M.Tech. credit seminar report,
Electronic Systems Group,
EE Dept, IIT Bombay,
Submitted: November 2002

Evolution of DSPs

Author: Kartik Kariya (Roll No. 02307923)
Supervisor: Prof. Vikram M. Gadre, Associate Professor, IIT Bombay.

Abstract

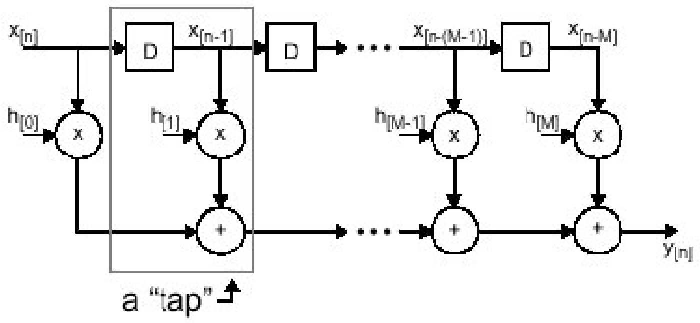
DSP algorithms and necessity to do complex computation with strict time and accuracy constraints pioneered to the development in the architecture of digital signal processors. Different low and high end DSPs, which satisfies cost and speed constrains of various application are available in the market. This report reviews the available DSP processor architecture and their features. There is growing need to design customized DSP core, which suits to particular application. This fact is explained with the case study of DSP architecture designed for handheld devices such as mobile phones.

1 Introduction:

DSP has become a key component in many consumer, communication, medical, and industrial products. The number and variety of products that include some form of digital signal processing has grown dramatically over the years. These products use a variety of hardware approaches to implement DSP, ranging from the use of microprocessors to field-programmable gate arrays (FPGAs) to custom integrated circuits (ICs).
In comparison to microprocessors, DSP processors often have an advantage in terms of speed, cost, and energy efficiency. In this report, we trace the evolution of DSP processors, from early architectures to current state-of-the-art devices. Some of the key differences among architectures, and their strengths and weaknesses are highlighted. Lastly, the case study of SPXK5, used in handheld applications is presented to explain how customized hardware are being used to satisfy need of particular application.

2 Differences in DSP and General-Purpose Processors Architecture:

From the beginning, DSP algorithms have driven DSP processor architectures. For nearly every feature found in a DSP processor, there are associated DSP algorithms whose computation, in some way are eased by inclusion of the feature. Therefore, perhaps the best way to understand the evolution of DSP architectures is to look at typical DSP algorithms and identify how their computational requirements have influenced the architectures of DSP processors [1]. As a case study, we will consider one of the most common signal processing algorithms, the FIR filter.



2.1 Fast Multipliers

The FIR filter is mathematically expressed as, where is a vector of input data, and is a vector of filter coefficients. For each "tap" of the filter, a data sample is multiplied by a filter coefficient, with the result added to a running sum for all of the taps. Hence, the main component of the FIR filter algorithm is "multiply and add". These operations are not unique to the FIR filter algorithm, in fact, multiplication (often combined with accumulation of products) is one of the most common operations performed in signal processing convolution, IIR filtering, and Fourier transforms also all involve heavy use of multiply accumulate operations.
Originally, microprocessors implemented multiplications by a series of shift and add operations, each of which consumed one or more clock cycles. In 1982, however, Texas Instruments (TI) introduced the first commercially successful "DSP processor," the TMS32010, which incorporated specialized hardware to enable it to compute a multiplication in a single clock cycle. As might be expected, faster multiplication hardware yields faster performance in many DSP algorithms, and for this reason all DSP processors include at least one dedicated single cycle multiplier or combined multiply-accumulate (MAC) unit.

2.2 Multiple Execution Units

DSP applications typically have very high computational requirements in comparison to other types of computing tasks, since they often must execute DSP algorithms (such as FIR filtering) in real time on lengthy segments of signals sampled at 10-100 KHz or higher. Hence, DSP processors often include several independent execution units that are capable of operating in parallel. For example, in addition to the MAC unit; they typically contain an arithmetic-logic unit (ALU) and a shifter.

2.3 Efficient Memory Accesses

Executing a MAC in every clock cycle requires more than just a single-cycle MAC unit. It also requires the ability to fetch the MAC instruction, a data sample, and a filter coefficient from memory in a single cycle. Hence, good DSP performance requires high memory bandwidth higher than was supported on the general-purpose microprocessors, which typically contained a single bus connection to memory and could only make one access per clock cycle. To address the need for increased memory bandwidth, early DSP processors developed different memory architectures that could support multiple memory accesses per cycle. The most common approach (still commonly used) was to use two or more separate banks of memory, each of which was accessed by its own bus and could be read or written during every clock cycle. Often, instructions were stored in one memory bank, while data was stored in another. With this arrangement, the processor could fetch an instruction and a data operand in parallel in every cycle.

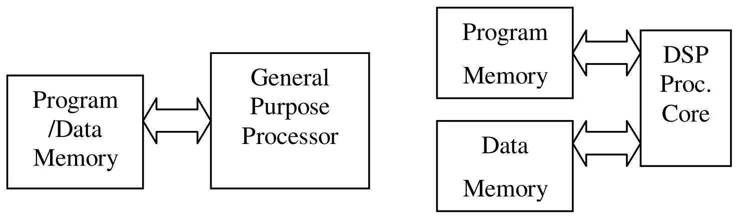


Figure 2: Difference between memory structures General Purpose Processor and DSPs

Above figure illustrates the difference in memory architectures for early general-purpose processors and DSP processors. Since many DSP algorithms (such as FIR filters) consume two data operands per instruction (e.g., a data sample and a coefficient), a further optimization commonly used is to include a small bank of RAM near the processor core that is used as an instruction cache. When a small group of instructions is executed repeatedly (i.e., in a loop), the cache is loaded with those instructions, freeing the instruction bus to be used for data fetches instead of instruction fetches—thus enabling the processor to execute a MAC in a single cycle.

High memory bandwidth requirements are often further supported via dedicated hardware for calculating memory addresses. These address generation units operate in parallel with the DSP processor's main execution units, enabling it to access data at new locations in memory (for example, stepping through a vector of coefficients) without pausing to calculate the new address.

Memory accesses in DSP algorithms tend to exhibit very predictable patterns; for example, for each sample in an FIR filter, the filter coefficients are accessed sequentially from start to finish for each sample, then accesses start over from the beginning of the coefficient vector when processing the next input sample. This is in contrast to other types of computing tasks, such as database processing, where accesses to memory are less predictable. DSP processor address generation units take advantage of this predictability by supporting specialized addressing modes that enable the processor to efficiently access data in the patterns commonly found in DSP algorithms. The most common of these modes is register-indirect addressing with post-increment, which is used to automatically increment the address pointer for algorithms where repetitive computations are performed on a series of data stored sequentially in memory. Without this feature, the programmer would need to spend instructions explicitly incrementing the address pointer. Many DSP processors also support "circular addressing," which allows the processor to access a block of data sequentially and then automatically wrap around to the beginning address. Circular addressing is also very helpful in implementing first-in, first-out buffers, commonly used for I/O and for FIR filter delay lines.

2.4 Data Format

Most DSP processors use a fixed-point numeric data type instead of the floating-point format. Though floating-point format have good numerical fidelity and virtually eliminates, numerical overflow etc, in many cases DSP processors face additional constraints i.e. they must be inexpensive and provide good energy efficiency. Fixed-point processors tend to be cheaper and less power-hungry than floating-point processors at comparable speeds, because floating-point formats require more complex hardware to implement. To ensure adequate signal quality while using fixed-point data, DSP processors typically include specialized hardware to help programmers maintain numeric fidelity throughout a series of computations. For example, most DSP processors include one or more "accumulator" registers to hold the results of summing several multiplication products. Accumulator registers are typically wider than other registers; they often provide extra bits, called "guard bits," to extend the range of values that can be represented and thus avoid overflow.

Sensitivity to cost and energy consumption also influences the data word width used in DSP processors. DSP processors tend to use the shortest data word that will provide adequate accuracy in their target applications. Most fixed-point DSP processors use 16-bit data words, because that data word width is sufficient for many DSP applications. A few fixed-point DSP processors use 20, 24, or even 32 bits to enable better accuracy in applications that are difficult to implement well with 16-bit data, such as high fidelity audio processing. In addition, DSP processors usually include good support for saturation arithmetic, rounding, and shifting, all of which are useful for maintaining numeric fidelity.

2.5 Zero-Overhead Looping

DSP algorithms typically spend the vast majority of processing time in relatively small sections of software that are executed repeatedly; i.e., in loops. Hence, most DSP processors provide special support for efficient looping.

Zero overhead Looping is implemented in a DSP as a block of logic that uses a small instruction cache and a few register to repeat a predetermined block of code (determined in advance by program) with zero overhead.

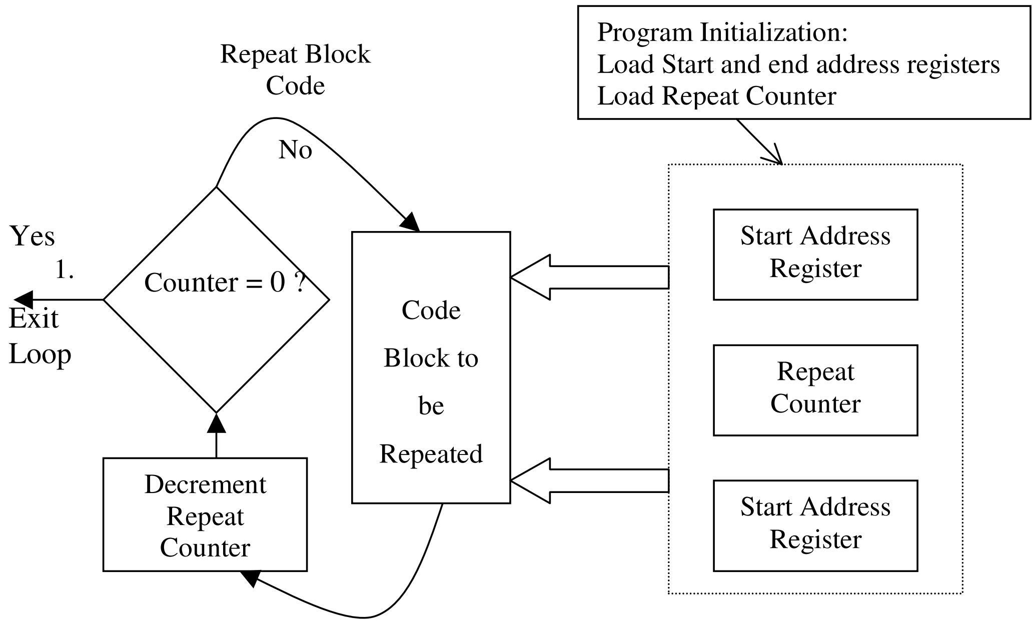


Figure 3: Zero Overhead Looping

When program reaches a block of code that will be repeated more than once, start address, and end-address register and a loop count register are loaded. The block of code indicated by the start and end address is then repeated automatically the number of times specified in the loop count register. The benefit is that instead of having to do test and branch operations repeatedly, the hardware in the hardware branching logic block (which comprises part of the Zero Overhead Looping hardware) does it all. However most disadvantage of this is whole block to be repeated must be accommodate in cache.

2.6 Streamlined I/O

To allow low-cost, high-performance input and output, most DSP processors incorporate one or more specialized serial or parallel I/O interfaces, and streamlined I/O handling mechanisms such as low-overhead interrupts and direct memory access (DMA) to allow data transfers to proceed with little or no intervention from the processor's computational units.

2.7 Specialized Instruction Sets

DSP processor instruction sets are designed with following two goals in mind. 1. To make maximum use of the processor's underlying hardware, thus increasing efficiency. 2. To minimize the amount of memory space required to store DSP programs, since DSP applications are often quite cost-sensitive and the cost of memory contributes substantially to overall chip and/or system cost.

To accomplish the first goal, conventional DSP processor instruction sets generally allow the programmer to specify several parallel operations in a single instruction, typically including one or two data fetches from memory (along with address pointer updates) in parallel with the main arithmetic operation. To reduce the number of bits required to encode instructions, DSP processors often offer fewer registers than other types of processors, and may use mode bits to control some features of processor operation (for example, rounding or saturation) rather than encoding this information as part of the instructions. The overall result of these features is that conventional DSP processors tend to have highly specialized, complicated, and irregular instruction sets. This characteristic has come to be viewed as a significant drawback of these processors, because it complicates the task of creating efficient assembly language software.

3 The DSP Landscapes

In this section we look at the DSP from historical perspective and see how DSP's got evolved [3].

3.1 Conventional DSP Processors

The performance and price range among DSP processors is very wide.
Some of the common features of these early (1980s) DSPs are:
1. Issue and execute one instruction per clock cycle, and use the complex, multi-operation instructions.
2. Typically include a single multiplier or MAC unit and an ALU, but few additional execution units.
Few examples of these groups are Analog Devices' ADSP-21xx family, Texas Instruments' TMS320C2xx family, and Motorola's DSP560xx family. These processors generally operate at around 20-50 MHz, and provide good DSP performance while maintaining very modest power consumption and memory usage. They are typically used in consumer and telecommunications products that have modest DSP performance requirements and stringent cost and/or energy consumption constraints, like disk drives and digital telephone answering machines.
The improvement in the performance of above processors was achieved through combination of increased clock speeds and somewhat more sophisticated architectures. DSP processors like the Motorola DSP563xx and Texas Instruments TMS320C54x operate at 100-150 MHz and often include a modest amount of additional hardware, such as a barrel shifter or instruction cache, to improve performance in common DSP algorithms. Processors in this class also tend to have deeper pipelines than their lower-performance cousins. However midrange DSP processors are more similar to their predecessors than they are different. By using this approach, midrange DSP processors are able to achieve noticeably better performance while keeping energy and power consumption low. Processors in this performance range are typically used in wireless telecommunications applications and high-speed modems, which have relatively high computational demands but often require low power consumption.

3.2 Enhanced-Conventional DSP Processors

The purpose of these DSPs is to capitalize the performance with addition of few parallel execution units, typically a second multiplier and adder. These hardware enhancements are combined with an extended instruction set that takes advantage of the additional hardware by allowing more operations to be encoded in a single instruction and executed in parallel. With this increased parallelism, enhanced-conventional DSP processors can execute significantly more work per clock cycle e.g. two MACs per cycle instead of one. The Lucent Technologies DSP16xxx is one such example of DSPs.

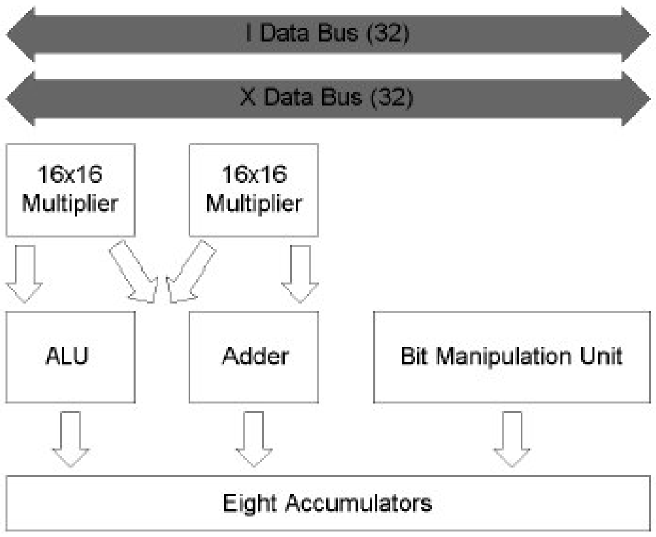


Figure 4: Lucent Tech's DSP 16xxx

Enhanced-conventional DSP processors are typically characterized by:

1. Wider data buses to allow them to retrieve more data words per clock cycle in order to keep the additional execution units fed with data.
2. Wider instruction words to accommodate specification of additional parallel operations within a single instruction.

Increases in cost and power consumption due to the additional hardware and architectural complexity are largely offset by increased performance.

3.3 Multi-Issue Architectures

Enhanced-conventional DSP processors provide improved performance by allowing more operations to be encoded in every instruction, however they are difficult to program in the assembly language and are unfriendly compiler targets.

With the goals of achieving high performance and creating architecture that lends itself to the use of compilers, DSP processors with multi-issue approach were designed. In contrast to conventional and enhanced-conventional processors, multi-issue processors use very simple instructions that typically encode a single operation. These processors achieve a high level of parallelism by issuing and executing instructions in parallel groups rather than one at a time. Using simple instructions simplifies instruction decoding and execution, allowing multi-issue processors to execute at higher clock rates than conventional or enhanced conventional DSP processors.

TI was the first DSP processor vendor to use this approach in a commercial DSP processor. TI's multi-issue TMS320C62xx, introduced in 1996, was dramatically faster than any other DSP processor available at the time. The two classes of architectures that execute multiple instructions in parallel are referred to as VLIW (very long instruction word) and superscalar. These architectures are quite similar, differing mainly in how instructions are grouped for parallel execution.

3.3.1 VLIW and Superscalar Architecture

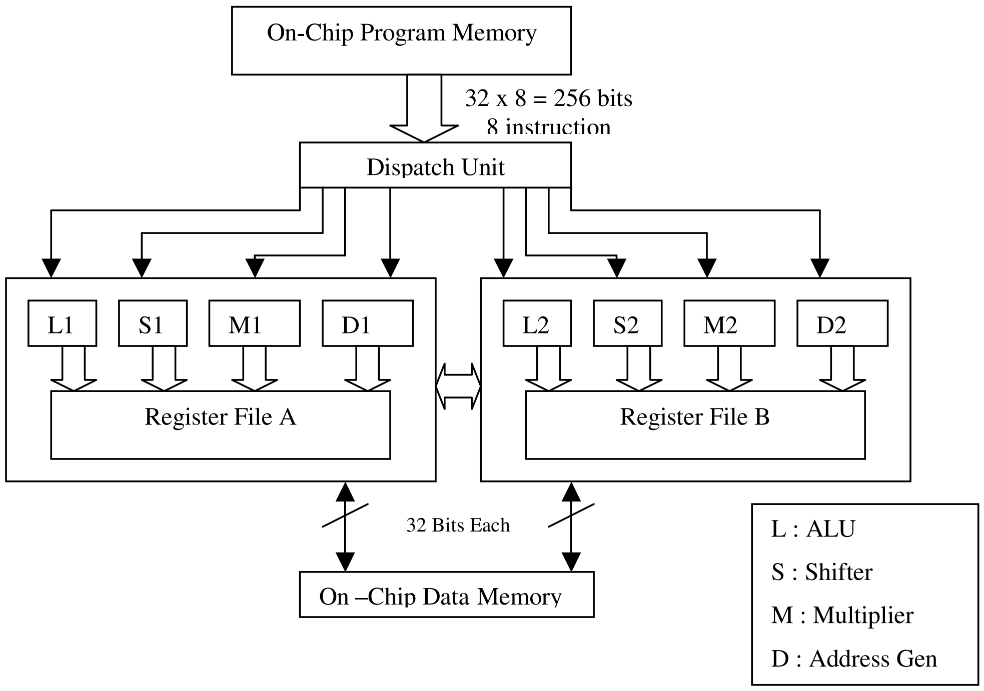


Figure 5: TMS320C6xx Execution Unit [6]

VLIW and superscalar architectures provide many execution units, more than that found in conventional or even enhanced conventional DSPs, each of which executes its own instruction. Figure 5 illustrates the execution units and buses of the TMS320C62xx, which contains eight independent execution units. VLIW DSP processors typically issue a maximum of between four and eight instructions per clock cycle, which are fetched and issued as part of one long super-instruction
In VLIW architecture, the assembly language programmer (or code-generation tool) specifies which instructions will be executed in parallel. Hence, instructions are grouped at the time the program is assembled, and the grouping does not change during program execution. Superscalar processors, in contrast, contain specialized hardware that determines which instructions will be executed in parallel based on data dependencies and resource contention, shifting the burden of scheduling parallel instructions from the programmer to the processor.
Superscalars may group the same set of instructions differently at different times in the program's execution e.g. it may group instructions one way the first time it executes a loop, then group them differently for subsequent iterations. This makes difficult for the programmer to predict exactly how long a given segment of software will take to execute. . The execution time may vary based on the particular data accessed, whether the processor is executing a loop for the first time or the third, or whether it has just finished processing an interrupt. The difference in the way these two types of architectures schedule instructions for parallel execution is important in the context of using them in real-time DSP applications because in this case execution time must be predictable.
VLIW processors use wider instruction words than conventional DSP processors. When a processor issues multiple instructions per cycle, it must be able to determine which execution unit will process each instruction. Traditionally, VLIW processors have used the position of each instruction within the super-instruction to determine to where the instruction will be routed. Some recent VLIW architectures do not use positional super-instructions, however, and instead include routing information within each sub-instruction. To support execution of multiple parallel instructions, VLIW and superscalar processors must have sufficient instruction decoders, buses, registers, and memory bandwidth. VLIW processors typically use either wide buses or a large number of buses to access data memory and keep the multiple execution units fed with data.

VLIW and superscalar processors often suffer from high-energy consumption relative to conventional DSP processors. In general, multi-issue processors are designed with an emphasis on increased speed rather than energy efficiency. Hence VLIW and superscalar processors have mainly targeted for applications, which have very demanding computational requirements but are not very sensitive to cost or energy efficiency. For example, a VLIW processor might be used in a cellular base station, but not in a portable cellular phone.

3.3.2 SIMD

SIMD, or single-instruction, multiple-data, is an architectural technique that can be used within any of the classes of architectures described so far. SIMD improves performance on some algorithms by allowing the processor to execute multiple instances of the same operation in parallel using different data. For example, a SIMD multiplication instruction could perform two or more multiplications on different sets of input operands in parallel in a single clock cycle. This technique can greatly increase the rate of computation for some vector operations that are heavily used in multimedia and signal processing applications.

On DSP processors with SIMD capabilities, the hardware that supports SIMD operations varies widely. Some of the approaches implemented are discussed here.

1. Analog Devices, for example, modified their basic conventional floating-point DSP architecture, the ADSP-2106x, by adding a second set of execution units that exactly duplicate the original set. The new architecture is called the ADSP-2116x. Each set of execution units in the ADSP-2116x includes a MAC unit, ALU, and shifter, and each has its own set of operand registers. The augmented architecture can issue a single instruction and execute it in parallel in both data paths using different data and effectively doubling its performance in some algorithms.
2. On the other hand instead of having multiple sets of the same execution units, some DSP processors can logically split their execution units (e.g., ALUs or MAC units) into multiple sub-units that process narrower operands. These processors treat operands in long (e.g., 32-bit) registers as multiple short operands (e.g., as two 16-bit operands or four 8-bit operands). This approach is been use in Analog Devices' TigerSHARC [4] processor.

To explain the concept of splitting the execution units, take example of FIR filter. FIR filter with length M can be expressed as

$$y(n) = \sum_{k=0}^{M-1} c(k)x(n-k)$$

To implement this filter with four parallel subunits simultaneously, equation can be expressed as

$$y_i(n) = \sum_{k'=0}^{\frac{M-1}{4}} c(4k'+1)x(n-4k'-1)$$

$$\text{and} \qquad y(n) = \sum_{i=0}^{3} y_i(n)$$

Each of the four sequences y_i (n) maps directly to the four sub-word MAC units.

4 Alternatives to DSP Processors

4.1 High-Performance CPUs

Many high-end CPUs, such as Pentiums and PowerPCs, have been enhanced to increase the speed of computations associated with signal processing tasks. The most common modification is the addition of SIMD-based instruction-set extensions, such as MMX for the Pentium, and AltiVec for the PowerPC. This approach is a good one for CPUs, which typically have wide resources (buses, registers, ALUs).

General-purpose processors are often able to achieve performance on DSP algorithms that is better than that of even the fastest DSP processors. This surprising result is partly due to the effectiveness of SIMD, but also because many CPUs operate at extremely high clock speeds in comparison to DSP processors; high-performance CPUs typically operate at upwards of 1- 1.5GHz, while the fastest DSP processors are in the 200-250 MHz range.
However following are the reasons why DSP processors are preferred over these high performance processors for many applications.

1. DSP processors provide better mixture of performance, power consumption, and price.
2. As discussed before for real-time applications, the superscalar architectures and dynamic features common among high-performance CPUs can be problematic for execution time estimation.
3. Another key advantage is the availability of DSP-specific development tools and off-the-shelf DSP software components.

4.2 DSP/Microcontroller Hybrids

Many applications require a mixture of control-oriented software and DSP software. An example is of the digital cellular phone, which must implement both supervisory tasks and voice-processing tasks. In general, microcontrollers provide good performance in controller tasks and poor performance in DSP tasks, and DSP processors have the opposite characteristics. Hence, until recently, combination control/signal processing applications were typically implemented using two separate processors: a microcontroller and a DSP processor. In recent years, however, a number of microcontroller vendors have begun to offer DSP-enhanced versions of their microcontrollers as an alternative to the dual-processor solution. Using a single processor to implement both types of software is attractive, because it can potentially:

1. Simplify the design task
2. Save circuit board space
3. Reduce total power consumption
4. Reduce overall system cost

Microcontroller vendors such as Hitachi, ARM, and Lexra have taken a number of different approaches to adding DSP functionality to existing microprocessor designs, borrowing and adapting the architectural features common among DSP processors. Many of these hybrid processors achieve signal-processing performance that is comparable to that of low-cost or mid-range DSP processors while allowing re-use of software written for the original microcontroller architecture.

5 Case Study: VLIW based Processor (SPXK5) for Mobile Applications

The fast bit-rate, multimedia applications like video codecs, audio codecs, echo cancellers speech recognition systems executing simultaneously, are some of the key requirements of modern mobile applications which naturally demands higher processing power [2]. As discussed before VLIW architecture are particularly suitable for such applications because they enable the development of high-level language compilers that generates efficient codes and in turns reduces the development time. However almost all VLIW-based DSPs are developed for high-end applications and consume too much power to use in the handheld devices. To address these issues recently NEC Corporation come up with DSP core SPXK5 useful particularly for 3G mobile devices. In incorporates customized VLIW approach as well as SIMD features to give better performance. Here we take the review of the architecture of it and then study implementation of some of the DSP algorithms.

5.1 Architecture of SPXK5

The SPXK5 is a 16-bit general purpose DSP core based on the NEC µPD7701x architecture. Its low-power consumption (0.15 mW / MIPS at 1.5V) and high performance makes it suitable for handheld devices:
As shown in the block diagram below it consists of control blocks, buses, registers and functional units. The functional units consist of

1. Two multiply-accumulate (MAC) units for 16-bit by 16-bit multiplications and 40/16-bit accumulations
2. Two arithmetic units (ALU) for addition, subtraction, shift and logical operations.

3.Two data address units (DAU) for load and store.
4.System control unit (SCU) for branch, zero overhead looping, and conditional execution.

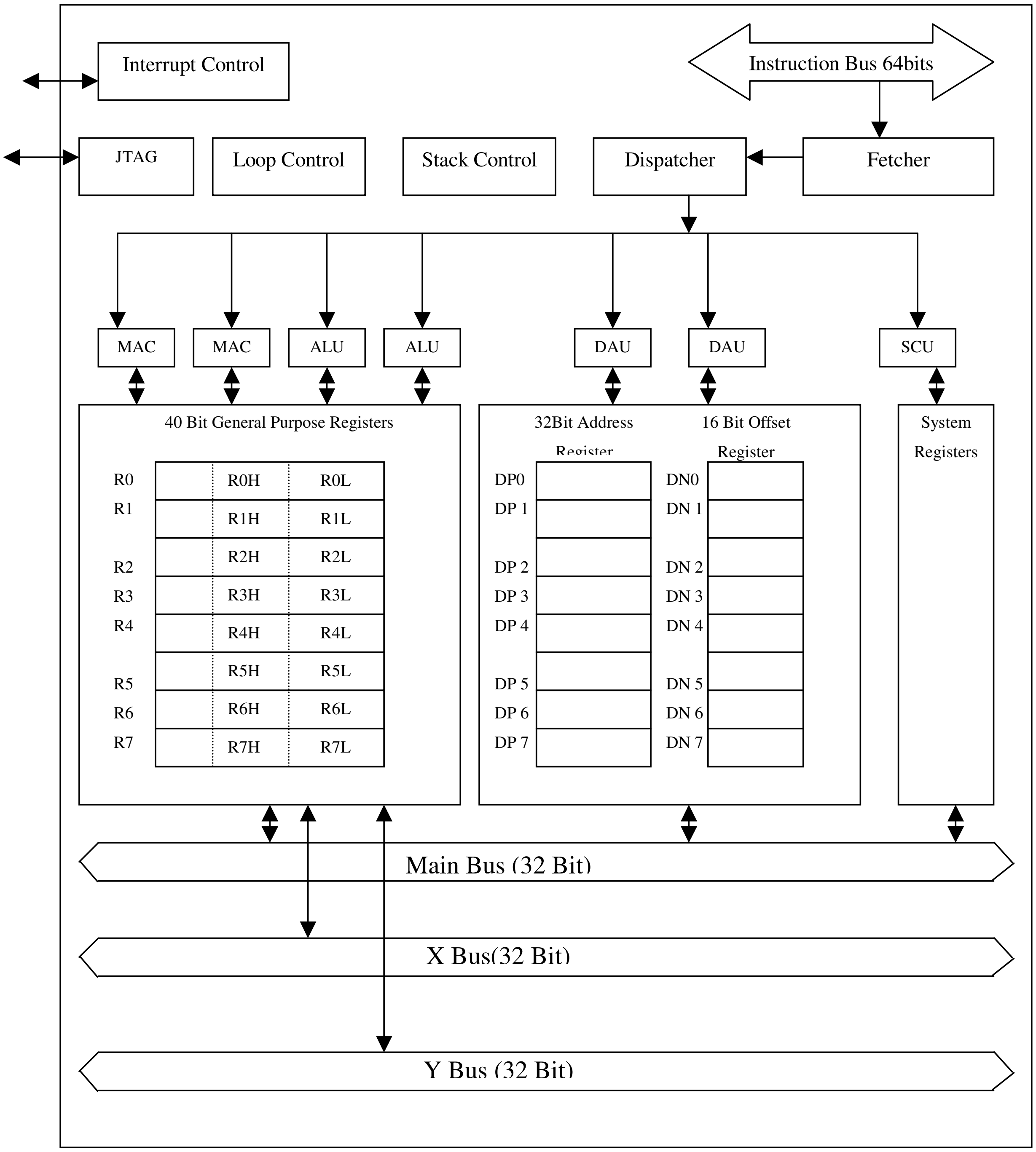


Figure 6: The SPXK5 block diagram showing functional units (Ref: [5])

5.2 Chip performance and Features

Mobile applications need processors, which deliver required functionality by consuming as much small power as possible. With the advent of 2.5G, and 3G devises, processors must provide architecture that helps to perform necessary algorithm efficiently. Following are few features that make it suitable for mobile applications. [5]:

1.Operational Frequency 250 MH_z; Average power Consumption: 0.15 mW/MIPS at 1.5 V and 0.05 mW/MIPS at 0.9 V.
2.SPXK5 is designed so that maximum four of seven functional units work simultaneously. Although it is desirable to run all seven blocks simultaneously it demands higher instruction width and in turn increasing the power dissipation beyond limit.
3.16 Kbyte instruction cache. Six-stage deep pipeline: Instruction fetch, dispatch queue, decode, DP register update, Execution phase I and II. However it relies on software for optimized instruction scheduling so that to have lesser hardware and consume less power.
4.Instruction are either 16 or 32 bits long while instruction packet size can vary from 16 to 64 bits. This is in contrast to the conventional VLIW architecture. The variable length instruction packet allows the elimination of redundant NOP operations used to fill packets and in turn produces higher code density.
5. It also contains eight special SIMD instructions (PADD, PSUB, PSHIFT, PADDABS, PACKV etc) to take advantage of data-level parallelism. These instructions are helpful to implement DSP algorithms such as video encoding/decoding, viterbi decoding, FFT etc. These instructions are explained in detail in next subtopic where implementation of some of the algorithms is explained.

5.3 Application Benchmarks

In this section some of the processing results for SPXK5 are compared with different low power DSPs, which employ different architectural approaches. Ref [5].

BenchMarks	Cycle Counts			
	PD77210	SPXK5	TI C55x	Intel ADI MSA
FIR filtering (N Samples, T taps)	NT	NT / 2	NT / 2	NT / 2
IIR filtering (N Samples, B biquads)	5NB	3NB	3NB	2.5NB
LMS adaptive filtering (N Samples, T Taps)	3NT	NT	2NT	1.5NT
256 point complex FFT	9196	2944	4786	3176

Table: Benchmark Comparison Ref [5].

5.4 Implementation of DSP Algorithm

In this section we consider implementation of implementation of MPEG – 4 video codec system components which is key application in the mobile terminals. The main components in the MPEG-4 video codec are Motion estimation (ME), Discrete cosine transform (DCT), inverse DCT (IDCT), Motion compensation (MC), quantization, variable length coding and decoding etc. Here we look in detail the motion estimation component.

5.4.1 Motion Estimation for video codec:

Motion estimation is most heavily demanding operations in video encoders. It uses block-matching techniques to search a motion vector for a desired macroblock. In a block-matching algorithm, both the mean absolute error (MAE) and mean square error (MSE) between a current macroblock and a reference macroblock are feasible as block-matching criteria.
The MAE and MSE are defined as

$$MAE = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left|a_{mn} - b_{mn}\right|$$

$$MSE = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(a_{mn} - b_{mn}\right)^2$$

where, a_{mn} and b_{mn} are pixel values in respectively current and reference macroblock.
Implementation diagram: Following diagram shows implementation of MEA.

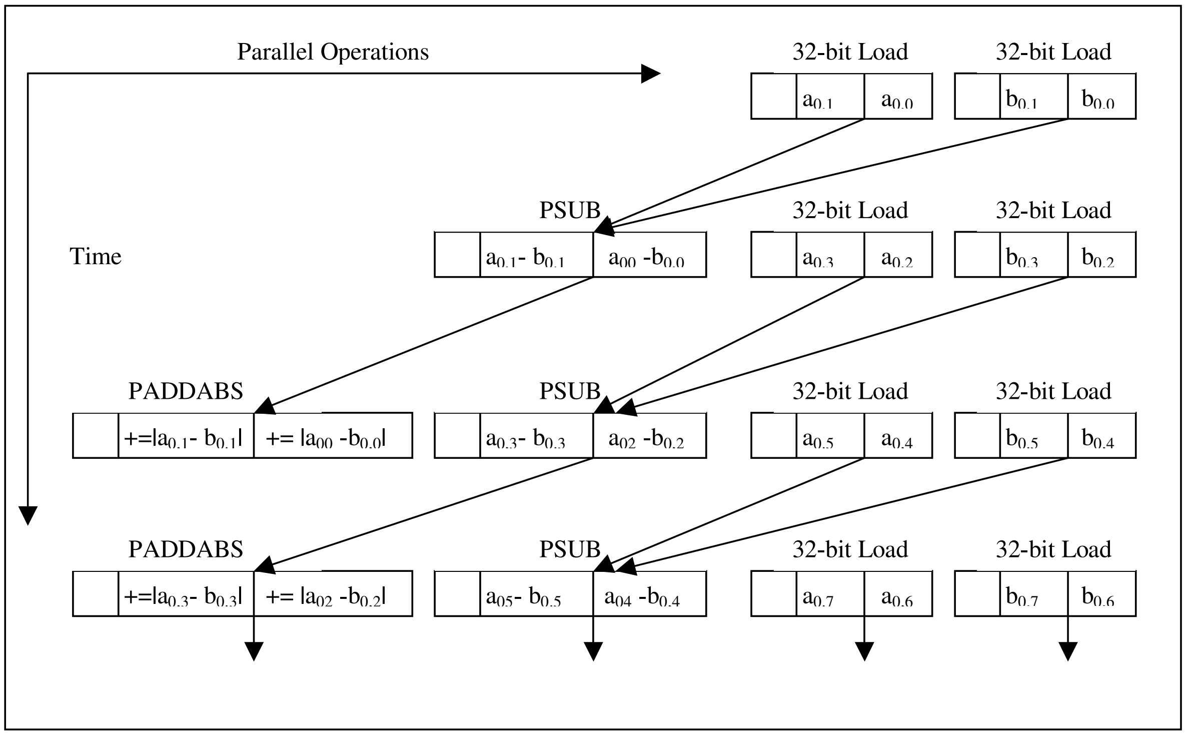


Figure7: Implementation of mean absolute error

As shown in the above flow diagram, to calculate mean absolute error we make use of two SIMD instructions, PSUB and PADDABS for doing two 16–bit subtraction simultaneously. Data is saved in registers with a pipeline structure as shown. MEA for 1 pixel can be calculated in 0.5 cycles and as the pixel size increase number of cycles will increase.

Other blocks of MPEG codec can also be implemented using the same approach. For MPEG-4 video (352 x 288 pixels, 15 frames), it has been observed that encoding and decoding both can be observed at around 105 MHz.

6 Conclusions

DSP processor architectures are evolving to meet the changing needs of DSP applications. There are many diverse architecture are designed to address different issues and goals. The forces driving the evolution of DSP processors today include the persistent push for increased speed, decreased energy consumption, decreased memory usage, and decreased cost, to better meet the needs of new and existing applications. As shown with example of DSP processors for low power Mobile application customized products are coming up to address the new generation applications.

The emphasis is for architectures that facilitate development of more efficient compilers, allowing DSP applications to be written primarily in high-level languages. This has become a focal point in the design of new DSP processors, because DSP applications are growing too large to comfortably implement and maintain in assembly language. As the needs of DSP applications continue to change, we expect to see a continuing evolution in DSP processors.

Reference:

[1] A.L. Lilein (TI) "Digital Signal Processors vs. Universal Microprocessors" ESIEE, Paris September 1996.
[2] Ichiro Kuoda, "Multimedia Processors", Proc. IEEE, Vol. 86, No.6, pp 1203-1221, 1998
[3] J. Eyre and J. Bier, "The evolution of DSP Processors", IEEE Signal Processing Magazine, vol17, no.2, pp43-51, Mar.2000
[4] J. Fridman, "Applying TigerSHARC Architecture: Sub-World Parallelism in Digital Signal Processing", IEEE Signal Processing Magazine, pp27-35, March 2000
[4] R. J. Higgins, "Digital Signal Processing in VLSI", Georgia Institute of Technology.
[5] Takahiro Kumara, Masao Ikekawa, Makoto Yoshida, and Ichiro Kuroda, "VLIW DSP for Mobile Applications" IEEE Signal Processing Magazine, pp10-21, July 2002
[6] TMS32060x Technical overview, Texas Instruments Inc.
[7] W. Strauss, "Digital Signal Processing Semiconductor Industry Technology Driver", IEEE Signal Processing Magazine, vol17, no.2, pp52-56, Mar.2000