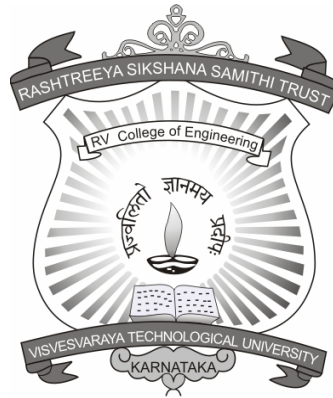


Seminar Report
On

The C6000 Family: Architecture, Pipelining and General Trends

2006 – 2007



Dept. of Telecommunication & PG Studies
R.V. College of Engineering
Bangalore – 59

SATISH B D
1RV03TE047

R. V. College of Engineering
(Affiliated to Visvesvaraya Technological University)

Department of Telecommunication Engineering & PG Studies

2006 – 2007



Certificate

This is to certify that the seminar work titled
**“The C6000 Family: Architecture, Pipelining and
General Trends ”**
has been successfully completed by the following student

SATISH B D
1RV03TE047

at R.V. College of Engineering, Bangalore,
during the academic year 2006 – 2007 as a partial fulfillment of the academic
requirement for the completion of the 8th semester of Visvesvaraya Technological
University.

Seminar Coordinator:
Mr. P. Nagaraju

Prof. K. N. Raja Rao
Head of the Department

CONTENTS

1	The Necessity of Digital Signal Processors	5
2	History of Digital Signal Processors	
2.1	The First Generation	7
2.2	The Second Generation	7
2.3	The Third Generation	8
2.4	The Fourth Generation	8
3	The Chip-Makers	10
4	The TMS320C6000 Family	
4.1	General Features of the C6000	12
4.2	Features Unique to C6400	14
4.3	Features Unique to C6700	15
5	Processor Architectures	
5.1	Von Neumann Architecture	16
5.2	Harvard Architecture	17
5.3	VLIW Architecture	18
5.4	The VelociTI™ Architecture	19
6	The C6000 DSP Core	
6.1	Data path and Control	21
6.2	Instruction Set Mapping	28
6.3	Register Usage	29
7	The Addressing Modes	
7.1	Types of Addressing Modes	30
7.2	Circular Addressing Mode	31
7.3	The Addressing Mode Register (AMR)	32
8	Pipelining and Parallelism	
8.1	The C6000 Pipeline Phases	34

8.2 Vector Summation : An Example	36
8.3 Vector Summation Without Pipelining	37
8.4 Vector Summation With Pipelining	38
9 The Memory Map of C6000	40
10 The Peripherals of C6000	41
11 The DaVinci™ Technology	
11.1 The Origin of the DaVinci™ Effect	43
11.2 Existing Systems	44
11.3 Peripherals and Operating System	44
11.4 Applications	45
12 Conclusions	46
13 References	47

1. The Necessity of Digital Signal Processors

In the 1960s it was predicted that artificial intelligence would revolutionize the way humans interact with computers and other machines. It was believed that by the end of the century we would have robots cleaning our houses, computers driving our cars, and voice interfaces controlling the storage and retrieval of information. This hasn't happened; these abstract tasks are far more complicated than expected, and very difficult to carry out with the step-by-step logic provided by digital computers.

However, the last forty years have shown that computers are extremely capable in two broad areas, (1) **data manipulation**, such as word processing and database management, and (2) **mathematical calculation**, used in science, engineering, and Digital Signal Processing. All microprocessors can perform both tasks; however, it is difficult (expensive) to make a device that is optimized for both. There are technical tradeoffs in the hardware design, such as the size of the instruction set and how interrupts are handled. Even more important, there are marketing issues involved: development and manufacturing cost, competitive position, product lifetime, and so on. As a broad generalization, these factors have made traditional microprocessors, such as the Pentium® which is primarily directed at data manipulation. Similarly, DSPs are designed to perform the mathematical calculations needed in Digital Signal Processing.

Figure 1 lists the most important differences between these two categories. Data manipulation involves storing and sorting information. For instance, consider a word processing program. The basic task is to store the information (typed in by the operator), organize the information (cut and paste, spell checking, page layout, etc.), and then retrieving the information (for example, printing a document with a laser printer). These tasks are accomplished by *moving* data from one location to another, and *testing* for inequalities ($A=B$, $A<B$, etc.).

	Data Manipulation	Math Calculation
Typical Applications	Word processing, database management, spread sheets, operating systems, etc.	Digital Signal Processing, motion control, scientific and engineering simulations, etc.
Main Operations	data movement (A --> B) value testing (If A== B then ...)	addition ($A+B=C$) multiplication ($A\times B=C$)

Figure 1: Data Manipulation and Math Calculation

In comparison, most DSPs are used in applications where the processing is continuous, not having a defined start or end. For instance, consider an engineer designing a DSP system for an audio signal, such as a hearing aid. If the digital signal is being received at 20,000 samples per second, the DSP must be able to maintain a sustained throughput of 20,000 samples per second. However, there are important reasons not to make it any faster than necessary. As the speed increases, so does the *cost*, the *power consumption*, the *design difficulty*, and so on. This makes an accurate knowledge of the execution time critical for selecting the proper device, as well as the algorithms that can be applied.

2. History of Digital Signal Processors

In 1978, Intel released the 2920 as an "analog signal processor". It had an on-chip ADC/DAC with an internal signal processor, but it didn't have a hardware multiplier and was not successful in the market. In 1979, AMI released the S2811. It was designed as a microprocessor peripheral, and it had to be initialized by the host. The S2811 was likewise not successful in the market.

- **2.1 The First Generation (1979 to 1987)**

In 1979, Bell Labs introduced the first single chip DSP, the Mac 4 Microprocessor. Then, in 1980 the first stand-alone, complete DSPs -- the NEC μ PD7720 and AT&T DSP1 -- were presented at the IEEE International Solid-State Circuits Conference '80. Both processors were inspired by the research in PSTN telecommunications.

The first DSP produced by Texas Instruments (TI), the TMS32010 presented in 1983, proved to be an even bigger success. It was based on the Harvard architecture, and so had separate instruction and data memory. It already had a special instruction set, with instructions like load-and-accumulate or multiply-and-accumulate. It could work on 16-bit numbers and needed 390ns for a multiply-add operation. TI is now the market leader in general purpose DSPs. Another successful design was the Motorola 56000.

- **2.2 The Second Generation (1988 to 1995)**

About five years later, the second generation of DSPs began to spread. They had 3 memories for storing two operands simultaneously and included hardware to accelerate tight loops; they also had an addressing unit capable of loop-addressing. Some of them operated on 24-bit variables and a typical model only required about

21ns for a MAC (multiply-accumulate). Members of this generation were for example the AT&T DSP16A or the Motorola DSP56001.

- **2.3 The Third Generation (1996 to 2002)**

The main improvement in the third generation was the appearance of application-specific units and instructions in the data path, or sometimes as coprocessors. These units allowed direct hardware acceleration of very specific but complex mathematical problems, like the Fourier-transform or matrix operations. Some chips, like the Motorola MC68356, even included more than one processor cores to work in parallel. Other DSPs from 1995 are the TI TMS320C541 or the TMS 320C80.

- **2.4 The Fourth Generation (2002 onwards)**

The fourth generation is best characterized by the changes in the instruction set and the instruction encoding/decoding. SIMD and MMX extensions were added; VLIW and the superscalar architecture appeared. As always, the clock-speeds have increased, a 3ns MAC became now possible.

DSPs in 2007

Today's signal processors yield much greater performance. This is due in part to both technological and architectural advancements like lower design rules, fast-access two-level cache, (E) DMA circuit and a wider bus system. Of course, not all DSPs provide the same speed and many-many kind of signal processors exist, each one of them being better suited for a specific task, ranging in price from about 1.50 to 300 dollars. A Texas Instruments C6000 series DSP clocks at 1Ghz and implements separate instruction and data caches as well as a 8Mbyte 2nd level cache, and its I/O speed is rapid thanks to its 64 EDMA channels. The top models are capable of even 8000 MIPS (million instructions per second), use VLIW encoding, perform eight operations per clock-cycle and are compatible with a broad range of external peripherals and various buses (PCI/serial/etc).

Another big signal processor manufacturer today is Analog Devices. The company provides a broad range of DSPs, but its main portfolio is multimedia processors, such as codecs, filters and digital-analog converters. Its SHARC-based processors range in performance from 66Mhz/198MFLOPS (million floating-point operations per second) to 400Mhz/2400MFLOPS. Some models even support multiple multipliers and ALUs, SIMD instructions and audio processing-specific components and peripherals. Another product of the company is the Blackfin family of embedded digital signal processors, with models like the ADSP-BF531 to ADSP-BF536. These processors combine the features of a DSP with those of a general use processor. As a result, these processors can run simple operating systems like μ CLinux, velOSity and Nucleus while operating relatively efficiently on real-time data.

Most DSPs use fixed-point arithmetic, because in real world signal processing, the additional range provided by floating point is not needed, and there is a large speed benefit; however, floating point DSPs are common for scientific and other applications where additional range or precision may be required. General purpose CPU's have ideas and influences from digital signal processors with extensions such as the MMX extensions in the Intel IA-32 architecture instruction set.

Generally, DSPs are dedicated integrated circuits; however DSP functionality can also be realized using Field Programmable Gate Array chips.

3. The Chip-Makers

The DSP market is very large and growing rapidly. As shown in Figure 2, it will be about 8-10 billion dollars/year at the turn of the century, and growing at a rate of 30-40% each year. This is being fueled by the incessant demand for better and cheaper consumer products, such as: cellular telephones, multimedia computers, and high-fidelity music reproduction. These high-revenue applications are shaping the field, while less profitable areas, such as scientific instrumentation, are just riding the wave of technology.

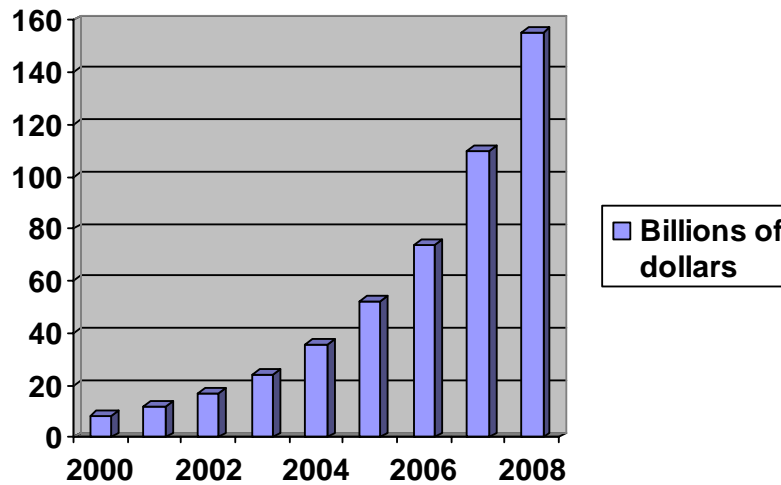


Figure 2: Revenue from the DSP Industry

DSPs can be purchased in three forms, as a **core**, as a **processor**, and as a **board level** product. In DSP, the term "core" refers to the section of the processor where the key tasks are carried out, including the data registers, multiplier, ALU, address generator, and program sequencer. A complete **processor** requires combining the core with memory and interfaces to the outside world. While the core and these peripheral sections are designed separately, they will be fabricated on the

same piece of silicon, making the processor a single integrated circuit. Lastly, there are several dozen companies that will sell you DSPs already mounted on a printed circuit board. These have such features as extra memory, A/D and D/A converters, EPROM sockets, multiple processors on the same board, and so on.

The present day Digital Signal Processor market (2007) is dominated by four companies. Here is a list, and the general scheme they use for numbering their products:

- **Analog Devices** (www.analog.com/dsp)
 - ADSP-21xx 16 bit, fixed point
 - ADSP-21xxx 32 bit, floating and fixed point

- **Lucent Technologies** (www.lucent.com)
 - DSP16xxx 16 bit fixed point
 - DSP32xx 32 bit floating point

- **Motorola** (www.mot.com)
 - DSP561xx 16 bit fixed point
 - DSP560xx 24 bit, fixed point
 - DSP96002 32 bit, floating point

- **Texas Instruments** (www.ti.com)
 - TMS320Cxx 16 bit fixed point
 - TMS320Cxx 32 bit floating point

4. The TMS320C6000 Family

In 1982, Texas Instruments (TI) introduced the TMS32010 — the first fixed-point DSP in the TMS320 family. Before the end of the year, Electronic Products magazine awarded the TMS32010 the title “Product of the Year”. Today, the TMS320 family consists of many generations:

- C1x, C2x, C2xx, C5x, and C54x fixed-point DSPs
- C3x and C4x floating-point DSPs, and
- C8x multiprocessor DSPs.

The TMS320C6000™ digital signal processor (DSP) platform is part of the TMS320™ DSP family. Refer Figure 3. The TMS320C62x™ DSP generation and the TMS320C64x™ DSP generation comprise fixed-point devices in the C6000™ DSP platform, and the TMS320C67x™ DSP generation comprises floating-point devices in the C6000 DSP platform. The TMS320C62x and TMS320C64x™ DSPs are code-compatible. The TMS320C62x and TMS320C67x DSPs are code-compatible. All three DSPs use the VelociTI™ architecture, a high-performance, advanced VLIW (very long instruction word) architecture, making these DSPs excellent choices for multi-channel and multifunction applications.

Now there is a new generation of DSPs, the TMS320C6x™ generation, with performance and features that are reflective of Texas Instruments commitment to lead the world in DSP solutions.

- **4.1 General Features of the C6000**

With a performance of up to 6000 million instructions per second (MIPS) and an efficient C compiler, the TMS320C6x DSPs give system architects unlimited possibilities to differentiate their products. High performance, ease of use and

affordable pricing make the TMS320C6x generation the ideal solution for multi-channel, multifunction applications, such as:

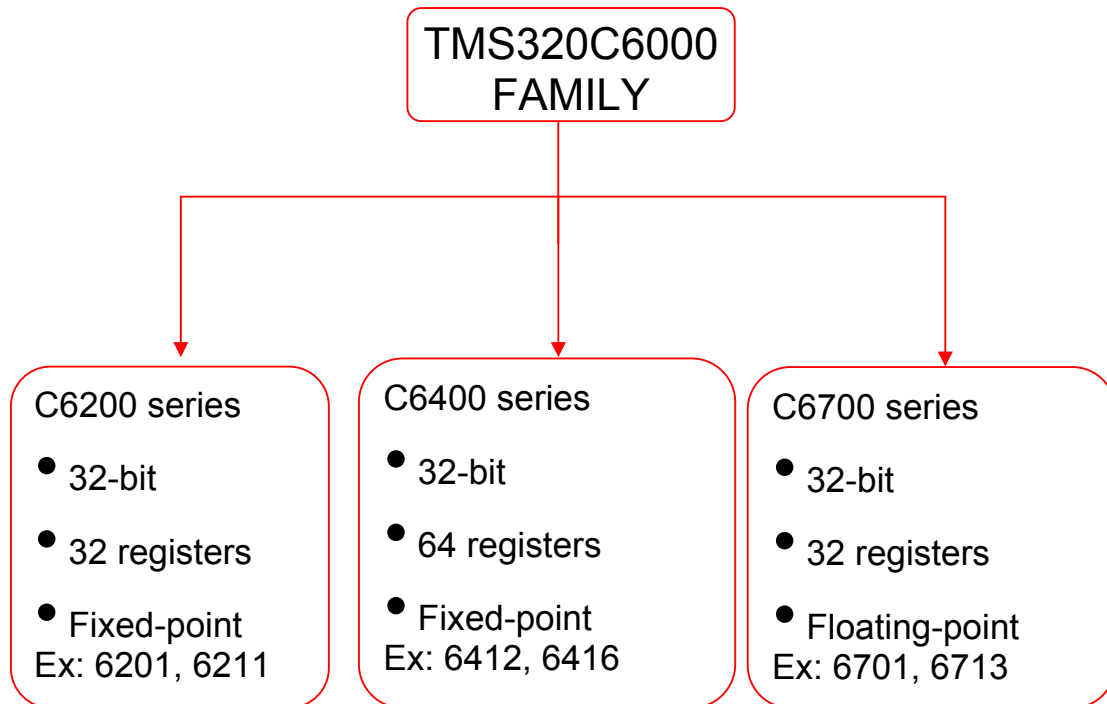


Figure 3: The TMS320C6000 Family

- Pooled modems
- Wireless local loop base stations
- Remote access servers (RAS)
- Digital subscriber loop (DSL) systems
- Cable modems
- Multi-channel telephony systems

The C6000 devices execute up to eight 32-bit instructions per cycle. The C62x/C67x device's core CPU consists of 32 general-purpose registers of 32-bit word length and eight functional units. The C64x core CPU consists of 64 general-purpose 32-bit registers and eight functional units. These eight functional units contain:

- Two multipliers
- Six ALUs

Features of the C6000 devices include:

- Advanced VLIW CPU with eight functional units, including two multipliers and six arithmetic units
 - Executes up to eight instructions per cycle for up to ten times the performance of typical DSPs
 - Allows designers to develop highly effective RISC-like code for fast development time
 - Instruction packing
 - Gives code size equivalence for eight instructions executed serially or in parallel
 - Reduces code size, program fetches, and power consumption
 - Conditional execution of all instructions
 - Reduces costly branching
 - Increases parallelism for higher sustained performance
 - Efficient code execution on independent functional units
 - Industry's most efficient C compiler on DSP benchmark suite
 - Industry's first assembly optimizer for fast development and improved parallelization
 - 8/16/32-bit data support, providing efficient memory support for a variety of applications
 - 40-bit arithmetic options add extra precision for vocoders and other computationally intensive applications
 - Saturation and normalization provide support for key arithmetic operations
 - Field manipulation and instruction extract, set, clear, and bit counting support common operation found in control and data manipulation applications.
- **4.2 Features Unique to the C6700**

The C67x has these additional features:

 - Hardware support for single-precision (32-bit) and double-precision (64-bit) IEEE floating-point operations
 - 32 x 32-bit integers multiply with 32- or 64-bit result.

- The TMS320C67x™ floating-point DSP uses all of the instructions available to the TMS320C62x™, but it also uses other instructions that are specific to the C67x™.
- These specific instructions are for 32-bit integer multiply, doubleword load, and floating-point operations, including addition, subtraction, and multiplication.

- **4.3 Features Unique to the C6400**

The C64x has these additional features:

- Each multiplier can perform two 16 x 16-bit or four 8 x 8 bit multiplies every clock cycle.
- Quad 8-bit and dual 16-bit instruction set extensions with data flow support
- Support for non-aligned 32-bit (word) and 64-bit (double word) memory accesses
- Special communication-specific instructions have been added to address common operations in error-correcting codes.
- Bit count and rotate hardware extends support for bit-level algorithms.

5. Processor Architectures

One of the biggest bottlenecks in executing DSP algorithms is transferring information to and from memory. This includes *data*, such as samples from the input signal and the filter coefficients, as well as *program instructions*, the binary codes that go into the program sequencer. For example, suppose we need to multiply two numbers that reside somewhere in memory. To do this, we must fetch three binary values from memory, the numbers to be multiplied, plus the program instruction describing what to do.

- **5.1 Von Neumann Architecture**

The Figure 4 shows how this seemingly simple task is done in a traditional microprocessor. This is often called **Von Neumann architecture**, after the brilliant American mathematician John Von Neumann (1903-1957). Von Neumann guided the mathematics of many important discoveries of the early twentieth century. His many achievements include: developing the concept of a stored program computer, formalizing the mathematics of quantum mechanics, and work on the atomic bomb. If it was new and exciting, Von Neumann was there!

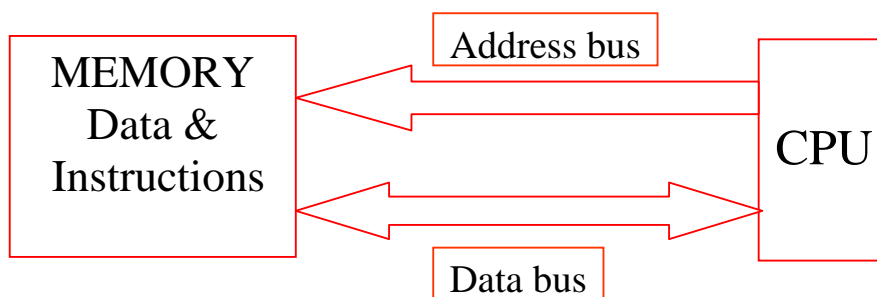


Figure 4: Von Neumann Architecture

As shown in Figure 4, Von Neumann architecture contains a single memory and a single bus for transferring data into and out of the central processing unit (CPU). Multiplying two numbers requires at least three clock cycles, one to transfer each of the three numbers over the bus from the memory to the CPU. We don't count the time to transfer the result back to memory, because we assume that it remains in the CPU for additional manipulation (such as the sum of products in an FIR filter).

The Von Neumann design is quite satisfactory when you are content to execute all of the required tasks in serial. In fact, most computers today are of the Von Neumann design. We only need other architectures when very fast processing is required, and we are willing to pay the price of increased complexity.

- **5.2 Harvard Architecture**

This leads us to the **Harvard** architecture, shown in Figure 5. This is named for the work done at Harvard University in the 1940s under the leadership of Howard Aiken (1900-1973). As shown in this illustration, Aiken insisted on separate memories for data and program instructions, with separate buses for each. Since the buses operate independently, program instructions and data can be fetched at the same time, improving the speed over the single bus design. Most present day DSPs use this quadruple bus architecture.

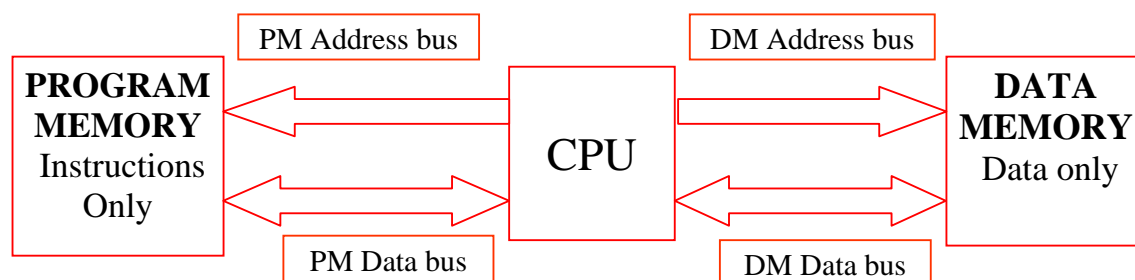


Figure 5: Harvard Architecture

PM: Program Memory

DM: Data Memory

A handicap of the basic Harvard design is that the data memory bus is busier than the program memory bus. When two numbers are multiplied, two binary values (the numbers) must be passed over the data memory bus, while only one binary value (the program instruction) is passed over the program memory bus.

- **5.3 VLIW Architecture**

VLIW stands for Very Long Instruction Word. This architecture was introduced by Texas Instruments. As the name itself denotes, we fetch a “long instruction”. Meaning, in the C6000, **eight** instructions (this is definitely long!) are always fetched in every clock cycle. This constitutes a *fetch packet*.

A traditional VLIW architecture consists of multiple execution units running in parallel, performing multiple instructions during a single clock cycle. Refer Figure 6. Here there are three ALUs (i.e. execution units) that share the same program and data memory. Each ALU has its own address and data bus which is independent of all other buses.

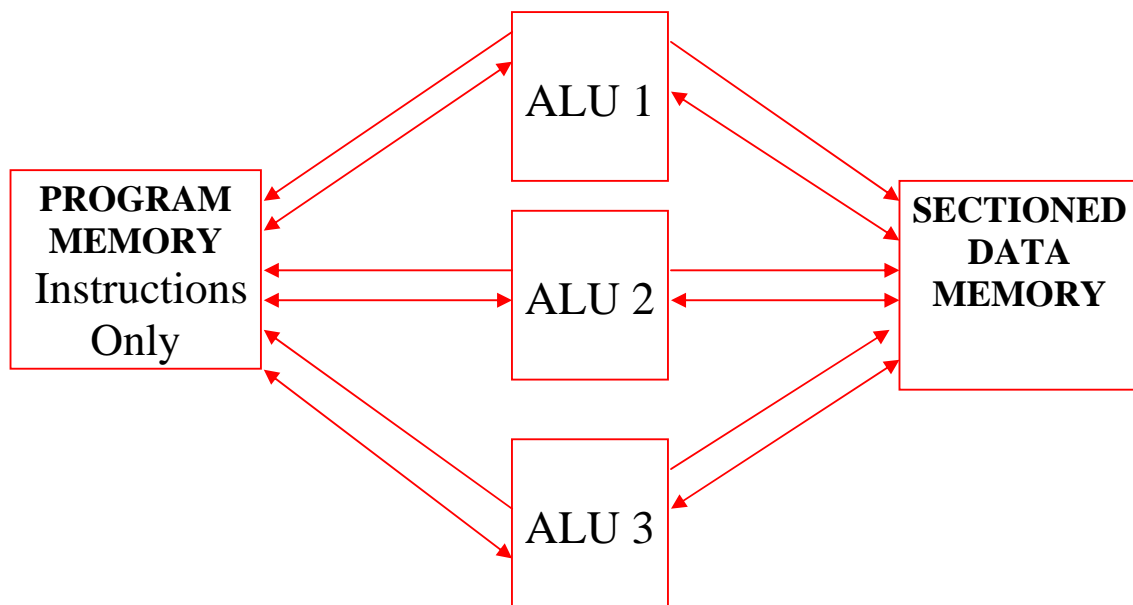


Figure 6: VLIW Architecture

A major difference between VLIW and the previous architectures is the presence of *sectioned* data memory. The memory is divided into different sections such as stack, heap, const, text, var (variables), args (arguments), cio (I/O in C language), etc. The user can explicitly decide the beginning and end of each memory section, which is not possible in previous architectures.

• **5.4 The VelociTI™ Architecture**

The VelociTI architecture of the C6000 platform of devices makes them the first off-the-shelf DSPs to use advanced VLIW to achieve high performance through increased instruction-level parallelism.

VelociTI is a highly deterministic architecture, having few restrictions on how or when instructions are fetched, executed, or stored. VelociTI's advanced features include:

- Instruction packing: reduced code size
- All instructions can operate conditionally: flexibility of code
- Variable-width instructions: flexibility of data types
- Fully pipelined branches: zero-overhead branching.

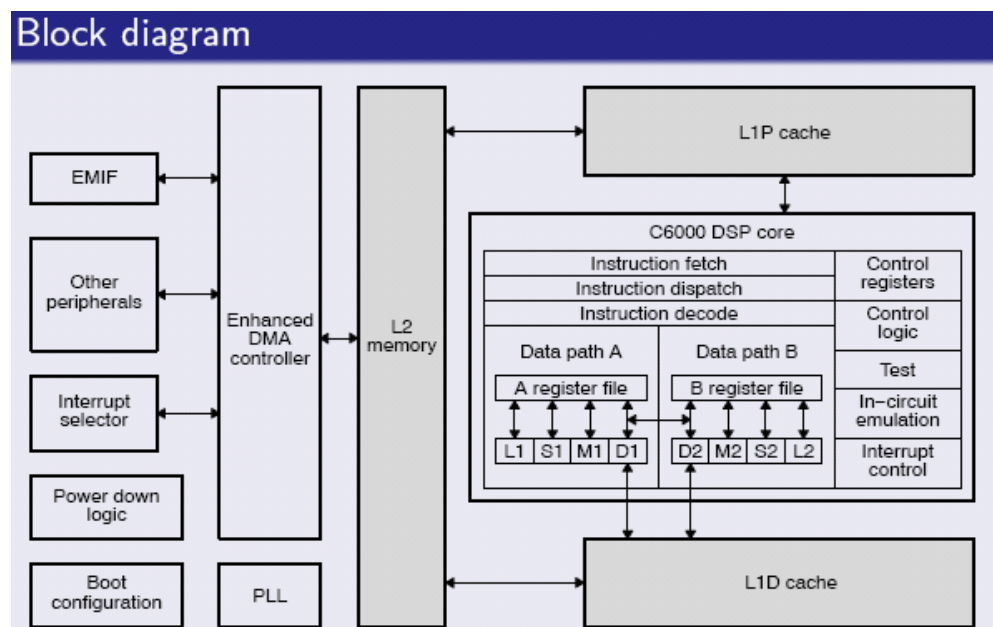


Figure 7: Block Diagram of the C6000

Other on-board peripherals include:

- (a) Enhanced Direct Memory Access (EDMA) controller
- (b) External Memory Interface (EMIF)
- (c) Interrupt selector
- (d) Power-down logic
- (e) Phase-Locked Loop (PLL) controller

6. The C6000 DSP Core

The DSP core consists of registers and the functional units which can be programmed by the user. The functions of the various components which make-up the C6000 core (Figure 8) are explained next.

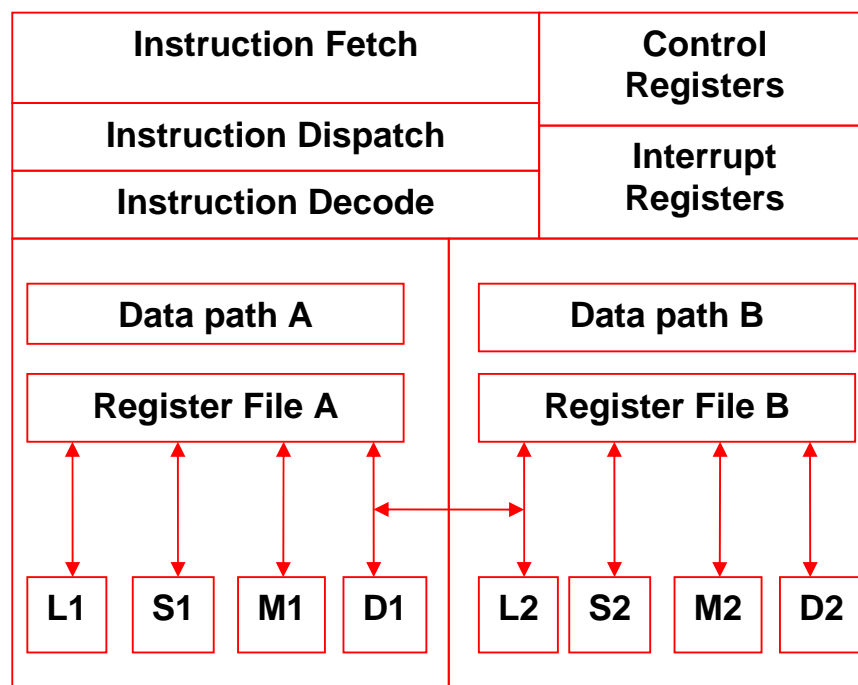


Figure 8: The C6000 Core

The C6000 CPU, shown in Figure 8, is common to all the C62x/C64x/C67x devices.

The CPU contains:

- Program fetch unit
- Instruction dispatch unit, advanced instruction packing (C64 only)
- Instruction decode unit
- Two data paths, each with four functional units

- 32 32-bit registers, 64 32-bit registers (C64 only)
- Control registers
- Control logic
- Test, emulation, and interrupt logic

The program fetch, instruction dispatch, and instruction decode units can deliver up to eight 32-bit instructions to the functional units every CPU clock cycle. The processing of instructions occurs in each of the two data paths (A and B), each of which contains four functional units (.L, .S, .M, and .D) and 16 32-bit general-purpose registers for the C62x/C67x and 32 32-bit general purpose registers for the C64x. The data paths are described in more detail in section 6.1. A control register file provides the means to configure and control various processor operations. To understand how instructions are fetched, dispatched, decoded, and executed in the data path, see Section 8, “Pipelining and Parallelism”.

Internal Memory

The C62x, C64x & C67x have a 32-bit, byte-addressable address space. Internal (on-chip) memory is organized in separate data and program spaces. When off-chip memory is used, these spaces are unified on most devices to a single memory space via the external memory interface (EMIF). The C62x and C67x have two 32-bit internal ports to access internal data memory. The C64x has two 64-bit internal ports to access internal data memory. The C62x, C64x & C67x have a single internal port to access internal program memory, with an instruction-fetch width of 256 bits.

- **6.1 Data path and Control**

Register Files

There are two general-purpose register files (A and B) in the C6000™ data paths. For the C62x/C67x DSPs, each of these files contains 16 32-bit registers (A0–A15 for file A and B0–B15 for file B). The general-purpose registers can be used for data; data address pointers, or condition registers. The C64x DSP register file

doubles the number of general-purpose registers that are in the C62x/C67x cores, with 32 32-bit registers (A0–A31 for file A and B0–B31 for file B).

The C62x/C67x general-purpose register files support data ranging in size from packed 16-bit data through 40-bit fixed-point and 64-bit floating point data. Values larger than 32 bits, such as 40-bit long and 64-bit float quantities are stored in register pairs (Figure 9). In these the 32 LSBs of data are placed in an even-numbered register and the remaining 8 or 32 MSBs in the next upper register (which is always an odd-numbered register). The C64x register file extends this by additionally supporting packed 8-bit types and 64-bit fixed-point data types. (The C64x does not directly support floating-point data types.) Packed data types store either four 8-bit values or two 16-bit values in a single 32-bit register, or four 16-bit values in a 64-bit register pair.

Figure 10 illustrates the register storage scheme for 40-bit long data. Operations requiring a long input ignore the 24 MSBs of the odd-numbered register. Operations producing a long result fill the 24 MSBs of the odd-numbered register with zeros. The even-numbered register is encoded in the opcode.

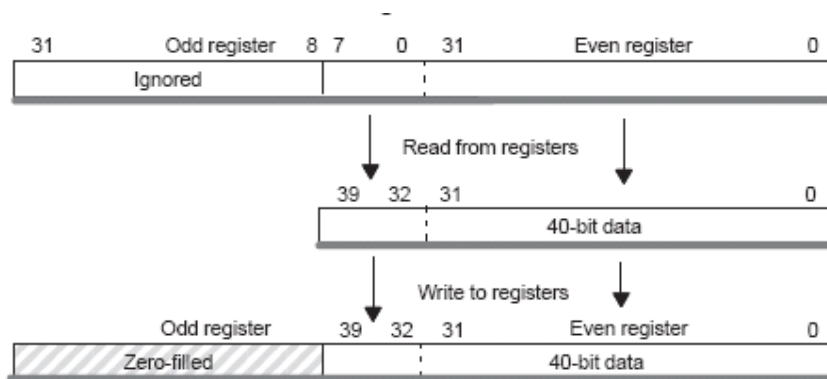


Figure 10: 40-bit long integer

Register Files		Applicable devices	
A	B		
A1:A0	B1:B0	C62x, C64x and C67x	
A3:A2	B3:B2		
A5:A4	B5:B4		
A7:A6	B7:B6		
A9:A8	B9:B8		
A11:A10	B11:B10		
A13:A12	B13:B12		
A15:A14	B15:B14		
A17:A16	B17:B16		C67x only
A19:A18	B19:B18		
A21:A20	B21:B20		
A23:A22	B23:B22		
A25:A24	B25:B24		
A27:A26	B27:B26		
A29:A28	B29:B28		
A31:A30	B31:B30		

Figure 9: Register Pair Structure

Functional Units

The eight functional units in the C6000 data paths can be divided into two groups of four; each functional unit in one data path is almost identical to the corresponding unit in the other data path. The functional units are described in Figure 11.

Besides being able to perform all the C62x instructions, the C64x also contains many 8-bit to 16-bit extensions to the instruction set. For example, the **MPYU4** instruction performs four 8x8 unsigned multiplies with a single instruction on an .M unit. The **ADD4** instruction performs four 8-bit additions with a single

instruction on an .L unit. The additional C64x operations are shown in boldface in Figure 11.

Functional Unit	Fixed-point Operations	Floating-point Operations
.L unit (.L1 and .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits Byte shifts Data packing/unpacking 5-bit constant generation Dual 16-bit arithmetic Quad 8-bit arithmetic Dual 16-bit min/max Quad 8-bit min/max	Arithmetic operations DP→SP, INT→DP, INT→SP conversion operations
.S unit (.S1 and .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) Byte shifts Data packing/unpacking Dual 16-bit compare Quad 8-bit compare Dual 16-bit shift Dual 16-bit saturated arithmetic Quad 8-bit saturated arithmetic	Compare Reciprocal and reciprocal square-root Operations Absolute value operations SP→DP conversion operations

.M unit (.M1 and .M2)	16 x 16 multiply operations 16 x 32 multiply operations Quad 8 x 8 multiply operations Dual 16 x 16 multiply operations Dual 16 x 16 multiply with add/subtract operations Quad 8 x 8 multiply with add Bit expansion Bit interleaving/de-interleaving Variable shift operations Rotation Galois Field Multiply	32 x 32-bit fixed-point multiply operations Floating-point multiply operations
.D unit (.D1 and .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit offset Loads and stores with 15-bit offset Load and store double words with 5-bit constant Load and store non-aligned words and double words 5-bit constant generation 32-bit logical operations	Load doubleword with 5-bit constant offset

Figure 11: Functional units and their operations

Control Register File

One unit (.S2) can read from and write to the control register file, as shown in this section. Figure 12 lists the control registers contained in the control register file and describes each. If more information is available on a control register, the table lists where to look for that information. Each control register is accessed by the **MVC** instruction.

Additionally, some of the control register bits are specially accessed in other ways. For example, arrival of a maskable interrupt on an external interrupt pin, $INT\ m$, triggers the setting of flag bit $IFR\ m$. Subsequently, when that interrupt is processed, this triggers the clearing of $IFR\ m$ and the clearing of the global interrupt enable bit, GIE. Finally, when that interrupt processing is complete, the B IRP instruction in the interrupt service routine restores the pre-interrupt value of the GIE. Similarly, saturating instructions like SADD set the SAT (saturation) bit in the CSR (Control Status Register).

Abbr.	Register Name	Description
AMR	Addressing mode register	Specifies whether to use linear or circular addressing for each of eight registers; also contains sizes for circular addressing
CSR	Control status register	Contains the global interrupt enable bit, cache control bits, and other miscellaneous control and status bits
IFR	Interrupt flag register	Displays status of interrupts
ISR	Interrupt set register	Allows manually setting pending interrupts
ICR	Interrupt clear register	Allows manually clearing pending interrupts
IER	Interrupt enable register	Allows enabling/disabling of individual interrupts
ISTP	Interrupt service table pointer	Points to the beginning of the interrupt service table
IRP	Interrupt return pointer	Contains the address to be used to return from a maskable interrupt
NRP	Nonmaskable interrupt return pointer	Contains the address to be used to return from a nonmaskable interrupt
PCE1	Program counter, E1 phase	Contains the address of the fetch packet that is in the E1 pipeline stage

Figure 12: Control Registers

- **6.2 Instruction Set Mapping**

Figure 13 shows the mapping between instruction set and the functional units.

.L Unit	.M Unit	.S Unit	.D unit
ABS	MPY	ADD SET	ADD
ADD	MPYU	ADDK SHL	ADDAB
ADDU	MPYUS	ADD2 SHR	ADDAH
AND	MPYSU	AND SHRU	LDB
CMPEQ	MPYH	B disp SSSL	LDBU
CMPGT	MPYHU	B IRP SUB	LDH
CMPGTU	MPYHUS	B NRP SUBU	LDHU
CMPLT	MPYHSU	B reg SUB2	LDW
CMPLTU	MPYHL	CLR XOR	MV
LMBD	MPYHLU	EXT ZERO	STB
MV	MPYHULS	EXTU	STH
NEG	MPYHSLU	MV	STW
NORM	MYPLH	MVC	SUB
NOT	MPYLHU	MVK	SUBAB
OR	MPYLUHS	MVKH	SUBAH
SADD	MPYLSHU	MVKLH	SUBAW
SAT	SMPY	NEG	ZERO
SSUB	SMPYHL	NOT	
SUB	SMPYLH	OR	
SUBU	SMPYH		
SUBC			
XOR			
ZERO			

Figure 13: Instruction Set Mapping

The fact that each functional unit can execute only a particular instruction is extremely important when writing assembly programs. This information must be entered in every assembly instruction.

- **6.3 Register Usage**

All instructions are conditional instructions. If no condition is specified, it is *always* executed. Conditional instructions are represented in code by using square brackets, [], surrounding the condition register name. The following execute packet contains two ADD instructions in parallel. The first ADD is conditional on B0 being nonzero. The second ADD is conditional on B0 being zero. The character “!” (exclamation mark) indicates the inverse of the condition.

```
[ B0 ] ADD .L1 A1,A2,A3      ;executes if B0 is nonzero
|| [ ! B0 ] ADD .L2 B1,B2,B3 ;executes if B0 = 00000000h
```

The above instructions are mutually exclusive. This means that only one will execute. Only A1, A2, B0, B1 and B2 registers can be used as conditional registers. No two instructions within the same execute packet can use the same resources. Also, no two instructions can write to the same register during the same cycle.

The registers A4, A5, A6, A7, B4, B5, B6 and B7 can be used for circular addressing. No other registers can be used for this purpose. Also some registers are assigned some special purpose:

- B15 is the Stack pointer (SP)
- A15 is the Frame Pointer (FP)
- A14 is the Data Page Pointer (DPP)

The programmer must avoid using A14, A15 and B15 as much as possible.

7. The Addressing Modes

Addressing Modes are methods used to specify the address of an operand in assembly instructions.

- **7.1 Types of Addressing Modes**

The addressing modes on the C62x, C64x, and C67x are

- Linear mode
- Circular mode using BK0
- Circular mode using BK1

The mode is specified by the addressing mode register, or AMR (Figure 12 and section 7.3). All registers can perform linear addressing. Only eight registers can perform circular addressing: A4–A7 are used by the .D1 unit and B4–B7 are used by the .D2 unit. No other units can perform circular addressing. LDB(U)/LDH(U)/LDW, STB/STH/STW, ADDAB/ADDAH/ADDAW/ADDAD, and SUBAB/SUBAH/SUBAW instructions all use the AMR to determine what type of address calculations are performed for these registers.

If no addressing mode is explicitly written into the AMR then linear addressing mode is used by default. The different ways of addressing modes are:

Register Addressing Mode: The operand is the contents of a processor register; the name of the register is given in the instruction.

Ex: ADD .L1 A1, A2, A3 ; A1 + A2 = A3
 SUB .L2 B1, B2, B6 ; B1 – B2 = B6

Note that the functional unit is a must in writing assembly instructions.

Immediate Addressing Mode: The operand is a numeric constant which is directly specified in the instruction.

Ex: ADD .L1 A1, 20, A3 ; A1 + 20 = A3
 SUB .L2 B1, 15, B6 ; B1 - 15 = B6

Indirect Addressing Mode: The effective address of the operand is the contents of a register that appears in the instruction. An asterisk (*) is used as an indirection operator. Also increment (++) and decrement (- -) operators are supported.

Ex: LDW .D2 *B0, B1
 STW .D1 A1,*A2++

- **7.2 Circular Addressing Mode**

A circular buffer is a fixed number of memory locations which is circular i.e. cyclic in nature. If you increment the address of last memory location in the buffer it points to the first location. Note that in linear mode, incrementing the last memory location in the memory map causes an overflow error.

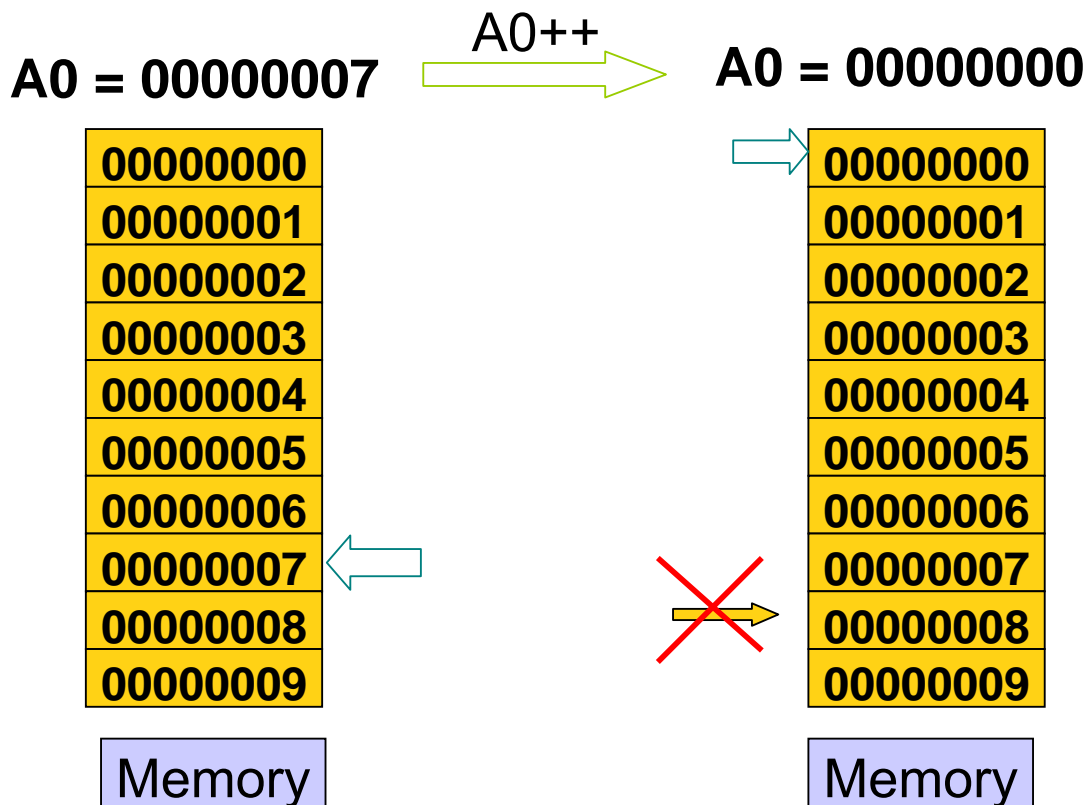


Figure 14: Circular Addressing

Consider the example shown in Figure 14. Let the block size of the circular buffer be $N = 8$. Also let $A0 = 00000007h$ initially. Assuming that $A0$ is set for circular mode in the AMR, we increment $A0$ by 1 ($A0++$). If it were linear addressing $A0$ would contain $00000008h$. But the block size is set to 8 itself (in circular mode) and since $\text{modulo}(8, N) = 0$, the $A0$ now points to zeroth memory location.

- **7.3 Addressing Mode Register**

For each of the eight registers ($A4$ – $A7$, $B4$ – $B7$) that can perform linear or circular addressing, the AMR specifies the addressing mode. A 2-bit field for each register selects the address modification mode: linear (the default) or circular mode. With circular addressing, the field also specifies which BK (block size) field to use for a circular buffer. In addition, the buffer must be aligned on a byte boundary equal to the block size. The mode select fields and block size fields are shown in Figure 15 and the mode select field encoding is shown in Figure 16 (next page).

Addressing Mode Register (AMR)

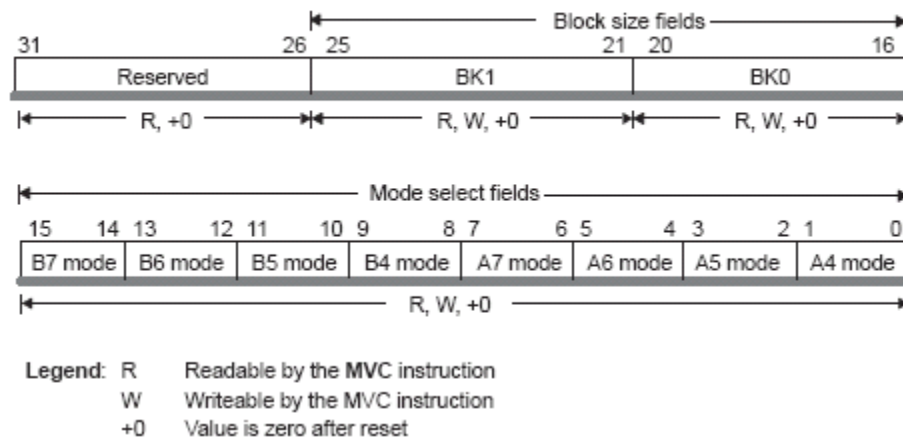


Figure 15: Addressing Mode Register

The reserved portion of AMR is always 0. The AMR is initialized to 0 at reset. The block size fields, BK0 and BK1, contain 5-bit values used in calculating block sizes for circular addressing.

$$\text{Block size (in bytes)} = 2^{(N+1)}$$

where N is the 5-bit value in BK0 or BK1

Mode	Description
00	Linear mode (default at reset)
01	Circular Addressing using BK0 field
10	Circular Addressing using BK1 field
11	Reserved

Figure 16: Mode Select Field Encoding

8. Pipelining and Parallelism

Pipelining is a technique of executing several instructions concurrently. The highlights of the C6000 pipeline are:

- The pipeline can dispatch eight parallel instructions every cycle.
- Parallel instructions proceed simultaneously through each pipeline phase.
- Serial instructions proceed through the pipeline with a fixed relative phase difference between instructions.

All instructions require the same number of pipeline phases for fetch and decode, but require a varying number of execute phases.

- **8.1 The C6000 Pipeline Phases**

The pipeline phases are divided into three stages:

- Fetch
- Decode
- Execute

All instructions in the C62x/C64x instruction set flow through the fetch, decode, and execute stages of the pipeline. The fetch stage of the pipeline has four phases for all instructions, and the decode stage has two phases for all instructions. The execute stage of the pipeline requires a varying number of phases, depending on the type of instruction. The stages of the C6000 fixed-point pipeline are shown in Figure 17.



Figure 17: Fixed-point Pipeline

The different phases in each stage of the pipelining are shown in Figure 18.

Stage	Phase	Symbol
Program Fetch	Program Address Generation	PG
	Program Address Send	PS
	Program Access Wait	PW
	Program Fetch Packet Receive	PR
Program Decode	Instruction Dispatch	DP
	Instruction Decode	DC
Program Execute	Execute Packet 1	E1
	:	:
	Execute Packet 5	E5

Figure 18: The Phases of Pipeline

During the **PG** phase, the program address is generated in the CPU. In the **PS** phase, the program address is sent to memory. In the **PW** phase, a memory read occurs. Finally, in the **PR** phase, the fetch packet is received at the CPU. In the **DP** phase of the pipeline, the fetch packets are split into execute packets. In the **DC** phase, the source registers, destination registers, and associated paths are decoded for the execution of the instructions in the functional units. The execute portion of the fixed-point pipeline is subdivided into five phases (**E1–E5**). Different types of instructions require different numbers of these phases to complete their execution.

Figure 19 shows an example of the pipeline flow of consecutive fetch packets that contain eight parallel instructions. In this case, where the pipeline is full, all instructions in a fetch packet are in parallel and split into one execute packet per fetch packet. The fetch packets flow in lockstep fashion through each phase of the pipeline. For example, examine cycle 7 in Figure 19. When the instructions from FP n reach E1, the instructions in the execute packet from FP $(n + 1)$ are being decoded. FP $(n + 2)$ is in dispatch while FPs $(n + 3)$, $(n + 4)$, $(n + 5)$, and $(n + 6)$ are each in one of four phases of program fetch.

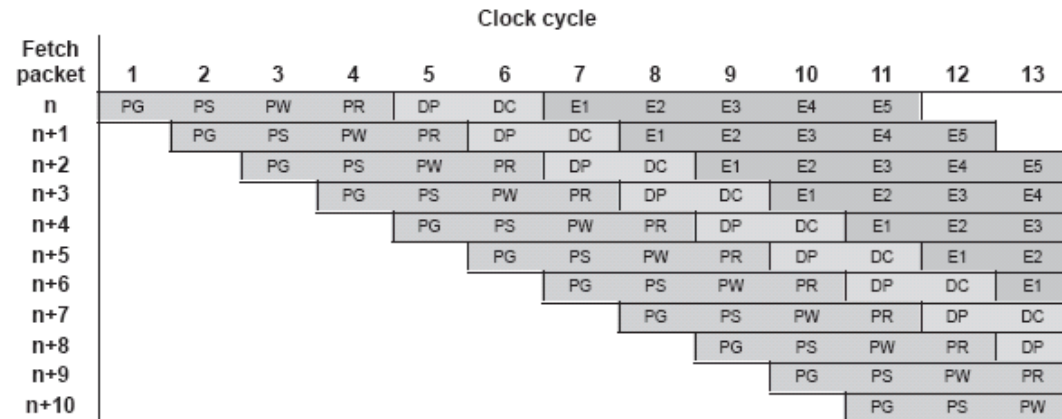


Figure 19: Pipeline Operation: One Execute Packet per Fetch Packet

• 8.2 Vector Summation: An Example

Consider a simple C function to add two N -element arrays **a** and **b**, element-by-element.

$$c[n] = \sum_{n=0}^{N-1} (a[n] + b[n])$$

```
void sum (int *a, int *b, int *c, int N)
```

```
{
    int n;
    for(n=0;n<N;n++)    c[n]=a[n]+b[n] ;
}
```

The inner loop can be implemented using the following assembly program:

```

    mvk .S1 10, A1      ; N = 10
LOOP :   ldw  .D1 *a++, A0    ; A0 = a[n]
        ldw  .D2 *b++, B0    ; B0 = b[n]
        add  .L2 A0,B0,B1    ;
        stw  .D2 B1, *c++    ; c[n] = A0+B0 , n++
        sub  .L1 A1, 1 , A1  ; N = N-1
[ A1 ]   b    LOOP          ;branch to LOOP if A1 != 0
```

We shall see next how the program operates with and without pipelining.

- **8.3 Vector Summation without Pipelining**

When pipelining is not applied, the resources (i.e. functional units) are utilized as shown below (Figure 20):

Cycle	.S1	.S2	.D1	.D2	.L1	.L2
1	mvk					
2			ldw	ldw		
3						add
4				stw	sub	

Figure 20: Resource Utilization

The first clock cycle simply loads the constant $N=10$. This is executed only once. The next three clock cycles keep repeating N times. In the second clock cycle only S1 and S2 are used. The remaining units are idle. Similarly in the third and fourth clock cycles, not all units are used. This leads to ineffective utilization of the functional units. Hence pipelining is needed. The entire program thus takes 32 clock cycles.

- **8.4 Vector Summation with Pipelining**

Software pipelining is a technique used to schedule instructions from a loop so that multiple iterations of the loop execute in parallel. Pipelining can be enabled in CCStudio™ using the option `-o1`, `-o2` or `-o3` (recommended) while running the compiler `cl6x`. Figure 21 illustrates a software pipelined loop. The stages of the loop are represented by A, B, C, D, and E. In this figure, a maximum of five iterations of the loop can execute at one time. The shaded area represents the *loop kernel*. In the

loop kernel, all five stages execute in parallel. The area above the kernel is known as the *pipelined loop prolog*, and the area below the kernel is known as *the pipelined loop epilog*.

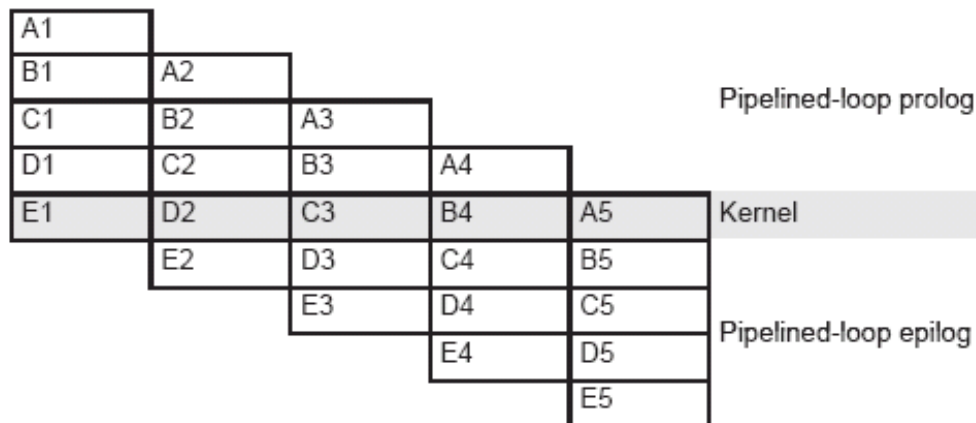


Figure 21: Software-pipelined loop

When pipelining is enabled, instead of performing one summation in one execution of the loop, the compiler *unrolls* the loop so that two summations are performed in one execution of the loop. The equivalent C function as seen by the compiler is:

```
void sum (int *a, int *b, int *c, int N)
{
    int n;
    for(n=0;n<N;n+=2)
        { c[n] = a[n] + b[n] ;
          c[n+1] = a[n+1] + b[n+1];
        }
}
```

The pipelined loop kernel is shown below (as generated by the cl6x compiler).

```
LOOP:          ; <PIPED LOOP KERNEL>
               ADD .L1X B7, A3, A3
|| [ B0]      B    .S1   L2
||           LDH  .D1T1 *++A4(4),A3
||           LDH  .D2T2 *++B4(4),B7
|| [!A1]     STH  .D1T1  A3,*++A0(4)
||           ADD  .L2X   B6,A5,B6
||           LDH  .D2T2 *+B4(2),B6
|| [ A1]     SUB  .L1    A1,1,A1
```

```
|| [!A1]   STH .D2T2 B6,*++B5(4)  
|| [ B0]   SUB .L2   B0,1,B0  
||        LDH .D1T1  *+A4(2),A5
```

Instructions marked with double bars (||) execute simultaneously in a single clock cycle. As a result, the entire program takes only 14 clock cycles now. Previously it was 32 cycles.

9. The Memory Map of C6000

The memory of any C6000 device in general consists of the following divisions (Figure 22):

Address	Memory Type
0x00000000	Internal Memory
0x00030000	Reserved Space or Peripheral Regs
0x80000000	EMIF CE0
0x90000000	EMIF CE1
0xA0000000	EMIF CE2
0xB0000000	EMIF CE3

Figure 22: Memory Map

Its features can be summarized as follows:

- Internal Memory of up to 2 MB with 1 MB being commonly found.
- External Memory Interface (EMIF) can support up to 24 MB
- External interface can be SDRAM, synchronous burst RAM
- Two-level (L1 and L2) cache up to 512 KB
- Separate cache for PMEM (program memory) and DMEM (data memory)

The actual memory map depends from device to device and cannot be generalized.

10. The Peripherals of C6000

The functions of various peripherals supported by the C6000 are:

(a) DMA (Direct Memory Access) Controller transfers data between address ranges in the memory map without intervention by the CPU. The DMA controller has four programmable channels and a fifth auxiliary channel.

(b) EDMA (Enhanced DMA) Controller performs the same functions as the DMA controller. The EDMA has 16 programmable channels, as well as a RAM space to hold multiple configurations for future transfers.

(c) HPI (Host Port Interface) is a parallel port through which a host processor can directly access the CPU's memory space. The host device has ease of access because it is the master of the interface. The host and the CPU can exchange information via internal or external memory. In addition, the host has direct access to memory-mapped peripherals.

(d) Expansion bus is a replacement for the HPI, as well as an expansion of the EMIF. The expansion provides two distinct areas of functionality (host port and I/O port) which can co-exist in a system. The host port of the expansion bus can operate in either asynchronous slave mode, similar to the HPI, or in synchronous master/slave mode. This allows the device to interface to a variety of host bus protocols. Synchronous FIFOs and asynchronous peripheral I/O devices may interface to the expansion bus.

(e) McBSP (Multi-channel Buffered Serial Port) is based on the standard serial port interface found on the TMS320C2000 and C5000 platform devices. In addition, the port can buffer serial samples in memory automatically with the aid of the

DMA/EDMA controller. It also has multi-channel capability compatible with the T1, E1, SCSA, and MVIP networking standards.

(f) Timers in the C6000 devices are two 32-bit general-purpose timers used for these functions:

- Time events
- Count events
- Generate pulses
- Interrupt the CPU
- Send synchronization events to the DMA/EDMA controller.

(g) Power-down logic allows reduced clocking to reduce power consumption. Most of the operating power of CMOS logic dissipates during circuit switching from one logic state to another. By preventing some or all of the chip's logic from switching, you can realize significant power savings without losing any data or operational context.

11. The DaVinci™ Technology

The Texas Instruments DaVinci™ Technology combines TI's offering of DSP and tools for developing a broad spectrum of optimized digital video end equipments.

- **11.1 The Origin of the DaVinci™ Effect**

A typical multimedia system such as a digital video recorder or digital camera can be split roughly into two pieces: control and media. The *control* portion handles tasks such as memory card or hard disk access, user interface, and networking, while the *media* portion covers tasks such as encoding and decoding of audio and video. A general-purpose processor performs well in control tasks, but all but the fastest of these processors are not sufficiently powerful for intensive media-related tasks such as real-time, high-quality video encoding. A DSP, on the other hand, is superb at the repetitive, easily parallelizable media-related tasks, but usually performs poorly in control-related jobs.

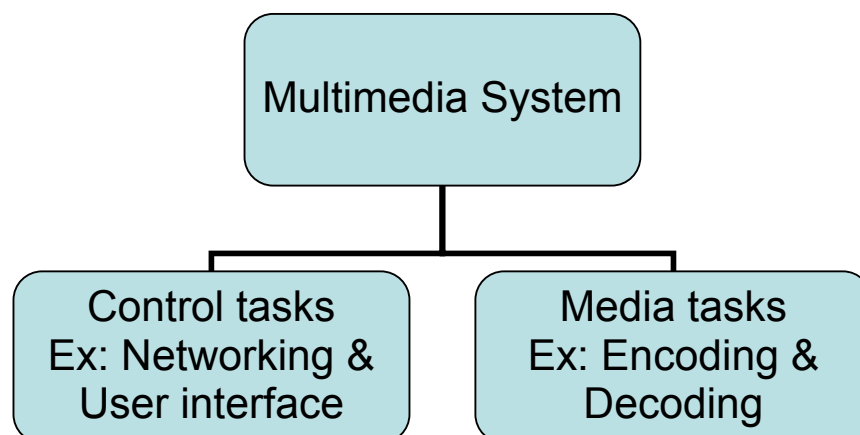


Figure 23: Multimedia System

The idea behind DaVinci is that by using both a general-purpose processor and a DSP, the control and media portions can both be executed by processors that excel at their respective tasks. The integration of these two components into one chip simplifies the system design and allows for more efficient communication between the two components.

- **11.2 Existing Systems**

DaVinci Technology has grown to support DSP only processors including TMS320DM6431, TMS320DM6433, TMS320DM6435 and TMS320DM6437.

Models

DM6443 = ARM9 + Texas Instruments TMS320C64x+ DSP + DaVinci Video (Decode) → Video Accelerator and Networking for display

DM6446 = ARM9 + Texas Instruments TMS320C64x+ DSP + DaVinci Video (Encode and Decode) → Video Accelerator and Networking for capture and display

- **11.3 Peripherals and Operating System**

The DaVinci includes a number of on-chip peripherals. These include:

- Support for memory cards such as CompactFlash, SD Card and MMC
- ATA interface
- CCD Controller for digital camera/camcorder applications
- Connectivity, including USB 2.0 Host and Client modes, VLYNQ (interface for FPGA, Wireless LAN, PCI), EMAC (Ethernet MAC) with MDIO
- Enhanced DMA
- Interrupt controller
- Digital LCD controller
- Serial interfaces, including SPI, I²C, and I²S, UART
- Histogram, autofocus, autoexposure, and auto-white-balance (H3A) acceleration
- Image resize acceleration
- A/D and D/A converters for analog video input and output

The DSP in the DaVinci generally runs TI's **DSP/BIOS** RTOS. DSP/BIOS Link drivers run on both the ARM processor and the DSP to provide communication between the two. A number of operating systems support DaVinci and the DSP/BIOS Link drivers:

- Green Hills Software INTEGRITY RTOS
- Montavista Linux
- QNX Neutrino
- Windows CE
- DaVinci Linux OS is currently (2007) under development.
- **Blackhawk XDS560™** is a real-time debugger released on March 19, 2007. It captures even the toughest real-time hardware bugs.

- **11.4 Applications**

High-end applications include:

- Stream live-video on to a portable, handheld device
- An on-board intelligence system in your car can record obstructions in front of the windscreen.
- Surveillance videos directly captured by a TV and transferred to a computer/controller

Other everyday applications which have a potential to exploit DaVinci are:

- 1200- to 56Kbps modems
- Digital Subscriber Loop
- X.25 packet switching
- Image Compression
- Homomorphic Processing
- Robotic Vision
- Pattern Recognition

12. Conclusions

- “Necessity is the mother of all inventions”. Hence the origin and four generations of DSPs were first considered.
- The failure of Von Neumann architecture led to the Harvard architecture, which was further developed into VLIW architecture.
- The data path and control registers are a dominating feature of the C6000’s architecture. Each instruction is mapped uniquely to every functional unit.
- The addressing modes, especially circular addressing, is extremely useful in implementing FIR filters and circular convolution.
- Pipelining is the watchword when it comes to parallel execution. Loop unrolling nearly doubles the execution speed.
- The DaVinci™ is a state-of-the-art technology that is designed with video and multimedia applications in mind.

13. References

Technical Documentation:

- *TMS320C6000 Programmer's Guide* (literature number 198G), Texas Instruments, August 2003. Describes ways to optimize C and assembly code for the TMS320C6000 DSPs and includes application program examples.
- *TMS320C6000 CPU and Instruction Set Reference Guide* (literature number SPRU189F), Texas Instruments, October 2004. Describes the C6000 CPU architecture, instruction set, pipeline, and interrupts for these digital signal processors.
- *TMS320C6000 Optimizing C Compiler* (literature number SPRU187L), Texas Instruments, May 2004. Describes the C6000 C compiler and the assembly optimizer. This C compiler accepts ANSI C source code and produces assembly language source code. The assembly optimizer helps to optimize assembly code.
- Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1999. A non-mathematical textbook that teaches DSP intuitively.

Newsletters:

- XML-RSS DaVinci™ feed from Texas Instruments.

Web Resources:

- http://en.wikipedia.org/wiki/Texas_Instruments_DaVinci: Describes the DaVinci™ technology (Chapter 11).
- http://en.wikipedia.org/wiki/Digital_signal_processor: History of DSPs (Chapter 2)
- <http://www.thedavincieffect.com>: Official website of DaVinci™ technology.