# A DSP ENGINE FOR A 64-ELEMENT ARRAY

S. W. ELLINGSON
The Ohio State University ElectroScience Laboratory
1320 Kinnear Road, Columbus, OH 43212 USA
E-mail: ellingson.1@osu.edu

This paper considers the feasibility of software-defined signal processing for a 64-element antenna array. In the proposed architecture, the signal from each element of the array is individually converted to a digital complex baseband format. These 64 outputs are then distributed among many DSP microprocessors, which take turns acquiring and processing the data. The processing is done entirely in software; no FPGAs or ASICs are used. The advantage of software implementation is the ability to dynamically and flexibly allocate the available computing resources to various tasks. Also, the architecture is flat as opposed to hierarchical; every DSP receives data from all antennas. A limited test of this approach using C language source code and commercial off-the-shelf hardware was conducted using the ADSP-21060 "SHARC" DSP. The following processing algorithm was considered as a test of the architecture: In-line calibration, 2D-FFT to form 64 beams, and then 64 length-256 1D-FFTs to obtain the frequency spectrum for each beam. The resulting performance equation is $F_S \cdot \text{(duty cycle)} \leq c_1 J$, where $F_S$ is the sample rate, $c_1 = 2.34 \times 10^4$ samples/s, and $J$ is the number of modular computing "clusters" used in the design. This study suggests that $c_1$ can be improved by one or two orders of magnitude using the existing hardware.

## 1 Introduction

This paper considers the feasibility of *software-defined* signal processing for large antenna arrays. This technical strategy has only recently become feasible, mainly due to recent technological advances in wideband digital receivers and DSPs. It is expensive compared to more traditional approaches. Nevertheless, it is useful when the ability to define system operation in software, and the ability for "on-the-fly" reconfiguration, are most important. An example of such an application is *space-frequency surveillance.* In this application the array is used to observe continuously over a large (potentially hemispherical) field-of-view, and is tasked to determine the joint distribution of incident signal power in the space domain (i.e., angle of arrival) and frequency domain. Applications for such a system can be found in radio astronomy, the search for extraterrestrial intelligence (SETI), parasitic bistatic radar. In SETI, for example, it is desired to detect very weak, intermittent narrowband signals with no *a priori* knowledge of frequency or direction of arrival. In this paper, it is assumed that the system includes an array of $P$ sensors, organized into a rectangular $M \times N$ grid. A design for $M = N = 8$ and $P = 64$ is considered specifically. The signal received by each sensor is processed by a dedicated receiver, which outputs digital samples in complex baseband ("zero IF") form. The following sequence of operations is then as follows:

- *Acquisition.* $L$ snapshots are captured from the array. Each snapshot consists of $P$ samples, one from each sensor. In this paper, we will consider specifically the case where $L = 256$ which (as explained below) is based on memory constraints.
- *Calibration.* In practice, the acquired snapshot data will be distorted by various factors including mutual coupling and unequal electronic delays. This distortion can be reduced by multiplying the $P \times 1$ vector of samples from a snapshot by a $P \times P$ calibration matrix. In the absence of mutual coupling, the calibration matrix reduces to a diagonal matrix, so that this step can be performed more efficiently as a length $P$ inner ("dot") product. In practice, it is anticipated that there will be strong coupling between adjacent elements but that more distant elements will be only weakly coupled. In this case, the

calibration matrix has a band-diagonal structure, and the required processing effort falls somewhere between that required for an inner product and that required for a full matrix multiply.

- *Spatial Processing.* A two-dimensional (2D) Fast Fourier Transform (FFT) of size $M \times N$ is applied to each snapshot. The $P$ bins now represent beams pointing in various directions.
- *Temporal Processing.* A length $L$ one-dimensional (1D) FFT is applied to all the samples associated with one beam. This is repeated $P$ times – once for each beam.
- *Storage.* The resulting set of $P$ frequency spectra are stored in preparation for the next iteration of processing. The data in storage is expected to be passed along for further processing, analysis, and decision making.

There is one additional issue – interference suppression – which is relevant but not addressed in this paper. However, assuming an interference suppression algorithm is implemented on an external processor, any spatial processing required can be implemented as a modification to the calibration matrix.

The motivation for a software-defined architecture in this application is as follows: It is a characteristic of most surveillance systems that they spend most of their time doing routine and tedious processing; in effect, waiting for something interesting to happen. An "interesting" event in this case is the emergence of a signal with unexpected direction or frequency. When this occurs, it may be desirable to allocate a larger portion of the available computing resources to study the new signal. For example, it may be desirable to abandon surveillance of some portion of the space-frequency domain and invest the additional resources in more computationally-intensive operations that would not normally be required: for example, calculating improved direction of arrival estimates, attempting to demodulate the signal, implementing more aggressive interference suppression algorithms, and running self-calibration routines and diagnostics to ensure the signal is not the result of interference or a malfunction. In a software-defined architecture, this rearrangement of priorities can be done in an elegant, flexible, and dynamic manner. This is as opposed to traditional "stovepipe" architectures, in which such a change in operation would require additional suites of equipment and/or downtime to reconfigure the equipment.

The following additional issues are considered in this paper.

- *Bandwidth/Duty Cycle Tradeoff.* The achievable bandwidth will be constrained by both the available computing resources (i.e., floating-point operations per second, or FLOP/s) *as well as* the available input/output (I/O) bandwidth. In this paper, it is shown that for continuous operation (100% duty cycle), a sample rate up to 10 million samples per second (MSPS) is feasible, albeit at very high cost. However, this system can be implemented for a dramatically reduced cost by trading off duty cycle and/or bandwidth. For example, 24.3 thousand samples per second (kSPS) can be achieved continuously in real time for cost on the order of US$70K. The proposed architecture allows the tradeoff between bandwidth and duty cycle to be implemented dynamically.
- *C language source code.* Usually, programming in a high-level language such as C results in reduced processing throughput, relative to coding in a processor's native assembly language. Nevertheless, system development in C is generally much faster and more flexible than development in assembly language. For the example presented in this paper, the code appears to run at about 40% of the realizable processing throughput. However, much of this degradation appears to be attributable to specific sections of the code as opposed to being uniformly distributed across all sections of the code.
- *Scalability.* The high cost of a software-defined architecture may preclude an immediate full implementation of the system. However, the architecture proposed here is scalable,

in the sense that one may achieve a more favorable bandwidth/duty cycle tradeoff simply by adding more identical, modular processing subsystems. In this paper, the subsystems are called *clusters.*

- *Single Platform.* Typically, combinations of field programmable gate arrays (FPGAs) and DSPs result in a more efficient system design than DSPs alone. However, multi-platform system development presents additional challenges in terms of integration and the need to master multiple sets of development tools. In this initial study, the simple case of single-platform (DSP-only) implementation is considered.
- *Non-Hierarchical ("Flat") Architecture.* Signal processing for large arrays can be greatly simplified using hierarchical processing. This means that array signal processing is done in multiple stages. For example, the first stage might be small groups of $P_1$ elements. This stage has fewer ($P/P_1$) outputs than inputs. Thus, the second stage is computationally less demanding. The drawback of this approach is that information is lost at each stage in the processing. For example, it is not possible to preserve the complete, original field-of-view using this approach. By contrast, a flat architecture allows all computing resources to be applied to any part, or all, of the sensor outputs. Although this paper considers only the flat architecture, such a system could easily be combined with other system elements to form a hierarchical architecture, or to act as a flat "overlay" to an existing hierarchical system.

## 2 Development of the Architecture

Based on the following considerations, the ADSP-21060 Super Harvard Architecture Computer (SHARC), a product of Analog Devices, Inc. (ADI), was selected for this study. First, this DSP has a large I/O bandwidth: It can transfer up to 240 megabytes-per-second (MB/s) across its native microprocessor bus, has six 4-bit (nibble)-wide serial link ports ("SLINKs") which each can transfer up to 40 MB/s each, and has 10 direct memory access (DMA) channels so that many of these resources can be used concurrently "in the background"; i.e., with no impact on core processing. Second, the ADSP-21060 has 4 megabits of static random access memory (SRAM) on chip. This allows data to be brought on-chip in large blocks, where it can be more efficiently accessed for signal processing. This in turn reduces contention for the microprocessor bus. Third, this device is a floating point processor, which is highly desirable for this application due to numerical dynamic range considerations. Finally, the ADSP-21060s used in this design are clocked at 40 MHz and are claimed by ADI to be capable of up to 80 million FLOP/s (MFLOP/s), sustained. In practice, such ratings are of little value since actual performance is highly dependent on coding details and I/O considerations. Nevertheless, this DSP is generally considered to be among the fastest in its class. It should be noted that ADI and Texas Instruments have recently released the next generation of these parts, which have superior performance. The newer parts were not considered for this study because they are still relatively expensive, and development tools for these parts are not as mature.

In space-frequency surveillance, each sensor outputs a digital complex baseband signal at $F_S$ samples per second. It is assumed that each sample consists of a 16-bit in-phase "I" word plus a 16-bit quadrature "Q" word, for a total of 4 bytes per sample, which is a common output format among digital receivers. The set of $P$ samples for a given instant is called a "snapshot", which consists of $4P$ bytes. Thus, the aggregate output data rate of the array is $4PF_S$ bytes per second.

The highest-bandwidth route into the SHARC is through it's local bus, which runs at 240 MB/s. If a single bus is used, $F_S$ must be less than 1 MSPS for $P = 64$. Another factor is that the sensor data must be time-division multiplexed onto the bus. The bus will thus have more than $P$ physical access points, which is difficult to implement due to mechanical and electromagnetic considerations. The next-highest bandwidth routes into the SHARC are the SLINKs, which run at 40 MB/s each. Each SHARC has 6 SLINK ports. For various reasons, such as the availability of DMA channels and physical layout considerations, it is difficult to use more than 2 SLINKs at a time. Let us define a *cluster* as a group of $K$ SHARCs working together to acquire the output of an array. Using SLINKs, a $P = 64$ array could be serviced by a cluster of $K = 32$ SHARCs, with $F_S$ up to 10 MSPS. Each SLINK consists of a 14-pin ribbon cable, and thus is electrically and mechanically simple to implement. Also, the conversion from 32-bit sample words to a sequence of 8 4-bit nibbles suitable for transmission on a SLINK is relatively simple compared to a bus interface. Although the SLINK approach appears promising, two problems emerge. First, note that each SHARC would receive only 2 sensors' worth of data; thus, a follow-up procedure is required to allow each SHARC to have access to all the sensor data necessary for a complete block of $L$ snapshots. The second problem is that at the maximum transmission rate of $F_S = 10$ MSPS, each SHARC would be able to do little more than simply absorb samples until internal memory was filled. There would be no time remaining to perform signal processing. Solutions to these problems are suggested below.

The requirement that each SHARC have access to all the sensor data for a given block of $L$ snapshots implies a high-bandwidth connection between a given SHARC and every other SHARC in the cluster. For a group of 6 or less SHARCs, the solution is simple: this many SHARCs can share the 240 MB/s SHARC bus. Moreover, the internal memory of each SHARC on the bus lies within the address space of every other SHARC on the bus. This special section of address space is known as the "multiprocessor memory space", and makes each SHARC's internal memory appear to be a virtual extension of every other SHARC's internal memory. To extend the number of SHARCs per cluster beyond 6, consider that six SHARCs can easily be hosted on a single board on the ubiquitous PCI bus. PCI has a nominal throughput of 132 MB/s, which equates to about 80-100 MB/s in practice, and a capacity of four boards without bridges. With bridges, the capacity can be extended to six or more boards. Thus, a $K = 32$ cluster could be implemented on a single backplane. Let us assume $L = 256$. This gives $4PL = 64$ KB (kilobytes), which is a practical goal for storage in the ADSP-21060's internal memory. Thus, a cluster should acquire $KL = 8192$ snapshots at a time, which provides enough data for every SHARC in the cluster to have a block of $L$ snapshots to process. The required time to redistribute a single block of 8192 snapshots may be as much as 25 ms over the PCI bus. This may or may not be acceptable depending on how much time is required for signal processing tasks.

The second problem with the SLINK data acquisition approach described above is that at the maximum transmission rate of $F_S = 10$ MSPS each SHARC would have insufficient time for signal processing. The cluster capture size of $KL = 8192$ array snapshots determined above requires about 819 $\mu$s at 10 MSPS. Since it is desired to perform real-time signal processing operations on this data, the DSP cluster must operate with a reduced duty cycle. That is, the cluster must be allowed to ignore input samples for a fraction of the time, so that it has time to process previously-acquired snapshots. To maintain real-time operation, additional clusters are required to ensure that that some cluster is always available to receive snapshots.

To achieve this, a "rotating acquisition" technique can be used. In this approach, the snapshot data is replicated and the full bandwidth is sent to $J$ identical clusters. However, each cluster is responsible for collecting snapshots only $100/J$ % of the time. Each cluster uses the time between collection periods to process the data received during the last collection. Assuming the number of SHARCs per cluster is $K = 32$, the required number of clusters $J$ depends on the time required for signal processing. This issue is considered in the next section.

## 3 Validation Test

It was shown in the previous section that various aspects of the architecture depend on the execution time for signal processing. This section describes a test for this purpose, using actual SHARC hardware and software. This test also serves as validation of the SHARC's ability to perform the various functions required of it in this architecture.

| Task ID | Description | Cycles | Effort |
|---|---|---|---|
| 0 | SLINK transfer | 32384 | 0.24% |
| 1 | Collect snapshots from other SHARCs in cluster | 130461 | 0.97% |
| 2 | Conversion to I-Q floating point values | 591360 | 4.39% |
| 3 | Full calibration (spatial filtering), 256 times | 9146880 | 67.95% |
| 4 | 2D (spatial) FFT, $8 \times 8$, 256 times | 2003976 | 14.89% |
| 5 | Transfer intermediate result to external SRAM | 218112 | 1.62% |
| 6 | Temporal window, length 256, 64 times | 131456 | 0.98% |
| 7 | 1D (temporal) FFT, length 256, 64 times | 931072 | 6.92% |
| 8 | Transfer final result to External SRAM | 262400 | 1.95% |
|  | Total for tasks 0 through 8 | 13448101 | 99.90% |
|  | Ultimate total, including test overhead | 13461556 | 100.00% |

Figure 1: Execution time for full calibration. Actual elapsed time was 336.5 ms.

This test was performed using a "Snaggletooth-PCI" board, a product of Bittware Research Systems, Inc. This board fits in a PCI slot in a PC and contains two SHARCs. SLINK connectors are available for both DSP-A and -B. Two additional SHARCs can be mounted on the board, each having two additional SLINKS. Thus, up to 8 external SLINKs could be supported with 2 SLINKs to each of four SHARCs. In this test, however, only the two on-board SHARCs were used. DSP-B served as the "device under test", and was set up to be as similar as possible to a DSP within the proposed architecture. Specifically, DSP-B was configured to play the role of "DSP-0" within the proposed architecture, with responsibility for sensors 0 and 1. DSP-A served several functions:

- First, it was used to generate a complete block of $KL$ array snapshots. The scenario assumed a single unmodulated carrier signal with baseband offset frequency = +1 MHz incident from $(\theta, \phi) = (30^\circ, 225^\circ)$, with the array is oriented in the $x - y$ plane and the $z$ axis corresponding to $\theta = 0$. No attempt was made to include noise. The sample rate was 10 MSPS.

- Each I-Q sample was converted to a 32-bit sample word, consisting of a 16-bit I word plus a 16-bit Q word.

- Emulating the digital receivers associated with sensors 0 and 1, DSP-A sent output across the two SLINKs to DSP-B. The SLINKs were implemented using two ribbon cables connecting the appropriate connectors on the board.

- Emulating DSP-1 (responsible for sensors 2 and 3), DSP-A positioned the data that would normally be received by this DSP in its internal memory, so DSP-0 could access it via the multiprocessor memory space; i.e., via the SHARC bus.

- After execution was complete, DSP-A was used to recover the processed data from external SRAM, where it is deposited by DSP-B. The data is searched to locate the space-frequency bin containing the greatest power. This result is compared to the correct result to verify proper operation: that is, the signal should appear in the correct beam and frequency bin.

The fundamental unit of time for a digital microprocessor is a *clock cycle*. Since the SHARCs in this test run at 40 MHz, one cycle corresponds to 25 ns. The source code for DSP-B included many timers which were used to determine how many cycles were required to perform various tasks. The results are given in Figure 1.

The execution time for a single SHARC to process a single block of $L$ snapshots was about $1.35 \times 10^6$ cycles; i.e., 336.5 ms. At this speed, the SHARC can process about 761 snapshots

| Task ID | Description | Cycles | Effort |
|---|---|---|---|
| 0 | SLINK transfer | 32384 | 0.70% |
| 1 | Collect snapshots from other SHARCs in cluster | 130461 | 2.82% |
| 2 | Conversion to I-Q floating point values | 591360 | 12.79% |
| 3 | Partial calibration (spatial filtering), 256 times | 308736 | 6.68% |
| 4 | 2D (spatial) FFT, $8 \times 8$, 256 times | 2003976 | 43.34% |
| 5 | Transfer intermediate result to external SRAM | 218112 | 4.72% |
| 6 | Temporal window, length 256, 64 times | 131456 | 2.84% |
| 7 | 1D (temporal) FFT, length 256, 64 times | 931072 | 20.14% |
| 8 | Transfer final result to External SRAM | 262400 | 5.67% |
|  | Total for tasks 0 through 8 | 4609957 | 99.70% |
|  | Ultimate total, including test overhead | 4623412 | 100.00% |

Figure 2: Execution time for partial calibration. Actual elapsed time was 115.6 ms.

per second on average, and a $K = 32$ cluster can process about $2.43 \times 10^4$ snapshots per second on average. If all snapshots are to be processed continuously at the 10 MSPS rate, $J = 411$ clusters are required using rotating acquisition.

Another important result from this test is the amount of effort required for full calibration: nearly 68%. This provides strong motivation to consider the reduced-complexity calibration proposed above. Results for the minimum-complexity (diagonal-only) calibration are shown in Figure 2. Note that calibration time improves by a factor of 30, and overall algorithm time improves by a factor of about 3.

Another interesting result from this test is the relatively small fraction of effort required for data transfer (sum of tasks 0, 1, 5 and 8). In the full-calibration scenarios, this is about 5% of the total effort, and rises to only 14% in the partial-calibration scenario. Therefore, this architecture is only lightly burdened by the additional chore of redistributing samples among the various SHARCs on a SHARC bus.

Finally, let us compare these results to theoretical results and to the rated performance of the SHARC. If we restrict our attention to the "core" signal processing tasks (tasks 3, 4, 6, and 7), it is simple to derive the theoretical computing effort required. Let us define a real addition or multiplication as 1 floating-point operation, or FLOP. Complex addition is then 2 FLOPs, and complex multiplication is 6 FLOPs. The nominal FLOP ratings for the various tasks can then be calculated as indicated below:

- *Calibration.* A single partial calibration, being an length-$P$ complex inner product, is $8P - 2$ FLOPs. A single full calibration can be viewed as $P$ complex inner products, and therefore requires $P(8P - 2)$ FLOPs.
- *Windowing.* Application of a real-valued window of length $L$ requires $2L$ FLOPs.
- *1D-FFT.* An FFT of length $L$ requires $L \log_2 L$ multiplies; therefore a complex FFT of length $L$ requires about $6L \log_2 L$ FLOPs.
- *2D-FFT.* A 2D-FFT of size $M \times N$ can be decomposed into $M$ FFTs of length $N$ followed by $N$ FFTs of length $M$. Therefore a complex 2D-FFT of size $M \times N$ requires $6MN \log_2 MN$ FLOPs.

Figures 3 and 4 compare nominal FLOPs to observed computational effort. The analysis seems to have provided only a crude estimate of the actual distribution of computational effort among tasks. If we compare the nominal FLOP requirements to the observed execution times for various tasks, we find that computational throughput of the SHARC is about 32 MFLOP/s and 18 MFLOP/s for full and partial calibration respectively. Compared to the rated throughput of 80 MFLOP/s, it appears that the coding is not optimal. Furthermore, the

| Task ID | Description | FLOPs | Effort | Observed Effort |
|---|---|---|---|---|
| 3 | Full calibration (spatial filtering), 256 times | 8355840 | 85.57% | 74.88% |
| 4 | 2D (spatial) FFT, $8 \times 8$, 256 times | 589824 | 6.04% | 16.41% |
| 6 | Temporal window, length 256, 64 times | 32768 | 0.33% | 1.08% |
| 7 | 1D (temporal) FFT, length 256, 64 times | 786432 | 8.05% | 7.62% |
|  | Total for tasks 3, 4, 6, and 7 | 9764864 | 100.00% | 100.00% |

Figure 3: Comparison of nominal FLOPs and observed computational effort – Full calibration case.

| Task ID | Description | FLOPs | Effort | Observed Effort |
|---|---|---|---|---|
| 3 | Partial calibration (spatial filtering), 256 times | 130560 | 8.48% | 9.15% |
| 4 | 2D (spatial) FFT, $8 \times 8$, 256 times | 589824 | 38.31% | 59.37% |
| 6 | Temporal window, length 256, 64 times | 32768 | 2.13% | 3.89% |
| 7 | 1D (temporal) FFT, length 256, 64 times | 786432 | 51.08% | 27.59% |
|  | Total for tasks 3, 4, 6, and 7 | 1539584 | 100.00% | 100.00% |

Figure 4: Comparison of nominal FLOPs and observed computational effort – Partial calibration case.

above data points to Task 4 (the 2D-FFT) specifically as being a problem. Further investigation revealed that this was attributable to inefficient source coding. Other inefficiencies are attributable to a combination of poor coding and shortcomings in the SHARC C compiler and runtime libraries. Custom-coding certain operations in native assembly code, though tedious, is often required to approach the rated performance for any DSP.

## 4 Managing Bandwidth and Duty Cycle Using Decimation

Based on the study so far, it is clear that continuous operation at 10 MSPS will result in a very large and expensive system. However, there may be cases where continuous operation is more important than bandwidth. For example, once an interesting narrowband (single bin) signal is identified, it may be acceptable to reduce bandwidth and then reinvest the additional computational throughput to sustain continuous observation within that bandwidth; i.e., at a reduced sample rate. This can be managed using *decimation*, in which one or more digital filters selectively weight or discard samples to achieve an output signal with the desired (reduced) sample rate and appropriate bandwidth limiting. Large reductions in bandwidth require very narrow filters, which become computationally intensive. Fortunately, there are a number of techniques (outside the scope of this paper) which make it possible to achieve a sample rate reduction $R$ on the order of $10^5$ or higher on a single integrated circuit. A candidate part for this application is the HSP43220, a product of Harris Semiconductor. It accepts a 16-bit digital input at up to 33 MSPS, and can apply decimation factors $R \leq 16384$. The decimation factor (in fact, the filter coefficients themselves) are programmable. Therefore, two of these parts (one for "I" and one for "Q") applied to the output of each of the $P$ digital receivers, could be used to implement a dynamically-variable decimator. For even greater flexibility, note that $2P$ decimation filters could be used for each of the $J$ clusters (i.e., total $2JP$ filters) as opposed to a single set of $2P$ decimation filters at the digital receiver output. This would be more expensive, but would allow the decimation factor to be varied on a cluster-by-cluster basis. For example, it would be possible to have $J/2$ clusters operating at 10 MSPS and the remaining clusters dedicated to narrowband surveillance around specified frequencies.

## 5 Concluding Remarks

To provide software-defined signal processing capability for a 64-element array, a $K = 32$ cluster design is proposed. It is hosted on a PCI backplane with at least 8 slots. Each slot holds a commercially-available DSP board, each with 4 SHARCs. Sensor data redistribution is via the PCI bus as proposed above. This is viable since the redistribution time of 25 ms is small compared to the signal processing time of 336.5 ms or 115 ms for full and partial calibration respectively. Using decimation, this design allows dynamic trade-off between duty cycle and bandwidth, ranging from 10 MHz bandwidth at 0.24% duty cycle to about 20 kHz for continuous processing. The cost of this system is directly proportional to the product of the bandwidth and the duty cycle, since these two things determine the number of clusters required for the system. In terms of performance, this architecture allows a flexible tradeoff between these parameters, within the constraint that

$$F_S \cdot (\text{duty cycle}) \leq (2.34 \times 10^4 \ \ \text{samples/s}) \cdot J. \tag{1}$$

$J$ can be increased simply (although not necessarily cheaply) by adding more clusters. In this sense, the system can be scaled up without limit.

## Acknowledgements

This work was supported by a grant from the SETI Institute, Mountain View, CA USA.