HSDPA Design Space Exploration and Implementation Guidance with Design-Trotter

Šaramentovas, Aleksandras; Ruzgys, Paulius; Abildgren, Rasmus; Le Moullec, Yannick

Published in:
IEEE ICICS 2007

DOI (link to publication from Publisher):
10.1109/ICICS.2007.4449771

Publication date:
2007

Document Version
Accepted author manuscript, peer reviewed version

HSDPA Design Space Exploration and Implementation Guidance with Design-Trotter

Rasmus Abildgren*, Aleksandras Šaramentovas†, Paulius Ruzgys†, and Yannick Le Moullec†
email: {rab,aleksara,paulius,ylm} @es.aau.dk

*Center for Embedded Software Systems (CISS)
Aalborg University, Selma Lagerlöfs Vej 300,
DK-9220 Aalborg East, Denmark

†Center for Software Define Radio (CSDR)
Aalborg University, Niels Jernes Vej 12,
DK-9220 Aalborg East, Denmark

***Abstract*—This work addresses some of the implementation challenges for recent and future wireless communication systems. More specifically this paper describes a design methodology for design space exploration and implementation guidance and illustrates its practical usage and benefits by applying it to some of the most critical sub-parts (i.e., the turbo-encoder and turbo-decoder) of the HSDPA concept. The implementation examples and results shows how the proposed methodology based on our tool Design-Trotter can guide system designers in selecting and/or building the most appropriate architecture for their application.**

I. Introduction

In order to satisfy the ever-increasing need for data-traffic and high-speed services in the wireless domain, new concepts have been developed to increase the spectral efficiency of current third generation systems to support high user data rates. The High Speed Downlink Packet Access (HSDPA) concept [1], which has been validated by the Third Generation Partnership Project (3GPP) in the specification of the Release 5 is a significant step to boost the WCDMA performance for downlink packet traffic, enabling user peak data rates up to 14 Mbps. HSDPA offers new opportunities for wireless communications but also raises a number of implementation challenges in terms of platform selection (e.g., DSP-processor, FPGA, GPP), optimizations (e.g., time, power and area), etc.

In order to alleviate system designers from time-consuming and error-prone design tasks which increase the time-to-market factor, a systematic and efficient design methodology is highly desirable. This work illustrates how designers can benefit from a guidance methodology for the implementation of the HSDPA technique by means of design space exploration (DSE) with the Design-Trotter tool [2]. In particular we investigate how DSE combined with the characterization of the application by means of metrics provide a design trajectory allowing early and right decisions for selecting and/or building the most appropriate target architecture according to the system specifications, and thus reducing the time-to-market factor.

The remainder of the paper is organized as follows:

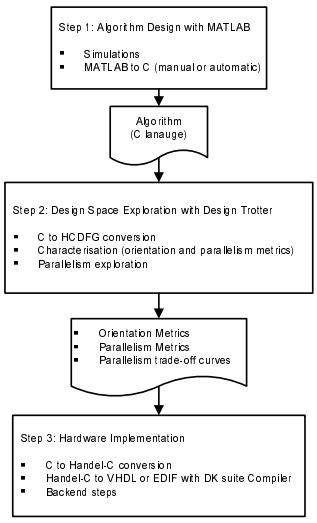

Fig. 1. Our mehodology for rapid development of wireless applicaitons.

Section II introduces the main steps of the proposed design methodology and discusses some of the key issues related to design space exploration and how to move from high-level estimates to an actual implementation. Then section III illustrates how the proposed methodology can be used for guiding the implementation of some of the most critical sub-parts of the HSDPA concept (i.e., the turbo-encoder and turbo-decoder) by means of design space exploration examples and implementation results. Finally we conclude in section IV.

II. Methodology

Modern system development requires a design methodology which enables the designer to explore different implementations in order to choose one which fulfils the performance requirements to the product. In this work we consider a design methodology build upon the design space exploration tool Design-Trotter. The overall design methodology is summarised in Fig.1 and the main steps are presented hereafter. Further details about the tool Design-Trotter can be found in [2].

The task of analysing an algorithm with a design space exploration tool consists of characterizing the algorithm, in such a way that the designer is able to get useful information about the performance of different implementations, typically in terms of a resource vs. execution time, or area vs. time curve.

Based on the design space exploration results, a particular solution can then be further explored and implemented. Some tools try to do this automatically based on the algorithm description, however this task is still done more or less manually in many situations. Due to the time-to-marked parameter, high-level languages are increasingly used for implementation. It is therefore important that the used high-level languages are able to express the detailed needs in order to implement the chosen design.

In the following we describe the individual tasks and illustrate some of the issues involved when going from DSE estimates to real implementations using high-level languages.

A. Design Space Exploration with Design-Trotter

Design-Trotter [2] is an academic design space exploration tool conjointly developed by LESTER lab, Université de Bretagne Sud, France and CSDR, Aalborg University, Denmark.

For guidance purposes metrics are computed to rapidly stress the proper architecture style for the application, e.g., the ratio of explicit parallelism versus the pipeline depth, the need for complex control structures, the requirements in terms of local memories and specific bandwidth, and the need for processing resources for specific computations or address generation. Design-Trotter computes three orientation metrics [3]: the Memory Orientation Metric (MOM), the Control Orientation Metric (COM), and the criticity (average potential par-

allelism) of a function (γ).

Since parallelism has a direct impact on several performance factors such as execution time, energy consumption, area, etc., Design-Trotter explores the potential parallelism of an application in terms of i) type (data-transfer and data processing), ii) granularity level, and iii) type (spatial, temporal). Design-Trotter rapidly provide dynamic exploration of an application by means of parallelism vs. delay trade-off curves on which a point corresponds to a potential architecture.

The analysis of the algorithm under consideration in Design-Trotter, is done automatically, and provide the design with the above mentioned information. The designer use these information to identify which possible solutions in the solution space are fulfilling the requirements. Since the solutions provided by Design-Trotter are estimates, it is important that these estimates are close to the performance of the real implementation.

B. Design-Trotter Solution to Handel-C

Having the algorithm and the design suggestions provided by Design-Trotter, the next task is to perform the actual implementation. To keep the development time short, we use the high-level language Handel-C [4].

Each solution proposed by Design-Trotter could be implemented on an FPGA using a HDL. Ideally this could also be the case for high-level languages however, the main problem at this point is how to achieve this precision. To settle it the following elements are used: the resource schedule details provided by Desing-Trotter and the ”par” statement in Handel-C. Firstly the C source code used in Design-Trotter is converted to Handel-C; secondly by referring to the schedule details of the desired Design-Trotter solution, we can manually express parallelism inside the top-level blocks of an algorithm using the ”par” construct in the corresponding

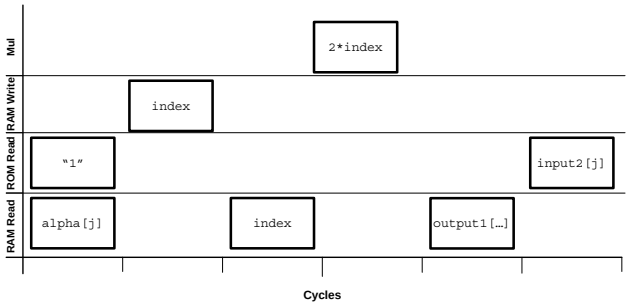

Fig. 2. Schedule details of the interleaver, derived from Design-Trotter.

Handel-C code parts; thirdly the Handel-C code in DK Suite [4] is compiled to an EDIF design file, used for further implementation on the FPGA.

The structure of the compiled Handel-C code depends on the assignments in the Handel-C code, meaning that every assignment in the Handel-C code have a corresponding circuit which takes a clock cycle to execute.

Due to this fact, one can imagine that the Design-Trotter result in terms of cycle-budget will differs from the corresponding Handel-C result.

As an example, lets depict the schedule details of the encoders interleaver block, shown in Fig 3, of the example, which we consider later on.

The C source code of this interleaver is shown below:

```
/* interleave output of the upper RSC encoder */
for(j=0; j<FrameLength; j++){
  index = alpha[j]; // statement #1
  input2[j] = output1[2*index]; // statement #2
  }
```

As we see in Fig 2, one iteration of the interleavers loop body takes 6 cycles in Design-Trotter, where statement #1 in the code above takes only 2 cycles to be performed: it takes 1 cycles to read from memory the constant 2 and `alpha[j]` values, then 1 cycle to write `alpha[j]` to index variable. Statement #2

takes 4 cycles to be performed: 1 cycle to read index value, 1 cycle to perform `2*index` multiplication, 1 cycles to read `output1[j]` value, and 1 cycles to store `output1[2*index]` in `input2[j]`.

In Handel-C, one iteration of this loop body takes only 2 cycles: 1 cycle both for statements #1 and #2.

The difference in cycle-budget between the Design-Trotter result and the corresponding Handel-C results, will in most cases, have a overweight of cycles in the Design-Trotter solution. The reason is that assignments and control statements in Handel-C are incorporated in the cycle circuit, whereas they have their individual cycle in Design-Trotter. This means that in practice the implemented solution typically is more efficient in terms of timing performance than the corresponding solution, shown in the Design-Trotter trade-off curves between resource usage and cycle-budget.

If we use Design-Trotter as a guidance tool and want to implement the Design Trotter solution as precisely as possible in hardware, in terms of timing performance, we need to cope with this difference by trying to minimize it as much as possible.

C. Timing mis-match between Design-Trotter and Handel-C

If we consider the nature of these differences, it is clear that the reduction in the cycle budget, in the Handel-C case comes from the assignments. Not that it a constant reduction for all assignment but the reduction are more or less equal for the individual assignments whether they are executed in parallel or sequential.

Let us therefore consider the purely sequential solution. If we denote the number of cycles taken from the Design-Trotter estimate c_{DT} and the corresponding hardware implementations execution time (in cycles) c_{HW} (i.e. the code without any parallel constructs in the code). We can the calculate the ratio $p = \frac{c_{DT}}{c_{HW}}$. Finally, we divide the cycle-budget axes of the Design-Trotter execution time vs resources trade-off by the obtained p ratio. The new cycle axes incorporate the differences between Design-Trotter and Handel-C, and makes the Design-Trotter estimates applicable.

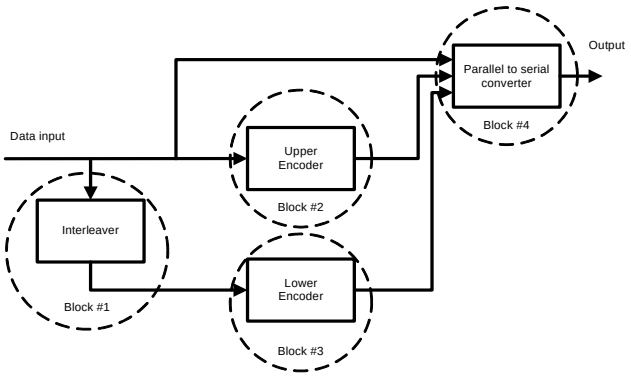

Fig. 3. Different blocks of the turbo encoder analysed.

Out tests shows that the new axes will to some extent match, the timing performance between Design-Trotter and Handel-C.

It should also be notice that when matching the timing performance in this way, it is important that all complex code assignments in the Handel-C code should be split into simpler assignments if possible, and that the same code should be used for the Design-Trotter analysis.

III. EXAMPLES

In the following we will illustrate the methodology applied on the turbo coder part of the HSDPA scheme. The turbo encoder and turbo decoder consist of different parts which are illustrated in Fig 3 and Fig 4 respectively.

A. Design-Trotter Characteristics

The characterization results derived by Design-Trotter for the turbo-encoder and turbo-decoder are given in Table I and Table II, respectively. The block numbers referring to the numbers shown in Fig 3 and Fig 4.

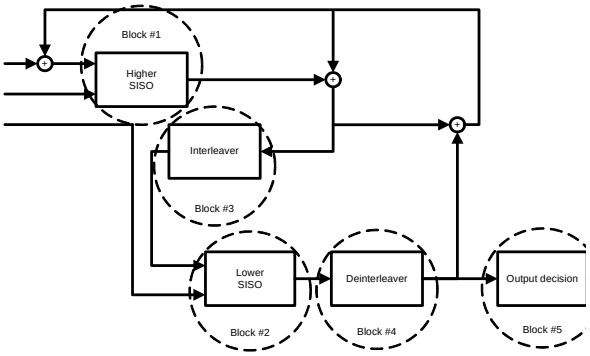

Fig. 4. Different blocks of the turbo decoder analysed.

TABLE I
METRIC OF THE ENCODER DERIVED FROM DESIGN TROTTER.

Block #	Block description	COM	MOM	γ
1	Interleaver	0.071	0.786	1.273
2	Upper RSC encoder	0.018	0.667	2.237
3	Lower RSC encoder	0.018	0.667	2.237
4	P/S converter (1 of 2)	0.03	0.636	3.286
5	P/S converter (2 of 2)	0.017	0.525	5.028

As seen in Table I and Table II, all blocks hav relatively low COM metric values denoting easily condi tioned data-flows, i.e., with almost no nondeterministi control operations in the algorithms. This is due to th fact that the loop indices are almost not data-dependent The MOM metric values, greater than $2/3$, indicate a important data accesses frequency: these blocks require high memory bandwidth in hardware. Finally, we ob-

TABLE II
METRIC OF THE DECODER DERIVED FROM DESIGN TROTTER.

Block #	Block description	COM	MOM	γ
1	Upper SISO processor	0.059	0.73	8.276
2	Lower SISO processor	0.059	0.73	8.276
3	Interleaver	0.077	0.846	1.183
4	Deinterleaver	0.143	0.793	1.473
5	Output formation	0.001	0.576	1.405

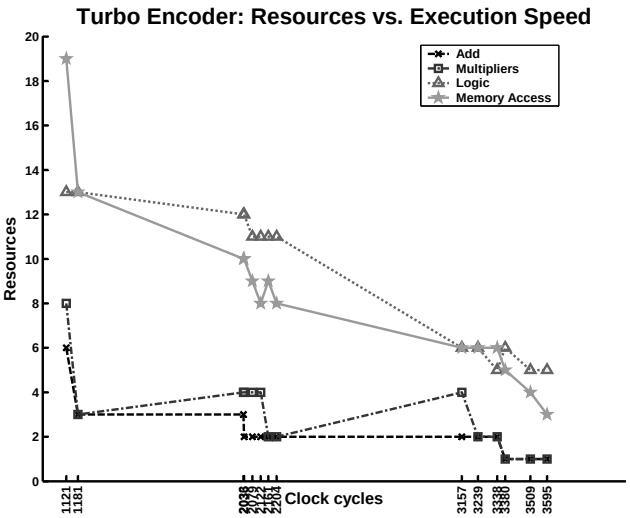

Fig. 5. Resource vs. Exec. Time graph for turbo encoder.

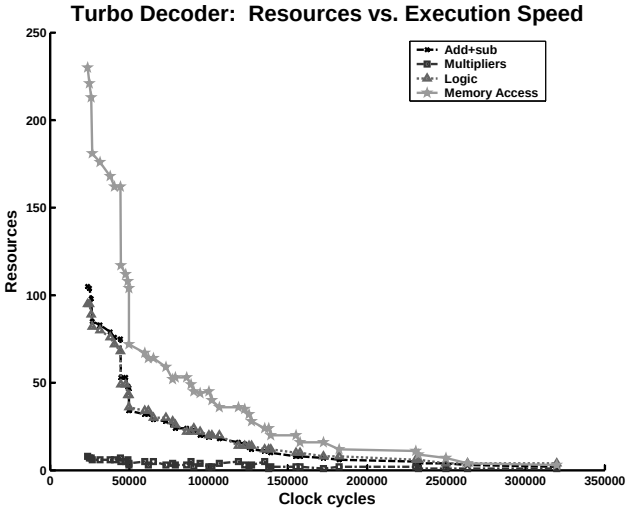

Fig. 6. Resource vs. Exec. Time graph for turbo decoder.

serve high parallelism (high γ value) in the encoders P/S converter, and in the decoders SISO processors. It means that these blocks can benefit from an architecture offering high parallelism capabilities (e.g., FPGA).

Design-Trotter also generates the resource vs. cycle-budget trade-off curves of the turbo-encoder and of the turbo-decoder as shown in Fig 5 and Fig 6 respectively.

For the turbo-encoder case, Fig 5 shows that the most-

right solution (3595 cycles) is purely sequential, i.e., at this point all of the turbo encoder operations are performed in a sequential manner using the minimum number of different operation resources.

With the most-left solution (1121 cycles), the maximum available parallelism for the encoder is achieved, where therefore this encoder operates in the fastest way as compared with other solutions. However, this solution is the most expensive in terms of resources. Finally the solutions in between offer different level of trade-off between the most sequential and most parallel ones.

For the decoder case, Fig 6 shows that there are more solutions than for the turbo-coder. This is mainly due to the fact that there is more processing and are less data-dependencies, thus more flexibility for scheduling the individual operations.

B. The Handel-C implementaion

Different versions, in terms of parallelism and split of complex statements are implementations of the encoder and decoder. These are shown in Table III.

The implementation results in terms of timing performance are presented in Table IV, and the implementation results in terms of resource usage are shown in Table V.

By examining the results of the different implementation of the turbo encoder, i.e., implementations #1, #2 and #3, from Table III, we see the following:

With implementation #1, the original sequential C source code of the encoder without complex statement splitting was used. Here we can observe (Table IV) that the cycle-budget of this encoder in Handel-C differs from the cycle-budget of the same encoder in Design-Trotter: in Handel-C it is about five times less (3595 cycles / 735 cycles) than the in Design-Trotter.

With implementation #2, all complex statements of the encoders code, used in implementation #1, are splitted into simpler statements. Here we see that the cycle-budget of the splitted code in Handel-C is now about three times less (5470 cycles / 1880 cycles) than the one of the same splitted code in Design-Trotter. It means that splitting of statements in some extent matches the timing performance between the related Handel-C and Design-Trotter codes. With implementation #3, the internal parallelism is expressed inside all blocks of the encoders code, used in implementation #2.

TABLE III
DIFFERENT IMPLEMENTATION USED IN THE EXAMPLE.

#	Algorithm	Solution	Statement splitting
1	Turbo encoder	Purely sequential	No
2	Turbo encoder	Purely sequential	Yes
3	Turbo encoder	Internal parallelism is max exploited	Yes
4	Turbo decoder	Purely sequential	No
5	Turbo decoder	Purely sequential	Yes

At this point we notice that the cycle-budget both in Handel-C and Design-Trotter is reduced about the same number of times (as compared with implementation #2), so using another solution from the trade-off curve gives the same trend in both Design-Trotter and Handel-C implementation.

Considering the implementation results of the turbo decoder, i.e., implementations #4 and #5, shown in Table IV. With implementation #4, the original C source code of the turbo decoder is implemented. With implementation #5, this code is splitted, i.e., all complex statements of this code are broken up into simpler statements.

When comparing implementations #4 and #5, we notice that splitting of complex statements increases the hardware clock speed.

TABLE IV

ESTIMATED EXECUTION TIME FROM DESIGN-TROTTER, AND EXECUTION TIME FROM HANDEL-C IMPLEMENTATION.

#	DT [Cycles]	Handel-C [Cycles]	HW clock [MHz]	Exec. time [μs]
1	3595	735	70.1	10.49
2	5470	1880	83.6	22.49
3	4357	1714	83.6	20.51
4	319691	39928	37.5	1066
5	249903	48213	57.6	836.7

TABLE V

RESOURCES USE OF THE DIFFERENT IMPLEMENTATIONS.

#	# of 4-input LUT	# slices	# RAM blocks
1	593	330	6
2	685	416	6
3	597	363	6
4	6348	3547	4
5	5662	3222	4

IV. CONCLUSION

In this paper some of the implementation challenges raised by recent and future wireless communication systems have been addressed. More specifically we have described a design methodology for design space exploration and implementation guidance for wireless systems. By applying the proposed methodology to the HSDPA concept we have illustrated its practical usage and benefits. The implementation examples and results have shown how the proposed methodology based on our tool Design-Trotter can alleviate system designers from time-consuming and error-prone design tasks, and thus reducing the time-to-market factor. In particular we have discussed the design space exploration and characterization of the turbo-encoder and turbo-decoder for HSDPA and illustrated how the exploration results can be used to guide the back-end implementation phase.

REFERENCES

[1] ETSI Mobile Competence Centre, "Overview of 3gpp release 5 - summary of all release 5 features - version 0.10," 2003.

[2] Yannick Le Moullec, Jean Philippe Diguet, Thierry Gourdeaux, and Jean-Luc Philippe, "Design-trotter: System-level dynamic estimation task - a first step towards platform architecture selection," *In Journal of Embedded Computing (IOSPRESS)*, vol. 1, no. 4, pp. 565–586, 2005.

[3] Y. Le Moullec, N. Ben Amor, J.-Ph. Diguet, M. Abid, and J-L. Philippe, "Multi-granularity metrics for the era of strongly personalized SOCs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2003.

[4] "Dk-design suite datasheet," http://www.celoxica.com, 2007.