SIEMENS

TriCore Architecture Overview Handbook

About this Document

This document was created with Adobe® FrameMaker® 5.5.3 at Siemens Microelectronics, Inc., 2480 North First Street, #220, San Jose, California 95131, USA. Revision number and date are shown on each page. This document is not controlled, meaning that no distribution list is maintained and the reader is responsible for ensuring that he/she is not using an obsolete version.

Revision History

Release Version	Release Date	Comments
1.0	06/01/97	Beta release
1.1	09/17/97	Preliminary release
1.2.0	01/29/99	Instruction set updated, other changes to content.
1.2.1	02/22/99	Reformatted for PDF creation, no changes to content

1. A critical component is a component used in a life-support device whose failure can reasonably be expected to cause the failure of that life-support device or system, and/or to affect the safety or effectiveness of that device or system.
2. Life-support devices or systems are intended: (a) to be implemented in the human body, or (b) to support and/or maintain human life. If they fail, it is reasonable to assume that the health of the user may be endangered.

SIEMENS

Contents

Preface

This document provides an overview of the TriCore Instruction Set Architecture (ISA). This document is written for engineering managers, hardware engineers, and software engineers.

Additional information about the TriCore product line can be found in the following publications. Please call your regional sales office to request these publications.

- TriCore Architecture Manual
- TriCore Instruction Set Simulator User's Guide
- Introducing TriCore (Brochure)
- TriCore Development Tools (Brochure)

1 Introducing the TriCore Family Architecture

Future trends for embedded systems include a convergence of microcontroller and DSP architectures, as well as superintegration of memory and logic. Embedded applications are evolving towards a single system-on-a-chip. This chip of the future will be comprised of a unified microcontroller-DSP core (32 bits), data and program memory (RAM, ROM, OTP, etc.), and custom application-specific logic (ASIC), as shown in Figure 1. The single core will provide virtual multiprocessing, which eliminates the need for multiple controllers and DSPs. On-chip memories enhance performance and reduce system power dissipation. The integration of system peripherals and customer-specific logic will increase overall system performance at a reduced cost. The resident (off-the-shelf) real-time operating system will have a compact kernel with appropriate plug-ins for debug, communications, etc. The application layer on top of the RTOS will be automatically generated with the help of app-builder programs that draw on rich library routines like DSP, floating-point, and peripheral management.

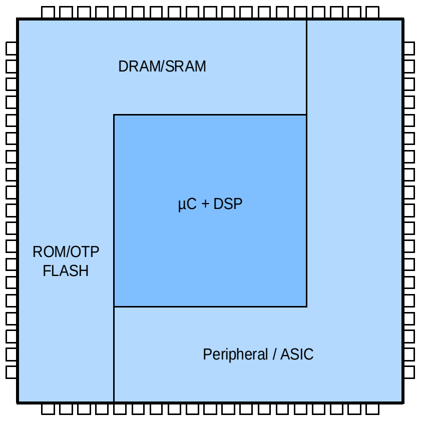

Figure 1: System-on-a-Chip for Embedded Applications

The scenario described above is imperative for the embedded systems of tomorrow. More and more applications demand higher system performance at a reasonable cost. System manufacturers are on the classic electronics “treadmill”—faster speeds and increased functionality/features for the same or even lower price. For example, cellular phones have migrated from analog to digital. Many cell phones incorporate features like paging and voice mail; some even provide internet access and PDA (personal digital assistant) functionality. Form factors have evolved from hand-held to matchbox size. Availability of low power dissipation components allow for increased talk and standby times. And, of course, market prices have dropped dramatically.

With cost-effective processor performance, more work can be off-loaded from hardware to software tasks running on these powerful multi-tasking CPUs. Combined microcontroller-DSP cores can eliminate the need for dual processors and dual development tool sets. On-chip Flash memory eases field programmability concerns.

The elements for tomorrow’s embedded systems exist today. The TriCore Instruction Set Architecture (ISA) from Siemens Semiconductor combines the real-time capability of a microcontroller, the computational power of a DSP, and the high-performance/price features of a RISC load/store architecture onto a compact, reprogrammable core. TriCore is the first single-core 32-bit microcontroller-DSP architecture optimized for real-time embedded systems. You can select peripheral functions (DMA, debug, etc.) from Siemens Semiconductors’ library of peripheral modules. You also can choose the type and size of on-chip memory: SRAM, DRAM, ROM, Flash, and OTP. The core and peripherals are easily connected to yield a high-performance, cost-effective system-on-a-chip, tailored to your application.

Key benefits to using the TriCore for your next real-time embedded system are:

- The single architecture merges both DSP and microcontroller features without sacrificing the performance of either
- Fast task switching (via an internal wide bus to on-chip memory) allows TriCore to be used effectively as a virtual multiprocessor. For example, it can switch from a DSP to a microcontroller task in two cycles.
- Large on-chip memory blocks (RAM, ROM, DRAM, OTP, FLASH) result in higher performance, more reliable operation, and reduced system power consumption
- The architecture allows direct control of on-chip peripherals without additional glue logic. TriCore supports a lean but powerful memory protection and on-chip debug support scheme.
- A freely intermixed 16-bit and 32-bit instruction format reduces code size for your application by approximately 30 to 40%.
- Interrupts are processed as injected calls and are handled by the same mechanism.

The architecture uses a RISC-like register model and load/store architecture to support HLL (High-Level Language) Compilers and their optimization strategies. Fast context switching and low interrupt latencies enable a flexible distribution of processor performance between concurrent tasks and effective control of peripheral events. Integrated debug hardware eases the software development cycle.

The TriCore architecture can save or store half the register context upon an interrupt within two cycles automatically. The architecture thus provides fast interrupt response without having to do a lot of housekeeping before entering the real interrupt service routine.

The architecture allows for a wide range of implementations, ranging from simple scalar to superscalar. Furthermore, the ISA is capable of interacting with different system architectures, including multiprocessing. This flexibility at the implementation and system levels allows for different trade-offs between performance and cost at any point in time.

The native microcontroller-DSP capabilities of the architecture allow you to tune through software, the microcontroller and DSP performance of each TriCore core. For instance, the performance of a 100-MHz TriCore-1 core with a sustained 130 MIPS rating is 80 microcontroller MIPS + 50 DSP MIPS, or 40 microcontroller MIPS + 90 DSP MIPS, depending on how the system designer implements load-sharing in software.

The key features of the TriCore instruction set architecture are:

- 4-GB unified data, program, and I/O space
- 16- and 32-bit instructions for reduced code size
- Low interrupt latency
- Fast context switch using wide pathway to on-chip memory
- Dual single-clock-cycle 16x16 multiply-accumulate unit
- Saturating integer arithmetic
- Extensive bit handling capabilities
- SIMD packed data operations

1.1 TriCore Instruction Categories

To optimize code space, the TriCore architecture offers a flexible set of instruction formats. Although the architecture is 32 bits, there are 16-bit instruction formats available to code the most needed instructions in a smaller amount of memory space. This reduces the instruction code space by an average of one third or more, over conventional RISC architectures.

The TriCore instructions are subdivided into the following categories.

- Branch
- Load/Store
- System
- 16-Bit Subset
- Arithmetic (Integer, DSP, and SIMD Packed Arithmetic)
- Comparison
- Bit Manipulation
- Address Arithmetic and Address Comparison

See “Instruction Set Highlights” on page 13.

1.2 Target Applications

TriCore has been optimized to meet the requirements of embedded applications like computer peripherals, automotive power-train controllers, vehicle dynamics systems, cellular communications, and networking equipment. An increasing number of embedded designs employ both a microcontroller or microprocessor and a DSP or hard-wired ASIC. A TriCore device can replace both these components due to its inherent microcontroller-DSP capabilities and its ability to switch between those tasks at breakneck speed.

1.3 TriCore Roadmap

The TriCore architecture is implemented as a family of cores. A core is a silicon implementation of the architecture. Figure 2 shows the future of the TriCore family architecture. The base group of cores is the TriCore-1 subgroup. TriCore-2 will be a true 64-bit microcontroller with higher degrees of superscalar execution, higher DSP performance, and fast clock speeds. TriCore-3 will perform multithreading, have increased DSP performance over the TriCore-2, and execute at clock speeds in excess of 300 MHz.

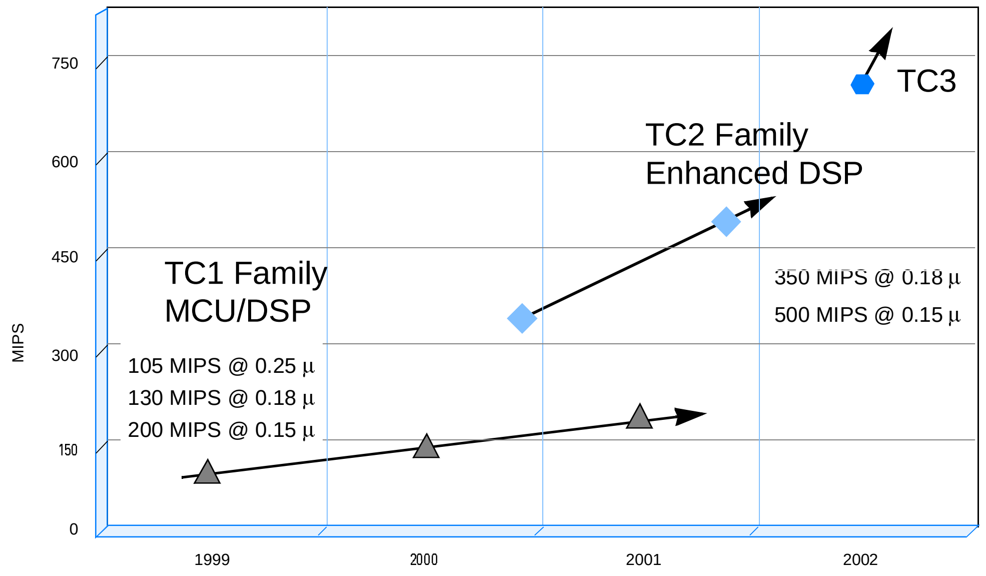

Figure 2: TriCore Roadmap

2 TriCore Programming Model

This section discusses the aspects of the TriCore architecture that are visible to software: the supported data types and formats, the various addressing modes that the architecture provides, and the memory model.

2.1 Architectural Registers

The TriCore architectural registers consist of 32 general-purpose registers (GPRs), two 32-bit registers with program status information (PCXI and PSW), and a program counter (PC). Four GPRs have special functions: D15 is used as an implicit data register, A10 is the stack pointer (SP), A11 is the return address register, and A15 is the implicit base address register. PCXI, PSW, and PC are core special function registers (CSFRs). The PCXI and PSW registers contain status flags, previous execution information, and protection information.

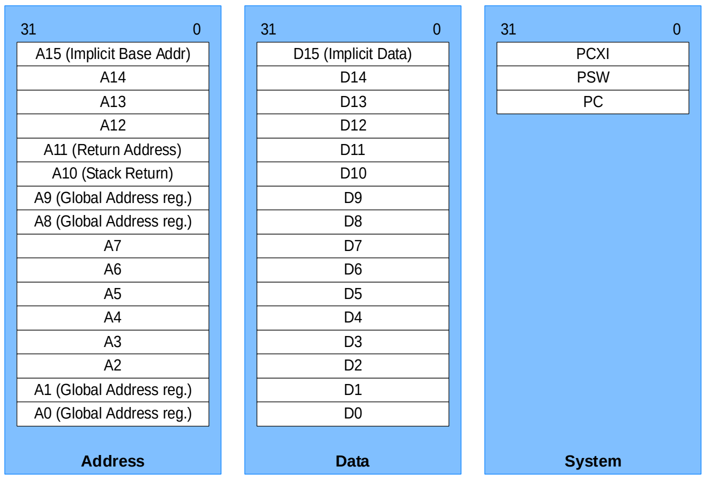

Figure 3: Architectural Registers (GPRs)

2.2 Data Types and Formats

The TriCore instruction set supports operations on booleans, bit strings, characters, signed fractions, addresses, signed and unsigned integers, and single-precision floating-point numbers. Most instructions work on a specific data type, while others are useful for manipulating several data types.

- Boolean
- Address
- Bit String
- Signed/Unsigned Integer
- Character
- Signed Fraction
- IEEE-754 single-precision floating-point

The general-purpose registers are all 32 bits wide, and most instructions operate on word (32-bit) values. Thus when data with fewer bits than a word is loaded from memory, it must be sign or zero-extended before operations can be applied to the full word. The sign or zero extension is done concurrently as part of the load operation.

The data memory and CPU registers store data in little-endian byte order (the least-significant bytes are at lower addresses). Little-endian memory referencing is used consistently for data and instructions. When the TriCore system is connected to an external big-endian device, translation between big- and little-endian format is performed by the bus interface.

Alignment requirements differ for addresses and data. Addresses (32 bits) must be aligned on a word boundary to permit transfers between address registers and memory. For transfers between data registers and memory, data may be aligned on any halfword boundary, regardless of size; bytes may be accessed an any valid byte address, with no alignment restrictions.

2.3 Memory Model

The TriCore architecture can access up to 4 Gbytes of unified program and I/O memory. The address width is 32 bits. The address space is divided into 16 regions or segments (0 through 15). Each segment is 256 Mbytes. The upper four bits of an address select the specific segment. The first 16-Kbytes of each segment can be accessed using either absolute addressing or absolute bit addressing with the bit set and bit clear instructions.

Figure 4 shows the TriCore architecture's address space mapping.

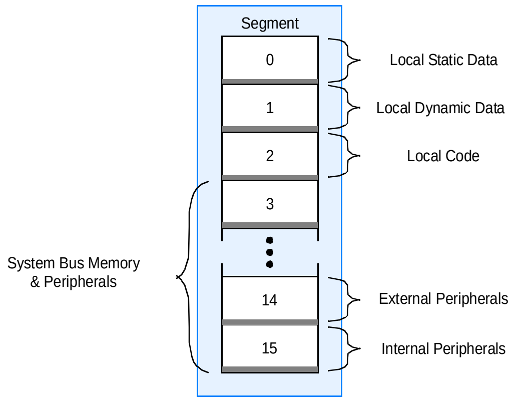

Figure 4: Address Map and Memory Model

2.4 Addressing Modes

Addressing modes allow load and store instructions to efficiently access simple data elements within data structures such as records, randomly and sequentially accessed arrays, stacks, and circular buffers. Simple data elements are 8, 16, 32, or 64 bits wide.

The TriCore architecture supports seven addressing modes, as listed in Table 1. These addressing modes support efficient compilation of C, easy access to peripheral registers, and efficient implementation of typical DSP data structures (circular buffers for filters and bit-reversed indexing for FFTs).

Table 1: Addressing Modes of the TriCore Architecture

Addressing Mode	Address Register Use	Offset Size (bits)
Absolute	None	18
Base + Short Offset	Address Register	10
Base + Long Offset	Address Register	16
Pre-increment	Address Register	10
Post-increment	Address Register	10
Circular	Address Register Pair	10
Bit-reverse	Address Register Pair	—

Addressing modes not supported directly in the hardware can be synthesized through short instruction sequences using indexed addressing, PC-relative addressing, or extended absolute addressing.

3 Tasks and Contexts

In this document, the term TASK refers to an independent thread of control. There are two types of tasks: SOFTWARE-MANAGED TASKS (SMTs) and INTERRUPT SERVICE ROUTINES (ISRs). Software-managed tasks are created through the services of a real-time kernel or OS, and are dispatched under the control of scheduling software.

Each task is allocated its own permission level, depending on the task's function. Individual permissions are enabled/disabled primarily through the IO mode bits in the Processor Status Word (PSW).

Associated with any task is a set of state elements known collectively as the task's CONTEXT. The context is everything the processor needs in order to define the state of the associated task and enable its continued execution. It includes the CPU general registers that the task uses, the task's program counter (PC), and its Program Status Information (PCXI and PSW). The TriCore architecture efficiently manages and maintains the tasks' contexts through hardware.

The context is subdivided into the UPPER CONTEXT and the LOWER CONTEXT. The upper context consists of the upper address registers, A10 - A15, and the upper data registers, D8 - D15. These registers are designated as non-volatile, for purposes of function calling. The upper context also includes PCXI and PSW. The lower context consists of the lower address registers, A2 through A7, and the lower data registers, D0 through D7, plus the PC. Registers A0 and A1 in the lower address registers and A8 and A9 in the upper address registers are defined as SYSTEM GLOBAL REGISTERS. These registers are not included in either context partition, and are not saved and restored across calls or interrupts. The operating system normally uses them to reduce system overhead.

The TriCore architecture uses linked lists of fixed-size CONTEXT SAVE AREAS (CSAs). A CSA is 16 words of on-chip memory storage, aligned on a 16-word boundary. Each CSA can hold exactly one upper or one lower context. CSAs are linked together through a LINK WORD.

The TriCore architecture saves and restores context much more quickly than conventional microprocessors and microcontrollers. Its unique memory subsystem design with a wide data path allows the TriCore architecture to perform rapid data transfers between processor registers and on-chip memory.

Context switching occurs when an event or instruction causes a break in program execution, resulting in the CPU needing to resolve this event before continuing with the program. These events and instructions consist of the following:

1. interrupt or service requests,
2. traps, or
3. function calls.

4 Interrupt System

One key feature of the TriCore architecture is its powerful and flexible interrupt system. The interrupt system is built around programmable Service Request Nodes (SRNs). A SERVICE REQUEST is defined as an interrupt request or a DMA request. A service request may come from an on-chip peripheral, external hardware, or software.

Conventional architectures handle service requests by loading a new Program Status from a vector table in data memory. With the TriCore architecture, service requests jump to vectors in code memory. This procedure reduces response time for service requests. The first instructions of the interrupt service routine (ISR) execute at least three cycles earlier than they would otherwise.

Service requests are prioritized, which enables nested interrupts. A service request can interrupt the servicing of a lower priority interrupt. Interrupt sources with the same priority cannot interrupt each other. The Interrupt Control Unit (ICU) determines which source will win arbitration based on the priority number.

All service requests are assigned priority numbers (SRPNs). Even the CPU has its own priority number. Different service requests must be assigned different priority numbers. The maximum number of interrupt sources is 255. Programmable options range from one priority level with 255 sources up to 255 priority levels with one source each.

Interrupt numbers are assumed to be assigned in linear order of interrupt priority. This is feasible, because interrupt numbers are not hardwired to individual sources. They are assigned by software executed during the power-on boot sequence.

Figure 5 shows several examples where Task 1 is interrupted. For a simple interrupt, the TriCore automatically saves the upper context upon entering the Interrupt Service Routine (ISR). Then the upper context registers can be used within the ISR. When the Return from Execution instruction is issued, the upper context from the time of the interrupt is automatically restored.

In the general interrupt, the upper context is automatically stored. The ISR explicitly saves the lower context using the SVLCX instruction. Both upper and lower context registers can be used within the rest of the ISR. Before returning to Task 1, the restore lower context instruction is issued followed by a return from exception, which automatically restores the upper context.

In the ISR in the persistent context example, explicit upper and lower context values are loaded from memory using the LDUCX and LDLCX instructions. These values were saved from a previous call or interrupt for explicit use in the ISR. At the end of the ISR, new values to be used in a subsequent ISR call are stored explicitly using the STUCX and STLCX instructions.

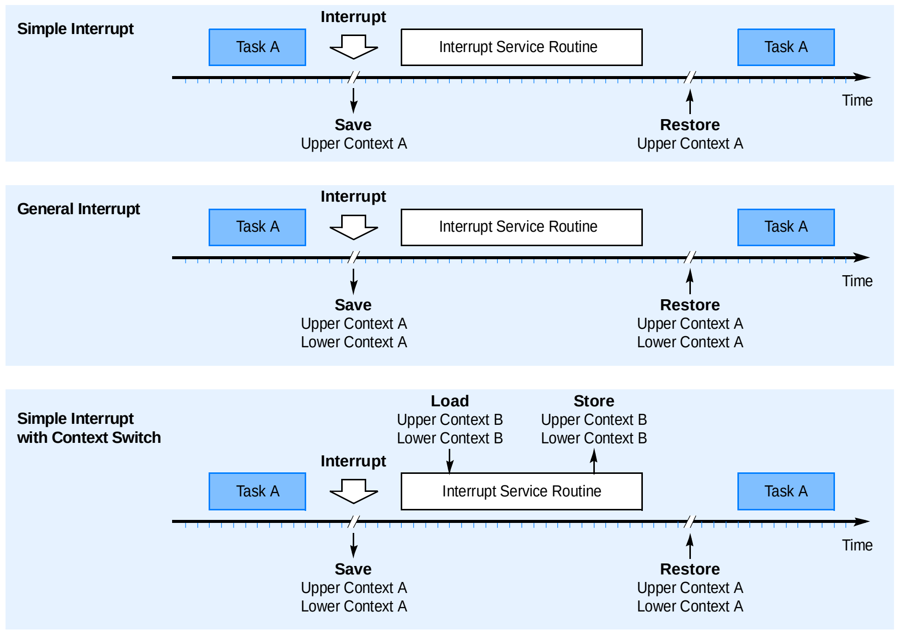

Figure 5: Interrupt Examples

5 Trap System

A trap occurs as a result of an event such as a non-maskable interrupt, an instruction exception, or illegal access. The TriCore architecture contains eight trap classes. These traps are further classified as synchronous or asynchronous, and hardware or software. Each trap is assigned a Trap Identification Number (TIN) that identifies the cause of the trap within its class.

The eight trap classes are:

- Reset
- Assertion
- Internal Protection
- System Bus & Peripheral Errors
- Instruction Errors
- System Call
- Context Management
- Non-Maskable Interrupt

6 Protection System

The protection system allows you to assign access permissions to memory regions for data and code. Protection capabilities are useful for protecting core system functionality from bugs that may have slipped through testing. They are also important aids to testing and debugging.

The TriCore's protection system provides the essential features to isolate errors and facilitate debugging. It protects critical system functions against both software and transient hardware errors.

The TriCore's embedded architecture allows each task to be allocated the specific permission level it needs to perform its function. The three permission levels are:

- USER-0 MODE is used for tasks that do not access peripheral devices.
- USER-1 MODE is used for tasks that access common, unprotected peripherals. Interrupts can be disabled at this level for a short period.
- SUPERVISOR MODE permits read/write access to system registers and protected peripheral devices.

The memory protection model for the TriCore architecture is based on address ranges, where each address range has an associated permission setting. Address ranges and their associated permissions are specified in two to four identical sets of tables residing in Core SFR (CSFR) space. Each set is referred to as a PROTECTION REGISTER SET (PRS).

When the protection system is enabled, the TriCore checks every load/store or instruction fetch address for legality before performing the access. To be legal, the address must fall within one of the ranges specified in the currently selected PRS, and permission for that type of access (read, write, execute) must be present in the matching range.

7 Instruction Set Highlights

This section provides high-level details on the TriCore instruction set. Complete information on all instructions can be found in Siemens Semiconductor's TriCore Architecture Manual.

7.1 Instruction Set Summary

The following table summarizes the TriCore instruction set. Shaded entries indicate 16-bit instructions.

Mnemonic	Definition	Mnemonic	Definition
ABS	Absolute value	DVADJ	Divide adjust
ABSDIF	Absolute value of difference	DVINIT	Divide initialization word
ABSDIFS	Absolute value of difference with saturation	DVSTEP	Divide step
ABSS	Absolute value with saturation	ENABLE	Enable interrupt
ADD	Add	EQ	Equal
ADDC	Add carry	EQANY	Multiple compare
ADDI	Add immediate	EQZ	Equal zero address
ADDIH	Add immediate high word	EXTR	Extract bit field
ADDS	Add with saturation	GE	Greater than or equal
ADDSC	Add scaled address	IMASK	Insert mask
ADDX	Add and generate carry	INS	Insert bit
AND	Logical AND	INSN	Insert bit Not
ANDN	Logical AND Not	INSERT	Insert
AND.comp	Compare, AND and accumulate	ISYNC	Synchronize instructions
AND.logic	Bit and logical accumulate	J	Jump unconditional
BISR	Begin ISR	JA	Jump unconditional absolute
CACHEA.I	Cache Address Invalidate	JEQ	Jump if equal
CACHEA.W	Cache Address Writeback	JGE	Jump if greater than or equal
CACHEA.WI	Cache Address Writeback and Invalidate	JGEZ	Jump if greater than or equal to zero
CADD	Conditional ADD	JGTZ	Jump if greater than zero
CADDN	Conditional ADD Not	JI	Jump indirect
CALL	Call	JL	Jump and link
CALLA	Call absolute	JLA	Jump and link absolute
CALLI	Call indirect	JLEZ	Jump if less than or equal to zero
CLO	Count leading ones	JLI	Jump and link immediate
CLS	Count leading signs	JLT	Jump if less than
CLZ	Count leading zeros	JLTZ	Jump if less than zero
CMOV	Conditional move	JNE	Jump if not equal
CMOVN	Conditional move Not	JNED	Jump if not equal and decrement
CSUB	Conditional subtract	JNEI	Jump if not equal and increment
CSUBN	Conditional subtract Not	JNZ	Jump if not equal to zero
DEBUG	Debug	JZ	Jump if zero
DEXTR	Double extract	LD	Load
DISABLE	Disable interrupt	LDLCX	Load lower context
DSYNC	Synchronize data	LDMDST	Load modify store

Mnemonic	Definition	Mnemonic	Definition
LDUCX	Load upper context	NE	Not equal
LEA	Load Effective address	NEZ.A	Not equal zero address
LOOP	Loop	NOP	No operation
LT	Less than	NOR	Logical NOR
MADD(S)	Multiply-Add (S = with Saturation)	NOT	Bitwise complement
MADDM(S).H	Packed Multiply-Add Q Format - Multiprecision	OR	Logical OR
MADDR(S).H	Packed Multiply-Add Q Format w/ Rounding	OR.comp	Compare, OR and accumulate
MADDR(S).Q	Multiply-Add Q Format with Rounding	OR.logic	Bit OR logical accumulate
MADDSU(S).H	Packed Multiply-Add/Sub Q Format	ORN	Logical OR Not
MADDSUM(S).H	Packed Multiply-Add/Sub Q Format - Multiprecision	RET	Return from call
MADDSUR(S).H	Packed Multiply-Add/Sub Q Format w/ Rounding	RFE	Return from Exception
MAX	Maximum value	RSLCX	Restore lower context
MFCR	Move from Core Register	RSTV	Reset overflow flags
MIN	Minimum value	RSUB	Reverse subtract
MOV	Move	RSUBS	Reverse subtract with saturation
MOVH(.A)	Move halfword to address	SAT	Saturate result
MOVZ.A	Move zero to address	SEL	Select
MSUB(S)	Multiply-Subtract (S = with Saturation)	SELN	Select Not
MSUBAD(S).H	Packed Multiply-Sub/Add Q Format	SH	Shift
MSUBADM(S).H	Packed Multiply-Sub/Add Q Format - Multiprecision	SH.comp	Compare accumulate and shift
MSUBADR(S).H	Packed Multiply-Sub/Add Q Format w/ Rounding	SH.logic	Bit shift logical accumulate
MSUBM(S).H	Packed Multiply-Subtract Q Format - Multiprecision	SHA	Arithmetic shift
MSUBR(S).H	Packed Multiply-Subtract Q Format w/ Rounding	SHAS	Arithmetic shift with saturation
MSUBR(S).Q	Multiply-Subtract Q Format w/ Rounding	ST	Store
MTCR	Move to Core Register	STLCX	Store lower context
MUL(S)	Multiply (S = with Saturation)	STUCX	Store upper context
MUL.H	Packed Multiply Q Format	SUB	Subtract
MUL.Q	Multiply Q Format	SUBC	Subtract with carry
MUL(S).U	Multiply Unsigned (S = with Saturation)	SUBS	Subtract signed with saturation
MULM.H	Packed Multiply Q Format - Multiprecision	SUBX	Subtract extended
MULR.H	Packed Multiply Q Format with Rounding	SVLCX	Save lower context
MULR.Q	Multiply Q Format with Rounding	SWAP	Swap
NAND	Logical NAND	SYSCALL	System call

Mnemonic	Definition	Mnemonic	Definition
TRAPV	Trap on overflow	XOR	Logical exclusive OR
TRAPSV	Trap on sticky overflow	XOR.comp	Compare, XOR and accumulate
XNOR	Logical exclusive NOR		

The TriCore architecture supports both 16- and 32-bit instructions formats. All instructions have a 32-bit format. The 16-bit instructions are a subset of the 32-bit instructions, chosen because of their frequency of use and included to reduce code space. The 16-bit instructions employ one or more of the following methods to allow encoding in 16 bits:

- 2-operand alternative to 3-operand ALU instructions (destination = second source operand)
- implicit source, destination, or base address operand
- small constants
- short branch displacements
- short load/store offsets

The width of the address/data is implicit in the opcode. The 32-bit instruction formats are shown in Figure 6, and the 16-bit instruction formats are shown in Figure 7. Refer to the TriCore Architecture Manual for more information on the instruction formats and their mnemonics.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABS	off18[9..6]				op2		off18[13..10]				off18[5..0]						off18[17..14]				s1/d				op1							
ABSB	off18[9..6]				op2		off18[13..10]				off18[5..0]						off18[17..14]				b	bpos3			op1							
B	disp24[15..0]																disp24[23..16]								op1							
BIT	d				p2					op2		p1					s2				s1				op1							
BO	off10[9..6]				op2						off10[5..0]						s2				s1/d				op1							
BOL	off16[9..6]				off16[15..10]						off16[5..0]						s2				s1/d				op1							
BRC	op2	disp15															const4				s1				op1							
BRN	op2	disp15															n[3..0]				s1				n4	op1						
BRR	op2	disp15															s2				s1				op1							
RC	d				op2							const9									s1				op1							
RCPW	d				p					op2		w					const4				s1				op1							
RCR	d				s3				op2			const9									s1				op1							
RCRR	d				s3				op2								const4				s1				op1							
RCRW	d				s3				op2			w					const4				s1				op1							
RLC	d				const16																s1				op1							
RR	d				op2										n		s2				s1				op1							
RRPW	d				p					op2		w					s2				s1				op1							
RRR	d				s3				op2						n		s2				s1				op1							
RRRR	d				s3				op2								s2				s1				op1							
RRRW	d				s3				op2			w					s2				s1				op1							
SYS						op2																			op1							

Figure 6: 32-Bit Instruction Formats

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SB	disp8								op1							
SBC	const4				disp4				op1							
SBR	s2				disp4				op1							
SBRN	n[3..0]				disp4				n4	op1						
SC	const8								op1							
SLR	s2				d				op1							
SLRO	off4				d				op1							
SR	op2				s1/d				op1							
SRC	const4				s1/d				op1							
SRO	s2				off4				op1							
SRR	s2				s1/d				op1							
SRRS	s2				s1/d				n		op1					
SSR	s2				s1				op1							
SSRO	off4				s1				op1							

Figure 7: 16-Bit Instruction Formats

7.2 Load and Store Instructions

The load and store instructions move data between registers and memory, using the seven addressing modes shown in Table 1 on page 7. The addressing mode determines the effective byte address for the load or store instruction and any update of the base pointer Address register.

7.3 Arithmetic Instructions

Arithmetic instructions operate on data and addresses in registers. Status information about the result of the arithmetic operations is recorded in five status flags. These instructions are categorized further into integer arithmetic, DSP arithmetic, and packed arithmetic instructions.

7.3.1 Integer Arithmetic

Move. The move instructions consist of MOV (sign-extends the value to 32 bits), MOV.U (zero-extends to 32 bits), MOVH (loads a 16-bit constant into the most-significant 16 bits of the register and zero fills the least-significant 16 bits).

Addition and Subtraction. The addition instructions are ADD (no saturation), ADDS (signed saturation), and ADDS.U (unsigned saturation), ADDX (extended precision addition), ADDC (Add with Carry), ADDI (Add Immediate), and ADDIH (Add Immediate High Word).

Because the large immediate of ADDI is sign-extended, it may be used for both addition and subtraction.

The RSUB (Reverse Subtract) instruction subtracts a register from a constant. Using zero as the constant yields negation as a special case.

Multiply and Multiply-Add. Multiplication of two 32-bit integers that produce a 32-bit result can be handled using MUL (Multiply Signed), MULS (Multiply Signed with Saturation), and MULS.U (Multiply Unsigned with Saturation). The MULM (Multiply with Multiword Result) and MULM.U (Multiply with Multiword Result Unsigned) instructions produce the full 64-bit result, which is stored to a register pair; MULM is for signed integers, and MULM.U is for unsigned integers. Special multiply instructions are used for DSP operations.

The multiply-add instruction (MADD) multiplies two signed operands, adds the result to a third operand, and stores the result in a destination. Because, the third operand and the destination do not use the same registers, the intermediate sums of a multi-term multiply-add instruction can be saved without requiring any additional register moves. The MADD, MADDS (Multiply-Add with Saturation), and MADDS.U (Multiply-Add with Saturation Unsigned) instructions operate on and produce 32-bit integers; MADDS and MADDS.U will saturate on signed and unsigned overflow, respectively. To add the 64-bit product to a 64-bit source and produce a 64-bit result, the instructions MADDM (Multiply-Add with Multiword Result), MADDM.U (Multiply-Add with Multiword Result Unsigned), MADDMS (Multiply-Add Multiword with Saturation), and MADDMS.U (Multiply-Add Multiword with Saturation Unsigned) can be used.

The set of Multiply-Subtract (MSUB) instructions, which supports the accumulation of products using subtraction instead of addition, provides the same set of variations as the MADD instructions.

Division. The TriCore ISA supports division of 32-bit by 32-bit integers for both signed and unsigned integers through a divide-step sequence that decreases interrupt latency (the length of time interrupts must be disabled). The divide instructions consist of DVINIT (Divide Initialization), DVSTEP (Divide Step), and DVADJ (Divide Adjust).

Absolute Value, Absolute Difference. The ABS and ABSDIF instructions compute the absolute value of a signed number or absolute value of the difference between two signed numbers, respectively. Each instruction has a version that saturates when the result is too large to be represented as a signed number.

Min, Max, Saturate. The MIN and MAX instructions calculate the minimum or maximum value between two operands, respectively. The SAT instructions saturate the result of a 32-bit calculation before storing it in a byte or halfword in memory or a register.

Conditional Instructions. The conditional instructions—Conditional Add (CADD), Conditional Add Not (CADDN) Conditional Subtract (CSUB), Conditional Subtract Not (CSUBN), Select (SEL), and Select Not (SELN)—provide efficient alternatives to conditional jumps around very short sequences of code. All conditional instructions use a condition operand that controls the execution of the instruction. The condition operand is a data register, with any non-zero value interpreted as TRUE, and a zero value interpreted as FALSE.

Logical. The TriCore architecture provides a complete set of two-operand, bit-wise logic operations: AND, OR, XOR, NAND, NOR, XNOR, and negations of one of the inputs (ANDN and ORN).

Count Leading Zeroes, Ones, and Signs. Three Count Leading instructions provide efficient support for normalization of numerical results, prioritization, and certain graphics operations: CLZ (Count Leading Zeros), CLO (Count Leading Ones), and CLS (Count Leading Signs). These instructions determine the amount of left shifting necessary to remove redundant zeros, ones, or signs.

The Count Leading instructions are useful for parsing certain Huffman codes and bit strings consisting of boolean flags, since the code or bit string can be quickly classified by determining the position of the first one (scanning from left to right).

Shift. The shift instructions support multi-bit left and right shifts. The shift amount is specified by a signed integer (n), which may be the contents of a register or a sign-extended constant in the instruction.

Bit-Field Extract and Insert. The TriCore architecture supports two bit-field extract instructions. The EXTR.U and EXTR instructions extract w (width) consecutive bits from the source, beginning with the bit number specified by the pos (position) operand. The width and position can be specified by two immediate values, by a data register and an immediate value, or by a data register pair.

7.3.2 DSP and Packed Arithmetic

DSP arithmetic instructions operate on 16-bit, signed fractional data in the 1.15 format (also known as Q15) and 32-bit signed fractional data in 1.31 format (also known as Q31). Data values in this format have a single, high-order sign bit, with a value of 0 or -1, followed by an implied binary point and fraction. Their values are in the range [-1, 1).

16-bit DSP data is loaded into the most significant half of a data register, with the 16 least-significant bits set to zero. The left alignment of 16-bit data allows it to be directly added to 32-bit data in 1.31 format. All other fractional formats can be synthesized by explicitly shifting data as required.

Operations created for this format are multiplication, multiply-add, and multiply-subtract. The signed fractional formats 1.15 and 1.31 are supported with the MUL.Q and MULR.Q instructions. These instructions operate on 2 left-justified, signed fractions and return a 32-bit signed fraction.

7.3.2.1 Scaling

The multiplier result can be shifted in two ways:

- Left shifted by 1
  - 1 sign bit is suppressed and the result is left-aligned, thus conserving the input format.
- Not shifted
  - The result retains its 2 sign bits (2.30 format).
  - This format can be used with IIR filters, in which some of the coefficients are between 1 and 2, and to have 1 guard bit for accumulation.

7.3.2.2 Special case = -1 * -1 => +1

When multiplying the two maximum negative values (-1), the result should be the maximum positive number (+1). For example,

0x8000 * 0x8000 = 0x4000 0000

is correctly interpreted in Q format as:

-1(1.15 format) * -1(1.15 format) = +1 (2.30 format)

However, when the result is shifted left by 1, the result is 0x8000 0000, which is incorrectly interpreted as:

-1(1.15 format) * -1(1.15 format) = -1 (1.31 format)

To avoid this problem, the result of a Q format operation (-1 * -1) that has been left-shifted by 1 (left-justified), is saturated to the maximum positive value. Thus,

0x8000 * 0x8000 = 0x7FFF FFFF

is correctly interpreted in Q format as:

-1(1.15 format) * -1(1.15 format) = (nearest representation of)+1 (1.31 format)

This operation is completely transparent to the user and does not set the overflow flags.

7.3.2.3 Guard bits

When accumulating sums (for example, in filter calculations) guard bits are often required to prevent overflow. The instruction set directly supports the use of 1 guard bit when using a 32-bit accumulator; when more guard bits are required, a register pair (64 bits) can be used.

7.3.2.4 Rounding

Rounding is used to retain the 16-bit most-significant bits of a 32-bit result. Rounding is combined with the MUL, MADD, MSUB instructions, and is implemented by adding 1 to bit 15 of a 32-bit register.

7.3.2.5 Overflow and Saturation

Saturation on signed and unsigned overflow is implemented as part of the MUL, MADD, MSUB instructions.

7.3.2.6 Sticky Advance Overflow and Block Scaling in FFT

The Sticky Advance Overflow (SAV) bit, which is set whenever an overflow “almost” occurred, can be used in block scaling of intermediate results during an FFT calculation. Before each pass of applying a butterfly operation, the SAV bit is cleared, and after the pass the SAV bit is tested. If it is set, then all of the data is scaled (using an arithmetic right shift) before starting the next pass. This procedure gives the greatest dynamic range for intermediate results without the risk of overflow.

7.3.3 Packed Arithmetic

The packed arithmetic instructions partition a 32-bit word into several identical objects, which can then be fetched, stored, and operated on in parallel. These instructions, in particular, allow the full exploitation of the 32-bit word of the TriCore architecture in signal and data processing applications.

The TriCore architecture supports two packed formats. The first format (Figure 8) divides the 32-bit word into two, 16-bit (halfword) values. Instructions which operate on data in this way are denoted in the instruction mnemonic by the ".H" and ".HU" data type modifiers.

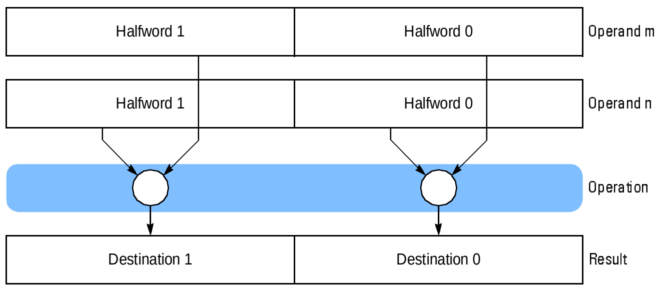

Figure 8: Packed Halfword Data Format

The second packed format (Figure 9) divides the 32-bit word into four 8-bit values. Instructions which operate on the data in this way are denoted by the ".B" and ".BU" data type modifiers.

The loading and storing of packed values into data registers is supported by the normal Load Word and Store Word instructions (LD.W and ST.W). The packed objects can then be manipulated in parallel by a set of special packed arithmetic instructions that perform such arithmetic operations as addition, subtraction, multiplication, etc.

Addition is performed on individual packed bytes or halfwords using the ADD.B and ADD.H instructions and their saturating variations ADDS.B and ADDS.H. ADD.B ignores overflow/underflow within individual bytes, while ADDS.B will saturate individual bytes to the most positive, 8-bit signed integer (127) on individual overflow, or to the most negative, 8-bit signed integer (-128) on individual underflow. Similarly, the ADD.H instruction ignores overflow/underflow within individual halfwords, while the ADDS.H will saturate individual halfwords to the most positive 16-bit signed integer (2^{15}-1) on individual overflow, or to the most negative 16-bit signed integer (-2^{15}) on individual underflow. Saturation for unsigned integers is also supported by the ADDS.BU and ADDS.HU instructions.

Besides addition, arithmetic on packed data includes subtraction, multiplication, absolute value, and absolute difference.

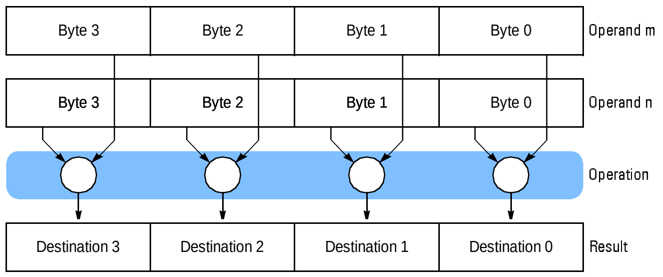

Figure 9: Packed Byte Data Format

7.4 Comparison Instructions

The compare (and conditional jump) instructions use a compare operation on the contents of two registers. The boolean result (1 = true and 0 = false) is stored in the least-significant bit of a data register, and the remaining bits in the register are cleared to zero. Figure 10 illustrates the operation of the LT (Less Than) compare instruction.

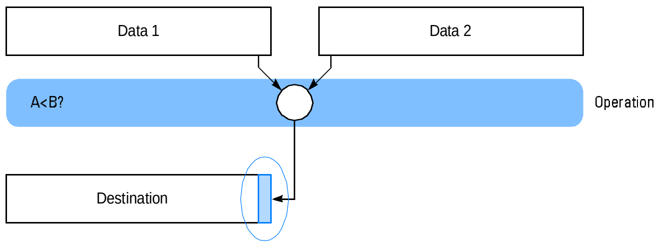

Figure 10: LT Comparison

7.5 Bit Operations

Some TriCore instructions operate on single bits. There are eight instructions for combinatorial logic functions with two inputs, and twelve instructions with three inputs.

The one-bit result of a two-input function is stored in the least-significant bit of the destination data register, and the most-significant 31 bits are set to zero (see Figure 11). The source bits can be any bit of any data register. The available Boolean operations are: AND, NAND, OR, NOR, XOR, XNOR, ANDN, and ORN.

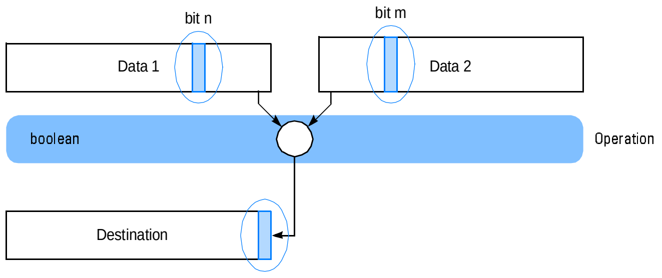

Figure 11: Two-Input Boolean Operations

The three-input Boolean operations are used to evaluate complex Boolean operations where the output of a two-input instruction together with the least-significant bit of a third data register, forms the input to a further operation. The result is written to bit 0 of the third data register, with the remaining bits unchanged. Refer to Figure 12. The available Boolean operations are: AND.AND.T, AND.ANDN.T, AND.NOR.T, AND.OR.T, OR.AND.T, OR.ANDN.T, OR.NOR.T, and OR.OR.T.

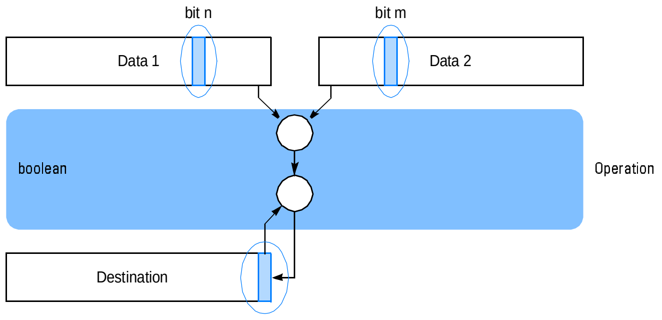

Figure 12: 3-Input Boolean Operation

7.6 Address Arithmetic and Address Comparison

The TriCore architecture provides selected arithmetic operations on the address registers. These operations supplement the address calculations inherent in the addressing modes used by the load and store instructions.

As with the comparison instructions that use the data registers, the comparison instructions using the address registers put the result of the comparison in the least-significant bit of the destination data register and clear the remaining register bits to zeros. An example using the Less Than (LT.A) instruction is shown in Figure 13.

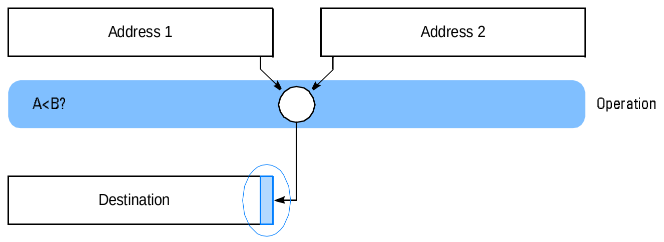

Figure 13: LT.A Comparison Operation

7.7 Branch Instructions

Branch instructions change the flow of program control by modifying the value in the PC register. There are two types of branch instructions: conditional and unconditional. Whether or not a conditional branch is taken depends on the result of a Boolean compare operation, rather than on the state of condition codes.

7.8 System Instructions

The system instructions allow user-mode and supervisor-mode programs to access and control various system services, including interrupts, the instruction and data caches, and the TriCore's debugging facilities. There are also instructions that read and write the PSW and PCXI registers, for both user and supervisor-only mode programs. The Load/Store Upper/Lower Context instructions explicitly save and restore a task's upper and lower contexts.

7.9 16-bit Instructions

The 16-bit instructions are a subset of the 32-bit instruction set, chosen because of their frequency of use. They significantly reduce static code size and thus reduce the cost of code memory and provide a higher effective instruction bandwidth. Because the 16-bit and 32-bit instructions all differ in the primary opcode, the two instruction sizes can be freely intermixed.

The 16-bit instructions are formed by imposing one or more of the following format constraints: smaller constants, smaller displacements, smaller offsets, implicit source, destination, or base address registers, and combined source and destination registers (the two-operand format). In addition, the 16-bit load and store instructions support only a limited set of addressing modes.

8 TriCore-1 Core and Modules

The minimum TriCore implementation consists of a CPU core. As per your design requirements, you can easily add peripheral and memory modules to the core design from Siemens' library. These modules connect via the FPI Bus (Flexible Peripheral Interconnect Bus). The core contains an interface to the FPI bus, for easy interconnection to all kinds of internal and external peripherals, memories, and different active bus agents like CPUs, DMA/PCP controllers, and coprocessors. Figure 14 shows the CPU core with optional data and instruction caches. The following subsections discuss the core and the optional modules that can comprise a TriCore-1 chip.

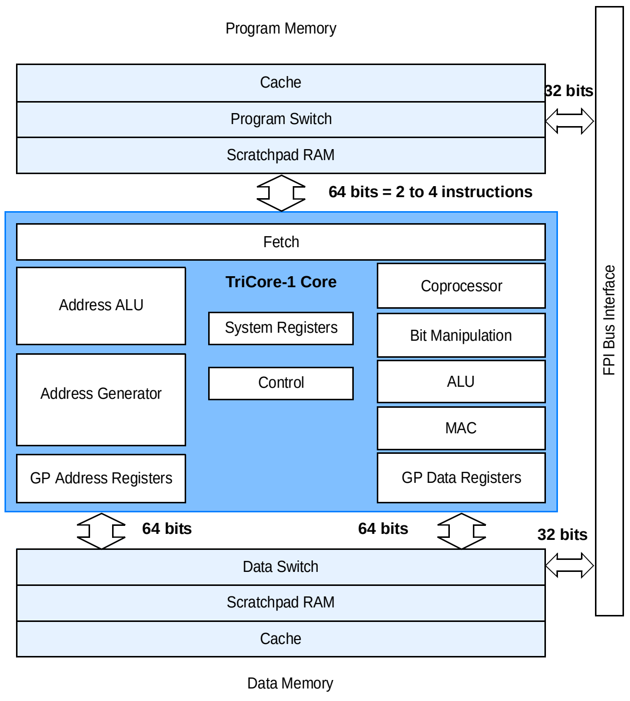

Figure 14: TriCore-1 Core and Optional Memory

8.1 TriCore-1 Core

The TriCore-1 core implements a Harvard architecture with separate address and data buses for program and data memories. Instruction fetches can be handled in parallel with data accesses. The TriCore-1 superscalar core consists of two major pipelines with four stages each, and one minor pipeline for loop control. The three pipelines operate in parallel, allowing up to three instructions to execute in one cycle.

The core is a RISC Load/Store machine. All arithmetic instructions use registers. The ISA contains a set of Load/Store instructions, which fetch the data from memory and store it back to memory. There are two General-Purpose Register Files; one is comprised of 16 address registers and the other is comprised of 16 data registers. The TriCore ISA provides a set of Load/Store instructions that fetch the data from the memory and store it back to the memories. The data side of the core has a 128-bit wide bus, which can save two data and two address registers in one cycle to the cache system. This configuration allows fast interrupt response.

The TriCore-1 core’s Integer Execute Unit consists of a dual Multiply Accumulate Module (MAC), an ALU, and a small tightly coupled Coprocessor interface, which has access to the Register File. The TriCore-1 core can process two Multiply-Accumulates per clock cycle.

The Flexible Peripheral Interconnect Bus (FPI Bus) easily connects the core to memory, internal and external peripherals, CPUs, coprocessors, etc. The data and instruction caches are both connected to the FPI Bus through individual interfaces. The data accesses from the FPI Bus are not cached in order to avoid coherency problems. The DMA accesses to the data memory are cached. The Scratchpad RAMs (SPRs) ensure the timing of critical routines without having to rely on the caches.

8.2 FPI Bus Overview

The FPI Bus (Flexible Peripheral Interconnect Bus) is an on-chip bus designed to be used in modular, highly integrated system chips. The FPI Bus is designed for memory and I/O mapped data transfers between its bus agents, where bus agents are on-chip function blocks (modules) that are equipped with an FPI Bus interface. It is a demultiplexed bus with up to 32 address bits and 64 data bits. Its peak throughput is 800 Mbytes/s at 100 MHz. There is no limit to the number of peripheral modules that can be connected to the FPI Bus.

Additional features of the FPI Bus are:

- Multimaster capability (up to 16 masters)
- Demultiplexed operation
- Clock synchronous
- 8-/16-/32- and 64-bit data transfers
- Broad range of transfer types from single to multiple data transfers
- Flexible bus protocol, which can be tailored to your application needs

There are three types of agents possible on the FPI Bus (see Figure 15):

- Master agents which can initiate and control transactions
- Slave agents, which only support simple read and write of registers and are not actively operating on the bus protocol.
- Master-Slave Agents, which support advanced features like split read transfer support and error handling. Depending on the type of transaction these agents may act as master or slave or both.

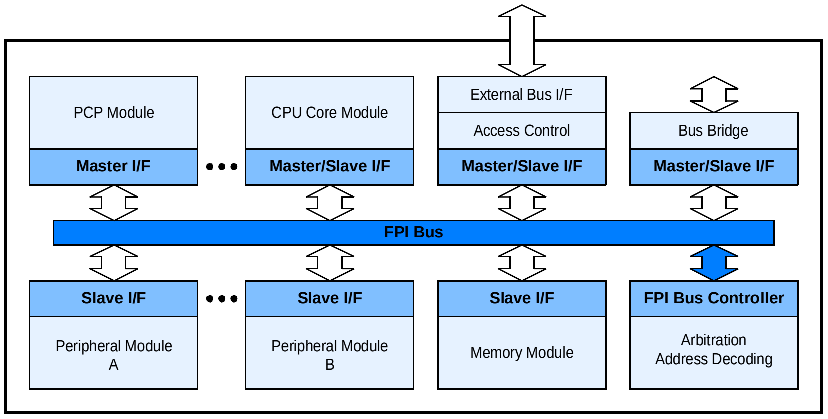

Figure 15: Examples of Modules within an FPI Bus-Based System

8.3 Peripheral Control Processor Module

The Peripheral Control Processor module (PCP) implemented in the TriCore architecture is a programmable data movement and manipulation device. It has up to 255 logical channels, which it services on a demand basis. It performs simple data transfers, monitors the transferred data values, and even performs operations on them. For example, the PCP can answer peripheral service requests by adding a new value to a compare register to set up a new time-out event without any CPU intervention. The PCP can service up to 64 peripheral events in parallel to the CPU, where in conventional systems the CPU has to undergo the burden of an interrupt service routine. Figure 16 shows a block diagram of the PCP.

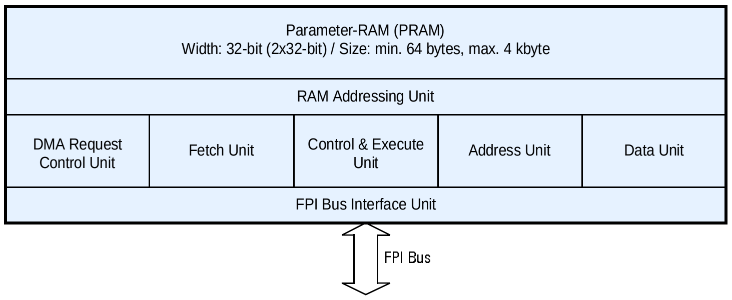

Figure 16: PCP Block Diagram

With the PCP, you can perform the following operations:

- Move data between any two memory or I/O locations
- Move data between the PCP parameter RAM and any memory or I/O location
- Read a data, modify it, and store the result
- Move data until a predefined data value has been detected
- Read data, compare it to predefined limits and conditionally perform appropriate actions
- Read data and accumulate it to previously read data
- Move data and accumulate it to previously read data
- Read two data values, perform an arithmetic or logical operation, and store the result

This functionality can be used to handle many service operations required by peripherals, which normally would be performed through an interrupt service routine via the CPU, with all the overhead involved. Here are some application examples for use of the PCP:

- Reload a peripheral register with a constant (e.g., reload a timer)
- Modify a bit or bit field in a peripheral register (e.g., start or stop a timer or the A/D converter)
- Accumulate values retrieved from peripherals (e.g., accumulate pulse period measurements)
- Move data only if its value matches predefined limits (e.g., monitor certain voltage limits on analog inputs)
- Add values to peripheral registers (e.g., calculate next compare event for PWM generation, etc.)

In addition, you can use the PCP to perform check operations on the CPU or peripherals. Calculation results from the CPU can be checked against predefined ranges with the use of the DMA/PCP. The integrity of peripheral control registers can also be monitored for example, by comparing their contents to a predefined table stored in memory) with the use of the PCP.

The Service Request Unit performs the arbitration of the different source's requests and grants service to the request that has the highest priority at a given time. The Bus Interface Unit provides the proper connection to the FPI Bus. The PCP channel control code may be stored locally to the PCP or to any memory accessible via the FPI bus.

8.4 Debug/Emulation Module

The Debug/Emulation Module provides on-chip debug support for your TriCore design. It easily connects to your design via the FPI bus (see Figure 17).

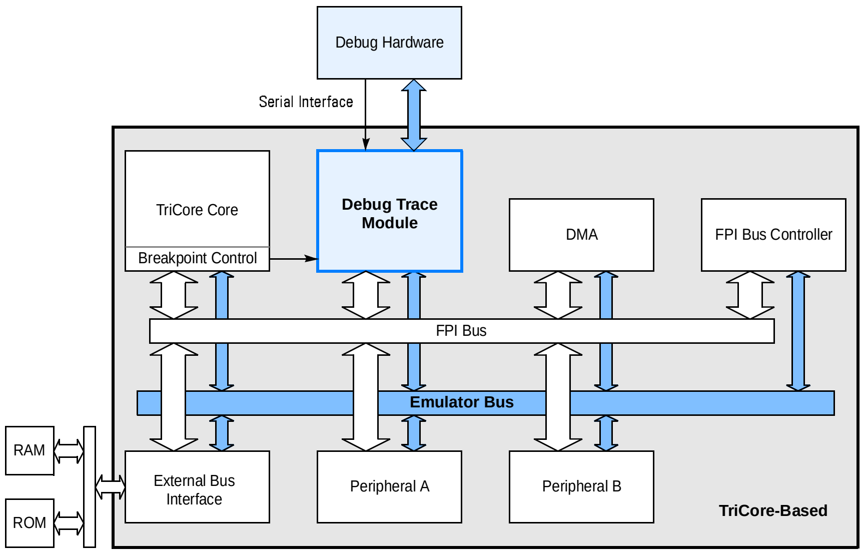

Figure 17: On-Chip Debug Support

The basic components of the Debug module are:

- Support mechanism in the core for breakpoints
- Debug port that provides access to breakpoint mechanism and system resources
- Real-time trace port

The Debug/Emulation module provides a mechanism for communicating with the design during simulation. The breakpoint mechanism allows you to view register and memory contents at various operating stages. The operation of the TriCore core, DMA, and Debug interface can be traced in real time via the TriCore real-time trace output.

9 TriCore Software Development Tools

The TriCore architecture is well supported by a robust set of hardware and software development tools (see Figure 18). These tools include the TriCore Instruction Set Simulator (TSIM), compiler-assembler debugger tool chain, real-time operating systems, and emulators. The instruction set architecture was developed in close consultation with the third party providers of these tools. The TriCore Instruction Set Simulator (TSIM) is bundled together with complete (debugger-compiler-assembler-linker-loader) tool chains from several vendors. Refer to the TriCore Development Tools brochure for the vendor names.

Their evaluation kits (both PC and UNIX versions) are available free of cost to qualified customers. System designers can not only perform price-performance trade-offs on this instruction accurate simulator, but can also begin their software development and debugging.

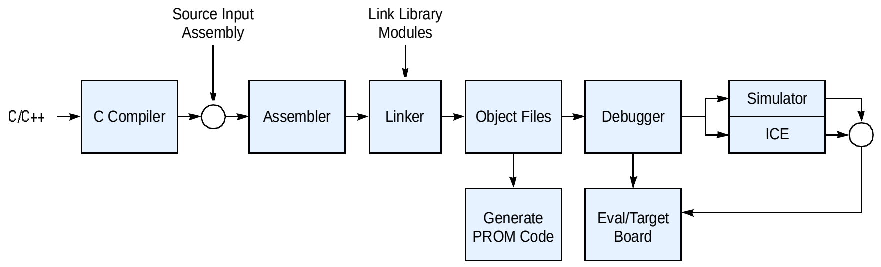

Figure 18: TriCore Development Tools

TSIM is a configurable, instruction-accurate model of the TriCore-1 core architecture that is integrated into all supported source-level debuggers. TSIM provides a simulation environment that models the TriCore core, memory configuration, and interrupt mechanism. TSIM is useful for performance and trade-off analysis and for developing and debugging your customized design.

You can reprogram the TriCore-1 core to evaluate your implementation approach by changing the memory parameters in the TSIM memory configuration file (**`MConfig`**). You can also specify interrupt events in the TSIM interrupt configuration file (**`IConfig`**) to evaluate interrupt operation and performance. The TSIM peripheral configuration file (**`PConfig`**) tells your program how to communicate with the external peripherals used in this implementation.

Figure 19 shows an overview of the simulation environment.

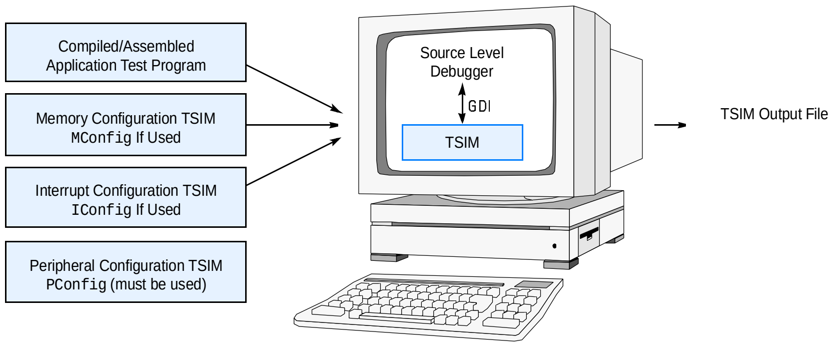
Compiled/Assembled Application Test Program
Memory Configuration TSIM MConfig If Used
Interrupt Configuration TSIM IConfig If Used
Peripheral Configuration TSIM PConfig (must be used)
Source Level Debugger
GDI
TSIM
TSIM Output File

Figure 19: TSIM Simulation Environment

Refer to Siemens Semiconductor’s TriCore Instruction Set Simulator User’s Guide for more information.

10 TriCore-1 Implementation Example

TriCore's convergent microcontroller-DSP architecture enables the lowest system cost design of embedded systems by offering "true" single-chip solutions with on-chip, high-density memories as well as peripherals and customer-specific logic.

Figure 20 shows a generic block diagram of Siemens' first silicon chip, a typical TriCore-1 implementation example. This superscalar implementation contains instruction and data caches, a DMA/PCP module, an interrupt request module, a debug/emulation module, and two miscellaneous peripheral modules. The core and the modules are interconnected via the FPI bus, with up to 32 address bits and 64 data bits, and a peak throughput of 800 Mbytes/s at 100 MHz.

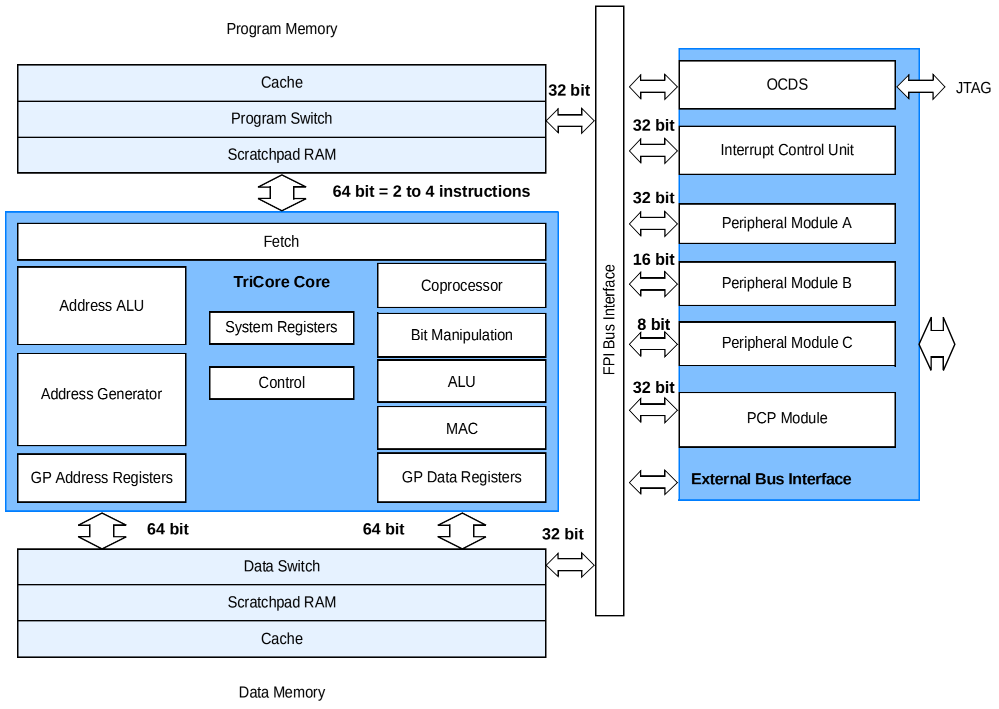

Figure 20: TriCore Chip Example

The DMA accesses to the data memory are cached. The instruction fetch from the FPI Bus is cached. This is necessary because the program is run very often from memories that are connected to the FPI or even the external bus.

The debug mechanism provides easy hardware-software integration through breakpoint support in the core, the debug port that offers access to the breakpoint, other system resources, and the real-time trace port.

11 DSP Example

The TriCore 1 superscalar architecture consists of three units, the Integer Execution Unit, the Load/ Store Unit and the Loop Unit, allowing the issue of up to three instructions per clock cycle. Figure 21 shows the different possible instruction issue combinations. The highest issue rate is achieved when a load/store, integer and loop instruction are all available. This issue rate is easy to reach during the inner loop of of many DSP routines, allowing TriCore to deliver a sustained DSP throughput of 2 16x16 MACs per clock. The example below shows how this works.

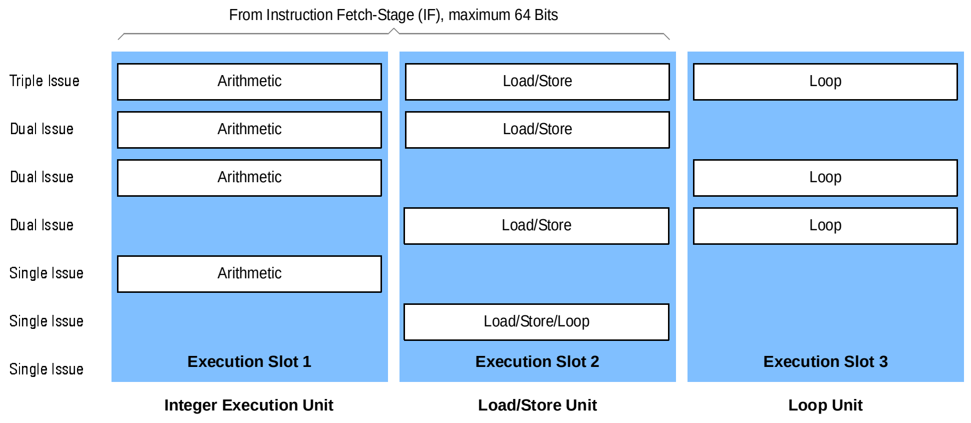

Figure 21: Superscalar Instruction Issue

This superscalar implementation can process two 16x16 Multiply-Accumulates per clock cycle. For example, assume the following equation needs to be calculated:

$$\sum_{i=0}^{n} c_i x_i = c_0 x_0 + c_1 x_1 + c_n x_n$$

Taking the case for n=255 (as in a 256-tap filter), the table below summarizes the execution unit utilization, assuming 16-bit fixed point data. In this example, eight 16x16 MACs are calculated for each loop iteration:

Clock	Integer Unit	Load/Store Unit	Loop Unit
clock 1	-	Load C_0, C_1, C_2, C_3	-
clock 2	-	Load X_0, X_1, X_2, X_3	-
clock 3	MAC C_0X_0, MAC C_1X_1	Load C_4, C_5, C_6, C_7	Loop Start
clock 4	MAC $C_2X_{2,}$, MAC C_3X_3	Load X_4, X_5, X_6, X_7	-
clock 5	MAC C_4X_4, MAC C_5X_5	Load C_8, C_9, C_{10}, C_{11}	-
clock 6	MAC $C_6X_{6,}$,MAC C_7X_7	Load X_8, X_9, X_{10}, X_{11}	Loop
...	...	...	-
clock 130	MAC $C_{254}X_{254}$, MAC $C_{255}X_{255}$	-	-
clock 131	-	Store Result	-

In this example, 16-bit operands are moved four-at-a-time into two 32-bit registers using 64-bit load operations. Eight operands are moved into four registers, then two dual-MAC operations process them. In parallel with this processing, the next 8 operands are moved into four other registers. These other registers are then used in the next two MAC operations. While the next two MACs are being performed, the first set of registers is loaded with the next 8 operands. Thus the loads and MACs are interleaved, with loads "ping-ponging" between two sets of registers. Sustained dual-MAC DSP throughput is thus obtained.

Global PartnerChip for Systems on Silicon

A

Siemens AG Österreich
Erdberger Lände 26
1030 Wien
☎ (++43)-1-1707-35611
Fax (++43)-1-1707-55973

AUS

Siemens Ltd., Head Office
544 Church Street
Richmond (Melbourne), Vic. 3121
☎ (03) 4207111
Tx 30425
Fax (03) 4207275

B

Siemens Electronic Components Benelux
Charleroisesteenweg 116/
Chaussée de Charleroi 116
B-1060 Brussel/Bruxelles
☎ (+32) 2-5362348
Fax (+32) 2-5362857

BR

ICOTRON S.A.
Indústria de Componentes
Eletrônicos
Avenida Mutinga, 3650-6o andar
05150 S_o Paulo-SP
☎ (011) 833-2211
Tx 11-81001
Fax (011) 831-4006

CDN

Siemens Electric Ltd.
Electronic Components Division
1180 Courtney Park Drive
Mississauga, Ontario L5T 1P2
☎ (416) 905-819-8000
Fax (416) 905-819-5744

CH

Siemens Schweiz AG
Bauelemente
Freilagerstraße 28
8047 Zürich
☎ (01) 495-3111
Fax (01) 495-5065

D

Siemens AG
Salzufer 6—8
10587 Berlin
☎ (030) 3863-2626
Fax (030) 3863-2490

Siemens AG
Lahnweg 10
40219 Düsseldorf
☎ (0211) 399-2930
Fax (0211) 399-1481

Siemens AG
Lindenplatz 2
20099 Hamburg
☎ (040) 2889-3819
Fax (040) 2889-3092

Siemens AG
Werner-von-Siemens-Platz 1
30880 Laatzen (Hannover)
☎ (0511) 877-2222
Fax (0511) 877-2078

Siemens AG
Halbleiter Distribution
Richard-Strauss-Straße 76
81679 München
☎ (089) 9221-3133
Fax (089) 9221-2071

Siemens AG
Von-der-Tann-Straße 30
90439 Nürnberg
☎ (0911) 654-7602
Fax (0911) 654-7624

Siemens AG
Weissacher Straße 11
70499 Stuttgart
☎ (0711) 1372864
Fax (0711) 1372448

DK

Siemens A/S
Borupvang 3
2750 Ballerup
☎ 44774477
Tx 1258222
Fax 44774017

E

Siemens S.A.
Dpto. Componentes
Ronda de Europa, 3
28760 Tres Cantos-Madrid
☎ (01) 8030085
Fax (01) 8033926

F

Siemens S.A.
39/47, Bd. Ornano
93527 Saint-Denis CEDEX 2
☎ (1) 49223100
Tx 234077
Fax (1) 49223970

GB

Siemens plc
Siemens House
Oldbury
Bracknell
Berkshire RG12 8FZ
☎ (0344) 396000
Fax (0344) 396632

GR

Siemens AE
Paradissou & Artemidos
P.O.B. 61011
15110 Amaroussio/Athen
☎ (01) 6864111
Tx 216292
Fax (01) 6864299

HK

Siemens Components Ltd
23/F., Tai Yau Building
181 Johnston Road, Wanchai
Hong Kong
☎ (852) 28320500
Fax (852) 28278421

I

Siemens S.p.A.
Semiconductor Sales
Via dei Valtorta, 48
20127 Milano
☎ (02) 6676-1
Fax (02) 6676-4395

IND

Siemens Ltd.
Head Office
134-A, Dr. Annie Besant Road,
Worli
P.O.B. 6597
Bombay 400018
☎ (022) 4938786
Tx 1175142
Fax (022) 4940240

IRL

Siemens Ltd.
Electronic Components Division
8 Raglan Road
Dublin 4
☎ (01) 6684727
Tx 93744
Fax (01) 684633

J

Siemens Components K.K.
Shinjuku Koyama Bldg. 2F
30-3, 4-Chome
Yoyogi, Shibuya-ku
Tokyo 151
☎ (81) 3-53888525
Fax (81) 3-33769792

N

Siemens A/S
_stre Aker vei 90
Postboks 10, Veitvet
0518 Oslo 5
☎ (02) 633000
Tx 78477
Fax (02) 633805

NL

Siemens Electronic Compo-nents Benelux
Postbus 16068
NL-2500 BB Den Haag
☎ (+31) 70-3332429
Fax (+31) 70-3332815

P

Siemens S.A.
Estrada Nacional 117, Km 2,6
Alfragide
2700 Amadora
☎ (01) 4170011
Tx 62955
Fax (01) 4172870

PL

Siemens Sp. z.o.o.
ul. Stawki 2
POB 276
00-950 Warszawa
☎ 6351619
Tx 825554
Fax 6355238

RC

Tai Engineering Co., Ltd.
6th Fl., Central Building
108, Chung Shan North Road, Sec. 2
P.O. Box 68-1882
Taipei 10449
☎ (02) 5234700
Tx 27860 taiengco
Fax (02) 5367070

ROC

Siemens Ltd.
Asia Tower Bldg, 10th floor
726 Yeoksam-dong, Kangnam-ku
CPO Box 3001, Seoul 135-080
Korea
☎ (822) 5277740
Fax (822) 5277779

Siemens AG
1. Donskoj pr., 2
Moskva 117419
☎ (095) 237-6476, -6911
Tx 414385
Fax (095) 237-6614

S

Siemens Components
Österögatan 1
Box 46
S-164 93 Kista
☎ (08) 7033500
Tx 11672
Fax (08) 7033501

FIN

Siemens Oy
P.O.B. 60
02601 Espoo
☎ (0) 51051, y 124465
Fax (0) 51052398

SGP

Siemens Components Pte. Ltd.
166 Kallang Way
Singapore 1334
☎ (65) 8400600
Fax (65) 7421080

TR

SIMKO Ticaret ve Sanayi A.S.
Meclisi Mebusan Cad. No. 125
P.K. 1001, 80007 Karaköy
80040 Findikli
☎ (01) 2510900
Tx 24233 sies tr
Fax (01) 2524134

USA

Siemens Microelectronics, Inc.
1730 North First Street
San Jose, CA 95112
☎ (408) 501-6000
Fax (408) 501-2424

ZA

Siemens Ltd.
Siemens House,
P.O.B. 4583
Johannesburg 2000
☎ (011) 3151950
Tx 450091
Fax (011) 3151968

http://www.siemens.de/Semiconductor/index.htm
USA: http://www.smi.siemens.com/

Total Quality Management

Quality takes on an all-encompassing significance at the Siemens Semiconductor Group. For us it means living up to each and every one of your demands in the best possible way. So we are not only concerned with product quality. We direct our efforts equally at quality of supply and logistics, service and support, as well as all the other ways in which we advise and attend to you.

Part of Siemens' quality is the very special attitude of our staff. Total Quality in thought and deed, towards co-workers, suppliers and you, our customer. Our guideline is "do everything with zero defects", in an open manner that is demonstrated beyond your immediate workplace, and to constantly improve. Throughout the corporation, we also think in terms of Time Optimized Processes (TOP), greater speed on our part to give you that decisive competitive edge.

Give us the chance to prove the best of performance through the best of quality—you will be convinced.