Analog Integr Circ Sig Process (2011) 69:207–218
DOI 10.1007/s10470-011-9712-8

Instructions and hardware designs for accelerating SNOW 3G on a software-defined radio platform

Chris Jenkins · Michael Schulte · John Glossner

Received: 1 February 2011 / Revised: 22 June 2011 / Accepted: 11 July 2011 / Published online: 11 August 2011

Abstract Software-defined radio (SDR) is a new technology transitioning from research into commercial markets. SDR moves hardware-dominant baseband processing of multiple wireless communication protocols into software on a single chip. New cellular standards, such as HSPA+, LTE, and LTE+, require speeds in excess of 40 Mbps. SNOW 3G is a new stream cipher approved for use in these cellular protocols. Running SNOW 3G in software on our SDR platform provides a throughput of 19.1 Mbps per thread for confidentiality and 18.3 Mbps per thread for integrity. To have secure cellular communications in SDR platforms for these new protocols, the performance of security algorithms must be improved. This paper presents instruction set architecture (ISA) extensions and hardware designs for cellular confidentiality and integrity algorithms using SNOW 3G. Our ISA extensions and hardware designs are evaluated for the Sandbridge Sandblaster™ 3011 (SB3011) SDR platform. With our new SNOW 3G instructions, the performance of confidentiality and integrity improve by 70 and 2%, respectively. For confidentiality, power consumption increased by 2%, while energy decreased by 40%. For integrity, power consumption remained consistent, while energy decreased by 2%.

Keywords Cryptography · Software-defined radio · Computer architecture · SNOW 3G · Cellular security

C. Jenkins (✉) · M. Schulte
Department of ECE, University of Wisconsin—Madison, 3632 Engineering Hall, 1415 Engineering Dr, Madison, WI 53713, USA
e-mail: cdjenkins@wisc.edu

M. Schulte
e-mail: Michael.Schulte@amd.com

M. Schulte
AMD, Inc. AMD Research, 7171 Southwest Pkwy, Austin, TX 78739, USA

J. Glossner
Optimum Semiconductor Technologies, Inc., 120 White Plains Rd, 4th Floor, Tarrytown, NY 10591, USA
e-mail: jglossner@optimumsemi.com

1 Introduction

Software-defined radios (SDRs) employ a combination of hardware and software to support multiple wireless communication standards dynamically. These devices have been widely recognized as one of the most important new technologies for wireless communication systems [1]. SDRs enable the efficient implementation of a diverse set of wireless communication systems using software, instead of hardware blocks, for each protocol. SDRs also provide the ability to change communication protocols and dynamically update communication systems through over-the-air software downloads [2].

Current cellular systems provide security by means of cryptographic operations such as confidentiality (i.e., encryption and decryption) and integrity (i.e., message authentication). These operations prevent eavesdropping on the transmission and ensure that each data packet has arrived without modification. As data rates over the air interface increase, cryptographic performance becomes a bottleneck. The 3rd Generation Partnership Project (3GPP™) has defined several algorithms for use in upcoming UMTS/SAE air-interfaces. One such cipher is SNOW 3G [3]. It is used to provide confidentiality and integrity. Future architectures will need to enable cryptographic processing at very high data rates. To expand, next-generation radio access networks seek to provide over-the-air throughputs of up to 100 Mbps

for mobile environments and 1 Gpbs in low-mobility environments. On our test SDR platform, our reference software implementation with SNOW 3G achieves 19.1 Mbps per thread for confidentiality and 18.3 Mbps per thread for message integrity. While both protocols use SNOW 3G, only confidentiality will need to achieve faster data rates. Integrity is only used for signaling messages and does not need substantial bandwidth.

This paper presents ISA extensions and hardware designs for accelerating SNOW 3G confidentiality and integrity algorithms on a multi-threaded SDR platform, the Sandbridge Sandblaster™ 3011 (SB3011). Our proposed instructions and hardware for acceleration have the following important features:

(1) The ability to operate efficiently in a multi-threaded processor micro-architecture.
(2) No hidden state is added to the programming model.
(3) Efficient use of the single-instruction/multiple-data (SIMD) unit.

Due to the placement of our hardware inside a SIMD unit, we believe our approach should also be useful in other SDR architectures, which commonly feature SIMD units. The primary contribution of this paper presents a programmable approach to accelerating SNOW 3G that utilizes existing hardware features present in many SDR platforms. In addition, this paper presents software profiling to demonstrate the execution time breakdown of algorithms that use SNOW 3G with and without software optimizations. Furthermore, we determine which portions of the SNOW 3G algorithm to accelerate. Power and energy of the different software versions are also analyzed, and trade-offs from using new instructions and hardware versus conventional software are presented.

This paper extends our previous research [4] by enhancing the methodology and analysis of performance, power, and energy profiling and examining implementation trade-offs in greater detail. This paper presents profiling with and without software optimizations and discusses how these optimizations affect power and energy consumption. Similar profiling is performed after adding the proposed ISA extensions. Furthermore, the paper proposes a substitution box structure that differs from a standard lookup table.

In the rest of this paper, Sect. 2 describes the SNOW 3G algorithm and its use for confidentiality and message integrity. Section 3 discusses related work. Section 4 details our platform architecture, simulation environment, and testing methodology. Section 5 and 6 presents SNOW 3G software performance, power, and energy profiling results. Section 7 presents our proposed functional units and ISA extensions for accelerating SNOW 3G. Section 7 demonstrates the performance improvements, energy benefits, and design characteristics of our solution. Section 8 presents conclusions.

2 Background

2.1 Snow 3G Cipher

SNOW 3G is a stream cipher approved for use in cellular networks to provide both confidentiality and integrity [5]. It uses a 128-bit initialization vector (IV) and 128-bit seed key. Two components make up the cipher: a finite state machine (FSM) and a linear feedback shift register (LFSR), as shown in Fig. 1. The FSM contains three 32-bit registers, *R1*, *R2*, and *R3*, plus two substitution boxes (or S-boxes), *S1* and *S2*. Each S-box maps one 32-bit value to another 32-bit value. The LFSR contains sixteen 32-bit registers, numbered *S0* through *S15*, with taps at registers *S0*, *S2*, and *S11*. The tap values are combined using bitwise XOR operations to produce a new value, *v* that is loaded into *S15*. Before the bitwise XOR operations, *S0* is processed by the function MUL_α and *S11* is processed by DIV_α. These functions are denoted α and α^{-1} in Figs. 1 and 2.

The cipher has two modes of operations: initialization (Fig. 1) and keystream (Fig. 2). As the name implies, initialization mode occurs before keystream mode. During initialization mode, the FSM registers are initialized to zero. The registers of the LFSR are initialized based on the IV and key, as described by the standard. Figure 1 illustrates how the cipher runs in initialization mode, during which the LFSR and FSM are clocked 32 times. For each clock, the output of the FSM is a 32-bit word, *F,* which is

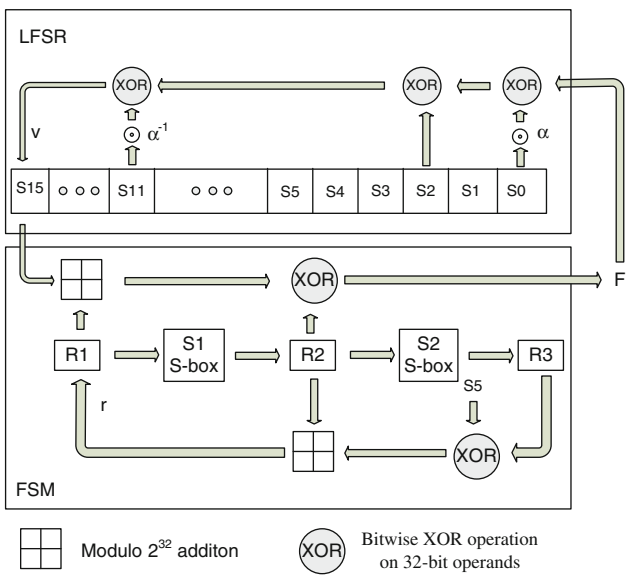

Fig. 1 SNOW 3G in initialization mode [3]

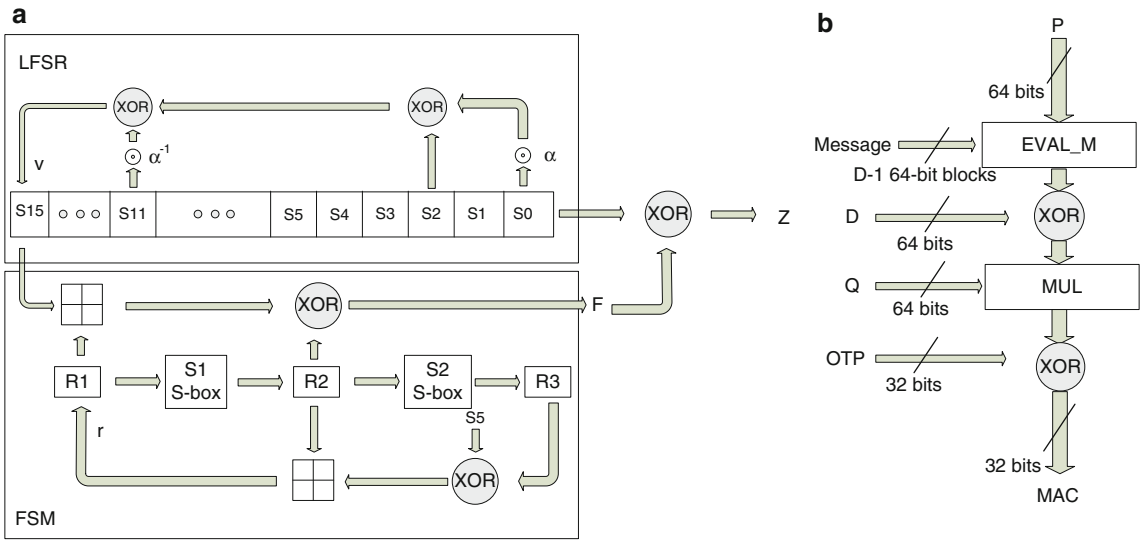

Fig. 2 SNOW 3G in keystream mode [3] and flow of the integrity algorithm

an input to the LFSR. Next, the cipher enters keystream mode, as shown in

In this mode, the FSM still produces *F*, but *F* is used to generate a 32-bit keystream word, *Z*. The LFSR then shifts and loads the value, *v*, into register S15. The process repeats until enough data bits are generated for the confidentiality or integrity algorithm.

2.2 Confidentiality and integrity algorithms

While SNOW 3G is the underlying cipher for both the confidentiality and integrity algorithms, the actual algorithms have data processing outside of the main cipher. Signaling and user data undergo confidentiality while only signaling data undergo integrity processing [6, 7]. For both integrity and confidentiality, the IV and keys have a lifetime of usage. For 3G wireless protocols, the same IV and key may be used to process up to 20,000 bits. After 20,000 bits, a new IV and key must be generated [8]. The confidentiality algorithm, called UEA2 or EEA1,[1] uses SNOW 3G to generate a keystream. For encryption, the keystream is XORed with the plaintext (data to be encrypted) to produce the ciphertext (encrypted data). For decryption, the ciphertext is XORed with the keystream to produce the original plaintext. This implies that both the sender and receiver produce identical keystreams. The number of key bits generated matches the length of the data.

The integrity algorithm, called UIA2 or EIA1,[2] uses the SNOW 3G cipher in a different manner. It runs the cipher to produce five 32-bit words (*z1, z2, z3, z4, z5*). It pairs the first four words into two 64-bit data values, called *P* (*z1*, *z2*) and *Q* (*z3, z4*), as shown in Fig. 2. The algorithm breaks the message into 64-bit data blocks and zero pads the last 64-bit block if the entire message is not a multiple of 64 bits. An additional 64-bit block, which contains the length, *D*, of the message, is appended to the end of the message. To perform the integrity algorithm, *D*-1 blocks of the message are multiplied by the binary polynomial *P* inside the function *EVAL_M*. The multiplication is performed over a Galois field of 64 bits, GF (2^{64}). *D* is XORed with the running data value and the result is multiplied by *Q* inside the function *MUL*, which is a single 64-bit Galois field multiply. The upper (left-most) 32 bits of the result are XORed with *OTP* (*z5*, the last 32-bit word). The upper (left-most) 32 bits of the resulting value are used as the message authentication code (MAC) for the message. The lower (right-most) 32 bits are discarded.

3 Related work

Related work on SNOW 3G has been performed in two major areas: commercial designs and ASIC-based research. Elliptic Technologies™ Inc. [9], provides an ASIC core that implements the SNOW 3G cipher and many other popular cryptographic algorithms. Another organization, IP Cores, Inc. [10], provides ASIC implementations as well as HDL source code versions for SNOW 3G and the algorithms that use it.

[1] UEA2 = Universal Mobile Telecommunications System (UMTS) Encryption Algorithm 2. EEA1 = Evolved Packet System Encryption Algorithm 1.

[2] UIA2 = UMTS Integrity Algorithm 2. EIA1 = Evolved Packet System Integrity Algorithm 1.

Work has been performed by the research community for SNOW 3G. Orhanou et al. [11] performed a complexity study of the SNOW 3G cipher. They analyzed time complexity and space complexity of SNOW 3G and the underlying functions. Kitsos et al. [12] analyzed and implemented the SNOW 3G cipher using a standard cell design flow. They focused on optimizing tables and adders in the critical path of the design, and provided a comparison with other implementations and other ciphers. Hessel et al. [13] analyzed various aspects of SNOW 3G and AES; both are used in LTE. They determined how the system changes with respect to data rate. They proposed and investigated different methods of encoding the S-boxes. Further work reduced the memory footprint of S-boxes and improved threading support for the core cipher [14]. Other work has looked into using smaller Galois field multipliers [4].

4 Methodology

4.1 Sandbridge SB3011

Our in-house simulation environment simulates the SB3011 SDR Platform. The platform contains four Sandblaster™ DSP cores, an ARM™ processor, and input and output peripherals found on many wireless handheld devices. Each Sandblaster™ core has eight hardware threads and uses a form of interleaved multi-threading (IMT) known as token-triggered threading (T^3) [2].

Each core is partitioned into three main units: an instruction fetch and branch unit, an integer and load/store unit, and a SIMD vector-processing unit (VPU), as shown in Fig. 3. The VPU consists of four vector processing elements (VPEs), a shuffle unit, a reduction unit, and an accumulator register file, as shown in Fig. 4. It executes logic and arithmetic operations on 16-, 32-, and 40-bit fixed-point data types concurrently in each VPE. The VPU loads either 128 or 64 bits depending on the data type. The VPU requires two load instructions to load 32-bit data into each VPE, while 16-bit data requires one load instruction. The instruction format allows for up to three source operands and one destination operand per VPU instruction.

4.2 Simulation environment and algorithm implementation

Our toolchain infrastructure provides a full-system, cycle-accurate simulator and allows us to analyze the current ISA and add custom-defined ISA-supported instructions that are implemented as intrinsics [15]. These instructions are mapped to user-defined functions inside the simulator. Each user-defined instruction takes one thread cycle, which is the same amount of time as each native instruction.

We use the C code provided in the SNOW 3G standard, but change recursive code to loop-based code. This helps saves stack space and provides the ability to use zero-overhead loop counters and various compiler optimization techniques to improve performance. We also modified encryption code to XOR 32-bit words instead of bytes to improve performance. Some functions of the standard C library are not supported in the Sandblaster™ DSP simulator. Slight modifications were done for integrity to remove the dependency on the library for a particular function. All other code was left intact and is what we call the "baseline" in the rest of the paper.

4.3 Performance analysis and power/energy modeling

To determine where to accelerate the SNOW 3G cipher, we profiled the SNOW 3G reference implementation [3, 16] code using the Sandblaster™ toolchain and full-system, cycle-accurate simulator [15]. For both the confidentiality and integrity algorithms, we started with the baseline code from the standard with the modifications described in Sect. 4.2. We used the test vectors from the standard to ensure correctness when making any code modifications. Next, we identified which parts of each algorithm consume a significant percentage of the execution time. SNOW 3G has some potential software optimizations to improve performance. As with the baseline case, we looked at how each optimization affected performance, power, and energy. We analyzed the potential benefit from these optimizations prior to looking at hardware acceleration.

To estimate power and energy of running the algorithm, we executed the Sandblaster™ instructions without our new SNOW 3G instructions on a fabricated SB3011 processor chip. During execution, we measured the actual power consumption using volt and amp meters. From these measurements, we determined that instructions within a particular instruction class (e.g., integer arithmetic, vector arithmetic, wide vector arithmetic, control, load/store, etc.) have roughly the same power dissipation, and individual functional units consume a small amount of power compared to other portions of the processor such as the clock tree, multi-threaded register file, instruction cache, and data memory. Our observations agreed with previous studies of real power dissipation in embedded processors [17].

From the observations, we calculated the average power dissipation for a program by multiplying the percentage of instructions in each instruction class by the corresponding average power for that instruction class. With this approach, we were able to estimate the power consumption with an average error of less than 8%. To find energy, we multiplied the average power by the processing time. For

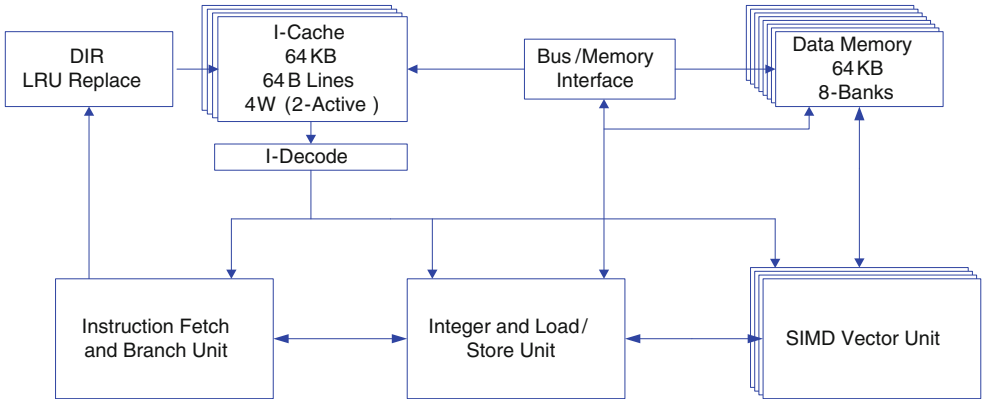

Fig. 3 Sandblaster™ DSP

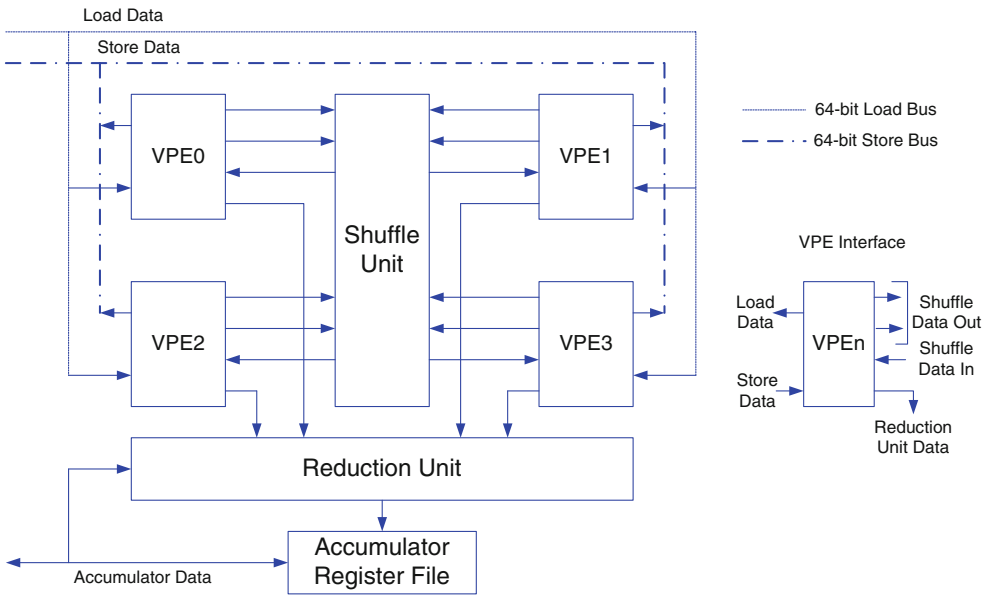

Fig. 4 Sandblaster™ DSP SIMD VPU

custom SNOW 3G instructions, we had to use a different approach. Since our custom SNOW 3G instructions are not present in a real processor, we assumed our SNOW 3G instructions consume power equal to the wide vector arithmetic instruction class, which corresponds to the class with the highest power consumption.

5 UEA2 performance profile

5.1 Baseline

Within the SNOW 3G algorithm, ten 1-KB tables may be utilized to accelerate different operations of the SNOW 3G cipher. The MUL and DIV tables implement the MUL_α and DIV_α functionality in the LFSR. Four tables implement the *S1* S-box; another four tables implement the *S2* S-box. Figure 5 shows the speed-up gained from using tables stored in level-1 (L1) data memory for the different functions. Each bar represents adding a table only for the specified function(s). Using all the tables provides a total speed-up of 26× as compared to the version without any tables. This comes with a cost of 10 KB of static data in L1 memory.

Other potential optimizations that are not mentioned in the standard are feasible. We employ two methods, using an array of 32-bit integers to implement the LFSR as opposed to the code from the standard, which uses sixteen independent 32-bit integers to implement the LFSR. Both methods use all the aforementioned software tables. The first method uses pointers to access the particular registers during computation. Typical pointer bounds checking occurs after an increment (listed as LFSR + MOD in Fig. 5). The second method involves using vector loads and

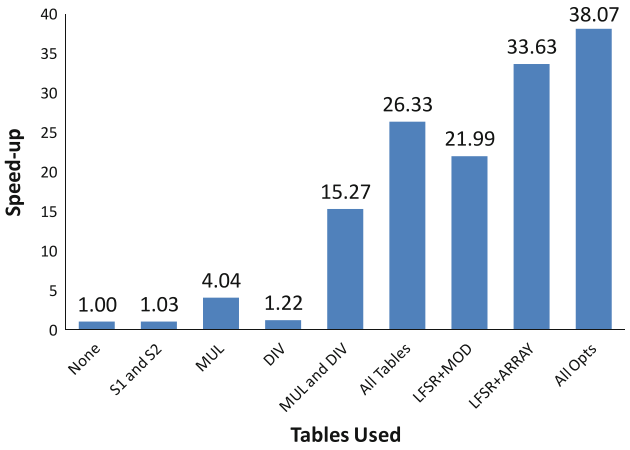

Fig. 5 UEA2 speed-ups due to using tables and other software optimizations

stores to move the data (LFSR + ARRAY in Fig. 5). Our simulation results show that pointer arithmetic actually performs slower than the baseline implementation. However, the version that utilizes vector loads and stores is faster than the baseline implementation. Vector loads read 64 bits of data versus an integer load of 32 bits. With sixteen 32-bit registers, there are eight vector loads and eight vector stores, instead of sixteen integer loads and sixteen integer stores. Even with all the tables present, plus the LFSR optimization (an approximate 34× total speed-up over the original code), the architecture only achieves a throughput of 16.9 Mbps per thread for confidentiality. The last software optimization applied to the code is function inlining. All other software optimizations are present as well. The result is listed as 'All Opts' in Fig. 5 and reaches 19.1 Mbps.

Table 1 shows that with our software optimizations applied, the FSM consumes more processing time than the LFSR. Hence, as we design hardware for SNOW 3G, priority goes to hardware that accelerates the FSM. Furthermore, since the FSM consumes a little more than half the processing time (56.89%), it is advantageous to include hardware for the LFSR also. Table 2 shows the breakdown of the FSM and LFSR functions.

5.2 Power/energy profile

We have a power profile for the Sandblaster™ instructions on the SB3011. Using the same configuration for determining speed-up, we analyzed the power and energy consumption. Since a majority of the processor power comes from the clock tree, register files, control logic, and L1/L2 memories, the different software implementations do not change power draw by a large amount. Since memory instruction access the L1 memory sub-system, the power draw increases with an increase in the number of tables utilized. This can be seen in Table 3. Having all tables present in L1 memory increases the power consumption by 5%. Performing all the software optimizations adds 7% to the baseline power draw. While there is an increase in power for the software optimizations, utilizing these optimizations saves energy due to the much shorter execution time (energy = power × time), as shown in Table 3. With all tables present, the energy consumption drops by 96% compared to the baseline configuration. With our optimizations, a 97% reduction in energy is obtained. However, the data rates achieved by the optimizations still fall below by the performance needed for next-generation data rates.

6 UIA2 performance profile

The other algorithm that uses the SNOW 3G cipher is the integrity algorithm. As with confidentiality, we profile the baseline implementation. Table 4 shows the processing time profile for the confidentiality algorithm. The second column, going left to right, depicts the relative processing time for the baseline implementation. The third column depicts the relative processing time after adding the software optimizations to the SNOW 3G cipher. The fourth and fifth columns are described next.

Table 1 Processing time profile with all software optimizations

Function	Percent of total processing time
Initialize	20.60
FSM	11.16
LFSR	8.41
Other processing	1.02
GenerateKeyStream	77.18
FSM	42.56
LFSR	30.84
Other processing	3.77

Table 2 FSM and LFSR keystream profiles

Function	Percent of total processing time
FSM	42.56
S1	15.57
S2	15.57
Other processing in FSM	17.89
LFSR	30.84
MUL_{α}	3.89
DIV_{α}	3.89
Other processing in LFSR	24.67

Table 3 UEA2 power and energy relative to baseline due to using tables and other software optimizations

	None	S1 and S2	MUL	DIV	MUL and DIV	All Tables	LFSR + MOD	LFSR + ARRAY	All Opts
Power	1.00	1.00	1.01	1.00	1.03	1.05	1.04	1.06	1.07
Energy	1.00	0.98	0.25	0.82	0.07	0.04	0.05	0.03	0.03

Table 4 UIA2 relative processing time with various optimizations

Function	Percent of total processing time			
	Baseline	With all SNOW 3G SW optimizations	With SNOW 3G and MUL tables	With function inlining
EVAL_M	97.97	99.86	41.53	29.36
MULP	–	–	16.62	20.56
Pre_mul_p	–	–	31.87	39.43
Initialize	1.65	0.06	4.88	4.61
GenerateKeystream	0.31	0.01	0.91	0.86
Other processing	0.06	0.06	2.95	3.61

6.1 Integrity specific software optimizations

As shown in Table 4, the EVAL_M function, which performs 64-bit Galois field multiplication over GF (2^{64}), consumes a significant percentage of the overall processing time. There are software optimizations to speed up the calculation of the 64-bit Galois field multiply. Eight tables of 2 KB each are used to perform a single 64-bit Galois field multiply. These tables are listed in the standard, and the relative processing time for each function using these eight tables and the other SNOW 3G software optimizations is represented in the fourth column of Table 4. The last column represents using all of these optimizations plus function inlining for SNOW 3G. There is also some function inlining specific to integrity. Figure 6 shows the throughput achieved as we add the SNOW3G tables and the Galois field multiplication tables. Implementing these tables takes 26 KB of static data in the L1 memory. With these optimizations, the integrity algorithm reaches a throughput of 18.3 Mbps. Integrity is primarily used for signaling and does not need large bandwidth. Therefore, we do not consider integrity acceleration hardware. However, we still provide power and energy numbers for completeness.

6.2 Power/energy profile

Similar to confidentiality, we use the ISA power profile of the Sandblaster™ DSP to determine power and energy usage for each of the optimizations. We see from Table 5 that power increases by 3% with a corresponding energy saving of 99%, when using software tables for optimizations as compared to the baseline (no tables).

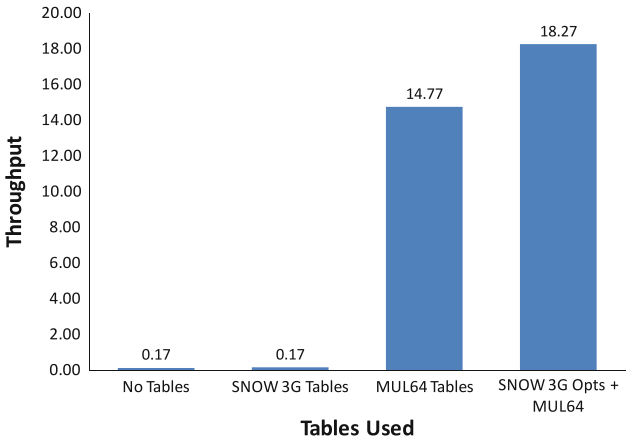

Fig. 6 UIA2 single-thread throughput due to using tables and other software optimizations

7 Proposed instructions and results

7.1 New instructions

As mentioned previously, our simulation infrastructure allows for custom instructions supported at the ISA level using intrinsics functions in the C code. We add new code to simulate the SNOW 3G functional units inside the VPU and use the new instructions to access this code. These instructions take up to three source operands and a single destination operand. From the previous analysis, we

Table 5 UIA2 power and energy relative to baseline due to using tables and other software optimizations

	No tables	SNOW 3G tables	MUL64 tables	SNOW 3G Opts + MUL 64
UIA2 power	1.00	1.00	1.03	1.03
UIA2 energy	1.00	0.98	0.01	0.01

accelerate both the LFSR and FSM of the SNOW 3G cipher. For the FSM to move forward, new values of *R1*, *R2*, and *R3* need to be produced. For the LSFR to move forward, a new *v* value must be produced and the entire array of integers should be shifted to the right.

We propose the following instructions to accelerate the SNOW 3G cipher:

- *snow3g_fsm* (*VRD, VRS1, VRS2, VRS3*). This instruction reads two 32-bit values from the first pair of VPE registers, specified by VRS1, and two 32-bit values from the second pair of VPE registers, specified by VRS2 and VRS3. It passes the values from VRS1 to the corresponding *S1* and *S2* transforms, shown in Figs. 1 and 2. It stores the resulting value in the appropriate registers specified by VRD in the first pair of VPEs. The second set of VPEs uses the values in VR2 and VR3 to implement the part of the FSM that generates a new *F* value and *r* value as $F = (S15 + R1) \oplus R2$ and $r = (R3 \oplus S5) + R2$. The resulting *F* and *r* values are stored in the appropriate registers specified by VRD in the second pair of VPEs.
- *snow3g_shuffle* (*VRD, VRS*). This instruction rearranges the 32-bit values in VRS and stores the result in VRD. The effective result is rotating a 128-bit register to the right by 32-bits.
- *snow3g_v* (*VRD, VRS1, VRS2, VRS3*[3]). This instruction produces the value $v = \alpha^{-1}(S11) \oplus S2 \oplus \alpha(S0)$ during initialization mode and $v = \alpha^{-1}(S11) \oplus S2 \oplus \alpha(S0 \oplus)$ F during keystream mode, as shown in Figs. 1 and 2. It takes the source registers specified by VRS1 and VRS2 and extracts the appropriate values to compute *v*. When the instruction uses three operands, the last register of VRS3 in the second pair of VPEs is also used. The instruction utilizes the MUL_α and DIV_α tables. The results from the table lookups and *S2* are combined to produce *v*, which is written to the first register of the first set of VPEs specified by VRD.

7.2 Performance

To determine the performance improvement, we analyzed the added benefit of the new instructions using the fully optimized SNOW3G with function inlining as the baseline. This is different from the baseline software implementation described in the previous sections. As with software, we perform a comparison between the different acceleration methods. Our test set corresponds to the highest data set specified in the standard [16]. This equates to 3,861 bits for confidentiality and 16,448 bits for integrity. We compare the ISA extensions in the SNOW 3G cipher as well as the Galois field multiplier extensions. Figure 7 shows the performance gains from adding the ISA extensions.

The FSM extensions provide a greater benefit to the core cipher than the LFSR extensions. The SNOW 3G ISA extensions provide around a 70% improvement in the UEA2 algorithm. Since a majority of the processing time in the UIA2 algorithm is spent performing Galois field multiplication, the SNOW 3G extensions do not show much improvement. With the 64-bit Galois field multiplier, we see a large improvement of 455% in the performance of the UIA2 algorithm.

The code size for core SNOW 3G functions are listed in Table 6. We see that with the FSM, the extensions reduce the code size by more than 66%. However, the LFSR extensions do not provide the same benefit. The LFSR function consists of address-offset calculations, memory loads, generating *v*, and shifting the LFSR. Even though the LFSR extension greatly speed up generation of *v*, a good percentage of the original memory loads are needed to load the data needed to generate *v*. Likewise, the shifting of the LFSR remains the same with and without ISA extensions. Hence, the movement of data becomes the bottleneck.

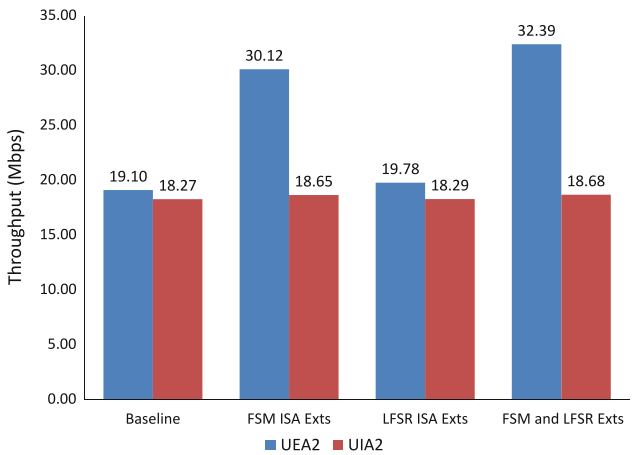

Fig. 7 Throughput with SNOW3G ISA extensions

[3] VRS3 is optional.

Table 6 Code size of core SNOW 3G functions

Function	Baseline (bytes)	ISA Extensions(bytes)	Reduction (%)
ClockLFSRInitializationMode	280	272	2.9
ClockLFSRKeyStreamMode	264	232	12.1
ClockFSM	448	152	66.1

Table 7 ISA extensions' power and energy relative to baseline

	Baseline	FSM ISA Exts	LFSR ISA Exts	Both Exts
UEA2 power	1.00	1.02	1.00	1.02
UEA2 energy	1.00	0.64	0.96	0.60
UIA2 power	1.00	1.00	1.00	1.00
UIA2 energy	1.00	0.98	1.00	0.98

7.3 Power and energy

One key aspect of adding ISA extensions stems from the energy saving of performing a task much more quickly at the cost of potentially increasing power. Table 7 illustrates how power and energy change when adding the ISA extensions for the UEA2 and UIA2 algorithms. Similar to the performance table, the LFSR extensions do not provide substantial energy saving. Fortunately, there is not major change in power either. The majority of the energy saving comes from the FSM extension, with a 2% increase in power. The UIA2 algorithm benefits the most by adding the 64-bit Galois field multiplier. The net result increases power by 3%, but pays off by reducing energy consumption by 83%.

7.4 Implementation

Our proposed implementation is designed to re-use existing hardware. While ASIC solutions exist [9, 10, 12], a programmable solution potentially has less additional area because bus interface logic, FIFOs, and extra registers, which are present with an ASIC, are not needed with ISA extensions. In the Sandblaster™ DSP, there are two locations considered for implementing the new functional units: the integer unit and the SIMD VPU. While both are viable alternatives, we believe the VPU is a better choice. First, the VPU has greater load/store bandwidth than the integer unit—64 bits versus 32 bits. Second, the VPU can access more data for a given instruction—480 bits versus 64 bits. Third, the VPU can hold more state than the integer unit—1,280 bits per thread versus 256 bits per thread.

An important aspect of our approach is our use of the SIMD VPU to implement our proposed SNOW3G and GFM64 instructions. Many SDR architectures employ multiple SIMD processing engines and/or pipelines with 16-bit or 32-bit data types [18–24]. We believe our design has the potential to be used in those architectures. While other computing platforms (e.g., IBM Cell™, x86, SPARC™, MIPS™, and ARM™) are not geared specifically for SDR, they use SIMD units and have the potential to use our design. We also utilize the SIMD storage and memory bandwidth with our approach. However, the functional processing we employ has the SIMD unit perform non-uniform data processing. Since cryptography does not make good use of traditional SIMD processing, we add units that can operate in parallel, but do not follow a pure SIMD processing paradigm. For the SB3011 processor, we add the proposed functional units illustrated in Fig. 8.

7.5 Description of functional units

As seen in Fig. 8, the different functional units are spread throughout the VPU. The MUL_{α} and DIV_{α} tables implement the MUL_{α} and DIV_{α} functionality. They use an 8-bit index and produce a 32-bit output. The XOR-adder gates correspond to the XOR-add operation specified in the standard. The XOR and adder are both 32 bits in length. Consideration was given to the implementation of the S1 and S2 S-boxes. A previous publication implemented the S1 and S2 S-boxes as eight separate 1-KB tables [12].

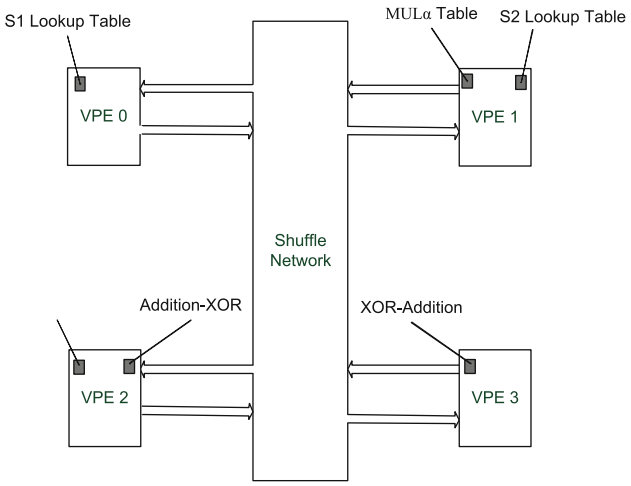

Fig. 8 Functional unit placement in the SandBlaster™ VPU

Table 8 Comparison between S-box implementations (65-nm TSMC CMOS library, 600 MHz target frequency)

Type	Area (μm^2)	Area ratio	Latency (ns)	Latency ratio
S1 [12]	7,860	1	0.96	1
S1 (this paper)	4,161	0.53	0.94	0.98
S2 [12]	8,024	1	1.00	1
S2 (this paper)	4,068	0.51	1.00	1

Table 9 Design characteristics (65-nm TSMC CMOS library, 600 MHz target frequency)

Unit	Delay (ns)	Area (μm^2)	Power (μW)
S1	0.93	4,161	372.5
S2	1.00	4,068	373.9
Addition-XOR	1.58	453	75.2
MUL_α	0.40	126	18.5
DIV_α	0.33	123	17.0
SNOW 3G total	1.58	9,384	932.3
SB3011 VMAC	1.65	6,864	1,543.1

VMAC vector multiply accumulate unit which is present in the Sandblaster™ DSP

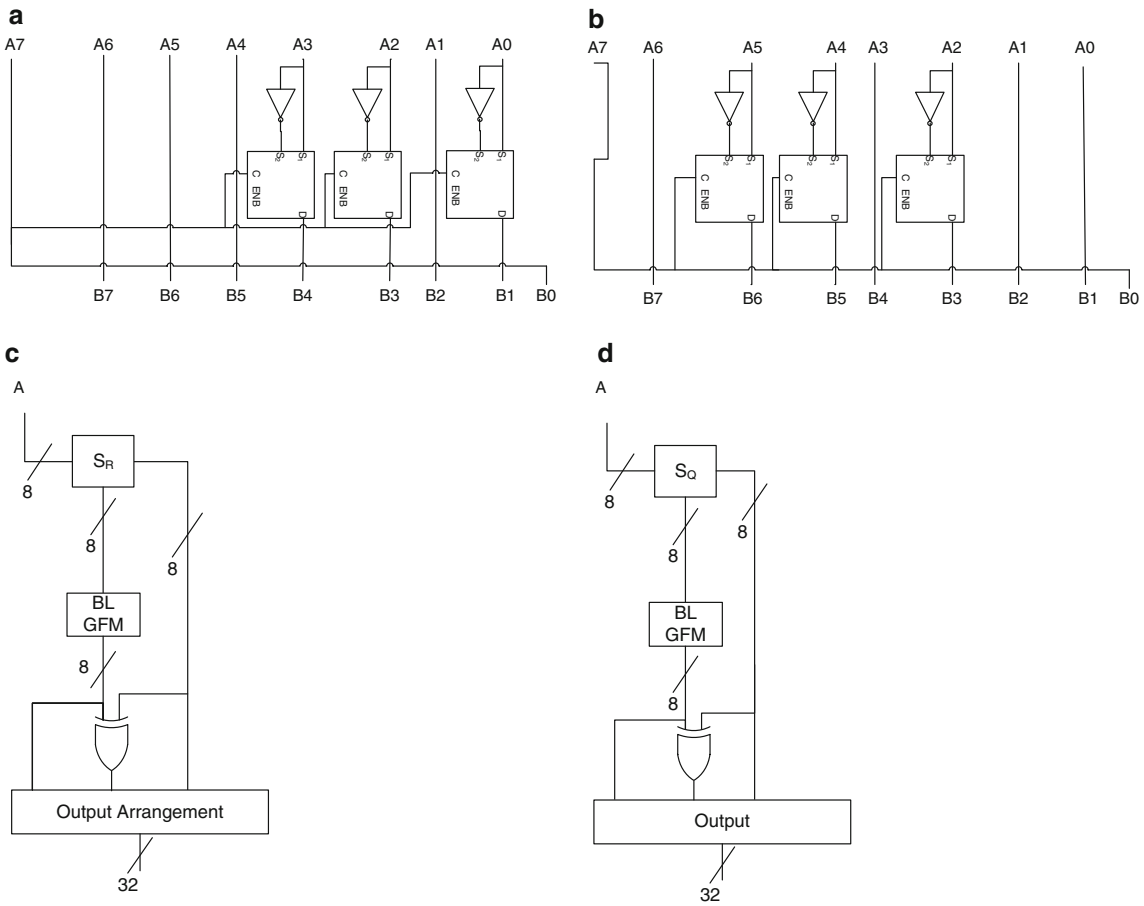

Fig. 9 Byte-level multiplier and 8-bit to 32-bit expander for S1 and S2 S-boxes

While this method works, we chose another method to implement the S-boxes using base tables. For S1, the base table corresponds to the AES S-box. For S2, the base table corresponds to S_Q, which is an S-box defined by the SNOW 3G standard. Logic around each base table expands the 8-bit output of the base table to a 32-bit output. Using this new method reduces the table lookup size from 8 to 2 KB. A comparison between our method for implementing S1

and S2 and the method presented in [12] is shown in Table 8. The latency does not change much using our approach, but the area is reduced about 50% for both the S1 and S2 S-boxes (Table 9).

Figure 9(a) and (b) illustrate the design of the two S-boxes. A byte-level multiplier (BLM) shifts the input left by one bit and reduces the shifted value to the underlying finite field (using 0x1B for S1 and 0x69 for S2). The output of the S-box transformation and BLM are fed into XOR gates. The results of the transformations are arranged in a pre-defined order according to the standard, as illustrated in Fig. 9(c) and (d). Four of these units implement the S1 S-box and four implement the S2 S-box.

8 Conclusion

This paper presented an in-depth look at profiling the SNOW 3G algorithm and its use for confidentiality and integrity algorithms in cellular networks. We profiled the core algorithm with and without software optimizations to determine bottlenecks. We proposed ISA extensions to address performance gaps and examined the resulting impact on both power and energy. Furthermore, we introduced a new design to reduce the area of standard implementations of S-boxes. The SNOW 3G ISA extensions improve performance by a factor of 1.7 and 1.0 for the UEA2 and UIA2 algorithms, respectively. Power increases by 2% and 0% with corresponding energy savings of 2% and 40% for the UEA2 and UIA2 algorithms, respectively. Further performance improvements can be obtained by implementing an LFSR shift operation.

Acknowledgements This material is based on work supported by the National Science Foundation under Grant No. CNS-0640880. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Mehta, M., Drew, N., Vardoulias, G., Greco, N., & Niedermeier, C. (2001). Reconfigurable terminals: an overview of architectural solutions. *IEEE Communications Magazine, 39*, 82–89.
2. Mamidi, S., Blem, E. R., Schulte, M. J., Glossner, J., Iancu, D., Iancu, A., Moudgill, M., & Jinturkar, S. (2005). Instruction set extensions for software defined radio on a multithreaded processor. In *Proceedings of the 2005 international conference on compilers, architectures and synthesis for embedded systems* (pp. 266–273).
3. ETSI/SAGE. (2006). Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2. Document 2: SNOW 3G specification. http://www.gsmworld.com/documents/snow_3g_spec.pdf.
4. Jenkins, C., Schulte, M., & Glossner, J. (2011). Instruction set extensions for accelerating SNOW 3G on a multithreaded software defined radio platform. In *Proceedings of the SDR '10 technical conference and product exposition* (pp. 94–100). Arlington, VA.
5. ETSI/SAGE. (2006). Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2. Document 5: Design and evaluation report.
6. 3rd Generation Partnership Project. (2010). 3G security: Security architecture. Technical Report. 3GPP TS 33.102.
7. Balderas, T., Cumplido, R., Cumplido-parra, R. A., Computacionales, C. D. C., Erro, L. E., & Balderas-contreras, T. (2004). Security architecture in UMTS third generation cellular networks.
8. 3rd Generation Partnership Project. (2009). Cryptographic algorithm requirements. http://www.3gpp.org/ftp/specs/html-info/33105.htm.
9. Elliptics. CLP-41: SNOW 3G cipher core. http://www.ellipticstech.com/products-clp-41.php.
10. IP Cores, Inc. SNOW 3G LTE cipher. http://www.ipcores.com/Snow3G.htm.
11. Orhanou, G., El Hajji, S., & Bentaleb, Y. (2010). SNOW 3G stream cipher operation and complexity study. *Contemporary Engineering Sciences-Hikari Ltd, 3*, 97–111.
12. Kitsos, P., Selimis, G., & Koufopavlou, O. (2008). A high performance ASIC implementation of the SNOW 3G stream cipher. In *Proceedings of the 16th international conference on very large scale integration* (*VLSI-SoC 2008*).
13. Hessel, S., Szczesny, D., Lohmann, N., Bilgic, A., & Hausner, J. (2009). Implementation and benchmarking of hardware accelerators for ciphering in LTE terminals. In *Proceedings of the 28th IEEE conference on global telecommunications* (pp. 2316–2322).
14. Traboulsi, S., Sbeiti, M., Bruns, F., Hessel, S., & Bilgic, A. (2010). An optimized parallel and energy-efficient implementation of SNOW 3G for LTE mobile devices. In *Proceedings of the 12th IEEE international conference on communication technology* (pp. 536–539). Nanjing, China.
15. Glossner, J., Jinturkar, S., Moudgill, M., Hokenek, E., Schulte, M., & Vassiliadisols, S. (2005). Sandbridge software tools. *Embedded Computer Systems: Architectures, Modeling, Simulation, 3553*, 269–278.
16. ETSI/SAGE. (2006). Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2. Document 4: Design conformance test data. http://www.gsmworld.com/documents/snow_3g_spec.pdf.
17. Meyer, B. H., Pieper, J. J., Paul, J. M., Nelson, J. E., Pieper, S. M., & Row, A. G. (2005). Power-performance simulation and design strategies for single-chip heterogeneous multiprocessors. *IEEE Transactions on Computers, 54*, 684–697.
18. Glossner, J., Iancu, D., Moudgill, M., Nacer, G., Jinturkar, S., & Schulte, M. (2006). The sandbridge SB3011 SDR platform. In *Symposium on trends in communications (SympoTIC'06), Invited Keynote.* Bratislava, Slovakia.
19. Lin, Y., Lee, H., Woh, M., Harel, Y., Mahlke, S., Mudge, T., et al. (2007). SODA: A high-performance DSP architecture for software-defined radio. *IEEE Micro, 27*, 114–123.
20. Moudgill, M., Glossner, J., Agrawal, S., & Nacer, G. (2008). The sandblaster 2.0 architecture and SB3500 implementation. In *Proceedings at the software defined radio technical forum* (pp. 2–5).
21. Ramacher, U. (2007). Software-defined radio prospects for multistandard mobile phones. *Computer-IEEE Computer Society, 40*, 62–69.
22. Tell, E., Nilsson, A., & Liu, D. (2005). A programmable DSP core for baseband processing. In *The 3rd international IEEE-NEWCAS conference* (pp. 403–406).
23. 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, 16*, 2613–2625.

24. Woh, M., Seo, S., Mahlke, S., Mudge, T., & Chakrabarti, C. (2009). AnySP: Anytime anywhere anyway signal processing. In *Proceedings of the 36th international symposium on computer architecture (ISCA)* (pp. 128–139). Austin, Texas, USA, June 2009.

Chris Jenkins is a PhD dissertator at the Madison Embedded Systems and Architectures Laboratory (MESA). He received a B.S. in computer engineering from the University of Illinois at Urbana-Champaign. He received a M.S. in electrical engineering from the University of Wisconsin–Madison. His research focuses on security acceleration architecture for multi-threaded software defined radio platforms using shared hardware resources inside a SIMD unit.

Michael Schulte received a B.S. degree in Electrical Engineering from the University of Wisconsin-Madison, and M. S. and Ph.D. degrees in Electrical Engineering from the University of Texas at Austin. He is currently a Fellow Design Engineer with AMD Research and an Associate Professor (on leave) in Electrical and Computer Engineering with the University of Wisconsin-Madison. His research interests include domain-specific processors, heterogeneous computing, computer arithmetic, hardware acceleration, and computer architecture. He is a recipient of an NSF CAREER Award, the Alfred Nobel Robinson Award, and the Frank Hook Assistant Professorship. His has served an Associate Editor for the IEEE Transactions on Computers and the Journal of VLSI Signal Processing, and as Program Chair and General Chair for the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP), the IEEE International Symposium on Computer Arithmetic (ARITH), and the Asilomar Conference on Signals, Systems and Computers.

John Glossner is co-founder, CTO, and Executive Vice President at Sandbridge Technologies. Prior to co-founding Sandbridge, John managed both technical and business activities in DSP and Broadband Communications at IBM and Lucent/Starcore. John received a Ph.D. in Computer Architecture from TU Delft in the Netherlands, M.S degrees in E.E. and Eng. Mgt from NTU, and a B.S.E.E. degree from Penn State. John is chair of the Commercial Baseband Working Group of the SDR Forum, a Senior member of the IEEE and has more than 120 publications and 36 issued patents.