A Multiprocessor Viterbi Decoder Using Xilinx FPGAs

David Yeh, Paul Chow, Gennady Feygin
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 1A4

1 Introduction

Before the introduction of FPGAs, attempts to build custom chips were costly, time consuming and no guarantees could be made on the functionality of the final product. However, with the introduction of FPGAs, prototyping time and cost have been reduced. Moreover, the reprogrammability of FPGAs allows designers to quickly repair errors in the design of a full system and permit the replacement of the FPGAs with ASICs once bugs in the system are removed, thus removing the need to redesign PC boards. As an added bonus, any systems developed with FPGAs may be reused by similar systems for future work. With all these advantages, FPGAs are ideal for our project and any other product that have cost and/or time constraints.

The project is to build a multiprocessor Viterbi Decoder using FPGAs. Viterbi Decoding is an error correcting algorithm that provides a maximum-likelihood decoding scheme for convolutional codes [1]. It attempts to find a path through a state transition diagram corresponding to the most likely transmitted data sequence using information from previous sequences.

1.1 Goals

The goals of this project are:

1) To build a demonstration system of the Galileo rate 1/4, K=14 Viterbi decoder [2] using FPGAs instead of custom ICs for the 26 processors, and attain data rates comparable to the present design (134.4 Kbits/s)

2) To get experience in designing large FPGA systems, and push for the highest speed possible.

2 Overview of Project

The system will be comprised of FPGAs organized to form a ring of 13 switches and 13 processor chips [3] as shown in Figure 1. For each pair of processor chips, there is a single RAM controller controlling the RAM associated with each processor. All switches and processor chips are implemented with the XC4010 FPGA. Ram controllers are realized with the XC4005 FPGAs.

This paper will be concerned primarily with the implementation of the processor chip and the techniques used to optimize its performance.

3 Processor Chip

The core of the implementation is the processor chip. It's function is to calculate the 2^{14} states required for each stage of the input. To improve throughput, a dual ring architecture has been implemented, meaning that each processor chip contains two processing elements (PE). Each PE takes a pair of 14-bit values, adds them to a pair of 10-bit values generated by the Branch Metrics Generator (BMG), compares the two

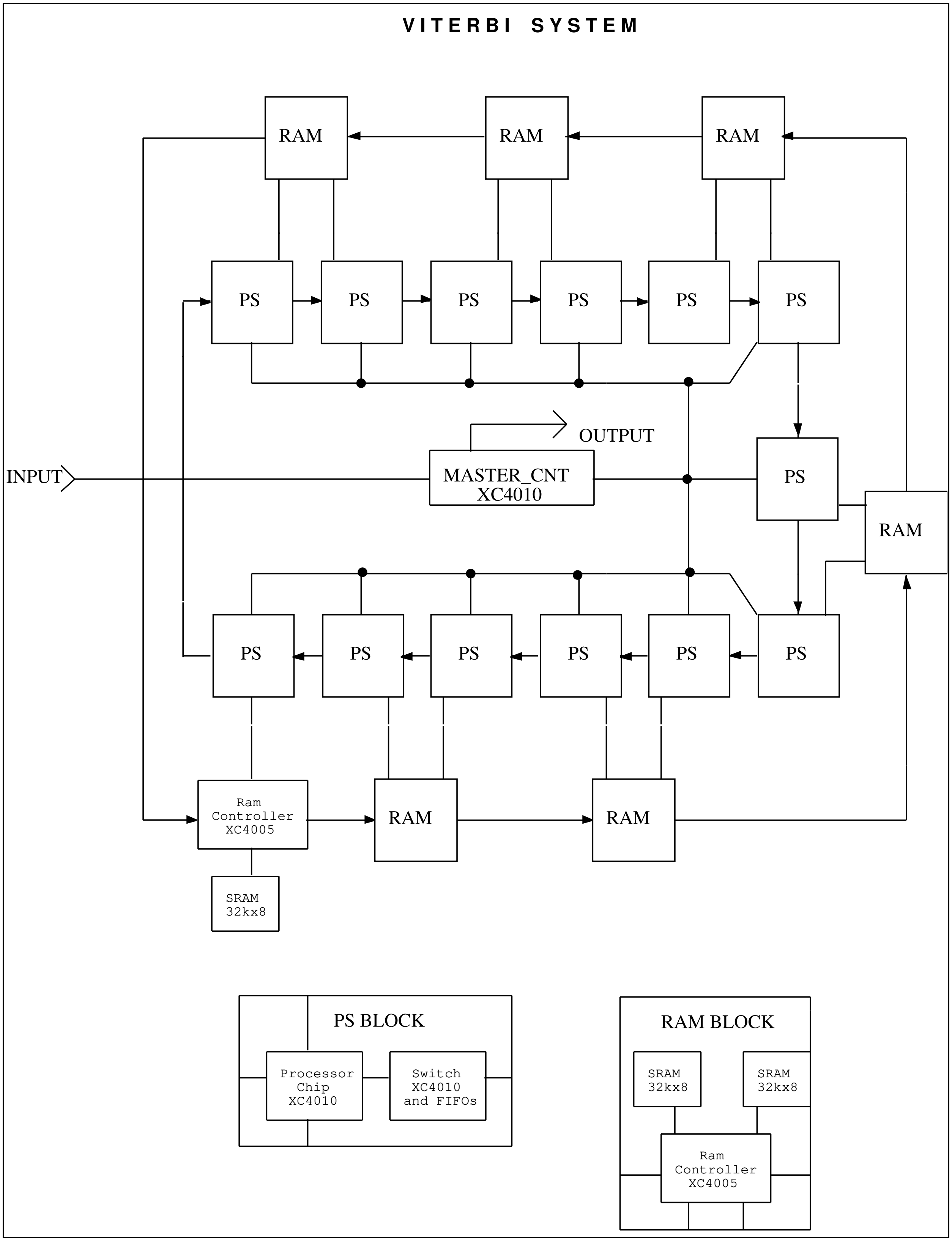


Figure 1: Viterbi System

results and selects the smallest value. This addition, comparison and selection, is also known as the Add-Compare-Select (ACS) portion of the PE. The basic structure of the processor chip is shown in Figure 2, and the pipelining of the BMG and ACS portion of the chip is shown in Figure 3. Note that there are two processors running in parallel inside the chip.

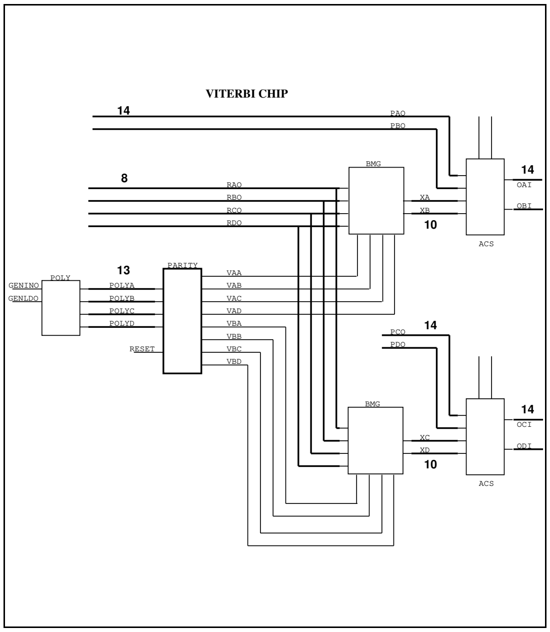


Figure 2: Chip Structure

The processor chip utilizes over 80% of a Xilinx XC4010 as shown in Table 1.

96% utilization of I/O pins	(154 of 160)
81% utilization of function generators	(646 of 800)
86% utilization of CLB flip-flops	(688 of 800)

Table 1: Utilization of XC4010 for Processor Chip

There are many difficulties that arise from the high utilization of an FPGA. The primary problem encountered in this particular design was the ability to place and route the entire chip. As can be seen, the processor chip is mostly datapath in nature, and it would be best if the logic in the datapath were placed in a regular manner as would be done in a custom VLSI implementation. We continue with a discussion of the techniques used to fit the circuit within a single XC4010.

3.1 Using the Pad Flip-Flops

Pipelining the chip effectively requires a large number of flip-flops. There are two flip-flops in each configurable logic block (CLB) for a total of 800 CLB flip-flops in the XC4010 FPGA. There are also 160 flip-flops in the 160 pads of the XC4010 FPGA. These pads can also be used as pipeline registers, reducing the need for using the CLB flip-flops.

3.2 Using Hard-Macros in Datapaths

If the datapath were specified as a netlist of logic, such as would be derived from a schematic definition, then the netlist would be given to the place-and-route software to try and find a logic mapping, and placement for each CLB. However, due to the nature of place and route algorithms, the result will be very random in nature, losing all of the regular structure that is available because the function is actually a datapath. This significantly impacts the density of logic that can be put inside the FPGA.

To try and preserve the regularity of the datapath, we devised a different strategy. Instead of allowing the circuit to be decomposed into functions that are mapped at the CLB level, we defined macros that

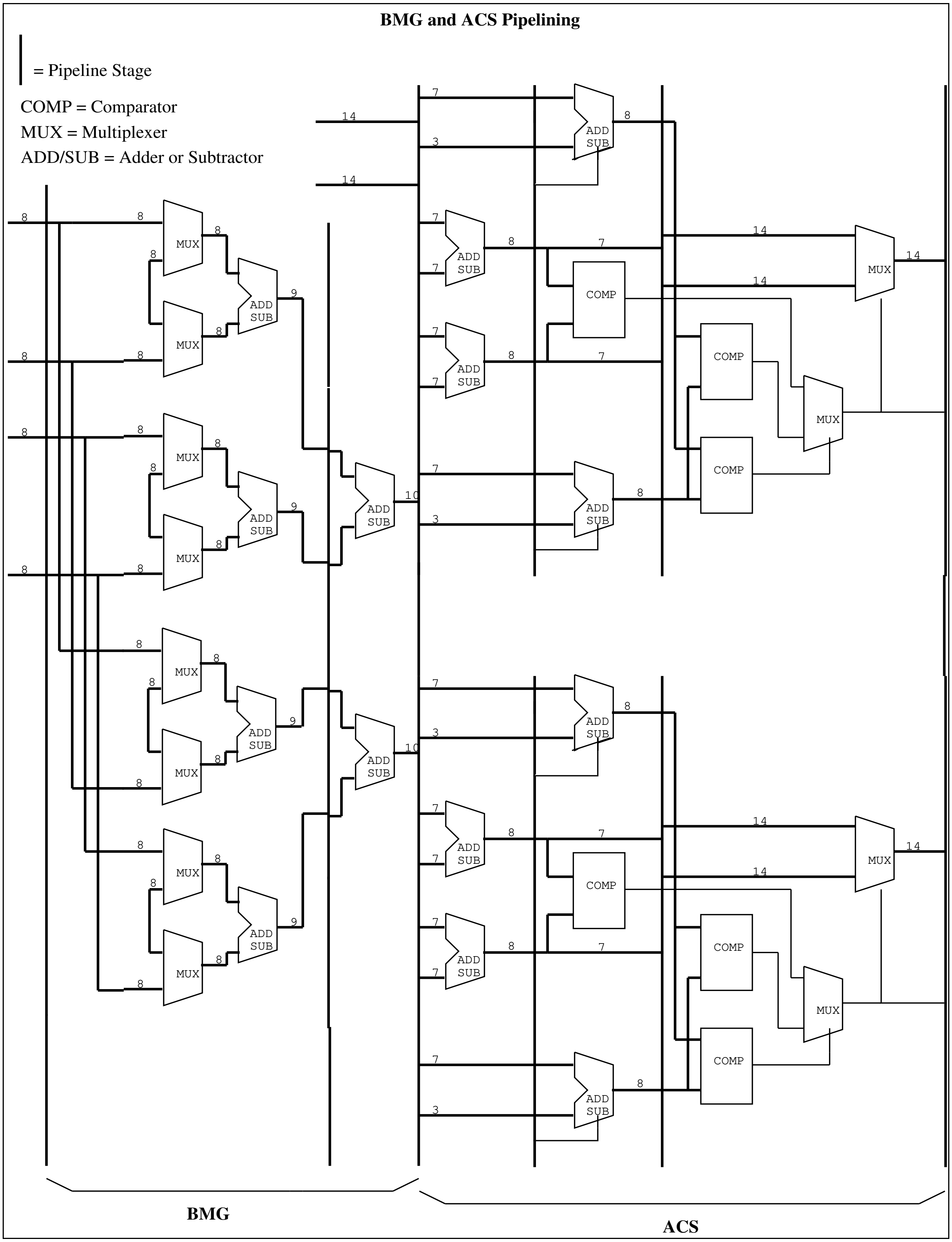


Figure 3: Chip Pipelining

consisted of functional slices of the datapath. Hard-macros are good in this case since they contain complete functions that have a predetermined mapping to the CLBs. The use of macros parallels how a custom VLSI datapath might be built. For example, one slice might be a complete nine-bit adder, and another might be an eight-bit wide set of 2:1 multiplexers with registered outputs. Once the hard-macros have been defined, they are manually placed exactly in the order they are to appear in the datapath. This leaves the router little choice in that the only routing necessary is to go from one hard-macro to the immediately neighbouring hard-macro, maximizing the usage of the available resources. Random control logic was allowed to be placed and routed automatically.

Two ways of manually placing macros can be used: either by schematic entry, where the placement is defined on a drawing, or by an external file to be read by the place and route program. An external file was preferred since modifications to placement could be performed via a text editor rather than direct editing of the schematic. For this particular design, a file with the extension of .cst was made to be read by the *ppr* (Xilinx placement and routing) program [4]. Figure 4 shows the placement of the macros. The grey area is one PE and the white area is the other.

3.3 Macro Choices

There were several factors considered when deciding which type of macro to use. Soft-macros, hard-macros, and XBLOX macros are the types provided by Xilinx. Soft-macros were used the least since even though there is a predefined structure, the

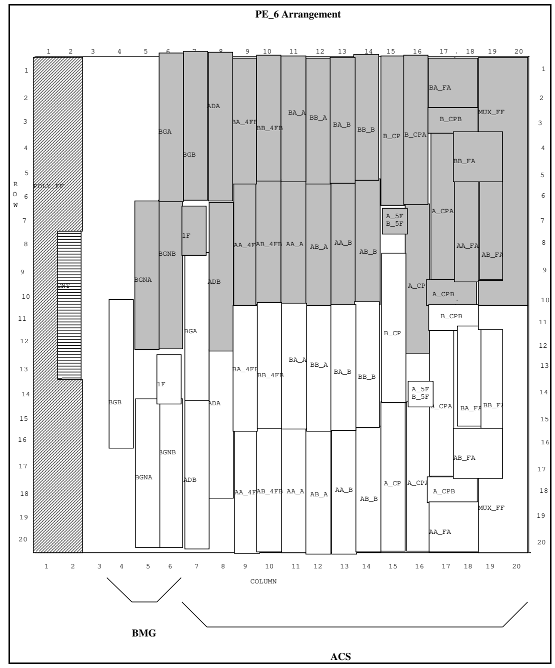


Figure 4: Placement Layout

CLBs within the structure are not fixed, leading to an unpredictable placement. An attempt was made to use hard-macros as described in Section 3.1, but a problem due to the predefined placement and routing structure was encountered. Any unused resources such as flip-flops and CLBs are unusable. Moreover, the hard and soft-macros provided have fixed bus widths, which are not ideal for efficiently creating a 14-bit data path when most components are given as eight-bits. This is where XBLOX has advantages [5].

XBLOX is a general macro program that contains both hard and soft-macros. What makes XBLOX unique is that it creates hard-macros, merges resources where possible, and has variable bus widths. An example is shown in Figure 5. When using traditional hard-macros, extra CLB's are required to include the pipeline flip-flops. On the other hand, XBLOX is able to

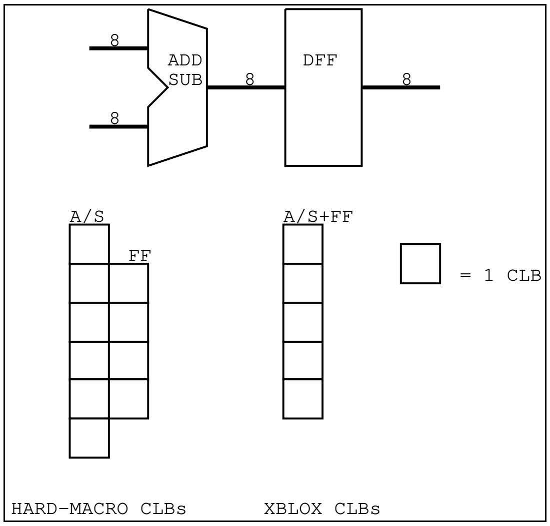


Figure 5: Example 1

combine the add/sub logic and flip-flops together to utilize the CLB resources as much as possible. As a result, XBLOX macros were used mostly for this design.

4 Other Tricks

Outlined below are a few tricks that may not be directly associated with the design of the processor chip, but could assist you in the design of future FPGA circuits.

4.1 Flip-Flop Placement for Speed

The use of pad flip-flops within a system could very easily save many CLB flip-flops if used correctly; however, one drawback is the placement of these pad flip-flops. If you are considering high speed operations, pad flip-flops may not be appropriate since they may not be close enough to the intended CLB; thus, increasing the propagation delay from the flip-flop to CLB. In such cases, it would be appropriate to use CLB flip-flops as opposed to pad flip-flops, since the delay from the CLB flip-flop to the destination CLB may be smaller than the delay from the pad flip-flop to the destination CLB.

4.2 XBLOX Macro and Flip-Flop Merging Techniques

The ability for XBLOX to merge logic and flip-flops within the hard-macro seems simple; however, XBLOX is not always intelligent enough to incorporate every flip-flop. For example, referring to Figure 6, case (a) has the output of the adder and its carry out merged into one bus and then followed by a pipeline stage. In this case, XBLOX will not combine any of the flip-flop information. In case (b), the bus and carry out have been defined separately. In this situation, everything but the carry out flip-flop will be merged into the hard-macro. A solution to this problem is to increase the bus width by one as shown in case (c). This allows the XBLOX software to combine the flip-flops with the adder without increasing resource usage.

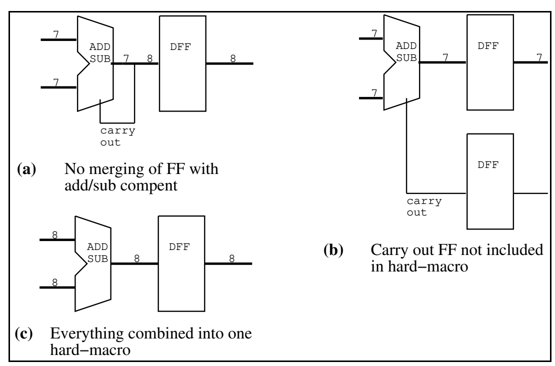


Figure 6: Example 2

4.3 Back-Annotation Problems

An inherent problem with XBLOX is the difficulty in probing specific nets during simulation. For the XC3000 FPGAs, the back-annotation program *bax* was used to back-annotate nets; however, there is no such program documented for XC4000 FPGAs. In attempting to resolve this problem, Xilinx does have a program called *xnfba* that does a better job at back-annotation for the XC4000 FPGA.

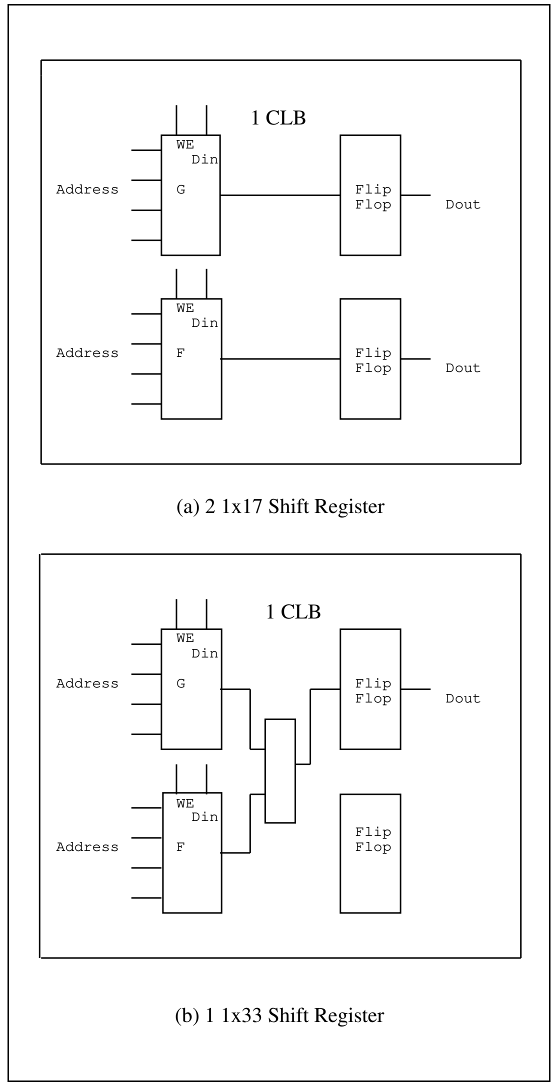
4.4 RAM Based Shift Registers

In some designs, it is desired to incorporate one or more shift registers. If there are a lot of flip-flop resources available, it is possible to design a small shift register; however, as the shift register's depth and width increase, it may be more desirable to make a RAM based shift register, as mentioned in the Xilinx Data Book [6].

When using XBLOX, the width and depth of the SRAM can be varied. However, for optimal speed and use of resources, pipelining of the shift register is required. By using SRAMs with a depth of 16 followed by a pipeline stage, optimal speed and use of resources are achieved. From Figure 7, the F and G function generators are both 1x16 RAM, followed by flip-flops. For example, to build a 2x32 shift register, using two CLBs of type 7(a) in series would be faster than using two CLBs of type 7(b) in parallel. This is because the first case is pipelined and does not require the extra delay in selecting the correct output as opposed to the second case.

5 Results and Conclusions

Simulated results for the XC4010-6 speed chips indicate that the processor can be clocked at 17MHz. Experience has shown that these simulated numbers are close to the actual speeds. This translates to



(a) 2 1x17 Shift Register

(b) 1 1x33 Shift Register

Figure 7:RAM Based Shift Register

17M ACS ops/s*2*13=442M ACS ops/s. and output at a rate of about 54Kbits/s. This is slower than the one built for Galileo [2] which ran at 134.4Kbits/s but that system required 256 identical VLSI chips compared to our 13 FPGAs for the maximum-likelihood decoding. Moreover, the FPGA system may be reprogrammed for other systems that may require the ring-like architecture of the Viterbi decoder.

The processor and memory system have actually been implemented on a field-programmable system called the TM-1 [7]. This is a board comprising four XC4010s, some memory and two APTIX [8] interconnect chips. The test ran at 10MHz and was limited by the clock distribution through the interconnect chips.

The Viterbi Decoder project using FPGAs has developed a set of techniques that allow high utilization of FPGA resources in datapath-dominated circuits. To accomplish such a task required extensive use of XBLOX to create hard-macros, and creative ideas in the placement of such macros. Other techniques such as using pad flip-flops helped to improve density.

6 ACKNOWLEDGEMENTS

This work could not have been done without the support of Micronet, NSERC, BNR, and Xilinx. Also, thanks to the Xilinx hotline support group (1-800-255-7778) for their timely help with figuring out some timing problems.

References

[1] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory,* IT-13(4): pages 260-269, 1967.

[2] Statman, J., Zimmerman, G., Pollara, F., and Collins, O. A Long Constraint Length VLSI Viterbi Decoder for the DSN. TDA Progress Report 42-95, Jet Propulsion Laboratory, Pasadena, California, 1988.

[3] Gennady Feygin, Patrick Glenn Gulak, and Paul Chow. A Multiprocessor Architecture for Viterbi Decoders with Linear Speed-Up. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, pages 2907-2917. IEEE, 1993.

[4] XILINX, Inc., 2100 Logic Drive, San Jose, CA 95124. *The XACT4000-Design Implementation Reference Guide,* pages 7.1-7.23, 1991.

[5] XILINX, Inc., 2100 Logic Drive, San Jose, CA 95124. *XACT-Design Interface-Macro Libraries*, pages 3.1-3.105, 1991.

[6] XILINX, Inc., 2100 Logic Drive, San Jose, CA 95214, *The Programmable Logic Data Book*, page 8-110 to 8-111, 1993.

[7] Dave Galloway, David Karchmer, Paul Chow, David Lewis, and Jonathan Rose. *The Transmogrifier: The University of Toronto Field-Programmable System*, to appear in 1994 Canadian Workshop on Field-Programmable Devices.

[8] Aptix Corporation, San Jose, California. *FPCB AXB-GP2 Data Sheet*, 1993.