Division and conquer

de Koning, W.

Award date:
2013

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Supervisor
dr.ir. Rob R. Hoogerwoord

Date
November 28, 2013

Division and Conquer

Master's Thesis

Wouter de Koning

Abstract

Integer division is an important arithmetic operation on microprocessors. To derive integer division algorithms we present an unconvential approach: a derivation technique in a calculational style, that guarantees that the derived algorithms are correct. Four different algorithms are derived using this method: restoring division, non-restoring divsion, radix-4 division and division by multiplication. We translate these to descriptions into combinatorial circuits, expressed in Verilog code. Then the circuits are compiled on a Spartan-3 Generation FPGA. At the end, we compare the propagation delays and area requirements for these circuits. We show that the division by multiplication is much faster than the other methods, however it only works for 18 bit integers.

Contents

Chapter 1

Introduction

Computer programs become more and more complex. To run these programs, microprocessors carry out massive mathematical computations mainly based on four basic arithmetic operations: addition, subtraction, multiplication and division. According to Soderquist and Leeser [9], division is the slowest operation compared to the other three. While division is not used very often, ignoring division in the microprocessor can result in system performance degradation and therefore division must be taken into account as well [6].

Implementing integer divisions with remainder on hardware is a classic problem that was already studied in the early days of the digital era [8,12]. Meanwhile, there are many approaches to this problem, which can be divided into three classes: digit recurrence or subtractive methods, functional iteration and table based functions [10]. These algorithms differ in propagation delay and area requirements, two relevant performance characteristics when implementing algorithms in hardware. A survey by Oberman and Flynn [7] presents an overview of the main division algorithms.

In this thesis we derive four algorithms for dividing integers using an unconventional approach. This approach follows the methods of Hoogerwoord [4], which were teached during the course Programming by Calculation at the TU/e. By using this approach, we prove the correctness of the solution during the derivations. The notation together with the predicate calculus gives us a tool to derive programs without introducing errors. This is useful to help with preventing errors such as the infamous Pentium FDIV bug in 1994, an error in the division part of the Intel P5 microprocessor floating point unit (FPU). This error was caused by a few missing entries in a lookup table part of the chips circuitry [1, 5].

Starting from a specification, we derive a functional program involving integer operations in a calculational style for each algorithm. We transform these functional programs such that they represent the numbers in binary digits. These solutions are translated to descriptions of combinatorial circuits, expressed in Verilog code [11]. Each of these circuits were tested on a Spartan-3 Generation Field Programmable Gate Array (FPGA) and have obtained reliable estimates of its propagation delay and area requirement.

1.1 Division algorithms

We derive four different algorithms for dividing integers. The division algorithms should compute for a (possible negative) integer A and positive integer B the quotient q ($= A \underline{\text{div}}\ B$) and remainder r ($= A \underline{\text{mod}}\ B$) specified as:

$$(0) \qquad q, r\text{:} \quad A = q{*}B + r \quad \wedge \quad 0 \leq r < B$$

For the first algorithm, we obtain from this specification an algorithm by formulating the solution for $2{*}A + b$ and B (with $b \in \{0, 1\}$) recursively in terms of the solution for A and B. This algorithm is called *restoring division* as explained in many books discussing computer arithmetic, such as [2, 3, 8].

For the second algorithm we weaken specification (0) to

$$(1) \qquad q, r\text{:} \quad A = q{*}B + r \quad \wedge \quad -B \leq r < B$$

The solution we obtain is *non-restoring division*. The weaker specification admits more manipulative freedom, such that we can derive a faster algorithm that requires less area when implemented. In the end, we only need a simple correction to get the actual quotient and remainder as specified by equation (0).

The third solution, *radix-4 division*, is similar to restoring division, but carried out in the quaternary number system. The algorithm is obtained by formulating the solution for $4{*}A + d$ and B (with $0 \leq d \leq 3$) recursively in terms of the solution for A and B. The resulting algorithm is faster but also larger than restoring division.

The fourth algorithm is *division by multiplication*. The relation $A/B = A * (1/B)$ can be exploited by obtaining an approximation for $1/B$ from a lookup table: Thus, a single multiplication suffices to obtain an approximation for A/B. This is attractive because the FPGA contains fast multipliers as standard components. Of course, we need a simple correction at the end to obtain the answer to $A \underline{\text{div}}\ B$ from the approximation. For this algorithm we have derived two variants: one with a smaller table, but with more propagation delay and one with a larger table and less propagation delay.

1.2 Overview

We introduce the notation in this thesis in Chapter 2; several useful definitions for lists are given and from those some useful properties are derived. Restoring division, non-restoring division, radix-4 division and division by multiplication are derived in Chapters 3 to 6. Each of those chapters can be read independently, without reading the previous ones. In Chapter 7 the results of the propagation delays and the area requirements are presented and discussed.

Chapter 2

Notation

2.1 Lists

A *list* is a linearly ordered set of elements, usually with the same type. For the remainder of this document, we use only finite lists: lists with a finite amount of elements. The empty list is denoted with the symbol [ ]. A list with only element b is written down as [b]. We denote the list composed from element b and list s as $b \triangleright s$. This operator is defined as follows:

Definition 0:

$$\begin{array}{ll} (b \triangleright s) \cdot 0 & = b \\ (b \triangleright s) \cdot (n+1) & = s \cdot n \end{array}$$

We use the shorthand notation s_n for $s \cdot n$.

Now we can define several operators on lists. The *count* operator # returns the amount of elements in a list. The operator is recursively defined as:

Definition 1:

$$\begin{array}{ll} \#[\,] & = 0 \\ \#(b \triangleright s) & = \#s + 1 \end{array}$$

From two lists s and t with length m and n we can construct a new list of length $m + n$ using the *concatenation* operator ($+\!\!+$). The result of $s +\!\!+ t$ is the list containing first all the elements of s and then all the elements of t. This is recursively defined as:

Definition 2:

$$\begin{array}{ll} [\,] +\!\!+ t & = t \\ (b \triangleright s) +\!\!+ t & = b \triangleright (s +\!\!+ t) \end{array}$$

The *drop* operator ($\lfloor$) removes the first n elements from a list s, for $0 \leq n \leq \#s$. This is defined by:

Definition 3:

$$\begin{array}{lcl} s \lfloor 0 & = & s \\ (b \triangleright s) \lfloor (n{+}1) & = & s \lfloor n \end{array}$$

The *take* operator ($\lceil$) takes the first n elements from a list s, for $0 \leq n \leq \#s$, thus dropping the last $\#s - n$ elements.

Definition 4:

$$\begin{array}{lcl} s \lceil 0 & = & [\,] \\ (b \triangleright s) \lceil (n{+}1) & = & b \triangleright s \lceil n \end{array}$$

From the previous definitions 2, 3 and 4 we prove several useful properties such as:

Property 5: for any list s and natural number n: $0 \leq n \leq \#s$:

$$s \;=\; (s \lceil n) +\!\!+ (s \lfloor n)$$

Property 6: for any list s and t:

$$(s +\!\!+ t) \lfloor \#s \;=\; t$$

and,

Property 7: for any list $s \neq [\,]$ and natural number n: $0 < n \leq \#s$:

$$s \lfloor (n{-}1) \;=\; s_{n-1} \triangleright s \lfloor n$$

Property 5 is proven using induction on n. We first prove the property for $n = 0$:

$$\begin{array}{ll} & (s \lceil 0) +\!\!+ (s \lfloor 0) \\ = & \{ \text{ definitions 3: } \lfloor \text{ and 4: } \lceil \text{ } \} \\ & [\,] +\!\!+ s \\ = & \{ \text{ definition 2: } +\!\!+ \text{ } \} \\ & s \end{array}$$

Now we prove the property for $1 \leq n \leq \#s$. The list s has the shape $b \triangleright t$, for some b and a list t, with length $n - 1$. Using this we derive:

$$\begin{array}{ll} & ((b \triangleright t) \lceil (n{+}1)) +\!\!+ ((b \triangleright t) \lfloor (n{+}1)) \\ = & \{ \text{ definitions 3: } \lfloor \text{ and 4: } \lceil \text{ } \} \\ & (b \triangleright (t \lceil n)) +\!\!+ (t \lfloor n) \\ = & \{ \text{ definition 2: } +\!\!+ \text{ } \} \end{array}$$

$$
\begin{array}{ll}
 & b \triangleright ((t \lceil n) +\!\!+ (t \lfloor n)) \\
= & \{ \text{ induction hypothesis } \} \\
 & b \triangleright t
\end{array}
$$

Property 6 is proven using induction on list s. We first prove the property for $s = [\,]$ and any list t:

$$
\begin{array}{ll}
 & ([\,] +\!\!+ t) \lfloor \#[\,] \\
= & \{ \#[\,] = 0 \} \{ \text{ definition 2: } +\!\!+ \} \\
 & t \lfloor 0 \\
= & \{ \text{ definition 3: } \lfloor \ \} \\
 & t
\end{array}
$$

With the induction hypothesis, we assume $(s +\!\!+ t) \lfloor \#s = t$. Now we prove the case $s = b \triangleright s$, for any b and list $s \neq [\,]$ as follows:

$$
\begin{array}{ll}
 & ((b \triangleright s) +\!\!+ t) \lfloor (\#s{+}1) \\
= & \{ \text{ definition 2: } +\!\!+ \} \\
 & (b \triangleright (s +\!\!+ t)) \lfloor (\#s{+}1) \\
= & \{ \text{ definition 3: } \lfloor \} \\
 & (s +\!\!+ t) \lfloor \#s \\
= & \{ \text{ induction hypothesis } \} \\
 & t
\end{array}
$$

By induction, property 6 holds for any list s and t.

Property 7 is derived as follows:

$$
\begin{array}{ll}
 & s \lfloor (n{-}1) \\
= & \{ \text{ choose } s = ss +\!\!+ [\, s_{n-1} \,] +\!\!+ st, \text{ with } \#ss = n{-}1 \ \} \\
 & (\ ss +\!\!+ [\, s_{n-1} \,] +\!\!+ st\ ) \lfloor (n{-}1) \\
= & \{ \text{ use property 6 } \} \\
 & [\, s_{n-1} \,] +\!\!+ st \\
= & \{ \#(ss +\!\!+ [s_{n-1}]) = n; \text{ use property 6 } \} \\
 & [\, s_{n-1} \,] +\!\!+ (\ (\ ss +\!\!+ [\, s_{n-1} \,] +\!\!+ st\ ) \lfloor n\ ) \\
= & \{ \ ss +\!\!+ [\, s_{n-1} \,] +\!\!+ st = s \ \} \\
 & [\, s_{n-1} \,] +\!\!+ s \lfloor n \\
= & \{ \text{ definition 2: } +\!\!+ \ (\ 2 \times\ ) \} \\
 & s_{n-1} \triangleright s \lfloor n
\end{array}
$$

2.2 Binary lists

To represent a natural number as a *binary list*, a list with elements in $\{0,1\}$, we use the abstraction function $v2 : \mathcal{L}2 \to Int$, where $\mathcal{L}2$ is the binary list. An element in $\{0,1\}$ is called a bit. The binary list in function $v2$, where the last bit is the most significant bit, is called the *binary representation* of a natural number. Function $v2$ is declared as follows:

Declaration 8:

$$\begin{array}{lll} & v2{\cdot}[\ ] & = 0 \\ \& & v2{\cdot}(\ b \triangleright s\ ) & = 2 * v2{\cdot}s + b \qquad \text{, for all } s \text{ in } \mathcal{L}2 \text{ and } b \text{ in } \{0,1\} \end{array}$$

To represent a signed integer as a binary list, we use the *two's complement representation*. The two's complement representation is similar to the binary representation with one additional bit, the sign bit. If the sign bit is 0, then the value is equal to the binary value of the first N bits, where $N+1$ is the total number of the bits. If the sign bit is 1, then 2^N is subtracted from the binary value of the first N bits. The abstraction function $tc : \mathcal{L}2 \to Int$ to represent the two's complement value for a given binary list, is declared as follows:

Declaration 9:

$$\begin{array}{lll} tc{\cdot}[\ b\ ] & = -b & \text{, for } b \text{ in } \{0,1\} \\ tc{\cdot}(\ b \triangleright s\ ) & = 2 * tc{\cdot}s + b & \text{, for all } s \text{ in } \mathcal{L}2 - \{\ [\ ]\ \} \text{ and } b \text{ in } \{0,1\} \end{array}$$

To remove the most significant bit in the binary representation from the a list, we use the following properties:

Property 10: for any list s and b in {0,1}:

$$v2{\cdot}(\ s +\!\!+ [b]\ ) \ = \ v2{\cdot}s + 2^{\#s} * b$$

and, in two's complement representation:

Property 11: for any list s and b in {0,1}:

$$tc\cdot(\ s +\!\!+ [b]\ ) \ = \ v2{\cdot}s - 2^{\#s} * b$$

These properties are derived as follows using induction on s. For property 10 we first prove the base case, where s is the empty list:

$$\begin{array}{ll} & v2{\cdot}(\ [\ ] +\!\!+ [b]\ ) \\ = & \{\ \text{definition 2: } +\!\!+\ \} \\ & v2{\cdot}[b] \\ = & \{\ \text{declaration 8: } v2\ (2\times)\ \} \end{array}$$

$$
\begin{array}{ll}
 & b \\
= & \{ \text{ algebra } \} \\
 & 0 + 2^0 * b \\
= & \{ \text{ declaration 8: } v2 \ \} \\
 & v2 \cdot [\ ] + 2^{\#[\ ]} * b
\end{array}
$$

With the induction hypothesis, we assume $v2 \cdot (s +\!\!+ [b]) = v2 \cdot s + 2^{\#s} * b$ is satisfied for some list s and bit b. We derive for $c \triangleright s$:

$$
\begin{array}{ll}
 & v2 \cdot (\ (c \triangleright s) +\!\!+ [b]\ ) \\
= & \{ \text{ definition 2: } +\!\!+ \ \} \\
 & v2 \cdot (\ c \triangleright (s +\!\!+ [b])\ ) \\
= & \{ \text{ declaration 8: } v2 \ \} \\
 & 2 * v2 \cdot (s +\!\!+ [b]) + c \\
= & \{ \text{ induction hypothesis } \} \\
 & 2 * (v2 \cdot s + 2^{\#s} * b) + c \\
= & \{ \text{ algebra } \} \\
 & 2 * v2 \cdot s + c + 2^{\#s+1} * b \\
= & \{ \text{ declaration 8: } v2 \ \} \\
 & v2 \cdot (c \triangleright s) + 2^{\#(c \triangleright s)} * b
\end{array}
$$

By induction, property 10 holds for all s and any bit b. The proof for property 11 is similar to property 10 and therefore it is omitted.

When working with two's complement representations, it is possible to use *sign extension* to increase, or *sign truncation* to decrease the length of a list, without changing the value that the list represents. To extend a list s with most significant bit b, add bits with value b to the end of list s. Similar, to truncate a list s we remove bits with value b from the end of s, as long as the most significant bit after the truncation has value b. More formally:

Property 12: for any list s and b in {0,1}:

$$tc \cdot (\ s +\!\!+ [b]\ ) \quad = \quad tc \cdot (\ s +\!\!+ [b] +\!\!+ [b]\ )$$

Property 12 is derived as follows:

$$
\begin{array}{ll}
 & tc \cdot (\ s +\!\!+ [b]\ ) \\
= & \{\ \text{property 11}\ \} \\
 & v2 \cdot s - 2^{\#s} * b \\
= & \{\ \text{algebra}\ \} \\
 & v2 \cdot s + 2^{\#s} * b - 2^{\#s+1} * b \\
= & \{\ \text{property 10}\ \} \\
 & v2 \cdot (\ s +\!\!+ [b]\ ) - 2^{\#s+1} * b \\
= & \{\ \text{property 11}\ \} \\
 & tc \cdot (\ s +\!\!+ [b] +\!\!+ [b]\ )
\end{array}
$$

Chapter 3

Restoring Division

For dividing two integers, we use the $\underline{\text{div}}$ and $\underline{\text{mod}}$ operators, where $A \mathbin{\underline{\text{div}}} B$ equals the quotient and $A \mathbin{\underline{\text{mod}}} B$ equals the remainder of the division. In this Chapter we derive an algorithm for computing the quotient and the remainder, called *restoring division*.

We are interested in computing the quotient and the remainder of an integer A and a positive integer B, provided $-2^N \leq A < 2^N$ and $1 \leq B < 2^N$; we choose variable B to be constant. We derive a function $f\colon Int \to \langle\, Int\,,\, Int\,\rangle$, that computes the quotient and the remainder, specified by:

Specification 13: for a given constant B: $1 \leq B < 2^N$ and variable a: $-2^N \leq a < 2^N$:

$$f{\cdot}a \quad = \langle\, q\,,\, r\,\rangle \quad \underline{\text{whr}} \quad q, r\colon \;\; a = q{*}B + r \;\; \wedge \;\; 0 \leq r < B \quad \underline{\text{end}}$$

Now $f{\cdot}A$ gives the solution to the quotient and the remainder.

From Specification 13 we derive the declarations for the base cases $f\cdot(-1)$ and $f\cdot 0$ as follows:

$$\begin{array}{ll} & f{\cdot}(\,-1\,) \\ = & \{\ \text{specification 13: } f\ \} \\ & \langle\ q\ ,\ r\ \rangle \quad \underline{\text{whr}} \quad q, r\colon \;\; -1 = q{*}B + r \;\; \wedge \;\; 0 \leq r < B \quad \underline{\text{end}} \\ = & \{\ \text{holds only for } q = -1 \text{ and } r = B - 1\ \} \\ & \langle\ -1\ ,\ B - 1\ \rangle \end{array}$$

and,

$$\begin{array}{ll} & f{\cdot}0 \\ = & \{\ \text{specification 13: } f\ \} \\ & \langle\ q\ ,\ r\ \rangle \quad \underline{\text{whr}} \quad q, r\colon \;\; 0 = q{*}B + r \;\; \wedge \;\; 0 \leq r < B \quad \underline{\text{end}} \\ = & \{\ \text{holds only for } q = 0 \text{ and } r = 0\ \} \\ & \langle\ 0\ ,\ 0\ \rangle \end{array}$$

For the recursive case we assume that q and r satisfy: $a = q{*}B + r$ and $0 \leq r < B$. Now we derive a declaration for the case $2{*}a + b$, for all integers a and bits b, with $b \in \{0, 1\}$:

$$
\begin{array}{ll}
 & 2*a + b \\
= & \{ \text{ assumption } \} \\
 & 2*(\,q*B + r\,) + b \\
= & \{ \text{ algebra } \} \\
 & (\,2*q\,)*B + (\,2*r + b\,) \\
= & \{ \text{ introduce } h = 2*r + b \} \\
 & (\,2*q\,)*B + h
\end{array}
$$

The pair $\langle\ 2{*}q\ ,\ h\ \rangle$ is a solution for $f \cdot (\,2{*}a + b\,)$, provided $0 \leq h < B$.

$$
\begin{array}{ll}
= & \{\ \text{ algebra }\ \} \\
 & (\,2*q + 1\,)*B + h - B
\end{array}
$$

And, the pair $\langle\ 2{*}q + 1\ ,\ h - B\ \rangle$ is a solution for $f \cdot (2*a + b)$, provided $0 \leq h - B < B$. Now, from $0 \leq r < B$ we derive the range of h:

$$
\begin{array}{ll}
 & 0 \ \leq\ r \ <\ B \\
\equiv & \{ \text{ algebra } \} \\
 & 0 \leq r \quad \wedge \quad r \leq B - 1 \\
\equiv & \{ \text{ algebra } \} \\
 & 0 \leq 2*r \quad \wedge \quad 2*r \leq 2*B - 2 \\
\Rightarrow & \{\ 0 \leq b\ \}\ \{\ b \leq 1\ \} \\
 & 0 \leq 2*r + b \quad \wedge \quad 2*r + b \leq 2*B - 1 \\
\equiv & \{ \text{ algebra } \} \\
 & 0 \ \leq\ 2*r + b \ <\ 2*B \\
\equiv & \{ \text{ definition } h \} \\
 & 0 \ \leq\ h \ <\ 2*B
\end{array}
$$

We conclude that $0 \leq h < 2 * B$. This proposition is split in two cases: $0 \leq h < B$, and $B \leq h < 2 * B$. The second case is equal to $0 \leq h - B < B$, thus both cases yields a solution for $f \cdot (\,2{*}a + b\,)$. Combining these cases, we obtain:

$$
\begin{array}{ll}
 & \underline{\text{if}} \quad 0 \leq h \quad\ \ < B \ \ \rightarrow \ \ \langle\ 2*q\ ,\ h\ \rangle \\
 & [] \quad 0 \leq h - B < B \ \ \rightarrow \ \ \langle\ 2*q + 1\ ,\ h - B\ \rangle \\
 & \underline{\text{fi}} \\
= & \{ \text{ algebra, using } 0 \leq h < 2*B \ \}
\end{array}
$$

$$\begin{array}{lll} \underline{\text{if}} & h - B < 0 & \rightarrow \ \langle\ 2*q\ ,\ h\ \rangle \\ [] & h - B \geq 0 & \rightarrow \ \langle\ 2*q+1\ ,\ h-B\ \rangle \\ \underline{\text{fi}} & & \end{array}$$

Now we obtain the following declarations for function f, that computes the quotient and remainder for a given integer A and a divisor B.

Declaration 14:

$$\begin{array}{lll} & f\cdot(-1) & = \ \langle\ -1\ ,\ B-1\ \rangle \\ \& & f\cdot 0 & = \ \langle\ 0\ ,\ 0\ \rangle \end{array}$$

$$\begin{array}{lll} \& & f\cdot(\ 2{*}a+b\ ) & = \ \underline{\text{if}} \quad h - B < 0 \ \rightarrow \ \langle\ 2*q\ ,\ h\ \rangle \\ & & \quad [] \quad h - B \geq 0 \ \rightarrow \ \langle\ 2*q+1\ ,\ h-B\ \rangle \\ & & \quad \underline{\text{fi}} \quad \underline{\text{whr}} \quad h \ = \ 2*r+b \\ & & \qquad\ \ \& \quad \langle\ q\ ,\ r\ \rangle \ = \ f\cdot a \\ & & \qquad \underline{\text{end}} \end{array}$$

3.1 From integer to binary (part 1)

We have derived a function f that computes the quotient and the remainder, following the restoring division method. In this section we implement function f that computes A $\underline{\text{div}}$ B and A $\underline{\text{mod}}$ B in the binary system.

First we represent the value A in binary, list as, with length $N{+}1$, provided $-2^N \leq A < 2^N$. Thus, the list as is the two's complement representation of A. Now we introduce a function, $g1$, that computes for a natural number n, with $0 \leq n \leq N$, the quotient and the remainder for the value of as without the least significant n bits.

The function $g1 : Int \rightarrow \langle\ Int\ ,\ Int\ \rangle$ is specified by:

Specification 15: for $0 \leq n \leq N$, a given constant B: $1 \leq B < 2^N$ and a given list as with $\#as = N{+}1$:

$$g1\cdot n \ = \ f\cdot(\ tc\cdot(\ as \lfloor n\ )\ )$$

For $f\cdot A$ we derive:

$$\begin{array}{ll} & f\cdot A \\ = & \{\ A = tc\cdot as\ \} \\ & f\cdot(\ tc\cdot as\ ) \\ = & \{\ \text{definition 3: } \lfloor\ \} \\ & f\cdot(\ tc\cdot as \lfloor 0\ ) \\ = & \{\ \text{specification 15: } g1\ \} \\ & g1\cdot 0 \end{array}$$

So, $g1{\cdot}0$ implements $f{\cdot}A$.

Now, we derive the declaration for $g1$ from specification 15. For the base case $n = N$, we derive:

$$
\begin{array}{ll}
 & g1{\cdot}N \\
= & \{\ \text{specification 15: } g1\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as \lfloor N\ )\ ) \\
= & \{\ \text{property 7: } \lfloor\ \text{, because } n-1 < N \text{ holds}\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as_N \triangleright as \lfloor (N{+}1)\ )\ ) \\
= & \{\ \#as = N+1\text{; property 6}\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as_N \triangleright [\ ]\ )\ ) \\
= & \{\ \text{definition 0: } \triangleright\ \} \\
 & f{\cdot}(\ tc{\cdot}[\ as_N\ ]\ ) \\
= & \{\ \text{declaration 9: } tc\ \} \\
 & f{\cdot}(\ -as_N\ ) \\
= & \{\ \text{declaration 14: } f\ \}\ \{\ as_N \in \{0,1\}\ \} \\
 & \underline{\text{if}}\quad as_N = 0 \quad\to\quad \langle\ 0\ ,\ 0\ \rangle \\
 & []\quad as_N = 1 \quad\to\quad \langle\ -1\ ,\ B-1\ \rangle \\
 & \underline{\text{fi}}
\end{array}
$$

and, for n: $0 \le n < N$:

$$
\begin{array}{ll}
 & g1{\cdot}n \\
= & \{\ \text{specification 15: } g1\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as \lfloor n\ )\ ) \\
= & \{\ \text{property 7}\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as_n \triangleright as \lfloor (n{+}1)\ )\ ) \\
= & \{\ \text{declaration 9: } tc\ \} \\
 & f{\cdot}(\ 2{*}tc{\cdot}(\ as \lfloor (n{+}1)\ ) + as_n\ ) \\
= & \{\ \text{declaration 14: } f\ \} \\
 & \underline{\text{if}}\quad h{-}B < 0 \quad\to\quad \langle\ 2{*}q\ ,\ h\ \rangle \\
 & []\quad h{-}B \ge 0 \quad\to\quad \langle\ 2{*}q+1\ ,\ h-B\ \rangle \\
 & \underline{\text{fi}}\quad \underline{\text{whr}}\quad h\ =\ 2{*}r + as_n \\
 & \qquad\ \&\quad \langle\ q\ ,\ r\ \rangle\ =\ f{\cdot}(\ tc{\cdot}(\ as \lfloor (n{+}1)\ )\ ) \\
 & \qquad \underline{\text{end}}
\end{array}
$$

From specification of $g1$, we rewrite $f{\cdot}(\ tc{\cdot}(\ as \lfloor (n{+}1)\ )\ )$ to $g1{\cdot}(n{+}1)$. Together with the result for the case $n = N$, we obtain the following declarations for the function $g1$:

Declaration 16:

$$
\begin{array}{lcll}
g1{\cdot}N & = & \underline{\text{if}} & as_N = 1 \;\rightarrow\; \langle\, -1\,,\; B-1\,\rangle \\
 & & [] & as_N = 0 \;\rightarrow\; \langle\, 0\,,\; 0\,\rangle \\
 & & \underline{\text{fi}} & \\
g1{\cdot}n & = & \underline{\text{if}} & h{-}B < 0 \;\rightarrow\; \langle\, 2{*}q\,,\; h\,\rangle \\
 & & [] & h{-}B \geq 0 \;\rightarrow\; \langle\, 2{*}q+1\,,\; h-B\,\rangle \\
 & & \underline{\text{fi}} & \underline{\text{whr}}\;\; h \;=\; 2{*}r + as_n \\
 & & & \;\;\&\;\;\; \langle\, q\,,\; r\,\rangle \;=\; g1{\cdot}(\, n{+}1\,) \\
 & & & \underline{\text{end}}
\end{array}
$$

3.2 From integer to binary (part 2)

We have derived a function that computes A $\underline{\text{div}}$ B and A $\underline{\text{mod}}$ B, where A is represented as a binary list. Here we present function $g2$, that represents B, the quotient and the remainder in the declaration of $g1$ as binary lists.

The remainder r in the declaration of $g1$ satisfies $0 \leq r < 2^N$, hence, we specify the length of rs, the binary list representing r, to be N; thus rs is in binary representation. The quotient q in the base case of $g1$ is either -1 or 0, which is represented by one bit in two's complement representation. In the recursive case, where $n < N$, the quotient is divided by 2 with each increase of n; in binary this is achieved with a bitshift. As a result, we specify the length of the binary list representing q to be $N{-}n+1$.

The function $g2 : \mathit{Int} \rightarrow \langle\, \mathcal{L}2\,,\; \mathcal{L}2\,\rangle$ is specified by:

Specification 17: for $0 \leq n \leq N$, a given list bs: $v2 \cdot bs = B \;\wedge\; \#bs = N$ and a given list as with $\#as = N+1$:

$$
\begin{array}{lcll}
g2{\cdot}n & = & \langle\, qs\,,\, rs\,\rangle & \underline{\text{whr}}\;\; qs, rs{:}\;\; \langle\, tc{\cdot}qs\,,\; v2{\cdot}rs\,\rangle \;=\; g1{\cdot}n \\
 & & & \;\;\wedge\; \#qs = N{-}n{+}1 \;\;\wedge\;\; \#rs = N \\
 & & & \underline{\text{end}}
\end{array}
$$

Following specification 17 we formulate first the quotient and the remainder of declaration 16 for the case $n = N$ in binary. The quotient q is -1 or 0. In two's complement representation with length $N - n + 1 = 1$, the quotient is represented by $[\,1\,]$ or $[\,0\,]$ respectively.

For the remainder, we formulate the values 0 and $B{-}1$ in binary representation with length N. We introduce for these values, respectively, the lists $zero$ and $bs1$, as defined by:

$$
\begin{array}{lcll}
0 & = & v2 \cdot zero & \wedge \;\; \#zero = N \\
B{-}1 & = & v2 \cdot bs1 & \wedge \;\; \#bs1 \;= N
\end{array}
$$

* * *

For the recursive case $0 \leq n < N$, we formulate first the quotient in binary. The quotient $2 * q$ is rewritten as follows:

$$\begin{array}{ll} & 2 * q \\ = & \{\ q = tc{\cdot}qs\ \} \\ & 2 * tc{\cdot}qs \\ = & \{\ \text{declaration 9: } tc\ \} \\ & tc{\cdot}(0 \triangleright qs) \end{array}$$

and $2 * q + 1$:

$$\begin{array}{ll} & 2 * q + 1 \\ = & \{\ q = tc{\cdot}qs\ \} \\ & 2 * tc{\cdot}qs + 1 \\ = & \{\ \text{declaration 9: } tc\ \} \\ & tc{\cdot}(1 \triangleright qs) \end{array}$$

For the remainder, we derive hs, the binary representation of h, as follows:

$$\begin{array}{ll} & h \\ = & \{\ h = 2 * r + as_n\ \} \\ & 2 * r + as_n \\ = & \{\ r = v2{\cdot}rs\ \} \\ & 2 * v2{\cdot}rs + as_n \\ = & \{\ \text{declaration 8: } v2\ \} \\ & v2{\cdot}(as_n \triangleright rs) \end{array}$$

Now hs is defined as $as_n \triangleright rs$.

Furthermore, for the remainder we introduce a list ks, which represents the value $h - B$ in two's complement representation:

$$tc{\cdot}ks \ = \ v2{\cdot}hs - v2{\cdot}bs$$

The length of ks is derived from $0 \leq h < 2 * B$ as follows:

$$\begin{array}{ll} & 0 \ \leq \ h \ < \ 2 * B \\ = & \{\ \text{algebra}\ \} \end{array}$$

$$
\begin{array}{ll}
 & -B \ \leq \ h{-}B \ < \ B \\
= & \{ \text{ definition } ks \ \} \\
 & -B \ \leq \ tc{\cdot}ks \ < \ B \\
\Rightarrow & \{ \ B < 2^N \ \} \\
 & -2^N \ \leq \ tc{\cdot}ks \ < \ 2^N
\end{array}
$$

We conclude that $\#ks = N{+}1$.

With ks, not only the remainder, but also the guards are rewritten:

$$
\begin{array}{ll}
 & h{-}B < 0 \\
\equiv & \{ \ h = v2 \cdot hs \text{ and } B = v2 \cdot bs \ \} \ \{ \text{ definition } ks \ \} \\
 & ks < 0 \\
\equiv & \{ \text{ the sign bit is on position } N \ \} \\
 & ks_N = 1
\end{array}
$$

As a result, the other guard $0 \leq h{-}B$ is rewritten to $ks_N = 0$.

The guards in declaration 16 for $0 \leq n < N$ provides $0 \leq r < B$, hence $0 \leq r < 2^N$. As a result, the remainder rs can be expressed in N bits. Hence, we use sign truncation (property 12), such that $\#rs = N$.

Now, by combining the guards, quotients and remainders, we obtain the following declaration for $g2$:

Declaration 18:

$$
\begin{array}{lll}
g2{\cdot}N & = & \underline{\text{if}} \quad as_N = 1 \ \rightarrow \ \langle \ [\ 1\ ] \ , \ bs1 \ \rangle \\
 & & [] \quad as_N = 0 \ \rightarrow \ \langle \ [\ 0\ ] \ , \ zero \ \rangle \\
 & & \underline{\text{fi}} \\
g2{\cdot}n & = & \underline{\text{if}} \quad ks_N = 1 \ \rightarrow \ \langle \ 0 \ \triangleright \ qs \ , \ hs \lceil N \ \rangle \\
 & & [] \quad ks_N = 0 \ \rightarrow \ \langle \ 1 \ \triangleright \ qs \ , \ ks \lceil N \ \rangle \\
 & & \underline{\text{fi}} \quad \underline{\text{whr}} \ \ hs \ = \ as_n \ \triangleright \ rs \\
 & & \qquad \& \quad ks \ = \ hs - bs \\
 & & \qquad \& \quad \langle \ qs \ , \ rs \ \rangle \ = \ g2{\cdot}(\ n{+}1\ ) \\
 & & \qquad \underline{\text{end}}
\end{array}
$$

3.3 Verilog implementation

From declaration 18 we construct the following Verilog program:

```
////////////////////////////////////////////////////////////////////////
//
// created by: Wouter de Koning
//
// This Verilog program implements g2; the restoring division
//
// as, bs, rs and qs are in two's complement representation (N + 1 bits)
//
////////////////////////////////////////////////////////////////////////

module DivModRestoring(as, bs, qs, rs);
  parameter N = 17;
      input [N:0]   as;
      input [N:0]   bs;
     output [N:0]   qs;
     output [N:0]   rs;

       wire [N-1:0] rss[N:0];   // rss[n] is the result of rs in g2 n
       wire [N:0]   hss[N-1:0]; // hss[n] is the result of hs in g2 n
       wire [N:0]   kss[N-1:0]; // kss[n] is the result of ks in g2 n

// - rss[0] is the result of rs in g2 0
 assign  rs    = { {1'b0} , rss[0] };

// - The case g2 N
 assign  qs[N] = as[N];
 assign rss[N] = (bs - 1) & {N{as[N]}};

// - The case g2 n for 0 = n < N
 genvar n;
     generate
         for (n=0; n<N; n=n+1)
         begin: RestDivMod
                assign hss[n] = {rss[n+1],as[n]};
                assign kss[n] =  hss[n] - bs;
                assign rss[n] =  kss[n][N] ? hss[n][N-1:0] : kss[n][N-1:0];
                assign  qs[n] = !kss[n][N];
         end
     endgenerate
endmodule
```

Chapter 4

Non-restoring Division

In Chapter 3 we derived an algorithm for implementing $A \underline{\text{div}}\ B$ and $A \underline{\text{mod}}\ B$, following the restoring division method. Instead of ensuring that the remainder in the intermediate steps remains non-negative, we create a function, which allows the remainder to be negative. In the postprocessing step, it is ensured that the remainder is non-negative. By weakening the specification, we derive a faster algorithm for division, called *non-restoring division*.

The functions in this chapter are very similar to the functions in Chapter 3, for this reason we use the same function names. So, we (re)derive a function $f\colon Int \to \langle\, Int\,,\, Int\,\rangle$, that computes the non-restoring division of A and B, specified by:

Specification 19: for a given constant B: $1 \leq B < 2^N$ and variable a: $-2^N \leq a < 2^N$:

$$f\cdot a \quad = \langle\, q\,,\, r\,\rangle \quad \underline{\text{whr}} \quad q, r\colon \ a = q{*}B + r \ \ \wedge \ \ -B \leq r < B \quad \underline{\text{end}}$$

Now $f\cdot A$ gives the solution to the non-restoring division of A and B.

For the base case $f\cdot(-1)$, we obtain the solutions $\langle\, 0\,,\, -1\rangle$ and $\langle -1\,,\, B-1\rangle$. And, for $f\cdot 0$ we obtain the solutions $\langle\, 0\,,\, 0\,\rangle$ and $\langle\, 1\,,\, -B\,\rangle$. In a later stage of the derivation of the algorithm, the solutions $\langle\, -1\,,\, B-1\,\rangle$ and $\langle\, 1\,,\, -B\,\rangle$ turn out to give better results.

* * *

For the recursive case we assume that q and r satisfy: $a = q{*}B + r$ and $-B \leq r < B$. We derive a solution for the case $2{*}a + b$, for all integers a and bits b, with $b \in \{0, 1\}$:

$$\begin{array}{ll} & 2{*}a + b \\ = & \{\ \text{assumption}\ \} \\ & 2{*}(\, q{*}B + r\,) + b \\ = & \{\ \text{algebra}\ \} \end{array}$$

$$
\begin{array}{ll}
 & (\,2{*}q\,) * B + (\,2{*}r + b\,) \\
= & \{\ \text{introduce } h = 2 * r + b\ \} \\
 & (\,2{*}q\,) * B + h
\end{array}
$$

Now the pair $\langle\ 2{*}q{-}1\ ,\ h{+}B\ \rangle$ is a solution for $f \cdot (\,2{*}a + b)$, provided $-B \leq h{+}B < B$, and the pair $\langle\ 2{*}q + 1\ ,\ h{-}B\ \rangle$ is a solution, provided $-B \leq h{-}B < B$. From $-B \leq r < B$ we derive the range of h:

$$
\begin{array}{ll}
 & -B\ \leq\ r\ <\ B \\
\equiv & \{\ \text{algebra}\ \} \\
 & -B \leq r \quad \wedge \quad r \leq B - 1 \\
\equiv & \{\ \text{algebra}\ \} \\
 & -2{*}B \leq 2 * r \quad \wedge \quad 2 * r \leq 2{*}B - 2 \\
\Rightarrow & \{\ 0 \leq b\ \}\ \{\ b \leq 1\ \} \\
 & -2{*}B \leq 2{*}r + b \quad \wedge \quad 2{*}r + b \leq 2{*}B - 1 \\
\equiv & \{\ \text{algebra}\ \} \\
 & -2{*}B\ \leq\ 2{*}r + b\ <\ 2{*}B \\
\equiv & \{\ \text{definition } h\ \} \\
 & -2{*}B\ \leq\ h\ <\ 2 * B
\end{array}
$$

So, we conclude that $-2 * B \leq h < 2 * B$. This proposition is split in two different cases: $-2 * B \leq h < 0$ and $0 \leq h < 2 * B$. The first is equal to $-B \leq h{+}B < B$ and the second to $-B \leq h{-}B < B$, thus each of those cases yields a solution for $f \cdot (\,2{*}a + b\,)$. Combining both cases, we obtain:

$$
\begin{array}{ll}
 & \underline{\text{if}} \quad -2 * B \leq h < 0 \quad \rightarrow\ \langle\ 2 * q - 1\ ,\ h{+}B\ \rangle \\
 & [] \qquad\ 0\ \leq h < 2 * B \quad \rightarrow\ \langle\ 2 * q + 1\ ,\ h{-}B\ \rangle \\
 & \underline{\text{fi}} \\
= & \{\ \text{logica, using } -2 * B \leq h < 2 * B\ \} \\
 & \underline{\text{if}} \quad h < 0 \quad \rightarrow\ \langle\ 2 * q - 1\ ,\ h{+}B\ \rangle \\
 & [] \quad h \geq 0 \quad \rightarrow\ \langle\ 2 * q + 1\ ,\ h{-}B\ \rangle \\
 & \underline{\text{fi}}
\end{array}
$$

Now we obtain the following declarations for function f, which computes the quotient and remainder for a given integer A and a divisor B:

Declaration 20:

$$
\begin{array}{lll}
 & f\cdot(-1) & = \langle\ -1\ ,\ B-1\ \rangle \\
\& & f\cdot 0 & = \langle\ 1\ ,\ -B\ \rangle
\end{array}
$$

$$
\begin{array}{lll}
\& & f\cdot(2{*}a+b) & = \underline{\text{if}}\quad h<0 \ \rightarrow\ \langle\ 2{*}q-1\ ,\ h+B\ \rangle \\
 & & \quad [] \quad h\geq 0 \ \rightarrow\ \langle\ 2{*}q+1\ ,\ h-B\ \rangle \\
 & & \quad \underline{\text{fi}} \quad \underline{\text{whr}}\quad h\ =\ 2{*}r+b \\
 & & \qquad\quad \&\quad \langle\ q\ ,\ r\ \rangle\ =\ f\cdot a \\
 & & \qquad\quad \underline{\text{end}}
\end{array}
$$

4.1 From integer to binary (part 1)

We have derived a function f that computes the quotient and the remainder, following the non-restoring division method. In this section we implement function f that computes A $\underline{\text{div}}$ B and A $\underline{\text{mod}}$ B in the binary system.

Similar to Section 3.1, we start by representing the value A in binary. We introduce as, the two's complement representation of A, with length $N+1$, provided $-2^N \leq A < 2^N$. We introduce a function, $g1$, which computes for a natural number n, with $0 \leq n < N$, the quotient and the remainder for the value of as without the least significant n bits. The maximal value of n is $N-1$; in the non-restoring division it is better to combine the two most significant bits of as in the base case.

The function $g1 : Int \rightarrow \langle\ Int\ ,\ Int\ \rangle$ is specified by:

Specification 21: for $0 \leq n < N$, a given constant B: $1 \leq B < 2^N$ and a given list as with $\#as = N+1$:

$$g1\cdot n \ =\ f\cdot(\ tc\cdot(\ as \lfloor n\ )\ )$$

For $f\cdot A$ we derive:

$$
\begin{array}{ll}
 & f\cdot A \\
= & \{\ A = tc\cdot as\ \} \\
 & f\cdot(\ tc\cdot as\ ) \\
= & \{\ \text{definition 3: } \lfloor\ \} \\
 & f\cdot(\ tc\cdot as \lfloor 0\ ) \\
= & \{\ \text{specification 21: } g1\ \} \\
 & g1\cdot 0
\end{array}
$$

So, $g1\cdot 0$ implements $f\cdot A$.

Now, from specification 21, we derive the declaration for $g1$. For the base case $n = N-1$, we derive:

$$
\begin{array}{ll}
 & g1 \cdot (\ N{-}1\ ) \\
= & \{\ \text{specification 21: } g1\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as \lfloor (N{-}1)\ )\ ) \\
= & \{\ \text{property 7: } \lfloor\ (\ {\times}2\ )\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as_{N-1} \triangleright as_N \triangleright as \lfloor (N{+}1)\ )\ ) \\
= & \{\ \#as\ =\ N+1, \text{ property 6 with } t = [\ ]\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as_{N-1} \triangleright as_N \triangleright [\ ]\ )\ ) \\
= & \{\ \text{definition 9: } tc\ (\ {\times}2\ )\ \} \\
 & f{\cdot}(\ 2 * (-as_N) + as_{N-1}\ )
\end{array}
$$

We distinguish two cases for as_N. The first case, if $as_N = 0$:

$$f{\cdot}(\ 2 * (\ 0\ ) + as_{N-1}\ )$$

We use declaration 20, with $a = 0$ and $b = as_{N-1}$. To find $\langle\ q\ ,\ r\ \rangle$ we use declaration 20 for $f \cdot 0$, hence $\langle\ q\ ,\ r\ \rangle = \langle\ 1\ ,\ -B\ \rangle$. Now, for h we derive:

$$
\begin{array}{ll}
 & h \\
= & \{\ h = 2{*}r + b\ \} \\
 & 2{*}(-B) + as_{N-1} \\
\leq & \{\ 1 \leq B\ \} \\
 & -2 + as_{N-1} \\
< & \{\ as_{N-1} \in \{0, 1\}\ \} \\
 & 0
\end{array}
$$

Applying $h < 0$, we derive for $f \cdot (\ 2{*}0 + as_{N-1}\ )$ the solution as follows:

$$
\begin{array}{ll}
 & f{\cdot}(\ 2 * (\ 0\ ) + as_{N-1}\ ) \\
= & \{\ \ h < 0\ \} \\
 & \langle\ 2{*}q - 1\ ,\ h + B\ \rangle \\
= & \{\ \ q = 1\ \ \}\ \{\ h = 2{*}(-B) + as_{N-1}\ \}\ \{\ \text{algebra}\ \} \\
 & \langle\ 1\ ,\ -B + as_{N-1}\ \rangle
\end{array}
$$

And for the second case, if $as_N = 1$:

$$f{\cdot}(\ 2 * (\ -1\ ) + as_{N-1}\ )$$

We use declaration 20, with $a = -1$ and $b = as_{N-1}$. To find $\langle\ q\ ,\ r\ \rangle$ we use declaration 20 for $f \cdot (-1)$, hence $\langle\ q\ ,\ r\ \rangle = \langle\ -1\ ,\ B - 1\ \rangle$. Now, for h we derive:

$$
\begin{array}{ll}
 & h \\
= & \{\ h = 2{*}r + b\ \} \\
 & 2{*}(B-1) + as_{N-1} \\
\geq & \{\ 1 \leq B\ \} \\
 & as_{N-1} \\
\geq & \{\ as_{N-1} \in \{0,1\}\ \} \\
 & 0
\end{array}
$$

Applying $0 \leq h$, we derive for $f \cdot (\ 2{*}(-1) + as_{N-1}\ )$ the solution as follows:

$$
\begin{array}{ll}
 & f{\cdot}(\ 2 * (\ -1\ ) + as_{N-1}\ ) \\
= & \{\ \ 0 \leq h\ \} \\
 & \langle\ 2{*}q + 1\ ,\ h - B\ \rangle \\
= & \{\ \ q = -1\ \ \}\ \{\ h = 2{*}(B-1) + as_{N-1}\ \}\ \{\ \text{algebra}\ \} \\
 & \langle\ -1\ ,\ B - 2 + as_{N-1}\ \rangle
\end{array}
$$

Both solutions combined we obtain:

$$
\begin{array}{lll}
g1{\cdot}(N-1) & = \langle\ \ \ 1\ ,\ -B + as_{N-1}\ \rangle & ,\ \text{if } as_N = 0 \\
g1{\cdot}(N-1) & = \langle\ -1\ ,\ B - 2 + as_{N-1}\ \rangle & ,\ \text{if } as_N = 1
\end{array}
$$

* * *

For the recursive case, we derive for $0 \leq n < N-1$:

$$
\begin{array}{ll}
 & g1{\cdot}n \\
= & \{\ \text{specification 21: } g1\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as \lfloor\ n\ )\ ) \\
= & \{\ \text{property 7}\ \} \\
 & f{\cdot}(\ tc{\cdot}(\ as_n \triangleright as \lfloor\ (n{+}1)\ )\ ) \\
= & \{\ \text{declaration 9: } tc\ \ \} \\
 & f{\cdot}(\ 2 * tc{\cdot}(\ as \lfloor\ (n{+}1)\ ) + as_n\ ) \\
= & \{\ \text{specification 19: } f\ \} \\
 & \underline{\text{if}}\ \ h < 0\ \ \rightarrow\ \ \langle\ 2{*}q - 1\ ,\ h + B\ \rangle \\
 & []\ \ h \geq 0\ \ \rightarrow\ \ \langle\ 2{*}q + 1\ ,\ h - B\ \rangle \\
 & \underline{\text{fi}}\ \ \ \underline{\text{whr}}\ \ h\ =\ 2{*}r + as_n \\
 & \qquad \&\ \ \ \langle\ q\ ,\ r\ \rangle\ =\ f{\cdot}(\ tc{\cdot}(\ as \lfloor\ (n{+}1)\ )\ ) \\
 & \qquad \underline{\text{end}}
\end{array}
$$

From specification of $g1$, we rewrite $f\cdot(\ tc\cdot(\ as\lfloor(n{+}1)\ )\ )$ to $g1\cdot(n{+}1)$. Together with the result for the case $n = N{-}1$, we obtain the following declarations for the function $g1$:

Declaration 22:

$$\begin{array}{lcl}
g1\cdot(N{-}1) & = & \underline{\text{if}}\quad as_N = 1 \quad\to\quad \langle\ -1\ ,\ B-2+as_{N-1}\ \rangle \\
 & & []\quad as_N = 0 \quad\to\quad \langle\quad 1\ ,\quad -B+as_{N-1}\ \rangle \\
 & & \underline{\text{fi}} \\
g1\cdot n & = & \underline{\text{if}}\quad h<0 \quad\to\quad \langle\ 2*q-1\ ,\ h+B\ \rangle \\
 & & []\quad h\geq 0 \quad\to\quad \langle\ 2*q+1\ ,\ h-B\ \rangle \\
 & & \underline{\text{fi}}\quad \underline{\text{whr}}\quad h\ =\ 2*r+as_n \\
 & & \qquad\&\quad \langle\ q\ ,\ r\ \rangle\ =\ g1\cdot(n{+}1) \\
 & & \qquad\underline{\text{end}}
\end{array}$$

4.2 From integer to binary (part 2)

We have derived a function that computes A $\underline{\text{div}}$ B and A $\underline{\text{mod}}$ B, where A is represented as a binary list. Here we present function $g2$, which represents B, the quotient and the remainder in the declaration of $g1$ as binary lists.

The remainder r in the declaration of $g1$ satisfies $-2^N \leq r < 2^N$, hence, we specify the length of rs, the binary list representing r, to be $N{+}1$; thus rs is in two's complement representation. The quotient q in the base case of $g1$ is -1 or 1, which is represented by two bits in two's complement representation. In the recursive case, where $n < N{-}1$, the quotient is divided by 2 with each increase of n; in binary this is achieved with a bitshift. As a result, we specify the length of the binary list representing q to be $N{-}n+1$.

Now the function $g2\colon \mathit{Int} \to \langle\ \mathcal{L}2\ ,\ \mathcal{L}2\ \rangle$ is specified by:

Specification 23: for $0 \leq n \leq N$, a given list bs: $v2\cdot bs = B\ \wedge\ \#bs = N$ and a given list as with $\#as = N+1$:

$$\begin{array}{lcll}
g2\cdot n & = & \langle\ qs\ ,\ rs\ \rangle & \underline{\text{whr}}\quad qs, rs\colon\ \langle\ tc\cdot qs\ ,\ tc\cdot rs\ \rangle\ =\ g1\cdot n \\
 & & & \quad\wedge\ \#qs = N{-}n+1\quad\wedge\quad \#rs = N \\
 & & & \underline{\text{end}}
\end{array}$$

Following specification 23 we formulate the quotient and the remainder of declaration 22 for the case $n = N-1$ in binary. The quotient q is -1 or 1. In two's complement representation with length $N-n+1 = 2$, this is represented by $(\ 1\triangleright[\,1\,]\ )$ and $(\ 1\triangleright[\,0\,]\ )$ respectively.

For the remainder we formulate the values $B-2+as_{N-1}$ and $-B+as_{N-1}$ in two's complement represenation with length $N{+}1$. We introduce for these values, respectively, the lists *bmin2plusas* and *minbplusas*, as defined by:

$$\begin{array}{rcl}
B-2+as_{N-1} = tc\cdot \mathit{bmin2plusas} & \wedge & \#\mathit{bmin2plusas} = N+1 \\
-B+as_{N-1} = tc\cdot \mathit{minbplusas} & \wedge & \#\mathit{minbplusas}\ = N+1
\end{array}$$

* * *

The operations on the quotient in the recursive definition of $g1$ are $2*q+1$ and $2*q-1$. The latter expression does not match the definition of tc. However, we observe that the result of both expressions are odd. Every odd number q can be written as $2*q1+1$, with $q1 \in \mathbb{Z}$. Substituting this into the expression $2*q+1$, we derive the following:

$$
\begin{array}{ll}
 & 2*q+1 \\
= & \{ \text{ substitute } q = 2*q1+1 \ \} \\
 & 2*(\ 2*q1+1\ )+1 \\
= & \{ \text{ introduce } q1s\text{: the two's complement representation of } q1 \ \} \\
 & 2*(\ 2*tc{\cdot}q1s+1\ )+1 \\
= & \{ \text{ definition 9: } tc\ (\times 2) \ \} \\
 & tc{\cdot}(\ 1 \triangleright 1 \triangleright q1s\ )
\end{array}
$$

and, for the expression $2*q-1$:

$$
\begin{array}{ll}
 & 2*q-1 \\
= & \{ \text{ substitute } q = 2*q1+1 \ \} \\
 & 2*(\ 2*q1+1\ )-1 \\
= & \{ \text{ algebra } \} \\
 & 2*(\ 2*q1\ )+1 \\
= & \{ \text{ definition } q1s\text{: the two's complement representation of } q1 \ \} \\
 & 2*(\ 2*tc{\cdot}q1s\ )+1 \\
= & \{ \text{ definition 9: } tc\ (\times 2) \ \} \\
 & tc{\cdot}(\ 1 \triangleright 0 \triangleright q1s\ )
\end{array}
$$

Now we express q in the recursive call of declaration 22 in terms of $q1s$ as follows:

$$
\begin{array}{ll}
 & q \\
= & \{ \ qs \text{ is the two's complement representation of } q \ \} \\
 & tc \cdot qs \\
= & \{ \ q = 2*q1+1 \ \} \\
 & 2*tc{\cdot}q1s+1 \\
= & \{ \text{ definition 9: } tc \ \} \\
 & tc{\cdot}(\ 1 \triangleright q1s\ )
\end{array}
$$

All expressions for the quotient in the base case and in the recursive case are in the form $tc \cdot (\ 1 \triangleright \ldots\ )$; thus, $1 \triangleright q1s$ is a viable expression for q in the recursive call.

For the remainder and the guard, we derive hs, the two's complement representation of h, as

follows:

$$
\begin{array}{ll}
 & tc{\cdot}hs \\
= & \{\ h = 2 * r + as_n\ \} \\
 & 2 * tc{\cdot}rs + as_n \\
= & \{\ \text{declaration 9: } tc\ \} \\
 & tc{\cdot}(as_n \triangleright rs)
\end{array}
$$

Now hs is defined as $as_n \triangleright rs$.

To verify whether the guard $h < 0$ is satisfied, one could check the value of the sign bit of hs, namely hs_{N+1}. But, because hs is defined as $as_n \triangleright rs$, the sign bits of rs and hs have the same value. So, instead of checking bit hs_{N+1}, we check rs_N.

For $h+B$ and $h-B$ in the remainder we introduce two lists, *hplusb* and *hminb* , respectively. If $-2{*}B \leq h < 0$, the remainder is equal to $h + B$. Now the length of *hplusb* is derived as follows:

$$
\begin{array}{ll}
 & -2 * B \leq h < 0 \\
= & \{\ \text{algebra}\ \} \\
 & -B \leq h + B < B \\
= & \{\ tc \cdot hplusb \text{ is equal to } h + B\ \} \\
 & -B \leq tc \cdot hplusb < B \\
\Rightarrow & \{\ B < 2^N\ \} \\
 & -2^N \leq tc \cdot hplusb < 2^N
\end{array}
$$

The minimal length of *hplusb* is $N + 1$, and following a similar proof, the minimal length of *hminb* is also $N + 1$. Now we define *hplusb* and *hminb* by:

$$
\begin{array}{lll}
h + B = tc \cdot hplusb & \wedge & \#hplusb = N + 1 \\
h - B = tc \cdot hminb & \wedge & \#hminb = N + 1
\end{array}
$$

The list *hplusb* is the result of the sum of two lists, namely hs with length $N{+}2$ and bs with length N. Then the last bit of the list is removed. However, we first remove the last bit of hs and then add list bs instead, as it gives the same result. Similar for the list *hminb* , we first remove the last bit of hs before bs is subtracted.

Because we first remove the last bit of hs before using it in the remainder, we do not use the last bit of hs at all. Hence, we redefine hs as $as_n \triangleright (\, rs \lceil N \,)$.

Now, by combining the guards, quotients and remainders, we obtain the following declaration for $g2$:

Declaration 24:

$$
\begin{array}{lcl}
g2\cdot(N{-}1) & = & \underline{\text{if}} \quad as_N = 1 \quad \rightarrow \quad \langle\ 1 \triangleright [\ 1\ ]\ ,\ \mathit{bmin2plusas}\ \rangle \\
 & & [] \quad as_N = 0 \quad \rightarrow \quad \langle\ 1 \triangleright [\ 0\ ]\ ,\ \mathit{minbplusas}\ \rangle \\
 & & \underline{\text{fi}} \\
g2\cdot n & = & \underline{\text{if}} \quad rs_N = 1 \quad \rightarrow \quad \langle\ 1 \triangleright 0 \triangleright qs\ ,\ \mathit{hplusb}\ \rangle \\
 & & [] \quad rs_N = 0 \quad \rightarrow \quad \langle\ 1 \triangleright 1 \triangleright qs\ ,\ \mathit{hminb}\ \rangle \\
 & & \underline{\text{fi}} \quad \underline{\text{whr}} \quad hs \ = \ as_n \triangleright (\ rs \lceil N\ ) \\
 & & \qquad\ \& \quad \langle\ 1 \triangleright qs\ ,\ rs\ \rangle \ = \ g2\cdot(n{+}1) \\
 & & \qquad \underline{\text{end}}
\end{array}
$$

4.3 Post-processing step

The remainder r satisfies $-B \leq r < B$. To implement the $\underline{\text{div}}$ and $\underline{\text{mod}}$ operators, the remainder in the end should be non-negative. Therefore, we add a post-processing step to add value B to the remainder if this remainder is negative. We introduce list *rplusb* defined by:

$$tc{\cdot}rs + v2{\cdot}bs = tc \cdot \mathit{rplusb} \quad \wedge \quad \#\mathit{rplusb} = N + 1$$

When we increase the remainder with B, the quotient decreases by one. A decrease of a list $(1 \triangleright s)$ results in a list $(0 \triangleright s)$, because $tc \cdot (1 \triangleright s) = tc \cdot (0 \triangleright s) + 1$. The function *post* is now declared as follows:

Declaration 25:

$$
\begin{array}{lcl}
\mathit{post}\cdot\langle\ 1 \triangleright qs\ ,\ rs\ \rangle & = & \underline{\text{if}} \quad rs_N = 1 \quad \rightarrow \quad \langle\ 0 \triangleright qs\ ,\ \mathit{rplusb}\ \rangle \\
 & & [] \quad rs_N = 0 \quad \rightarrow \quad \langle\ 1 \triangleright qs\ ,\ rs\ \rangle \\
 & & \underline{\text{fi}}
\end{array}
$$

4.4 Verilog implementation

From declaration 24 and declaration 25, we construct the following Verilog program:

```
////////////////////////////////////////////////////////////////////////
//
// created by: Wouter de Koning
//
// This Verilog program implements g2; the non-restoring division
//
// as, bs, rs and qs are in two's complement representation (N + 1 bits)
//
////////////////////////////////////////////////////////////////////////

module DivModNonRestoring(as, bs, qs, rs);
  parameter N = 17;
     input [N:0] as;
     input [N:0] bs;
    output [N:0] qs;
    output [N:0] rs;

      wire [N:0] rss[N-1:0]; // rss[n] is the result of rs in g2 n
      wire [N:0] hss[N-1:0]; // hss[n] is the result of hs in g2 n

// -- Post-processing step
 assign  qs[0]   = !rss[0][N];
 assign  rs      =  rss[0][N] ? rss[0][N:0] + bs : rss[0][N:0];

// -- The case g2 (N-1)
 assign  qs[N]   = as[N];
 assign rss[N-1] = as[N] ? bs - 2 + as[N-1] : -bs + as[N-1];

// -- The case g2 n for 0 = n < N - 1
 genvar n;
     generate
         for (n=0; n<N-1; n=n+1)
         begin: NonRestDivMod
                assign  hss[n] = {rss[n+1][N-1:0],as[n]};
                assign  rss[n] =  rss[n+1][N] ? hss[n] + bs : hss[n] - bs;
                assign qs[n+1] = !rss[n+1][N];
         end
     endgenerate
endmodule
```

Chapter 5

Radix-4 Division

In Chapter 3 we have derived an algorithm, called restoring division, for computing $A \underline{\text{div}}\ B$ and $A \underline{\text{mod}}\ B$ in the binary number system. Essentially the same algorithm can be formulated in any number system. In this chapter we present a version of this algorithm for radix 4.

The advantage of using radix 4—or radix 2^k, for any natural number k—is that the digits 0,1,2,3 can be represented in binary, by groups of two bits—or groups of k bits, in the general case. As a result we obtain an algorithm that processes 2 bits of A at a time, instead of one bit at a time for the binary version of the algorithm. Thus, the algorithm will require roughly half the amount of steps compared to the binary; every step, however, will be more complicated, but nevertheless the algorithm may be expected to be faster.

Similar to the restoring division, we derive a function $f\colon \mathit{Int} \to \langle\, \mathit{Int}\,,\, \mathit{Int}\,\rangle$ that computes the quotient, $A \underline{\text{div}}\ B$, and the remainder, $A \underline{\text{mod}}\ B$, specified by:

Specification 26: for a given constant B: $1 \leq B < 2^N$ and variable a: $-2^N \leq a < 2^N$:

$$f{\cdot}a \;=\; \langle\, q\,,\, r\,\rangle \quad \underline{\text{whr}} \quad q, r\colon \;\; a \;=\; q * B + r \;\wedge\; 0 \leq r < B \quad \underline{\text{end}}$$

Now $f{\cdot}A$ gives the solution to the quotient and the remainder.

We distinguish four base cases for f; we choose as arguments $-2, -1, 0$ and 1, which are the different values of a 2 bit group in the two's complement representation. We have already derived the declarations for the case $f \cdot 0$ and $f \cdot (-1)$ in Chapter 3:

$$\begin{array}{ll} f{\cdot}0 & = \langle\, 0\ ,\ 0\,\rangle \\ f{\cdot}(-1) & = \langle\, -1\ ,\ B-1\,\rangle \end{array}$$

For the case $f \cdot 1$ we derive:

$$\begin{array}{ll} & f{\cdot}1 \\ = & \{\ \text{specification 13: } f\ \} \end{array}$$

$\langle\ q\ ,\ r\ \rangle \quad \underline{\text{whr}} \quad q,r\colon\ 1\ =\ q*B+r\ \wedge\ 0\le r<B \quad \underline{\text{end}}$

$=$ { if $B=1$, the only solution is $q=1$ and $r=0$, otherwise $q=0$ and $r=1$ }

$\underline{\text{if}}\ \ 1\ =\ B\ \ \rightarrow\ \ \langle\ 1\ ,\ 0\ \rangle$
$[]\ \ 2\ \le\ B\ \ \rightarrow\ \ \langle\ 0\ ,\ 1\ \rangle$
$\underline{\text{fi}}$

and,

$f\cdot(-2)$

$=$ { specification 13: f }

$\langle\ q\ ,\ r\ \rangle \quad \underline{\text{whr}} \quad q,r\colon\ -2\ =\ q*B+r\ \wedge\ 0\le r<B \quad \underline{\text{end}}$

$=$ { if $B=1$, the only solution is $q=-2$ and $r=0$, otherwise $q=-1$ and $r=B-2$ }

$\underline{\text{if}}\ \ 1\ =\ B\ \ \rightarrow\ \ \langle\ -2\ ,\ 0\ \rangle$
$[]\ \ 2\ \le\ B\ \ \rightarrow\ \ \langle\ -1\ ,\ B-2\ \rangle$
$\underline{\text{fi}}$

* * *

For the recursive case we assume that q and r satisfy: $a=q*B+r$ and $0\le r<B$. Now we derive a declaration for the case $4*a+d$, for all integers a and digits d, with $0\le d<4$:

$4*a+d$

$=$ { assumption }

$4*(\,q*B+r\,)+d$

$=$ { algebra }

$(\,4*q\,)*B+(\,4*r+d\,)$

$=$ { introduce $h=4*r+d$ }

$(\,4*q\,)*B\ +\ h$

$=$ { algebra; introduce integer x }

$(\,4*q+x\,)*B\ +\ h-x*B$

For any integer x, the pair $\langle\ 4*q+x\ ,\ h-x*B\ \rangle$ is a solution for $f\cdot(4*a+d)$, provided $0\le h-x*B<B$. Now, from $0\le r<B$ we derive the range of h:

$0\ \le\ r\ <\ B$

$\equiv$ { algebra }

$0\le r\ \ \wedge\ \ r\le B-1$

$\equiv$ { algebra }

$$
\begin{array}{ll}
 & 0 \le 4*r \quad \wedge \quad 4*r \le 4*B - 4 \\
\Rightarrow & \{\ 0 \le d\ \}\ \{\ d \le 3\ \} \\
 & 0 \le 4*r + d \quad \wedge \quad 4*r + d \le 4*B - 1 \\
\equiv & \{\ \text{algebra}\ \} \\
 & 0\ \le\ 4*r + d\ <\ 4*B \\
\equiv & \{\ \text{definition } h\ \} \\
 & 0\ \le\ h\ <\ 4*B
\end{array}
$$

We conclude that $0 \le h < 4*B$ holds. This proposition is split in the following four cases:

$$0 \le h < B \ , \quad B \le h < 2{*}B \ , \quad 2{*}B \le h < 3{*}B \ , \quad \text{and} \quad 3{*}B \le h < 4{*}B$$

Each of these cases can be denoted in the form: $0 \le h - x*B < B$, for $x \in \{0, 1, 2, 3\}$. Thus, each of these cases yields a solution for $f \cdot (4*a + d)$. Combining these cases, we obtain:

$$
\begin{array}{l}
\underline{\text{if}}\ \ 0\ \le\ h - 0{*}B\ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 0\ ,\ h - 0{*}B\ \rangle \\
[]\ \ 0\ \le\ h - 1{*}B\ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 1\ ,\ h - 1{*}B\ \rangle \\
[]\ \ 0\ \le\ h - 2{*}B\ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 2\ ,\ h - 2{*}B\ \rangle \\
[]\ \ 0\ \le\ h - 3{*}B\ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 3\ ,\ h - 3{*}B\ \rangle \\
\underline{\text{fi}}
\end{array}
$$

Now we obtain the following declarations for function f, which computes the quotient and remainder for a given integer A and a divisor B.

Declaration 27:

$$
\begin{array}{lll}
 & f{\cdot}(-2) & = \underline{\text{if}}\ \ 1 = B\ \ \rightarrow\ \ \langle\ -2\ ,\ 0\ \rangle \\
 & & \quad []\ \ 2 \le B\ \ \rightarrow\ \ \langle\ -1\ ,\ B - 2\ \rangle \\
 & & \quad \underline{\text{fi}} \\
\& & f{\cdot}(-1) & = \langle\ -1\ ,\ B - 1\ \rangle \\
\& & f{\cdot}0 & = \langle\ 0\ ,\ 0\ \rangle \\
\& & f{\cdot}1 & = \underline{\text{if}}\ \ 1 = B\ \ \rightarrow\ \ \langle\ 1\ ,\ 0\ \rangle \\
 & & \quad []\ \ 2 \le B\ \ \rightarrow\ \ \langle\ 0\ ,\ 1\ \rangle \\
 & & \quad \underline{\text{fi}}
\end{array}
$$

$$
\begin{array}{lll}
\& & f{\cdot}(\ 4{*}a + d\ ) & = \underline{\text{if}}\ \ 0\ \le\ h\ \ \ \ \ \ \ \ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q\ \ \ \ ,\ h\ \ \ \ \ \ \ \ \rangle \\
 & & & \quad []\ \ 0\ \le\ h - B\ \ \ \ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 1\ ,\ h - B\ \ \ \ \rangle \\
 & & & \quad []\ \ 0\ \le\ h - 2{*}B\ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 2\ ,\ h - 2{*}B\ \rangle \\
 & & & \quad []\ \ 0\ \le\ h - 3{*}B\ <\ B\ \ \rightarrow\ \ \langle\ 4{*}q + 3\ ,\ h - 3{*}B\ \rangle \\
 & & & \quad \underline{\text{fi}}\ \ \ \underline{\text{whr}}\ \ h\ =\ 4{*}r + d \\
 & & & \qquad\ \ \&\ \ \ \langle\ q\ ,\ r\ \rangle\ =\ f{\cdot}a \\
 & & & \qquad \underline{\text{end}}
\end{array}
$$

5.1 From integer to binary (part 1)

We have derived a function f that computes the quotient and the remainder. In this section we implement function f that computes $A \underline{\text{div}}\ B$ and $A \underline{\text{mod}}\ B$ in the binary system.

First we represent the value A in binary, as, with length $N{+}1$, provided $-2^N \leq A < 2^N$. We assume N is odd, hence $\#as$ is even. Now we introduce a new function, $g1$, which computes for a natural number n, $2{*}n < N{+}1$, the quotient and the remainder for the value of as without the least significant $2 * n$ bits.

The function $g1 : Int \to \langle\, Int\,,\, Int\,\rangle$ is specified by:

Specification 28: for $0 \leq 2 * n < N{+}1$, a given constant B: $1 \leq B < 2^N$ and a given list as with $\#as = N + 1$:

$$g1{\cdot}n \quad = \quad f{\cdot}(\ tc{\cdot}(\ as \lfloor (2 * n)\ )\ )$$

For $f{\cdot}A$ we derive:

$$\begin{array}{ll} & f{\cdot}A \\ = & \{\ A = tc{\cdot}as\ \} \\ & f{\cdot}(\ tc{\cdot}as\ ) \\ = & \{\ \text{definition 3: } \lfloor\ \} \\ & f{\cdot}(\ tc{\cdot}as \lfloor 0\ ) \\ = & \{\ \text{specification 28: } g1\ \} \\ & g1{\cdot}0 \end{array}$$

So, $g1{\cdot}0$ implements $f{\cdot}A$.

Now, from specification 28, we derive the declaration for $g1$. For the base case $n = M$, where M is the maximal value of n—hence, $M = (N{-}1)\ \underline{\text{div}}\ 2$—we derive:

$$\begin{array}{ll} & g1 \cdot M \\ = & \{\ \text{specification 28: } g1\ \} \\ & f{\cdot}(\ tc{\cdot}(\ as \lfloor (2{*}M)\ )\ ) \\ = & \{\ (N{-}1) \text{ is even, hence } 2 * M = N{-}1\ \} \\ & f{\cdot}(\ tc{\cdot}(\ as \lfloor (N{-}1)\ )\ ) \\ = & \{\ \text{property 7: } \lfloor\ (\times 2)\ \} \\ & f{\cdot}(\ tc{\cdot}(\ as_{N-1} \triangleright as_N \triangleright as \lfloor (N{+}1)\ )\ ) \\ = & \{\ \#as = N{+}1\text{; property 6}\ \} \end{array}$$

$$
\begin{array}{ll}
 & f\cdot(\ tc\cdot(\ as_{N-1} \triangleright as_N \triangleright [\,]\ )\ ) \\
= & \{\ \text{definition 0: } \triangleright\ \} \\
 & f\cdot(\ tc\cdot(\ as_{N-1} \triangleright [\ as_N\ ]\ )\ ) \\
= & \{\ \text{declaration 9: } tc\ (\times 2)\ \} \\
 & f\cdot(\ as_{N-1} - 2{*}as_N\ ) \\
= & \{\ \text{declaration 27: } f\ \}\ \{\ (as_{N-1} - 2{*}as_N) \in \{-2,-1,0,1\}\ \ \}
\end{array}
$$

$$
\begin{array}{l}
\underline{\text{if}}\ \ as_{N-1}=0\ \wedge\ as_N=0\ \ \to\ \ \langle\ 0\ ,\ 0\ \rangle \\
[]\ \ as_{N-1}=1\ \wedge\ as_N=0\ \ \to\ \ \underline{\text{if}}\ \ 1=B\ \ \to\ \ \langle\ 1\ ,\ 0\ \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad []\ \ 2\le B\ \ \to\ \ \langle\ 0\ ,\ 1\ \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{\text{fi}} \\
[]\ \ as_{N-1}=0\ \wedge\ as_N=1\ \ \to\ \ \underline{\text{if}}\ \ 1=B\ \ \to\ \ \langle\ -2\ ,\ 0\ \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad []\ \ 2\le B\ \ \to\ \ \langle\ -1\ ,\ B-2\ \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{\text{fi}} \\
[]\ \ as_{N-1}=1\ \wedge\ as_N=1\ \ \to\ \ \langle\ -1\ ,\ B-1\ \rangle \\
\underline{\text{fi}}
\end{array}
$$

and, for the case $0 \le n < M$:

$$
\begin{array}{ll}
 & g1{\cdot}n \\
= & \{\ \text{specification 28: } g1\ \} \\
 & f\cdot(\ tc\cdot(\ as\lfloor(2{*}n)\ )\ ) \\
= & \{\ \text{property 7 } (\times 2)\ \} \\
 & f\cdot(\ tc\cdot(\ as_{2*n} \triangleright as_{2*n+1} \triangleright as\lfloor(2{*}n+2)\ )\ ) \\
= & \{\ \text{declaration 9: } tc\ (\times 2)\ \} \\
 & f\cdot(\ 4{*}tc{\cdot}as\lfloor(2*n+2) + 2{*}as_{2*n+1} + as_{2*n}\ ) \\
= & \{\ \text{specification 26: } f, \text{ with } h\ =\ as_{2*n} + 2{*}as_{2*n+1} + 4{*}r\ \}
\end{array}
$$

$$
\begin{array}{l}
\underline{\text{if}}\ \ 0\ \le\ h\ \ \ \ \ \ \ \ <\ B\ \ \to\ \ \langle\ 4{*}q\ \ \ \ \ ,\ h\ \ \ \ \ \ \ \ \rangle \\
[]\ \ 0\ \le\ h-B\ \ \ \ \ <\ B\ \ \to\ \ \langle\ 4{*}q+1\ ,\ h-B\ \ \ \ \rangle \\
[]\ \ 0\ \le\ h-2{*}B\ <\ B\ \ \to\ \ \langle\ 4{*}q+2\ ,\ h-2{*}B\ \rangle \\
[]\ \ 0\ \le\ h-3{*}B\ <\ B\ \ \to\ \ \langle\ 4{*}q+3\ ,\ h-3{*}B\ \rangle \\
\underline{\text{fi}}\ \ \ \underline{\text{whr}}\ \ \langle\ q\ ,\ r\ \rangle\ =\ f\cdot(\ tc\cdot(\ as\lfloor(2*n+2)\ )\ )\ \ \underline{\text{end}}
\end{array}
$$

From specification of $g1$, we rewrite $f \cdot (\ tc \cdot (\ as\lfloor(2{*}n+2)\ )\ )$ to $g1 \cdot (n{+}1)$. Together with the result for the case $n = M$, we obtain the following declarations for function $g1$, with $0 \le n < M$:

Declaration 29:

$$
\begin{array}{lllllll}
 & g1{\cdot}M & = & \underline{\text{if}} & as_{N-1}=0 \ \wedge \ as_N=0 & \rightarrow & \langle\ 0\ ,\ 0\ \rangle \\
 & & & [] & as_{N-1}=1 \ \wedge \ as_N=0 & \rightarrow & \underline{\text{if}}\ \ 1=B \ \rightarrow \ \langle\ 1\ ,\ 0\ \rangle \\
 & & & & & & [] \ \ 2\le B \ \rightarrow \ \langle\ 0\ ,\ 1\ \rangle \\
 & & & & & & \underline{\text{fi}} \\
 & & & [] & as_{N-1}=0 \ \wedge \ as_N=1 & \rightarrow & \underline{\text{if}}\ \ 1=B \ \rightarrow \ \langle\ -2\ ,\ 0\ \rangle \\
 & & & & & & [] \ \ 2\le B \ \rightarrow \ \langle\ -1\ ,\ B{-}2\ \rangle \\
 & & & & & & \underline{\text{fi}} \\
 & & & [] & as_{N-1}=1 \ \wedge \ as_N=1 & \rightarrow & \langle\ -1\ ,\ B{-}1\ \rangle \\
 & & & \underline{\text{fi}} & & & \\
\& & g1{\cdot}n & = & \underline{\text{if}} & 0\le h\ <\ B & \rightarrow & \langle\ 4{*}q\ ,\ h\ \rangle \\
 & & & [] & 0\le h-B\ <\ B & \rightarrow & \langle\ 4{*}q+1\ ,\ h-B\ \rangle \\
 & & & [] & 0\le h-2{*}B\ <\ B & \rightarrow & \langle\ 4{*}q+2\ ,\ h-2{*}B\ \rangle \\
 & & & [] & 0\le h-3{*}B\ <\ B & \rightarrow & \langle\ 4{*}q+3\ ,\ h-3{*}B\ \rangle \\
 & & & \underline{\text{fi}} & \underline{\text{whr}}\ \ h\ =\ as_{2*n}+2{*}as_{2*n+1}+4{*}r & & \\
 & & & & \ \ \&\ \ \langle\ q\ ,\ r\ \rangle\ =\ g1{\cdot}(n{+}1) & & \\
 & & & & \underline{\text{end}} & &
\end{array}
$$

5.2 From integer to binary (part 2)

We have derived a function that computes A $\underline{\text{div}}$ B and A $\underline{\text{mod}}$ B, where A is represented as a binary list. Here we present function $g2$, which represents B, the quotient and the remainder in the declaration of $g1$ as binary lists.

The remainder r in the declaration of $g1$ satisfies: $0 \le r < 2^N$, hence, we specify the length of rs, the binary list representing r, to be N; thus rs is in binary representation. The quotient q in the base case of $g1$ is one of the digits $-2, -1, 0, 1$, which can be represented by two bits in two's complement representation. In the case where $n \neq M$, the quotient q is divided by 4 with each increase of n; in binary this is achieved with a bitshift of 2 to the left. As a result, we specify the length of the binary list representing q to be $(M{-}n) * 2 + 2$.

Now the function $g2 : Int \rightarrow \langle\ \mathcal{L}2\ ,\ \mathcal{L}2\ \rangle$ is specified by:

Specification 30: for $0 \le n \le M$, where $M = (N{-}1)$ $\underline{\text{div}}$ 2, a given list bs: $v2 \cdot bs = B$ $\wedge$ $\#bs = N$ and a given list as with $\#as = N + 1$:

$$
\begin{array}{lll}
g2{\cdot}n & = \langle\ qs\ ,\ rs\ \rangle & \underline{\text{whr}}\ \ qs, rs{:}\ \ \langle\ tc{\cdot}qs\ ,\ v2{\cdot}rs\ \rangle\ =\ g1{\cdot}n \\
 & & \ \ \wedge\ \#qs = 2 + (M{-}n){*}2\ \ \wedge\ \ \#rs = N \\
 & & \underline{\text{end}}
\end{array}
$$

Following specification 30 we first formulate the quotient and the remainder of declaration 29 for the case $n = M$ in binary. The quotient q is one of the integers $-2, -1, 0, 1$. The length of the quotient in two's complement representation, qs, is $2 + (M{-}n) * 2 = 2$. By using the declaration of tc twice (declaration 9) we get the following equations:

$$\begin{array}{lcl} 0 & = & tc{\cdot}[\,00\,] \\ 1 & = & tc{\cdot}[\,10\,] \\ -2 & = & tc{\cdot}[\,01\,] \\ -1 & = & tc{\cdot}[\,11\,] \end{array}$$

For the remainder, we formulate the values $0, 1, B-1$ and $B-2$ in binary representation with length N. We introduce for these values, respectively, the lists *zero*, *one*, $bs1$ and $bs2$. The value $B-2$ could be negative, therefore, we use in this case the two's complement representation instead of the binary representation. These lists are defined as follows:

$$\begin{array}{lcll} 0 & = & v2 \cdot zero & \wedge \quad \#zero = N \\ 1 & = & v2 \cdot one & \wedge \quad \#one = N \\ B{-}1 & = & v2 \cdot bs1 & \wedge \quad \#bs1 = N \\ B{-}2 & = & tc \cdot bs2 & \wedge \quad \#bs2 = N{+}1 \end{array}$$

If $0 \leq B-2$, which is true for $B-2$ in the remainder, then $bs2$ can be expressed in binary representation with length N. Hence, we use sign truncation (property 12), such that the length of the remainder is N.

The list $bs2$ is also useful for rewriting the guards. Whether a list in two's complement representation is negative or non-negative, can be verified by the value of the sign bit. This is used for verifying whether a guard holds. The guard $1 = B$ is rewritten as follows:

$$\begin{array}{ll} & 1 = B \\ \equiv & \{\ 1 \leq B\ \} \\ & B \leq 1 \\ \equiv & \{\ \text{algebra}\ \} \\ & B - 2 < 0 \\ \equiv & \{\ B{-}2 = tc \cdot bs2\ \} \\ & tc{\cdot}bs2 < 0 \\ \equiv & \{\ \text{sign bit of } bs2 \text{ is on position } N\ \} \\ & bs2_N = 1 \end{array}$$

As a result, the other guard $2 \leq B$ is rewritten to $bs_N = 0$.

* * *

We formulated the remainder, quotient and the guards for the case $n = M$ in binary. Now, for the recursive case $0 \leq n < M$, we formulate first the quotient in two's complement representation. The quotient (declaration 29) is of the form $4{*}q + x$, where $x \in \{0, 1, 2, 3\}$; consequently, it is rewritten in the two's complement representation as follows:

$$
\begin{array}{ll}
 & 4 * q + x \\
= & \{\ q = tc{\cdot}qs\ \} \\
 & 4 * tc{\cdot}qs + x \\
= & \{\ \text{declaration 9: } tc\ (\times 2)\ \} \\
 & \begin{array}{llllll}
\underline{\text{if}} & x & = & 0 & \rightarrow & tc{\cdot}(0 \triangleright 0 \triangleright qs) \\
[] & x & = & 1 & \rightarrow & tc{\cdot}(1 \triangleright 0 \triangleright qs) \\
[] & x & = & 2 & \rightarrow & tc{\cdot}(0 \triangleright 1 \triangleright qs) \\
[] & x & = & 3 & \rightarrow & tc{\cdot}(1 \triangleright 1 \triangleright qs) \\
\underline{\text{fi}} & & & & &
\end{array}
\end{array}
$$

For the remainder, we derive hs, the binary representation of h, as follows:

$$
\begin{array}{ll}
 & h \\
= & \{\ \text{definition } h\ \} \\
 & as_{2*n} + 2 * as_{2*n+1} + 4 * r \\
= & \{\ r = v2{\cdot}rs\ \} \\
 & as_{2*n} + 2 * as_{2*n+1} + 4 * v2{\cdot}rs \\
= & \{\ \text{declaration 8: } v2\ (\times 2)\ \} \\
 & v2{\cdot}(as_{2*n} \triangleright as_{2*n+1} \triangleright rs)
\end{array}
$$

Now hs is defined as $as_{2*n} \triangleright as_{2*n+1} \triangleright rs$.

With bs and hs we introduce the lists $h0b, h1b, h2b$ and $h3b$ in two's complement representation with length $N + 3$. These lists are useful to express the remainders and the guards:

$$
\begin{array}{lcl}
tc{\cdot}h0b & = & v2{\cdot}hs \\
tc{\cdot}h1b & = & v2{\cdot}hs - v2{\cdot}bs \\
tc{\cdot}h2b & = & v2{\cdot}hs - 2 * v2{\cdot}bs \\
tc{\cdot}h3b & = & v2{\cdot}hs - 3 * v2{\cdot}bs
\end{array}
$$

The four guards in the recursive part of declaration 29 are rewritten as follows:

$$
\begin{array}{ll}
 & (\ 0 \le h < B\ ) \vee \\
 & (\ 0 \le h - B < B\ ) \vee \\
 & (\ 0 \le h - 2 * B < B\ ) \vee \\
 & (\ 0 \le h - 3 * B < B\ ) \\
\equiv & \{\ \text{algebra}\ \}\ \{\ 0 \le h < 4 * B\ \} \\
 & (\ h - B < 0\ ) \vee \\
 & (\ 0 \le h - B \wedge h - 2 * B < 0\ ) \vee \\
 & (\ 0 \le h - 2 * B \wedge h - 3 * B < 0\ ) \vee \\
 & (\ 0 \le h - 3 * B\ ) \\
\equiv & \{\ h = v2{\cdot}hs \text{ and } B = v2{\cdot}bs\ \}\ \{\ \text{definition } h0b, h1b, h2b \text{ and } h3b\ \}
\end{array}
$$

$$
\begin{array}{l}
\quad (\quad\quad\quad\quad\quad tc{\cdot}h1b \;<\; 0\;) \quad \vee \\
\quad (\;0 \le tc{\cdot}h1b \;\wedge\; tc{\cdot}h2b \;<\; 0\;) \quad \vee \\
\quad (\;0 \le tc{\cdot}h2b \;\wedge\; tc{\cdot}h3b \;<\; 0\;) \quad \vee \\
\quad (\;0 \le tc{\cdot}h3b\quad\quad\quad\quad\quad\;\;) \\
\equiv \quad \{\text{ the sign bit of each list is on position } N{+}2 \;\} \\
\quad (\quad\quad\quad\quad\quad h1b_{N+2} = 1\;) \quad \vee \\
\quad (\;h1b_{N+2} = 0 \;\wedge\; h2b_{N+2} = 1\;) \quad \vee \\
\quad (\;h2b_{N+2} = 0 \;\wedge\; h3b_{N+2} = 1\;) \quad \vee \\
\quad (\;h3b_{N+2} = 0\quad\quad\quad\quad\quad\;\;)
\end{array}
$$

The guards in declaration 29 for $0 \le n < M$ provide $0 \le r < B$, hence $0 \le r < 2^N$. As a result, the remainder rs can be expressed in N bits. Hence, we use sign truncation (property 12), such that $\#rs = N$.

By combining the guards, quotients and remainders, we obtain the following declaration for $g2$:

Declaration 31:

$$
\begin{array}{lll}
g2{\cdot}M & = & \underline{\text{if}} \;\; as_{N-1} = 0 \;\wedge\; as_N = 0 \;\to\; \langle\, [00]\,,\; zero \,\rangle \\
& & [] \;\; as_{N-1} = 1 \;\wedge\; as_N = 0 \;\to\; \underline{\text{if}} \;\; bs2_N = 1 \;\to\; \langle\, [10]\,,\; zero \,\rangle \\
& & \qquad\qquad\qquad\qquad\qquad\quad [] \;\; bs2_N = 0 \;\to\; \langle\, [00]\,,\; one \,\rangle \\
& & \qquad\qquad\qquad\qquad\qquad\quad \underline{\text{fi}} \\
& & [] \;\; as_{N-1} = 0 \;\wedge\; as_N = 1 \;\to\; \underline{\text{if}} \;\; bs2_N = 1 \;\to\; \langle\, [01]\,,\; zero \,\rangle \\
& & \qquad\qquad\qquad\qquad\qquad\quad [] \;\; bs2_N = 0 \;\to\; \langle\, [11]\,,\; bs2\lceil N \,\rangle \\
& & \qquad\qquad\qquad\qquad\qquad\quad \underline{\text{fi}} \\
& & [] \;\; as_{N-1} = 1 \;\wedge\; as_N = 1 \;\to\; \langle\, [11]\,,\; bs1 \,\rangle \\
& & \underline{\text{fi}}
\end{array}
$$

$$
\begin{array}{lll}
\& \quad g2{\cdot}n & = & \underline{\text{if}} \;\; (\quad\quad\quad\quad h1b_{N+2} = 1\;) \;\to\; \langle\, 0 \triangleright 0 \triangleright qs\,,\; h0b\lceil N \,\rangle \\
& & [] \;\; (\;h1b_{N+2} = 0 \;\wedge\; h2b_{N+2} = 1\;) \;\to\; \langle\, 1 \triangleright 0 \triangleright qs\,,\; h1b\lceil N \,\rangle \\
& & [] \;\; (\;h2b_{N+2} = 0 \;\wedge\; h3b_{N+2} = 1\;) \;\to\; \langle\, 0 \triangleright 1 \triangleright qs\,,\; h2b\lceil N \,\rangle \\
& & [] \;\; (\;h3b_{N+2} = 0\quad\quad\quad\quad\quad\;\;) \;\to\; \langle\, 1 \triangleright 1 \triangleright qs\,,\; h3b\lceil N \,\rangle \\
& & \underline{\text{fi}} \quad \underline{\text{whr}} \;\; h0b \;=\; as_{2*n} \triangleright as_{2*n+1} \triangleright rs \\
& & \qquad \& \;\; h1b \;=\; h0b - bs \\
& & \qquad \& \;\; h2b \;=\; h0b - 2*bs \\
& & \qquad \& \;\; h3b \;=\; h0b - 3*bs \\
& & \qquad \& \;\; \langle\, qs\,,\; rs \,\rangle \;=\; g2{\cdot}(n{+}1) \\
& & \quad \underline{\text{end}}
\end{array}
$$

5.3 Verilog implementation

We have derived a function that computes $A \underline{\text{div}}\ B$ and $A \underline{\text{mod}}\ B$ in binary. Next we formulate this function in an efficient Verilog program: however, the declaration is still fairly complex. To resolve this, we express the first and second bit of the quotients independently as boolean equations, which can be implemented straightforward in Verilog. Moreover, we simplify the computation for the remainders.

For the case $n = M$ in declaration 31, we create the following truth table:

$bs2_N$	as_{N-1}	as_N	qs_{N-1}	qs_N	rs
0	0	0	0	0	*zero*
1	0	0	0	0	*zero*
0	1	0	0	0	*one*
1	1	0	1	0	*zero*
0	0	1	1	1	$bs2\lceil N$
1	0	1	0	1	*zero*
0	1	1	1	1	$bs1$
1	1	1	1	1	*zero*

Table 5.1: truth table for $g2$ with $n = M$

If the value of a bit b is 1, we write b, otherwise, we write $!\,b$. From the truth table shown in Table 5.1, we obtain the following equations:

Equations 0:

$$
\begin{array}{lcl}
qs_N & = & as_N \\
qs_{N-1} & = & (as_{N-1} \ \wedge \ bs2_N) \ \ \vee \ \ (as_N \ \wedge \ !\,bs2_N) \\
rs & = & \underline{\text{if}} \quad bs2_N \ \ \rightarrow \ \ zero \\
 & & [] \ \ !\,bs2_N \ \ \rightarrow \ \ \underline{\text{if}} \quad as_{N-1} \ \wedge \ \ as_N \ \ \rightarrow \ \ bs1 \\
 & & \qquad\qquad\qquad\quad [] \ \ !\,as_{N-1} \ \wedge \ \ as_N \ \ \rightarrow \ \ bs2\lceil N \\
 & & \qquad\qquad\qquad\quad [] \ \ \ as_{N-1} \ \wedge \ !\,as_N \ \ \rightarrow \ \ one \\
 & & \qquad\qquad\qquad\quad [] \ \ !\,as_{N-1} \ \wedge \ !\,as_N \ \ \rightarrow \ \ zero \\
 & & \qquad\qquad\qquad\quad \underline{\text{fi}} \\
 & & \underline{\text{fi}}
\end{array}
$$

And, for the case $0 \leq n < M$, we create the following truth table where the non-valid combinations of booleans are omitted:

$h1b_{N+1}$	$h2b_{N+1}$	$h3b_{N+1}$	qs_{2*n}	qs_{2*n+1}	rs
0	0	0	1	1	$h3b\lceil N$
0	0	1	0	1	$h2b\lceil N$
0	1	1	1	0	$h1b\lceil N$
1	1	1	0	0	$h0b\lceil N$

Table 5.2: truth table for $g2$ with $0 \leq n < M$

We obtain from the table the following equations, where $N2 = N + 2$:

Equations 1:

$$
\begin{array}{lcl}
qs_{2*n+1} & = & !\,h2b_{N2} \\
qs_{2*n} & = & (!\,h3b_{N2} \vee h2b_{N2}) \wedge !\,h1b_{N2} \\
rs & = & \underline{\text{if}} \quad h2b_{N2} \wedge h1b_{N2} \rightarrow h0b\lceil N \\
 & & [] \quad h2b_{N2} \wedge !\,h1b_{N2} \rightarrow h1b\lceil N \\
 & & [] \quad !\,h2b_{N2} \wedge h3b_{N2} \rightarrow h2b\lceil N \\
 & & [] \quad !\,h2b_{N2} \wedge !\,h3b_{N2} \rightarrow h3b\lceil N \\
 & & \underline{\text{fi}}
\end{array}
$$

* * *

Now from declarations 31 and equations 0 and 1 we construct the following Verilog program:

```
////////////////////////////////////////////////////////////////////////
//
// created by: Wouter de Koning
//
// This Verilog program implements g2; the radix-4 division
//
// as, bs, rs and qs are in two's complement representation (N + 1 bits),
// where N must be odd
//
////////////////////////////////////////////////////////////////////////

module DivModRadix4(as, bs, qs, rs);
  parameter  N = 17;   // must be odd
 localparam  M = (N-1)/2;
      input  [N:0]   as;
      input  [N:0]   bs;
     output  [N:0]   qs;
     output  [N:0]   rs;

       wire  [N-1:0] B;
       wire  [N-1:0] Bs1;
       wire  [N:0]   Bs2;
       wire  [N:0]   Btimes2;
       wire  [N+1:0] Btimes3;

       wire[N-1:0] rss[M:0];
       wire[N+2:0] h0b[M-1:0], h1b[M-1:0], h2b[M-1:0], h3b[M-1:0];
```

```
// -- assign constants
 assign        B = bs[N-1:0];
 assign      Bs1 = B - 1;
 assign      Bs2 = B - 2;
 assign  Btimes2 = { B , {1'b0} };
 assign  Btimes3 = Btimes2 + B;

 assign       rs = rss[0];

// -- The case g2 M
 assign   qs[N] = as[N];
 assign qs[N-1] = (as[N] & !Bs2[N]) | (as[N-1] & Bs2[N]);
 assign  rss[M] = ( (Bs2[N] | (!as[N] & !as[N-1]))
                  ? 0
                  : ( as[N-1] ? (as[N] ? Bs1 : 1) : Bs2[N-1:0] )
                  ) ;

// -- The case g2 n for 0 <= n < M
 genvar n;
     generate
         for (n=0; n<M; n=n+1)
         begin:  RadixDivMod
                 assign    h0b[n] = { {1'b0} , rss[n+1] , as[2*n+1:2*n] };
                 assign    h1b[n] = h0b[n] - B;
                 assign    h2b[n] = h0b[n] - Btimes2;
                 assign    h3b[n] = h0b[n] - Btimes3;

                 assign qs[2*n+1] = !h2b[n][N+2];
                 assign   qs[2*n] = (!h3b[n][N+2] | h2b[n][N+2]) & !h1b[n][N+2];

                 assign    rss[n] = ( h2b[n][N+2]
                                    ? ( h1b[n][N+2]
                                      ? h0b[n][N-1:0]
                                      : h1b[n][N-1:0]
                                      )
                                    : ( h3b[n][N+2]
                                      ? h2b[n][N-1:0]
                                      : h3b[n][N-1:0]
                                      )
                                    ) ;
         end
     endgenerate
endmodule
```

Chapter 6

Division by Multiplication

In the other chapters we have derived programs for the restoring division, non-restoring divison and radix 4 division, relatively. Another way to divide integers is by using multiplications. We wish to exploit the Spartan-3 Generation FPGA's fast dedicated 18×18 multipliers for this. Therefore, we are particularly interested in computing $A \underline{\text{div}} B$ and $A \underline{\text{mod}} B$, with A and B in the two's complement representation with 18 bits ($-2^{17} \leq A < 2^{17}$ and $1 \leq B < 2^{17}$). We use the following alternative definitions for quotient q and remainder r:

$$\begin{array}{lll} q\text{:} & A \underline{\text{div}} B = \left\lfloor \frac{A}{B} \right\rfloor, & \text{if } 1 \leq B \\ r\text{:} & A \underline{\text{mod}} B = A - q * B, & \text{if } 1 \leq B \end{array}$$

With the quotient we can compute the remainder with one additional multiplier and subtracter, hence our main focus is to compute the quotient. For the quotient we observe that:

$$\frac{A}{B} = A * \frac{1}{B}$$

A very fast solution is to create a lookup table with for each possible value of B the value of $1/B$. However, there are 131071 possible values of B. Such a table will be too large to be practical; the amount of storage on a FPGA is limited. Therefore, we investigate whether we can use a smaller table. A table with p-bits indices and elements of w bits gives rise to a table containing $2^p \times w$ bits. Our goal now is to find sufficiently small values for p and w, and still obtain a fast algorithm.

For the algorithm, it suffices to compute an approximation for q, named $q1$. The remainder of $q1$ is defined as: $r1 = A - q1 * B$. If $q1$ differs at most 1 from q, it is possible to obtain q from $r1$ with only two comparisons as follows:

$$\begin{array}{ll} & q-1 \leq q1 \leq q+1 \\ \equiv & \{\ q1 \text{ and } q \text{ are integers } \} \end{array}$$

$$
\begin{array}{ll}
 & q1 \ \in \ \{\ q{-}1,\ q,\ q{+}1\ \} \\
\equiv & \{\ \text{algebra}\ \} \\
 & A - q1 * B \ \in \ \{\ r{+}B,\ r,\ r{-}B\ \} \quad \underline{\text{whr}} \quad r = A - q * B \quad \underline{\text{end}} \\
\equiv & \{\ \text{definition remainder } r;\ \text{definition remainder } r1\ \} \\
 & r1 \ \in \ \{\ r{+}B,\ r,\ r{-}B\ \}
\end{array}
$$

Because $0 \leq r < B$, we obtain q and r from $q1$ and $r1$ as follows:

$r1$	q	r
$r1 < 0$	$q = q1 - 1$	$r = r1 + B$
$0 \leq r1 < B$	$q = q1$	$r = r1$
$B \leq r1$	$q = q1 + 1$	$r = r1 - B$

So, if $q1$ satisfies $q{-}1 \leq q1 \leq q{+}1$, then we consider $q1$ a good approximation for q. We first remove A out of this equation, where $q = A \ \underline{\text{div}}\ B$.

For the implementation of $q1$ we wish to use multipliers, hence we define $q1$ as $\lfloor A * x \rfloor$, for some x still to be chosen. Now we derive boundaries for x:

$$
\begin{array}{ll}
 & q - 1 \ \leq \ q1 \ \leq \ q + 1 \\
\equiv & \{\ \text{definition of } q \text{ and } q1\ \} \\
 & \left\lfloor \frac{A}{B} \right\rfloor - 1 \ \leq \ \lfloor A * x \rfloor \ \leq \ \left\lfloor \frac{A}{B} \right\rfloor + 1 \\
\equiv & \{\ \text{property '}\lfloor\ \rfloor\text{'}\ \} \\
 & \left\lfloor \frac{A}{B} - 1 \right\rfloor \ \leq \ \lfloor A * x \rfloor \ \leq \ \left\lfloor \frac{A}{B} + 1 \right\rfloor \\
\Leftarrow & \{\ \text{monotonicity of '}\lfloor\ \rfloor\text{'}\ \} \\
 & \frac{A}{B} - 1 \ \leq \ A * x \ \leq \ \frac{A}{B} + 1
\end{array}
$$

We have three cases for A: If $A = 0$, then the above proposition is always true, independent of x. For $A > 0$ the derivation continues as follows:

$$
\begin{array}{ll}
 & \frac{1}{B} - \frac{1}{A} \ \leq \ x \ \leq \ \frac{1}{B} + \frac{1}{A} \\
\Leftarrow & \{\ A < 2^{17}\ \} \\
 & \frac{1}{B} - \frac{1}{2^{17}} \ \leq \ x \ \leq \ \frac{1}{B} + \frac{1}{2^{17}}
\end{array}
$$

And, for $A < 0$:

$$
\begin{array}{ll}
 & \frac{A}{B} - 1 \ \leq \ A * x \ \leq \ \frac{A}{B} + 1 \\
\equiv & \{\ \text{algebra}\ \}
\end{array}
$$

$$
\begin{array}{ll}
 & -\left(\frac{A}{B}-1\right) \;\geq\; -(A*x) \;\geq\; -\left(\frac{A}{B}+1\right) \\
\equiv & \{\text{ algebra }\} \\
 & \frac{-A}{B}-1 \;\leq\; -A*x \;\leq\; \frac{-A}{B}+1 \\
\equiv & \{\; 1 \leq -A \;\} \\
 & \frac{1}{B}+\frac{1}{A} \;\leq\; x \;\leq\; \frac{1}{B}-\frac{1}{A} \\
\Leftarrow & \{\; -2^{17} \leq A \;\} \\
 & \frac{1}{B}-\frac{1}{2^{17}} \;\leq\; x \;\leq\; \frac{1}{B}+\frac{1}{2^{17}}
\end{array}
$$

By combining the cases for $A = 0$, $A < 0$ and $A > 0$, we obtain the following boundaries for x:

$$(2) \quad \frac{1}{B}-\frac{1}{2^{17}} \;\leq\; x \;\wedge\; x \;\leq\; \frac{1}{B}+\frac{1}{2^{17}}$$

So, the algorithm should compute some value x satisfying the boundaries of equation (2), before multiplying this value by A. For every applicable B we wish to obtain x by using a table of size $2^p \times w$. In the table we can only store $1/B$ for 2^p different values of B. So, we investigate for which subset of all applicable B's to store $1/B$. We observe that when B is small, the value $1/B$ is relatively large; a small change in B has a greater effect on value $1/B$. On the contrary, when B is large, a small change in B has less effect on $1/B$. The choice for the index of p bits should take this into account. Therefore, we choose as index the value of the p most significant digits of the binary representation of B. We wish to ignore the leading zeroes. Consequently, different B's with different amounts of leading zeroes in the binary representation could have the same significant digits, while the difference between the B's is large. Therefore we use *normalization*; we shift bs to the right until there are no more leading zeroes. As a result, the difference between all B's with the same significant digits is small after the normalization. In a later stage, we have to remove the amount of added bits to get the correct result.

To normalize B, we define:

$$Bk \;=\; B*2^k$$

where $0 \leq k \leq 16$ and $2^{16} \leq Bk < 2^{17}$. This k exists and is unique, because $2^0 \leq B < 2^{17}$. We define $Bk = B*2^k$. Because $2^{16} \leq Bk \leq 2^{17}-1$, the last bit—at position 17—of the binary representation of Bk always is 1. Hence, the binary representation of Bk is a list of the form: [b_0 b_1 ... b_{14} b_{15} 1]. As index in the table we use bits [b_{16-p} ... b_{14} b_{15}]. The right-most bit always equals 1, hence this bit gives no information: there is no point in using this constant bit in the index. The value of those p bits is named β and the value of the remaining $16-p$ bits is named ϵ. An example how to obtain β and ϵ is shown in Figure 6.1. The values β and ϵ are defined by:

$$Bk \;=\; 2^{16} + \beta * 2^{16-p} + \epsilon \qquad \text{, with } 0 \le \beta < 2^p \text{ and } 0 \le \epsilon < 2^{16-p}$$

$$\begin{aligned} B &= [\ 0101\ 0011\ \ 1011\ \underrightarrow{0000\ 0}\ ] \\ Bk &= [\ \underbrace{0000\ 0010\ 10}_{\epsilon}\underbrace{01\ 1101}_{\beta}\ 1\ ] \end{aligned}$$

Figure 6.1: obtaining β and ϵ for a given $B = 3530$ and $p = 6$

Now, we wish that value x only depends on β and k, not on ϵ. Hence, we introduce a function f, with β and k as parameters only and choose $x = f \cdot \beta \cdot k$. In the next section we derive a function f, which satisfies:

$$\frac{1}{B} - \frac{1}{2^{17}} \;\le\; f{\cdot}\beta{\cdot}k \quad\wedge\quad f{\cdot}\beta{\cdot}k \;\le\; \frac{1}{B} + \frac{1}{2^{17}}$$

for all B, that correspondends with these k and β.

The above equation requires $7 \le p \le 16$, see Appendix 9.1. For these values of p, we know a function f exists that satisfies the above equation. Although the minimal value of p is 7, we start with $p = 8$. Using this larger table size an algorithm is produced that is both simpler and faster. Moreover, the analysis is easier to follow.

6.1 Derivation for $p = 8$

We have shown that from an approximation, $q1$, the quotient q can be computed. To calculate $q1$, we multiply A by a value computed by function f with arguments β and k, both unique variables for a given B. It is important that the value given by f is computed as fast as possible, while it is precise enough to calculate $q1$.

In function f we use a $2^p \times w$ table. Here, we investigate the case where $p = 8$. Now β and ϵ, with $0 \le \beta < 2^8$ and $0 \le \epsilon < 2^8$, are defined as follows:

$$(3) \quad B * 2^k \;=\; 2^{16} + \beta * 2^8 + \epsilon$$

We derive function f, with arguments β and ϵ, such that

$$(4) \quad \frac{1}{B} - \frac{1}{2^{17}} \;\le\; f{\cdot}\beta{\cdot}k \quad\wedge\quad f{\cdot}\beta{\cdot}k \;\le\; \frac{1}{B} + \frac{1}{2^{17}}$$

is satisfied for all B, that correspondends with these k and β. We derive the smallest value for $f \cdot \beta \cdot k$ that is at least the lower bound. Then, we prove the correctness for the upper boundary. First we eliminate ϵ as follows:

$$
\begin{array}{ll}
 & \frac{1}{B} - \frac{1}{2^{17}} \;\leq\; f \cdot \beta \cdot k \\
\equiv & \{ \text{ equation (3) } \} \\
 & \frac{2^k}{2^{16} + 2^8 * \beta + \epsilon} - \frac{1}{2^{17}} \;\leq\; f \cdot \beta \cdot k \\
\Leftarrow & \{ \text{ the left-hand side is maximal if } \epsilon = 0 \;\} \\
 & \frac{2^k}{2^{16} + 2^8 * \beta} - \frac{1}{2^{17}} \;\leq\; f \cdot \beta \cdot k
\end{array}
$$

This left-hand side is a candidate for $f \cdot \beta \cdot k$. However, we wish to remove the division. We continue as follows:

$$
\begin{array}{ll}
 & \frac{2^k}{2^{16} + 2^8 * \beta} - \frac{1}{2^{17}} \\
= & \{ \text{ algebra } \} \\
 & 2^{k-8} * \frac{1}{2^8 + \beta} - \frac{1}{2^{17}}
\end{array}
$$

We use a table, which contains the value of $\frac{1}{2^8 + \beta}$ as accurately as needed.

$$
\begin{array}{ll}
 & 0 \;\leq\; \beta \;<\; 2^8 \\
\equiv & \{ \text{ algebra } \} \\
 & 2^8 \;\leq\; 2^8 + \beta \;<\; 2^9 \\
\equiv & \{ \text{ algebra } \} \\
 & 2^{-9} \;<\; \frac{1}{2^8 + \beta} \;\leq\; 2^{-8}
\end{array}
$$

We choose the value of w to be 16, thus the elements in the table consist of 16 bits.

$$
\begin{array}{ll}
\equiv & \{ \text{ algebra } \} \\
 & 2^{16} \;<\; \frac{2^{25}}{2^8 + \beta} \;\leq\; 2^{17} \\
\equiv & \{ \text{ algebra } \} \\
 & 0 \;<\; \frac{2^{25}}{2^8 + \beta} - 2^{16} \;\leq\; 2^{16}
\end{array}
$$

This value is not necessarily an integer. By rounding down we obtain:

$$0 \;\leq\; 2^{25} \;\underline{\text{div}}\; (2^8 + \beta) - 2^{16} \;\leq\; 2^{16}$$

These values, except when $\beta = 0$, are representable in 16 bits. In the case $\beta = 0$, we store $2^{16} - 1$ instead and accept the additional error. Now the lookup table of 256×16 is defined as follows:

$$\begin{aligned} tab{\cdot}\beta &= 2^{25} \operatorname{\underline{div}} (2^8{+}\beta) - 2^{16} && \text{, for } 1 \le \beta < 2^8 \\ tab{\cdot}\beta &= 2^{16} - 1 && \text{, for } \beta = 0 \end{aligned}$$

We use this definition of tab in the derivation of f, as follows:

$$\begin{aligned} & 2^{k-8} * \frac{1}{2^8 + \beta} - \frac{1}{2^{17}} \\ = \quad & \{ \text{ algebra } \} \\ & \frac{\dfrac{2^{25}}{2^8 + \beta} - 2^{16-k}}{2^{33-k}} \\ = \quad & \{ \text{ algebra } \} \\ & \frac{\dfrac{2^{25}}{2^8 + \beta} - 2^{16} + 2^{16} - 2^{16-k}}{2^{33-k}} \\ \le \quad & \{ \; x \le \lfloor x \rfloor + 1 \; \} \\ & \frac{\left\lfloor \dfrac{2^{25}}{2^8 + \beta} \right\rfloor - 2^{16} + 1 + 2^{16} - 2^{16-k}}{2^{33-k}} \\ \le \quad & \{ \; 1 \le 2^{16-k} \; \} \\ & \frac{\left\lfloor \dfrac{2^{25}}{2^8 + \beta} \right\rfloor - 2^{16} + 2^{16}}{2^{33-k}} \end{aligned}$$

Now we have two cases.

Case $1 \le \beta$:

$$\begin{aligned} = \quad & \{ \text{ definition } tab{\cdot}\beta \text{ with } 1 \le \beta \; \} \\ & \frac{tab{\cdot}\beta + 2^{16}}{2^{33-k}} \\ = \quad & \{ \text{ choose } f{\cdot}\beta{\cdot}k = \frac{tab{\cdot}\beta + 2^{16}}{2^{33-k}} \; \} \\ & f{\cdot}\beta{\cdot}k \end{aligned}$$

Case $\beta = 0$:

$$\begin{aligned} = \quad & \{ \; \beta \; = \; 0 \; \} \\ & \frac{\left\lfloor \dfrac{2^{25}}{2^8} \right\rfloor - 2^{16} + 2^{16}}{2^{33-k}} \\ = \quad & \{ \text{ algebra } \} \end{aligned}$$

$$
\begin{array}{ll}
 & \frac{2^{16} + 2^{16}}{2^{33-k}} \\
= & \{ \text{ definition } tab \cdot \beta \text{ with } \beta = 0 \ \} \\
 & \frac{tab{\cdot}\beta + 1 + 2^{16}}{2^{33-k}} \\
\geq & \{ \text{ remove the } +1 \text{; additional proof requirement (see below) } \} \\
 & \frac{tab{\cdot}\beta + 2^{16}}{2^{33-k}} \\
= & \{ \text{ choose } f{\cdot}\beta{\cdot}k = \frac{tab{\cdot}\beta + 2^{16}}{2^{33-k}} \ \} \\
 & f{\cdot}\beta{\cdot}k
\end{array}
$$

To justify the removal of the $+1$, we need to prove for $\beta = 0$ that the following inequality still holds:

$$\frac{2^k}{2^{16} + 2^8 * \beta} - \frac{1}{2^{17}} \ \leq \ f{\cdot}\beta{\cdot}k$$

The proof is in Appendix 9.2.

* * *

We have now derived a function f with a 256×16 table, such that $\frac{1}{B} - \frac{1}{2^{17}} \leq f \cdot \beta \cdot k$ for any B. In Appendix 9.4 it is verified that $f \cdot \beta \cdot k \leq \frac{1}{B} + \frac{1}{2^{17}}$ holds for any B. As a result, we know that $q - 1 \leq \lfloor A * f{\cdot}\beta{\cdot}k \rfloor \leq q + 1$. Hence, we define $q1$ as follows:

Definition 32:

$$
\begin{array}{lllll}
q1 & = & \lfloor A * f{\cdot}\beta{\cdot}k \rfloor & & \\
 & & \underline{\text{whr}} \quad f{\cdot}\beta{\cdot}k & = & \frac{tab{\cdot}\beta + 2^{16}}{2^{33-k}} \\
 & & \& \quad tab{\cdot}\beta & = & 2^{25} \ \underline{\text{div}} \ (2^8 + \beta) - 2^{16} \qquad \text{, for } 1 \leq \beta < 2^8 \\
 & & \& \quad tab{\cdot}\beta & = & 2^{16} - 1 \qquad\qquad\qquad\qquad\quad \text{, for } \beta = 0
\end{array}
$$

In Section 6.3 we implement this definition as a Verilog program.

6.2 Derivation for $p = 7$

We have now derived a definition for computing $q1$ with $p = 8$. Here, we investigate the case $p = 7$. The values β and ϵ, with $0 \leq \beta < 2^7$ and $0 \leq \epsilon < 2^9$, are defined as follows:

$$(3) \quad B * 2^k \ = \ 2^{16} + \beta * 2^9 + \epsilon$$

Similar to Section 6.1, we derive function f, with arguments β and ϵ, such that:

$$(4) \quad \frac{1}{B} - \frac{1}{2^{17}} \;\leq\; f{\cdot}\beta{\cdot}k$$

is satisfied for all B, that correspondends with these k and β. We derive the smallest value for $f \cdot \beta \cdot k$ by first eliminating ϵ as follows:

$$\begin{array}{ll} & \frac{1}{B} - \frac{1}{2^{17}} \;\leq\; f{\cdot}\beta{\cdot}k \\ \equiv & \{ \text{ equation (3) } \} \\ & \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon} - \frac{1}{2^{17}} \;\leq\; f{\cdot}\beta{\cdot}k \\ \Leftarrow & \{ \text{ the left-hand side is maximal if } \epsilon = 0 \;\; \} \\ & \frac{2^k}{2^{16} + 2^9 * \beta} - \frac{1}{2^{17}} \;\leq\; f{\cdot}\beta{\cdot}k \end{array}$$

This left-hand side is a candidate for $f \cdot \beta \cdot k$. However, the left-hand side still contains a division, that we wish to remove. We continue as follows:

$$\begin{array}{ll} & \frac{2^k}{2^{16} + 2^9 * \beta} - \frac{1}{2^{17}} \\ = & \{ \text{ algebra } \} \\ & 2^{k-9} * \frac{1}{2^7 + \beta} - \frac{1}{2^{17}} \end{array}$$

We use a table containing the value of $\frac{1}{2^7 + \beta}$ as accurately as needed.

$$\begin{array}{ll} & 0 \;\leq\; \beta \;<\; 2^7 \\ \equiv & \{ \text{ algebra; we use similar steps of Section 6.1 with } w = 13 \; \} \\ & 0 \;<\; \frac{2^{21}}{2^7 + \beta} - 2^{13} \;\leq\; 2^{13} \\ \equiv & \{ \text{ rounding down } \} \\ & 0 \;\leq\; 2^{21} \;\underline{\text{div}}\; (2^7{+}\beta) - 2^{13} \;\leq\; 2^{13} \end{array}$$

These values are representable in 13 bits, except for $\beta = 0$. In this case we store $2^{13}-1$ instead and accept the additional error. Now the lookup table of 128×13 is defined as follows:

$$\begin{array}{lll} tab{\cdot}\beta & = & 2^{21} \;\underline{\text{div}}\; (2^7{+}\beta) - 2^{13} \qquad \text{, for } 1 \leq \beta < 2^7 \\ tab{\cdot}\beta & = & 2^{13} - 1 \qquad\qquad\qquad\qquad\quad \text{, for } \beta = 0 \end{array}$$

We use this definition of tab in the derivation of f, as follows:

$$
\begin{array}{ll}
 & 2^{k-9} * \dfrac{1}{2^7 + \beta} - \dfrac{1}{2^{17}} \\
= & \{ \text{ algebra } \} \\
 & \dfrac{\dfrac{2^{21}}{2^7 + \beta} - 2^{13-k}}{2^{30-k}} \\
= & \{ \text{ algebra } \} \\
 & \dfrac{\dfrac{2^{21}}{2^7 + \beta} - 2^{13} + 2^{13} - 2^{13-k}}{2^{30-k}} \\
\leq & \{ \; x \leq \lfloor x \rfloor + 1 \; \} \\
 & \dfrac{\left\lfloor \dfrac{2^{21}}{2^7 + \beta} \right\rfloor - 2^{13} + 1 + 2^{13} - 2^{13-k}}{2^{30-k}}
\end{array}
$$

Now we have two cases.

Case $1 \leq \beta$:

$$
\begin{array}{ll}
= & \{ \text{ definition } tab{\cdot}\beta \text{ with } 1 \leq \beta \; \} \\
 & \dfrac{tab{\cdot}\beta + 1 + 2^{13} - 2^{13-k}}{2^{30-k}} \\
= & \{ \; 2^{13-k} \text{ is not an integer; algebra } \} \\
 & \dfrac{2^3 * tab{\cdot}\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \\
= & \{ \text{ choose } f{\cdot}\beta{\cdot}k = \dfrac{2^3 * tab{\cdot}\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \; \} \\
 & f{\cdot}\beta{\cdot}k
\end{array}
$$

Case $\beta = 0$:

$$
\begin{array}{ll}
= & \{ \; \beta \; = \; 0 \; \} \\
 & \dfrac{\left\lfloor \dfrac{2^{21}}{2^7} \right\rfloor - 2^{13} + 1 + 2^{13} - 2^{13-k}}{2^{30-k}} \\
= & \{ \text{ algebra } \} \\
 & \dfrac{2^{13} + 1 + 2^{13} - 2^{13-k}}{2^{30-k}} \\
\geq & \{ \text{ remove the } +1 \text{; additional proof requirement (see below) } \}
\end{array}
$$

$$
\begin{array}{ll}
 & \dfrac{2^{13} + 2^{13} - 2^{13-k}}{2^{30-k}} \\
= & \{ \text{ definition } tab{\cdot}\beta \text{ with } \beta = 0 \ \} \\
 & \dfrac{tab{\cdot}\beta + 1 + 2^{13} - 2^{13-k}}{2^{30-k}} \\
= & \{ \ 2^{13-k} \text{ is not an integer; algebra } \} \\
 & \dfrac{2^3 * tab{\cdot}\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \\
= & \{ \text{ choose } f{\cdot}\beta{\cdot}k = \dfrac{2^3 * tab{\cdot}\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \ \} \\
 & f{\cdot}\beta{\cdot}k
\end{array}
$$

To justify the removal of the $+1$, we need to prove for $\beta = 0$ that the following inequality still holds:

$$\frac{2^k}{2^{16} + 2^9 * \beta} - \frac{1}{2^{17}} \ \leq \ f{\cdot}\beta{\cdot}k$$

The proof is in Appendix 9.2.

* * *

We have now derived a function f with a 128×13 table, such that $\frac{1}{B} - \frac{1}{2^{17}} \leq f \cdot \beta \cdot k$ for any B. In Appendix 9.5 it is verified that $f \cdot \beta \cdot k \leq \frac{1}{B} + \frac{1}{2^{17}}$ holds for any B. As a result, we know that $q - 1 \leq \lfloor A * f{\cdot}\beta{\cdot}k \rfloor \leq q + 1$. Hence, we define $q1$ as follows:

Definition 33:

$$
\begin{array}{lllll}
q1 & = & \lfloor A * f{\cdot}\beta{\cdot}k \rfloor & & \\
 & & \underline{\text{whr}} \quad f{\cdot}\beta{\cdot}k \ = \ \dfrac{2^3 * tab{\cdot}\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} & & \\
 & & \& \quad tab{\cdot}\beta \ = \ 2^{21} \ \underline{\text{div}} \ (2^7 + \beta) - 2^{13} & , \text{ for } 1 \leq \beta < 2^7 \\
 & & \& \quad tab{\cdot}\beta \ = \ 2^{13} - 1 & , \text{ for } \beta \ = \ 0
\end{array}
$$

In the next section we implement this definition as a Verilog program.

6.3 Verilog implementation with a larger table

First we implement the version with the 256×16 table as described in Section 6.1.

To implement $q1$, we first obtain value β from bs, the binary representing of B. We use the module *normalizeS* as developed by R.R. Hoogerwoord (see Appendix 9.6), which returns the list bk from the input bs. The bits of bk from the second position to the ninth position are β. We use β as index for the table to retrieve $tab \cdot \beta$. The list $tab \cdot \beta$ contains 16 bits. To add 2^{16} to the binary representation of this list, we add a 1 as significant bit (property 10). This is list c in the Verilog program. Next we multiply list c with as, the two's complement representation of A, using a 18×18 multiplier. Therefore, the length of c should be 18, which is obtained by adding a 0 as most-significant bit.

After the multiplication, we still have to divide the result by 2^{33-k} and round it down to obtain $q1$. This is done by doing a bit shift of $33-k$ places to the right, where a bit shift of x bits to the right means, that the least significant x bits are removed from the list and x bits with value 0 are inserted at the most significant side. Likewise, a bit shift of x bits to the left means that x bits with value 0 are inserted as the least significant bits; thus, it multiplies the value of the list by 2^x. To perform a bitshift of $33-k$, we first shift 33 places to the right and then, by using the module *normalizeS*, shift it k places to the left. After the bitshift we reconstruct q from $q1$ by calculating the remainder of $q1$ and compare it to the value B and 0 as mentioned at the start of this chapter.

* * *

We construct the following Verilog program:

```
//////////////////////////////////////////////////////////////////////
//
// created by: Wouter de Koning
//
// This Verilog program implements the division by multiplication
// using a table with 256 x 16 bits
//
// as, bs, rs and qs are in two's complement representation (18 bits)
//
// Dependencies: DivTable256x16, NormalizeS
//
//////////////////////////////////////////////////////////////////////

module DivModMultiply256(as, bs, qs, rs) ;
  input [17:0] as, bs ;
 output [17:0] qs, rs ;

   wire [16:0] bk     ;
   wire [ 7:0] beta   ;
   wire [15:0] tab    ; // value of tab[beta]
   wire [17:0] c      ; // value of tab[beta] + 2^16
```

```
   wire [35:0] amulc  ;
   wire [17:0] preq1  ; // before shifting k bits to left
   wire [33:0] postq1 ; // after shifting k bits to left
   wire [35:0] q1mulb ;
   wire [17:0] q1     ;
   wire [18:0] r1     ;

   wire [18:0] aminb  ;
   wire [18:0] aplsb  ;
   wire [17:0] q1min1 ;
   wire [17:0] q1pls1 ;
   wire [18:0] r1minb ;
   wire [18:0] r1plsb ;

   NormalizeS NormB(bs[16:0],bk,preq1,postq1) ;

   assign beta = bk[15:8] ;

  // -- DivTable returns the value of tab[beta]
   DivTable256x16 Ctab(beta,tab) ;

   assign c = {{2'b01},tab} ;

   MULT18X18 atimc(amulc,as,c) ;

   assign  preq1 = amulc[34:17];

  // -- postq1 is calculated in NormB
   assign     q1 = postq1[33:16];

   MULT18X18 QtimB(q1mulb,q1,bs);

   assign  aminb = {{as[17]},{as}} - {{1'b0},{bs}};
   assign  aplsb = {{as[17]},{as}} - {{1'b0},{bs}};
   assign     r1 = {{as[17]},{as}} - q1mulb[18:0];
   assign r1minb = aminb - q1mulb[18:0];
   assign r1plsb = aplsb - q1mulb[18:0];

  // -- in the case q1min1 or q1pls1 overflows it is not used in the final step
   assign q1min1 = q1 - 1;
   assign q1pls1 = q1 + 1;

   assign qs = ( r1[18] ? q1min1 : ( r1minb[18] ? q1 : q1pls1 ) ) ;
   assign rs = ( r1[18] ? r1plsb[17:0] : ( r1minb[18] ? r1[17:0] : r1minb[17:0] ) ) ;

endmodule
```

6.4 Verilog implementation with a smaller table

Now we implement the version with the 128×13 table as described in Section 6.2.

The program is very similar to the version with the 256×16 table. The implementation of the list c is a bit different. The multiplication by 2^3 is implemented with 3 zeroes as least-significant bits. Furthermore, we add the value $8 - 2^{16-k}$ to c, which is implemented in the module *NormalizeX* as described in Appendix 9.7.

* * *

We construct the following Verilog program:

```
//////////////////////////////////////////////////////////////////////
//
// created by: Wouter de Koning
//
// This Verilog program implements the division by multiplication,
// using a table with 128 x 13 bits
//
// as, bs, rs and qs are in two's complement representation (18 bits)
//
// Dependencies: DivTable128x13, NormalizeX
//
//////////////////////////////////////////////////////////////////////

module DivModMul128(as, bs, qs, rs) ;
  input [17:0] as, bs ;
 output [17:0] qs, rs ;

   wire [16:0] bk     ;
   wire [ 6:0] beta   ;
   wire [12:0] tab    ; // value of tab[beta]
   wire [17:0] c      ; // value of tab[beta] + 2^16
   wire [35:0] amulc  ;
   wire [17:0] preq1  ; // before shifting k bits to left
   wire [33:0] postq1 ; // after shifting k bits to left
   wire [35:0] q1mulb ;
   wire [17:0] q1     ;
   wire [18:0] r1     ;
   wire [17:0] ex     ; // value of 8 - 2^(16-k)

   wire [18:0] aminb  ;
   wire [18:0] aplsb  ;
   wire [17:0] q1min1 ;
```

```
   wire [17:0] q1pls1 ;
   wire [18:0] r1minb ;
   wire [18:0] r1plsb ;

   NormalizeX NormB(bs[16:0],bk,preq1,postq1,ex) ;

   assign beta = bk[15:9] ;

  // -- DivTable returns the value of tab[beta]
   DivTable128x13 Ctab(beta,tab) ;

  // -- ex is calculated in NormB
   assign c = {{2'b01},tab,{3'b000}} + ex ;

   MULT18X18 atimc(amulc,as,c) ;

   assign  preq1 = amulc[34:17];

  // -- postq1 is calculated in NormB
   assign     q1 = postq1[33:16];

   MULT18X18 QtimB(q1mulb,q1,bs);

   assign  aminb = {{as[17]},{as}} - {{1'b0},{bs}};
   assign  aplsb = {{as[17]},{as}} - {{1'b0},{bs}};
   assign     r1 = {{as[17]},{as}} - q1mulb[18:0];
   assign r1minb = aminb - q1mulb[18:0];
   assign r1plsb = aplsb - q1mulb[18:0];

  // -- in the case q1min1 or q1pls1 overflows it is not used in the final step
   assign q1min1 = q1 - 1;
   assign q1pls1 = q1 + 1;

   assign qs = ( r1[18] ? q1min1 : ( r1minb[18] ? q1 : q1pls1 ) ) ;
   assign rs = ( r1[18] ? r1plsb[17:0] : ( r1minb[18] ? r1[17:0] : r1minb[17:0] ) ) ;

endmodule
```

Chapter 7

Results and Discussion

In the previous chapters we have derived five circuits: restoring division, non-restoring division, radix-4 division and two variants of division by multiplication. We have obtained the performance of these five circuits by compiling them on a Spartan-3 Generation FPGA. Specifically, we are interested in the propagation delay and area requirement in amount of LUTs for each circuit. A LUT is a LookUp Table with four inputs and one output, the standard building block of the Spartan-3 Generation FPGAs. The amount of LUTs used on a FPGA gives us a good measure for the area requirement. We obtain the propagation delays and area requirements from the synthesis reports, produced by the Xilinx compiler. The results are shown in Table 7.1.

Verilog program	Maximum delay (in ns)	Number of 4 input LUTs
DivModRestoring	88	629
DivModNonRestoring	75	441
DivModRadix4	54	975
DivModMultiply256	32	622
DivModMultiply128	33	538

Table 7.1: propagation delay and area requirement for the Verilog programs

Non-restoring division is faster than restoring division. When we take a closer look at the synthesis reports of both circuits, the difference in delay can be explained by the fact that each recursive step of the non-restoring version takes approximately 4.2 ns and the restoring version approximately 5.1 ns. Hence, the recursive steps of the non-restoring division are implemented more efficiently than the steps of the restoring division.

Radix-4 division has fewer but more complicated recursive steps than the (non-)restoring versions. As expected, radix-4 division is a lot faster, but also requires more LUTs. From the synthesis reports we obtain a propagation delay of approximately 6.2 ns for each recursive step. The relatively low delay for this complicated step can be explained by the fact that several computations are executed in parallel.

The multiplication circuits give better results than the recursive circuits. This is because these circuits do not need to perform multiple additions/subtractions. The multiplication with the larger table has a slightly lower delay than the one using the smaller table. The circuit with the smaller table has one additional adder, which adds more delay than the speed-up gained by using the smaller table. However, the circuit with the smaller table needs fewer LUTs to store the actual table.

For the multiplication circuits, we also tried different versions of the normalize module. We were able to speed-up the normalize module, such that the propagation delay of the multiplication circuit with the larger table decreases with 5%. However, the amount of LUTs increases with 73%, which is a relatively large increase for a small decrease in delay. Therefore we did not include these versions for the normalize module in this thesis.

Chapter 8

Conclusion

We have derived five circuits for dividing integers using a calculational approach. The calculational style of reasoning is a very convenient method to derive algorithms. It is immediately verified that the algorithms are correct. However, at all times one needs to be careful, mistakes are easily made. One could derive other algorithms by starting with a different specification or making different design decisions in the process. For example, if one chooses to use the specification of the restoring division and formulate the solution for $16 * A + b$ and B with $0 \leq b \leq 15$, recursively in terms of A and B, instead of $2 * A + b$ with $0 \leq b \leq 1$, one probably would obtain a radix 16 algorithm.

We only derived algorithms of the classes digit recurrence (subtractive method) and table based functions as mentioned in the introduction. It is worthwile to derive algorithms from the functional iteration methods as described in [7]. The functional iteration method utilizes the multiplication as the fundamental operator, instead of the subtractor. By using this multiplication, in each recursive step the number of correct quotient bits is doubled, while in the digit recurrence method the amount of correct quotient bits is increased with a constant. In theory, this should be faster than the digit recurrence, especially with the fast dedicated multipliers as provided by the Spartan-3 Generation FPGAs.

From all the algorithms we have derived, the multiplication method is by far the most interesting. The main disadvantage of our multiplication method is that the input and output lists are fixed to 18 bits. It is interesting to investigate how to extend this method, such that it works with lists containing more bits and what the effect is on the maximal propagation delay and area requirement. It could be possible to use a hybrid version of two methods, such as first using a few recursive steps of radix 4, before using the multiplication method, and then to combine the results to obtain a division method for more than 18 bits.

Chapter 9

Appendices

9.1 Appendix: Derive range of p

Derive the range of p, such that $\frac{1}{B1} - \frac{1}{2^{17}} \leq \frac{1}{B2} + \frac{1}{2^{17}}$ holds for any $\beta, \epsilon_1, \epsilon_2$ and k

$$\begin{array}{lrcl} \text{where} & B1 & = & (\ 2^{16} + 2^{16-p} * \beta + \epsilon_1\ ) * 2^{-k} \\ \& & B2 & = & (\ 2^{16} + 2^{16-p} * \beta + \epsilon_2\ ) * 2^{-k} \end{array}$$

The following ranges are given for integers $k, p, \epsilon_1, \epsilon_2$ and β:

- $0 \leq k \leq 16$
- $p \leq 16$
- $0 \leq \epsilon_1 \leq \epsilon_{max} < 2^{16-p}$
- $0 \leq \epsilon_2 \leq \epsilon_{max} < 2^{16-p}$
- $0 \leq \beta < 2^p$

First we derive a definition for ϵ_{max} in terms of p and k. We know from the definition of $B1$ and $B2$ that 2^k is a divisor of ϵ_{max}. As a result, if $\epsilon_{max} < 2^k$, then the only possible value for ϵ_{max} is 0. We know from $\epsilon_{max} < 2^{16-p}$, that $16 - p \leq k$ is satisfied. Otherwise, if $k < 16 - p$, then ϵ_{max} is the largest number smaller than 2^{16-p}, which is divisble by 2^k, hence $\epsilon_{max} = 2^{16-p} - 2^k$. In summary, we get the following definition for ϵ_{max}:

$$\begin{array}{lcll} \epsilon_{max} & = & 2^{16-p} - 2^k & , \text{for } k < 16-p \\ \epsilon_{max} & = & 0 & , \text{for } 16-p \leq k \end{array}$$

* * *

We solve the inequality as follows:

$$
\begin{array}{ll}
 & \frac{1}{B1} - \frac{1}{2^{17}} \;\leq\; \frac{1}{B2} + \frac{1}{2^{17}} \\
\equiv & \{ \text{ algebra } \} \\
 & \frac{1}{B1} - \frac{1}{B2} \;\leq\; \frac{1}{2^{16}} \\
\equiv & \{ \text{ definition } B1 \text{ and } B2 \} \\
 & \frac{2^k}{2^{16} + 2^{16-p} * \beta + \epsilon_1} - \frac{2^k}{2^{16} + 2^{16-p} * \beta + \epsilon_2} \;\leq\; \frac{1}{2^{16}} \\
\Leftarrow & \{ \text{ choose minimal } \epsilon_1 \text{ and maximal } \epsilon_2 \} \\
 & \frac{2^k}{2^{16} + 2^{16-p} * \beta} - \frac{2^k}{2^{16} + 2^{16-p} * \beta + \epsilon_{max}} \;\leq\; \frac{1}{2^{16}} \\
\equiv & \{ \text{ algebra } \} \\
 & 2^k + \frac{2^k * \epsilon_{max}}{2^{16} + 2^{16-p} * \beta} - 2^k \;\leq\; 1 + 2^{-p} * \beta + \frac{\epsilon_{max}}{2^{16}} \\
\equiv & \{ \text{ algebra } \} \{ \text{ worst case if } \beta = 0 \} \\
 & \frac{2^k * \epsilon_{max}}{2^{16}} \;\leq\; 1 + \frac{\epsilon_{max}}{2^{16}} \\
\equiv & \{ \text{ algebra } \} \\
 & 2^k * \epsilon_{max} - \epsilon_{max} \;\leq\; 2^{16}
\end{array}
$$

If $16-p \leq k$, then $\epsilon_{max} = 0$, hence the proposition holds for any p. If $k < 16-p$, the proof continues as follows:

$$
\begin{array}{ll}
 & 2^k * (2^{16-p} - 2^k) - (2^{16-p} - 2^k) \;\leq\; 2^{16} \\
\Leftarrow & \{ \text{ worst case: } k \text{ is maximal, hence } k = 16-p-1 \} \\
 & 2^{16-p-1} * (2^{16-p} - 2^{16-p-1}) - (2^{16-p} - 2^{16-p-1}) \;\leq\; 2^{16} \\
\equiv & \{ \text{ algebra } \} \\
 & 2^{15-p} * 2^{15-p} - 2^{15-p} \;\leq\; 2^{16}
\end{array}
$$

This is true if and only if $7 \leq p$. Hence, the total range of possible p values is $7 \leq p \leq 16$

9.2 Appendix: Justify the removal of the $+1$ for $p = 8$

To prove for $\beta = 0$: $\dfrac{2^k}{2^{16} + 2^8 * \beta} - \dfrac{1}{2^{17}} \;\leq\; f \cdot \beta \cdot k$

$$\begin{array}{rrcl} \text{where} & f{\cdot}\beta{\cdot}k & = & \dfrac{tab{\cdot}\beta \,+\, 2^{16}}{2^{33-k}} \\ \text{and} & tab \cdot 0 & = & 2^{16} - 1 \end{array}$$

The range for integer k is:

- $0 \leq k \leq 16$

* * *

First we simplify the proposition as follows:

$$\begin{array}{ll} & \dfrac{2^k}{2^{16} + 2^8 * 0} - \dfrac{1}{2^{17}} \;\leq\; f{\cdot}0{\cdot}k \\ \equiv & \{ \text{ algebra } \} \\ & \dfrac{2^k}{2^{16}} - \dfrac{1}{2^{17}} \;\leq\; f{\cdot}0{\cdot}k \end{array}$$

The function f is rewritten as:

$$\begin{array}{ll} & f \cdot 0 \cdot k \\ = & \{ \text{ definition } f \; \} \\ & \dfrac{tab \cdot 0 + 2^{16}}{2^{33-k}} \\ = & \{ \text{ definition } tab \; \} \\ & \dfrac{2^{16} - 1 + 2^{16}}{2^{33-k}} \\ = & \{ \text{ algebra } \} \\ & \dfrac{2^k}{2^{16}} - \dfrac{1}{2^{33-k}} \\ \leq & \{ \; 0 \leq k \leq 16 \; \} \\ & \dfrac{2^k}{2^{16}} - \dfrac{1}{2^{17}} \end{array}$$

Hence, the proposition holds.

9.3 Appendix: Justify the removal of the $+1$ for $p = 7$

To prove for $\beta = 0$: $\dfrac{2^k}{2^{16} + 2^9 * \beta} - \dfrac{1}{2^{17}} \leq f \cdot \beta \cdot k$

$$\begin{array}{rrcl} \text{where} & f{\cdot}\beta{\cdot}k & = & \dfrac{2^3 * tab{\cdot}\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \\ \text{and} & tab \cdot 0 & = & 2^{13} - 1 \end{array}$$

* * *

First we simplify the proposition as follows:

$$\begin{array}{ll} & \dfrac{2^k}{2^{16} + 2^9 * 0} - \dfrac{1}{2^{17}} \leq f{\cdot}0{\cdot}k \\ \equiv & \{ \text{ algebra } \} \\ & \dfrac{2^k}{2^{16}} - \dfrac{1}{2^{17}} \leq f{\cdot}0{\cdot}k \end{array}$$

The function f is rewritten as:

$$\begin{array}{ll} & f \cdot 0 \cdot k \\ = & \{ \text{ definition } f \text{ } \} \\ & \dfrac{2^3 * tab \cdot 0 + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \\ = & \{ \text{ algebra } \} \\ & \dfrac{tab \cdot 0 + 2^{13} + 1 - 2^{13-k}}{2^{30-k}} \\ = & \{ \text{ definition } tab \text{ } \} \\ & \dfrac{2^{13} - 1 + 2^{13} + 1 - 2^{13-k}}{2^{30-k}} \\ = & \{ \text{ algebra } \} \\ & \dfrac{2^k}{2^{16}} - \dfrac{1}{2^{17}} \end{array}$$

Hence, the proposition holds.

9.4 Appendix: The upper boundary for $f\cdot\beta\cdot k$, if $p = 8$

To prove: $f\cdot\beta\cdot k \;\leq\; \frac{1}{B} + \frac{1}{2^{17}}$

where $\quad f\cdot\beta\cdot k \quad = \quad \frac{tab\cdot\beta + 2^{16}}{2^{33-k}}$

The following ranges are given for integers B, k, ϵ and β:

- $1 \leq B < 2^{17}$
- $0 \leq k \leq 16$
- $0 \leq \epsilon \leq \epsilon_{max} < 2^8$
- $0 \leq \beta < 2^8$

We use the following definitions:

$$
\begin{array}{lll}
B & = (\ 2^{16} + 2^8 * \beta + \epsilon\ ) * 2^{-k} & \\
tab[\beta] & = 2^{25} \ \underline{\text{div}}\ (2^8 + \beta) - 2^{16} & \text{, for } 1 \leq \beta \\
tab[\beta] & = 2^{16} - 1 & \text{, for } \beta = 0 \\
\epsilon_{max} & = 2^8 - 2^k & \text{, for } k \leq 7 \\
\epsilon_{max} & = 0 & \text{, for } 8 \leq k
\end{array}
$$

* * *

To prove the inequality, we first rewrite it to a proposition, where the outcome only depends on the variables β and k. Then we prove that the proposition holds for $\beta = 0$ and any k, followed by a proof for $1 \leq \beta$ and any k. The proof starts as follows:

$$
\begin{array}{ll}
 & f\cdot\beta\cdot k \;\leq\; \frac{1}{B} + \frac{1}{2^{17}} \\
\equiv & \{\ \text{definition } f \text{ and definition } B\ \} \\
 & \frac{tab\cdot\beta + 2^{16}}{2^{33-k}} \;\leq\; \frac{2^k}{2^{16} + 2^8 * \beta + \epsilon} + \frac{1}{2^{17}} \\
\equiv & \{\ \text{algebra}\ \} \\
 & tab\cdot\beta + 2^{16} \;\leq\; \frac{2^{33}}{2^{16} + 2^8 * \beta + \epsilon} + 2^{16-k} \\
\Leftarrow & \{\ \text{worst case if } \epsilon = \epsilon_{max}\ \} \\
 & tab\cdot\beta + 2^{16} \;\leq\; \frac{2^{33}}{2^{16} + 2^8 * \beta + \epsilon_{max}} + 2^{16-k}
\end{array}
$$

Proof for $\beta = 0$:

$$
\begin{array}{ll}
 & tab \cdot 0 + 2^{16} \;\leq\; \dfrac{2^{33}}{2^{16} + 2^8 * 0 + \epsilon_{max}} + 2^{16-k} \\
\equiv & \{ \text{ definition } tab \text{ for } \beta = 0; \text{ algebra } \} \\
 & 2^{17} - 1 \;\leq\; \dfrac{2^{33}}{2^{16} + \epsilon_{max}} + 2^{16-k}
\end{array}
$$

If $8 \leq k$, then $\epsilon_{max} = 0$ and because $-1 \leq 2^{16-k}$ the proposition holds. If $k \leq 7$, the proof continues as follows:

$$
\begin{array}{ll}
\equiv & \{ \text{ algebra } \} \\
 & 2^{33} + 2^{17} * \epsilon_{max} - 2^{16} - \epsilon_{max} \;\leq\; 2^{33} + 2^{32-k} + 2^{16-k} * \epsilon_{max} \\
\Leftarrow & \{ \text{ algebra; } 0 \leq 2^{16-k} * \epsilon_{max} \; \} \\
 & 2^{17} * \epsilon_{max} - 2^{16} - \epsilon_{max} \;\leq\; 2^{32-k}
\end{array}
$$

Because $\epsilon_{max} < 2^8$ and $k \leq 7$, the proposition holds.

* * *

Proof for $1 \leq \beta$:

$$tab \cdot \beta + 2^{16} \;\leq\; \frac{2^{33}}{2^{16} + 2^8 * \beta + \epsilon_{max}} + 2^{16-k}$$

The left-hand side of the inequality can be rewritten as follows:

$$
\begin{array}{ll}
 & tab \cdot \beta + 2^{16} \\
= & \{ \text{ definition } tab \text{ for } 1 \leq \beta \; \} \\
 & 2^{25} \;\underline{\text{div}}\; (2^8 + \beta) - 2^{16} + 2^{16} \\
= & \{ \text{ algebra } \} \\
 & 2^{25} \;\underline{\text{div}}\; (2^8 + \beta) \\
\leq & \{ \; x \;\underline{\text{div}}\; y \;\leq\; x/y \; \} \\
 & \dfrac{2^{25}}{2^8 + \beta} \\
= & \{ \text{ algebra } \} \\
 & \dfrac{2^{33}}{2^{16} + 2^8 * \beta}
\end{array}
$$

Now the inequality is proven as follows:

$$
\begin{array}{ll}
 & tab \cdot \beta + 2^{16} \;\leq\; \dfrac{2^{33}}{2^{16} + 2^8 * \beta + \epsilon_{max}} + 2^{16-k} \\
\Leftarrow & \{ \text{ rewrite left-hand side } \} \\
 & \dfrac{2^{33}}{2^{16} + 2^8 * \beta} \;\leq\; \dfrac{2^{33}}{2^{16} + 2^8 * \beta + \epsilon_{max}} + 2^{16-k}
\end{array}
$$

If $8 \leq k$, then $\epsilon_{max} = 0$ and because $0 \leq 2^{16-k}$ the proposition holds. If $k \leq 7$, the proof continues as follows:

$$
\begin{array}{ll}
 & \dfrac{2^{33}}{2^{16} + 2^8 * \beta} \;\leq\; \dfrac{2^{33}}{2^{16} + 2^8 * \beta + \epsilon_{max}} + 2^{16-k} \\
\equiv & \{ \text{ multiply by } 2^{16} + 2^8 * \beta + \epsilon_{max} \; \} \\
 & 2^{33} + \dfrac{2^{33} * \epsilon_{max}}{2^{16} + 2^8 * \beta} \;\leq\; 2^{33} + 2^{32-k} + 2^{16-k} * (2^8 * \beta + \epsilon_{max}) \\
\Leftarrow & \{ \text{ algebra; } 0 \leq 2^{16-k} * (2^8 * \beta + \epsilon_{max}) \; \} \\
 & \dfrac{2^{33} * \epsilon_{max}}{2^{16} + 2^8 * \beta} \;\leq\; 2^{32-k} \\
\Leftarrow & \{ \; \epsilon_{max} < 2^8 \; \} \\
 & \dfrac{2^{41}}{2^{16} + 2^8 * \beta} \;\leq\; 2^{32-k}
\end{array}
$$

Because $\dfrac{2^{41}}{2^{16} + 2^8 * \beta} \leq 2^{25}$ and $2^{25} \leq 2^{32-k}$ for $k \leq 7$, the inequality holds.

9.5 Appendix: The upper boundary for $f\cdot\beta\cdot k$, $p = 7$

To prove: $f\cdot\beta\cdot k \;\leq\; \frac{1}{B} + \frac{1}{2^{17}}$

$$\text{where} \quad f\cdot\beta\cdot k \;=\; \frac{2^3 * tab\cdot\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}}$$

The following ranges are given for integers B, k, ϵ, β:

- $1 \leq B < 2^{17}$
- $0 \leq k \leq 16$
- $0 \leq \epsilon \leq \epsilon_{max} < 2^9$
- $0 \leq \beta < 2^7$

We use the following definitions:

$$\begin{array}{lcll} B & = & (\ 2^{16} + 2^9 * \beta + \epsilon\ ) * 2^{-k} & \\ tab[\beta] & = & 2^{21} \ \underline{\text{div}}\ (2^7 + \beta) - 2^{13} & \text{, for } 1 \leq \beta \\ tab[\beta] & = & 2^{13} - 1 & \text{, for } \beta = 0 \\ \epsilon_{max} & = & 2^9 - 2^k & \text{, for } k \leq 8 \\ \epsilon_{max} & = & 0 & \text{, for } 9 \leq k \end{array}$$

* * *

To prove the inequality, we first rewrite it to a proposition, which outcome only depends on the variables β and k. Then we prove that the proposition holds for $\beta = 0$ and any k, followed by a proof for $1 \leq \beta$ and any k. The proof starts as follows:

$$\begin{array}{ll} & f\cdot\beta\cdot k \;\leq\; \frac{1}{B} + \frac{1}{2^{17}} \\ \equiv & \{\ \text{definition } f \text{ and definition } B\ \} \\ & \frac{2^3 * tab\cdot\beta + 2^{16} + 8 - 2^{16-k}}{2^{33-k}} \;\leq\; \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon} + \frac{1}{2^{17}} \\ \equiv & \{\ \text{algebra}\ \} \\ & \frac{tab\cdot\beta + 1 + 2^{13}}{2^{30-k}} \;\leq\; \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon} + \frac{1}{2^{16}} \\ \Leftarrow & \{\ \text{worst case if } \epsilon = \epsilon_{max}\ \} \\ & \frac{tab\cdot\beta + 1 + 2^{13}}{2^{30-k}} \;\leq\; \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon_{max}} + \frac{1}{2^{16}} \end{array}$$

Proof for $\beta = 0$:

$$\begin{array}{ll} & \frac{tab \cdot 0 + 1 + 2^{13}}{2^{30-k}} \;\leq\; \frac{2^k}{2^{16} + 2^9 * 0 + \epsilon_{max}} + \frac{1}{2^{16}} \\ \equiv & \{ \text{ definition } tab \text{ with } \beta = 0 \; \} \\ & \frac{2^{13} - 1 + 1 + 2^{13}}{2^{30-k}} \;\leq\; \frac{2^k}{2^{16} + 2^9 * 0 + \epsilon_{max}} + \frac{1}{2^{16}} \\ \equiv & \{ \text{ algebra } \} \\ & \frac{2^k}{2^{16}} \;\leq\; \frac{2^k}{2^{16} + \epsilon_{max}} + \frac{1}{2^{16}} \end{array}$$

If $9 \leq k$, then $\epsilon_{max} = 0$ and because $0 \leq \frac{1}{2^{16}}$ the proposition holds. If $k \leq 8$, the proof continues as follows:

$$\begin{array}{ll} & \frac{2^k}{2^{16}} \;\leq\; \frac{2^k}{2^{16} + 2^9 - 2^k} + \frac{1}{2^{16}} \\ \equiv & \{ \text{ algebra } \} \\ & 1 \;\leq\; \frac{2^{16}}{2^{16} + 2^9 - 2^k} + \frac{1}{2^k} \\ \equiv & \{ \text{ algebra } \} \\ & 2^{16} + 2^9 - 2^k \;\leq\; 2^{16} + 2^{16-k} + 2^{9-k} - 1 \\ \equiv & \{ \text{ algebra } \} \\ & 2^9 - 2^k + 1 \;\leq\; 2^{16-k} + 2^{9-k} \end{array}$$

For $0 \leq k \leq 8$ it holds that $2^9 - 2^k \;\leq\; 2^{16-k}$ and $1 \;\leq\; 2^{9-k}$, thus the proposition holds.

* * *

Proof for $1 \leq \beta$:

$$\frac{tab \cdot \beta + 1 + 2^{13}}{2^{30-k}} \;\leq\; \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon_{max}} + \frac{1}{2^{16}}$$

The left-hand side of the inequality is rewritten as follows:

$$\begin{array}{ll} & \frac{tab \cdot \beta + 1 + 2^{13}}{2^{30-k}} \\ = & \{ \text{ definition } tab \text{ with } 1 \;\leq\; \beta \; \} \\ & \frac{2^{21} \;\underline{\text{div}}\; (2^7 + \beta) - 2^{13} + 1 + 2^{13}}{2^{30-k}} \\ \leq & \{ \; x \;\underline{\text{div}}\; y \;\leq\; x/y \; \} \end{array}$$

$$\frac{\frac{2^{21}}{2^7+\beta} - 2^{13} + 1 + 2^{13}}{2^{30-k}}$$

$=$ { algebra }

$$\frac{2^k}{2^{16} + 2^9 * \beta} + \frac{1}{2^{30-k}}$$

Now the inequality is proven as follows:

$$\frac{tab \cdot \beta + 1 + 2^{13}}{2^{30-k}} \leq \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon_{max}} + \frac{1}{2^{16}}$$

$\Leftarrow$ { rewrite left-hand side }

$$\frac{2^k}{2^{16} + 2^9 * \beta} + \frac{1}{2^{30-k}} \leq \frac{2^k}{2^{16} + 2^9 * \beta + \epsilon_{max}} + \frac{1}{2^{16}}$$

We have three different cases for k.

Case $k \leq 8$.
As a result, $\epsilon_{max} = 2^9 - 2^k$ and the proof continues as follows:

$$\frac{2^k}{2^{16} + 2^9 * \beta} + \frac{1}{2^{30-k}} \leq \frac{2^k}{2^{16} + 2^9 * \beta + 2^9 - 2^k} + \frac{1}{2^{16}}$$

$\equiv$ { multiply by $2^{16} + 2^9 * \beta + 2^9 - 2^k$ }

$$2^k + \frac{2^k * (2^9 - 2^k)}{2^{16} + 2^9 * \beta} + \frac{2^{16} + 2^9 * \beta + 2^9 - 2^k}{2^{30-k}} \leq 2^k + 1 + \frac{\beta}{2^7} + \frac{1}{2^7} - \frac{2^k}{2^{16}}$$

$\equiv$ { algebra }

$$\frac{2^k * (2^9 - 2^k)}{2^{16} + 2^9 * \beta} + \frac{2^k}{2^{14}} + \frac{2^k * \beta}{2^{21}} + \frac{2^k}{2^{21}} - \frac{2^k * 2^k}{2^{30}} + \frac{2^k}{2^{16}} \leq 1 + \frac{\beta}{2^7} + \frac{1}{2^7}$$

$\Leftarrow$ { worst case, $\beta = 1$ }

$$\frac{2^k * (2^9 - 2^k)}{2^{16} + 2^9} + \frac{2^k}{2^{14}} + \frac{2^k}{2^{21}} + \frac{2^k}{2^{21}} - \frac{2^k * 2^k}{2^{30}} + \frac{2^k}{2^{16}} \leq 1 + \frac{1}{2^6}$$

$\Leftarrow$ { worst case, $k = 8$ }

$$\frac{2^{16}}{2^{16} + 2^9} + \frac{1}{2^6} + \frac{1}{2^{13}} + \frac{1}{2^{13}} - \frac{1}{2^{14}} + \frac{1}{2^8} \leq 1 + \frac{1}{2^6}$$

Verification shows that this inequality is true.

Case $9 \leq k \leq 14$.
As a result, $\epsilon_{max} = 0$ and the proof continues as follows:

$$\frac{2^k}{2^{16} + 2^9 * \beta} + \frac{1}{2^{30-k}} \leq \frac{2^k}{2^{16} + 2^9 * \beta + 0} + \frac{1}{2^{16}}$$

$\equiv$ { algebra }

$$\frac{1}{2^{30-k}} \leq \frac{1}{2^{16}}$$

Which holds for $k \leq 14$, thus also for $9 \leq k \leq 14$.

Case $15 \leq k$.
As a result, $\epsilon_{max} = 0$ and the proof continues as follows:

$$\begin{aligned} & \frac{2^k}{2^{16} + 2^9 * \beta} + \frac{1}{2^{30-k}} \leq \frac{2^k}{2^{16} + 2^9 * \beta + 0} + \frac{1}{2^{16}} \\ \equiv \quad & \{ \text{ definition } tab \text{ with } 1 \leq \beta \ \} \\ & \frac{2^{21} \ \underline{\text{div}} \ (2^7 + \beta) + 1}{2^{30-k}} \leq \frac{2^k}{2^{16} + 2^9 * \beta} + \frac{1}{2^{17}} \end{aligned}$$

According the definition of B, a combination of β and k should satisfy $2^k | 2^9 * \beta$. Because $1 \leq \beta$, there is only one possible combination of β and k left, i.e. $\beta = 64$ and $k = 15$. Verification shows that this inequality is true.

9.6 Appendix: NormalizeS

The module *NormalizeS* is developed by Hoogerwoord. The module normalizes a given input list; in other words, it shifts the given input list to the left until the most significant bit is a 1. Furthermore, a second input list will be shifted to the left by the same amount of bits.

The module creates a list xx with only zeroes and exactly one 1. This list is obtained by linearly traversing bs. The bit with value 1 is on the same position as the most significant bit of bs with value 1. We name the position of this bit l. If $l = 0$, then the least significant bit is the most significant bit of bs with value 1.

To obtain bk, we have to shift bs an amount of $16 - i$ bits to the left, where $xx_i = 1$. This is implemented as follows: For all i, the module copies the value xx_i 17 times and uses an *and*-operation on this bundle with the list bs shifted $16 - i$ places to the left. If $xx_i = 0$, then the list contains after the *and*-operation only zeroes. If $xx_i = 1$, then the list after the *and*-operation is equal to bs shifted $16 - i$ places to the left. Then the *or*-operation is used on all those outputs, and as result, it produces list bs shifted $16 - l$ positions to the left, which is exactly the list bk.

The list xx is used a second time, to shift a second input list the same amount of bits to the left as the list bs.

* * *

```
////////////////////////////////////////////////////////////////////////////////
//
// Engineer:       R.R. Hoogerwoord
//
// This Verilog program normalizes B;
//   NB    = B * 2^k
//   PostQ = PreQ * 2^k
//
////////////////////////////////////////////////////////////////////////////////

module NormalizeS(B, NB, PreQ, PostQ);
     input [16:0] B;
     input [17:0] PreQ;
    output [16:0] NB;
    output [33:0] PostQ;

      wire [16:0] xx, yy ;
      wire [33:0] PreQExt;

assign xx[16] =  B[16] ;
assign yy[16] = ~B[16] ;
```

```
genvar i;
generate
  for (i=0; i<16; i=i+1)
  begin: AndStep
         MUXCY muxcystep(yy[i],yy[i+1],1'b0, ~B[i]);
         assign xx[i] = yy[i+1] && B[i] ;
  end
endgenerate

assign NB =
       ({17{xx[16]}} & B)           | ({17{xx[15]}} & B[15:0]<<1)  |
       ({17{xx[14]}} & B[14:0]<<2)  | ({17{xx[13]}} & B[13:0]<<3)  |
       ({17{xx[12]}} & B[12:0]<<4)  | ({17{xx[11]}} & B[11:0]<<5)  |
       ({17{xx[10]}} & B[10:0]<<6)  | ({17{xx[ 9]}} & B[ 9:0]<<7)  |
       ({17{xx[ 8]}} & B[ 8:0]<<8)  | ({17{xx[ 7]}} & B[ 7:0]<<9)  |
       ({17{xx[ 6]}} & B[ 6:0]<<10) | ({17{xx[ 5]}} & B[ 5:0]<<11) |
       ({17{xx[ 4]}} & B[ 4:0]<<12) | ({17{xx[ 3]}} & B[ 3:0]<<13) |
       ({17{xx[ 2]}} & B[ 2:0]<<14) | ({17{xx[ 1]}} & B[ 1:0]<<15) |
       ({17{xx[ 0]}} & B[ 0:0]<<16) ;

assign PreQext = {{16{PreQ[17]}},{PreQ}};

assign PostQ =
       ({34{xx[16]}} & PreQext)     | ({34{xx[15]}} & PreQext<<1)  |
       ({34{xx[14]}} & PreQext<<2)  | ({34{xx[13]}} & PreQext<<3)  |
       ({34{xx[12]}} & PreQext<<4)  | ({34{xx[11]}} & PreQext<<5)  |
       ({34{xx[10]}} & PreQext<<6)  | ({34{xx[ 9]}} & PreQext<<7)  |
       ({34{xx[ 8]}} & PreQext<<8)  | ({34{xx[ 7]}} & PreQext<<9)  |
       ({34{xx[ 6]}} & PreQext<<10) | ({34{xx[ 5]}} & PreQext<<11) |
       ({34{xx[ 4]}} & PreQext<<12) | ({34{xx[ 3]}} & PreQext<<13) |
       ({34{xx[ 2]}} & PreQext<<14) | ({34{xx[ 1]}} & PreQext<<15) |
       ({34{xx[ 0]}} & PreQext<<16) ;

endmodule
```

9.7 Appendix: NormalizeX

This is an extension of the module *NormalizeS* as described in Appendix 9.6.

Now, the module also implements $8 - 2^{16-k}$, which is needed in the module *DivModMul128*.

```
//////////////////////////////////////////////////////////////////////////////////
//
// Engineer:       R.R. Hoogerwoord
// Modified by:    Wouter de Koning
//
// This Verilog program normalizes B;
//   NB    = B * 2^k
//   PostQ = PreQ * 2^k
//   con   = 8 - 2^(16-k)
//
//////////////////////////////////////////////////////////////////////////////////

module NormalizeX(B, NB, PreQ, PostQ, con);
     input [16:0] B;
     input [17:0] PreQ;
    output [16:0] NB;
    output [33:0] PostQ;
    output [17:0] con;

      wire [16:0] xx, yy ;
      wire [33:0] PreQExt;


assign xx[16] =  B[16] ;
assign yy[16] = ~B[16] ;

genvar i;
generate
  for (i=0; i<16; i=i+1)
  begin: AndStep
         MUXCY muxcystep(yy[i],yy[i+1],1'b0, ~B[i]);
         assign xx[i] = yy[i+1] && B[i] ;
  end
endgenerate

assign NB =
       ({17{xx[16]}} & B)           | ({17{xx[15]}} & B[15:0]<<1)  |
       ({17{xx[14]}} & B[14:0]<<2)  | ({17{xx[13]}} & B[13:0]<<3)  |
       ({17{xx[12]}} & B[12:0]<<4)  | ({17{xx[11]}} & B[11:0]<<5)  |
       ({17{xx[10]}} & B[10:0]<<6)  | ({17{xx[ 9]}} & B[ 9:0]<<7)  |
```

```
       ({17{xx[ 8]}} & B[ 8:0]<<8)  | ({17{xx[ 7]}} & B[ 7:0]<<9)  |
       ({17{xx[ 6]}} & B[ 6:0]<<10) | ({17{xx[ 5]}} & B[ 5:0]<<11) |
       ({17{xx[ 4]}} & B[ 4:0]<<12) | ({17{xx[ 3]}} & B[ 3:0]<<13) |
       ({17{xx[ 2]}} & B[ 2:0]<<14) | ({17{xx[ 1]}} & B[ 1:0]<<15) |
       ({17{xx[ 0]}} & B[ 0:0]<<16) ;

assign PreQext = {{16{PreQ[17]}},{PreQ}};

assign PostQ =
       ({34{xx[16]}} & PreQext)     | ({34{xx[15]}} & PreQext<<1)  |
       ({34{xx[14]}} & PreQext<<2)  | ({34{xx[13]}} & PreQext<<3)  |
       ({34{xx[12]}} & PreQext<<4)  | ({34{xx[11]}} & PreQext<<5)  |
       ({34{xx[10]}} & PreQext<<6)  | ({34{xx[ 9]}} & PreQext<<7)  |
       ({34{xx[ 8]}} & PreQext<<8)  | ({34{xx[ 7]}} & PreQext<<9)  |
       ({34{xx[ 6]}} & PreQext<<10) | ({34{xx[ 5]}} & PreQext<<11) |
       ({34{xx[ 4]}} & PreQext<<12) | ({34{xx[ 3]}} & PreQext<<13) |
       ({34{xx[ 2]}} & PreQext<<14) | ({34{xx[ 1]}} & PreQext<<15) |
       ({34{xx[ 0]}} & PreQext<<16) ;

// - con implements 8 - 2^16-k
assign con =
       ({18{xx[16]}} & {  {2{1'b1}} , {12{1'b0}} , {4'b1000} }) |
       ({18{xx[15]}} & {  {3{1'b1}} , {11{1'b0}} , {4'b1000} }) |
       ({18{xx[14]}} & {  {4{1'b1}} , {10{1'b0}} , {4'b1000} }) |
       ({18{xx[13]}} & {  {5{1'b1}} ,  {9{1'b0}} , {4'b1000} }) |
       ({18{xx[12]}} & {  {6{1'b1}} ,  {8{1'b0}} , {4'b1000} }) |
       ({18{xx[11]}} & {  {7{1'b1}} ,  {7{1'b0}} , {4'b1000} }) |
       ({18{xx[10]}} & {  {8{1'b1}} ,  {6{1'b0}} , {4'b1000} }) |
       ({18{xx[ 9]}} & {  {9{1'b1}} ,  {5{1'b0}} , {4'b1000} }) |
       ({18{xx[ 8]}} & { {10{1'b1}} ,  {4{1'b0}} , {4'b1000} }) |
       ({18{xx[ 7]}} & { {11{1'b1}} ,  {3{1'b0}} , {4'b1000} }) |
       ({18{xx[ 6]}} & { {12{1'b1}} ,  {2{1'b0}} , {4'b1000} }) |
       ({18{xx[ 5]}} & { {13{1'b1}} ,  {1{1'b0}} , {4'b1000} }) |
       ({18{xx[ 4]}} & { {14{1'b1}}              , {4'b1000} }) |
       ({18{xx[ 2]}} & {              {14{1'b0}} , {4'b0100} }) |
       ({18{xx[ 1]}} & {              {14{1'b0}} , {4'b0110} }) |
       ({18{xx[ 0]}} & {              {14{1'b0}} , {4'b0111} }) ;
endmodule
```

Bibliography

[1] Randal Bryant. Bit-Level Analysis of an SRT Divider Circuit. In *Proceedings of the 33rd annual Design Automation Conference*, pages 661–665. ACM, 1996.

[2] John Carpinelli. *Computer Systems Organization and Architecture.* Addison-Wesley Boston, San Francisco, New York, 2001.

[3] Vincent Heuring. *Computer Systems Design and Architecture.* Addison-Wesley Boston, San Francisco, New York, 2008.

[4] Rob Hoogerwoord. *Programming by Calculation: techniques and applications.* 2013.

[5] Cleve Moler. A Tale of Two Numbers. *SIAM News*, 28(1):16–16, 1995.

[6] Stuart Oberman and Michael Flynn. Design Issues in Division and Other Floating-Point Operations. *IEEE Transactions on Computers*, 46(2):154–161, 1997.

[7] Stuart Oberman and Michael Flynn. Division Algorithms and Implementations. *IEEE Transactions on Computers*, 46(8):833–854, 1997.

[8] Richard Richards. *Arithmetic Operations in Digital Computers.* Van Nostrand, 1955.

[9] Peter Soderquist and Miriam Leeser. Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations. *ACM Computing Surveys (CSUR)*, 28(3):518–564, 1996.

[10] Kaur Sukhmeet, Suman, Manna Manpreet, and Agarwal Rajeev. VHDL Implementation of Non Restoring Division Algorithm Using High Speed Adder/Subtractor. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2(7):3317–3324, 2013.

[11] Donald Thomas and Philip Moorby. *The Verilog Hardware Description Language*, volume 2. Springer, 2002.

[12] John von Neumann, Arthur Brucks, and Herman Goldstine. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. *The Institute for Advanced Study*, 1946.