CODE GENERATION TOOLS FOR HARDWARE IMPLEMENTATION OF FEC CIRCUITS

Christian Schuler
GMD FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
email: schuler@fokus.gmd.de

ABSTRACT
During the last decade the use of hardware description languages (HDL) became an important method for the specification and the design of VLSI circuits. In particular the combination with logic synthesis tools allows a significant acceleration of the design cycle. Further advantages are the use of top down design methodology, reuse of code and retargeting to different semiconductor technologies. Nevertheless there is a lot of effort spend to develop tools and languages, which allow an higher level of abstraction and by this a further improvement of the design efficiency.

This paper focuses on application specific code generation tools, which might be used in many areas of digital hardware design. The concept is validated on the example of parameterized Forward Error Correction (FEC) circuits, which are used for error detection or correction in most communication systems. The code generators are able to generate the synthesizable VHDL (Very High Speed Integrated Circuits Hardware Description Language) description of complete FEC encoders and decoders within less than a second.

1. INTRODUCTION
In the recent years there are several approaches to improve the efficiency of HDL based hardware design. In particular in the area of Application Specified Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FGPA) the tendency is to allow the designer to work on a higher level of abstraction to reduce the development times and improve the reusability. In this context the use of code generators for standard hardware functions, which are used in many different hardware designs, is a promising concept. The code generators described in this paper can be considered as an example for this design approach. They allow to reduce the development time for an error correcting decoder from several month to seconds.

FEC codes are used in many datacomm and telecomm applications as well as in storage systems like CD or DVD to improve the transmission efficiency or the system capacity. In this paper they are taken as an example for a complex part of a VLSI circuit, which usually requires a high engineering effort. The code generator concept might as well be applied to other areas of hardware design, e.g. DMA controllers, bus interfaces, display drivers or serial interfaces. Code generators are a consequent step towards the system-on-a-chip philosophy, which is encouraged by the rapidly increasing integration density. Figure 2 explains the location of the code generator in the chip level design flow.

2. FEC IMPLEMENTATION
The FEC code generation tools, which are implemented in ‘C’ for the Unix operating system, allow to generate most types of binary FEC encoders (e.g. Hamming, CRC and BCH codes) and non-binary RS codes for symbol sizes up to 10 bit. The main part of the paper is dedicated to the RS decoder, which is the most complex component generated by the tools with up to 10000 gates. The decoding of Reed-Solomon FEC codes is performed by a set of mathematical algorithms based on Galois field arithmetic. A mathematical description of the decoding procedure is given in [1,2,7]. A flow chart of the used decoding procedure is shown in figure 2, where the Forney algorithm for computation of the error values is repeated for each error location. The decoding is based on the Euclid algorithm, which is known to be more suited for hardware implementation than the Berlekamp-Massey algorithm [11,12,13].

All calculations work on polynomials with coefficients in the Galois field corresponding to the symbol size of the code, e.g. GF(256) for 8 bit symbols. At the
end of the decoding procedure either the errors in the received codeword should have been corrected or the decoder should detect a decoding error. Nevertheless in every FEC code there is a certain probability for an undetected error, which is inherent to the code itself and cannot be influenced by a specific software or hardware implementation.

![Reed-Solomon Decoder Flow Chart](image)

Fig. 2: Reed-Solomon Decoder Flow Chart

Figure 3 shows a simplified block diagram with the main VHDL components. The syndrome is denoted as $S$, the normalized error locator polynomial as $\sigma$, and the error evaluator polynomial $\Omega$. RS decoding requires various arithmetic algorithms to process Galois field elements, e.g. adders, constant and general multipliers, inverters, dividers and power functions. The complex algorithms are implemented as VHDL functions to keep the source code readable.

![RS Decoder Hardware Modules](image)

Fig. 3: RS Decoder Hardware Modules

The most hardware extensive logic functions, in particular the general Galois field multipliers and dividers, are implemented as shared resources, which can be used by several modules in subsequent computation steps. This allows to save chip area, but requires tristate buses and reliable scheduling of the steps to avoid collisions. The exact pattern of the decoding procedure depends on the error structure, i.e. the interaction of the state machines is non-deterministic.

The VHDL implementation makes extensive use of parameterization in form of constants, which depend on the RS code parameters. All common constants are concentrated within a single VHDL package, which can be accessed by the 13 specific hardware modules during hardware synthesis. This technique is an important advantage of HDL design over schematic design. It allows to limit the FEC code dependent part of the RS decoder to about 50 lines of source code from a total of over 2700 lines. The current version of the RS decoder requires on chip RAM cells, which are available on most of today’s ASIC and FPGA technologies. The RAM architecture consumes less chip area than register/flip-flop based designs. On the other hand the synthesis of the RAM cells for the specific target technology is not standardized, and it might require source code modifications to map to a different technology.

An important issue is the test of the RS decoder on VHDL simulation level. For this purpose a VHDL testbench is automatically generated, which allows to simulate the design before and after synthesis. The implementation of a testbench for an error correcting code can be simplified, if the information word is assumed to contain only zeros. An arbitrary error pattern can be generated by simply adding ones in the received symbol sequence. In the case of correct decoding the decoded word should again contain only zeros, which can be easily verified.

In addition to correct decoding of errors at various positions the testbench has to ensure the correct recognition of uncorrectable codewords, which are caused by many errors. With the generated VHDL testbench several hundred test cases can be simulated, which gives a fairly good error coverage.

### 3. DESIGN RESULTS

The designs generated by the VHDL code generators have been extensively analyzed to get a picture on the required chip area and the achievable system clock speed. All results are based on the Exemplar Logic Leonardo V4.2 synthesis tool with XILINX XC4000EX as target technology.

In figure 4 two different RS decoder architectures with a symbol size of $m = 8$ have been compared. They differ in the implementation of the Euclid algorithm, which can be generated as register based or as RAM based design. The register based architecture can be used with every target technology, while the RAM based design requires on chip RAM cells, which are assigned during the optimization process of the synthesis tool.
The x axis in figure 4 shows the error correction capability $t$, which is equal to number of symbols, which can be corrected. The correction capability depends on the number of parity symbols. The graph indicates, that the size of the decoder grows linear with $t$. The absolute size of the codeword $n$ has only an influence on the size of the code word RAM and can be neglected for larger $t$. For example a RS(255,223) decoder and a RS(64,32) decoder are of approximately the same size. The chip area in equivalent logic gates can be estimated for the RAM based FPGA architecture by

\[ A = 1000 + 67t \]  \hspace{1cm} (1)

and for the register based architecture by

\[ A = 1000 + 190t \]  \hspace{1cm} (2)

As expected the register based design is considerably larger for higher error correction capabilities. The other main parameter of interest in the development of high speed hardware is the critical path through the logic elements of the circuit, which relates to the maximum system clock speed.

For the RAM based RS decoders this parameter has been analyzed in figure 5. The critical path is almost constant for all values of $t$, which indicates a good pipeline structure. The critical factor for the system timing is the delay of the general GF multiplier, which is the most complex part of combinatorial logic in this architecture. The total decoder delay depends on the used code and the number of symbol errors. For a RS(45,39) decoder running with 20 MHz clock it was 14 microseconds or 280 clock cycles.

The dependence of area and delay of the RS decoder on the RS symbol size $m$ has been analyzed for symbol sizes between 4 and 10. For larger symbols the hardware design becomes impractical large, mainly due to the size of the GF multipliers.

To obtain comparable results, the absolute rate $k/n$ of the codes for the different symbol sizes was approximately constant. Table 1 lists the code parameters in detail.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$k$</th>
<th>$t$</th>
<th>$rate$ $k/n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>13</td>
<td>1</td>
<td>0.867</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>27</td>
<td>2</td>
<td>0.871</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>55</td>
<td>4</td>
<td>0.873</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>111</td>
<td>8</td>
<td>0.874</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
<td>223</td>
<td>16</td>
<td>0.875</td>
</tr>
<tr>
<td>9</td>
<td>511</td>
<td>447</td>
<td>32</td>
<td>0.875</td>
</tr>
<tr>
<td>10</td>
<td>1023</td>
<td>895</td>
<td>64</td>
<td>0.875</td>
</tr>
</tbody>
</table>

Table 1: 7/8 codes with different symbol size

For a fair comparison it should be considered, that the code performance differs with the symbol sizes as well. Figure 6 shows the residual bit error rate BER versus the channel BER for a channel with exponentially distributed errors. While all analyzed codes in this example have the same rate, i.e. the same amount of redundancy, two areas can be distinguished. For channel BER higher than $5 \times 10^{-3}$ the codes with smaller symbol size perform better, while for smaller BERs the codes with larger symbol size lead to much smaller residual error rates. For the selection of a suitable FEC coding the expected BER must be taken into account. The additional hardware resources needed for larger symbol sizes will not in any case lead to a performance improvement.

Figure 7 shows the dependency between the size of the RS code symbols and the required chip area. The results are given in form of the gate count and the number of XILINX Configurable Logic Blocks CLB, which represent the smallest logic cells of XILINX FPGA circuits.
4. CONCLUSIONS

The use of code generators seems to be a promising approach to reduce the design cycle. If the HDL code is well structured, it can be embedded in a parameterized code generation tool with a comparatively small development effort. The optimum number of parameters has to be determined by the design experiences. Possible parameters for the presented FEC circuits are the input/output bus size, automatic wait state generation for large logic functions, selection of different RAM architectures or implementation in other HDLs. Another interesting issue is the incorporation of typical synthesis constraints into the code generator. The higher abstraction level of a code generator allows architectural decisions with much more significant effects than on the RTL logic synthesis level. For example the use of additional multipliers in the Euclid algorithm of the RS decoder would allow to increase the system throughput, but the would result in a much higher area cost as well. With code generators the design engineer will be able to find the optimum trade-off for the particular application by try and error methods. The parametrization, which becomes possible through the use of VHDL, allows a high gain in flexibility and a reduction of development times. Another application for code generators is the possibility to generate benchmarks for synthesis tools or complete design flows. The automatically generated designs of variable size and complexity can be used to evaluate processing time, delay and area performance of different hardware design tools.

5. REFERENCES