8-2018

Design and Verification of a Round-Robin Arbiter

Aung Toe
axt1937@rit.edu

Design and verification of a Round-Robin Arbiter

by
Aung Toe

Graduate Paper

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in Electrical Engineering

Approved by:

Mr. Mark A. Indovina, Lecturer
Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

---

Dr. Sohail A. Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering

Department of Electrical and Microelectronic Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
August 2018

To my family and friends, for all of their endless love, support, and encouragement.

Abstract

As the number of bus masters increases in chip, the performance of a system largely depends on the arbitration scheme. The throughput of the system is affected by the arbiter circuit which controls the grant for various requestors. An arbitration scheme is usually chosen based on the application. A memory arbiter decides which CPU will get access for each cycle. A packet switch uses an arbiter to decide which input packet will be scheduled to the output. This paper introduces a Round-robin arbitration with adjustable weight of resource access time. The Round-robin arbiter mechanism is useful when no starvation of grants is allowed. The arbiter quantizes time shares each requestor is allowed to have. A minimal fairness is guaranteed by granting requestors in Round-robin manner. The requestors can prioritize their time shares by the weight. For example, if requestor A has a weight of two and requestor B has a weight of four, arbiter will allocate requestor B with time slice two times longer than that of requestor A's. The verification of the design is carried out using SystemVerilog. The inputs of the arbiter are randomized, outputs are predicted in a software model and verification coverage is collected. The work in this paper includes design and verification of a weighted Round-robin arbiter.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This paper is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Aung Toe

August 2018

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor/mentor, Mark A. Indovina, for the continuous support throughout the project. Under his mentor-ship, not only did I learn to love complex digital systems but also to become a good engineer.

I would also like to thank my professor, Dorin Patru, for introducing me to various practical design concepts during my coursework at Rochester Institute of Technology (RIT).

My sincere thanks to all my friends who shared my journey at RIT for the sleepless nights working together before deadlines.

Last but not least, I want to thank David Coumou, Aaron Radomski and Daniel Gill for their patience and guidance in my career development.

Contents

List of Figures

List of Listings

List of Tables

Chapter 1

Introduction

Scheduling algorithms are required when multiple requestors require access to a shared resource. In a System on Chip (SoC), multiple devices in the chip are needed to work together. As a result, an SoC may have multiple bus masters. A fast and powerful arbiter becomes important to service all the bus masters. Another example of arbitration system application is network switches. In a network switch, packets from multiple input ports need to go through a single output port[4, 5]. As the number of parallel processes increases, accessing a shared resource becomes the bottleneck in performance[6]. One of the goals of a scheduler is to maximize throughput. The throughput of a system can be maximized by minimizing wait time for each request. The advantages of utilizing arbiters include access fairness for the requestors to the resources, utilization without wasting cycles, re-usability, arbitration speed, power and resource overhead[7, 8]. Different types of Round-robin arbiters such as baseline arbiters, time speculative arbiters, acyclic arbiters, parallel prefix arbiters, priority based arbiters, etc... are used in various applications[9].

In the case of multiple bus masters, all masters require access to a shared resource at the same or similar level of priority. A Round-robin arbitration mechanism fits the

application of fairness without starving the requestors. A Round-robin arbiter allocates fixed time slices for the masters for each Round-robin turn. This time slice limitation allows the predictability of worst-case time when the grant will get granted[10].

In this paper, a Round-robin arbiter is designed using weight decoder, next grant precalculate logic and granting logic. The weight of each granted requestor is decoded using a weight decoder logic. Based on the current grant, the next possible grant is precalculated in Round-robin mechanism. Finally, the granting logic checks for requests and precalculated next grant mask to select a single grant.

Weight decoder, next grant precalculator and grant state-machine are designed to be configurable. The number of requestors and the bit width of the weight can be set before synthesis. The request inputs and grant outputs are in one-hot format. The weights requested by requestors are concatenated in a weight bus. The weight decoder logic dictates the grant logic how long each grant is needed for each request. The next grant calculator algorithm enforces the grant logic to be in Round-robin order. The calculator acts like a record book to keep track of current grant and next possible grants. Every clock cycle, the grant logic state-machine services appropriate grant by checking the weight, current grant, requests and precalculated next grant.

1.1 Organization

This paper is organized as follows:

- Chapter 2 discusses different arbitration algorithms.
- Chapter 3 discusses the design and implementation of weighted Round-robin arbiter.
- Chapter 4 discusses the design and implementation of SystemVerilog verification.

- Chapter 5 discusses the test results and statistics of 8 port arbiter.
- Chapter 6 is the conclusion and possible future work.

Chapter 2

Background Research

Arbiters play an important role when multiple requests are sent to access a single resource. In a network switching router, the packets received on input ports are sent out to the respective output ports. The arbiter acts a middle man to direct which input gets to send its packet to the designated output. The arbitration speed of the arbiter has a large factor in determining the speed of switching performance. Of the many metrics to benchmark an arbiter, fairness is a good unit to measure the performance, and there are a few types typically utilized:

- Weak Fairness
- Weighted Fairness
- Last Served Lowest Priority (LSLP)

Weak Fairness means a request may have to wait indefinitely until it gets served. There may be higher priority requestors holding on to the grant. Section 2.1 discusses a Fixed Priority Arbiter that demonstrates the weak fairness metric. The Lottery Arbiter discussed in Section 2.2 updates the weight of the lottery ticket as it arbitrates. The weight of the

ticket increases the chances of winning the grant. This algorithm has the property of Weighted Fairness. The Matrix Arbiter in Section 2.3 has the property of LSLP. The last served requestor will have the lowest priority in the arbiter.

2.1 Fixed Priority Arbitration

Fixed priority arbiter is the simplest form of arbiter. It is also known as per-emptive arbiter due to the nature of its scheduling algorithm. Each master is given a priority from high to low. As shown in Eqn 2.1, master $i-1$ has higher priority than master i[7, 9]. For master i to get the grant, all the masters higher priority than master i must not be requesting to the arbiter.

$$grant_i = \overline{req_0} \cdot \overline{req_1} \cdot \overline{req_2} \cdots \overline{req_{i-1}} \cdot req_i \tag{2.1}$$

For example, if there are 3 masters, master 0 is given priority 0, master 1 priority 1 and master 2 priority 2. Grant is given to the master that has the highest priority. If master 0 and master 1 request at the same time, master 0 will get the grant since it has higher priority. As a result, a higher priority master can starve other masters by monopolizing the bus. However, due to the simplicity of the design, fixed priority arbiters are very useful in applications where high priority tasks need immediate servicing and low priority tasks can wait indefinitely to get grant.

2.2 Lottery Arbiter

Lottery arbitration scheduling is based on the weighted probabilistic distributions. The algorithm utilizes a lottery manager to mange the drawing of grants. As in Figure 2.1,

lottery manger gives a numbered ticket/request to each master. The weight of the ticket number is increased each time a specific master requests.

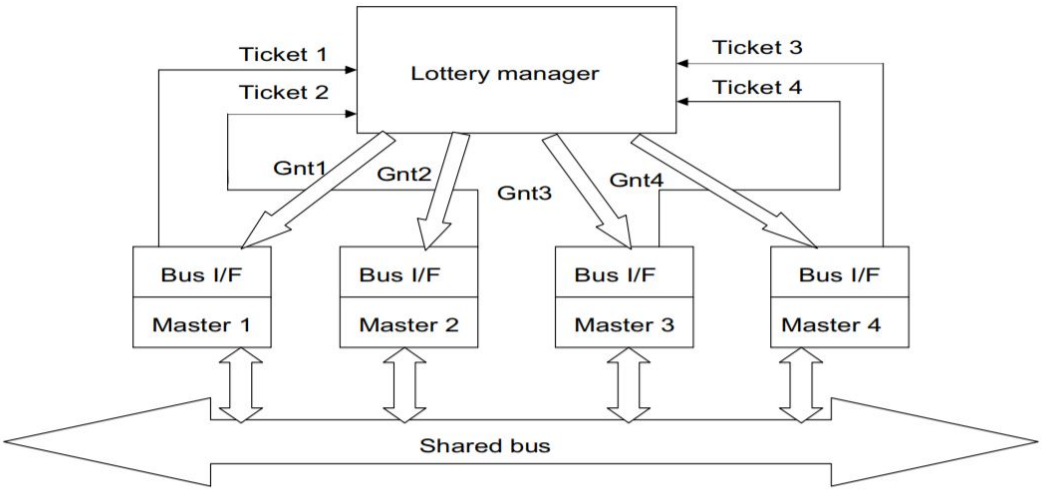

Figure 2.1: Lottery Manager [1]

Assuming a non-empty set of weights $\{w_1, w_2, ..., w_n\}$, the probability of winning a ticket can be calculated as in Eqn 2.2.

$$p_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{2.2}$$

The manager draws the highest numbered ticket as a winner. The ticket count of the granted master is reset on winning the lottery. The reset makes the current winner less likely to be chosen on the next draw. In case of a tie, the manager may choose any master. If there is only one master requesting, the manger will choose the trivial solution. As a result of this pseudo-randomization, the masters get a fair share of bus time dictated by the weight of the lottery ticket.

2.3 Matrix Arbiter

Matrix arbiters are designed to enforce last served master to have the lowest priority on the shared resources. It keeps track of the priority in a square matrix form. The rows and columns of the matrix represents the requestors. The i^{th} row can be linked to requestor i and j^{th}column requestor j. Figure 2.2 shows the 4 requestors mapping in a 4 by 4 matrix.

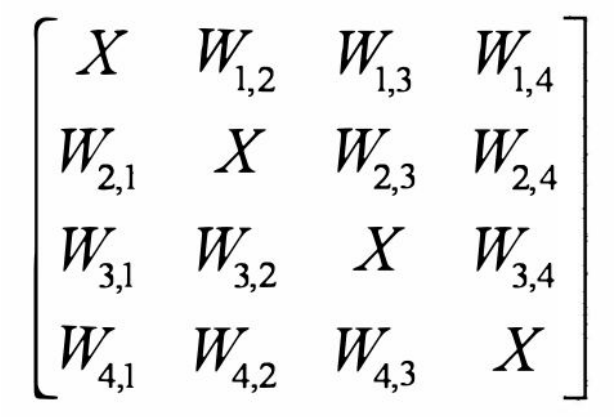

Figure 2.2: Priority Matrix[2]

The rule of the matrix arbiter is if there is a 1 in i^{th} row and j^{th} column, requestor i has priority over requestor j. As in Figure 2.3, if requestor 2 sends a request, the grant will be issued to requestor 2. The elements in row 2 are set to zero. It forces requestor 2 to have the lowest priority. At the same time the elements in column 2 are set to 1. It makes other requestors beat requestor 2 in the next iteration. Matrix arbiters are useful when the number of inputs are small. If the number of requests increases, the structure of the arbiter increases leading to larger area overhead[11].

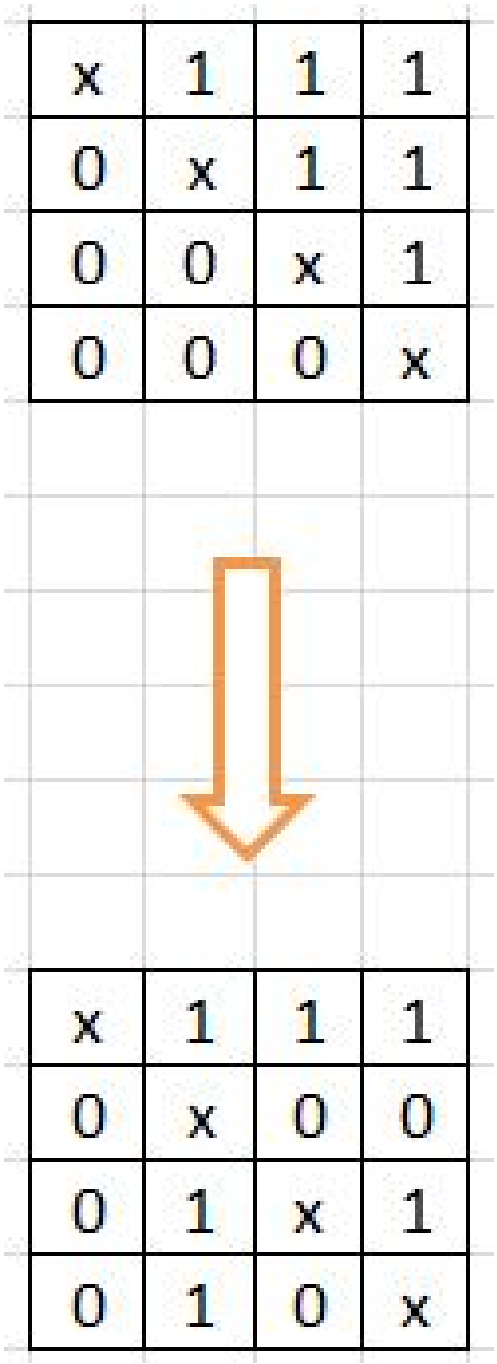

Figure 2.3: Matrix Transition

Chapter 3

Weighted Round-Robin Arbiter

Round-robin arbitration has multiple flavors to fit the desired application. In some applications two-pick Round-robin arbiters are used instead of one pick arbiters[12]. However, the final goal, starvation prevention and statistical fairness, is the same[13]. The algorithms introduced in Chapter 2 give the grant to the master that has the higher priority. It means a master has the ability to monopolize the bus for a long time. This causes bus starvation to the masters with lower priorities. Weighted Round-robin arbiter design is based on the algorithm that the scheduling of grants must go on in a Round-robin manner. This work is based on a two-step approach. The arbiter monitors the requests and give them grants in the next clock. In best case condition, the request at time t_i will get serviced at time t_{i+1}[14]. This scheduling algorithm makes sure each master gets its share of time slice in a fair amount of time. A good analogy would be if there are 4 masters in x cycle arbiter, each master will get a quantized time slice of x/4 cycles. However, in some applications, one bus master may require more bus time than others. Figure 3.1 shows top level view of Round-robin arbiter in a network packet switching system.

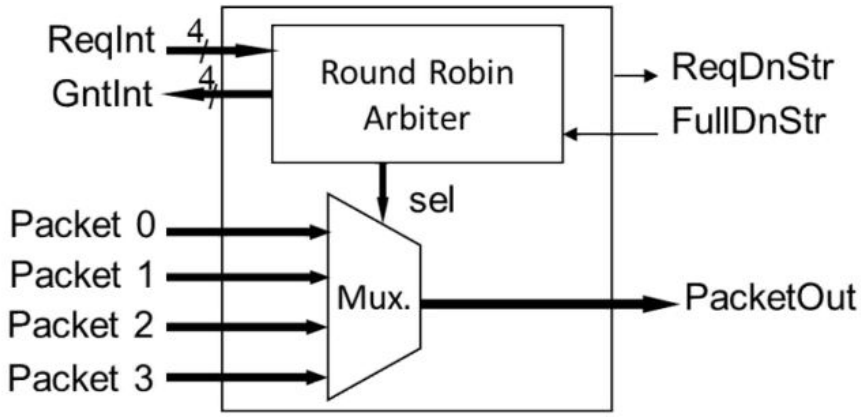

Figure 3.1: Packet Switching Architecture

This paper introduces another configurable variable called weight. The weight of each master can be defined as the grant time slice that the master can configure in the arbiter. If all the masters have the same amount of weight, each master will get an equal time share of the pie. If master A requests 20 cycles and master B requests 10 cycles, master A will get grant 2 times longer than master B. One disadvantage of letting the masters to configure the weight is a master may configure a very large weight. To reduce this large weight monopoly, another global configurable maximum allowed weight is added. A master may request a very large weight value, but the arbiter will only grant up to the maximum allowed weight if there are other masters waiting.

3.1 Weight Decoder

Weight decoder decodes one-hot grant to decode the correct weight of the granted master. As shown in Figure 3.2, the weight of the masters are concatenated to form a weight bus. As in Eqn 3.1, the width of the bus can be calculated by the width of a single weight and the total number of masters in the arbiter.

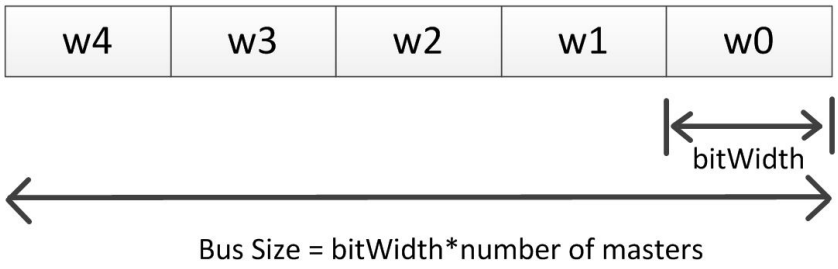

Figure 3.2: Weight Bus Size

$$busWidth = weightWidth * numOfMasters \tag{3.1}$$

Weight decoder takes current grant as an one-hot input. The input grant is decoded to produce an index for correct bit slice positions of the weight bus. Figure 3.3 shows the flowchart to produce the correct index. For example, if the grant is $b'0010$, the index output is 1. Index of 1 stands for the master no. 1. The weight of master 1 is decoded as an output. If the grant is $b'0100$, index output is 2. The weight of master 2 is decoded as an output.

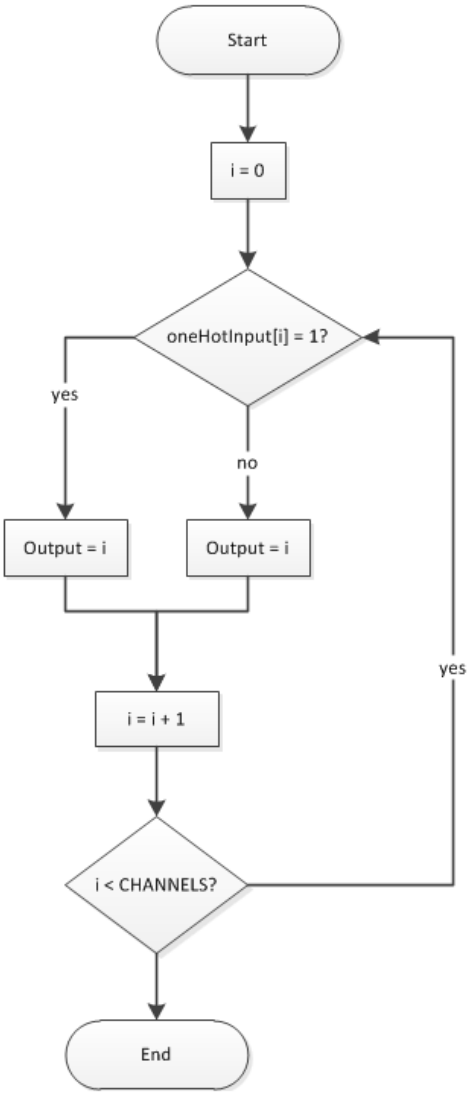

Figure 3.3: One-hot Index Flowchart

3.2 Next Grant Precalculator

Next Grant PReCalculator(NGPRC) calculates the next possible grants mask based on current grant. By precalculating the next possible grants, NGPRC dictates the Round-robin arbitration of the arbiter. As in Figure 3.4, if all 4 masters in the arbiter are

requesting and current grant is master 1, next possible grant is restricted to be in the order of master 2, master 3 and master 0. The arbiter cannot skip master 2 to grant master 3. It would violate Round-robin scheme and it is not allowed. By giving next possible grant priority to the Grant State-machine, it forces the grant to be in strict Round-robin order.

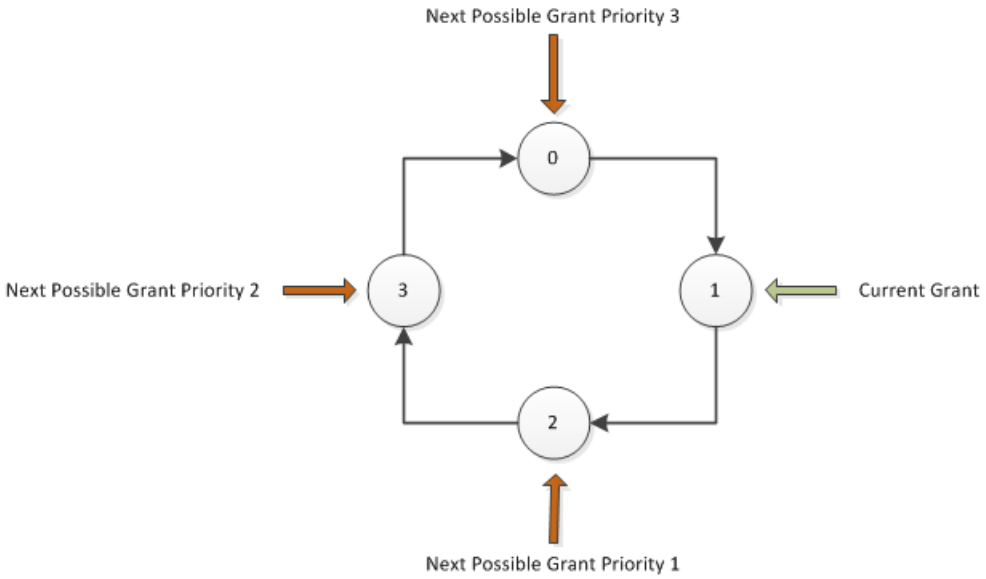

Figure 3.4: Next Grant PreCalculator

Figure 3.5 shows the calculation steps NGPRC takes to compute the next possible grant priority. For example, if current grant is $b'0010$, rotate left gives $b'0100$. After inversion, the bits become $b'1011$. After increment by 1, the next possible grant becomes $b'1100$. It means the leftmost 2 bits are in line in priority.

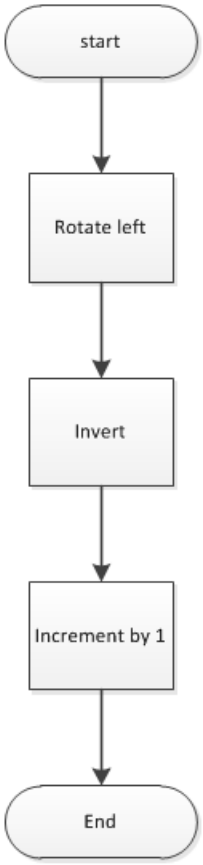

Figure 3.5: NGPRC Calculation Steps

3.3 Grant State Machine

Grant state machine is the logic to calculate which master gets the grant and for how long based on the weight. The grant logic is based on the requests and next grant priority mask created by NGPRC. Figure 3.6 shows the state flow diagram of grant state machine. "Grant Process" state masks requests using precalculated mask to grant the next requesting master. After the grant is decided, it moves to "Get Weight" state to fetch the weight of the grant from Weight Decoder. After that, it moves to "Count" state to count the clock cycles until local counter reaches the desired weight.

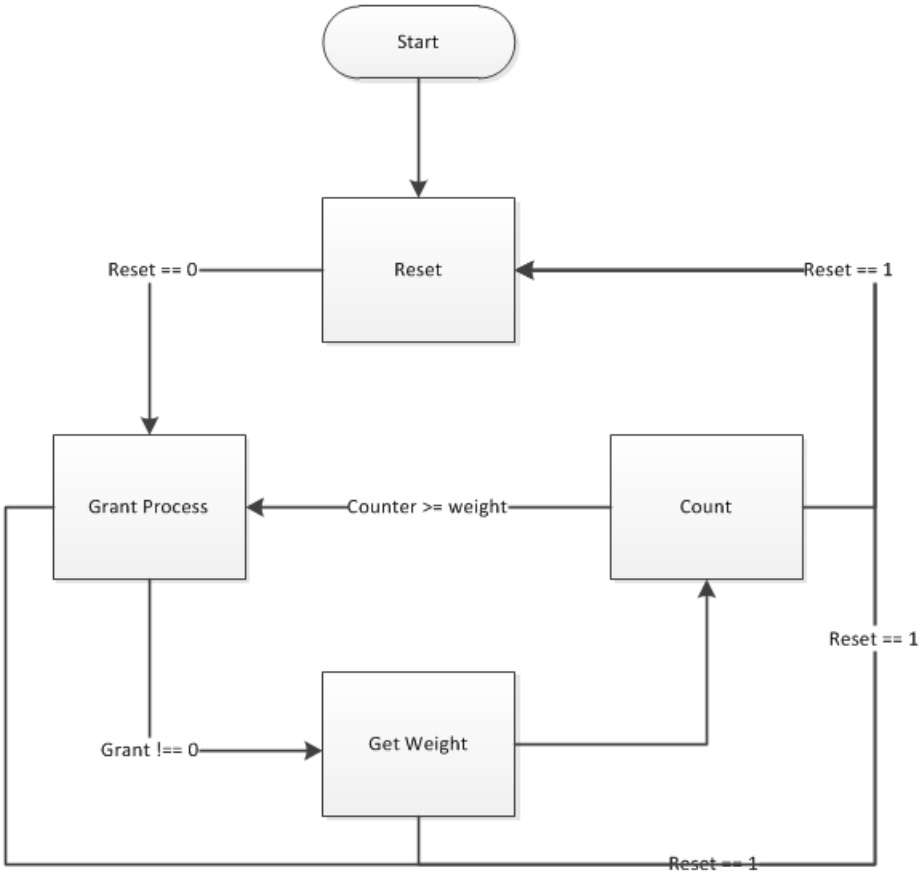

Figure 3.6: Grant State Machine

Chapter 4

SystemVerilog Verification Design

As the sizes and complexity of electronic design increases, faster integration of design and verification of complex systems become mandatory[15]. The beginning of a new feature starts with architectural exploration and ends with functional verification. The studies find that the verification of a design occupies the most amount of time in a project life-cycle[16]. The time required to verify a design from the end of a design life-cycle can be defined as a verification gap. As engineers reuse Intellectual Property (IP) cores, design engineers can produce complex features in short time. However, these new features are still needed to be verified. As a result, the verification gap increases as the product cycle rotates between debugging and verification. To reduce the gap, design engineers and verification engineers have to come to an agreement of having a universal verification process[17]. SystemVerilog language is introduced as a common language to design verification environment as a Universal Verification Methodology (UVM). A unified verification methodology is important because about 70% of the design cycle time is used to develop verification environment[18]. The re-usability of the verification environment shortens the verification gap. A general verification includes multiple layers that can be re-used with

minimal change to the design[19]. As in Figure 4.1:

- Test Layer : Different test cases with constraint random and/or direct stimulus.
- Scenario Layer : Generates random stimulus based on the test cases.
- Functional Layer : This layer predicts possible Device Under Test (DUT) outputs (golden test vectors) based on the random inputs using a reference mode[20]. It may also contain a scoreboard to keep track of the results.
- Command Layer : Command layer is a pin-level layer. On the input side, it receives stimulus from functional layer and drives the DUT. DUT output is also monitored to be compared with golden test vectors.

SystemVerilog is used to verify the proposed Round-robin arbiter. SystemVerilog verification can be designed in an object oriented way to allow classes, inheritance, class routine sharing (polymorphism). It has the ability to randomize inputs and set constraints on the randomness. Assertions are used to check for undesired behavior. The functional converge sampling bins can be obtained through the defined coverage points. Mailboxes and event triggers can be used to control synchronization between the modules. One of the main advantages of developing SystemVerilog environment is it allows engineers to reuse the classes and modules for a different project. The proposed SystemVerilog environment contains requestor, generator, agent, driver, monitor, checker and scoreboard. Figure 4.1 shows the top level view of proposed SystemVerilog test environment.

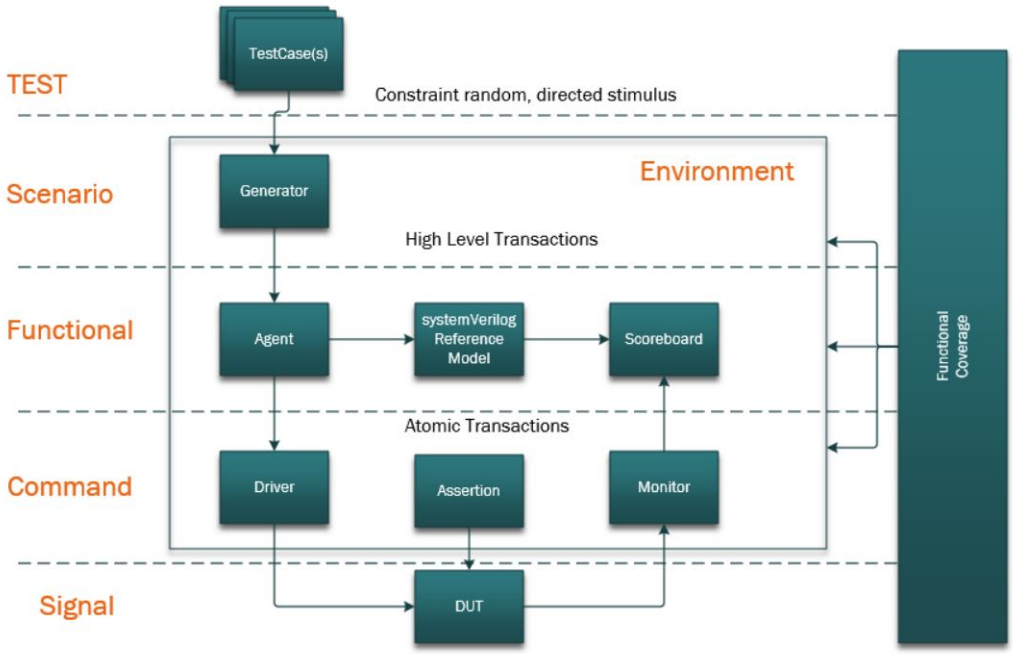

Figure 4.1: SystemVerilog Environment

4.1 Requestor

Requestor class is designed to behave like a master that would request access to the shared resource from the arbiter. Requestor class contains two members, request and weight. The members are of the type "rand". It allows the calling class to be able to randomize to provide random stimuli to the DUT. Having random test cases is important because the verification engineers might not be able to consider many combinations of test cases for complex systems. However, randomization can produce test cases not applicable to DUT. Adding constraints to the random variable makes the random stimuli applicable to the target DUT verification. Verification engineers spend most of the time on iterating runs by adding constraints and various random seeds. To achieve full coverage, some direct test

cases may be required to fully close the loop. Figure 4.2 shows the functional coverage loop.

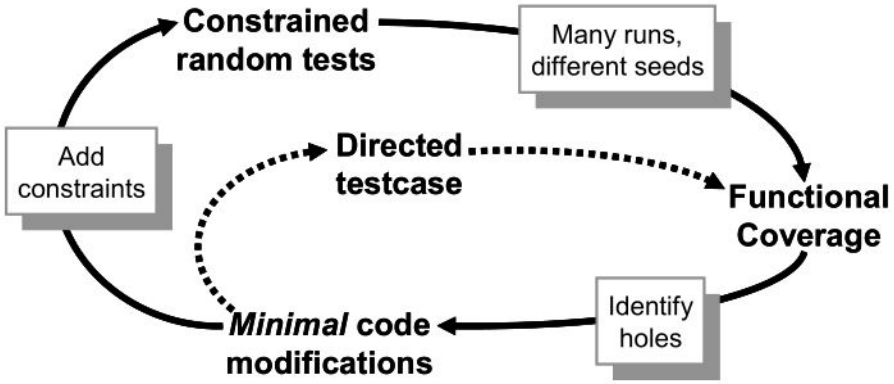

Figure 4.2: Function Coverage Loop [3]

4.2 Generator

Generator is a part of Scenario Layer. The purpose of the generator is to create different test stimuli based on scenarios. For example, the verification of a TV remote have many test scenarios. One of the test scenarios could be pressing mute button. A different test scenario could be changing channels or adjust volumes. In case of Round-robin arbiter, random requests/non-request could be simulated with different weights. The requestor class is instantiated in generator. The generator class generates different requests by randomizing the requestor class. Generator module can be used to generate different random stimuli according to the different test case scenarios.

4.3 Agent

Agent is located in functional level. As in Figure 4.1, agent acts as a mediator between Scenario Layer and Command Layer. At functional level, agent class is responsible to receive stimuli and predict output of DUT according to that stimuli. To achieve that, agent class usually has a functional model of Device Under Test. The input stimuli and predicted output test vectors can be called "golden" test vectors. Engineers can analyze the input test vector set to predict the expected output results.

In this verification environment, the agent class instantiates the generator class. The stimuli generated by the generator are used to convert to golden test vectors by predicting the expected outcome of the given stimuli. Each randomization contains a bit representing request or no-request accompanied by the weight. The model checks for the bit and if the bit is request bit, the weight value is recoded as the number of clock cycles the bit should be granted. If the bit is no-request the the output is recorded as no request with zero cycles. The test vectors are passed onto the driver module and expected golden vectors to checker module.

4.4 Driver

The driver located in Command Layer is closest to hardware. Driver class connects the input of DUT to the rest of the verification environment. It is responsible to drive signals synchronously to DUT. It can contains functions such as reset conditions. The test vectors from agent are driven to DUT by the driver in synchronous with system clock. In this verification environment, request bits and weights are driven to DUT using DUT system clock. Because driver controls the input of DUT, Monitor class described in the following section needs to know when the driver finished driving a particular test vector. The

synchronization between class modules is done using event triggering. After driver finished sending a set of test vector to DUT, it raises an event for monitor class to catch. Monitor class uses this event to know the respective DUT output. By having synchronization, driver can insure that all the input test vectors are aligned correctly with the expected golden test vectors.

4.5 Monitor

Similar to driver class, monitor class resides in Command Layer. Monitor class is connected to the output of DUT to capture data from DUT output ports synchronously. The purpose of monitor is to record DUT output for each input driven by driver. The recorded data can be transferred to checker module to check for errors.

In this verification environment, monitor module starts counting once it sees a grant signal of a request. The number of cycles or the amount of time a request is granted can be determined by the counter value of the monitor. For example, if a requestor 1 is granted for 5 clock cycles, monitor will get a count of 5 for requestor 1. In other words, monitor collects the grants and their respective granted cycles. The collected data is sent to the check to be matched with the golden test vectors.

4.6 Checker

Like Agent class, Checker class is located in Functional Layer. Checker class is responsible to match the output of DUT collected by monitor and the golden test vectors predicted by reference model. In this verification environment, checker class receives grants and grant time (cycles) from monitor module. It also receives the expected output of golden test

vectors. For each test data set, checker verifies DUT output against golden test vectors. It also records the verification statistics in scoreboard.

4.7 Scoreboard

As the name stands, Scoreboard module contains the statistics of current verification. Checker module calls the "record" member function of scoreboard after each test vector. The function records which requestor (master) is granted by updating member variables. Using the recorded data, statistical analysis can be performed.

4.8 Determinism

If a bug were found during the verification process, it is important for the design and verification engineers to be able to reproduce easily. Knowing the sets of test vectors caused the failure is crucial for debugging purposes. Therefore, when the requestor module is instantiated in Generator module, the seed of the requestor's random variables can be set. Each "randomize" function call is based on a different seed. If a failure occurs, the seed of the failure test vectors can be extracted for the design engineers to debug. It allows verification engineers to reproduce the failing inputs easily without restarting the whole verification process. This approach of seeding makes the randomization process to be deterministic every iteration in every run.

Chapter 5

Tests and Results

This section discusses the simulation results of the arbiter, as well as the area and power overhead of the top level design.

5.1 Simulation

5.1.1 Weight Decoder

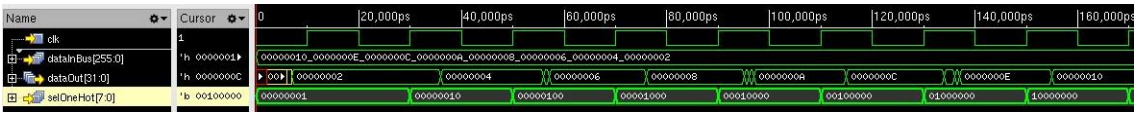

Figure 5.1: Weight Decoder

Figure 5.1 shows the simulation of weight decoder. Input dataInBus contains the weights of the channels preconfigured. Input selOneHot port/grant the input used to decode the weight of current grant. The decoded weight is outputted to the dataOut port.

5.1.2 Next Grant PreCalculator

Figure 5.2: Next Grant PreCalculator

Figure 5.2 shows the simulation waveform of Next Grant PreCalculator. Based on the input request and grant, next grant mask is created to dictate Round-robin order to restrict the grant order.

5.1.3 Round-Robin Top Level/Grant State Machine

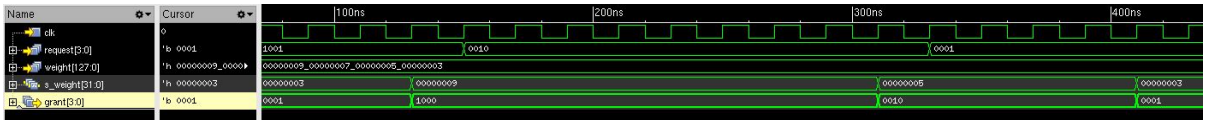

Figure 5.3: Top Level Simulation

The Figure 5.3 shows the simulation top level Round-Robin Arbiter. Grant is serviced based on the requests and precalculated mask from NGPRC. The grant is given the access time for the number of weight cycles before servicing the next request.

5.2 SystemVerilog Verification Results

5.2.1 Scoreboard scores

Table 5.1 shows the results of SystemVerilog verification of 20,000 iterations. As seen in the table, the hit score distribution is fairly uniform since the randomization of test vector generation is based on uniform probability distribution.

Table 5.1: SystemVerilog Verification Hit Scores

	Score
Channel 0	1021
Channel 1	972
Channel 2	1036
Channel 3	980
Channel 4	1048
Channel 5	995
Channel 6	1009
Channel 7	971

5.3 System Overhead

5.3.1 Area Overhead

The arbiter is synthesized using a TSMC 65 nm technology library and Synopsis Design Compiler using a two step process. The first step, RTL synthesis, performs logic synthesis and produces what is called a pre-scan netlist. In the second step, test synthesis, Design For Test (DFT) structures for full scan testing are added to the design and optimized producing what is called a post-scan netlist. The top level design is targeted at 8 channels to be able to arbitrate 8 different requestors. Table 5.2 shows the pre-scan and post-scan area overhead for Weight Decoder, Next Grant PreCalculator and Grant Statemachine.

5.3.2 Power Overhead

Similarly, Table 5.3 shows power overhead for Weight Decoder, Next Grant PreCalculator and Grant state machine.

Table 5.2: Area Overhead

Module	**Pre-Scan**			
	Combinational (µm²)	Non-Combinational (µm²)	Percent Total (%)	Total (µm²)
Weight Decoder (MUX)	612.0576	0.0000	13.4	612.0576
Next Grant PreCalculator (NGPRC)	755.0928	691.8912	31.8	1446.9840
Grant Statemachine (Grant)	1167.5664	1327.2336	54.8	2494.8000
Top Level Arbiter Module (8 Channels)	2534.7168	2019.1248	100	4553.8416
	Post-Scan			
Weight Decoder (MUX)	612.0576	0.0000	12.4	612.0576
Next Grant PreCalculator (NGPRC)	755.0928	828.2736	32.0	1583.3664
Grant Statemachine (Grant)	1167.5664	1580.0401	55.6	2747.6065
Top Level Arbiter Module (8 Channels)	2534.7168	2408.3137	100.0	4943.0305

Table 5.3: Power Overhead

Netlist Type	**Group**	**Internal**	**Switching**	**Leakage**	**Total**	**Percentage**
Pre-Scan Netlist	Register	9.9823e-02 mW	4.3251e-03 mW	9.9996 nW	0.1042 mW	76.50 %
	Combinational	1.1323e-02 mW	2.0664e-02 mW	9.8877 nW	3.1997e-02 mW	23.50 %
	Total	0.1111 mW	2.4989e-02 mW	19.8873 nW	0.1362 mW	100 %
Post-Scan Netlist	Register	0.1240 mW	9.2250e-03 mW	11.8287 nW	0.1332 mW	74.45 %
	Combinational	1.7618e-02 mW	2.8082e-02 mW	9.8877 nW	4.5710e-02 mW	25.55 %
	Total	0.1416 mW	3.7307e-02 mW	21.7164 nW	0.1789 mW	100 %

Chapter 6

Conclusions

This work discussed the design and verification of Round-Robin Arbitration. As the number of masters requires access the shared resources increases, a good arbitration system becomes essential. This work focused on the fairness metric to measure the performance of an arbiter. However, it is important to not overlook the area overhead and power consumption. A complex system may have good fairness at the trade-off of large overhead. Therefore, the design preference is based on the application of the arbiter. Round-robin arbiter is chosen because of its fairness in granting access. Bus grant time quantization allows the requestors to be able to predict the maximum amount of time to get grant. However, in some applications, one requestor might require to have the grant twice as long. This work introduced the weight or the number of clock cycle that the requestor can configure during synthesis. This makes the fairness adjustable (more fair or less fair). At the same time, increasing or decreasing the weights allows the time quantization adjustable.

SystemVerilog is used as a verification environment for the design. By having a verification environment, engineers can have a higher confidence on the release of the product. Code maintenance is easier for each time the design changes due to bugs or feature intro-

duction as the verification can filter out issues before reaching to the customers. This work discussed that design process from the aspect of customer requirements and applications.

To summarize, although fairness is used as a performance measurement, fairness is only good when a particular application requires it. This Round-Robin Arbiter design is customized to be more fair or less fair. The design trade-off between fairness and/or overhead remains at the process of the intended design application.

6.1 Future Work

Since the arbiter are application specific, for future work, this implementation of Round-robin arbiter can be modified to suit the intended usage. One of the applications of Round-robin arbiter is system-on-chip shared memory. In this application, two independent Round-robin arbiters are used, one for address and one for data. For read access, the two arbiters can operate independently[21]. However, for write back operations, both the data and address needs to go together[22]. It might be beneficial to implement a modified version that is aware of the condition when address or data arbiter needs to freeze in order to write back.

Another applications is communication arbiter for Network-On-Chip (NOC), where communication between IP cores are usually non-uniform or hot-spot in traffic[22, 23]. The arbiter in this work only allow a fixed time slice preconfigured. It would be beneficial to implement logic to detect the load of the inputs and adjust priority dynamically. By adjusting priority or grant time based on the traffic would make sure that busy master/requestor traffic is well balanced and not starved.

References

[1] K. Warathe, D. Padole, and P. Bajaj, "A Design Approach to AMBA (Advanced Microcontroller Bus Architecture) Bus Architecture with Dynamic Lottery Arbiter," in *2009 Annual IEEE India Conference*, Dec 2009, pp. 1–4.

[2] Z. Fu and X. Ling, "The design and implementation of arbiters for Network-on-chips," in *2010 2nd International Conference on Industrial and Information Systems*, vol. 1, July 2010, pp. 292–295.

[3] C. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. New York NY: Springer, 2006.

[4] Y. Li, N. Zeng, W. N. N. Hung, and X. Song, "Enhanced symbolic simulation of a round-robin arbiter," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Oct 2011, pp. 102–107.

[5] S. Q. Zheng and M. Yang, "Algorithm-Hardware Codesign of Fast Parallel Round-Robin Arbiters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 1, pp. 84–95, Jan 2007.

[6] M. Abdelrasoul, M. Ragab, and V. Goulart, "Impact of Round Robin Arbiters on

router's performance for NoCs on FPGAs," in *2013 IEEE International Conference on Circuits and Systems (ICCAS)*, Sept 2013, pp. 59–64.

[7] Y. Yang, R. Wu, L. Zhang, and D. Zhou, "An Asynchronous Adaptive Priority Round-Robin Arbiter Based on Four-Phase Dual-rail Protocol," *Chinese Journal of Electronics*, vol. 24, no. 1, pp. 1–7, 2015.

[8] K. A. Helal, S. Attia, T. Ismail, and H. Mostafa, "Priority-select arbiter: An efficient round-robin arbiter," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, June 2015, pp. 1–4.

[9] R. Kamal and J. M. M. Arostegui, "RTL implementation and analysis of fixed priority, round robin, and matrix arbiters for the NoC's routers," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, April 2016, pp. 1454–1459.

[10] E. S. Shin, V. J. Mooney, and G. F. Riley, "Round-robin Arbiter Design and Generation," in *15th International Symposium on System Synthesis, 2002.*, Oct 2002, pp. 243–248.

[11] M. Oveis-Gharan and G. N. Khan, "Index-Based Round-Robin Arbiter for NoC Routers," in *2015 IEEE Computer Society Annual Symposium on VLSI*, July 2015, pp. 62–67.

[12] H. F. Ugurdag, F. Temizkan, O. Baskirt, and B. Yuce, "Fast two-pick n2n round-robin arbiter circuit," *Electronics Letters*, vol. 48, no. 13, pp. 759–760, June 2012.

[13] K. C. Lee, "A variable round-robin arbiter for high speed buses and statistical multiplexers," in *[1991 Proceedings] Tenth Annual International Phoenix Conference on Computers and Communications*, Mar 1991, pp. 23–29.

[14] K. Yoghigoe, K. J. Christensen, and A. Roginsky, "Design of a high-speed overlapped round robin (ORR) arbiter," in *Local Computer Networks, 2003. LCN '03. Proceedings. 28th Annual IEEE International Conference on Local Computer Networks*, Oct 2003, pp. 638–639.

[15] S. Marconi, E. Conti, J. Christiansen, and P. Placidi, "Reusable SystemVerilog-UVM design framework with constrained stimuli modeling for High Energy Physics applications," in *2015 IEEE International Symposium on Systems Engineering (ISSE)*, Sept 2015, pp. 391–397.

[16] D. Rich, "The unique challenges of debugging design and verification code jointly in SystemVerilog," in *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, Sept 2013, pp. 1–7.

[17] M. Rashid, M. W. Anwar, and F. Azam, "Expressing embedded systems verification aspects at higher abstraction level - SystemVerilog in Object Constraint Language (SVOCL)," in *2016 Annual IEEE Systems Conference (SysCon)*, April 2016, pp. 1–7.

[18] P. Gurha and R. R. Khandelwal, "SystemVerilog Assertion Based Verification of AMBA-AHB," in *2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, Sept 2016, pp. 641–645.

[19] R. Sethulekshmi, S. Jazir, R. A. Rahiman, R. Karthik, S. Abdulla M, and S. Sree Swathy, "Verification of a RISC processor IP core using SystemVerilog," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, March 2016, pp. 1490–1493.

[20] Y. Zhu, T. Li, J. Guo, H. Zhou, and F. Fu, "A novel low-cost interface design for Sys-

temC and SystemVerilog Co-simulation," in *2015 IEEE 11th International Conference on ASIC (ASICON)*, Nov 2015, pp. 1–4.

[21] R. Bhaktavatchalu, B. S. Rekha, G. A. Divya, and V. U. S. Jyothi, "Design of AXI bus interface modules on FPGA," in *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, May 2016, pp. 141–146.

[22] J. Reed and N. Manjikian, "A dual round-robin arbiter for split-transaction buses in system-on-chip implementations," in *Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513)*, vol. 2, May 2004, pp. 835–840 Vol.2.

[23] A. A. Khan, R. N. Mir, and N. ud din, "Buffer aware arbiter design to achieve improved QoS for NoC," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, Nov 2017, pp. 2494–2499.

[24] J. Bromley, "If SystemVerilog is so good, why do we need the UVM? Sharing responsibilities between libraries and the core language," in *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, Sept 2013, pp. 1–7.

[25] J. Wang, Y. Li, Q. Peng, and T. Tan, "A dynamic priority arbiter for Network-on-Chip," in *2009 IEEE International Symposium on Industrial Embedded Systems*, July 2009, pp. 253–256.

Appendix I

Arbiter Source Code

Listing I.1: Weight Decoder Module

```
// MUX module
// This module selects one of the inputs according to the input select
    signal
// Combinational Logic
// Input : selOneHot - signal ONE HOT STYLE
//       : daataInBus - input data bus for all channels concat from
   ch0  to
//        chN
// Output : dataOut - output data according to select signal
//

module MUX
        #(
            parameter WIDTH = 32,       // width of each channel
            parameter CHANNELS = 8      // number of channels
        )
        (
            reset,
            clk,
            scan_in0,
            scan_in1,
            scan_in2,
            scan_in3,
            scan_in4,
            scan_enable,
```

```
            test_mode,
            scan_out0,
            scan_out1,
            scan_out2,
            scan_out3,
            scan_out4,
            selOneHot,                  // one hot select input
            dataInBus,                  // input bus
            dataOut                     // output data
        );

input
    reset,                          // system reset
    clk;                            // system clock

input
    scan_in0,                       // test scan mode data input
    scan_in1,                       // test scan mode data input
    scan_in2,                       // test scan mode data input
    scan_in3,                       // test scan mode data input
    scan_in4,                       // test scan mode data input
    scan_enable,                    // test scan mode enable
    test_mode;                      // test mode

output
    scan_out0,                      // test scan mode data output
    scan_out1,                      // test scan mode data output
    scan_out2,                      // test scan mode data output
    scan_out3,                      // test scan mode data output
    scan_out4;                      // test scan mode data output

input [(CHANNELS-1) : 0] selOneHot;             // one hot select input
input [(CHANNELS*WIDTH)-1 : 0] dataInBus;  // input data bus

output reg
    [(WIDTH-1) : 0] dataOut;                // output data after select

//reg [(CHANNELS*WIDTH)-1 : 0] tempData;

// generate variable
genvar gv;
```

```
//--- COMBINATIONAL SECTION ---//

// temporary array to hold input channels
wire [(WIDTH-1) : 0] inputArray [0 : (CHANNELS-1)];

generate
    // generate statement to assign input channels to temp array
    for(gv = 0; gv < CHANNELS; gv = gv+1) begin : arrayAssignments

        assign inputArray[gv] = dataInBus[ ( (gv+1)*WIDTH )-1 : (gv*
           WIDTH)];


    end // arrayAssignments
endgenerate

// function to convert one hot to decimal
function integer decimal;
input [CHANNELS-1 : 0] oneHotInput;
integer i;
for(i = 0; i<CHANNELS; i = i+1)
    if(oneHotInput[i])
        decimal = i ;

endfunction

// select the output according to input oneHot
always@*
begin
    dataOut = inputArray[decimal(selOneHot)];
end // end always

endmodule // MUX
```

Listing I.2: Weight Decoder Test Module

```
module test;
localparam WIDTH = 32;
localparam CHANNELS = 8;

// clock period
localparam CLOCK_PERIOD = 20; //20ns (50Mhz)


wire  scan_out0, scan_out1, scan_out2, scan_out3, scan_out4;
reg  clk, reset;
reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4, scan_enable,
   test_mode;

// inputs
reg [(CHANNELS-1) : 0] test_selOneHot;
reg [(CHANNELS*WIDTH)-1 : 0] test_dataInBus;

//output
wire [(WIDTH-1) : 0] test_dataOut;

// flow control flags
integer testDoneFlag = 0;
integer i = 1;
integer j = 0;
integer k = 0;
integer waveCounter = 1;

// temp reg/variables
reg [(WIDTH-1) : 0] tempDataIn;
reg [(WIDTH-1):0] dataArray [(CHANNELS-1) : 0]; // array to check
   output

MUX top(
        .reset(reset),
        .clk(clk),
        .scan_in0(scan_in0),
        .scan_in1(scan_in1),
        .scan_in2(scan_in2),
        .scan_in3(scan_in3),
        .scan_in4(scan_in4),
```

```
        .scan_enable(scan_enable),
        .test_mode(test_mode),
        .scan_out0(scan_out0),
        .scan_out1(scan_out1),
        .scan_out2(scan_out2),
        .scan_out3(scan_out3),
        .scan_out4(scan_out4),
        .selOneHot(test_selOneHot),
        .dataInBus(test_dataInBus),
        .dataOut(test_dataOut)
    );


initial
begin
    $timeformat(-9,2,"ns", 16);
`ifdef SDFSCAN
    $sdf_annotate("sdf/ADDC_tsmc18_scan.sdf", test.top);
`endif
    clk = 1'b0;
    reset = 1'b0;
    scan_in0 = 1'b0;
    scan_in1 = 1'b0;
    scan_in2 = 1'b0;
    scan_in3 = 1'b0;
    scan_in4 = 1'b0;
    scan_enable = 1'b0;
    test_mode = 1'b0;

    // initialize input to 1
    test_selOneHot = 1;

    // set the very first weight to 2 (channel 0)
    test_dataInBus = 2;
    tempDataIn = 2;
    dataArray[0] = 2;

    // input weight data bus generation
    for(j = 1; j < CHANNELS; j = j+1)
    begin
        // manipulate test data for each channel (increment by 2 in
           this case)
        tempDataIn = tempDataIn + 2;
```

```
            // set weight data bus by shifting and bitwise or
            test_dataInBus = test_dataInBus | (tempDataIn << WIDTH*j);

            // save data to output checker array as well
            dataArray[j] = tempDataIn;


        end

        while(!testDoneFlag)
        begin
            @(posedge clk)
            begin
                // assign i to sel input as test vector
                test_selOneHot = i;

                // i=1 will be shifted by 1 from bit 0 to bit (WIDTH-1)
                i = i << 1;

                // reset if we overflow
                if (test_selOneHot == 0)
                    i = 1;
            end

        end

    end


    // check output in parallel on negative edge
    always @(negedge clk)
    begin
        for(k = 0; k < CHANNELS; k = k+1)
        begin
            // make sure input is valid (one hot)
            if (test_selOneHot == 1 << k)
                // check if DUT output matches expected output
                if (test_dataOut !== dataArray[k])
                begin
                    // display useful information if the outputs don't
                       match
                    $display("Wrong output at %0t", $time);
```

```
                $display("Expected %H, Actual %H", dataArray[k],
                    test_dataOut);

                // stop if we see error
                //$finish;
            end

    end

    // count waves
    waveCounter = waveCounter + 1;

    // stop if we looped through all channel values (*2 to see some
        extra
    // length)
    if (waveCounter >= CHANNELS*2)
        $finish;
end


// clock generation
always #(CLOCK_PERIOD/2)
    clk = ~clk;

endmodule
```

Listing I.3: Next Grant Precalculator Module

```
// NGPRC module
// Next Grant PReCalculate
//
//
// This module precalculate the mask for the Grant Process
// The mask is shfted left to dictate round robin manner
// Input : request , grant
// Output : nextGrant mask
//
module NGPRC
        #(
            parameter CHANNELS = 8 // total number of requestors
        )
        (
            reset,
            clk,
            scan_in0,
            scan_in1,
            scan_in2,
            scan_in3,
            scan_in4,
            scan_enable,
            test_mode,
            scan_out0,
            scan_out1,
            scan_out2,
            scan_out3,
            scan_out4,

            // inputs
            request,                    // request input
            grant,                      // grant input

            // outputs
        nextGrant                   // next grant output

       );

input
    reset,                          // system reset
    clk;                            // system clock
```

```
input
    scan_in0,                       // test scan mode data input
    scan_in1,                       // test scan mode data input
    scan_in2,                       // test scan mode data input
    scan_in3,                       // test scan mode data input
    scan_in4,                       // test scan mode data input
    scan_enable,                    // test scan mode enable
    test_mode;                      // test mode

output
    scan_out0,                      // test scan mode data output
    scan_out1,                      // test scan mode data output
    scan_out2,                      // test scan mode data output
    scan_out3,                      // test scan mode data output
    scan_out4;                      // test scan mode data output

input [(CHANNELS-1) : 0] request;
input [(CHANNELS-1) : 0] grant;

output reg [(CHANNELS-1) : 0] nextGrant;


reg [(CHANNELS-1) : 0] priorityMask;

//--- Internal Constants ---//
localparam SIZE = 2;

// STATES
reg [(SIZE-1) : 0] state;

localparam RESET = 'b01;        // 3'b001
localparam NEXT_GRANT = 'b10;   // 3'b010


//--- Code Starts --//

// always block for state transition
always@(posedge clk, posedge reset)
begin : preCalStateTransition

    if (reset == 1'b1)
    begin
```

```
        state = RESET;
    end
    else
        // state transition
        case(state)
            // check if we are out of reset
            RESET :
            begin
                // transition right away once NOT in reset
                 state = NEXT_GRANT;

            end

            NEXT_GRANT :
            begin
                //go back to reset if there is reset
                state = state;

            end

            default :
            begin
                // stay in the same state
                state = RESET;
            end

    endcase


end


// output logic
always @(posedge clk, posedge reset)
begin : preCalOutputLogic
    if(reset == 1'b1)
    begin

        nextGrant = 0;
        priorityMask = 0;
    end
    else
```

```
case(state)
    // reset signals in reset state
    RESET :
    begin

        nextGrant = 0;
        priorityMask = ~0;

    end

    // set next grant and priorityMask
    // Handle wrap around case
    NEXT_GRANT :
    begin

        // calculate priorityMask
        // Rotate left, invert and add 1
        priorityMask = ~{grant[CHANNELS-2 : 0], grant[CHANNELS-1]}
            + 1;

        // if grant somehow becomes zero, set priorityMask to all
           1
        if (priorityMask == 0)
            priorityMask = ~0;
        else
            priorityMask = priorityMask;

        // calculate nextGrant
        nextGrant = request & priorityMask;
        //nextGrant = priorityMask;

        // if we see a request but nextGrant is zero
        // it means we wrap around
        if ((nextGrant == 0) && (request != 0) )
            nextGrant  = request;

    end

    // if statemachine never goes out of wack
    // we should NOT reach to this case
    default :
    begin
        // keep all the signals the same
```

```
            priorityMask = priorityMask;
            nextGrant = nextGrant;

        end


    endcase
end


endmodule //NGPRC
```

Listing I.4: Next Grant Precalculator Test Module

```
module test;
localparam WIDTH = 32;
localparam CHANNELS = 8;

// clock period
localparam CLOCK_PERIOD = 100; //20ns (50Mhz)


wire  scan_out0, scan_out1, scan_out2, scan_out3, scan_out4;
reg  clk, reset;
reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4, scan_enable,
   test_mode;

// inputs
reg [(CHANNELS-1) : 0] test_request;
reg [(CHANNELS-1) : 0] test_grant;

// flow  control
reg [(CHANNELS-1) : 0] expectedNextGrant;
reg sticky;

//output
wire [(CHANNELS-1) : 0] test_nextGrant;
wire [4 : 0] test_debugPreCal;



NGPRC top(
        .reset(reset),
        .clk(clk),
        .scan_in0(scan_in0),
        .scan_in1(scan_in1),
        .scan_in2(scan_in2),
        .scan_in3(scan_in3),
        .scan_in4(scan_in4),
        .scan_enable(scan_enable),
        .test_mode(test_mode),
        .scan_out0(scan_out0),
        .scan_out1(scan_out1),
```

```
        .scan_out2(scan_out2),
        .scan_out3(scan_out3),
        .scan_out4(scan_out4),
            // input
        .request(test_request),
        .grant(test_grant),

            // output
        .nextGrant(test_nextGrant)
        //     .debugPreCal(test_debugPreCal)
    );


initial
begin
    $timeformat(-9,2,"ns", 16);
`ifdef SDFSCAN
    $sdf_annotate("sdf/ADDC_tsmc18_scan.sdf", test.top);
`endif
    clk = 1'b0;
    reset = 1'b1;
    scan_in0 = 1'b0;
    scan_in1 = 1'b0;
    scan_in2 = 1'b0;
    scan_in3 = 1'b0;
    scan_in4 = 1'b0;
    scan_enable = 1'b0;
    test_mode = 1'b0;

    sticky = 0;
    test_request = 0;
    test_grant = 0;
    // release reset
    @(posedge clk);
    reset = 1'b0;

    // test case 1
    // request = 0000_000
    // grant = don't care
    // nextGrant =  0000_0000
    @(posedge clk);
    test_request = 0;
    test_grant = 0;
```

```
expectedNextGrant = 0;
$display("-----Test case 1-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);



// test case 2
// request = 1111_1111
// grant = 0001
// nextGrant = 1111_1110
@(posedge clk);
test_request = 8'hFF;
test_grant = 1;
expectedNextGrant = 8'b1111_1110;
$display("-----Test case 2-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);


// test case 3
// request = 1111_1111
// grant = 0010
// nextGrant = 1111_1100
@(posedge clk);
```

```
test_request = 8'hFF;
test_grant = 8'b0000_0010;
expectedNextGrant = 8'b1111_1100;
$display("-----Test case 3-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);


// test case 4
// request = 1111_1111
// grant = 0000_0100
// nextGrant = 1111_1000
@(posedge clk);
test_request = 8'hFF;
test_grant = 8'b0000_0100;
expectedNextGrant = 8'b1111_1000;
$display("-----Test case 4-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);

// test case 5
// request = 1111_1111
// grant  = 1000_0000
// nextGrant = 1111_1111
@(posedge clk);
```

```
test_request = 8'hFF;
test_grant = 8'b1000_0000;
expectedNextGrant = 8'b1111_1111;
$display("-----Test case 5-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);


// test case 6
// request = 0000_0000
// grant = don't care
// nextGrant = 0000_0000
@(posedge clk);
test_request = 8'h00;
test_grant = 8'b1;
expectedNextGrant = 8'b0;
$display("-----Test case 6----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);



// test case  7
// request = 0000_0010
// grant = 0000_0010
```

```
// nextGrant = 0000_0010
// nextGrant = 1111_1100 ??  maybe?
@(posedge clk)
test_request = 8'b0000_0010;
test_grant = 8'b0000_0010;
expectedNextGrant = 8'b0000_0010;
$display("-----Test case 7-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);

// test case 8
// request = 0000_0010
// grant = 0
// nextGrant = 0000_0010
@(posedge clk)
test_grant = 8'b0000_0010;
test_grant = 8'b0;
expectedNextGrant = 8'b0000_0010;
$display("-----Test case 8-----");
$display("Request = %b", test_request);
$display("grant = %b", test_grant);
@(negedge clk);
if(test_nextGrant != expectedNextGrant)
begin
    sticky = 1;
    $display("Expected next grant = %b, Actual = %b",
       expectedNextGrant, test_nextGrant);
end
else
    $display("next grant = %b", test_nextGrant);


@(posedge clk)
reset = 1'b1;
```

```
    @(posedge clk)
    reset = 1'b0;

    @(posedge clk);
    @(posedge clk);
    @(posedge clk);


    if (sticky == 1)
        $display("Test failed");
    else
        $display("Test passed");

    $finish;

end

// check output in parallel on negative edge
//always @(negedge clk)
//begin
//    for(k = 0; k < CHANNELS; k = k+1)


//    begin
      // make sure input is valid (one hot)
//      if (test_selOneHot == 1 << k)
        // check if DUT output matches expected output
//        if (test_dataOut !== dataArray[k])
//        begin
            // display useful information if the outputs don't
                match
 //           $display("Wrong output at %0t", $time);
//            $display("Expected %H, Actual %H", dataArray[k],
   test_dataOut);

            // stop if we see error
            //$finish;
//        end

//    end

    // count waves
```

```
//     waveCounter = waveCounter + 1;

    // stop if we looped through all channel values (*2 to see some
       extra
    // length)
//     if (waveCounter >= CHANNELS*2)
//         $finish;
//end


// clock generation
always #(CLOCK_PERIOD/2)
    clk = ~clk;

endmodule
```

Listing I.5: Grant Module

```
// GRANT module
//
//
module GRANT
        #(
            parameter CHANNELS = 8, // total number of requestors
            parameter WIDTH = 32,    // the width of each  requestor's
                weight
            parameter WEIGHTLIMIT = 16
         )
         (
            reset,
            clk,
            scan_in0,
            scan_in1,
            scan_in2,
            scan_in3,
            scan_in4,
            scan_enable,
            test_mode,
            scan_out0,
            scan_out1,
            scan_out2,
            scan_out3,
            scan_out4,

            // input
            request,          // request input
            nextGrant,        // nextGrant from NGPRC
            weight,           // weight of current grant

            // output
            grant           // grant output
        );

input
    reset,                          // system reset
    clk;                            // system clock

input
    scan_in0,                       // test scan mode data input
```

```
    scan_in1,                   // test scan mode data input
    scan_in2,                   // test scan mode data input
    scan_in3,                   // test scan mode data input
    scan_in4,                   // test scan mode data input
    scan_enable,                // test scan mode enable
    test_mode;                  // test mode

output
    scan_out0,                  // test scan mode data output
    scan_out1,                  // test scan mode data output
    scan_out2,                  // test scan mode data output
    scan_out3,                  // test scan mode data output
    scan_out4;                  // test scan mode data output

// input
input [(CHANNELS-1) : 0] request;
input [(CHANNELS-1) : 0] nextGrant;
input [(WIDTH-1) : 0] weight;

// output
output reg [(CHANNELS-1) : 0] grant;


// internal registers
reg [(WIDTH-1) : 0] s_counter;
reg [(CHANNELS-1) : 0] s_request;
reg [(WIDTH-1) : 0] s_weight;
//reg update;
//--- Internal Constants ---//
localparam SIZE = 4;

// STATES
reg [(SIZE-1) : 0] state;
localparam RESET = 'b0001;              // 'b00001
localparam GRANT_PROCESS = 'b0010;      // 'b00100
localparam COUNT = 'b0100;              // 'b01000
localparam GETWEIGHT = 'b1000;          // 'b10000

//--- Code Starts --//

// registeri/delay request
always@(posedge clk, posedge reset)
begin : requestDelay
```

```
    if(reset == 1'b1)
        s_request = 0;
    else
        s_request = request;
end

// always block for state transition
always@(posedge clk, posedge reset)
begin : grantStateTransition
    // reset condition
    if(reset == 1'b1)
    begin
        state = 0;
    end

    // out of reset
    else
    begin
        // state transisiton
        case(state)
            // grant process this output grant
            GRANT_PROCESS :
            begin
                // if there is request
                // go to COUNT state to count
                if(grant != 0)
                    state = GETWEIGHT;

                // just stay here and process next
                else
                    state = state;

            end

            GETWEIGHT :
            begin
                state = COUNT;
            end

            // count clock cycle according to weight
            COUNT :
            begin
                // if counter is up
```

```
                    // move to grant next
                    if(s_counter >= s_weight)
                        state = GRANT_PROCESS;

                    // fairness limit set by user
                    // default is 16
                    else if(s_counter >= WEIGHTLIMIT)
                        state = GRANT_PROCESS;

                    // else
                    // keep counting
                    else
                        state = state;

                end

                // if statemachine never goes out of wack
                default :
                begin
                    state = GRANT_PROCESS;
                end
            endcase

    end

end

// output logic
always@(posedge clk, posedge reset)
begin:grantStateMachineOutputLogic
    if(reset == 1'b1)
    begin
        grant = 0;
        s_counter = 0;
        s_weight = 0;
    end
    else
    case(state)
        RESET :
        begin
            // reset everything in reset state

            grant = 0;
```

```
            s_counter = 0;
            //s_mask = ~0;
        end

        GRANT_PROCESS :
        begin
            // update mask
            //s_mask = nextGrant & (~nextGrant + 1);

            // granting logic
            grant = request & nextGrant & (~nextGrant + 1);

            // it takes 3 cycle to look back here
            // so set the  counter for when weight >= 2
            s_counter = 2;
        end

        GETWEIGHT:
        begin
            s_weight = weight;
        end

        COUNT :
        begin
            // count up until weight is reached account for clock
               cycle
            s_counter = s_counter + 1;

            // no change to grant
            grant = grant;
        end

        // if statemachine never goes out of wack
        default :
        begin
            grant = grant;
            s_counter = s_counter;

        end

    endcase
end
```

```
endmodule // GRANT
```

Listing I.6: Round Robin Arbiter Module

```
// RRBTOP module
// Top level of Round Robin Arbiter
// This module connects MUX, Next Grant Precalculator and Grant
    statemachine
// Input : request, weight bus
// Output : grant
`include "include/RRB_verification.h"


module RRBTOP
        #(
            parameter CHANNELS = `CHANNELS,
            parameter WIDTH = `WIDTH,
            parameter WEIGHTLIMIT = `WEIGHTLIMIT
        )
        (
            reset,
            clk,

            // input to RRB
            request,
            weight,      // weight bus each having bit size of WIDTH for
                each channel
            // output from RRB
        grant,

            scan_in0,
            scan_in1,
            scan_in2,
            scan_in3,
            scan_in4,
            scan_enable,
            test_mode,
            scan_out0,
            scan_out1,
            scan_out2,
            scan_out3,
            scan_out4
        );

input
```

```
    reset,                          // system reset
    clk;                            // system clock

input
    scan_in0,                       // test scan mode data input
    scan_in1,                       // test scan mode data input
    scan_in2,                       // test scan mode data input
    scan_in3,                       // test scan mode data input
    scan_in4,                       // test scan mode data input
    scan_enable,                    // test scan mode enable
    test_mode;                      // test mode

output
    scan_out0,                      // test scan mode data output
    scan_out1,                      // test scan mode data output
    scan_out2,                      // test scan mode data output
    scan_out3,                      // test scan mode data output
    scan_out4;                      // test scan mode data output

input [(CHANNELS-1) : 0] request;
input [(CHANNELS*WIDTH)-1 : 0] weight;

output wire [(CHANNELS-1) : 0] grant;

wire [(CHANNELS-1) : 0] s_selOneHot;
wire [(WIDTH-1) : 0] s_weight;
wire [(CHANNELS-1) : 0] s_nextGrant;




// COMBINATIONAL SECTION //
assign grant = s_selOneHot;

// MUX
MUX #(
    .WIDTH(WIDTH),
    .CHANNELS(CHANNELS)
    )
    MUX(
    .reset(reset),
    .clk(clk),
```

```
    .scan_in0(scan_in0),
    .scan_in1(scan_in1),
    .scan_in2(scan_in2),
    .scan_in3(scan_in3),
    .scan_in4(scan_in4),
    .scan_enable(scan_enable),
    .test_mode(test_mode),
    .scan_out0(scan_out0),
    .scan_out1(scan_out1),
    .scan_out2(scan_out2),
    .scan_out3(scan_out3),
    .scan_out4(scan_out4),

    // input
    .selOneHot(s_selOneHot),
    .dataInBus(weight),

    // output
    .dataOut(s_weight)

    );

NGPRC #(
    .CHANNELS(CHANNELS)
    )
    NGPRC(
    .reset(reset),
    .clk(clk),
    .scan_in0(scan_in0),
    .scan_in1(scan_in1),
    .scan_in2(scan_in2),
    .scan_in3(scan_in3),
    .scan_in4(scan_in4),
    .scan_enable(scan_enable),
    .test_mode(test_mode),
    .scan_out0(scan_out0),
    .scan_out1(scan_out1),
    .scan_out2(scan_out2),
    .scan_out3(scan_out3),
    .scan_out4(scan_out4),

    // input
    .request(request),
```

```
        .grant(s_selOneHot),

        // output
        .nextGrant(s_nextGrant)

        );

GRANT #(
        .CHANNELS(CHANNELS),
        .WIDTH(WIDTH),
        .WEIGHTLIMIT(WEIGHTLIMIT)
        )
       GRANT(
        .reset(reset),
        .clk(clk),
        .scan_in0(scan_in0),
        .scan_in1(scan_in1),
        .scan_in2(scan_in2),
        .scan_in3(scan_in3),
        .scan_in4(scan_in4),
        .scan_enable(scan_enable),
        .test_mode(test_mode),
        .scan_out0(scan_out0),
        .scan_out1(scan_out1),
        .scan_out2(scan_out2),
        .scan_out3(scan_out3),
        .scan_out4(scan_out4),

        // input
        .request(request),
        .nextGrant(s_nextGrant),
        .weight(s_weight),

        // output
        .grant(s_selOneHot)



        );
endmodule // RRB
```

Listing I.7: Round Robin Arbiter Test Module

```
module test;
localparam WIDTH = 32;
localparam CHANNELS = 4;
localparam WEIGHTLIMIT = 100;

// clock period
localparam CLOCK_PERIOD = 20; // 20ns(500MHZ)

wire  scan_out0, scan_out1, scan_out2, scan_out3, scan_out4;
reg  clk, reset;
reg  scan_in0, scan_in1, scan_in2, scan_in3, scan_in4, scan_enable,
   test_mode;


//inputs
reg [(CHANNELS*WIDTH-1) : 0] test_weight;
reg [(CHANNELS-1) : 0] test_request;
//reg test_ack;

// output
wire [(CHANNELS-1) : 0] test_grant;

//
reg [(WIDTH-1) : 0] tempData;

// flow control flags
integer j = 0;

RRBTOP  #(
        .CHANNELS(CHANNELS),
        .WIDTH(WIDTH),
        .WEIGHTLIMIT(WEIGHTLIMIT)
        )
        top(
        .reset(reset),
        .clk(clk),
        .scan_in0(scan_in0),
        .scan_in1(scan_in1),
        .scan_in2(scan_in2),
        .scan_in3(scan_in3),
```

```
        .scan_in4(scan_in4),
        .scan_enable(scan_enable),
        .test_mode(test_mode),
        .scan_out0(scan_out0),
        .scan_out1(scan_out1),
        .scan_out2(scan_out2),
        .scan_out3(scan_out3),
        .scan_out4(scan_out4),

        // input
        //
        .request(test_request),
        .weight(test_weight),
//          .ack(test_ack),


        // output
        .grant(test_grant)

    );


initial
begin
    $timeformat(-9,2,"ns", 16);
`ifdef SDFSCAN
    $sdf_annotate("sdf/ADDC_tsmc18_scan.sdf", test.top);
`endif
    clk = 1'b0;
    reset = 1'b1;
    scan_in0 = 1'b0;
    scan_in1 = 1'b0;
    scan_in2 = 1'b0;
    scan_in3 = 1'b0;
    scan_in4 = 1'b0;
    scan_enable = 1'b0;
    test_mode = 1'b0;

//    test_ack = 1'b0;
    test_request = 0;

    // set the very first weight to 2 (channel 0)
    tempData = 3;
```

```
    test_weight = 3;

    // input weight data bus generation
    for(j = 1; j < CHANNELS; j = j+1)
    begin
        // manipulate test data for each channel  (increment by 2 in
           this
        // case)
        tempData = tempData + 2;

        // set weight data bus by shifting and  bitwise or
        test_weight = test_weight | (tempData << WIDTH*j);

    end

    // pull reset high
    @(posedge clk);
    @(posedge clk);
    reset = 1'b0;


    @(posedge clk);
    test_request = 'b1001;

    #100

    @(posedge clk);
  //  test_ack = 1'b1;
    test_request = 'b0010;

    @(posedge clk);
//     test_ack = 1'b0;

    #160
    @(posedge clk);
   // test_ack = 1'b1;
    test_request = 'b0001;

    @(posedge clk);
//     test_ack = 1'b0;

    #500
    $finish;
```

```
end


// clock generation
always #(CLOCK_PERIOD/2)
    clk = ~clk;

endmodule
```

Appendix II

Test Bench Source Code

Listing II.1: Requestor Module

```
// requestor.sv
// requestor module
// this module generates
// random request (0 or 1)
// weight (0 to 2^32-1)

class requestor;
    rand bit request; // request
    rand bit [31:0] weight; // weight

    int seed;
    int weightLow;
    int weightHigh;

    // constraint weight between the limits
    constraint weight_range {
        weight inside {[weightLow : weightHigh]};
    }

    // constructor
    function  new(int seed = 1, int weightLow = 2, int weightHigh =
       100);
        this.request = 0;
        this.weight = weightLow;
```

```
        this.seed = seed;
        this.weightLow = weightLow;
        this.weightHigh = weightHigh;

        // initialize random seed
        this.srandom(seed);

    endfunction : new


endclass
```

Listing II.2: Generator Module

```
// generator.sv
// generator class
// This class generates random stimulus
// in this case, it instantiates multiple requestors

`include "include/RRB_verification.h"

class generator;
    requestor req;

    // constructor method
    function new(requestor req);
        this.req = req;
    endfunction : new

    // generate random requests
    extern function void generate_requestor;

    // get request
    extern function bit get_request;

    // get weight
    extern function bit[`WIDTH-1 : 0] get_weight;

endclass : generator

//----------External Methods----------//

// generate function
// no arg input
// randomize the requestor
function void generator::generate_requestor;
    begin : randomize_requestor

        //generate random request/weight in requestor
        assert(req.randomize());

        if(`DEBUG_GENERATOR)
            $display("generated requestor : %p\n", req);

    end : randomize_requestor
```

```
endfunction : generate_requestor

// get method for requestor's request
function bit generator::get_request;
    begin
        return req.request;
    end

endfunction : get_request

// get method for requestor's weight
function bit[`WIDTH-1 : 0] generator::get_weight;
    begin
        return req.weight;
    end
endfunction : get_weight
```

Listing II.3: Agent Module

```
// agent.sv
// agent class
// agent instantiates generator module
// using that generator to generate random requests

`include "include/RRB_verification.h"

class agent;
    generator gen;
    bit [`CHANNELS-1 : 0] requests;
    bit [(`CHANNELS*`WIDTH)-1 : 0] weights;

    bit [`CHANNELS-1 : 0] golden_grants;
    int golden_grant_weights [`CHANNELS] = '{default:0};
    int temp_weights [`CHANNELS] = '{default:0};

    // constructor method
    function new(generator gen);
        this.gen = gen;
        this.requests = 0;
        this.weights = 0;

        this.golden_grants = 0;
        this.golden_grant_weights = '{default:0};
        this.temp_weights = '{default:0};
    endfunction : new

    // generate new requests
    extern function void generate_requests;

endclass : agent

//----------External Methods----------//

// generate_requests function
// no arg input
// generate n number of requestors
function void agent::generate_requests;
    begin : generateNewRequests

        // only get golden vector if requests are not zero
```

```
// by getting golden vector at the start of this routine
// we are essentially delaying the update by one test vector
   set
// this allows golden vectors to be in sync with checker/
   monitor/DUT
if(requests != 0)
begin
    // golden test data to check  DUT
    golden_grants = requests;
    golden_grant_weights = temp_weights;

    $display("request at update %b", requests);
    $display("weight at update %h", weights);
    $display("golden update at %g", $time);
end

// reset weights
weights = 0;

// loop through to generate n random requests
for(int i = 0; i < `CHANNELS; i++)
begin
    gen.generate_requestor();

    // random request data
    requests[i] = gen.get_request();
    weights = weights | gen.get_weight() << (i * `WIDTH);
    temp_weights[i] = gen.get_weight();


    // display if debug agent flag is set
    // debug only
    if(`DEBUG_AGENT)
    begin
    //     $display("i = %d", i);
    //     $display("request = %d", gen.get_request());
    //     $display("weight = %d", gen.get_weight());
    end

end

// debug
if(`DEBUG_AGENT)
```

```
        begin
            $display("requests = %b", requests);
            $display("weights = %h", weights);

            $display("golden grants = %b", golden_grants);

            for(int i = 0; i < `CHANNELS; i++)
            begin
                $display("golden grant weights %d = %h", i,
                    golden_grant_weights[i]);
            end
        end

    end : generateNewRequests
endfunction : generate_requests
```

Listing II.4: Driver Module

```
// driver.sv
// driver class
`include "include/RRB_verification.h"

class driver;

    // handle for interface
    virtual intf_rrb intf;

    // handle for agent
    agent agt;

    // event handle
    event e_start;
    event e_drv_done;

    // constructor method
    function new(virtual intf_rrb intf, agent agt, event e_drv_done);
        this.intf = intf;
        this.agt = agt;
        this.e_drv_done = e_drv_done;

    endfunction : new

    // reset method
    extern task reset();

    // drive new data
    extern task drive_new_data();

    // event logic
    extern task event_logic();
endclass : driver

//----------External Methods----------//
// drive new data
task driver::drive_new_data();
    // we need to know how long to wait to know when drive new data
    // so wait for weight cycles before sending new data
    bit [(`CHANNELS*`WIDTH)-1 : 0] waitCycles, temp;
    waitCycles = 0;
```

```
    temp = 0;

    // get the sum of all the weights
    for(int i = 0; i < `CHANNELS; i++)
    begin
        temp = agt.weights >> (i*`WIDTH);
        waitCycles = waitCycles + temp[`WIDTH-1 : 0] * (agt.requests[i
            ]);
    end

    // debug only
    if(`DEBUG_DRIVER)
    begin
        $display("agt %p\n", agt);
        $display("waitCycles %d\n", waitCycles);
    end

    // send request to DUT synchronously
    // only need to wait if waitCycle is not zero
    if(waitCycles > 0)
    begin
        // wait for waitCyclesi for DUT to perform arbiteration
        for(int i = 0; i < (waitCycles); i++)
        begin
            @(posedge intf.DRIVER.clk);
            intf.DRIVER.request = agt.requests;
            intf.DRIVER.weight = agt.weights;
        end

        // after wait cycle is done, tell event logic to start counter
        -> e_start;
        //$display("start at %g", $time);
    end

endtask : drive_new_data

// Event logic
// to wait 2 cycles after driver is done driving
// to sync up with end of DUT output
task driver::event_logic();
    forever
    begin
        //@(posedge intf.DRIVER.clk);
```

```
        @(e_start);
        //$display("caught at %g", $time);

        //repeat(2)
            //@(posedge intf.DRIVER.clk);

        -> e_drv_done;
        //$display("driver done issued");
    end
endtask : event_logic

// Reset Method
task driver::reset();
    // initialize everything
    intf.DRIVER.reset = 1'b0;
    intf.DRIVER.request = 0;
    intf.DRIVER.weight = 0;

    // reset on negative edge
    @(negedge intf.DRIVER.clk);
    intf.DRIVER.reset = 1'b1;

    // wait for a few cycles
    @(negedge intf.DRIVER.clk);
    @(negedge intf.DRIVER.clk);

    intf.DRIVER.reset = 1'b0;
    // wait for a few cycles
    @(negedge intf.DRIVER.clk);
    @(negedge intf.DRIVER.clk);


endtask : reset
```

Listing II.5: Monitor Module

```
// monitor.sv
// monitor class
// Monitor captures DU output

`include "include/RRB_verification.h"

class  monitor;

    // handle for interface
    virtual intf_rrb intf;

    // DUT output vectors
    bit [`CHANNELS-1 : 0] dut_grants;
    int dut_weight_array [`CHANNELS] = '{default:0};

    // data to be sent to checker
    bit [`CHANNELS-1 : 0] mon_grants;
    int mon_weight_array [`CHANNELS] = '{default:0};

    // event handles
    event e_drv_done;
    event e_mon_done;

    // bit to indicate data needs to be cleared
    bit clearData;

    // constructor method
    function new(virtual intf_rrb intf, event e_drv_done, event
       e_mon_done);
        this.intf = intf;

        this.dut_grants = 0;
        this.dut_weight_array = '{default:0};
        this.mon_grants = 0;
        this.dut_weight_array = '{default:0};

        this.e_drv_done = e_drv_done;
        this.e_mon_done = e_mon_done;

        this.clearData = 0;
    endfunction : new
```

```
    // monitor the output
    extern task run;

    // event logic
    extern task event_logic;
endclass : monitor

//----------External Methods----------//
// run method
task monitor::run;

    // wait time for driver

    forever
    begin

        // capture data on positive edge of clock
        @(posedge intf.MONITOR.clk);
        if(intf.MONITOR.grant != 0)
        begin

            // if we need to clear counters, do it first before
               couting again
            if(clearData)
            begin
                dut_grants = 0;
                dut_weight_array = '{default:0};
                clearData = 0;
            end

            dut_grants[$clog2(intf.MONITOR.grant)] = 1;
            dut_weight_array[$clog2(intf.MONITOR.grant)] =
               dut_weight_array[$clog2(intf.MONITOR.grant)] + 1;

        end


    end


endtask : run
```

```
// event logic
// when driver is done driving, it means one test vector is done
// we need to register data for monitor to be read
// at the same time set clear bit to clear out counters
task monitor::event_logic;
    forever
    begin
        //@(posedge intf.MONITOR.clk);
        //wait(e_drv_done.triggered);
        @(e_drv_done);
        $display("eventtriggered at %g", $time);

        repeat(2)
            @(posedge intf.MONITOR.clk);

        // driver/DUT finished one set of test vectors
        // register/copy output for monitor
        mon_grants = dut_grants;
        mon_weight_array = dut_weight_array;

        // indicate monitor is ready for checker
        -> e_mon_done;

        // if we finished one set of test vectors
        // we need to clear counters
        clearData = 1;

        if(`DEBUG_MONITOR)
        begin
            $display("---");
            $display("dut grant = %b at %g", dut_grants, $time);
            $display("mon grant = %b at %g", mon_grants, $time);

            //$display("weight = %h", dut_weight_array[$clog2(intf.
               MONITOR.grant)]);
            for(int i = 0; i < `CHANNELS; i++)
            begin
                $display("dut weight %d = %h", i, dut_weight_array[i])
                    ;
                $display("mon weight %d = %h", i, mon_weight_array[i])
                    ;

            end
```

```
        end
    end
endtask : event_logic
```

Listing II.6: Checker Module

```
//check.sv
// check class
// get output of DUT and agent data
// and compare and recored in scoreboard

`include "include/RRB_verification.h"

class check;
    // class handles
    scoreboard sb;
    agent agt;
    monitor mon;

    // event handles
    event e_mon_done;
    event e_drv_done;

    // golden test vectors
    bit [`CHANNELS-1 : 0] chk_golden_grants;
    int chk_golden_grant_weights [`CHANNELS] = '{default:0};


    // constructor method
    function new(scoreboard sb, agent agt, monitor mon, event
       e_mon_done, event e_drv_done);
        this.sb = sb;
        this.agt = agt;
        this.mon = mon;

        this.e_mon_done = e_mon_done;
        this.e_drv_done = e_drv_done;

        this.chk_golden_grants = 0;
        this.chk_golden_grant_weights = '{default:0};
    endfunction : new

    // task to check the output of DUT
    extern task check_output;

    // test to update the golden data
    // it is always running but in sync with end of driver done event
```

```
    extern task update_golden_data;

endclass : check

//----------External Methods----------//
// check the output of DUT and Agent golden data
task check::check_output();
    forever
    begin
        @(e_mon_done);

        if(`DEBUG_CHECKER)
        begin
            $display("local golden grant = %b", chk_golden_grants);
            $display("mon DUT grant = %b at %g", mon.mon_grants, $time
                );

            for(int i = 0; i < `CHANNELS; i++)
            begin
                $display("mon DUT weight %d = %h", i, mon.
                    mon_weight_array[i]);
                $display("local golden weight %d = %h", i,
                    chk_golden_grant_weights[i]);
            end
        end

        // loop through all the channels and check the output
        for (int i = 0; i < `CHANNELS; i++)
        begin
            // if the request is 1, check the weight
            if(chk_golden_grants[i] == 1)
            begin
                //$display("golden index position = %h", i);
                //$display("dut bit output = %h", mon.mon_grants[i]);

                // check DUT and golden and record
                if((mon.mon_grants[i] == 1) & chk_golden_grant_weights
                    [i] == mon.mon_weight_array[i])
                begin
                    sb.record(i);
                end
                else
                begin
```

```
                    // if the test fails, display some helpful
                       information
                    $display("failure at %g, index %h, expected weight
                       = %h, actual weight = %h", $time, i,
                       chk_golden_grant_weights[i], mon.
                       mon_weight_array[i]);
                    $finish;

                end


            end
        end


        sb.display();
        //$display("cov %e", $get_coverage());

    end
endtask : check_output

// update golden test vectors when driver is done driving
// it is always runnig - forever
task check::update_golden_data();
    forever
    begin
        @(e_drv_done)
        chk_golden_grants = agt.golden_grants;
        chk_golden_grant_weights = agt.golden_grant_weights;

       //$display("local golden grant = %b", chk_golden_grants);

        for(int i = 0; i < `CHANNELS; i++)
        begin
            //$display("local golden weight %d = %h", i,
               chk_golden_grant_weights[i]);
        end
    end
endtask : update_golden_data
```

Listing II.7: Scoreboard Module

```
// scoreboard.sv
// scoreboard class
// records the number of hits

class scoreboard;
    int score_array[`CHANNELS];

    // default construction
    function new();
        this.score_array = '{default:0};
    endfunction : new

    // record score
    extern function void record(int reqIndex);

    // display score
    extern function void display;
endclass : scoreboard

//----------External Methods----------//
// record function
// update the requestor channel count
function void scoreboard::record(int reqIndex);
    // increment
    score_array[reqIndex] = score_array[reqIndex] + 1;

endfunction : record

// display current score
function void scoreboard::display();
    for(int i = 0; i < `CHANNELS; i++)
    begin
        $display("CHANNEL %d score = %d", i, score_array[i]);
    end

endfunction : display
```

Listing II.8: Assertion Module

```
// assertion.sv
// This module defines the assertion properties

module assertion(intf_rcc intf);
    // reset sequence
    sequence reset_seq;
        (intf.reset == 1) ##1 (intf.grant == 0);
    endsequence

    // reset condition
    property reset_property;
        @(posedge intf.clk)
            (intf.reset == 1) |-> reset_seq;
    endproperty
endmodule
```

Listing II.9: Interface Module

```
// interface.sv
// interface  module
// this class declares and defines interface between  various blocks

interface intf_rrb(input bit clk);

    bit reset;                          // system reset
    bit test_mode = 0;                  // DFT test_mode

    bit [`CHANNELS-1 : 0] request;  // request input to RRB
    logic [`CHANNELS*`WIDTH - 1 : 0] weight; // weight input to RRB

    bit [`CHANNELS-1 : 0] grant;    // grant output from RRB

    // modport for RRB module
    modport RRB (input reset, clk, request, weight,
                 output grant);

    // modport for driver class
    modport DRIVER (input clk,
                    output reset, request, weight);

    // modport for monitor class
    modport MONITOR (input clk, grant);


    // reset condition  assert
    property reset_state;
        @(posedge clk) reset |-> grant==0;

    endproperty

    resetAssert : assert property(reset_state);


endinterface
```

Listing II.10: Environment Module

```
// environment.sv
// environment class
// Defines all modules to create a test environment

class environment;

    // single requestor
    requestor req;

    // class instances
    virtual intf_rrb intf;
    generator gen;
    agent agt;
    driver drv;
    monitor mon;
    scoreboard scb;
    check chk;

    // events
    event e_drv_done;
    event e_mon_done;

    // default constructor
    function new(virtual intf_rrb intf);
        this.intf = intf;

        this.req = new(`SEED, `WEIGHTLOW, `WEIGHTHIGH);

        // intialize transactors
        gen = new(req);
        agt = new(gen);
        drv = new(intf, agt, e_drv_done);
        mon = new(intf, e_drv_done, e_mon_done);
        scb = new();
        chk = new(scb, agt, mon, e_mon_done, e_drv_done);
    endfunction : new

endclass : environment
```

Listing II.11: TopLevel Module

```
// topLevel.sv
// top level SV test module for RRB

module topLevel();

    //testModules testModules();
    reg clk = 0;

    // clock generator
    initial
    forever #(`CLOCK_PERIOD/2) clk = ~clk;

    // interface instance
    intf_rrb intf(clk);

    // DUT
    RRBTOP #(
            .CHANNELS(`CHANNELS),
            .WIDTH(`WIDTH),
            .WEIGHTLIMIT(`WEIGHTLIMIT)
            )
            top(
                .reset(intf.RRB.reset),
                .clk(intf.RRB.clk),
                .request(intf.RRB.request),
                .weight(intf.RRB.weight),
                .grant(intf.RRB.grant)
            );

    // test case
    testcase test(intf);

endmodule
```

Listing II.12: TestCase Module

```
// test.sv
// test case for RoundRobin Arbiter

`include "include/RRB_verification.h"

program testcase(intf_rrb intf);
    genvar i;
    // coverage info
    covergroup din_cov@(posedge intf.RRB.clk);
        request_coverage : coverpoint intf.RRB.request;
        weight_coverage : coverpoint ignoreFunction(intf.RRB.weight){
                          bins bin_1 = {1'b1};
                            }

        //option.per_instance=1;
    endgroup

    //Function to ignore weights less than  3
    integer n;
    function bit ignoreFunction(logic [`CHANNELS*`WIDTH - 1 : 0]
       weight);

        //$display("weight = %h", weight);
        for(n = 0; n < `CHANNELS; n++)
        begin
            //$display("N = %d",n);
            //$display("w coverage test : %h", (weight>> (n*`WIDTH) &
               `WEIGHTMASK));
            if( ((weight>>(n*`WIDTH)) & `WEIGHTMASK) <= 2)
            begin
                //return 0;
                //$display("returning false");
            end
        end

        //$display("returning true");
        return 1;
    endfunction

    covergroup dout_cov@(posedge intf.RRB.clk);
        grant_coverage : coverpoint intf.RRB.grant{
```

```
                        bins ch0 = {0};
                        bins ch1 = {1<<1};
                        bins ch2 = {1<<2};
                        bins ch3 = {1<<3};
                        bins ch4 = {1<<4};
                        bins ch5 = {1<<5};
                        bins ch6 = {1<<6};
                        bins ch7 = {1<<7};
                    }

    option.per_instance=1;
endgroup

// coverage handle
din_cov din_covergroup = new();
dout_cov dout_covergroup = new();

// env object (interface)
environment env = new(intf);

initial
begin
    $timeformat(-9,2,"ns",16);
    $set_coverage_db_name("testCov");

    // start converage collection
    din_covergroup.start();
    dout_covergroup.start();

      // reset dut
    env.drv.reset();

    // Test started
    $display("-----Test started-----");

    // generate new test vectors
    env.agt.generate_requests();
    $display("--------------------------------------------------");

    // run monitor, mon/drv event sync logic
    fork
        env.mon.run();
        env.drv.event_logic();
```

```
            env.mon.event_logic();
            env.chk.update_golden_data();
            env.chk.check_output();
        join_none

        fork
            // env.chk.check_output();
        join_none

        @(posedge intf.DRIVER.clk);
        // drive the test vector, wait for sum(request*weight), and
            generate again
        for(int i = 0; i < `NUM_ITERATIONS; i++)
        begin
            $display("iteration = %d", i);
            fork
                // send the test vectors to DUT
                env.drv.drive_new_data();
            join
                // generate new test vectors (requests)
                env.agt.generate_requests();
                $display("------------------------------------------------");
                //$display("%g join", $time);
        end


        @(posedge intf.DRIVER.clk);
        @(posedge intf.DRIVER.clk);
        @(posedge intf.DRIVER.clk);
        @(posedge intf.DRIVER.clk);
        @(posedge intf.DRIVER.clk);
        @(posedge intf.DRIVER.clk);

    end

    final
    begin
        // display scoreboard results
        env.scb.display();


        // display coverage results
        $display("Input request coverage = %e", din_covergroup.
```

```
            request_coverage.get_coverage());
        $display("Input weight coverage = %e", din_covergroup.
            weight_coverage.get_coverage());
        $display("Output grant coverage = %e", dout_covergroup.
            grant_coverage.get_coverage());

    end

endprogram : testcase
```