Georgia Institute of Technology
Center for Experimental Research in Computer Systems

A Virtualized Quality of Service Packet Scheduling Accelerator

Technical Report: GIT-CERCS-08-02

Kangtao Kendall Chuang, Sudhakar Yalamanchili,
Ada Gavrilovska, and Karsten Schwan

May 21, 2008

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

DB	Decision Block.
DWCS	Dynamic Window-Constrained Scheduler.
FPGA	Field Programmable Gate Array.
QoS	Quality of Service.
RBB	Register Base Block.
VPID	Virtual Process Identifier.

LIST OF LISTINGS

SUMMARY

This paper introduces the virtualization of a Quality of Service Packet Scheduler. Virtualization in terms of resource sharing among multiple processes of virtual packet schedulers implementing the DWCS algorithm on an FPGA is implemented in the ShareStreams-V architecture. This work builds on the previous work in ShareStreams, which implemented the Dynamic Window-Constrained Scheduler algorithm. This implementation is parametric, permitting tradeoffs between packet decision latency, decision throughput, and the number of virtual packet schedulers supported. ShareStreams-V is able to schedule minimal size packets faster than one decision per 51.2 ns for up to 64 streams, the throughput required for 10Gbps Ethernet.

CHAPTER I

INTRODUCTION

The recent trends in computing have been an rise in the number of cores and the use of custom accelerators to increase performance for a variety of applications. Multicore processors are able to achieve greater parallelism, while custom accelerators such as graphics processors (GPU), digital signal processors (DSPs), and field programmable gate arrays (FPGA) achieve greater performance for specific applications. Virtualization, a technology that allows the abstraction and sharing of resources, has emerged as a technology to manage the sharing of these resources.

Virtualization of FPGA-based accelerators in particular presents several challenges ranging from the choice of programming abstractions, management of accelerator state, and minimization of overhead of switching between virtualized accelerators. This paper addresses the challenges of FPGA-based virtualization in a specific context: virtualization of accelerators for wire-speed quality of service (QoS) packet scheduling. It presents the design and implementation of ShareStreams-V, a QoS packet scheduling accelerator with support for virtualization.

The background to this work is introduced in the next three sections. Section 1.1 introduces the concept of virtualization, section 1.2 presents the Dynamic Window-Constrained Scheduler (DWCS) algorithm [15], and section 1.3 presents the non-virtualized hardware realization of the algorithm ShareStreams [10] [11]. Finally, section 1.4 presents an outline of this paper.

1.1 Virtualization

It is anticipated that operationally future systems will have time-varying number of processes each of which will utilize some number of packet streams that will be subject to QoS requirements. Customized hardware is typically necessary to meet real-time constraints since embedded processors cannot deliver the necessary throughput in scheduling decisions per second. However, given that the scheduler should be sized to meet a range of deployment scenarios ASIC solutions are not cost effective. Replication and static sizing of hardware packet schedulers typically leads to over-design and consequently inefficient use of hardware resources. However, virtualization of FPGA-based solutions can enable consolidation and efficient use of hardware scheduling resources that can be reconfigured over periods of time, e.g., size of the scheduler, to adapt to changing workloads. This aspect of virtualization is different from past characterizations of virtualization in an FPGA context.

An overview of resource virtualization in general, including topics of security, performance, and reliability, has been presented by Figueiredo et al. in [5]. Resource virtualization, specifically the virtualization of network routers not in the context of FGPAs, has been studied by McIlory, R. and Sventek, J. in [12]. McIlroy et al. study the use of "virtual routelets" for Quality of Service network routing using "virtual routelets." Their work is different from the current work in terms of the application and the target architecture – this work studies the virtualization of a packet scheduler accelerator implemented on an FPGA.

A recent paper has categorized approaches to FPGA virtualization as follows [13]: temporal partitioning, virtualized execution, and virtual machine approaches. Temporal partitioning techniques facilitate the realization of designs too large to fit in the FPGA necessitating partitioning and sequencing. FPGA virtualization in terms of partitioning of a design for a larger or smaller FPGA fabric is also presented by Fornaciari and Piuri in [6]. Virtualized execution refers to portability across a family of reconfigurable devices. Virtual machine refers to the implementation of an "abstract computing architecture" in the FPGA such that a target application or operating system can be run.

Cardoso presents an algorithm and design flow for partitioning and virtualization of units during high-level synthesis of a hardware design [4]. The algorithm presented aims at minimizing the number of temporal partitions and the execution latency.

Goldstein et al. implement a specific architecture and virtualization of the logic for the architecture on FPGA fabric in [7]. In their architecture, the dynamic reconfiguration capability of FPGAs was used to enable larger designs to be implemented on an FPGA. Designs are partitioned during compilation, and configuration and execution of different "pipeline stages" on the FPGA occurs in parallel during runtime to achieve greater utilization of the FPGA.

Guo et al. present a packet scheduler that balances loads from multiple streams in a heterogeneous network processor in [8]. This is different from ShareStreams-V in that ShareStreams-V provides scheduling for streams that are scheduled independently for different processes (i.e. streams belonging to one process will not affect the priority of streams belonging to another process).

1.2 Dynamic Window-Constrained Scheduler Algorithm (DWCS)

DWCS is a framework for scheduling multiple streams [15] with quality of service constraints. While the framework is quite general, the main topic of concern is with scheduling packet streams. The scheduler examines the attributes of the packet at the head of each stream, for example deadline, and determines the packet with highest priority based on pair-wise comparisons from Table 1. The highest priority packet is allocated to the wire, and then the packet attributes for all of the streams are adjusted based on the winner packet.

For any stream, DWCS ensures that at most X packets within a window of Y packets will miss their deadline (and therefore be dropped), where each packet has a periodic deadline determined by a request period. The window constraint parameter for each stream is $\frac{X}{Y}$. The window constraints (numerators and denominators) and deadlines are updated every time a winner is determined.

Table 1: DWCS Packet Priority [15]

1. Earliest deadline first
2. Equal deadlines, order lowest window-constraint ($\frac{X}{Y}$) first
3. Equal deadlines and zero window-constraints, order highest window-denominator (Y) first
4. Equal deadlines and equal non-zero window-constraints, order lowest window-numerator (X) first
5. All other cases: first-come-first-serve

Hardware acceleration of the Dynamic Window-Constrained Scheduling algorithm in the ShareStreams architecture [10, 11] was motivated by the high scheduling decision throughput required for operation over 10Gbps Ethernet links. The ShareStreams architecture was designed to optimize the DWCS algorithm in hardware in terms of area and decision time. ShareStreams was originally implemented on the Xilinx Virtex-I and Virtex-II FPGAs in [11]. ShareStreams-V is a redesigned version of ShareStreams with both hardware and software support for virtualization. The original ShareStreams architecture is briefly reviewed in the next section.

1.3 ShareStreams Architecture

Table 2: Register Base Block (RBB) [11]

Stream Attribute Registers
Deadline and Current Time Comparator
Deadline Miss Window Constraint Update Logic
Winner Window Constraint Update Logic
Window Constraint Update Multiplexers

The base ShareStreams design for an n stream scheduler, where n is a power of 2, consists of three main components: the control units, the register base blocks (RBB), and the recirculating shuffle-exchange network comprised of multiplexers and decision blocks

(DB). The control units are responsible for communication with the host CPU as well as execution of the recirculating shuffle-exchange network. The register base blocks store and update unique stream state information - this is the information for the packet at the head of the stream such as deadline or arrival time. This stream state information is quality of service attributes that are stored in the packet header. An overview of the registers and the logic in the RBB is shown in Table 2.

The control block sends signals to the register blocks, network, and the decision blocks. After each scheduling decision, the RBBs must be updated. For example, the RBB corresponding to the winner must be updated with information from the next packet in the stream. All other streams must have their window constraints updated.

The recirculating shuffle-exchange network is the core of the DWCS algorithm and chooses the winner packets based on the pair-wise comparisons of the stream states. These pair-wise comparisons for two packets are based on a valid bit comparison and the DWCS priorities, and produce both a winner and a loser. The recirculating shuffle-exchange network is used to route the comparisons of the n streams with n/2 decision blocks (DB), and produces the winner packet within $\log_2(n)$ cycles. Then an additional cycle is used for the "priority update", in which the stream state information is updated based on the winner packet. Thus the latency of a ShareStreams decision is $\log_2(n) + 1$, and the throughput is $\frac{1}{\log_2(n)+1}$.

As an example, the Four-Stream Scheduler is shown in Figure 1. The controller writes the stream state information to the RBBs, and then sends control signals to the multiplexers and latches after the decision blocks each cycle. This design can be easily extended to create an n-stream scheduler, where n is a power of 2, by creating n RBBs, n multiplexers, and $n/2$ decision blocks. The control block shown communicates with the host software. At the end of each decision cycle, there is one winner packet whose ID is added to the winner first-in-first-out (FIFO). As a function of the number of streams, the ShareStreams scheduler requires a linear increase in logic with logarithmic increase in decision time. This

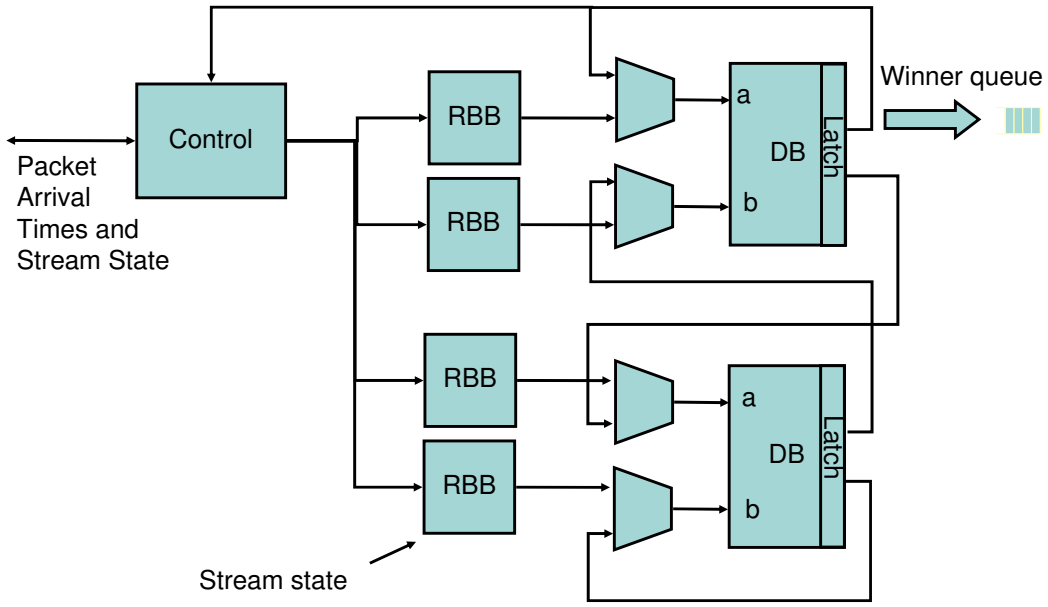

Figure 1: Four-Stream Scheduler Data Flow Diagram

property should be preserved in the virtualized version.

1.4 Outline

This work presents a virtualized hardware accelerator which implements of the Dynamic Window-Constrained Scheduling algorithm. This paper first details alternative approaches to the virtualization of a FPGA-based scheduler accelerator and as a consequence proposes ShareStreams-V - a virtualized hardware implementation of the Dynamic Window-Constrained Scheduler (DWCS) [15] algorithm for quality of service packet scheduling. This work builds upon the previous work on ShareStreams in [10, 11] by adding support for virtualization.

The hypothesis of this work is the following: Virtualization of a quality of service

packet scheduler accelerator through dynamic spatial partitioning is a feasible approach to sharing the accelerator, and will produce one additional decision for an additional process at the cost of one additional cycle of latency.

The goal of virtualization is to share an n stream physical scheduler among multiple processes such that each process can access a distinct k stream virtual scheduler. The objectives of the ShareStreams-V architecture are to implement support for virtualization and maximize the performance in terms of packet decision throughput while minimizing the costs in terms of decision latency clock frequency and area. The target application and benchmark of the implementation is scheduling packet streams at wire-speeds for 10Gbps Ethernet.

The implementation is parametric, permitting tradeoffs between packet decision latency, decision throughput, and degree of virtualization - this is the number of virtual packet schedulers available to be allocated across host processes. The FPGA implementation enables new schedulers representing different trade-offs to be swapped into the FPGA over time as a function of workload mix. For example, serving a large number of processes with a smaller number of packet streams each over a 1Gbps link will engender a different virtualized solution than serving a small number of processes with a larger number of streams over a 10Gbps link. ShareStreams-V is able to schedule minimal size packets faster than one decision per 51.2ns for up to 64 streams, the throughput required for 10gbps Ethernet. At this rate, the bottleneck is currently the host-accelerator HW/SW interface (e.g. over PCI Express link). It is anticipated that future designs will be over tightly coupled native or commercial interfaces, for example AMD's HyperTransport or Intel's Common System Interface.

This paper presents four techniques for virtualization of the DWCS accelerator and how they differ from the base architecture in Chapter 2. One of these techniques is implemented in the ShareStreams-V system, and a description of the implementation is given in Chapter

3. The experimental results of synthesized and tested on a Xilinx Virtex-4 FPGA to measure the scheduling performance with regards to 10Gbps Ethernet. The synthesis results, analysis of design tradeoffs, and software runtime results are discussed in Chapter 4, and concluding remarks and directions for future work are provided in Chapter 5.

CHAPTER II

VIRTUALIZATION TECHNIQUES

The goal of virtualization is to share an n stream physical scheduler among multiple processes such that each process can access a distinct k stream virtual scheduler. A few of the challenges involved in the design include managing context switch overhead and multiplexing logic, minimizing decision latency, and increasing decision throughput. Functionally the decision throughput of the physical scheduler should be allocated across the virtual schedulers to produce the behavior shown in Figure 2 where all streams that belong to a process. The streams belonging to a particular virtual scheduler are labeled with a virtual process identifier or VPID, and are scheduled together, but independent from streams belonging to other processes. This property, that streams belonging to one VPID do not affect the priority of streams belonging to another VPID, is important to maintain the correctness of the DWCS algorithm.

In this chapter, the following four design approaches to virtualization are introduced and compared:

1. Coarse-Grained Temporal Partitioning
2. Fine-Grained Temporal Partitioning
3. Spatial Partitioning
4. Dynamic Spatial Partitioning

The first two approaches are temporal sharing of a hardware scheduler where the state is swapped much like threads sharing a processor core. Such sharing can be coarse- or fine-grained. The advantage is that the full hardware decision bandwidth of the physical design

is available to each VPID while having to bear the overhead of context switches (coarse-grained) or additional hardware (fine-grained). The second two approaches are based on spatial partitioning. In this case the hardware is partitioned into smaller designs that are allocated to VPIDs. This is simpler but places an upper bound on decision throughput that can be allocated to a process, independent of how many processes are actually using the scheduler at any time while this upper bound decreases as maximum number of possible VPIDs increases. A virtualization solution that is based on dynamic spatial partitioning is proposed and implemented based on its high performance increase at a low cost in terms of additional logic .

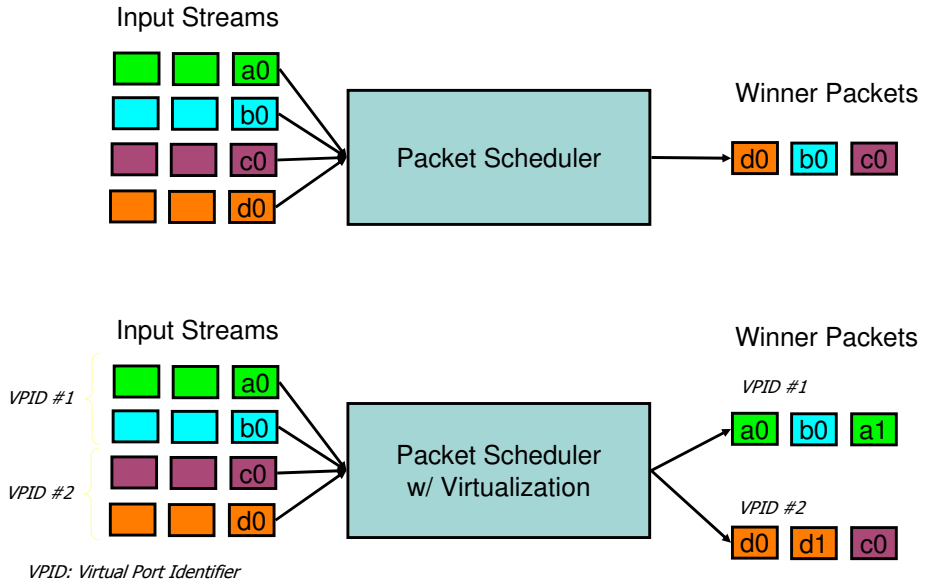

Figure 2: Virtual Packet Scheduler

2.1 Coarse-Grained Temporal Partitioning

The coarse-grained temporal partitioning design is implemented in terms of a context switch when switching between VPIDs. The cost of the context switch is the time spent loading and unloading stream state information from the register base blocks, which is effectively lost scheduling time. With the addition of VPIDs the scheduler can be shared across multiple processes in a straightforward manner. The context switch time overhead will be high for the coarse-grained temporal partitioning implementation, as all of the stream information will have to be loaded and unloaded from the hardware scheduler whenever a context switch occurs.

A similar context switching is discussed in the original implementation of ShareStreams [11]. In that version, software support and on-chip memory are used to implement context switches in terms of sets of streams, at the cost of increased decision latency. The advantage of that approach is that a larger number of streams may be scheduled than the hardware resource is designed to handle. For example, 64 streams can be scheduled on a 16-stream hardware scheduler by scheduling 16 streams each in four stages and then taking the four winner packets and scheduling those in the fifth stage. The main difference between that approach and the approach discussed here is in sharing across processes via the use of VPIDs.

2.2 Fine-Grained Temporal Partitioning

Fine-grained temporal partitioning exploits the fact that the core scheduler cannot be pipelined since a winner is necessary before the next scheduling round can begin. Therefore the scheduler can be pipelined to d stages, and the execution of d virtual schedulers can be interleaved. Effectively, the recirculating shuffle-exchange decision network is unrolled and pipelined such that each pipeline stage contains the shuffle routing, the decision blocks, and latches, as can be seen in Figure 3. The advantage of pipelining is that the decision

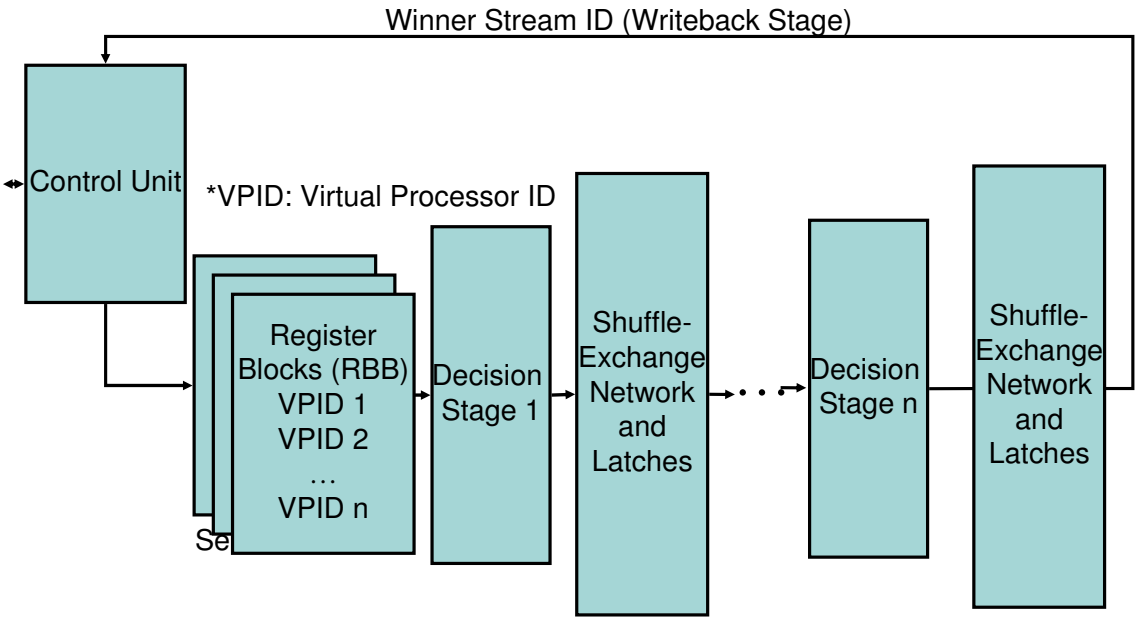

Figure 3: Fine-Grained Temporal Partitioning

throughput in total is increased by a factor of d, although each individual scheduler is limited to that of the base physical design. The increased hardware is also accompanied by a need to have each stage of decision blocks handle a distinct VPID.

Fine-grained temporal partitioning is also referred to as C-Slow. This is a more generic term for pipelining a feedback loop. The dependencies in the feedback loop are removed by having each pipeline stage that is created correspond to a thread that is independent of all other threads in other pipeline stages.

To add support for multiple VPIDs, support for more than one set of RBBs must be added. This is enabled by switching between the sets of register blocks. An example of this is shown in Figure 4. In order to maintain (and not increase) the latency of $\log_2(n) + 1$ cycles for n streams, a maximum of $\log_2(n) + 1$ VPIDs is supported in this design. In the

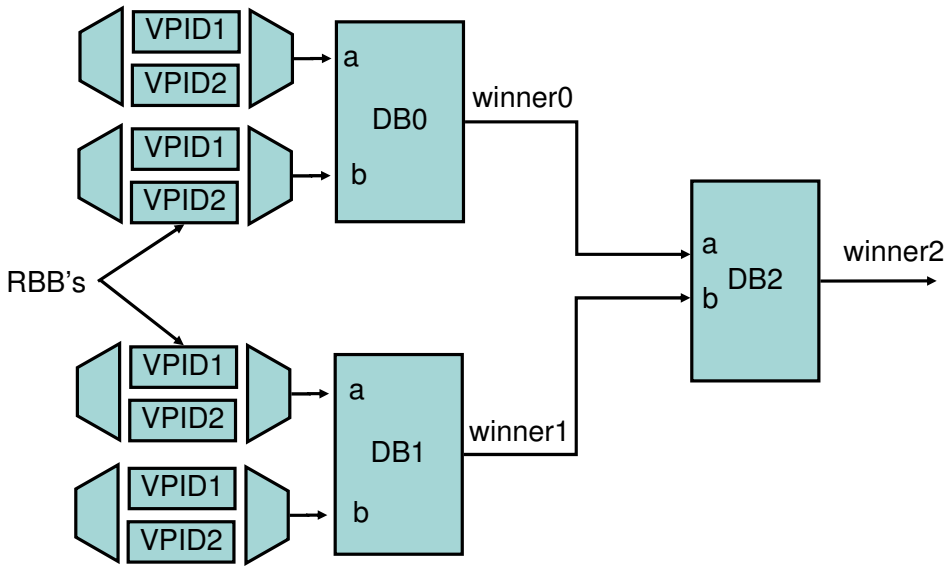

Figure 4: Unrolling and Pipelining Shuffle-Exchange Network

design in Figure 4, up to two VPIDs are supported.

The advantage of this method is that the loading can be overlapped with execution between different processes, and once all processes have finished loading and begun execution the maximum amount of parallelism can be achieved. The disadvantage is the addition of multiple pipeline stages, decision blocks, and multiplexers will require more logic, and additional control signals and routing will be needed to manage the additional multiplexers, demultiplexers, and registers.

This method is different from the original ShareStreams approach to pipelining [11]. In that work, the architecture was pipelined in terms of stream state without increasing the number of register blocks, but at the cost of decision latency (which increases linearly with the number of additional streams). In the fine-grained temporal partitioning approach,

pipelining increases both the number of register blocks and the number of decision blocks, but does not increase decision latency.

2.3 Spatial Partitioning

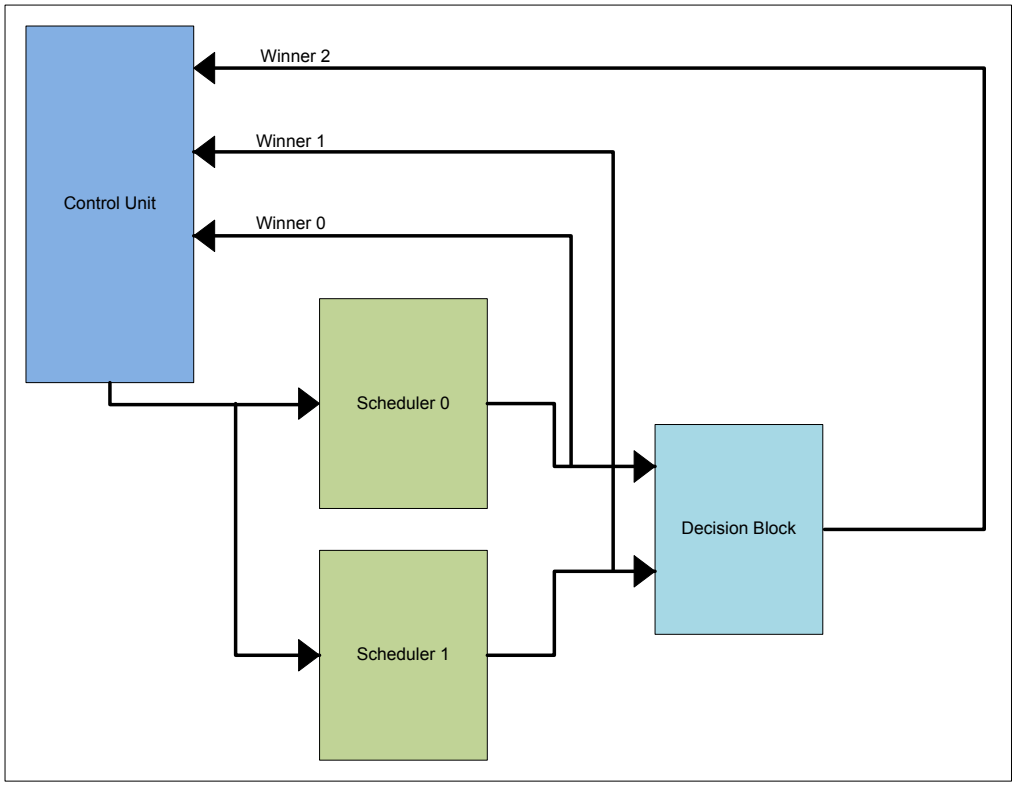

Figure 5: Spatial Partitioning

The base scheduler architecture is virtualized through spatial partitioning by splitting the scheduler into several basic scheduler blocks and connecting them through a pipelined network of decision blocks. Each basic scheduler block is functionally equivalent. If one process on the host processor requires a larger scheduler than can fit in a basic scheduler block, it can request access to two or more scheduler blocks. An example of such a design is shown in Fig. 5. In this design, results can be read from the Winner0 and Winner1 lines if Scheduler0 and Scheduler1 correspond to different VPIDs. Otherwise if they correspond

to the same VPID then the result would be read from the Winner2 line.

This design has some flexibility in that the granularity can be increased or decreased by making the size of the basic scheduler blocks larger or smaller. The tradeoff is that the number of winner lines will have to be increased or decreased accordingly, which will lead to additional complexity in terms of reading winners from the block outputs. In addition, each base scheduler must have its own control unit, which will lead to additional area on the chip. One advantage to this technique is that latency can be reduced for processes which use fewer streams and thus fewer basic scheduler blocks. In the design shown in Fig. 5, Winner0 and Winner1 will be ready one cycle before Winner2 is ready. This means that if a process only needs to use one basic scheduler it will save one clock cycle in the hardware. The disadvantage is that software complexity is increased, in terms of stream allocation and reading out the winner streams.

2.4 Dynamic Spatial Sharing

Based in part on the preceding observations, an approach that integrates the VPIDs into the register base blocks and decision blocks, as can be seen in Figure 6, was chosen for implementation. This addition of VPIDs to the RBBs and DBs would involve support for another register in each Register Base Block and an additional comparison in each Decision Block. The pair-wise comparisons of the streams will use the rules shown in Table 2.4.

The advantages of this technique are that any register block can hold stream state information for any process, and the performance improvement, in terms of decision throughput for proportional increase in unit area, is greater than for the other techniques. Thus, this technique was chosen to be implemented in ShareStreams-V.

By adding the comparison of VPID to the decision block pair-wise comparisons, the dynamic spatial sharing architecture will compare the packets based on both their priority and the VPID. And by making the VPID comparison first, packets will be grouped together by VPID. This will facilitate the partitioning of the winner streams at the output. There is

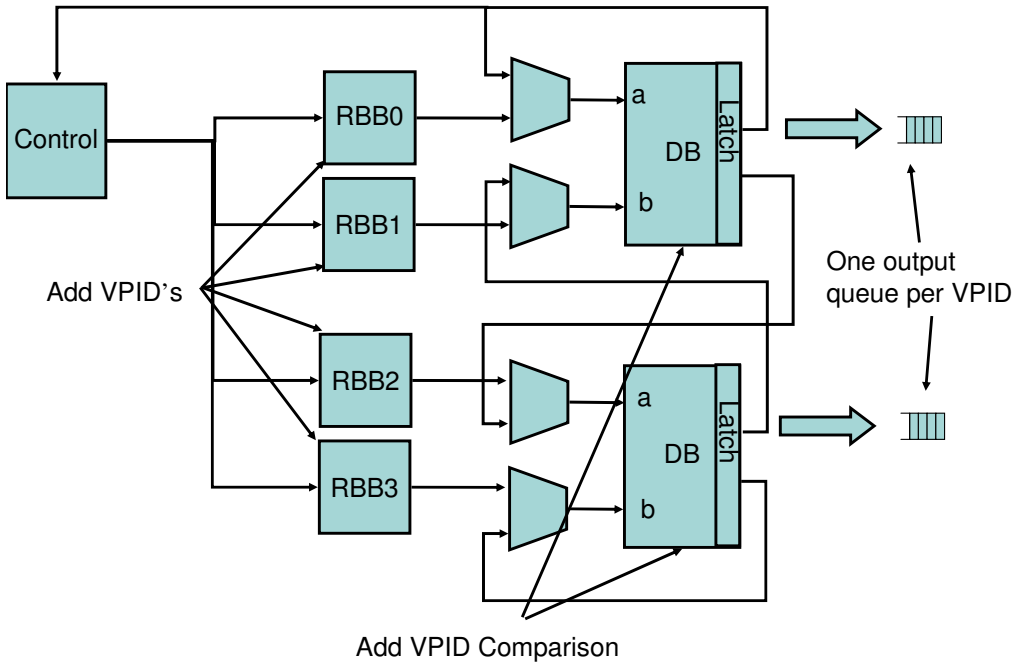

Figure 6: Four-Stream Scheduler with Virtual Port Identifiers

Table 3: ShareStreams-V Packet Priority

1. Lowest VPID First
2. Highest valid bit first
3. Earliest deadline first
4. Equal deadlines, order lowest window-constraint (X/Y) first
5. Equal deadlines and zero window-constraints, order highest window-denominator (Y) first
6. Equal deadlines and equal non-zero window-constraints, order lowest window-numerator (X) first
7. All other cases: first-come-first-serve

a latency cost of one additional cycle per additional VPID above the first in the "priority update" stage as discussed before in 1.3; this is necessary because the streams states for

Table 4: Comparison of Virtualization Techniques; v is the degree of virtualization; d is the maximum pipeline depth = $\log_2(n) + 1$; g is the maximum partition granularity = $\frac{n}{2}$

	Base Design	Dynamic Spatial Sharing	Spatial Partitioning	Coarse-Grained Temporal Partioning	Fine-Grained Temporal Partioning
VPID	1	$v \leq \frac{n}{2}$	$v \leq g$	v	$v \leq d$
RBB	n	n	n	n	$n \times v$
DB	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2} + \sum_{k=0}^{\log_2(g)-1} 2^k$	$\frac{n}{2}$	$\sum_{k=0}^{\log_2(g)-1} 2^k$
Throughput	$\frac{1}{\log_2(n)+1}$	$\frac{v}{(\log_2(n)+v)}$	$\frac{v}{log_2(\frac{n}{v})+1}$	$\frac{1}{\log_2(n)+1}$	$\frac{v}{\log_2(n)+1}$
Latency	$\log_2(n) + 1$	$\log_2(n) + v$	$\geq \log_2(\frac{n}{v}) + 1$ $\leq \log_2(n) + 1$	$\log_2(n) + 1$	$\log_2(n) + 1$

different VPIDs must be updated independently from each other.

Therefore, a design for eight VPIDs will add eight cycles of latency for a scheduling decision. However, the throughput of scheduling decisions per unit time is increased over the base design. Essentially, in this longer latency cycle, multiple scheduling decisions for multiple VPIDs are being concurrently being generated. Thus, the choice of the degree of virtualization is a tradeoff between overall system scheduling throughput, latency of individual scheduling decisions, and degree of sharing (virtualization).

2.5 *Summary of Techniques*

The number of RBB and DB modules and performance of each of the designs is shown in Table 4.

Each of the techniques discussed leads to design tradeoffs. Coarse-grained temporal partitioning has the largest area overhead due to context switch, as can be seen in the RBB and DB columns as compared to the base implementation in Table 4. The advantage is that it requires the lowest increase in physical logic to enable, as most of the support is configured in software. However, this requires that software complexity be increased. On the plus side, this technique can be combined with any of the other techniques to allow

for a higher degree of virtualization supported at the time cost of context switching. Fine-grained temporal partitioning is able to achieve a very high throughput but at a large area cost for the multiplexing logic required to implement it (this cost is not shown in the table).

Spatial partitioning is able to achieve both higher throughput and lower latency at the cost of control complexity, and thus an increased area required for control logic in the hardware. This technique has a greater potential granularity in terms of flexibility of streams available to processes than fine-grained temporal partitioning, and also has a different type of area cost.

The final technique, dynamic spatial partitioning, was chosen for implementation based on to the degree of virtualization and performance improvement achieved through a small increase in area (i.e. one additional register in each RBB, one additional comparator in each DB, and one additional winner stream multiplexer for the design).

Part of the hypothesis proposed by this work, that dynamic spatial partitioning will produce one additional decision for an additional process at the cost of one additional cycle of latency, is shown in the formulas for latency and throughput. The latency formula for this architecture is $\log_2(n) + v$ and the throughput formula for this architecture is $\frac{v}{(\log_2(n)+v)}$. One additional decision is produced for an additional process (corresponding to a VPID v) at the cost of an additional cycle of latency for the priority update (corresponding to the v in the latency equation).

The degree of virtualization v in these formulas is equivalent to the maximum number of processes (or number of VPIDs) supported in a particular architecture. The variable n is the number of physical RBBs in the architecture. For each process above the first one, the original decision latency, $\log_2(n) + v$, will increase by one. The reason for this increase in latency is because in this architecture the stream registers corresponding to each VPID must be updated through the same bus, and thus in a different cycle. Thus, there is one "priority update" cycle for each VPID.

The theoretical performance statement of the hypothesis has been shown for the dynamic spatial partitioning design. The other part of the hypothesis, the feasibility of the implementation of dynamic spatial partitioning, is demonstrated in the following chapters. Chapter 3 describes the system implementation including both the hardware and software, and Chapter 4 describes the experimental results of the implementation.

CHAPTER III

SHARESTREAMS-V IMPLEMENTATION

The ShareStreams-V system is divided such that the control and virtualization interface is mostly handled by the software, while the core execution of the DWCS algorithm occurs in hardware. The rest of this chapter will present the feasibility of the hypothesis through an overview of the ShareStreams-V implementation: the system, the software and execution model, and the hardware.

3.1 System

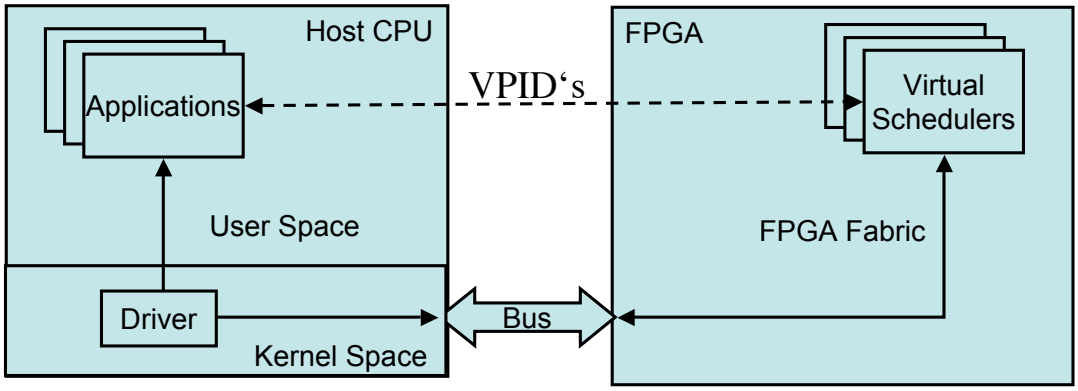

Figure 7: ShareStreams-V System

The ShareStreams-V system is implemented with a host processor coupled to an FPGA through a bus as illustrated in Figure 7. The processes on the host CPU access hardware through the ShareStreams-V software interface which communicates with the FPGA. On

the FPGA resides the physical scheduler core, which implements the core DWCS algorithm.

Control and data signals flow through the interconnection bus as can be seen in Fig. 7. Ideally this is a low-latency bus so that the scheduler input data (packet arrival times, stream state attributes, etc) can be streamed in and output data (winner packet streams) can be streamed out of the scheduler with low time overhead.

In the current implementation the software interface resides completely on the host processor core, a general purpose processor. The software was compiled using gcc 4.1.2 and experiments are conducted on a dual Intel(R) Core(TM) 2 Quad processor, which has four cores at 2.66GHz. This platform running Fedora 7 as the operating system serves as the host CPU for the ShareStreams-V system. The physical packet scheduler accelerator is implemented on a Alpha Data ADPe-XRC-4 PCI Express FPGA board is running on a 4x PCI Express bus, and the control was implemented with a 32-bit bus running at clock speed to the local FPGA. The Alpha Data board has a Virtex-4 FX140 chip which has an embedded PowerPC 405 processor [2]. In future implementations this 32-bit bus may be connected to the PowerPC 405 hard core embedded in the Xilinx Virtex-4 fabric.

3.2 Software

The ShareStreams-V software portion running on the host CPU is key to enabling the virtualization of the hardware in the current implementation. The management of virtualization and VPIDs, allocation of register base blocks, and main control signals.

The principle control functions between the hardware and software include:

- Add Stream - Load RBB
- Delete Stream - Reset RBB
- Pause Scheduler
- Run Scheduler

- Update Stream Information - Packet Arrival Times
- Read Winner Stream - Stream ID, Packet Arrival Time/Packet ID, and VPID

For "client" processes that wish to access the virtual scheduler through ShareStreams-V, the following functions are available to them:

- Add Scheduler
- Delete Scheduler
- Add Stream
- Delete Stream
- Update Stream Information
- Read Winner Stream

3.2.1 ShareStreams-V Execution Model

The execution model is one wherein a process p on the host CPU requests a k-stream scheduler that is allocated by the Sharestreams-V system that returns a handle for the virtual scheduler, called the Virtual Process Identifier (VPID). The process can subsequently start the scheduling, add and delete streams, query for the next packet to be scheduled and update the state of winner streams with information about the next packet in the winning stream.

The ShareStreams-V software application supports an API that provides for interaction between the process and its allocated virtual scheduler. The application supports functions to reset the hardware IP, add and delete streams or sets of streams (for an application), write packet arrival times to the input FIFOs, read the winner packets from the winner FIFO, and start and pause the scheduler execution. It implements the allocation algorithm described below to allocate register base blocks to virtual schedulers that are assigned to applications.

3.2.1.1 *Initialization Functions*

The ShareStreams-V software application runs the following steps to initialize the hardware scheduler:

1. Reset Scheduler
2. Initialize VPIDs for RBBs
3. Load physical scheduler with stream states (into RBBs) from each process (i.e. for each VPID)
4. Load packet arrival times
5. Start physical scheduler execution

After resetting the physical scheduler, the RBBs are initialized to enable the shuffle-exchange partitioning that is essential for the dynamic spatial partitioning technique to function. The RBBs corresponding to each active virtual scheduler are loaded, and then an initial set of packet arrival times loaded. Then the execution starts.

3.2.1.2 *Main Loop*

The main loop is as follows:

```
// main loop
while(1) {
    if (p.request) { // request to add or delete a scheduler
        Pause(); // pause the scheduler
        // service the request here
        if(add_scheduler) {
            Allocate(p); // allocate RBBs to scheduler
            LoadRBB(p); // load stream state information
        }
        else if(delete_scheduler) {
            DeleteRBB(p); // delete stream state information in ↓
→               hardware
            InitRBB(p); // reinitialize RBBs corresponding
```

```
                DeAllocate(p); // deallocate RBBs from scheduler in ↓
→                   software
            }
            Start(); // continue the scheduler execution
16      }
        ReadWinner(); // read the winner packet from the scheduler
    }
```

Listing 1: The main loop pseudocode.

As can be seen in the code, the main loop consists of updates to the stream states and streaming packet arrival times into the physical scheduler and reading the winner stream out from the physical scheduler. The less time spent adding/deleting streams or sets of streams means more time spent in the core execution of the DWCS algorithm, which results in a higher throughput. The next section presents the partitioning of the shuffle-exchange network and the RBB allocation algorithm.

3.2.2 Partitioning and Allocation

The allocation algorithm is a critical part of the ShareStreams-V dynamic spatial partitioning design. The following two parts describes the partitionability of the shuffle-exchange network, and then the buddy allocation algorithm, both of which enable ShareStreams-V to produce separate winner streams for each VPID.

3.2.2.1 Shuffle-Exchange Network Partitioning

The concept that enables dynamic space sharing and efficient partitioning of the hardware scheduler into virtual schedulers as described above is the partitionability of the Shuffle-Exchange Network as described by Siegel in [14]. The key to partitioning the Shuffle-Exchange Network in ShareStreams-V is to add an additional VPID comparison in the decision blocks and then utilize the inherent partitionability of the shuffle exchange network.

For an n-stream scheduler, where n is a power of two, the number of recirculating

shuffle-exchange cycles required to obtain a scheduling decision is $\log_2(n)$. This is functionally equivalent to having a network with $\log_2(n)$ shuffle-exchange stages rather than recirculating the packet information through a single stage. This latter $\log_2(n)$ stage network is the well know Omega Network. The ability to partition the Omega network has been well studied and is governed by two constraints in [14]:

1. The size of each partition must be a power of two.

2. The physical addresses of the input/output ports of a partition of size 2^s must all agree in any fixed set of $m - s$ bit positions, where m is the number of stages of the shuffle-exchange.

The first constraint is met by only allocating virtual schedulers whose size is a power of two. For example, a hardware scheduler that can support eight streams can be partitioned into two four-stream virtual schedulers, four two-stream virtual schedulers, or one four-stream virtual scheduler and two two-stream schedulers. The scheduler allocation algorithm is described in Section 3.2.2.2.

The host software uses the following steps in order to satisfy the second criteria for partitioning:

1. Initialize the RBBs so that they are ordered in VPID from low to high

2. Allocate a power of 2 number of RBBs to a process by using the buddy allocation algorithm described below.

3. Assign a VPID for the process which maintains the low to high ordering within the physical scheduler

The second constraint is met by the addition of a VPID register to the to each RBB as well as a comparator for two VPIDs in the DBs. The VPIDs are always ordered from lowest to highest when assigned to the RBBs, thus the first RBB in the physical scheduler

will always have the lowest VPID and the last RBB in the scheduler will have the highest VPID. The VPID comparisons correspond to the address routing of the Omega Network.

The Omega Network routes the source to the destination through m stages using an m-bit address, where m is $log_2(n)$ for an n-node network (which corresponds to an n-stream scheduler). At each stage i, the $m-i$ bit of the address determines the type of exchange that occurs; this is either pass-through or cross-over when Decision Blocks are used, since there is always one winner and one loser packet. The m-cycle recirculating Shuffle-Exchange is functionally equivalent to an Omega Network in that each stage of the Omega Network corresponds to one cycle of the recirculating Shuffle-Exchange.

For the first $m - s$ stages, different VPIDs will be compared against each other and routing will always be pass-through because of the lowest to highest ordering of the VPIDs with respect to the RBBs.

3.2.2.2 Scheduler Allocation

In order to create isolated partitions of the scheduler, the software must allocate RBBs to schedulers and assign VPIDs such that they meet the conditions outlined in the previous section. Towards this end, the Buddy Allocation Algorithm described by Knuth in [9] is implemented in the host software to meet the first criteria of partitioning that "the size of each partition must be a power of two."

This algorithm has an efficient implementation and permits the allocation of virtual schedulers with the *reserve* function, and de-allocation with the *liberate* function. The buddy algorithm utilizes a binary tree data structure where each level of a binary tree corresponds to a certain scheduler size (number of streams or RBBs), 2^k. Allocation of a scheduler of a certain size requires searching the nodes at a specific level and higher - the latter in case a larger block of RBBs must be split to get a scheduler of the required size. While the buddy allocation does produce efficient use of RBBs (following its performance analysis in applications to memory allocation), it is simple, and as will be shown in the next

section, quite effective.

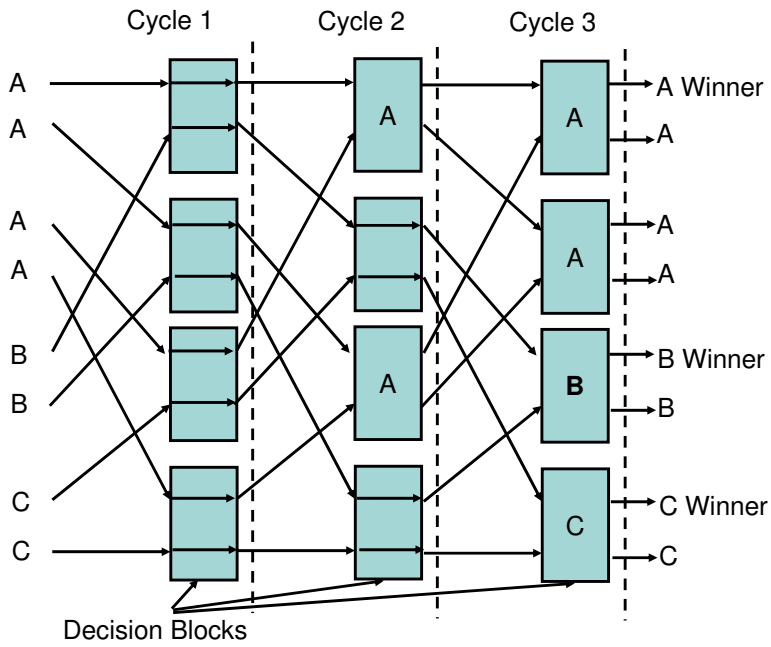

Figure 8: Partitioning Eight-Stream Scheduler into Subnetworks

An example of an allocated ShareStreams-V scheduler is shown in Fig. 8. In this example, an 8-stream ShareStreams-V scheduler is allocated using the buddy allocation algorithm described in section 3.2.2.2 with the three VPIDs A, B, and C. The scheduler is partitioned in the first stage into a subnetwork with virtual scheduler A and a subnetwork with virtual schedulers B and C. In the second stage the scheduler is partitioned again into two subnetworks B and C.

As mentioned earlier, each stage indicates a different cycle in the recirculating Shuffle-Exchange network that is actually implemented in ShareStreams-V. In the first stage, the contents from the RBBs are read into the DBs, and the winner and loser streams are latched. Then in the following $m - 1$ stages the streams are routed (i.e. the "shuffle" occurs) from

the latched results from the previous stage and then compared in the DBs (i.e. they are "exchanged").

3.3 Hardware

The hardware implements the core execution of the DWCS algorithm on the FPGA. This section describes the changes required from the base, unvirtualized hardware design to implement support for virtualization.

The basic functions and control stages are the same for the base architecture as for the virtualized architecture. However, the configuration and data transferred is slightly different, as the virtualized architecture will have the addition of the VPIDs to relevant data signals (stream state information stored in RBBs, packet arrival time information, and winner state information).

3.3.1 Implementation Details

For the virtualized version, the bitwidths for the stream state information are shown in Table 5. Of the stream state registers, the variables that are required to identify a single valid packet valid bit (1 bit), the packet arrival time (16 bits, to identify the packet), the stream ID (10 bits), and the VPID (5 bits). Thus, 32 bits can identify a packet in the virtualized design.

Table 5: Stream State Registers and Bitwidths

Valid Bit	1
Packet Arrival Time	16
Stream ID	10
VPID	5
X	8
Y	8
Deadline	16
Request Period	16

The main change to the unvirtualized architecture in the RBBs is to add the VPID, which is five bits, in order to keep the packet identifier information to a total of 32 bits. This allows one winner packet identifier to be sent on the 32-bit bus to the host CPU in one cycle. Five bits for the VPID are sufficient for up to a 32 VPID architecture, which requires a 32-bit 32-to-1 winner multiplexer such as the multiplexer seen in Figure 9. In general, support for more VPIDs requires a larger winner multiplexer, which will require more logic to implement and may potentially slow the clock frequency due to the routing of signals in the design or increasing delays in the critical path.

3.3.2 Hardware Execution

Next the execution of the hardware is described. The control functions from the software to the hardware are:

1. Start - begin scheduler execution of DWCS algorithm
2. Pause
3. Reset
4. Load RBB
5. Unload RBB
6. Load Packet Arrival Time into input FIFO
7. Read Winner Stream from winner FIFO

The hardware data flow model is shown in Figure 9. Upon a start command from the host CPU, the scheduler will go through the $\log_2(n)$ shuffle-exchange cycles. During these cycles, the winner packet for each VPID will be calculated. Then there will be v priority update cycles. During each of these priority update stages, one winner stream identifier (including VPID info) will be broadcast to all of the RBBs, so all RBBs corresponding to

that particular VPID can be updated if they are the winner or if they missed their deadline. This cycle repeats until another control signal is read from the host CPU.

The ShareStreams-V architecture was made with small changes to a base, unvirtualized architecture. This base, unvirtualized, architecture was implemented with the same parameters (data bitwidths, organization) as the original ShareStreams architecture [11]. A description of the original architecture was described on ShareStreams in section 1.3. The differences between the virtualized and the base architectures are:

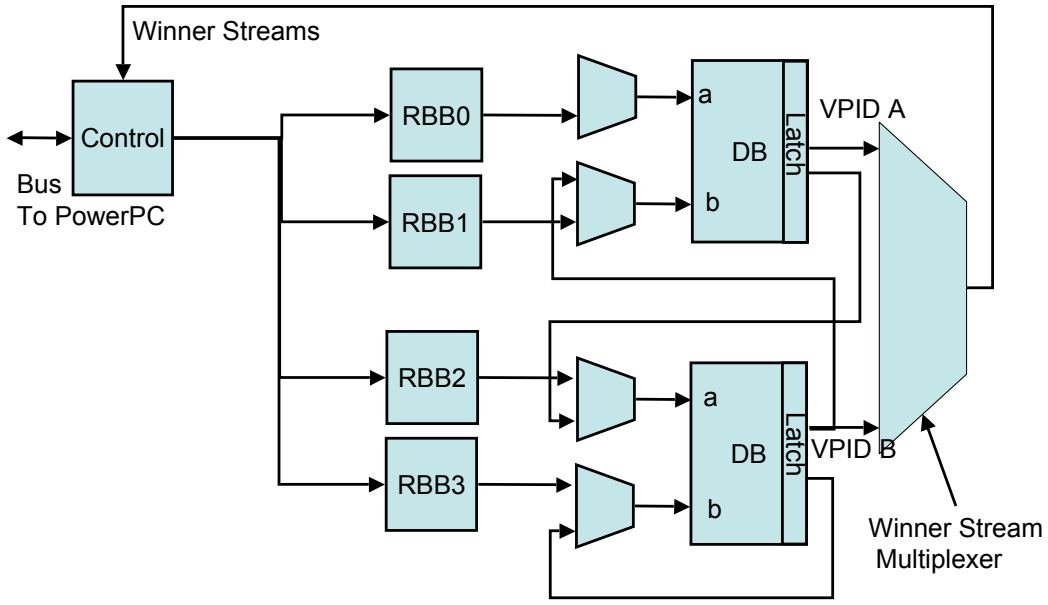

Figure 9: Dynamic Spatial Partitioning Data Flow Diagram

There is control logic to allow for the additional priority update cycles, which contribute to the increase in decision latency. During the priority update cycle, the register base blocks must be updated with new stream state information. The output multiplexer, the rightmost block in Fig. 9, is v-to-one, where v is the degree of virtualization (i.e. the number of

VPIDs). The value of v is fixed for a specific architecture, and must be a power of 2. The value of v is also the number of priority update cycles in the architecture, since each VPID will have its own winner stream. Thus the decision latency for this scheduler is $\log_2(n) + v$.

1. Addition of the VPID registers to register base blocks
2. Addition of VPID comparisons to decision blocks
3. Winner stream multiplexer at the output of the decision blocks
4. Additional priority update stages, one stage per VPID

3.4 Summary

This chapter has described the implementation of the ShareStreams-V system. The system consists of the hardware implemented on a PCI Express board with a Xilinx Virtex-4 FPGA, and the software on the host CPU. The dynamic spatial partitioning technique is enabled by adding support partitioning of the shuffle-exchange network, including the two main changes of using VPIDs in the hardware and using the buddy allocation algorithm in the software. The next chapter will describe the synthesis and testing results for the implementation described.

CHAPTER IV

EXPERIMENTAL RESULTS

The ShareStreams-V was implemented on a general purpose CPU with an FPGA daughter card connected on the PCI Express bus. This chapter discusses the results of the experiments conducted using the current implementation. Section 4.1 presents the validation of the hardware design. Section 4.2 introduces the target environment and results of the hardware synthesis. Section 4.3 presents the results of the software runtime, and the final section summarizes the results.

4.1 Validation

A C++ program has been developed that implements the DWCS algorithm and simulates the base hardware implementation as well as the virtualized ShareStreams-V scheduler. This implementation is used for generating inputs for (example stream states for test) the hardware design as well as for generating inputs for testing the FPGA implementation.

4.2 Synthesis

ShareStreams-V has been developed and synthesized for the Xilinx Virtex-4 FX140 chip with speed grade -10, and the FF1517 footprint [2]. The Virtex-4 uses 90nm process technology and has 63,168 slices, 552 Block RAM's, and 192 Xilinx DSP48 slices.

Synplicity Synplify Pro 8.9 and Xilinx ISE 9.1 were used for synthesis, mapping, translation, and place and route [1,3]. Pipelining and best-effort timing were enabled in Synplify Pro, but retiming was disabled due to the size of the design. The results shown are post place-and-route.

For the first set of tests three architectures were synthesized: the unvirtualized design

(SS), the virtualized design with support for four VPIDs (SSV4), and the virtualized design with support for up to 32 VPIDs (SSV32). The winner multiplexers in the virtualized designs are a limiting factor in the 4 and 32 bit designs as they are on the critical path of the scheduler and also add additional logic to the design; testing both of these designs will help in examining the effect of the multiplexer on performance of a design.

The upper limit of virtualization for an n-stream scheduler is $\frac{n}{2}$, due to the fact that each virtual packet scheduler must have at least two streams. With just one stream, a packet scheduler is not necessary as all packets in the stream will be scheduled. Thus for a 64-stream packet scheduler, there can be support for up to 32 VPIDs.

SS and SSV4 architectures with 16, 32, 64, and 128 streams were synthesized, and SSV32 was synthesized with 64 and 128 streams. A 256-stream design, both with and without virtualization, was too large for the Virtex-4 FX140 chip.

The change in area between the unvirtualized (SS) and the virtualized (SSV4, SSV32) designs in terms of slices on the Virtex-4 ranged from between -1.1 percent and 6.6 percent for SSV4, and between 0.9 percent and 2.9 percent for SSV32. Though there is additional logic on the FPGA to store the additional registers and comparators for the VPIDs and for the output winner multiplexer, the change in area is small due to the variances in placement and routing as well as greater slice utilization. The Xilinx Virtex-4 slices contain two Look-Up-Tables (LUT's) each [2].

Figure 10 shows the maximum frequency achievable post Place-and-Route. The critical path is the bus to the RBBs for loading stream state and for the priority update cycle, and there is a dramatic drop-off of 20-30MHz between the 64-stream design and a 128-stream design due to additional routing required for the bus. This fundamentally is an issue of interconnect for writing to a large number of locations and future optimizations may address this problem.

The change in maximum frequency was between 4.4 percent and -2.7 percent for SSV4, and between -8.6 percent and -20.3 percent for SSV32. This dramatic decrease in the

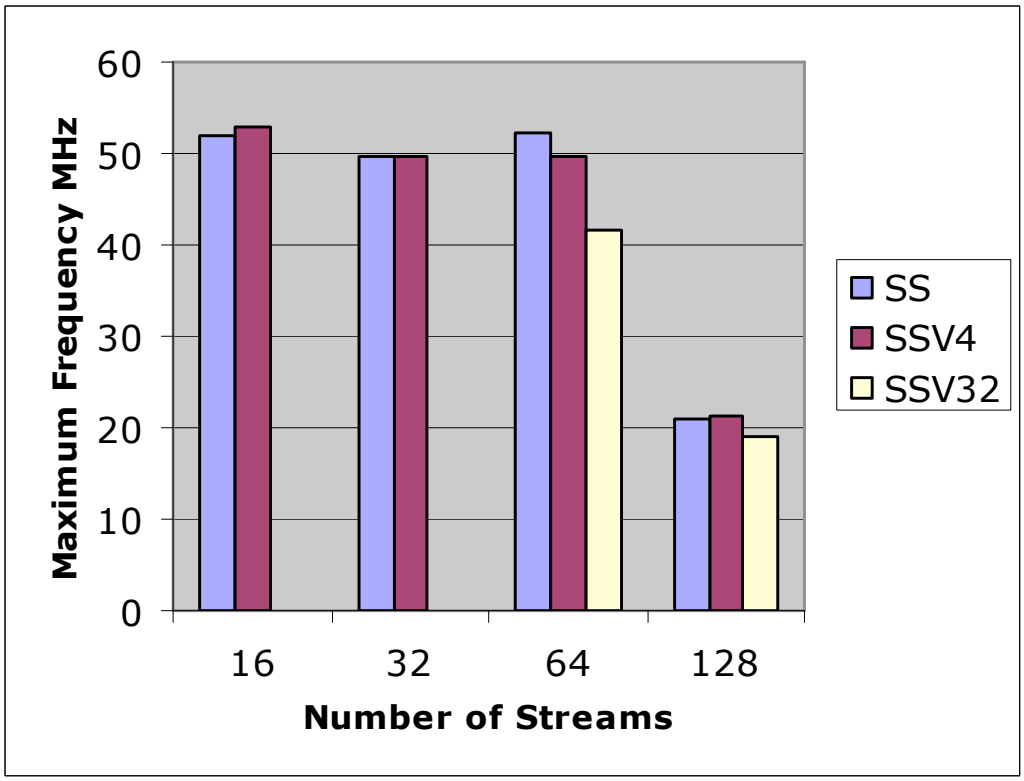

Figure 10: Clock Speed

maximum frequencies is due to the addition of a multiplexer in the virtualized designs. For example in the 32 VPID implementations, the 32-to-1 winner multiplexer is on the critical path, to support the 32-VPID priority update.

Figure 11 shows the decision latency for each VPID on a log scale. The latency is calculated by the formula $\log_2(n) + v$ from Table 4, where n is the number of streams and v is the degree of virtualization (i.e. 4 and 32 for the SSV4 and SSV32 designs, respectively). The latencies for all of the schedulers except for the 32-VPID 128-stream scheduler meet the requirement for 1500-byte packets for a 10Gbps Ethernet link, which is the top line shown in the graph. For each individual VPID none of the schedulers able to make the requirement for 64-byte packets (which is the bottom line). However, with virtualization, the latencies can be overlapped and a greater maximum throughput achieved.

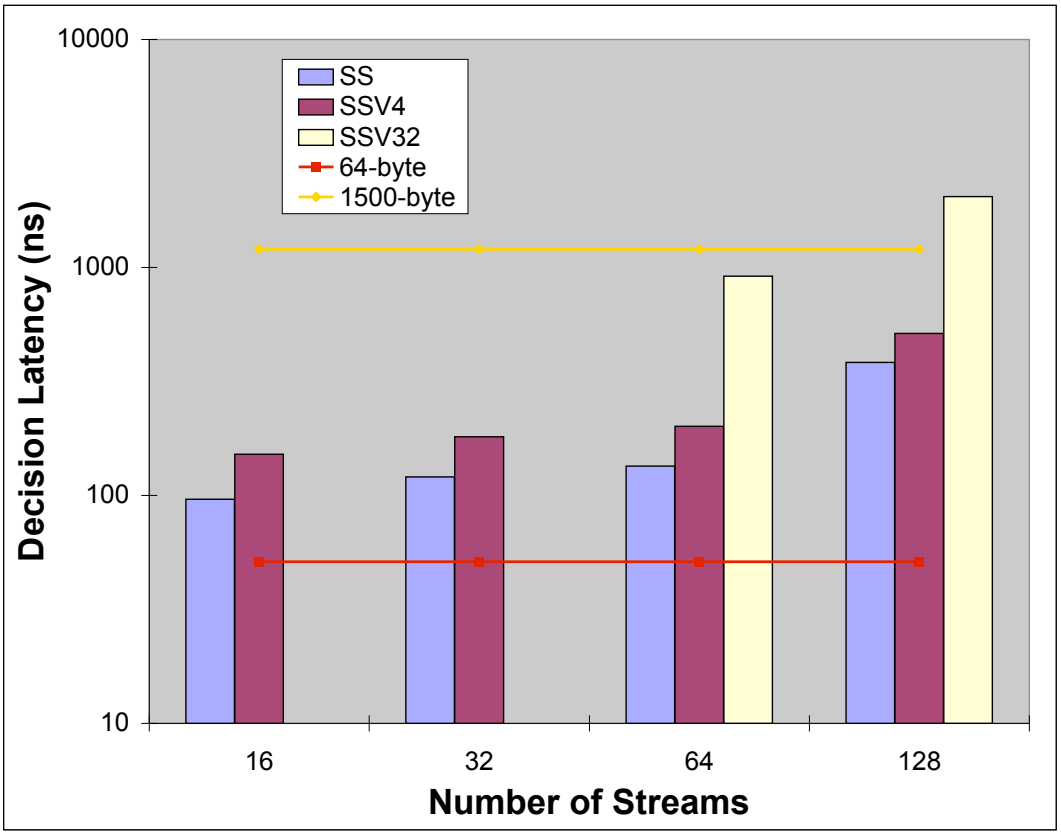

Figure 11: Decision Latency per VPID

The individual latencies are affected by the degree of virtualization v supported by the hardware (4 or 32 in the designs synthesized) and by the winner multiplexer size. One option to dynamically decrease the latency would be to only have Priority Update stages for the valid VPIDs. This would be implemented with additional control logic that would look at the valid signals for the "winner stream" for each VPID. As an example, a 32-input winner multiplexer which normally takes a fixed 32 cycles for the priority update stages can run the PU in only four cycles if there are only four VPIDs which are active in the scheduler. The disadvantage of this technique is that the scheduling latency varies based on the current number of VPIDs in the design, which requires the software to adjust how frequently it streams packet arrival times into and reads winner packets out of the scheduler.

Additionally, the size of the winner stream multiplexer affects the overall clock speed,

as is evidenced by the lower clock frequency for the SSV32 designs in Figure 10. As this multiplexer is on the critical path for the SSV32 designs, a smaller winner multiplexer such as a 4-to-1, 8-to-1, or 16-to-1 mux, can be used instead to increase the clock speed at the cost of a lower degree of virtualization v supported.

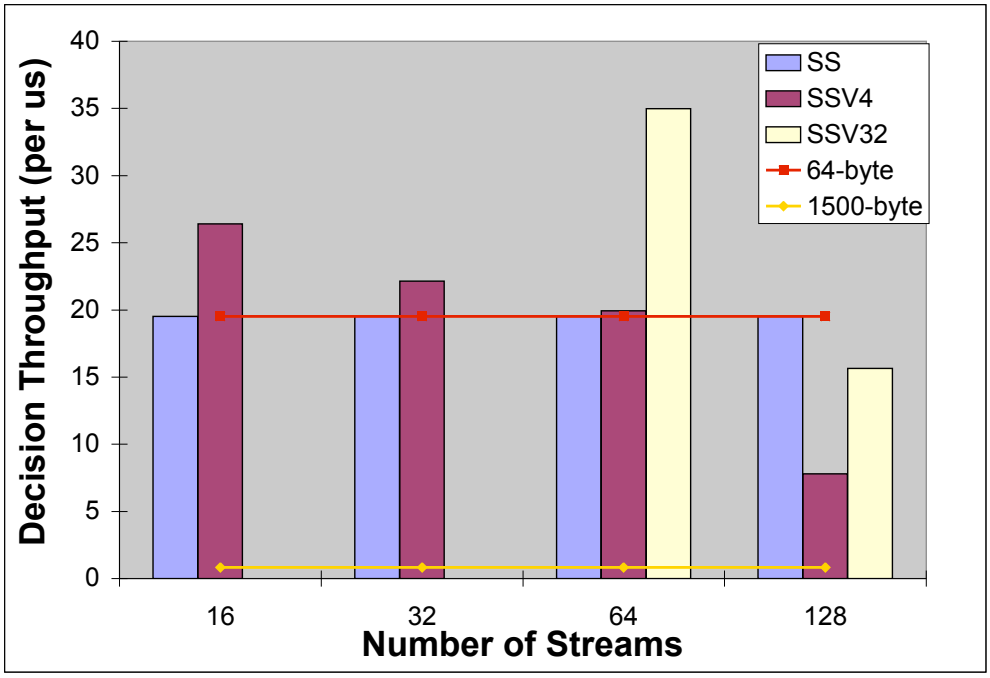

Figure 12: Maximum Decision Throughput

The maximum achievable throughput is seen for the ten architectures in Figure 12; the formula for the throughput is $\frac{v}{(\log_2(n)+v)}$, where n is the number of RBBs and v is the degree of virtualization, as can be seen from Table 4. Ethernet frames sizes are between 64-bytes and 1500-bytes, for which the required throughput for 10Gbps is shown by the two lines in the figure. Scheduling 64-byte Ethernet frames at 10Gbps wire-speeds (i.e. one decision per 51.2ns) can be achieved by the 16, 32, and 64-stream 4-VPID scheduler, and by the 64-stream 32-VPID scheduler. And for 1500-byte frames, each of the 10 configurations

synthesized can achieve 10gbps scheduling throughput (i.e. one decision per 1200ns).

The tradeoff between latency and throughput is seen in the latency formula $\log_2(n) + v$ and the throughput formula $\frac{v}{(\log_2(n)+v)}$ for this design. The potential increase in throughput is mitigated by the actual scalability of the design, as for 128-stream designs the clock speed falls more than 50 percent for all of the designs synthesized.

For the second set of tests, the design of the decision blocks in the 4-VPID architecture was changed to use the built-in Xilinx DSP48 slices, which include 18x18 multipliers, and also by decreasing the number of rules as shown in Table 2.4 (and thus the number of comparators in the DB) to just the top four to examine the effects on area and frequency.

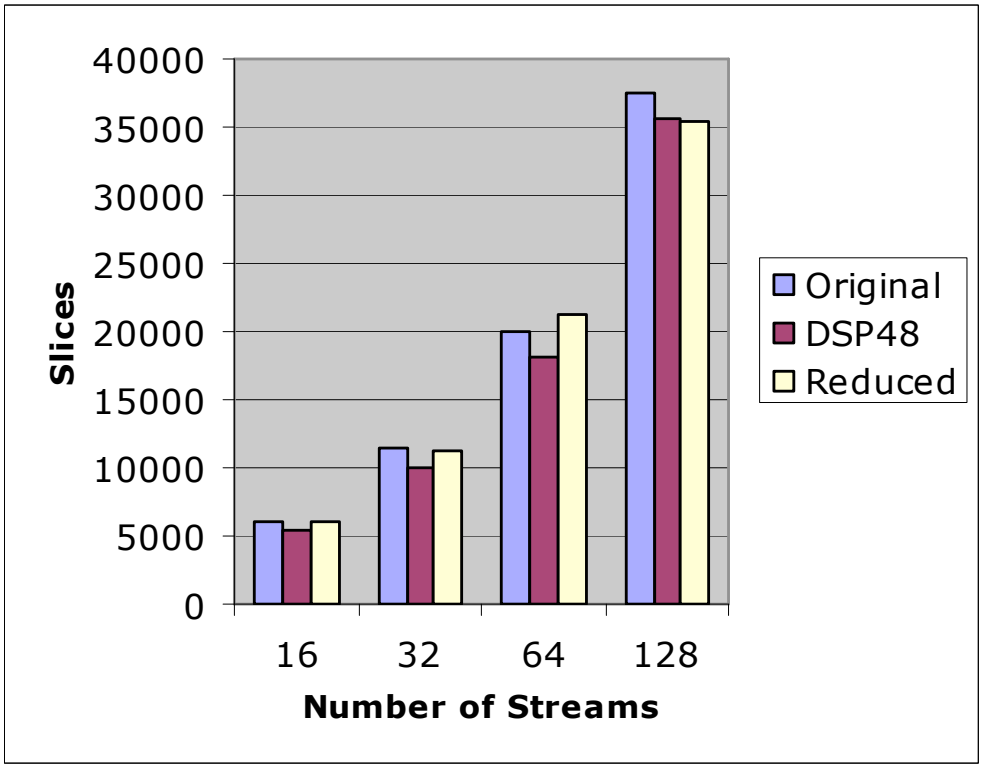

Figure 13: Logic Utilization with in Four-VPID Architecture

The designs with Xilinx DSP48 slices (which contain hard-coded multipliers) used 5.6-13.1 percent fewer of the general slices, as can be seen in Figure 13. However, they

use one DSP48 slice for each stream in the architecture. Since the Virtex-4 FX140 only contains 192 DSP48 slices, the maximum scheduler size may be limited by the number of these slices if the hard-coded multipliers are used. Figure 13 shows also that reduced-decision DB design occupied 0.7 to 6.4 percent fewer slices. These designs for 128-stream schedulers occupied 56 percent of the slices on the chip, so the architecture would require additional design changes or area optimizations to synthesize a 256-stream scheduler for this particular FPGA.

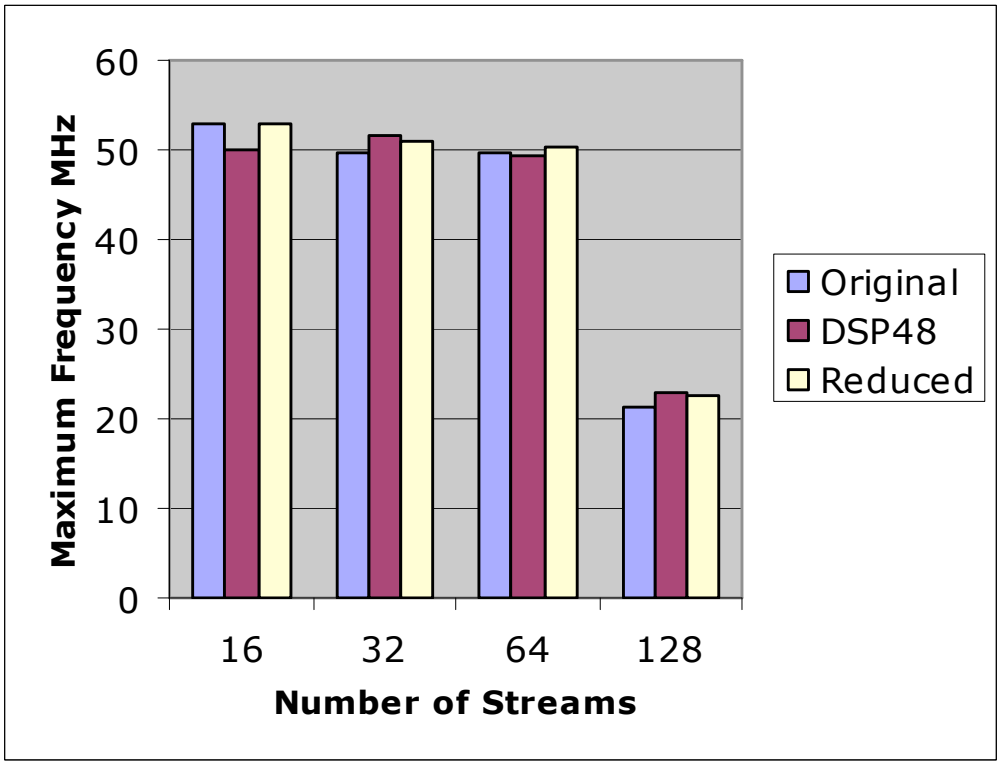

Figure 14: Clock Speed in Four-VPID Architecture

The clock speed improvement of the DSP48 designs was up to 5.6 percent, and up to 6.7 percent for the Reduced designs, as can be seen in Figure 14. This has a directly proportional effect on throughput, and an inversely proportional effect on latency.

To optimize the latency and also to save general logic for designs with 64 or 128

streams, using Xilinx multipliers is recommended. Using Xilinx multipliers the DWCS algorithm is maintained with a close speedup to and a larger area saving than the Reduced design.

4.3 Software Runtime

The ShareStreams-V software was compiled using gcc 4.1.2 and experiments are conducted on a dual Intel(R) Core(TM) 2 Quad processor, which has four cores at 2.66GHz. This platform running Fedora 7 as the operating system serves as the host CPU for the system. The software communicates to the hardware accelerator through a 4x PCI Express bus. A general software test was run to test performance of the current system.

The ctime functions *time* and $diffime$ were used to measure (based on the current system time) the approximate time of the core execution of the scheduler. This was combined with a count of the number of winners that could be read from the scheduler to determine the decision throughput.

The software runtime results had much lower performance than the hardware results. For the system described, the highest throughput for a SS128 FPGA design was $\frac{6000000 decisions}{30 seconds}$, or one decision per 5ms. This does not even meet the throughput requirements for maximum-size 1500-byte packets on 10Gbps, or one decision per 1.2ms.

The overhead is not just in terms of bus latency, but also on the ShareStreams-V software runtime as well as the multithreaded OS. The software tests were run on the Linux-based Fedora 7 OS, so there is additional overhead from the OS process scheduling. The results for the hardware are the upper limit on performance, and the entire system must take into account the overhead from bus bandwidth and latency, as well as the software overhead.

To achieve higher overall system performance for the virtualized scheduler, part or all of the virtualization management and control should be tightly coupled through a low-latency bus to the FPGA. This may take the form of a NIC with a network processor directly

connected to an on-board FPGA through a local bus, or through the use of one or both of the PowerPC 405 core(s) that is embedded on the Virtex-4 FX. The ShareStreams-V system requires a complex allocation algorithm, and software is better suited for handling both the allocation algorithm as well as management of virtualization (VPIDs and the virtual scheduler data structures to store sets of stream information).

4.4 Summary

Overall, the hardware synthesis results for ShareStreams-V show that the hypothesis is both feasible, and the ShareStreams-V hardware is able to perform scheduling decisions at 10Gbps wire rates. However, the bus and supporting software overhead in the tested system were high enough to cause the performance to drop significantly in the tested system. It is recommended in future implementations for the host CPU to be tightly coupled in a low-latency link to the packet scheduler hardware in order to minimize the overhead in terms of time.

CHAPTER V

CONCLUSIONS

This work has examined and demonstrated the hypothesis that virtualization of a quality of service packet scheduler accelerator based on the DWCS algorithm through dynamic spatial partitioning is a feasible approach to sharing the accelerator, and will produce one additional decision for an additional process at the cost of one additional cycle of latency. It has presented and compared the techniques of temporal and spatial partitioning. The Dynamic Spatial Partitioning virtualization technique was presented with the theoretical decision latency as in the hypothesis. The proposed solution permits fairly fine grained trade-off between throughput and per-stream decision latency as a function of the number of processes sharing the scheduler.

The hardware design was synthesized for a Xilinx Virtex-4 FPGA in the ShareStreams-V implementation and shown to be feasible. All of the designs synthesized and tested were able to meet 10Gbps Ethernet line scheduling throughput for 1500-byte packets, while only the 16, 32, and 64-stream 4-VPID schedulers, and the 64-stream 32-VPID scheduler were able to make wire speed decision throughput for 64-byte packets.

The software was also tested with an Intel chip as the host CPU for ShareStreams-V, and the FPGA on a PCI Express card. The results for this fell far short of the throughput requirements for 10Gbps wire speed scheduling. Thus, for future implementations it is recommended that part or most of the control functionality be implemented on a CPU that is either embedded in the FPGA or tightly coupled to the FPGA on the same board, such as in a NIC with the FPGA as a daughter card.

Future research may be conducted to test effect on performance of sharing between a larger number of processes. Dynamic spatial partitioning is a specific implementation

that is enabled with the shuffle-exchange network in ShareStreams-V. There is significant opportunity for further research in temporal and spatial partitioning virtualization methods for other packet scheduler architectures.

REFERENCES

[1] “"synplicity synplify pro software",” 2007.

[2] “"xilinx data sheet: Virtex-4 family overview",” September 2007.

[3] “"xilinx ise software",” 2007.

[4] Cardoso, J., “On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures,” *Computers, IEEE Transactions on*, vol. 52, no. 10, pp. 1362–1375, Oct. 2003.

[5] Figueiredo, R., Dinda, P., and Fortes, J., “Guest editors’ introduction: Resource virtualization renaissance,” *Computer*, vol. 38, no. 5, pp. 28–31, May 2005.

[6] Fornaciari, W. and Piuri, V., “General methodologies to virtualize fpgas in hw/sw systems,” *Circuits and Systems, 1998. Proceedings. 1998 Midwest Symposium on*, pp. 90–93, 9-12 Aug 1998.

[7] Goldstein, S., Schmit, H., Budiu, M., Cadambi, S., Moe, M., and Taylor, R., “Piperench: a reconfigurable architecture and compiler,” *Computer*, vol. 33, no. 4, pp. 70–77, Apr 2000.

[8] Guo, J., Yao, J., and Bhuyan, L., “An efficient packet scheduling algorithm in network processors,” *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, pp. 807–818 vol. 2, 13-17 March 2005.

[9] Knuth, D. E., *The Art of Computer Programming*, vol. 1 Fundamental Algorithms. Addison-Wesley Publishing Company, second ed., 1973.

[10] Krishnamurthy, R., Yalamanchili, S., Schwan, K., , and West, R., “Leveraging block decisions and aggregation in the sharestreams qos architecture,” *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pp. 10 pp.–, 22-26 April 2003.

[11] Krishnamurthy, R., Yalamanchili, S., Schwan, K., and West, R., “Sharestreams: a scalable architecture and hardware support for high-speed qos packet schedulers,” *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, pp. 115–124, 20-23 April 2004.

[12] McIlory, R. and Sventek, J., “Resource virtualisation of network routers,” *High Performance Switching and Routing, 2006 Workshop on*, pp. 6 pp.–, 7-9 June 2006.

[13] Plessl, C. and Platzner, M., "Virtualization of hardware – introduction and survey," in *Proc. 4th Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 63–69, CSREA Press, June 2004.

[14] Siegel, H. J., *Interconnection Networks for Large-Scale Parallel Processing*. McGraw-Hill Inc., second ed., 1990.

[15] West, R.; Poellabauer, C., "Analysis of a window-constrained scheduler for real-time and best-effort packet streams," *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, pp. 239–248, 2000.