VALVED DATAFLOW FOR FPGA MEMORY HIERARCHY SYNTHESIS

M. Milford and J. McAllister

eStreams,
The Institute of Electronics, Computer Science and Information Technology (ECIT),
School of Electrical Engineering, Electronics and Computer Science,
Queen's University Belfast
Email: {mmilford01, jp.mcallister}@qub.ac.uk

ABSTRACT

For modern FPGA, implementation of memory intensive processing applications such as high end image and video processing systems necessitates manual design of complex multi-level memory hierarchies incorporating off-chip DDR and on-chip BRAM and LUT RAM. In fact, automated synthesis of multi-level memory hierarchies is an open problem facing high level synthesis technologies for FPGA devices. In this paper we describe the first automated solution to this problem. By exploiting a novel dataflow application modelling dialect, known as Valved Dataflow, we show for the first time how, not only can such architectures be automatically derived, but also that the resulting implementations support real-time processing for current image processing application standards such as H.264. We demonstrate the viability of this approach by reporting the performance and cost of hierarchies automatically generated for Motion Estimation, Matrix Multiplication and Sobel Edge Detection applications on Virtex-5 FPGA.

Index Terms— FPGA, High Level Synthesis, Dataflow, Memory

1. INTRODUCTION

The computational capacity and memory bandwidth resources of modern FPGA are unprecedented, motivating their use in very high performance embedded signal and image processing systems [1]. These characteristics, along with the emergence of viable High Level Synthesis (HLS) design tools (i.e. those which can synthesise efficient FPGA processing architectures from behavioural algorithm specifications [2]) suggest FPGA are a programmable embedded DSP device of unequalled potential.

This potential, however, is tempered by the relative lack of on-chip data storage capacity on modern FPGA [3]. This presents a severe implementation problem for image processing applications in particular, where significant capacity and bandwidth are required for their frame-based operations. To overcome this issue, manual development of multi-layered hierarchies of memory, composed of numerous layers of RAM implemented in the FPGAs programmable logic, or from dedicated on-chip RAMs, and off-chip RAM is required. Whilst methodologies deriving such architectures from behavioural models are evident [2][4] automated solutions do not exist, a situation which has been identified as a key open challenge threatening the viability of HLS tools for FPGA [2].

In this paper we present the first such automated solution. By exploiting a novel dataflow application modelling approach, known as Valved Dataflow (VDF), we show how multi-level memory hierarchies may be automatically derived, and we present the synthesis results of the first automatically generated implementations for Motion Estimation, matrix multiplication and Sobel Edge Detection applications.

The remainder of this paper is as follows. Section 2 describes the memory resources of modern FPGA and surveys current work which addresses synthesis of such structures. Section 3 introduces VDF, and Section 4 the automated mapping approach for translating VDF models to multi-level memory architectures. Section 5 presents the results from the automated synthesis of image processing kernels.

2. BACKGROUND

The relative capacities and bandwidths of the levels of a memory hierarchy resident on a state-of-the-art FPGA, such as the Xilinx Virtex-6 are illustrated in Fig. 1 [3]. On-chip, these devices host Look Up Table (LUT)-based RAM, which offer a low capacity but very high bandwidth data storage facility. These are augmented by configurable, dedicated on-chip Block RAM (BRAM) components which offer a higher capacity storage facility with lower bandwidth. Finally, off-chip DDR3 memory offers low-bandwidth bulk data storage.

Consider these resources in the context of an image processing operation such as Full Search Motion Estimation algorithm (FSME) [5]. This processes 352×288 pixel frames of CIF video data, where each pixel contains three 8 bit values. Given a block-size of 16×16 pixels and a search-window of 32×32 pixels, FSME requires 657 KB of storage capacity accessible at an approximate rate of 14.5 GB/s. Given these lev-

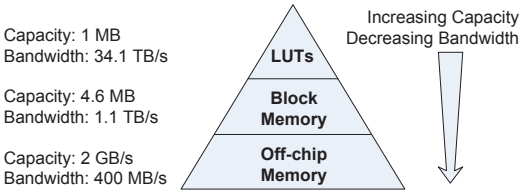


Fig. 1. Virtex-6 FPGA Memory Hierarchy

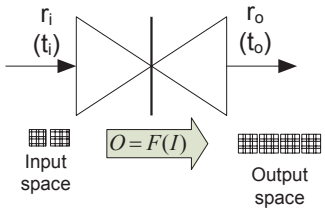


Fig. 2. A Valve

els, any substantial image processing application will quickly saturate the on-chip capacity of FPGA, necessitating off-chip storage, despite the fact that the requisite memory bandwidth is not availalbe off-chip. Typically, the solution to this problem is the development of a multi-level memory architecture spanning all levels of the hierarchy.

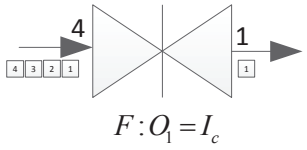
Whilst work has shown how refining the application to exploit all three levels of the hierarchy enables real-time implementation [4], this is typical of current approaches to this problem which offer only manual solutions [6][2], or automatic derivation of two-level hierarchies [7]. To the best of the authors' knowledge, there is as yet no approach which can automatically derive a memory hierarchy of more than two levels.

In this paper we solve this problem, presenting the first such approach. Given the proliferation of dataflow application models in approaches to this problem, either as the entry point for the HLS process [4][8][9], or as an intermediate language used during the compilation process [10], we propose a novel dataflow modelling approach as the entry point for this automated solution in Section 3.

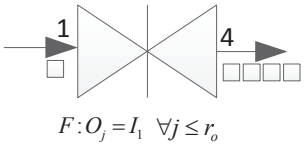
3. VALVED DATAFLOW

We propose the use of a novel dataflow [11] modelling dialect, known as Valved Dataflow (VDF) as the entry point for the memory hierarchy synthesis process. The key novelty in VDF is in the strict separation of the *computational* parts of the algorithm from the parts of the algorithm which change the rate and nature of the data as it moves between computational elements. A VDF graph (VDFG) $G = \{A, E\}$ is composed of a set of actors A and a set of edges E. Each actor $a \in A = \{P\}$ where P is a set of ports. When an actor fires, it consumes tokens through its input ports, transforms them and produces tokens through its output ports. Hence each port $p \in P = \{r, t\}$, where $r \in \mathbb{N}$ defines the rate of each port, i.e. the number of data tokens which pass the port per firing and $t \in \mathbb{N}^m$ describes the size of the *m*-dimensional token the port encounters.

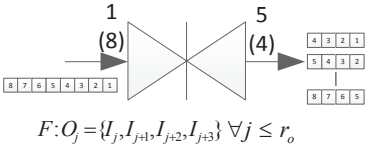
The major differentiating feature of VDF from other dataflow languages is in its definition of an edge $e \in E = \langle prd, cns, V \rangle$ where $prd \in A, cns \in A$ are source and sink vertices respectively and V is a sequence of valves. A valve, shown in Fig. 2 is a single-input, single-output entity. It has no functional capability (i.e. it cannot perform arbitrary mathematical operations) but rather a firing of a valve changes the rate and form of data flowing along the edge, performing for instance interpolations, decimations, token conversions or subtoken selection operations. A valve maps the token impinging on its input port *i* (said to be in its *input space* $I \in t_i^{r_i}$) onto tokens produced into its output space $O \in t_o^{r_o}$ through its output port o. Hence it may be said to implement the function $F : I \to O$. To illustrate the effect of different variations of F on the behaviour of a valve, consider how it may be used to implement decimate, interpolate or sliding window operators as outlined in Fig. 3.



(a) Decimate



(b) Interpolate



(c) Sliding Window

Fig. 3. Valve Configurations

The key motivation for distinguishing valves from actors is to allow valves (i.e the data transportation infrastructure) to be synthesised independently of the mathematical operations applied to the data. This refinement process is described in Section 4.

4. FPGA-BASED IMPLEMENTATION OF VDF MODELS

4.1. Overview

The process for mapping a VDFG to an FPGA architecture is show in Fig.4. It proceeds through two major steps: *Technology* and *Node* Synthesis. During technology synthesis, VDFG valves are mapped to specific layers of the memory hierarchy of the target device and the actors to specific Processing Elements (PEs). This defines the network-level architecture of the implementation, with the node level synthesis following to finalise the final Register Transfer Level (RTL) architecture.

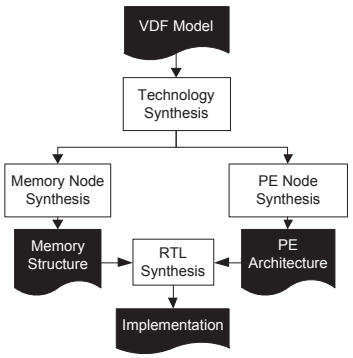


Fig. 4. VDFG Synthesis

Disparate components are used for actors and the valves, interconnected via point-to-point communication links represented by the VDFG edges. The separation of the computation and memory requirements of the application allows independent synthesis of each. Hence for the remainder of this work we assume PEs are realised either using predesigned components, or created by a HLS synthesis tool such as Catapult-C or AutoESL [2].

The RTL architecture of a valve is shown in Fig. 5. It is composed of a single buffer element with a decoupled read-/write controller. The buffer contains, at any one time, the entire contents of the input space for a single firing of the valve. The read machine is responsible for loading the buffer with the input space data. Once loaded, the write machine reads the data according to F to satisfy the output production rules. Note that to allow the static analyses exploited for VDFG mapping described in the remainder of this section, F must take the form of a static affine function. This restriction has been shown to enable static compiler time analysis of signal and image processing operations, whilst enabling sufficient expressive power to specify their behaviours [9].

Synthesis of a sequence of VDF valves to a hierarchy of memory resources requires resolution of three key issues:

1. Assign each valve to a level of the memory hierarchy.

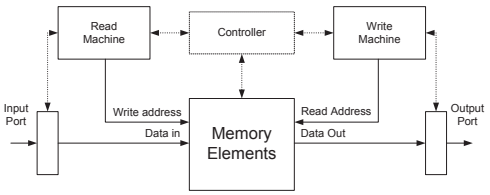


Fig. 5. Valve Architecture

2. Determine the structure of each level of the hierarchy.

3. Realise the RTL architecture of each memory element.

The RTL architecture is as described in Fig. 5, hence we focus on addressing the assignment and merging problem in the remainder of this section.

4.2. Valve To Hierarchy Mapping

To address the assignment problem, i.e. mapping the n valves along an edge onto the m levels of memory hierarchy, valves are initially mapped according to their capacities. Specifically, this process adheres to the mapping rule in Table 1.

Table 1. Valve Mapping Rules

Capacity ($\times 10^3$ words)	Target Memory Resource
0.256	LUT RAM
0.256 - 36	BRAM
> 36	Off-chip

As a result of this mapping, the capacity constraints of the application will be met, but it may be that the bandwidth constraints of the application are violated. In particular, this is likely to be the case at the interface between off-chip and on-chip layers, since the bandwidth of off-chip memory is quite restricted (see Table 1). In the case where the aggregate bandwidth of the valves mapped off-chip exceeds the maximum which may be supported by the target platform, the valves are redistributed. This redistribution process is outlined in Fig. 6, where the most bandwidth-demanding valves mapped off-chip are moved on-chip (i.e into BRAM) until the aggregate off-chip bandwidth can be feasibly supported by the platform.

To deal with bandwidth violations in the on-chip layers of memory hierarchy an alternative strategy may be employed, i.e. exploiting more than one memory unit in parallel to boost the bandwidth of that hierarchy layer. The minimum number of disparate memory components required for the output valves at each level is given by $W = \frac{BW(o)}{BW(M)}$, where $BW(o)$ is the bandwidth of the valve output port, $BW(M)$ is the maximum bandwidth which the buffer may support. In practice a more suitable value, normally the next largest interger factor of $BW(o)$ should be chosen so that the bandwidth is evenly spread across the *W* elements.

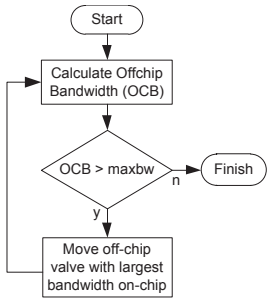


Fig. 6. Bandwidth-Based Valve Redistribution

5. RESULTS

We have tested this approach by automating the synthesis of SystemC-based VDF descriptions of three typical image processing kernels to Xilinx Virtex 5 SX95T FPGA architectures. The chosen kernels are:

1. Motion Estimation (ME) on 30 fps CIF video (as per H.264 Level 1.3).
2. 128×128 Matrix Multiplication (MM) at 500 matrices/s.
3. Sobel Edge Detection (SED) of 30 fps 720 p frames (as per H.264 Level 3.1).

The implementations are summarised in Table 2. Synthesis was performed using Xilinx ISE 11.3. In all cases the requested real-time performance metrics were met.

Table 2. Synthesis Results

Operation	**ME**	**MM**	**SED**
Frequency (MHz)	160	96	212
DSP48es	387	32	20
BRAMs	247	321	4
LUTs ($\times 10^3$)	36.4	13.7	0.92
Throughput	30.8 fps	688 mat/s	38.4 fps

6. CONCLUSION

The synthesis of multi-level memory hierarchies for memory intensive image processing applications on FPGA is currently a manual process, and is an open issue challenging the viability of HLS synthesis tools for these platforms. In this paper we have presented the first automated solution to this problem. By exploiting a novel dataflow dialect, VDF, which explicitly separates computation from data manipulation and communication, we have shown how these applications may be automatically mapped to a multi-level hierarchy of off-chip DDR and a combination of dedicated and programmable logic-based RAM components. As experimental evidence of the viability of this approach, we have presented the synthesis and real-time performance results of automatically generated FPGA memory hierarchies for Motion Estimation, Matrix Multiplication and Sobel Edge Detection applications. In all cases real-time processing was enabled.

7. REFERENCES

[1] BDTI, *FPGAs for DSP*, 2008.

[2] J. Cong, Bin Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs : From Prototyping to Deployment," *IEEE Trans. CAD*, vol. 30, no. 4, pp. 473–491, 2011.

[3] Xilinx Inc., "Virtex-6 FPGA Memory Resources User Guide," Tech. Rep., 2011.

[4] S. Fischaber, R. Woods, and J. McAllister, "SoC Memory Hierarchy Derivation from Dataflow Graphs," *Journal of Signal Processing Systems*, vol. 60, no. 3, pp. 345–361, 2010.

[5] S.H. Nam, J.S. Baek, and M.K. Lee, "Flexible VLSI Architecture of Full Search Motion Estimation for Video Applications," *VLSI Design*, vol. 40, no. 2, pp. 176–184, 1994.

[6] K. Denolf, S. Neuendorffer, and K. Vissers, "Using C-to-gates to program streaming image processing kernels efficiently on FPGAs," in *International Conference on Field Programmable Logic and Applications*. 2009, pp. 626–630, IEEE.

[7] Q. Liu, G.A. Constantinides, K. Masselos, and P.Y.K. Cheung, "Combining Data Reuse with Data-Level Parallelization for FPGA Targeted Hardware Compilation: A Geometric Programming Framework," *IEEE Trans. CAD*, vol. 28, no. 3, pp. 279–280, 2009.

[8] E. Deprettere and T. Stefanov, "Affine Nested Loop Programs and their Binary Parameterized Dataflow Graph Counterparts," in *Int. Conf. Application-specific Systems, Architectures and Processors*, 2006, pp. 186 – 190.

[9] H. Nikolov, T. Stefanov, and E. Deprettere, "Multi-processor system design with ESPAM," *4th Int Conf on Hardware/Software Codesign and System synthesis*, p. 211, 2006.

[10] Michael Fingeroff, *HLS Blue Book*, 2009.

[11] E A Lee and T M Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.