Evaluation of the Tagged Up/Down Sorter Priority Queue

Michael Couglin
University of Colorado, Boulder

1 Introduction

In this paper, I present an evaluation of the Tagged up/-down sorter priority queue first presented by Moore, et. al. against other hardware priority queues for my final project for ECEN 5139 [**?**]. This evaluation is of the performance of this queue compared to the published results of other queues when implemented in an FPGA in terms of speed, queue size and resource utilization. This project was first motivated by the presentation of this particular priority queue during class and how it compares to other priority queues. Priority queues themselves are a classic example of a hardware task and as such, fall directly into the scope of computer-aided verification. Queuing is an important task that needs to be performed in many different applications, with a very frequent usage in networking, but is also used in other areas, including databases or even in some sorting applications [**?**].

The tagged up/down sorter queue that is the subject of this was first presented in 1995, but it appears to have not been used by the hardware community to any significant extent. However, the paper claims to achieve very good performance in both speed and resource utilization when implemented in hardware, so a comparison to the state-of-the-art solutions may reveal a potential new application of this queue, or a reason for why this solution is not used today. The primary works being used for comparison are the hybrid hardware-software priority queue presented by Kumar et. al. [**?**], the hybrid BRAM-based tree priority queue presented by Huang et. al., and a canonical priority queue implemented as a min-heap as a baseline. The first two of these implementations are from very recent publications, both published in 2014, and so can be considered examples of the state-of-the-art of this field.

An analysis of the tagged up/down sorter compared to the two state-of-the-art solutions shows a theoretical advantage in performance, as this queue is able to achieve enqueue/dequeue operations in a single FPGA cycle, whereas the state-of-the-art solutions only claim "nearly" one cycle performace (in the case of Huang et. al.) [**?**]. In addition, both of these solutions claim low resource utilization compared to other solutions when implemented in an FPGA, whereas the tagged up/down sorter was only tested in simulation. Therefore, an analysis of both the performance and utilization may yield insight into the potential trade-offs for using each solution.

In the remainder of this paper, I will present my evaluation methodology (Section 2), the results of the evaluation (Section 3), a discussion of these results (Section **??**), and finally, a conclusion (Section **??**).

2 Evaluation Methodology

In order to evaluate this queue against the performance of state-of-the-art designs, I required an implementation of the queue that can be programmed to an FPGA, a testbench to evaluate the performance of the queue and an FPGA device to target and use for the evaluation. For the implementation of the queue, I used Vivado High-level synthesis (HLS) to implement the queue using C++ and then have it synthesized to hardware, as well as a Verilog implementation provided by Professor Somenzi for a direct hardware implementation. I also used an HLS and Verilog implementation of a min-heap priority queue for the baseline. I chose to use HLS partly due to my familiarity with this particular tool and my lack of experience with Verilog, and partly because these algorithms are generally only expressed in software programming languages (there are, in fact, no easy-to-find implementations of hardware min-heap and few accessible priority queue implementations in general).

For the FPGA used for evaluation, I chose the Zedboard evaluation board, as this board was accessible to me from prior research. This board includes a Zynq7000 SoC, which incorporates an ARMv7 dual-core CPU and a 7000-series FPGA, allowing for a Linux OS to be able

to access and program the FPGA. Using this capability, I instantiated a testbench hardware module in the FPGA along with a particular queue implementation that runs the same test on each queue, referred to as the ”runner.” This module then returns the results of this test to a program running in the operating system as the number of enqueue operations performed. The software application in the OS accesses this module as a memory-mapped peripheral and records the time to execute the test, as well as the result returned by the module. The runner itself runs a test to enqueue as many elements as possible with incrementing and random priorities, and dequeue these items until the queue is empty, repeatedly for 10,000 iterations. The number of iterations is important, as there is significant overhead when interfacing with the runner module from software, so the FPGA needs to execute a time-consuming operation in order for it to be measurable. Due to time constraints, only a single queue size was used for each evaluation, as the queue size must be specified before synthesis and is time consuming to repeat.

Evaluation of the resource utilization of this queue is much more straight-forward, as utilization metrics are an output of synthesis. Therefore, each design only needs to be synthesized with different queue sizes to determine the utilization metrics. These output metrics are then compared to the published results of the state-of-the-art implementations.

3 Evaluation