SoC Simulator on FPGA using Bluespec System Verilog

Mohsen Ghasempour, Mikel Luján, and Jim Garside
School of Computer Science
The University of Manchester
Oxford Road, Manchester, M13 9PL, UK
{ghasempm , mikel.lujan , jgarside}@cs.manchester.ac.uk

Abstract— **Building large computing systems requires first to model them. Modern hardware systems are so complex that their software models in the desired detail may be too slow. Thus abstract hardware modelling can be appropriate. This paper presents an example software/hardware model built using Bluespec System Verilog (BSV) design flow to give rapid simulation of a hardware system. The chosen example was a hardware model of the on-chip router, on-chip and off-chip network of SpiNNaker for understanding the behaviour of the traffic in the system. A model of a 5×5 SpiNNaker topology has been designed in Virtex-5 FPGA using BSV and a Graphical User Interface (GUI) was developed in LabVIEW for graphical representation of the results.**

I. INTRODUCTION

Complex computer architectures, such as current multicores and SoCs, are time and cost expensive to design. Therefore, a precise model of the system is required to avoid costly mistakes. Due to the level of complexity of modern hardware architectures, software models (of such systems) are generally too slow so hardware models are preferred. In recent years, demands for FPGA (Field Programmable Gate Array) based hardware emulators and software accelerators have increased [1-6]. The reason is that modern FPGAs provide large capacity, high-speed and enough flexibility to map almost any application and run it faster than software. A reconfigurable hardware model on FPGA makes it possible to refine and modify a design easily to get the desired outcome. Also, FPGAs are able to emulate the functionality of the model (clock accurate simulation or emulation). These attributes make FPGAs an excellent candidate for abstracted hardware modelling of computer architectures. On the other hand, developing an accurate model of SoCs or multicores on FPGAs is not straightforward and requires in depth knowledge about computer architecture design and Hardware Description Language (HDL); such as VHDL, Verilog, System Verilog or Bluespec. As systems become more complex, more effort is required to achieve a reliable model of the system. In this situation a high level HDL can speed up the modelling process as well as producing a more reliable design. Bluespec is able to model computer architecture at a high level of abstraction as well as guarantee the correctness of functionality.

In this study a hardware model of the SpiNNaker network is developed on a Virtex-5 FPGA using BSV for understanding the traffic behaviour on the system (Fig. 1). The network traffic is subject to queuing, contesting for links and, in extreme cases, packet dropping. It is the behaviour of this complex scalable network under dynamic loads which is the primary modelling target. LabVIEW was used as a front-end GUI for graphical presentation of the results. A brief introduction of SpiNNaker system, Bluespec and LabVIEW are presented in Sections II, III and IV, respectively. Section V presents the experimental work and design process. Conclusions and future work are drawn in Section VI.

Figure 1. SpiNNaker chips and Virtex 5 FPGA.

II. SPINNAKER ARCHITECTURE

SpiNNaker is a hardware-based real time simulator of spiking neural network which consist of up to 65,536 SpiNNaker nodes (chips) or one million processors. This system is able to simulate around one billion neurons in real time [7]. Models of the neurons are run in software, the architecture provides acceleration of the delivery of neuron spikes (i.e. one packet in SpiNNaker).

A. SpiNNaker Node

A SpiNNaker node is a SoC consists of 18 ARM968 processors. Each processor is able to simulate up to 1000 neuron in real time [8]. There are two Networks on Chip (NoC) in each SpiNNaker node, a Communications NoC (Comms NoC) and System NoC. The primary function of System NoC is to connect processors to the SDRAM interface. The Comms NoC carries packets between the

processors on the same or different chips. A block diagram of SpiNNaker chip is presented in Fig. 2.

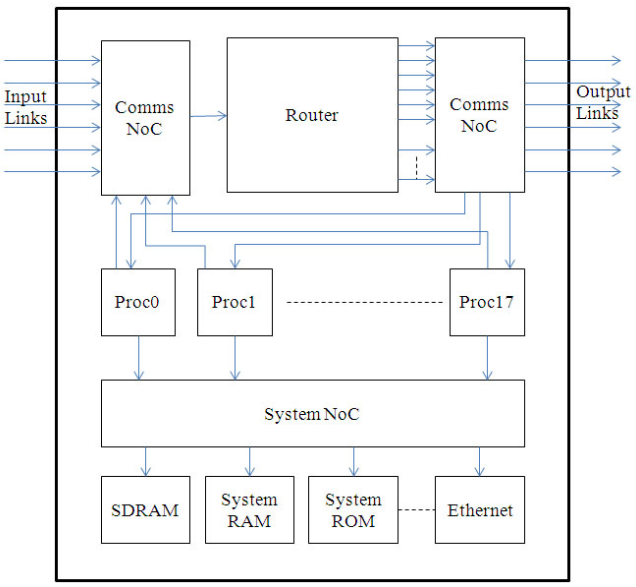

Figure 2. Schematic model of SpiNNaker chip.

B. Router

One of the main components of SpiNNaker chip is the router. This is the heart of the Communications NoC and it provides the internal interconnection (between cores) and external interconnection (between SpiNNaker nodes). A hardware simulator of an SpiNNaker network with point-to-point router was developed in this work. In point-to-point communications a packet can be sent from one processor in one SpiNNaker node to other processor in another SpiNNaker chip anywhere in the system [9, 10].

C. Topology of Interonnection netwrok

SpiNNaker chips are arranged in a 2D triangular mesh topology wrapped into a torus. A fully expanded SpiNNaker system is composed of 256×256 SpiNNaker node. Each SpiNNaker node has six bidirectional links in six different directions, North, South, East, West, North-East and South-West (Fig 3). An example of 5×5 structure of SpiNNaker system with wrap-around links is presented in Fig. 4.

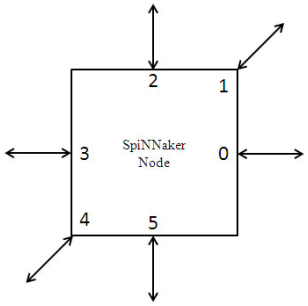

Figure 3. SpiNNaker node's external links.

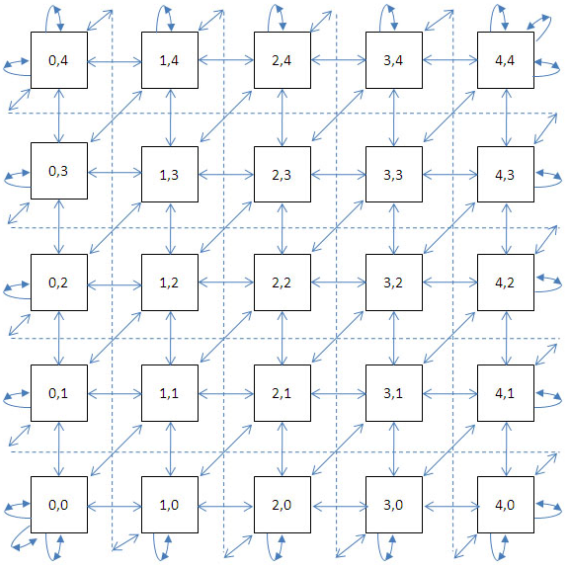

Figure 4. Example of a 5x5 SpiNNaker topology.

III. BLUESPEC SYSTEM VERILOG

BSV (Bluespec System Verilog) is a *high level* hardware description language which is fully *synthesizable* to hardware. These two attributes make BSV an excellent tool for the fast prototyping of complex systems. BSV uses a behavioural model which is called *Atomic Rules and Interfaces*. There are two reasons why BSV uses this model. First, the model of atomic rules is basically parallel so this makes BSV suited to the massive parallelism in a complex hardware design. Second, atomicity allows the functional correctness of a design to be achieved by considering the one-rule-at-a-time semantic (looking at each of the rules in isolation, without considering the action of other rules). BSV has a very strong type elaboration mechanism which provides great power to express computer architecture with high level of reliability. In the experimental results section, it will be explained how BSV can speed up the design process. A sample of BSV code is presented in Fig. 5. More examples and explanation about BSV can be found in [11, 12].

```
package Tb;

(* synthesize *)
module mkTb (Empty);

   Ifc_type  ifc <- mkModuleDeepThought;

   rule theUltimateAnswer;
    $display ("Hello World! The answer is: %0d",ifc.the_answer (10, 15, 17));
    $finish (0);
   endrule

endmodule: mkTb

interface Ifc_type;
   method int the_answer (int x, int y, int z);
endinterface: Ifc_type

(* synthesize *)
module mkModuleDeepThought (Ifc_type);

   method int the_answer (int x, int y, int z);
      return x + y + z;
   endmethod

endmodule: mkModuleDeepThought
endpackage: Tb
```

Figure 5. An example of Bluespec code [4].

IV. LABVIEW

Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW) is a system development environment for visual programming language from National Instruments Corporation. LabVIEW uses a dataflow programming language called "G". In contrast with sequential (programming) languages, "G" is capable of parallel execution. Its graphical environment and variety of libraries for different communication protocols (such as RS232, RS485, TCP/IP) make LabVIEW an excellent candidate for creating front-end monitoring platforms and GUIs for custom designed hardware [13]. Fig 6 shows an example code in LabVIEW.

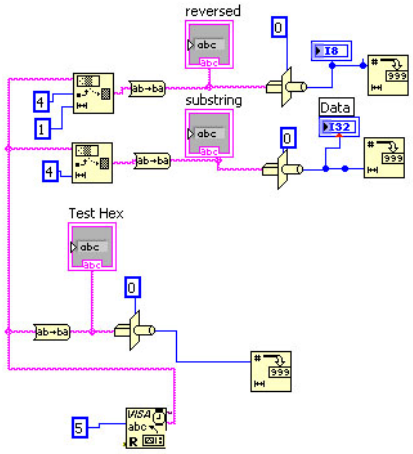

Figure 6. Example of LabVIEW code.

V. EXPERIMENTAL WORK

A. *Model of the system*

The focus of this work is to investigate the traffic behaviour in the Comms NoC in a multi SpiNNaker chip system. Thus, some simplifications were done to the structure of SpiNNaker to have a simple model of fundamental blocks involved in the Comms NoC. Fig. 7 presents the implemented model in this work which consists of a 5×5 SpiNNaker structure, a RAM unit and a Monitor unit. The Monitor unit gathers the desired information from all SpiNNaker nodes and sends it to the serial port (RS232). The RAM unit is used for initialization of the network parameters.

Each SpiNNaker node is simplified and modelled as shown in Fig. 8. There is no need to have the System NoC and SDRAM for analysing the network traffic in a chip, therefore these parts are eliminated from the model. All 18 ARM processors are modelled as an injection queue with room for four packets at its output and a queue with the same room in its input for receiving packets.

A 64-bit programmable register has been implemented in the modelled processor which makes the system more flexible in injecting a desired traffic pattern. Also, for each processor, another 64-bit program register dedicated to the RAM unit (in this model twenty five 64-bit program registers are fitted in the RAM unit).

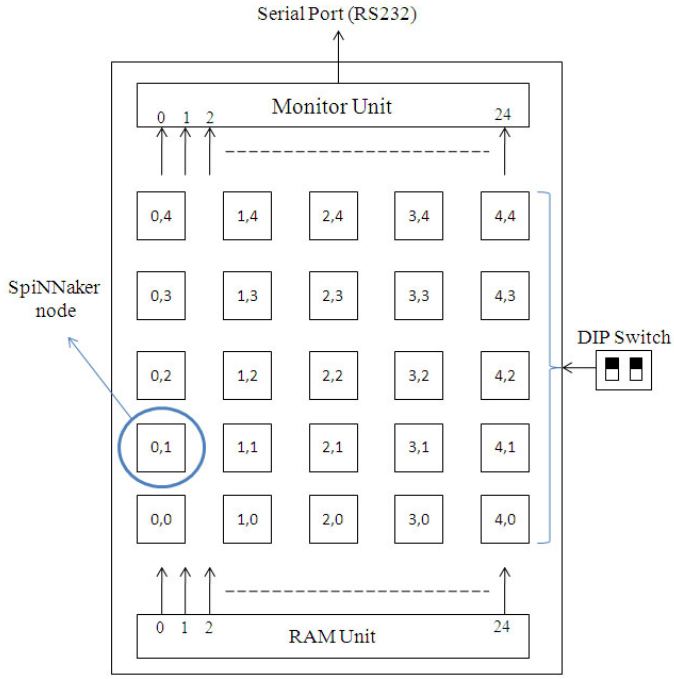

Figure 7. Implemented model in BSV.

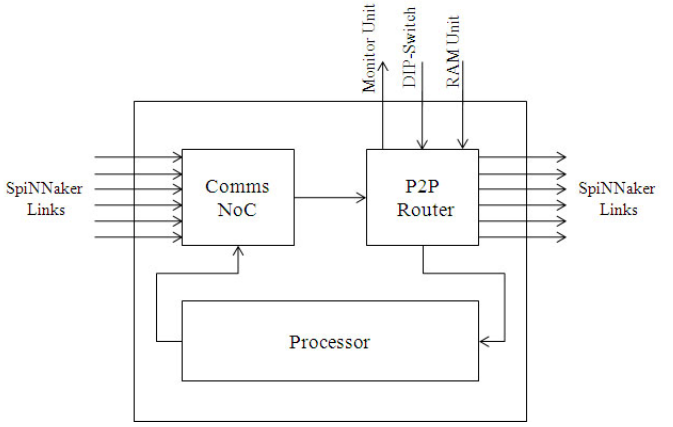

Figure 8. Simplified model of SpiNNaker network components.

In the example presented in Fig 9, the first 16 bits of the program register specify the location (coordinate) of each node in the system. 8 bits for X coordinate and 8 bits for Y coordinate. The next 8 bits are dedicated for clock intervals (delay) between injecting each packet by a processor. Bits 48 to 55 present the number of packets which each processor should inject and bit 24 specifies the mode of injection, if it is 0 the processor will only send the number of specified packets (by bits 48-55) once and then stop injection, otherwise, if bit 24 is 1, the processor keeps injecting packets continuously. Bits number 32 to 47 (16 bits) specify the location of the first destination chip (processor will send the first packet to this node).

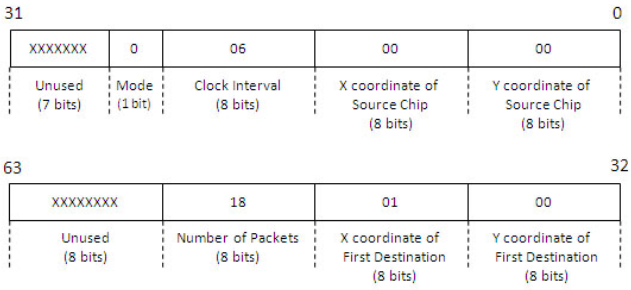

Figure 9. An example of program register.

According to the specified value in the program register (hex format), the SpiNNaker node which is located in coordinate 0,0 will send the first packet to the node at coordinates 1,0 and keep injecting 24 packets to all other SpiNNaker nodes according to the specified path way in Fig. 10. Using this algorithm it is possible to generate point-to-point uniform and non-uniform traffic patterns.

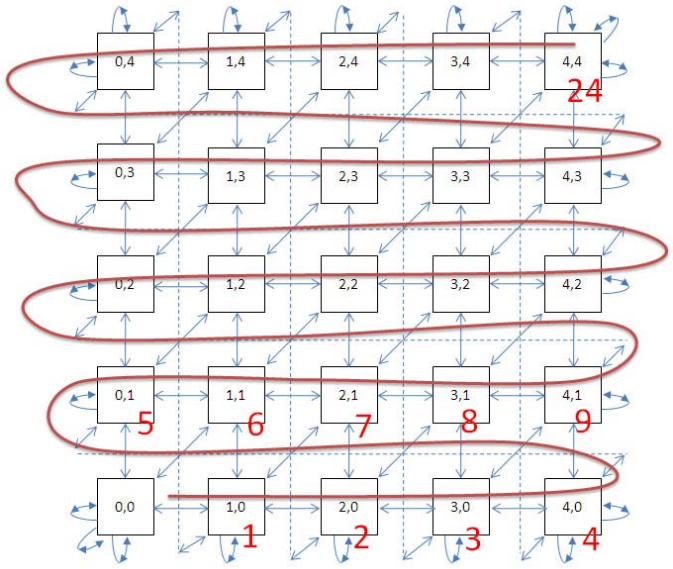

Figure 10. Path way of injection of 24 packets in a 5x5 SpiNNaker system.

The Comms block serialises the input packets coming from 6 neighbour SpiNNaker chips and local processors and feed them into the router one at a time. The second part of Comms NoC (right hand side) presented in Fig. 2 was eliminated for the sake of simplicity and packets are routed directly (Fig,11).

The router plays a significant role in the communication network functionality so an accurate model of the point-to-point router has been developed and implemented. In the original SpiNNaker, the point-to-point router uses routing tables. In our model Dimension Order Routing (DOR) [14] has been used instead for the sake of simplicity and efficient use of FPGA resources. Fig. 12 presents a block diagram of the implemented point-to-point router. Three pipeline stages have been considered for implementation of the point-to-point router, Packet decode, Routing engine and Output select.

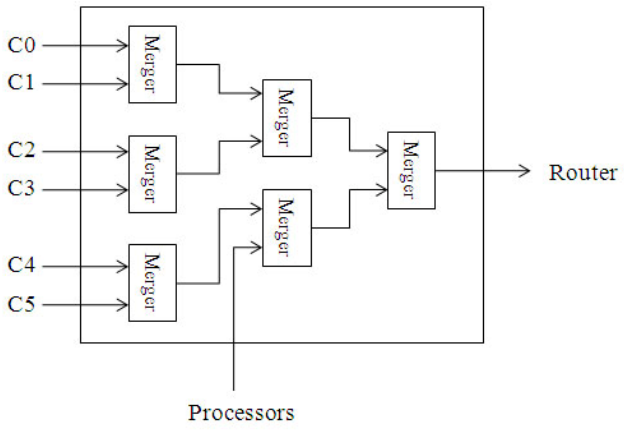

Figure 11. Comms NoC block diagram.

In the first stage, the input packet is analysed and required information for the next stage will extracted. In the Routing engine stage two main operations are executed. If a packet comes from the first stage then DOR will be applied to it and its destination will be determined. Otherwise, the packet comes from RAM unit so the Routing stage will extract the source chip ID of the packets (program register) and route the program packet to the internal processor. In the case that the input packet comes from the decode stage, after determining the destination (output link) with the Routing engine, the packet will be sent to the Output select stage. Here the router tries to send the packet to the specified link. If there is no congestion or failure on the specified link, the router will send the packets successfully. If a packet cannot be sent, two programmable timer intervals are applied. The first allows a period for working on retrying; the second allows attempts of rerouting (if link 1 fails, link 0 will be used (Fig 3)). If the second timer expires the packet is dropped.

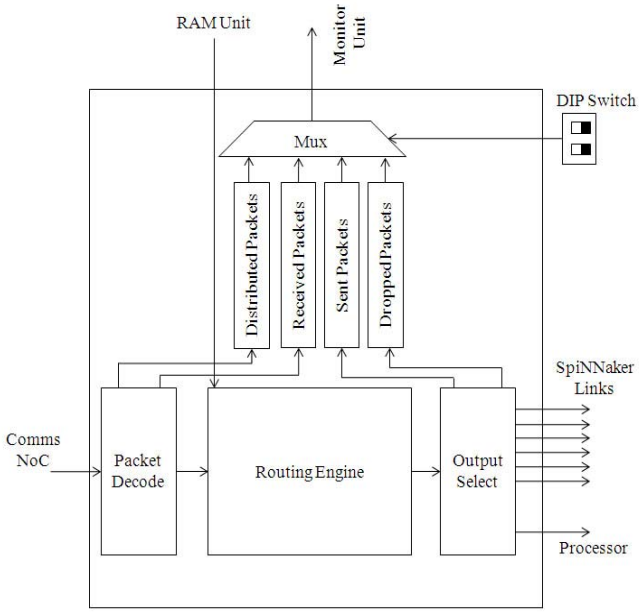

Figure 12. Router block diagram.

During the routing process an internal monitor unit counts the number of dropped (due to failure or congestion in output links), injected (by local processor), received (by local processor) and distributed packets (packets which come through the router only from 6 bidirectional link not from the local processor) in the router. Two switches choose the desired monitor parameter. By changing the position of the switches, number of dropped, injected, received or distributed packet will be sent to the monitor unit. Different positions of the switches are depicted in Table 1.

Table 1. DIP-Switch Positions

Dip Switch	MUX output
00	Dropped Packet
01	Injected Packet
10	Distributed Packet
11	Received Pakcet

These switches are wired to all of SpiNNaker nodes so that at any time the monitor unit will receive 25 monitor packets from 25 SpiNNaker nodes.

A monitor unit was designed which continuously receives the dropped, received, injected or distributed packets (depending on the position of the switches) from all (SpiNNaker) nodes and sends these to the serial port which is connected to the general purpose computer. A front-end GUI was developed using LabVIEW depicts a graphical presentation of the traffic behaviour of the network (Fig. 13).

A RAM unit with 25 output ports was designed to initialise the required parameter of routers such as source chip ID and to program the register of each processor in the system. These parameters can be saved in a text file and then loaded into the RAM unit block during the synthesis and implementation process. This makes it possible to generate different traffic patterns easily by modifying twenty five 64-bit registers in a text file.

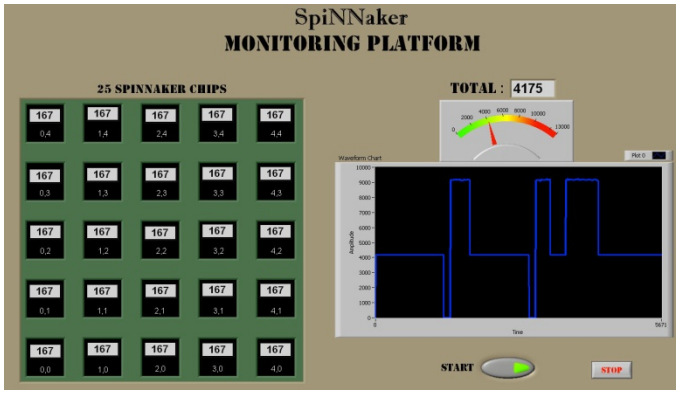

Figure 13. SpiNNaker monitoring platform implemented in Labview.

B. Design process

The whole design was implemented using BSV and simulation results have been analysed using Bluesim, which is a clock accurate BSV simulator, to investigate the correctness of the system. GTKwave has been used for precise analysing of simulation results which generated as VCD format from Bluespec compiler. Then Verilog files corresponding to each implemented BSV module were generated using the BSV compiler. Generated Verilog files have been synthesized using ISE (Xilinx synthesis tool) to generate the bit stream file. Finally the generated bit stream file was loaded into the FPGA and the functionality of the whole system was analysed in real time (Fig 14).

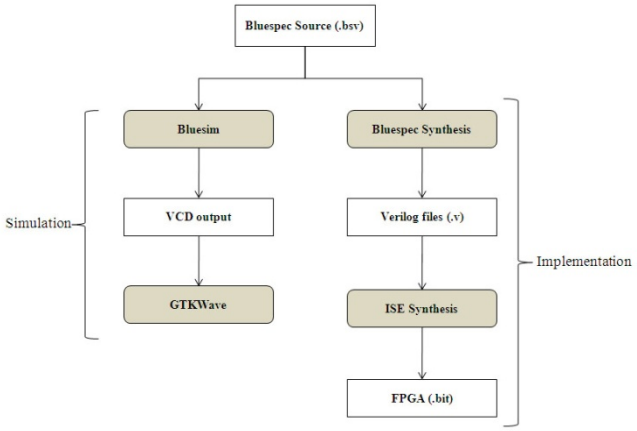

Figure 14. Design flow.

Highlighted boxes in Fig. 14 are different tools which are used during the design process.

C. Experimental Results

ML509 (Virtex-5 XC5VLX110T) Xilinx evaluation board is used as a target platform for the purpose of this paper. Table 2 shows a brief summary of available resources on selected Virtex-5 FPGA.

Table 2. Virtex-5 XC5VLX110T resources

Resources	Available
MAX user I/O	680
Slice LUTs	69,120
Max Distributed RAM (Kb)	1,120
Block RAM blocks (Kb)	5,328

Different traffic patterns were injected into the system and the behaviour of the system was analysed. First a uniform traffic pattern was injected by defining the program registers of all SpiNNaker chips to the desired value (each SpiNNaker node injects 24 packets continuously starting from its adjacent node). Each SpiNNaker node will count and accumulate dropped, injected, received and distributed packets in each clock cycle (clock accurate monitoring). Then after 1000 clock cycles SpiNNaker nodes send the desired monitor packet (specified by the switches) to the monitor unit. This adds an 8-bit tag to each packet to specify the node's ID and then sends it to the serial port. In the front-end software, LabVIEW, the received packet is decoded and according to its tag will be displayed in the corresponding

position depicted graphically in fig 13. Also, a real time graph shows the total number of dropped, received, injected or distributed packets.

For precise analysis of the traffic behaviour of the system, a single graph has been dedicated for each SpiNNaker node. Therefore the variation of dropped, received, injected and distributed packets in each node can be observed (Fig 15). Also a graphical presentation of network traffic density was implemented (Figs 16 and 17).

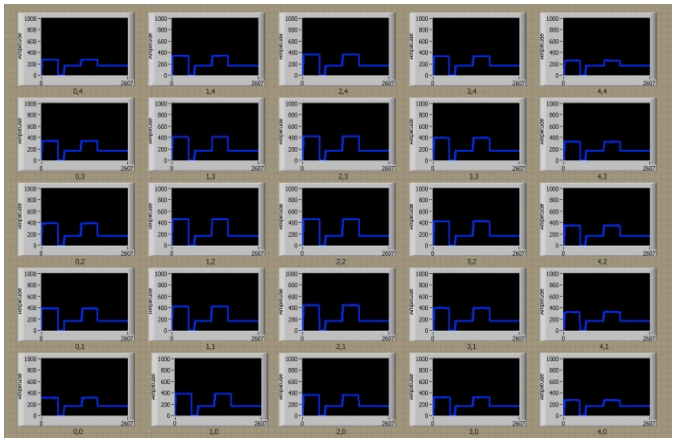

Figure 15. Dedicated graph for each SpiNNaker node.

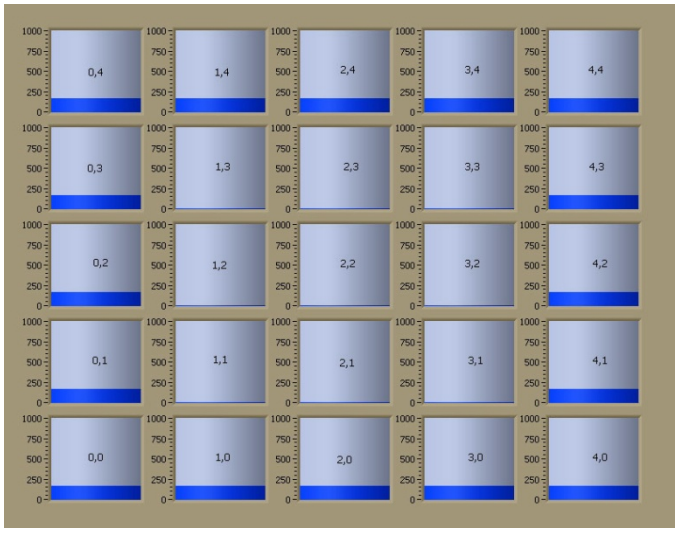

Figure 16. An example of injected pattern (SpiNNaker nodes, except 9 nodes located at the centre, are injecting packet toward node located at coordinate 2,2).

Only the injected packets from different SpiNNaker nodes can be observed in Fig. 16. The front-end software displays one of the dropped, received, injected or distributed packets at a time. Also Fig. 17 only presents the distributed packets in the system. The graphical presentation of traffic behaviour gives good visibility of the probable fault in the design. For instance if there is any fault in DOR implementation, in the router design, the outcome will be non-uniformed traffic distribution in the system which is clear from GUI in LabVIEW.

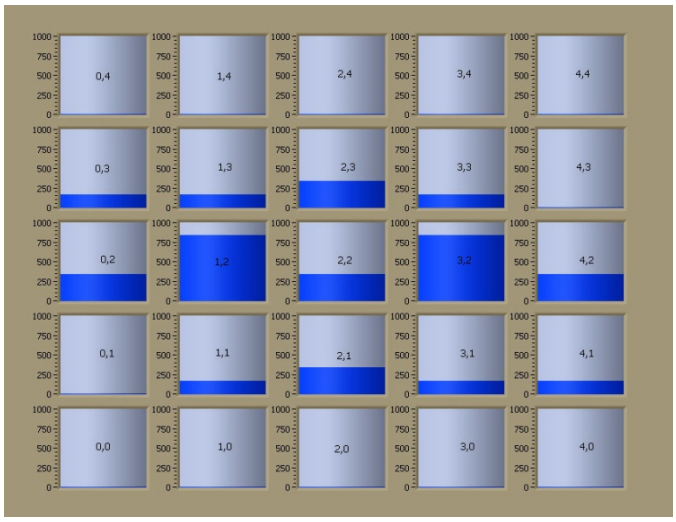

Figure 17. Presentation of network traffic distribution for presented example of injected packet in Fig. 16.

The bottleneck of the system performance is the serial communication since it limits the speed of the monitoring process. The maximum baud rate of serial communication is 115,200. Each monitor packet is 32 bits wide, but after adding an 8-bit tag and some start and stop bits, it will be 50 bits wide (40 bits data plus 10 control bits). Therefore, in the best situation, 433 µs (approximately) is required for sending each packet to the front end software (286 clocks interval for each bit if using 33 MHz clock and 868 clocks interval if using 100 MHz clock). Also, in practice, some extra delay should be considered between sending packets to avoid conflict in receiving packets in the front end software. Although these unwanted delays degrade the performance of the system, the hardware model is still much faster than the software simulator. A comparison has been done between Bluesim (BSV simulator) and the hardware implementation. The results show that the FPGA implementation is 160,000 times faster than Bluesim. Simulation ran on a core i7-2.8 GHz computer with 6 GB RAM. Table 3 shows the FPGA utilization summary for 5×5 SpiNNaker topology modelled in this study. The results show that significant number of slice LUTs (87%) in FPGA has been used.

Table 3. Utilization summary

Resources	Utilization
Slice Registers	47%
Slice LUTs	87%
Occupied Slices	96%
Bonded IOs	1%

Table 4 presents the number of code lines for BSV implementation and corresponding generated Verilog code by Bluespec synthesis tool. According to these results, Verilog implementation of the same system can requires up to 4 times more effort. It could be claimed that generated Verilog code by BSV is not as efficient as handwritten Verilog code but still we found that BSV can speed up the design process.

Table 4. Number of code line in BSV and generated Verilig files

Modules	Bluespec (.bsv)	Generated Verilog (.v)	Ratio
SpiNNaker_Board	1324	6203	4.6
SpiNNaker	241	765	3.1
Processor	105	299	2.8
Comms NoC	153	575	3.7
Router	342	1055	3.08
Monitor_Unit	223	1195	5.3
RAM	159	1211	7.6
Total	**2547**	**11303**	**4.4**

VI. CONCLUSION AND FUTURE WORK

In this study an FPGA-based platform was designed for understanding the traffic behaviour in the SpiNNaker network. The SpiNNaker architecture was introduced as a hardware-based real time simulator of spiking neural network. Fundamental components in a SpiNNaker chip such as Comms NoC, System NoC, Router and Processors were briefly analysed and their functionality discussed. A brief introduction of BSV has been presented and its main attributes were discussed. For the purpose of analysing the network traffic behaviour in the system an efficient model of the main components of the network was implemented. The System NoC, SDRAM and some other peripherals were eliminated from the model since these have no effect on traffic in the network. By eliminating these components there are enough FPGA resources to have a more accurate model of important components such as the router. The local processors were modelled as a single queue with room for four packets in its input and output.

The Comms NoC was modelled to serialise incoming packets from local processors and other SpiNNaker nodes and feed them into the router one at a time. Then an accurate model of the point-to-point router was proposed and discussed. In the process of routing 4 main effective parameters for analysing the traffic behaviour has been monitored which are sent, received, injected and distributed packets (clock accurate monitoring). The router will send one of these parameters to the monitor unit according to the position of dedicated switches. Therefore a SpiNNaker network model was implemented using the mentioned three components: Processor, Comms NoC and Router.

A model of a 5×5 SpiNNaker topology (triangular torus) with wrap-around connection (Fig. 4) was designed in BSV. A monitor unit was designed to gather monitor packets of all SpiNNaker nodes and send them to the front-end software in LabVIEW. A RAM unit has been implemented for initializing the network parameter during system start up. Finally the whole system was developed in BSV and the functionality was analysed using Bluesim. ISE has been used for the synthesis of generated Verilog files by Bluespec compiler and producing the bit stream file for FPGA. A Virtex-5 XC5VLX110T was used as a FPGA target platform for this study.

BSV was a productive way of fast prototyping and developing the computer architecture models for FPGA design. Its powerful compiler makes it easy to debug the design in a very short time and generate a reliable implementation of a hardware model. Graphical environment of LabVIEW makes it easy to generate desirable GUI for hardware interfaces as well as speed up the design process. Also, inherently parallel nature of “G” language makes LabVIEW an excellent choice for developing the front-end software for FPGA based systems.

Synthesis results present 87% utilization of Virtex-5 slice LUTs. Serial communication degraded the performance of the system due to its low baud rate. Ethernet could be a better choice for communicating with the front-end software which may be considered in the future work. Navaridas et al. [15] developed a software simulator for analysing the traffic behaviour of the largest (256×256) SpiNNaker system (65K nodes). By some modification on the software simulator, some accurate comparison could be done between our hardware accelerator and software simulator. Finally, as future work we intend to develop a real time monitoring platform, using multiple FPGA boards, for analysing the behaviour of the network traffic for the multicast routing packets in SpiNNaker.

ACKNOWLEDGEMENTS

Dr. Luján is supported by a Royal Society University Research Fellowship.

VII. REFERENCES

[1] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun, "Accelerating Millions of Short Reads Mapping on a Heterogeneous Architecture with FPGA Accelerator," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, Toronto, Canada, 2012, pp. 184-187.

[2] C. Dennl, D. Ziener, and J. Teich, "On-the-fly Composition of FPGA-Based SQL Query Accelerators Using a Partially Reconfigurable Module Library," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, Toronto, Canada, 2012, pp. 45-52.

[3] M. Psarakis, A. Pikrakis, and G. Dendrinos, "FPGA-based Acceleration for Tracking Audio Effects in Movies," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, Toronto, Canada, 2012, pp. 85-92.

[4] E.S. Chung, M.K. Papamichael, E. Nurvitadhi, J.C. Hoe, K. Mai, and B. Falsafi, "Protoflex: Towards scalable, full-system multiprocessor simulations using fpgas," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, p. 15, 2009.

[5] E.S. Chung, E. Nurvitadhi, J.C. Hoe, B. Falsafi, and K. Mai, "PROTOFLEX: FPGA-accelerated hybrid functional simulator," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1-6.

[6] J. Wawrzynek, D. Patterson, M. Oskin, S.L. Lu, C. Kozyrakis, J.C. Hoe, D. Chiou, and K. Asanovic, "RAMP: Research accelerator for multiple processors," *Micro, IEEE*, vol. 27, pp. 46-57, 2007.

[7] X. Jin, M. Luján, L.A. Plana, S. Davies, S. Temple, and S. B. Furber, "Modeling spiking neural networks on SpiNNaker," *Computing in Science & Engineering*, vol. 12, pp. 91-97, 2010.

[8] X. Jin, S.B. Furber, and J.V. Woods, "Efficient modelling of spiking neural networks on a scalable chip multiprocessor," in *Neural Network (IEEE world congress on computional intelligence)*, 2008, pp. 2812-2819.

[9] L.A. Plana, S.B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS infrastructure for a massively parallel multiprocessor," *Design & Test of Computers, IEEE*, vol. 24, pp. 454-463, 2007.

[10] M. Khan, DR. Lester, L.A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *Neural Network (IEEE world congress on computional intelligence)*, 2008, pp. 2849-2856.

[11] R.S. Nikhil and K.R. Czeck. *BSV by Example (The next-generation language for electronic system design), 2010*.Available:http://csg.csail.mit.edu/6.S078/6_S078_2012_www/resources/bsv_by_example.pdf

[12] Bluespec. *Bluespec System Verilog Reference Guide, 2012*.Available:http://www.google.co.uk/url?sa=t&rct=j&q=bluespec+system+verilog+reference+guide+2011&source=web&cd=2&ved=0CE8QFjAB&url=http%3A%2F%2Fwww.bluespec.com%2Fforum%2Fdownload.php%3Fid%3D158%26sid%3Dc7f911b9d549faaa21aa10e81273b0cb&ei=OQHWT5aFFaa_0QWTs9GxBA&usg=AFQjCNGIIiWyrUEGZtrHVAavOtPePfubCQ&sig2=dJV0x-Y0RE7cRhRlKSDEYg

[13] *National Instruments, What is LabVIEW?* Available: http://www.ni.com/labview/whatis/

[14] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*: Morgan Kaufmann, 2004.

[15] J. Navaridas, M. Luján, J. Miguel-Alonso, L.A. Plana, and S. B. Furber, "Understanding the interconnection network of SpiNNaker," in *23rd international conference on Supercomputing*, 2009, pp. 286-295.