Embedded Systems 1: On-Chip Bus

Davide Zoni PhD
email: davide.zoni@polimi.it
webpage: home.deib.polimi.it/zoni
Additional Material and Reference Book

- Reference Book Chapter
  - “Principles and Practices of Interconnection Networks”, Dally 2004 Chapter 22

- On-Chip Bus Specification
  - AMBA Bus
    - Developed by ARM
    - Coherence support through the AMBA Coherence Extensions (ACE)
    - Implements split transactions and burst mode
  - WISHBONE
    - http://opencores.org/opencores,wishbone
    - No copyright and public use
    - Supported by OpenCores
    - No Split Transaction, No Coherence Support
    - Extensively exploited in Open Hardware Projects, i.e. OpenRisc
On-chip Bus: Basic Concepts

- The Bus Specification Document:
  - Topology
  - Architectural Implementation
  - Data Transfer Protocol
  - Arbitration Scheme
  - Extensions

- A compliant bus implementation has to satisfy the bus specification apart from the optional/extension components
On-chip Bus: Bus Signals

- **Address Bus**
  - Routes the destination address provided by the master
  - It can be shared with the data bus
- **Data Bus**
  - Transfers data to/from master and slave during a bus transaction
  - It can be shared with the data bus
- **Control Bus**
  - It carries out all the protocol specific signals to steer the bus transactions, i.e. read/write signals, ack errors, data validity
- **NOTE:** We use a simplified protocol spec
  - no explicit address/control bus
  - Ack to the data bus
On-chip Bus: Architecture

- Tri-state: mainly used for off-chip communication
  - low pin count design constraint
  - High delay, high power
  - Custom logic cells
- Mux and AndOr:
  - more efficient
  - Standard cell logic is required
**On-chip Bus: Components**

- **Master**: each module that can independently trigger a bus transaction
- **Slave**: each module that answers to a bus transaction started from others
- **Decoder**: decodes the destination address and activate the proper source-destination path, i.e. master-slave
  - Tightly coupled with the arbiter
  - Distributed and centralized implementations are possible
- **Arbiter**: grants the bus access to the requesting master avoiding deadlock starvation and collisions
  - The granted master depends on the implemented arbitration policy (static, dynamic)
- **Bridge**: Interfaces two bus architectures that differs because of the protocol, technological parameters
On-chip Bus: Simple Data Transfer

- **Master Signals**
  - Bus Request, Command, address
  - Data Input, Data Output (for read and write data)

- **Arbiter**
  - Collects the bus requests and commands
  - Outputs a grant signal

- **Slaves**
  - Data Input Data Output (provide data, signal ack/nack), validity bit
On-chip Bus: The Simplified Bus Model

How to Optimize the Performance

- Increase the frequency or Reduce the Pipeline Depth
- Optimize the bus utilization between concurrent transactions

The Pipelined Bus

- Address and Data transfers are partially overlapped
- Additional arbitration cycle (Dally’s book compliant)
- Shared/Exclusive Cycles
  - Gray: shared ops
  - Black: exclusive bus usage
- No control bits bu the R/W command and the ACK

### Write Transaction

<table>
<thead>
<tr>
<th>Cycles</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ArbReq
- Arb
- Grant
- Bus

### Read Transaction

<table>
<thead>
<tr>
<th>Cycles</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ArbReq
- Arb
- Grant
- Bus

[Table diagrams showing the pipeline stages for write and read transactions]
On-chip Bus: Starting Point Simplifications

The Pipelined Bus

- The required memory cycles are fixed (IDEAL)
- The read/write transaction shape is known
- The Bus Transaction is pipelined
- Once the transaction receives the grant the arbiter has no more control over it
- The arbiter grants based on the active transactions, since the transaction shape is known (IDEAL)
- The exact number of memory cycles to provide an answer cannot be foreseen in the real world

Three optimizations to increase the bus utilization and reduce the simplifications

- Split Transaction Support
- Burst Transaction Support
- Out of Order Transaction Support
Optimize the bus usage

Each transaction is split in two parts: request

Both requests and responses must arbitrate first

This optimization removes the need to know in advance the number of memory cycles per each transaction
On-chip Bus: Burst Mode Optimization

**Cache line and bus width differs**
- Bus width optimized for control packets
- Cache line depends on the hierarchy and the whole architecture
- In general bus width < cache line

**L1-L2 and L2-Mem exchanged data are cache lines not data words.**

The burst transaction mode allows to arbitrate the bus once to send multiple data chunks
Improves the split transaction mode

- The same master can issue multiple transactions before the pending ones complete
- The slave can reorder the responses to the masters to improve its own metrics

To take advantage of this optimization

- The CPU should be able to issue multiple load stores
- The cache, if any, must be non blocking

We don’t have exercises with such an optimization since such optimization stresses more other architectural modules, i.e. cache hierarchy, CPU, instead of the bus.