Enhanced Reliability Design Automation Methodology considering the Generation of Parallel CRC Modules based on arbitrary CRC Polynomials and unlimited Data-Word Widths

Timo Brenningmeyer
University of Applied Sciences Osnabrueck
Laboratory of Micro- and Optoelectronics
Albrechtstr. 30, D-49076 Osnabrueck
Timo.Brenningmeyer@fh-osnabrueck.de

Ralf Göttsche
Intel GmbH
Theodor-Heuss-Straße 7,
D-38122 Braunschweig
Ralf.Goettsche@intel.com

Prof. Dr. Arno Ruckelshausen
University of Applied Sciences Osnabrueck
Laboratory of Micro- and Optoelectronics
Albrechtstr. 30, D-49076 Osnabrueck
A.Ruckelshausen@fhos.de

Abstract

Presentation of an enhanced design flow with automated generation of high performance and parallel CRC modules based on user-defined CRC polynomials and unlimited data word width.

1. Introduction

This paper describes an enhanced design flow providing parallel HDL CRC[1] modules for high performance data paths. The algorithm generating the parallel equations is implemented in a tool that integrates into the standard design automation flow to offer the best possible usability and efficiency.

CRC modules are usually implemented as linear feedback shift registers (LFSR) handling serial data streams as shown in figure 1. To calculate a checksum (black) for a n-bit data word, n clock cycles are needed. Each row in the table in figure 1 represents one calculation step (clock cycle). The starting point for parallel CRC calculations are the data word (grey) and the initial value of the register (light grey).

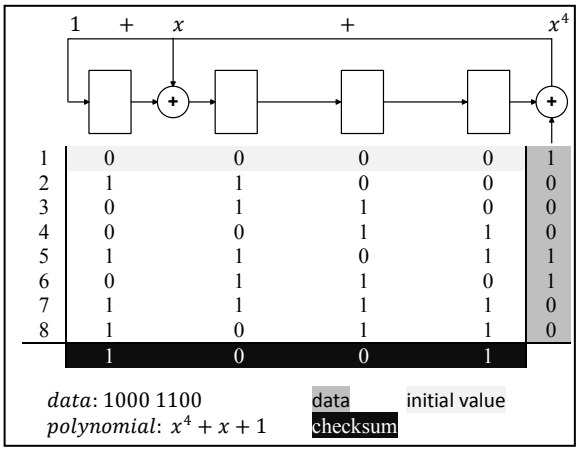

Figure 1: Serial CRC operation

Yi et al. ([1]) have developed an advanced algorithm which generates highly optimized results regarding the number of logic levels in the data path. Due to complex algorithmic operations it is difficult to implement. In [2], a parallel CRC function is described that focuses on generic applicability and does not require mathematical knowledge of the user. It is a very efficient and easy-to-use methodology to generate and implement parallel CRC circuits. But for large data-words, the area and critical path delay achieved by logic synthesis do not even match relaxed constraints.

The following sections describe an easy-to-use automated approach to generate highly optimized parallel CRC modules.

2. Parallel CRC generation methodology

The algorithm establishing parallel equations is based on the serial CRC operation shown in figure 1. For a parallel calculation of a CRC checksum, only the data word and the initial value of the CRC register are needed.

The operation of the algorithm is illustrated in figure 2. Each bit in the checksum is traced back in the serial calculation steps shown in figure 1. Every time a xor operation affects the bit or the initial value is reached, the particular coefficients are added to the equation. This operation is repeated until all coefficients that were written to the equation are belonging to the initial value or the data word.

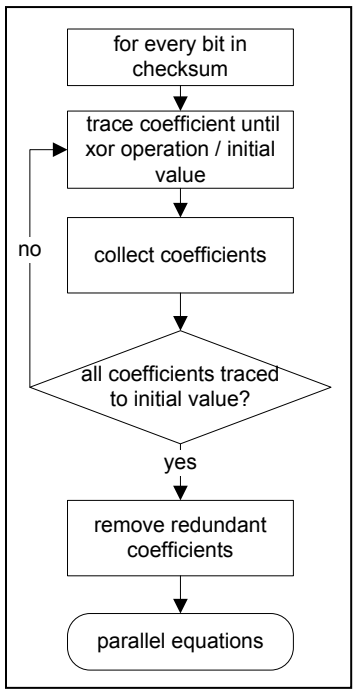

Figure 2: Algorithm delivering parallel CRC equations

The resulting equations contain multiple redundant coefficients that are eliminated according to the logical axioms for exclusive-or operations ($c_1 \oplus c_1 = 0$) to achieve optimized synthesis results.

Instead of executing these optimizations at the end of the generation, it is implemented in the iterative operation described above in order to reduce run time and memory usage of the tool.

[1] Hardware Description Language (HDL) Cyclic Redundancy Check (CRC)

3. Parallel CRC netlist generator

The algorithm mentioned above is implemented in a Perl based tool to provide an automated generation of parallel HDL CRC modules. The user interface is based on shell commands to enable an automated execution.

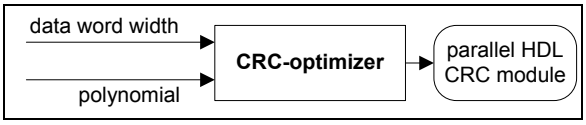

Figure 3: CRC-optimizer tool

The tool delivers the desired parallel CRC Verilog netlist and takes the data word width and the polynomial as input (see figure 3). Additionally, a testcase for verification against a reference serial CRC module is generated to enable the user to assure proper functionality.
The run time of the generation flow increases exponentially with the data word width (see table 1). For most common needed data word widths (<1024) the run time suits the expectations. Possible requests for larger data words would justify further optimization of the algorithm.

data word width [bit]	run time [s]	levels of logic (input to register)	area (normalized)
8	0	4	1,0
16	0	3	1,4
32	0	4	2,7
64	1	4	4,5
128	1	6	7,2
256	4	6	12,0
512	22	11	24,7
1024	164	8	45,4
2048	1296	15	85,9
4096	10053	16	141,0

Table 1: Tool run time and synthesis results

Synthesis of the generated parallel CRC modules shows that the critical path length increases slowly when increasing the data word width (see table 1 and figure 4).
In addition, the slowly growing area also demonstrates the quality of optimization reached by the tool.
Even when reaching very large data word widths (such as 4096 bit), the critical path does not exceed reasonable lengths (16 levels of logic).

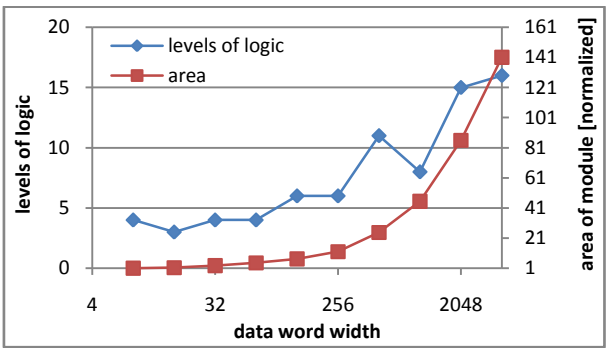

Figure 4: Area and logic levels of parallel CRC modules

4. Integration into design flow

The CRC-optimizer tool integrates into the design flow parallel to the RTL coding phase. The generated CRC HDL netlists are instantiated in the RTL description of the design. The HDL netlists are realized in Verilog. The tool is part of the standard iteration flow that is located prior to synthesis and physical implementation (see figure 5).
Due to the option of executing the tool automatically, no severe changes to the standard flow are necessary.
Furthermore the easy-to-use interface enables very flexible and efficient applications.

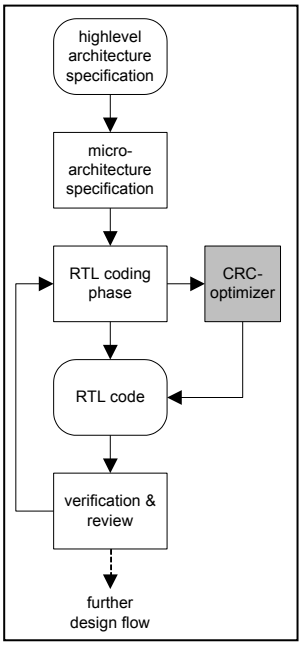

Figure 5: Location of CRC-optimizer in design

5. Conclusion

In this paper a methodology for the automatic generation of parallel CRC Verilog netlists is described.
The CRC-optimizer tool provides optimized parallel CRC Verilog netlists that show very good characteristics regarding synthesis.
The tool was seamlessly integrated into the existing standard design flow.
Future plans include an improvement of the algorithm to achieve linear growth of the run time reliant on the data word width.

6. References

[1] H. Yi, J. Song, C. Park, and S. Park, *"Parallel crc logic optimization algorithm for high speed communication systems", ICCS 2006*, pages 1–5, October 2006.

[2] M. Sprachmann, *"Automatic generation of parallel crc circuits"*, *IEEE Design & Test of Computers,* pages 108–114, May-June 2001.

[3] T.-B. Pei, C. Zukowski, *"High-speed parallel CRC circuits in VLSI"*, *IEEE Transactions on Communications*, pages 653–657, April 1992.

[4] G. Campobello, G. Patane, M. Russo, *"Parallel CRC realization"*, *IEEE Transactions on Computers*, pages 1312 – 1319, Oct. 2003.

Acknowledgments

The author would like to thank Norbert Förster for his knowledgeable teaching and Stefan Kuhnert for his expertise support during synthesis.