Low Power DSP's for Wireless Communications

Ingrid Verbauwhede
UCLA, EE Dept.
7440B Boelter Hall
Box 951594
Los Angeles CA 90095-1594
Ingrid@ee.ucla.edu

Chris Nicol
Bell Laboratories
Lucent Technologies
North Ryde, 2113,
Australia
chrisn@lucent.com

ABSTRACT

Wireless communications and more specifically, the fast growing penetration of cellular phones and cellular infrastructure are the major drivers for the development of new programmable Digital Signal Processors (DSP's). In this tutorial, an overview will be given of recent developments in DSP processor architectures, that makes them well suited to execute computationally intensive algorithms typically found in communications systems. DSP processors have adapted instruction sets, memory architectures and data paths to execute compute intensive communications algorithms efficiently and in a low power fashion. Basic building blocks include convolutional decoders (mainly the Viterbi algorithm), turbo coding algorithms, FIR filters, speech coders, etc. This is illustrated with examples of different commercial and research processors. Please note that the authors do not endorse the processors used in this tutorial. These processors are used to illustrate how different solutions are proposed for the same problem.

Keywords: Digital Signal Processing, architectures, programmable processors, wireless communications.

1. INTRODUCTION

Mobile wireless communications show an incredible growth, as is illustrated in Figure 1. It is estimated that by the year 2010 wireless phones will surpass wire line phones, each having a world-wide penetration of more than 20%.

The market for DSP processors has a growth rate of 40%. In 1996 it was a $2B market, by 1999 it has grown to a $4.4B market and Forward Concepts forecast a $19B market in 2004 [17]. Almost half of all DSP processors will end up in equipment for wireless communications, such as cellular phones, basestations, cordless phones, GPS, etc. There are not only new cellular phone users but it is estimated that in 2000, half of the handsets sold are replacement units [17].

ISLPED '00, Rapallo, Italy.

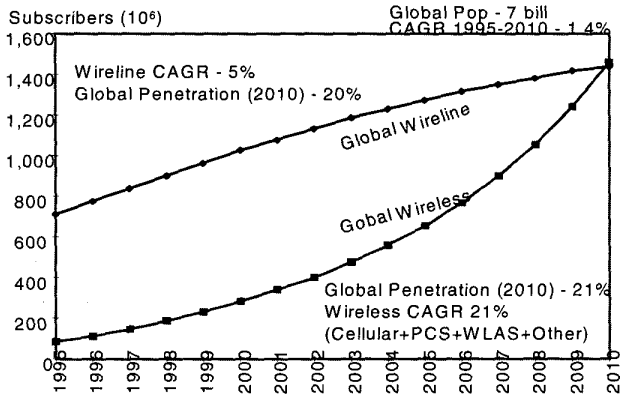

Figure 1: Mobile Wireless Trends

The first successful DSP processors were introduced in the early 80's. Many good overview papers are available that describe the evolution of these processors and the special features to support signal processing applications [10][11][5]. Examples in this category are the Texas Instruments TMS320C1x, C2x, C5x, series or the Lucent DSP16A and DSP1600 series. This tutorial will focus on the evolution of DSP processors during the last couple of years and especially will focus on the special features in the processors to support the demands from wireless communications.

Up till recently, the same DSP processors are used both in the mobile terminal, i.e. the cell phone itself, and the base stations. However, a trend starts to emerge to place *different processors in the mobile terminal and the base stations*. The main drivers for the processors in the mobile are cost and very low energy consumption. This leads to processors that have a very compact but complex instruction set (CISC). The processors in the base stations need to be very high performant and they tend to become more compiler friendly, because the software complexity requires it. Hence, the recent success of VLIW processors for DSP for the base station applications.

Because of this, this tutorial is also split in two main parts. In section 2, DSP processors for mobile terminals are covered. In section 3, processors for base stations are discussed. In section 4, future trends are discussed. Each section has a similar style: the main driver functions are explained first, followed by several processor solutions to the same problem.

It is insightful to first define the meaning of Million Instructions per Second (MIPS) and Million Operations per Second (MOPS). Most traditional DSP processors belong to the class of Complex Instruction Set machines, called CISC processors. This means that in one instruction, typically 16 bits wide, several operations, sources and destinations for the operations are coded. For instance, in one dual-multiply-accumulate operations of the LODE processor, 6 different instructions are performed: two memory read operations, two address calculations and two multiply-accumulate instructions [19]. Assuming the processor runs at 100 MHz, this corresponds to 100 Mips and 600 Mops. If the multiply and add's are considered 2 operations, this becomes 800 Mops. Similarly, in one dual-Mac instruction on the Lucent 16210, 7 different instructions are executed: one 3 input addition, two multiplications, 2 memory reads and 2 address pointer updates. This corresponds to 700 Mops.

Cisc type processors are usually compared on the amount of Mips. Sometimes, to make things confusing, the two multiply-accumulate operations are counted separately. So, it might be a 100 MHz processor, advertised for "200 Mips".

One instruction of a very large instruction word (VLIW) processor consists of a set of small, e.g. 6 or 8, primitive instructions, issued in parallel (more about these processors later). It is custom to multiply the clock frequency of these processors by the number of parallel units and define these as "MIPS". E.g. the TI C6x is a 200 MHz processor with 8 processing units, corresponding to 1600 Mips [18].

To make a fair comparison between processors, we will use the MIPS terminology and count the primitive operations for both the CISC and VLIW machines.

2. DSP's for the Mobile Terminal.

DSP processors are made to support hard real-time signal processing applications. This often translates in the rule that 10% of the code is executed 90% of the time and 90% of the code is executed 10% of the time. The code executed all the time, tends to sit in tight loops, of which every instruction or clock cycle counts. DSP processors are compared based on the number of instructions and the number of clock cycles it takes to execute basic DSP kernels.

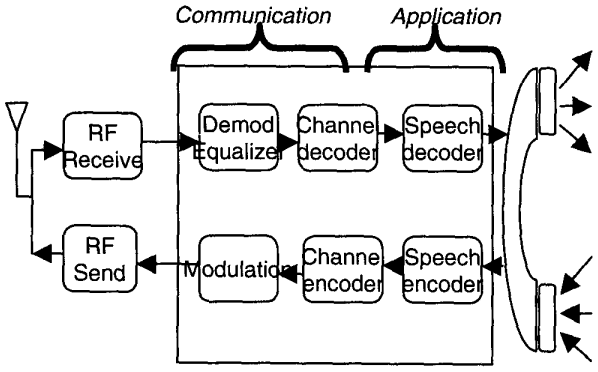

Figure 2: Cellular communication system

The main baseband building blocks of a 2nd generation cellular phones, such as for GSM, GSM+ and IS-95, are shown in Figure 2 [15]. About half of the processing functions are at the physical layer, implementing the modulation/demodulation, consisting mainly of the equalizer, and the channel coder and decoder. The other half of the processing occurs in the application level. For 2nd generation phones this means the speech coder.

All functions of the system of Figure 2, can be implemented in one state-of-art DSP processor running at a clock frequency between 80 MHz to 150 MHz. The differentiation between the processors and implementations sits in either the power consumption, and/or the extra features that are included in the processor such as noise cancellation or more advanced equalizers.

3rd generation cellular wireless standards put higher demands on the modem functions as well as the application functions. More advanced equalizers and more advanced coding algorithms, such as turbo coding algorithms will be used. This is combined with a higher bandwidth requirement. Also on the application side, more advanced features are required such as lower rate speech coders, video communication, data communication, etc. Most of the computations are spent on the following basic building blocks:

- Filters (FIR, IIR), autocorrelations and other "traditional" signal processing functions.
- Convolutional decoders based on the Viterbi algorithm.
- Code book search, max-min search, etc for speech coders and vector search algorithms.
- Turbo decoding for data processing.

2.1 FIR implementation

The basic FIR equation is the following:

$$y(n) = \sum_{i=0}^{i=N-1} c(i) \cdot x(n-i)$$

When this equation is executed in software or assembly code, output samples y(n) are computed in sequence. This means that to compute one output sample, there are N multiply-accumulate operations and 2N memory read operations to fetch the data and the coefficients. N is the number of taps in the filter. It is well known that DSP processors include datapaths to execute multiply accumulate operations in an efficient way. Therefore, we will focus on the memory architecture, which is a much more fundamental design issue for DSP processors.

Memory architectures:

On a traditional von Neumann architecture, 3N access cycles are needed to compute one output: for every tap one needs to fetch one instruction, read one coefficient and read one data sample sequentially from the unified memory space. Already early on, DSP processors differentiated themselves from von Neumann architectures by implementing a Harvard or modified-Harvard architecture [10]. The main characteristic is the use of two memory banks instead of one common memory space in the von Neumann architecture. The Harvard architecture has a separate data memory from program memory. This reduces the number of access cycles from 3 to 2, since the instruction fetch from the program memory can be done in parallel with one of the data fetches. The modified Harvard architecture improves this even further. It is combined with a "repeat" instruction. In this case, one multiply-accumulate instruction is fetched from program

memory and kept in the one instruction deep instruction cache. Then the data access cycles are performed in parallel: the coefficient is fetched from the program memory in parallel with the data sample being fetched from data memory. This architecture is found in all early DSP processors and is the foundation for all following DSP architectures.

Newer generation of DSP processors have even more memory banks, accompanying address generation units and control hardware, such as the repeat instruction, to support multiple parallel accesses. The execution of a 32 tap FIR filter on the dual Mac architecture of the Lucent DSP 16210 is shown in Figure 3. The corresponding pseudo code is the following:

```
do 14 { //one instruction !
    a0=a0+p0+p1
    p0=xh*yh p1=xl*yl
    y=*r0++ x=*pt0++
}
```

This code can be executed in 19 clock cycles with only 38 bytes of instructions code. The inner loop takes one cycle to execute and as can be seen from the assembly code, 7 operations are executed in parallel: one addition, 2 multiplications, 2 memory reads and 2 address pointer updates.

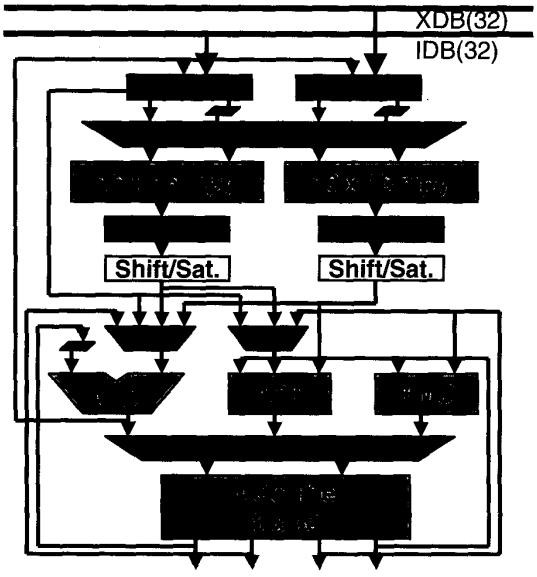

Figure 3: Lucent DSP16210 architecture

The difficult part in the implementation of this tight loop is the arrangement of the data samples in memory. To supply the parallel data paths, two 32 bit data items are read from memory and stored in the X and Y register, as shown in Figure 3. Then the data items are split in an upper half and a low half and supplied to the two 16x16 multipliers in parallel. It requires a correct alignment of the data samples in memory, which is usually a tedious work done by the programmer, since compilers are not able to handle this. A similar problem exists in SIMD instructions on general purpose micro-processors.

A similar approach is used in [9]. Instead of two multipliers, only one multiplier working at double the frequency, is used. But the problem of alignment of data items in memory remains.

In the Lode architecture, a delay register is introduced between the two MAC units as shown in Figure 4. This halves the amount of memory accesses. Two output samples are calculated in parallel as indicated in the pseudo code below. One data bus will read the coefficient from memory, the other data bus will read the data sample from memory. The first Mac will compute a multiply-accumulate for output sample y(n). The second multiply-accumulate will compute in parallel on y(n+1). It will use a delayed value of the input sample.

y(0) = [redacted] + c(1)x(-1) + c(2)x(-2) + . . . + c(N-1)x(1-N);

y(1) = [redacted]x(1) + c(1)[redacted] + c(2)x(-1) + . . . + c(N-1)x(2-N);

y(2) = c(0)x(2) + c(1)x(1) + c(2)x(0) + . . . + c(N-1)x(3-N);

. . .

y(n) = c(0)x(n) + c(1)x(n-1) + c(2)x(n-2)+ . . + c(N-1)x(n-(N-1));

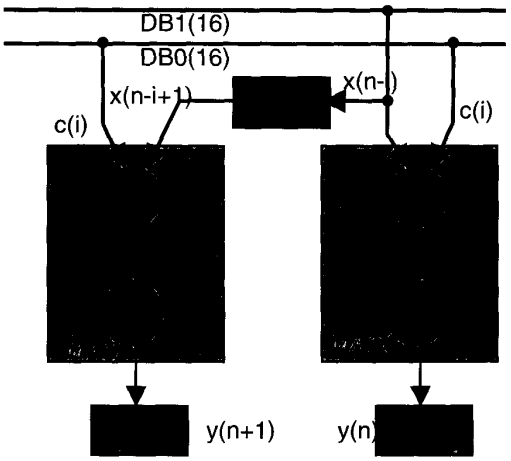

Figure 4: Lode's Dual Mac Architecture with delay register.

This concept of inserting a delay register can be generalized. When the datapath has P Mac units, P-1 delay registers can be inserted and only 2N/(P+1) memory accesses are needed. These delay registers are pipeline registers and hence if more delay registers are used, more initialization and termination cycles need to be introduced. This is summarized in Table 1.

Table 1: Data memory accesses, MAC operations, instruction cycles and instructions for a N tap FIR filter.

DSP	Data memory accesses	MAC's	Instruction cycles	Instructions
Von Neumann	2N	N	3N	2N
Harvard	2N	N	2N	2N
Modified Harvard	2N	N	N	2 (repeat instruction)
Dual Mac	2N	N	N/2	2 (same)
Dual Mac with 1 reg	N	N	N/2	2 (same)
Dual Mac with P reg	2N/(P+1)	N	N/(P+1)	2

2.2 Viterbi acceleration

The Viterbi decoders are used as forward error correction (FEC) devices in many digital communication devices, not only in cellular phones but also in digital modems, etc. The Viterbi

algorithm is a dynamic programming technique to find the most likely sequence of transitions that a convolutional encoder has generated.

Most practical convolutional encoders are rate 1/n (which means that one input bit generates n coded output bits). A convolutional encoder of "constraint length K" can be represented as a Finite State Machine (FSM) with K-1 memory bits. This means that the FSM has 2^{K-1} possible states, also called trellis states. If the input is binary, there are two possible next states starting from a current state, since the next state is computed from the current state and the input bit. The task of the Viterbi decoding algorithm is to reconstruct the most likely sequence of state transitions based on the received bit sequence. This approach is called the "most likelihood sequence estimation." To compute this most likely path, a trellis diagram is constructed. It will compute from every current state, the likelihood of transitioning to one out of two next states. This leads to the kernel of the Viterbi algorithm, called the Viterbi butterfly. This is illustrated in Figure 5.

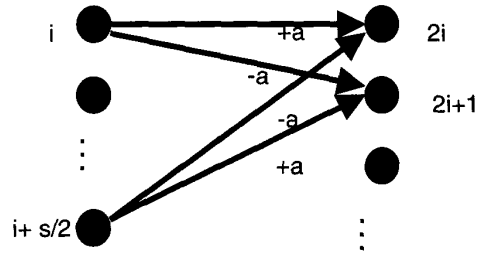

Figure 5: Viterbi butterfly

The basic equations executed in this Butterfly are:

$$d(2i) = \min\{d(i) + a, d(i + s/2) - a\}$$

$$d(2i+1) = \min\{d(i) - a, d(i + s/2) - a\}$$

These equations are implemented by an "Add-Compare-Select" operation. Indeed, one needs to add or subtract the branch metric from states *i* and *i+s/2*, compare them and select the minimum. Similarly, state 2*i*+1 is updated. The butterfly arrangement is chosen because this reduces the amount of memory accesses by half.

DSP processors have special hardware and instructions to implement the Add Compare Select (ACS) operation in the most efficient way. The Lode architecture uses the two MAC units and the ALU to implement the ACS operation as shown in Figure 6. The Dual Mac operates as a dual add/subtract unit. The ALU finds the minimum. The shortest distance is saved to memory and the path indicator, i.e. the decision bit is saved in a special shift register A2. This results in 4 cycles per butterfly.

The Texas Instruments TMS320C54x and the processors described in [9] use a different approach which also results in 4 cycles per butterfly. This is illustrated in Figure 7. The ALU and the accumulator are split into two halves (much like SIMD instructions), and the two halves operate independently. A special compare, select and store unit (CSSU) will compare the two halves, will select the chosen one and write the decision bit into a special register TRN. The processor described in [13] describes two ACS units in parallel. To illustrate the importance of an efficient implementation of the ACS butterflies, consider the IS-95 cellular standard. The IS-95, uses a rate ½ convolutional encoder with a constraint length 9 [15]. It has a window size of 192 samples. This corresponds to $2^8 \times 192 \times (ACS)$ operations. The most efficient implementation requires four cycles per butterfly. This still corresponds to close to 100 MIPS.

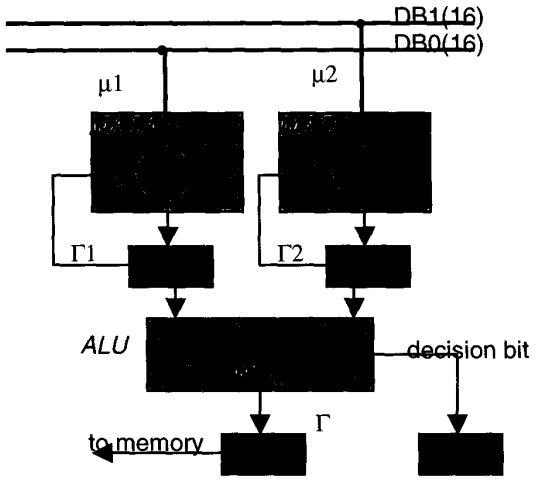

Figure 6: Add compare select on the Lode architecture.

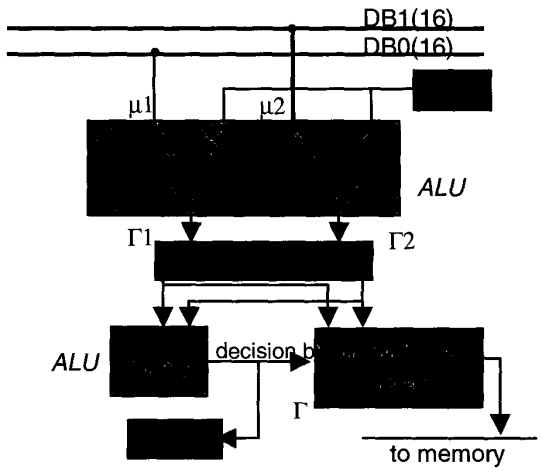

Figure 7: Add compare select on C54x and on [13].

The hardware support for the Viterbi algorithm on the 16210 allows for the automatic storage of decision bits from the ACS computations. This functionality can be switched on or off as required. When the built-in comparison function `cmp1()` is called, the associated decision bit is shuffled into the auxiliary register `ar0` as shown in Figure 8.

As the additions must be carried out manually, each ACS takes two cycles (one for the additions, one for the compare/select) and thus a single butterfly takes a total of four cycles. The following code segment performs the butterfly computations:

```
do 8 {
     a0=a4+y  a1=a5-y  *r3++=a0h
```

```
    a2=a4-y  a3=a5+y  *r5++=a2h
    a0=cmp1(a1,a0) yh=*r0 r0=r1+j j=k k=*pt1++
    a2=cmp1(a3,a2) a4_5h=*pt0++
}
*r2++=ar0
```

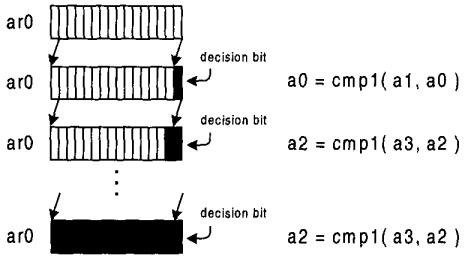

Figure 8: Hardware support for Viterbi on the DSP16210

The 16210 has hardware looping support as described above, and there is only a single cycle required to initialize this looping support before the loop executes with zero overhead. When decoding a standard GSM voice channel, which has a constraint length of 5 or 16 states in the trellis, the `ar0` register is filled with 16 decision bits after the 8 butterflies are processed. Thus, with a single memory access the decision bits can be stored in memory and the next symbol pair can be processed. This is an efficient use of memory bandwidth. For codes with higher constraint lengths and thus more states, the code segment can merely be executed multiple times with each decision bit word written to memory as required.

3. DSPs for Wireless Infrastructure

Whereas most low power wireless research efforts focus on portable handsets, this tutorial also discusses the implications of next-generation standards on the design of mobile infrastructure systems. Basestations (BTS) will require flexible, low cost integrated solutions that are capable of supporting several-standards. There are several trends in the baseband mobile wireless infrastructure market that have significant implications on the design of the DSPs used within.

1. Increased capacity for packet data services.
2. Reduced cost through integration.
3. Multi-standard support.

Increasing the capacity of a network requires an increase in both the receiver sensitivity and the co-channel interference rejection. Suburban cells have relatively static user populations and large area whereas urban cells have dynamic load requirements and small size. By increasing the receiver sensitivity in the BTS, we can reduce the transmit power of the mobile handset. This reduces the co-channel interference in urban cells (and therefore increases capacity by increasing the frequency re-use). It also extends the battery life of the handset. Alternatively, we increase the size of the suburban cells to reduce the cost of network deployment. Furthermore, increasing the receiver sensitivity through advanced channel estimation and decoding techniques reduces the packet-error rate, thereby increasing the capacity of packet-data networks.

Cost-reduction is best-achieved through integration (of both RF and Baseband) into ASICs. As an example, a GSM BTS requires approximately 300 DSP-MIPS to implement a single carrier (transmit & receiver for 8 users). A multi-carrier board therefore has several programmable DSPs – that dominate the cost and power consumption of the baseband processing. These may be integrated into a single-chip solution to reduce cost. A side-effect is the power reduction that impacts the rating of the power supply and the cooling requirements of the cabinet – issues that are crucial in Pico-cell BTS platforms.

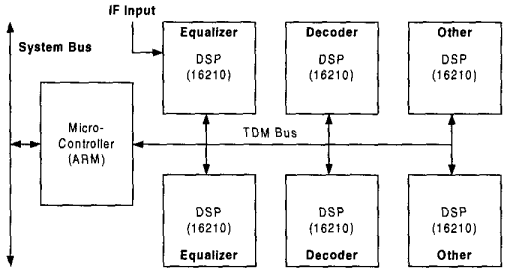

Figure 9: Pool of High-Performance DSPs for Infrastructure Baseband Processing

To satisfy the different bandwidth requirements for different standards, software architectures can be used. Rather than sampling a narrowband signal, a wideband "chunk" of the received signal is sampled at high rates to provide flexibility (such as multi-band and multi-standard support). The digital part of receiver is subdivided into two parts: inner receiver and baseband receiver. The inner receiver performs analog-to-digital conversion (from RF or IF) and filtering. Filtering (including digital down-conversion, narrow-band filtering, decimation and interpolation) provides a 'good' signal for the base-band receiver. The processing of the high frequency signals implies that the inner receiver is normally implemented by an ASIC and a micro-controller. The baseband receiver structure includes demodulation, convolutional/turbo decoding, voice decoding, deciphering, and a microcontroller for interfacing. A baseband receiver can be implemented by either a pool of high-performance DSPs, or DSP+IP-cores.

These trends in wireless infrastructure require very high performance DSPs. There are a number of new high-performance DSPs on the market that will be deployed in future basestation systems. Some have targetted raw clock speed through simple RISC instruction sets, while others opt for complex DSP instructions sets. We now describe some of these processors and describe the mapping of channel decoding and equalisation algorithms on them.

3.1 DSPs for Wireless Infrastructure

3.1.1 TI C6x Family of DSPs

The Texas Instruments 'C6x is a 1600 RISC MIPS, 8-way VLIW DSP with two separate execution clusters. The RISC nature of the 'C6x means there is no hardware support for the Viterbi algorithm, however the eight execute units of the 'C62 allow a great deal of parallel execution.

An implementation of the Viterbi algorithm on the 'C62 [18] performs each butterfly computation in three cycles.

```
JLOOP:
```

```
  [B1]    B      .S1   JLOOP          ;** for j
||[B1]    SUB    .S2   B1,1,B1        ; j++
||[!A2]   STH    .D1   B12,*+A6[8]    ; store new[j+8] = a8
||[!A2]   ADD    .D2   B0,B14,B14     ; tr  |= t8
||        CMPGT  .L1   A11,A10,A1     ; t0 = (b0 > a0)
||        CMPGT  .L2   B11,B10,B0     ; t8 = (b8 > a8)
||        MPY    .M1X  1,B5,A4        ; copy mj

  [A2]    SUB    .S1   A2,1,A2        ; decrement priming
||[!A2]   STH    .D1   A12,*A6++      ; store new[j] = a0
```

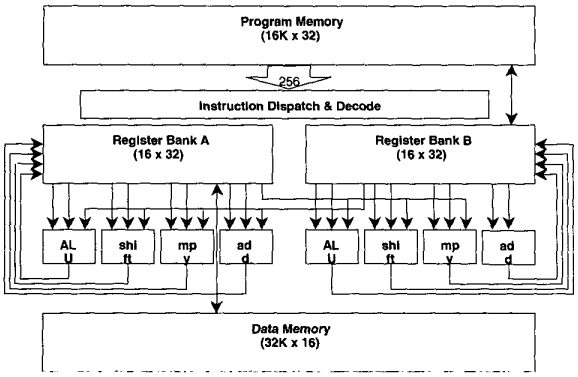

Figure 10. Architecture of the TI C6x DSP

```
||[A1]    ADD    .S2   2,B0,B0        ; t8  |= (t0 << 1)
||[B0]    MPY    .M2   1,B11,B12      ; if (t8) a8 = b8
||        MPY    .M1   1,A10,A12      ; copy a0
||        SUB    .L2X  A7,B5,B10      ; a8 = old0 - mj
||        LDH    .D2   *++B9,B5       ; load mj = m[j]

          SHL    .S2   B14,2,B14      ; tr <<= 2
||[A1]    MPY    .M1   1,A11,A12      ; if (t0) a0 = b0
||        ADD    .S1   A7,A4,A10      ; a0 = old0 + mj
||        SUB    .L1X  B13,A4,A11     ; b0 = old1 - mj
||        ADD    .L2   B13,B5,B11     ; b8 = old1 + mj
```

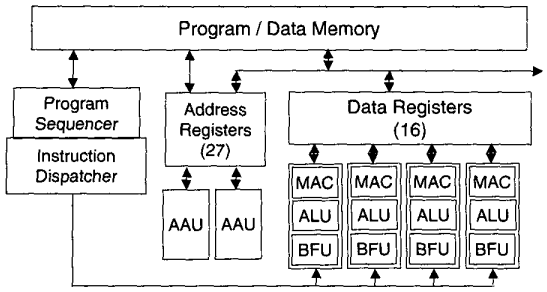

Figure 11. Architecture of Lucent/Motorola Starcore SC140

```
||        MPY    .M2   1,B10,B12      ; copy a8
||        LDH    .D2   *B4++[2],A7    ;* load old0 = old[2*j]
||        LDH    .D1   *A5++[2],B13   ;* load old1 = old[2*j+1]
; end of JLOOP
```

This implementation is bound by memory bandwidth, as the 'C62 has only two addressing/data units and can thus load/store a maximum of two 32-bit words per cycle. The code segment shows the computation requires the loading of the current path metric values, loading the branch metrics from a table, and then storing the new path metrics. Thus, if the branch metrics are stored in a table, there is no way to perform the computations in less than three cycles. The 'C62 has two banks of 16 registers, one for each side of the processor. The implementation of Viterbi uses only 19 of the 32 registers, and yet is essentially storing the two branch metric values in a table and taking a full cycle to look these values up.

3.1.2 The Lucent/Motorola Starcore

The StarCore SC140 is a quad-ALU, 1200 MIPS 6-way VLIW DSP capable of 1200 MMAC/s at 300 MHz [16]. Like the DSP16210, the StarCore features hardware support for the Viterbi algorithm. The StarCore has been specifically targeted towards 3G Wireless infrastructure. The emerging 3G Wireless standards demand greater DSP MIPS than the current 2G standards, and thus it is important that it handles wireless algorithms like Viterbi well. Indeed the StarCore is the industry-best programmable DSP at Viterbi, able to process a butterfly every cycle – a factor of four better than the DPS16210 and three better than the 'C6x.

The hardware Viterbi support also takes the form of the automatic storage of decision bits, via a specialized version of the maximum operation called `max2vit`. StarCore is a 32-bit processor capable of performing arithmetic operations on the higher and lower 16-bit words independently, via instructions such as `add2` and `sub2`. This doubles the throughput in computations that can use these double operands, and the Viterbi algorithm is an ideal candidate for such an optimization as it does not require any multiplications. Butterflies can then be done in parallel within the core loop kernel. The `max2vit` flavor of the `max2` instruction independently selects the maximum in both the higher and lower words of the two operands, storing the result in the second register operand. It also sets the Viterbi Flags in the status which are extracted when the decision bit words are stored.

The extraction of decision bits is done with a specialized store instruction called vsl, or Viterbi Shift Left. This instruction takes two new path metric values and two decision bit words, shifts in the new decision bits and write all four words back into memory.

```
move.2l (r2)+,d0:d1          move.2l (r3)+,d2:d3 tfr d7,d4 tfr d7,d6
add2 d0,d4      sub2 d4,d0   sub2 d6,d2    add2 d2,d6
max2vit d4,d2                max2vit d0,d6
vsl.w d2:d6:d1:d3,(r4)+n0    vsl.f d2:d6:d1:d3,(r5)+n0
```

3.2 Turbo Decoding

While convolutional decoding remains a top priority (the decoding requirement for EDGE has been identified at greater than 500 MIPS), the performance needed for Turbo decoding is an order of magnitude greater. We therefore describe the Turbo decoders needed in 3G systems. Turbo decoding (shown in Figure 12) is a collaborative structure of soft-input/soft-output (SISO) decoders with the inclusion of interleaver memories between decoders to scatter burst errors [3]. Either Soft-Output Viterbi Algorithm (SOVA) [8] or Maximum A Posteriori (MAP) [2] can be used as SISO decoders. Within a turbo decoder, the two decoders can operate on the same or different codes. Turbo codes have been shown to provide coding performance to within 0.7dB of the Shannon limit (after a number of iterations).

The Log-MAP algorithm can be implemented in a manner very similar to the standard Viterbi algorithm. Perhaps the most important difference between the algorithms when they are implemented is the use of a correction factor on the new 'path metric' value (the alpha, beta and log-likelihood ratio values in

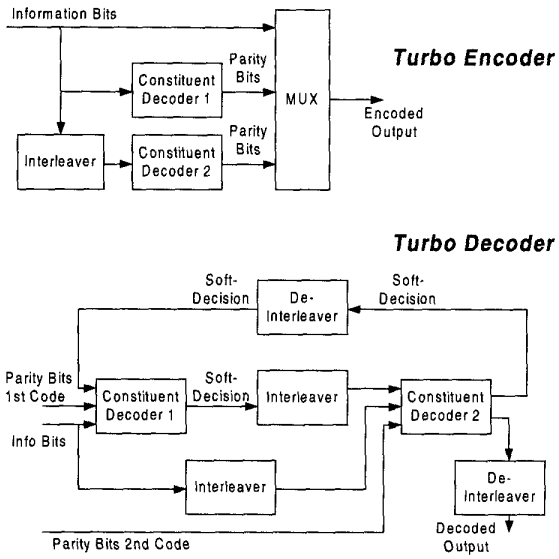

Figure 12: Turbo decoding

Log-MAP) from each ACS, which is dependant on the difference between the values being compared. This is typically implemented using a lookup table, with the absolute value of the difference used as an index into this table and the resulting value added to the selected maximum before it is stored.

Without hardware assistance, these operations alone will increase the number of cycles far beyond those needed for a standard Viterbi kernel as shown in the following Starcore kernel:

```
move.w (r0)+,d0   move.w (r1)+,d1      ; load current data
add d0,d6,d0      sub d6,d0,d5         ; add/subtract gamma values
sub d6,d1,d4      add d1,d6,d1
sub d0,d4,d2      sub d1,d5,d3         ; calculate difference
max d0,d4         max d1,d5            ; calculate maximum
abs d2            abs d3               ; absolute value of index
move.l d2,n0                           ; load index
move.l d3,n0      move.w (r6+n0),d2    ; load index, do lookup
add d4,d2,d4      move.w (r6+n0),d3    ; add correction, do lookup
add d5,d3,d5                           ; add correction
move.2w d4:d5,(r2)+                    ; store new data
```

4. Future Trends

As previously described, the trends in decoding algorithms are moving away from standard Viterbi and towards more computationally-expensive algorithms like SOVA and MAP. These soft-output algorithms, used in turbo decoding and iterative channel equalization, place heavy MIPS requirements on wireless infrastructure. However, there is barely enough support for standard Viterbi in programmable DSPs. The 16210 and other earlier DSPs do have elegant hardware support, but lack the raw computational power to handle next-generation wireless standards. Next-generation DSPs like the 'C62 have increased MIPS, but lack any form of hardware support for the algorithms critical to wireless infrastructure. *It is critical for next generation programmable DSP to address the requirements of algorithms such as SOVA or MAP, since these algorithms are essential for improved 2G and 3G wireless communications.* It is clear that there needs to be a shift in focus for programmable DSPs - to have the levels of performance required to implement a software radio system for wireless infrastructure.

4.1 Multi-Processor DSP Systems

The VLIW architectures presented thus-far provide a certain level of speed-up from a single thread of DSP code. These solutions are not scalable to provide very high levels of DSP performance needed by next-generation wireless infrastructure. Furthermore, the power consumption does not scale linearly with the number of execution units due to the overhead of the instruction dispatch logic.

Some type of multi-processor DSP approach is needed to achieve true performance and power scalability. Fortunately – many DSP applications can be partitioned into parallel threads. Examples of this are the channel equalisation and channel decoding functions. These can be executed on separate DSP processors. Integrating multiple DSP cores into a single chip is achieved using a MIMD DSP architecture like Daytona [1][21] (shown in Figure 13) which uses a high performance split transaction bus to connect multiple DSP cores with a memory hierarchy. Each DSP core has a cache for both instructions and data – to minimize the memory connected to each DSP.

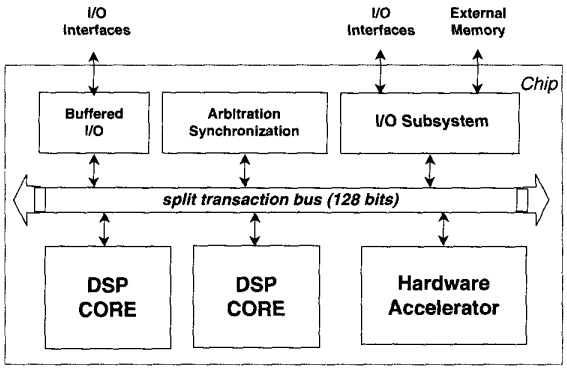

Figure 13. Architecture of Daytona MIMD DSP.

5. ACKNOWLEDGMENTS

We would like to acknowledge the experts of DSP processors whose sources we use and whose interaction we appreciate: Wanda Gass, Gareth Hughes, Mihran Touriguian, Katsuhiko Ueda, Bing Xu.

6. REFERENCES

[1] B. Ackland & P. D'Arcy, "A New Generation of DSP Architectures", *Proc. IEEE CICC99*, Paper 25.1.

[2] L. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", IEEE Trans. Information Thoery, V IT-20, pp 284-287, Mar 1974

[3] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes (1)", *Proc. ICC'93*, May 1993.

[4] G. D. Forney, Jr., "Maximum Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference", *IEEE Trans. Inform. Theory*, V IT-18, pp. 363-378, May 1972.

[5] W. Gass, D. Bartley, "Programmable DSPs" Chapter 9 *Digital Signal Processing for Multimedia Systems*, Marcel Dekker Inc. 1999.

[6] A. Gatherer, T. Stelzler, M. McMahan, E. Auslander, "DSP-Based Architectures for Mobile Communications: Past, Present and Future," IEEE Communications Magazine, pg. 84-90, January 2000.

[7] L. C. Godara, "Application of Antenna Arrays to Mobile Communications: Part 1", *Proc. IEEE*, Vol 85, No. 7. pp 1031-1060, July 97.

[8] J. Hagenauer, P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications", *Proc. Globecom '89*, Nov 1989, pp 47.1.1-47.1.7

[9] H. Kabuo, M. Okamoto et al. "An 80 MOPS Peak High Speed and Low Power consumption 16-bit Digital Signal Processor, *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 4, pp. 494-503, 1996.

[10] P. Lapsley, J. Bier, A. Shoham, E.Lee, *DSP Processor Fundamentals* IEEE Press, 1997.

[11] E.A. Lee, "Programmable DSP Processors: Part I and II," *IEEE ASSP Magazine*, Oct. 1988 and Jan. 1989.

[12] W. Lee et al., "A 1-V Programmable DSP for Wireless Communications," *IEEE Journal of Solid-State Circuits*, Vol. 32, no. 11, Nov. 1997.

[13] M. Okamoto, K. Stone, T. Sawai, H. Kabuo, S. Marui, M. Yamasaki, Y. Uto, Y. Sugisawa, Y. Sasagawa, T. Ishikawa, H. Suzuki, N, Minamida, R. Yamanaka, K. Ueda, "A High Performance DSP Architecture for Next Generation Mobile Phone Systems," 1998 IEEE DSP Workshop.

[14] M.W. Oliphant, "The Mobile Phone meets the Internet", *IEEE Spectrum* pp. 20-28, Aug. 1999.

[15] T. Rappaport, *Wireless Communications, Principles & Practices*, Prentice Hall, 1996.

[16] "Starcore Launched First Architecture", *Microprocessor Report*, V12, No. 14. pp 22, Oct 1998.

[17] W. Strauss, "DSP Markets head for record," EE Times, May 29, 2000, midyear forecast.

[18] J. Turley, H. Hakkaraainen, "TI's new 'C6x DSP Screams at 1600 MIPS", *Microprocessor Report*, Vol 11, No. 2, pp 14, Feb 1997.

[19] I. Verbauwhede, M. Touriguian, "A Low Power DSP Engine for Wireless Communications," *Journal of VLSI Signal Processing* 18, pg. 177-186, 1998, Kluwer Academic Publishers.

[20] I. Verbauwhede, M. Touriguian, "Wireless digital signal processors," Chapter 11 in *Digital Signal Processing for Multimedia Systems*, Edited by K.K. Parhi, T. Nishitani, Publisher: Marcel Dekker Inc., New York, 1999.

[21] J. Williams, K.J. Singh, C.J. Nicol, B. Ackland, "A 3.2 GOPs Multiprocessor DSP for Communication Applications", *Proc. IEEE ISSCC2000*, Paper 4.2.