FPGA Resource Estimation for Loop Controllers

Tom Degryse, Harald Devos and Dirk Stroobandt

Ghent University, Department of Electronics and Information Systems
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
{Tom.Degryse, Harald.Devos, Dirk.Stroobandt}@elis.UGent.be

Abstract

High-level synthesis overcomes the high design effort required by using an FPGA by moving the hardware design to a higher abstraction level. At this higher level, loop transformations are used to improve the characteristics of the program. These transformations have a large impact on the resulting hardware, but their impact is only known after the time-consuming synthesis steps. This hinders a fast design space exploration.

In this paper, we tackle this issue by estimating the performance of the hardware loop controller, an often overlooked component in other approaches. We present an equation based model to estimate the area and clock frequency of the loop controller during high-level synthesis. In our approach, we manage to keep estimation errors reasonably low, so our estimation model can be used during design space exploration. Due to its simplicity, the overhead is minimal, which is critical when lots of design variants need to be estimated.

1. Introduction

Many of today's embedded systems need a custom hardware accelerator to meet the stringent timing and/or power constraints. Field Programmable Gate Arrays or FPGAs have become increasingly popular as a hardware platform thanks to their lower price for small and medium volumes and their reconfigurability. Although the clock frequency on an FPGA is much lower than on a processor, many applications can be accelerated thanks to the high degree of parallelism available. Algorithms that have a high degree of regularity and high levels of parallelism can thus benefit the most from a hardware implementation.

However, exploiting the computational power offered by such devices is not straightforward. The translation from a sequential software representation of the algorithm into a hardware description language such as VHDL, is a time-consuming and error-prone process. The exploitation of the available parallelism is only one of the issues that the designer needs to tackle. When the design suffers from a bad data locality, the execution speed will be limited by the available memory bandwidth.

To shorten this design time, the hardware design process is moved to a higher abstraction level. The design is now specified at the algorithmic or behavioral level. A High-Level Synthesis (HLS) tool translates this high-level representation into a low-level synthesizable hardware description. In our HLS tool, called CLooGVHDL [4], we use the polyhedral model [6] as intermediate representation during hardware generation. The polyhedral program representation and the hardware generation from it, will be briefly tackled in Section 2.

During hardware generation, the HLS tool transforms the program to obtain an improved hardware implementation. These transformations can for instance introduce more parallelism or improve the data locality. The application of loop transformations is one well known optimization technique, both in software compilation and hardware synthesis. The polyhedral model offers a flexible program representation that allows to automate these loop transformations.

These loop transformations can have a great impact on the quality of the resulting hardware, but their impact is only known after completing all the time-consuming synthesis steps, which can take up to more than a day. This long synthesis time, combined with the large number of possible loop transformations, results in the fact that not every solution can be fully implemented during design space exploration. This can be overcome by estimating the characteristics of the final hardware implementation. These estimations can then be used to guide the designer or an automated design space exploration framework in selecting a good sequence of transformations.

In this work, we address the estimation of the loop controller directly from a polyhedral program representation. The loop controller is an often overlooked component in other a priori estimation techniques, such as [5, 9].

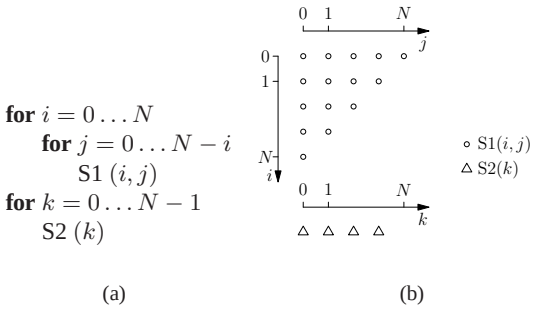


Figure 1. Program example (a) with the corresponding polyhedral representation (b).

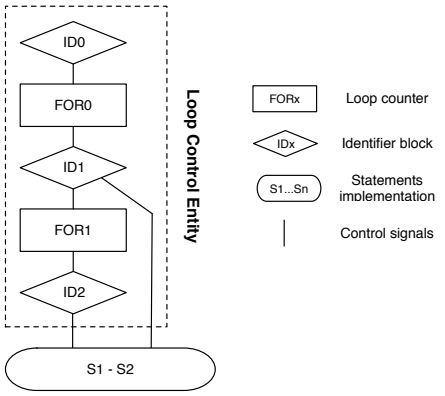


Figure 2. Hardware architecture of the loop controller of the program in Fig. 1. The block *FOR0* implements the i- and k-loops. Which of the two is selected depends on the value received from *ID0*.

These estimation techniques only estimate the data path. In Section 3 we discuss why the loop controller has a non-negligible impact on the complete design. We present a simple equation based model (Section 4) to estimate the number of logic elements needed to implement the loop controller. Despite its simplicity, our model is still accurate enough to be useful during design space exploration. Our experiments (Section 5) indicate that the relative error is not more than 8% on average. This is comparable with the results obtained in other papers (Section 6). We conclude this paper with some final thoughts and some indications for future work in Section 7.

2. Polyhedral model

Our work is based on loop transformations described in the polyhedral model [6]. Therefore, this section discusses the polyhedral program representation (Section 2.1), as well as loop transformations within this model (Section 2.2). The last part of this section (Section 2.3) explains how this high-level representation is translated into a hardware implementation.

2.1 Program representation

The polyhedral model presents a geometrical representation of the control inside a program. A single execution of a statement inside a loop or a loop nest is represented as a point in an n-dimensional space, with n the number of loops surrounding the statement. The set of points for which a statement is executed, defines a bounded polyhedron. Hence the name polyhedral model. These polyhedra are determined by the loop bounds and conditionals surrounding the statement. Figure 1 presents a small example and the corresponding polyhedral representation.

A polyhedron does not contain any information about the functionality implemented by a statement. The polyhedral model makes thus an abstraction of the statement implementations. As a result, there is a clear separation between the loop control and the data path.

2.2 Loop transformations

Previous work has shown that loop transformations can improve the characteristics of a program, e.g. an improved data locality or the introduction of parallelism. In the Data Transfer and Storage Exploration (DTSE) [3] methodology, loop transformations are applied to minimize the storage and bandwidth requirements. The Riverside Optimizing Compiler for Configurable Computing or ROCCC [7] uses loop transformations to enable more parallelism in the program amongst other optimizations.

Loop transformations change the execution order of the statement invocations in a loop nest. All loop transformations in the polyhedral model are performed by transforming the matrices that describe the iteration domains. Loop transformations are easily combined by combining the corresponding matrix operations. During loop transformations the statement definitions remain untouched. The only things that change are the execution order of the statements and their arguments.

2.3 Hardware generation

Generating code from a polyhedral representation of a program can be done with a tool called CLooG (Chunky Loop Generator) [2]. CLooG generates efficient code for scanning all the integral points of one or more parametrized

polyhedra. This is done by transforming the input polyhedra with an extended version of the Quilleré algorithm [14].

CLooG has been extended to generate VHDL code in a tool called CLooGVHDL [4]. The hardware architecture used in CLooGVHDL consists of two different entities: one with the statement implementations and one with the loop controller. At the moment, CLooGVHDL only supports a sequential execution of the statements. In this case, the statement implementations are not influenced by loop transformations, so this entity can be reused during design space exploration. In the case of parallel execution, the statement implementations itself do not change, there are only multiple statement instances placed on the FPGA. However, this has a minor impact on the structure of the loop controller. Hence, our estimation methodology presented in Section 4 can easily be adapted to support parallelism.

The controller is composed of several communicating automata. There are two different types of automata in the hardware architecture used in GLooGVHDL. The first type is the loop counter, which is responsible for the iterators. It calculates the loop bounds and the stride in function of the parameters and the iterators of the surrounding loops. The identifier blocks are the second type of automata. These blocks enumerate all statements and inner loops. Fig. 2 gives an overview of the hardware architecture of the example presented in Fig. 1. The loop counter *FOR0* implements the i- and k-loops. Which of the two is selected depends on the value of the identifier block *ID0*.

3. Loop controller impact

Modern FPGAs have lots of dedicated hardware blocks to facilitate the data path implementation. One can for instance think about the multipliers in some of the Xilinx FPGAs or Altera's DSP blocks. This makes that the data path can be efficiently implemented without using lots of logic elements. Opposed to this, the loop controller needs to be completely built out of custom logic elements. As a result, the loop controller consumes a major part of the logic elements needed by a hardware implementation.

To measure the impact of the loop controller on the area usage (Table 1), we synthesized the loop controller, the statements entity and the complete design with Xilinx ISE for several variants, resulting from different sequences of loop transformations, of the 2-Dimensional Inverse Discrete Wavelet Transformation (2D-IDWT). These variants range from the Row-Column (RC) based transformations to the more complex Line-Based (LB) variants, which have a better data locality. Several loop tiling strategies were explored in this experiment.

The area usage is expressed as the number of slices needed. A slice contains two logic cells, each having a 4-input LookUp Table (LUT) inside them. The statements entity remains the same for all design variants and consumes 1185 slices on a Xilinx Virtex II FPGA. The total number of slices is not always equal to the sum the loop controller and the statements, due to some optimizations across both entities. However, these only have a limited impact.

Table 1. Impact of the loop controller on the area usage of several 2D-IDWT designs.

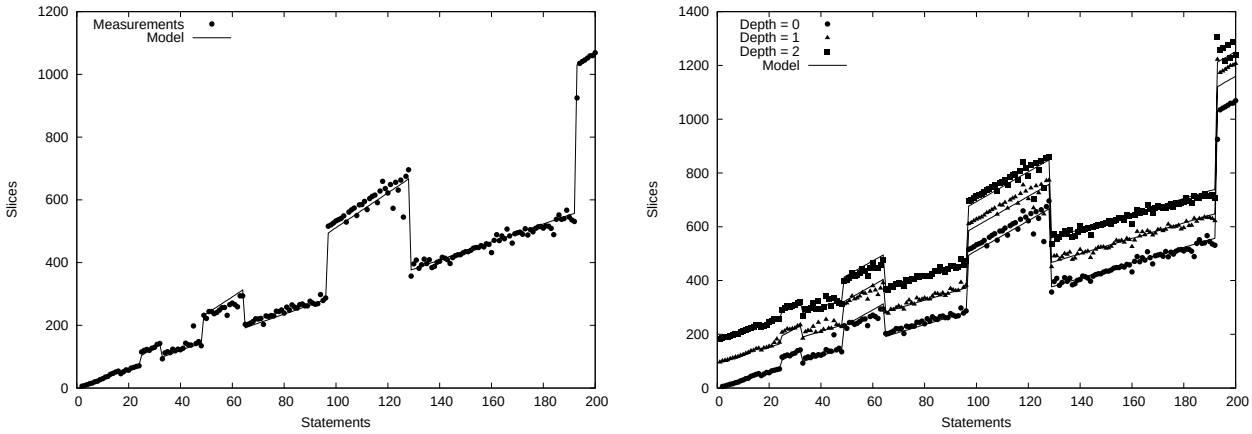
Design	Area (slices)		
	Controller	Total	%
RC	180	1326	13,57
RC tile X	992	2009	49,38
RC tile Y	731	1761	41,51
RC tile X + Y	1545	2520	61,31
RC tile Y + X	1913	2978	64,24
RC tile Y + X + LB	2863	3607	79,37
LB	527	1477	35,68
LB tile X	3499	4575	76,48
LB tile Y	1215	2034	59,73
LB tile Y + X	2686	3878	69,26

Table 1 shows that the loop controller consumes a substantial amount of resources, ranging from 13.57% of the total design cost to 79.37% in the more complex variants. We conclude that the loop controller has a non-negligible impact on the area utilization of a design made with CLooGVHDL. The same conclusion holds for the clock frequency. In case of the more simple design, like the RC variant, the clock frequency of the complete design is determined by the clock frequency of the statements, which is 71.582MHz. When the designs get more complex, and the clock frequency of the loop controller drops, the clock frequency of the full design drops almost evenly.

4. Estimation model

In this section we introduce our estimation model for both the area usage (Section 4.1) and clock frequency (Section 4.2) for Xilinx FPGAs. The designs are synthesized with Xilinx ISE 9.1.03i.

Our proposed estimation strategy uses empirical equations drawn from detailed observations of the synthesis results of a set of synthetically generated benchmarks. In the next step we fit the equations obtained from the synthetic benchmarks with a large number of loop controllers extracted from the SpecFP 2000 benchmarks provided with CLooG [1]. We try to minimize the average of the absolute value of the relative error. The reason why we do not use the Root Mean Square Error (RMSE) fitting strategy is that this favors the larger, more complex benchmarks too much.



(a) Area vs. statements

(b) Area vs. loop depth

Figure 3. Parameters affecting the controller area for Xilinx. The points in the figure indicate the measured area, while the full line represents the estimation model. Although the shape of the curve is irregular, we manage to estimate the number of slices reasonably accurate.

4.1 Area

This section discusses the two most relevant parameters (the number of statements and the maximum loop nest depth) affecting the area usage of the loop controller implemented on a Xilinx FPGA.

4.1.1 Number of statements

The first parameter affecting the loop controller FPGA resource usage is the total number of statements in the program. To investigate the impact of the number of statements on the number of slices needed, we generate a set of synthetic benchmarks. Each benchmark consists of a number of sequentially executed statements, without any loops or conditionals surrounding the statements.

The synthesis results from this experiment are presented in Fig. 3(a). These results are surprising: the area usage characteristic is not as regular as one could expect. Several regions with a different area usage characteristic can be identified in the curve. These regions can be divided into two categories ($i = 1, 2$ respectively), each modeled by an equation of the following form:

$$A_s = a_i \times x + b_i \times log_2(x) + c_i \ , \ i = 1, 2 \ , \qquad (1)$$

with x the number of statements in the program and a_i, b_i and c_i constants, A_s is the number of slices needed to implement the loop nest controller.

We were able to determine the bounds of those regions, so we could model the area usage characteristic fairly accurately, as indicated by the full line in Fig. 3(a). For instance, all sets in the lowest region of the curve ($i = 1$), L_n, can be expressed as follows:

$$L_n = \{x \in \mathbb{N} \mid x \in \ [2^n + 1, 2^n + 2^{n-1}]\}, n \in \mathbb{N} \ .$$

For instance, all the points from 17 to 24 and from 33 to 48 satisfy this condition. The selection between both sets of coefficients (i.e. both sets of regions in the curve) is thus determined by the fractional part of the base-2 logarithm of the number of statements.

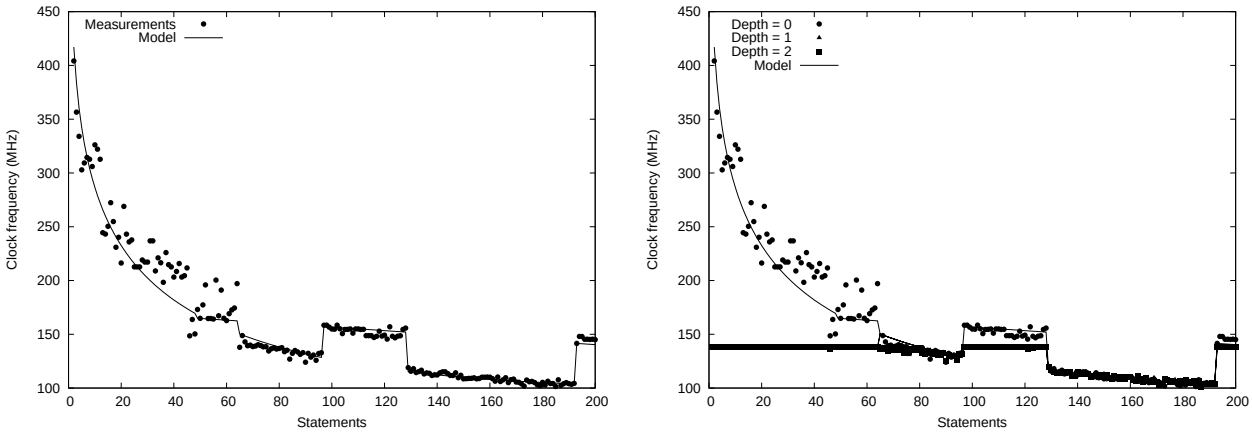
4.1.2 Maximum loop nest depth

CLooGVHDL generates a loop counter and an identifier block for each loop level in the program. For instance, in the hardware architecture presented in Fig. 2, two loop counters and three identifier blocks are used to implement a maximum loop nest depth of two. As a result, the maximum loop nest depth should have a large impact on the area usage. To measure this impact, we generate a large set of loop controllers with a varying number of statements and a varying loop nest depth. As one can see in Fig. 3(b), the irregular shape of the curve is preserved.

The impact of the loop nest depth can be modeled by adding an extra term A_{depth} to (1):

$$A_d = d \times y \ . \qquad (2)$$

In this additional term, the maximum loop nest depth y is multiplied by a constant factor d, which is the same for both regions in the curves. This constant depends on whether the loop bounds are parametric or constant. If the loop bounds are constant, then the automata implementing the



(a) Frequency vs. statements

(b) Frequency vs. loop depth

Figure 4. Parameters affecting the clock frequency for Xilinx. The points in the figure indicate the measured clock frequency, while the full line represents the estimation model. Regions with a higher area utilization also have a higher clock frequency.

loop counters are smaller and thus this constant factor d is smaller.

There is only one loop counter and one identifier block for each level in the program. As you can see in Fig. 2, the loop counter *FOR0* implements both i- and k-loops. As a result, the size of the controller should thus be mainly determined by the deepest loop nest in the program and not by the other loop nests in the program. This was confirmed by the experiments.

4.1.3 Other parameters

Loop nests in real-world applications are not always as simple as in our synthetic benchmarks. Therefore some additional parameters that affect the loop controller area usage, such as the presence of conditionals or operations in the loop boundaries, were identified. All these additional parameters can easily be extracted from the polyhedral program representation used in CLooG. For instance, if a certain equation of a polyhedral domain, contains more than two non-zero elements, one or more operations is needed in the calculation of the upper or lower loop boundary.

The impact of these additional parameters on the area usage is modeled by adding an extra term A_o to the equations presented above. However, in the majority of our benchmarks, this term is rather small compared to A_s and A_d.

This leads to our final estimation model:

$$A = A_s + A_d + A_o \; . \tag{3}$$

with A the total area needed by the loop controller.

4.2 Clock frequency

We will investigate the influence of the parameters discussed in the previous section on the maximum clock frequency of the loop controller.

4.2.1 Number of statements

If we plot the maximum clock frequency of the loop controller as a function of the number of statements for a sequential program (Fig. 4(a)), we see that there are irregularities in the curve at the same positions as in Fig. 3(a). Regions with a higher area utilization also have a higher maximum clock frequency.

Opposed to the area estimation model, both regions are now modeled by a different equation. In the first set of regions, corresponding with the lower regions in our area estimation model, the clock frequency decreases logarithmically with the number of statements. In the other regions of the curve, there is a linear dependency between the number of statements and the clock frequency. This leads to the following set of equations:

$$F_s = \begin{cases} a_1 \times ln(x) + b_1 \times x + c_1 & x \in L_n \\ b_2 \times x + c_2 & x \notin L_n \end{cases} \; . \tag{4}$$

with x the number of statements in the program and a_1, b_1 and b_2 negative constants and c_1 and c_2 positive constants, F_s is the clock frequency a sequential loop controller.

The maximum clock frequency of designs with less than 64 statements is distributed more randomly. As a result, our model can not estimate these designs as accurately as larger

controllers, but this turned out not to be a real problem during the validation of our model.

4.2.2 Maximum loop nest depth

Only the presence of loops in the program has an an impact on the clock frequency. The maximum depth of the loop nests does not influence the clock frequency, as is shown in Fig. 4(b). If the number of statements in the program is smaller than 64 or the L_n condition is not satisfied, then the clock frequency has a constant value. Otherwise, it can be described by logarithmically decreasing function in Eq 4. This gives us the following formula:

$$F_d = \begin{cases} d & x \notin L_n \text{ or } x \leq 64 \\ a_1 \times ln(x) + b_1 \times x + c_1 & \text{otherwise} \end{cases}. \quad (5)$$

where x is again the number of statements and d is a constant value. Constants a_1, b_1 and c_1 are the same as in Eq. 4.

4.2.3 Number of operations

We have also investigated several additional parameters that can affect the maximum clock frequency. During our experiments, it turned out that the major parameter affecting the clock frequency is the number of operations used in the loop bounds. As opposed to the area estimation model, this parameter can have a huge impact on the achievable clock frequency. This can be modeled by subtracting an additional term F_o from the equations presented above, where F_o is:

$$F_o = e \times ln(z). \quad (6)$$

where z is the total number of operations and e is a constant value. Our overall clock frequency estimation model can then be written as follows:

$$F = \begin{cases} F_s - F_o & \text{sequential program} \\ F_d - F_o & \text{otherwise} \end{cases}. \quad (7)$$

5. Experimental results

A second set of benchmarks provided with CLooG [1] was used to validate our model from the previous section. This set contains a number of loop controllers extracted from the PerfectClub benchmark suite. We managed to keep the estimation error reasonably low, with an average estimation error of not more than 7.14% for our area estimations and 6.79% if we estimate the maximum achievable clock frequency.

Next to the validation with the PerfectClub suite, we have also applied our model to a real-world application. We used our model to estimate the loop controllers from the 2D-IDWT example presented in Table 1. The area estimations and the actual synthesis values are presented in Table 2. There is a good correlation between the synthesis results and our estimations, with an average error of 5.79%. Moreover, our estimation technique leaves the relative order of the different designs intact and similar designs have similar estimates.

Table 2. Area estimations and synthesis results from several 2D-IDWT variants.

Design	Area	Estimation	Rel. error
RC	180	185	-2.98
RC tile X	992	920	7.22
RC tile Y	731	704	3.74
RC tile X + Y	1545	1695	-9.71
RC tile Y + X	1913	1913	0.00
RC tile Y + X + LB	2863	2888	0.87
LB	527	473	10.31
LB tile X	3499	4249	-21.45
LB tile Y	1215	1216	-0.07
LB tile Y + X	2686	2644	1.58
Average Error			5.79%

Due to the simplicity of our model, we managed to keep the estimation overhead minimal. This is critical when there are lots of design variants that need to be estimated. The accurate estimations, combined with the low estimation overhead makes our model very useful during design space exploration. Our model can thus be used to guide a designer or an automated design space exploration framework in selecting a good sequence of transformations.

The clock frequency estimation results are similar to those from the area estimations, although at this moment the accuracy is not as good when estimating the 2D-IDWT variants.

6. Related work

The need to estimate the performance during high-level synthesis has been recognized before by many researchers. In [5, 9], estimations are derived from a Data Flow Graph (DFG). As a DFG does not contain any control information, the controller is not taken into account. These approaches are complementary to our work, since they can be used to estimate the area and the delay of the data path. The accuracy reported in [9] is about 5%.

The problem of estimating the controller from a high-level specification was addressed by [11] for FPGAs and by [8] for ASICs. Both papers use an equation based approach to estimate the performance. These techniques estimate the low level controller implementation, which is responsible for steering the multiplexers and registers. Our work ad-

dresses the complete, high-level control in the design, including loops.

The techniques presented in [10, 13, 15] address the estimation of complete designs. In [15], estimations are constructed from a netlist, which is then mapped onto the CLBs for the Xilinx XC4000. However, netlist creation is a part of the work done by the synthesis tools. Our technique skips this expensive step by estimating at a more abstract level. In contrast to our work, the approach in [15], is also very technology dependent and hence can not be ported easily to other architectures.

The technique presented in [13] makes use of the MATCH compiler to translate a MATLAB specification into VHDL code. Area and delay estimations are made after the scheduling and register allocation steps. The overhead for this method is again larger than in our approach. Meeuws et al. [10] present a very fast prediction model based on software quality metrics. Their approach is aimed towards a hardware/software partitioning tool. They report a mean error of more than 69%, which is much too large for our application.

Design Trotter [12] is a complete design space exploration framework. The DFG is expanded to include control information, resulting in a hierarchical control and data flow graph. As a result, both the data path and the controller are estimated. However, estimations are less accurate than the results obtained by our approach. An average error of 20% for area values is reported, which could lead to large errors during design space exploration.

7. Conclusions and Future work

We have presented a model to estimate the FPGA resources needed by the loop controller and the maximum achievable clock frequency during high-level synthesis. The models presented in this paper are accurate enough to be used during design space exploration. Due to their simplicity and the abstract level of estimating, the estimation overhead is minimal, which is critical when lots of design variants have to be evaluated. The presented methodology can easily be adapted to new FPGA design flows and architectures. Next to the estimation models for Xilinx FPGAs, we have also developed similar models for Altera FPGAs.

Future work will focus on the data path during the estimations. At this point, the techniques presented in other papers, like in [5, 9], could be used in conjunction with this work. By combining our area estimation error of about 7% with the 5% obtained by [9], we strongly believe that we can estimate complete designs within 10% accuracy, which is much better than the 20% from Design Trotter, as stated in [12]. Next to this, additional research is needed to move towards an automated design space exploration framework.

Acknowledgments

Tom Degryse is supported by a BOF grant from Ghent University. Part of this research is supported by the IWT grant 060068.

References

[1] CLooG benchmarks. `http://www.CLooG.org`.

[2] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT*, pages 7–16, 2004.

[3] F. Catthoor, E. de Greef, and S. Wuytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.

[4] H. Devos, K. Beyls, M. Christiaens, J. Van Campenhout, E. H. D'Hollander, and D. Stroobandt. Finding and applying loop transformations for generating optimized FPGA implementations. *Transactions on High Performance Embedded Architectures and Compilers I, LNCS*, 4050:159–178, 2007.

[5] R. Enzler, T. Jeger, D. Cottet, and G. Tröster. High-level area and performance estimation of hardware building blocks on FPGAs. In *FPL*, pages 525–534, 2000.

[6] S. Girbal, N. Vasilache, C. Bastoul, A. Cohen, D. Parello, M. Sigler, and O. Temam. Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies. *International Journal of Parallel Programming*, 34(3):261–317, 2006.

[7] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers. Optimized generation of data-path from C codes for FPGAs. In *DATE*, pages 112–117, 2005.

[8] G. R. Gupta, M. Gupta, and P. R. Panda. Rapid estimation of control delay from high-level specifications. In *DAC*, pages 455–458, 2006.

[9] D. Kulkarni, W. A. Najjar, R. Rinker, and F. J. Kurdahi. Compile-time area estimation for LUT-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems*, 11(1):104–122, 2006.

[10] R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis. A quantitative prediction model for hardware/software partitioning. In *FPL*, pages 735–739, 2007.

[11] C. Menn, O. Bringmann, and W. Rosenstiel. Controller estimation for FPGA target architectures during high-level synthesis. In *ISSS*, pages 56–61, 2002.

[12] Y. L. Moullec, J.-P. Diguet, T. Gourdeaux, and J.-L. Philippe. Design-trotter: System-level dynamic estimation task a first step towards platform architecture selection. *Journal of Embedded Computing*, 1(4):565–586, 2005.

[13] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee. Accurate area and delay estimators for FPGAs. In *DATE*, pages 862–869, 2002.

[14] F. Quilleré, S. V. Rajopadhye, and D. Wilde. Generation of efficient nested loops from polyhedra. *International Journal of Parallel Programming*, 28(5):469–498, 2000.

[15] M. Xu and F. J. Kurdahi. Accurate prediction of quality metrics for logic level designs targeted toward lookup-table-based FPGA's. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(4):411–418, 1999.