A Globally-Asynchronous Locally-Synchronous VLSI Circuit for the SAFER Cryptoalgorithm

T. Villiger, J. Muttersbach, H. Kaeslin, N. Felber, W. Fichtner
Integrated Systems Laboratory, Swiss Federal Institute of Technology

ACiD-WG Workshop, Neuchatel, Switzerland

Funded by: Infineon Technologies, Germany
Philips Semiconductors Zürich, Switzerland
Commission for Technology and Innovation, Switzerland
Outline

- GALS principle
- GALS building blocks
- Data channels
- SAFER cryptoalgorithm
- Architecture of the cryptochip
- Design flow
- Results
- Conclusion, Future work
GALS Principle

- Locally-synchronous (LS) modules perform all functionality
- Data are transferred in self-timed manner between LS modules

Benefits:
- Facilitates clocking of SOCs
- Modularity enhances optimization and re-use of LS blocks
- Provides a hook for low-power operation
- Natural inclusion of totally asynchronous modules
The asynchronous wrapper

- Asynchronous port controllers allow for fast handshake processing
- Metastability of data is **prevented** by pausing the clock
- No extra latency (synchronizers, FIFOs... ) introduced
- Wrapper shall be assembled from predesigned elements
Pausable clock generation

- Ring oscillator for local clock generation
- Arbitration with Mutual Exclusion (ME) elements
- Programmable delayline
Delayline and Arbitration

Principle of delayline using slices

- every slice can bypass rest of delayline
- delay adjustable over a wide range
- small delay increment (≈ 350ps per slice with 0.25μm technology)

Multiple Arbitration block

- safely arbitrates between incoming requests and rising edges on rclk
- row of mutual exclusion elements
- scales linearly to multiple ports
GALS building blocks

- Port controller responsible for managing all data transfers
- Each unit is captured by a structural VHDL description
- Asynchronous port controllers achieve cycle times less than 350ps

- A **Poll-type** ports ask for clock stretching only to prevent metastability and ensure data correctness: “proceed while waiting”
- **Demand-type** ports also ensure data integrity but stop the local clock as soon as they are enabled: “sleep while waiting”
- **ROM (LUT) port:** Interface at the ROM port is just a fake delay between \( Rp \) and \( Ap \) matching the data delay
- **RAM access:** Essentially a Demand-Out port with bidirectional data transfer (two data vectors in opposite direction controlled by a common handshake pair)
• Performs 2-phase to 4-phase conversion
• Transfer acknowledge (Ta) indicates successful transfer
• 3D-tools available for synthesis (by K.Y. Yun)
D-input port (cont.)

Synthesis results

\[ Ri = Rp \overline{Ri} + \overline{Den} Z0 + Den \overline{Ap} \overline{Z0} \]

\[ Ap = Rp Ai + Ai Ap \]

\[ Z0 = \overline{Rp} Z0 + \overline{Ai} Z0 + Den \overline{Rp} Ap \]

2-level AND-OR Implementation

Compact, fast and hazard-free implementation of async FSM
The transfer channels all work with rendezvous scheme.

- Push channels only, but pull channels would also be possible.
- Data latches needed due to undetermined clock relation between the modules.
Data channel simulation

P-out to D-in channel:

- Clock stretching infrequent for P-ports
- Fast transfer processing
- Local clocks restart in phase with data
SAFER cryptoalgorithm

- Secret-key iterated block cipher
- Encryption and decryption slightly different
- Byte oriented: blocks of 8 bytes
- Recommended number of rounds 10 to 12
- Additional input/output transformation
- Comes with variable key lengths (Implemented version: SK-128)
Cryptographic operation modes

ECB Mode

\[ p \rightarrow \text{SAFER encryption} \rightarrow c \rightarrow \text{SAFER decryption} \rightarrow p \]

\[ k \]

\[ k^{-1} \]

CFB mode

\[ IV \rightarrow \text{SAFER encryption} \rightarrow k \rightarrow c \rightarrow \text{SAFER decryption} \rightarrow p \]

\[ c \]

CBC mode

\[ p \rightarrow \text{SAFER encryption} \rightarrow c \rightarrow \text{SAFER decryption} \rightarrow p \]

\[ k \]

\[ k^{-1} \]

OFB mode

\[ IV \rightarrow \text{SAFER encryption} \rightarrow k \rightarrow c \rightarrow \text{SAFER encryption} \rightarrow p \]

\[ c \]

\[ k \]

\[ p : \text{Plaintext} \]

\[ c : \text{Ciphertext} \]

\[ k : \text{Key} \]

\[ IV : \text{Initialization Vector} \]
Design Example: MARILYN SAFER cryptochip

- SAFER block cipher algorithm
- Supports ECB, CBC, CFB, OFB
- Implements both encryption and decryption
- 5 “true” clock domains
- Synchronous counterpart named “MERLIN”
## Area figures

<table>
<thead>
<tr>
<th></th>
<th>nominal cycle time</th>
<th>(sync.) module $[\mu m^2]$</th>
<th>wrapper area w/o sync $[\mu m^2]$</th>
<th>wrapper area with sync $[\mu m^2]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>controller</td>
<td>2.1ns</td>
<td>38 457</td>
<td>12 501</td>
<td>18 315</td>
</tr>
<tr>
<td>key prep</td>
<td>3.4ns</td>
<td>250 299</td>
<td>13 059</td>
<td>18 531</td>
</tr>
<tr>
<td>bias ROM</td>
<td>1.2ns</td>
<td>32 805</td>
<td>4 104</td>
<td>4 104</td>
</tr>
<tr>
<td>datapath</td>
<td>3.3ns</td>
<td>214 956</td>
<td>28 629</td>
<td>37 863</td>
</tr>
<tr>
<td>mux1</td>
<td>2.3ns</td>
<td>150 831</td>
<td>29 628</td>
<td>39 204</td>
</tr>
<tr>
<td>mux2</td>
<td>2.2ns</td>
<td>126 747</td>
<td>26 361</td>
<td>34 227</td>
</tr>
<tr>
<td>exp/log ROM</td>
<td>1.7ns</td>
<td>291 672</td>
<td>12 708</td>
<td>12 708</td>
</tr>
<tr>
<td>subkey RAM</td>
<td>–</td>
<td>237 415</td>
<td>17 784</td>
<td>17 784</td>
</tr>
<tr>
<td>async FIFO</td>
<td>–</td>
<td>34 182</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>area</strong></td>
<td><strong>1 377 364</strong></td>
<td><strong>144 774</strong></td>
<td><strong>182 736</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>100%</strong></td>
<td><strong>10.5%</strong></td>
</tr>
</tbody>
</table>
Design flow
MARILYN

- **Technology:**
  0.25µm, 5 metal, CMOS
- **Core:** 1.7 x 1.7 mm
- **Die:** 2 x 2 mm
- **66 pads (→ JLCC68 package)**
- **Contains extra testblocks for GALS components**
- **GALS overhead ≈ 10%**
- **Throughput 232Mbit/s at 10 rounds ECB**
- **Max. throughput up to 780Mbit/s (feedthrough)**
## Results

<table>
<thead>
<tr>
<th>CBC</th>
<th>ECB</th>
<th>Dec.</th>
<th>10 rounds</th>
<th>12 rounds</th>
<th>10 rounds</th>
<th>12 rounds</th>
<th>GALS benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB enc.</td>
<td>ECB</td>
<td>ECB</td>
<td>232 MBit/s</td>
<td>194 MBit/s</td>
<td>227 MBit/s</td>
<td>191 MBit/s</td>
<td>−23%</td>
</tr>
<tr>
<td>ECB dec.</td>
<td>ECB</td>
<td>ECB</td>
<td>303 MBit/s</td>
<td>255 MBit/s</td>
<td>303 MBit/s</td>
<td>255 MBit/s</td>
<td>−24%</td>
</tr>
<tr>
<td>ECB dec.</td>
<td>ECB</td>
<td>ECB</td>
<td>555 nJ</td>
<td>737 nJ</td>
<td>973 nJ</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>ECB dec.</td>
<td>ECB</td>
<td>ECB</td>
<td>577 nJ</td>
<td>733 nJ</td>
<td>965 nJ</td>
<td>21%</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• Complete methodology for GALS architectures developed
• Over 25% energy reduction (energy per MBit throughput)
• Area overhead below 10%
• Throughput up to 25% lower than synchronous version (due to datapath not running at full speed, reason currently being investigated)
• GALS building blocks:
  – Consist mainly of technology independent structural VHDL
  – Only mutual exclusion (ME) element requires cell design
• And most important: GALS works on silicon!
Future work

- Improve testability
- Extend the communication schemes beyond point to point links
- Address system level aspects like deadlock analysis and system partitioning
- Improve tool flow support (hierarchical flow, timing verification, integration of asynchronous tools)
- Possibility to withdraw a data transfer request (bus deferral)
- Power estimation theory
- Finer time-slice resolution of ring-oscillator