High Performance Space Computing

John W. Rooks, Dr Richard Linderman
Information Directorate
Air Force Research Laboratory, USAF
Rome, NY 13441
{rooksj, lindermanr}@rl.af.mil

Abstract—This paper describes a fully programmable 6 processor die that was implemented in IBM's 130 nanometer 8SF process. It is capable of operating as two triple voted processors each with 6 Mbytes of Embedded Dynamic Random Access Memory (EDRAM) or 6 independent processors each with 2 Mbytes of EDRAM. In triple vote mode the processor counts and recovers from single event upsets. It supports external Double Data Rate (DDR) Synchronous Dynamic Random Access Memory (SDRAM), and has two Spacewire ports, a 4 GBit input port, and a 4 Gbit output port. The processor core with memory performs approximately 2.5 Giga Floating Point Operations Per Second (GFLOPS) per Watt, with worst case input/output power its performance is approximately 1 GFLOPS per Watt. The processor's efficient messaging allows hundreds of processors to be applied to applications such as radar[1][2].

TABLE OF CONTENTS

1. INTRODUCTION

Space processing presents some technical challenges beyond those faced by ground based systems. Radiation hardening, reliability, and power efficiency are much more important than for ground systems. Sensors such as radars can generate tremendous amounts of raw data. This data can be greatly reduced with on-board processing, resulting in a more robust system. With on-board processing the satellite can have a much lower downlink bandwidth and can respond to sensor data with lower latency.

Previously, the Air Force Research Laboratory's Information Directorate (AFRL/IF), had designed a fully programmable processor with record setting power efficiency.[1,2] This processor, while capable of standard operations, was highly optimized for power efficient signal processing functions, making it suitable for the intensive processing needs of satellite systems, such as radars and imaging sensors.

Over the course of several programs and five years, AFRL's Information Directorate, Sensors Directorate, and Space Vehicles Directorate studied the challenges of a Space Based Radar. As a result, AFRL had the knowledge of what was needed in terms of processing algorithms (Sensors Directorate), the best approach to implementing a high performance computer (Information Directorate), and how to radiation hardening the processor (Space Vehicles Directorate). Lessons learned from large DOD acquisitions with extensive on-board processing requirements also informed the team.

Using the knowledge gained from studying space radar and the existing processor core, a multi-processor embedded high performance computer was designed and fabricated. The multi-processor chip is very scalable allowing hundreds of chips to be used together. This paper begins by describing the resulting processor and support software. Test results for two benchmarks are then presented prior to illustrating how a processing task can be readily distributed across hundreds of processors for a radar Moving Target Indicator (MTI) example. The processor is applicable to a broad range of applications besides the radar example. It is particularly well suited to applications that perform a large amount of floating point digital signal processing.

2. PROCESSOR

A six processor chip was built through IBM's 8SF process, CU-11 cells and IBM's Application Specific Integrated Circuit (ASIC) flow. IBM's 8SF process uses 130 nanometer (nm) technology, copper interconnects and has EDRAM available. The chip design, fabrication, packaging, and testing was completed within seven months. This rapid turn around was possible due to the maturity of the VHDL (synthesizable hardware description), (90% complete), dedicated effort by AFRL/IF, ITT Corporation and IBM, and IBM's ASIC flow methodology. The extremely short schedule prevented all of the desired features from being added to the processors, but the basic

1
[1] U.S. Government work not protected by U.S. copyright.
[2] IEEEAC paper #1327, Version 11, Updated October 24, 2006.

features needed for space operation, (i.e. triple voting and Error Detection And Correction, (EDAC)), were included.) A ceramic column grid array package was selected to improve the processor's reliability in a temperature cycling environment, see figure 1. The relatively tall columns allow the chip and the board that it is mounted on to expand at different rates with temperature changes. Figure 2 is a block diagram of the FPASP7.0 chip.

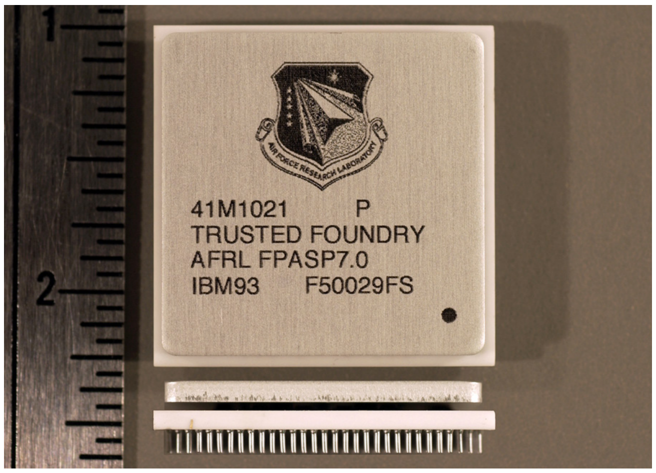

Figure 1: FPASP7.0 Column Grid Array

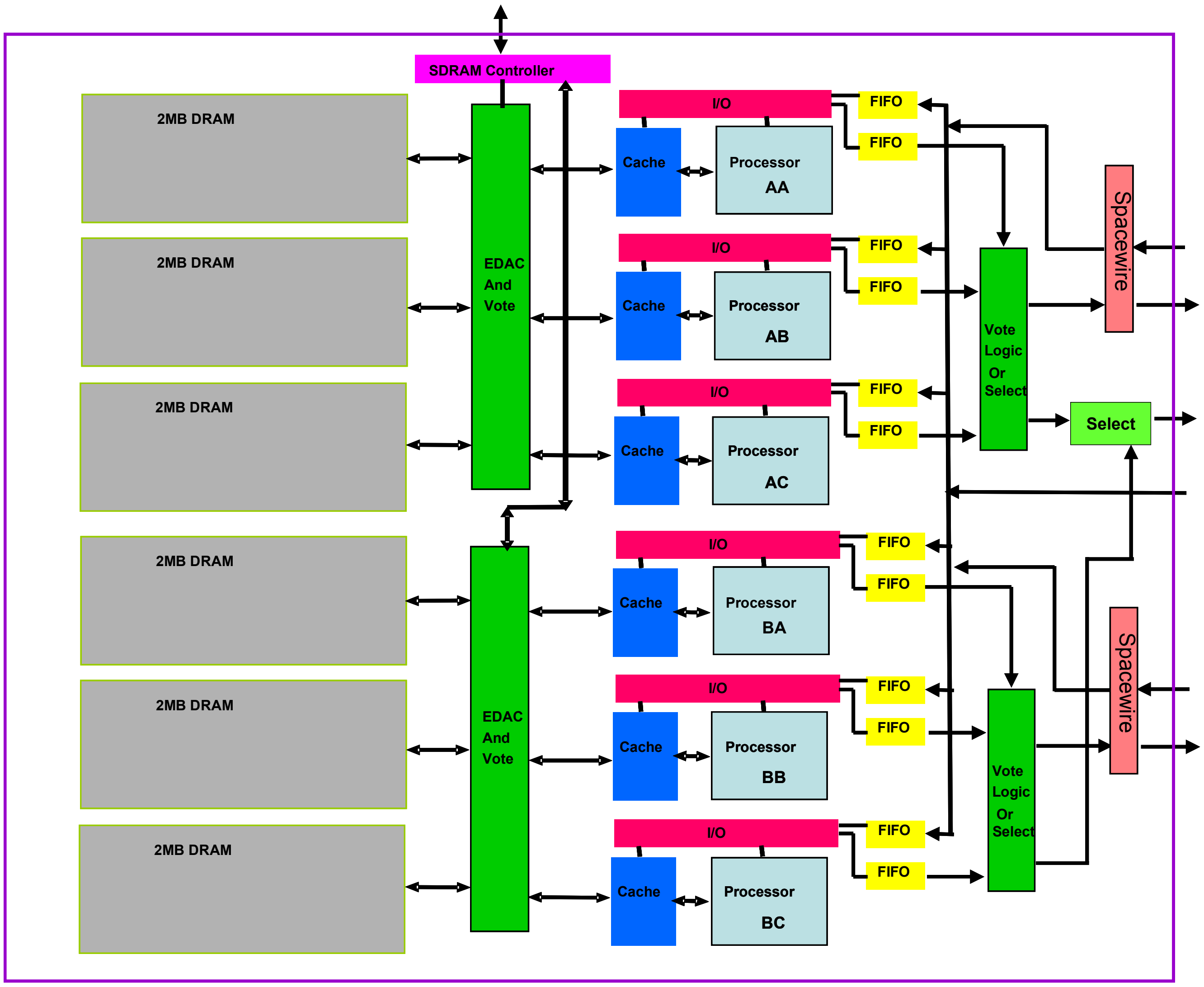

Figure 2: FPASP7.0 Block Diagram

Processor core—The core of the processor is the Floating Point Application Specific Processor (FPASP). The FPASP is a relatively simple architecture with 3 pipeline stages for floating point operations. The FPASP has a complete set of general purpose instructions, numerous digital signal processing (DSP) instructions, and the ability to add new instructions during operation. The processor can perform two 32 bit floating point multiplies and two 32 floating point add/sub per clock cycle or one 64 bit multiply and one 64 bit add/sub per clock cycle. The processor core has a memory interface similar to a synchronous static memory, but it can accept a ready signal from the memory interface which allows a cache or other device to stall the processor during a memory request. The VHDL is 100% synthesizable, though typically it is optimized to the selected process by using macros for some of the registers arrays.

Embedded Dynamic Random Access Memory—EDRAM is desirable due to the reduced leakage current relative to SRAM, reduced Single Event Upset (SEU) rate due to its large capacitance, and because it consumes much less chip area than static memory. For these reasons, an EDRAM was used with the processor. The EDRAM is 592 bits wide, (512 data bits, 64 error correction bits, and 16 unused bits). An 8 bit error correction code is stored with each 64 bit double word within the 512 data bits. During a read, any single bit errors are corrected prior to being written into the cache. The cache line is then marked dirty ensuring it is eventually written back to the EDRAM. Double bit errors are detected but cannot be corrected. Single and double bit errors each cause a unique maskable interrupt to the processor.

Cache—There are multiple pipeline delays accessing the EDRAM so a cache was designed to mask this latency and to efficiently utilize the wide EDRAM. This cache has a synchronous static random access memory type interface to the processor that is 64 bits wide, with a capability to stall the processor if the data is not ready within one clock cycle. On the EDRAM side, it has a 512 bit wide interface. There are 32 cache lines each 512 bits wide. 16 of the cache lines are used primarily to hold data waiting to be written to memory that was not previously read by the processor. The cache was designed to match the EDRAM, (width and speed), to the processor, to minimize the power consumption, and to allow sustained processor operation on DSP instructions. The next sequential EDRAM address is prefetched by the cache when an existing line has a hit. This allows the cache to stay ahead of the processor requests on complex DSP instructions.

Inputs and Outputs—The primary I/O for the die is a First In First Out memory (FIFO) style physical interface, with 64 inputs and 64 outputs, as well as typical FIFO control and status signals. Each of the 6 processors has an 8 Kbyte input and an 8 Kbyte output FIFO that can be connected to the chips FIFO I/O pads allowing them each to have multiplexed access to the chip I/O. Arbitration amongst the 6 sets of FIFOs for the I/O pads is controlled off chip. State machines within each processor allow data to be transferred to and from the cache as well as to and from the processors I/O registers. In addition to raw data, the state machines can assist in formatting the data for transfer over Ethernet networks.

In Ethernet mode, the I/O output state machine will take two pointers, one to a header data and one to the payload data. After sending the header data, it will add an Ethernet Frame Check Sequence (FCS), then continue sending the payload data. Another FCS is added to the end of the payload data, which is also the end of the packet. The header FCS is used by the receiving processor's I/O state machine to verify the packet header for a special direct write type packet. Direct write packets deliver the payload directly into the source specified memory location as opposed to being buffered at another location first. This mode is helpful when dealing with high data rates and/or many short messages. The independent input and output ports allow 4 GBits per second in and 4 GBits per second out. Time limitations prevented accelerating these ports to 10 GBits/second, though it is not a difficult task.

There are two Spacewire ports per chip. Each triplet (or the A processors when in independent mode) has access to a Spacewire port. Spacewire is a European developed standard for spacecraft communication. [3] Spacewire uses two low voltage differential signaling pairs in each direction allowing moderate data rates up to 625 Mbits per second. This implementation is designed for about half of that rate. Messages into and out of triplets or A processors (shown in figure 2) can be directed to the Spacewire port by replacing the Ethernet pre-amble and start byte with the logical Spacewire address and a non-valid Ethernet start byte. The non-valid start byte causes the packet to be directed to the Spacewire port.

Other I/Os include; thirty-two generic inputs that all processors can view, thirty-two generic outputs that are divided amongst the 6 processors, two maskable interrupts that go to all the processors, and load timer inputs that cause the processors timer register to be stored in one of two registers that are used to record important timing events.

Timers—By having timer input pins, the processor is able to record its internal 64 bit timer value when either load-timer input is asserted. When operated with a timing device, such as a Global Positioning System receiver, this allows the processor to update its time in a controlled manner. Abrupt changes in time can be disruptive to many algorithms that use time. For instance, time may need to go backwards, and many algorithms don't respond well to time going backwards. So the processor can look at its own timer count at the known time of the pulse and gradually adjust the time as needed. There is never a need to change the actual time counter, just the constants that are added to it and the multiplier used on it. If the processor's clock frequency is different than expected, the processor can adjust for that and keep very accurate time between time updates. This also allows the processor to do a sanity check on the time pulse prior to using the updated time, filtering out and reporting erroneous pulses on the timing signal.

Voting and error recovery—The 6 processors can operate independently or as two triplets. When operated as a triplet, the EDRAMs for the three processors are organized as one 6 Mbyte contiguous block. All transactions between the three caches and the EDRAM are voted. Additionally, all I/O between the three I/O state machines and the I/O pads are voted. The voting logic reports any SEUs by causing a maskable interrupt to the three processors. The interrupt handler for a triple voting error causes the processors to save all of their internal data, flush the cache, and issue a soft reset. The soft reset reloads the internal data and resumes operation. Because the internal data is voted prior to being written to the EDRAM any flipped bits are corrected during the restore state that is performed after the soft reset. This makes SEUs transparent to application programmers. Only a few hundred clock cycles are lost as a result of a mis-compare.

Processor networking—The processors are designed to have 10 Gbit Ethernet crossbar switches interconnecting them. Currently the network is provided with a separate chip. The next generation part will likely have the switch integrated on-chip. Sufficient 10 Gbit ports on and off the next generation chip will allow assembly of a large processing system primarily with just that chip.

Simulation and Emulation—The processor was simulated using VHDL, a digital logic hardware description language, and a multi-processor system was emulated using Field Programmable Gate Arrays (FPGAs). The entire hardware design is written in VHDL, as well as board level simulations. The same software tools that support the hardware also support the VHDL with the same or similar load files and interfaces. A set of C and assembly language self checking programs form the regression tests. These regression tests verify the functionality of each instruction, inputs, outputs, error detection, etc. A limitation of simulation is that it runs at approximately 10 clock cycles per second. This is sufficient for our regression test suite but does not allow us to simulate running the operating system and complex applications.

An emulation of the processor; with no attempt at increasing the frequency, runs at 24 MHz, which is over a million times faster than the simulation. The emulation is run on the Heterogeneous High Performance Computer (HHPC) at AFRL in Rome, New York. [4] The HHPC is a 48 node processor with each node containing an Annapolis Wildstar II FPGA card. The Wildstar cards each hold two Xilinx 6 million gate Virtex II FPGAs with associated memory and support chips. The HHPC allows emulation of up to 96 individual FPASP processor cores with their caches and I/O. Using the emulation, we were able to run more extensive tests including various tests of random numbers to verify the floating point adder and multiplier. These random tests identified a few errors. The errors were then recreated in simulations, corrected in the VHDL and added to our standard regression tests. The emulation also identified I/O related bugs, where a similar methodology was used to simulate, correct, and add them to our standard regression tests. Emulation is sufficiently fast to allow testing to detect bugs involving complex state machine interactions, and other rarely occurring bugs.

Emulation allows early software development and debugging features that are not available on the hardware. Having the emulation allows the processor to be modified so that debugging hardware, such as buffers that trigger on certain events and record information, can be inserted. These buffers act as internal logic analyzers allowing hardware designers to monitor the internal processor signals identifying bugs that occur too infrequently to easily catch with simulation. Once the root cause of a bug is identified in emulation it can usually be recreated consistently and quickly with a short simulation.

ISA Simulator—The FPASP7.0's Instruction Set Architecture (ISA) simulator is the central piece of software in the development suite. It simulates the behavior of the FPASP processing element at the level of the assembly instruction code. The ISA simulator simulates the behavior of the FPASP down to the number of clock cycles used by every assembly instruction.

In addition to the simulation of the processing element, the ISA simulator also simulates the cache, memory, and the network through which messages are exchanged between processors and different nodes in the system.

With the capability of simulating the number of clock cycles taken by each assembly instruction, the ISA simulator can accurately profile the performance of the code. The simulator provides the user with a complete list of performance measures, such as total number of clock cycles, and the total number of vector or scalar floating point operations. The ISA simulator's profiler can be set to report the percentage of time in each subroutine relative to the total time of the whole program, or to report the number of branch calls, or cache hits/misses to get a good evaluation of the system performance.

The ISA simulator is also used as an interface to the hardware. During software development it makes all the resources of the host system available, providing file access and display services to the embedded processors. It is also possible to run a system of simulated processing elements mixed with actual hardware processors. This enables localizing the debugging process by replacing the failing hardware processor with a simulated one and watching the exact replica of the program running in the simulation environment where more information is available. The ISA simulator runs several million clock cycles per second.

The GNU compiler, assembler, and debugger have been modified to support the FPASP. [5] The debugger also comes with the DDD GUI interface. With the assistance of the debugger, the developer can monitor the status of any program. It provides the ability to step through the code in a single or multiple step modes. The debugger can also be accessed remotely through a secure shell with X-Windows capability. The remote access is done through communication with sockets. This approach reduces the response time by running the core software on the remote access machine, and exchanging only data through the remote socket.

The remote accessibility is one of the major advantages of the software development tools of the FPASP environment. Users can run and debug their applications without having to be physically on-site. This feature has been used with an FPGA emulation of the processor, allowing remote programmers to emulate on multiple FPGAs over the internet.

The Real-Time Executive for Multi-Processor Systems (RTEMS), operating system has been ported to the FPASP. [6] This open operating system, along with a choice of several message passing middleware options completes the processor's entirely open software suite.

Evaluation Board—An evaluation board, (see figure 3), is available for the processor. The board has numerous I/O options including 10/100 Ethernet, 10 GBit Ethernet, Spacewire, USB, RS-232, LEDs, and header pins. Currently the 100 MBit Ethernet port is used by the ISA simulator to communicate to the processors. An FPGA sits between the processor and the 10/100 chip adapting the data width and setting up the 10/100 chip.

Test results—The processor has been exercised on its evaluation board. The regression tests all pass. The RTEMS operating system with various test programs such as infinite print tests and message passing tests all pass. Benchmarks have been performed with several common DSP functions.

The power consumption for all 6 processor cores, caches, and EDRAMs, running at 125 MHz and performing almost 3 Giga Floating Point Operations Per Second, (GFLOPS), is about 1 watt. The current evaluation board has several hundred resistors that ensure the signal integrity of the high speed Double Data Rate (DDR) signals. Not all of these resistors are needed and some applications would not use any of them, resulting in reduced I/O power consumption. I/O power consumption with the extra resistors is about 2 Watts.

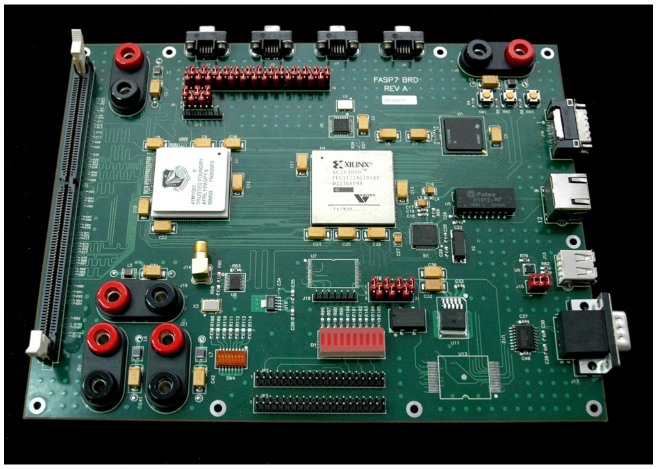

Figure 3: FPASP7.0 Evaluation Board

Table 1 shows the power efficiency of a 4096 point complex single precision FFT. Results are given for various combinations of voted and independent processors. On this FFT, the cache stalls the processor for a total of 18% of the clock cycles. Also, the two multipliers and two adders are not fully utilized due to the inherent imbalance in the ratio of multiplies and adds required for an FFT. The processor implements a radix-4 FFT. The 4096 point FFT performs a total of 245,760 floating point operations. That is an average of 2.52 floating point operations per clock cycle, giving a 63%, (2.52/4), utilization rate. Since FFTs can be computed in several ways and different processors use different approaches, usually the flops count is calculated based on the radix-2 butterfly which is less complex to implement but requires more floating point operations. If the processor is credited with the higher radix-2 floating point operations count, the result is 2.97 operations per clock cycle or a 74% utilization rate. The results are presented in Table 1 as actual and normalized to radix-2, i.e. for 6 independent processors the power efficiency excluding I/O power is 1.60/1.89 GFLOPS/Watt.

Results for 4096 point FIR_8 instruction are shown in Table 2. FIR_8 performs a finite impulse response filter, with 8 complex filter coefficients. For this instruction, 3.5% of the clock cycles are wait states caused by the cache. Including cache induced wait states, the processors averages 3.86 floating point operations per clock cycle for this instruction.

Table 1: 4096 Point FFT Power Efficiency

Mode	Core Power (Watts) at 1.2V	I/O Power (Watts) at 2.5V	GFLOPS /Watt for the Proc, cache, EDRAM	GFLOPS /Watt for including I/O
Processors off, clocks on, EDRAM Refresh on	0.56	2.00	N/A	N/A
1 Indep.	0.65	2.00	0.40/0.48	0.11/0.13
1 Triplet	0.76	2.00	0.35/0.41	0.11/.013
1 Trip. and 3 Indep.	1.01	2.00	1.04/1.23	0.39/0.46
6 Indep.	1.06	2.00	1.49/1.75	0.58/0.68

Table 2: 4096 Point FIR_8 Filter Power Efficiency

Mode	Core Power (Watts) at 1.2V	I/O Power (Watts) at 2.5V	GFLOPS/ Watt for the Proc, cache, EDRAM	GFLOPS/Watt for including I/O
Processors off, clocks on, EDRAM Refresh on	0.56	2.00	N/A	N/A
1 Indep.	0.62	2.00	0.65	0.18
1 Triplet	0.73	2.00	0.55	0.17
1 Trip. and 3 Indep.	0.91	2.00	1.77	0.62
6 Indep.	0.96	2.00	2.51	0.92

3. MULTI-PROCESSOR MTI EXAMPLE

A challenging moving target indication (MTI) problem requiring on the order of 100 billion floating point operations per second (GFLOP/sec) sustained throughput can serve as an example of how real-time radar signal processing algorithms can be mapped onto collections of the low power FPASP processors, in this case 256 chips.

In this example there are 16 channels of incoming data, (from 16 receivers). We assume the A/D converters are sampling at 50 MHz and a 40 km window in range is to be processed (13333 range bins). The Coherent Processing Interval (CPI) consists of 128 pulses and the pulse repetition frequency is 1 KHz. The data set is often referred to as a data cube. The general flow of processing steps follows the PRI-staggered post Doppler approach to space-time adaptive processing (STAP) as discussed in [7,8]. Steps are as follows: 1) pulse compress in range, 2) Doppler process, 3) calculate beamforming weights with PRI-staggered STAP, and 4) form multiple adaptive receive beams and make detections.

The incoming data cube for each CPI consists of 16 channels, 13333 range cells, and 130 pulses, see figure 4. This cube is sent to a clique of 16 FPASP chips nominally collocated on a board, see figure 5. Each chip receives the input data from one channel totaling 6.9 MBytes. The chips are operating as a pair of triplets for fault tolerance and both triplets work together on the incoming data, which is placed in the shared DRAM. Across the 16 chips, 111 Mbytes is input per CPI, corresponding to a real time flow of 866 Mbytes/sec from the sensor to the embedded processor which contains 16 cliques.

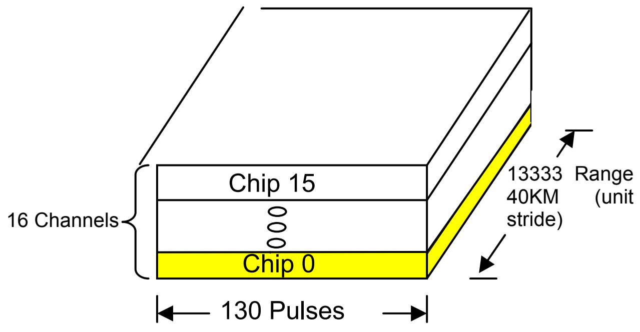

Figure 4: Data Partitioning for Range-Doppler Processing

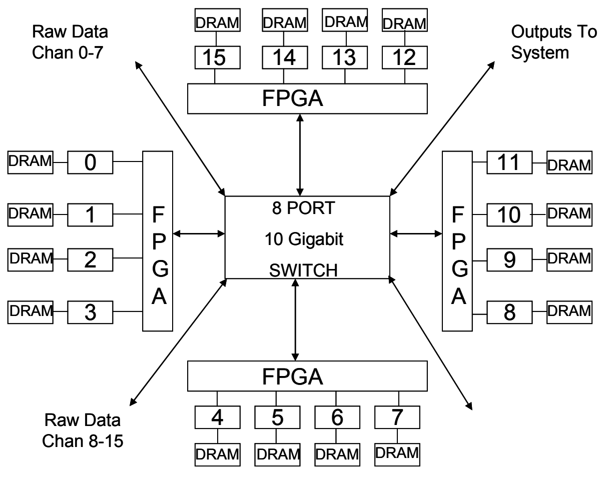

Figure 5: 16 Chip/32 Processor Board

Range processing—Range processing consists of adjusting the return signal strength to compensate for the signal reduction at increasing distances and compressing the returned pulse. First there is a multiply of the returns point by point by a real vector (2N ops) to normalize the signal strength and apply a window prior to pulse compression to improve range sidelobes. Next, to compress the pulse, a Fast Fourier Transform (FFT) (5NlogN ops) is performed followed by a complex point by point multiply (6N ops) and an inverse FFT (5NlogN). These steps cost the chip 314 million floating point operations (MFLOP) to process the assigned channel over the CPI.

After the range processing, the data must be transposed in memory (cornerturned) to prepare for unit stride FFTs across pulses which will form the Doppler bins. This is done completely within each node, using a custom instruction which cornerturns 8 cache lines (each 8 complex samples wide) at a time. The operation requires just two clock cycles per sample to read and then write each eight byte word.

Doppler processing—Since each chip contains the entire range-Doppler plane for a given channel, as shown in fig. 4, the processors can move directly into the Doppler processing without any interprocessor communications. The PRI-staggered approach to STAP allows multiple temporal degrees of freedom to be weighted in the adaptive filter to effect a cancellation very similar to the 2-pulse and 3-pulse MTI cancellation filters used historically. Here we generate 3 temporal degrees of freedom by computing three sets of 128 point Doppler FFTs, shifting by one pulse in time between each FFT, hence the input of 130 pulses in the CPI. Computing these triplets of Doppler FFTs in each of the 13333 range bins requires 180 MFLOP on each channel (chip).

After Doppler processing, each processor gathers 144 samples in each of six regions for each Doppler bin to input into the STAP processing which is spread out across all the nodes. This requires 2.5 Mbytes of output to other nodes and the same amount of input to the node performed across the 10 gigabit Ethernet link.

Space Time Adaptive processing—The STAP algorithm closely follows the flow of earlier work on real-time implementation of PRI-staggered post Doppler STAP [7,8] base with much finer range resolution and more output adaptive beams. Here we form adaptive weights to construct 16 beams finely spaced in azimuth to cover the surveillance volume. In addition, the 40 km range extent is divided, not necessarily equally, into 6 regions featuring homogeneous clutter distribution (e.g. farmland, forest, lake, city, etc). The partitioning of the range extent into homogeneous regions is an opportunity to leverage knowledge-based STAP techniques [9] which will not be discussed further here, since they do not significantly drive computational complexity beyond necessitating multiple adaptive regions.

After completing the range-doppler processing assigned, each processor in the clique extracts 144 training data samples within each of the six regions in each of the 128 doppler bins. The training data is sent to the chip within the clique assigned to compute adaptive weights for that Doppler bin. This data distribution is depicted in Fig. 6. Upon receiving channel data from each of the other 15 chips, processors begin computing adaptive weights for each of the 16 receive beams by QR factorizing 48x144 matrices augmented with beam shape constraints. This approach is a least squares solution which minimizes clutter in the output (represented by the training samples of homogeneous clutter) while preserving the desired beamshape and pointing direction. This work requires 156 MFLOP per CPI on each of the 16 chips.

Beamforming and MTI detection—With the 48 point adaptive weight vectors calculated, beamforming can proceed, but first the data from across all channels must be brought to a common location. This requires a cornerturn operation and all-to-all communication amongst the processors in the clique to reach the data cube orientation shown in Fig. 7. To reach this configuration, each chip sends and receives 38 Mbytes per CPI. Beamforming can then proceed, costing 8 operations per range-doppler point per beam. Overall, this requires each chip compute 218 MFLOPS. A detector is then run across the formed beams looking for anomalies, but this only costs on the order of 14 MFLOPS.

The total computation sums to 882 MFLOP per chip per CPI and 14.1 GFLOP across the clique of 16 chips. A targeted latency to derive detections of 2 seconds mandates 448 MFLOP/sec be sustained per chip, which is 44% of peak given the 125 MHz clock. This percentage seems reasonable with the FFTs sustaining 74% of peak and the complex dot products sustaining over 90% of peak. The dot products are key to both QR factorization for STAP and the beamforming operation. Since each clique handles 128 pulses of the 2000 pulses in 2 seconds, 16 cliques (16 boards and 256 chips/512 triplet processors) handle the real-time rate. Overall the system would sustain a computational throughput of 110 GFLOP/sec. If the total operations are divided by the number of input data points, the computational complexity of this algorithm is 517 operations per data point.

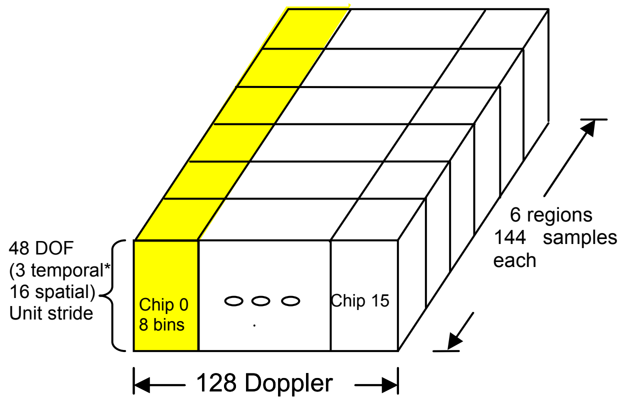

Figure 6: Data Partitioning for PRI-Staggered STAP

Overall each processor inputs 6.9 Mbytes/CPI of raw data and then inputs and outputs approximately 40 Mbytes (each direction) with other processors on its board during the computation. When the radar frontend drives inputs to this board, data flows in at 866 Mbytes/sec which is a good match to a 10 gigabit Ethernet link assuming either direct write Ethernet packets or User Datagram Protocol (UDP) packets. If needed, the data can be transferred over two channels as shown in figure 5.

4. Future Work

Future work includes continued performance testing of the FPASP7.0, radiation testing, space flight experiment support, and design of the next generation processor. New evaluation boards are being fabricated without the DDR SDRAM resistors which will allow the power consumption of the system without DDR SDRAM to be accurately quantified. Radiation testing is planned to demonstrate the tolerance of this commercial process to radiation. Flight worthy circuit boards are being designed for the AFRL/VS Plug and Play Satellite (PnPSat), and software is being ported to the FPASP7.0 to support PnPSat.

AFRL/IF intends to team with other Government agencies and/or industry to fund fabrication of the next generation processor. It is expected that the next generation processor will include on-chip networking switches allowing a large system to be composed of primarily this chip and DRAM chips when they are needed.

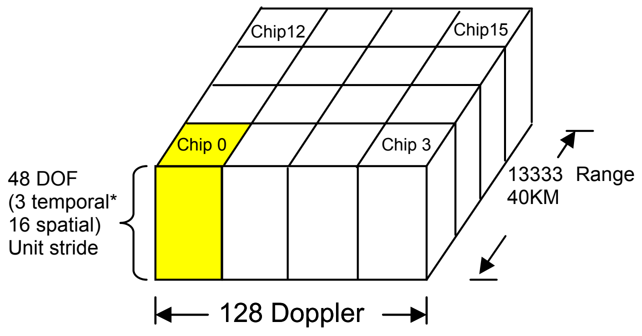

Figure 7: Data Partitioning for Beamforming

5. Conclusions

A power efficient high performance space computing building block was fabricated on IBM's 8SF line with CU-11 cells. SEU mitigation is built into the design with triple voting of three processors and EDAC on the EDRAM. The 6 processor chip will be radiation tested in the near future and plans are being made for a flight test. Power efficiency for the processor, cache and 12 Mbytes of embedded memory is close to 1 GFLOPS per watt.

The processor design and fabrication, including completing the processor's core, I/O addition, layout, and fabrication were completed within 7 months using IBM's ASIC methodology. While using the ASIC flow was restrictive compared to a full custom design it does allow the latest processing technology to be rapidly leveraged for space use where power efficiency is critical.

This processor is suitable for use as a single chip or in large multi-chip systems. The architecture with its low overhead messaging and digital signal processing instructions is well suited for high data rate floating point intensive processing. While many applications are satisfied with the 12 Mbytes of on-board memory per chip there is also a DDR SDRAM port providing additional application versatility.

References

[1] J. Rooks, "An Embedded Fusion Processor", NATO SET Sumposium on Space-Based Obeservation Technology Samos Greese, October 18, 2000.

[2] R. Linderman, R. Kohler, M. Linderman, "A Dependable High Performance Wafer Scale Architecture for Embedded Signal Processing", IEEE Transactions on Computers, Vol 47, No 1, January 1998, pp 125-128.

[3] Spacewire Web site http://spacewire.esa.int/tech/spacwire/

[4] R. Linderman et al, “Heterogeneous High Performance Computer Emulation of a Space Based Radar On-Board Processor”,HPCMP Users Group Conference, June 2004.

[5] GNU Web site http://www.gnu.org/

[6] RTEMS Web site http:/www.rtems.com/

[7] R. Linderman, M. Linderman, “Real-Time STAP Demonstration on an Embedded High Performance Computer”, IEEE Aerospace and Electronics System Magazine, Vol. 13, No. 3,March 1998, pp.15-31.

[8] A. Choudary et al, “Design Implementation an Evaluation of Parallel Pipelined STAP on Parallel Computers”, IEEE Tranactions on Aerospace and Electronic Systems, Col 36, No 2, April 2000, pp 529-548.

[9] W. Melvin et al, “Knowledge-Based Space-Time Adaptive Processing for Airborne Early Warning Radar”, IEEE AES Systems Magazine, April 1998, pp 37-42.

BIOGRAPHY

John W. Rooks received the B.S. degree in Electrical Engineering from the University of Vermont, Burlington VT, and the M.S. degree in Electrical Engineering from Syracuse University, Syracuse NY, in 1986 and 1990, respectively. He has been with the Air Force Research Laboratory’s Information Directorate, Rome NY, since 1986. Currently he works in the Advanced Computing Architectures Branch developing Embedded High Performance Computers.

Dr. Richard Linderman is the Senior Scientist for Advanced Computing Architectures, Information Directorate, Air Force Research Laboratory, Rome, NY. This area includes high performance computers, embedded computer architectures, fault tolerant architectures, distributed architectures, and next generation architectures.

Dr. Linderman is an IEEE Fellow and an AFRL Fellow. Dr. Linderman received his PhD from Cornell University in 1984 and taught at AFIT until 1988, when he joined the Rome Air Development Center which eventually became AFRL.