ProNoC

System Overview

Alireza Monemi

1 Project Summary

Prototype-network-on-chip (ProNoC) is an EDA tools that facilitates prototyping of custom heterogeneous NoC-based many-core-SoC (MCSoC). ProNoC is enhanced using a parameterizable virtual channel based low-latency NoC that is optimized for FPGA implementation (see *NoC Specification* Section for more details). Moreover, ProNoC can also be used as a custom Wishbone bus based SoC generator (SoC without NoC) using available Intellectual Properties (IPs) in ProNoC library. The ProNoC IP library can be easily extended to support more IPs.

2 ProNoC GUI MCSoC Generator

Writing the whole RTL code of a complex heterogeneous MCSoC manually can be time-consuming and error prone due to the huge number of possible configurations as well as high similarity among sub-components code portions. In order to facilitate the design of such complex systems, ProNoC, an open-source EDA tools that generates the complete heterogeneous customized NoC-based MCSoC RTL code is developed.

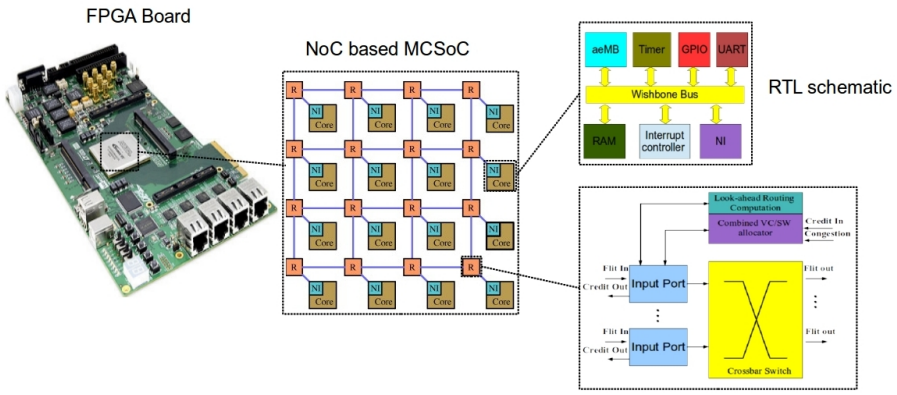


Figure 1: ProNoC system overview

The design effort increases when developing a heterogeneous MCSoC as each processing tile must be designed separately. To ease and speed up the development of such platform, a graphical user interface (GUI) is developed to generate a custom NoC-based MCSoC. The ProNoC GUI is written with Perl programming language and GTK2 library. Figure 2 illustrates the ProNoC design flow. ProNoC consists of four main windows corresponding to each layer in MCSoC design as follow.

2.1 Interface Generator

The components interconnection is facilitated by defining the interface. Interface is the combination of several ports that provide specific functionality. Each interface is

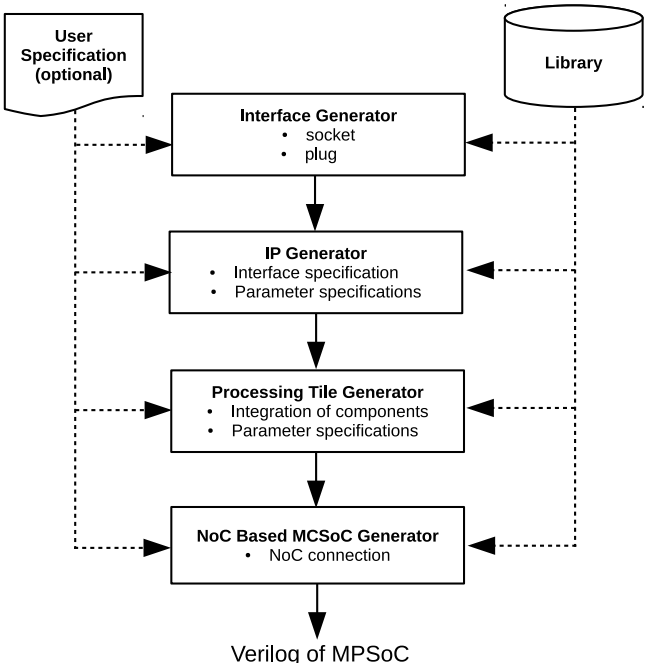


Figure 2: ProNoC design flow.

divided into two groups namely as *socket* and *plug*. Components having the *socket* interface can be connected to other components having the *plug* type of the same interface. In processing tile generator, only the *plug* interface are shown in IP box where user can connect them to the list of available *sockets*. For instance, the master and slave interface of the Wishbone bus are defined as *sockets* type whereas all other components that are connected to the Wishbone bus would have the *plug* type interfaces. For more information on Interface Generator, please refer to `/trunk/mpsoc/perl_gui/doc/interface_gen.pdf` located in project folder.

2.2 IP Generator

The IP generator facilitates the process of making a library for each IP. An IP can either be a processor or a peripheral device such as memory, timer, bus or an interrupt controller. The IP generator reads the Verilog file containing the top-level module of the IP and user can define the number of interfaces and map them to the IP ports. One advantage of the IP generator is that it can also detect the Verilog file parameters and allow the user to choose an appropriate GUI interface such as *spin-bottom* or *combo-box* for redefining the parameter when each IP is called by the PT generator. User can also define the preferable memory-mapped range and required address width (e.g. the address width can be a Verilog parameter) for slave Wishbone bus interface(s). Hence, the PT generator can automatically assign Wishbone bus addresses. For more

information on IP Generator, please refer to `/trunk/mpsoc/perl_gui/doc/ip_gen.pdf` located in project folder.

2.3 Processing Tile Generator

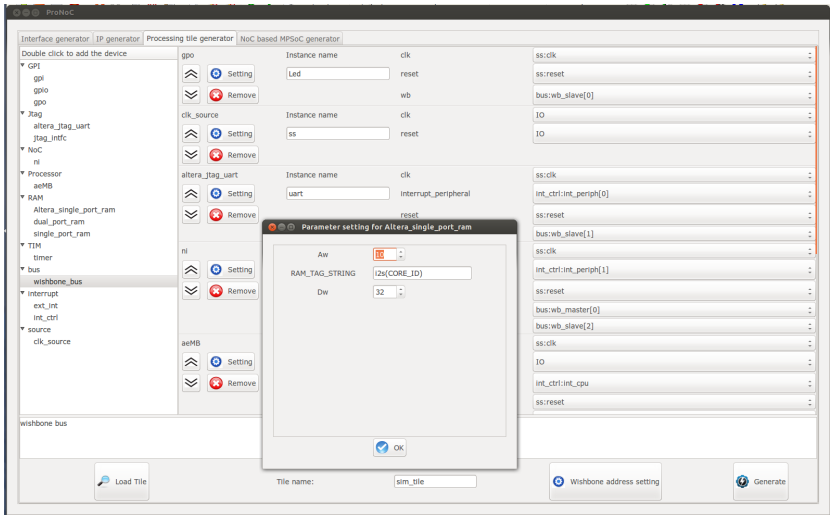


Figure 3: Processing tile generator snapshot.

The processing tile (PT) generator contains the list of all IPs that can be connected to each other using available defined interfaces. This integration tool provides some facilities such as automatic generation of interconnect logic and automatic Wishbone address setting. It also provides graphical interface for setting different IP parameters. The PT generator can generate any custom PT, generate the Verilog file containing the top level design, and generate a header file containing IP's Wishbone addresses and functions. Figure 3 shows a snapshot of the PT generator. For more information on PT Generator, please refer to to `/trunk/mpsoc/perl_gui/doc/pt_gen.pdf` located in project folder.

2.4 NoC-based MCSoC Generator

The MCSoC generator facilitates the generation of a heterogeneous NoC-based MCSoC by providing GUI interface for setting the NoC's and PTs' parameters. It checks all processing tiles which have been previously generated using the PT generator and lists all the tiles containing the NI to connect to the NoC.

2.4.1 NoC Simulator

ProNoC is developed in Verilog language, the code is verified using Modelsim simulator. However, RTL simulation is too slow to efficiently evaluate the performance of a

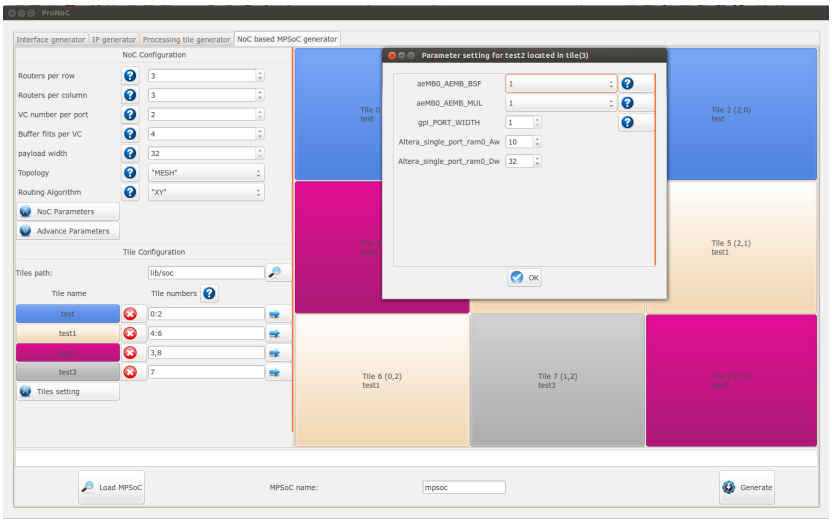


Figure 4: NoC-based MCSoC generator snapshot.

medium or large NoC under different network traffic patterns and NoC parameters. In order to accelerate the NoC simulation, Verilator simulator is used to generate a C++ model of an NoC router from the Verilog RTL code. A C++ testbench code connects multiple generated routers to generate the NoC. It also connects NoC routers to packet generators which are able to inject packets with six different synthetic traffic patterns namely: uniform random, Matrix-transposed 1 and 2, Bit-complement, Bit-reversal, and Tornado. Figure 5 shows the ProNoC simulator processes flowchart.

2.4.2 NoC Emulator

Although Verilator simulator can speed up simulation time, it consumes a lot of time especially for large NoCs. ProNoC comes up with a GUI interface for emulating of actual NoC using Altera FPGAs. To do this, a programmable packet injector module is developed which can be programed at run time using Altera JTAG. These modules inject/sink packets to the prototype NoC using different synthetic traffic patterns. In order to emulate specific NoC configuration, two files must be provided to the emulator, which are the SRAM Object File (.sof) of the compiled top-level design and the NoC configuration information file (.info file). The emulator provides the GUI interface for generating the top-level module containing the NoC, packet injectors and, remotely controllable reset and counters. The generated Verilog code is compiled using Quartus II compiler to provide the .sof file.

Having the .sof and .info files of different NoC configurations, the emulation can be done using steps shown in Figure 6. The ProNoC emulator GUI snapshot is illustrated in Figure 7.

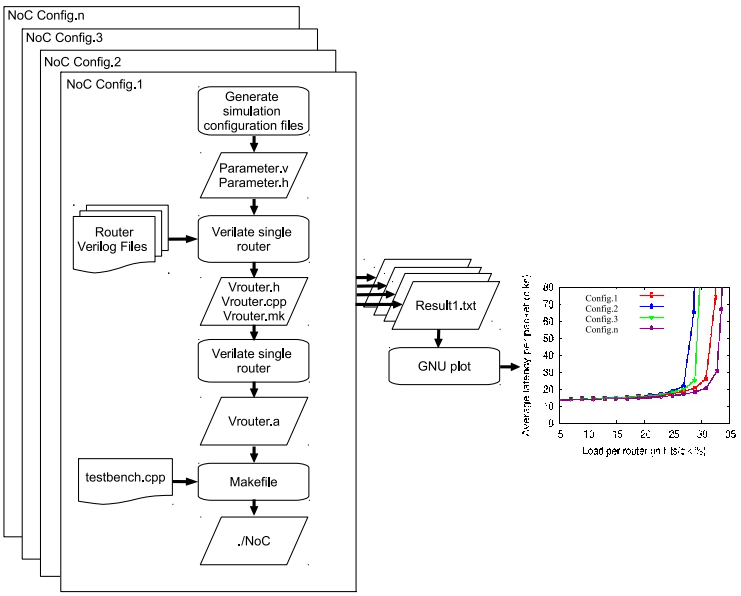


Figure 5: NoC simulator flowchart.

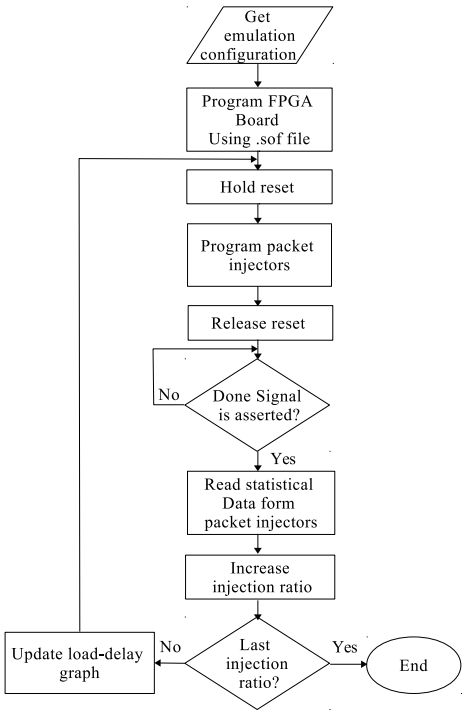


Figure 6: NoC emulator runtime stages flow chart.

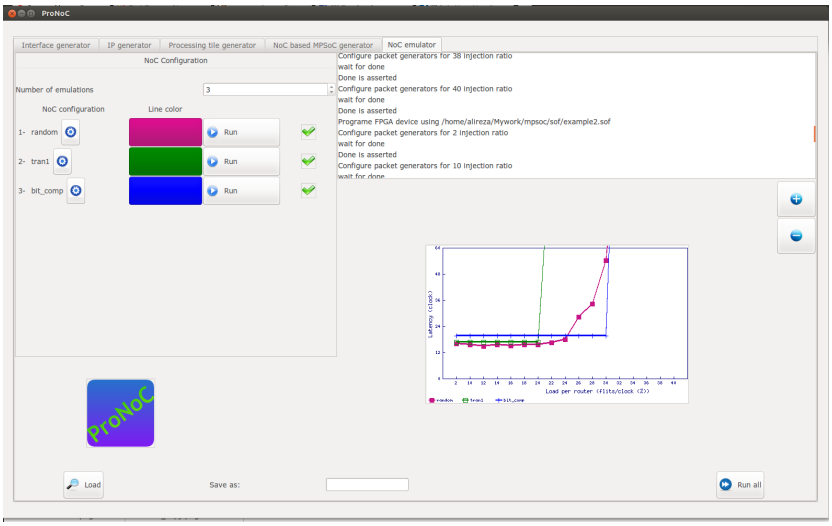


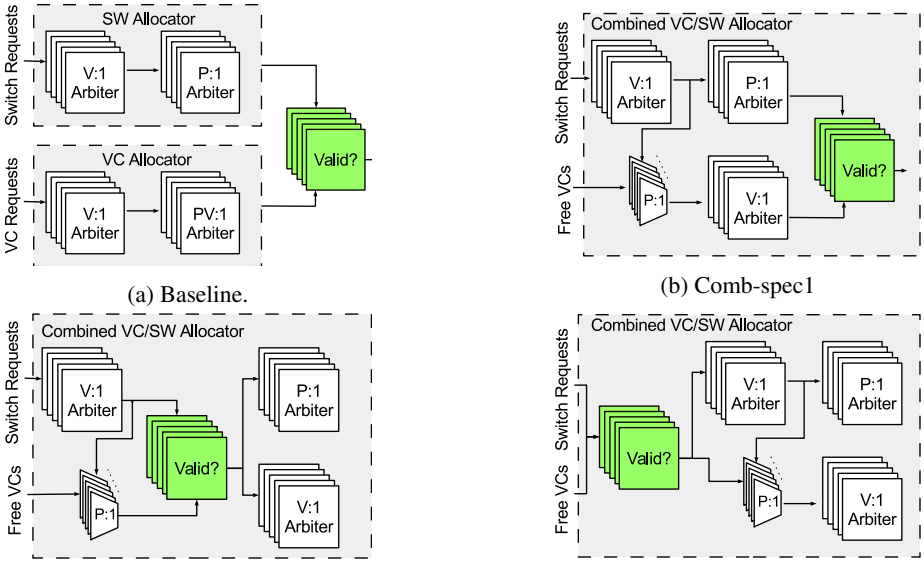
Figure 7: ProNoC NoC emulator snapshot.

3 NoC Specification

Network-on-Chip (NoC) is a scalable on-chip interconnects for complex and large-scale MCSoC. Prototyping modern NoCs on field programmable gate array (FPGA) platform can provide a functional system model that allows evaluation of the state-of-the-art NoC-based MCSoC. However, most existing FPGA-based NoC routers have simple implementation that can not represent the state-of-the-art application-specific integrated circuit (ASIC) NoCs which limits the evaluation of FPGA based MCSoC prototypes to simple NoC parameterizes. ProNoC presents an FPGA-optimized NoC-based MCSoC with ASIC-based NoC functionalities. The NoC specifications are as follow.

- **Wormhole packet switching flow-control**: Wormhole allows storing of different flits of the same packet in several routers along the path and requires low buffer.
- **Virtual Channel(VC)**: ProNoC supports parameterizable number of VCs. All VCs that are located in the same input port of the router share one BRAM memory. Typically, several VCs on a single physical channel can be implemented for various reasons such as to increase NoC throughput, prevent deadlock condition in both network-level and protocol-level, or to generate virtual networks (VNs) to support QoS for different application classes. In case that none of aforementioned features are required, the user can define VC number as one which results in a simple non-VC based router architecture.
- **Combined VC/switch allocator**: combined allocator allows simultaneous allocation of VC and switch stages in the same clock cycle to reduce the router

latency. ProNoC can be configured with three different combined VC/SW allocators:



(a) Baseline.

(b) Comb-spec1

(c) Comb-spec2.

(d) Comb-nonspec.

Figure 8: Parallel VC/SW allocation functional block diagram.

1. *comb-spec1*: VC allocator combined with speculative switch allocator where the validity of speculative requests are checked after the switch allocation stage.
2. *comb-spec2*: VC allocator combined with speculative switch allocator where the validity of speculative requests are checked after the first level arbitration stage of the switch allocator.
3. *comb-nonspec*: VC allocator combined with non- speculative switch allocator where the validity of speculative requests are checked at the beginning of switch allocation.

Combing VC and switch allocation has the benefit of lower area in comparison with separated speculative VC and SW allocator (This configuration also can be selected with setting combination type as *baseline*. However, it is not a recommended configuration as it results in high area-overhead without any significant improvement in router performance.

- **Non-atomic or atomic VC reallocation**:

  In atomic VC reallocation, a free VC can be reallocated only when it is empty. whereas in non-atomic VC reallocation a non-empty VC can be reallocated once it receives the tail flit. Atomic VC reallocation may results in inefficient buffer utilization and performance degradation.

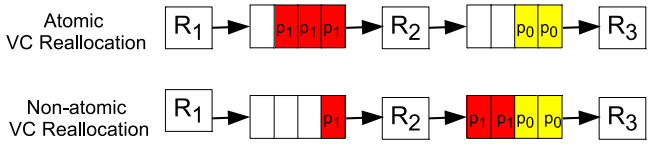


Figure 9

- **NoC topology**: Currently ProNoC supports two regular topologies of 2D Mesh and Torus.
- **Different routing algorithm:** ProNoC supports deterministic (DoR), partial adaptive (turn models and odd-even) and fully adaptive routing. To avoid a deadlock condition in torus topology due to wrap-around wires, packets are routed based on TRANC routing. ProNoC proposes an improved flow control for 2D mesh. For more information see our NoCArc'16 paper [2].
- **Router pipeline stages:** ProNoC NoC router has two pipeline stages. In the first stage three processes of look-ahead route computation, VC allocation and SW allocation are done in parallel. The second stage is switch traversal.

  ProNoC can also be configured with a static strength allocator (SSA), which allows packets traversing to the same direction pass NoC router withing 1-clk latency. More information about SSA can be found in: our ICITACEE 2016 [3].

3.1 FPFA Synthesis Results

The FPGA synthesis results of some 4×4 Mesh NoCs with different configuration are shown in this section. In all tables NoC are configured with flit size of 32 bit and 4-flit buffer size per VC on Stratix IV EP4SGX230KF40C2 Altera FPGA. Note that the reported values in percentages indicates the amount of target FPGA hardware resource usage. We also provide the synthesis results of CONNECT as it also targets ASIC style NoC router implementation. See [3] for more information including performance comparison between CONNECT and proposed NoC router.

Table 1: No-VC and 2-VC based comparison.

DoR NoC	Max freq.	Total BRAM num. (M9k)	Total LCs of 4x4 mesh	Avg. LCs of a 5-port router
no-VC	258 MHz	64 (5.2%)	11296 (6.2%)	673 (0.4%)
2-VC	218 MHz	64 (5.2%)	18578 (10.2%)	1105 (0.6%)

Table 2: 2-clk and 2clk-SSA synthesis results comparison using 4-VCs.

DoR NoC	Max freq.	Total BRAM num. (M9k)	Total LCs of 4x4 mesh	Avg. LCs of a 5-port router
2-cycle	165 MHz	64 (5.2%)	41406 (23%)	3234 (1.8%)
2-cycle-SSA	152 MHz	64 (5.2%)	46296 (25%)	3616 (2.0%)
CONNECT	94 MHz	-	67666 (43%)	5286 (3.6%)

Table 3: 4-VCs adaptive NoC synthesis results.

	Max freq.	Total BRAM num. (M9k)	Total LCs of 4x4 mesh	Avg. LCs of a 5-port router
Fully/partially adaptive	161-163 MHz	64 (5.2%)	48,668 - 49940 (27 %)	3802 - 3901 (2%)

4 In-system Communication

The communication with the FPGA board including memory content updating (programming the processors) or reading/writing to the probes/sources has been done using JTAG interfaces. To do this a `Jtag_wb` interface module is developed which can be connect the Wishbone bus to the Altera Virtual JTAG TAB. The communication via host PC and FPGA board is handled using `Jtag_man` software which is written in C to allow simple data transferring to the `Jtag_wb` via USB blaster cable.

5 Target Platform

ProNoC has been developed and verified using ALtera FPGAs. In order to use other FPGA vendors you probably needs to replace the Altera VJTAG TAB with the specific target FPGA Jtag TAB and modify the `Jtag_man.c` to adapt with the new TAB.

6 How to Cite

If you found ProNoC useful please cite following references in your publications:

[1] Alireza Monemi, Chia Yee Ooi, and Muhammad Nadzir Marsono. Low latency network-on-chip router microarchitecture using request masking technique. *International Journal of Reconfigurable Computing*, 2015:2, 2015.

[2] Alireza Monemi, Chia Yee Ooi, Muhammad Nadzir Marsono, and Maurizio Palesi. Improved flow control for minimal fully adaptive routing in 2D mesh NoC. In *Proceedings of the 9th International Workshop on Network on Chip Architectures*, NoCArc'16, pages 9–14. ACM, 2016.

[3] Alireza Monemi, Chia Yee Ooi, Maurizio Palesi, and Muhammad Nadzir Marsono. Low latency network-on-chip router using static straight allocator. In *Proceedings of 3rd International Conference on Information Technology, Computer and Electrical Engineering*, ICITACEE'16. IEEE, 2016.

7 Additional Documentation

For more information and tutorials, please check `trunk/mpsoc/perl_gui/doc` directory.