

# FPGA acceleration of an Adaptive Beamformer within GEDAE

Richard Walke

QinetiQ Ltd  
Malvern Technology Centre  
Malvern, WR14 3PS, UK  
+44 (0)1684 895912

walke@signal.QinetiQ.com

Jasmine Lam

QinetiQ Ltd  
Malvern Technology Centre  
Malvern, WR14 3PS, UK  
+44 (0)1684 895912

j.lam@signal.QinetiQ.com

John McAllister

Queen's University Belfast,  
Belfast,  
BT9 5AH, UK  
+44 (0)28 274275

jp.mcallister@ee.qub.ac.uk

## ABSTRACT

In this paper the implementation of an adaptive beamformer through Gedae, in which some of the computation is performed using a Field Programmable Gate Array (FPGA), is presented. A description of the application, its description as a Gedae graph and its optimisation for FPGA are considered. In particular, the optimisation of the memory used is examined, and a number of transformations to the graph are presented to show how it may be reduced manually. Extensions to the Gedae design flow to support FPGA are proposed, and the board support infrastructure to enable this in a fashion consistent with programmable processors is discussed. Finally, an overview of a demonstrator that was built to explore the integration of FPGA within the Gedae design flow is given.

## 1. INTRODUCTION

Digital signal processing (DSP) is the enabling technology for many commercial and military systems. The design tools and technology necessary to achieve in a rapid manner a high-performance implementation with low power, weight and volume are of general interest. A challenging DSP application of interest to both the military and commercial worlds that could benefit from improvements in the tools and implementation technology is adaptive beamforming.

### 1.1 Adaptive beamforming

Adaptive beamforming plays an important role in sensor array systems in countering interference outside the direction of interest. The output of a number of sensors is time-delayed in order to provide high gain in the direction of interest and low gain in the directions of interference. This can be performed digitally, but high bandwidths and large numbers of sensor requires large numbers of programmable processors. A non-programmable alternative is to use Application Specific Integrated Circuits (ASICs). However, these are expensive to fabricate, inflexible, and present legacy issues. Field programmable gate arrays (FPGAs) offer a programmable alternative to ASICs.

### 1.2 Field Programmable Gate Arrays

FPGAs consist of a large array of simple programmable logic and interconnect resources. Current devices provide the equivalent of 6 million programmable gates[1], and include dedicated fixed-point multipliers and memories to support signal processing. Performance in the range of 50 GOPS for fixed-point filtering and beamforming, as required by radar and communications systems, is now possible. Furthermore, devices are emerging that contain

embedded processors and high-speed serial interfaces that simplify the integration of FPGAs with conventional processors[2]. This enables heterogeneous systems to be built that exploit the different capabilities of both FPGA and conventional programmable processors.

However, there is currently a lack of tool support for designing and implementing multi-processor heterogeneous systems. Specifically, the issues are in the creation of the FPGA components from a specification using an imperative language, such as C, and the integration of these components with an array of conventional processors.

### 1.3 FPGA design

The effort to design FPGA implementations is generally an order of magnitude greater than programmable processors. Highly parallel, optimised, architectures need to be created, and efficient implementation relies on the simplification of the arithmetic type and minimisation of its wordlength. In essence, more design work must be done, and with current tools, most of this must be done manually.

A further issue with FPGA is that hardware description languages, such as VHDL or Verilog, have been necessary to describe the design to the required level of detail. Also, designs must be taken through to chip programs via vendor specific implementation flows, which are fragile and require considerable specialist knowledge.

### 1.4 Library-based design approach

Tools are emerging that allow abstract, functional, descriptions in C or Matlab to be taken through to hardware. These are being investigated and one avenue is the subject of a complementary

paper[10]. However, the tools are relatively immature, and have only manual or limited automatic capability to create parallel circuits. There is no doubt they will improve with time, but in the meantime a more immediate method is required. One approach is to design circuits from a library of parameterised cores.

There is already a wide range of targeted cores available, such as digital receivers, Viterbi decoders and FFT processors. This range can be extended to incorporate primitive components, from which custom graphs can be created to describe the necessary functionality albeit with a high-level of detail and in an application specific way. This can be complemented by a pre-defined, more generic library of vector, matrix and signal processing operations. Such a library can be implemented using efficient, parallel architectures that are parameterised for wordlength, function and level of parallelism. Control over the level of parallelism enables the throughput of cores to be balanced, ensuring efficient utilisation of FPGA resources. Furthermore, a library of cores common to both FPGA and conventional processor would enable designs to be migrated between technologies.

The integration of conventional and FPGA technologies, therefore, requires a system design tool to manage the design, its partitioning and mapping to technologies, and the communications between the parts.

## 1.5 Heterogeneous system design

Gedae already provides a graphical tool for designing, partitioning, mapping and implementing multi-processor systems. It is based upon a dataflow model of computation that is well suited to systems processing streams of data. Such applications are commonplace in the military world, and include sonar, radar and information processing systems. Other applications have grown in importance in the commercial world with the advent of mobile computing and communications, where compression and wireless networking are key requirements.

In striving to create a heterogeneous design flow several approaches are possible that range from enhancing either software or hardware tools, to starting from scratch. As most systems are currently software based it makes sense to start with a tool that already addresses software design, and add a hardware capability. Gedae represents a particularly appropriate starting point, as its graphical design entry method and model of computation are highly compatible with hardware design.

## 1.6 Overview

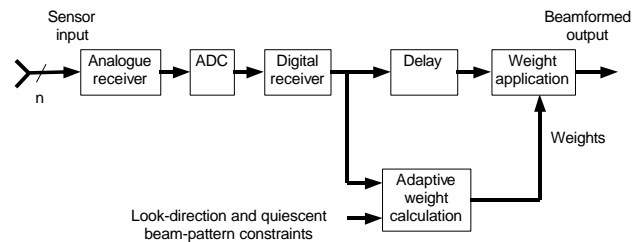
In this paper, the implementation of an adaptive beamformer is presented, in which some of the processing is performed on FPGA. This work was undertaken as a means of exploring the issues of enhancing Gedae to perform heterogeneous system

design. In section 2, an introduction to the application is provided along with its implementation as a Gedae graph. Whilst simplification of arithmetic type and wordlength are obvious requirements for efficient FPGA implementation, the optimisation of the memory requirements is also critical due to the limited resources on FPGA. So, in section 3, a number of simple graph optimisations are presented to show how memory can be minimised at the graph level. The board support infrastructure must also be implemented for FPGA and in section 4 an overview of how this may be done in a manner consistent with current methods is given. A hardware demonstration has been produced to explore some of the issues, and this is described in section 5. Conclusions are presented in section 6.

## 2. ADAPTIVE BEAMFORMING

### 2.1 System overview

Figure 1 shows a block diagram of a generic beamforming system that combines the outputs of an array of sensors in a way that places nulls in the direction of interference, whilst maintaining a high gain in the direction of interest.



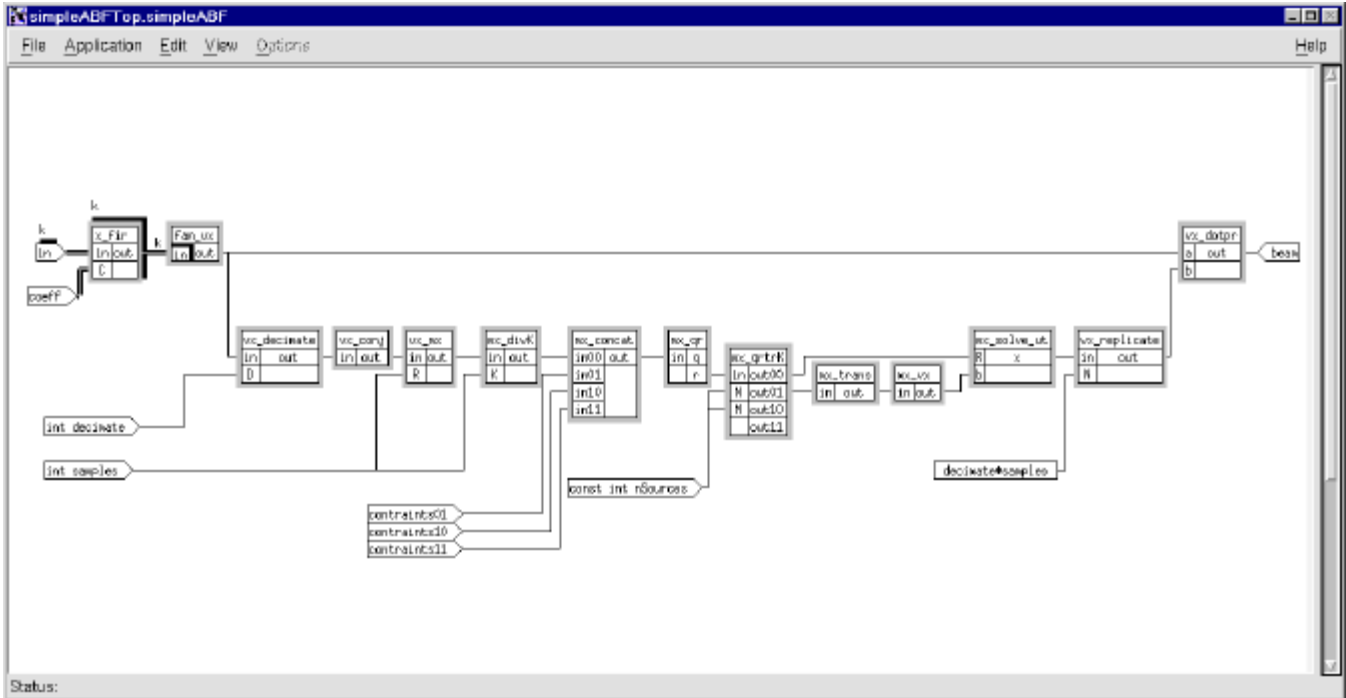
**Figure 1. Block diagram of generic adaptive beamformer**

The frequency, at the sensor inputs shown in Figure 1, is generally higher than its bandwidth, and therefore a combination of analogue and digital receivers is used to shift the input frequency down to base-band. A filter, which forms the bulk of the digital receiver, removes image frequencies introduced by the frequency shift and sets the signal bandwidth.

The beamformer weights,  $w$ , are calculated over a block of input data and applied by the weight application unit to give the beamformed output. At the heart of the weight calculation process is the solution of a least-squares problem, which aims to minimise the power out of the beamformer, subject to the constraints of a fixed gain in a given look direction and a pre-defined quiescent beam pattern. These constraints are shown as inputs to the adaptive weight calculation unit in Figure 1.

### 2.2 Gedae graph

Figure 2 shows a Gedae graph for a narrowband adaptive beamformer constructed from predefined Gedae primitives.



**Figure 2. Geda graph of adaptive beamformer**

The filter in the digital receiver is implemented using a Finite Impulse Response (FIR) filter. Weight calculation is performed using QR decomposition, as this requires low wordlength arithmetic, and can be efficiently implemented in hardware using a scalable parallel processor[3][4].

Only some of the input data is required to calculate the weights (between  $2n$  and  $3n$  samples is generally sufficient, where  $n$  is the number of antennas). Decimation has been used here to select data, although in practice a more intelligent process might be used to avoid samples containing the signal of interest. The complex conjugate of the selected data is scaled and then appended to the constraints to provide an input to the QR decomposition primitive. The output of the QR decomposition is two matrices R and Q (where Q is the unitary rotation matrix, which when applied to the input matrix X, generates an upper triangular matrix R). R is retained and then split into two parts, which are back-substituted (using *mx\_solve\_ut* in Figure 2) to obtain the weights. The weights are replicated, to obtain the right number of values for application to the input data via a dot-product operation (*vx\_dotpr* in Figure 2).

### 3. OPTIMISATION

#### 3.1 Introduction

Efficient FPGA implementations are obtained by reducing wordlength, simplifying the type of arithmetic and minimising memory.

#### 3.2 Wordlength and arithmetic type

The arithmetic circuits to perform DSP on FPGAs must be constructed from a combination of programmable and dedicated resources on the FPGA. The amount of resources consumed by a function depends upon the wordlength and type of arithmetic. Fixed-point arithmetic uses fewest resources, whilst floating-point requires extra logic, mainly for the adder. In both cases, the

smaller the wordlength the greater the computation capability of an FPGA, and the lower its power consumption for a given computation requirement. Currently, Geda only supports 32-bit signed integer and single precision floating-point. A means to specify and simulate reduced precision arithmetic is required by Geda if effective use of the FPGA resource is to be made.

The wordlength of fixed-point arithmetic can be defined more generally using two parameters: wordlength of the integer part and total wordlength (the latter being the sum of the wordlengths of the integer and fractional parts). This representation has been standardised by the SystemC community, and a simulation library is freely available[5].

Floating-point wordlength can be generalised using two parameters to allow both exponent and mantissa wordlength to be specified. The IEEE standard can be adopted for both the bit-level representation and behaviour to exceptions. When exponent and mantissa wordlengths of 8 and 24, and 12 and 52 are selected then single and double precisions respectively are obtained. This is the behaviour of the Quixilica® floating-point cores[6].

#### 3.3 Memory requirements

Implementation of the graph, shown in Figure 2, on FPGA requires components for each of the primitives, a network to interconnect them and, in the case of dataflow, buffering. The implementation of the network and the primitives is specific to FPGA, and is considered in section 4. The memory requirement, which can greatly affect the efficiency of FPGA implementation, is a function of the graph and can be optimised at that level. Such optimisations are presented here.

FPGAs have large numbers of on-chip memories, but they are small in size. If the total on-chip memory is exceeded, then it is necessary to use off-chip memory. This off-chip memory has a bandwidth that is several orders of magnitude less than on-chip memory due to the limited number and speed of pins between FPGA and external memory. As a result, the computation



and  $vx\_divK$  operations can be replaced by scalar  $x\_conj$  and  $x\_divK$  operations by first converting the vector into a scalar. This reduces the memory on the two components from 16 words to only 2 words.

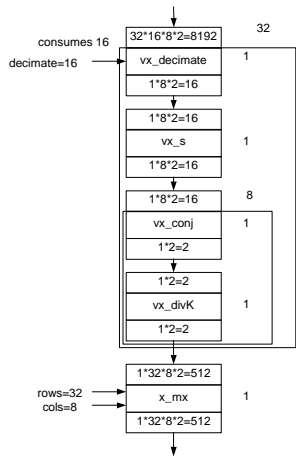


Figure 6. Replacing vectors with scalars

In general, memory can be reduced in a primitive by storing only enough input data to calculate the output. For example, an element-by-element matrix multiply can be implemented using a scalar multiply on a stream of input data in which only one element of the input is applied at a time.

This *streaming* optimisation has been exploited in *hardware* design for many years, and is opposite to that applied in programmable processor implementation, where tokens are processed in larger blocks to minimise the overhead of switching between functions. In Gedaie increasing the granularity does this, whereas for FPGA, the objective is to reduce the granularity. Indeed, the process of implementing a matrix operation in a streaming fashion, using a scalar operation, can be viewed as employing a granularity less than 1.

The processor design community is exploiting this optimisation in an emerging and revolutionary class of processor design known as streaming processors (e.g. the Imagine[8] and Raw processors[9]).

## 4. IMPLEMENTATION FLOW

### 4.1 Extending flow to include FPGA

The current Gedaie design flow already provides support for different types of embedded processor by allowing the C compiler, library functions and board support infrastructure to be specified by the vendor for each type of embedded processor.

FPGA could be targeted using the current flow by implementing an array of C-programmable processors connected by a runtime programmable communications network. Whilst this may be appropriate in some system configurations, it is less efficient in FPGA resources than employing an optimised network of application specific primitives, i.e. an application specific circuit.

In this section, extensions to the current design flow are proposed to enable graphs to be optimised for FPGA. Figure 7 presents the extended design flow for FPGA. The main addition is that Gedaie provides, at compile time, a network of the primitive interconnectivity for the part of the graph mapped to the FPGA. This also allows Gedaie to target a new class of multi-processor

devices where the interconnection must be specified at compile time[7].

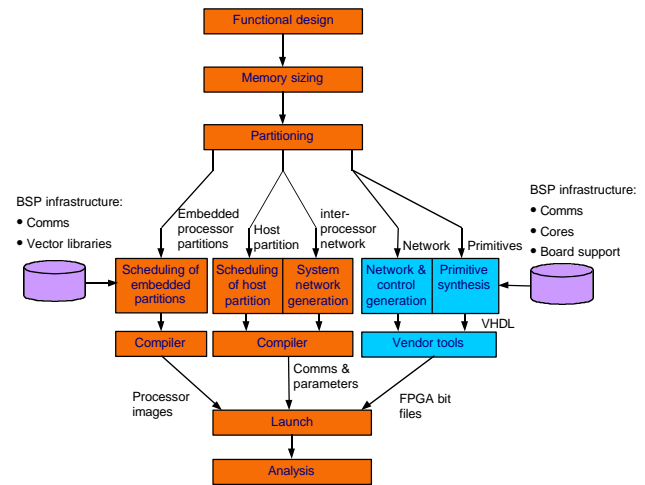


Figure 7. Extended Gedaie implementation flow

### 4.2 Partitioning

The Gedaie graph is partitioned into host, embedded and FPGA parts. Gedaie creates programs for the embedded and host partitions using a C compiler and the current flow. The embedded partitions communicate through *send* and *recv* primitives, which are automatically inserted on partition boundaries by Gedaie. The information to allow a *send* primitive to transmit to a particular *recv* primitive is supplied as a parameter by the host when the programs are launched. In this way, inter-processor communication can be reconfigured at runtime.

### 4.3 FPGA partition

The FPGA part of the graph needs to be handled differently if optimised implementations are to be achieved. Specifically, the FPGA part of the graph must be optimised, scheduled and mapped. Some of the optimisations have been considered previously. Here the scheduling and mapping is considered further.

A graph may be scheduled and mapped onto the FPGA in many ways. This represents a second level to the partitioning of the graph. The implementation may be fully parallel, fully sequential or a near infinite number of combinations in between the two. The search for an optimum solution is part of the *design space exploration*, and is similar to that which must be performed for an implementation on an array of programmable processors. The problem is compounded on FPGA due to the high levels of parallelism that may be employed and the need to make decisions on partitioning between FPGA and conventional processor. Automation of the scheduling and mapping processes makes design space exploration possible, and we consider this in [10].

Initially, it will be assumed that the primitives in the graph are mapped to components on a one-to-one basis, to provide a fully parallel implementation on the FPGA. The more general case requires Gedaie to provide a mechanism to specify sequential and parallel mappings, and the vendor to provide a means of scheduling more than one primitive on a component (using hardware sharing if the primitive is the same or programmable processor if it is different).

### 4.3.1 Primitive synthesis/compilation

The FPGA part consists of a network of primitives. The primitives are either synthesised from source code (in C, or an alternate representation such as RTL) or replaced with a pre-defined core from a library. The latter is similar to how the E\_library functions are employed on conventional processors.

### 4.3.2 Network implementation

The network specifies the interconnection of primitives at compile time. This knowledge allows dedicated connections to be made between primitives on the FPGA. This avoids the logic overhead of implementing an on-chip general communications infrastructure configured at launch-time by the host.

For off-chip connections, however, it is proposed that the current, more general, network infrastructure be retained, as this provides consistency with the rest of the system, and allows failures in the less reliable off-chip communications to be circumvented by reconfiguration by the host.

### 4.3.3 Control and parameter interface

An embedded processor on the FPGA, or on a processor on the card hosting the FPGA, can run a command program to enable control of the FPGA part of the graph. Reset, trace and parameter update capabilities can be implemented by the embedded processor, in a way that makes the FPGA look like a standard embedded processor. Parameters can be written from the host processor to the embedded control processor and then to the primitive via an on-chip parameter bus. Transactions with each primitive will be enabled according to an address map built into the FPGA version of the graph and compiled into the command program during the build process.

## 4.4 Board support infrastructure

A board support infrastructure is required for the FPGA part to implement the functionality described in previous section, and this is very similar to that of a conventional processor. A board-support layer is important as it provides an abstraction from the board hardware and allows migration of Gedae graphs between different boards.

As stated, it is proposed that off-chip communications use current mechanisms i.e. *send* and *recv* primitives that interface with a board-specific communications method. In the FPGA context, where implementations are likely to be parallel, we need a means of connecting a single physical communication link to one or more inputs or outputs of the graph. In Figure 8 a bus between the communications interface and the primitives has been implemented to allow this in a scalable manner.

Each separate physical interface should be identified within the configuration file as a particular transfer method. When selected by the user for a particular communication path, the FPGA compilation process should implement the necessary connections to the requested interface.

It is suggested that the BSP defined communication paths also be used when partitioning RTL graphs across multiple FPGAs, as this will allow pin allocation and inter-FPGA communication paths to be managed. To allow an RTL partition to be connected to an arbitrary communications infrastructure (e.g. Ethernet, wire and custom), it is necessary to implement an enable signal on the RTL partition to control its execution and associated data input and output. This is consistent with the use of a top-level dynamic schedule on embedded processors, and avoids the need to lock clocks on separate FPGAs.

## 4.5 Vendor support

In summary, to support an FPGA board the vendor must provide:

- An FPGA compilation process (to support network synthesis & address allocation);
- A primitive library, or primitive synthesis method;
- A command program (start, stop, reset, trace and parameter update functions);
- A communications infrastructure (communications interface and *send* and *recv* primitives);
- A launch mechanism (downloading of bit-file).

## 5. Adaptive implementation

### 5.1 Overview

In the previous section, we have proposed a very general method for including FPGA as a target device in Gedae in a way that is consistent with the current implementation flow. In this section, a demonstrator is described that was built to explore some of the issues.

### 5.2 Demonstrator

**Error! Reference source not found.** shows a block diagram of the main components of the demonstrator. The demonstrator was designed to implement both the FIR and QR components on FPGA (using the Quixilica® software configurable digital receiver (DRx) and QR processor). Communication between host and FPGA has been implemented using *send* and *recv* components whose channel number has been set manually at compile time. These components interface with an embedded communications interface that provides a connection to the physical medium. A range of physical media and interfaces are becoming available. High-speed serial interfaces on the latest generation of FPGA supports the physical interface of a wide range of emerging standards e.g. Infiniband, 1Gbit and 10Gbit Ethernet and serial RapidIO. We have implemented our embedded

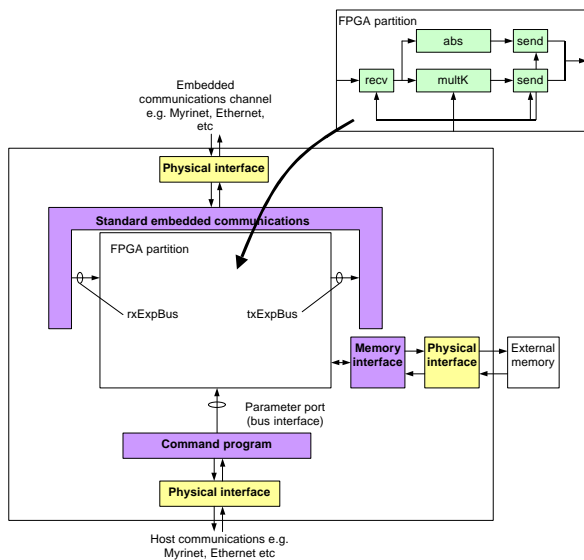


Figure 8. Board support infrastructure

communications interface over Myrinet and PCI physical interfaces.

The parameter interface has been implemented using a call to a memory interface component. This primitive takes in a set of coefficients as Gedae parameters and writes them into the digital receiver core (DRx) through the *mem write* function call shown in **Error! Reference source not found.**

## 6. Conclusions

Gedae is highly suited to the rapid design of heterogeneous systems. Its graphical representation is helpful in identifying parallelism, and for managing partitioning and mapping of the graph. The dataflow model of computation is highly compatible with the nature of systems that map well onto FPGA.

The enhancements to Gedae are primarily required to enable efficient implementations on FPGA. Minimisation of wordlength, simplification of arithmetic to fixed-point, and dedicated inter-processor communications enable FPGA to provide computation increases and power consumption reduction over conventional techniques, such as programmable digital signal processors.

There are many ways of generating an FPGA implementation from a Gedae graph. What is required is a range of methods. Firstly, RTL offers the ability to design efficient implementations on FPGA by restricting the behaviour of primitives to single cycle, single token. The approach addresses a wide range of

requirements. Secondly, the ability to target cores will allow Gedae to exploit a wide range of existing intellectual property, and a general library of vector, matrix and more general signal processing operations will complement these. Thirdly, Gedae can exploit the emerging architectural synthesis tools that offer circuit synthesis from algorithm level descriptions in imperative languages, such as C. Gedae provides a system-level tool for modelling and for partitioning the system into parts of low enough complexity to be managed by these tools. The use of SystemC and CoCentric Studio is discussed in a complementary paper[10]. This approach enables architectural transformations to be applied automatically to obtain different trade-offs between area and speed, enabling the rapid exploration of the design space and greater opportunity to obtain more optimum solutions.

## 7. Acknowledgements

GEDAE is a trademark of Blue Horizon Development Software.

This work has been sponsored by the UK Ministry of Defence Corporate Research Programme and has been undertaken in collaboration with BAE SYSTEMS ATC, Gt. Baddow, UK.

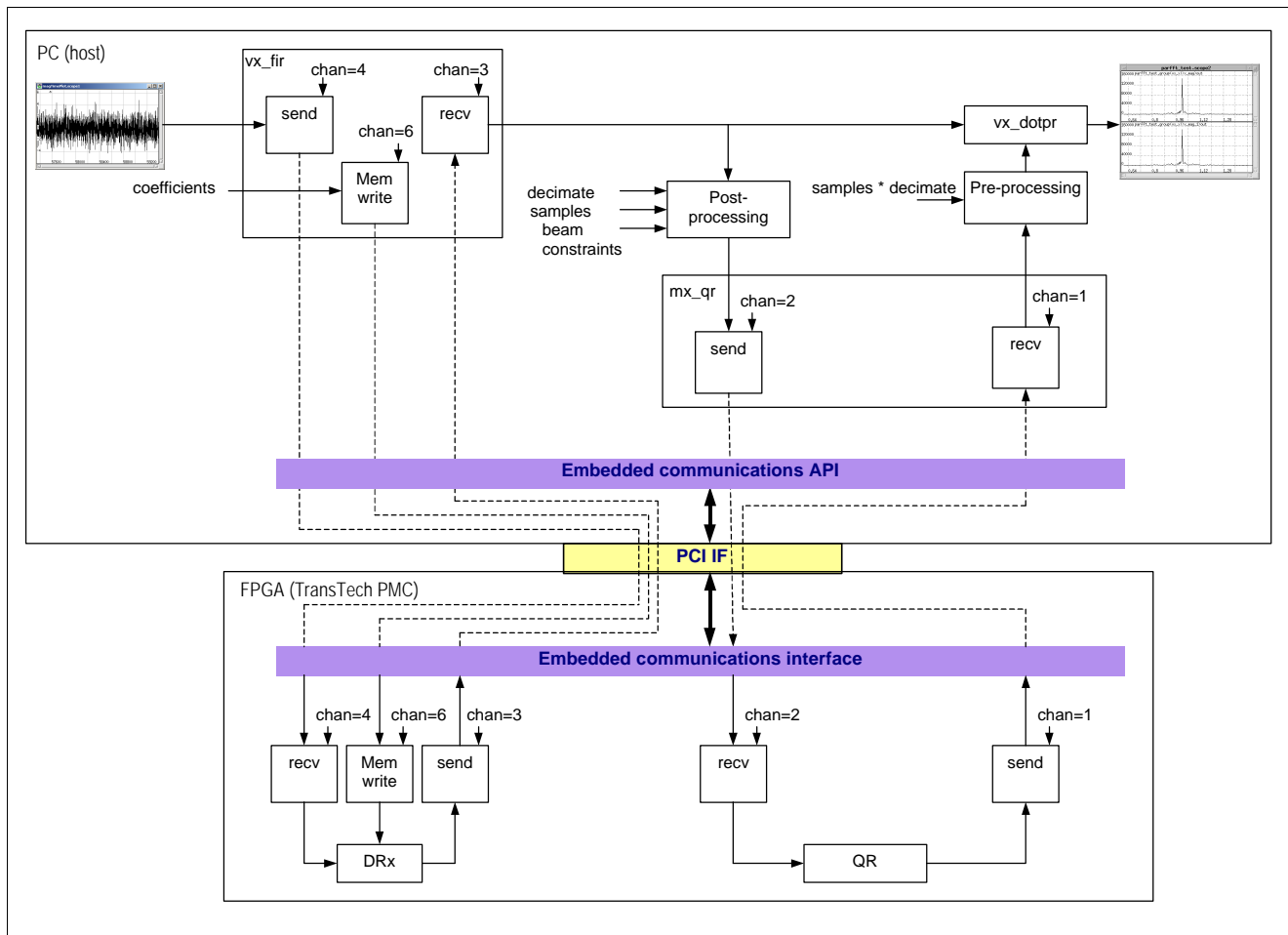


Figure 9. Adaptive beamformer based on FPGA and host PC

## 8. REFERENCES

- [1] Xilinx, Inc, "Virtex-II FPGAs Datasheet", DS031, <http://www.xilinx.com>, 2003.
- [2] Xilinx, Inc, "Virtex-II Pro Platform FPGAs Datasheet", DS083, <http://www.xilinx.com>, 2003.
- [3] Walke R. L., Smith R. W. M. Lightbody G., "20GFLOPS QR processor on a Xilinx Virtex-E FPGA", Proc. SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations X, pp. 300-310, 2000.
- [4] Walke R., Smith R.W. M., and Lightbody G, "Architectures for Adaptive Weight Calculation on ASIC and FPGA", 33<sup>rd</sup> Asilomar Conference on Signals, Systems and Computers, 1999.
- [5] Open SystemC Initiative, <http://www.SystemC.org/>, 2002.
- [6] "Quixilica Floating-Point FPGA Cores", Datasheet, <http://www.quixilica.com/>, Sept. 2001.
- [7] PicoChip, <http://www.picochip.com>, 2003.
- [8] Kapasi, U. J., Dally W. J., Rixner S., Owens J. D., and Khailany B., Proceedings of the IEEE International Conference on Computer Design, pp. 282-288, 2002.
- [9] Taylor M. B., Kim J., Miller J., Ghodrat F., Greenwald B., Johnson P., Lee W., Ma A., Shnidman N., Strumpen V., Wentzlaff D., Frank M., Amarasinghe S., and Agarwal A. "The Raw Processor - A Scalable 32-bit Fabric for Embedded and General Purpose Computing", Hotchips XIII, Palo Alto, California, 2001.
- [10] McAllister, J. Colgan, K., Woods, R and Walke R. L., "Architectural Exploration Using GEDAE for Heterogeneous Architectures", Gedae Users Conference, Philadelphia, 2003.

© Copyright QinetiQ Ltd 2003.