IDENTIFYING OPPORTUNITIES FOR DYNAMIC CIRCUIT SPECIALIZATION

Tom Davidson, Karel Bruneel, Dirk Stroobandt

ELIS CSL, HES group
Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
email: [name].[surname]@elis.ugent.be

1. INTRODUCTION

This work describes the identification of designs that benefit from a Dynamic Circuit Specialization (DCS) implementation on FPGAs. In DCS, the circuit is specialized for slowly changing inputs, called parameters. For certain applications or cores, a DCS implementation is faster and smaller than the original implementation. DCS implementations can benefit from the possibility in modern FPGAs to reconfigure specific configuration bits at run-time. This means DCS can be used to specialize an FPGA circuit during the run-time of the FPGA.

Initially, in DCS, it is assumed that the parameters are constant. Constant propagation then allows the specialization of the circuit by partially evaluating the circuit. This specialized circuit is smaller and faster than the original circuit but is only correct for one specific parameter value. Of course, the parameters will not stay constant for ever, eventually they will change. This is solved using the run-time reconfiguration capabilities of the FPGA. A DCS system has both an FPGA and a configuration manager. The configuration manager is responsible for generating the specialized circuit and reconfiguring the FPGA.

In an implemented DCS system, the FPGA contains a specialized circuit for the current parameter value. The FPGA is working normally until the value of a parameter changes. Using the new parameter value, a new specialized circuit is generated by the configuration manager and the FPGA is reconfigured with this new circuit. During this process the FPGA is halted. Once the reconfiguration has finished, the FPGA is working normally again. Until the next parameter change, then the specialization process starts over. The time and resources needed to generate the new circuit and to reconfigure the FPGA are overheads DCS introduces. The time of one specialization is called the *single specialization overhead.*

Such a DCS system could be implemented in multiple ways. In [1] an efficient method for DCS, developed by Ghent University, is described. It includes an FPGA tool flow adapted for DCS, which will be used in the following sections. For the moment, it only implements the reconfiguration of LUT truth tables, and not the routing infrastructure. Only a select number of LUTs are run-time reconfigured, these LUTS are called TLUTs.

Previous papers have shown that this method for DCS can achieve significant area reductions in a number of hardware designs. In [1], an adaptive 16-TAP FIR-filter is implemented using 56% less area. It uses only 1301 LUTs, while the size of the original implementation was 2999 LUTs. The resulting DCS implementation is also 27% faster than the original implementation. The single specialisation overhead is 166μs. This is the design used for the profiler run-time measurements in Section 3. The results for larger FIR-filters are similar. The same publication also shows a 66% LUT reduction for Ternary Content-Addressable Memories. Both of these results, and the adapted FPGA tool flow, were verified by building these DCS implementations on an actual FPGA, the Xilinx Virtex II Pro.

[2] shows that a number of key-based encryption algorithms also see a significant area reduction, 20.6% for AES. 27.8% for Triple DES and a 72.7% reduction for the rc6-algorithm. Finally, in [3] this method for DCS is shown to achieve similar results as hand-crafted run-time reconfigurable implementations of a Network Intrusion Detection System (NIDS). This paper also presents improvements to make the NIDS implementation fully run-time reconfigurable, using this DCS method.

In this paper, we present a profiling tool to aid the designer in analysing the feasibility of a DCS implementation (Section 3). It automatically provides a functional density estimate (see Section 2) for the most interesting DCS implementations.

2. DCS IDENTIFICATION OF A DESIGN

Determining whether or not a certain design will benefit from a DCS implementation is a difficult task for the designer. First, it requires the designer to be familiar with the exact DCS-method. Secondly, in order to find a good parameter selection, insight into the dynamic behaviour of the signals in the design is required. This is typically not something that is important in the normal design process. In addition, there are no tools that allow the designer to analyse the dynamic behaviour of large groups of signals. VHDL and Verilog Simlators do exist, but they focus on verifying the behaviour of a limited number of signals. Even after the parameter selection it is very difficult to predict the gains of a DCS implementation without actually implementing it. To solve this problem, we first decided on a metric that allows an easier comparison between implementations. This metric is the functional density, and is explained in detail below. The next section presents a profiler that uses this metric to automatically analyse an existing implementation.

To identify designs that benefit from DCS, different implementations of the design have to be compared. A good measure for this comparison is the functional density (FD) [4]. It is the number of computations per unit of area and per unit of time (Equation 1). T_{FPGA} is the complete execution time of the FPGA. N is the number of operations, and A_{FPGA} is the number of LUTs in the implementation.

$$FD_{DCS} = \frac{N}{A_{FPGA} \times T_{FPGA}} \qquad (1)$$

The number of operations, N, can always be expressed as the number of clock cycles times a correction factor (C). In the case of the FIR-filter, where one input sample is processed every clock cycle, N can be redefined as exactly the number of clock cycles.

DCS implementations where the benefits are high will occupy less area and therefore have a higher FD. On the other hand, if DCS introduces a large time overhead, the total execution time will increase, while the number of operations stays the same, leading to a lower FD. A design benefits from DCS if a DCS implementation with a higher FD that the original implementation can be found.

In the DCS implementation a number of signals will be selected as parameters. We call this the parameter set. The most exhaustive way to find the best DCS-implementation is to calculate the FD for all possible parameter sets. However, this would take a prohibitively long time, because (i) the number of signals in complex designs is very high and (ii) calculating the FD itself requires up to hours for complex designs.

To address (i), the most interesting parameters for DCS are identified based on Equation 2. This equation expresses the FD as a function of the average single specialization overhead ($\hat{T}_{SST}$) and the average time the FPGA is working for a single parameter value ($\hat{T}_{FPGA}$).

$$FD_{DCS} = \frac{1}{\frac{\hat{T}_{SST}}{\hat{T}_{FPGA}} + 1} \frac{N}{A_{FPGA} \times T_{FPGA}} \qquad (2)$$

The first part of Equation 2 expresses the degradation of the FD, caused by the single specialization overhead. It is clear that the degradation will be small if the single specialization time is much smaller than the average time for each parameter value. The influence of the degradation can be seen clearly in Figure 1. This figure shows how the functional density is dependent on the average time between parameter changes ($\hat{T}_{FPGA}$). Looking more closely at the first part of Equation 2, this means only signals for which $\hat{T}_{SST}$ is (much) smaller than $\hat{T}_{FPGA}$ will have a low degradation. In other words, only signals for which the time between transitions ($\hat{T}_{FPGA}$) is much longer that the overhead for a single reconfiguration ($\hat{T}_{SST}$) are interesting parameter candidates. To reduce the number of signals under consideration, signals for which $\hat{T}_{SST}$ is larger than $\hat{T}_{FPGA}$ are ignored. A good value for $\hat{T}_{SST}$ is discussed in Section 3.

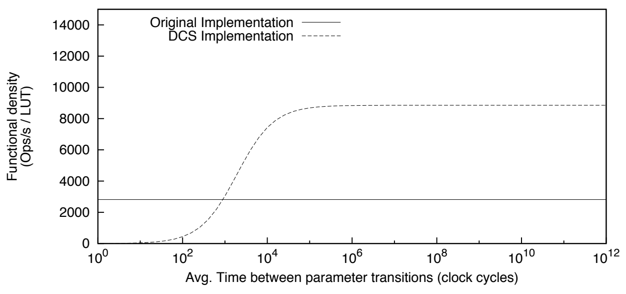

Fig. 1. Influence of the degradation on the functional density of a DCS-implementation. Compared to the functional density of a generic implementation.

As for (ii, the long time needed to calculate the FD), the FD has to be determined for each of the remaining parameter candidates. Calculating the FD exactly requires running the full FPGA tool flow, which can take hours for complex designs [5]. Most time is spent in the last two steps, placement and routing. In addition, both of these steps scale very badly with increasing circuit size, since both are NP-complete problems.

However, the FD can also be estimated, before placement and routing. For this estimation, Equation 3, a rewritten version of Equation 2, is used. Where, $\hat{T}_O$ is the average time between parameter changes in the original implementation. T^1_{DCS} and T^1_O are the clock periods of the DCS and the original implementation, respectively. All these variables can be estimated based on the FPGA tool flow before placement and routing. A more detailed discussion in Section 3.

$$FD_{DCS} = \frac{C}{\frac{\hat{T}_{SST}}{\hat{T}_O \frac{T^1_{DCS}}{T^1_O}} + 1} \frac{1}{A_{FPGA} \times T^1_{DCS}} \qquad (3)$$

3. PROFILER

To help the designer analyse the different DCS implementations of designs based on the FD metric, an automatic profiling tool was developed. It requires an RTL description of the design and a test bench with realistic input data. The profiler uses a two-step approach. In the first step a number of parameter candidates are selected from all the signals in the design. In the second step, the functional density is estimated for each of those candidates. Both steps are discussed in more detail below.

Selecting the parmeter candidates: First, the profiler runs the test bench through a simulator to gather data on the dynamic behaviour of all signals. This data is then analysed in order to remove all signals for which $\hat{T}_O$ is smaller than the chosen $\hat{T}_{SST}$. This is the criterion explained in the previous section. However, because the exact overhead of the DCS-implementation is not known yet, a minimal value for $\hat{T}_{SST}$ was chosen. This minimal $\hat{T}_{SST}$ is the time needed to run-time reconfigure one single LUT. This removes all the signals that have a large FD degradation even with the smallest possible overhead. All the remaining signals, the parameter candidates, are given to the next step of the profiler, which will estimate the FD for each candidate.

Functional Density estimate: This FD estimate is based on Equation 3. Here, $\hat{T}_O$ is collected from the analysis of the dynamic signal behaviour in the previous step. All the other variables (A_{FPGA}, $\hat{T}_{SST}$,T^1_{DCS} and T^1_O) are collected by running an abbreviated FPGA tool flow for DCS [1], without placement and routing. This flow is run for each parameter candidate. After this flow has finished, A_{FPGA} is known exactly and $\hat{T}_{SST}$, T^1_{DCS} and T^1_O can be estimated. The details of these estimates are discussed below.

3.1. Single Specialization Time ($\hat{T}_{SST}$)

The single specialization time is the run-time overhead introduced by DCS. It has two parts: the time needed for generating a new circuit ($T_{generation}$) and the time needed for the actual reconfiguration of the FPGA ($T_{reconfiguration}$).

The FPGA tool flow presented in [1] uses Boolean functions to express how the TLUT truth tables are dependent on the parameter values. The new circuit is generated by evaluating these Boolean functions for the new parameter values. This *generation time* is estimated using the number of Boolean operations ($BoolOps$) and the chosen computation unit (K). K represents the average total overhead of one Boolean operation. It is determined by running the complete run-time reconfiguration flow for a large design multiple times, while each time measuring only the time required for the Boolean evaluation.

$$T_{generation} = BoolOps \times K \tag{4}$$

This computation unit is generally also the configuration manager. A good option is the embedded CPU in a lot of modern FPGAs. For the Xilinx Virtex II Pro FPGA, this is the PowerPC 405. In that case, K is 3.32 clock cycles.

The *reconfiguration time* is dependent on the chosen reconfiguration method. [1] proposes two methods, one method using the HWICAP and one using the Shift Register LUT (SRL) capability of Xilinx FPGA's.

The HWICAP is the standard configuration interface provided by Xilinx. In this case, the FPGA is reconfigured frame by frame. To estimate the *reconfiguration time* we estimate number of frames that needs to be reconfigured, assuming the TLUTs are spread out randomly over the total number of LUTs. The reconfiguration time is then Equation 5. For the Virtex II Pro, a single frame is reconfigured in 98.23 μs.

$$T^{HWICAP}_{reconfiguration} = E[\#frames] \times T_{frame} \tag{5}$$

The second method for run-time reconfiguration uses the Shift-Register LUT capabilities, present in some modern FPGAs. This allows the TLUTs to be combined in one or more shift registers chains. This method of reconfiguration is much faster because each chain can be reconfigured in parallel and only the actual truth table bits are sent. A HWICAP frame carries a lot more overhead. The SRL chains are clocked at the design speed. For the above information, the *reconfiguration time* using SRLs can be estimated easily. It uses the number of TLUTS, the number of chains and the clock speed of the DCS implementation (Equation 6).

$$T^{SRL}_{reconfiguration} = \frac{\#TLUTS \times 16 \times T^1_{DCS}}{\#chains} \tag{6}$$

3.2. Clock periods (T^1_{DCS},T^1_O)

The clock periods, T^1_{DCS}, of the DCS implementation, and T^1_O, of the original implementation, are estimated by the number of LUTs in the longest path of the mapping result of each implementation. This depth is then multiplied with a worst-case estimate of one complete LUT delay for the target FPGA. To get the complete clock period estimation a pre and post delay are added. It is assumed that the longest path will be from FF to FF, not from I/O Block to I/O Block. For the Virtex II Pro, the LUT gate delay is 0.275 ns and the LUT net delay is 0.575. The pre and post delays are 0.886 ns and 0.208 ns respectively.

$$T^1_{estimate} = T_{pre} + depth \times T_{LUTdelay} + T_{post} \tag{7}$$

3.3. Profiling time

Using FD estimates instead of exact FD calculations, reduces the execution time of the profiler significantly. As discussed earlier, the most time intensive parts of the FPGA tool flow are the placement and the routing [5]. Both of these steps are avoided by using the estimates. The impact of using the estimates is shown in the execution time measurements described below. These experiments were done for a Virtex II Pro (xc2vp30-7ff1152), the profiler was run on a computer with 8 GBs of RAM and an Intel Core2 Quad Q9650 (3GHz, 1333MHz, 12MB).

Two adaptive FIR filters, a 16 TAP and a 32 TAP version, and corresponding test benches were analysed by the profiler. The profiler was run two times for each FIR filter, one run used the FD estimates, the other the exact FD calculation. In each case the execution time for each parameter candidate was measured. We will discuss the average execution time per parameter candidates for all cases.

Table 1. The average execution time for one parameter candidate

	FD Estimate	Exact FD
16 TAP filter	36.12 s	107.12 s
32 TAP filter	41.67 s	248.10 s

The FD estimate for one parameter candidate in a 16 TAP adaptive FIR filter requires 36.12 seconds. Calculating the FD exactly instead of estimating it requires an extra 71 seconds. In that case, the total analysis time for this filter would increase from 10.23 minutes to 30.35 minutes. This extra time is the time needed for the placement and the routing of the design.

In addition, because the placement is an NP-complete problem, it scales badly with increasing circuit size [5]. E.g. for a 32 TAP adaptive FIR filter, the time needed for the exact calculation of the FD is already 248.10 seconds for each parameter candidate, while estimating the FD still only requires 41.67 seconds, a difference of 206.43 seconds. So, even though the size of the circuit has only doubled, the time for placement and routing has increased by 2.9x. The 32 TAP adaptive FIR filter still only uses 15% of the Virtex II Pro area. This effect will be even more pronounced in designs that use a larger FPGA area.

We are currently preparing an extensive discussion on the accuracy of the FD estimates. However, it is already clear that if the FD estimates predict a significant gain, then the exact, calculated, FDs will also show a significant gain.

4. CONCLUSION

This paper shows how to identify designs that benefit from DCS implementations, using the functional density (FD) as a metric. In addition, a profiler that implements this metric is presented. It automatically analyses the quality of the most interesting DCS implementations of a given design. This allows the designer to determine more easily whether a certain core benefits from a DCS implementation or not. To reduce the execution time of the profiler an FD estimate is used instead of an exact calculation.

This profiler is the first step towards a completely automatic tool flow for DCS. This flow would allow the implementation of DCS systems without any intervention of the designer. Additionally, self-aware reconfigurable systems could also use this automatic flow to analyse their own dynamic behaviour, which could lead to optimal DCS implementations. Off course, this kind of flow would, most probably, require running the full FPGA tool flow at runtime. A process which can take up to hours for complex systems. However, dependent on the practical implementation of these self aware systems, this kind of elaborate self-analysis would only be done very infrequently.

Acknowledgements

This research was supported by both the IWT and the European Commission in the context of the FP7 FASTER project (#287804).

5. REFERENCES

[1] K. Bruneel, "Efficient circuit specialization for dynamic reconfiguration of FPGAs," Ph.D. dissertation, Ghent University, 2011.

[2] T. Davidson, F. Abouelella, K. Bruneel, and D. Stroobandt, "Dynamic circuit specialisation for key-based encryption algorithms and DNA alignment," *International Journal of Reconfigurable Computing*, 2012.

[3] T. Davidson, M. Merlier, K. Bruneel, and D. Stroobandt, "A dynamically reconfigurable pattern matcher for regular expressions on fpga," in *ParCo2011*.

[4] A. M. Dehon, *Reconfigurable architectures for general-purpose computing*. Massachusetts Institute of Technology, 1996.

[5] C. Ababei, "Speeding up FPGA placement via partitioning and multithreading," *International Journal of Reconfigurable Computing*, 2009.