Ultra-Low-Power Digital Circuit Design

Christoph WALTER

Professor: Yusuf LEBLEBICI

Supervisors: Armin TAJALLI
Alessandro CEVRERO

January 21, 2011
Subthreshold source-coupled logic (STSCL) is a logic family that contains a constant bias current in each logic gate. Logic operation is based on the steering of this current towards one of two load resistors using subthreshold differential pairs. STSCL was proposed for its low power consumption due to the small voltage signal swing, as well its suitability for configurable circuits.

In this work, the feasibility of an ECC processor using subthreshold source-coupled logic was studied. It was shown that the STSCL design flow can be successfully applied to a real-world design. However, comparison with a standard low-voltage CMOS implementation showed that STSCL in its current state is not as energy-efficient as CMOS. Two main reasons for this have been identified:

The most consequential problem in the design of STSCL circuits in deep sub-micron technologies are device mismatches. They make it necessary to use transistors much larger than the minimum size, leading to increased device capacitances as well as interconnect parasitics due to the larger overall area of the circuits. The special load devices used for STSCL further aggravate the problem of large cell areas, since they require two isolated n-wells in the layout of each logic gate.

The other issue that has been identified is the importance of the logic depth of the design. Since the power dissipation in STSCL is directly proportional to the speed of the individual gates, a high logic depth requires each gate to be faster, which leads to poor performance in comparison to CMOS.

Even in its current implementation, STSCL offers some of the expected advantages over CMOS. Most notably, the replica bias circuit used to generate the bias voltages for the current source and load resistors of each gate makes it possible to use the same circuit over a wide range of target frequencies, independent of the supply voltage.

As expected, the current profile of STSCL gates is very flat; this is an important advantage for cryptographic applications where an attacker might try to gain information on the data being processed by studying the supply current.

The performance issues with the present design suggest that certain steps be taken to improve the library and design flow:

- Redesigning a STSCL standard cell library with less conservative device sizing would somewhat reduce the cell area and could help amend the problems of large interconnect capacitance. Further improvements could be made by building a more complete standard library with high fan-in gates.

- More importantly, the activity rate is the factor that appears to carry the largest potential for improvement; therefore, circuit techniques such as one-stage pipelining should be taken into consideration.
# Contents

1 Introduction 6
   1.1 Low-power circuits for RFID 6
   1.2 Low-Power Logic Families 6
      1.2.1 CMOS adapted for low power consumption 6
      1.2.2 Current-mode logic 7
   1.3 Report organization 7

2 Subthreshold Source-Coupled Logic 8
   2.1 Overview 8
   2.2 Description of STSCL Circuits 8
      2.2.1 NMOS Tail Transistor 9
      2.2.2 NMOS Switching Network 9
      2.2.3 PMOS Load Devices 10
      2.2.4 Replica Bias Circuit 10
   2.3 Use of STSCL for cryptographic hardware 10
   2.4 Performance analysis 10
      2.4.1 Gate Delays 10
      2.4.2 Power consumption 11
      2.4.3 Effects of Process Variations and Mismatch 12
      2.4.4 Noise margin analysis 13
   2.5 Design Flow 15
      2.5.1 Synthesis 16
      2.5.2 Placement and Routing 16

3 Elliptic curve cryptographic processor 17
   3.1 Introduction 17
   3.2 Elliptic curve cryptography 17
      3.2.1 Montgomery algorithm for scalar multiplication 19
   3.3 Specifications of the cryptographic processor 19
   3.4 Architecture 19
      3.4.1 ALU 20
      3.4.2 Finite State Machine 21
      3.4.3 Registers 21
   3.5 Standard CMOS implementation 21
   3.6 STSCL implementation 21
      3.6.1 Modifications of the design flow and library 21
Contents

3.6.2 Library Specifications ........................................... 22

4 Results ........................................................................ 23
  4.1 Design and simulation flow ........................................... 23
  4.2 Performance comparison .............................................. 23
  4.3 Interpretation of results ............................................. 24
    4.3.1 Advantages and disadvantages STSCL ..................... 26

5 Outlook ........................................................................ 29
  5.1 Possible improvements in the STSCL library .................... 29
    5.1.1 Device sizing optimization .................................... 29
    5.1.2 Shallow pipelining ............................................. 30

6 Conclusion .................................................................... 31

Bibliography .................................................................... 33

Appendix ......................................................................... 34
1 Introduction

1.1 Low-power circuits for RFID

The practical part of this work was the implementation of a cryptographic processor which could be used in a RFID system for secure authentication.

RFID systems require a unique ID for each tag that is to be read by the system. Unsecured wireless transmission of these IDs makes it relatively easy for an attacker to detect the protocol and read the IDs remotely. The use of asymmetric cryptography makes it possible for a reader to verify the identity of the tag using a challenge-response scheme. In this case, each tag has a public-private key pair, and the reader knows the public key for each tag, which allows any tag to securely prove its identity without exposing the private key.

Asymmetric cryptography for RFID requires a considerable amount of processing on a very tight power budget, which justifies the study of ultra-low-power circuits for this application.

1.2 Low-Power Logic Families

1.2.1 CMOS adapted for low power consumption

CMOS logic families are an obvious choice for low-power logic due to their simplicity and the fact that simply scaling the supply voltage can allow the same circuit to be used under a wide performance range in terms of speed and power consumption. Lowering the supply voltage results in a quadratically lower dynamic energy per operation. At the same time, the speed of operation reduces because of lower gate bias voltages.

The average power dissipation in a CMOS circuit can be separated into static (leakage) power and dynamic (switching) power:

\[ P = P_{\text{leak}} + P_{\text{dynamic}} = V_{DD}I_{\text{leak}} + \alpha V_{DD}^2 C_L f, \]

where \( \alpha \) is the activity rate, \( C_L \) the total load capacitance and \( f \) the clock frequency.

In CMOS circuits that are operating at regular supply voltages (higher than the threshold voltage), dynamic power is the dominant source of power dissipation. When the voltage is reduced and devices enter the subthreshold region, leakage becomes a more important part of the total power.

In this case, the ratio of current flowing through an 'on' transistor and current leaking through an 'off' transistor depends on the supply voltage as well as the subthreshold slope, as described in the expression for subthreshold current:
\[ I_{DS} \approx I_0 \cdot e^{\frac{V_{GS}-V_T}{nU_T}} \]

where \( V_T \) is the threshold voltage and \( n \) the subthreshold slope factor.

In the subthreshold region, both the speed and the power dissipation of CMOS circuits depend strongly on the voltage. For this reason, subthreshold CMOS designs require very precise control over the supply voltage.

### 1.2.2 Current-mode logic

Current-mode logic (CML) is a group of logic styles where operation is based on controlled switching (steering) of a current from one branch of the circuit to another. CML has a long history: Emitter-coupled logic (ECL) using bipolar transistors and resistors was patented in 1956. ECL was used for high-performance bipolar circuits because of the low voltage swing and the fact that the transistors would not enter saturation, leading to very fast gate delays.

More recently, the logically equivalent source-coupled logic using CMOS technology has been implemented for its low power supply noise injection, as well as its high speed.

Subthreshold source-coupled logic \(^{[1, 2, 3]}\) has been proposed as an alternative to CMOS in ultra-low power applications, due to the precise control it offers over speed and power consumption.

### 1.3 Report organization

Chapter 2 first gives an overview of STSCL circuits and then continues with an analysis of the performance in presence of device variations. In Chapter 3, an elliptic curve cryptography processor is introduced which will serve as an example for the top-down design flow using the STSCL library. The results of this implementation will be summarized and commented in Chapter 4. Chapter 5 gives an outlook on possible improvements of the STSCL standard library and design flow. The final Chapter 6 is a short conclusion.
2 Subthreshold Source-Coupled Logic

2.1 Overview

Source-coupled logic, also known under the more general term current-mode logic (CML) is a group of logic families that use transistor differential pairs to switch a constant bias current towards one of two branches representing the two terminals of a differential output signal. In a MOS implementation, an NMOS tail transistor biased with a constant gate voltage acts as a current source that draws a constant current $I_{ss}$ from the supply. Logic operation takes place by steering the tail current to one of the two load devices. This can be achieved by a network of differential pairs controlled by the (differential) gate input voltages.

The output signal is created by two load ‘resistors’ that convert the difference in current in their respective branches into a differential output voltage. The value of these resistors is chosen such that $I_{ss}$ creates a voltage drop of $V_{SW} = I_{SS}R_L$ when the tail current passes through them. One of the output terminals will therefore be at a voltage of $V_L = V_{DD} − V_{SW}$, the other at $V_H = V_{DD}$.

Subthreshold source-coupled logic (STSCL, [1, 2, 3]) is a variant of current-mode logic in which all transistors are operating in the subthreshold region. Since in STSCL, operation is based on switching a subthreshold current between the two branches of the logic gate, the problem of leakage power consumption is virtually nonexistent.\footnote{As long as the currents in the p-n junction formed by the source and bulk of the PMOS load devices are small compared to the bias current.}

2.2 Description of STSCL Circuits

Figure 2.1 shows the STSCL logic style proposed in [3]. It uses a replica bias circuit which allows the tail current to be adjusted over a wide range, enabling circuits that can dynamically adapt to available power and required speed. Operation of the circuits is indifferent to variations in supply voltage, as long as the replica bias circuit is able to generate a large enough bias voltage for the desired tail current. As shown in Figure 2.2, supply voltages as low as 0.3 V are possible for a tail current of 100 pA. Gates with higher driving strength require a slightly higher $V_{DD}$ if device sizes are kept the same, because the gate-to-source voltage of the active differential pair transistor is higher.

One particularity with current-mode logic styles is the fact that they draw a constant current, even when no switching takes place. For good energy-per-operation efficiency, it is therefore important to design circuits with a high activity rate.
2.2.1 NMOS Tail Transistor

A single reference current source $I_{BIAS}$ is required to bias all logic gates of a given driving strength. If $I_{BIAS}$ is implemented using a programmable current mirror, the tail current and therefore the speed of the logic gates can be adjusted dynamically. In the current implementation of the STSCL library in 90 nm technology, the different driving strengths of each logic gate are implemented through the use of different bias voltages. 12 different bias voltage signals have to be routed on the chip to bias the NMOS tail transistors and PMOS loads for six different driving strengths ($x_1 - x_{32}$).

2.2.2 NMOS Switching Network

A network of combined NMOS differential pairs controlled by the differential input voltages steers the current towards one of the two load devices. The differential input voltage needs
Chapter 2. Subthreshold Source-Coupled Logic

to satisfy \( V_{SW} = \Delta V_{in} > 4 \cdot n \cdot U_T \) (\( n \) is the subthreshold slope factor and \( U_T \) the thermal voltage) in order to completely switch the current. In a technology with \( n = 1.5 \), the voltage swing has to be at least 150 mV.

For more complex gates, a systematic approach is needed to identify all the logic functions that can be implemented with a given number of logic stages. A binary decision diagram can be used to systematically generate all possible gate topologies for a given number of inputs [5].

2.2.3 PMOS Load Devices

STSCL gates require a pair of load ‘resistors’ with a high resistivity that can be precisely controlled and that is relatively insensitive to process variations. This can be achieved by using a pair of PMOS transistors biased by a gate voltage \( V_P \), and with their bulk terminal (the n-well tap) tied to the drain. The resulting load device has been shown to have a reasonably linear resistivity and low sensitivity to process variations [3].

2.2.4 Replica Bias Circuit

In order to maintain the desired circuit performance in the presence of PVT (process - voltage - temperature) variations, a feedback loop containing a replica circuit is used to set the gate voltage of the PMOS load devices. This replica circuit consists of a tail transistor with a bulk-drain connected load transistor, both using the same dimensions as their counterparts inside the logic gates. The output voltage of this replica stage is equal to \( V_{DD} - I_{SS}R_L \), the low voltage in a differential output pair. The desired value of \( V_{SW} = I_{SS}R_L \) is fed as an input to a negative-feedback loop which controls the gate voltage \( V_P \) of the load device and therefore its resistance \( R_L \).

2.3 Use of STSCL for cryptographic hardware

Figure 4.4 shows the power supply current waveform for the STSCL implementation of the ECC core presented in Chapter 3. Contrary to CMOS, STSCL exhibits a very flat power profile, with supply current fluctuations of less than 5%. The (partially) symmetric nature of STSCL gates means that the transient current waveform is also much less data-dependent. This reduces the risk of exposing secrets (e.g. the private key) to a side-channel attacker.

2.4 Performance analysis

2.4.1 Gate Delays

The bulk-drain connected load device acts like a resistor with a large-signal resistance \( R = \frac{V_{SW}}{I_{SS}} \) and together with the load capacitance \( C_L \) creates an RC network at the output node.

The differential output of an STSCL gate switches with a time constant given by:
\[ \tau_{STSCL} = R_L \cdot C_L \approx \frac{V_{SW}}{I_{SS}} \cdot C_L, \]

It can be shown that the propagation delay is given by [6]:

\[ t_{d,STSCL} = \ln(2) \cdot \tau_{STSCL} = \ln(2) \cdot \frac{V_{SW}}{I_{SS}} \cdot C_L \]

With \( P = V_{DD}I_{SS} \), the power-delay product (PDP) for each gate can thus be written as

\[ PDP_{STSCL} = \ln(2) \cdot V_{DD} \cdot V_{SW} \cdot C_L \]

It must be noted that, for a given choice of \( V_{DD} \) and \( V_{SW} \), the PDP is proportional to the output capacitance, but independent of \( I_{SS} \). In other words, by scaling \( I_{SS} \), the speed of the circuit can be adjusted while keeping the PDP constant.

### 2.4.2 Power consumption

In a system with a target clock frequency \( f = \frac{1}{T} \) and an average logic depth (number of gates in a register-to-register path) \( d \), the power consumption can be estimated as follows:

For the sake of simplicity, all gates are assumed to have identical load capacitances \( C_L \).

The delay for each gate (assuming equal delays) has to be less than or equal:

\[ t_d = \ln(2) \cdot \frac{V_{SW}}{I_{SS}} \cdot C_L = \frac{T}{d} \]

The required gate bias current is

\[ I_{SS} = \ln(2) \cdot \frac{d}{T} \cdot V_{SW} \cdot C_L = \ln(2) \cdot d \cdot f \cdot V_{SW} \cdot C_L \quad (2.1) \]

In a system with \( N \) gates, the minimum total supply current is

\[ I_{total} = \ln(2) \cdot C_{total} \cdot d \cdot f \cdot V_{SW} \quad (2.2) \]

where \( C_{total} \) is the total single-ended load capacitance in the system. Because (2.1) is linear in \( C_L \), this expression is valid even if \( C_{total} \) is not equally distributed among the gates, if the current in each gate can be adjusted such that \( t_d \) is the same for all gates.

**Comparison to CMOS**  Figure 2.3 shows a comparison of the power dissipation of an STSCL an a CMOS implementation of the finite field multiplier design (Appendix, pg. 34). The STSCL core was run with a clock period of 6\( \mu \)s (red dot); the dashed red line shows the theoretical (linear) power-frequency characteristic of STSCL. In reality, there is a maximum \( I_{SS} \) that depends on \( V_{DD} \) and which sets an upper limit on the operating frequency.

---

The design used for this analysis is a modified version of the ALU contained in the ECC processor of Chapter 3, consisting of the ALU and 3 registers with 163 bits each and the control signals needed for the shift-and-add multiplication algorithm.
It can be seen from the figure that the power dissipation of CMOS circuits is dominated by leakage currents up to frequencies in the order of 1 MHz. At higher speeds, power dissipation increases in a linear fashion with the frequency, up to the maximum possible operating frequency (indicated by the sudden drop in power).

For this design, STSCL uses less power than CMOS with a supply voltage of 0.4 V for frequencies of about 100 kHz and lower.

### 2.4.3 Effects of Process Variations and Mismatch

For a basic analysis of the nominal STSCL performance as well as variability, a buffer (or inverter) is used, since it represents the simplest possible logic gate and the effects of variations are the same for all other gates. Three components can be identified that influence the performance of this gate:

1. The NMOS tail transistor which provides the constant current required for CML operation. Mismatch between the tail transistor in the replica circuit and the one in the actual logic gate can lead to a lower current value in the gate, and therefore a lower output voltage swing. Since the replica bias circuit will be located far away from some of the logic gates, the amount of mismatch can be considerable, but it is hard to estimate during the design phase.

2. The NMOS differential pair (several pairs for more complex gates). The exponential subthreshold conduction law dictates the minimally required input voltage difference for complete current switching. In addition to that, any threshold voltage mismatch between the two differential pair devices will appear as an input offset; it has therefore
to be added to the voltage swing as a margin.

3. The PMOS load transistors. Their large-signal resistance is set by the replica circuit to result in a voltage drop equal to $V_{SW}$. Again, the distance to the replica bias circuit will create significant variation in $V_{SW}$. Mismatch between the two devices translates into an input-referred offset at the gate input.

In order to guarantee correct operation, the gates are required to have a positive noise margin (NM) under the presence of variations.

It can be shown that the NM for an STSCL gate with ideal resistor loads is given by [6]:

$$\frac{NM}{V_{SW}} = \sqrt{1 - \frac{1}{A_V}} - \frac{1}{A_V} \cdot \tanh^{-1}\left(\sqrt{1 - \frac{1}{A_V}}\right) \tag{2.3}$$

where $A_V$ is the DC voltage gain. As long as the load devices are close to ideal resistors, $A_V$ (and therefore $\frac{NM}{V_{SW}}$) is determined by the subthreshold slope factors for the given technology. Considering only nominal performance, the library designer is left with choosing $V_{SW}$ to achieve the desired NM. On the other hand, device variability has an important consequence. If all gates are to work under worst-case mismatch conditions, the output voltage swing has to be overdesigned, therefore requiring a higher bias current. The amount of mismatch can be reduced by using transistors with a large gate area. Standard cell design will therefore be dominated by the trade-off between circuit area and power dissipation.

2.4.4 Noise margin analysis

Under the presence of device variations, the noise margin can be estimated using [6]:

$$NM \approx NM_0 - \left(\frac{\partial NM}{\partial V_{SW}}\right) \cdot \Delta V_{SW} - V_{OS}$$

where $NM_0$ is the nominal noise margin, $\Delta V_{SW}$ is the variation of output low voltage and $V_{OS}$ is the input referred offset of the gate. This expression shows device variations affecting the noise margin on both the input side (offset voltage) and the output side (reduced output differential voltage).

The sensitivity to output swing variations can be estimated using [2.3]:

$$K_{NM} = \frac{\partial NM}{\partial V_{SW}} \approx \sqrt{1 - \frac{1}{A_V}}$$

Noise margin variance becomes:

$$\sigma_{NM}^2 \approx K_{NM}^2 \sigma_{SW}^2 + \sigma_{OS}^2$$

The variances of output voltage swing ($\sigma_{SW}^2$) and input referred offset voltage ($\sigma_{OS}^2$) are both dependent on device dimensions. Assuming that the main source of variability are threshold voltage variations it follows that $\sigma_{SW}^2$ depends mainly on the gate area of the tail and load transistors, whereas $\sigma_{OS}^2$ depends on $V_{TH}$ mismatch between the differential
CHAPTER 2. SUBTHRESHOLD SOURCE-COUPLED LOGIC

Figure 2.4: Dependency of NM variance on tail transistor gate area

\[ \sigma^2_{NM} \approx C_N S_N + C_B S_B + C_P S_P \]

Where \( S_N, S_B, S_P \) are the per-device transistor gate area \((W \times L)\) for the NMOS differential pair, NMOS current source, and PMOS bulk-drain shorted load devices.

Monte Carlo analysis In order to obtain numerical values for the coefficients \( C_N, C_B \) and \( C_P \), Monte Carlo simulations were performed on an inverter gate over a range of different sizes for the three types of transistors.

A MATLAB program (Appendix, pg. [38]) was written to calculate the mean and standard deviation of the noise margin (cf. [7]) for a series of voltage sweeps. Figure 15 shows the output of this program. The values of \( \sigma^2_{NM} \) were calculated for a range of different gate areas for each of the three transistor types. The coefficients \( C_N, C_B \) and \( C_P \) were then found by linear regression.

As an example, Figure 2.4 shows how the area of the NMOS tail transistor affects NM variability. Table 2.1 shows the coefficients for the three types of transistors.

<table>
<thead>
<tr>
<th>coefficient</th>
<th>value ([V^2 \cdot \mu m^2])</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_N )</td>
<td>( 7.4 \times 10^{-6} )</td>
</tr>
<tr>
<td>( C_B )</td>
<td>( 83.5 \times 10^{-6} )</td>
</tr>
<tr>
<td>( C_P )</td>
<td>( 47.5 \times 10^{-6} )</td>
</tr>
</tbody>
</table>

Table 2.1: Relative contributions to NM variance

pair devices as well as between the load devices. The total NM variation can therefore be approximated as a function of device sizes for the three independently sized transistors:

These simulation results are unable to take into account the degree of matching among pairs of transistors. Inside the cell layouts, transistors are placed close together and with a similar environment. It can therefore be expected that the actual matching is better than
When designing a standard cell library in STSCL, the total budget for noise margin variations should be distributed among these three types of devices while keeping the total area at a minimum. It should be noted, however, that the gate area of the differential pair transistors, and, to a lesser extent, the n-well of the load devices’ bulk-drain terminal constitute parasitic capacitances for the gate. Therefore, the differential pair and load transistors should be made somewhat smaller for fast operation and the tail transistor larger to keep $\sigma_{NM}^2$ at an acceptable level.

### 2.5 Design Flow

The novel characteristics of current-mode logic families requires modified tools for top-down design of digital circuits. A major difficulty with existing design software is their inability to route differential signals. (Cadence Encounter has support for routing differential pairs, but this only applies to pre-defined pairs of nets, so no optimization is possible). [8] introduced a series of custom programs that use standard synthesis and Place and Route tools for a MCML design flow. Their particularity lies in the use of two versions of the standard cell library, one differential and one single-ended. The differential library represents the physical circuits; each input or output consists of a pair of signals. In the single-ended library (also called ‘fat’ library), logic gates are represented as if they were single-ended, meaning a single pin is used to represent one signal.

In this work, a design flow based on the scripts presented in [4] is being used. The scripts had to be adapted for compatibility with more recent EDA tools, and functionality for routing of the bias voltages as well as clock tree synthesis have been added.

In a first step, the design is synthesized, placed, and routed using the single-ended library. This allows the designer to make full use of existing methods for power and timing...
optimization, clock tree synthesis, and so on. During this step, wide wires are used to route signals.

As a second step, a custom software tool processes the design, replacing the single-ended gates with their physical (differential) counterparts, and splitting signal wires into differential wire pairs.

2.5.1 Synthesis

Synthesis of the RTL code is straightforward using an existing logic synthesis tool (Synopsys Design Compiler). It must be noted, however, that one main difference between CMOS and STSCL may cause the tool to produce a non-optimum netlist: In CMOS, the driving strength of a gate is roughly proportional to its input capacitance (for a given logic function), whereas in the STSCL library, different driving strengths use the same device sizes with different bias voltages.

2.5.2 Placement and Routing

For the first P&R iteration, the design is loaded using a single-ended setup. The structural Verilog netlist generated by Design Compiler is used together with the ‘fat’ libraries, that is, the LEF files for the standard library and routing wires. The design is then placed and routed using a standard flow.

The routing information for the V_P and V_N bias voltages is then temporarily saved to a DEF file. This allows the bias routing to be done at an earlier stage, keeping the amount of routing required after wire splitting at a minimum.

Wire splitting is performed by exporting the cell placement and signal routing to a DEF file, which can then be processed by the custom-made [8 5] split_wires tool. This tool replaces all wire shapes (rectangles) by two narrower rectangles and updates all cell instances to their respective differential versions. split_wires outputs again a DEF file which can then be read back into Encounter to continue with the P&R flow.

Mechanically replacing all wires by differential pairs obviously creates numerous design rule violations. Therefore, the routing command has to be called once again to fix these violations and to complete the routing in places where shapes had previously been deleted.

In order to keep the energy consumption low, a special option (leakage power optimization) was used in place and route. Using this option enables a final pass where gates are replaced by alternative implementations that have lower leakage current, which in STSCL translates to lower overall power.
3 Elliptic curve cryptographic processor

3.1 Introduction

For demonstration purposes, an existing design for an elliptic curve cryptographic core
(provided by [9]) was implemented in both a standard CMOS library as well as STSCL.
Both were synthesized starting from the same VHDL register-transfer-level (RTL) code.
The two implementations were then compared in terms of area, power consumption and
supply current profile.

3.2 Elliptic curve cryptography

Public-key cryptographic systems rely on computationally ‘hard’ mathematical problems.
Traditionally, public-key systems used the fact that it is computationally expensive to
factor large integers. Elliptic curve cryptography instead relies on the discrete logarithm
problem on elliptic curves.

Recently, elliptic curve cryptography has been proposed as a viable encryption/authen-
tication technology for RFID applications, because it can be implemented with a compar-
atively low hardware cost [10].

In order to understand the required hardware for elliptic curve cryptography, the follow-
ing concepts need to be defined [11]:

Finite Fields  A finite field is a system consisting of a finite set $F$ (the “numbers”) to-
gether with operations $+$ and $\times$ (“addition” and “multiplication”) that satisfy a number of
properties:

- Closure: for all $a, b \in F$, we have $a + b \in F$ and $a \times b \in F$.
- Associativity: for all $a, b, c \in F$, $(a \times b) \times c = a \times (b \times c)$.
- Existence of an identity element
- Existence of an inverse
- Abelian property (commutativity)
- Distributivity

For any prime number $p$, the prime (finite) field $\mathbb{F}_p$ is defined to be the set of integers from
0 to $p - 1$, together with the operations defined as follows:
**Addition** \( a + b = r \), where \( r \) is the remainder of the division of \( a + b \) by \( p \).

**Multiplication** \( a \times b = s \), where \( s \) is the remainder of the division of \( a \times b \) by \( p \).

**The finite field \( \mathbb{F}_2^m \)** The binary finite field, \( \mathbb{F}_2^m \), is a vector space of dimension \( m \) over the prime field \( \mathbb{F}_2 \). A polynomial basis of \( \mathbb{F}_2^m \) can be introduced as follows: Let \( f(x) \) be an irreducible polynomial of degree \( m \) over \( \mathbb{F}_2 \) called the reduction polynomial. Each element \( a \) of \( \mathbb{F}_2^m \) can now be written as a binary polynomial of degree \( m - 1 \) or less:

\[
a = a_{m-1}x^{m-1} + ... + a_1x + a_0
\]

When a polynomial basis is specified, an element of \( \mathbb{F}_2^m \) can therefore be written as a bit vector of length \( m \).

**Finite field operations** Using a polynomial basis for \( \mathbb{F}_2^m \), the following field operations can be defined:

- **Addition**: \( a + b = c = (c_{m-1}...c_0) \), where \( c_i = (a_i + b_i) \mod 2 \). Field addition is the bitwise XOR of the bit vectors representing elements of \( \mathbb{F}_2^m \).

- **Multiplication**: \( a \cdot b = c = (c_{m-1}...c_0) \), where \( c(x) = \sum_{i=0}^{m-1} c_i x^i \) is the remainder of the division of the polynomial \((\sum_{i=0}^{m-1} a_i x^i)(\sum_{i=0}^{m-1} b_i x^i)\) by \( f(x) \). Field multiplication can be performed by the shift-and-add method, where one bit of \( b \) is considered at a time, starting at the MSB. If the bit is equal to one, \( a \) is added (using XOR) to a running sum \( c \). After each step, \( c \) is left-shifted by one bit and reduced modulo \( f(x) \).

**Elliptic curves over \( \mathbb{F}_2^m \)** The elliptic curve \( E(\mathbb{F}_2^m) \) over \( \mathbb{F}_2^m \) for the parameters \( a, b \in \mathbb{F}_2^m, b \neq 0 \) is defined to be the set of points \( P = (x, y) \) for \( x, y \in \mathbb{F}_2^m \) that are solution to the equation

\[
y^2 + xy = x^3 + ax^2 + b
\]

together with the special point \( \mathcal{O} \) called the point at infinity.

**Addition on elliptic curves** Addition of two points \( P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_2^m), P \neq \pm Q \), results in a new point \( P + Q = (x_3, y_3) \in E(\mathbb{F}_2^m) \), where

\[
x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \quad y_3 = \lambda(x_1 + x_2) + x_3 + y_1 \quad \left( \lambda = \frac{y_2 + y_1}{x_2 + x_1} \right).
\]

A similar expression can be found for the double of a point \( P \).

**Elliptic scalar multiplication** Scalar multiplication of a point \( P \) on an elliptic curve by the integer \( k \) is defined to be the result of adding \( P \) to itself \( k \) times. This operation is the underlying principle of all elliptic curve cryptographic schemes. There are efficient algorithms for calculating \( kP \). On the other hand, it is very hard to find \( k \) if only \( P \) and \( kP \) are known. This is the so-called discrete logarithm problem.
### 3.2.1 Montgomery algorithm for scalar multiplication

The *binary method* for scalar multiplication on elliptic curves can be implemented as follows:

```
set k ← (k_l−1...k_1k_0)_2
set P_1 ← P, P_2 ← 2P
for i = l − 2 to 0 do
    if k_i = 1 then
        set P_1 ← P_1 + P_2, P_2 = 2 · P_2.
    else
        set P_2 ← P_1 + P_2, P_1 = 2 · P_1.
    end if
end for
```

One fast algorithm for fast scalar multiplication on elliptic curves is the ladder algorithm first proposed by Montgomery [12]. It is based on the observation that in the binary method, the difference between $P_1$ and $P_2$ is always equal to $P$. This makes it possible to implement scalar multiplication with fewer registers.

### 3.3 Specifications of the cryptographic processor

The ECC core that was chosen as an example implementation for this project is designed for elliptic curve calculations over the field $\mathbb{F}_{2^{163}}$. It provides a hardware implementation of the “add-and-double” operation that is at the heart of Montgomery’s ladder multiplication algorithm.

The ALU and the register file are controlled by a Finite State Machine (FSM), which is also part of the hardware implementation. An external controller implements the physical interface for encryption or authentication and repeatedly calls the ECC core to perform the operations on two curve points.

The main specifications are given in table 3.1. The core is designed to operate at a very low clock frequency of 100 kHz.

<table>
<thead>
<tr>
<th>Technology</th>
<th>UMC 90nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target clock frequency</td>
<td>100 kHz</td>
</tr>
<tr>
<td>Power dissipation</td>
<td>min.</td>
</tr>
</tbody>
</table>

**Table 3.1: Specifications of the ECC core**

### 3.4 Architecture

Shown in Figure 3.1, the architecture of the ECC core consists of a core containing the Arithmetic and Logic Unit (ALU) on the left, and a 6-word register file on the right.
The processor’s register file consists of six registers with 163 bits each, named with the letters A through F. Each register has an enable signal which has to be asserted by the state machine if a different value is to be loaded into the register. A global reset signal (one of the inputs to the processor) can be used to initially set all registers to zero. A global clock signal clocks the flip-flops in the register file and in the counter.

To start a new operation, the system is reset and then the start signal is asserted. The start signal causes the FSM to leave the default state and progress through a number of states, first loading the input data, then calculating the sum of the two input points, and then the double of one of them.

The first two registers are equipped with multiplexers that are controlled from the FSM. Register A has the most diverse functionality. It can load the output of the ALU, the data_in input, the output of register B, or one of three predefined values.

Register B can take either the output of register A or a left-shifted copy of its own value. This allows cyclic shifting of values in register B, used for feeding the operand to the ALU in a bit-serial fashion.

The remaining registers C to F serve to store intermediate results. They each take the output of the previous register.

3.4.1 ALU

The finite field multiplier is implemented as a bit-serial unit. One of the operands is stored in register B and its MSB is an input to the ALU. During multiplication, register B is left-shifted at each clock cycle; thus the operand is entered into the ALU bit-serially.

The value of the current bit position of B is multiplied with the other operand (register C) by an array of 163 AND gates. In each cycle, this partial product is added to the running sum (register A) using an array of 162 XOR gates. If the result has a ‘1’ at the
CHAPTER 3. ELLIPTIC CURVE CRYPTOGRAPHIC PROCESSOR

bit-position $m$ (the degree of the sum is equal to the degree of the reduction polynomial),
the bits at positions corresponding to the reduction polynomial are flipped, again using
XOR gates. This step corresponds to the modulo operation which guarantees that the
degree of the running sum is less than $m$ after each clock cycle.

3.4.2 Finite State Machine

The Finite State Machine that is part of this hardware ECC implementation loads the
coordinates of the two points $P_1$ and $P_2$ into registers and then cycles through a number
of states in order to compute the sum $P_1 + P_2$ and the value $2 \cdot P_1$. The result is then
output in projective coordinates at the ports $x1_{out}$, $x2_{out}$ and $z_{out}$ and the ready
signal is asserted to signal completion of the calculations. The entire operation requires
roughly 1800 clock cycles.

3.4.3 Registers

The ECC core uses a register file with six 163-bit registers called regA, regB, ... regF
to store intermediate results. Registers regA and regB have an input multiplexer which
selects the value to store according to the select bits generated in the FSM. The other
registers form a circular shift memory where each register takes the output of the previous
register as its input.

3.5 Standard CMOS implementation

The ECC core was implemented in the UMC 90nm logic/mixed-mode CMOS process using
the Faraday standard cell library. The RTL code was synthesized using Synopsys Design
Compiler and then placed and routed using Cadence Encounter. Synthesis constraints
were chosen to favor low power consumption over high performance. No special power
optimization techniques were used and the library was not re-characterized for lower supply
voltages. Therefore, power consumption measurements can be expected to give pessimistic
results.

3.6 STSCL implementation

3.6.1 Modifications of the design flow and library

The STSCL implementation used the existing library described in [4] with some small
modifications.

An analysis of the routing process with the existing library showed that the design was
difficult to route because of the many D flip-flops in the design. The existing layout of
these flip-flops had been assembled from two copies of the existing layout for the 2-input
multiplexer. This lead to a bad cell layout with a long wire in the metal 3 layer, creating
obstructions during the routing process. The D Flip-Flop layout was thus optimized to
eliminate most of the routing in metal 3.
3.6.2 Library Specifications

The STSCL implementation of the ECC core uses the library with the characterization parameters listed in Table 3.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{DD}$</td>
<td>1.2 V</td>
</tr>
<tr>
<td>$V_{SW}$</td>
<td>0.2 V</td>
</tr>
<tr>
<td>$I_{SS,x1}$</td>
<td>1 nA</td>
</tr>
</tbody>
</table>

Table 3.2: Library corner

In order to find the best configuration, the library was characterized for different values for the bias current (200 pA, 1 nA, 5 nA). It was found that the 1 nA variant is the best choice for the present design, due to the fact that the range of available gates (driving strengths from $x1$ to $x32$) covers the requirements for synthesis. The final P&R results show that most of the gates have an intermediate driving strength, and only relatively few $x1$ and $x32$ gates are present. This suggests that the chosen bias current is appropriate for this design.
4 Results

4.1 Design and simulation flow

The two implementations were synthesized, placed and routed as described in the previous chapter. The CMOS design was then imported in Cadence Virtuoso and a device and capacitance extraction from the layout was performed. This extracted netlist was then simulated using Cadence Spectre, with a VCD (Verilog value change dump) file providing the input stimuli as well as the correct output values for verification.

The STSCL design could not be simulated correctly using this approach. It was simulated in Nanosim using the Verilog netlist and extracted capacitance file (SPEF) generated in Cadence Encounter. A supply voltage of 1 V was used.

![Figure 4.1: Output waveform of one of the regB flip-flops with the corresponding clock signal](image)

A comparison of output signals with the results from VHDL simulation shows that the design is operating correctly at a frequency of 100 kHz. Figure 4.1 shows an example of a clock and signal waveform.

4.2 Performance comparison

Table 4.1 gives a summary of the results obtained for the implementations of the ECC core using the Faraday CMOS library as well as the STSCL library. Figure 4.2 shows the final layouts for both implementations.

It can be seen from Figure 4.2 that the area of the STSCL implementation is roughly 8.5 times larger. This is mainly due to the large size of cells in the STSCL library. For
CHAPTER 4. RESULTS

Figure 4.2: Layout of the CMOS (left) and STSCL implementation (to scale)

<table>
<thead>
<tr>
<th>Implementation</th>
<th>CMOS</th>
<th>STSCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (including power rings, no pads)</td>
<td>36000 $\mu m^2$</td>
<td>308000 $\mu m^2$</td>
</tr>
<tr>
<td>Number of standard cells</td>
<td>2568</td>
<td>4541</td>
</tr>
<tr>
<td>Number of D flip-flops</td>
<td>993</td>
<td>993</td>
</tr>
<tr>
<td>Power consumption (average)</td>
<td>9 $\mu W$ @ 0.45V</td>
<td>56 $\mu W$ @ 1V</td>
</tr>
<tr>
<td>% of area in DFFs</td>
<td>54%</td>
<td>27%</td>
</tr>
</tbody>
</table>

Table 4.1: comparison of CMOS and STSCL implementations of the ECC core

example, an AND gate with minimum driving strength has an area of about 3.9 $\mu m^2$ in the CMOS library, and 42 $\mu m^2$ in the STSCL library. The layout area of STSCL gates is significantly larger because of the large gate areas. Furthermore, the two load devices need to be placed in separate n-wells because of their drain-bulk connection. This adds a lot of area due to spacing rules that have to be met, such that in the end, the gates of the two load devices need to have a distance of more than 2 $\mu m$ between them.

The different number of standard library cells in the two implementations can be explained by the fact that the commercial CMOS library offers a large selection of gates to the synthesis tools. For example, the CMOS version of the design contains 6-input AOI gates, whereas the STSCL library only has gates with up to 3 inputs.

Running the STSCL core at the same supply voltage $V_{DD} = 0.45V$ can be expected to reduce the power consumption to 25$\mu W$.

4.3 Interpretation of results

In the present design, the critical register-to-register path contains 15 logic gates, even though the design had been synthesized with tight timing constraints. For STSCL a high logic depth means that each gate is doing a useful operation only during a small fraction of the clock cycle. Even if every node were to switch once per clock period, the gates are
‘wasting’ current for more than 90% of the clock period.

Since STSCL allows reduction of the power consumption at the cost of speed down to tail currents of a few pA, the STSCL design could offer better performance at lower frequencies. However, this is not feasible due to the speed constraint imposed on the ECC core.

Interconnect capacitance  Another factor is the large area of the STSCL block, which leads to wires being roughly three times longer on average. Post-layout capacitance extraction shows a total routing capacitance of 89 pF in STSCL (counting both wires of each differential pair), whereas the CMOS design has only 13 pF of capacitance.

Equation 2.2 can be used to calculate an estimate of the theoretical lower limit on power dissipation. The average logic depth in each path was estimated to be 10, and the single-ended gate capacitance 2 fF for each gate input. For a conservative estimate, it can be assumed that all 4500 gates have only two inputs, resulting in a total capacitance of

\[ C_{\text{total}} \approx \frac{89 \text{pF}}{2} + 4500 \cdot 2 \cdot 2 \text{fF} = 62.5 \text{pF} \]

\[ I_{\text{total}} \geq \ln(2) \cdot C_{\text{total}} \cdot d \cdot f \cdot V_{SW} \approx \frac{0.7 \cdot 62.5 \text{pF} \cdot 10 \cdot 10^5 \text{s}^{-1} \cdot 0.2 \text{V}}{2} = 8.8 \mu \text{A} \]

This value is significantly lower than the simulated result of 56 µA. Several reasons for this difference can be identified. First, the estimate is overly conservative because in reality, many gates have more than two inputs, and D flip-flops use twice the current of a normal gate. On the other hand, in the final design, some paths are significantly faster than they need to be (Figure 4.3). The EDA tools did not correctly replace gates with low-power equivalents in those paths. Moreover, the delays in a path may not be equal at all; for instance, if one stage has to drive a very large fan-out, the delay of that stage will be greater, requiring higher current consumption in the other stages to compensate for this delay.

While STSCL was expected to be very efficient due to the low voltage swing, it has to be noted that the differential nature of the signals entails a switching current that is at least twice as high as in the single-ended case. In fact, the routing method used leads to a large coupling capacitance between the two wires of the differential signal. If the leakage current in CMOS can be kept at acceptable values, it can thus be expected that the power dissipation of STSCL circuits with a differential voltage swing of ±0.2 V will not be much better than that of a CMOS circuit operating at \( V_{DD} = 0.4 \text{V} \).

Supply Current  Figure 4.4 shows the current flowing into the \( V_{DD} \) node, for CMOS and STSCL\(^1\). Whereas in CMOS, the current is concentrated in peaks (when the flip-flops are switching), STSCL, as expected, draws a nearly constant current with only minor transients due to switching.

\(^1\)For CMOS, a higher frequency (10 MHz) was used to generate this waveform; otherwise the switching currents would be very narrow peaks.
CHAPTER 4. RESULTS

Figure 4.3: Register-to-register path slack distribution in the STSCL core

4.3.1 Advantages and disadvantages STSCL

With regard to the present implementation of the ECC core in STSCL, the following conclusions can be drawn:

**Advantages**

- **Operation at very low speed:** there is almost no lower limit to the power consumption of STSCL gates. For clock frequencies in the kilohertz range and below, STSCL is an ideal choice. In CMOS, leakage power is determined by the supply voltage, which has to meet a minimum noise margin requirement in the presence of PVT variations. In STSCL, current consumption and noise margin are controlled separately, and so the power dissipation can be reduced to a very low value while keeping a reasonable voltage.

- **Shallow logic depth:** Similarly, circuits with very shallow pipelining can be implemented more efficiently in STSCL since low logic depth means each gate is switching during a significant fraction of the clock period. That way, less current is ‘wasted’ in inactive gates.

- **Tunability over a wide range of frequencies:** The use of a single replica bias circuit makes it possible that the same circuit can be used over a range of frequencies of several orders of magnitude. Using a single constant current source and a programmable current mirror, STSCL circuits can be used in applications where the operating speed, and therefore the power consumption, has to be adjusted dynamically to meet performance demands.
Figure 4.4: Supply current waveform for CMOS (top) and STSCL.
CHAPTER 4. RESULTS

Disadvantages

- **Large area**: The large area due to matching requirements and the overhead for the load devices present in each gate are a significant problem.

- **Power consumption**: In their current version, larger STSCL designs implemented using a top-down design flow are not competitive due to the issues with large cell area and interconnect capacitance.
5 Outlook

5.1 Possible improvements in the STSCL library

5.1.1 Device sizing optimization

As discussed in Section 2.4.1, optimal sizing of standard cell devices takes into account the relative weight with which the different transistors contribute to noise margin variability.

In order for STSCL to be more area-efficient, logic synthesis should favor cells with many inputs. Figure 5.1 shows the relative sizes of the differential pairs, load devices, and tail current source. Even though the differential pair devices implement the actual ‘logic’ of the gate, they only occupy a comparatively small area. For this reason, gates with a small number of inputs are very inefficient in terms of area: a 3-input XOR gate has an area of 50.5\(\mu m^2\), whereas a simple buffer has an area of 39\(\mu m^2\).

In CMOS, using low fan-in gates is justified by the higher driving strength that these gates offer for a given input capacitance. In STSCL however, the driving strength only depends on the tail current. For these reasons, it is more efficient to use more complex gates in STSCL, both in terms of area (due to the overhead for tail and load devices) and power consumption. An STSCL library should therefore contain a large selection of high fan-in gates, possibly custom-made for a specific design.
5.1.2 Shallow pipelining

It has been suggested in [3] that in order to increase the power efficiency of STSCL, single-stage pipelining is to be used. In this scenario, the system is clocked with two clock phases which alternately latch the outputs of two consecutive gates by switching the tail current to a lower value. An output stage consisting of a pair of cross-coupled NMOS transistors biased with a small current can be used as a keeper to ensure that the output state does not degrade during the hold phase.

While this shallow-pipelining method is very promising for manually designed circuits blocks like multipliers, it would be difficult to integrate in a top-down design flow.
6 Conclusion

This work successfully demonstrated the design of an elliptic curve cryptographic core in subthreshold source-coupled logic using a top-down design flow. The ECC core runs correctly at the specified frequency of 100 kHz.

The comparison to a standard CMOS implementation of the same core shows, however, that in the current state, the STSCL library is not competitive in terms of power dissipation. The area required by the PMOS load devices and sizing constraints imposed by device variations make the STSCL standard cells considerably larger than their CMOS counterparts. On the system-level, this leads to an excessive amount of device and interconnect capacitance.

Advantages of STSCL over CMOS have been identified. The flat power profile of STSCL circuits makes it difficult to extract information on the data being processed by studying the supply current. This is an important advantage for cryptographic applications.
Bibliography


Appendix

VHDL code for the finite field multiplier

Listing 6.1: multiplier.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity multiplier is
  generic (nbits : natural := 163);
  Port ( clk : in std_logic;
          start : in std_logic;
          A, B : in std_logic_vector (nbits-1 downto 0);
          output : out std_logic_vector (nbits-1 downto 0);
          done : out std_logic
        );
end multiplier;

architecture struct of multiplier is
  signal xorshift_out : std_logic_vector(nbits-1 downto 0);
  signal sum_reg : std_logic_vector (nbits-1 downto 0);
  signal reduce_out : std_logic_vector (nbits-1 downto 0);
  signal shift_reg, B_reg : std_logic_vector (nbits-1 downto 0);

  component counter is
    port( clk : in std_logic;
          start : in std_logic;
          done : out std_logic
      );
  end component;

  component reduce is
    generic (nbits : natural);
    port( enable : in std_logic;
          input : in std_logic_vector (nbits-1 downto 0);
          output : out std_logic_vector (nbits-1 downto 0)
      );
  end component;

  component xorshift is
    generic (nbits : natural);
    port ( shifted_bit : in std_logic;
           multiplicand : in std_logic_vector (nbits-1 downto 0)
          ;
           sum_in : in std_logic_vector (nbits-2 downto 0);
           sum_out: out std_logic_vector (nbits-1 downto 0)
          );
  end component;
```
end component;
begin
xs : xorshift
 generic map ( nbits => nbits )
 port map ( shifted_bit => shift_reg(nbits -1),
 multiplicand => B_reg,
 sum_in => sum_reg(nbits -2 downto 0),
 sum_out => xorshift_out);
red : reduce
 generic map ( nbits => nbits )
 port map ( enable => sum_reg(nbits -1),
 input => xorshift_out,
 output => reduce_out);
cnt : counter
 port map ( clk => clk,
 start => start,
 done => done);
process ( clk )
begin
 if rising_edge ( clk ) then
  if start = '1' then
   shift_reg <= A;
   B_reg <= B;
   sum_reg <= ( others => '0' );
  else
   sum_reg <= reduce_out;
   shift_reg( nbits -1 downto 0 ) <= shift_reg( nbits -2
downto 0 ) & '0';
  end if;
 end if;
end process;
output <= sum_reg;
end architecture struct;

Listing 6.2: counter.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
  Port ( clk : in  std_logic;
         start : in  std_logic;
         done : out std_logic);
end counter;

architecture Behavioral of counter is
signal tmp: std_logic_vector(7 downto 0);
begin
process (clk)
begin
  if rising_edge(clk) then
    if start = '1' then
      tmp <= "10100011";
    else
      tmp <= tmp -1;
    end if;
  end if;
end process;

process (tmp)
begin
  if tmp = 0 then
    done <= '1';
  else
    done <= '0';
  end if;
end process;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

tenity reduce is
  generic (nbits : natural := 8);
  port ( enable : in std_logic;
        input : in std_logic_vector (nbits-1 downto 0);
        output : out std_logic_vector (nbits-1 downto 0) );
end reduce;

architecture arch of reduce is
  signal mask : std_logic_vector (nbits-1 downto 0);
begin
  mask(nbits-1 downto 0) <= (0 => enable, 3 => enable, 6 => enable,
    7 => enable, others => '0');
  output(nbits-1 downto 0) <= input xor mask;
end arch;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xorshift is
  generic (nbits : natural := 8);
  port (shifted_bit : in std_logic;
        multiplicand : in std_logic_vector (nbits-1 downto 0);
        sum_in : in std_logic_vector (nbits-2 downto 0);
        sum_out : out std_logic_vector (nbits-1 downto 0)
  );
end xorshift;

architecture arch of xorshift is
  component and_2
    port(
      a : in std_logic;
      b : in std_logic;
      c : out std_logic
    );
  end component and_2;
  
  component xor_2
    port(
      a : in std_logic;
      b : in std_logic;
      c : out std_logic
    );
  end component xor_2;

  signal and_temp : std_logic_vector (nbits-1 downto 0);
begin
  and_gate : for i in 0 to nbits-1 generate
    Comp: and_2 port map (a => shifted_bit, b => multiplicand(i),
                             c => and_temp(i));
  end generate;

  xor_gate : for i in 1 to nbits-1 generate
    Comp: xor_2 port map (a => and_temp(i), b => sum_in(i-1),
                           c => sum_out(i));
  end generate;

  sum_out(0) <= and_temp(0);
end architecture;

Listing 6.5: xor_2.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity xor_2 is
  Port ( a : in std_logic;
         b : in std_logic;
         c : out std_logic);
end xor_2;

architecture Behavioral of xor_2 is
begin
c <= a xor b;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_2 is
  Port ( a : in std_logic;
         b : in std_logic;
         c : out std_logic);
end and_2;

architecture Behavioral of and_2 is
begin
c <= a and b;
end Behavioral;

MATLAB code to measure noise margin statistics

function snm(FILEF1, FILEF2)
% use FILEF1 = FILEF2 for NM of a gate with itself

  close all;
  AF1 = importdata(FILEF1);
  AF2 = importdata(FILEF2);
  Nsamples = length(AF1.colheaders) / 2;
  interp = size(AF1.data,1);
  % Voltage swing:
  Vdd = 0.4;
xo = AF1.data(:,1);
v1 = AF1.data(:,2:2:Nsamples*2);
v2 = AF2.data(:,1);
v2 = AF2.data(:,2:2:Nsamples*2);
F1 = v1;
F2 = [];
for i = 1:Nsamples
    F2 = interp1(v2(:,i),v1,xpoints);
end
plot(xpoints,F1)
hold on
plot(xpoints,F2)
axis([0,Vdd,0,Vdd]);
axis equal;

% rotated system of coordinates
v1 = (xpoints*ones(1,Nsamples) + v1)/sqrt(2);
u1 = (xpoints*ones(1,Nsamples) - v1)/sqrt(2);
v2 = (xpoints*ones(1,Nsamples) + v2)/sqrt(2);
u2 = -(xpoints*ones(1,Nsamples) - v2)/sqrt(2);

% points on the new 'x'-axis
upoints = linspace(-0.9*Vdd/sqrt(2),0.9*Vdd/sqrt(2),interp);

v1resamp = [];
v2resamp = [];

% interpolate the curve at the new points
for i = 1:Nsamples
    v1resamp = [v1resamp interp1(u1(:,i),v1(:,i),upoints)];
    v2resamp = [v2resamp interp1(u2(:,i),v2(:,i),upoints)];
end

%v1resamp(isnan(v1resamp)) = 0;
%v2resamp(isnan(v2resamp)) = 0;

plot(upoints,v1resamp,upoints,v2resamp);
axis equal;

snr = [];
for i = 1:Nsamples
    diff = (v1resamp(:,i)*ones(1,Nsamples)-v2resamp);
    snr = [snr; min(max(diff(1:round(interp/2),:),[]),1),
           max(-diff(round(interp/2):interp,:),[]),1])'/sqrt(2);
end

figure()
histfit(snr,20);
[mu, sig] = normfit(snr)
xlabel('Static Noise Margin [V]');
\texttt{v = axis();}

\texttt{text(v(1) + 0.7*(v(2)-v(1)), v(3) + 0.7*(v(4)-v(3)),}
\texttt{strvcat(['\mu = ', num2str(mu)], ['\sigma = ', num2str(sig)],
\texttt{['N = ', num2str(Nsamples) 'x ' num2str(Nsamples)]));}

\texttt{minSNR = min(snr)}