Master Thesis

Design of Source Coupled Logic Circuits
for Ultra-Low Power Applications

Student: Michele MERCALDI
Project Supervisors: A.Tajalli, M.Stanisavljevic
Professor: Y. Leblebici

July 2007
Abstract

The trade-off between power consumption and performance has remained the main challenging issue in many modern applications, especially in low-power systems; this thesis will present an attempt to overcome the main difficulties in the design of ultra low-power circuits using new circuit techniques. The main goal of this work is the development of ultra low-power source-coupled logic (SCL) circuits. The idea is based on a circuit design technique to implement relatively linear and very high impedance resistors, which is very desirable in low power applications. To show the performances of the proposed circuits, a "demonstration" digital circuit (in this case a 8x8 bit Carry-Save Multiplier) has been designed and evaluated by simulation and implementation on a test chip.


Contents

1 Introduction ................................................. 1
  1.1 Motivation ............................................ 1
  1.2 Thesis Overview ...................................... 1

2 SCL for low-power consumption .......................... 3
  2.1 Source-Coupled Logic ................................ 3
  2.2 Design of SCL for Low-Power ......................... 5
    2.2.1 nMOS Source-Coupled Pairs Network ............... 6
    2.2.2 Bias Current Source .............................. 7
    2.2.3 pMOS Resistive Load ............................. 8
      2.2.3.1 Standard pMOS Load .......................... 8
      2.2.3.2 Drain-Bulk Connected pMOS Load .............. 9

3 SCL Gates Overview ....................................... 15
  3.1 SCL General Topologies ............................... 15
  3.2 SCL Gates for the Multiplier ......................... 16
    3.2.1 Layout ........................................ 17
    3.2.2 Timing .......................................... 19

4 SCL Demonstrator Circuit .................................. 22
  4.1 The Multiplier ...................................... 22
    4.1.1 Definitions ................................... 22
      4.1.1.1 Partial-Product Generation ................... 23
      4.1.1.2 Partial Product Accumulation ................. 23
      4.1.1.3 Final Addition ................................ 25
    4.1.2 The SCL Multiplier ................................ 25
      4.1.2.1 Realization ................................ 25
      4.1.2.2 Simulations ................................ 28
  4.2 Voltage Level Shifters ................................ 30
    4.2.1 Input Level Shifter .............................. 30
List of Figures

2.1 Topology of a generic SCL gate ................................. 4
2.2 Schematic of the cascode current mirror used as the bias current source 7
2.3 Drain-Bulk Connected pMOS .................................. 9
2.4 Cross section view of the proposed pMOS load device and parasitic elements .................................................. 10
2.5 Comparison between source-bulk connected (SB) and drain-bulk connected (DB) (W=240n L= 400n) ............................ 11
2.6 Drain current at $V_{SD} = V_{SWING} = 200mV$ in function of the voltage $V_{SG}$ in linear (a), and logarithmic scale (b) ..................... 12
2.7 Drain current for the nominal case $I_{bias} = 10nA$, $V_{swing} = 200mV$ ........ 13
2.8 Parasitic Diode in standard pMOS (left) and in Drain-Bulk Connected configuration (right) .................................................... 14
2.9 Topology of an ideal SCL inverter (a) and model of its actual implementation (b) ................................................................. 14
3.1 Topologies of a 2-level (a) and of 3-level generic gate (b) ........... 16
3.2 Implementation of an AND/NAND/OR/NOR gate with a unique structure ................................................................. 17
3.3 HALF ADDER components: 2-input XOR (a) and 2-input AND (b) 18
3.4 FULL ADDER components: 3-input XOR (left) and 3-input Carry (right) ................................................................. 18
3.5 Layout of a basic SCL gate (3-input XOR) .............................. 19
3.6 Test-bench for an AND gate ........................................... 20
3.7 Delay of the 2-input AND gate in function of the bias current and the supply voltage ....................................................... 21
4.1 Partial-product generation ........................................... 23
4.2 Array Multiplier ...................................................... 24
4.3 Carry Save Multiplier .................................................. 26
4.4 Layout of the SCL 8x8 bit Carry-Save Multiplier ....................... 27
4.5 Current consumption of the SCL multiplier .................................. 28
4.6 Voltage Levels for the SCL Multiplier ............................................ 30
4.7 SCL input (a) and output (b) level shifter schematics ..................... 31
4.8 Input Level Shifter Transient Simulation ........................................ 31
4.9 Input Level Shifter Transient Simulation ........................................ 32
4.10 SCL Input Level Shifter Layout .................................................... 33
4.11 SCL output Level Shifter Layout .................................................. 33
4.12 Principle of the Voltage Swing Controller (feedback loop is drawn in blue) .............................................................. 34
4.13 Schematic of the folded-cascade OTA ............................................. 35
4.14 Schematic of the voltage swing control circuit ................................. 36
4.15 Layout of the voltage swing control circuit ..................................... 37
4.16 Static Gain of the voltage swing control ........................................ 38
4.17 Phase Margin of the voltage swing control ...................................... 39
4.18 Limitations on the usable range of the SCL due to the voltage swing control (blue: fulfilled, green: not fulfilled) .................. 40
4.19 Schematic of the current divisor for OTAs and level shifters ............ 42
4.20 Layout of the current divisor for OTAs and level shifters ................ 42
4.21 Bias currents of the OTAs and the level shifters ............................... 42
4.22 Ratios between the current $I_{bias_{OTA,LS}}$ supplied from the pin and the bias currents of OTAs and level shifters respectively .................. 43
4.23 Schematic of the current divisor for the SCL gates ........................................ 43
4.24 Layout of the current divisor for the supply current of SCL gates ...... 44
4.25 Bias Current of the SCL gate .......................................................... 46
4.26 Ratios between the current $I_{bias_{SCLs}}$ supplied from the pin and the bias current of a SCL gate .................................................. 47

5.1 SCL Circuit Layout ............................................................... 49
5.2 Eye diagrams of output signal before level shifters ........................ 49
5.3 Eye diagram of output signal before level shifters: not operational case. ($I_{BIAS} = 100nA$, $f = 10$MHz) ................................. 50
5.4 Eye diagram of the output signal .................................................... 51
5.5 Current Consumption of the SCL Circuit for $I_{bias} = 1nA$, $Vdd = (0.5, 0.75, 1, 1.25)V$ .................................................. 52
5.6 Current Consumption of the SCL Circuit $I_{bias} = 1nA$ and $I_{bias} = 100nA$ 53

6.1 CMOS Multiplier Layout ........................................................ 55
6.2 Current consumption of the CMOS multiplier .................................. 57
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Structure of the CMOS Interface</td>
<td>58</td>
</tr>
<tr>
<td>6.4</td>
<td>CMOS Interface Layout</td>
<td>59</td>
</tr>
<tr>
<td>6.5</td>
<td>Finite State Machine of the control of the CMOS Interface</td>
<td>61</td>
</tr>
<tr>
<td>7.1</td>
<td>Merging of SCL gates</td>
<td>64</td>
</tr>
<tr>
<td>7.2</td>
<td>SCL gate performing ((A \cdot B) \oplus Ci \oplus D)</td>
<td>64</td>
</tr>
<tr>
<td>7.3</td>
<td>SCL gate with different driving capabilities (N=3)</td>
<td>65</td>
</tr>
<tr>
<td>8.1</td>
<td>Layout of the test circuit</td>
<td>67</td>
</tr>
<tr>
<td>A.1</td>
<td>(\alpha) vs (I_{bias})</td>
<td>70</td>
</tr>
<tr>
<td>A.2</td>
<td>(T_{RISE,SIM}) vs (T_{RISE,MOD})</td>
<td>71</td>
</tr>
<tr>
<td>A.3</td>
<td>Relative error of (T_{RISE}) value</td>
<td>71</td>
</tr>
<tr>
<td>A.4</td>
<td>Output node capacitances</td>
<td>72</td>
</tr>
<tr>
<td>B.1</td>
<td>2-input AND</td>
<td>74</td>
</tr>
<tr>
<td>B.2</td>
<td>2-input XOR</td>
<td>74</td>
</tr>
<tr>
<td>B.3</td>
<td>3-input CARRY</td>
<td>75</td>
</tr>
<tr>
<td>B.4</td>
<td>3-input XOR</td>
<td>75</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Functions obtained from the SCL 2-input AND gate by input/output inversion. ........................................... 16
3.2 SCL Gates Delays ......................................................... 20
4.1 Dimensions of the transistors of the OTA ............................. 36
5.1 Operating regions of the SCL multiplier (\(\checkmark\): correct computation; \(\times\): not correct computation) ................. 50
6.1 Performances of the CMOS carry-save multiplier ..................... 56
Chapter 1

Introduction

1.1 Motivation

Ultra low-power logic circuits are very desirable in many modern applications such as portable equipment or implanted biomedical systems. The source-coupled logic (SCL) topology has always been a very good candidate for implementing high speed logic circuits [3]. The state of the art already demonstrated that, compared to traditional CMOS logic, SCL gates exhibit various interesting features that make them suitable for an increasingly wide range of applications: SCL gates are faster (as the steering of the current between two branches is a very fast event), have a better power efficiency at high frequencies (as CMOS power consumption is frequency-dependent), and generate a much lower switching noise[1].

However, the design of ultra low-power SCL-based circuits still remains an important challenge. While several techniques for implementing CMOS logic circuits with transistors in the subthreshold regime and with very low power dissipation have been already introduced, the design of SCL circuits operating in subthreshold is yet an open research subject. The purpose of this project is to investigate a circuit design technique to implement relatively linear, very high impedance and compact resistors: in SCL gates such a resistor allows to reduce drastically the bias current and consequently the power consumption.

1.2 Thesis Overview

The thesis will present at first the characteristics of the SCL Circuit Design, with a deeper analysis about the ultra low-power features (Chapter 2). In Chapter 3 an overview of the general topologies of the SCL gates is presented, as well as a description of the ones used for the Demonstrator Circuit. The design of this circuit
is described in Chapter 4, while in Chapter 5 the results of its characterization are presented. The CMOS comparation circuit and the related simulations are presented in Chapter 6. In Chapter 7 a comparison between the performances of the SCL circuit and of the CMOS one is discussed, and some further improvements are briefly presented.
Chapter 2

Source-Coupled Logic for Ultra Low Power Consumption

2.1 Source-Coupled Logic

Source-Coupled Logic (SCL), also known as MOS Current-Mode Logic (MCML), is a differential logic based on the differential pair structure, as shown in Figure 2.1. The structure of an SCL gate can be divided in three parts:

- a **DC current source** to furnish the bias current $I_{bias}$ to the differential pair. It is usually realized with a current mirror, which can be cascaded in order to increase the output resistance of the mirror.

- a **nMOS network** composed by stacked source-coupled pairs which steer the bias current to one of the output branches according to the value of the differential input voltages $V_{id,i} = V_{ip,i} - V_{in,i}$

- a **resistive load** implemented with a pMOS transistor at each output branch, which converts the steered current into a differential output voltage $V_{od} = V_{op} - V_{on}$. 
2.1. Source-Coupled Logic

Figure 2.1: Topology of a generic SCL gate

The basic working principle of Source-Coupled Logic is to apply at each differential input $i$ of the SCL gate a sufficiently large voltage ($|V_{id,i}| > V_{id,min}$) in order to steer all the current into one of the two output resistances $R_{load}$.

The resistance where the bias current is flowing has a voltage drop equal to $\Delta V = R_{load}I_{bias}$, and pulls its output node to $Vdd_{SCL} - \Delta V$. The other resistance, where no current is flowing, has no voltage drop ($\Delta V = 0$) and pulls up its output node to $Vdd_{SCL}$. So the differential output voltage $V_{od}$ will switch between

$$V_{OH} = Vdd_{SCL} - (Vdd_{SCL} - \Delta V) = \Delta V = R_{load}I_{bias}$$

and

$$V_{OL} = (Vdd_{SCL} - \Delta V) - Vdd_{SCL} = -\Delta V = -R_{load}I_{bias}$$

The logical swing is then equal to

$$V_{LOG,swing} = V_{OH} - V_{OL} = 2 \cdot R_{load}I_{bias}$$

However, the voltage swing at the output $V_{SWING}$ will be considered as

$$V_{SWING} = R_{load}I_{bias}.$$

(2.1)
2.2. Design of SCL for Low-Power

We can notice that the output voltage swing \( V_{SWING} \) is independent from the supply voltage \( V_{ddSCL} \) and depends only on the resistance \( R_{load} \) and on the current \( I_{bias} \). It is important that the value of the output voltage, used to drive the following gate, is sufficiently high to completely steer the current in one branch of the next stage. So the voltage swing has to be larger than a minimum input value \( V_{id_{min}} \). This gives a first condition for the design of SCL which can be written as

\[
V_{SWING} > V_{id_{min}} \tag{2.2}
\]

Now it is possible to make an estimation of the delay of a SCL gate. The transient behavior of the switching operation is dominated by the effect of the load capacitance \( C_L \), associated to the load resistance \( R_{load} \). The propagation delay \( D_{SCL} \) of an SCL gate is proportional to the time constant \( R_{load}C_L \):

\[
D_{SCL} \propto R_{load}C_L = \frac{V_{SWING} \cdot C_L}{I_{bias}} \tag{2.3}
\]

The study of the parameters \( R_{load} \) and \( C_L \) is presented in Appendix A.

The power consumption \( P_{SCL} \) can be usually divided in a static and a dynamic component. The dynamic component, which is rather small, is due to the switching operation and is associated to the charging of the capacitance \( C_L \). It can be proved [2] that the sum of the static and dynamic components is a constant to first order, equal to the static power consumption, given by the product

\[
P_{SCL} = V_{ddSCL} \cdot I_{bias} \tag{2.4}
\]

Thus, in the following, the power consumption is assumed to be a purely static one. Eq.(2.4) brings the two straightforward key conditions to reduce the power consumption, i.e. the reduction of the supply voltage or/and the reduction of the bias current. Using Eq.(2.3) and Eq.(2.4), we obtain the power-delay product \( PD_{SCL} \)

\[
PD_{SCL} = V_{ddSCL} \cdot I_{bias} \cdot \frac{V_{SWING} \cdot C_L}{I_{bias}} = V_{ddSCL} \cdot V_{SWING} \cdot C_L \tag{2.5}
\]

2.2 Design of SCL for Low-Power

In this section, we will discuss the way to design a SCL gate which minimizes its power consumption, but also its power-delay product. Firstly we would like to

\[\text{The capacitance } C_L \text{ models the capacitance of the next gate (fanout), the wire capacitance as well as the transistor’s parasitic capacitances at the output node.}\]
reduce the supply current and voltage. Secondly, we want to analyze each term of the Eq. (2.5) and find its origin in the three parts of the SCL gate (the DC current source, the nMOS network, the pMOS load). Then we will focus on each part in order to reduce their effect on the power-delay.

The capacitance $C_L$ in Eq.(2.5) is due to the parasitic capacitances, to the transistors of the pMOS load and of the nMOS network, to the wire capacitances and to the input gate capacitance of the next gate.

The output voltage swing $V_{SWING}$ is limited by the minimum input voltage $V_{id,min}$ which allows to steer all the current into one output resistance and it is directly related to the nMOS network.

The supply voltage has to ensure a dynamic high enough for the voltage swing, for the saturation voltage $V_{ds,sat}$ of each level of the nMOS network and for the saturation of the transistors in the current source $V_{CS,sat}$:

$$V_{ddSCL,min} = V_{CS,sat} + i \cdot V_{ds,sat} + V_{SWING}$$

(2.6)

The technology used in this project is the UMC-180nm CMOS process. The parameters used to size the transistors of the SCL gate are summarized in Appendix C.

### 2.2.1 nMOS Source-Coupled Pairs Network

The nMOS transistors of the source-coupled pairs network are directly affecting the value of the minimum supply voltage, which is a very important parameter as the main purpose of this study is the low-power consumption. So it is necessary to have a minimum value for the voltage $V_{ds,sat}$, that is reached when the transistors are in weak inversion ($V_{ds,sat} \approx 3U_T - 5U_T \approx 100mV$).

The weak inversion condition is fulfilled only if the inversion factor $I_C$ is less than 0.1:

$$I_C = \frac{I_{bias}}{2n\beta U_T^2} = \frac{I_{bias}}{2n\mu C_{OX} \frac{W}{L} U_T^2} = \frac{I_{bias}}{2nK_p \frac{W}{L} U_T^2} < 0.1$$

(2.7)

As the expected tail current $I_{bias}$ ranges from 1nA to 200nA, we should ensure the weak inversion condition for the worst case ($I_{bias} = 200nA$), this results in:

$$\frac{W}{L} = \frac{I_{bias}}{2nK_p I_C U_T^2} \geq 4.5$$
2.2. Design of SCL for Low-Power

In order to ensure the weak inversion of the transistors, tighter conditions have been used \((I_C = 0.05 \text{ with } I_{bias} = 200\,\text{nA})\). Choosing minimal length to reduce area and capacitance, the following dimensions are obtained: \(W=1620\,\text{nm}\) and \(L=180\,\text{nm}\).

The condition of weak inversion for the nMOS transistors is fundamental for the aim of the project, as we will try to find a working frequency where the SCL technology offers advantages respect to the CMOS one. As in weak inversion the current flowing through the nMOS transistors will not depend on their dimensions, we will be allowed to change the tail current \(I_{BIAS}\) in order to tune the desired frequency without any modification in the layout.

2.2.2 Bias Current Source

The DC current source for the tail current is designed with a cascode current mirror, as shown in Figure 2.2. A high output resistance is very suitable to reduce the switching noise of the gate, so it is increased thanks to the cascode stage (transistors T2 and T2’). The transistors of the current mirror should work in saturation in order to ensure a current independent from the drain voltage of T2, i.e. a current source behavior. So the output voltage must be higher than the saturation voltage of the current mirror \(V_{CS,sat}\): to minimize \(V_{CS,sat}\) the transistors have to be carefully biased. Ideally we would like to have \(V_{CS,sat} = V_{P1} + V_{P2}\), where \(V_{P1}\) and \(V_{P2}\) are the pinch-off voltages of transistors T1’ and T2’. The topology is proposed in Figure 2.2, where the cascode stage is realized with low threshold voltage nMOS T2 and T2’, allows to tune the internal voltage \(V_A\). Thus, this configuration permits to have a self biased structure with a low saturation voltage \(V_{CS,sat}\). In order to have a good precision, we choose for the transistors the working condition of strong inversion. Let’s analyze more deeply how to bias the transistors in order to get a minimal \(V_{CS,sat}\).

![Figure 2.2: Schematic of the cascode current mirror used as the bias current source](image)

Both transistors T1 and T2 have to be in saturation. As T2 is in diode-connected
configuration, T2 is always in saturation as:

\[ V_{D2} = V_G > V_{P2} = \frac{V_G - V_{th2}}{n} \]

For T1 the condition of saturation can be written as:

\[ V_{D1} = V_A > V_{P1}. \tag{2.8} \]

The voltage at node A is given by:

\[ V_A = V_G - V_{DS,sat2} = V_G - \sqrt{\frac{2I_{ref}}{n\beta_2}} \]

Thus, the Eq.(2.8) can be written as:

\[ V_G - \sqrt{\frac{2I_{ref}}{n\beta_2}} \geq V_{P1} \tag{2.9} \]

\[ \beta_2 \geq \frac{2I_{ref}}{(V_G - V_{P1})^2} \approx \frac{2I_{ref}}{(V_{bias})^2} \tag{2.10} \]

With a nominal current \( I_{ref} = 10nA \) we get approximately the condition for the dimensions of the low-voltage transistor \( \frac{W}{L} > 5 \). So we get \( W = 2\mu m \) and \( L = 240nm \). Notice that this topology allows to get a minimal output voltage \( V_{CS,sat} \), a high output resistance due to the cascode stage and does not need any complex bias structure. Hence, the area and the power consumption will be minimized.

### 2.2.3 pMOS Resistive Load

The resistive load is used to convert the current into an output voltage and the desired output voltage swing is \( \Delta V = R_{load}I_{bias} = 200mV \). So for the requested bias current range \( (I_{bias} \in [1nA; 200nA]) \) the resistive load has to be tunable between \( 1M\Omega \) and \( 200M\Omega \). A huge resistance like that has to be implemented with a pMOS transistor, otherwise the dimensions of a “common” resistor would be dramatically big.

#### 2.2.3.1 Standard pMOS Load

The main condition to be verified in order to design such a load is that the voltage drop on the pMOS must be sufficiently high to control the next SCL gate. This means that the saturation voltage \( V_{DS,sat} \) should be larger than the voltage swing
2.2. Design of SCL for Low-Power

\[ V_{SWING} = V_{DSat} > V_{SWING} \]  

(2.11)

As the saturation voltage in weak inversion is too low \( V_{DSat,WI = 100mV} \), the pMOS transistor has to be in strong inversion. In this condition:

\[ V_{DSat} = \sqrt{\frac{2I_D}{n\beta}} \]

The minimum current \( I_D \) which ensures a sufficient voltage drop is obtained in the limit case \( V_{DSat} = V_{SWING} \) so:

\[ I_{D,\text{min}} = \frac{\beta n}{2} V_{DSat,\text{min}}^2 = \frac{\beta n}{2} V_{SWING}^2 = \frac{\mu C_{OX} W}{2} V_{SWING}^2 \]  

(2.12)

The way to minimize the drain current is to increase the length \( L \) of the pMOS transistor. In order to obtain the required \( V_{SWING} \) and to stay in the desired range of bias current \( I_{\text{bias}} \), we obtain resulting dimensions that are excessively big compared to the rest of the design. In this case the price to pay in term of area (and also parasitic capacitance) is too high.

2.2.3.2 Drain-Bulk Connected pMOS Load

The problem of the previous configuration of the pMOS load is due to the narrowness of the linear region. In this section we will try to improve this linear region thanks to the dependence of the threshold voltage \( (V_{TH}) \) on the substrate potential. The proposed load device consists of a pMOS transistor where the bulk terminal (n-well) is connected to the drain \cite{3}, as shown in Figure 2.3.

![Figure 2.3: Drain-Bulk Connected pMOS](image)

Due to the well-known dependence of the threshold voltage \( V_{TH} \) on the substrate potential, the drain current of this device keeps increasing with increasing \( V_{SD} \), instead of reaching a saturation value. As shown in Figure 2.5, this will result in
Figure 2.4: Cross section view of the proposed pMOS load device and parasitic elements

a finite large equivalent resistance instead of almost infinite output impedance of conventional pMOS device. Notice that the “linear behavior” is really extended beyond the requested nominal swing $V_{SWING} = 200mV$. 
2.2. Design of SCL for Low-Power

Figure 2.5: Comparison between source-bulk connected (SB) and drain-bulk connected (DB) (W=240n L= 400n)

Using the EKV model for the weak inversion [4], we get Eq.(2.13) for the drain current $I_{SD}$; notations are referred to Figure 2.3.

$$I_{SD} = 2n\beta U_T^2 \exp \left( \frac{V_G - V_{T0}}{nU_T} \right) \left( \exp \left( \frac{-V_S}{U_T} \right) - \exp \left( \frac{-V_D}{U_T} \right) \right)$$

(2.13)

Yet, we have $V_D = V_{DB} = 0$, $V_S = -V_{SD}$ and $V_G = V_{SG} - V_{SD}$; then Eq.(2.13) becomes:

$$I_{SD} = I_{SD0} \exp \left( \frac{-V_{SD}}{nU_T} \right) \left( \exp \left( \frac{V_{SD}}{U_T} \right) - 1 \right).$$

(2.14)

where

$$I_{SD0} = 2n\beta U_T^2 \exp \left( \frac{-V_{T0}}{nU_T} \right) \exp \left( \frac{V_{SG}}{nU_T} \right)$$

For a source-drain voltage $V_{SD}$ lower to 0.4V, the behavior of $I_{SD}$ is typical of the subthreshold region and follows the one described in Eq.(2.14). However, if
Figure 2.6: Drain current at $V_{SD} = V_{SWING} = 200\text{mV}$ in function of the voltage $V_{SG}$ in linear (a), and logarithmic scale (b)

$V_{SD}$ exceeds 0.4V, the PN junction of source-well becomes forward biased and the current $I_{SD}$ increases. So, for what concerns the voltage swing needed to switch the differential pair of an SCL gate, we should have $V_{SD} < 400\text{mV}$ (the typical value is $V_{SD} = 200\text{mV}$).

The current $I_{SD}$ (and thus the resistance $R_{SD}$) can be tuned thanks to the “constant” term $I_{SD0}$, which depends on the gate voltage $V_{SG}$. Thus the current can be easily adjusted thanks to $V_{SG}$. The dependence between $I_{SD}$ and the voltage $V_{SG}$ for a transistor whose the dimensions are $W=240\text{n}$ and $L=400\text{n}$ is illustrated in Figure 2.6 (simulation with BSIM3V3 model).

For a nominal voltage swing $V_{SWING} = V_{SD} = 200\text{mV}$, the current $I_{SD}$ covers all the desired range [$1\text{nA}; 200\text{nA}$] when the gate voltage varies between $V_{SG} = 540\text{mV}$. Increasing the dimensions of the pMOS transistor allows to reduce the voltage $V_{SG}$ to reach 200nA.

According to the bias current desired for the speed requirements, the voltage has to be fixed at the good value. This is done thanks to a voltage swing controller (or replica bias) which will be discussed in Chapter 4.3. Figure 2.7 shows as an example the nominal case,
2.2. Design of SCL for Low-Power

![Graph showing current vs. voltage for a nominal case.]

Figure 2.7: Drain current for the nominal case $I_{bias} = 10 nA$, $V_{swing} = 200 mV$

By using this kind of configuration it is possible to implement the load resistor using a small size transistor, which is relatively linear for a wide voltage swing range. However, the $I-V$ characteristics of the device depends exponentially on $V_{TH}$, so the behaviour of the device is sensitive to the mismatch. Another drawback respect to the conventional pMOS load is that with this configuration the n-well / p-substrate junction parasitic diode is connected to the output, as shown in Figure 2.8. This diode must be included in the simulations in order to perform a correct analysis of the circuit.
2.2. Design of SCL for Low-Power

Figure 2.8: Parasitic Diode in standard pMOS (left) and in Drain-Bulk Connected configuration (right)

Figure 2.9: Topology of an ideal SCL inverter (a) and model of its actual implementation (b)
Chapter 3

SCL Gates Overview

3.1 SCL General Topologies

The complexity of an SCL gate is related to the structure of the nMOS differential pairs network. As the number of stacked differential pairs (we will refer to them as “levels”) increases, the number of possible implementations is higher.

We can notice from Fig. 3.1 that a wide range of boolean functions can be implemented by properly configuring an SCL universal block. This standard structure allows to have a common physical framework for all the gates. Since inverted and non-inverted outputs are available, and the negation of one input of a gate is achieved by switching the positive and negative ends of the differential signal, the function implemented by a gate can be varied in multiple ways [6]. For example, a gate realizing the 2-input AND function will also realize the NAND function, as well as many functions resulting of inverting the inputs and output in all combinations, as shown in Table 3.1.

So using a 2-level configuration it is possible to realize a wide range of logic functions: 2-input AND / NAND, 2-input OR / NOR, 2-input XOR / XNOR, 2-to-1 MUX, D-Latch. In the same way a 3-levels configuration can be used to implement various logical functions: 3-input AND/NAND, 3-input OR / NOR, 3-input XOR / XNOR, 3-input AND-OR, 3-input OR-AND. An example of this capability is shown in Figure 3.2, where a multi-function gate is obtained using the same SCL topology. In terms of timing characteristics, this means that we have a very uniform distribution of the delays (for gates with the same number of levels). This allows to have a path delay that is more predictable and less sensible to variability [5]. Notice that when a differential pair of the nMOS network is not needed (for example in the 2-input AND gate) it is not simply substituted with a short circuit but with an nMOS transistor whose gate is connected to $V_{dd_{SCL}}$, in order to keep the same
“current path length” from the current mirror to the pMOS loads for all the possible inputs.

### 3.2 SCL Gates for the Multiplier

The number of differential pair levels directly affects the value of the minimum supply voltage $V_{dd_{SCL}}$. As the aim of the project is the low power consumption, 2 and 3-levels gates have been chosen to implement cells of the multiplier. As explained in 4.1, the multiplier is composed mainly by Adder gates. Half Adder

<table>
<thead>
<tr>
<th>$A \cdot B$</th>
<th>$\overline{A} \cdot \overline{B}$</th>
<th>$A + B$</th>
<th>$\overline{A} + \overline{B}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \cdot B$</td>
<td>$\overline{A} \cdot \overline{B}$</td>
<td>$A + B$</td>
<td>$\overline{A} + \overline{B}$</td>
</tr>
<tr>
<td>$\overline{A} \cdot \overline{B}$</td>
<td>$A \cdot B$</td>
<td>$A + B$</td>
<td>$A + \overline{B}$</td>
</tr>
<tr>
<td>$A \cdot \overline{B}$</td>
<td>$\overline{A} \cdot \overline{B}$</td>
<td>$A + B$</td>
<td>$A + \overline{B}$</td>
</tr>
<tr>
<td>$\overline{A} \cdot \overline{B}$</td>
<td>$\overline{A} \cdot B$</td>
<td>$\overline{A} + \overline{B}$</td>
<td>$\overline{A} + \overline{B}$</td>
</tr>
</tbody>
</table>

Table 3.1: Functions obtained from the SCL 2-input AND gate by input/output inversion.
cells are realized with a 2-input XOR gate to calculate the sum and an AND gate to calculate the carry-out (Figure 3.3). Full Adder cells are realized with a 3-input XOR gate to calculate the sum and 3-input Carry gate (Figure 3.4). Other AND gates are needed in order to calculate the partial products (see 4.1.1.1).

### 3.2.1 Layout

As explained in 3.1, different SCL gates can be implemented with very similar structures. This characteristic is even more evident when the layout is designed. In fact the general position of the components is the same for all the gates, and only the nMOS network is modified. The layout has been designed in order to have all the supply voltages running in horizontal direction, and the I/O pins on the same side of the structure. A sample structure is shown in Figure 3.5. It is possible to distinguish the cascaded current mirror branch on the top, over the 2 pMOS loads, and the nMOS network at the bottom. The resulting area is about 65μm². The layouts of the other cells used for the realization of the SCL multiplier are presented in Appendix B.
3.2. SCL Gates for the Multiplier

Figure 3.3: HALF ADDER components: 2-input XOR (a) and 2-input AND (b)

Figure 3.4: FULL ADDER components: 3-input XOR (left) and 3-input Carry (right)
3.2. SCL Gates for the Multiplier

3.2.2 Timing

In order to characterize each cell in the correct way, a proper test-bench has been realized. The differential input signals goes through a buffer stage before being processed by the gate under test; this step is necessary in order to obtain a realistic input waveform instead of an ideal one. The output load includes another identical cell to be driven and the wire capacitances. The value of the 2 wire capacitances (one for each line of the differential signal) has been calculated as:

\[ C_{L,WIRE} = \varepsilon_{SiO_2} \cdot \frac{L \cdot W}{t} \]

where \( \varepsilon_{SiO_2} = 3.9 \times \varepsilon_0, \varepsilon = 8.85 \times 10^{-12} \), \( t \) is the thickness of the \( SiO_2 \) layer \((23 \times 10^{-7} m)\), \( W \) is the width of a routing metal line \((0.28 \mu m)\), and the value of \( L \) is conventionally calculated as 20 times the diagonal of the layout of the gate under test.

To characterize each gate all the possible combinations of switching input have been studied, and finally only the worst case has been considered. For example to characterize the AND gate it has been observed that the worst case delay is verified when one input is stable at the logic value “1” and the other switches from “0” to “1”. Notice that, as shown in Figure 3.7, the inputs of the gate can not be considered symmetrically: in fact when the gate is driven by the input A (lowest level in NMOS network) the delay is slightly larger than in the case of input B (highest level in NMOS network). This is due to a different electrical path between
3.2. SCL Gates for the Multiplier

![Test-bench for an AND gate](image)

Figure 3.6: Test-bench for an AND gate

<table>
<thead>
<tr>
<th>I_bias</th>
<th>AND</th>
<th>CARRY3</th>
<th>XOR2</th>
<th>XOR3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delay</td>
<td>T rise</td>
<td>T fall</td>
<td>Delay</td>
</tr>
<tr>
<td>1 nA</td>
<td>4.0us</td>
<td>12us</td>
<td>13.5us</td>
<td>4.0us</td>
</tr>
<tr>
<td>10 nA</td>
<td>460ns</td>
<td>1.1us</td>
<td>1.3us</td>
<td>391ns</td>
</tr>
<tr>
<td>100 nA</td>
<td>46ns</td>
<td>106ns</td>
<td>165ns</td>
<td>45ns</td>
</tr>
<tr>
<td>200 nA</td>
<td>23ns</td>
<td>54ns</td>
<td>88ns</td>
<td>23ns</td>
</tr>
</tbody>
</table>

Table 3.2: SCL Gates Delays

the input transistor and the output. The lowest transistor will see an additional time constant RC due to the upper transistor.

The resulting delay values of the designed gates are shown in Table 3.2. Simulations have been run for different values of bias current $I_{\text{BIAS}}$. 
3.2. SCL Gates for the Multiplier

Figure 3.7: Delay of the 2-input AND gate in function of the bias current and the supply voltage
Chapter 4

SCL Demonstrator Circuit

4.1 The Multiplier

Multiplications are slow and expensive operations. The performance of many computational problems is often dominated by the speed at which a multiplication operation can be executed [8]. That’s why complete multiplication units are often integrated in state-of-the-art signal and microprocessors. A multiplier has been chosen here to make a complete “demonstrator” of the new approach in SCL circuit design technique.

4.1.1 Definitions

Consider two unsigned binary numbers X and Y that are M and N bits wide respectively. To introduce the multiplication operation, it is useful to express X and Y in a binary representation,

\[ X = \sum_{i=0}^{M-1} X_i 2^i \]

\[ Y = \sum_{j=0}^{N-1} Y_j 2^j \]

with \( X_i, Y_i \in \{0, 1\} \).

The multiplication operation is defined as follows:

\[ Z = X \times Y = \sum_{i=0}^{M+N-1} Z_k 2^k = \left( \sum_{i=0}^{M-1} X_i 2^i \right) \left( \sum_{j=0}^{N-1} Y_j 2^j \right) = \]

22
The simplest way to perform a multiplication is to use a single two-input adder; for inputs that are M and N bits wide, the multiplication takes M cycles using an N-bit adder. This shift-and-add algorithm for multiplication adds together M partial products. Each partial product is generated by multiplying the multiplicand with a bit of the multiplier, and by shifting left the result based on the position of the multiplier bit.

A faster way to implement multiplication is to resort to an approach similar to manually computing a multiplication: all the partial products are generated at the same time and organized in an array, then a multi-operand addition is applied to compute the final product. The resulting structure is called “array multiplier”. It combines three functions: partial-product generation, partial-product accumulation, and final addition.

### 4.1.1.1 Partial-Product Generation

Partial products result from the logical AND of multiplicand X with a multiplier bit Y<\text{i} \rangle \ (Figure \ 4.1).

Each row in the partial product array is either a copy of the multiplicand, or a row of zeroes.

### 4.1.1.2 Partial Product Accumulation

After the partial products are generated, they have to be summed. This accumulation is essentially a multi-operand addition. A straightforward way to accumulate partial products is by using a number of adders that form an array. This is why it is called an array multiplier.
Array Multiplier  The composition of an array multiplier is shown in Fig.4.2. There is a one-to-one topological correspondence between this hardware structure and the manual multiplication shown in the figure. The generation of N partial products requires \( N \times M \) 2-bit AND gates. Most of the area of the multiplier is devoted to the adding of the N partial products, which requires N-1 M-bit adders. The shifting of the partial products for their proper alignment is performed by simple routing and does not require any logic. The overall structure can easily be compacted into a rectangle, resulting in a very efficient layout.

Due to the array organization, determining the propagation delay of this circuit is not straightforward as a large number of almost identical length paths can be identified:

\[
t_{MULT} \approx [(M - 1) + (N - 2)] t_{CARRY} + (N - 1) t_{SUM} + t_{AND} \quad (4.1)
\]

where \( t_{CARRY} \) is the propagation delay between input and output carry, \( t_{SUM} \) the delay between the input carry and sum bit of the full adder, and \( t_{AND} \) the delay of the AND gate.

Since all critical paths have the same length, speeding up just one of them - for instance by replacing one adder by a faster one such as a carry-select adder - does not make much sense. All critical paths have to be attacked at the same time. From Eq.
4.1. The Multiplier

4.1, it can be deduced that the minimization of $t_{MULT}$ requires the minimization of both $t_{CARRY}$ and $t_{SUM}$. In this case it could be beneficial for $t_{CARRY}$ to be equal $t_{SUM}$. This contrasts with the requirements for adder cells, where a minimal $t_{CARRY}$ has prime importance.

**Carry-Save Multiplier** Due to the large number of almost identical critical paths, increasing the performance of the previous structure through transistor sizing yields marginal benefits. A more efficient realization is obtained by noticing that the multiplication result does not change when the output carry bits are passed diagonally downwards instead only of to the right. As shown in Figure 4.3, an extra adder is added (called a vector-merging adder) to generate the final result. Such a multiplier is called a carry-save multiplier, because the carry bits are not immediately added, but are rather "saved" for the next adder stage. In the final stage, carries and sums are merged in a fast carry-propagate adder stage. While this structure has a slightly increased area cost (one extra adder), it has the advantage that its worst-case critical path is uniquely defined:

$$t_{MULT} = t_{AND} + (N + 1) t_{CARRY} + t_{MERGE}$$

where $t_{MERGE} = (N - 1) t_{CARRY}$

still assuming that $t_{AND} = t_{CARRY}$.

4.1.1.3 Final Addition

The final step to complete the multiplication is the combination of the result through the “merging vector” at the bottom of the structure, which is necessary if the Carry-Save configuration is used. As said before an area overhead is needed to realize such a vector [8].

4.1.2 The SCL Multiplier

4.1.2.1 Realization

The Carry-Save structure has been chosen to implement the SCL multiplier. The price to pay in terms of area (for the realization of the final merging vector) is well spent as the speed of the multiplication is much higher. The Carry-Save configuration does not complicate the routing, as it is still very regular.

Implementing the multiplier with the gates presented in (3) we obtain:
$t_{MULT} = t_{AND} + t_{HAC} + (N - 2) \cdot t_{FA,C} + t_{HAC} + (N - 3) \cdot t_{FA,C} + t_{FA,S} = \\
= 3 \cdot t_{AND} + (2N - 5) \cdot t_{FA,C} + t_{FA,S}$

(4.2)

Where $t_{AND}$, $t_{FA,C}$, $t_{FA,S}$, $t_{HAC}$ are respectively the propagation delays of the 2-input Gate, of the Carry bit of the Full Adder, of the Sum bit of the Full Adder and of the Carry bit of the Half Adder respectively. All the values are shown in Table 3.2. Decomposing the adders in their basic gates and using $N = 8$, the worst-case delay becomes

$t_{mult} = 3 \cdot t_{AND} + 12 \cdot t_{Carry}$

As all the basic gates of the SCL multiplier (AND, XOR2, XOR3, Carry) have the same structure and area, the overall structure can easily be compacted into a rectangular and efficient layout, shown in Figure 4.4. The area of the layout is about $22400 \mu m^2$. The multiplicand $A<7:0>$ is coming from the left side, while the multiplicand $B<7:0>$ comes from the top. Horizontal routing (Vdl, Vss, Vbp, Vbn) is realized in Metal 2, vertical routing (input data, and current mirrors biasing) is
realized in Metal 3. The total circuit contains:

- 48 Full Adder Gates
- 8 Half Adder Gates
- 64 AND Gates

Hence, the total current consumption should be around 176 times the basic current:

\[ I_{bias,TOT} = (2 \cdot 48 + 2 \cdot 8 + 64) \cdot I_{bias} = 176 \cdot I_{bias} \]
4.1. The Multiplier

4.1.2.2 Simulations

In Figure 4.5 the results of the simulated current consumption are shown. This analysis shows that the total one is really 176 times the bias current of one gate. There is just an unexpected result for \( I_{\text{bias}} = 100\,nA \) and \( Vdd_{\text{SCL}} = 0.5V \).

In the normal cases the current consumption is 17.6\( \mu A \) for \( I_{\text{bias}} = 100\,nA \) and 178.1\( nA \) for \( I_{\text{bias}} = 1\,nA \). Delay and switching noise are well independent from the supply voltage. At \( I_{\text{bias}} = 100\,nA \) we have a delay around 410ns while for \( I_{\text{bias}} = 100\,nA \) it is about 51\( \mu s \). The switching noise is about 6\% for \( I_{\text{bias}} = 100\,nA \) and 16\% for \( I_{\text{bias}} = 1\,nA \).

However, for \( I_{\text{bias}} = 100\,nA \) and \( Vdd_{\text{SCL}} = 0.5V \) the current consumption is reduced to 16.5\( \mu A \) and the delay is increased of 100ns. The switching noise is about 13\%, i.e. the double of the normal case. This is due to the fact that in these working conditions the current mirror of the SCL gate starts to leave saturation. Therefore it cannot produce the desired current and the global consumption is reduced. During the switching operation, the node between the NMOS network and the drain...
of the current mirror has a small voltage variation. This voltage is directly translated in current variation, due to the non-saturation condition of the current mirror. Therefore the switching noise is increased.
4.2 Voltage Level Shifters

Voltage Level Shifters are needed in order to allow the interfacing of the SCL multiplier with the external signals and supplies. In fact the SCL gates need a differential signal with a swing between $V_{dd_{SCL}}$ and $V_{REF} = V_{dd_{SCL}} - \Delta V$, while the external signals are single-ended and have a swing between $V_{dd}$ and ground. In Figure 4.6 the needed voltage levels are shown, as well as an overview of the whole system.

4.2.1 Input Level Shifter

This level shifter is actually performing a transformation of the single-ended signal into the differential one, needed for the SCL gates. The pull-up network is realized with 2 cross-coupled pMOS transistors, driven by a dual pull-down network (Figure 4.7). In this way a differential signal between ground and the supply voltage is obtained; the reduction of the swing to $\Delta V$ is not necessary, as it will be reached automatically inside the multiplier, right after the first SCL gate.

Transistors are sized in order to not affect the speed of the whole circuit, being its delay negligible respect to the one of the multiplication. This layout results very compact (Figure 4.10) and it is not affecting the area of the whole structure. Only power supply connections are needed. Post-Layout simulations show a global delay of about 2ns.
4.2. Voltage Level Shifters

Figure 4.7: SCL input (a) and output (b) level shifter schematics

Figure 4.8: Input Level Shifter Transient Simulation
4.2. Voltage Level Shifters

4.2.2 Output Level Shifter

The SCL Multiplier generates 16 differential signals with a swing of $\Delta V$; to analyze these results it is necessary to obtain single-ended signals with a proper swing. Each level converter is realized with a comparator whose output saturates to $vdd$ or $vss$ according to his differential input (Figure 4.7). In this way the OUT signal is single-ended and has a swing between ground and the required $vdd_s$.

Transistors are sized with minimal length; output transistors width has been chosen high enough to ensure high speed, with:

$$W_{pMOS} = 3 \times W_{nMOS}$$

to obtain the same value for rise and fall time. The need of the current mirror makes the layout a bit larger respect to the input converter one, but it still does not affect the total area (Figure 4.11).

The delay of the circuit is about $t_{LS,\uparrow} = 220\text{ns}$ for the rising edge and about $t_{LS,\downarrow} = 110\text{ns}$ for the falling edge. This big delay has to be taken into account when the SCL circuit is tested, particularly when the SCL gates are biased with a large current.
4.2. Voltage Level Shifters

Figure 4.10: SCL Input Level Shifter Layout

Figure 4.11: SCL output Level Shifter Layout
4.3 Voltage Swing Control

4.3.1 Principle of the Voltage Control

The main purpose of the voltage swing control circuit (also called replica bias) is to adjust the pMOS load resistance, which is controlled by the gate voltage \( vbp \). The idea is to use a feedback loop in order to set the voltage swing \( V_{SWING} \) equal to a reference one \( (V_{ref}) \). This feedback loop, drawn in blue in the Figure 4.12, is composed by an OTA and one pMOS load transistor. An excellent robustness of the circuit is reached when we have a gain \( A_0 \) superior to 60dB and a phase margin higher than 60°. Here the regulation system is submitted to the large variations of the current \( I_{bias_{SCL}} \) and of the supply voltage \( V_{dd_{SCL}} \). The changes of \( I_{bias_{SCL}} \) have a big impact on the gain of the pMOS transistor behavior, as well as the variation of the supply voltage have an impact on both the OTA and the pMOS transistor. This means that the circuit has to work in very large range, and it is very tricky or even impossible to fulfill the upper gain and phase margin requirements for all the cases. For this circuit the minimum gain ensuring a good functioning is about 40dB\(^1\), and the minimum phase margin has to be higher than 45°.

\[ \Delta V = V_{swing} \]

\[ V_{dd_{SCL}} \]

\[ V_{ref} \]

\[ vbp \]

\[ I=0 \]

\[ I_{bias_{SCL}} \]

\[ vbn \]

Figure 4.12: Principle of the Voltage Swing Controller (feedback loop is drawn in blue)

\(^1\)A gain of 40dB means that the voltage \( V_{SWING} \) can have a difference of 1% from the expected value.
4.3. Voltage Swing Control

The design of the OTA is realized for a nominal case with the parameters: $I_{bias_{SCL}} = 10\, \text{nA}$ and $V_{dd_{SCL}} = 1\, \text{V}$. For the nominal case we expect $a$ to have $A_0 > 60\, \text{dB}$ and $PM > 60^\circ$.

The voltage swing control or replica bias can be used for biasing several gates. This is possible because all the transistors of the pMOS load have the same size for all the gates. For the multiplier circuit we will use replica bias on each side of the multiplier, so as to guarantee a uniform distribution of the bias voltage $v_{bp}$. In Figure 4.12 the nMOS network is composed by a single pair for the sake of simplicity. However in the real implementation of the replica bias, we use an AND gate in order to get a better matching with the gates of the multiplier.

4.3.2 The OTA

The operational transconductance amplifier (OTA) has to be able to work in low-voltage range and to have a big gain to ensure the stability. This can be achieved with the Folded-Cascode OTA presented in Figure 4.13. The bias current $I_{bias_{OTA}}$ of the OTA is fixed to $10\, \text{nA}$. The corresponding sizes of the transistors are mentioned in Table 4.1.
4.3. Voltage Swing Control

![Figure 4.14: Schematic of the voltage swing control circuit](image)

<table>
<thead>
<tr>
<th>Transistor names</th>
<th>Width W [m]</th>
<th>Length L [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>500n</td>
<td>5μ</td>
</tr>
<tr>
<td>lVCM</td>
<td>2μ</td>
<td>240n</td>
</tr>
<tr>
<td>lVD</td>
<td>3μ</td>
<td>300n</td>
</tr>
<tr>
<td>lVF</td>
<td>5μ</td>
<td>500n</td>
</tr>
<tr>
<td>P1</td>
<td>3μ</td>
<td>300n</td>
</tr>
<tr>
<td>P2</td>
<td>1μ</td>
<td>3μ</td>
</tr>
</tbody>
</table>

Table 4.1: Dimensions of the transistors of the OTA

### 4.3.3 Implementation of the Voltage Swing Control

The voltage swing control necessitates 2 bias currents: \( I_{bias_{SCL}} \) and \( I_{bias_{OTA}} \). The first is realized with a current mirror, which is completely implemented into the cell, as represented in Figure 4.14. The gate voltage \( vbn \) of the current mirror is then used to polarize the current mirrors of the SCL core as well as the internal AND gate. So, this cell provides the two bias voltages \( vbn \) and \( vbp \) needed for the SCL gates.

The current \( I_{bias_{OTA}} \) is set into the OTA by the gate voltage \( vbn_{OTA} \), exactly in the same way we provide the \( vbn \) voltage for the core. The voltage \( vbn_{OTA} \) comes from the Current Divider (see Chapter 4.4).

The corresponding layout requires an area equal to 685μm² and is drawn in Figure 4.15.
4.3. Voltage Swing Control

4.3.4 Voltage Swing Control Simulations

The simulation of the voltage swing control is done from the av_extracted view of the layout (see Figure 4.15) of voltage swing control. The capacitance $C_{OTA}$ is needed to ensure a sufficient phase margin. The phase margin falls down when the bias current $I_{biasSCL}$ tends to 1nA. In order to limit this effect, we use a huge capacitance $C_{OTA} = 447 \text{fF}$, whose area is equal to $75 \mu\text{m}^2$. Figure 4.16 shows the static gain $A_0$ in function of the supply voltage for different bias current $I_{biasSCL}$. The lowest value of the gain which guaranties the correct operation of the voltage swing control is 40dB. So when the $I_{biasSCL}$ is larger than 15nA, the supply voltage has to be bigger than 500mV. For instance, when $I_{biasSCL} = 50nA$ we have $V_{ddSCL,min} = 576mV$ and in the worst case for $I_{biasSCL} = 200nA$ we have $V_{ddSCL,min} = 645mV$.

The phase margin, depicted in Figure 4.17, remains in any conditions higher than 45°, and when the current exceeds slightly 1nA, the phase margin is higher than 60°.

Concerning the gain bandwidth product, we obtain a value higher than 150kHz for all the cases that had a gain superior to 40dB. As the voltage swing control should work in DC regime, this value is sufficient.

In summary, due to the gain drop for the big currents, the functioning of the SCL gates cannot be guaranteed for the whole range of $I_{biasSCL}$ and $V_{ddSCL}$. A good estimation of the limit between working and not working is given by the 40dB border, which is represented in Figure 4.18.
Figure 4.16: Static Gain of the voltage swing control
4.4. Current Mirrors

Reference currents for the SCL gate, for the OTAs and for the level shifters have to be supplied from the output of the chip. As we are dealing with very low currents (down to 1nA), it is very hard or even impossible to furnish it directly from the pins. The design of current divisors is thus needed for the demonstrator chip. The bias current of the level shifters and of the OTAs are fixed and have the same order of magnitude, then the current divisor will share the same circuit. On the other hand SCL gates use supply currents varying between 1nA to 200nA. This means that SCL gates require their own circuit.

For both circuits, the specifications will be the same, As the minimum current that can be supplied from the pin is in the order of 1μA, the ratio of the current divisors is fixed to $1/1000$. Moreover, the precision of the dividing factor has to guarantee a relative error lower than 5%, for common measurements.
4.4 Current Mirrors

Figure 4.18: Limitations on the usable range of the SCL due to the voltage swing control (blue: fulfilled, green: not fulfilled)

4.4.1 Current Mirrors for OTAs and Level Shifters

The current divisor for OTAs and level shifters is driven by a $10\mu A$ input current but has to produce an output voltage, which is the bias voltage for every replica bias and all the level shifters. The global framework of the current divisor is drawn in the Figure 4.19.

Figure 4.19: Schematic of the current divisor for OTAs and level shifters
4.4. Current Mirrors

The basic idea is to realize the division of the current thanks to two cascode current mirrors, a NMOS one and a PMOS one. The structure of the current mirror rest on the same principle of the current mirror used in the SCL gate (see 2.2.2). The first one, composed with NMOS transistors realizes a division by 40. This means that we need \(41^2\) cascode cells with exactly the same transistors, for matching purpose. The diode-connected transistors N1' and N2' are duplicated 40 times while we have only one element with the transistors N1 and N2. Therefore we get a current equal to \(I_{bias_{OTA,LS}}\), which flows directly in the diode-connected transistors P1' and P2' of the PMOS current mirror. These transistors are duplicated 25 times to make a second division. At the output of the PMOS current mirror, we will have 3 different branches: one branch for the level shifters and two identical branches for the OTAs.

The OTAs need a bias current of 10 nA, so we just have one cell with P1 and P2. This cell is connect to the half-part of a NMOS current mirror. This half-part contains the diode-connected transistors and is went trough by the reference current. Two identical branches are devoted for the OTAs, one for the OTAs of each side of the multiplier block.

The last output is used by the level shifters, which need a 30nA bias current. Thus we have merely three cells P1-P2 in parallel that provide the current. Exactly as the OTA case there is a half-part current mirror (see Figure 4.19).

The corresponding layout\(^3\) is depicted in the Figure 4.20 and exhibits four distinct parts. Note that the input current \(I_{bias_{OTA,LS}}\) comes from the upper side and the supply voltages and the gnd arrives in the middle of the layout. The area covered by the layout is around \(2370\mu m^2\). The highest metalization of the layout is Metal 2.

After the extraction from the layout of the parasitic capacitances, the circuit is simulated using the Virtuoso® Spectre® circuit simulator. The bias currents of level shifters and OTAs are obtained in an “real” (av_extracted) external circuit in order to have an accurate simulation. The circuit is tested with a supply voltage \(V_{dd}=1.8\text{V}\) for the current mirror, and a variable supply voltage for the OTAs. The input current \(I_{bias_{OTA,LS}}\) is equal to \(10\mu\text{A}\). The corresponding bias currents for OTAs and for level shifters are plotted in Figure 4.21.

\(^2\)Actually, two dummy cascode cells are added on each side of the layout for matching purpose. The total number of NMOS cascoded cells is thus equal to 43

\(^3\)The dimensions used are the ones indicated in the Figure 4.19
4.4. Current Mirrors

Figure 4.20: Layout of the current divisor for OTAs and level shifters

Figure 4.21: Bias currents of the OTAs and the level shifters

The bias current present an evident drift with the supply voltage of the OTA. However it remains completely negligible as the maximum relative error is less than 0.7%. In term of current ratios, the divisor is plotted in Figure 4.22.
4.4. Current Mirrors

Figure 4.22: Ratios between the current $I_{bias_{OTA,LS}}$ supplied from the pin and the bias currents of OTAs and level shifters respectively

4.4.2 Current Mirrors for SCL Gates

The topology of the current divisor for SCL gates, depicted in Figure 4.23, is very similar to the one used for OTAs and level shifters. The main difference is that outputs are reference currents, $I_{bias_{SCL}} < 0 : 15 >$, and not bias voltages. As previously, the NMOS current mirror makes a division by 40 and the PMOS one divides the current by 25 in such a way that the global ratio is $1/1000$. 
4.4. Current Mirrors

Figure 4.23: Schematic of the current divisor for the SCL gates

The corresponding layout\textsuperscript{4} is depicted in the Figure 4.24 and exhibits one part for the NMOS current mirror and another one for the PMOS current mirror. As previously, the input current $I_{\text{bias}_{\text{SCL}}}$ comes from the upper side while the supply voltage and the ground arrives in the middle of the layout. The area of the current divisor with the direct connections represents about 2380$\mu$m$^2$.

Figure 4.24: Layout of the current divisor for the supply current of SCL gates

The simulation with Virtuoso® Spectre® circuit simulator of the layout is realized for all the full range 1$\mu$A - 200$\mu$A and for diverse supply voltages of the

\textsuperscript{4}The dimensions used are the ones indicated in the Figure 4.23
4.4. Current Mirrors

core (0.5V-1.2V). The measured currents are not the output currents, but come from an external SCL gate (exactly like an av_extracted gate from the multiplier). The circuit is tested with a supply voltage $V_{dd}=1.8V$ for the current mirror. The corresponding bias current of a SCL gate is plotted in Figure 4.25. Most of the obtained results have a relative error smaller than 2% for the full range of input currents (see Figure 4.26). However, when the supply voltage of the SCL gates is lower than 600mV, the current mirror is not able to provide big currents in such a way that the relative error grows dramatically (up to 20% for $I_{bias_{SCLs}} = 200\mu A@V_{dd_{SCL}} = 0.5$). Nevertheless, this error happens for supply conditions that are in the green region of Figure 4.18. So this large relative error, will be completely hidden by the voltage swing control limits.

When the input current is closed to $1\mu A$, the regime of the transistors can leave the strong inversion to tend towards the weak inversion. Even if it is not a complete change, this change of regime imply that the ratio between the transconductance and the bias current $\frac{gm}{I_{bias_{SCL}}}$ will increase. This means the global ratio won’t be 1/1000, but tends towards 1/985, as shown in Figure 4.26. Nevertheless this effect remains controlled (<2%) for all the input range.
Figure 4.25: Bias Current of the SCL gate
Figure 4.26: Ratios between the current $I_{biasSCL}$ supplied from the pin and the bias current of a SCL gate.
Chapter 5

SCL Circuit Characterization

All the blocks presented previously are now associated to the carry-save multiplier. The whole SCL Layout is shown in Figure 5.1. The total area is about 46200µm². The SCL multiplier has to be re-simulated with all the peripheral elements. The most significant simulations concern the maximum frequency achievable and the current consumption.

5.1 Frequency vs Bias Current

As explained in 2.1, the speed of an SCL gate is almost proportional to the value of the tail current $I_{\text{BIAS}}$. Transient simulations have been run in order to determine the maximum frequency achievable for the values of current of the desired range $I_{\text{BIAS}} \in (1nA; 200nA)$. The results are shown in Table 5.1.

In Figure 5.2 we can see some eye diagrams of the output signal still not processed by the level shifters, for 2 values of tail current. In each figure the results for different values of supply voltage $Vdd \in (0.5V, 0.75V, 1V; 1, 25V)$ are presented. Notice that the supply voltage $Vdd$ does not affect the speed of operation.

However, the $Vdd$ value becomes significant in the situation shown in Figure 5.3. With a tail current of 100$nA$, at a frequency of 1$MHz$, the multiplier is not operational for the case $Vdd = 500mV$. This is due to the fact that to drive a current of 100$nA$, the necessary $vbp$ value is higher than $Vdd$ (about 550$mV$).

After being processed by the output level shifters, the signal becomes single ended with a swing equal to the desired supply voltage (see Figure 4.6). In figure 5.4, the eye diagrams of the output signal are shown. Notice that this delay (4.2.2) does not affect the output waveforms in case of low-frequency signal (5.4,a and 5.4,b) while at higher frequency its effect is not negligible anymore (5.4,c)
5.1. Frequency vs Bias Current

Figure 5.1: SCL Circuit Layout

Figure 5.2: Eye diagrams of output signal before level shifters.
5.1. Frequency vs Bias Current

<table>
<thead>
<tr>
<th>frequency</th>
<th>Ibias</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10nA</td>
</tr>
<tr>
<td>10kHz</td>
<td>V</td>
</tr>
<tr>
<td>20kHz</td>
<td>V</td>
</tr>
<tr>
<td>50kHz</td>
<td>V</td>
</tr>
<tr>
<td>75kHz</td>
<td>X</td>
</tr>
<tr>
<td>100kHz</td>
<td>X</td>
</tr>
<tr>
<td>200kHz</td>
<td>X</td>
</tr>
<tr>
<td>500kHz</td>
<td>X</td>
</tr>
<tr>
<td>1MHz</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 5.1: Operating regions of the SCL multiplier (V: correct computation; X: not correct computation)

Figure 5.3: Eye diagram of output signal before level shifters: not operational case. ($I_{BIAS} = 100nA$, $f = 10MHz$)
Figure 5.4: Eye diagram of the output signal.

5.2 Current Consumption

Respect to the simulations on the simple SCL multiplier the measured current consumption has increased: this is because of the 32 replica bias circuits which use an SCL gate supplied with the same voltage as the SCL multiplier. The measured current consumption has increased by $\frac{32}{178} = 18\%$. The measurements present peaks which are much higher than in the previous simulations, about 2 orders of magnitude\(^1\).

\(^1\)This effect is currently being analysed
5.2. Current Consumption

Figure 5.5: Current Consumption of the SCL Circuit for $I_{bias} = 1nA$, $V_{dd} = (0.5, 0.75, 1, 1.25)V$
5.2. Current Consumption

Figure 5.6: Current Consumption of the SCL Circuit $I_{bias} = 1nA$ and $I_{bias} = 100nA$
Chapter 6

CMOS Comparative Circuit

6.1 CMOS Multiplier

6.1.1 Design

The CMOS multiplier is realized with a semi a custom approach, using the UMC-180nm standard cells library. In order to perform a fair comparison between the SCL circuit and the CMOS one, it is necessary to ensure that in both cases the multiplications are performed using the same algorithm. That is why the VHDL code for the CMOS multiplier (shown in Appendix D) describes exactly which gates and connections have to be synthesized. This is practically the structural VHDL code of the carry-save multiplier mapped into the library. The layout of the CMOS multiplier (shown in Figure 6.1) has been automatically generated by Cadence SoC Encounter. The resulting area is about $10000 \ \mu m^2$.

6.1.2 Characterization

The comparison between SCL and CMOS performances is not straightforward. For an SCL circuit, as explained in Chapter 5, the speed of the circuit is related to the amount of furnished tail current while the supply voltage (if sufficient to ensure the correct functionality) is not affecting such a performance. On the other hand, the speed of a CMOS circuit is related mainly to the supply voltage. In order to make a fair comparison between the 2 multipliers, and find a working region where the SCL one presents better performances, we have chosen to fix a certain frequency and to calculate the power consumption in the 2 cases; several simulations have been run for different values of frequencies.

For the CMOS multiplier, we have found the supply voltage $V_{dd_{MIN}}$ sufficient to ensure the correct functionality at the chosen frequency, and calculated the value
Figure 6.1: CMOS Multiplier Layout
of the current consumption $I_{dd}$. Some values of such a characterization are shown in Table 6.1, while the transient simulations of the current consumption $I_{dd}$ are shown in Figure 6.2.

### 6.2 CMOS Interface

The limited number of pins of the chip obliges to devote only 16 pins for the data: 8 pins for the input bits and 8 pins for the output ones. This means that we have to use some registers in order to store the input and the output data, before being computed. These registers, as well as their controller and some testing electronics are designed into the cell called CMOS interface. Basically the CMOS interface, whose structure is depicted in Figure 6.3, controls all the inputs and outputs of the chip. So, it is the controller of the CMOS interface which fix the testing sequence.

Let’s have a look inside the structure of the CMOS interface in Figure 6.3. Basically, the interface between the input pins and the input of the multipliers is composed by three registers, InMSBReg, InLSBReg and In16Reg, which are used to buffer the input data. In this way, both multiplicands $A<7:0>$ and $B<7:0>$ are presented simultaneously to the multipliers. The computation is then proceeded in parallel in CMOS and in SCL multiplier. At the output of each multiplier, we have a register which stores the result. We expect to get the result from the CMOS multiplier after one cycle of the main clock ClkxClk. Then a comparison is effectuated between the SCL and the CMOS results. When the SCL computation is finished, i.e. the comparison matches, we stop the counter ClkCNT, which has been incremented during all the SCL computation. At the output of the chip, the result is given byte after byte. One byte for the 8 Most Significant Bits (MSB) of the result, then the 8 Least Significant Bits (LSB) of the result and finally the value of the ClkCNT is read. Moreover, the result of the SCL or of the CMOS multiplier can be selected.

<table>
<thead>
<tr>
<th>$V_{dd}$ [V]</th>
<th>$f$ [Hz]</th>
<th>$I_{dd}$ [A]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1.5 M</td>
<td>7.7u</td>
</tr>
<tr>
<td>0.4</td>
<td>200 k</td>
<td>883n</td>
</tr>
<tr>
<td>0.3</td>
<td>30 k</td>
<td>236n</td>
</tr>
<tr>
<td>0.2</td>
<td>3 k</td>
<td>44n</td>
</tr>
<tr>
<td>0.2</td>
<td>1 k</td>
<td>14n</td>
</tr>
</tbody>
</table>

Table 6.1: Performances of the CMOS carry-save multiplier
Figure 6.2: Current consumption of the CMOS multiplier
6.2. CMOS Interface

Figure 6.3: Structure of the CMOS Interface
Figure 6.4: CMOS Interface Layout
6.3 Testing Sequence

In this section, we will discuss the way to run the chip in order to test the correct functionality of the chip and to extract the speed and the power consumption of the chip. The sequence is governed by the Finite-State Machine illustrated in Figure 6.5. The strobe signal STxSI allows to move step by step. If the strobe signal is always at high level, the FSM will change state at each clock cycle. Note that the transitions S2-> S3 and S3->S4 are automatic. Therefore the system remains during only one clock cycle in each state S2 and S3. For the other states, the transition depends on the value of the strobe signal STxSI. If the signal STxSI is always high, the FSM will jump to the next state on each rising edge of the main clock ClkxCl. Then it is possible to remain in the state S0, S1, S4, S5 or S6 if the strobe signal returns to its idle value '0'.

1. State S0: Initial and idle state, reached by resetting (RstxDI='0')
2. Transition S0->S1: Store multiplicand A<7:0>
3. State S1: Enable to Store multiplicand B<7:0>
4. Transition S1->S2: Store multiplicand B<7:0>
5. Transition S2->S3: Start Multiplication
6. Transition S3->S4: Start comparison between SCL and CMOS results
7. State S4: Write Output Data : Multiplication Result <15:8>
8. State S5: Write Output Data : Multiplication Result <7:0>
9. State S6: Write Output Data : ClkCNT<7:0>

Once the simulation has started, if the CMOS and the SCL results match, the signal EndSCLxSO becomes active. This allows to measure the timing characteristics of the diverse paths. Using the correct input vectors we can sensitize each output path to measure its delay [7].
6.3. Testing Sequence

```
Defaults:
StatexDN  => StatexDP;
RDxSN     => '0';
InMSBRegxDN  => InMSBRegxDP;
InLSBRegxDN  => InLSBRegxDP;
In16RegxDN  => In16RegxDP;
ClkCNTsttxtxSN  => ClkCNTsttxtxSP;
CompEnxSN  => CompEnxSP;
DataOutRegxDN  => (others=>'0');
OutSCLMux31xD  => OutSCLBfuxDP (15 downto 8);
OutCMOSMux31xD  => OutCMOSRegxDP (15 downto 8);

STxS1='1'
  InMSBRegxDN  => DataInxDI;
  RDxSN       => '1';

STxS1='1'
  InLSBRegxDN  => DataInxDI;
  RDxSN       => '1';

In16RegxDN (15 downto 8) => InMSBRegxDP;
In16RegxDN (7 downto 0)  => InLSBRegxDP;
ClkCNTsttxtxSN

CompEnxSN  => '1';

OutSCLMux31xD  => OutSCLBfuxDP (15 downto 8);
OutCMOSMux31xD  => OutCMOSRegxDP (15 downto 8);

STxS1='1'
  DataOutRegxDN  => DataOutxDI;
  RDxSN         => '1';

OutSCLMux31xD  => OutSCLBfuxDP (7 downto 0);
OutCMOSMux31xD  => OutCMOSRegxDP (7 downto 0);
```

Figure 6.5: Finite State Machine of the control of the CMOS Interface
Chapter 7

SCL-CMOS Comparison

The effort to minimize the Power Delay Product (Eq. 2.5) has been done for one SCL gate. The realization of the 8x8 Carry-Save Multiplier of Chapter 4.1 has been done with a direct implementation of the basic gates. This has resulted in a circuit which is still not competitive with the CMOS technology. Without any change of the general architecture of the multiplier, we will discuss in this section the way to make the SCL technique more competitive. Firstly we want to re-express some properties of the SCL topology, but this time regarding more than only one gate.

7.1 Power-Delay Product Comparison

Let’s consider our circuit as composed by a linear chain of N identical gates, all with identical load capacitance $C_L$ on each output node. The total propagation delay of the chain of gates can be expressed as [2]:

$$P_{SCL} = N \cdot V_{ddSCL} \cdot I_{bias}$$

With the assumptions made in 2.1 we can write the expressions for power:

$$D_{SCL} = \frac{N \cdot C_L \cdot V_{SWING}}{I_{bias}}$$

From which we can directly extract the power-delay expression:

$$PD_{SCL} = N^2 \cdot V_{ddSCL} \cdot V_{SWING} \cdot C_L$$

(7.1)

If we consider the same circuit, but composed by CMOS gates we get:

$$D_{CMOS} = \frac{N^2 C_L V_{ddCMOS}}{2(V_{ddCMOS}-V_T)^2}$$

(7.2)
\[ P_{CMOS} = N \cdot C_L \cdot Vdd_{CMOS}^2 \cdot \frac{1}{D_{CMOS}} \]

So the power-delay becomes:

\[ PD_{CMOS} = N \cdot C_L \cdot Vdd_{CMOS}^2 \]  \hspace{1cm} (7.3)

where \( k \) and \( \alpha \) are process and transistor size dependent parameters.

The most important conclusion from the equations (7.1) and (7.3) comes from the effect of logic depth, \( N \). The performance of SCL gates in relation to CMOS decreases linearly with \( N \). This is due to the fact that SCL consumes static power, even when not switching.

In conclusion in order to minimize the power-delay product in SCL circuits it is very important to maintain a shallow logic depth. This is not the case of the multiplier, where in fact the SCL implementation does not result competitive with the CMOS one.

### 7.2 Further Improvements

#### 7.2.1 Merging of SCL Gates

In the SCL circuits the value of power consumption is proportional to the number of gates to be constantly biased with a tail current. This means that as we increase the logical function of each gate, decreasing their total number \( N \), the current supply needed will be reduced proportionally with \( N \). The main drawback concerns mainly the value of the minimum supply voltage: for a higher number of differential-pair levels, an higher supply voltage is needed. However this increase is not so high as the nMOS network transistors are in weak inversion and their voltage drop is minimum.

In this case the simulations made on the SCL multiplier show an evident improvement of the performances. Referring to the Carry-Save Multiplier shown in Figure 4.3, the idea is to merge each AND gate with the below Full-Adder (see Figure 7.1). This results in a 4 levels gate implementing the AND + XOR3 function, whose schematic is shown in Figure 7.2.

With this kind of configuration we also obtain a significant reduction of the total area and of the routing complexity. Not last, the frequency of operation can be slightly improved thanks to the reduction of the internal glitches.
7.2. Further Improvements

Figure 7.1: Merging of SCL gates

Figure 7.2: SCL gate performing \((A \cdot B) \oplus Ci \oplus D\)
7.2. Further Improvements

![Diagram of SCL gate with different driving capabilities](image)

Figure 7.3: SCL gate with different driving capabilities (N=3)

### 7.2.2 Design of SCL Gates for Different Driving Capabilities

A full library with SCL gates for different driving strengths is currently being designed. One of the possibilities is to use a unique structure, as presented in Chapter 3. In order to give the ability to drive different loads, \( N^1 \) different bias voltages \( vbp \) and \( vbn \) are provided. In this way \( N \) voltage swing control circuits will be needed. The price in area is almost null, as to associate a gate to its driving capability it is be sufficient to connect it to the correct \( vbp \) and \( vbn \) lines, as shown in Figure 7.3. This would be an SCL possibility which is clearly impossible in CMOS technology.

---

\(^1\)\( N \) is the number of required driving strengths.
Chapter 8

Conclusions

The simulation results obtained at the moment show clearly that in the case of the carry-save multiplier the SCL technique is not competitive with the CMOS one. Other simulations are currently being performed to find a working region where the two technologies might show similar performances. The solution proposed in 7.2.1 shows a consistent improvement respect to the classic SCL multiplier in terms of power consumption, but the power-delay product is, in the best case, almost double respect to the CMOS case. As discussed in 7.1, we experienced the need to keep a shallow logic depth.

However, several interesting features and advantages respect to the CMOS technology have been verified. For example the static power consumption, that is almost independent from the switching activity. The SCL multiplier generates a much lower switching noise respect to the CMOS one. In fact the constant current supply needed by SCL avoids the typical current spikes of CMOS logic that determine large variations on the supply voltage, which can couple with the analog circuits on the same substrate and degrade their resolution.

As seen in 2.2.1, one of most interesting features verified for the SCL circuit is the speed tunability. In fact a variation of the furnished current (and so of the power consumption) determines an almost linear variation of the maximum frequency achievable. In this way we would be allowed to tune the desired frequency without any modification in the layout.
Figure 8.1: Layout of the test circuit
Bibliography


Appendix A

Speed Limitations: Study of $R_{LOAD}$ and $C_L$ Parameters

The maximum operating frequency is limited by the pole at the output of the differential pair. This pole is characterized by a resistance (the pMOS load), and a capacitance. For design simplification, the pMOS load is approximated by a linear resistance $R_{MOD}$ defined as:

$$R_{MOD} = \frac{\Delta V}{I_b} = \frac{0.2}{I_b}$$

So $R_{swing}$ varies according to the tail current. From this model, we get the time constant of the output node:

$$\tau = R_{MOD}C_{out}$$

From which we can directly estimate the corresponding rising time:

$$t_{RISE,MOD} = \ln(9) \cdot \tau = \ln(9) \cdot R_{MOD}C_{out}$$

On the other hand, we can extract the equivalent resistance $R_{SIM}$ from the simulated values of rise time:

$$R_{SIM} = t_{RISE,SIM}/(\ln(9) \cdot C)$$

Comparing $R_{MOD}$ with $R_{SIM}$ we get the correction factor $\alpha = R_{SIM}/R_{MOD}$ which takes into account the first-order variation of the pMOS resistance. Actually the factor $\alpha$ has a drift (due to the non-linearity versus $I_b$) in function of the tail current, as depicted in the following figure A.1.

To reduce the approximation error, the average value $\alpha_{AV}$ is taken for finding
\[ R_{SIM} = R_{MOD} \cdot \alpha_{AV}. \]

Thanks to this linear resistance we can straightforwardly draw the estimated rising time, which gives a closed result to simulation \( t_{RISE,SIM} = \ln(9) \cdot R_{SIM}C_{out} \) (see Figure A.2).

The remaining error (<15%) on the rising time due to the approximation of the pMOS resistance by \( R_{SIM} \) is drawn in Figure A.3.

The capacitance \( C \) at the output pole is composed by the wire capacitance \( C_w \), the fan-out of the next input stage \( C_i \), and the intrinsic parasitic capacitances at the pole. This total capacitor is linear (in the expected frequency range of the circuit).

\[ C_{POLE} = C_{INTRINSIC} + C_{OUTPUT} \]

where

\[ C_{INTRINSIC} = C_{gd,pMOS} + C_{bs,pMOS} + C_{nwell} + C_{gd,nMOS} + C_{bd,nMOS} \]

\[ C_{OUTPUT} = C_W + C_L \]

Where the capacitances are defined on the pseudo-small signal schematic of the output node depicted in Figure A.4.

Intrinsic parasitic capacitances of MOSFETs:
\[ C_{pMOS} = C_{gd} + C_{bs} \simeq C_{gdo}W = 5.64e - 17F \]
\[ C_{nMOS} = C_{gd} + C_{bd} \simeq C_{gdo}W = 9.4e - 17F \]
Figure A.2: $T_{RISE, SIM}$ VS $T_{RISE, MOD}$

Figure A.3: Relative error of $T_{RISE}$ value
n-Well capacitance:
\[ C_{nw} \simeq 4.5 \, fF \text{ (in parallel with } dionw_{mm}) \]

Next stage capacitance \( C_l \) (\( C_{in} \) of an inverter gate):
\[ C_{ox} = \varepsilon_{ox}/t_{ox} = 3.9\varepsilon_o/4.2e - 09 = 3.9 \times 8.854e - 12/4.2e - 09 = 8.22e - 03 \, [F/m] \]

Gate Capacitance: \( C_g = C_{ox} W L = 5.92e - 16 \, F \)
Gate-source overlap capacitance: \( C_{gs} = C_{gso} W = 2.35e - 10 \times e - 9 = 9.4e - 17 \, F \)
Gate-drain overlap capacitance: \( C_{gd} = C_{gdo} W = 9.4e - 17 \, F \)
Gate-bulk overlap capacitance: \( C_{yb} = C_{gbo} L \simeq 0 \, F \)

\[ C_l = C_{gate} + C_{gs} + 2C_{gdo} = 0.87 \, fF \text{ (where factor 2 is due to Miller effect)} \]
Appendix B

Layouts of Basic Cells used in the Carry-Save Multiplier
Figure B.1: 2-input AND

Figure B.2: 2-input XOR
Appendix C

UMC 0.180\mu m Process Parameters

<table>
<thead>
<tr>
<th>nMOS transistor</th>
<th>pMOS transistor</th>
<th>Low-Vt transistor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{ox}$</td>
<td>4.20E-09 [m]</td>
<td>$t_{ox}$ 4.20E-09 [m]</td>
</tr>
<tr>
<td>$C_{ox}$</td>
<td>8.22E-03 [F/m$^2$]</td>
<td>$C_{ox}$ 8.22E-03 [F/m$^2$]</td>
</tr>
<tr>
<td>$V_{T0}$</td>
<td>0.31 [V]</td>
<td>$V_{T0}$ -0.46 [V]</td>
</tr>
<tr>
<td>$\mu_o$</td>
<td>314.1 [cm$^2$/Vs]</td>
<td>$\mu_o$ 114.5 [cm$^2$/Vs]</td>
</tr>
<tr>
<td>$n$</td>
<td>1.27 [1]</td>
<td>$n$ 1.54 [1]</td>
</tr>
<tr>
<td>$U_T$</td>
<td>2.60E-02 [V]</td>
<td>$U_T$ 2.60E-02 [V]</td>
</tr>
</tbody>
</table>

Where:

Thermodynamic voltage $U_T$: $U_T = \frac{kT}{q} \approx 26mV$

Transfer parameter $\beta$: $\beta = \mu_0 C_{ox}\frac{W}{L} = K_p\frac{W}{L}$

Inversion factor $I_C$: $I_C = \frac{I_D}{2n\beta U_T^2}$

Pinch-off voltage $V_P$: $V_P \approx \frac{V_T - V_{th}}{n}$
Appendix D

VHDL Code of the CMOS Multiplier

library IEEE, UMCI8_CORE_ART;

use IEEE.std_logic_1164.all;
use UMCI8_CORE_ART.umci8_VCOMPONENTS.all;

entity mult2 is

port( A, B : in std_logic_vector (7 downto 0); MULT_OUT : out
     std_logic_vector (15 downto 0));

end mult2;

architecture SYN_SYN Behav of mult2 is

signal ABx0xx1x, ABx1xx0x, Cx1x, ABx0xx2x, ABx1xx1x, Sx2x, Cx2x, ABx0xx3x, ABx1xx2x, Sx3x, Cx3x, ABx0xx4x, ABx1xx3x, Sx4x, Cx4x, ABx0xx5x, ABx1xx4x, Sx5x, Cx5x, ABx0xx6x, ABx1xx5x, Sx6x, Cx6x, ABx0xx7x, ABx1xx6x, Sx7x, Cx7x, ABx2xx0x, Cx8x, ABx2xx1x, Sx9x, Cx9x, ABx2xx2x, Sx10x, Cx10x, ABx2xx3x, Sx11x, Cx11x, ABx2xx4x, Sx12x, Cx12x, ABx2xx5x, Sx13x, Cx13x, ABx2xx6x, ABx1xx7x, Sx14x, Cx14x, ABx3xx0x, Cx15x, ABx3xx1x, Sx16x, Cx16x, ABx3xx2x, Sx17x, Cx17x, ABx3xx3x, Sx18x, Cx18x, ABx3xx4x, Sx19x, Cx19x, ABx3xx5x, Sx20x, Cx20x, ABx3xx6x, ABx2xx7x, Sx21x, Cx21x, ABx4xx0x, Cx22x, ABx4xx1x, Sx23x, Cx23x, ABx4xx2x, Sx24x, Cx24x, ABx4xx3x, Sx25x, Cx25x, ABx4xx4x, Sx26x, Cx26x, ABx4xx5x, Sx27x, Cx27x, ABx4xx6x, ABx3xx7x, Sx28x, Cx28x, ABx5xx0x, Cx29x, ABx5xx1x, Sx30x, Cx30x, ABx5xx2x, Sx31x, Cx31x, ABx5xx3x, Sx32x, Cx32x, ABx5xx4x, Sx33x, Cx33x, ABx5xx5x, Sx34x, Cx34x, ABx5xx6x, ABx4xx7x, Sx35x, Cx35x, ABx6xx0x, Cx36x, ABx6xx1x, Sx37x, Cx37x, ABx6xx2x, Sx38x, Cx38x, ABx6xx3x, Sx39x, Cx39x, ABx6xx4x, Sx40x, Cx40x, ABx6xx5x, Sx41x, Cx41x, ABx6xx6x, ABx5xx7x, Sx42x, Cx42x, ABx7xx0x, Cx43x, ABx7xx1x, Sx44x, Cx44x, ABx7xx2x, Sx45x, Cx45x, ABx7xx3x, Sx46x, Cx46x, ABx7xx4x,
Sx47x, Cx47x, ABx7xx5x, Sx48x, Cx48x, ABx7xx6x, ABx6xx7x, Sx49x, Cx49x, Cx50x, Cx51x, Cx52x, Cx53x, Cx54x, Cx55x, ABx7xx7x : std_logic;

begin

HA1 : ADDHX4 port map( A => ABx0xx1x, B => ABx1xx0x, S => MULT_OUT(1), CO => Cx1x);  
HA2 : ADDHX4 port map( A => ABx0xx2x, B => ABx1xx1x, S => Sx2x, CO => Cx2x);  
HA3 : ADDHX4 port map( A => ABx0xx3x, B => ABx1xx2x, S => Sx3x, CO => Cx3x);  
HA4 : ADDHX4 port map( A => ABx0xx4x, B => ABx1xx3x, S => Sx4x, CO => Cx4x);  
HA5 : ADDHX4 port map( A => ABx0xx5x, B => ABx1xx4x, S => Sx5x, CO => Cx5x);  
HA6 : ADDHX4 port map( A => ABx0xx6x, B => ABx1xx5x, S => Sx6x, CO => Cx6x);  
HA7 : ADDHX4 port map( A => ABx0xx7x, B => ABx1xx6x, S => Sx7x, CO => Cx7x);  
FA8 : ADDHX4 port map( A => ABx2xx0x, B => Sx2x, CI => Cx1x, S => MULT_OUT(2), CO => Cx8x);  
FA9 : ADDHX4 port map( A => ABx2xx1x, B => Sx3x, CI => Cx2x, S => Sx9x, CO => Cx9x);  
FA10 : ADDHX4 port map( A => ABx2xx2x, B => Sx4x, CI => Cx3x, S => Sx10x, CO => Cx10x);  
FA11 : ADDHX4 port map( A => ABx2xx3x, B => Sx5x, CI => Cx4x, S => Sx11x, CO => Cx11x);  
FA12 : ADDHX4 port map( A => ABx2xx4x, B => Sx6x, CI => Cx5x, S => Sx12x, CO => Cx12x);  
FA13 : ADDHX4 port map( A => ABx2xx5x, B => Sx7x, CI => Cx6x, S => Sx13x, CO => Cx13x);  
FA14 : ADDHX4 port map( A => ABx2xx6x, B => ABx1xx7x, CI => Cx7x, S => Sx14x, CO => Cx14x);  
FA15 : ADDHX4 port map( A => ABx3xx0x, B => Sx9x, CI => Cx8x, S => MULT_OUT(3), CO => Cx15x);  
FA16 : ADDHX4 port map( A => ABx3xx1x, B => Sx10x, CI => Cx9x, S => Sx16x, CO => Cx16x);  
FA17 : ADDHX4 port map( A => ABx3xx2x, B => Sx11x, CI => Cx10x, S => Sx17x, CO => Cx17x);  
FA18 : ADDHX4 port map( A => ABx3xx3x, B => Sx12x, CI => Cx11x, S => Sx18x, CO => Cx18x);  
FA19 : ADDHX4 port map( A => ABx3xx4x, B => Sx13x, CI => Cx12x, S => Sx19x, CO => Cx19x);  
FA20 : ADDHX4 port map( A => ABx3xx5x, B => Sx14x, CI => Cx13x, S => Sx20x, CO => Cx20x);  
FA21 : ADDHX4 port map( A => ABx3xx6x, B => ABx2xx7x, CI => Cx14x, S => Sx21x, CO => Cx21x);  
FA22 : ADDHX4 port map( A => ABx4xx0x, B => Sx16x, CI => Cx15x, S => MULT_OUT(4), CO => Cx22x);  
FA23 : ADDHX4 port map( A => ABx4xx1x, B => Sx17x, CI => Cx16x, S => Sx23x, CO => Cx23x);
FA24 : ADDFH4 port map( A => ABx4xx2x, B => Sx18x, CI => Cx17x, S => Sx24x, CO => Cx24x);
FA25 : ADDFH4 port map( A => ABx4xx3x, B => Sx19x, CI => Cx18x, S => Sx25x, CO => Cx25x);
FA26 : ADDFH4 port map( A => ABx4xx4x, B => Sx20x, CI => Cx19x, S => Sx26x, CO => Cx26x);
FA27 : ADDFH4 port map( A => ABx4xx5x, B => Sx21x, CI => Cx20x, S => Sx27x, CO => Cx27x);
FA28 : ADDFH4 port map( A => ABx4xx6x, B => ABx3xx7x, CI => Cx21x, S => Sx28x, CO => Cx28x);
FA29 : ADDFH4 port map( A => ABx5xx0x, B => Sx23x, CI => Cx22x, S => MULT_OUT(5), CO => Cx29x);
FA30 : ADDFH4 port map( A => ABx5xx1x, B => Sx24x, CI => Cx23x, S => Sx30x, CO => Cx30x);
FA31 : ADDFH4 port map( A => ABx5xx2x, B => Sx25x, CI => Cx24x, S => Sx31x, CO => Cx31x);
FA32 : ADDFH4 port map( A => ABx5xx3x, B => Sx26x, CI => Cx25x, S => Sx32x, CO => Cx32x);
FA33 : ADDFH4 port map( A => ABx5xx4x, B => Sx27x, CI => Cx26x, S => Sx33x, CO => Cx33x);
FA34 : ADDFH4 port map( A => ABx5xx5x, B => Sx28x, CI => Cx27x, S => Sx34x, CO => Cx34x);
FA35 : ADDFH4 port map( A => ABx5xx6x, B => ABx4xx7x, CI => Cx28x, S => Sx35x, CO => Cx35x);
FA36 : ADDFH4 port map( A => ABx6xx0x, B => Sx30x, CI => Cx29x, S => MULT_OUT(6), CO => Cx36x);
FA37 : ADDFH4 port map( A => ABx6xx1x, B => Sx31x, CI => Cx30x, S => Sx37x, CO => Cx37x);
FA38 : ADDFH4 port map( A => ABx6xx2x, B => Sx32x, CI => Cx31x, S => Sx38x, CO => Cx38x);
FA39 : ADDFH4 port map( A => ABx6xx3x, B => Sx33x, CI => Cx32x, S => Sx39x, CO => Cx39x);
FA40 : ADDFH4 port map( A => ABx6xx4x, B => Sx34x, CI => Cx33x, S => Sx40x, CO => Cx40x);
FA41 : ADDFH4 port map( A => ABx6xx5x, B => Sx35x, CI => Cx34x, S => Sx41x, CO => Cx41x);
FA42 : ADDFH4 port map( A => ABx6xx6x, B => ABx5xx7x, CI => Cx35x, S => Sx42x, CO => Cx42x);
FA43 : ADDFH4 port map( A => ABx7xx0x, B => Sx37x, CI => Cx36x, S => MULT_OUT(7), CO => Cx43x);
FA44 : ADDFH4 port map( A => ABx7xx1x, B => Sx38x, CI => Cx37x, S => Sx44x, CO => Cx44x);
FA45 : ADDFH4 port map( A => ABx7xx2x, B => Sx39x, CI => Cx38x, S => Sx45x, CO => Cx45x);
FA46 : ADDFH4 port map( A => ABx7xx3x, B => Sx40x, CI => Cx39x, S => Sx46x,
CO => Cx46x);
FA47 : ADDFH4 port map( A => ABx7xx4x, B => Sx41x, CI => Cx40x, S => Sx47x, CO => Cx47x);
FA48 : ADDFH4 port map( A => ABx7xx5x, B => Sx42x, CI => Cx41x, S => Sx48x, CO => Cx48x);
FA49 : ADDFH4 port map( A => ABx7xx6x, B => ABx6xx7x, CI => Cx42x, S => Sx49x, CO => Cx49x);
HA50 : ADDFH4 port map( A => Sx44x, B => Cx43x, S => MULT_OUT(8), CO => Cx50x);
FA51 : ADDFH4 port map( A => Sx45x, B => Cx50x, CI => Cx44x, S => MULT_OUT(9), CO => Cx51x);
FA52 : ADDFH4 port map( A => Sx46x, B => Cx51x, CI => Cx45x, S => MULT_OUT(10), CO => Cx52x);
FA53 : ADDFH4 port map( A => Sx47x, B => Cx52x, CI => Cx46x, S => MULT_OUT(11), CO => Cx53x);
FA54 : ADDFH4 port map( A => Sx48x, B => Cx53x, CI => Cx47x, S => MULT_OUT(12), CO => Cx54x);
FA55 : ADDFH4 port map( A => Sx49x, B => Cx54x, CI => Cx48x, S => MULT_OUT(13), CO => Cx55x);
FA56 : ADDFH4 port map( A => ABx7xx7x, B => Cx55x, CI => Cx49x, S => MULT_OUT(14), CO => MULT_OUT(15));
C69 : AND2X4 port map( A => A(0), B => B(0), Y => MULT_OUT(0));
C70 : AND2X4 port map( A => A(0), B => B(1), Y => ABx0xx1x);
C71 : AND2X4 port map( A => A(0), B => B(2), Y => ABx0xx2x);
C72 : AND2X4 port map( A => A(0), B => B(3), Y => ABx0xx3x);
C73 : AND2X4 port map( A => A(0), B => B(4), Y => ABx0xx4x);
C74 : AND2X4 port map( A => A(0), B => B(5), Y => ABx0xx5x);
C75 : AND2X4 port map( A => A(0), B => B(6), Y => ABx0xx6x);
C76 : AND2X4 port map( A => A(0), B => B(7), Y => ABx0xx7x);
C77 : AND2X4 port map( A => A(1), B => B(0), Y => ABx1xx0x);
C78 : AND2X4 port map( A => A(1), B => B(1), Y => ABx1xx1x);
C79 : AND2X4 port map( A => A(1), B => B(2), Y => ABx1xx2x);
C80 : AND2X4 port map( A => A(1), B => B(3), Y => ABx1xx3x);
C81 : AND2X4 port map( A => A(1), B => B(4), Y => ABx1xx4x);
C82 : AND2X4 port map( A => A(1), B => B(5), Y => ABx1xx5x);
C83 : AND2X4 port map( A => A(1), B => B(6), Y => ABx1xx6x);
C84 : AND2X4 port map( A => A(1), B => B(7), Y => ABx1xx7x);
C85 : AND2X4 port map( A => A(2), B => B(0), Y => ABx2xx0x);
C86 : AND2X4 port map( A => A(2), B => B(1), Y => ABx2xx1x);
C87 : AND2X4 port map( A => A(2), B => B(2), Y => ABx2xx2x);
C88 : AND2X4 port map( A => A(2), B => B(3), Y => ABx2xx3x);
C89 : AND2X4 port map( A => A(2), B => B(4), Y => ABx2xx4x);
C90 : AND2X4 port map( A => A(2), B => B(5), Y => ABx2xx5x);
C91 : AND2X4 port map( A => A(2), B => B(6), Y => ABx2xx6x);
C92 : AND2X4 port map( A => A(2), B => B(7), Y => ABx2xx7x);
C93 : AND2X4 port map( A => A(3), B => B(0), Y => ABx3xx0x);
C94 : AND2X4 port map( A => A(3), B => B(1), Y => ABx3xx1x);
C95 : AND2X4 port map( A => A(3), B => B(2), Y => ABx3xx2x);
C96 : AND2X4 port map( A => A(3), B => B(3), Y => ABx3xx3x);
C97 : AND2X4 port map( A => A(3), B => B(4), Y => ABx3xx4x);
C98 : AND2X4 port map( A => A(3), B => B(5), Y => ABx3xx5x);
C99 : AND2X4 port map( A => A(3), B => B(6), Y => ABx3xx6x);
C100: AND2X4 port map( A => A(3), B => B(7), Y => ABx3xx7x);
C101: AND2X4 port map( A => A(4), B => B(0), Y => ABx4xx0x);
C102: AND2X4 port map( A => A(4), B => B(1), Y => ABx4xx1x);
C103: AND2X4 port map( A => A(4), B => B(2), Y => ABx4xx2x);
C104: AND2X4 port map( A => A(4), B => B(3), Y => ABx4xx3x);
C105: AND2X4 port map( A => A(4), B => B(4), Y => ABx4xx4x);
C106: AND2X4 port map( A => A(4), B => B(5), Y => ABx4xx5x);
C107: AND2X4 port map( A => A(4), B => B(6), Y => ABx4xx6x);
C108: AND2X4 port map( A => A(4), B => B(7), Y => ABx4xx7x);
C109: AND2X4 port map( A => A(5), B => B(0), Y => ABx5xx0x);
C110: AND2X4 port map( A => A(5), B => B(1), Y => ABx5xx1x);
C111: AND2X4 port map( A => A(5), B => B(2), Y => ABx5xx2x);
C112: AND2X4 port map( A => A(5), B => B(3), Y => ABx5xx3x);
C113: AND2X4 port map( A => A(5), B => B(4), Y => ABx5xx4x);
C114: AND2X4 port map( A => A(5), B => B(5), Y => ABx5xx5x);
C115: AND2X4 port map( A => A(5), B => B(6), Y => ABx5xx6x);
C116: AND2X4 port map( A => A(5), B => B(7), Y => ABx5xx7x);
C117: AND2X4 port map( A => A(6), B => B(0), Y => ABx6xx0x);
C118: AND2X4 port map( A => A(6), B => B(1), Y => ABx6xx1x);
C119: AND2X4 port map( A => A(6), B => B(2), Y => ABx6xx2x);
C120: AND2X4 port map( A => A(6), B => B(3), Y => ABx6xx3x);
C121: AND2X4 port map( A => A(6), B => B(4), Y => ABx6xx4x);
C122: AND2X4 port map( A => A(6), B => B(5), Y => ABx6xx5x);
C123: AND2X4 port map( A => A(6), B => B(6), Y => ABx6xx6x);
C124: AND2X4 port map( A => A(6), B => B(7), Y => ABx6xx7x);
C125: AND2X4 port map( A => A(7), B => B(0), Y => ABx7xx0x);
C126: AND2X4 port map( A => A(7), B => B(1), Y => ABx7xx1x);
C127: AND2X4 port map( A => A(7), B => B(2), Y => ABx7xx2x);
C128: AND2X4 port map( A => A(7), B => B(3), Y => ABx7xx3x);
C129: AND2X4 port map( A => A(7), B => B(4), Y => ABx7xx4x);
C130: AND2X4 port map( A => A(7), B => B(5), Y => ABx7xx5x);
C131: AND2X4 port map( A => A(7), B => B(6), Y => ABx7xx6x);
C132: AND2X4 port map( A => A(7), B => B(7), Y => ABx7xx7x);

end SYN_SYN_behav;
Appendix E

VHDL Code of the CMOS Interface

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity CMOS_interface is
port(
    -- Control Signals I/O
    RstxRBI : in std_logic;
    ClkxCl : in std_logic; -- Main Clock
    STxSI : in std_logic; -- Strobe signal
    SelSCLxSI : in std_logic; -- Select CMOS or SCL output
    ExtClkxCl : in std_logic; -- Fast Clock for SCL Time measurement
    RDxSO : out std_logic; -- Ready Signal
    EndSCLxSO : out std_logic; -- SCL computation Ended Flag
    -- Data Signals I/O
    DataInxDIR : in std_logic_vector(7 downto 0); -- Chip’s input data
    InMulxDO : out std_logic_vector(15 downto 0); -- Multipliers’ input data
    OutSCLMulxDI : in std_logic_vector(15 downto 0); -- SCL_Mul’s output data
    OutCMOSMulxDI : in std_logic_vector(15 downto 0); -- CMOS_Mul’s output data
    DataOutxDO : out std_logic_vector(7 downto 0)); -- Chip’s output data
end entity CMOS_interface;
architecture rtl of CMOS_interface is
    -- Internal Data Signals Present State
    signal InMSBRegxD : std_logic_vector(7 downto 0);
    signal InLSBRegxD : std_logic_vector(7 downto 0);
    signal In16RegxD : std_logic_vector(15 downto 0);
    signal OutSCLRgexDP : std_logic_vector(15 downto 0);
    signal OutSCLBfgxDP : std_logic_vector(15 downto 0);
    signal OutCMOSRegxD : std_logic_vector(15 downto 0);
    signal OutCLKRegxD : std_logic_vector(7 downto 0);
    signal ClkCNTxD : unsigned (7 downto 0);
    signal DataOutRegxD : std_logic_vector(7 downto 0);
signal OutSCLMux31xD : std_logic_vector(7 downto 0);
signal OutCMOSMux31xD : std_logic_vector(7 downto 0);
signal DataOutxD : std_logic_vector(7 downto 0);
-- Internal Data Signals Next State
signal InMSBRegxDN : std_logic_vector(7 downto 0);
signal InLSBRegxDN : std_logic_vector(7 downto 0);
signal In16RegxDN : std_logic_vector(15 downto 0);
signal OutSCLRegxDN : std_logic_vector(15 downto 0);
signal OutSCLBufxDN : std_logic_vector(15 downto 0);
signal OutCMOSRegxDN : std_logic_vector(15 downto 0);
signal OutCLKRegxDN : std_logic_vector(7 downto 0);
signal ClkCNTxnDN : unsigned (7 downto 0);
signal DataOutRegxDN : std_logic_vector(7 downto 0);
-- Internal Helper Signals
signal CompEnxS : std_logic;
signal CompENxS : std_logic;
signal EndSCLxS : std_logic;
signal ClkCNTstartxS : std_logic;
signal ClkCNTENxS : std_logic;
-- Internal Control Signal
signal CompENxSP, CompENxSN : std_logic;
signal ClkCNTstartxSP, ClkCNTstartxSN : std_logic;
-- Output Signals
signal RDxSP, RDxSN : std_logic;
signal EndSCLxSP, EndSCLxSN : std_logic;
-- FSM States
type FSM_states is (S0, S1, S2, S3, S4, S5, S6);
signal StatexDp, StatexDN : FSM_states;
begin
-- set the outputs
InMuxxD <= In16RegxDP;
RDxSO <= RDxSP;
EndSCLxSO <= EndSCLxSP;
DataOutxD <= DataOutRegxDP;
-- set the inputs
OutCMOSRegxDN <= OutCMOSMuxDI;
OutSCLRegxDN <= OutSCLMuxDI;
-- FSM
p_FSMcomb : process (StatexDP, RDxSP, InMSBRegxDP, InLSBRegxDP, CompENxSP,
                     ClkCNTstartxSP, OutCMOSRegxDP, OutSCLBufxDP, OutCLKRegxDP, DataOutRegxDP,
                     DataOutxD, STxSI)
begin
StatexDN <= StatexDP;
RDxSN <= '0';
InMSBRegxDN <= InMSBRegxDP;
InLSBRegxDN <= InLSBRegxDP;
In16RegxDN <= In16RegxDP;
ClkCNTstartxSN <= ClkCNTstartxSP;
CompENxSN <= CompENxSP;
DataOutRegxDN <= (others => '0');
OutSCLMux31xD <= OutSCLBufxDP(15 downto 8);
OutCMOSMux31xD <= OutCMOSRegxDP(15 downto 8);
case StatexDP is
  when S0 =>
    if STxSI = '1' then
      InMSBRegxDN <= DataInxDI;
      R DxSN <= '1';
      StatexDN <= S1;
    end if;
  when S1 =>
    if STxSI = '1' then
      InLSBRegxDN <= DataInxDI;
      R DxSN <= '1';
      StatexDN <= S2;
    end if;
  when S2 =>
    In16RegxDN(15 downto 8) <= InMSBRegxDP;
    In16RegxDN(7 downto 0) <= InLSBRegxDP;
    ClkCNTstartxSN <= '1';
    StatexDN <= S3;
  when S3 =>
    CompENxSN <= '1';
    StatexDN <= S4;
  when S4 =>
    OutSCLMux31xD <= OutSCLBufxDP(15 downto 8);
    OutCMOSMux31xD <= OutCMOSRegxDP(15 downto 8);
    if STxSI = '1' then
      DataOutRegxDN <= DataOutxD;
      R DxSN <= '1';
      StatexDN <= S5;
    end if;
  when S5 =>
    OutSCLMux31xD <= OutSCLBufxDP(7 downto 0);
    OutCMOSMux31xD <= OutCMOSRegxDP(7 downto 0);
    if STxSI = '1' then
      DataOutRegxDN <= DataOutxD;
      R DxSN <= '1';
      StatexDN <= S6;
    end if;
  when S6 =>
OutSCLMux31xD <= OutCLKRegxD (7 downto 0);
OutCMOSMux31xD <= OutCLKRegxD (7 downto 0);
when others => StatexDN <= S1;
end case;
end process p_FSMcomb;

--- Main synchronization processes for CLK and EXT_CLK

mainClk : process (RstxBRI, ClkxCI)
begin
if RstxBRI = '0' then
StatexDP <= S0;
RDxSP <= '0';
CompENxSP <= '0';
ClkCNTstartxSP <= '0';
EndSCLxSP <= '0';
InMSBRegxDP <= (others => '0');
InLSBRegxDP <= (others => '0');
In16RegxDP <= (others => '0');
OutSCLBufxDP <= (others => '0');
OutCMOSRegxDP <= (others => '0');
OutCLKRegxD <= (others => '0');
DataOutRegxD <= (others => '0');
elsif ClkxCI'event and ClkxCI = '1' then
StatexDP <= StatexDN;
RDxSP <= RDxSN;
CompENxSP <= CompENxSN;
ClkCNTstartxSP <= ClkCNTstartxSN;
EndSCLxSP <= EndSCLxSN;
InMSBRegxDP <= InMSBRegxDN;
InLSBRegxDP <= InLSBRegxDN;
In16RegxDP <= In16RegxDN;
OutSCLBufxDP <= OutSCLBufxDN;
OutCMOSRegxD <= OutCMOSRegxDN;
OutCLKRegxD <= OutCLKRegxDN;
DataOutRegxD <= DataOutRegxDN;
end if;
end process mainClk;

mainExtClk : process (RstxBRI, ExtClkxCI)
begin
if RstxBRI = '0' then
OutSCLRegxD <= (others => '0');
ClkCNTxDP <= ('others' => '0');
elsif ExtClkxCI'event and ExtClkxCI = '1' then
OutSCLRexxDp <= OutSCLRexxDN;
ClkCNTxDP <= ClkCNTxDN;
end if;
end process mainExtClk;

-- Counter
p_ClkCNT : process(ClkCNTxDP, ClkCNTENxS)
begin
ClkCNTxDN <= ClkCNTxDP;
if ClkCNTENxS = '1' then
ClkCNTxDN <= ClkCNTxDP + 1;
end if;
end process p_ClkCNT;

-- Comparator
CompEqxS <= '1' when std_match(OutSCLRexxDP, OutCMOSRegxDP) else '0';

-- Internal Connections
EndSCLxS <= CompENxSP AND CompEqxS;
EndSCLxSN <= EndSCLxS;
ClkCNTstartxS <= ClkCNTstartxSP;
ClkCNTENxS <= ClkCNTstartxS AND (NOT EndSCLxS);
OutSCLBufxDN <= OutSCLRexxDP;
OutClkRegxDN <= std_logic_vector(ClkCNTxDP);

-- Output Mux
with SelSCLxSI select
DataOutxD <= OutCMOSMux31xD when '0',
OutSCLMux31xD when others;
end architecture rtl;