Design and Implementation of a GALS Adapter for ANoC based Architectures

Yvain Thonnart, Edith Beigné, Pascal Vivet
MINATEC, CEA-LETI, Grenoble, France
ASYNC’09, Chapel Hill, NC, USA
May 18th, 2009
GALS Adapter for ANoC

- Asynchronous NoC
  - 2D-Mesh based,
  - Wormhole Source Routing,
  - 2 Virtual Channels for QoS,
  - QDI (4-rail, 4-phase)

- Low Power scheme:
  - DVFS is used at IP level

- GALS scheme:
  - IP are distinct frequency domains
  - synchronous Off-chip NoC interfaces

→ Need efficient GALS interfaces
Outline

- Introduction
- ANoC GALS adapter architecture
- New bi-synchronous FIFOs using Johnson code
- Design of ANoC Interfaces
- Implementation & Results
- Conclusion
Outline

- Introduction
- ANoC GALS adapter architecture
  - Objectives & Principles
- New bi-synchronous FIFOs using Johnson code
- Design of ANoC Interfaces
- Implementation & Results
- Conclusion
ANoC GALS Adapter Objectives

- **Design Objectives**
  - QDI asynchronous (4-rail/4-phase) vs Synchronous NoC protocols
  - Virtual Channel policy (VC0 / VC1)
  - Local Clock Generator for easy frequency decoupling
  - *High throughput, low latency, low area*

- **Implementation Objectives**
  - Standard-Cell based Design
  - Delivered as a Hard Macro for easy design flow integration
  - *Hide asynchronous complexity to the final user*
ANoC GALS Adapter Architecture

- Efficient A-S and S-A FIFOs based on Johnson encoding
  - Protocol conversion between QDI & Synchronous logic
  - Virtual Channel multiplexing / demultiplexing

- Programmable Local Clock Generator
  - Using a standard-cell based delay-line
  - Robust & efficient programming interface using pausable clocking
Outline

- Introduction
- ANoC GALS Adapter Architecture Proposal

- New bi-synchronous FIFOs using Johnson encoding
  - Metastability issues in FIFOs
  - Johnson Code
  - FIFO Micro-Architecture

- Design of ANoC Interfaces
- Implementation & Results
- Conclusion
Dual clock FIFO principles

- Decoupled timing domains
  - Write pointer updated with the write clock
  - Read pointer updated with the read clock
  - High throughput: 1 transfer / cycle
  - Latency & minimal FIFO depth depend on the synchronization costs

- Synchronization issue?
  - pointers are cross timing domains ➔ need synchronization with opposite clock
  ➔ *Needs ad-hoc encoding to ensure proper detection of full and empty states*
From Gray encoding to Johnson encoding

- **Hamming distance of 1**
  - always synchronizes either last value or new value

- **Disadvantages of Gray**
  - Increment & comparison operations are not trivial
    - needs to be done in binary, needing converters
    - or with consequent translation tables
  - Natural Gray is limited to $2^N$ FIFO depths: area consuming
    - Gray adaptations to non $2^N$ depths need complex logic

- **Johnson encoding is to Gray what 1-hot is to Binary**
  - Also has a Hamming distance of 1
  - Less dense, not restricted to $2^N$ values
  - Increment & comparisons are trivial
Johnson counter

- Also known as “twisted ring”
  - A single bit change propagates from LSB to MSB
  - Implemented by a shift register looped by an inverter

- Increment and comparison are trivial

- N bit Johnson counter encodes 2N values

[Johnson 74]
Johnson encoding for FIFO design

- Use a N-bit Johnson counter (2N values) for a FIFO of depth N
  - Each bit defines a FIFO register
  - The bit value is the access parity for that given register

- Control logic equations
  - Wen for register i:
    \[ \text{Wptr}[i] \text{ xor } \text{Wptr\_incr}[i] \]
  - Ren for register i:
    \[ \text{Rptr}[i] \text{ xor } \text{Rptr\_incr}[i] \]
  - Empty: \( \text{Rptr} = \text{Wptr} \)
  - Full: \( \text{Rptr} = \text{NOT Wptr} \)

<table>
<thead>
<tr>
<th>Johnson Code</th>
<th>Register id</th>
<th>Parity</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>00001</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>00011</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>00111</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>01111</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>11111</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11110</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11100</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>11000</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>10000</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Detect code transition to access register i

Full condition?
no cell lost in the FIFO to handle Full state:
Read and Write on same register
Johnson encoding FIFO architecture

- The FIFO is cross timing domains between distinct synchronous or asynchronous domains

- R/W pointer synchronization?
  - using 2 (or more) Flip-Flops for synchronous domains
  - using glitchless logic for asynchronous side of the AS/SA interfaces
Outline

- Introduction
- ANoC GALS Adapter Architecture Proposal
- New bi-synchronous FIFOs using Johnson encoding

- Design of ANoC Interfaces
  - A-S Interface & S-A Interface
  - Local Clock Generator

- Implementation & Results
- Conclusion
A-to-S interface

**Design constraints**
- write clock is locally generated (output of the Input Data word completion tree)
- write clock edge must occur after 4rail-BD conversion
- FIFO size must be at least 5 due to AS interface round trip

Full / Empty detector for A-S interface
S-to-A interface

Design constraints
- read clock is locally generated (output of the asynchronous data ack. completion tree)
- Asynchronous token generation must occur after BD-4rail conversion
- FIFO size must be at least 5 due to SA interface round trip
Local Clock Generator

- **Use Pausible Clock Principle**

**Design constraints**
- AFSM for controlling the req/ack pause signals for reprogramming
- Linear delay increment for total delay line delay
- Clock division for lower frequencies for DVFS and test
- Reprogramming is only done during low phase of the clock (all signals are stable equal to 0)
- Full standard-cell implementation

**Latch-Mux based programmable delay line**
Outline

- Introduction
- ANoC GALS Adapter Architecture Proposal
- New bi-synchronous FIFOs using Johnson encoding
- Design of ANoC Interfaces

- Implementation & Results

- Conclusion
Design Flow

- **GALS adapter implementation**
  - Full standard-cell (CORELIB + TAL)
  - Mixed RTL / gate instantiation
  - Synthesis with definition of the 3 clock domains
  - Max-delay constraints on cross-domain paths
  - Standard place&route design flow
  - Validation on SDF back-annotated netlist in all corners
  - Delivered as a hard-macro with all CAD views

- **GALS adapter usage**
  - Use the hard-macro as a synchronous IP with CTS + .lib files

- Easy top-level integration
  - Hiding asynchronous complexity to the final user
- Provides full timing domain decoupling
  - Using a GALS NoC scheme
Hard-Macro Layout (STMicroelectronics CMOS65LP)
ANoC GALS Adapter Performances

Nominal Case, 1.20V, 25°C

Worst Case, 1.05V, 105°C

<table>
<thead>
<tr>
<th></th>
<th>Frequency Range (Delay Line Prog)*</th>
<th>A-to-S Throughput</th>
<th>S-to-A Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst-Case</td>
<td>200MHz – 500MHz</td>
<td>400MHz</td>
<td>280MHz</td>
</tr>
<tr>
<td>Nominal-Case</td>
<td>380MHz – 980MHz</td>
<td>680MHz</td>
<td>510MHz</td>
</tr>
</tbody>
</table>

(*) plus additional values through division factors

- Using QDI, ANoC and the GALS adapter will provide about 500MFlit/s.
- The Delay Line will provide local clock frequency up to 1GHz

(Less precision for higher frequencies)
## Comparison with previous solutions

<table>
<thead>
<tr>
<th>Design</th>
<th>Max clock freq (MHz)</th>
<th>Max A-S throughput (Mflit/s)</th>
<th>Max S-A throughput (Mflit/s)</th>
<th>Layout area (µm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAUST(*)</td>
<td>350</td>
<td>300</td>
<td>250</td>
<td>34,000</td>
</tr>
<tr>
<td>(2005)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALPIN</td>
<td>400</td>
<td>220</td>
<td>180</td>
<td>9,000</td>
</tr>
<tr>
<td>(2007)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGALI</td>
<td>980</td>
<td>710</td>
<td>520</td>
<td>12,500</td>
</tr>
<tr>
<td>(this work)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FAUST [Async’06] : use of FIFO interfaces based on Gray code  
ALPIN [NOCS’08] : use of Pausable clocking

(*) 130nm values converted to 65nm
Outline

- Introduction
- ANoC GALS Adapter Architecture Proposal
- New bi-synchronous FIFOs using Johnson encoding
- Design of ANoC Interfaces
- Implementation & Results

- Conclusion
Lessons from the past...

- Proposed to (re-)use Johnson encoding [Johnson74]
  - Allow efficient FIFO design & any FIFO depth compared to std Gray code
  - The FIFO can be sized to the minimum ➔ area reduction

- Pausable clocking?
  - Provides very low area overhead at the cost of very low bandwidth
  - Limits the maximum clock frequency [Dobkin, Ginosar 05]
  - Well suited to low-power solutions, not for high performance
  - Nevertheless efficient to provide robust interface for DFS scheme

- NoC Virtual Channels are costly for GALS interfaces
  - Do not demux the VCs in the GALS interface when VCs are not interleaved within the synchronous unit
  - Share all you can!

- GALS Design Flow
  - Provide the GALS interface as a Hard Macro with all CAD views
  - For easy integration at top level