A PROGRAMMABLE BASEBAND PROCESSOR DESIGN FOR SOFTWARE DEFINED RADIOS

Sridhar Rajagopal, Scott Rixner, and Joseph R. Cavallaro

Computer Systems Laboratory
Department of Electrical and Computer Engineering
Rice University, Houston, TX 77005
{sridhar,rixner,cavallar}@rice.edu

ABSTRACT

Future wireless systems need extremely fast and flexible architectures to support varying standards, algorithms and protocols with data rates in the range of 10-100 Mbps. Software Defined Radios (SDRs) based on DSP-FPGAs are a widely proposed solution for these systems. However, these SDR solutions have not been able to meet real-time requirements. We propose a programmable architecture solution for SDRs using a stream-based architecture based on the *Imagine* media processor. The configurable *Imagine* simulator allows us to investigate issues such as memory bottlenecks, number and type of functional units needed, and the utilization of those functional units. To evaluate stream-based architectures for baseband processing, we parallelize and implement sophisticated baseband algorithms including multiuser estimation, multiuser detection and Viterbi decoding on this simulator. We present the bottlenecks in such a stream-based architecture for efficient communications processing. Comparisons with current generation DSP-based solutions show orders-of-magnitude performance improvements, both due to the stream-based nature of computations as well as the increase in the number of functional units having a high utilization factor. The result is a baseband processor designed with broad system functionality and flexibility that approaches real-time performance for future wireless systems.

1. INTRODUCTION

Next generation wireless systems [1, 2] are being designed to provide a wide variety of multimedia services and to seamlessly switch between different wireless standards, such as wireless LAN and wideband CDMA. Each of these standards require different physical layer algorithms to be implemented. Also, algorithmic parameters such as the coding rate and constraint length for decoding need to be configured based on the channel environment. The wide range of configuration parameters and flexibility in the choice of algorithms to be implemented motivates the need for a software defined radio (SDR) solution.

Figure 1 shows the number of adders and multipliers theoretically needed to meet a real-time data rate of 4 Mbps (aggregate) for a W-CDMA cellular system. The number of adders and multipliers needed depends on the environment and the frequency of estimating the channel. Most SDR solutions are based on DSP-FPGA implementations and are usually a prototyping or a proof-of-concept effort [3–5]. From Figure 1, a 32-user CDMA system requires around 15 additions and 15 multiplication operations per cycle (with a 500 MHz clock), to meet real-time of 128 Kbps/user

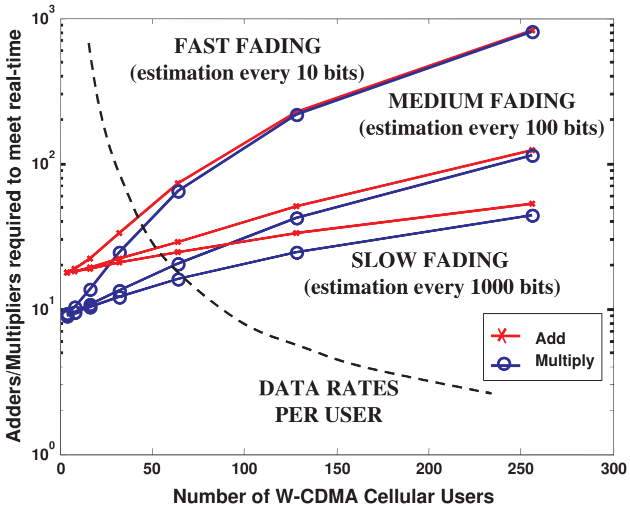

Figure 1: *Number of adders and multipliers needed to achieve 4 Mbps (aggregate), assuming a 500 MHz clock.*

even for a slowly fading channel. This is assuming that the functional units in the programmable processor are operating at 100% efficiency and hence, in a practical system, around 20-25 adders and multipliers may be needed to meet real-time requirements. A programmable DSP-based SDR does not have enough functional units to meet the targeted data rates. ASIC and FPGA support are often used for implementing these systems in real-time. Newer trends in SDR solutions include re-configurable computing based solutions for wireless systems such as Stallion and Chameleon [6, 7]. However, these solutions require significantly higher programming effort and have been unable to achieve these extremely high data rates.

Figure 2 shows a SDR with a proposed communications processor that does the frontend base-band processing in real-time and acts as the interface between the RF and the higher MAC and network layers in the mobile device. We use a streaming processor simulator, based on the Imagine architecture [8], as the tool to investigate the communications processor design. An end-to-end suite of key sophisticated algorithms modeling a future W-CDMA system is implemented on the simulator to study the computational workload and its characteristics. We present the bottlenecks present in the base Imagine implementation that need to be overcome for a stream processor architecture to perform effi-

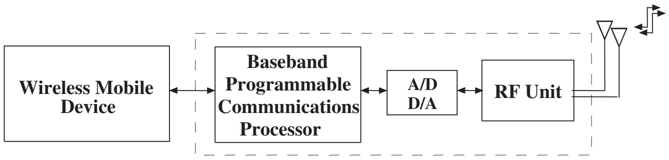

Figure 2: *A programmable communications processor for SDR. The figure shows a SDR having a communications processor to do the frontend (physical layer) baseband processing.*

cient communications processing. Comparisons with SDR solutions based on current generation DSPs show orders-of-magnitude performance improvements due to the larger number of functional units available and their efficient utilization.

2. THE IMAGINE ARCHITECTURE AND SIMULATOR

Imagine is a high performance media processor built at Stanford [8]. We choose to explore stream processors based on the Imagine architecture because many workload characteristics for communications such as FIR and FFT are similar to media processing. The Imagine simulator allows us to simulate high performance architectures with great flexibility without the added baggage of caches, branch prediction units and out-of-order schedulers present in conventional general purpose simulators which are not useful for communications processing. The base Imagine architecture is shown in Figure 3. It has 8 VLIW-based computational clusters arranged in a SIMD fashion. Each cluster contains multiple functional units. A large general purpose stream register file (SRF) forms the heart of the chip and is connected to the clusters, the memory system and the network. The stream register file is program-controlled and serves as a storage area for data used by other units. The number of memory accesses are minimized by keeping frequently used data in the SRF. The Imagine memory system allows multiple streaming accesses simultaneously and provides enough memory bandwidth for computational units. The chip is controlled by a host processor and issues commands via a stream controller. The architecture details can be obtained from [8].

The Imagine simulator is a cycle-accurate simulator that gives us insight on the operations being performed in functional units, register files and memory of the processor every clock cycle. The simulator allows us to vary parameters such as the number and type

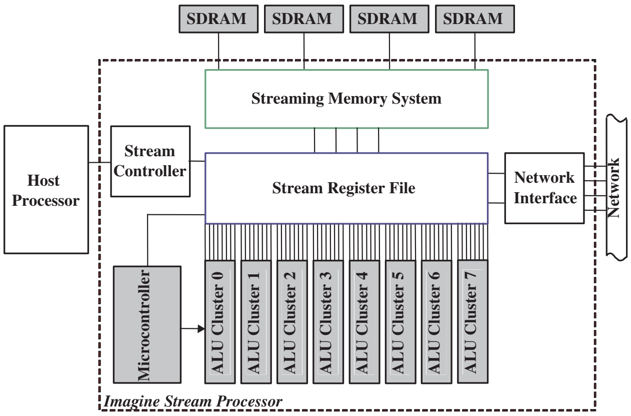

Figure 3: *The Imagine stream processor architecture.*

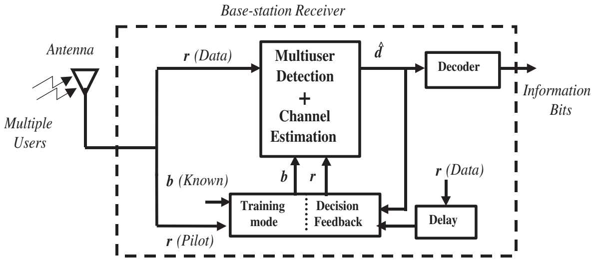

Figure 4: *W-CDMA algorithm suite studied for implementation*

of functional units, clusters, register sizes and memory and study the effects of these variations on the algorithms. New blocks such as special-purpose functional units can also be integrated into the simulator. Graphical tools are available that show the algorithm schedule, including functional unit utilization and memory bottlenecks.

The Imagine processor is currently programmed in a two-level fashion using C++. The computations of the algorithms are scheduled in the eight clusters and arranged as kernel operations and are written in C++ (called KernelC). The communication between the different kernels and between the main memory and SRF is then handled using another C++ code (called StreamC).

3. ALGORITHMS

A proposed end-to-end W-CDMA based base-station receiver that implements key sophisticated algorithms such as multiuser channel estimation, multiuser detection [9] and Viterbi decoding [10] is shown in Figure 4. The algorithms proposed are highly parallelizable, use only multiplications and additions and show good fixed-point behavior [9].

The channel information is obtained by the transmission of a pilot signal, $\mathbf{b}(Known) \in \{-1,+1\}^{2K}$, which is a sequence of bits that are known at the receiver (training sequence). The received pilot signal, $\mathbf{r}(Pilot) \in \mathbb{C}^N$, is compared with the known bits to form an estimate of the channel. However, the pilot sequence may not be available continuously [11]. The channel estimates may need to be updated once for every 1000 bits of detection for a slow fading channel to around once every 10 detection bits for a fast fading channel. In this scenario, the decisions from the multiuser detection block, $\hat{\mathbf{d}}$, are fed back to the channel estimation block along with the received data bits, $\mathbf{r}(Data)$, delayed by the time required for detection, for tracking the channel estimates when the pilot signal is absent.

The derivation of the joint multiuser estimation and detection algorithm chosen for implementation is detailed in [9]. Channel estimation consists of the following computations:

$$\mathbf{R}_{br}^{(i)} = \mathbf{R}_{br}^{(i-1)} + \mathbf{b}_i\mathbf{r}_i^H - \mathbf{b}_{i-L}\mathbf{r}_{i-L}^H \quad (1)$$

$$\mathbf{R}_{bb}^{(i)} = \mathbf{R}_{bb}^{(i-1)} + \mathbf{b}_i\mathbf{b}_i^T - \mathbf{b}_{i-L}\mathbf{b}_{i-L}^T \quad (2)$$

$$\hat{\mathbf{A}}^{(i)} = \hat{\mathbf{A}}^{(i-1)} - \mu(\mathbf{R}_{bb}^{(i)} * \hat{\mathbf{A}}^{(i-1)} - \mathbf{R}_{br}^{(i)}) \quad (3)$$

where $\mathbf{R}_{br} \in \mathbb{C}^{2K \times N}$ is the cross-correlation matrix between the synchronization bits $\mathbf{b}_i$ and the received signal $\mathbf{r}_i$, and $\mathbf{R}_{bb} \in \mathbb{R}^{2K \times 2K}$ is the autocorrelation matrix. The channel estimate, $\mathbf{A} \in \mathbb{C}^{2K \times N}$, is obtained by an iterative method, suitable for fading channels. The matrix $\mathbf{A}$ is rearranged into its odd and even columns

$\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{C}^{K \times N}$ which corresponds to the data bits $\hat{\mathbf{d}}_{i-1}$ and $\hat{\mathbf{d}}_i \in \{-1, +1\}^{2K}$ in the estimate. L is the length of the preamble window which is dependent on the amount of fading in the channel. Typical values [11] for N, the spreading code length, and K, the number of users are 32 each.

The computations used in the detection phase are as shown:

$$\mathbf{L} = \Re[\hat{\mathbf{A}}_1^H \hat{\mathbf{A}}_0] \quad (4)$$

$$\mathbf{C} = \Re[\hat{\mathbf{A}}_0^H \hat{\mathbf{A}}_0 + \hat{\mathbf{A}}_1^H \hat{\mathbf{A}}_1] - diag(\hat{\mathbf{A}}_0^H \hat{\mathbf{A}}_0 + \hat{\mathbf{A}}_1^H \hat{\mathbf{A}}_1)] \quad (5)$$

$$\mathbf{y}_i^{(0)} = \Re[\mathbf{A}_1^H \mathbf{r}_{i-1} + \mathbf{A}_0^H \mathbf{r}_i] \quad (6)$$

$$\hat{\mathbf{d}}_i^{(l)} = sign(\mathbf{y}_i^{(0)}) \quad (7)$$

$$\mathbf{y}_i^{(l)} = \mathbf{y}_i^{(0)} - \mathbf{L}\hat{\mathbf{d}}_{i-1}^{(l-1)} - \mathbf{C}\hat{\mathbf{d}}_i^{(l-1)} - \mathbf{L}^H \hat{\mathbf{d}}_{i+1}^{(l-1)} \quad (8)$$

$$\hat{\mathbf{d}}_i^{(l)} = sign(\mathbf{y}_i{}^{(l)}). \quad (9)$$

Equation (8) performs parallel interference cancellation (PIC) and may be thought of subtracting the interference from the past bit of users having more delay, and the future bits of the users having less delay, than the desired user. The left matrix $\mathbf{L}$, stands for the partial correlation between the past bits of the interfering users and the desired user, the right matrix $\mathbf{L^H}$, stands for the partial correlation between the future bits of the interfering users and the desired user. The center matrix $\mathbf{C}$, is the correlation of the current bits of interfering users and the diagonal elements are made zeros to avoid self-cancellation. The $\mathbf{L}$ and $\mathbf{C}$ matrix calculations in equations (4),(5) need to be performed only once per channel estimation while the matched filter and PIC in equations (6),(8) needs to be performed for every detection bit.

The output of the multiuser detector is then fed to the Viterbi decoder [10]. The Viterbi decoder then decodes the bits for each of the 32 users in the system. The parameters of the decoder are also flexible and they can vary from a rate 1/2, constraint length 5 to a rate 1/3, constraint length 9 system [11] where the complexity increases exponentially with constraint length.

4. RESULTS

We implemented the above baseband algorithms on the streaming processor simulator to investigate their performance and functional unit utilization. We present the bottlenecks in stream-based processors that need to be overcome for efficient communications processing.

The various computation kernels and communication kernels for the algorithms are shown in Figure 5. Equations (1)-(2) form the 'correlation update' kernel. The matrix-matrix multiplication in equation (3) forms the 'matrix mul' kernel. The 'matrix mul' kernel is implemented as a series of matrix-vector multiplication kernels and hence, a loop is shown in the figure to signify repetitions. The remaining iteration update in equation (3) forms the 'iteration update' kernel. The data communication between these kernels is fairly regular until this point. However, the channel estimate matrix $\mathbf{A}$ is now rearranged into its odd and even columns $\mathbf{A}_0, \mathbf{A}_1$. This data rearrangement induces expensive memory accesses in order to rearrange the data, during which the computation units in Imagine remain idle, producing memory stalls on the Imagine processor. Similarly, the matrix transpose in equation (8) also requires data rearrangement and induces expensive memory accesses. The 'matrix mul L' and 'matrix mul C' kernels are obtained from equations (4) and (5). The matched filter 'MF' kernel

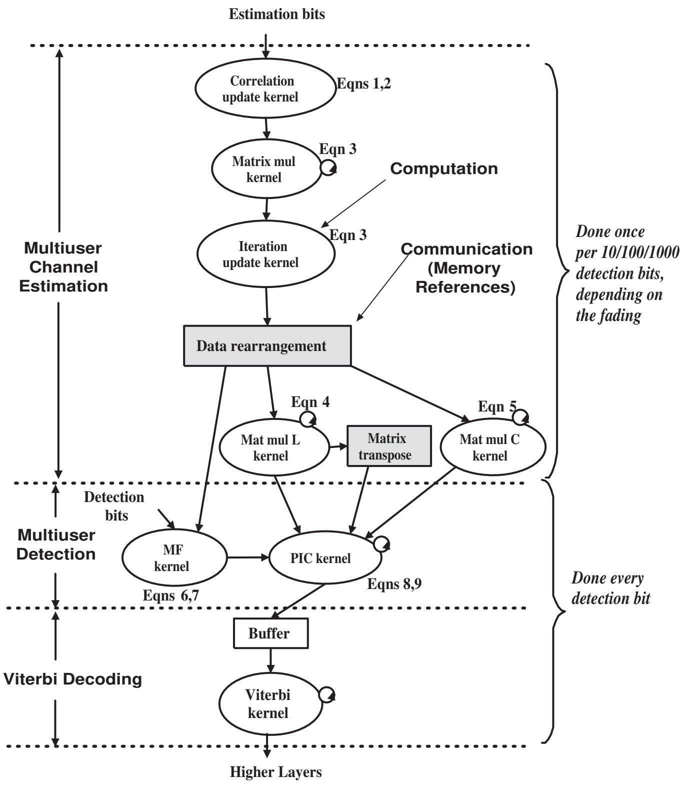

Figure 5: *An end-to-end W-CDMA system for SDR implemented on the streaming processor simulator.*

obtained from equation (6), takes the bits to be detected as input and produces a rough estimate of the result to the PIC stages where the interference from other users are canceled for a more accurate estimate. The loop around PIC kernel signifies that PIC is repeated 3 times for convergence after which the detected bit is sent to the Viterbi decoder for decoding each of the 32 users.

The performance of the base Imagine architecture is shown in Table 1. The base Imagine architecture has 8 clusters, each with 3 adders, 2 multipliers and 1 division unit. Since the algorithms are highly parallel, all 16 multipliers show high utilization of around 90-100% for most kernels. The 24 adders are, however, underutilized with 53-70% efficiency as they are waiting for the multiplications to complete. The division unit (not shown) is idle due to a lack of divisions in the implemented algorithms. From Figure 1, we saw that the total number of additions and multiplications in the kernels are approximately the same. This suggests replacing the division unit by an additional multiplication unit to have a 3 adder, 3 multiplier per cluster configuration as shown in Table 1. Having equal number of adders and multipliers gave significant speedup to most kernels due to increased adder efficiency. Since most kernels were compute intensive, we optimized the kernels to produce high functional unit efficiency at the cost of communication overhead between kernels. The sorting of the path metrics and survivor memory in Viterbi were performed within the kernel. This significantly reduced the functional unit utilization but still produced better performance due to the elimination of the communication between the trellis stages which required memory references. Viterbi(a) shows the performance of a rate 1/2 constraint length 5 Viterbi decoder and Viterbi(b) shows the performance of

Table 1: *Functional unit utilization of different kernels on the base Imagine architecture with 24 adders, 16 multipliers and 8 dividers and the modified architecture with 24 adders and 24 multipliers.*

Kernel	Base			Modified		
	Adder util	Mult util	Time (cycles)	Adder util	Mult util	Time (cycles)
corr update	70%	100%	1224	79%	78%	1064
matrix mul	53%	91%	22720	85%	99%	14360
iteration	55%	42%	1058	55%	28%	1058
mat mul L	59%	91%	7468	78%	84%	5573
mat mul C	63%	96%	12192	68%	71%	11084
mf	67%	100%	366	90%	89%	275
pic	67%	96%	996	89%	84%	760
viterbi(a)	13%	2%	8044	13%	1.4%	8044
viterbi(b)	35%	9%	80006	35%	6%	80006

a rate 1/3 constraint length 9 Viterbi decoder.

Comparisons with a TI C67 DSP are shown in Table 2. The DSP simulations were performed using the TI C67 simulator that assumes a flat memory model for external memory references. We observe a speedup of 48× for channel estimation and 42× for detection for our design using the stream processor simulator. The C67 comparisons for Viterbi decoding were obtained from [12]. The DSP shows better performance as the functional units in our design have very poor utilization. The real-time requirements allow only 4000 cycles per bit for a 500 MHz system. From Table 2, we see that excluding decoding, we can easily meet real-time for both the slow and medium fading cases. We are investigating reducing the memory stalls by exploring different techniques such as pipelining other kernels in the stall time and re-arranging data within kernels.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a programmable communications processor for SDRs that approaches the real-time system requirements. We implement key baseband algorithms for a W-CDMA based cellular system on the Imagine simulator to investigate its performance. We design kernels and functional units with high functional unit utilization for multiuser estimation and detection and we are investigating ways to minimize the memory references between the computations during the data rearrangement and during matrix transposes. The Viterbi implementation, though optimized to eliminate memory references, suffers from the reverse problem of having extremely poor functional unit utilization.

The Imagine simulator allows us to explore different variations in the architecture including the functional unit design and usage, number of clusters, memory references etc. for classes of scenarios. Our current research aim is to solve the bottlenecks posed by the algorithms to design a real-time programmable communications processor with high functional unit utilization and minimum communication overhead between kernels.

Table 2: *Comparison with a C67 DSP for different channels. The figure shows the number of cycles per bit taken by our design based on Imagine against the TI C67 DSP with perfect external memory.*

Stage	Slow Fading		Medium Fading		Fast Fading	
	Imagine	DSP	Imagine	DSP	Imagine	DSP
Est.	46	2228	459	22288	4594	222879
Det.	1035	44077	1035	44077	1035	44077
Dec.	80006	72736	80006	72736	80006	72736
Stalls	104	-	606	-	5620	-

6. ACKNOWLEDGMENTS

We would like to thank the Imagine group at Stanford, especially Ujval Kapasi, for timely help with their tools. This work was supported in part by Nokia Corporation, Texas Instruments Inc., the Texas Advanced Technology Program under grant 1999-003604-080, and by NSF under grants NCR-9506681 and ANI-9979465.

7. REFERENCES

[1] R. Berezdivin, R. Breinig, and R. Topp, "Next-generation wireless communications concepts and technologies," *IEEE Communications Magazine*, vol. 40, no. 3, pp. 108–117, March 2002.

[2] B. Aazhang and J. R. Cavallaro, "Multi-tier wireless communications," *Wireless Personal Communications, Special Issue on Future Strategy for the New Millennium Wireless World, Kluwer*, vol. 17, pp. 323–330, June 2001.

[3] J. Mitola, "The Software Radio Architecture," *IEEE Communications Magazine*, vol. 33, no. 5, pp. 26–38, May 1995.

[4] I. Seskar and N. B. Mandayam, "A software radio architecture for linear multiuser detection," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, pp. 814–823, May 1999.

[5] M. Cummings and S. Haruyama, "FPGA in the software radio," *IEEE Communications Magazine*, vol. 37, no. 2, pp. 108–112, February 1999.

[6] S. Srikanteswara, J. H. Reed, P. Anthanas, and R. Boyle, "A software radio architecture for reconfigurable platforms," *IEEE Communications Magazine*, vol. 38, no. 2, pp. 140–147, February 2000.

[7] B. Salefski and L. Caglar, "Re-configurable computing in wireless," in *Design Automation Conference*, Las Vegas, NV, June 2001, pp. 178–183.

[8] B. Khailany, W. J. Dally, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, A. Chang, and S. Rixner, "Imagine: media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, March-April 2001.

[9] S. Rajagopal, S. Bhashyam, J. R. Cavallaro, and B. Aazhang, "Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers," *IEEE Transactions on Wireless Communications*, vol. 1, no. 3, pp. 468–479, July 2002.

[10] H-L. Lou, "Implementing the viterbi algorithm," *IEEE Signal Processing Magazine*, vol. 12, no. 5, pp. 42–52, September 1995.

[11] "Third Generation Partnership Project," http://www.3gpp.org.

[12] G. Kang and P. Zhang, "The implementation of Viterbi decoder on TMS320C6201 DSP in W-CDMA system," in *IEEE International Conference on Communications Technology*, 2000, vol. 2, pp. 1693–1696.