An Overlay Architecture for FPGA-based Industrial Control Systems Designed with Functional Block Diagrams

Taisei Segawa[1], Yuichiro Shibata[1], Yudai Shirakura[1], Kenichi Morimoto[1], Hidenori Maruta [1], Fujio Kurokawa[1], Masaharu Tanaka[2], and Masanori Nobe[3]

[1]Nagasaki University 1-14, Bunkyo-machi, Nagasaki,852-8521 Japan
[2]Mitsubishi Heavy Industries, LTD. 2-16-5 Konan, Minato-ku, Tokyo 108-8225 Japan
[3]Mitsubishi Hitachi Power Systems, LTD. 3-3-1 Minatomirai, Nishi-ku, Yokohama, 220-8401 Japan

Abstract— This paper discusses FPGA implementation of industrial control logic described in a function block diagram (FBD) language. First, we evaluate an approach where FBD descriptions are directly translated to FPGA hardware using a high level synthesis technique. Second, aiming at improving resource utilization efficiency, we proposed an overlay architecture which helps resource sharing of the same arithmetic structure utilized in different control logic sheets. Evaluation results show that the proposed architecture can significantly reduce resource requirements per control logic sheet, at a cost of acceptable performance degradation.

I. INTRODUCTION

Recently, demands for highly responsive real-time OS, highly intelligent communication, and high performance control operations are increasing in control systems for industrial infrastructure such as thermal power generation plants. Although a standard solution to the demands is multicore CPUs, their complex architecture is not necessarily suited to the industrial control systems.

Generally, long-term utilization of the same CPU architecture is difficult, since product cycles of CPUs are shorter than those of plant equipments. When the CPU architecture installed in industrial control systems is exchanged with new architecture, often software applications and OS are also needed to be revised due to architectural changes such as the number of cores. Note that dedicated OS needs to be often used for industrial control systems. Regarding development environments for industrial control systems, function block diagram (FBD) languages are widely utilized. Usually, FBD descriptions are translated to standard programming languages such as C, and then executed sequentially on a CPU. However, this design and execution flow does not exploit any inherent parallelism in control logic.

As a solution to these problems, FPGAs are attracting attention. A logical design layer of FPGA circuits can be inherited and enables long-term utilization of the design, even when physical FPGA chips are exchanged. Also, parallelism included in FBD descriptions will be exploited on FPGAs, by using a high-level synthesis (HLS) technique.

Although research attempts that translate FBD descriptions to HDL descriptions using HLS have been carried out [1][2][3], it has not been well addressed how these designs should be implemented on FPGAs in terms of performance and resource utilization. In this paper, we discuss FPGA implementation of control logic described in an FBD language. We evaluate two approaches: (1) direct conversion from FBD to FPGA hardware using HLS and (2) introduction of an overlay architecture on the FPGA. While many overlay architectures have been proposed to speedup application mapping time on FPGAs [4][5][6][7], our main aim of introducing an overlay architecture is to improve efficiency of resource utilization.

This paper is organized as follows. Section II explains about FBD languages. Section III describes direct implementation of control logic described in FBD and shows evaluation results. Section IV presents a proposed overlay architecture. Section V explains how the proposed architecture is implemented. Section VI discusses evaluation results on performance and resource utilization of the proposed architecture. Finally, Section VII concludes the paper.

II. FBD (Function Block Diagram)

FBD is a graphical diagram, where instruction blocks (function blocks), which are reusable functional elements, and flows of data signals between function blocks are described. FBD languages are also widely utilized for program description of programmable logic controllers (PLCs). In this work, we use a sort of FBD language called IDOL, which was developed by Mitsubishi Hitachi Power Systems for their DIASYS Netmation control systems.

Typically, control logic for industrial plants consists of a

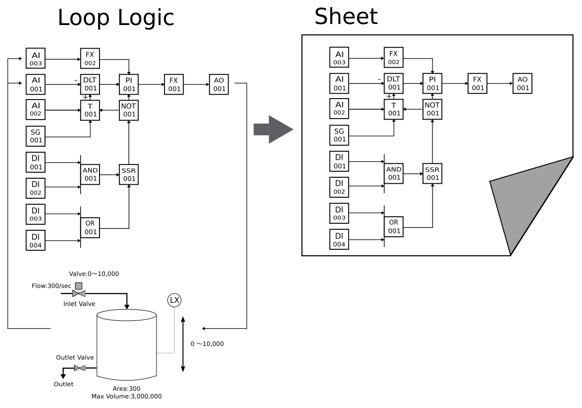
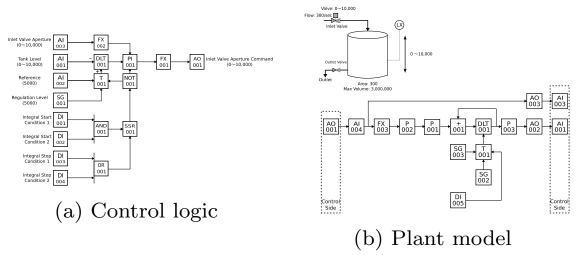
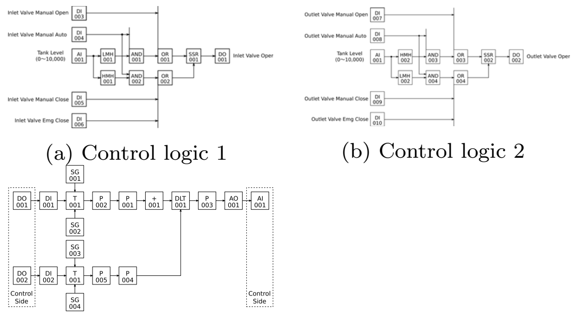


Fig. 1. Loop logic and a sheet



(a) Control logic (b) Plant model

Fig. 2. Control logic and plant model of PI control



(a) Control logic 1 (b) Control logic 2

(c) Plant model

Fig. 3. Control logic and plant model of PI control

number of feedback control loops. In IDOL, each feedback loop is called loop logic, and is basically drawn in one sheet as shown in Figure 1.

III. Direct Translation from FBD to FPGA by HLS

We implemented two simple control logic loops with HLS to evaluate performance and resource utilization of direct translation from FBD to FPGA hardware: proportional-integral (PI) control logic and on-off control logic for controlling a liquid level in a water tank.

Figure 2 (a) shows diagram of the evaluated PI control loop, and Figure 2 (b) shows an assumed plant model as a control target. Based on input data such as the current liquid level, the PI loop logic controls the valve aperture to keep the liquid level at the desired level. The AI and DI blocks shown in Figure 2 are used for data input, while the AO block is used for data output. The other blocks such as FX and PI are arithmetic blocks. In this evaluation, the target liquid level was set to 5,000.

As Figure 3 illustrates, the on-off control logic consists of two sheets of control loops, since its plant model has two control inputs, a water inlet valve and an outlet valve. Comparing the current liquid level to predefined threshold levels, each control loop decides whether to open or close the corresponding valve. In this evaluation, the inlet valve was controlled to be opened and closed when the level falls below 2,000 and exceeds 9,000, respectively. For the outlet valve, the two threshold levels were set to 3,000 and 8,000, respectively. The control cycle time of the systems was assumed to be 50 msec, which is a typical case in current industrial plants.

A. Implementation

In the DIASYS IDOL system, a script is prepared for each function block, which is eventually executed on a CPU. Figure 4 shows the script of the HMH block as an example. Although IDOL scripts are similar to C descriptions, IDOL has a dedicated grammar and has original variable types. Therefore, to absorb a grammatical gap between the IDOL scripts and HLS languages, we implemented a class library where IDOL original types and system functions are defined as C++ classes. By using this class library, IDOL scripts can be interpreted as standard C++ descriptions to be processed with HLS tools. In this evaluation, Xilinx Vivado-HLS 2015.2 was utilized for high-level synthesis. Single-precision floating point arithmetic was utilized for real number calculations. The control logic was mapped on a Xilinx Kintex-7 XC7K325T FPGA with a clock frequency constraint of 100 MHz. The control target models were also described in C++ and mapped on the same FPGA to emulate the plant behavior.

B. Evaluation

Figure 5 and Figure 6 show results of the emulation experiments. For the PI control, the tank level converged at the target level of 5,000. We confirmed the behavior of the control logic was appropriately changed by parameters of PI control. A 'k' and 't' in Figure 5 show a proportional gain and a time constant, respectively.

Also from the results of the on-off control, it is confirmed that the inlet and outlet valves were surely opened and closed according to the designed threshold levels. The average arithmetic execution time per control loop was 0.93 μsec for the PI control and 0.02 μsec for the on-off control, respectively. Considering a typical calculation cycle time for many plant equipments is a few msec, the

```
/*********************** HMH*/
void SCR_HMH(Ain X,Ain H,Ain D,Dout Y)
{
if(IS_MA_SET(Q(Y))) return;

if(D<0.0){
Y=1;
SET_CALC_ERROR(Q(Y));
return;
}

if (X>H) Y=1;
else if(X<=H-D) Y=0;
}
```

Fig. 4. Script of HMH block

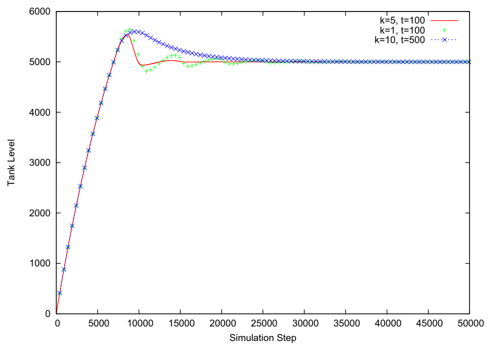


Fig. 5. Emulation result of the PI logic

control logic generated by HLS has a large performance advantage, more than three orders of magnitude.

Table I shows the amounts of FPGA resources consumed by each control logic. When attention is paid to the PI control, a total of 906 SLICEs, each of which consists of 4 look-up tables (LUTs) and flip-flops (FFs), were needed to implement this control logic. Since a Kintex-7 XC7K325T FPGA provides 50,950 SLICEs, this FPGA can implement 56 sheets of PI control loops on the chip. However, recent control systems for large-scale energy plants execute about 1,000 sheets of logic. Since even a high-end Virtex-7 XC7V200T FPGA, which offers 30,5400 SLICEs, can implement only 337 sheets, it is essential to improve area efficiently of the system. Considering that the calculation speed of the control logic is fast enough for the required performance, we propose an architecture where arithmetic hardware for multiple loop logics is not simply expanded in space, but hardware resources for the same function blocks used in different sheets are shared and reused.

IV. PROPOSED ARCHITECTURE

In this section, we explain a proposed architecture, which focuses on sharing of hardware resources.

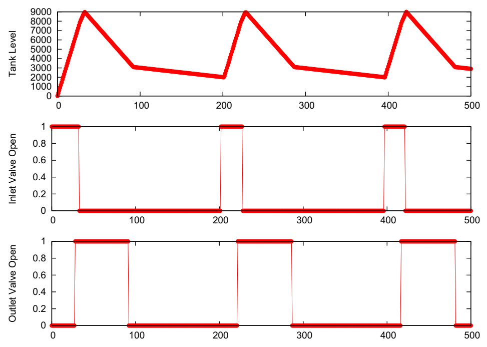


Fig. 6. Emulation results of the on-off logic

TABLE I
Resource utilization

Resource	PI Control		On Off Control	
	Utilization	Utilization Rate[%]	Utilization	Utilization Rate[%]
SLICE	906	1.78	104	0.204
LUT	3093	1.52	332	0.162
FF	1934	0.474	17	0.00417
DSP	12	1.42	0	0
BRAM	2	0.224	0	0

As shown Figure 7, processing elements (PEs) needed for desired control logic are placed to form an array of $(n \times m)$. The PEs in the same row and column are connected with horizontal buses and vertical buses, respectively. For simplicity of the layout, PEs of the same type of function block are placed in the same column. That is, n corresponds to how many different types of function blocks, and m corresponds to the maximum number of PEs of the same type of function block are utilized.

A PE that finishes its calculation transmits the calculation result to the PE that will use the result for the next operation. The data transmission is performed by the 2-D buses in the following manner. Here, we denote the horizontal axis as x-axis and the vertical axis as y-axis for explanation.

1. A PE which finished its calculation broadcasts the result to all the PEs on the same horizontal bus. The coordinate of the destination PE is appended in the header of the packet.

2. Each PE that receives the broadcast data checks the destination coordinate of the data. Only the PE that has the same x-coordinate with the destination retains the data, while the other PEs just discard the data.

3. The PE which retains the data broadcasts the data to all the PEs on the same vertical bus.

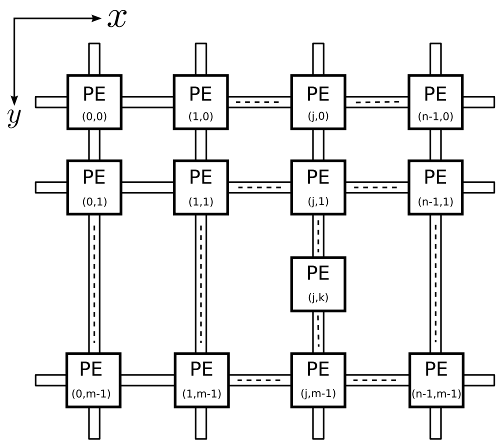


Fig. 7. Proposed architecture

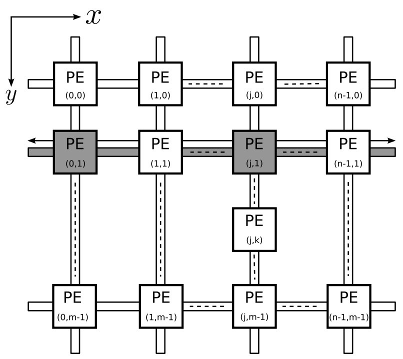


Fig. 8. Transmission to horizontal direction

4. Each PE that receives the data verifies the destination, and only the PE that has the same y-coordinate with the destination retains the data.

For example, when the PE (0,1) transmits data to the PE (j,k), first, the sender PE broadcasts the data on the same horizontal bus and the PE (j,1) retains data as shown in Figure 8. Next, the PE (j,1) sends the data on the vertical bus as shown in Figure 9, and lastly PE (j,k) retains the data. In this way, communication between any combinations of PEs can be completed in two-hop routing.

A. Execution Order Controller

All PEs are connected to an execution order controller, which controls the order of PE execution as shown Figure 10. The execution controller activates the PE to be executed next by sending the coordinate of the PE. The execution controller also sends the sheet number to be executed. The order of PE execution is stored in the table in the execution order controller, whose contents are written when an application is mapped on the architecture.

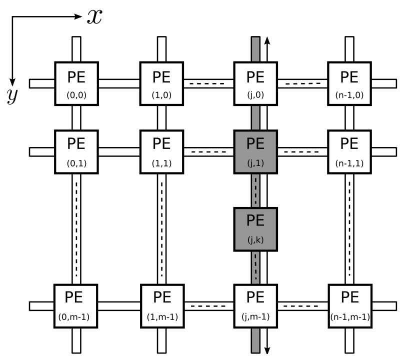


Fig. 9. Transmission to vertical direction

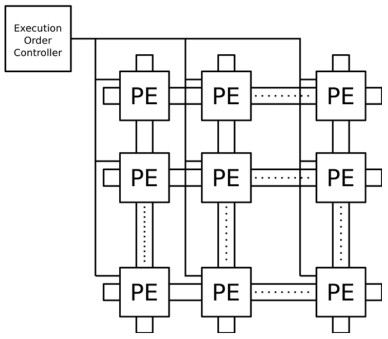


Fig. 10. Execution order controller

B. PE (Processing Element)

Figure 11 shows the structure of a PE in the proposed architecture. The PE consists of a router, a register file, and execution module. The router controls data transmission and bus interfaces. In concrete, the router checks the destination of transmitted data and decides whether it accepts or discards the data. When the execution module finish its calculation, the router appends the routing header to the result data and sends the packets. When a PE receives data, it is stored in the register file. The register file can hold data individually for each sheet and each input port, so that the functionality of the execution module is shared by different sheets. When a PE is activated by the execution controller, execution module reads required data from the register file and starts its calculation.

V. IMPLEMENTATION OF PROPOSED ARCHITECTURE

Figure 12 and Figure 13 illustrate how the control logic shown in Figure 2 and Figure 3 were implemented on the proposed overlay architecture, respectively.

To simplify the structure and routing logic in the architecture, we placed different types of PEs along the horizontal direction and the same types of PEs along the ver-

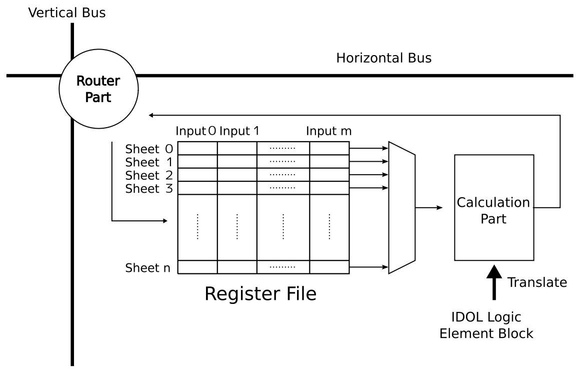


Fig. 11. Inside of the PE

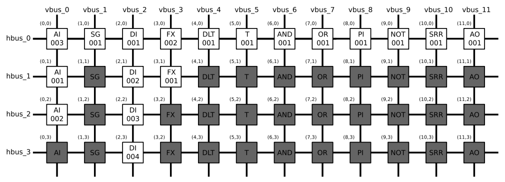


Fig. 12. Implementation of the PI logic on the overlay architecture

tical direction, forming a rectangular array. The number of PEs in the vertical direction, which was four in both examples in this evaluation, corresponds to the number of the PEs of the type with the most usage. Each number at the upper left corner of PEs shows the coordinate of the PE. The router module and the register file for each PE and the execution order controller were designed in a register transfer level with Verilog-HDL. RTL designs for execution module were translated from IDOL scripts using HLS. In this evaluation, the depth of the register file was set to 4096, so that 4096 sheets of control logic can be executed. As in the case of the implementation using only HLS, single-precision floating point arithmetic was utilized with the target clock frequency of 100 MHz. Finally, the overlay architecture was mapped on a Kintex-7 XC7K325T FPGA with Xilinx Vivado 2015.2.

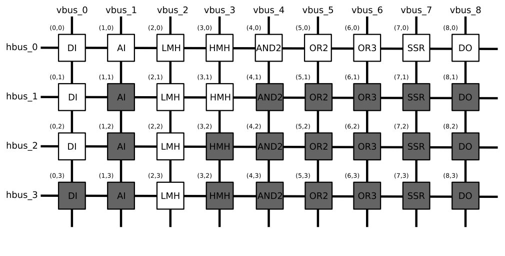
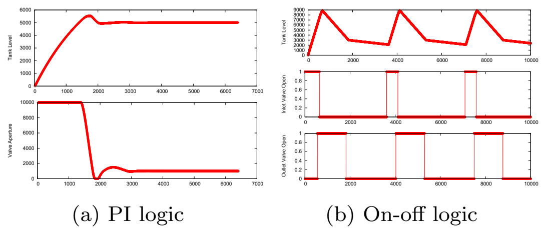


Fig. 13. Implementation of the on-off logic on the overlay architecture



(a) PI logic (b) On-off logic

Fig. 14. Emulation results of the proposed architecture

TABLE II
A Quantity of resource utilization of a proposed architecture

Resource	PI Control		On Off Control	
	Utilization	Utilization Rate[%]	Utilization	Utilization Rate[%]
SLICE	4126	8.01	1154	2.26
LUT	12292	1.76	3595	1.76
FF	10883	0.566	2308	0.566
DSP	50	5.95	16	1.90
BRAM	76	17.1	19	4.27

VI. Evaluation

Figure 14 show the emulation results of the control logic implemented on the proposed overlay architecture. For both examples, it was confirmed that the control logic worked appropriately to regulate the tank level.

Table II shows resource utilization for both control logic. Compared with the HLS implementation shown in Table I, BRAM utilization increased by 30 times and other resources increased by 4 to 5 times. This increase in the resources is mainly due to newly added mechanism such as the routers and register files in PEs. In addition, since we formed a rectangular array of PEs to simplify the architecture, some PEs were not utilized for the control logic. Also, unlike HLS implementations, the resource sharing between different PEs cannot be performed with this architecture.

However, the proposed architecture can process 4096 sheets of control logic, in contrast to the HLS implementation that can execute only one sheet. Table III shows comparison results of resource utilization for PI control per one sheet. It is shown that, utilization efficiency of BRAM was improved by about 100 times for BRAM. Also for other resources, improvement of approximately 800 times was achieved by introducing the overlay architecture.

To analyze the hardware overhead imposed by the additional mechanisms, we evaluated the breakdown of the resource usage, which is summarized in Table IV. This time, all BRAMs were utilized for the register files in PEs, since any function blocks in the evaluated control logic did not use BRAMs. For other resources, about 80 % were devoted to the arithmetic circuits. It is revealed that the

TABLE III
Comparison resource utilization for PI control per one sheet

Resource	Proposed Architecture	HLS Implementation	Utilization Efficiency Improvement
SLICE	1.01	906	897.0
LUT	3.00	3093	1031
FF	2.66	1934	727.1
DSP	0.0122	12	983.6
BRAM	0.0186	2	107.5

TABLE IV
Breakdown of resource utilization used overlay architecture

Resource	SLICE[%]	LUT[%]	FF[%]	DSP[%]	BRAM[%]
Calculation	79.8	90.9	93.0	100	0
Router · Register File	19.2	8.3	6.9	0	100
Execution Order Controller	1.0	0.8	0.1	0	0

resource overhead of the proposed overlay architecture is about 20 % in terms of SLICEs.

Next, we evaluated the execution performance of the control logic implemented on the proposed architecture. The calculation times per control loop (minimum control cycles) of the PI control and on-off control were 2.34 μsec and 1.66 μsec, respectively. The performance degradation compared to the HLS direct implementation was 3 times for the PI control and 83 times for the on-off control. The main reason is the data transfer on the two-dimension buses, which is required for every interval between PE executions. Especially in the on-off control logic, this overhead was significant, since the time required for arithmetic execution for this logic was originally quite small. Another cause is the proposed architecture executes PEs sequentially and does not extract parallelism from different PEs, while the HLS direct implementation spatially expands arithmetic hardware so that inter PE parallelism is fully exploited.

However, the proposed architecture achieved the control cycle of several micro seconds, which is still several orders of magnitude faster than the required performance in general industrial control systems. Therefore, the performance degradation observed by the proposed architecture is considered to be acceptable as a trade-off for the 800 times improvement in the resource utilization efficiency.

VII. Conclusion

In this paper, we proposed an overlay architecture focusing on sharing of hardware resources, in which the resources for the same function blocks that are utilized in different sheets of control logic. We compared the performance and resource utilization between the proposed architecture and HLS direct implementation using two examples of control logic.

As a result of evaluation, it was shown that the proposed architecture still achieved practical control performance while an overhead of data communication degraded the performance to some extent. On the other hand, the resource sharing mechanism of the proposed architecture improved the resource requirements per sheet by about 100 times for BRAM and by about 800 times for the other resources.

As future work, we will evaluate the architecture with more practical control logic sheets that are utilized in actual industrial plants. Although the proposed architecture does not exploit inter-PE parallelism at this moment, we will address improvement of the architecture so that logic sheets that have no dependency each other are executed in parallel.

References

[1] D. A. Lee, E. s. Kim, J. Yoo, J. S. Lee, and J. G. Choi. Fbdtoverilog 2.0: An automatic translation of fbd into verilog to develop fpga. In *Proc.2014 International Conference on Information Science Applications (ICISA)*, pages 1–4, May 2014.

[2] J. Yoo, J. G. Choi, Y. J. Lee, and J. S. Lee. A technique for demonstrating safety and correctness of program translators: Strategy and case study. In *Proc.Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 210–215, Nov 2014.

[3] J. Yoo, E. S. Kim, D. A. Lee, J. G. Choi, Y. J. Lee, and J. S. Lee. Nude 2.0: A model-based software development environment for the plc amp; fpga based digital systems in nuclear power plants. In *Proc.2014 International Symposium on Integrated Circuits (ISIC)*, pages 604–607, Dec 2014.

[4] A. K. Jain, S. A. Fahmy, and D. L. Maskell. Efficient overlay architecture based on dsp blocks. In *Proc.Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 25–28, May 2015.

[5] A. K. Jain, D. L. Maskell, and S. A. Fahmy. Throughput oriented fpga overlays using dsp blocks. In *Proc.2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1628–1633, March 2016.

[6] G. Stitt and J. Coole. Intermediate fabrics: Virtual architectures for near-instant fpga compilation. *IEEE Embedded Systems Letters*, 3(3):81–84, Sept 2011.

[7] D. Capalija and T. S. Abdelrahman. A high-performance overlay architecture for pipelined execution of data flow graphs. In *Proc.2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–8, Sept 2013.