LOW COMPLEXITY AND LOW POWER BIT-SERIAL MULTIPLIERS

Kenny Johansson

LiTH-ISY-EX-3391-2003
Linköping 2003
LOW COMPLEXITY AND LOW POWER
BIT-SERIAL MULTIPLIERS

Master thesis in Electronics Systems
at Department of Electrical Engineering,
Linköping University

by

Kenny Johansson

LiTH-ISY-EX-3391-2003

Supervisors: Oscar Gustafsson
Henrik Ohlsson

Examiner: Lars Wanhammar

Linköping, 27 May 2003.
Abstract
Bit-serial multiplication with a fixed coefficient is commonly used in integrated circuits, such as digital filters and FFTs. These multiplications can be implemented using basic components such as adders, subtractors and D flip-flops. Multiplication with the same coefficient can be implemented in many ways, using different structures. Other studies in this area have focused on how to minimize the number of adders/subtractors, and often assumed that the cost for D flip-flops is neglectable. That simplification has been proved to be far too great, and further not at all necessary. In digital devices low power consumption is always desirable. How to attain this in bit-serial multipliers is a complex problem.

The aim of this thesis was to find a strategy on how to implement bit-serial multipliers with as low cost as possible. An important step was achieved by deriving formulas that can be used to calculate the carry switch probability in the adders/subtractors. It has also been established that it is possible to design a power model that can be applied to all possible structures of bit-serial multipliers.
# TABLE OF CONTENTS

1 Introduction ................................................. 1  
   1.1 Background ............................................ 1  
   1.2 Restrictions ........................................... 1  
   1.3 Outline ............................................... 2  

2 Multiplier Principle ................................. 3  
   2.1 Shift Operation with D Flip-flop ................. 3  
   2.2 Addition and Subtraction ............................. 3  
   2.3 From Graph to Implementation ...................... 4  
   2.4 Implementation Costs ............................... 5  

3 Graph Theory .............................................. 7  
   3.1 Topologies ............................................ 7  
      3.1.1 Types ............................................ 7  
   3.2 Pipelining ........................................... 10  
      3.2.1 Pipeline from Outside or Internal .......... 10  
      3.2.2 Automatization .................................. 10  
   3.3 Complete Graph Search ............................... 10  
      3.3.1 Sign Combinations ....................... 11  
      3.3.2 Search Algorithm ..................... 11  
      3.3.3 Statistics ........................................ 12  

4 Components ............................................... 15  
   4.1 About the Components ............................... 15  
   4.2 Logic ................................................ 15  
   4.3 D Flip-flop ............................................ 17  
   4.4 Adder ............................................... 18  
   4.5 Subtractor ........................................... 19  
   4.6 Multiplier .......................................... 20
1

INTRODUCTION

1.1 BACKGROUND

Bit-serial multiplication with a fixed coefficient is commonly used in integrated circuits, such as digital filters and FFTs. These multiplications can be implemented using adders, subtractors and D flip-flops. The basic function is shown in Figure 1.1, where \(X\) is the input, \(\alpha\) is the coefficient and \(Y\) is the output. Multiplication with the same coefficient can be implemented in many ways, using different structures. The goal with this thesis is to develop a method that make it possible to always choose the best structure, with respect to area, latency or power consumption.

\[
X \rightarrow \alpha \rightarrow Y = \alpha \cdot X
\]

Figure 1.1: Basic function for multiplication with a fixed coefficient.

1.2 RESTRICTIONS

The thesis is restricted to cover integer coefficients in the interval \([-4096, 4096]\). The structures are pipelined and contain up to four adders/subtractors, see [3]. The data to be multiplied is assumed to be a random bit-serial sequence.
1.3 OUTLINE

The outline of this report is as follows.

Chapter 2: Multiplier Principle - How multipliers can be described using graphs and which implementation costs that will be studied.

Chapter 3: Graph Theory - Different graph types and how the complete set of graphs can be searched.

Chapter 4: Components - Describes the components that are used when implementing bit-serial multipliers, i.e. adders, subtractors and D flip-flops.

Chapter 5: Switch Probability - Formulas that describe the carry switch probability are derived and how these can be applied to bit-serial multipliers is shown.

Chapter 6: Power Model - Models that describes the power consumption of adders, subtractors and flip-flops are designed and then applied to bit-serial multipliers.

Chapter 7: Matlab Program - The derived results are used in a program from which it is easy to receive the best implementations.

Chapter 8: Summary - Conclusions and suggestions for future work.
2

MULTIPLIER PRINCIPLE

2.1 SHIFT OPERATION WITH D FLIP-FLOP

The most fundamental function in the multiplication procedure is to multiply with two. This is done with a D flip-flop, see Figure 2.1.

\[ x(n) \longrightarrow [D] \longrightarrow x(n - 1) \]

<table>
<thead>
<tr>
<th>n</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(n)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x(n-1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ 2 + 4 + 32 = 38 \]

\[ 4 + 8 + 64 = 76 \]

Figure 2.1: Multiplication with two.

Because of this easy way to double it is not necessary to study multiplication with even coefficients, since that always can be obtained from a multiplication with an odd coefficient followed by a number of shifts. Therefore the total number of positive coefficients to be studied in the restricted interval is 2047, where 3 is the smallest and 4095 is the largest.

2.2 ADDITION AND SUBTRACTION

The basic circuit to perform addition and subtraction is a full adder, see [2]. Bit-serial signals can be added/subtracted as described in Figure 2.2, see [1].
4

Low Complexity and Low Power Bit-Serial Multipliers

Figure 2.2: Adder and subtractor.

If we, for example, study an addition with $A = x(n)$ and $B = x(n-1)$, as in Figure 2.1, we will get a multiplication with the coefficient three, see Figure 2.3.

Figure 2.3: Multiplication with three.

2.3 FROM GRAPH TO IMPLEMENTATION

The implementation of a multiplier can be described with a directed acyclic graph, where the nodes (except for the initial one) represent adders/subtractors and the branches correspond to shift operations, see [3]. Multiplication
with fixed coefficients is described in [1]. An example of a graph and the equivalent implementation is shown in Figure 2.4.

![Figure 2.4](image)

Figure 2.4: Multiplication with the coefficient 45 using three adders/subtractors.

### 2.4 IMPLEMENTATION COSTS

The different implementation costs considered in this thesis are:

- Number of adders/subtractors.
- Number of D flip-flops.
- Latency.
- Power consumption.

The question is how to find a structure that has as low costs as possible. We can, for example, reduce the number of add/sub-and-shift operations for the realization in Figure 2.4 by choosing another structure. This is shown in Figure 2.5.

![Figure 2.5](image)

Figure 2.5: Multiplication with the coefficient 45 using two adders.
The latency cost is defined as the computation time it takes to generate an output value from the corresponding input value, see [1]. If we only study integer multiplication, the latency can be defined as the number of introduced pipeline stages multiplied with the clock period. We use pipelining to divide the critical path so that two adders/subtractors always have at least one intermediate D flip-flop. We also introduce pipelining at the output. The minimum latency is one and the maximum is the same as the number of adders/subtractors. The previously studied examples, after introducing pipelining, are shown in Figure 2.6. If the clock period is $T$, the structures have latency $3T$ and $2T$, respectively.

![Figure 2.6: Pipelined multipliers.](image)

The most complex cost to calculate is the power consumption, which of course is affected by the number of add/sub-and-shift operations, but also by other circumstances such as switching activity, see [2]. This cost can not be calculated exactly, but how it can be estimated will be discussed in Chapter 6.
3

GRAPH THEORY

3.1 TOPOLOGIES

All possible graph topologies with up to four nodes are shown in Figure 3.1. These, and the 32 different graph topologies with five nodes, are discussed in [3], while results for up to six nodes are discussed in [6].

3.1.1 TYPES

Most graphs can be used in different ways by altering which branches that corresponds to a multiplication larger than one. In Figure 3.2 the different types of a graph is shown. The different types of this graph can be used to implement multiplication with the same coefficients, but that is not the case for all graphs. We can see that the number of flip-flops needed for type 1 is \(x+y+z\) and for type 4 \(\max(x, y, z)\). Notice that the variables have different values to implement multiplication with the same coefficient. We can also see that type 1 is much easier to pipeline than type 4. There are 16 different graph types with four nodes and 127 with five nodes.
Figure 3.1: All graphs with up to four nodes.
Figure 3.2: Different types of the same graph.
3.2 PIPELINING

A very important graph property is how easy it is to perform pipelining, because this affects latency, number of flip-flops and power consumption.

3.2.1 PIPELINE FROM OUTSIDE OR INTERNAL

Pipelining can be performed in two different ways; from outside or internal. It is always possible to pipeline from outside, and for type 2 and 4 in Figure 3.2 that is the only choice. But if we have the situation in type 3 it is possible to use internal pipelining if \( z \) is bigger than one. This is done by moving one flip-flop from the \( z \)-branch to the left side of the adder/subtractor, as shown in Figure 3.3. When internal pipelining is used the latency is not increased. Further the number of flip-flops is not always increased either.

![Internal pipelining](image)

Figure 3.3: Internal pipelining.

3.2.2 AUTOMATIZATION

It would be very time consuming to describe how pipelining can be performed for all different graph types by hand. Therefore a code generator, that investigates all possible ways to pipeline the structures, was implemented. The generator first pipelines from the outside and then, if possible, performs different ways of internal pipelining. The results, with adequate conditions, are printed to a file that can be used as a look-up-table.

3.3 COMPLETE GRAPH SEARCH

To find the best way to implement multiplication with all different coefficients, it is necessary to search through all different graph types with all possible combinations of adders/subtractors and flip-flops.
3.3.1 SIGN COMBINATIONS

For each node that represents an adder/subtractor there are three different sign combinations; (+, +), (+, -), (-, +). This imply that the total number of sign combinations are $3^{\text{nodes} - 1}$.

3.3.2 SEARCH ALGORITHM

The algorithm shown below was used to search through all possible implementations. If the variable $\text{coeff\_max}$ is 4096 all variables will loop from 1 to 13. Since changing sign of the output is assumed to be free, the absolute value of the coefficient is used. For each coefficient four different (some of them may be the same) implementations are saved, one optimized for each cost. The costs are considered in the order; latency, power consumption, number of flip-flops, number of adders/subtractors. This imply that if we for example are looking for the implementation with lowest number of flip-flops and several different implementations have the same cost, we choose the one with the lowest latency, and if this also is the same we choose the one with the lowest power consumption.

Algorithm:

$W := \text{ceil(log}_2(\text{coeff\_max})) + 1$

loop over all graph types

loop all variables from 1 to $W$

loop over all sign combinations

$c := \text{abs(calculated coefficient value)}$

if $(c > 2)$ and $(c \leq \text{coeff\_max})$

calculate the costs:

* number of adders/subtractors
* number of flip-flops
* latency
* power consumption

if any cost is lower than earlier found for $c$

save structure, variables, sign combination and costs

end if

end if

end loop

end loop

end loop
3.3.3 STATISTICS

In Table 3.1 some statistics over the best implementations are given. Out of the total 148 tested graph types, it is only 30 that in some aspect is the best for at least one of the 2047 coefficients. All graphs with up to five nodes have been completely investigated, and in addition the standard case with six nodes, see Figure 3.4. The column \textit{Realizeable} gives how many of the 2047 coefficients that is possible to implement with that graph type. The number of coefficients that is best implemented with each graph type considering the four different costs, respectively, are given in the last four columns. One interesting thing that can be seen in the table is that a minimum number of adders/subtractors does not always give the lowest power consumption. Compare, for example, the structures \([\text{nodes } 4, \text{ graph } 2, \text{ type } 3]\) and \([\text{nodes } 5, \text{ graph } 1, \text{ type } 1]\). The first structure is best for many more coefficients when considering number of adders/subtractors than when considering power consumption, while the opposite situation applies to the second structure.

![Figure 3.4: The standard graph type with six nodes.](image-url)
<table>
<thead>
<tr>
<th>Nodes</th>
<th>Graph Type</th>
<th>Type</th>
<th>Realizable</th>
<th>Add/Sub</th>
<th>Flip-flops</th>
<th>Latency</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>21</td>
<td>21</td>
<td>17</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>183</td>
<td>162</td>
<td>63</td>
<td>162</td>
<td>161</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>62</td>
<td>0</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>743</td>
<td>487</td>
<td>271</td>
<td>548</td>
<td>473</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>696</td>
<td>212</td>
<td>0</td>
<td>0</td>
<td>133</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>535</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>895</td>
<td>123</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>683</td>
<td>263</td>
<td>0</td>
<td>0</td>
<td>238</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>535</td>
<td>171</td>
<td>130</td>
<td>172</td>
<td>169</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
<td>172</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1583</td>
<td>128</td>
<td>557</td>
<td>510</td>
<td>211</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1705</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1835</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1652</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>1645</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1368</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1699</td>
<td>99</td>
<td>207</td>
<td>175</td>
<td>121</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1404</td>
<td>56</td>
<td>181</td>
<td>101</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1862</td>
<td>32</td>
<td>58</td>
<td>36</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>1770</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>1955</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>3</td>
<td>1981</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>1</td>
<td>1645</td>
<td>158</td>
<td>369</td>
<td>250</td>
<td>226</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
<td>1755</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>1</td>
<td>948</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>1</td>
<td>1404</td>
<td>14</td>
<td>61</td>
<td>40</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>1</td>
<td>1784</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
<td>1264</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>1</td>
<td>697</td>
<td>2</td>
<td>46</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>2015</td>
<td>0</td>
<td>73</td>
<td>16</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.1. Statistics over best implementations.
Low Complexity and Low Power Bit-Serial Multipliers
4

COMPONENTS

4.1 ABOUT THE COMPONENTS

The main focus of this thesis was not to find the best individual components. For that reason basic components have been used, but they are however briefly presented in this chapter. The circuits are designed in Cadence using 0.35 μm technology. PMOS transistors are tripled in width size with respect to NMOS devices. When there are two transistors connected in serie the width is doubled, and in the same way the width is tripled for three transistors connected in serie. This can, for complementary CMOS, be explained as that the circuit output resistance is the same as that of an inverter, see [2].

4.2 LOGIC

Two basic complementary CMOS circuits are used as building blocks in the more complex circuits. The inverter, see Figure 4.1, and the three-input NAND gate, see Figure 4.2. To implement an SR flip-flop, two NAND gates are used. The special thing about this SR flip-flop is that it has two S and two R inputs, see Figure 4.3.
Figure 4.1: Complementary CMOS inverter.

Figure 4.2: Complementary CMOS three-input NAND gate.
4.3 D FLIP-FLOP

Since the D flip-flop is the most commonly used building block in a bit-serial multiplier, it is the most important circuit. It is used for three purposes:

- Multiplication with two, see Section 2.1.
- Save the carry in adders/subtractors, see Section 2.2.
- Pipelining, see Sections 2.4 and 3.2.

The D flip-flop that has been used is the StrongARM flip-flop, see [4]. It has been modified to include asynchronous set/clr inputs, see Figure 4.4. Before a multiplication is started the flip-flops used to save carries in subtractors are set and all other flip-flops are reset.
The basic function in the adder is performed by a full adder. There are a lot of different designs of full adders but a very fundamental design was selected, see Figure 4.5. Because of the design it is called mirror adder, see [2]. Originally this design only produced the inverted sum and carry, but since the StrongARM flip-flop need the standard outputs as well, the mirror adder is complemented with two inverters. The bit-serial carry-save adder is completed by combining a mirror adder and a StrongARM flip-flop, see Figure 4.6 (compare with Figure 2.2).
4.5 SUBTRACTOR

The subtractor implementation is very similar to the adder. The only differences are that one input is inverted and that the D flip-flop is set instead of cleared when the operation is started, see Figure 4.7 (compare with Figure 2.2).
4.6 MULTIPLIER

The presented components are enough to implement bit-serial multipliers. Figure 4.8 shows a pipelined implementation of a multiplier with the coefficient 45 (compare with Figure 2.6).
SWITCH PROBABILITY

5.1 FORMULA SEARCHING

A main factor for power consumption in CMOS circuits is the switching activity, see [2]. It is therefore of interest to calculate the probability for logical switching in the adders/subtractors in a bit-serial multiplier.

5.1.1 STATE DIAGRAM FOR A SPECIFIC ADDER CIRCUIT

To start with we consider the situation in Figure 5.1. As we can see the circuit contain three D flip-flops, which gives eight different states, see Table 5.1. It is now possible to draw a state diagram which shows how transitions between states depend on the input signals, see Figure 5.2.

Figure 5.1: An adder where one input is delayed two clock periods.
Table 5.1. State definitions.

<table>
<thead>
<tr>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 5.2: State diagram with input signals stated on the branches.
5.1.2 TRANSITION PROBABILITY

We define $P(x)$ as the probability that the signal $x$ is 1. Assume that $P(A) = P(B) = 1/2$. We also define the correlation between $A$ and $B$ as $\text{corr}(A, B) = P(A \land B)$. This imply that the maximum correlation is 1/2 and occur when $A$ and $B$ always have the same logic value. There are four possible input combinations, but most interesting is if the inputs are equal or not. The probability that they have the same logic value is $2 \cdot \text{corr}(A, B)$. This can be used to describe the probability for each state as stated in (5.1), where $\lambda$ is a function of the correlation as shown in (5.2) and $P$ is used to make the total probability equal to one as stated in (5.3).

\[
P(\text{state}_i) = 2(1 + \lambda) P_i, \quad 0 \leq i \leq 7 \quad \text{(5.1)}
\]

\[
\lambda = \frac{1}{2 \cdot \text{corr}(A, B)} - 1, \quad \text{corr}(A, B) \neq 0 \quad \text{(5.2)}
\]

\[
P = \frac{1}{2(1 + \lambda) \sum_{i=0}^{7} P_i} \quad \text{(5.3)}
\]

We can also describe the state diagram in Figure 5.2 with a table where we beside states and input signals also include the outputs, $S$ and $C$, and the probability for each transition, see Table 5.2.

<table>
<thead>
<tr>
<th>From</th>
<th>AB = 00</th>
<th>AB = 01</th>
<th>AB = 10</th>
<th>AB = 11</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>To S C</td>
<td>To S C</td>
<td>To S C</td>
<td>To S C</td>
<td>$A = B$</td>
</tr>
<tr>
<td>0</td>
<td>4 0 0</td>
<td>4 1 0</td>
<td>4 1 0</td>
<td>4 1 0</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0</td>
<td>4 1 0</td>
<td>1 0 1</td>
<td>5 0 1</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td>4 1 0</td>
<td>1 0 1</td>
<td>5 0 1</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>3</td>
<td>1 0 1</td>
<td>5 0 1</td>
<td>1 1 1</td>
<td>5 1 1</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>4</td>
<td>2 0 0</td>
<td>6 0 0</td>
<td>2 1 0</td>
<td>6 1 0</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>5</td>
<td>2 1 0</td>
<td>6 1 0</td>
<td>3 0 1</td>
<td>7 0 1</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>6</td>
<td>2 1 0</td>
<td>6 1 0</td>
<td>3 0 1</td>
<td>7 0 1</td>
<td>$P_iP$</td>
</tr>
<tr>
<td>7</td>
<td>3 0 1</td>
<td>7 0 1</td>
<td>3 1 1</td>
<td>7 1 1</td>
<td>$P_iP$</td>
</tr>
</tbody>
</table>

Table 5.2. State diagram in table format.
Table 5.2 is symmetric and it is obvious that \( P_0 = P_7 \), \( P_1 = P_6 \), \( P_2 = P_5 \) and \( P_3 = P_4 \). We can use this to set up a system of equations that is based on the fact that the probability to get to a specific state has to be the same as the probability to leave that state, see (5.4). If we, for example, assume that \( P_0 = 1 \), which is alright to assume because the transition probabilities will be regulated by \( P \), the system of equations can be solved, see (5.5). We can then calculate \( P \) as stated in (5.3), see (5.6).

\[
\begin{bmatrix}
-(2 + \lambda) & \lambda & 1 + \lambda \\
1 & -(1 + 2\lambda) & 1 + \lambda \\
\lambda & \lambda & -2(1 + \lambda)
\end{bmatrix}
\begin{bmatrix}
P_1 \\
P_2 \\
P_3
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
-(1 + \lambda)
\end{bmatrix}

\begin{bmatrix}
P_0 \\
P_0 \\
P_0
\end{bmatrix}
\cdot P
\quad (5.4)
\]

\[
\begin{bmatrix}
P_1 \\
P_2 \\
P_3
\end{bmatrix}
= \begin{bmatrix}
\frac{1}{4}(1 + 3\lambda) \\
\frac{1}{4}(3 + \lambda) \\
\frac{1}{2}(1 + \lambda)
\end{bmatrix}
\quad (5.5)
\]

\[
P = \frac{1}{2(1 + \lambda)2(1 + P_1 + P_2 + P_3)} = \frac{1}{2(1 + \lambda)(5 + 3\lambda)}
\quad (5.6)
\]

5.1.3 CORRELATIONS AND SWITCH PROBABILITIES

It is now possible to express the correlations between the inputs and the output \( S \) as a function of \( \lambda \), see (5.7) and (5.8). We can also calculate \( P(\text{switch } C) \), which is the probability that the output \( C \) goes from logic 1 to logic 0 or the other way around, see (5.9). In Table 5.2 we can see that from each state, \( i \) where \( 0 \leq i \leq 7 \), there are two branches where \( S = 1 \) with the probabilities \( P_i P \) and \( \lambda P_i P \), respectively. This imply that (5.10) applies. Another interesting result that will be used later is shown in (5.11).

\[
\text{corr}(A,S) = (1 + \lambda)(P_0 + P_3 + P_4 + P_7)P = \frac{3 + \lambda}{2(5 + 3\lambda)}
\quad (5.7)
\]

\[
\text{corr}(B,S) = (P_0 + P_3 + P_4 + P_7)P + \lambda(P_1 + P_2 + P_5 + P_6)P = \frac{2\lambda^2 + 3\lambda + 3}{6\lambda^2 + 16\lambda + 10}
\quad (5.8)
\]
Chapter 5 – Switch Probability

\[
P(\text{switch C}) = (1 + \lambda)(P_1 + P_2 + P_5 + P_6)P = \frac{1 + \lambda}{5 + 3\lambda} 
\] 

\[
P(\text{switch S}) = \frac{1}{2} 
\] 

\[
corr(A,S) + P(\text{switch C}) = \frac{1}{2} 
\]

We can now calculate \(corr(A,S), corr(B,S)\) and \(P(\text{switch C})\) for different values of \(corr(A,B)\), some results are shown in Table 5.3. In the table a new variable, \(\delta\), is included. \(\delta\) is, in the same way as \(\lambda\), a function of the inputs correlation, see (5.12). The reason to introduce \(\delta\) is that it gives easier formulas than \(\lambda\), especially in the coming where we are going to make the formulas more general. The new expressions for the correlations between the inputs and S are shown in (5.13) and (5.14), respectively.

<table>
<thead>
<tr>
<th>(corr(A,B))</th>
<th>(\lambda)</th>
<th>(corr(A,S))</th>
<th>(corr(B,S))</th>
<th>(P(\text{switch C}))</th>
<th>(\delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>0</td>
<td>3/10</td>
<td>3/10</td>
<td>1/5</td>
<td>1</td>
</tr>
<tr>
<td>1/3</td>
<td>1/2</td>
<td>7/26</td>
<td>10/39</td>
<td>3/13</td>
<td>3</td>
</tr>
<tr>
<td>3/10</td>
<td>2/3</td>
<td>11/42</td>
<td>53/210</td>
<td>5/21</td>
<td>5</td>
</tr>
<tr>
<td>1/4</td>
<td>1</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td>---</td>
</tr>
<tr>
<td>1/5</td>
<td>3/2</td>
<td>9/38</td>
<td>24/95</td>
<td>5/19</td>
<td>-5</td>
</tr>
<tr>
<td>1/20</td>
<td>9</td>
<td>3/16</td>
<td>3/10</td>
<td>5/16</td>
<td>-5/4</td>
</tr>
</tbody>
</table>

Table 5.3. Results for some input correlations.

\[
\delta = \frac{1}{4 \cdot corr(A,B) - 1}, \quad corr(A,B) \neq \frac{1}{4} 
\] 

\[
corr(A,S) = \frac{2\delta + 1}{\delta^2 + 2} 
\] 

\[
corr(B,S) = \frac{4\delta^2 + \delta + 1}{16\delta^2 + 4\delta} 
\]
5.1.4 GENERALIZED ADDER CIRCUIT

So far we have only studied the case in Figure 5.1, but it is now time to explore other cases. The circuit shown in Figure 5.3 is generalized so that the B input is delayed an arbitrary number of clock periods.

![Figure 5.3: An adder where one input is delayed d clock periods.](image)

We can now repeat all the work we did for \( d = 2 \), for other values of \( d \). The results from this, with \( \text{corr}(A,B) = 3/10 \), is presented in Table 5.4. To start with we establish that (5.11) still applies to calculate \( P(\text{switch C}) \). It is quite simple to see the strong connection between values in the table, and we summarize this with the formulas in (5.15) and (5.16). The most interesting relation is the one between \( \text{corr}(A,B) \) and \( P(\text{switch C}) \), see Figure 5.4.

<table>
<thead>
<tr>
<th>( d )</th>
<th>( \text{corr}(A,S) )</th>
<th>( \text{corr}(B,S) )</th>
<th>( P(\text{switch C}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/11</td>
<td>14/55</td>
<td>5/22</td>
</tr>
<tr>
<td>2</td>
<td>11/42</td>
<td>53/210</td>
<td>5/21</td>
</tr>
<tr>
<td>3</td>
<td>21/82</td>
<td>103/410</td>
<td>10/41</td>
</tr>
<tr>
<td>4</td>
<td>41/162</td>
<td>203/810</td>
<td>20/81</td>
</tr>
<tr>
<td>5</td>
<td>81/322</td>
<td>403/1610</td>
<td>40/161</td>
</tr>
</tbody>
</table>

Table 5.4. Results for \( 1 \leq d \leq 5 \) with \( \delta = 5 \).

\[
\text{corr}(A,S) = \frac{\delta \cdot 2^d - 1 + 1}{\delta \cdot 2^d + 1 + 2}
\]  

(5.15)

\[
\text{corr}(B,S) = \frac{\delta^2 \cdot 2^d + \delta + 1}{\delta^2 \cdot 2^d + 2 + 4\delta}
\]  

(5.16)
5.1.5 CORRELATIONS FOR A SUBTRACTOR CIRCUIT

With the equations in (5.11), (5.12), (5.15), and (5.16) the adder is sufficiently described. Next thing to consider is to find similar expressions for the subtractor, where two different cases exists as shown in Figure 5.5.

The expressions can of course be found in the same way as for the adder, i.e. with a state diagram and so on, but here are only the results presented. For the first subtractor situation, that is when the delayed input is inverted, we get the formulas in (5.17), (5.18) and (5.19) ($d$ is derived in the same way as stated in (5.12)). Corresponding equations for the other subtractor situation are shown in (5.20), (5.21) and (5.22). The relation between corr(A,B) and P(switch C) is shown in Figure 5.6. Notice that the carry switch probability is the same independent of which of the input signals that is inverted. Further this diagram is reversed compared to the one for the adder.
Figure 5.5: Subtractors where one input is delayed $d$ clock periods.

\[
corr(A, S) = \frac{\delta \cdot 2^{d-1} - 1}{\delta \cdot 2^{d+1} - 2} \quad (5.17)
\]

\[
corr(B, S) = \frac{\delta^2 \cdot 2^d - \delta - 1}{\delta^2 \cdot 2^{d+2} - 4\delta} \quad (5.18)
\]

\[
P(\text{switch C}) = \frac{1}{2} - corr(A, S) \quad (5.19)
\]

\[
corr(A, S) = \frac{1}{2} - \frac{\delta \cdot 2^{d-1} - 1}{\delta \cdot 2^{d+1} - 2} \quad (5.20)
\]

\[
corr(B, S) = \frac{1}{2} - \frac{\delta^2 \cdot 2^d - \delta - 1}{\delta^2 \cdot 2^{d+2} - 4\delta} \quad (5.21)
\]

\[
P(\text{switch C}) = corr(A, S) \quad (5.22)
\]
5.1.6 THE PROBLEM WITH UNCORRELATED SIGNALS

As stated in (5.12) we cannot derive $\delta$ if $\text{corr}(A,B) = 1/4$ because $\delta$ will then be infinite. But if we let $\delta$ go towards infinity in (5.15) and (5.16) we get the correct results as shown in (5.23). This also applies to corresponding expressions for the subtractor.

\[
\lim_{\delta \to \infty} \frac{\delta \cdot 2^{d-1} + 1}{\delta \cdot 2^{d+1} + 2} = \frac{1}{4} \quad \lim_{\delta \to \infty} \frac{\delta^2 \cdot 2^d + \delta + 1}{\delta^2 \cdot 2^{d+2} + 4\delta} = \frac{1}{4}
\]  

(5.23)
5.2 HOW TO USE THE FORMULAS

The derived formulas in Section 5.1 can be used to calculate the switch probabilities in bit-serial multipliers. This will now be shown with an example.

EXAMPLE 5.1

Calculate the carry switch probabilities for the structure in Figure 5.7.

![Figure 5.7: Structure for multiplication with the coefficient 347.](image)

The switch probabilities in the two adders and the subtractor are calculated from left to right. The subscripts indicate which adder/subtractor the variable \( \delta \) and the signals \( A, B \) and \( S \) are referring to.

1) \( \text{corr}(A_1, B_1) = \frac{1}{2} \)

\[
(5.12) \quad \Rightarrow \quad \delta_1 = 1
\]

\[
(5.15) \quad \Rightarrow \quad \text{corr}(A_1, S_1) = \frac{1 \cdot 2^{2-1} + 1}{1 \cdot 2^{2+1} + 2} = \frac{3}{10}
\]

\[
(5.11) \quad \Rightarrow \quad P(\text{switch } C_1) = \frac{1}{2} - \text{corr}(A_1, S_1) = \frac{1}{5}
\]

2) \( \text{corr}(A_2, B_2) = \text{corr}(A_1, S_1) = \frac{3}{10} \)

\[
(5.12) \quad \Rightarrow \quad \delta_2 = 5
\]

\[
(5.15) \quad \Rightarrow \quad \text{corr}(A_2, S_2) = \frac{5 \cdot 2^{1-1} + 1}{5 \cdot 2^{1+1} + 2} = \frac{3}{11}
\]

\[
(5.11) \quad \Rightarrow \quad P(\text{switch } C_2) = \frac{1}{2} - \text{corr}(A_2, S_2) = \frac{5}{22}
\]
Chapter 5 – Switch Probability

3) \[(5.16) \Rightarrow \text{corr}(A_3, B_3) = \text{corr}(B_2, S_2) = \frac{5^2 \cdot 2^1 + 5 + 1}{5^2 \cdot 2^{1+2} + 4 \cdot 5} = \frac{14}{55}\]

\[(5.12) \Rightarrow \delta_3 = 55\]

\[(5.20) \Rightarrow \text{corr}(A_3, S_3) = \frac{1}{2} \cdot \frac{55 \cdot 2^{5-1} - 1}{55 \cdot 2^{5+1} - 2} = \frac{440}{1759}\]

\[(5.22) \Rightarrow P(\text{switch } C_3) = \text{corr}(A_3, S_3) = \frac{440}{1759}\]

Notice that the switch probabilities in the two adders differ significantly from 1/4 while the switch probability for the subtractor is very close to 1/4. The reason for this is that the inputs to the subtractor are almost uncorrelated and in addition one input is delayed as much as five clock periods.

---

**EXAMPLE 5.2**

Simulate the carry switch probabilities for the structure in Figure 5.7 and compare with the results calculated in Example 5.1.

Matlab is used to run a simulation with 1 000 000 random input values. The S output switch probability, which according to (5.10) is supposed to be 1/2, is also simulated. The results is presented in Table 5.5, and corresponds well with the calculated values. The reason is, of course, the large number of input values.

<table>
<thead>
<tr>
<th>Adder/ Subtractor</th>
<th>Simulated</th>
<th>Calculated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P(switch C)</td>
<td>P(switch S)</td>
</tr>
<tr>
<td>1</td>
<td>20.03%</td>
<td>50.08%</td>
</tr>
<tr>
<td>2</td>
<td>22.72%</td>
<td>49.96%</td>
</tr>
<tr>
<td>3</td>
<td>25.01%</td>
<td>49.91%</td>
</tr>
</tbody>
</table>

Table 5.5. Simulated and calculated switch probabilities.
5.3 LIMITATIONS

For most graphs with up to four nodes the switch probabilities can be calculated with the derived formulas. But there are two graphs with four nodes and 19 graphs with five nodes where this is not possible. One example of such a graph is shown in Figure 5.8.

![Figure 5.8: Graph where the switch probability can not be calculated.](image)

In this graph there are no problem to calculate the switch probabilities for the first two adders/subtractors, but it can not be done for the last one since we do not know the correlation between the inputs to this node. It is, however, possible to calculate this correlation for a specific case, that is when all branches are known including sign and number of D flip-flops. This can be done in a similar way as was shown in Section 5.1.2, i.e. by setting up and solving a system of equations. The solution is to loop through all unsolveable graphs, as done by the algorithm in Section 3.3.2, and for each specific case calculate the correlation and save the value in a look-up-table. An example of this for the graph in Figure 5.8 is shown in Table 5.6, which contain the correlation between the input signals to the last adder/subtractor when the two previous nodes correspond to adders. This imply that we for this graph get nine different tables, one for each possible sign combination, see Section 3.3.1, of the first two adders/subtractors.
Table 5.6. Correlation look-up-table for the graph in Figure 5.8.

The maximum number of D flip-flops for which the correlation was calculated was in this case nine, but there are also one flip-flop in each adder. This gives $2^9 = 2^{11} = 2048$ number of states, which correspond to a system of equations where the main matrix is $1023 \times 1023$. The systems can obviously be very complicated and it is not recommended to use this method for a large number of D flip-flops. When there are many flip-flops in a circuit, and the correlation between the input signals to an adder/subtractor cannot be calculated with the derived formulas, it is a good approximation to assume that the signals are uncorrelated, i.e. that the correlation is $1/4$. This can also be seen in Table 5.6 where all values in the diagonal $x + y = 9$ are very close to $1/4$.

Except the big dimension of these systems, there are another unpleasant problem; singular matrixes. When the main matrix is singular it is impossible to solve the system of equations, see [5]. It is however possible to rewrite the system, trying to avoid the singularity. In this thesis, the systems were calculated in three ways, as described in (5.24), (5.25) and (5.26). The number of states is $2n$. $A$ is a matrix with the dimension $n-1 \times n-1$ and $B$ is a matrix with the dimension $n-1 \times 1$.

$$A \cdot \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{n-1} \end{bmatrix} = B \cdot P_0 \quad \text{assumed } P_0 = 1$$  \hspace{1cm} (5.24)
Another way to find missing values is to make use of symmetry. It is obvious that Table 5.6 is symmetric and if for example the value corresponding to \((x = 3, y = 5)\) could not have been calculated due to singularity it would have been possible to assume that it should be the same as the value corresponding to \((x = 5, y = 3)\).

Some statistics on how many searched correlation values it was possible to find with these methods is presented in Table 5.7. From this it is clear that most values could be found with the first method, and the reason that the other methods did not result in that many more new values is that if a system of equations is singular a rewriting of it is much more likely to be singular. In total almost 97% of the searched correlation values could be found. The remaining correlation values are estimated to be 1/4.

<table>
<thead>
<tr>
<th>Total number of searched correlation values</th>
<th>77 211</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation values found with the method in (5.24)</td>
<td>72 085</td>
</tr>
<tr>
<td>New correlation values found with the method in (5.25)</td>
<td>1 946</td>
</tr>
<tr>
<td>New correlation values found with the method in (5.26)</td>
<td>229</td>
</tr>
<tr>
<td>New correlation values found using symmetry</td>
<td>437</td>
</tr>
</tbody>
</table>

**Table 5.7. Statistics on searched correlation values.**
6

POWER MODEL

6.1 POWER MODEL FOR THE ADDER

To study the power consumption of the adder, a circuit based on the schematic in Figure 5.3 with \( \text{corr}(A,B) = 1/2 \) can be used. From simulations in NanoSim, with 1000 random input values and a clock frequency of 4 MHz, the results in Table 6.1 were generated. Notice that when power is mentioned in this chapter it refers to the total average power and the unit is always \( \mu W \).

| \( d \) | \( \text{Switches} \) | \( \text{Power} \) (\( \mu W \)) |
|---|---|---|---|---|---|
|   | S | C | StrongARM | MirrorAdder | Total |
| 1 | 505 | 162 | 25.33 | 15.42 | 40.75 |
| 2 | 606 | 213 | 26.75 | 16.89 | 43.64 |
| 4 | 605 | 252 | 27.89 | 17.55 | 45.44 |
| 8 | 612 | 251 | 27.88 | 17.61 | 45.49 |

Table 6.1. Simulation result for the adder.

6.1.1 GLITCH PROBABILITY

An unexpected result in Table 6.1 is that \( P(\text{switch } S) \) seems to be bigger than \( 1/2 \) when \( d \neq 1 \). However this is not so surprising after a closer look in Figure 4.5, where it is possible to establish that a glitch will occur if \( C \) switches but not \( S \). This can be expressed as in (6.1). An example of how
these glitches may occur is shown in Figure 6.1, where \(A(n), B(n)\) and \(C(n-1)\) are the input signals to the mirror adder and \(S(n)\) and \(C(n)\) are the outputs. In this figure two glitches, one positive and one negative, occur on the \(S(n)\) signal. To make the glitches clear the used clock frequency in this figure is as high as 500 MHz.

Figure 6.1: Glitches in a mirror adder.

The glitch probability can be calculated in a manner similar to how the switch probability was calculated in Chapter 5. A comparison between Table 6.2 and Table 5.4 make statement (6.2) trustworthy, and this formula can also be applied to the subtractor.

<table>
<thead>
<tr>
<th>(d)</th>
<th>(P(\text{glitch } S))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/55</td>
</tr>
<tr>
<td>2</td>
<td>5/84</td>
</tr>
<tr>
<td>3</td>
<td>5/82</td>
</tr>
<tr>
<td>4</td>
<td>5/81</td>
</tr>
<tr>
<td>5</td>
<td>10/161</td>
</tr>
</tbody>
</table>

Table 6.2. Results for \(1 \leq d \leq 5\) with \(\delta = 5\).
Chapter 6 – Power Model

\[ P(\text{glitch S}) = P((\text{switch C}) \land \neg (\text{switch S})) \]  
(6.1)

\[ P(\text{glitch S}) = \frac{P(\text{switch C})}{4}, \quad d \neq 1 \]  
(6.2)

So the problem is when \( d = 1 \), which needs to be calculated separate, see Table 6.3. From these values it is possible to derive the expression in (6.3). Corresponding expression for the subtractor is shown in (6.4). If \( \delta \) goes towards infinity the correct result for \( \text{corr}(A,B) = 1/4 \) is received, see (6.5).

<table>
<thead>
<tr>
<th>\text{corr}(A,B)</th>
<th>\delta</th>
<th>\text{P(\text{glitch S})}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1/3</td>
<td>3</td>
<td>1/21</td>
</tr>
<tr>
<td>3/10</td>
<td>5</td>
<td>3/55</td>
</tr>
<tr>
<td>1/4</td>
<td>---</td>
<td>1/16</td>
</tr>
<tr>
<td>1/5</td>
<td>-5</td>
<td>1/15</td>
</tr>
<tr>
<td>1/20</td>
<td>-5/4</td>
<td>3/80</td>
</tr>
</tbody>
</table>

Table 6.3. Results for some input correlations, with \( d = 1 \).

\[ P(\text{glitch S}) = \frac{\delta^2 - 1}{16\delta^2 + 8\delta}, \quad d = 1 \]  
(6.3)

\[ P(\text{glitch S}) = \frac{\delta^2 - 1}{16\delta^2 - 8\delta}, \quad d = 1 \]  
(6.4)

\[ \lim_{\delta \to \infty} \frac{\delta^2 - 1}{16\delta^2 + 8\delta} = \frac{1}{16} \]  
(6.5)

6.1.2 MIRROR ADDER

The power consumption for the mirror adder can be expressed as in (6.6). In Table 6.1 the difference for \( P(\text{glitch S}) \) corresponding to \( d = 2 \) and \( d = 4 \) is neglectable. From this \( P_{\text{switch}} \) can be derived, see (6.7). Further the first and last row in Table 6.1 can be used to derive \( P_{\text{base cost}} \) and \( P_{\text{glitch}} \), see (6.8). Notice that all switches for the S signal over 500 are assumed to arise from glitches, and that each glitch is composed of two switches.
Low Complexity and Low Power Bit-Serial Multipliers

\begin{equation}
\text{Power} = P_{\text{base cost}} + P(\text{switch C}) \cdot P_{\text{switch}} + P(\text{glitch S}) \cdot P_{\text{glitch}}
\end{equation}

\begin{equation}
17.55 - 16.89 = (0.252 - 0.213) \cdot P_{\text{switch}} \Rightarrow P_{\text{switch}} = 16.92
\end{equation}

\begin{align*}
15.42 - 0.162 \cdot P_{\text{switch}} &= P_{\text{base cost}} + 0.0025 \cdot P_{\text{glitch}} \\
17.61 - 0.251 \cdot P_{\text{switch}} &= P_{\text{base cost}} + 0.056 \cdot P_{\text{glitch}}
\end{align*}

\begin{align*}
P_{\text{base cost}} &= 12.65 \\
P_{\text{glitch}} &= 12.78
\end{align*}

6.1.3 CARRY SAVING D FLIP-FLOP

The power consumption for the carry saving flip-flop is dependent on the switch probability. In Figure 6.2 the results from Table 6.1 and Table 6.4 are marked, and to this is a straight line adjusted. The line has the equation shown in (6.9).

\begin{equation}
\text{Power} = 20.72 + 28.42 \cdot P(\text{switch C})
\end{equation}

Figure 6.2: Relation between P(switch C) and power consumption.
6.2 POWER MODEL FOR THE SUBTRACTOR

Simulation results for the subtractor was generated in the same way as for the adder, see Table 6.4.

<table>
<thead>
<tr>
<th>d</th>
<th>Switches</th>
<th>Power (μW)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>524</td>
<td>522</td>
</tr>
<tr>
<td>2</td>
<td>666</td>
<td>342</td>
</tr>
<tr>
<td>4</td>
<td>645</td>
<td>276</td>
</tr>
<tr>
<td>8</td>
<td>660</td>
<td>266</td>
</tr>
</tbody>
</table>

Table 6.4. Simulation result for the subtractor.

The glitch probability and the power consumption for the carry saving D flip-flop was investigated in Section 6.1. The power consumption for the mirror adder in a subtractor can be expressed in the same way as for the adder, see (6.6). We let the base cost be the same as for the adder to make the comparison easier. Further we choose to use the rows corresponding to d = 1 and d = 4 to calculate the other values, see (6.10).

\[
\begin{align*}
22.44 - P_{\text{base cost}} &= 0.522 \cdot P_{\text{switch}} + 0.012 \cdot P_{\text{glitch}} \\
18.80 - P_{\text{base cost}} &= 0.276 \cdot P_{\text{switch}} + 0.0725 \cdot P_{\text{glitch}} \\
\begin{cases}
P_{\text{switch}} = 18.45 \\
P_{\text{glitch}} = 14.60
\end{cases}
\end{align*}
\]

(6.10)

If we as a test calculate the power consumption of the mirror adder on the rows corresponding to d = 2 and d = 8, we get 20.17 μW and 18.73 μW, respectively. These results are sufficiently close to the simulated values. Notice that the switches and glitches give rise to higher power consumption in the subtractor than in the adder.

6.2.1 INVERTER

The power consumption for the inverter is not dependent on switch or glitch probabilities, and is therefore constant which is clearly seen in Table 6.4. However an average value based on this table was shown to be a bit to large when performing more realistic simulations and a slightly lower power consumption of 8.60 μW was used.
6.3 POWER MODEL FOR THE D FLIP-FLOP

All D flip-flops, except the ones in adders/subtractors, have switch probability 1/2. Therefore the switch probability is not of interest when the power model should be designed. Instead it is the load on the output that is the important factor. The consumed power for different loads is shown in Table 6.5 (sub+ and sub- refer to the input without and with inverter, respectively). In this table we can see that the load cost is linear and that the cost for add and sub+ is the same, which is not surprising since they are identical. The load cost for add can be calculated as in (6.11). The other costs can be calculated in a similar way, and the results are summarized in Table 6.6.

\[
\frac{(46.33 - 37.30) + (55.41 - 46.33)}{2} = 9.055
\]

(6.11)

<table>
<thead>
<tr>
<th>Load</th>
<th>Power ((\mu)W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>37.30</td>
</tr>
<tr>
<td>2 add</td>
<td>46.33</td>
</tr>
<tr>
<td>3 add</td>
<td>55.41</td>
</tr>
<tr>
<td>sub+</td>
<td>37.30</td>
</tr>
<tr>
<td>sub-</td>
<td>29.42</td>
</tr>
<tr>
<td>2 sub-</td>
<td>30.39</td>
</tr>
<tr>
<td>3 sub-</td>
<td>31.36</td>
</tr>
<tr>
<td>D + add</td>
<td>38.88</td>
</tr>
<tr>
<td>D + 2 add</td>
<td>47.92</td>
</tr>
</tbody>
</table>

Table 6.5. Simulation results with different load.

<table>
<thead>
<tr>
<th>Load</th>
<th>Cost ((\mu)W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, sub+</td>
<td>9.05</td>
</tr>
<tr>
<td>sub-</td>
<td>0.97</td>
</tr>
<tr>
<td>D</td>
<td>1.57</td>
</tr>
</tbody>
</table>

Table 6.6. Load costs for the D flip-flop.
A difference between flip-flops in a bit-serial multiplier is what kind of gate that drives the flip-flop. By simulations this was however shown not to affect the power consumption to a great extent, and is therefore just summarized in Table 6.7.

<table>
<thead>
<tr>
<th>Driving gate</th>
<th>Cost (µW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>0.00</td>
</tr>
<tr>
<td>add, sub</td>
<td>0.50</td>
</tr>
<tr>
<td>Multiplier input</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 6.7. Driving gate costs for the D flip-flop.

The most important value in the power model for the D flip-flop is of course the base cost. This can be estimated from Table 6.5 and Table 6.6, by subtracting the load costs, see Table 6.8. The base cost is set to 28.25 µW.

<table>
<thead>
<tr>
<th>Load</th>
<th>Base cost (µW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>28.25</td>
</tr>
<tr>
<td>2 add</td>
<td>28.23</td>
</tr>
<tr>
<td>3 add</td>
<td>28.26</td>
</tr>
<tr>
<td>sub+</td>
<td>28.25</td>
</tr>
<tr>
<td>sub-</td>
<td>28.45</td>
</tr>
<tr>
<td>2 sub-</td>
<td>28.45</td>
</tr>
<tr>
<td>3 sub-</td>
<td>28.45</td>
</tr>
<tr>
<td>D + add</td>
<td>28.26</td>
</tr>
<tr>
<td>D + 2 add</td>
<td>28.25</td>
</tr>
</tbody>
</table>

Table 6.8. Base cost with different load.

### 6.4 POWER MODEL FOR THE MULTIPLIER

Since adders, subtractors and D flip-flops are the only components in a bit-serial multiplier it is now possible to calculate the total power consumption for all multiplier structures. Additionally the power for the last pipelining stage and the input inverter are estimated by simulations to 29.05 µW and 2.85 µW, respectively. Since the values are included in all multipliers they do
not affect the comparison between different structures. To show how the total power can be calculated we continue Example 5.1.

---

**EXAMPLE 6.1**

Calculate the total average power for the structure in Figure 5.7.

- **Power for the first adder**

  \[ P_{\text{glitch } S_1} = \frac{P_{\text{switch } C_1}}{4} = \frac{1}{20} \]

  \[ P_{\text{total,add1}} = P_{\text{D}} + P_{\text{add}} = 43.08 \]

- **Power for the second adder**

  \[ P_{\text{glitch } S_2} = \frac{5^2 - 1}{16 \cdot 5^2 + 8 \cdot 5} = \frac{3}{55} \]

  \[ P_{\text{D}} = 20.72 + 28.42 \cdot \frac{5}{22} \approx 27.18 \]

  \[ P_{\text{total,add2}} = P_{\text{D}} + P_{\text{add}} = 44.37 \]

- **Power for the subtractor**

  \[ P_{\text{glitch } S_3} = \frac{P_{\text{switch } C_3}}{4} = \frac{110}{1759} \]

  \[ P_{\text{D}} = 20.72 + 28.42 \cdot \frac{440}{1759} \approx 27.83 \]

  \[ P_{\text{INV}} = 8.60 \]

  \[ P_{\text{total,sub}} = P_{\text{D}} + P_{\text{add}} + P_{\text{INV}} = 54.61 \]
• Power for the D flip-flops

Notice that we need to pipeline to avoid a direct connection from the first adder to the subtractor. The different costs can be summarized as shown in Table 6.9.

<table>
<thead>
<tr>
<th>Description</th>
<th>Cost (μW)</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base cost</td>
<td>28.25</td>
<td>9</td>
</tr>
<tr>
<td>add load</td>
<td>9.05</td>
<td>3</td>
</tr>
<tr>
<td>sub-load</td>
<td>0.97</td>
<td>1</td>
</tr>
<tr>
<td>D load</td>
<td>1.57</td>
<td>6</td>
</tr>
<tr>
<td>D driving</td>
<td>0.00</td>
<td>6</td>
</tr>
<tr>
<td>add driving</td>
<td>0.50</td>
<td>2</td>
</tr>
<tr>
<td>input driving</td>
<td>0.20</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.9. Costs for all design specific flip-flops.

\[
\text{Power}_{\text{flip-flops}} = 292.99
\]

• Power for the input inverter and the last pipelining stage

\[
\text{Power}_{\text{input INV}} = 2.85 \\
\text{Power}_{\text{last pl}} = 29.05
\]

• Total power for the bit-serial multiplier

The power consumption for the different components are summarized in Table 6.10. The total power consumption for the multiplier is:

\[
\text{Power}_{\text{tot}} = \\
= \text{Power}_{\text{tot,add1}} + \text{Power}_{\text{tot,add2}} + \text{Power}_{\text{tot,sub}} + \\
+ \text{Power}_{\text{flip-flops}} + \text{Power}_{\text{input INV}} + \text{Power}_{\text{last pl}} = 466.95
\]
EXAMPLE 6.2

Simulate the glitch probabilities for the structure in Figure 5.7 and compare with the results calculated in Example 6.1.

As in Example 5.2 Matlab is used to run a simulation with 1 000 000 random input values. The results is presented in Table 6.11. The simulated results agree very well with the calculated values due to the large number of input values.

<table>
<thead>
<tr>
<th>Adder/</th>
<th>Simulated P(glitch S)</th>
<th>Calculated P(glitch S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtract</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5.01%</td>
<td>5.00%</td>
</tr>
<tr>
<td>2</td>
<td>5.46%</td>
<td>5.45%</td>
</tr>
<tr>
<td>3</td>
<td>6.25%</td>
<td>6.25%</td>
</tr>
</tbody>
</table>

Table 6.11. Simulated and calculated glitch probabilities.
EXAMPLE 6.3

Simulate the total average power for the structure in Figure 5.7.

To start with we validate the function of the designed circuit. The easiest way to do this is to set the input signal to 1, and check that the output signal is the coefficient, see Figure 6.3. Notice that $347_{10} = 101011011_2$ and that the latency is two clock periods.

![Validation of multiplier with the coefficient 347.](image)

From a simulation with 10 000 random input values, the result in Table 6.12 was generated. These results do not differ much from the calculation in Example 6.1, see Table 6.10. The exception is the mirror adders, which consumes more power than calculated. The main reason for this is that there are other kind of glitches then the one studied in Section 6.1.1. These other glitches are much smaller but of course they consumes power, and this was not included in the calculation.

<table>
<thead>
<tr>
<th>Component</th>
<th>INV</th>
<th>add</th>
<th>D</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>First adder</td>
<td></td>
<td>19.27</td>
<td>26.41</td>
<td>45.68</td>
</tr>
<tr>
<td>Second adder</td>
<td></td>
<td>20.32</td>
<td>27.00</td>
<td>47.32</td>
</tr>
<tr>
<td>Subtractor</td>
<td></td>
<td>9.01</td>
<td>22.86</td>
<td>28.00</td>
</tr>
<tr>
<td>D flip-flops</td>
<td></td>
<td></td>
<td></td>
<td>290.46</td>
</tr>
<tr>
<td>Input inverter</td>
<td></td>
<td></td>
<td></td>
<td>3.28</td>
</tr>
<tr>
<td>Last pipelining stage</td>
<td></td>
<td></td>
<td></td>
<td>28.95</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>475.56</strong></td>
</tr>
</tbody>
</table>

Table 6.12. Simulated power consumption.
6.5 FROM POWER TO ENERGY

To avoid that the results are dependent on the used simulation clock frequency, the power is transformed to energy. This is simply done by dividing with the frequency, and the result is the average energy per bit, see (6.12). Notice that power consumption is linear to the clock frequency, so if for example the frequency is doubled so is also the power, and the energy is therefore constant.

\[
\text{Energy} = \frac{\text{Power}}{\text{Frequency}} \quad (6.12)
\]

EXAMPLE 6.4

Calculate the energy for the structure in Figure 5.7.

Power (see Example 6.1): 466.95 µW.

Simulation clock frequency: 4 MHz.

\[
\text{Energy} = \frac{466.95 \cdot 10^{-6}}{4 \cdot 10^6} \quad \text{Ws} \approx 116.74 \quad \text{pJ}
\]

6.6 LIMITATIONS

As mentioned in Chapter 4, the focus of this thesis is to find a strategy on how to find the best way to implement bit-serial multipliers. The part of this strategy discussed in this chapter, that is to find power models for the components, is far from optimal. More simulations are needed to get accurate results and if other components are used maybe other parameters have to be introduced. One big drawback is that nothing outside the multiplier is considered, the consequences of this is for example that the driving cost of inputs to adders/subtractors that is directly connected to the multiplier input is not included. However the main goal of this chapter, which was to show that it is possible to design a power model that can be applied to all possible structures of bit-serial multipliers, is achieved.
MATLAB PROGRAM

7.1 ABOUT THE MATLAB PROGRAM

To make use of the models developed in this report a Matlab program was developed. The input to the program is the fixed coefficient that is to be used in the multiplier. The output is the best way to implement the multiplier, with respect to number of adders/subtractors, number of flip-flops, latency and power consumption, respectively.

7.2 CHANGE PARAMETERS

The first thing to do before using the program is to change the parameters so that they fit the components to be used in the design. All parameters, with values derived in Chapter 6, are listed in Table 7.1.
### Table 7.1. Parameters

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameter name</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>clock_frequency</td>
<td>4</td>
<td>MHz</td>
</tr>
<tr>
<td>Multiplier</td>
<td>input_inverter_cost</td>
<td>2.85</td>
<td>µW</td>
</tr>
<tr>
<td>Multiplier</td>
<td>last_pl_cost</td>
<td>29.05</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>base_cost</td>
<td>28.25</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>input_driving_cost</td>
<td>0.20</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>D_driving_cost</td>
<td>0.00</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>add_driving_cost</td>
<td>0.50</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>D_load_cost</td>
<td>1.57</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>add_load_cost</td>
<td>9.05</td>
<td>µW</td>
</tr>
<tr>
<td>D flip-flop</td>
<td>sub_load_cost</td>
<td>0.97</td>
<td>µW</td>
</tr>
<tr>
<td>Adder</td>
<td>base_cost_add</td>
<td>12.65</td>
<td>µW</td>
</tr>
<tr>
<td>Adder</td>
<td>switch_cost_add</td>
<td>16.92</td>
<td>µW</td>
</tr>
<tr>
<td>Adder</td>
<td>glitch_cost_add</td>
<td>12.78</td>
<td>µW</td>
</tr>
<tr>
<td>Subtractor</td>
<td>base_cost_sub</td>
<td>12.65</td>
<td>µW</td>
</tr>
<tr>
<td>Subtractor</td>
<td>switch_cost_sub</td>
<td>18.45</td>
<td>µW</td>
</tr>
<tr>
<td>Subtractor</td>
<td>glitch_cost_sub</td>
<td>14.60</td>
<td>µW</td>
</tr>
<tr>
<td>Subtractor</td>
<td>inv_cost_sub</td>
<td>8.60</td>
<td>µW</td>
</tr>
<tr>
<td>Carry saving D flip-flop</td>
<td>base_cost_D</td>
<td>20.72</td>
<td>µW</td>
</tr>
<tr>
<td>Carry saving D flip-flop</td>
<td>switch_cost_D</td>
<td>28.42</td>
<td>µW</td>
</tr>
</tbody>
</table>

#### 7.3 GENERATE BEST CHOICE DATA

When the parameters have been changed a new complete graph search, as discussed in Section 3.3.2, is needed to find the best way to implement multiplication with all coefficients. This search algorithm is automatized, but is very time consuming.
CHAPTER 7 – Matlab Program

7.4 RUN THE PROGRAM

When the graph search is completed it is easy to find the best way to implement bit-serial multiplication with any fixed coefficient, see Example 7.1.

EXAMPLE 7.1

Find how to implement bit-serial multiplication with the coefficient 347 to as low energy cost as possible. Also find details such as carry switch probabilities. Use Matlab and the function \texttt{get\_coeff}.

\begin{verbatim}
>> get\_coeff(347, 'energy', 'details')

*** Best choice considering energy ***
Adders/Subtractors: 3, Graph: 2, Type: 1

\begin{tikzpicture}
  \node at (0,0) (A) {A};
  \node at (1,0) (B) {B};
  \node at (2,0) (D) {D};
  \node at (3,0) (F) {F};
  \node at (1,1) (C) {C};
  \node at (1,-1) (E) {E};

  \draw (A) -- (C);
  \draw (B) -- (D);
  \draw (D) -- (F);
  \draw (B) -- (E);

  \node at (0,-1.5) {A=1 B=4 C=1 D=2 E=-1 F=32};
  \node at (0,-2.5) {Flip-flops: A:0 B:2 C:0 D:1 E:0 F:5};
  \node at (0,-3.5) {Pipelining: A:0 B:0 C:0 D:0 E:1 F:1 output:1};

  \node at (0,-4.5) {Energy for design specific flip-flops: 73.25 pJ};

\end{tikzpicture}

\end{verbatim}

Notice that this is the same structure as studied in previous examples. The \texttt{exact} statement for each adder/subtractor refer to if the switch probability is calculated exact or not, see Section 5.3. Other optional arguments, besides \texttt{energy} and \texttt{details}, are \texttt{adder}, \texttt{flipflop} and \texttt{latency}. If no optional argument is given (or only \texttt{details}), one implementation for each of the four costs is received.
8

SUMMARY

8.1 CONCLUSION

The aim of this thesis was to find a strategy on how to implement bit-serial multipliers with as low cost as possible. Other studies in this area have focused on how to minimize the number of adders/subtractors, and often assumed that the cost for flip-flops is negligible. That simplification has in this report been shown to be far too great, and further not at all necessary. With a complete graph search it is rather straightforward to find the best structures considering latency, number of adders/subtractors and number of flip-flops. However the most interesting problem is how to minimize the power consumption. An important step to solve this problem was achieved by deriving formulas that can be used to calculate the carry switch probability in the adders/subtractors. It has also been established that it is possible to design a power model that can be applied to all possible structures of bit-serial multipliers.

8.2 FUTURE WORK

There is almost an endless number of matters that can be examined further. Here are some examples.

- Extend the graph set with all structures containing five adders/subtractors. This would make it possible to implement multiplication with all coefficients in the interval [-65536, 65536], see [6].
• Simplify the graph search. This can be done by making the code more efficient and by sorting out structures that in practice have been proven to be useless.

• Try to find formulas that solves the carry switch probability problem for all structures, and thereby replacing the correlation tables. This would of course have the advantage of exact results, but it would also speed up the graph search.

• Investigate how sign extension affects the switch probabilities.

• Improve the power model. To design an exact model is in principle impossible, but at least some improvement can be done.

• Develop a simple strategy to derive the needed parameters when new component designs or a new technology is to be used. A start can be to produce realistic testbenches.
REFERENCES


