Lilah Kelly
Professor Traver
ECE 318
May 15, 2021

ECE 318 Paper: Q100 DPU

Those who work closely with computers tend to be familiar with central processing units, or CPUs. However, there tends to be an overall lessened awareness of data processing units. A data processing unit, or DPU, can be viewed as a helper to the CPU. This means that it typically handles tasks such as encryption, decompression, and compression using a "specialized design that combines processor cores with a collection of hardware accelerator blocks" (Robinson). Although many share these similarities, the tasks and architecture of the DPU can vary by distributor.

One such variation is the Q100 DPU. What sets the Q100 apart from other DPUs is that it is an energy-efficient way to process large quantities of data. It does so by maximizing pipeline and data parallelism, and minimizing the need to time multiplex the accelerator tiles and spill intermediate results to memory" (Wu). For this paper, pipeline and data parallelism can simply be thought of as concurrent statements processing the data as inputs. In terms of the time multiplex, it is a mux component that chooses the one piece of data to transmit using time as a select bit. To accomplish this, the Q100 uses a variety of ASIC tiles, also known as an "application specific integrated circuit...a microchip designed for a special application" (PeopleVine). These tiles include function tiles, which are boolean generators, aggregators, column filters, joiners, partitioners, sorters, and an ALU, as well as auxiliary tiles, which are table appenders, column selectors, column concatenators, and column stitchers. An example query and its corresponding spatial instruction plan, or flow of instructions, is included as it

shows how some of these listed tiles are used in the Q100 (Figures 1 and 2). This paper will touch upon what each function tile does and provide a brief explanation of each auxiliary tile.

In sequential order, the first tile to be used by the Q100 is the partitioner. This is because the Q100 is designed to process large quantities of data, and "as buffers and sorting networks are costly, this limits the number of items that can be sorted at once," it is best to partition it first (Wu). Partitioning means exactly what it sounds like; partitions are placed throughout a large table of data to divide it into smaller and more manageable sized tables of data. The particular type of partitioning used in the Q100 is range partitioning, "which splits the space of keys into contiguous ranges. We chose this because it is tolerant of irregular data distributions...and produces ordered partitions…" (Wu). In this statement, keys are the same as partitions.

Once the data is partitioned, the sorter is used. As the name suggests, a sorter is used to initially sort the input table of data using a designated key column and a sorting method called bitonic sorting (Wu). Bitonic sorting essentially means in a column of data, the individual elements of data are paired up with their immediate neighbors and swapped if they are not in order. After they are swapped, pairs are then made with every other element of data, then again with their neighbors. This continues, where the space between the two data elements increases by a power of two until there is no more space in the column or the column is fully sorted. In other words, if given the list of numbers [1 4 9 5 8 6 2 3] and the task is to sort them sequentially, using bitonic sorting will give an output of [1 4 5 9 6 8 2 3]. To continue, the list of numbers will next be [1 4 5 8 2 3 9 6], then [1 4 5 8 2 3 6 9]. Increasing the space between pairings again, we get [1 3 5 8 2 4 6 9], then [1 3 2 4 5 8 6 9], then [1 3 2 4 5 8 6 9], and so on

until we reach a fully sorted list. To give a visualization of the inputs and outputs of this tile, an example picture is provided (Figure 3).

After all the data has been partitioned and then sorted, it is then typically fed through the joiner. Again, as the name indicates, a joiner rejoins together the previously partitioned data. The specific technique used by the Q100 is called inner-equi join, which essentially means that only data that is equal to another will get joined. In other words, inner-equi join means to decrease the number of duplicates in the data after sorting.

Following the joiner, the ALU is used. ALU stands for the arithmetic logic unit, which can be thought of as a basic calculator. This means that ALU can make all basic arithmetic calculations such as adding, subtracting, dividing, multiplying, as well as the logic gates AND, NOT, and OR. In this course, several labs have been dedicated to developing multipliers and adders. These would be implemented into one datapath along with the other remaining functions to make up the ALU.

The next tile to be used is the boolean generator. This tile is what best resembles the digital designs seen in ECE 318 out of all of the components discussed thus far. The tile has two inputs, an enable pin and an operator pin, and a single output column (Figure 4). Essentially, this tile is a slightly more complex comparator that “compares an input column with either a constant or a second input column, producing a column of boolean values” (Wu). The output boolean column is a result of whichever operation is meant to be looked for. For example, if the operator pin was ‘==”, it would output 1 for each of the elements in the input column that was

equal to the second input element. This tile is used with another tile called a column filter, which “takes in a column of booleans (from a boolean generator) and a second data column. It outputs the same data column but dropping all rows where the corresponding bool is false” (Wu).

Finally, there is the aggregator. Aggregate, meaning a sum or total, is used similarly to the boolean generator tile, with two inputs, an operation pin, an enable pin, and a single output (Figure 5). This means that once again, the purpose and use are very similar to the comparator, except this time the operation options are maximum, minimum, count, sum, and average. The example given for the two inputs is “if the query sums purchases by zip code, the data column are the purchase totals while the group-by is the zipcode” (Wu).

Outside of the function tiles, there are also four auxiliary tiles, which are the column selector, column stitcher, column contatenator, and the table appender. To put it simply, the column selector selects and then “extracts a column from a table, and the column stitcher does the inverse” (Wu). For the remaining two, the column contatenator concatenates two inputs into one output, and the table appender appends two tables at a time.

All of these tiles were implemented using Verilog, which was touched upon at the beginning of this class. As a quick review, Verilog is another type of HDL that is largely based on the programming language of C (“ECE 318”). Overall, the Q100 DPU chip is much more complex than anything touched upon so far in ECE 318. It is important to note that once the chip is modularized into all of the various tiles such that each tile can be individually analyzed,

a deeper understanding of the Q100 DPU chip can be gained. In this way, this course serves as an essentially foundation for exploring much more advanced topics in the world of digital design.

APPENDIX

▷ *Sample query written in SQL*

```
SELECT    S_SEASON,
          SUM(S_QUANTITY) as SUM_QTY
FROM      SALES
WHERE     S_SHIPDATE <= '1998-12-01' - INTERVAL '90' DAY
GROUP BY  S_SEASON
ORDER BY  S_SEASON
```

▷ *Sample query plan converted to proposed DPU spatial instructions*

```
Col1          ← ColSelect(S_SEASON from SALES);
Col2          ← ColSelect(S_QUANTITY from SALES);
Col3          ← ColSelect(S_SHIPDATE from SALES);
Bool1         ← BoolGen(Col3, '1998-09-02', LTE);
Col4          ← ColFilter(Col1 using Bool1);
Col5          ← ColFilter(Col2 using Bool1);
Table1        ← Stitch(Col4, Col5);
Table2..Table5 ← Partition(Table1 using key column Col4);
Col6..7       ← ColSelect(Col4..5 from Table2);
Col8..9       ← ColSelect(Col4..5 from Table3);
Col10..11     ← ColSelect(Col4..5 from Table4);
Col12..13     ← ColSelect(Col4..5 from Table5);
Table6 ← Append(Aggregate(SUM Col7 from Table2 group by Col6),
                Aggregate(SUM Col9 from Table3 group by Col8));
Table7 ← Append(Aggregate(SUM Col11 from Table4 group by Col10),
                Aggregate(SUM Col13 from Table5 group by Col12));
FinalAns ← Append(Table6, Table7);
```

Figure 1: Sample Query

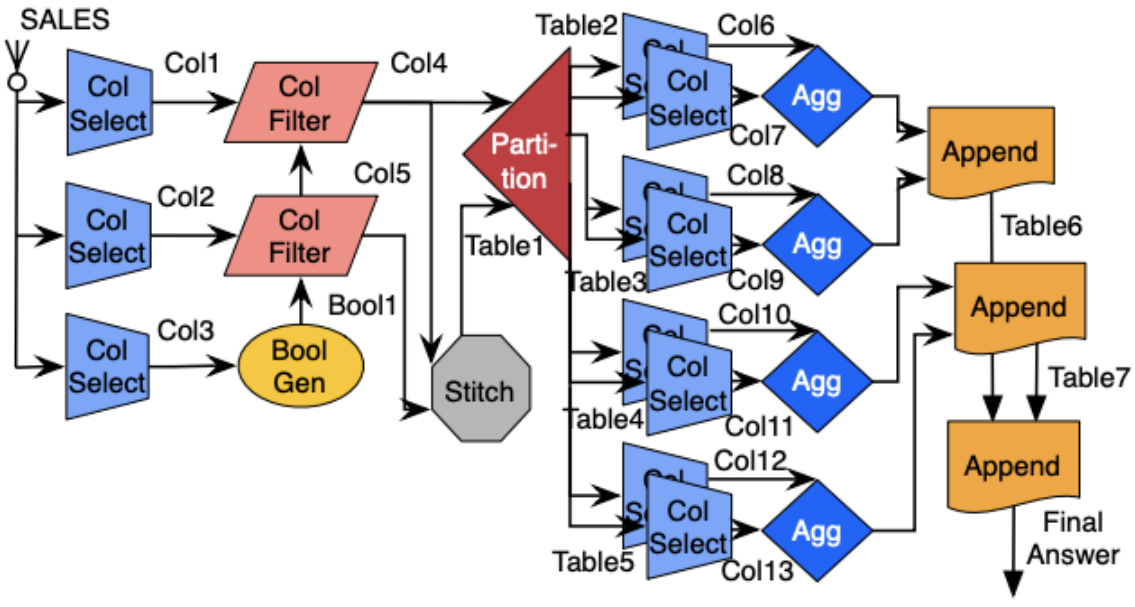

Figure 2: Corresponding Spatial Instructions

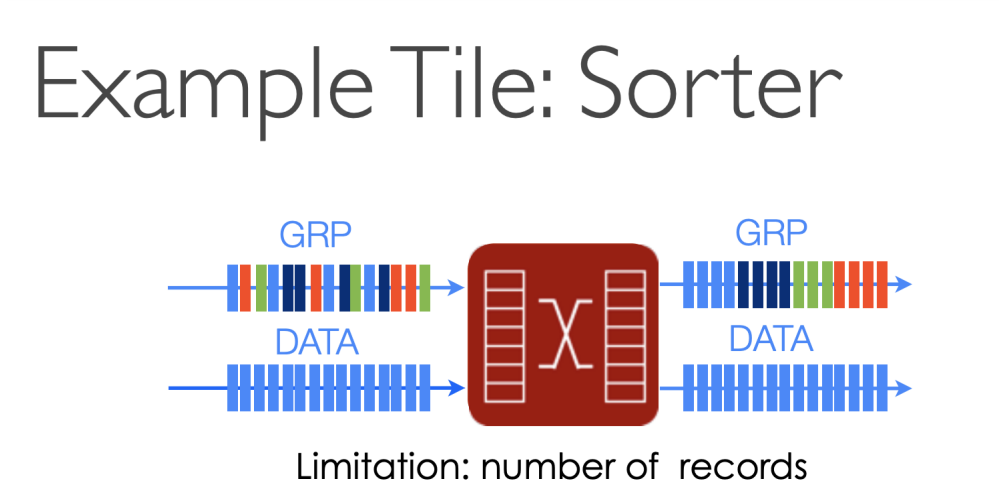

Figure 3: Example Tile for Sorter

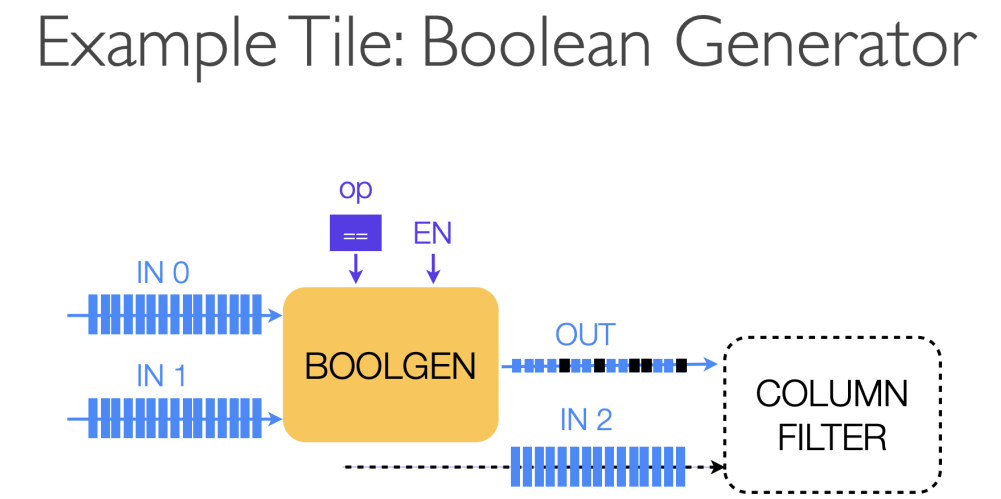

Figure 4: Example Tile for Boolean Generator

Figure 5: Example Tile for Aggregator

WORKS CITED

"ECE 318: Digital Design" zybooks. https://learn.zybooks.com/zybook/UNIONECE318TraverSpring2021

PeopleVine, Sigenics via. "What Is an ASIC, and Why Is Everyone Using Them?" SIGENICS, www.sigenics.com/blog/what-is-an-asic.

Robinson, Daniel. "Data Processing Units Accelerate Infrastructure Performance." SearchDataCenter, TechTarget, 13 Apr. 2021, searchdatacenter.techtarget.com/feature/Data-processing-units-accelerate-infrastructure-performance.

Wu, Lisa, et al. "Q100: The Architecture and Design of a Database Processing Unit." Columbia University, New York, NY, arcade.cs.columbia.edu/q100-asplos14.pdf.

Wu, Lisa, et al. Q100: The Architecture and Design of a DATABASE PROCESSING UNIT. Columbia University, New York, NY, arcade.cs.columbia.edu/q100-asplos14-slides.pdf.