

A SCALABLE ARCHITECTURE FOR GENERAL REAL-TIME ARRAY-BASED DSP ON FPGAS WITH APPLICATION TO THE WAVE EQUATION

Ross P. Kirk,

Audio Lab, Department of Electronics,
University of York
York, UK

Jeremy J. Wells,

Music Research Centre, Department of Music,
University of York
York, UK
jez.wells@york.ac.uk

ABSTRACT

This paper describes a scheme for parallel execution on FPGAs of DSP tasks which rely heavily on MAC operations. Multiple operations are assigned to a single ‘processing node’ such that each node can operate just in real-time. Where the number of MACs required exceeds the capability of a single processing node additional nodes are added until the capacity of the FPGA is exhausted. Additional requirements beyond the capability of a single FPGA are accommodated by extension across multiple devices, offering significant scalability. Resource usage, performance results for an example acoustic modelling application on a modest single FPGA and development system are presented.

1. INTRODUCTION

DSP algorithms often require numerically intensive, hard deadline, real-time processing. A typical algorithm, as seen in FFT, convolution and digital filter systems, is based on the classic ‘sum of products’ calculation:

$$y_i = \sum_{n=0}^i a_{n-k} x_{n-l}, \quad k, l \in \mathbb{Z} \quad (1)$$

This is supported by multiply-accumulate (MAC) architectures in many DSP devices.

As is well known, the conventional Von Neumann processor structure, consisting of a single arithmetic processing unit and a single memory unit interconnected over a system bus, is not well suited to this task. Limitations in performance arise through the need to share the single processing unit over many algorithmic nodes in the calculation (such as the individual product terms in (1) above) and through the finite bandwidth of the system bus carrying data transfers between the two units.

Many architectural innovations have passed into common usage over the years in an attempt to mitigate these problems. For example, the ‘Harvard’ architecture approaches the bus bandwidth problem by separating instruction and data flows over dedicated buses. DSP devices now commonly have pipelined multiple arithmetic and register units to provide increased processor performance and on-board cache memory is used in an attempt to limit processor-memory traffic. However, given that there may be a very large number of product terms in (1) such innovations will ultimately be compromised in a similar way to the basic Von Neumann structure, for the simple reason that they are variants of that structure.

Modern gate array devices provide opportunities to address the processing of algorithmic nodes in a radically different way, independent of the Von Neumann architecture. They have enough hardware resource to provide many multiply-accumulate (MAC) structures and many independent memory units which may be associated directly with the algorithmic nodes. A dense interconnect medium is provided within the device whose point-to-point connectivity means that the bandwidth bottleneck of a ‘system bus’ can be avoided.

The origin of the work described here lies in an attempt to quantify any architectural advantage which Field Programmable Gate Arrays (FPGAs) can provide in adopting this approach. Of course, graphical processing units (GPUs) possess an architecture which is well-suited to DSP problems such as MAC since they consist of multiple processors enabling algorithmic nodes to be distributed, at least partly, in parallel. There has been considerable interest in GPU implementations of such tasks, for example in room acoustic modelling. However there is still a limitation since the parallel processors use monolithic rather than distributed memory [1]. Massively parallel MAC operations have been considered before (e.g. [8]) but the architecture presented here offers a spatial paradigm for parallel processing, with address generation enabling adjacent processors to remain in time and location synchronisation with each other. Distributed memory offers genuinely localised processing without the bandwidth required to transfer data from central memory to local processing nodes. This paradigm is particularly appropriate for solving the 2D and 3D wave equations.

2. SCALABLE ARCHITECTURE FOR MAC-BASED PROCESSING

This paper describes the application of an FPGA-based parallel architecture for solving the 2D wave equation in real-time. There is existing work describing the use of FPGAs specifically for this problem [2]. However, the use of the MAC as the fundamental processing unit in the study described here means that the resulting architecture is quite generic, so that the approach can be applied to a wide variety of signal processing tasks, including banks of digital filters, correlators, FFT/convolution engines as well as finite element models, cellular automata and similar algorithms. The centralised address generation that is used enables the scaling of networks of MAC processors across a variety of algorithmic architectures and multiple FPGAs.

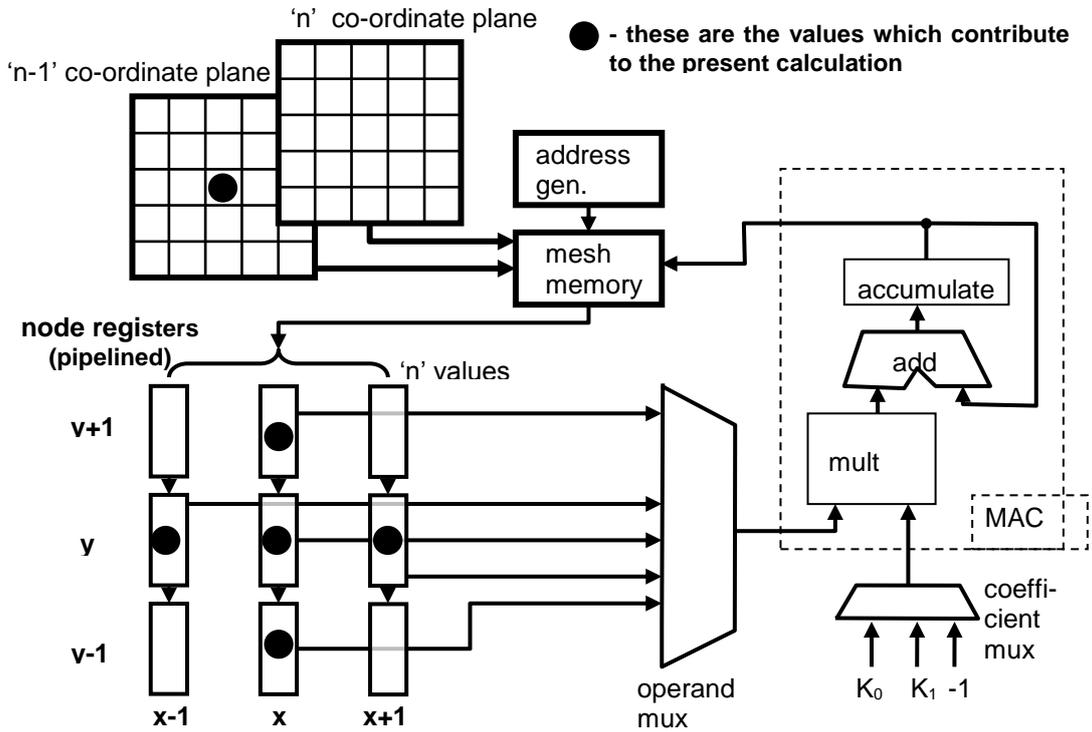


Figure 1: Node processor operation in the proposed scheme

The 2D wave equation enables the modelling of the propagation of travelling waves in a two-dimensional acoustic medium and their reflection from boundaries surrounding and within the space. The advantage of this approach over, for example, ray-tracing methods is that wave behaviour, such as diffraction, is inherent in the model [3]. The free-space propagation of these waves can be described by the equation

$$p_{x,y}^{n+1} = k_0 p_{x-1,y}^n + p_{x+1,y}^n + p_{x,y-1}^n + p_{x,y+1}^n + k_1 p_{x,y}^n - p_{x,y}^{n-1} \quad (2)$$

where $p_{x,y}^{n+1}$ describes the air pressure of a cell at point x,y in a 2D rectangular mesh at time update instant $n+1$, in terms of the air pressure values of the cell at the present and previous update instants, together with the present pressure values of adjacent cells in the mesh. k_0 and k_1 are constants. A simple extension is available which deals with 3D spaces and which is straightforwardly compatible with the architecture described here. There is a specific case of (2) where $k_1 = 0$, which both simplifies the calculation and offers the possibility of 'sub-gridding' which reduces the required memory by 50% as only one mesh is required, rather than two [4].

It would often be profligate to assign one MAC unit to each of the processing nodes in (1), given that it may be possible to use one MAC to process many nodes within a typical sampling interval in a DSP system. In these cases it makes sense to multiplex the processing of a group of nodes through one MAC unit, up to a limit M defined by:

$$M = \left\lfloor \frac{\tau_s}{m \tau_0} \right\rfloor \quad (3)$$

where τ_s is the sampling interval, τ_0 is the cycle time of the MAC and m is the number of MAC cycles required to complete one sample calculation (often determined by the number of operands in the calculation).

For the system described in this paper Fig. 1 shows the architecture used to support such a group of nodes for the acoustic mesh application defined by (2). The mesh memory contains two sample (co-ordinate) planes, one of which accommodates the samples of the present sample interval (n) which contribute to the calculation of the next pressure value ($n+1$) of the node under consideration. The other plane ($n-1$) accommodates the samples from the previous sample interval. As these 'previous' pressure values are consumed by the application of equation (2), they are replaced by the newly calculated $n+1$ values so that the $n-1$ plane gradually becomes the $n+1$ plane. In the next sample interval the two planes are swapped so that n becomes $n-1$ and $n+1$ becomes n and so on. The address generator preloads the n values into pipeline registers for the operand multiplexer.

For this audio application, a sample rate of 44.1 kHz is chosen. (However it should be noted that, due to dispersion error which increases with increasing frequency, a higher sampling rate may be required for best quality results [5].) A MAC cycle time of 10 nS is easily attainable using standard automated optimisation techniques provided by the FPGA synthesis tools used (Xilinx ISE v10.1) and the number of MAC cycles required by the architecture is 7 for each sample (taking into account internal

pipeline latencies in the MAC and the total number of operands). Equation (3) therefore gives the number of nodes supported by the node processor M as 323 in each plane. 32 bit integer arithmetic and saturated overflow are used.

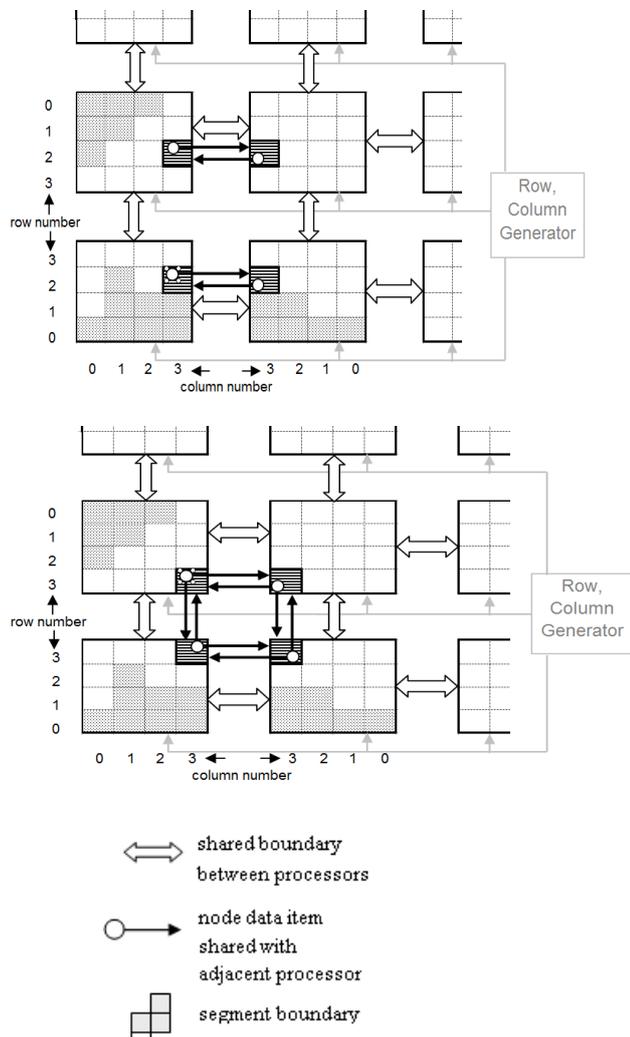


Figure 2: Node processing for two successive node steps at coordinates $\{2,3\}$ (top) and $\{3,3\}$ (bottom)

The architecture is scaled so that multiple node processors can be used within one gate array device, and with suitable inter-device interfaces, across many devices. The acoustic node space is divided into clusters containing M nodes, addressed in a row/column manner by a master row/column generator fed to all node processors. The node processors are all driven in common by this row/column generator and they therefore operate in lock-step synchronisation.

Fig. 2 shows this arrangement for a portion of the processor array for two successive steps in the cluster space in a simplified system with a cluster space $M=16$. (Each processor supports 16 nodes in this example.) Node processing increments by row number, and then by column number. The next node to be processed in each processor following the step shown in the lower figure would therefore be $\{row,col\}=\{0,0\}$.

When the address generator detects that its processor is operating at the boundary of its jurisdiction it exchanges data with the adjacent processor(s) which are also operating at the same point in their boundary because of the lock-step. This exchanged data is sent to the operand multiplexer in the place of local node memory data. A form of superscalar operation is thus attained by having the multiple node processors operate with true concurrency. The lock-step synchronisation mechanism can be preserved across multiple FPGAs using either using high speed serial links or simple point to point between I/O on the devices. Address generation for all devices would remain on one single master device.

The cluster space is segmented so that the address generator can track the boundaries and also so that different segments can operate with different propagational characteristics, for example by feeding different coefficients to the MAC's coefficient multiplexer. This is implemented with a 'segment RAM' associated with each address generator. This is a dual-port device, loadable from a controller processor, thus providing a dynamic configuration capability. The content of the segment RAM may be unique to each node processor, meaning that node processors are not constrained to have the same configuration. The address algorithm run by the address generators mean that precious RAM resource need not be allocated to row-column addresses lying outside the node space.

Parameters exist in the parameter RAM to tell relevant node processors that they are operating at the boundaries of the entire acoustic space. In the simple demonstration implementation here, this causes total in-phase reflection (pressure doubling) at the boundaries, although this can be replaced with a more flexible strategy based on the use of boundary digital filters, such as those described in [6]. These would integrate well with the node processor/MAC structures described in this paper. Further detail and illustrative examples are available from the on line resource which accompanies this paper [7].

Every node has a multiplexer which routes data into the MAC. This may be data from adjacent nodes (including reflected data from boundaries, as determined by boundary reflection coefficients) or from the local node where the data relates to propagation values from earlier update cycles. Another source of data is that which corresponds to direct external acoustic excitation of the node (again see the online resource for more information [7]). This data can be transferred into the node's multiplexer input using a chain of shift registers driven from a control processor embedded in the FPGA. Data input appears externally as a single input port which is acquired via a handshake protocol. Similarly output data from any node is acquired by parallel loading the shift register chain from the node array and clocking it serially out of the chain.

3. EXAMPLE IMPLEMENTATION

The question remains as to how many such node processors can be fitted into one commonly available device. We scaled the design up and targeted the result to a Xilinx *Virtex 5 vsx95t* device. The Xilinx ISE design tool reported that it could fit 90 node processors into the device for a MAC cycle time of 10nS. A Windows XP machine fitted with 3 GB of RAM was used. The node capacity (M) was successfully increased to 680 locations per memory plane, even though the maximum useable was 323 for real-time operation, as explained above.

ISE reported that 92% of the device's BlockRAM, 56% of DSP48Es (essentially MACs) and 75% of primary logic units

(LUTs) were used. Attempts were made to further decrease the MAC cycle time so that the greater node capacity could be used, but these attempts failed on timing constraints. Additional progress might be made, for example by increasing the use of pipelining in the design, but this would require significant redesign of the processor, for possibly marginal results.

An attempt to increase total node space by increasing the number of processors to 100 failed on place and route in ISE synthesis because of memory capacity limitations in the XP Windows machine (i.e. the task became intractable on the design system, rather than un-implementable on the FPGA itself). Larger FPGA devices (e.g. the vsx240t device) could not be used for the same reason. The 90 processor/10nS/680 node design was therefore taken as the largest implementation attainable with this design under Windows XP. Nevertheless the scale of the logic implemented within this limitation remains impressive. It is important to realise that the memory capacity of the development PC is an important constraint with large FPGA designs (as is the time for the synthesis - runs of 36 and 48 hours were not uncommon).

4. CONCLUSION

This paper, and the detailed online resources that accompany it, has described an architecture for real-time implementation of highly parallel, distributed DSP algorithms. The specific case used to illustrate this architecture has been the solution in real-time of the 2D wave equation using a rectilinear finite-difference time-domain (FDTD) mesh. This has demonstrated that modern FPGAs can accommodate useful scaled DSP algorithms at sample rates which would be challenging for conventional processor designs. The design takes advantage of localised memory and the bus bandwidth limitation between node processors has therefore largely been eliminated. Whilst useful DSP capability and performance can be achieved with this design, the memory available in the development PC is a significant constraint. However this constraint will be eased as systems are developed over time, allowing more capacious devices to be used as they are produced by gate array manufacturers. This means that the use of FPGA devices for array-based DSP algorithms is a technology with valuable potential. Of course, the suitability of this approach to a particular problem, is determined by the extent to which it is characterised by MAC operations and amenable to parallelisation. However, simulations of wave propagation (and other phenomena that are comprised of a large number of local processes distributed in space and occurring over time) are particularly well-matched in this regard.

5. REFERENCES

- [1] Savioja, L., "Real-Time 3D Finite-Difference Time-Domain Simulation of Low- and Mid-Frequency Room Acoustics", *Proceedings of the 13th International Conference on Digital Audio Effects(DAFx-10)*, Graz, Austria, September 2010, pp. 43-50.
- [2] Motuk, E., et al, "Design Methodology for Real-Time FPGA-Based Sound Synthesis", *IEEE Transactions on Signal Processing*, vol. 55, no. 12, Dec. 2007, pp. 5833-5845
- [3] Murphy, D. and Beeson, M., "Modelling Spatial Sound Occlusion and Diffraction Effects Using the Digital Waveguide Mesh", *24th International Conference of the Audio Engineering Society*, Banf, Canada, June 2003, pp. 207-216.
- [4] Bilbao, S., *Wave and Scattering Methods for Numerical Simulation*, Wiley-Blackwell, 2004.
- [5] van Walstijn, M. and Kowalczyk, K., "On the numerical solution of the 2D wave equation with compact FDTD schemes", *Proceedings of the 11th International Conference on Digital Audio Effects(DAFx-08)*, Espoo, Finland, September 2008, pp. 205-212.
- [6] Kowalczyk, K. and Walstijn, M., "Modelling Frequency-Dependent Boundaries as Digital Impedance Filters in FDTD and K-DWM Room Acoustics Simulations", *Journal of the Audio Engineering Society*, vol. 56, no. 7/8, July 2008, pp.569-583.
- [7] http://www.jezwells.org/Kirk_Wells_DAFx13_support_files.zip
- [8] Cadambi, S. et al, "A Massively Parallel FPGA-based Co-processor for Support Vector Machines", *17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp.115-122