GALS System Design
Side Channel Attack Secure Cryptographic Accelerators

Frank K. Gürkaynak
Integrated Systems Laboratory
ETH Zurich

22 November 2005
Ph.D. Thesis Presentation
Outline

1. Outline

2. Globally-Asynchronous Locally-Synchronous (GALS) Design

3. Cryptography

4. GALS implementation of the AES Algorithm

5. Results and Conclusions
What is wrong with the way we design chips now?

Modern System-on-Chip circuits ...

- Contain millions of transistors
- Require clock rates exceeding 100s of MHz
- Include 100s of subblocks
- Use 10s of different clock domains
What is wrong with the way we design chips now?

Modern System-on-Chip circuits ...

- Contain millions of transistors, trend **increasing**
- Require clock rates exceeding 100s of MHz, trend **increasing**
- Include 100s of subblocks, trend **increasing**
- Use 10s of different clock domains, trend **increasing**
What is wrong with the way we design chips now?

Modern System-on-Chip circuits ...  
- Contain millions of transistors, trend **increasing**  
- Require clock rates exceeding 100s of MHz, trend **increasing**  
- Include 100s of subblocks, trend **increasing**  
- Use 10s of different clock domains, trend **increasing**

... are not easy to design  
- The clock signal must be distributed to an increasing number of elements with increased precision  
- Many independently designed components must be combined to a large system.  
- All subsystems must be able to reliably exchange data
Globally-Asynchronous Locally-Synchronous Design

GALS is a methodology to enable the design of complex digital systems on chip.

- System is divided into smaller GALS modules
- Each module works synchronously
- Interconnected modules communicate asynchronously
Globally-Asynchronous Locally-Synchronous Design

GALS is a methodology to enable the design of complex digital systems on chip.

- System is divided into smaller GALS modules
- Each module works synchronously
- Interconnected modules communicate asynchronously
- Was first developed by D. Chapiro in 1984
- First chip implementation by J. Muttersbach in 1999
GLOBALLY-ASYNCHRONOUS LOCALLY-SYNCHRONOUS DESIGN

GALS is a methodology to enable the design of complex digital systems on chip.

- System is divided into smaller GALS modules
- Each module works synchronously
- Interconnected modules communicate asynchronously
- Was first developed by D. Chapiro in 1984
- First chip implementation by J. Muttersbach in 1999

GALS implementations differ in:

- synchronization method between blocks
- specific asynchronous communication protocol used
Basic GALS Structure

Synchronous system
Two large functional blocks of a synchronous system
Local clock generators

GALS modules are formed by adding a local clock generator for each functional block
Basic GALS Structure

GALS system
Port controllers are added to regulate data transfers between GALS modules
GALS Works

GALS at IIS

- J. Muttersbach
  First implementation
- T. Villiger
- S. Oetiker
- F. K. Gürkaynak
GALS Works

GALS at IIS

- J. Muttersbach
- T. Villiger
  Multi-point interconnect
- S. Oetiker
- F. K. Gürkaynak
GALS Works

GALS at IIS

- J. Muttersbach
- T. Villiger
- S. Oetiker
  Local clock generators
- F. K. Gürkaynak
GALS Works

GALS at IIS

- J. Muttersbach
- T. Villiger
- S. Oetiker
- F. K. Gürkaynak

Design and test flow
Why GALS?

Advantages

- No global clock distribution problems
- Modular design flow
- Potential for low-power design
- Offers new possibilities for designers
Cryptography 101

Private key ciphers

- Alice encrypts **plain-text** information by using a **cipher-key**.
- Bob can decrypt the resulting **cipher-text** only if he has access to the same cipher-key.
Cryptography 102

**Security**

- Oscar wishes to obtain the plain-text
- Oscar knows everything about the cryptographic algorithm
- Oscar can observe/modify the cipher-text
- but..
Cryptography 102

**Security**
- Oscar wishes to obtain the plain-text
- Oscar knows everything about the cryptographic algorithm
- Oscar can observe/modify the cipher-text
- but.. Oscar **does not know** the cipher-key
Advanced Encryption Standard (AES)

**AES Standard**
- by NIST 2001
- 128 bit data
- 128 bit key
- 10/12/14 rounds

**Components**
- **ShiftRows**

**AES Standard**

- **by NIST 2001**
- **128 bit data**
- **128 bit key**
- **10/12/14 rounds**

**Components**

- **ShiftRows**
Advanced Encryption Standard (AES)

AES Standard
- by NIST 2001
- 128 bit data
- 128 bit key
- 10/12/14 rounds

Components
- ShiftRows
- AddRoundKey
Advanced Encryption Standard (AES)

AES Standard
- by NIST 2001
- 128 bit data
- 128 bit key
- 10/12/14 rounds

Components
- ShiftRows
- AddRoundKey
- SubBytes
Advanced Encryption Standard (AES)

AES Standard
- by NIST 2001
- 128 bit data
- 128 bit key
- 10/12/14 rounds

Components
- ShiftRows
- AddRoundKey
- SubBytes
- MixColumns
Side-Channels

Once an otherwise secure algorithm is implemented in either Hardware or Software it gains physical properties that can be observed:

- Time required to finish the operation
- Power consumption
- Electromagnetic Radiation
- Heat dissipation
- Sound

These properties are called **Side Channels**

**Side-Channel Attacks**

In 1996, P. Kocher showed that it is possible to obtain additional information on the cipher-key by observing these side-channels.
Differential Power Analysis (DPA)

1. Select a *subkey* and a *target operation*.
2. Use a simple model to *predict* the power consumption for *S input vectors*.

![Diagram of Differential Power Analysis (DPA)]
Differential Power Analysis (DPA)

1. Select a subkey and a target operation

2. Use a simple model to predict the power consumption for $S$ input vectors

3. predict the power consumption for all $K$ subkey permutations
Differential Power Analysis (DPA)

1. Select a subkey and a target operation
2. Use a simple model to predict the power consumption for S input vectors
3. Predict the power consumption for all K subkey permutations
4. Measure the power consumption using the same S input vectors
Differential Power Analysis (DPA)

1. Select a *subkey* and a *target operation*.
2. Use a simple model to *predict* the power consumption for $S$ input vectors.
3. Predict the power consumption for all $K$ subkey permutations.
4. *Measure* the power consumption using the same $S$ input vectors.
5. *Determine* if one of the power hypotheses shows a *distinctively higher correlation* to the measurement.
The GALS implementation is called **Acacia**.

- Operations are divided between a 128-bit **Goliath** and a 32-bit **David** unit.
The GALS implementation is called **Acacia**.

- Operations are divided between a 128-bit **Goliath** and a 32-bit **David** unit
- David and Goliath are separate GALS modules
Block Diagram

The GALS implementation is called **Acacia**.

- Operations are divided between a 128-bit **Goliath** and a 32-bit **David** unit.
- David and Goliath are separate GALS modules.
- There is a second David unit running in parallel.
- One round of AES requires 1 Goliath and 4 David operations.
Normal Operation

The attacker will normally target a single operation, and will measure the power consumption of this particular clock cycle.
Implemented countermeasures

Inserting dummy operations

Inserting random dummy cycles will confuse the attacker, since the targeted operation will not always be executed at a specific clock cycle. Unfortunately, this also increases the run-time.
Implemented Countermeasures

Change ordering of operations

Independent operations can be re-ordered arbitrarily. Contrary to inserting dummy cycles, this does not increase the run-time.
Implementated Countermeasures

Parallelization

Executing operations in parallel creates more activity at the same time, this appears as noise for the attacker.
Parallelization

Executing operations in parallel creates more activity at the same time, this appears as noise for the attacker.
Implemented Countermeasures

Parallelization

Executing operations in parallel creates more activity at the same time, this appears as noise for the attacker.
Implemented Countermeasures

Introducing GALS modules

GALS modules have their own local clock generator, their clocks are independent and can not be controlled by the attacker
Implemented Countermeasures

Variable clock periods

Each GALS module can randomly change its own clock period. This adds even more uncertainty.
Chip Photo

Acacia
- UMC 0.25 µm CMOS
- Total area 1.75 mm²
  - David 0.221 mm²
  - Goliath 0.687 mm²
  - Sync. 0.584 mm²
- Rate 177.7 Mb/s
- Energy 1.232 mJ/Mb

This part of the chip occupied by two independent AES designs: Baby and Pampers

Synchronous Interface & Reference Design

Goliath

David

David

Clockgen

g2s
d2g
g2d
g2d

Clockgen

Clockgen
Conclusions

- A novel **GALS based crypto ASIC** implementing the AES algorithm was presented.
- In addition to traditional DPA countermeasures, the chip also includes GALS modules that use **randomly varying clocks** which make known attacks extremely difficult.
- The GALS design methodology was refined. The presented design was designed **using mainly standard EDA tools**.
- A combination of functional and scan-chain based testing allows a **stuck-at-coverage** of more than 99.8%.

Is this really secure? We don’t know yet. The security has to be evaluated by cryptanalysts.
Conclusions

- A novel GALS based crypto ASIC implementing the AES algorithm was presented.

- In addition to traditional DPA countermeasures, the chip also includes GALS modules that use randomly varying clocks which make known attacks extremely difficult.

- The GALS design methodology was refined. The presented design was designed using mainly standard EDA tools.

- A combination of functional and scan-chain based testing allows a stuck-at-coverage of more than 99.8%.

Is this really secure?

We don’t know yet. The security has to be evaluated by cryptanalysts.
QUESTIONS?

Acknowledgements for GALS
Stephan Oetiker, Thomas Villiger, Hubert Kaeslin, Norbert Felber

Acknowledgements for Crypto-chips
Stefan Achleitner, Gérard Basler, Andres Erni, Dominique Gasser, Peter Haldi, Franco Hug, Adrian Lutz, Norbert Pramstaller, Stefan Reichmuth, Pieter Rommens, Jürg Treichler, Stefan Zwicky
and
Andreas Burg, Matthias Braendli, Stefan Eberli, Simon Haene Stefan Mangard, Elisabeth Oswald, S. Berna Örs
GALS System Design

1 GALS port is activated by the Pen signal, which enables Req
The receiving GALS module sets **Ack**, clock pause request **Ri** is set.
The clock pause is acknowledged by $Ai$, the LS Island is paused.
Data transfer is complete, $Ta$ is set and $Req$ is reset
Timing Diagram

Locally Synchronous Island

- Clock
- Pen
- Ta

Receiving GALS Module

- Req
- Ack
- Ri
- Ai

Local Clock Generator

5

All handshake signals return to their initial values, local clock is released.

Local clock is paused
Normal operation resumes, $Ta$ remains active until the $Pen$ is reset.
Local Clock Generator

GALS System Design

kgf, Integrated Systems Laboratory (IIS)
Mutual exclusion element

![Mutual exclusion element](image_url)
Design flow for GALS (as used in Shir-Khan)
AES implementations at IIS

<table>
<thead>
<tr>
<th></th>
<th>Riddler</th>
<th>Fastcore</th>
<th>Ares</th>
<th>Baby / Pampers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 128 bit parallel</td>
<td>2 x 128 bit parallel</td>
<td>128 bit</td>
<td>128 / 32 bit</td>
<td>16 bit</td>
</tr>
<tr>
<td>2.16 Gb/s (pipelined)</td>
<td>2.12 Gb/s</td>
<td>1.15 Gb/s (128 bit)</td>
<td>2.12 Gb/s</td>
<td>0.285 / 0.230 Gbit/s</td>
</tr>
<tr>
<td>37.8 mm(^2) (0.6 (\mu)m)</td>
<td>3.56 mm(^2) (0.25 (\mu)m)</td>
<td>1.2 mm(^2) (0.25 (\mu)m)</td>
<td>1.2 mm(^2) (0.25 (\mu)m)</td>
<td>0.35 / 0.58 mm(^2) (0.25 (\mu)m)</td>
</tr>
<tr>
<td>En/Decryption (ECB)</td>
<td>En/Decryption (all)</td>
<td>Encryption (ECB/OFB)</td>
<td>Encryption (ECB/OFB)</td>
<td>Encryption (ECB/OFB)</td>
</tr>
<tr>
<td>Parallel Datapath</td>
<td>Independent Enc/Dec</td>
<td>Includes masking</td>
<td>Plain / Countermeasure</td>
<td></td>
</tr>
</tbody>
</table>

---

GALS System Design

kgf, Integrated Systems Laboratory (IIS)
SubBytes determines AES performance

<table>
<thead>
<tr>
<th>Datapath width</th>
<th>8-bit</th>
<th>16-bit</th>
<th>32-bit</th>
<th>64-bit</th>
<th>128-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel SubBytes units</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Complexity (gate eq)</td>
<td>5,052</td>
<td>6,281</td>
<td>7,155</td>
<td>11,628</td>
<td>20,410</td>
</tr>
<tr>
<td>Area (normalized)</td>
<td>1</td>
<td>1.266</td>
<td>1.472</td>
<td>2.432</td>
<td>4.269</td>
</tr>
<tr>
<td>Clock cycles for AES-128</td>
<td>160</td>
<td>80</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Critical path (normalized)</td>
<td>1.349</td>
<td>1.341</td>
<td>1.206</td>
<td>1.133</td>
<td>1</td>
</tr>
<tr>
<td>Total time (normalized)</td>
<td>21.580</td>
<td>10.729</td>
<td>4.825</td>
<td>2.227</td>
<td>1</td>
</tr>
</tbody>
</table>
Protection your weak spots

- **DPA measures power consumption**
  Add **Noise** (unrelated switching activity) to confuse the measurements

- **DPA targets a specific operation**
  Change the operation order by inserting **Random Operations**

- **Power consumption of CMOS is data dependent**
  Use **Alternative Logic Styles**

- **There are direct operations between input and output**
  Prevent direct operations by **Masking** the key with random data
DPA attacks work

Key 0x73

Number of measurements

Correlation

-0.25 -0.2 -0.15 -0.1 -0.05 0 0.05 0.1 0.15 0.2 0.25

Key 0x73

Number of measurements
DPA attack setup
## Area overhead of GALS

<table>
<thead>
<tr>
<th></th>
<th>David</th>
<th>Goliath</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Area µm²</strong></td>
<td>183,007</td>
<td>551,194</td>
</tr>
<tr>
<td><strong>Area µm²-LSFRs</strong></td>
<td>26,928</td>
<td>73,512</td>
</tr>
<tr>
<td><strong>Area µm²-ClockGen</strong></td>
<td>7,579</td>
<td>7,626</td>
</tr>
<tr>
<td><strong>Area µm²-Ports</strong></td>
<td>6,225</td>
<td>11,412</td>
</tr>
<tr>
<td><strong>Area µm²-GALS</strong></td>
<td>196,811</td>
<td>570,233</td>
</tr>
<tr>
<td><strong>Area µm²-TOTAL</strong></td>
<td>963,855</td>
<td></td>
</tr>
</tbody>
</table>
# Latency overhead of GALS

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>GALS+DPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical path (ns)</td>
<td>5.43</td>
<td>5.84</td>
</tr>
<tr>
<td>Latency (cycles)</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Clock freq. (MHz)</td>
<td>170.96</td>
<td>250.8</td>
</tr>
<tr>
<td>Enc(clock cycles)</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Enc time (ns)</td>
<td>40.88</td>
<td>42.38</td>
</tr>
</tbody>
</table>
Block diagram of David

- **Data IN** x DI → **Data OUT** x DO
- **Reg-32**
- **89-bit LFSR**
- **Mult. Inverse**
- **Output Select**
- **Affine Transformation**
- **MixColumns**
- **InvMixColumns**
- **Inverse Affine T.**
Block diagram of Goliath

![Block diagram of Goliath](image-url)
Goliath to Synchronous Interface
Goliath to David
David to Goliath

<table>
<thead>
<tr>
<th></th>
<th>Pen+</th>
<th>Req+</th>
<th>Ack+</th>
<th>Ri+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pen-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ai-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ri-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ack-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Req-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ta-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ta+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ai+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- AO21D1
- AND2D1
- BUFDL
- INVD1
- OR2D1
- AND3D1
- Pen
- Z
- req
- ri
- ta
- ai
- Ack
- Ta-
Scan-test configuration
Stuck-at-fault test coverage

**Stuck-at-fault testing**

- There are a **total** of 154,604 stuck-at faults in the entire circuit.
- **Only 182** of these faults are within the asynchronous finite state machines.
- A straightforward test vector generation using TetraMax **fails** to detect **3.089** faults.
- Using a simple encryption/decryption operation **2.796** of these faults were **detected by simulation**.
- The **total test coverage** obtained by combining these two methods exceeds **99.8%**.
Distribution of the first SubBytes operation

Mode 00: As fast as possible

Mode 01: Slightly Random

Mode 10: Mostly Random

Mode 11: Pre-programmed Policy
Simulation result

![Simulation Diagram](chart.png)

- **Interface**
- **Clk**, **Req**, **Ack**, **Strb**

**Input for Next Operation**

**Output of Last Operation**

**Request to start**

**Acknowledge**

**Goliath**

**David Tschopp**

**David Perels**

**Clock**

- 100MHz
- 120MHz
- 170MHz
- 190MHz
- 200MHz
- 190MHz
- 180MHz
- 150MHz

**Period**

- 7200 ns
- 7600 ns
- 8 us

**Power/Mode**

- ~100MHz
- ~120MHz
- ~170MHz
- ~190MHz
- ~200MHz

- Clock Pause

**More random**

**Less random**

**Fast**

**Less random**

**More random**
## Operation modes of Acacia

<table>
<thead>
<tr>
<th>Operation Mode</th>
<th>I/O Clock [MHz]</th>
<th>Encr. [ns]</th>
<th>Throughput [Mb/s]</th>
<th>Energy [mJ/Mb]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acacia - 00</td>
<td>50</td>
<td>720.0</td>
<td>177.7</td>
<td>1.232</td>
</tr>
<tr>
<td>Acacia - 01</td>
<td>50</td>
<td>880.0</td>
<td>145.4</td>
<td>1.362</td>
</tr>
<tr>
<td>Acacia - 10</td>
<td>50</td>
<td>2,440.0</td>
<td>57.1</td>
<td>2.704</td>
</tr>
<tr>
<td>Acacia - 11</td>
<td>50</td>
<td>920.0</td>
<td>139.1</td>
<td>1.198</td>
</tr>
<tr>
<td>Synchronous</td>
<td>150</td>
<td>779.2</td>
<td>164.2</td>
<td>0.976</td>
</tr>
</tbody>
</table>
Clock period versus delay-line settings

Acacia Local Clock Generator Period (Chip #1)

Local Clock Period [ns] versus Number of Delay Slices for Acacia

- Goliath
- D. Perels
- D. Tschopp
- Simulation
Clock frequency versus delay-line settings

Goliath Clock Generator Frequency (Chip #1-Chip #14)

- Chip #1
- Chip #14
- Simulation

Local Clock Frequency [MHz] vs. Number of Delay Slices

GALS System Design
kgf, Integrated Systems Laboratory (IIS)
Power consumption vs maximum GALS module frequency

Acacia Power Dissipation

Clock Frequency of GALS Modules [MHz]

Power Consumption [mW]
Power consumption of different operation modes

Acacia Power Dissipation with Different Operation Modes

Mode00
Mode01
Mode10
Mode11

Power Consumption [mW]
I/O Clock Period [MHz]

GALS System Design
kgf, Integrated Systems Laboratory (IIS)
SBB: Das Handy ist jetzt auch ein Billett