

Towards A Multi-FPGA Infrared Simulator

Vinay Sriram

Research fellow
Defence and Systems Institute
University of South Australia
vinay.sriram@unisa.edu.au

David Kearney

Associate Professor
University of South Australia
david.kearney@unisa.edu.au

High speed infrared (IR) scene simulation is used extensively in defense and homeland security to test sensitivity of IR cameras and accuracy of IR threat detection and tracking algorithms used commonly in IR missile approach warning systems (MAWS). A typical MAWS requires an input scene rate of over 100 scenes/second. Infrared scene simulations typically take 32 minutes to simulate a single IR scene that accounts for effects of atmospheric turbulence, refraction, optical blurring and charge-coupled device (CCD) camera electronic noise on a Pentium 4 (2.8GHz) dual core processor [7]. Thus, in IR scene simulation, the processing power of modern computers is a limiting factor. In this paper we report our research to accelerate IR scene simulation using high performance reconfigurable computing. We constructed a multi Field Programmable Gate Array (FPGA) hardware acceleration platform and accelerated a key computationally intensive IR algorithm over the hardware acceleration platform. We were successful in reducing the computation time of IR scene simulation by over 36%. This research acts as a unique case study for accelerating large scale defense simulations using a high performance multi-FPGA reconfigurable computer.

Keywords: computer architecture, electronics, electro-optics

1. Introduction

There are many processing tasks that are far too complex for standard processors. Traditional means of accelerating simulations using PC based processors is no longer an attractive option. As PC based processors have hit the so called “power wall”, i.e. the era of rapid processor frequency increases has come to an effective end. As a result it has brought to an end the speedup to numerical software and simulations which benefited from such frequency increases. This explains the rise of special purpose graphic hardware and PC based multi-core processors. However not all processing tasks can be accelerated on commodity platforms.

High speed infrared scene simulation is an important application in defense and homeland security because in recent times there has been an increased risk to military and civilian personal and equipment from portable handheld missiles. MAWS, sensitive to a missile's unique rocket propulsion exhaust, are capable of detecting and negating these threats in real time. Due to the high cost in carrying out field trials to test MAWS, IR scene simulations are used for this purpose. Also IR scene simulations have the added advantage that they allow complex sensor target threat engagements to be modelled. However, MAWS have typical input scene rates of over 100 scenes per second.

Simulating an IR scene is not as simple as constructing an image with all scene elements embedded onto the generated scene. As typical IR environments contain many benign sources of IR radiation of various size, shape and intensity, known collectively as 'clutter'. An accurate IR scene simulation needs to account for these complexities. Generation of high fidelity IR scenes needs to take into account the effects of atmospheric phenomena such as refraction and scintillation as well as optical phenomena like blurring and CCD camera electronic noise. A typical IR scene is shown in figure 1 below.



Figure 1: IR scene

Advanced computing based on programmable logic technology has been long established as a platform for parallel computing [2]. Image processing operations have achieved massive speedups due to the use of pixel level parallelism see [5] and [8]. Furthermore, the recent availability of large clusters of programmable logic devices and the ability to incorporate these devices into commodity PCs are further motivating factors for this line of research. Such computers, which combine a standard (possibly multi-core) processor with a field programmable gate array device (FPGA) are called reconfigurable computers. On such a computer the computationally complex algorithms are accelerated in hardware (i.e. FPGA) through the development of customized parallel digital circuits.

The development of such circuits is an active area of research in reconfigurable computing.

Applying reconfigurable computing to accelerate tasks traditionally performed in software is an active area of research with specific examples from various narrow domains being regularly published in international journals. This project also falls into the area of parallel computing research. Active investigations are continuing to determine which parts of the application should be placed in hardware as opposed to software and what the granularity of parallel techniques should be. In addition the inter-disciplinary nature of the project with the domains of scene generation and high performance computing are likely to produce new results that will be of interest to the computer graphics community and in particular computer generated synthetic environments outside the visual spectrum.

There are a number of choices for accelerating algorithms using FPGAs [2]. The simplest platform is a single FPGA attached to a standard PC motherboard. Module development will be carried out first on these platforms because of their ease of use and quick compilation times. However, after module development is undertaken, the development of a platform which consists of a cluster of FPGAs and CPUs will be carried out. Integrating hardware modules into the cluster will form a major part of this stage including decision making about communications between elements of the cluster and middleware.

In this paper we present results of our initial research to accelerate an infrared scene simulation over a hardware acceleration platform consisting of an array of programmable logic devices. We show how reconfigurable computing can overcome the computing limitations imposed by modern technologies like PC based dual core processors. This work is significant as other computationally intensive defence modelling and simulation applications can be accelerated in a similar fashion. In the next section we briefly discuss the methodology we employ. In section 3 we present the profiling results and identify the research problem. In section 4 we present multiple hardware implementations for one of the most computationally intensive infrared algorithms. In section 5 we present our multi-FPGA hardware acceleration platform and our middleware architecture.

2. Research Methodology

The conceptual framework for this research is to build a system embodying both software and reconfigurable hardware and to evaluate its performance with the objective of answering the question: How can hardware acceleration best be incorporated into an IR scene generation simulation to complement existing software while providing a sizeable acceleration of performance? We present the methodology we undertook to carry out this research project. The conceptual research framework is divided into four stages as illustrated in figure 2 below.

2.1 Stage 1: Profiling

The first task involved building a computational profile of an infrared scene simulation software, which is written in FORTRAN. This involved evaluating each software module with regard to the following factors:

1. Execution time
2. Frequency of execution
3. Communication with other modules
4. Memory requirements

For a given module, the outcome of studying these four factors determined where the performance of the simulation could be improved by shifting it into an FPGA (programmable logic) and whether it is practical to do so. The relationship between modules was important in developing the system architecture for integrating the hardware and software components.

2.2 Stage 2: System Design

From the software profile it was clear where the largest performance gains could be made by shifting current software modules into hardware. This stage also involved designing the overall architecture of the accelerated system. Efforts were made wherever possible to limit the overhead for data exchange so that it does not unduly slow the simulation down.

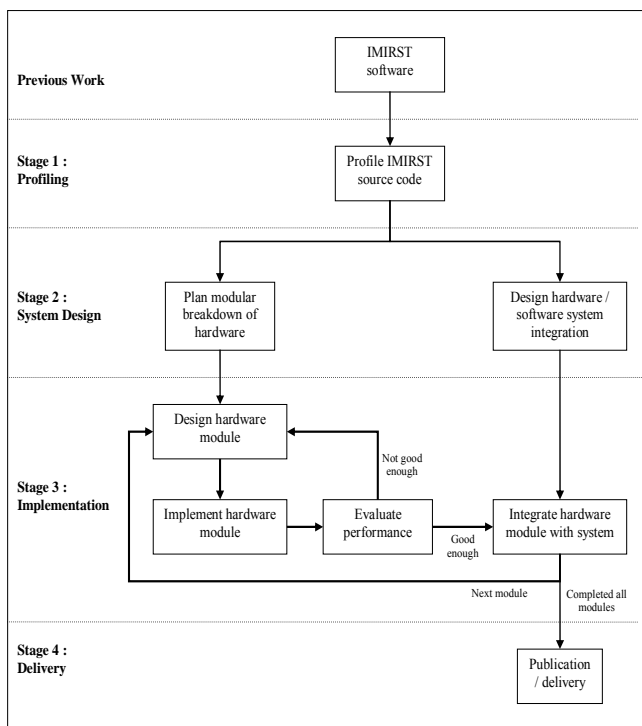


Figure 2: Research framework

The system was designed in such a way that it can operate in a software-only mode (as it currently does), without any hardware acceleration. When the FPGA accelerator board (programmable logic accelerator board) is present in the computer, the simulation software detects it and makes use of it in a manner transparent to the user. Even though

the user will not see any difference from the outside, the simulation runs much faster with the board installed.

2.3 Stage 3: Implementation

The implementation stage involved developing the hardware modules as planned in Stage 2, and integrating them with the simulation. How each module fits in, and its interaction with the other modules was already decided in Stage 2. Each module was developed and tested independently. This was followed by modular integration of each component independently.

The design was first simulated in software to check for correctness. It was then synthesised and placed into a programmable logic chip (FPGA chip), to allow evaluation of its performance.

2.4 Stage 4: Delivery

The final stage involved finalizing the documentation on all the enhancements made to the system.

3. Research Problem

The first research problem involved the identification and acceleration of key computationally intensive IR simulation algorithms. These algorithms were identified by profiling the infrared scene simulation. The second research problem involved the development of an optimal reconfigurable computing architecture on which the hardware accelerated implementations can be realized.

3.1 Infrared scene simulation in real time

In table 1 below, we list the profiling results for a simulation of a single IR scene. We identify the frequency of execution of the various algorithms and their percentage of execution time.

Algorithm	Freq. of execution	% Execution time
Scintillation	143724	65.6
Random number generator (RNG)	691897304	33.7
Refraction + ray tracing	21616	0.2
Convolution for scintillation	28	0.4

Clipping	52039680	0.1
----------	----------	-----

Table 1:
Profiling results

The table above shows that the cause of slow simulation performance involves one or more of the following computations:

1. generation of large quantities of uncorrelated random numbers for modelling a number of atmospheric and optical phenomena like atmospheric turbulence and refraction, blurring due to lens optics and photon noise effects
2. scintillation of infrared sources due to turbulence in the atmosphere
3. simulation of the effects of optical aberration blurring through the use of convolution operations at the image sensor plane

However, it must be noted that, although modelling the effects of atmospheric scintillation is the most computationally expensive algorithm, it requires large quantities of random numbers. In order to accelerate this algorithm in hardware it is therefore necessary to first accelerate random number generation. We have developed multiple hardware implementations of uniform random number generation in hardware. In section 4 we present this in more detail.

4. FPGA Based Pseudo Random Number Generators

When simulating IR scenes that take into account the effects of a turbulent atmosphere and of CCD camera sensor electronic noise, each typical IR 352×352 scene generated by the simulation requires 691×10^6 Gaussian random numbers [5]. A real time simulation sequence at 50 scenes/sec thus needs more than 35×10^9 random samples generated per second. Since a typical software uniform generator [4] can only manage 10×10^6 per second, 33,950 PCs would be needed. A key requirement of IR scene simulation is the necessity to generate large sequences of random numbers on the provision of single a seed. It has been recommended in [6] that in order to prevent possible correlations between output sequences in parallel implementation of the same algorithm using different initial seeds, it is necessary to use a random number generator that has a period greater than 2^{200} . By surveying literature we have identified that TT800 pseudo random

number generator meets the above mentioned requirements and has not yet been accelerated using FPGAs. In this section we present multiple hardware implementations of the TT800 random number generator. Our best hardware implementation, the 24 port TT800, achieved a throughput of 4.6×10^9 uniform random numbers per second. It has the lowest area time metric and only half the area requirement than the previously best published multi-port, single seed generator with at least a 2800 period.

4.1 TT800 Algorithm

The TT800 algorithm is a variant of the twisted generalized feedback shift register. It is based on the linear recurrence relation given below, which consists of a sequence x_0, x_1, x_2, \dots of w -bit integers (known as a TGFSR sequence) where x_i is a row vector of bits and \oplus is a bitwise exclusive-or operation. In the equation given below, x_{n-N}^U implies the upper $(w-R)$ bits of x_{n-N} , while x_{n-N+1}^L implies the lower R bits of x_{n-N+1} , A is a w by w bit array and $|$ implies a concatenation operation. For derivation and proof of the formula see paper [4].

$$x_n = x_{n-(N-M)} \oplus (x_{n-N}^U | x_{n-N+1}^L) A$$

The origins of the TT800 algorithm are from the tausworthe generator, which was a multiplicative recursive generator that produced a polynomial. The period of this polynomial, which was irreducible, depended on the characteristic of the polynomial. The period is the smallest integer n for which x^n+1 is divisible by the characteristic polynomial. The polynomial has the following form:

$$x_{n+1} = (A_1 x_n + A_2 x_{n-1} + \dots + A_k x_{n-k+1}) \bmod 2, \text{ where } x_i, A_i \in \{0, 1\} \text{ for all } i.$$

Although this algorithm produced uniform random bits, it was slow. This algorithm was later revised by Lewis and Payne, who developed a variant known as the generalised feedback shift register.

$x_i = x_{i-p} \mathbf{xor} x_{i-q}$, where each x_i is a vector of size w with components 0 or 1.

The maximal possible period of $2p-1$ of this generator is achieved when the primitive trinomial $x^p + x^q + 1$ divides $x^n - 1$ for $n = 2p-1$, for the smallest value of n . The maximum period can be achieved by selecting n as a Mersenne Prime.

4.2 Single Port Hardware Implementation of TT800

This section describes our first hardware implementation of TT800 which we call the single port version. Generation of random numbers is carried out in 3 stages, namely the seed initializer, seed value modulator and output generator. Typically the user provides one number as a seed, however the TT800 algorithm requires a pool of 24 seeds, so that the initializer stage generates 24 seeds of the single input from the user. In stage two (the seed value modulator), which is the core of the algorithm, two values $\text{seed}[i]$ and $\text{seed}[i+7]$ are read from the pool and based on the computation defined in the algorithm,

seed[i] is updated. In the final stage, the output generator reads one of the pool values and generates the output uniform random number from this value.

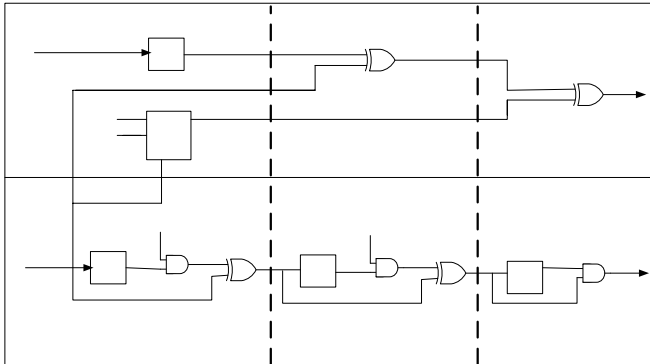


Figure 3: Internal logic of stage 2 and stage 3

The logic used to generate values out of stage 2 and 3 is shown in figure 3. The simplest form of parallelism for TT800 is to perform stage 2 and 3 in parallel as illustrated in figure 1. Note that it is not possible to more finely pipeline the output generator because its processing rate is tied to the seed value modulator, which can only be pipelined into 3 stages. If the data comes from a dual port on chip memory, Block RAM (BRAM), only one value can be read and one written in the same clock cycle. Since we need two values to be read, we use 2 dual port BRAMs. We then need logic to decide which BRAM to write to. The write back selection logic forms another stage in the seed value modulator, which now has 4 stages. In figure 4a, mag1 and mag2 are constants given in the algorithm definition.

4.3 24 Port TT800

Previously BRAMs were used to store the initial seeds. In our attempts to parallelize TT800, the seeds had to be distributed across many different BRAMs. The problem with such an approach was that the address generation for reading the relevant seeds from the various BRAM locations and writing of the values to the respective BRAM locations made the design area inefficient, see [6].

So the BRAMs are replaced with registers because each BRAM would hold only one value. A careful examination of the addressing scheme shows that the seeds can be divided into 7 groups in such a way that there is no need for the logic in one group to access the seeds in another group. We call these groups seed pools and these are shown in figure 4a below. There are two types of seed pools, those which have four concurrent seed value modulators and generators attached and those that have three attached. We present a generic model which makes use of each of these seed pools to modify the seed value and generate new random numbers per clock cycle. On each seed pool 4 TT800 instances work together to modify each seed value and generate a new one. This is illustrated in figure 4b below. No registers are shared by more than two reading channels and one writing channel, from the point of view of circuit speed and complexity. This results in simpler, smaller and faster register access logic.

Output generator

seed[i+7] >> 1

mag1
mag2 mux

0x2b5b2500

seed[i] << 7

Stage 1

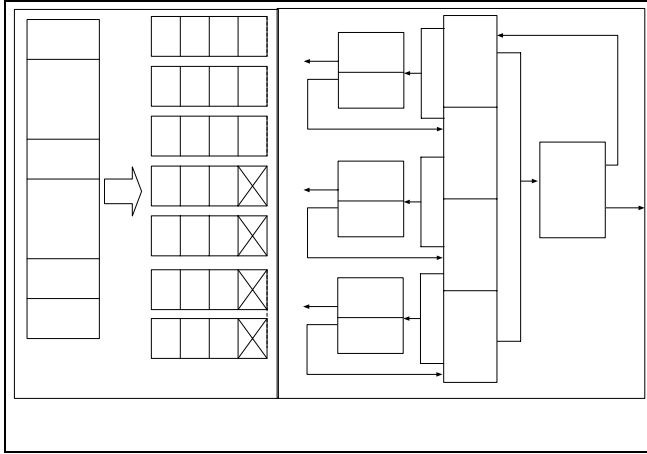


Figure 4: 24 port version

Seed 24

Seed 1 Seed 8 Seed 15 Seed 22

4.4 Results

As a preliminary test, the outputs of the hardware implementations were successfully verified against the output of the software implementation. For a more complete test, all of the hardware implementations have been tested using the die hard test. The die hard test produces p values in the range [0,1). The p values are to be above 0.025 and below 0.975 for the test to pass. All the implementations pass this test, see [7].

Seed 6

Table 2: Period, area, time and throughput comparisons

	Period	Xilinx Slices	LUTs	Clock (MHz)	Area (slices x per number x10 ⁻⁶)	Time (sec) x10 ⁻⁶	Seed 1	Seed 8	Seed 15	Seed 22
TT800 [This work]							Seed 5	Seed 12	Seed 19	
Single port	2 ⁸⁰⁰	81	-	319	0.34	0.24				
3 port	2 ⁸⁰⁰	132	-	287	0.022	0.6				
24 port	2 ⁸⁰⁰	253	-	280	0.05	4.6	Seed 6	Seed 13	Seed 20	
Software ¹	2 ⁸⁰⁰	-	-	2800	-	0.097				
MT19937 [10]	2 ¹⁹⁹³⁷	420	-	76	5.5	0.076				
MT19937 [3]							Seed 1	Seed 2	Seed 14	Seed 21
SMT	2 ¹⁹⁹³⁷	298	-	128.02	2.32	3.7				
PMT52	2 ¹⁹⁹³⁷	5,700	-	71.63	1.52	8.2				
FMT52	2 ¹⁹⁹³⁷	22,926	-	157.6	2.8	2.2				
PMT52_in	2 ¹⁹⁹³⁷	5,827	-	64.24	1.8	5.8				
FMT52_in	2 ¹⁹⁹³⁷	5,925	-	74.16	1.54	1.1				
LUT [11]										
4-tap,k=32	2 ³²	-	33	309	0.06 ²	0.3				
4-tap,k=64	2 ⁶⁴	-	65	310	0.05 ²	0.6				
4-tap,k=96	2 ⁹⁶	-	97	298	0.05 ²	1.1				

Seed Pool 7 Seed Pools
 a. Storage of seeds in 7 pools

4-tap,k=128	2^{128}	-	127	287	0.05^2	1.8
4-tap,k=256	2^{256}	-	257	246	0.06^2	1.8
4-tap,k=1248	2^{1248}	-	1249	168	0.09^2	1.9
3-tap,k=32	2^{32}	-	33	302	0.06^2	0.31
3-tap,k=64	2^{64}	-	65	319	0.05^2	0.64
3-tap,k=96	2^{96}	-	97	308	0.06^2	1.2
3-tap,k=128	2^{128}	-	127	287	0.06^2	1.7
3-tap,k=256	2^{256}	-	257	243	0.07^2	1.9
3-tap,k=1248	2^{1248}	-	1249	173	0.09^1	6.7

In table 2, we compare our designs on the basis of area time rating and throughput with other published FPGA implementations of uniform random number generators. In contrasting these solutions we take into account the amount of total resources used, including slices, LUTs and flip flops. From table 2 it can be seen that there are only three random number generators from current literature that are able to achieve a throughput of greater than 4.6×10^9 random numbers per second. Those are the FMT52, 4-tap, k=1248 and 3-tap,k=1248 random number generators. However, the price they pay in achieving this high throughput is the area they consume. The area time rating of FMT52 is 1.54×10^{-6} and the other two are 0.92×10^{-9} . The design presented herein has an AT rating of only 0.55×10^{-9} for a throughput of 4.6×10^9 random numbers per second. A further criticism of [11] is that the specialized feedback register matrix used in the implementation was not completely published.

5. A Multi-FPGA Hardware Acceleration Platform

Advanced computing based on programmable logic provides a platform for parallel computing with complete freedom of choice about the granularity and structure of the parallel algorithm.

It appears that a multi-FPGA hardware acceleration platform may be better suited for large scale IR scene simulation. A fast simulation would benefit from the concurrent processing of multiple algorithms at the same time. Thus, by scaling the hardware implementations of key IR algorithms across an array of FPGAs, faster computation may be achieved. This will be investigated during performance evaluation. Our proposed multi-FPGA system consists of one PCI-FPGA chip and four FPGA chips dedicated for hardware processes all interconnected on a single processor board, as illustrated in figure 5. Such a system requires a sophisticated memory hierarchy, bus structure and communication network [1]. For this purpose we propose the development of a multi-FPGA cluster execution environment.

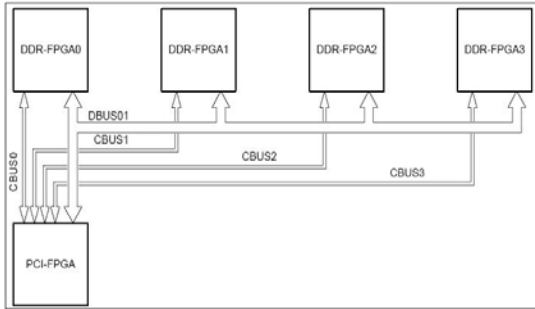


Figure 5: Cluster of FPGAs

Acceleration of IR scene simulation using a multi-FPGA cluster requires the transfer of large amounts of data to and from the cluster. A constraint in such a setup would be slow speed data transfer between the cluster and software. The likely cause of this problem is latency imposed by the operating system during data transfer. We wish to investigate operating system independent data transfer using remote direct memory access (RDMA). Furthermore, there currently exists a need for an efficient communication scheme between the microcontroller, FPGAs, external memory and the system bus [3]. For this purpose we propose the development of a standardized execution environment.

The standardized execution environment we developed comprises of the cluster middleware, multi-FPGA cluster daemon and a custom built device driver, as illustrated in figure 6. The middleware is responsible for path setup, resource sharing, RDMA, task scheduling and network management. The FPGA cluster daemon abstracts the cluster of FPGAs as a set of individual processes. We have constructed such a prototype architecture.

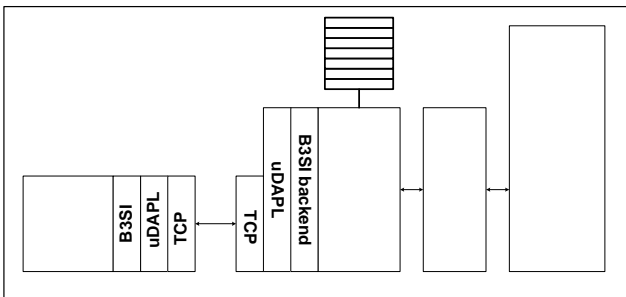


Figure 6: Accelerating random number generation over a multi-FPGA cluster

The current setup is such that the hardware provides IR algorithms a constant supply of uniform random numbers. The result of this measure is that we have been able to reduce the computation time by over 11 minutes. It now only takes approximately 21.4 minutes to simulate the same IR scene as before. In table 3, we present the profiling results of new simulation. Random number generation is now no longer a major computation burden on the simulation.

Table 3: New profiling result

Algorithm	Freq. of execution	% Execution time
Scintillation	143724	92
Refraction + ray tracing	21616	3
Convolution for scintillation	28	4
Clipping	52039680	.6
Random number generator (RNG)	691897304	.4

We believe that greater performance benefits can be achieved by developing a hardware implementation for the scintillation algorithm. For example, acceleration of convolution using FPGAs has been widely published [5], [8].

6. Concluding Remarks and Future Work

The results presented in this paper are important in that we have been able to demonstrate that random number generation has benefited from hardware acceleration. In this manner the other identified computationally intensive IR scene simulation algorithms can be accelerated using FPGAs. Also, we have been able to deploy and access the FPGA devices of the hardware acceleration platform. We have accelerated the 24-port TT800 hardware random number generator on this platform.

The hardware acceleration platform and the middleware presented above are useful in that this platform could be further developed to act as a massive hardware pipeline. In such a design, only the initial IR scene simulation data is fed to the beginning of the pipeline and the completed scene may be collected from the last FPGA of the hardware acceleration platform.

7. References

- [1] Bhatia, D. "Reconfigurable Computing". Proceedings of IEEE, 1996.
- [2] Compton, K. and Hauck, S. "Reconfigurable Computing: A Survey of Systems and Software". ACM Computing Surveys, 2002, pp. 171-210.
- [3] Konuma, S and Ichikawa, S. "Design and Evaluation of Hardware Pseudo-Random Number Generator MT19927". IEICE Transactions of Information and Systems, pp.2876-2879, December 2005.
- [4] Matsumoto, M and Kurita, Y. "Twisted GFSR generators II" ACM Trans. On Modelling and Computer Simulation, 4 (1994), 254-266.
- [5] Kim, W, Hong, K and Shim B, "Novel approach for high speed convolution". In Proc SPIE: Intelligent Robots and Computer Vision Algorithms, Techniques and Active Vision, SPIE 2001.
- [6] Srinivasan, A and M. Ceperley. D. "Testing parallel random number generators", Parallel Computing, v.29 n.1, p.69-94, January 2003.
- [7] Sriram, V and Kearney, D. "A high throughput area time efficient uniform random number generator based on the TT800 algorithm". In Proc of IEEE

- International Conference on Field Programmable Logic and Applications, Amsterdam, Netherlands, August 2007, IEEE Press.
- [8] Sriram, V and Kearney, D. "A FPGA Implementation of Variable Kernel Convolution", In Proc of IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies, Adelaide, Dec 2007, IEEE CS Press.
- [9] Sriram, V and Kearney, D. "High Speed High Fidelity Infrared Scene Simulation Using Reconfigurable Computing". In Proc of IEEE International Conference on Field Programmable Logic and Applications, Madrid, Spain, August 2006, IEEE Press.
- [10] Sriram, V and Kearney, D. "An Area Time Efficient Field Programmable Mersenne Twister Uniform Random Number Generator". In Proc of International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, USA, June 2006, CSREA Press.
- [11] Thomas, B and Luk, W. "High quality uniform random number generation through LUT optimised linear recurrences". In Proc of IEEE International Conference on Field-Programmable Technology, Singapore, Dec. 2005, IEEE Press.

Acknowledgements

The authors would like to acknowledge the funding provided by BAE Systems Australia and useful discussions with Mr. Gerry Smith of BAE Systems Australia.

Authors Biographies

Vinay Sriram is a research fellow at the Defence and Systems Institute. For the last three years Vinay has been actively involved in the development of a multi-FPGA hardware acceleration platform and in the acceleration of many computationally intensive algorithms over this platform. In particular he has been involved in the development of high performance defence applications (like that of IR missile approach warning systems and IR threat simulation) that make use of this reconfigurable hardware. In the past Vinay worked as a research engineer for the Defence Science and Technology Organisation and as a system support engineer for Sun Microsystems.

David Kearney is an associate professor at the University of South Australia. He is the program director for research degrees. Dr. Kearney's research has focused in the last ten years on computer hardware and reconfigurable computing. David has published more than 60 refereed papers in journals and conferences. In reconfigurable computing David has published papers particularly related to run-time partial reconfiguration of applications and the design of high bandwidth hardware platforms for reconfigurable computing. For the past three years he has been working on reconfigurable computing operating systems for UAVs. He is currently the director of the Sir Keith and Sir Ross Smith Reconfigurable Computing Laboratory within the Advanced Computing Research Group at the University of South Australia.