VirtualSoC
HW Platforms

MicrelLab - DEI
Università di Bologna

version 0.1

Abstract

This document is intended to give an overview of the HW platforms and configurations of VirtualSoC.

Contacts
daniele.bortolotti@unibo.it

Cluster Components

In this section we give a brief overview of the cluster components, please consider Figure **??** for clarity.

Cores : The shared L1 cluster consists of a configurable number of 32-bit ARMv6 processor.
Interconnect : The logarithmic interconnect module has been modeled, from a behavioral standpoint, as a parametric, Mesh-of-Trees (MoT) interconnection network (see Figure **??**). The module is intended to connect processing elements to a multi-banked memory on both data and instruction side. Data routing is based on address decoding: a first-stage checks if the requested address falls within the intra-cluster memory address range or has to be directed off-cluster. To increase module flexibility this stage is optional, enabling explicit L3 data access on the data side while, on the instruction side, can be bypassed letting the cache controller take care of L3 memory

accesses for lines refill. The last $log_2(interleaving\ size)$ bits of the address determine the destination. The crossing latency consists of one clock cycle. In case of multiple conflicting requests, for fair access to memory banks, a round-robin scheduler arbitrates access and a higher number of cycles is needed depending on the number of conflicting requests, with no latency in between. In case of no banking conflicts data routing is done in parallel for each core. Read broadcast has been implemented and no extra cycles are needed when broadcast occurs.

TCDM : On the data side, a multi-ported, multi-banked, Tightly Coupled Data Memory (TCDM) is directly connected to the logarithmic interconnect. The number of memory ports is equal to the number of banks to have concurrent access to different memory locations. Once a read or write requests is brought to the memory interface, the data is available on the negative edge of the same clock cycle, leading to two clock cycles latency for conflict-free TCDM access. If conflicts occur there is no extra latency between pending requests, once a given bank is active, it responds with no wait cycles.

L1 Instruction Cache Module : The Instruction Cache Module has a core-side interface for instruction fetches and an external memory interface for refill. The inner structure consists of the actual memory (TAG + DATA) and the cache controller logic managing the requests. The module is configurable in its total size, associativity, line size and replacement policy (FIFO, LRU, random).

L3 memory : Each cluster has its own (L3) main memory. Depending on the chosen I$ (private or shared) the number of requestors to the L3 memory can vary. Accesses to the memory are coordinated by a simple L3 BUS, concurrent requests are served in a round robin fashion.

Synchronization : To coordinate and synchronize cores execution, we modeled two different synchronization mechanisms. The first one consists of *HW semaphores* mapped in a small subset of the TCDM address range. They consist of a series of registers, accessible through the data logarithmic interconnect as a generic slave, associating a single register to a shared data structure in TCDM. By using a mechanism such as a hardware *test&set*, we are able to coordinate access: if reading returns '0', the resource is free and the semaphore automatically locks it, if it returns a different value, typically '1', access is not granted. Theoretically all cores can be resumed at the same time (reading broadcast the value of the semaphore), but there is no guarantee that this happens because of execution misalignment. To get tight execution alignment, we developed two *fast synchronization primitives*

based on a *HW Synchronization Handler Module* (SHM). This device acts as an extra slave device of the logarithmic interconnect and has a number of hardware registers equal to the number of cores, where each register is mapped in a specific address range. When a write operation is issued to a given register, a synchronization signal is raised to the corresponding core suspending its execution after one cycle, when the synchronization signal is lowered the execution is resumed. The SHM is accessible in different ways from the software level via APIs (see `APPSUPPORT.pdf`).
DMA : The DMA engine will be available in a forthcoming release.

Clocks in the System

The HW platforms described above makes use of different clocks. The motivation behind this choice is to correctly model instructions and data latencies from the ARM11 wrapper point of view (see `WRAPPING.doc` for a description of its internal behavior). The protocol involved in a transaction, based on `request` and `ready` signals, introduces an intrinsic extra latency for data of one clock cycle. The data can be an instruction opcode from the I$, or a value from TCDM. To obtain the goal of instruction fetched (hit) in one clock cycle from the I$ (i.e. requested at cycle *i-th* and read at cycle *(i+1)-th*), we clocked the I$ banks at a double frequency, thus working around the intrinsic delay. The same was done for the TCDM banks and for the shared I$ banks. The same mechanism was applied where necessary, to reduce the intrisic delay of modules and by eventually applying the some well-known delay to meet the timing requirements.

Single Cluster Platforms

For instructions on how to simulate the single-cluster platform, please refer to the documentation in `SIMULATOR.pdf` located in `$SWARDMDIR/../doc`.

Private I$ Architecture

All the previously described architectural elements are combined together to form the private instruction cache architecture as shown in Figure **??**. The cluster is made of 16 ARMv6 cores, each one has its own private instruction cache with separate line refill paths while the L1 data memory is shared among them. An optional DMA engine[1] can be used to carry out L3 to TCDM data transfers. Access to the off-cluster L3 memory is coordinated by the L3 BUS, requests are served in a round-robin fashion. On the data side all cores are able to perform access to TCDM, L3 memory and eventually to HW semaphores or SHM. The logarithmic interconnect is responsible of data routing based on address ranges.

Shared I$ Architecture

Shared instruction cache architecture is shown in Figure **??**. From the data side there is no difference between the private architecture except for the reduced contention for data requests to L3 memory (line refill path is unique in this architecture).

Shared cache inner structure is represented in Figure **??**. A slightly modified version of the logarithmic interconnect connects processors to the shared memory banks operating line interleaving (1 line consists of 4 words). A round robin scheduling guarantees fair access to the banks. In case of two or more processors requesting the same instruction, they are served in broadcast not affecting hit latency. In case of concurrent instruction miss from two or more banks, a simple MISS BUS handles line refills in round robin towards the L3 BUS.

[1] the DMA engine will be available in a forthcoming release

Multi-Cluster Platform

For instructions on how to simulate the single-cluster platform, please refer to the documentation in `SIMULATOR.pdf` located in `$SWARDMDIR/../doc`.

The multi-cluster platform consists of 4 clusters globally interconnected through a 2x2 NoC (Figure **??**). Each cluster can be configured in the same parameters (also its I$ architecture) as in the single cluster version. The logarithmic interconnect is slightly different, enabling inter-cluster communication, if an issued address falls outside the cluster address range (each cluster has a dedicated section of the global address space as explained in the next sections). Two extra modules (`bridges`) are required to operate protocol translation from the cluster-wide protocol (`PINOUT`) to the system-wide, i.e. NoC level, protocol (standard `OCP`).

Local Address Space

Address space management, how the different components are addressable by the processors in the system, is a key part of the overall simulator structure. The ARMv6 core has a 32-bit address field, addressing up to 4 GB. Every single cluster is associated with an address space (TAS - *Tile Address Space*) of `0x10000000` (256 MB). In Figure **??** is shown a representation of the address space of a single tile. The first hexadecimal digit identifies one among the 16 possible tiles (in case of a 4x4 platform), so this figure is related to tile with ID equal to 0.
The size of the components in a cluster are statically defined inside simulator's source code. Default configuration is as follows:

```
FILE: "$SWARMDIR/core/config.h"
...
TILE_SPACING            0x10000000  (256 MB)
PLR_L3_BASE             0x00000000
PLR_L3_SIZE             0x08000000  (128 MB)
PLR_TCDM_BASE           0x08000000  (128 MB)
PLR_TCDM_SIZE           0x00040000  (256 KB)
PLR_BANK_SPACING        PLR_BANK_SIZE
PLR_SEM_BASE            0x09000000  (512 MB)
PLR_SEM_SIZE            0x00004000  (16 KB)
...
```

Global Address Space

Every tile has its own local address space and all tiles together make up a non overlapping, globally visible address space (PGAS), see Figure **??**.
Considering the architecture depicted in Figure **??**, a 2x2 NoC platform where every Network Interface is connected to its tile, identified by its own TID (Tile ID). The first hexadecimal digit is used to identify a single tile, while other digits are meaningful at local level and interpreted by the logarithmic interconnect to connect processors with local slave devices.
For example if a core inside TILE 0 wants to access TCDM memory inside TILE 3, reading at address `0x3800AB00`, logarithmic interconnect of TILE 0 will forward to the Network Interface (after a proper protocol conversion),

NoC routers will route the transaction to TILE 3 and then, logarithmic interconnect of TILE 3 will grant access to the local TCDM. This situation is depicted in Figure **??**.