J Sign Process Syst
DOI 10.1007/s11265-009-0402-z

Analyzing the Next Generation Software Defined Radio for Future Architectures

Mark Woh · Yuan Lin · Sangwon Seo · Scott Mahlke · Trevor Mudge

Received: 21 October 2008 / Revised: 23 January 2009 / Accepted: 23 July 2009

Abstract Commercial and research work in the field of software defined radio (SDR) has produced designs which have been able to deliver the efficiency and computational power needed to process 3G wireless technologies. Though efficient 3G processing has been achieved by these designs, next generation 4G SDR technology requires 10–1000x more computational performance but limits the power budget increase to 2–5x. In this paper, we present a breakdown of the major 4G kernels and analyze two methods of increasing performance and reducing power consumption. Specifically, we consider the effect of SIMD width and reduction in number of register file accesses on the performance and energy consumption of a SDR architecture, SODA. We show that by increasing SIMD width we can gain almost 2–8x performance increase while increasing total energy used by 1–2x for different SIMD widths. We also show that by reducing SIMD register accesses we can reduce the total energy used by 5–20% for the 4G kernels.

Keywords Software defined radio · Single instruction multiple data · 4G wireless

M. Woh (✉) · Y. Lin · S. Seo · S. Mahlke · T. Mudge
University of Michigan—Ann Arbor,
2260 Hayward Street, Ann Arbor,
MI 48109, USA
e-mail: mwoh@umich.edu

1 Introduction

Wireless communication has grown dramatically over the years. Accessing the web, downloading video, and listening to music is a growing demand with wireless users. Third generation wireless (3G) technologies have been able to provide people with access to these services. With the number of users increasing and the demand for higher quality content, the bandwidth needed exceeds what 3G can provide. Fourth generation wireless (4G) technology has been proposed by the International Telecommunications Union (ITU) [4] to increase the bandwidth to maximum data rates of 100 Mbps for high mobility situations and 1 Gbps for stationary and low mobility situations like internet hot spots. With this increase in bandwidth there will also be an increase in the number of computations needed to process this standard on software defined radio (SDR) systems.

Baseband signal processing for mobile terminals has been a computing challenge for computer architects. Several architectures have been successful at achieving the super computer like workloads of 3G wireless systems while maintaining the mobile device power budget. Though we were able to meet the requirements for 3G, the next generation 4G seems to be an even larger hurdle. With computational requirements increasing from 10–1000x compared to 3G and a power envelope that can increase by only 2–5x [19], we need even more efficient designs to complete these tasks. We need more powerful processors that consume less energy because device scaling is delivering less performance improvements and not reducing power consumption for existing designs.

In the past ITRS [3] had suggested that in future generations of process technologies, we would still be

Published online: 26 August 2009

able to scale frequencies higher. ITRS has been very optimistic with its predictions and manufacturers in the past have been able to meet these targets through material and process improvements. This continuation of process technology improvement is not sustainable anymore and current data shows that frequency and power are reaching a plateau [12]. The only benefit for changing technologies would be reduction in transistor size allowing us to pack more logic onto the same size die.

This slowdown in performance gain requires us to extract more computation with better architectural design. One way to extract more computational performance is by increasing the width of the datapath and exploiting more data-level parallelism (DLP) through the use of SIMD. Not only does this support increased performance, we can do it with little or no increase in energy. Though wider SIMD widths will consume more energy per operation, we can take advantage of the DLP and reduce the amount of control code and memory accesses thus reducing the total number of cycles and total energy overall. In our study, we analyze the effect that SIMD width has on computational performance and energy for 4G wireless systems. We will show the maximum possible DLP that can be extracted from each kernel and the percent of the algorithm which will benefit from exploiting the DLP.

Another way to extract more computational efficiency is by reducing the total energy consumed. If performance stays the same but energy is reduced the total power is also reduced. We found that register file (RF) access energy accounts for a large percentage of the total energy in many processors. In order to reduce this large percentage, we analyzed the RF access patterns within the kernels and found that many of the kernels have instructions which produce values which are consumed by the next instruction and never used again. By storing these values into a small temporary RF or forwarding the data without writing the value into the RF, we can save the wasted energy of writing and reading these values from the main RF.

This paper is organized as follows. In Section 2, we present the SDR architecture used for the SIMD width and RF access study. In Section 3, we present a simplified 4G system and describe some of the major kernels: an OFDM demodulator /modulator, a MIMO encoder/decoder, and a channel decoder. In Section 5, we analyze the kernels and show their potential DLP and also the instruction breakdown of the algorithm. We also present the effects that varying SIMD width has on the computation and energy efficiency. In Section 6, we analyze the RF access patterns in the kernels and we show the amount of energy saved by not writing back unneeded values into the main RF. The summary and concluding remarks are given in Section 7.

2 SDR Processors

Most current processor solutions for SDR utilize SIMD to exploit the large amounts of DLP. These include Infineon's MuSIC [7], Analog Device's TigerSHARC [9],

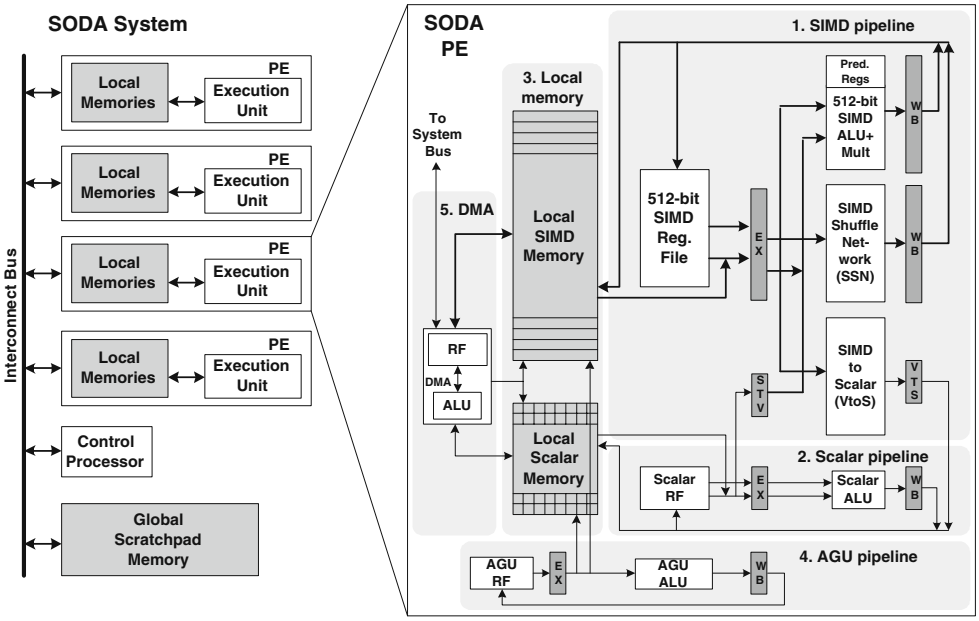


Figure 1 SODA architecture for SDR. The system consists of 4 data processing elements (PEs), 1 control processor, and global scratchpad memory, all connected through a shared bus. Each PE consists of a 32-wide 16-bit SIMD pipeline, a 16-bit scalar pipeline, two local scratchpad memories, an Address-Generation-Unit(AGU) for calculating memory addresses, and a direct-memory-access (DMA) unit for inter-processor data transfer.

Phillips's EVP [6], and Sandbridge's Sandblaster [10]. Though all these processors are SIMD processors, the widths for each are different. Recent work by [15], suggests building wide SIMD processors of up to 96 lanes in width to support the different error correction algorithms within SDR. For our work, we used the SODA processor [14] to explore the scalability of SIMD width and to analyze the potential for reduced register accesses.

A block diagram of the architecture is shown in Fig. 1. SODA is a control-data decoupled multi-core architecture. The SODA architecture consists of processing elements which uses a wide 512-bit SIMD unit that is capable of operating on 32 16-bit elements concurrently. SODA also has a non-uniform memory architecture, with local memories on the processing elements and a shared global memory. In SODA, RF accesses are the same as typical DSPs where each instruction accesses one or two registers from the RF and then write the results back into the RF. Even if the next instruction accesses the register value being written to, the data is written into the RF.

3 4G Wireless Kernels

Though there is no standard yet for 4G, Fig. 2 shows a high level block diagram of the physical layer of a NTT DoCoMo test 4G wireless system setup [18]. The major components of the physical layer consists of 3 major blocks: de/modulator, MIMO encoder/decoder, and the channel encoder/decoder. These blocks compose the majority of the total computation in 4G. The role of the modulator is to map data sequences into amplitudes and phases which then are converted to the time domain and transmitted. The demodulator performs the operations in reverse order to reconstruct the original data sequences. This is typically done by the Fast Fourier Transform (FFT) algorithm.

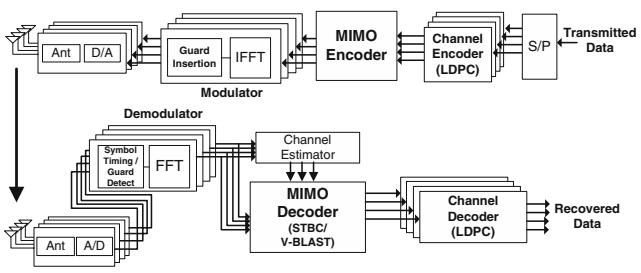


Figure 2 Block diagram overview of a 4G system.

The Multiple Input Multiple Output (MIMO) encoder multiplexes many data signals over multiple antennae. The MIMO decoder receives all the signals from the antennae and then tries to either decode all the streams for increased data rates or combine all the signals in order to increase the signal strength. The algorithm used to increase data rate is Vertical Bell Laboratories Layered Space-Time (V-BLAST) and the algorithm used to increase the signal quality is Space Time Block Codes (STBC). Finally the channel encoder and decoder performs forward error correction that enables receivers to correct errors in the data sequence without retransmission. There are many FEC algorithms which are used in wireless systems but LDPC is the most computationally intensive kernel and used for the highest data rates. LDPC has also been proposed in many standards like TGnSync and WWise [13] for IEEE 802.11n, which leads us to believe it may be used in 4G systems as well.

3.1 Fast Fourier Transforms (FFT and IFFT)

The transmission path uses an inverse FFT (IFFT) for modulation and the receiver uses an FFT for demodulation. We will only discuss the algorithm for FFT because IFFT is almost identical. The FFT operation consists of a data movement operation followed by a multiplication and addition on a complex number. An N-point radix-2 decimation in frequency (DIF) FFT consists of $\log_2 N$ stages. Between each stage, the N points of data are shuffled in a butterfly pattern. As an example, Fig. 3a shows the data movement pattern of an 8-point FFT which consists of three stages. Each stage shows a different but regular data movement pattern. The operation of each stage can be divided into several 2-point FFT operations. We used a radix-2 FFT because other FFT implementations have more complex shuffle patterns which require more cycles to implement even though the arithmetic may be simpler (as is the case with radix-4 FFT).

Figure 3b shows the data dependence graph (DDG) on SODA for one of the major inner loops of FFT. Each node represents the SODA instruction and the edges represent the data dependency. These instructions implement the butterfly operation on the SODA processor [14]. There are many edges within the DDG where there is only one producer immediately followed by the consumer. This means that the values produced by those instructions will only be used once and then never referenced again. From this DDG we can see there is a common instruction pattern that occurs as shown in Fig. 3c. From this pattern we can see that only one of the multiply values produced needs to

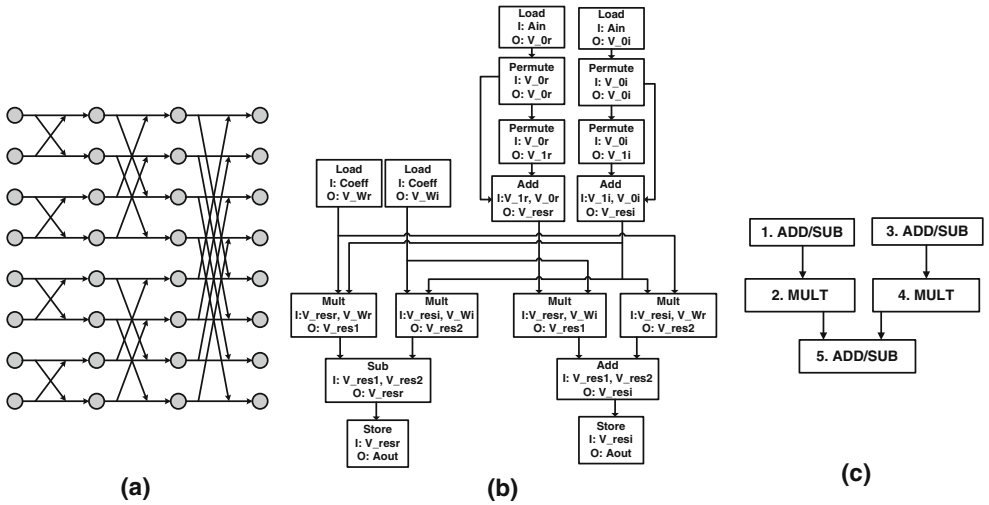


Figure 3 For the 8-point FFT, the butterfly pattern is the cross between two different elements in the vector. The major inner loop of FFT performs the operation of the butterfly across a SIMD. The nodes represent SODA instructions and the edges represent data dependencies. The instruction pattern represents the actual butterfly operation that is performed in each lane. **a** 8-Point FFT data movement. **b** Major inner loop of FFT. **c** Common instruction pattern in FFT.

be stored if we were executing the instructions in the shown order.

The vector width of an N-point FFT is equal to the number of data points, N. For a 1024-point FFT used in the NTT DoCoMo test setup, we perform 1024-point FFTs, meaning that the vector width is also 1024. FFT has a large amount of DLP because all 2-point FFT operations required between stages can be done in parallel. This means that the SIMD can be utilized almost 100%, suggesting we should increase the SIMD width to be as large as the vector width of the algorithm in order to achieve maximum performance.

3.2 Space Time Block Codes (STBC)

As stated before, STBC is used to increase the signal quality of the transmitted signal. Figure 4 shows the general operation of STBC. The same data is transmitted through each antenna but its representation is different for each antennae. Specifically, the signal is transmitted through multiple antennae in conjugate forms with different orderings. Signal quality is increased by receiving the redundant copies of the same data signal and optimally combining the information from each receiver to produce better quality

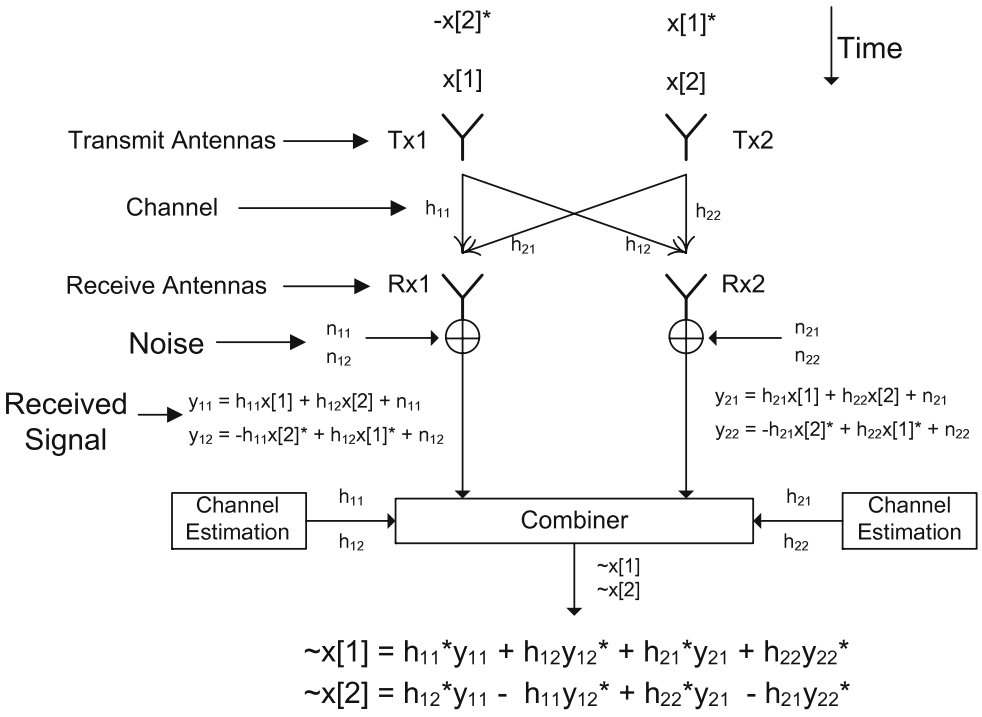


Figure 4 STBC general operation of Alamouti scheme.

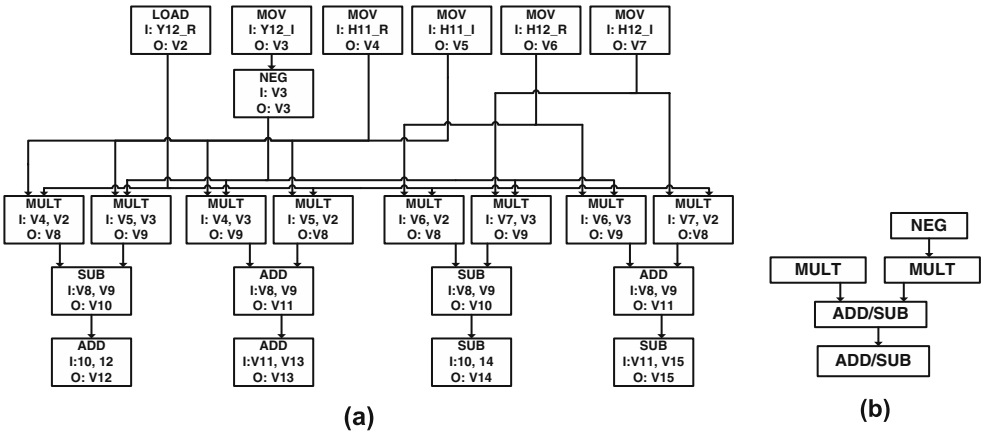


Figure 5 The inner loop of STBC shown calculates the values x[1] and x[2] of Alamouti Scheme. The common instruction pattern as shown is the majority of the inner loop code. **a** Major inner loop of STBC. **b** Common instruction pattern in STBC.

estimations of the original data signal. The implementation we used is based on Alamouti's 2x2 scheme [5].

In the STBC encoder and decoder, the vector width is only 4 elements. Though the vector width is small, each data set is independent. This means that we can join many data sets together and process one large set. The set size would be limited only by the amount of data the FFT provides. In our case, this would suggest that a 1024 width data set would be most optimal though this will be dependent on the final 4G standard.

Figure 5a shows the DDG for one of the major inner loops of STBC. These instructions implement part of the STBC signal detection operation on the SODA processor. Like FFT there exists many edges in the DDG where the values produced by instructions will only be used once and then never referenced again. We see a common instruction pattern that occurs as shown in Fig. 5b. From this pattern we can see that only one of the multiply values produced needs to be stored if we were executing the instructions in order.

3.3 Vertical Bell Laboratories Layered Space-time

V-BLAST is a spatial multiplexing scheme that improves the data rate by transmitting independent data streams over multiple antennae. This technique combines the multiple signals to obtain higher data rate rather than combine the same signal like STBC.

The general decoding process of V-BLAST consists of two major steps: channel estimation and signal detection as shown in Fig. 6. The channel matrix is estimated

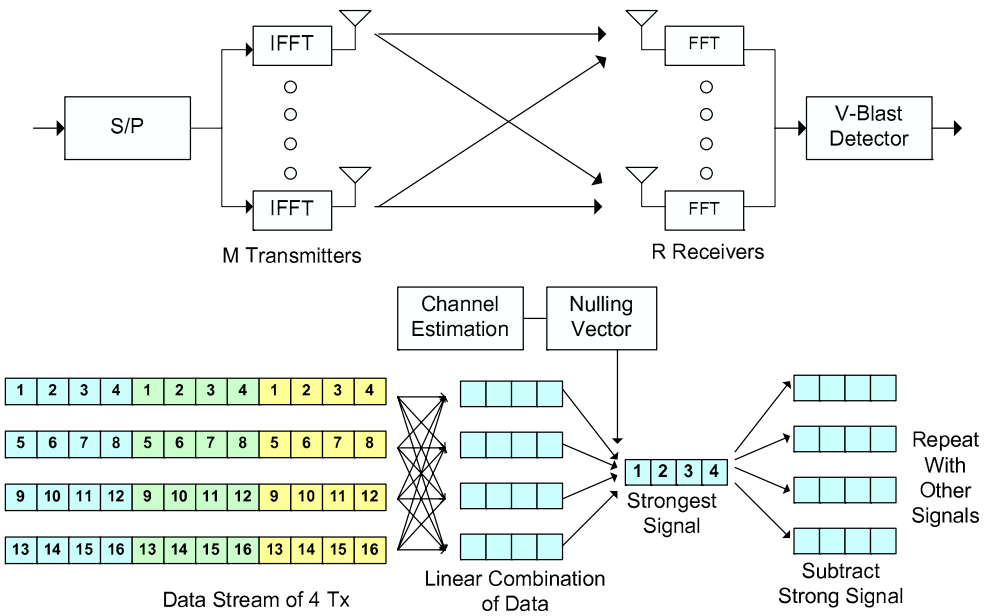


Figure 6 General decoding process of V-BLAST.

based on pre-defined training symbols. The operations for channel estimation are relatively simple with shift and sign-change operations. Once the channel matrix has been estimated, the detection order is determined. The detection order is based on signal strength found among all the signals received. The strongest signal is selected and extracted from the received signal. This process is repeated for the remaining signals. This process is iterative. The V-BLAST algorithm we implemented was based on work from [20], which reduces the computational complexity of the general V-BLAST decoder.

Because our system is based on a 4 transmit and 4 receive V-BLAST, the vector width of V-BLAST is 4 elements. The dimension of the channel matrix is 4x4 and the data signal is 4xN, where N is the number of data points in the FFT. The calculations performed are matrix operations with the channel matrix. Though the vector width is only 4, we can exploit larger SIMD widths because we can process larger sections of the 4xN data signal. The algorithm itself can support SIMD widths up to 4N.

Unlike FFT and STBC, V-BLAST has lots of control and predicate operations within it's algorithm. Though this control and predication can be parallelized, the DDG becomes very complicated. Figure 7 shows some of the few common instruction patterns that are found in the algorithm. Figure 7a is often used within matrix operations. The shuffle and add operations help align the different rows and columns together and perform the complex number operations. Figure 7b is used frequently in complex number operations. Because this V-BLAST deals with complex numbers the most common operation is the complex multiply. Within these two operations there are values which are produced and consumed and never used again outside the small set of instructions. These values potentially do not have to be written into the RF.

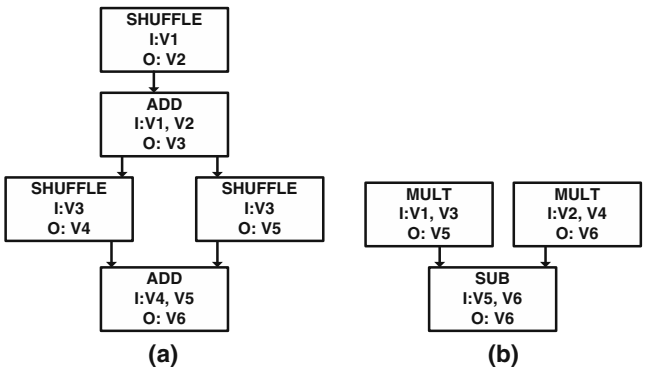


Figure 7 These two instruction patterns are reused many times within V-BLAST. They are common in many of the matrix operations. **a** Subgraph related to matrix operations. **b** Subgraph related to complex multiply.

3.4 Low Density Parity-Check Codes (LDPC)

LDPC is an error correcting code that can perform closer to Shannon's limit than any other code. This means that LDPC can be used to achieve the highest data transmission rate possible over a wireless channel. LDPC is made up of only simple adds, subtracts and compares. LDPC has no serial dependency in operation unlike Turbo Codes that have to process SISO decoders serially after the interleaver. Our implementation of LDPC is based on [17] which was optimized for the SODA processor and 802.16e.

The graphical representation of LDPC is shown in Fig. 8a. The check nodes represents the number of rows in the parity check code and the bit nodes represent the number of columns. The edges connecting the check nodes and bit nodes are the 1's in the parity check code matrix—all other values are 0. The LDPC decoding operation is broken down into 4 stages as shown in Fig. 8b. These four stages are the Initialization, Bit Node, Check Node, and Bit Update operation. This implementation is based on the Min-Sum algorithm.

Figure 9 shows the main operations within the inner loops of the LDPC decoder. These operations are responsible for the majority of the work within LDPC. A

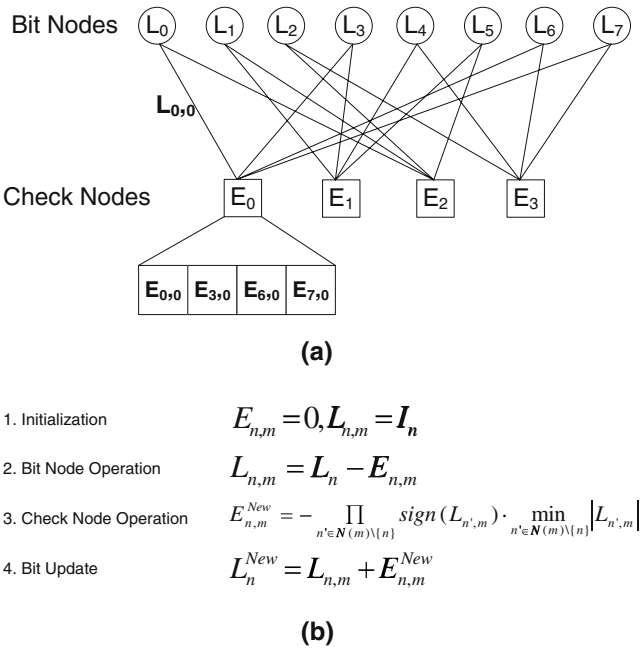


Figure 8 LDPC graphical representation and decoding operations. **a** Graphical representation of LDPC code. **b** LDPC decoding in 4 steps.

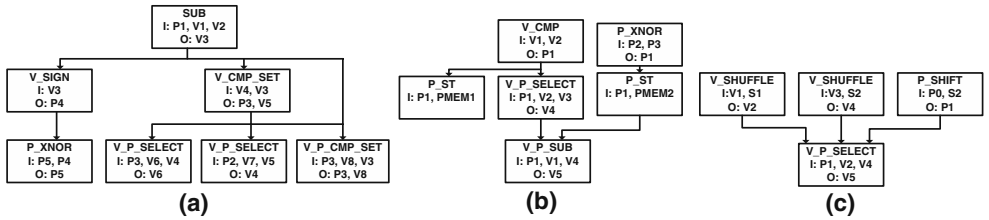


Figure 9 The inner loop operations of LDPC. These three operations represent the majority of the LDPC workload. They correspond closely to steps 2–4 of the LDPC decoding operation. **a** Inner loop for bit node and check node operation. **b** Inner loop for bit update. **c** Inner loop for bit and check node alignment.

large amount of time in LDPC is spent within the bit and check node alignment. The alignment operation, as shown in Fig. 9c, shuffles two vectors and then combines different sections of the two vectors into one vector. There is no need to save the values that are produced by the two shuffle operations because they are used only by the following select instruction. Outside this operation the values produced by the shuffles are never used.

LDPC is naturally parallel unlike other error correction codes. The vector width of the algorithm is related to the z size of the circulant shifted identity matrix *(zxz)*. The z value we used was 96 which corresponds to the maximum LDPC block size in 802.16e [2], which we assume is the highest data rate because it allocates the most number of subchannels. This means that we can benefit from SIMD widths up to 96 elements using this z value. After this limit there is no performance benefit. Unlike the other algorithms, we cannot overlap multiple z wide block rows of the LDPC matrix because there exists a data dependency between block rows. This prevents us from utilizing SIMD widths larger than 96 elements. Though 802.16e uses a z values of 96, 4G may use larger z values allowing SIMD widths larger than 96 to be beneficial.

4 Methodology

We used the SODA architecture as our SDR SIMD architecture to explore the scalability of SIMD width and also the modified RF. We used SODA because we could modify the implemented Verilog hardware model for different SIMD widths and change the RF. We modify the kernels' assembly code to support these multiple widths and to use the modified RF. All assembly code was written based on implementations cited previously in SODA assembly. All of the optimizations and assembly code were done by hand. We synthesized SODA using Synopsys' Physical Compiler in 0.13 micron technology for 400 MHz. Energy values were then extracted from the models and total energy was estimated based on the execution of the kernels.

5 SIMD Width Analysis

In Table 1, we analyze the DLP contained within the kernels. The instructions are broken down into 3 categories: overhead workload, scalar workload and vector workload. Overhead workload consists of all the instructions that assist SIMD computations, for example SIMD loads, stores and shuffle operations. The scalar workload consists of all the instructions that are not parallelizable and have to be run on the scalar unit. The vector workload consists of all the raw SIMD computations that use the ALU, multiplier, and shift units.

From the table, we can see that FFT is dominated by the overhead workload of loading the SIMD data and shuffling it. STBC and V-BLAST have a very high SIMD computation which suggests that these algorithms are dominated by raw computations and may be adaptable to very wide SIMDs. Finally, LDPC seems to have a mixture of all three types of instructions that

Table 1 Data level parallelism analysis for major 4G kernels.

Algorithm	Overhead workload (%)	Scalar workload (%)	Vector workload (%)	Vector width (elements)
FFT/IFFT	61	5	34	1024
STBC	14	5	81	4
V-BLAST	24	6	70	4
LDPC	33	18	49	96

suggests that performance may be limited by the non-vector workloads.

The table also presents the natural vector widths of the algorithms. Because we are using a 1024 point FFT the natural vector width is 1024. This means that this algorithm can support a SIMD of up to 1024 and still show performance improvement. This is very different from the other 3 algorithms whose vector widths are much narrower. For STBC and V-BLAST the vector widths may be small, but each set of elements are independent of each other allowing us to map multiple sets of 4 element computations onto any larger sized SIMD to increase performance. LDPC vector width may be its limiting factor because after a SIMD width of 96, the algorithm is constrained by the overhead and scalar workload, which prevents mapping of multiple sets of the vector elements. Any SIMD width larger than 96 will see no benefit in performance.

We took each of the major kernels of the 4G system and mapped them onto wider versions of the SODA architecture. Most of the algorithms parallelized quite easily. FFT and STBC were especially easy to parallelize. As we can see in Fig. 10, the instruction breakdown hardly changed from widths 32 to 256 because these algorithms are composed mainly of loops containing SIMD computations. By increasing the SIMD width, only the number of loop iterations changed. Lastly, V-BLAST is somewhat of an exception, because as we increase the width, the scalar instructions start to dominate. Thus the performance of V-BLAST will eventually be bounded by the scalar workload but, as we can see, the benefit of SIMD width can still provide major benefits.

In Fig. 11, we show the normalized speed up of the kernels for different SIMD widths. All the values were normalized to the 32 wide SODA implementation. For comparison purposes, the 32 wide SODA implementation can perform FFT almost 10 times faster than the TI TMS320C6203 [1]. For FFT, STBC and V-BLAST, the speed up is linear with width: doubling the width, yields 2x return for each of the algorithms. The performance benefit of increasing SIMD width is apparent. LDPC, though, is a different story. Because of the natural vector width of 96 we can only extract limited parallelism within the kernel. For SIMD widths greater than 128, there is no improvement in performance. The large jump between 64 and 128 width SIMD occurs because we cannot map all 96 values onto on a 64 wide SIMD machine. This forces us to do two iterations instead of the one possible with the 128 and 256 width SIMD. Because LDPC is the major performance bottleneck of 4G, this suggests that increasing SIMD alone may not meet the processing requirements.

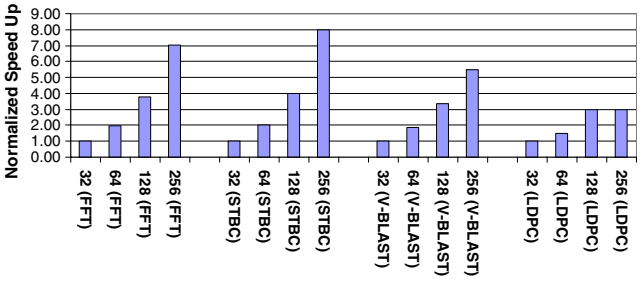


Figure 11 Normalized speedup of each kernel with respect to SIMD width.

Finally in Fig. 12, we show the energy consumption of each kernel for different widths. We computed the energy consumed taking into account leakage. As we can see, for FFT, STBC and V-BLAST there is a great benefit from increasing SIMD width. We get the greatest performance increase with reasonable increase in energy consumption. The exception again is LDPC. Going from 32 to 64, we take a large energy penalty mainly due to the fact that many SIMD lanes are wasted because the algorithm cannot map perfectly onto widths that don't divide into the natural vector width. Between 64 and 128, there is not much change

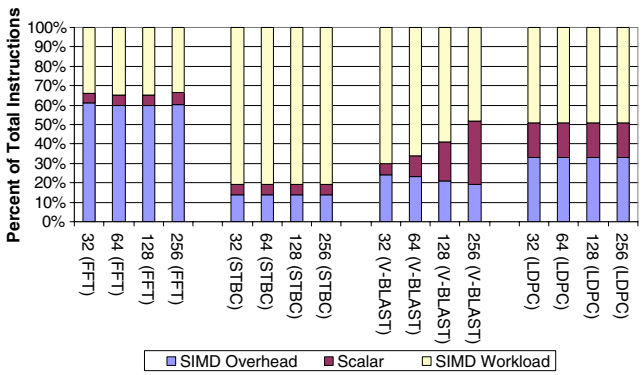


Figure 10 Instruction breakdown of each kernel with respect to SIMD width.

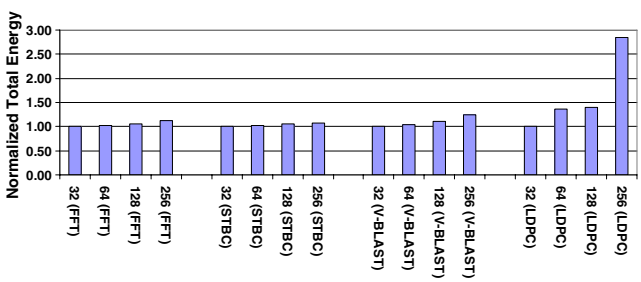


Figure 12 Total energy consumption of each kernel with respect to SIMD width.

because the algorithm is actually mapped relatively efficiently on the SIMD. The biggest jump is at 256 because all SIMD widths greater than 96 will waste energy doing unneeded work on the remaining 160 lanes. SODA does not support clock gating, which can help alleviate this problem by shutting off the lanes not being used, but will still result in inefficient use of silicon area for this algorithm and wasted potential performance.

We find that for these kernels increasing SIMD width does give us a good increase in performance with little increase in energy. The exception to this was seen in LDPC because the algorithm did not gain any improvement when SIMD width was greater than 128. As stated in Section 3, future implementations of LDPC may have larger z values which would allow us to efficiently use larger SIMD widths.

6 Register Access Analysis

Figure 13 shows the power breakdown of SODA running the 4G kernels. We can see the vector RF consumes almost 37% of the total energy. Others have shown that for other processors, RF power consumed almost 25% of the total power [11]. By analyzing the RF access patterns of the kernels we find that we can reduce the total energy consumed by the RF.

In Section 3, the inner loop operations of each kernel were discussed. Each kernel had instruction patterns within the inner loops where register values produced were then used by the next following instruction and then the value was never referenced again. This was apparent in FFT and STBC which had simple inner loops. In the SODA architecture, as with many common DSPs, the result value of each instruction is written back to the RF and when the value is needed it is read out of the RF. If the value is only referenced within a few instructions then there is no reason the value itself has to be written back into the main RF. Energy is wasted by writing and then reading the register value. Instead, the value can be either forwarded or stored in a small temporary buffer which can be accessed by any instruction. Techniques to implement this have been shown in [8, 16, 21].

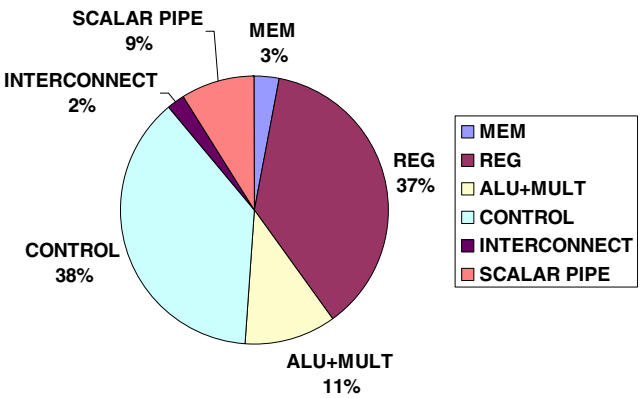


Figure 13 Power breakdown for the kernels on SODA.

Table 2 Percent of total register file access that are produced and consumed within a subgraph.

Algorithm	FFT/IFFT	STBC	V-BLAST	LDPC
RF reduction (%)	59	53	12	37

These register file accesses can use the small register file.

By analyzing the kernels running on SODA, we found the percent of RF access that had potential for registers not writing into the main RF. This is listed in Table 2. From the table we find that FFT, STBC and LDPC have large amounts of RF accesses that can bypass the main RF. Our implementation of V-BLAST does not have large amounts of registers that can be bypassed because there are large amounts of control within the algorithm. As seen in the SIMD width analysis, the bottleneck of this kernel are the scalar control instructions. Because of this control, the registers need to be kept until the condition that chooses the value is calculated. This requires the use of the full RF. Though there are lots of register value which need to be kept, there are operations within the kernel which can be bypassed. These operations are related to all the matrix and complex number arithmetics that are done within the kernel.

The modification to SODA, shown in Fig. 14, was a small two-entry RF and register write bypass which is explicitly controlled by the instruction. The two-entry RF is a partition of the main RF similar to split register files [21]. We implemented it by mapping two registers of the main RF registers to the smaller RF. When a request for those registers is made, we disable the read/write to the main RF and mux the data from/to the smaller RF. Register write bypass prevents the value from being written to the RF and forwards it directly to the input of the next instruction. This is done by an extra bit in the instruction. If the bit is set then the instruction will not write the value into the RF because

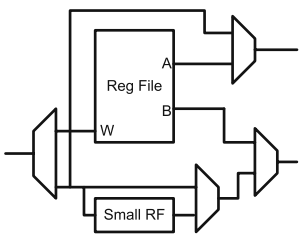


Figure 14 Modified register file. Data can either be written to the main register file, the small register or bypassed completely from both allowing data to be forwarded directly to the next functional unit.

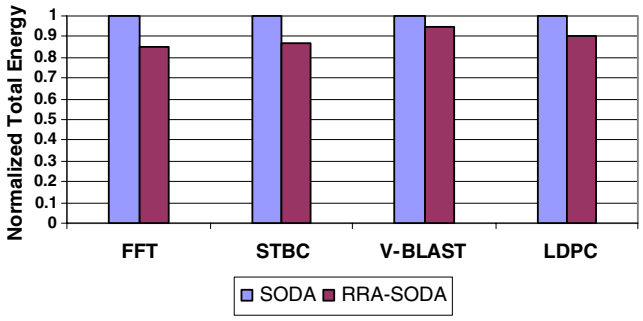


Figure 15 Total energy consumption of each kernel with reduced register file access. SODA is the baseline processor where each instruction always writes back to the register file then is read when needed. RRA-SODA is the modified SODA architecture that can write the values to a smaller register file

the next instruction will use the value and the value will not be used again. The next instruction, which references the same register as the current instructions' register destination, will get the data forwarded directly from the write port of the RF to the correct read port without the value being written to the RF. Figure 15 shows the amount of energy savings that can be achieved on a modified SODA architecture. From the figure we see that we are able to save between 6–15% of the total energy in the processor while maintaining the same performance. These results were as expected, following what we saw in Table 2. FFT, STBC and LDPC showed substantial energy reduction by reducing RF accesses while V-BLAST showed far less reduction in energy. This total reduction in energy translates into lower total power of the processor for these set of kernels.

7 Conclusion

Though the power and performance requirements of 4G is a significant challenge for designers, scaling SIMD width and reducing register accesses can help us gain major performance increases and reduce power consumption. We have seen almost a doubling of performance with doubling SIMD width. Not all kernels benefited from the increase in SIMD width. LDPC clearly is a major limiting factor in 4G. By increasing the SIMD width, FFT, STBC and V-BLAST benefits but LDPC benefits less. This suggests that LDPC may better be implemented on an accelerator or another specialized core with a different SIMD width. By reducing the RF accesses, we reduced the total energy by between 6–15% across the kernels. While FFT, STBC, and LDPC had the most energy reduction because many of the RF accesses could be reduced, V-BLAST had less reduction in energy because many of the values had to be kept.

SIMD scalability and register access reduction are just two techniques in the processor design where we can extract more performance and reduce energy. Referring back to Section 3, we notice that many of the kernels had similarities between their common subgraphs. In future work we will see if we can take advantage of these common subgraphs to improve performance. Another open issue is compilation for SIMD architectures. In this work, the kernels were hand written in assembly and optimized for varying with SODA. This prevents code compatibility among other types of systems with varying widths. Further work has to be done on kernel compilation for SIMD architectures and scheduling the code for varying widths. With a combination of the two architectural techniques presented and future work in SDR processor design, we may eventually be able to process 4G efficiently, within the power and performance requirements.

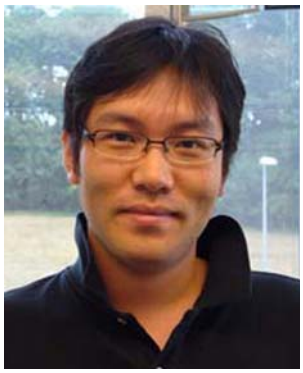
References

1. Dsp developers' village. Texas Instruments. http://dspvillage.ti.com.
2. IEEE Std 802.16e. *Part 16: Air interface for fixed and mobile broadband wireless access systems*. http://standards.ieee.org/getieee802/download/802.16e-2005.pdf.
3. *International technology roadmap for semiconductors*. http://public.itrs.net .
4. ITU-R M.1645. Framework and overall objectives of the future development of IMT-2000 and systems beyond IMT-2000. *International telecommuncations union M.1645 recommendation*. http://www.ieee802.org/secmail/pdf00204.pdf.
5. Alamouti, S. (1998). A simple transmit diversity technique for wireless communications. *IEEE Journal on Selected Areas in Communications, 16*(8), 1451–1458.
6. van Berkel, K., Heinle, F., Meuwissen, P. P. E., Moerman, K., & Weiss, M. (2005). Vector processing as an enabler for software defined radio in handheld devices. *EURASIP Journal on Applied Signal Processing,* 2005(1), 2613–2625.
7. Bluethgen, H.-M., Grassmann, C., Raab, W., & Ramacher, U. (2003). A programmable platform for software-defined radio. *International symposium on system-on-chip* (p. 15), 19–21 Nov. 2003.
8. Corporaal, H., & Mulder, H. J. M. (1991). MOVE: A framework for high-performance processor design. In *Proc. of the 1991 ACM/IEEE conference on supercomputing* (pp. 692–701). Albuquerque, New Mexico, USA.
9. Fridman, J., & Greenfield, Z. (2000). The TigerSharc DSP architecture. In *IEEE micro* (pp. 66–76), Jan. 2000.
10. Glossner, J., Hokenek, E., & Moudgill, M. (2004). The sandbridge sandblaster communications processor. In *3rd workshop on application specific processors* (pp. 53–58), Sept. 2004.
11. Guan, X., & Fei, Y. (2008). Reducing power consumption of embedded processors through register file partitioning and compiler support. In *International conference on application-*

specific systems, architectures and processors (ASAP) (pp. 269–274), 2–4 July 2008.
12. Haensch, W., Nowak, E., Dennard, R., Solomon, P., Bryant, A., Dokumaci, O., et al. (2006). Silicon CMOS devices beyond scaling. *IBM Journal of Research and Development, 50*(4/5), 339–361.
13. Lestable, T., & Zimmermann, E. (2005). LDPC options for next generation wireless systems. *Proceedings of the 14th wireless world research forum (WWRF)*. San Diego, CA, Jul. 2005.
14. Lin, Y., Lee, H., Woh, M., Harel, Y., Mahlke, S., Mudge, T., et al. (2006). SODA: A low-power architecture for software radio. In *Proc. ISCA* (pp. 89–101), 17–21 June 2006. Boston, MA.
15. Naessens, F., Bougard, B., Bressinck, S., Hollevoet, L., Raghavan, P., Van der Perre, L., et al. (2008). A unified instruction set programmable architecture for multi-standard advanced forward error correction. In *Proc. IEEE SiPS*, 8–10 Oct. 2008. Washington D.C., USA.
16. Park, S., Shrivastava, A., Dutt, N., Nicolau, A., Yunheung, P., & Earlie, E. (2008). Register file power reduction using bypass sensitive compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27*(6), 1155–1159.
17. Seo, S., Mudge, T., Zhu, Y., & Chakrabarti, C. (2007). Design and analysis of LDPC decoders for software defined radio. In *Proc. IEEE SiPS*, 17–19 Oct. 2007. Shanghai.
18. Taoka, H., Higuchi, K., & Sawahashi, M. (2006). Field experiments on real–time 1-Gbps high–speed packet transmission in MIMO–OFDM broadband packet radio access. *IEEE 63rd vehicular technology conference, 2006 (VTC 2006-Spring)* (Vol. 4, pp. 1812–1816), 7–10 May 2006.
19. Woh, M., Seo, S., Lee, H., Lin, Y., Mahlke, S., Mudge, T., et al. (2007). The next generation challenge for software defined radio. In SAMOS (Ed.), *Lecture notes in computer science* (Vol. 4599, pp. 343–354).
20. Zhu, H., Lei, Z., & Chin, F. (2004). An improved square-root algorithm for BLAST. *IEEE Signal Processing Letters, 11*(9), 772–775.
21. Zyuban, V., & Kogge, P. (1998). Split register file architectures for inherently lower power microprocessors. In *Proc. power-driven microarchitecture workshop, in conjunction with ISCA '98*, (pp. 32–37), June 1998.

Mark Woh is a PhD candidate in electrical engineering and computer science at the University of Michigan Ann Arbor. His research interests include low-power microarchitecture and wireless communication. Woh has a BS in electrical engineering and computer science from the University of Michigan at Ann Arbor.

Yuan Lin is currently a senior engineer at Sigmatix. His research interests include high performance DSP architecture, algorithm, and compiler design for the next generation wireless communication protocols. He has a PhD in electrical engineering and computer science from the University of Michigan at Ann Arbor. He is a member of the ACM and IEEE.



Sangwon Seo is a Ph.D. candidate in electrical engineering and computer science at the University of Michigan, Ann Arbor. His research interests include low-power microarchitecture and wireless signal processing. He received his B.S. degree in electrical engineering from Seoul National University, Korea in 2005.

Scott Mahlke is an Associate Professor in the Electrical Engineering and Computer Science Department at the University of Michigan where he leads the Compilers Creating Custom Processors group (http://cccp.eecs.umich.edu). The CCCP group

delivers technologies in the areas of compilers for multicore processors, application-specific processors for mobile computing, and reliable system design. Mahlke received the Ph.D. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1997. Mahlke's achievements were recognized by being named the Morris Wellman Assistant Professor in 2004 and being awarded the Most Influential Paper Award from the Intl. Symposium on Computer Architecture in 2007.

Trevor Mudge is the first Bredt Family Professor of Electrical Engineering and Computer Science at the University of Michigan at Ann Arbor. His research interests include computer architecture, CAD, and compilers. Mudge has a PhD in computer science from the University of Illinois, Urbana-Champaign. He is a member of the ACM, Institution of Engineering and Technology, and British Computer Society and a fellow of the IEEE.