Title

J.R. Author

May 24, 1999

Chapter 1

Introduction

- Why did we do this?
- What is component-based design?
- What question were we trying to answer?
- Outline of the report

Chapter 2

Viterbi Algorithm

Introduction

2.1 Our specification

2.2 Metrics & Analysis Benchmarks

Chapter 3

Implementions

We chose to implement the Viterbi decoder in four different component design styles, for comparison. The different implementations are software running on a general-purpose processor, software running on a DSP, software running on a customizable processor, and an ASIC soft core implementation using standard cells. These styles covers the broad range of choices faced by today's system designer, and so it is hoped that this study reflects real-world practices.

For the general-purpose processor, the ARM8 core from ARM was studied. This processor is known for its good performance and low power qualties, and so was chosen as a viable competitor against the DSP and ASIC solutions.

The TMS320C54x processor from Texas Instruments was chosen for the DSP implementation. As will be explained, this DSP is geared specially towards the Viterbi algorithm, and so this would be a prime choice in a real-world design.

Also studied here is the new Xtensa configurable microprocessor core by Tensilica. This processor allows some customization of the instruction set, and thus allows some of the benefits of the design flexibility available in hardware with the ease-of-design of a software-based platform.

Finally, an ASIC solution was studied to try to quanify the performance and power benefits available when designing in a hardware style.

Note that there are other interesting design styles available, such as an FPGA-based or other reconfigurable logic system. We chose to focus on only the above four styles due to limited resources.

3.1 Software on ARM

The ARM family of microprocessors are 32-bit processors, featuring a scalar integer unit, with no floating point hardware. The architecture is a fairly standard 5-state RISC pipeline. Most ALU operations (including a barrel shifter, capable of arbitrary shifts of data) require one cycle to complete; some instructions can be issued in one cycle, but delay their results for several cycles.

3.1.1 Implementation Details

Using Lookup Tables

The branch metric computation specified in the project description can be very inefficient if implemented using ALU operations. Two multiplications and an addition are required for each branch computation. Multiplications are especially costly, as the result from these operations is only available 3 cycles after the instruction is issued; these pipeline slots must be filled with instructions which are independent of the multiplication results.

In contrast, implementing the branch metric computation as a lookup table results in a significant performance gain. Memory accesses require only a 1 cycle delay slot, as opposed to the potential 3 for multiplication. Thus not only are the number of instructions decreased, but we then have fewer constraints on instruction scheduling.

The only drawback to using a lookup table is the potentially large size of the table. Since the I and Q components of the metric act orthogonally, we only require a 64-element array to for each functionally-distinct metric computation. There are two such tables (one for (x-1)(x-1) and one for (x+1)(x+1)), so 128 words (512 bytes) are required.

Additionally, the Viterbi code computation (essentially a parity computation) is efficiently done with a lookup table as well; this takes 64 words, so a total of 768 bytes is needed for lookup tables.

Caching Strategy

Early on, we recognized that poor cache management would result in a large performance degradation; minimization of memory utilization would be essential to achieving good data rates. To this end, we chose to implement the traceback buffer as a packed array of bits; each traceback only contains one bit of information, the LSB of the previous state, and the upper bits can be derived from the current state. Thus 32 bits of traceback can be packed into a single word in memory. Extraction of the packed bits is made efficient through the barrel shifter, and so the performance lost due to unpacking is minimized.

Quality of Results

The bit error rate (BER) achieved using 16-bit fixed point computation and a 128-step traceback buffer was 1.50e-4 for the reference data set. Using the SNR degradation estimation provided with the reference data gives a resulting 0.196 dB degradation. This is exactly the same results obtained from the reference implementation, so we assume the performance of the ARM implementation is acceptable.

We attempted to adjust the algorithm to obtain better performance; an easy way to do this is to reduce the traceback depth. Reducing the traceback to 64-levels increased the BER to 7.00e-4 with a corresponding 0.725 dB SNR degradation. This is unacceptable, given a 0.05 dB SNR budget.

We did another experiment where we used a single-bit encoding scheme; using single bits may give an easy way to perform many path metric computations in parallel,

by using the full 32-bit width of the ALU. However, the quality degradation is severe; the corresponding BER is 1.36e-2, with a 1.963 dB SNR degradation.

The final implementation was thus chosen to use 16-bit resolution and a 128-level traceback, in order to meet the quality of results requirements.

Development and Simulation

An instruction-level simulator for the ARM8 core was available from Prof. Rabaey's group. This simulator estimates power consumed for a given program based on profiling of the code.

Power models were also available for the SA-110 microcontroller, but the parameters of the project dictated that we use a standalone core, so the ARM8 was chosen here. The models also include power for an 8k unified I+D cache.

Unfortunately, power characterization was only available for the core running at 125 MHz and 3.3 V. As we will see in the next section, we require higher performance to match the desired bitrate for the project. We will subsequently show how to scale these results to meet the performance requirements.

3.1.2 Observations

Raw Results

For decoding 4096 bits, the simulator indicated that 9.67 Mcycles of computation would be required, with 53.14 mW of power used. This gives a datarate of 52947 bits/s. Again, the power is characterized for 125 MHz operation and 3.3 V. These figures include the power used for the cache.

Result Scaling

We note that the ARM architecture is capable of higher speeds than that given by the simulator; 275 MHz SA-110 devices are available in current commercial products. As well, the SA-110 family from Intel operates at 2.0 V. Thus we assume that these are achievable goals for our design. If we assume that power varies with the square of the voltage and linearly with the frequency, then our results scale to give a 116484 bits/s datarate and 42.94 mW power consumption.

Design Cost (Area)

The ARM8 core proper occupies an area of 3.75 mm x 1.65 mm, while the 8k cache needs 3.2 mm x 0.4 mm, both in a 0.25 um process. Thus the total area for the design is $7.47mm^2$

Design Effort

An estimated 4 person-days of design work was required for this design, including a small amount of time for familiarization with the ARM architecture for design optimization.

The software is written completely in ANSI C code, so this design should be very portable. Some of the architecture-dependent design optimizations may need to be rethought based on the new architecture, however.

3.1.3 Summary

Clock Speed	275 MHz
Execution Performance	116.5 kb/s
Area	7.47 mmˆ2 (0.25 um)
Power Dissipation	42.94 mW
	5.68 mW/mmˆ2

3.2 Software on TI 320C54

The TI TMS320C54x DSP is a new, low power line of DSPs unveiled by TI in June 1998. These chips use 16 bit fixed point words, and can run at as low as .45 mW / MIPS for chips built in 0.25 micron technology.

For our implementation, we choose to use the VC5402, which has the following features:

- Operates at 100 MIPS with a core voltage of 1.8V
- I/O pins operate at 3.3V
- 16K word x 16 bits of dual-access RAM
- 4K word x 16 bits of ROM
- Internal DMA
- Is created in 0.18 micron technology (Results are later scaled up to a .25 micron implementation for comparison purposes)

Hardware capabilities of the entire C54x series include:

- Three 16-bit data buses and one 16-bit program memory bus
- 40 bit ACC with 40 bit barrel shifter and two independant accumulators
- A single cycle non-pipelined MAC
- Single-instruction repeat and block-repeat operations for program code
- Block-memory-move instructions for better program and data management
- Arithmetic instructions with parallel store and parallel load
- Up to 168K single access RAM
- Up to 32K dual access ram
- Up to 8M word external memory access
- Six channel DMA controller

3.2.1 Implementation Details

Texas instruments provides a reference document describing how to implement a Viterbi Decoder on a C54 DSP. This document can be downloaded at:

`http://www-s.ti.com/sc/psheets/spra071/spra071.pdf`

The C54x is highly optimized to perform the Viterbi Decomposition due to the following features of its ALU:

- A single cycle compare, select, and store unit (CSSU) is used to compare branch metrics, record the larger value, and store the appropriate decision bit, all in one cycle
- Dual accumulators allow for dual add/subtract operations to occur in one clock cycle
- Address pointer registers can be incremented/decremented in a circular buffer as part of these instructions
- The above 2 instructions allow a butterfly to occur in 5 cycles - 1 load local distance, 2 dual add/subtract, 2 CSSU

An assembly program was created using code snippets from the TI Viterbi Decoder Appplication Report which showed how to best take advantage of these features. The code can be found in the Appendices.

The only deviation from the specifications listed above is that a length 128 traceback was used, with only the final 64 bits being recorded. A multiple of 16 was chosen to simplify the code for traceback, and does not degrade the performance of the algorithm (in fact, it enhances it). Since 16 bit arithmetic was used upon 6 bit input words, there was no quantization loss in the algorithm (beyond input quantization noise), so it easily met the SNR and BER requirements.

The input I and Q data was assumed to have been put into a partiuclar memory location by one of the on-chip DMA channels, and the output data words are assumed to have been moved to their final destination by another on-chip DMA channel. Alternatively, since the Viterbi decoding algorithm does not require all of the processing power of the chip, the input data might have been placed in its memory location after the TI chip was used to run some filtering or other DSP algorithm upon the datapoints.

3.2.2 Observations

The code mentioned above was simulated on a cycle-accurate TI simulator, which included a memory map specific to this processor. To decode 64 bits, the decoding algorithm took, on average, 13780 cycles to run (13714 cycles if the path metrics did not need to be decremented, 13914 cycles if the path metrics did need to be updated). At this rate, to decode 100 Kbits/second, which can be handled if the chip runs at 22 MHz. Alternatively, the chip could handle a maximum of 464.7 Kbits/second at 100 MHz. Since we are scaling the technology up from 0.18 to 0.25 technology, it is unlikely that the 1.8V core could be clocked at 100 MHz, but 22 MHz should be easily attainable.

As seen below, the code needed used only about a fourth of the 4Kx16 ROM, and the data storage necessary was a little over 1/16 of the available 16K x 16 dual access RAM.

```
Code Size: 1032 (16 bit) Program Words
Data Storage: 1280 (16 bit) Data Words
Cycles Required (100 Kbps) 13780
Maximum Speed (100 MIPS) 464.7
```

The results closely follow the estimates from TI. From the TI Appplication Report, we have the following predictions for required operations/frame: (FS = frame size, FR = frame rate)

Metric update: Cycles/frame = (#States/2 butterflies butterfly calculation + TRN store + local dist calculation.) # bits = (2 K-2 5 + 2 K-5 + 1 + n 2 n-1) FS Traceback: Cycles/frame = (loop overhead and data storage + loop 16) # bits/16 = (9 + 12 16) FS/16 = 201 FS/16 Data reversal: Cycles/frame = 43 FS/16 Total MIPS = Frame rate (metric update + traceback + data reversal) cycles/frame = FR [(2 K-2 5 + 2 K-5 + 1 + n 2 n-1) FS + (201/16) FS + (43/16) FS] = FR FS (2 K-2 5 + 2 K-5 + 1 + n 2 n-1 + (201 + 43)/16) = FR FS (2 K-2 5 + 2 K-5 + n 2 n-1 + 16.25)

So, for a frame size of 100K bits and frame rate of 1 Hz, the estimate was18.425 MIPS at 100 Kbits/sec, or 582 Kbits/sec maximum at 100 MIPS. The difference between the estimate and the actual implementation is due to necessary overhead of setting up pointers, creating loops, decrementing path metrics, and doing a traceback twice for all bits of data (since only the last 64 bits of a length 128 traceback are stored).

Power

Using another Application Report from TI, Calculation of TMS320LC54x Power Dissipation, I examined every instruction used in my program. Since TI estimates current usage based upon half nop and half MAC instructions, I concluded that my program averaged 1.08 times more current usage than their standard figure of .45 mA/MIPS (which uses half MACs, half NOPs) for .25 micron technology. Their chips are made using static CMOS designs, made to run at frequencies approaching 0 Hz, so for a clock rate of 22 MHz the power should scale linearly. Thus, we calculate that to run the Viterbi decoder at 100 Kbits/sec using a C5402 DSP requires 13.78 mW.

```
           Voltage   Current    Power
DSP Core   1.8 V     10.69 mA   19.25 mW
```

This figure does not include any estimate of power to drive I/O pins since our comparison is upon the core only.

Physical Chip Size

TI does not publish their die sizes, and we were unable to find a suitable figure anywhere in the literature. This chip (and other ones with larger memories) is implemented

on a 144 pin ball grid array (BGA) measuring 12 mm per side. The area in this package usable for a die measures 3.2 mm on a side, so the maximum size of the die and memory is $10.24mm^2$.

Cost

```
Cost of Chip (50K quantities) $5 - $75 ea.
Emulator Development Kit $2,995
Code Composer Development Environment $3156.45
Simulator $1578.22
C Compiler/Assembler/Linker $2367.33
Debugger $3156.45
```

Development Tools

TI boasts a fairly sophisticated, integrated set of development tools. The Code Composer Development Environment claims to seamlessly integrate the compiler/assembler/linker, debugger, simulator and emulator. It includes a signal probe, profiler, multiprocessor debugging, data visualization, interactive compiling, and a development environment similar to Microsoft Visual C++. Run-time libraries are available to speed code development and assure code optimality, and TI claims efficiency of compiled code to be close to hand-assembled code.

This project did not use the C compiler at all, the entire program was created in assembly language. We found that many features of the debugger seemed to be better suited to debugging C code rather than assembly. The tools overall were well integrated and helpful, although there is still plenty of work to be done that would make them better.

Development Time

This project could be completed in approximately 3 engineer days, from start to finish, using the latest tools. This is based upon the somewhat sophisticated (comparitively) development tools available for the TI line of DSPs, the availability of run-time libraries, the availability of the Viterbi algorithm application report (with included code snippets), and the availability of configurable emulator boards with JTAG pins. It also assumes that the engineer is familiar with the TI tools and the TI C54x assembly language. Since I was familiar with neither, this project took me somewhat longer to finish.

3.2.3 Summary

This chip is an excellent choice for a low data rate implementation of the Viterbi Decoder. The chip is fast, low power, and has specialized instructions that greatly accelerate the speed of decoding. An application report from TI will speed development, and the with less than 25% of the chip operation time and only a fraction of its memory being utilized for a 100 Kbit/sec decoder, there is plenty of processing power remaining to implement other blocks in the receiver chain.

3.3 Software on Tensilica Configurable Processor

The Tensilica Xtensa configurable processor is a RISC core which allows designers to customize certain aspects of the processor to tune it towards a particular application. A potential designer uses the Tensilica Processor Generator tool to configure a microprocessor. The designer has the ability to do such things as configure caches, configure memories, set up interrupt mechanisms, add datapath units such as a multiplier or MAC, and add Tensilica Instruction Extension (TIE) instructions. Based on the configurations, a compiler and instruction set simulator are generated. The compiler and instruction set simulator are used to determine the performance of the applications on the described architecture. Based on the feedback from the simulator, the designer can update the architecture and restructure the C code. The process continues until the designer is satisfied with the results.

FIXME show flow picture?

3.3.1 Implementation without TIE (CP0)

Implementation details

Observations

Summary & Results

3.3.2 Implementation with TIE (CP3)

Implementation details

Observations

Summary & Results

3.4 ASIC Soft Core Hardware Implementation

Introduction

This section describes the ASIC implementation of the Viterbi decoder. This implementation is based on the Mentor Viterbi soft core. Some of the features include:

- Flexible coding parameters: code rate, constraint length, traceback depth, code generating functions, soft decision word length
- Two orders of magnitude flexibility in speed and area: full parallel for high speed, and parameterized resource sharing for area efficiency
- Robust algorithmic implementation: self-normalizing traceback and saturation arithmetic in ACS computation
- RTL model optimized for high quality synthesis results
- Additional features (which can be disabled): channel bit error rate monitoring, Viterbi encoder, and synchronization monitor

The items provided as part of the soft core were:

- Bit-true cycle-based C and VHDL behavioral models
- Synthesizable and simulatable VHDL models with simulation scripts for Mentor VHDL simulator (which did not work for Synopsys VHDL simulator however)
- Self verifying test bench
- Synopsys synthesis scripts
- User's guide

3.4.1 Implementation details

Functionality verification and bit error rate (BER) simulations

As part of the Viterbi decoder soft core deliverables, three functionally equivalent models are provided: a synthesizable VHDL RTL model, a behavioral bit-true VHDL model, and a bit-true C behavioral model. The equivalency of the models for the particular decoder parameters of interest was verified through C and VHDL (modelsim) simulations, while the simulation times for the three models differ significantly (the behavioral VHDL simulation was about 10 times slower than behavioral C simulation, and the VHDL RTL simulation was about 1,000 times slower). The C behavioral model was used for BER simulations. It is a true cycle based implementation offering the highest simulation speed. The decoder is implemented using integer arithmetic with identical precision and rounding algorithms as the RTL model. The figure below shows the BER against SNR (signal to noise) curves for both "ideal" (D = 100 and floating point or swidth = 11 with modulo arithmetic in the ACS unit) and Mentor core (D = 48 and swidth = 9 with normalization) cases. The SNR degradation is about 0.15 dB.

FIXME FIGURE HERE

The synthesized net list of the Viterbi decoder is simulated and the gate level simulation results are compared with those form behavioral simulation to verify the functionality of the synthesized hardware (i.e. no synthesis related errors were introduced and timing constraints are met under target condition). After place and route, a transistor level net list with parasitic capacitance can be extracted and final verification could be done using Epic tools.

Technology

The technology used to implement this core was a $0.25\mu m$ six-metal layer process with a good standard cell library from ST Microelectronics.

Place and Route

The place and route of the Viterbi SRAM macro cells and standard cells was done in Cadence. Silicon Ensemble was used for routing. Parasitics for timing simulation were extracted from the final placed and routed nets in Silicon Ensemble.

There were significant problems in attempting to route the chip without routing violations. The smallest number of routing violations observed was 6, despite trying a large chip area of 12 mm^2.

The routing congestion appears to be worst at the 16 by 64 bit SRAM outputs - an SRAM cell design with pins spread over a wider length might solve the problem. Previous experience has found problems in Silicon Ensemble when routing pins not spaced at least 2 um apart, as wires are restricted to being 1 um apart, and vias to higher metal layers are quite large. Thus, the design analysis proceeded despite the unrouted nets, under the assumption that these nets would be routable given a different SRAM layout.

Different areas were tried, along with different placements (with width x height in um):

- chip area 2.5 mm2, 1575x1560: about 4700 violations
- chip area 3.2 mm2, 1803x1790: 57 violations
- chip area 3.6 mm2, 1850x1950: 57 violations, 64 violations with external pin locations changed
- chip area 4.0 mm2, 1850x2150: 64 violations, 77 violations with one 64 bit SRAM macro cell above the other two (significant routing congestion between all three) - in both placements the 64 bit SRAMs were vertical
- chip area 4.5 mm2, 1850x2450: 78 violations initially, reduced to 15 violations, then 9 violations with better placement
- Final placement, chip area 4.0 mm2, 1250x3200: 9 violations, 64 bit SRAMs were placed sideways next to each other

3.4.2 Observations

SRAM Simulations

Assumptions:

- Clock Speed is 66 Mhz
- 1000 iterations of activity (read, write, both)
- Voltage 2.5 V

```
1 ns set up time
2 ns hold time
1.8 ns to read address after rising clock edge


1 ns set up time
2 ns hold time
5 ns to read address after rising clock edge
```

Operations	Average Current (μA)	Average Power (mW)	Average Energy per Operation (pJ)
All Read	664	1.66	24.9
All Write	563	1.41	21.1
Random R/W	612	1.53	23.0

Table 3.1: Smaller ACS SRAM (16 by 8 bits) Power Simulation, Without Parasitics.

Operations	Average Current (μA)	Average Power (mW)	Average Energy per Operation (pJ)
All Read	2170	5.43	81.4
All Write	1890	4.73	71.0
Random R/W	2090	5.22	78.3

Table 3.2: Larger ACS SRAM (16 by 64 bits) Power Simulation, Without Parasitics.

```
1 ns set up time
2 ns hold time
1.9 ns to read address after rising clock edge
```

```
1 ns set up time
8 ns hold time
5 ns to read address after rising clock edge
```

Power and Timing Analysis

The capacitance per net was reported from the router (Silicon Ensemble) using the `report simcap` command. From this extraction, a Design Compiler capacitance annotation script was generated. Switching activity was measured from a 16000 cycle (1000 symbol) gate level VHDL simulation, and this was given to Design Compiler for annotation as well.

The timing analysis was divided into two parts, the analysis of paths which do not include an SRAM and the paths which do include an SRAM. Among all paths which do not include an SRAM, the longest was found to have a delay of 8.65 ns before parasitic annotation, and 16.7 ns after annotation. Thus the maximum operating frequency is 60 MHz. As expected, this critical path was through an ACS unit.

The longest path which passes through an SRAM was found to have a delay of 11.3 ns after parasitic extraction. Adding in an SRAM delay of 1.8 ns and a setup time of 1.0 ns, the path delay becomes 13.1 ns for this case. Thus the SRAM is not in the critical path.

Power numbers for gate designs (i.e. not including the SRAM) were found using the `report_power` command. Table 3.5 shows these results. The power used for the

Operations	Average Current (μA)	Average Power (mW)	Average Energy per Operation (pJ)
All Read	950	2.37	35.6
All Write	773	1.93	29.0
Random R/W	851	2.13	31.9

Table 3.3: Smaller ACS SRAM (16 by 8 bits) Power Simulation, Wit Parasitics.

Operations	Average Current (μA)	Average Power (mW)	Average Energy per Operation (pJ)
All Read	2480	6.21	93.2
All Write	2680	6.69	100.0
Random R/W	2570	6.44	96.5

Table 3.4: Larger ACS SRAM (16 by 64 bits) Power Simulation, With Parasitics.

SRAM is shown in Table 3.6. Operation is assumed at 2.5 V and 60 MHz.

Power	Without Annotation (mW)	With Switching Activity (mW)	With Capacitances and S.A. (mW)
Cell	28	20	20
Net	15	6	9
Total	43	26	29
Leakage	750 nW	810 nW	810 nW

Table 3.5: Summary of Power Compiler Results (Decoder Gates only)

Some of the blocks in this design would not be implemented in an actual Viterbi decoder implementation, for example the BER monitor, scrambler, etc. The `report_power` command was used on each sub-block of the design (without any back annotation), and it was found that 11% of the gate power was used by these blocks. Thus the final power figure is (29 mW x 89%) + 7.9 mW + 18 mW = 50.6 mW.

EPIC Powermill Simulation

FIXME results questionable?

Memory Module	Element Count	Energy Per Operation (pJ)	Measured Switching Activity	Total (mW)
9x8 ACS Mem. (Small SRAM)	16	31.9	0.5	7.9
64x16 Traceback Mem. (Large SRAM)	3	96.5	0.94	18

Table 3.6: Summary of Memory Module Power

Scaling

In order to take advantage of the fact that the implemented Viterbi decoder soft core can achieve much higher sample rate with 2.5 V supply voltage than required, voltage scaling is applied to trade in the "extra" performance for low power consumption. Since the standard cell library is characterized only under 2.5 V, all the scaling is based on the two figures below, which are critical path delay vs. supply voltage and energy-delay-product (EDP) vs. supply voltage. The figures are based on results from HSPICE simulations of a 16 bit ripple adder design (with parasitic capacitance) using the same 0.25 um technology. The voltage range under consideration is from 0.8 V to 2.5 V.

FIXME FIGURE HERE

Different supply voltage could be chosen depending on figure of merit of the design, for example, chose 2.5 V for highest performance, 0.8 V for lowest energy/power consumption, and 1.25 V for lowest energy-delay-product. The table below lists the scaled performance for the implemented Viterbi decoder, where clock rates are chosen to be approximately 1 / (critical path delay).

Optimized for	Supply Voltage	Clock Rate (MHz)	Symbol Rate (Msps)	Energy-Delay Product (fJs)	Power (mW)
Performance	2.5	60	3.75	3.60	50.6
Power	0.8	7.46	0.47	2.96	0.64
EDP	0.8	7.46	0.47	2.96	0.64

Table 3.7: Performance Results for Variable Clock Frequency

3.4.3 Summary & Results

```
Performance     3.75 Msps
Power           50.6 mW (2.5 V, 60 MHz)
Area            4.0  mm^2
Design Time     30 designer days
```

Chapter 4

Pulling it all Together

4.1 Introduction

4.1.1 Methods for comparison, scaling issues, approximations used, etc.

4.2 Summary of Results & Conclusions

Chapter 5

References

FIXME

Chapter 6

Appendices

Code, etc.

FIXME