Example bus cycles for external data memory access are shown in Figure 8 through Figure 15 on page 25. Figure 8 shows an external data memory read cycle without stretch cycles.

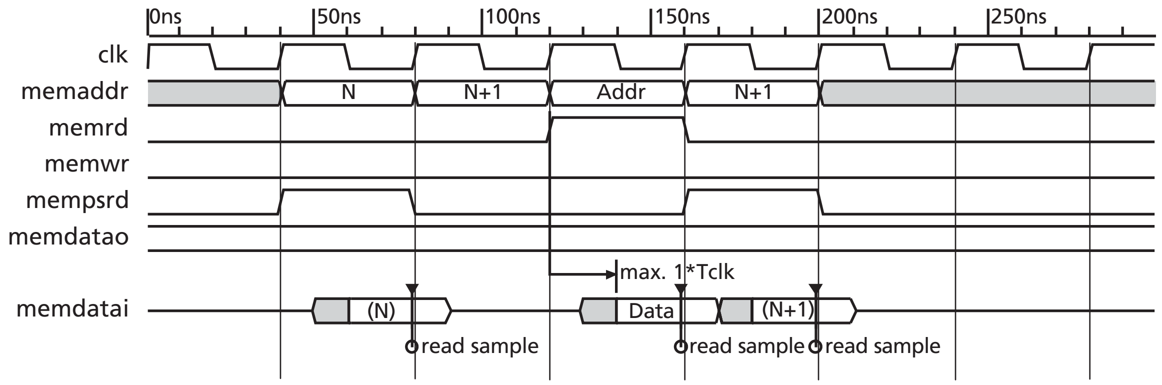

Figure 8 • **External Data Memory Read Cycle without Stretch Cycles**

Figure 9 shows an external data memory read cycle with one stretch cycle.

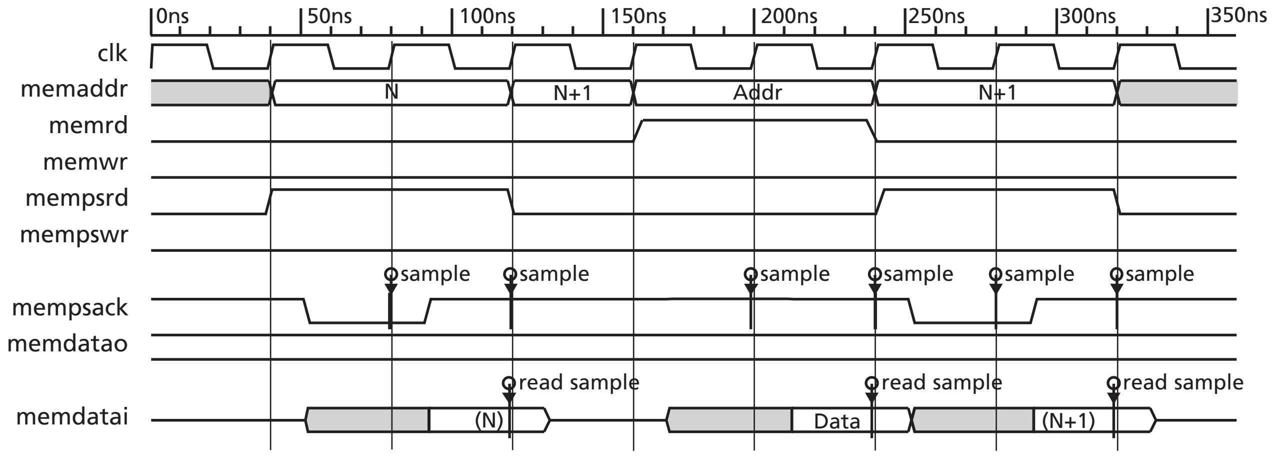

Figure 9 • **External Data Memory Read Cycle with One Stretch Cycle**

Figure 10 shows an external data memory read cycle with two stretch cycles.

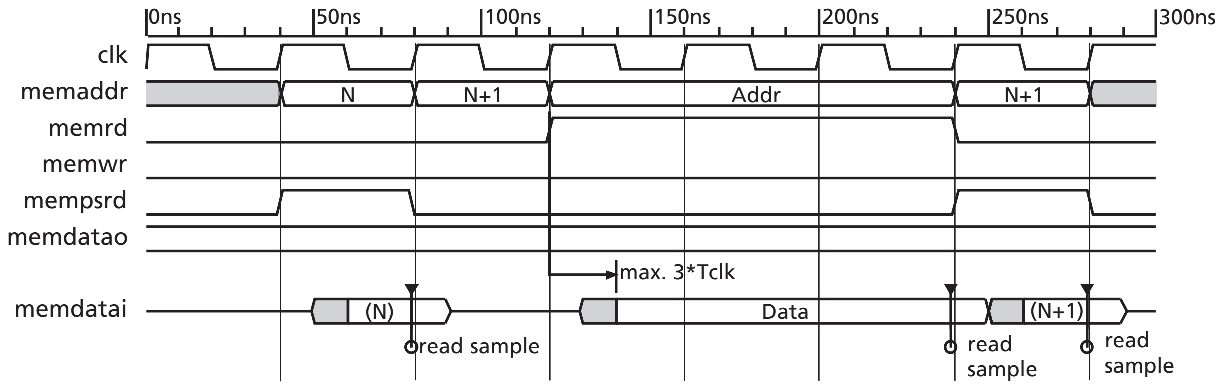

Figure 10 • **External Data Memory Read Cycle with Two Stretch Cycles**

Figure 11 shows an external data memory read cycle with seven stretch cycles.

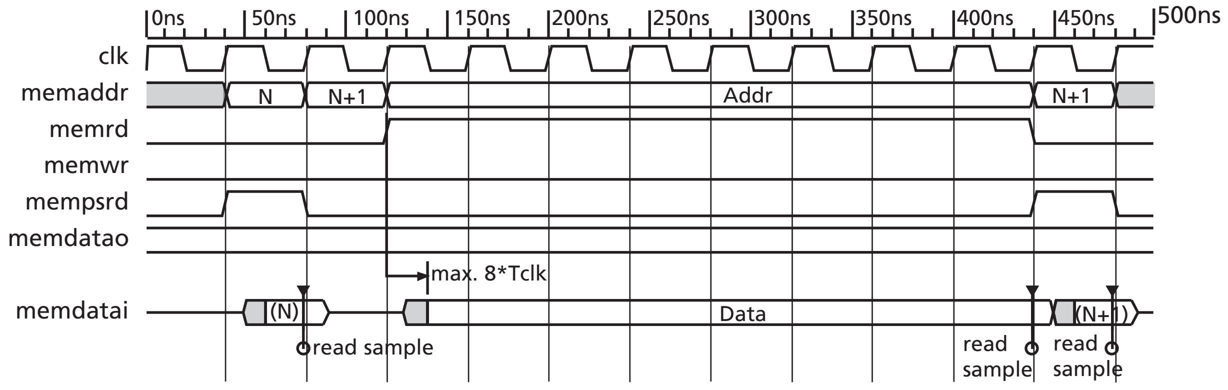

Figure 11 • **External Data Memory Read Cycle with Seven Stretch Cycles**

Figure 12 shows an external data memory write cycle without stretch cycles.

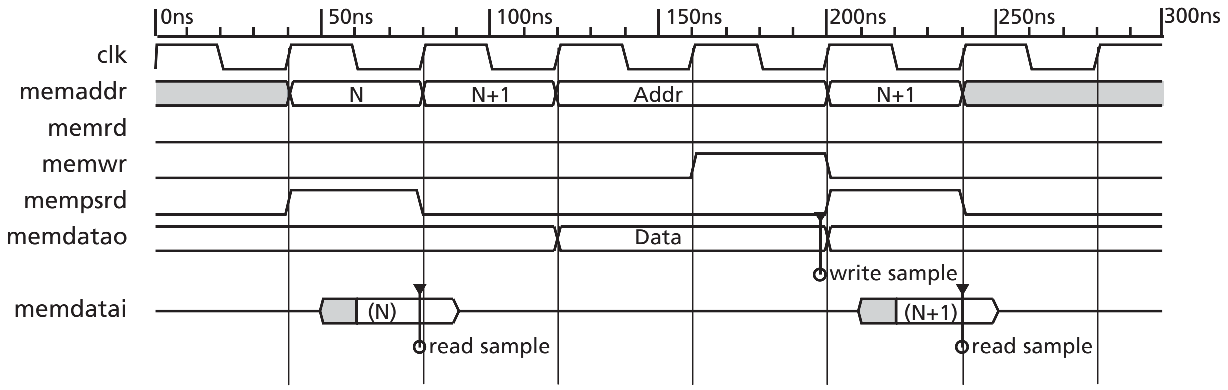

Figure 12 • **External Data Memory Write Cycle without Stretch Cycles**

Figure 13 shows an external data memory write cycle with one stretch cycle.

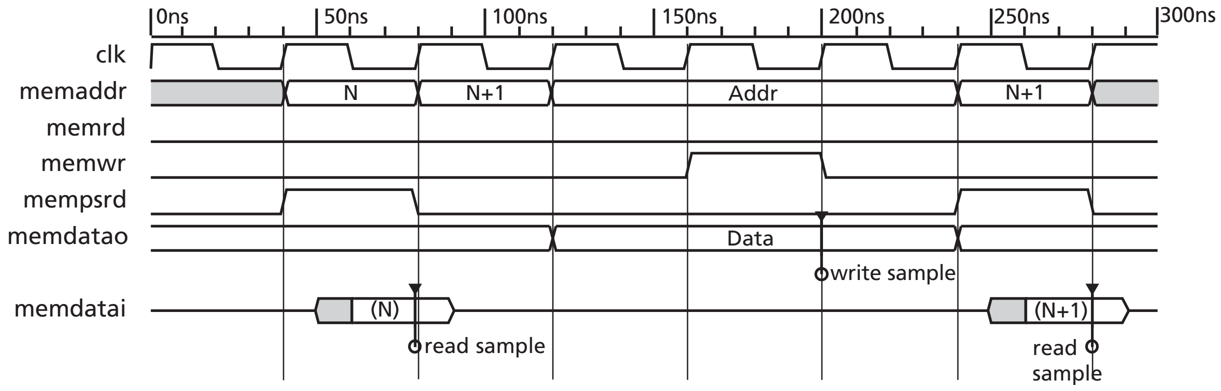

Figure 13 • **External Data Memory Write Cycle with One Stretch Cycle**

Figure 14 shows an external data memory write cycle with two stretch cycles.

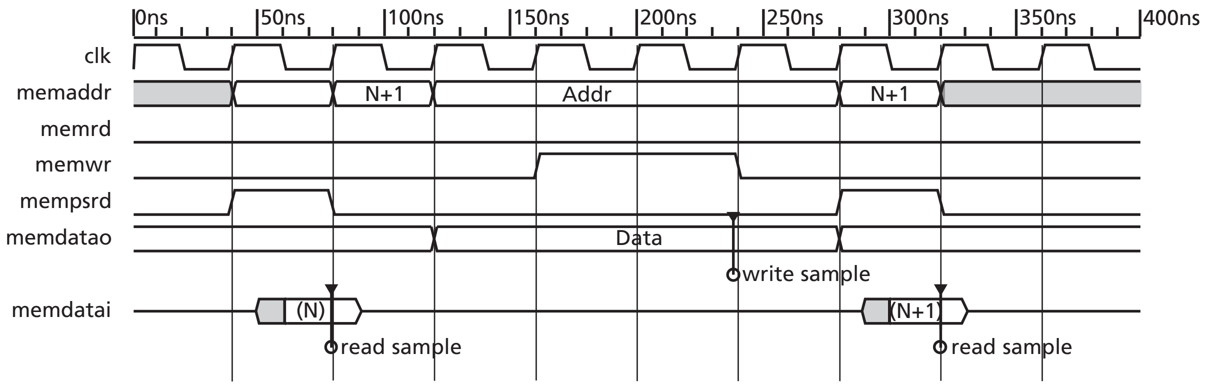

Figure 14 • **External Data Memory Write Cycle with Two Stretch Cycles**

Figure 15 shows an external data memory write cycle with seven stretch cycles.

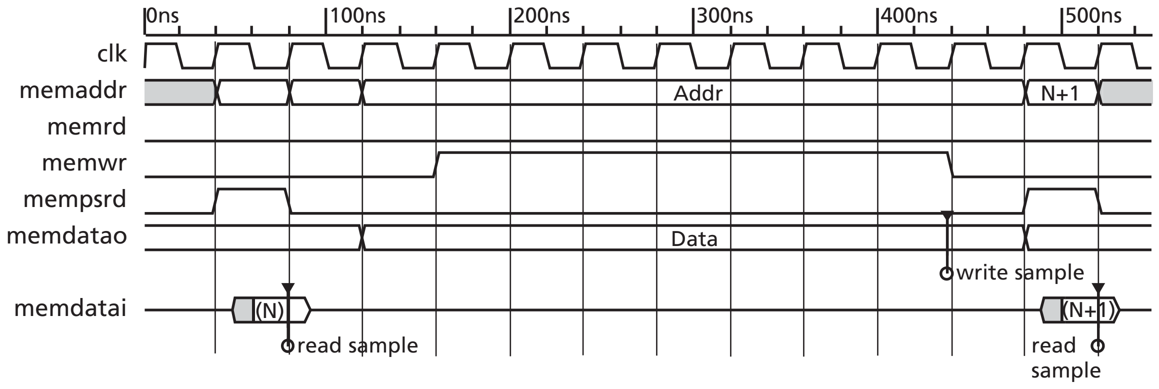

Figure 15 • **External Data Memory Write Cycle with Seven Stretch Cycles**