2010 International Symposium on Nonlinear Theory and its Applications
NOLTA2010, Krakow, Poland, September 5-8, 2010

Mapping of High Performance Data-flow Graphs into Programmable Logic Devices

Csaba Nemes[†],Zoltán Nagy[‡],Miklós Ruszinkó[§], András Kiss[†] and Péter Szolgay[†] [‡]

†Faculty of Information Technology, Péter Pázmány Catholic University
Práter u 50/a, Budapest, Hungary
‡Cellular Sensory and Wave Computing Laboratory Computer and Automation Institute,
Hungarian Academy of Sciences, Lágymányosi u 11, Budapest, Hungary
§Applied Mathematics Research Laboratory Computer and Automation Institute,
Hungarian Academy of Sciences, Lágymányosi u 11, Budapest, Hungary
Email: nemcs@digitus.itk.ppke.hu, nagyz@sztaki.hu, ruszinko@sztaki.hu, kissa@digitus.itk.ppke.hu, szolgay@sztaki.hu

Abstract—In high-performance processors computation time and communication delay are comparable. Only those design methodologies can be successful which take care of the precedence of locality. In this article new design methodology is introduced which partitions the execution units and assigns a locally distributed control unit to each partition. Execution and control are relatively fast inside the partition and this results in a speed gain contrast to the global control unit where the fan out of the wiring can cause a slow operation. An optimization problem is described and an algorithm is developed which targets to find the optimal partitioning where fast local control units can be used with relatively small area increase. The optimal solution of the partitioning problem is NP complete [1] but a reasonable algorithm can be constructed for practical engineering applications. We have successfully designed a greedy algorithm and tested on few test cases.

1. Introduction

Having a computational problem defined on 2D or 3D array (NxM, NxMxL) and the operation on every element is described as a mathematical expression, acyclic data flow graph or UMF diagram [2]. The problem to be solved is how to map a computational problem on a virtual array to a given physical FPGA where area/processor (logic slices, DSP slices), on-chip memory (BRAM) and off-chip memory bandwidth are limited. Depending on the complexity of the operator a small amount of physical execution units can be implemented n << NxM (in 2D case) or NxMxL (in 3D case). The operator can be decomposed into small basic blocks which use either the logic resources (such as adders) or the dedicated resources (embedded multipliers) of the FPGA. The result of this process is a Physical Cellular Machine optimized for the given application. The optimization can be focused on speed, area, accuracy etc. Main components are the on-chip memory and the specialized execution unit.

Using current high speed DDR2/3 SDRAM and SRAM memories data read and write operations can be carried out in consecutive bursts. Additionally the available memory bandwidth might be fluctuating, therefore the execution unit should be halted during the computation if no data available.

The simplest and most area efficient solution of this problem is to use one global control unit to monitor the state of the I/O buffers and enable the operation of the entire system by using a global enable signal. The global enable signal has very high fan-out and is hard to route even if global wires are available on FPGA. As wire delay dominates over gate (LUT) delay on the current state-of-the-art FPGAs this solution results in very low operating frequency.

One possible solution of this problem is to create a data driven pipeline where a basic processing unit is halted automatically when no input data is available or the results cannot be processed by the next unit. Therefore local control unit can be added to every operator (adder, multiplier). In this case the control units are the simplest, but area requirements are significantly increased by the large number of FIFOs.

Alternative solution is to share the control unit among several basic processing units thus the FIFO buffers inside the groups can be eliminated significantly reducing area requirements. Determining the parameters of the groups carefully significant loss in operating frequency can be avoided.

2. Resources on an FPGA

The main configurable element of the new Xilinx Virtex family[3] is the Advanced Silicon Modular Block (ASMBL). The architecture is column based where each ASMBL column has specific capabilities, such as logic, memory, I/O, DSP, hard IP and mixed signal. By using different mix of the ASMBL columns domain specific devices can be manufactured. Currently four families are available optimized for different application areas: logic intensive (LX), logic intensive with serial transceiver (LXT), high performance DPS with serial transceiver (SXT), and embedded processing (FXT). Due to the smaller transistor dimensions the total net delay is mainly determined by the wire delay, hence the CLBs of the Virtex-5 architecture are completely redesigned. In the new architecture traditional 4-input LUTs are replaced by 6-input LUTs. Each CLB is divided into two slices and every slice contains 4 6-input LUTs, 4 registers, and carry logic.

In the new FPGAs the simple multipliers are replaced by complex DSP blocks called XtremeDSP (DSP48E) slices. The heart of the DSP48E is a 25bit by 18bit 2's complements signed multiplier. It also contains a 48bit ALU unit with optional registered accumulation feedback. Additionally, hardwired 17 bit shift capability simplifies the construction of large multipliers, while optional pipeline registers enable even 550MHz operation. The currently available largest Virtex-5 device contains 1056 DSP48E slices, while the largest member of the recently introduced Virtex-6 family contains 2016 DSP48E slices.

3. SystemC and Mathematical representation

Using a high-level description language the computationally intensive algorithm can be efficiently described by using the data-flow model. This model can be transformed to an abstract mathematical graph representation, where operations and connections are represented by nodes and arcs of the graph respectively.

In case of distributed control units data driven pipelines are created where basic processing units are halted automatically when no input data is available or the results cannot be processed by the next unit. Synchronization of the processing elements is done by using FIFO buffers.

If we assign one control unit to every processing unit and attach a FIFO to each of its output we obtain a restricted case of the Kahn process networks [4] where independent processes are communicating over bounded FIFO channels. Each processing unit can be treated as a process because it reads its input from a FIFO attached to previous processor or the memory interface and writes the updated result into a FIFO. The enable signal and the FIFO control signals are local to the given processor; their state depends on the state of the connected FIFOs.

However it is more practical to partition the processing units into groups where each group has one control unit. In this case FIFOs inside the locally controlled groups can be omitted. The disadvantage of this solution is that each control unit should handle more FIFOs, which results in more complex control logic and smaller operating frequency as shown in Figure 1.

There is a design trade of between the speed of the control unit and the area requirements of the circuit. Larger partitions have more inputs and outputs which results in a slower control unit, on the other hand small partitions increase the area requirements of the circuit because of the overhead generated by the larger quantity of FIFOs. We can formulate an optimization problem where we would

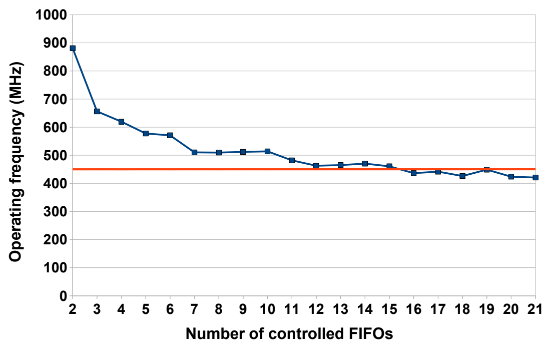

Figure 1: Operating frequency of the control block. Red line indicates the operating frequency of the multiplier unit.

like to enlarge the size of the locally controlled groups as long as the operating frequency of our control blocks does not limit the operation of the entire circuit.

Mathematical formulation of the partitioning problem:

1. Optimization: Make partitions from the nodes of a directed acyclic hyper graph where the number of cut edges are minimal. (In this case a hyperlink can be cut many times.)

2. Constraint: Any given partition shall have less than or equal connection to the other partitions than a given upper threshold.

Finding the optimal solution is an NP-complete problem, however our goal is to find a reasonable solution in polynomial time. The result of the algorithm will be an optimized graph, where the control is local and data driven. According to our experiments one control unit can handle 10 input/output FIFOs without decreasing the expected 450MHz operating frequency of the entire data path significantly (see Figure 1).

In our implementation the input of the algorithm is an initial solution of a computationally intensive task implemented in C++ via the SystemC library [5]. The only restriction for the implementation is that the classes of the modules have to inherit our interfaces as well. This is an elegant way to extend the SystemC model with some extra information without the modification of the library itself. While the modeling features of the SystemC library are still available, the extra information is used to build up the graph representation via the Lemon Graph Library [6].

4. The proposed partitioning algorithm

Hereby we propose a heuristic greedy algorithm for the previously described partitioning problem. While the pseudo code of the algorithm is shown in Algorithm 1. the key steps are also summarized below:

Algorithm 1

```
function runPartitioning(G,T):
1: Assign a number to each node, which indicates the number of cycles required for the data to reach the given node through the pipeline.
2: Sort the nodes based on the assigned numbers.
3: repeat
4:     Create a new partition P.
5:     Move the lowest-numbered node, which is unpartitioned to P.
6:     growSubgraph(P,T)
7: until there is any unpartitioned node
function growSubgraph(P,T):
1: for all $N$ nodes which gets input from one of the nodes of $P$ do
2:     P1 := P
3:     Move N to partition P1.
4:     Move all ancestor nodes of N which is unpartitioned to P1.
5: end for
6: if the number of the incoming and outgoing arcs of the subgraph $\leq T$ then
7:     P := P1
8:     growSubgraph(P,T)
9:     break
10: end if
```

1. The input of the algorithm is a directed acyclic hypergraph.

2. Based on the delays of the arithmetic units different levels can be assigned to the nodes. These levels indicate the number of cycles required for the data to reach the given node through the pipeline.

3. Based on the levels the nodes are numbered. The order of two nodes on the same level are arbitrary but nodes on lower level always have lower numbers than the ones on higher levels.

4. The algorithm starts from the lowest-numbered node and a subgraph is grown from this node. After the subgraph cannot be grown further the first partition is created from the nodes of the subgraph.

5. In every upcoming iteration the lowest-numbered node id selected which has not been partitioned yet. If there are no more unpartitioned nodes the algorithm ends.

6. The subgraph is also created iteratively. In the first iteration the subgraph contains only one node. In every upcoming iteration the algorithm tries to deepen the subgraph by selecting one node "below" the subgraph and includes all the ancestors of the selected node which have not been partitioned yet.

```
...
Prod<INSIZE,INSIZE,OUTSIZE> p1,p2;
Sum<INSIZE,INSIZE,OUTSIZE> s1;
...
p1− >c(signal1);
p2− >c(signal2);
s1− >a(signal1);
s1− >b(signal2);
...
```

Figure 2: Simple SystemC code snippet

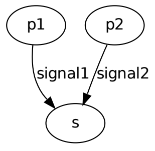

Figure 3: Simple data-flow graph generated from Figure 2.

7. Extension of the subgraph is only successful if the number of incoming and outgoing arcs of the subgraph does not exceed the upper threshold.

8. If no node can be selected below the subgraph which is suitable to a successful extension the subgraph cannot be extended and the algorithm is continued from step 4.

5. Examples

5.1. Simple SystemC code and its graph representation

Simple SystemC code fragment and an equivalent graph representation is shown in Figure 2 and Figure 3 respectively. Sum and Prod modules are base modules which are already implemented in SystemC. Instances of these modules will be the nodes of the graph. Signal1 and signal2 are wires corresponding to the interconnections.

5.2. Base template operation of the CNN state equation

Our first test case was the frequently used 3x3 template operation of the CNN state equation [7]. It can be regarded as a convolution with a 3x3 kernel where 9 state values should be multiplied by 9 template parameters. The final result is computed by summing the partial results and the old state value. However this example is relatively simple and has a limited size, it is ideal for the demonstration of our algorithm. The result of the algorithm is shown in Figure 4. The first partition (partition#1) is grown from the top-left node of the graph (node 1). The algorithm extends the partition by stepping one level down and recursively selecting all the ancestors of the newly founded node in every iteration, therefore nodes 2 and 12 are added to the partition in the first iteration. Partitions are extended as long as the number of input/output arcs are still smaller then 10,

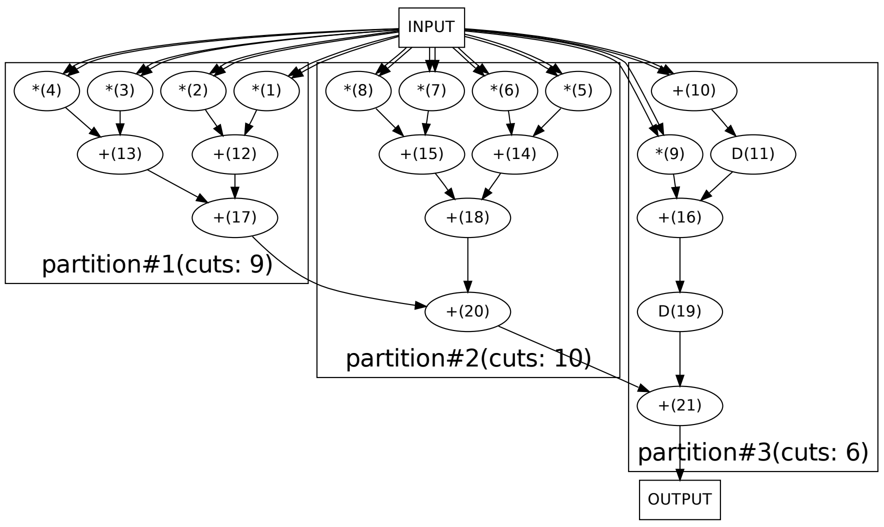

Figure 4: Result of the partitioning algorithm on the graph of the CNN template operator. The total number of cut arcs is 23.

which was determined earlier as an upper threshold. In this example the first two extensions are successful. The third extension failed because the extended partition would have 17 input/output arcs. The second partition (partition#2) is grown from the next non-partitioned node (node 5) which has the lowest number.

Area requirements and operating frequency of the optimized control unit are shown on Table 1. The optimized control unit requires less additional area than fully distributed control unit but operates on higher frequency than global control unit.

6. Conclusions

A design methodology is described to implement customized high-performance data-flow architectures using distributed control unit. A data-flow graph of a mathematical expression constructed from a high-level language description is given. An optimization problem has been described to efficiently partition the execution units between control structures without significantly increasing the area of the circuit or decreasing the operating frequency. A heuristic algorithm has been proposed to give an affordable solution. The number and the size of the FIFOs are optimized in the design process to reach high speed with small increase in implementation area. The operation of the algorithm was presented by optimizing a simple CNN state equation example.

Table 1: Implementation results of the control unit

		Fully distributed control unit	Global control unit	Optimized control unit
FIFO Area (slice)	Input/output	549	549	549
	Inside	0	855	98
	All	549	1404	647
Clock frequency (MHz)		423,908	881,057	512,032

Acknowledgments

This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

[1] *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[2] T. Roska, “An overview on emerging spatial wave logic for spatial-temporal events via cellular wave computers on flows and patterns,” *Proc. of NOLTA 2008*, pp. 98–100, 2008.

[3] “Xilinx product homepage,” http://www.xilinx.com, 2010.

[4] T. M. Parks, “Bounded Scheduling of Process Networks,” Ph.D. dissertation, University of California at Berkeley, 1995.

[5] “The open systemc initiative,” http://www.systemc.org/, 2010.

[6] “Lemon graph library,” http://lemon.cs.elte.hu/trac/lemon, 2010.

[7] P. Nagy, Z. Szolgay, “Configurable multilayer cnn-um emulator on fpga,” *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, pp. 774 – 778, 2003.