divgen: a divider unit generator
Romain Michard, Arnaud Tisserand and Nicolas Veyrat-Charvillon
Arénaire project (CNRS – ENS Lyon – INRIA – UCBL)
LIP, ENS Lyon. 46 allée d’Italie. F–69364 LYON Cedex 07, FRANCE

ABSTRACT
In this work, we present a tool that generates division hardware units. This generator, called divgen, allows a fast and wide space exploration in circuits that involve division operations. The generator produces synthesizable VHDL descriptions of optimized division units for various algorithms and parameters. The results of our generator have been demonstrated on FPGA circuits.

Keywords: Computer arithmetic, division, SRT, circuit generator, FPGA

1. INTRODUCTION
Hardware support for the evaluation of complicated computations has become an important demand in many current applications. Division is frequently used in scientific computing, digital signal processing and computer graphics. A wide variety of algorithms and parameters for the implementation of division is possible. The choice of a division algorithm and the numerous internal parameters is not a straightforward problem.

divgen is a fixed-point divider unit generator. Given a set of parameters and options, it automatically generates an optimized VHDL description of the corresponding divider operator for some specific implementation target constraints. divgen is an ongoing research project released under the GPL license. More details and new versions can be found in

This article is organized as follows. Section 2 presents definitions and notations. Section 3 details the algorithms used, the parameters and options available in the program. Section 4 presents some internal technical details. Performance results on Xilinx FPGAs are given in Section 5.

2. NOTATIONS
The notations proposed in will be used. \( x \) stands for the dividend, \( d \) for the divisor, \( q \) for the quotient and \( \text{rem} \) for the final remainder. The division operation is defined as:

\[
x = q d + \text{rem}.
\]

The division algorithms use radix-\( r \) quotient representation (a power of 2). In the following, we will present the case of division of integer values, but our generator handles fixed-point values in arbitrary formats.

Each iteration of the digit-recurrence algorithms used for division computes a new radix-\( r \) digit of the quotient, noted \( q_j \) at iteration (or step) \( j \). The value \( w[j] \) denotes the partial remainder (or residual) obtained at step \( j \). The final quotient is an \( n \)-bit value. The quotient after \( j \) steps is:

\[
q[j] = \sum_{i=1}^{j} q_i r^{j-i},
\]

Further author information:
Romain Michard: E-mail: Romain.Michard@ens-lyon.fr
Arnaud Tisserand: E-mail: Arnaud.Tisserand@ens-lyon.fr
Nicolas Veyrat-Charvillon: E-mail: Nicolas.Veyrat-Charvillon@ens-lyon.fr
and the final quotient is $q = q[n]$, with

$$n = \left\lceil \frac{q\text{size}}{\log_2 r} \right\rceil.$$

The supported number systems are: unsigned format, sign-and-magnitude format, 2’s complement format, carry-save format.

When using a redundant representation of numbers, a particular number system is specified by the radix $r$ and the largest digit used $\alpha$, e.g., 4–2 (resp. 4–3) denotes the radix-4 representation using digits from the set $\{-2, -1, 0, 1, 2\}$ (resp. $\{-3, -2, -1, 0, 1, 2, 3\}$). Only symmetrical digit sets are considered in this work, and with $\frac{r}{2} \leq \alpha < r$.

RCA (ripple-carry adder) denotes a sequential carry-propagate adder, CPA (carry-propagate adder) denotes a standard adder and CSA (carry-save adder) a constant time redundant adder. The RCAs are usually found in FPGA implementations where fast carry-lines are present, and CSA are used in ASIC implementations.

When referring to the options, typewriter font is used for the option name, italic font for a numerical value, sans-serif font for a choice to make between some possible values. For example when setting the dividend number system and quotient digits radix for SRT division:

```
dividend_representation [ unsigned | sign-magnitude | 2s_complement ]

q_radix [ integer ]
```

### 3. SUPPORTED ALGORITHMS AND PARAMETERS

#### 3.1. General divider architecture

After execution of the generator, the output file contains a VHDL description of a set of subcomponents and a top level component called "Divider" (presented in Figure 1 and Figure 2, and its port list in Table 1).

![Diagram of Divider](image)

**Figure 1.** The general architecture of the generated divider

<table>
<thead>
<tr>
<th>port name</th>
<th>input/output</th>
<th>size</th>
<th>activity</th>
<th>use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>input</td>
<td>integer</td>
<td>N/A</td>
<td>parallel input for dividend</td>
</tr>
<tr>
<td>$d$</td>
<td>input</td>
<td>integer</td>
<td>N/A</td>
<td>parallel input for divisor</td>
</tr>
<tr>
<td>start</td>
<td>input</td>
<td>1</td>
<td>high</td>
<td>starts the computation</td>
</tr>
<tr>
<td>clock</td>
<td>input</td>
<td>1</td>
<td>N/A</td>
<td>clock</td>
</tr>
<tr>
<td>reset</td>
<td>input</td>
<td>1</td>
<td>high</td>
<td>resets the operator</td>
</tr>
<tr>
<td>$q$</td>
<td>output</td>
<td>integer</td>
<td>N/A</td>
<td>parallel output for quotient</td>
</tr>
<tr>
<td>overflow</td>
<td>output</td>
<td>1</td>
<td>high</td>
<td>overflow leading to a false output</td>
</tr>
<tr>
<td>complete</td>
<td>output</td>
<td>1</td>
<td>high</td>
<td>computation completion</td>
</tr>
</tbody>
</table>

**Table 1.** Divider component port list
Some constraints exist on the input signals: reset should be zero before the start signal is raised. Start should be synchronous and active for exactly one clock period. Dividend and divisor inputs have to be stable from the raise of start until the computation is complete (complete signal raised). When the computation has ended, quotient, overflow and complete stay stable until the next reset signal.

3.2. General options

$x$, $d$ and $q$ are integers or fixed-point numbers (floating-point numbers are not handled in this version), characterized by their size and their representation (unsigned, sign-and-magnitude and 2’s complement). The carry-save format is only possible for the internal residual. The supported algorithms have been tested for sizes between 8 and 64 bits.

Three division algorithms can be selected so far: restoring, nonrestoring and SRT. All are digit-recurrence algorithms, that produce a fixed number of quotient bits at each iteration (one quotient digit of one or more bits), most significant digit first. Those algorithms are based on shift and addition operations and some additional logical elements. It is well known that the choice of the radix and quotient digit set influences the overall latency of the algorithm. Roughly, increasing the radix decreases the number of iterations required for the same quotient precision. Unfortunately, as the radix increases, every iteration becomes more complicated and the overall latency may not be reduced as expected. Additionally, it becomes impractical to generate the required divisor multiples for higher radices (costly multiplication in the iteration).

The basic two digit-recurrence algorithms are the restoring and the nonrestoring algorithms. Those algorithms only rely on very simple radix-2 iterations, i.e., one bit of the quotient is produced at each iteration. The third available algorithm is the SRT division, which may rely on a higher radix. This algorithm needs less iterations to compute a quotient but at the cost of a more complicated iteration. Many variations of these algorithms exist, and it is very difficult to decide which one is best suited with respect to some constraints.

The computation of digit-recurrence algorithms is based on a similar recurrence step:

\[
\begin{align*}
    w[0] &= x \\
    q[0] &= 0 \\
    q_j &= Sel(w[j-1], d) \\
    w[j] &= rw[j-1] - q_jd \\
    q[j] &= \sum_{i=1}^{j} q_ir^{-1}
\end{align*}
\]

They differ in the radix $r$, the values that $q_j$ can attain and in the definition of the quotient digit selection function $Sel$. Another difference between the division algorithms is that the restoring and nonrestoring algorithms...
can take any inputs, and will be able to compute, although some inputs may cause an overflow, whereas the SRT division needs a normalized divisor, i.e., we must have:

\[ 2^{d_{\text{size}}-1-\delta \text{ulp}(d)} \leq |d| < 2^{d_{\text{size}}-\delta \text{ulp}(d)}, \]

with \( \delta = \begin{cases} 0 & \text{if } d \text{ is unsigned} \\ 1 & \text{otherwise}. \end{cases} \)

These are the options common to all algorithms and the values they can take:

- **dividend_representation** [unsigned | sign-magnitude | 2s_complement] specifies the number system of the dividend
- **divisor_representation** [unsigned | sign-magnitude | 2s_complement] specifies the number system of the divisor
- **quotient_representation** [unsigned | sign-magnitude | 2s_complement] specifies the number system of the quotient
- **dividend_size** [integer] specifies the size of the dividend in number of bits
- **divisor_size** [integer] specifies the size of the divisor in number of bits
- **quotient_size** [integer] specifies the size of the quotient in number of bits
- **component_algorithm** [restoring | nonrestoring | SRT] the chosen division algorithm

Below, an example of configuration file is presented in case of a 32-bit restoring divider:

```plaintext
x_representation unsigned
d_representation unsigned
q_representation unsigned
x_size 32
d_size 32
q_size 32
algorithm restoring
```

### 3.3. Algorithms and specific options

#### 3.3.1. Restoring division

This algorithm is very similar to the "paper-and-pencil" method learnt in school, using radix \( r = 2 \) with quotient digit set \( \{0, 1\} \). At each iteration, the algorithm subtracts the divisor \( d \) from the previous partial remainder \( w[j-1] \) multiplied by \( r \). The quotient digit selection function is derived by comparing the residual at each step to 0. If the result of the subtraction is positive (or null), the new digit of the quotient is one, otherwise it is set to zero and the previous value of the partial remainder should be restored (by adding \( d \) to the result). Usually, this restoration is not performed using an addition, but by selecting the value \( 2w[j-1] \) instead of \( w[j] \), which replaces an addition in the critical path by a multiplexer. This nonperforming division, presented in Figure 3, is the one generated by `divgen`.
3.3.2. Nonrestoring division
Nonrestoring division is an improved version of the restoring method in the sense that it completely avoids the restoration step by combining restoration additions with the next recurrence, thus, reducing the overall latency. Moreover, it uses the quotient digit set \{-1, 1\} to perform directly the recurrence with the selection function:

\[ q_j = Sel(w[j - 1]) = \begin{cases} +1 & \text{if } w[j - 1] \geq 0, \\ -1 & \text{if } w[j - 1] < 0. \end{cases} \]

The nonrestoring division algorithm presented in Figure 4 allows the same small amount of computations at each iteration. The conversion of the quotient from the digit set \{-1, 1\} to the chosen output format can be done either by a subtraction or on the fly by using a simple algorithm. We chose that last solution.

3.3.3. SRT division
The main problem in division schemes is to determine the new quotient digit \( q_j \) at each iteration. The SRT methods use a redundant quotient digit set in order to speed up the computation of \( q_j \). More precisely, comparing the partial remainder to all the divisor multiples can offset or possibly diminish the performance gained by increasing the radix. To avoid this, redundancy is introduced in the set of possible quotient digits \( |q_i| \leq \alpha \). This allows to simplify the quotient digit selection function in the sense that comparisons are now done with limited precision constants only (a constant time operation). This is done using an estimation of the divisor \( d \) and the partial remainder \( w[j] \).

The hardware implementation is done using a table (or any structure that acts like a table: PLAs, LUTs) addressed by a few of the most significant bits of \( d \) and \( w[j] \). Some techniques can be used in order to reduce the size and the critical path of the divider, which are described below.

The specific options for the SRT algorithm are:
1 if $\alpha = r - 1$ then
2 $w[0] \leftarrow x/2$
3 else $w[0] \leftarrow x/4$
4 for $j$ from 1 to $n$ do
5 $q_j = \text{Sel}(w[j - 1], d)$
6 $w[j] \leftarrow rw[j - 1] - qjd$
7 end for

Figure 5. Radix-$r$ SRT division algorithm

- quotient_radix [power_of_2] 
  the value $r$
- quotient_max_digit [integer] 
  the value $\alpha$
- partial_remainder_representation [2s_complement | carry-save] 
  use 2's complement for FPGAs and carry-save for ASICs
- step_adder [RCA | CSA] 
  use RCA for FPGAs and CSA for ASICs
- #guard_bits [integer] 
  additional bits due to the conversion of the estimation of the redundant residual to a non-redundant form
- SRT_table_folding [yes | no] 
  folding method proposed in\textsuperscript{9} to decrease the selection table area
- gray_encoding [yes | no] 
  encoding of the selection table output for preserving the redundancy whenever possible

All these options impact the selection function and the whole divider’s performances in several ways (see\textsuperscript{9} for instance). We will show some examples of situations where they may be useful.

To illustrate the selection function behavior, we use P-D diagrams, which are $rw[j]$ versus $d$ diagrams of the valid quotient digit values. The theoretical bounds where the choice of quotient digit $q_j = k$ is valid are the solid plotted lines originating from $(0, 0)$ (they are the $U_k$ and $L_k$ from\textsuperscript{3}).

In the following, we will characterize \textit{a priori} the complexity of a selection table by its number of terms (i.e., the number of boxes on the P-D diagram).

- Impact of $r$ and $\alpha$:
  By increasing the radix $r$, one decreases the number of iterations. But this also means that more multiples of $d$ have to be computed for the product $q_jd$ and the selection function is more complicated since each iteration must cancel $\log_2 r$ digits of the partial remainder (there are more possible digits for $q_j$ since $\alpha \geq \frac{r}{2}$).

Increasing $\alpha$ for a constant radix $r$ increases the number of required multiples of $d$. But this also increases the overlapping of quotient digit selection areas then this leads to a simpler (and smaller) selection function.

Figure 6 gives a general idea of how we compute the multiples of $d$. We have $m_0 \in \{-2, -1, 0, 1, 2\}$, $m_1 \in \{-4, 0, 4\}$, $m_2 \in \{-8, 0, 8\}$.

Figure 7 presents the P-D diagrams for several possible internal representations of the quotient. Table (a) is defined by 272 terms, (b) by 28 and (c) by 400.
Figure 6. Structures computing \( w[j] = rw[j - 1] + q_jd \) for \( \alpha = 2 \) (left) and \( 10 < \alpha \leq 14 \) (right)

- **Short CPA based estimation of** \( w[j] \) **and guard bits impact:**
  The number of digits used for the estimation \( \hat{w}[j] \) of the residual is an important parameter in the selection function. It has to be large enough to ensure a converging algorithm but is should not be too high because the selection table is addressed by \( \hat{w}[j] \).

  When using a redundant number system for the residual, the accuracy of the estimation is less than using a non redundant representation. To avoid the use of a selection table with a large number of address bits, one can use a short CPA to get an estimation of the partial remainder \( \hat{w}[j] \). It clearly leads to a tradeoff between the additional time required by this short CPA and the time required to decode the address in the selection table. The parameter \#guard_bits \( \) allows to tune the length of the short CPA (see\( ^9 \) for more details).

  For example, in an SRT 8–6 division, the selection table is defined by 686 terms when using a 2’s complement partial remainder. If the partial remainder is represented in carry-save representation, for \( g = 0 \) we have 2702 terms in the table, 1358 if \( g = 1 \) and only 686 for \( g = 4 \).

- **Table folding impact:**
  Due to the symmetrical digit set being used, the selection table is almost symmetrical with respect to \( w[j] = 0 \). “Almost” because the truncation error done while estimating the partial remainder is not symmetrical for 2’s complement and carry-save number system (it would be for a borrow-save one). We can try to fold the table over the \( w[j] = 0 \) to try to diminish its size. This is almost always possible, sometimes at the cost of a better estimation made by increasing the number of bits from \( d \) and \( w[j] \) used to drive the selection table, which can offset and even cancel the gain made by folding. The result is often better than expected, because instead of using \( \hat{w}[j] \) as the table input, which needs a carry-propagation addition in order to be computed, we use \( \hat{w}'[j] \) where \( \hat{w}'[j] = \hat{w}[j] \oplus \text{sign}(\hat{w}[j]) \). Doing this for a 2’s complement representation, while being faster than taking the actual absolute value of \( \hat{w}[j] \), introduces an error that compensates the truncation error made before.

  For example, a selection table for SRT 4–3 (resp. 4–2) with a 2’s complement partial remainder contains
28 (resp. 272) terms. If we try to fold it using $|\hat{w}[j]|$ as an input, it needs (a) 52 (resp 269) terms ! Using the XOR-based conversion described above, the table contains only (b) 14 (resp. 136) terms, we halved the table. The corresponding tables are presented in Fig 8.

4. TECHNICAL DETAILS

divgen consists in a set of C++ classes (about 5000 lines in version 0.1) used to create a description of an operator which can be converted to VHDL.

All components are derived from the element class which implements the basic component attributes and methods. An instance of element may contain several other elements linked with each other and with the ports.
of the surrounding element, possibly included in iterative or conditional structures. A component is instantiated by setting his ports and generic parameters. Wires and sets of wires are related to the signal class, for which some operators have been overloaded in order to allow an easy manipulation, similar to the VHDL one.

As divgen is executed, it first reads the configuration file and creates the desired component along with all its subcomponents in memory. Then the main component is parsed and translated to VHDL. All signal assignations and components instantiations are translated automatically to VHDL.

```
element
    | logic
    |   | and2
    |   | or2
    |   | ...
    | sequential
    |   | register
    | arithmetic
    |   | RCA
    |   | divider
    |       | restoring
    |       | nonrestoring
    |       | SRT
```

### 5. PERFORMANCE RESULTS

Some implementation results of our generator are presented below. The circuit targets are FPGAs from Xilinx (Virtex-E 300, 3072 slices, medium speedgrade).

The overall performances of the generated dividers are presented for several quotient word lengths and the several division algorithms and parameters. Figure 9 reports the clock period results, Figure 10 reports those for the total computation time, Figure 11 reports those for the area and Figure 12 reports the results for area-time product.

In Table 2 and 3 are reported the impact of various synthesis and place and route efforts on the divider performances (in case of a 32-bit divided by 16-bit values for a SRT 4–3 algorithm). Table 4 presents some examples of parameters impact on the performances.
Figure 10. Total computation time obtained for several dividers and quotient word lengths

Figure 11. Circuit area obtained for several dividers and quotient word lengths

Table 2. Synthesis and Place&Route effort impact for high speed target

<table>
<thead>
<tr>
<th>Synthesis effort</th>
<th>Speed Normal</th>
<th>Speed High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>Medium</td>
</tr>
<tr>
<td>slices</td>
<td>193</td>
<td>193</td>
</tr>
<tr>
<td>period [ns]</td>
<td>18.6</td>
<td>18.5</td>
</tr>
</tbody>
</table>

Table 3. Synthesis and Place&Route effort impact for small area target

<table>
<thead>
<tr>
<th>Synthesis effort</th>
<th>Area Normal</th>
<th>Area High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>Medium</td>
</tr>
<tr>
<td>slices</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>period [ns]</td>
<td>26.4</td>
<td>28.2</td>
</tr>
</tbody>
</table>
Figure 12. Area-time product obtained for several dividers and quotient word lengths

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Partial Remainder</th>
<th>Folding</th>
<th>Gray Encoding</th>
<th>#Guard Bits</th>
<th>Slices</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>60</td>
<td>23.8</td>
</tr>
<tr>
<td>4-2</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>111</td>
<td>28.1</td>
</tr>
<tr>
<td>4-3</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>100</td>
<td>26.4</td>
</tr>
<tr>
<td>8-4</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-6</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>220</td>
<td>33.4</td>
</tr>
<tr>
<td>8-7</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>158</td>
<td>34.6</td>
</tr>
<tr>
<td>4-3</td>
<td>2s</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td>104</td>
<td>31.3</td>
</tr>
<tr>
<td>4-3</td>
<td>2s</td>
<td>Y</td>
<td>N</td>
<td>0</td>
<td>101</td>
<td>24.3</td>
</tr>
<tr>
<td>4-3</td>
<td>2s</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>104</td>
<td>31.6</td>
</tr>
<tr>
<td>4-2</td>
<td>2s</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td>117</td>
<td>28.1</td>
</tr>
<tr>
<td>4-2</td>
<td>2s</td>
<td>Y</td>
<td>N</td>
<td>0</td>
<td>95</td>
<td>29</td>
</tr>
<tr>
<td>4-2</td>
<td>2s</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>101</td>
<td>27.4</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>108</td>
<td>18.7</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>Y</td>
<td>N</td>
<td>0</td>
<td>108</td>
<td>20.2</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>N</td>
<td>N</td>
<td>1</td>
<td>108</td>
<td>20.1</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>N</td>
<td>N</td>
<td>2</td>
<td>108</td>
<td>22.2</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>N</td>
<td>N</td>
<td>3</td>
<td>108</td>
<td>23.1</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>N</td>
<td>N</td>
<td>4</td>
<td>110</td>
<td>24</td>
</tr>
<tr>
<td>4-3</td>
<td>cs</td>
<td>N</td>
<td>N</td>
<td>5</td>
<td>111</td>
<td>21.6</td>
</tr>
<tr>
<td>8-4</td>
<td>2s</td>
<td>N</td>
<td>N</td>
<td>0</td>
<td>378</td>
<td>41.4</td>
</tr>
<tr>
<td>8-7</td>
<td>2s</td>
<td>Y</td>
<td>N</td>
<td>0</td>
<td>128</td>
<td>31.1</td>
</tr>
<tr>
<td>8-7</td>
<td>2s</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>129</td>
<td>31.1</td>
</tr>
</tbody>
</table>

Table 4. Examples of parameters impact
Some numerical tests have been performed in order to validate the behavior of the generated dividers using VHDL simulations. Special cases values and random values have been tested for all the supported algorithms, various parameters, representations and sizes.

6. CONCLUSION & FUTURE PROSPECTS

In this paper, we have presented a tool for the automatic generation of optimized divider hardware units. This program, called divgen, generates optimized VHDL descriptions of division operators for various algorithms and parameters with respect to several implementation constraints. The behavior and the performances of the generated dividers have been tested on FPGA circuits from Xilinx.

In the future, we aim at extending this generator to the square root function and other parameters and targets constraints.

REFERENCES