REPRINT OF IRISH SIGNALS AND SYSTEMS CONFERENCE 2000, DUBLIN, IRELAND, PP. 368-375, JUNE 2000.

# Constant Divider Structures of the Form $2^n \pm 1$

A.Th. Schwarzbacher[1,3,++], M. Brutscheck[1,2], O. Schwingel[1,2] and J.B. Foley[3]

[1]Dublin Institute of Technology, Dublin, Ireland
[1]Fachhochschule Merseburg, Germany
[3]Trinity College, Dublin, Ireland

***Abstract***

Constant operations are one of the key elements in digital signal processing (DSP) systems. The implementation of such constant operations is usually performed through the use of standard modules which are not optimised for one particular function. The primary reason for the use these standard modules is that in commercial projects time to market is always one of the main objectives. This paper addresses the particular problem of the implementation of constant divider structures of the form $2^n \pm 1$. As will be shown alternative algorithms can have significantly improved performance over standard modules.

***Keywords***: Standard Binary Division, Constant Division Algorithm, High-Level CMOS Design.

## 1 Theoretical Background

This paper was originally inspired by a problem encountered during the implementation of a high performance image processing system. In this system a divider by three was required. However, the initial approach of using a standard module revealed an insufficient timing behaviour of the module. Therefore, alternative implementations were investigated using various division algorithms that have been suggested in the past. In the following sections six approaches will be presented before their features in respect to a silicon implementation are discussed.

### 1.1 A Constant Division Algorithm by Petry and Srinivasan

The constant division algorithm as proposed by Petry and Srinivasan [1] is an iterative algorithm, which was developed for division of the form $2^n \pm 1$. The computation of the quotient can be described as

$$Q = \sum_{i=1}^{m} (h)^{i-1} 2^{-in} A \tag{1}$$

As illustrated in Figure 1, it is possible to describe (1) as an array of grouped dividend bits. Therefore, the equation can be rewritten as shown in (2). Using this equation only shift and addition operations are required to solve the quotient $Q$, where $Q$ consists of quotient bits $q_i$ and the remainder $R$.

++ Author to whom correspondence may be directed: Andreas Schwarzbacher, aschwarzbacher@electronics.dit.ie
Dublin Institute of Technology, Dept. of Electronics and Communication Engineering, Dublin8, Ireland.

$$
\begin{array}{llllllll}
 & h^0 * & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 . & a_0 \\
+ & h^1 * & & a_{m-1} & a_{m-2} & \cdots & a_2 . & a_1 \\
+ & h^2 * & & & a_{m-1} & \cdots & a_3 . & a_2 \\
 & \vdots & & & & \cdots & \vdots & \vdots \\
+ & h^{m-2} * & & & & & a_{m-1} . & a_{m-2} \\
+ & h^{m-1} * & & & & & . & a_{m-1} \\
\hline
 & & q_{m-2} & q_{m-3} & q_{m-4} & \cdots & q_0 . & R
\end{array}
$$

*Figure 1: Iterative Division by $2^n+1$ and $2^n-1$*

Equation (1) was analysed and optimised for the task of carrying out the division by three. It can be seen from (1) that the use of $h = 1$ is preferable to perform this division to avoid an alternating function. The reason for this is that alternating functions require more area and also have a higher power consumption than non-alternating functions [2]. The dividend $A$ is independent of $i$ and the equation can be simplified to

$$Q = A\sum_{i=1}^{m} 2^{-in} \tag{2}$$

In this equation $n$ is equal to two because the divisor is three while $m$ only depends on the bitwidth of the dividend $A$. For a 10 bit wide input bus $m$ is equal to 10. Hence, the whole equation can be rewritten as

$$Q = A\sum_{i=1}^{10} 2^{-2i} \tag{3}$$

If a deviation of 1 bit is acceptable, this constant division algorithm, as presented in [2], achieves a higher accuracy as required. An investigation of (3) shows that only four terms instead of 10 are required. For this reason the maximum deviation can be written as

$$\textit{maximum deviation} = \frac{256}{2^8} = 1\,\text{bit}$$

As described above a deviation of one bit occurs for a quotient range of 0 to 255. The optimised formula can be generalised as

$$Q = A\sum_{i=1}^{4} 2^{-2i} = A*(2^{-2} + 2^{-4} + 2^{-6} + 2^{-8}) \tag{4}$$

In (4) it can be seen that the division by three is reducible to a multiplication by a value which is a close approximation of 1/3. As illustrated in Figure 2 only shift steps and one addition operation is required for an implementation.

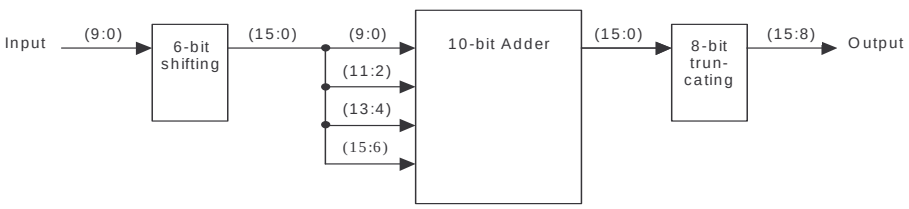


*Figure 2: Implementation of the Optimised Algorithm*

### 1.2 A Residue Number System based Division Architecture for Constant Divisors

The Residue Number System (RNS) [3] can be used for constant divisors of the form $2^n \pm 1$. A dividend $A$ can be written as follows.

$$A = a_{N-1}a_{N-2}....a_2a_1a_0 \tag{5}$$

In (5), the dividend is a positive $N$ bit number where the number of bits $m = N/n$. The dividend $A$ can be written as a number of $m$ digits where each digit $A_i$ consists of $n$ bits. For a hardware implementation it may be necessary to append zero bits to the most significant bits of $A$ to present $N=n*m$ as follows.

$$A = A_{m-1}A_{m-2}....A_1A_0 = A_{m-1}2^{n(m-1)} + ....+ A_2 2^{2n} + A_1 2^n + A_0 2^0 \tag{6}$$

Taking $A=D*Q+R$ and replacing $A$ in (6) the equation can now be written as

$$\frac{A}{D} = \frac{A_0 2^0 + A_1 2^n + A_2 2^{2n} + ....+ A_{m-1}2^{n(m-1)}}{2^n - 1} \tag{7}$$

Since $2^{2n} = 2^n * 2^n$ and $2^n = (2^n - 1) + 1$, (7) can be simplified by using $2^n - 1$ as shown in (8).

$$\frac{A}{D} = \frac{\sum_{i=0}^{m-1} A_i}{2^n - 1} + \sum_{i=1}^{m-1} A_i + \sum_{i=2}^{m-1} A_i 2^n + \sum_{i=3}^{m-1} A_i 2^{2n} + ....+ A_{m-1}2^{(m-2)n} \tag{8}$$

The components of the remainder $R$ and the quotient $Q$ can be written as follows.

$$R = \frac{\sum_{i=0}^{m-1} A_i}{2^n - 1} \tag{9}$$

$$Q = \sum_{i=1}^{m-1} A_i + \sum_{i=2}^{m-1} A_i 2^n + \sum_{i=3}^{m-1} A_i 2^{2n} + ....+ A_{m-1}2^{(m-2)n} \tag{10}$$

An analysis shows that it is useful to focus on the form $2^n - 1$ to avoid the implementation of an alternating function. For an implementation the input must be analysed and split into different bits. This analysis is necessary to calculate $m$.

$$m = \frac{N}{n} = 5 \tag{11}$$

After determining $m$ (10) can be written as follows

$$Q = \sum_{i=1}^{4} A_i + \sum_{i=2}^{4} A_i * 2^2 + \sum_{i=3}^{4} A_i * 2^4 + \sum_{i=4}^{4} A_i * 2^6 \tag{12}$$

As seen in (12), it is not necessary to compute the digit $A_0$. Only the digits $A_1$-$A_4$ are required to compute $Q$. Figure 3 presents the block diagram of the VLSI implementation of the RNS algorithm. Here, it can be seen that the division by three was reduced to a combination of shift and add operations. Furthermore, Figure 3 shows that the input bus was split into smaller buses each containing 2 bits.

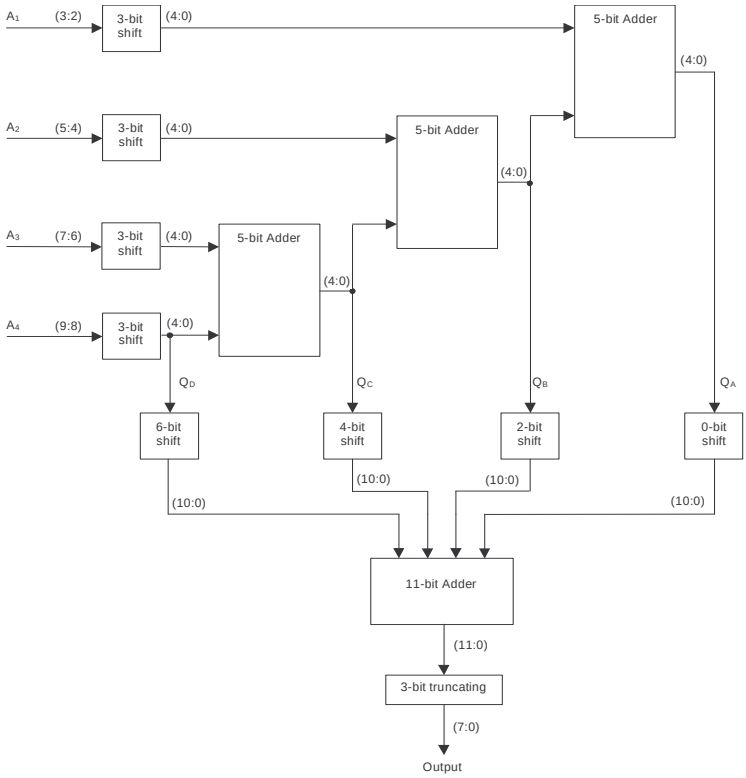


*Figure 3: Implementation of the RNS based Algorithm*

### 1.3 A fast Constant Division Routine by Shuo-Yen Robert-Li

A constant-division routine was presented by Shuo-Yen Robert-Li in [5]. By using *Euler`s function* and *Fermat`s Little theorem* this constant division algorithm can be generalised as a multiplication with the reciprocal number of the divisor *D*. Equation (13) shows that the multiplication is an approximation to 1/*D*.

$$\frac{1}{D} = Binary(0.0b_1b_2...b_{n-1})\prod_{i=0}^{\infty}(1+2^{-n2^i}) \tag{13}$$

It can be seen that (13) contains a multiplication by $Binary(0.0b_1b_2...b_{n-1})$ followed by multiplication by factors of the form $1+2^{-m}$. First, an odd constant divisor *D* has to be found which will be used in the algorithm. Then, *n* has to be defined as a next smaller integer value to *D*. In this way, the first mathematical term can be expressed as follows.

$$Binary(0.0b_1b_2...b_{n-1}) = \frac{2^n - 1}{D} \tag{14}$$

As shown in (14), it is necessary to calculate the bitwidth of the binary expression depending on *D* and *n*. The form $1+2^{-m}$ can be expressed as an approximation of a infinite product series as shown in (13). Using the equations above it is not necessary to solve all products of the infinite series. This is because in an implementation of this algorithm the divisor *D* has to be approximated as described in Section 1.1. This is required to minimise the logic. The only difference to the algorithm be Petry and Srinivasan is the expression 1/*D*. In the case of the constant division routine by Shuo-Yen Robert Li, a multiplication by $Binary(0.0b_1b_2...b_{n-1})$ followed by multiplication by factors of the form $1+2^{-m}$. However, in respect to a hardware implementation this algorithm will have no difference to the algorithm proposed by Petry and Srinivasan and is therefore not further discussed.

### 1.4 Lookup Tables

Due to the limited input range, the use of a Lookup Table (LUT) was also considered. LUT's are simple storage devices where all possible output values are precomputed and stored. Therefore, no computations are necessary which results in a fast design. Another advantage of LUT's is that the output always corresponds to the most accurate possible value. However, LUT's are limited by the amount of input and output values because all possible output values have to be stored.

### 1.5 The Standard Binary Divider

Binary division is basically a procedure to determine how many times the divisor *D* divides the dividend *A* thus resulting in the quotient *Q*. At each step in the process the divisor *D* either divides *A* into a group of bits or it does not. The divisor divides a group of bits when the divisor has a value less than or equal to the value of those bits. Therefore, the quotient is either 1 or 0. This is repeated as long as the bitwidth *m* is smaller than the number of input bits.

There are two possibilities to implement the Standard Binary Divider (SBD). First, the binary division can be implemented using a standard module. This is often done in industry because time to market is one of the dominant factors in any project. However, VLSI design environments can be used to optimise these standard modules. This leads as the results will show to a significantly improved performance. In the remainder of this paper the divider obtained through this optimisation is called SBD(opt).

## 2 Results

This chapter presents the results of the implementation of the previously described divider structures. All dividers were implemented under the following constraints. The input has a width of 10 bits and the output a width of 8 bits. All designs were implemented into hardware using the European Silicon Structures 0.7µ CMOS technology. The algorithmic description of the designs was made in *VHDL* and the designs were synthesised without constraints using the *Synopsys Design Environment*. The power consumption, timing behaviour, required area, accuracy and error deviation were measured. The results are presented in this chapter.

### 2.1 Timing Behaviour

The first feature of the divider structures to be investigated was the operational speed of the different implementations. Figure 4 compares the timing of the divider circuits.

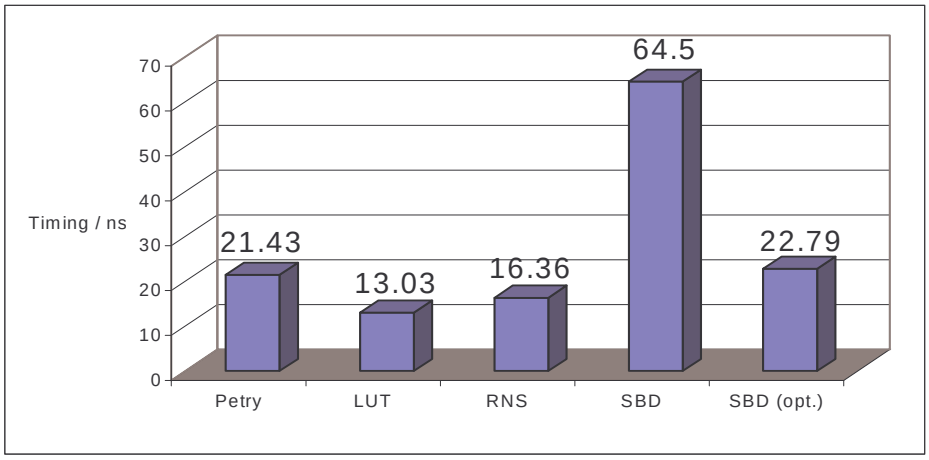


*Figure 4: The Timing Behaviour*

The timing behaviour of the different divider algorithms is shown in Figure 4. As illustrated in Figure 4 the SBD has the poorest performance in respect to operational speed with a time of 64.5ns. The reason for this is that 10 comparator stages are required in this design. All other implementations compute the result at least three times faster. Using a simple structure and no computation processes the Lookup Table has the fastest timing behaviour with 13.03ns. The RNS based algorithm requires approximately 25% and the SBD(opt) requires only 35% of the time of the slowest structure.

### 2.2 Area Requirements

The Figure below shows the required area in $\mu m^2$ of the different divider designs.

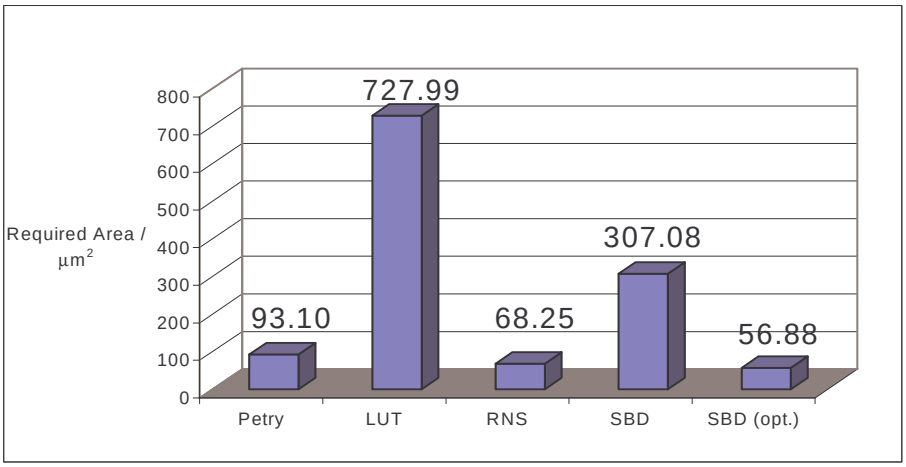


*Figure 5: The Required Area*

It can be seen that the Lookup Table requires the most area. This is an anticipated result because all possible output values must be precomputed and stored, which limits the LUT to small designs. As illustrated in Figure 5, the SBD needs 42.2% of the area of the LUT which has an area of $728mm^2$. The algorithm by Petry requires 12.8% of the area of the Lookup Table. Using the RNS-based algorithm it is possible to perform the division by three with an area of 9.4% of that of the LUT. The best solution towards the required area is the SBD(opt) because it only needs 7.8% of the area of the LUT.

### 2.3 Power Consumption

Figure 6 shows the power consumption for the different implemented algorithms. The power consumption was measured using PowerCount [7] at a frequency of 33MHz and a supply voltage of 5V.

In comparison to the algorithm by Petry and Srinivasan and the RNS based algorithm, the SBD requires approximately 95% more power. The SBD consumes with 30.53µW the most power. The SBD(opt) consumes only 7% of the power of the SBD. The LUT requires with 6.59µW or 21% of the power of the SBD. Therefore, the SBD and the LUT are not useful to implement because of the large power consumption.

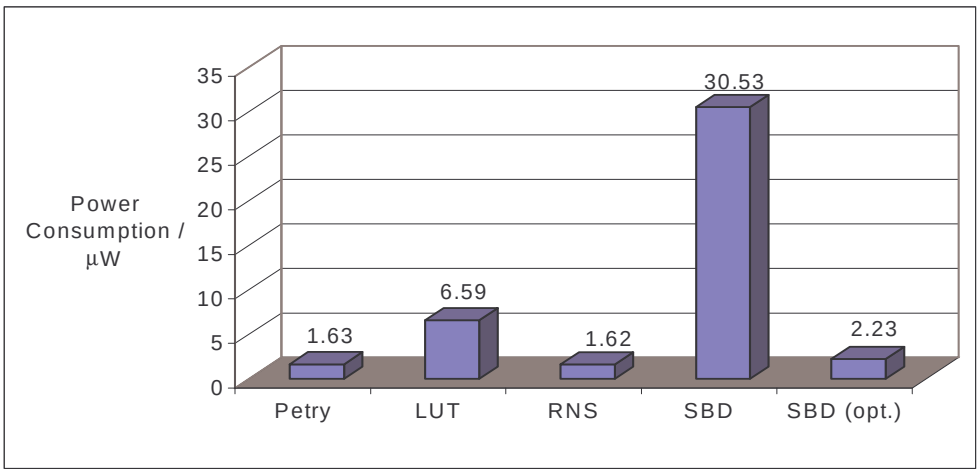


*Figure 6: The Power Consumption*

## 2.4 Accuracy and Error Deviation

Figure 7 shows the accuracy and error deviation of the six investigated structures in comparison to the exact division by three (A/3).

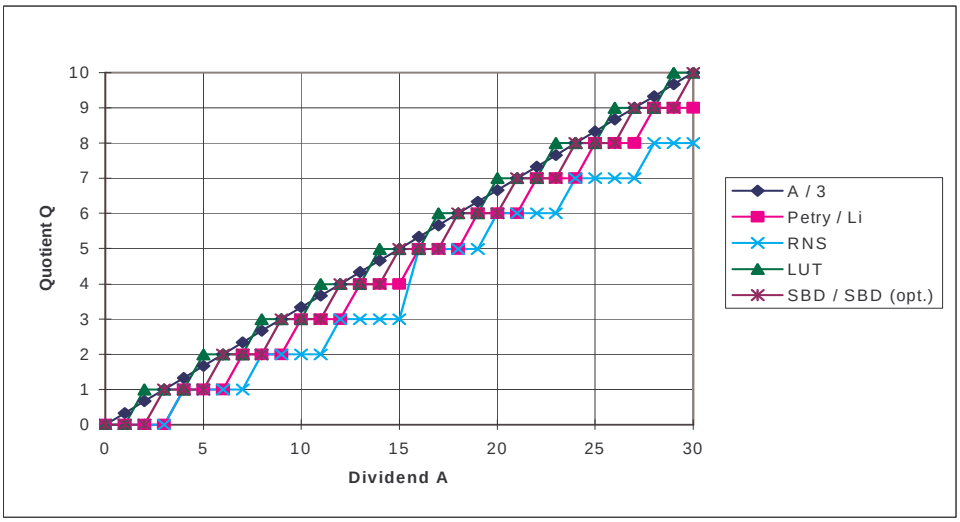


*Figure 7: The Accuracy and Error Deviation*

The quotient of the LUT has the best possible accuracy possible. This is because each output value is precalculated and rounded. Therefore, the LUT can be seen as the benchmark in respect to accuracy. It also can be seen that the constant-division by Petry and Srinivasan has the same accuracy and deviation as the algorithm of Shuo-Yen Robert Li. The real deviation is at least –1 bit in comparison to the usual division by three. SBD and the SBD(opt) have exactly the same accuracy as truncated division by three. The computed negative deviations are 0, 1/3 and –2/3. Using the RNS based algorithm the deviation is different on each step of the computation. The presented structure contains a maximum deviation of 4 bit. Therefore, this algorithm is not suitable to perform the division by three.

## 3 Conclusions

This paper has presented a detailed analysis of the implementation of a divider by three. Six designs were investigated. It has been shown that algorithms which are optimised for division by constants achieved the best results. In terms of power consumption, the results were all quite similar. If area and speed are also taken into consideration, the RNS based implementation appears preferable. However, as has been shown, this algorithm is less accurate than the other implementations presented. Therefore, the algorithm proposed by Petry has the best overall performance.

The LUT is, as expected, the fastest and largest implementation. However, as the implementation of Petry is capable of operating at frequencies of up to 45 MHz with smaller area and power consumption, the LUT should only be used if speed if the design constraint. This also demonstrates the advantages of using optimised designs, as opposed to standard modules. Standard modules are frequently used in industry to shorten the design cycle. However, as was shown in this paper, the use of optimised designs results in an improvement in area, speed and power consumption of the IC by a factor of three.

## Acknowledgement

Mr. Brutscheck and Mr. Schwingel would like to thank Dr. Monica Liebe for her support in preparing the research visit to the Dublin Institute of Technology.

Furthermore, Mr. Schwarzbacher would like to acknowledge the funding received through the IRCARUS2 (DG-XII) initiative.

## References

[1] A.Th. Schwarzbacher, P.A. Comiskey and J.B. Foley, "Reduction of the power consumption at the algorithmic level of CMOS circuits," *Electronic Systems and Devices Conference*, Bruno, Czech Republic, pp. 5-8, June 1998.

[2] P. Srinivasan, F.E. Petry, “Constant-Division Algorithms,“ *IEE Proc.-Comput. Digit. Techn.*, Vol. 141, No. 6, November 1994.

[3] B. Al-Besher, A.Bouridane, A. S. Ashur, “An RNS-based Division Architecture for Constant Divisors of the Form $2^n$+1 and $2^n$-1,“ *Irish Signals & Systems Conference*, June 1997.

[4] V. P. Nelson, H. T. Nagle, B. D. Carroll, J. D. Irwin, “Digital Logic Circuit Analysis & Design,” Prentice Hall Englewood Cliffs, 1995.

[5] S.-Y. R. Li, “Fast Constant Division Routines,” *IEEE Transactions on Computers*, Vol. C-34, No. 9, September 1985.

[6] F. E. Petry, P. Srinivasan, “Division Techniques for Integers of the Form $2^n$+1 and $2^n$-1,“ *Int. J. Electronics*, Vol. 74, No. 5, 1993.

[7] A.Th. Schwarzbacher, P.A. Comiskey, J.B. Foley, J. Rodrigoues and F. Klemenz, "Rapid estimation of the active node capacitance of VLSI circuits," *Proc. of Programmable Devices and Systems 2000*, Ostrava, Czech Republic, February 2000.