M-Wide Systolic Sorter

With Thermometer Code Inputs

Portland State University

ECE 590 Spring 2015

Paul Long

paul@thelongs.ws

# Overview

This report documents the creation of a hardware systolic sorter written in SystemVerilog. The design takes as inputs M, N-bit signals (X0, X1, … Xm) and outputs M, N-bit signals (Y0, Y1, … Ym) sorted in increasing order where Y0 = minimum input value and Ym = maximum input value. A block diagram is depicted in Figure 1. Inputs and outputs are provided in thermometer code (explained below).

The sorter is expandable in the M dimension by virtue of it’s architecture, i.e. it is comprised of individual constituent cells each of which takes two inputs and produces two (sorted) outputs. These inputs/outputs are parameterized which provides expandability in the N dimension. Figure 2 shows the constituent cell block diagram.

Figure - Sorter Block Diagram

Figure - Min/Max Block Diagram

Figure - Min/Max Cell Internal Structure

# Thermometer Code

Thermometer code (sometimes called Unary code) is an encoding scheme that represents an unsigned integer number, *n*, with n ones zero padded to the desired vector width. For example, the decimal number 10 encoded in 16 bits is 0000\_0011\_1111\_1111. Compare this to the 16-bit binary representation: 0000\_0000\_0000\_1010. Thermometer code is considerably less dense than binary, especially as the values to be encoded grow large. The minimum number bits required to encode *n* in thermometer code is n; in binary the minimum is $\left⌊log2\left(n\right)\right⌋+1$. This size penalty is compensated for by the resulting simplification of the sorting logic. The Max function reduces to logical OR and the Min function reduces to logical AND. Consider the decimal numbers five and eight, represented respectively as 0001\_1111 and 1111\_1111 in thermometer code.

MIN = 0001\_1111 AND 1111\_1111 = 0001\_1111 = 510

MAX = 0001\_1111 OR 1111\_1111 = 1111\_1111 = 810

Assuming thermometer code operands, the min/max cell has the structure illustrated in Figure 3.

# Sorter Dataflow

What makes this design different from a traditional processor design is the importance of the constituent min/max cell topology—merely connecting the cells in a specific manner leverages the min/max functionality to provide sorting. Lets start with a simple example where M=3 and M=4. This results in a device that can sort three 4-bit numbers with the topology depicted in Figure 5. Figure 4 shows the labeling scheme for the individual cells and their interconnections. Each cell is fed from the bottom by the ColWire with the same indices as the cell itself and similarly on the right by the RowWire with the same indices. The Max/Min outputs are fed to the incremented ColWire and RowWire, respectively. This allows the cells and connecting wires to be programmatically generated at processor compile time in response to the M and N parameter settings.

Figure - Row/Column Labeling and Cell Latencies

Figure - Three-Digit Sorter Topology

The unlabeled squares in Figure 5 are registers which are used to ensure the two-dimensional pipeline remains properly synchronized. Cell[0,0] (the bottom-most cell) has a latency of 1. That means it’s Min/Max outputs will be valid 1 cycle after the inputs X[0] and X[1] are valid. Cell[1,0] has a latency of 2, indicating that it’s Min/Max outputs will be valid 2 cycles after X[0] and X[1] are valid and similarly for the remaining cells. Therefore, if X[3] is not delayed for one cycle with respect to X[0] it will arrive at Cell[1,0] too early and will not be compared to it’s data cohorts X[0] and X[1]; it will be mistakenly compared to the previous data cohort’s result from Cell[0,0]. These delays are referred to as “input delays” in the SystemVerilog source code. A similar delay is needed when the results from the left most cell of each row are passed up to the next higher row. These are referred to as “corner delays” in the source code. This synchronization is necessary at one more point in the design—the output—conveniently referred to in the source code as “output delays”. If this processor fed into another processor with similar input delay requirements, these output delays, and the input delays on the next processor, could be omitted. Fig depicts a cohort of data flowing through the processor. For clarity, only one set of data is shown flowing, wave-like, through the two-dimensional pipeline. One can easily imagine new data flowing immediately behind this initial wave and, indeed, this would be the most efficient use of the processor. This architecture produces an overall latency of 2M-3 cycles.

Figure - A Data Cohort Flowing Through the Sorter. Read the figure from left-to-right and top-to-bottom. The blue squares depict the data moving through the sorter in consecutive clock cycles.

# SystemVerilog Module Construction

The Min/Max cell is constructed as an individual module with the simple structure indicated in Figure 3. This module was unit tested before being included in the larger sorter design. In addition a parameterized shift register was created to be used as the input, corner and output delays as indicated in Figure 5. This module was also verified independently before it was incorporated into the larger design. With the two main modules tested and the interconnection naming scheme determined the only remaining difficulty was to construct the proper doubly-indexed looping structure that will produce the desired triangular array of Min/Max cells such that each row contains as many cells as the index of that row up to a total of M-1 rows. This indexing scheme is shown in the following code snippet:

// instantiate and wire up the cells

 **genvar** row**,** col**,** j**;**

 **generate**

 // instantiate the cells

 **for** **(**row**=**0**;** row **<** **(**M**-**1**);** row**++)** **begin**

 **for** **(**col**=**0**;** col **<** **(**row**+**1**);** col**++)** **begin**

 nBitThermMinMax Cell **(**

 **.**clk **(**Clk**),**

 **.**rst **(**Reset**),**

 **.**a **(**ColWire**[**row**][**col**]),**

 **.**b **(**RowWire**[**row**][**col**]),**

 **.**min **(**RowWire**[**row**][**col**+**1**]),**

 **.**max **(**ColWire**[**row**+**1**][**col**])**

 **);**

This is where the regularity of the triangle architecture really pays off; the majority of the work in constructing the sorter is done in that small chunk of code. However, as the code loops to generate each row, care must be taken to determine when we are in the first row, the first column, the corners and the last row as those regions require connections to the various delay pipelines. All Project source code is available at <https://github.com/pwlong/peaches>.

# Calculating Power Consumption

In order to estimate power consumption, the design was implemented in Xilinx Vivado with M=5 and N=4. The target FPGA was an Artix-7, xc7a100tcsg324-1. Vivado has a built-in power estimator that produced the dynamic and static power estimations shown in Figure 7. For these calculations, the clock speed was set to 100MHz.

Figure - Vivado Produce Power Estimations. Target FPGA is an Artix-7, xc7a100tcsg324-1

For use as an in-class demo to show the Min/Max circuit working and to show the procedure of downloading code onto an FPGA, a simple top-level module was created with M=2 and N=4. This module was targeted to the Nexys 4 FPGA development board from Digilent. Eight de-bounced slide switches were used to set the inputs and the outputs were displayed on two LEDs. If the left LED was lit, the left switches were said to be encoding the max, if the right LED was lit the right switches were said to be encoding the max. The Vivado project files used for this demonstration and the presentation power point slides are available at the Github repository mentioned earlier.

# Future Work

This design could be expanded, via the addition of a layer of control circuitry, to allow the constituent cells to produce any arbitrary function of two inputs and two outputs. The design could be altered to allow inputs to come from other directions and thus implement different functions. The cells could be modified to be multipurpose and derive the desired function from an opcode embedded in the data itself. The shape and systolic nature of the architecture open up innumerable possibilities for the creative thinker.