VirtualSoC
Application Support Library

MicrelLab - DEI
Università di Bologna

version 0.1

Abstract

This document is intended to give a brief description of all
support functions available to user applications and runtime library
as well as an overview of the DMA APIs

Contacts
daniele.bortolotti@unibo.it[1]

Memory Map: LOCAL_SHARED

A new section of memory, named LOCAL_SHARED, has been added to the linker script to enable explicit allocation of data in TCDM. This can be done by appending the macro LOCAL_SHARED to the declaration of a variable:

```
int a LOCAL_SHARED;
```

Data statically assigned to the LOCAL_SHARED section, will be loaded on top of TCDM banks at simulator startup.
The linker script used for compiling applications is located in $VSOC_APP_DIR/support/simulator/ and the file is named vsoc.ld.

[1] please use as subject of your email : [VirtualSoC] Support

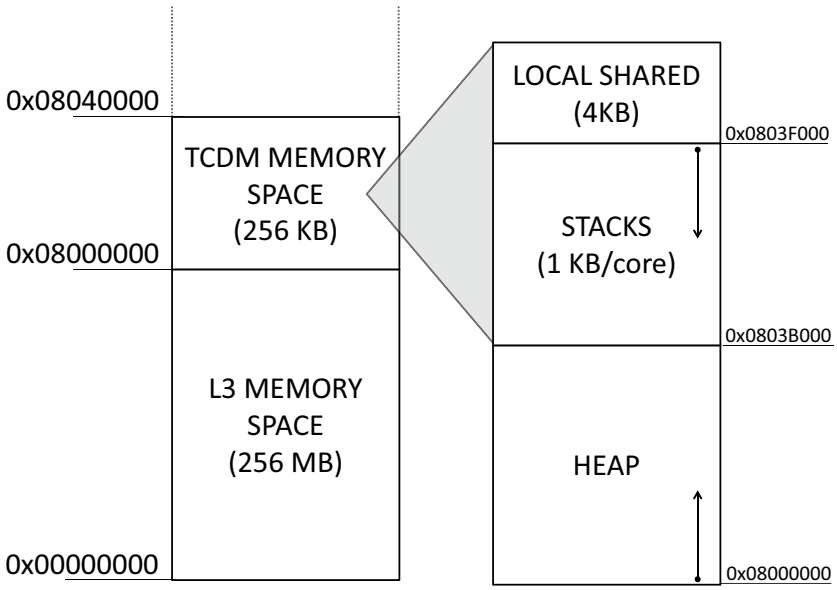

Figure 1: VirtualSoC cluster memory map

Appsupport

The `SimSupport` module of VirtualSoC offers a set of services (such as getting each core's ID) useful for the simulation to the user through the APIs offered by the `appsupport`. Its mechanism is based on read or write issued in a given address range: all the transactions within this range are intercepted by the SystemC wrapper and no traffic occurs at bus level.

All the support functions are defined in file `appsupport.c`, located in `$VSOC_APP_DIR/support/simulator`.

Memory-mapped support functions

- `void pr(char *msg, unsigned long int value, unsigned long int mode)`

  Allows printing debug info even without OS support. Input parameters:

  - `char *msg` : the string to be printed;
  - `unsigned long int value` : the numerical value to be printed;

- `unsigned long int mode` : allows printing some information and to format the output. The `mode` is set by a series of macros like `PR_CPU_ID` (to enable printing processor's ID), `PR_STRING` (to enable printing che string, otherwise no string will be printed), `PR_DEC` (to print the `value` in decimal format), `PR_HEX` (to print the `value` in hexadecimal format, only one among `PR_DEC` and `PR_HEX` muste be used), `PR_NEWL` (to have a newline right after the printed line), `PR_TSTAMP` (to enable printing current simulation time). These macros are combined together in an OR fashion.

Example of usage:

- ```
  pr("hello", 0x10, PR_CPU_ID | PR_STRING |
                    PR_HEX | PR_NEWL)
  ```
  will give you as output:
  ```
  Processor 0 - hello 0x10
  ```
- ```
  pr("dummy string", 0x10, PR_CPU_ID | PR_DEC |
                           PR_TSTAMP | PR_NEWL)
  ```
  will give you:
  ```
  Processor 0 - 16 @ 4350 ns
  ```

Assuming Core 0 is using the support library.

- `unsigned int get_proc_id()`

  Allows getting global (system-wide) processor ID (from 1 onwards).

- `unsigned int get_proc_num()`

  Allows getting global (system-wide) number of processors in the platform.

- `unsigned int get_tile_id()`

  Allows getting global (system-wide) cluster ID.

- `unsigned int get_proc_loc_id()`

  Allows getting local (cluster-wide) processor ID.

- `unsigned int get_proc_tile_num()`

  Allows getting local (cluster-wide) number of processors.

- `void start_metric()`

  Starts statistic collection for a processor.

- `void stop_metric()`

  Stops statistic collection for a processor.

- `void stop_core()`

  The calling processor goes idle.

- `unsigned int get_argc()`

  Allows getting the `argc` command line parameter.

- `char **get_argv()`

  Allows getting the `argv` command line parameter.

- `char **get_envp()`

  Allows getting the environment.

- `unsigned long long int get_time()`

  Allows getting the current simulation time.

- `void opt_get_time()`

  Optimized version (asm).

- `unsigned long long int get_cycle()`

  Allows getting the current simulation cycle.

- `void opt_get_cycle()`

  Optimized version (asm).

- `unsigned long int access_file(char *filename,`
  `unsigned long int mode)`

  Allows reading or writing one file on the simulation host.

DMA APIs

All the DMA APIs are defined in file `dmasupport.c`, located in `$VSOC_APP_DIR/support/simulator`.

NOTE: The DMA module is not fully tested in the multicluster architecture.

- ```
  unsigned char dma_prog(
        unsigned char id,
        unsigned int addr1,
        unsigned int addr2,
        unsigned int size,
        unsigned char direction,
        unsigned char async,
        unsigned char sleep,
        unsigned char trigger);
  ```

  Allows programming a job in the DMA (local to the cluster). Input parameters:

  - `id` : core's id;
  - `addr1` : address 1 (source or dest);
  - `addr2` : address 2 (source or dest);
  - `size` : size of the transfer (Bytes) - needs to be aligned;
  - `direction` : sets direction of transfer : if '1' from addr1 to addr2, '0' viceversa;
  - `async` : sets asynchronous ('1') or synchronous ('0') copy;
  - `sleep` : when '1', suspend core after programming dma, wake up when transfer is completed. Based on event signals;
  - `trigger` : starts transfer right after programming the dma ('1') or later ('0');

  The API returns the `job id` (`unisgned char`) to be used to collect job's status.

- `void dma_start(unsigned int job_id);`

  In case the job was programmed without starting the transfer right after (`trigger = 0`), it can be triggered with this API.
  ```

- `unsigned char dma_wait(unsigned int job_id);`

  API to collect job completion ('1' job done). Two behavior according to how the job was programmed (`job_id` distinguishes behavior):

  - `0 <job_id <15` : no sleep-mode (busy waiting - polling)
  - `16 <job_id <31` : sleep-mode (idle mode and event signal)