CGRA structure

This pages describes the structure of the CGRA.

The compute module

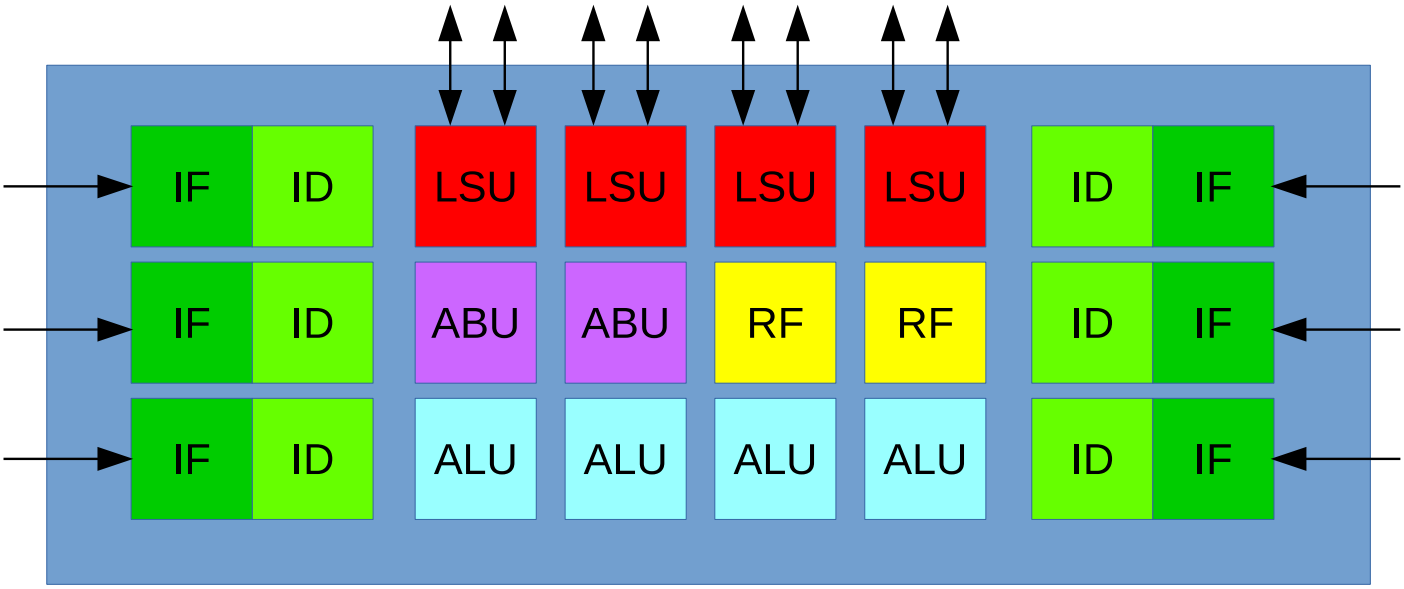
The most fine grained modules used in the CGRA are the functional units (FUs), the following FUs are available:

- Load Store Unit (LSU)
- Register File Unit (RF)
- Arithmetic Logic Unit (ALU)
- Immediate Unit (IU)
- Accumulate and Branch Unit (ABU)
- Multiplier Unit (MUL)

Besides these units there are Instruction Decoders (IDs), Instruction Fetch units (IFs) and switchboxes (swb).

These units are instantiated based on the atchitecture description, a XML file describing the FUs available, their properties and interconnect. The interconnect can be either fixed (specified at design time in the XML file) which we call ‘static’ or reconfigurable, which we call dynamic. Dynamic CGRAs use switchboxes to make connections between functional units and between IDs and FUs. By doing so processors can be constructed, either at design time with a static CGRA or run-time with a dynamic CGRA.

The FUs, IDs, IFs and switchboxes are contained in the ‘compute module’. An example of such a compute module is shown in the figure below, please note that this is not meant thos show a particular instantiation and merely aims to show the structure of the CGRA. The switchboxes are not shown for clarity.



Picture source

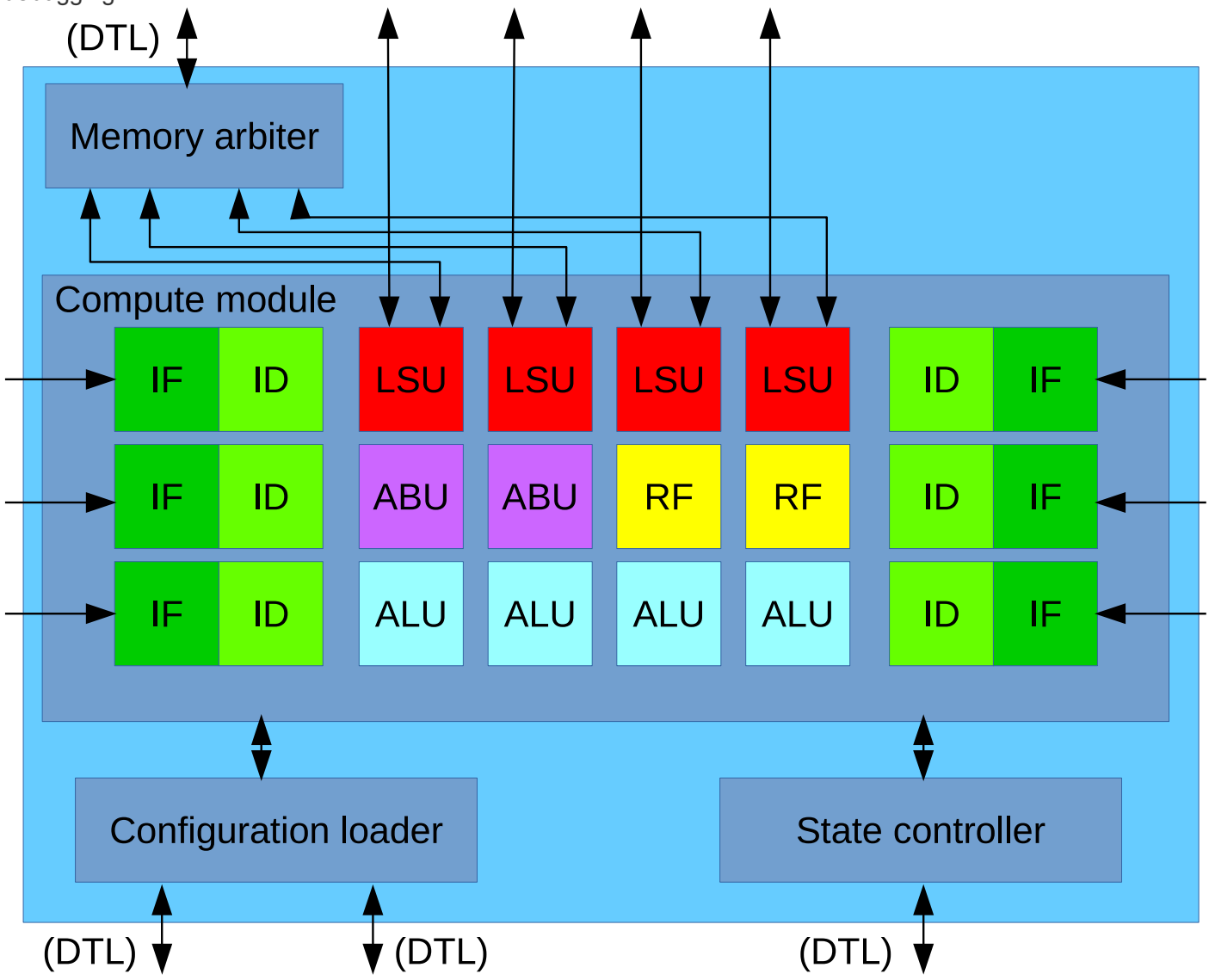
The arrows in the figure above indicate interfaces to a higher hierarchical level, for the LSUs these are interfaces to local or global memory. For the IF units the arrows indicate a interface to the instruction memory.

The compute module also contains scan-chains for loading the CGRA configuration and for reading or writing the processor state. The control wires are available to the higher hierarchical level, the compute-wrapper module.

The compute-wrapper module

The compute-wrapper module contains the compute module of the CGRA and adds some ‘administrative’ functionality. The independent global memory connections for each of the LSUs are connected to an arbiter which manages access to the global memory bus. The local memory connections pass through this module since they do not need to be arbited, as do the instruction memory connections to the IFs. Configuration loading is managed by the ‘configuration loader’, this module also contains some status and control registers which are described in CGRA external interfaces. The ‘configuration loader’ also takes care of filling the instruction memories with the operations defined in the binary. Another, optional, module is the state controller. This module allows reading and writing of the entire state of the compute module, this can be useful for multi-threading or

debugging.



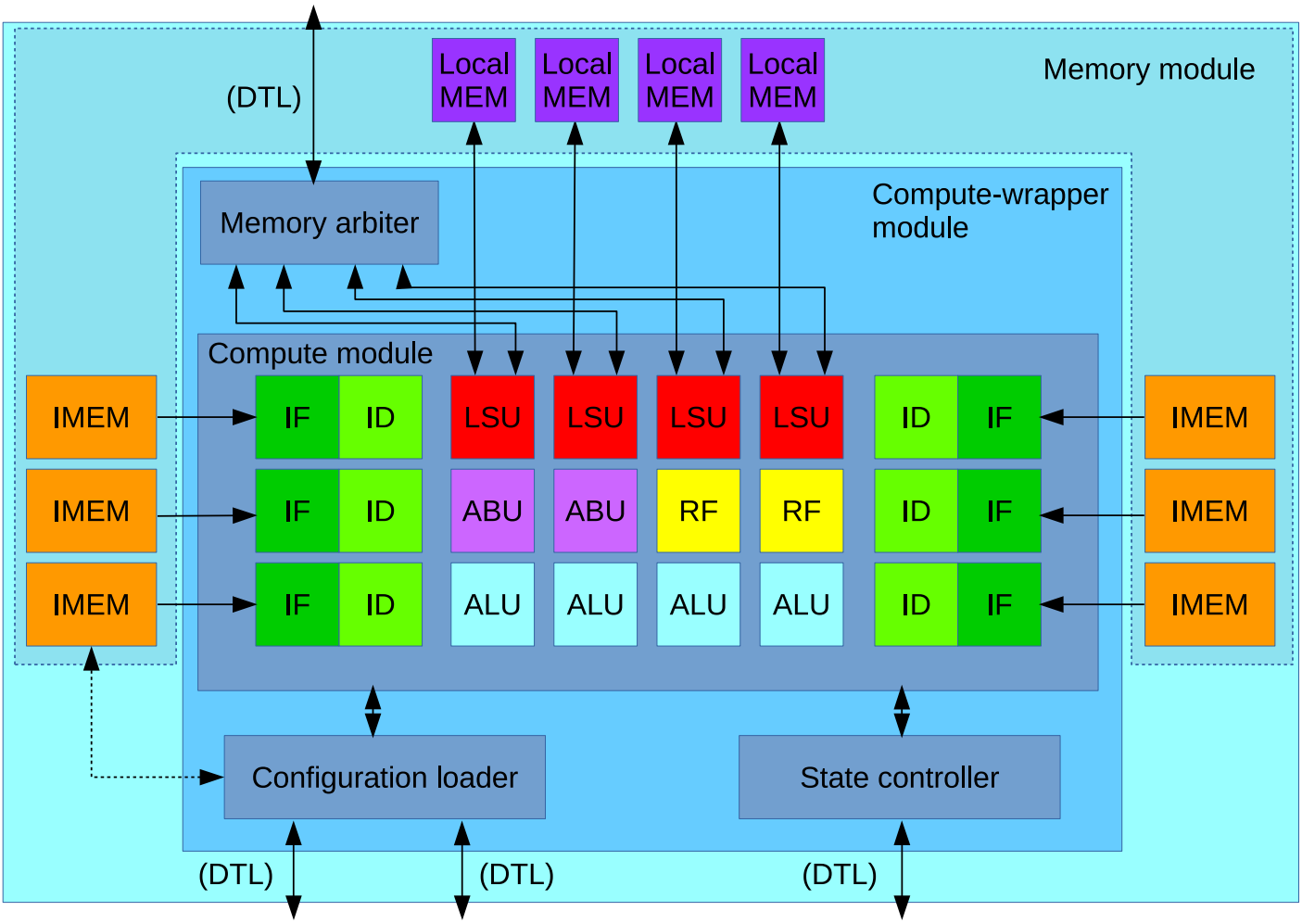
Picture source

The connections for the data memory, loader and state controller are all DTL (Device Transaction Layer) busses, essentially a simplified AXI bus. These busses are compatible with the CompSoC platform but are also very convenient to connect to other devices via DTL to AXI translators for example. The DTL port for the state controller is an optional port and will only be instantiated when the global define 'INCLUDE_STATE_CONTROL' is defined.

The connections between the local memories and the LSUs, as well as the instruction memories and the IFs are simple direct memory interfaces. These memories are not contained within the compute-wrapper module since the ASIC toolflow typically has no memory generator. **Therefore the compute-wrapper module is the top-level module for ASIC synthesis.**

The core module

The core module combines the compute-wrapper module with the local data memories and instruction memories. These memories are contained in a separate module, very imaginatively called, the 'memory module'. Besides adding the memories to the compute-wrapper module, the core module does not introduce any new functionality. The DTL busses simply pass through this module.



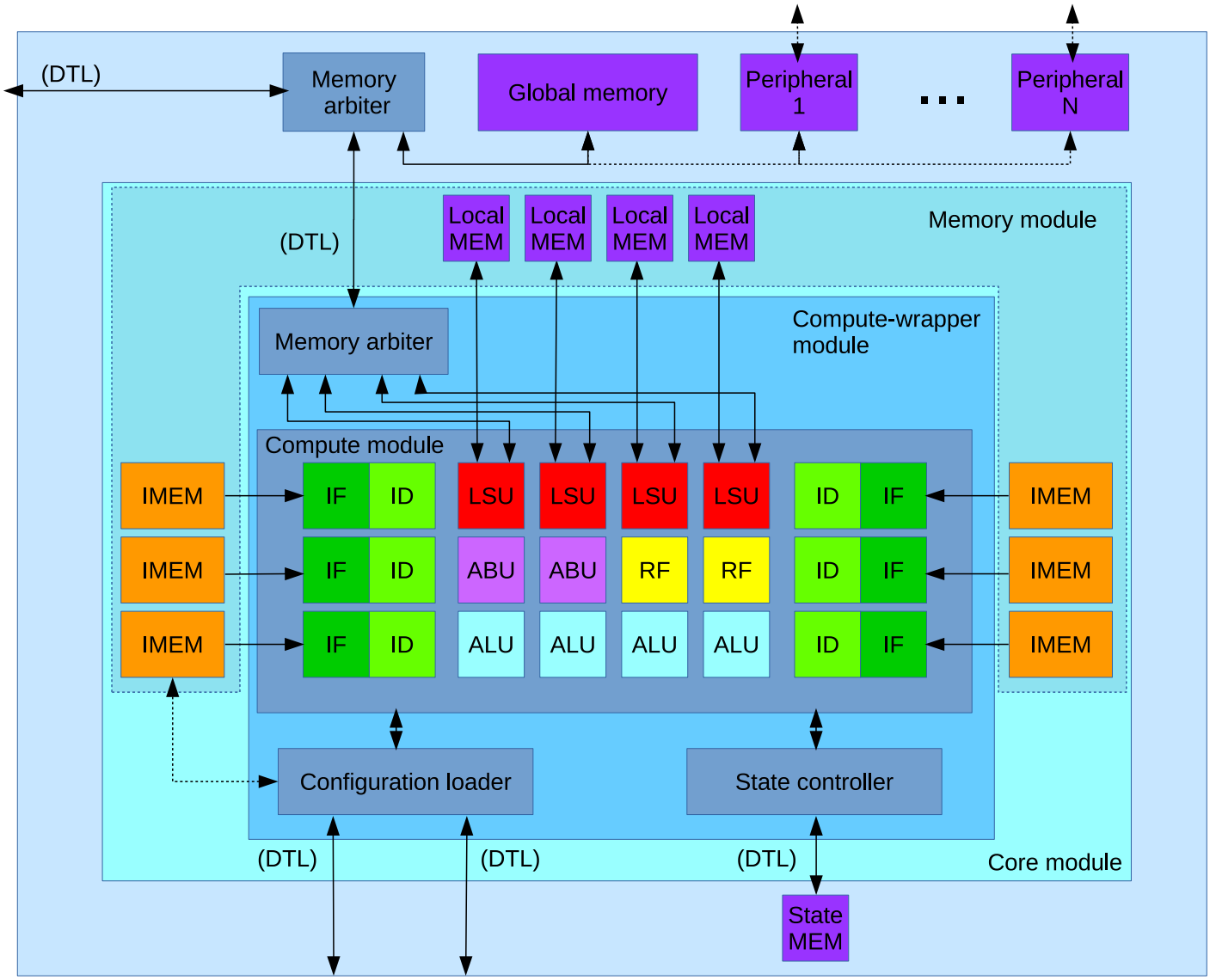
Picture source

The core module is a ready to use CGRA block that can be included into FPGA designs (or the CompSoC platform), the DTL ports have to be connected to the host processor and the required external memories, more about this in Using the CGRA.

The top module

The top module is a (almost) standalone version of the CGRA and is mostly used for simulation, all required memories such as the global and state storage memory and peripherals are contained in this module. The oly memory not contained in this module (because it is assumed to be supplied by a external system) is the memory where the application binary resides. The top module assumes a simple hardware initiator that sends a pointer to the address in the ‘binary memory’ where the application binary resides. Peripherals specified in the architecture XML file will be instantiated in the top module, any required inputs and outputs will be added based on the archtecture description.

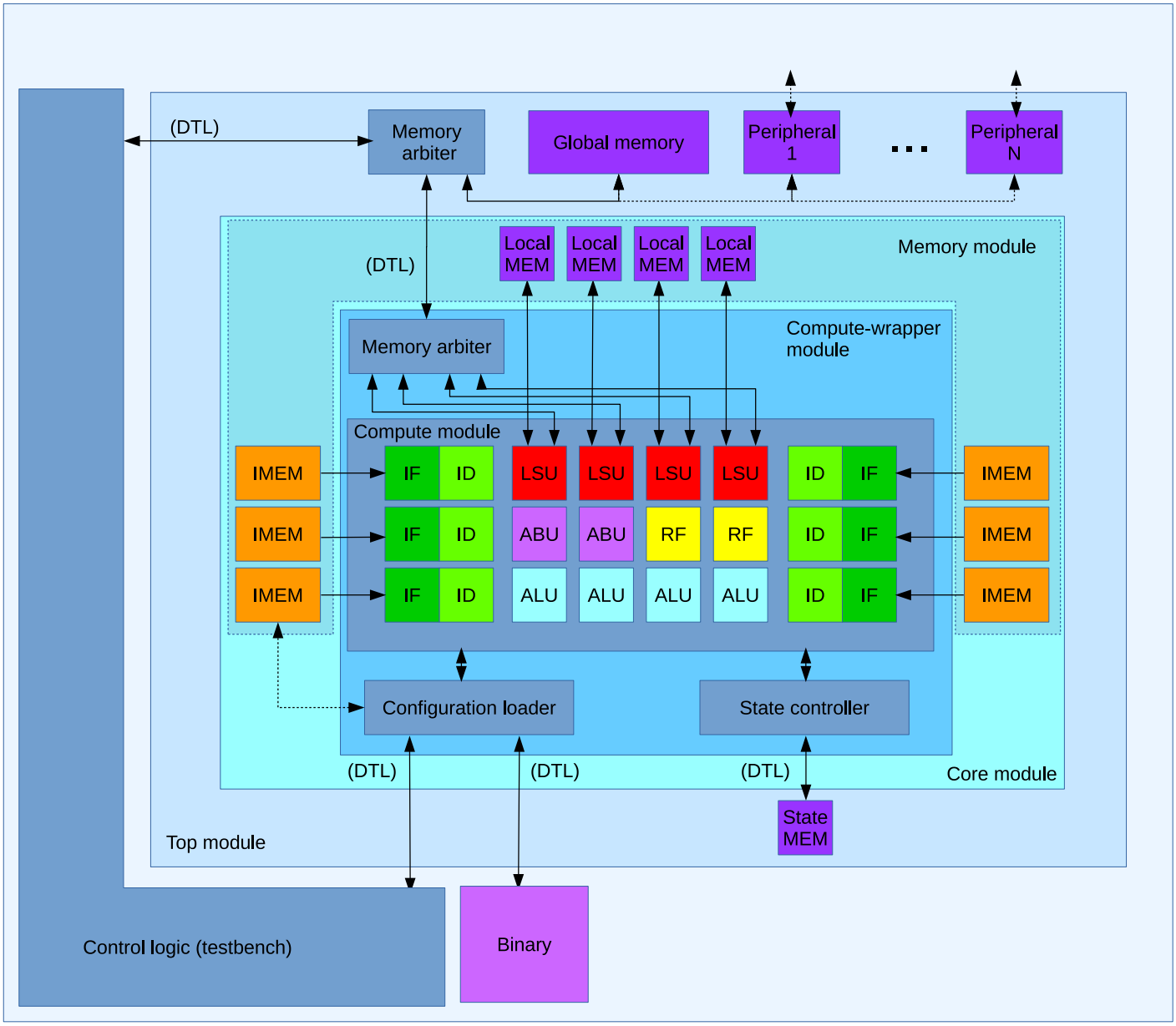
The global memory bus, and therefore also the peripherals, are arbited between the external world (a host processor can therefore also controll the peripherals) and the CGRA. The arbiter is is round robin but does not grant slots to ports without requests, if there are no requests from the external world there is no penalty in CGRA performance (in cycles) by having the arbiter present.



Picture source

Testbench

The testbench contains the simulation logic and application binary. It asserts the proper DTL control signals to configure and start the CGRA. It can also be used to poll the state of the CGRA or retrieve data from the global memory. Control signals for peripherals are available within the testbench, it is up to the user however to use these signals (not included in the default control logic).



Picture source

Verilog code structure

This section describes the structure of the toolflow generated by the CGRA toolflow. We use the following conventions:

- [design] : is the name of the design, e.g. the testbench name 'BinarizationStatic'
- [unit] : is the name of a functional unit, e.g. lsu

The code is structured as follows:

- **TB_TOP** : top level testbench loading the binary and initiating configuration of the DUT.
  dut : The Device Under Test, in our case the CGRA instance used.
  - **GM_inst** : Global memory.
  - **DTL_[peripheral name]_inst** : Peripheral connected to the global memory bus.
  - **[design]_Core_inst : Wrapper around the memory and compute modules**
    - **[design]_Memory_inst : Module containing all the memories for the CGRA (instruction-, local data- and global data memories)**
      - **LM_[unit] : Memory for functional units, usually LSUs, that are connected to a local scratchpad memory.**
      - **IM_[unit] : Instruction Memory for units such as the IDs and the immediate units.**
    - **[design]_Compute_Wrapper_inst : Module containing a loader and the compute module.**
      - **Loader_inst : Module that manages configuration of the network and FUs. It also manages loading the instruction memories.**

- **[design]_Compute_inst : Module containing all FUs, switchboxes (if present) and all wiring in between.**
  - **IF_[unit]_inst : instruction fetcher for a ID**
  - **SWB_DATA_[unit]_inst : Switchbox module for the data network**
  - **SWB_CONTROL_[unit]_inst : Switchbox module for the control (decoded instructions) network**
  - **[unit]_inst : instance of a functional unit, can be a ID, IU, ALU, ABU, LSU, RF or MUL.**