# Evaluation of the potential of the VLIW Digital Signal Processor TMS320C6201 for UMTS FDD standard baseband processing implementation

Geneviève Baudoin
ESIEE
*baudoing@esiee.fr*

Roman Marsálek, Jiri Prokes
Technical University of Brno
*marsaler@esiee.fr*

## Abstract

*In this paper, we evaluate the potential of the new VLIW Digital Signal Processor TM320C6201, to implement the baseband processing of a UMTS (Universal Mobile Telecommunication System) base station receiver. We have focussed our attention on the DSP implementation of the Viterbi decoder and of the synchronization task of the base station receiver.*

## 1 Introduction

The UMTS is a new generation standard for mobile communications. The standardization process is still under development and the first commercial use is planned to year 2002. Compared to preceding generations such as GSM, UMTS will provide higher data rates for multimedia communications. One of the multiple access techniques for UMTS is W-CDMA Wideband Code Divsion Multiple Access [4].

The UMTS standard will require a heavy processing load from the mobile and the base station. But at the same time new powerful DSPs, such as Very Long Instruction Word VLIW DSP, are appearing on the market. The typical speed of these DSP is above 1000 Mips. As their cost and power consumption are quite high, they cannot not be used in mobile terminals but they are interesting for base stations.

## 2 UMTS FDD schematic for Uplink

Through our work, simple UMTS link on one dedicated physical channel [3] was considered. In this paper, only the uplink transmission is described, the downlink being very similar. There are two types of dedicated physical channels: Dedicated Physical Data Channel DPDCH and Dedicated Physical Control Channel DPCCH. They are multiplexed. Figure 1 is a Simple schematic of uplink transmitter. User data are first coded with convolutional encoder (rate 1/3 and constraint length K=9), rate matched and interleaved. After interleaving, data are spread with a channelization code with a basic chip rate of 4.096 Mcps (different codes $c_d$ and $c_c$ are used for control and data channels). The signal is then scrambled with a complex scrambling code $(a_r + ja_q)$. The resulting complex signal is filtered with shaping root raised cosine filter of 5 MHz bandwidth and subsequently modulated.

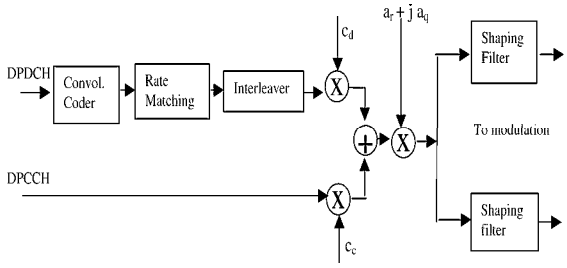


Figure 1: Uplink transmitter

In the receiver, Rake structure with coherent channel estimation is used. Outputs from all fingers are added together and resulting signal is deinterleaved, rate-dematched and decoded (using Viterbi algorithm): see Figure 2. But as a first step, baseband signal (after demodulation) must be synchronized, the first chip of the spreading sequence has to be identified. There are two devices for synchronization: Acquisition and Tracking device. Initially, acquisition is used to find beginning of the frame (length of frame is 10ms) and then synchronization is refined using Tracking device.

## 3 Choice of tested algorithms

We have choosen to test 2 algorithms for the DSP implementation: the Viterbi algorithm and the correlator algorithm which is used intensively in the synchronization process. Both algorithms require a lot of calculations, the Viterbi decoder relying more on logical operations and the correlation only on arithmetic operations.

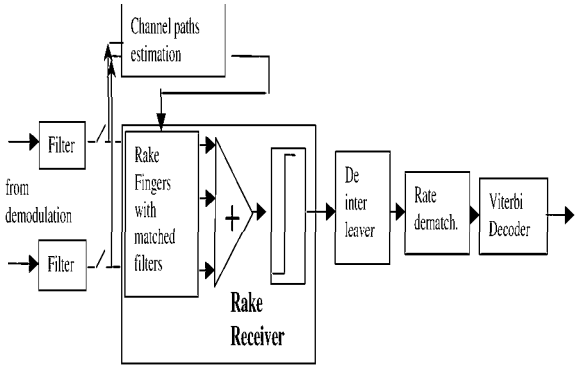


Figure 2: Uplink receiver

## 4 Presentation of DSP TMS320C6201

The TMS320C6201 is a new powerful 32-bit fixed point DSP from Texas Instruments. The TMS320C62xx family of DSPs use the VelociTI architecture, a high-performance, advanced VLIW (Very Long Instruction Word) architecture. The VLIW architecture makes use of multiple execution units running in parallel, performing multiple instructions during a single clock cycle. There are 8 execution units in the C6201. Instructions are packed in Very Long Words consisting of 8 32-bits instructions, and when the 8 instructions are executed in parallel the DSP can operate at a maximum of 1600 Mips.

### VelociTI's advanced feature

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

### Main features of TMS320C6201

- Performance up to 1600 million instructions per second (MIPS),
- CPUs frequency is 200 MHz (5-ns clock cycle time),
- Up to eight 32-bit instructions every cycle (executes up to eight instruction in parallel),
- 32 general purpose registers of 32-bit word length,
- 8 control registers,
- 8 functional units,
- 2 multipliers,
- 6 ALUs (Arithmetic Logic Unit),
- 8/16/32-bit data support.

### Internal memory

The TMS320C62xx have a 32-bit byte-addressable address space. Internal (on-chip) memory is organized in separate data and program spaces. In the C62xx there are two 32-bit internal ports to access internal data memory. The C62xx have a single internal port to access internal program memory, with an instruction-fetch width of 256-bits.

### Functional unit types and performed operations:

There are 2 sets of 4 functional units: named L, S, M and D units.

- .L unit
  - 32/40-bit arithmetic and compare operations,
  - Leftmost 0 or 1 bit counting for 32 bits,
  - 32-bit logical operations.
- .S unit
  - 32-bit arithmetic and logical operations,
  - 32/40-bit shift and 32-bit bit-field operations,
  - Branches,
  - Constant generations,
  - Register transfer to/from the control register file (.S2 only).
- .M unit
  - 16 x 16 bit multiply operations.
- .D unit
  - 32-bit add, subtract, linear and circular address calculation,
  - Loads and stores with a 5-bit constant offset,
  - Loads and stores with a 15-bit constant offset (.D2 only).

For more details about TMS320C62xx architecture see [1].

## 5 Program optimization techniques

There are 4 main phases in the code development flow, which are used according to required code efficiency, see [1, 2].

1. Writing C code
2. Refining C code
   - C compiler optimization,
   - Using intrinsics to replace complicated C code,
   - Using word access to operate on 16-bit data stored in the high and low parts of a 32-bit register,
   - Software pipelining the instruction manually,
   - Using down counting loop counter,
   - Eliminating redundant loops,
   - Loop unrolling.
3. Writing linear assembly Linear assembler is a simplified assembler in which the programmer

don't have to specify neither on which functionnal unit an instruction should run, neither which instructions are to be run in parallel.
- Translating C code to Linear Assembly,
- Drawing a Dependency Graph,
- Linear Assembly Resource Allocation,
- Using the assembly optimizer.

4. Writing final assembly
   - Modulo iteration interval scheduling,
   - Determining the minimum iteration interval.

# 6 Viterbi decoder

Viterbi decoder is an efficient technique for the decoding of convolutionaly coded data, first introduced in 1967. Convolutional coder in UMTS has a constraint length K=9 and coding rate 1/3. Decoding is based on comparing likelihoods between received signal and all possible sent sequences. It can use different distances (either Hamming distance or Euclidean type distance). For our simulations, we used Hamming distance (hard decision).

### Principle of Viterbi decoder

Viterbi decoder principle can be well illustrated using trellis representation. As the trellis of the UMTS coder is too big for drawing (constraint K=9 gives 256 states of trellis), we give in Figure 3 an example of trellis for coder with K=3 There are three main parts of trellis. In the first one we simply calculate hamming distances for K-1 bits. In the second one, main part, for each node there are two possible incoming paths - upper and lower, depending on wether a 0 or a 1 is transmitted. Algorithm must choose the one with smaller hamming distance, the survivor, and store information about this decision. Then in the last tail part we generally know that zeros were sent to set coder into initial zero state. Afterwards, traceback routine is performed to obtain estimate of received sequence.

### Implementation

For the implementation of the main part of the algorithm, we are using the butterfly structure which is highlighted on figure 3. A butterfly has 2 input nodes and 2 output nodes. The 2 input nodes correspond to states with the same 8 bits except for the LSB. So whatever bit is sent, these nodes lead to the same output nodes.

In one 32-bit word, we store the possible coder outputs for the whole butterfly, which can be loaded from memory together. In each byte, the value for one local path is stored. Then hamming distance is calculated using XOR between received sequence and

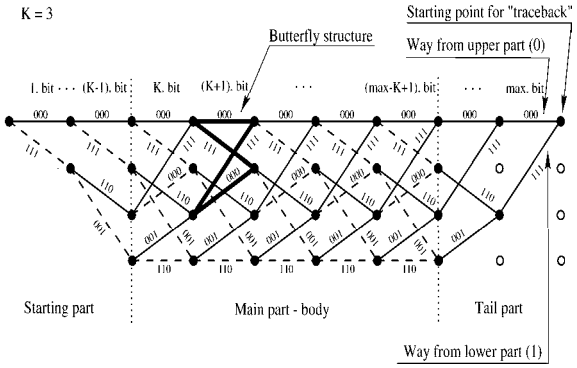


Figure 3: Example of trellis

4 possible butterfly sequences by successively calculating the number of ones in each byte.

### Optimization results

The following results were obtained for a sequence with 50-bit before coding (42 information bits and 8 tail bits). The first version was written in C with no compiler optimization. Then compiler optimization was used (compilers parameter -o1..-o3). Up to level -o3, program works correctly, but for -o3, functionality is disrupted by optimizer. Next versions were written in linear assembler and then also in pure assembler using "hand-made" optimization.

| Type of optimization | Benchmark in cycles |
|---|---|
| No optimization | 3.059.298 |
| Compiler option -o1 | 1.876.316 |
| Compiler option -o2 | 1.734.911 |
| Compiler option -o3 | 1.674.622 |
| Linear assembler | 269.925 |
| Hand optimized assembler | 87.832 |

Table 1: Benchmarks for Viterbi decoder

As you can see from the table 1, rewritting of code into linear assembler gives 6 times faster code comparing with optimized C program. Hand optimized assembler code can improve speed 20 times. For another length (defined by parameter *max*) of coded sequence, the required number of cycles for decoding can be calculated (approximately) by the following equation:

$$\text{Number\_of\_cycles} = 3400 + (max - 8) * 2010 \quad (1)$$

So for 8kb/s speech service, with 104 bits before convolutional coding in each frame, decoding time is less than 1ms (frame duration is 10 ms).

# 7 Correlator - hypothesis testing

### Principle of synchronization

The most important part of synchronization devices is hypothesis testing device (correlator), which is used

for acquisition as well as in early-late tracking device. In correlator, demodulated signal is firstly descrambled with the same scrambling sequence as in transmitter. Then channelization sequence is removed also using multiplication with the same channelisation sequence as in transmitter. Result is accumulated over N chips and phase dependancy is removed with squaring and addition of I and Q branches. This proccess is performed for a large number of possible scrambling sequence shifts and the one with highest output correlation value is chosen to be right delay.

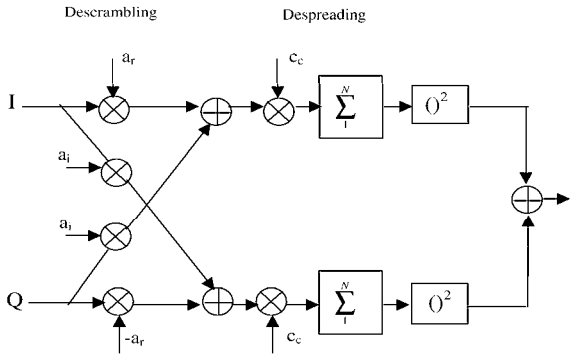


Figure 4: Correlator

### Implementation

For reducing the calculations load, scrambling sequence was first multiplied with control channel channelisation sequence and then only operations corresponding to descrambling with this new sequence had to be done. Two samples per chip were used for obtaining smaller timing error. Data were stored in integer fixed point format.

### Optimization and Results

For correlator implementation, only C language was used till now. Length of scrambling sequence used for calculations was 1536 chips (corresponding to 6 pilot bits present in one control channel slot). The results presented here were obtained for 1000 examined shifts of descrambling sequence.

| Used optimization | Data in external memory | Data in internal memory |
|---|---|---|
| Short access, no optimization | 307.29 Mcycles | 102.20 Mcycles |
| Short access, -o3 | 108.09 Mcycles | 4.69 Mcycles |
| Word access, intrinsics | 257.82 Mcycles | 52.85 Mcycles |
| Word access, intrinsics, -o3 | 54.83 Mcycles | 3.91 Mcycles |

Table 2: Results for the correlator

In all simulations data were expected to have short type. In the first two programs, short access to data was used, without and with optimization. In the 2 remaining cases, word access to data was used, i.e. in each word two short data were stored. Then intrinsics (special C functions representing assembler instructions) for multiplication between low and high parts of word were used to improve performance.

Allocation of data in internal or external memory was examined, and results show that it is obligatory to store data into internal memory, because in other case, it is not possible to optimize code speed due to the slow access to external memory. Assumming 5 ns instruction cycle of DSP, our best result is less then 20 ms.

It is probably possible to improve correlators code efficiency by rewritting code into assembler or at least into linear assembler.

## 8 Conclusion

The C6201 DSP is very well suited for basestation processings requiring a lot of arithmetic operations, such as the correlators of synchronization process. But for purely logical processing the performances are not very interesting. On example for Viterbi decoding, classical DSPs with integrated Viterbi accelerator give equivalent results. while the performance for the correlator on a C6201 are significantly better than those of a C541. This can be explained because the correlator use all the functional units while the Viterbi decoder never uses the 2 multiplier units and there can be only 6 instructions executing in parallel. Even if the C development tools of the DSP are rather powerful, hand-written assembler is much more efficient, as it can be seen with the Viterbi example.

## 9 References

## References


[1] Texas-Instruments, LaTeX, *TMS320C62x/C67x CPU and Instruction Set - Reference Guide*, March 1998.

[2] Texas-Instruments, LaTeX, *TMS320C62xx DSP Design Workshop - Student Guide*, April 1997.

[3] ITU submission editor, LaTeX, *The ETSI UMTS Terrestrial Radio Access (UTRA) ITU-R RTT Candidate Submission*, 1998.

[4] Viterbi A.J. , LaTeX, *CDMA, Principles of Spread Spectrum Communication*, Addison-Wesley, 1995.