Network on Chip Scheduling Modified i-SLIP Scheduler for High-Speed Virtual Output Queuing Packets

Ihsen BEN MBAREK[1], Dhia BELHAJALI[1], Mohamed MAZOUZI[1], Salem HASNAOUI[1], Khaled JALASSI[2]
[1]Networking and Distributed Computing: Sys'Com
[2]Electrical Systems Laboratory of Tunis
[1, 2]National Engineering School of Tunis
University of Tunis El Manar, Tunis, Tunisia
ben_mbarek_ihsen@yahoo.fr

ABSTRACT: *This paper presents a design and an implementation of a hardware scheduler in VHDL for High-Speed VoQ using a modified i-SLIP algorithm. The aims for the scheduling algorithm is to match input queues with output queues to achieve the maximum throughput while maintaining stability and eliminating starvation. The N-input by N-output scheduler manages VoQ packets to avoid a HOL blocking. This implementation requires N Grant Arbiters, N Accept Arbiters, three input-to-output Swizzles and a FSM. The Modified i-SLIP arbiter requires perforce a state pointer for the highest priority input (or output). The simulation and synthesis results are shown for N = 8. The design was implemented in VHDL RTL, simulated with Modelsim 6.2 and synthesized using Xilinx-ISE and the Virtex-4 device XC4VFX100-12FFG1517 of 90 nm technology to achieve a maximum frequency of 265.88 MHz, a minimum slices utilization of 771 and a total estimated power consumption about 915 mW.*

Keywords: i-SLIP, Scheduler, VHDL, VoQ, HOL blocking, Virtex-4, Xilinx-ISE

Received: 11 November 2012, Revised 25 December 2012, Accepted 30 December 2012

1. Introduction

Technology advancement permits the implementation of large VLSI systems on a single silicon chip. Thus, Network-on-chip (NoC) is the ideal paradigm for communications in core-based system design. NoC seems as a "*public transportation*". NoC represents a shared resource between components such as processor cores, memories and IP blocks. By analogy to multitasking, NoC and packets are analog to shared processor and tasks, respectively. Therefore, we need a NoC manager called scheduler. In fact, it is the responsible that decides which packets should be transferred within the NoC from input ports to output ports in a given time slot. As we are talking about VLSI systems, there is no doubt that a hardware solution must be the best. That is why we decided to design a hardware scheduler as a real-time solution. On the one hand, there is many scheduling algorithm which can be hardware implemented such as Parallel Iterative Matching (PIM), Round Robin Matching (RRM), Serial Line Internet Protocol (SLIP), iterative Serial Line Internet Protocol (i-SLIP). On the other hand, many ways of packets processing to enhance their management such as Output Buffer, FIFO Input Buffer and Virtual output Queuing (VoQ).

In our case, we settled on Modified i-SLIP algorithm for VoQ packets.

1.1 Scheduling Algorithms

There are various scheduling algorithms where the most employed in manufactory are PIM, RRM, SLIP and i-SLIP.

1.1.1 Parallel Iterative Matching

The iterative scheduling algorithm PIM affects randomly inputs to outputs [1-3]. PIM requires at most N iteration where N represents the devices number [4]. In most of cases, it manages VoQ packets (see *B*)) [4, 5]. Each iteration consists of three steps.

The first step is called request. Each unconnected input that has at least a queued cell for an output sends requests to the required outputs.

In the second step called also grant, a free output that receives at least a request grants to one input randomly.

The last step is called accept. Each input that receives at least a grant, it accepts only one by selecting an output randomly.

On the one hand, PIM can be considered as a starvation free algorithm. Indeed, its iterations number required to converge on a maximal-sized match is optimized [1]. On the other hand, it is too difficult to implement a hardware PIM arbiter. Also, it leads not only to unfairness under heavy loads but also to a throughput about 63 % for single iteration [5].

1.1.2 Round Robin Matching

To resolve both of unfairness and complexity problems, the RRM algorithm took place. RRM consists of three steps: request, grant, accept [1, 6, 7].

• **Step 1: Request:** Each input sends a request to every output for which it has a queued cell.

• **Step 2: Grant:** when an output receives requests, it grants to only one input. Using a fixed round-robin schedule, the granted input is the nearest one to the highest priority element. An output notification is sent to each input to confirm or not the grant. The grant pointer (g_i) points to the highest priority element. The latter pointer must be incremented (*modulo N*) to one position beyond the granted one.

• **Step 3: Accept:** when an input gets a grant, it accepts only one output. Starting from the highest priority element, the accepted one is that who appears next in a fixed round-robin schedule. The highest priority element is pointed by the accept pointer (a_i) of the round-robin schedule. The latter pointer must be incremented (*modulo N*) to one position beyond the accepted one.

Despite the fairness of RRM [8, 9], its synchronized output arbiter leads to a throughput of just 50% [2, 9].

1.1.3 i-SLIP algorithm

i-SLIP is an iterative scheduling algorithm where "*i* " is the number of iteration that can reach at most N iteration where N is the number of Input/output ports. The single iteration 1-SLIP or simply SLIP algorithm [7, 10] is not only the basis of i-SLIP but also an improvement upon the RRM algorithm. Indeed, the only difference between SLIP and RRM is that the grant pointer is updated unless the grant is accepted in the third step. Since SLIP tries to fiend input/output pairs in only one iteration, it is very likely to find unutilized input/output pairs. That is why i-SLIP algorithm uses several iterations to find I/O pairs as many as possible till it converges where there is no more possible matches [10].

To conclude, i-SLIP algorithm assigns the lowest priority to the most recently made connection. So, it is fair and starvation free [6, 7]. As their output arbiters are asynchronous thanks to the conditional update of the grant pointer, under heavy load, all queues with a common output have the same throughput [5].

1.1.4 Modified i-SLIP algorithm

Modified i-SLIP algorithm consists also of three steps [7, 11].

• **Step 1: Request:** It is the same as the RRM first step where each unconnected input sends a request to every output for which it has a queued cell.

• **Step 2: Grant:** when an output receives requests, it grants to only one input. Using a fixed round-robin schedule, the granted input is the nearest one to the highest priority element. An output notification is sent to each input to confirm or not the grant. The grant pointer (g_i) points to the highest priority element. The latter pointer must be incremented (*modulo N*) to one position beyond the granted one if and only if the grant is accepted in Step 3 of the first iteration.

• **Step 3: Accept:** when an input gets a grant, it accepts only one output. Starting from the highest priority element, the accepted one is that who appears next in a fixed round-robin schedule. The highest priority element is pointed by the accept pointer (a_i) of the round-robin schedule. The latter pointer must be incremented (*modulo N*) to one position beyond the accepted one only if this input was matched in the first iteration.

The question is "*what is the significance of this algorithm*?" Indeed, this algorithm not only shuns starvation and HOL blocking problems, but also reduces burstness and latency [7, 11]. The modified i-SLIP converges in $O(log_2 N)$ where the i-SLIP converge in N iteration [7]. Moreover, throughput under Modified i-SLIP scheduling reaches 100% [6, 10, 12].

The figure 1 represents an example of Modified i-SLIP scheduling of 3×3 VoQ packets. The number of iteration is fixed for 3.

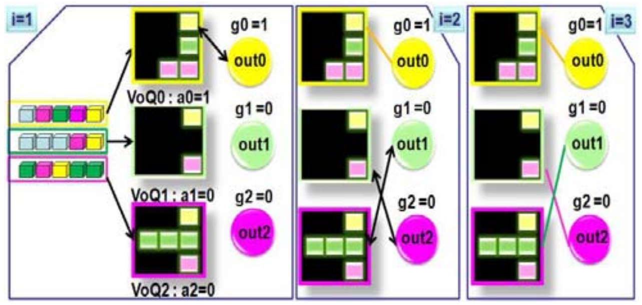


Figure 1. 3×3 VoQ packets scheduled with Modified 3-SLIP

1.2 Queuing mecanisms

Since we can have more than one packet at the same time for the same output, packets queuing became a need. We can distinguish three queuing types which are output queuing, input queuing and shared buffer [1, 11, 12]. In our case, we choose the VoQ as an input queuing mechanism since it is the only way to resolve the Head of Line (HoL) blocking [1]. Also, it reduces the loss probability [11, 12].

2. Scheduler design

The goal is to manage the communication of N hardware devices. Each device can be an input block as well as an output block. Thus, our design consists of N input arbiter, called accept arbiter, and N output arbiter or also grant arbiter. In addition, we need to have three input-to-output Swizzles to reorganize request vectors, grant vectors and accept vectors. In order to control the iteration number and arbiters updating, we utilize a Finite State Machine (FSM).

Both of grant arbiter and accept arbiter have the same architecture.

Our design has a structural architecture. In fact, an arbiter basically consists of a Programmable Priority Encoder (PPE). Moreover, programmable priority encoder can be made from tow Simple Priority Encoder (SPE) where the corresponding pseudocode is shown in figure 2.

```
Component SPE (Request_Vector) (Grant_Accept)
{
    for Device_i := 1 to Device_Nmbr do
        if (Request_Vector[Device_i] := 1) then
            Set_to_Zero_Vector (Grant_Accept)
            Grant_Accept [Device_i] =1
            Set_to_Zero_Vector (Request_Vector)
        else
            Grant_Accept = Grant_Accept
        end if
    end for
    Return Grant_Accept
}
```

Figure 2. Simple Priority Encoder pseudocode

PPE principal is described within its pseudocode presented in figure 3.

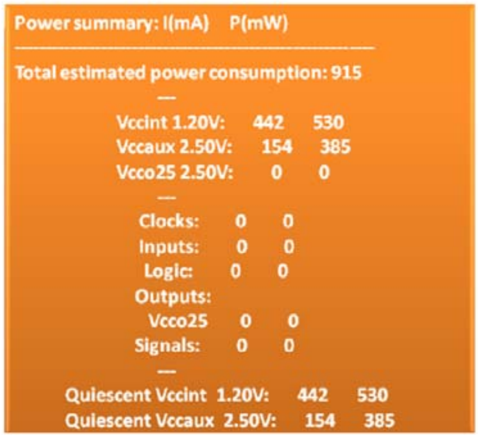
```
Component PPE (Request_Vector, HightPriority_Device)
                    (Grant_Accept , AnyGrant_Accept)
{
  Set_to_Zero_Vector (Grant_Accept)
      for Device_i := HightPriority_Devise to Device_Nmbr do
            if  (Request_Vector [Device_i] := 1) then
                        Grant_Accept [Device_i] = 1
                   Set_to_Zero_Vector (Request_Vector)
            else
                               Grant_Accept = SPE (Request_Vector)
            end if
            if (Grant_Accept != 0) then
                          AnyGrant_Accept = 1
            else
                    AnyGrant_Accept = 0
            end if
         end for
         Return Grant_Accept
}
```

Figure 3. Programmable Priority Encoder pseudocode

Figure 4 shows the arbiter pseudocode of the Modified i-SLIP.

```
Component Arbiter (Request_Vector, Arbiter_Enable,
Update_Enable) (Grant_Accept, AnyGrant_Accept)
{
 integer HightPriority_Devise = 1
 if (Arbiter_Enable := 1) then
                    Grant_Accept   =   PPE.Grant_Accept
(HightPriority_Devise,
                                          Request_Vector)
    AnyGrant_Accept = PPE.AnyGrant_Accept (
                                        HightPriority_Devise,
Request_Vector)
       if (Update_Enable := 1) then
          for Device_i := 1 to Device_Nmbr do
                if (Grant_Accept [Device_i] := 1) then
                     HightPriority_Device = Device_i + 1
                 else
                     HightPriority_Device = Device_i
                end if
             end for
        else HightPriority_Device = HightPriority_Device
  else
      Set_to_Zero_Vector (Grant_Accept)
      AnyGrant_Accept = 0
  end if
  }
```

Figure 4. Modified i-SLIP Arbiter pseudocode



```
Power summary: I(mA)    P(mW)
---------------------------------
Total estimated power consumption: 915
---
Vccint 1.20V:     442    530
Vccaux 2.50V:     154    385
Vcco25 2.50V:       0      0
---
Clocks:     0     0
Inputs:     0     0
Logic:      0     0
Outputs:
  Vcco25    0     0
Signals:    0     0
---
Quiescent Vccint 1.20V:     442    530
Quiescent Vccaux 2.50V:     154    385
```

Figure 5. Power sammary

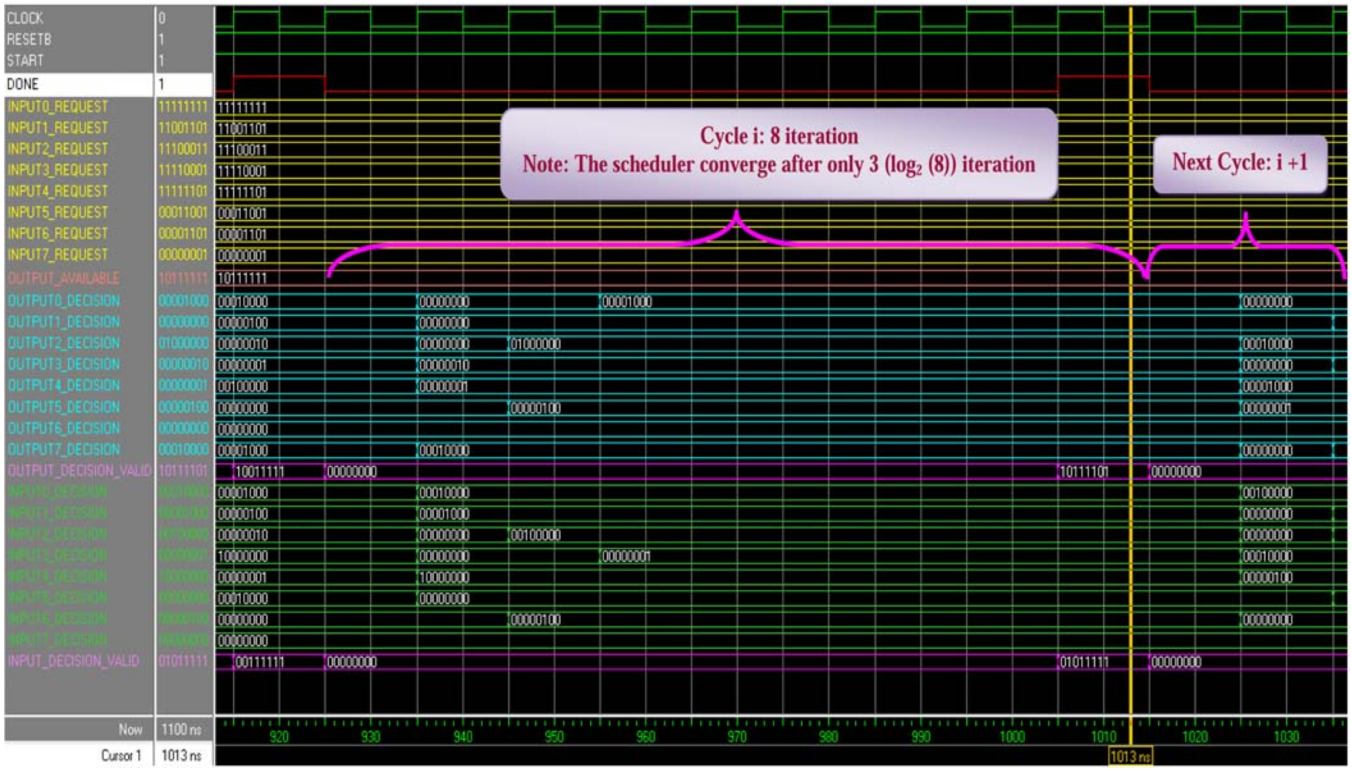


Figure 6. Modified i-SLIP scheduler simulation

3. Scheduler simulation and implementation

In this part, we will present simulation and implementation results of 8×8 input/output using Modelsim SE 6.2 full version as VHDL simulator and Xilinx ISE 8.1i WebPACK version as VHDL synthesizer.

The simulated example is characterized by :

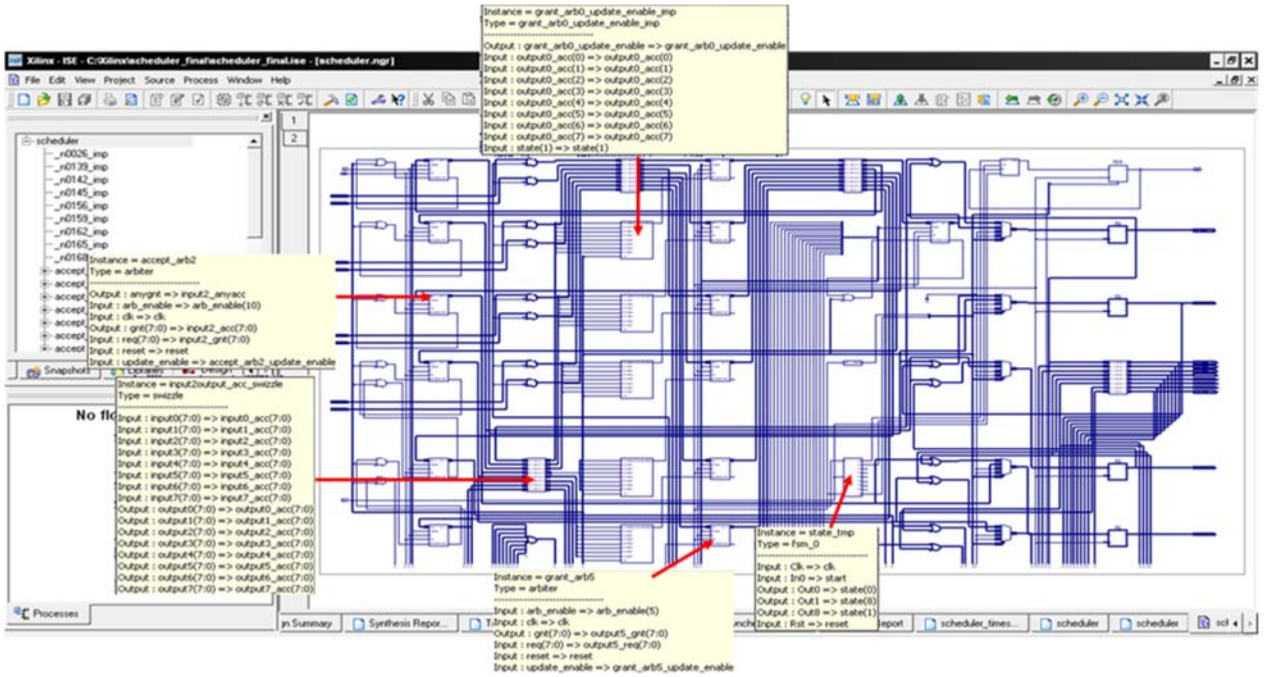
• $N = 8$; $m = 3$ (where $2^m = N$); $i = 8$;

• packet width = 72bits

• 8 grant arbiters (output side)

• 8 accept arbiters (input side)

• 3 swizzles


Xilinx - ISE - C:\Xilinx\scheduler_final\scheduler_final.ise - [scheduler.ngr]

```
Instance = grant_arb0_update_enable_imp
Type = grant_arb0_update_enable_imp
------------------------------
Output : grant_arb0_update_enable => grant_arb0_update_enable
Input : output0_acc(0) => output0_acc(0)
Input : output0_acc(1) => output0_acc(1)
Input : output0_acc(2) => output0_acc(2)
Input : output0_acc(3) => output0_acc(3)
Input : output0_acc(4) => output0_acc(4)
Input : output0_acc(5) => output0_acc(5)
Input : output0_acc(6) => output0_acc(6)
Input : output0_acc(7) => output0_acc(7)
Input : state(1) => state(1)
```

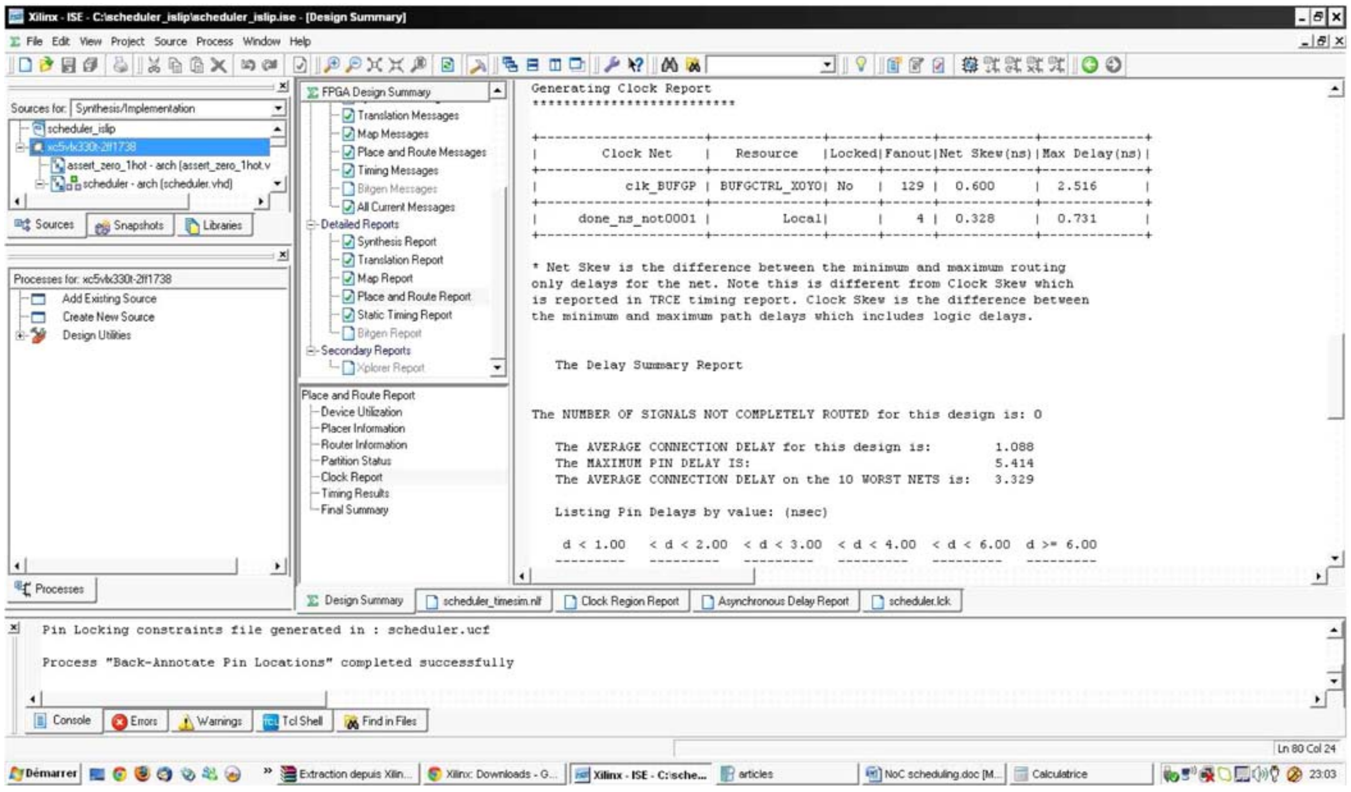
```
Instance = accept_arb2
Type = arbiter
------------------------------
Output : anygnt => input2_anyacc
Input : arb_enable => arb_enable(10)
Input : clk => clk
Output : gnt(7:0) => input2_acc(7:0)
Input : req(7:0) => input2_gnt(7:0)
Input : reset => reset
Input : update_enable => accept_arb2_update_enable
```

```
Instance = input2output_acc_swizzle
Type = swizzle
------------------------------
Input : input0(7:0) => input0_acc(7:0)
Input : input1(7:0) => input1_acc(7:0)
Input : input2(7:0) => input2_acc(7:0)
Input : input3(7:0) => input3_acc(7:0)
Input : input4(7:0) => input4_acc(7:0)
Input : input5(7:0) => input5_acc(7:0)
Input : input6(7:0) => input6_acc(7:0)
Input : input7(7:0) => input7_acc(7:0)
Output : output0(7:0) => output0_acc(7:0)
Output : output1(7:0) => output1_acc(7:0)
Output : output2(7:0) => output2_acc(7:0)
Output : output3(7:0) => output3_acc(7:0)
Output : output4(7:0) => output4_acc(7:0)
Output : output5(7:0) => output5_acc(7:0)
Output : output6(7:0) => output6_acc(7:0)
Output : output7(7:0) => output7_acc(7:0)
```

```
Instance = grant_arb5
Type = arbiter
------------------------------
Input : arb_enable => arb_enable(5)
Input : clk => clk
Output : gnt(7:0) => output5_gnt(7:0)
Input : req(7:0) => output5_req(7:0)
Input : reset => reset
Input : update_enable => grant_arb5_update_enable
```

```
Instance = state_tmp
Type = fsm_0
------------------------------
Input : Clk => clk
Input : In0 => start
Output : Out0 => state(0)
Output : Out1 => state(0)
Output : Out8 => state(1)
Input : Rst => reset
```

Figure 7. 8 × 8 Scheduler RTL Schematic


Xilinx - ISE - C:\scheduler_islip\scheduler_islip.ise - [Design Summary]

```
Generating Clock Report
**************************

+---------------------+--------------+------+------+------------+-------------+
|      Clock Net      |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+--------------+------+------+------------+-------------+
|           clk_BUFGP | BUFGCTRL_X0Y0| No   |  129 |  0.600     |  2.516      |
+---------------------+--------------+------+------+------------+-------------+
|     done_ns_not0001 |         Local|      |    4 |  0.328     |  0.731      |
+---------------------+--------------+------+------+------------+-------------+

* Net Skew is the difference between the minimum and maximum routing
only delays for the net. Note this is different from Clock Skew which
is reported in TRCE timing report. Clock Skew is the difference between
the minimum and maximum path delays which includes logic delays.


   The Delay Summary Report


The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0

   The AVERAGE CONNECTION DELAY for this design is:        1.088
   The MAXIMUM PIN DELAY IS:                               5.414
   The AVERAGE CONNECTION DELAY on the 10 WORST NETS is:   3.329

   Listing Pin Delays by value: (nsec)

    d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 6.00  d >= 6.00
   ---------   ---------   ---------   ---------   ---------   ---------
```

```
Pin Locking constraints file generated in : scheduler.ucf

Process "Back-Annotate Pin Locations" completed successfully
```

Figure 8. 8 × 8 Scheduler Clock Report

• 1 finite state machine

• 1 update_enable

3.1 Simulation

The simulation results of our Modified i-SLIP scheduler are illustrated in figure 6.

3.2 Synthesis results

The design was implemented in VHDL RTL, simulated with Modelsim 6.2 and synthesized using Xilinx-ISE and the Virtex-4 device XC4VFX100-12FFG1517 of 90 nm technology to achieve a maximum frequency of 265.88 MHz, a minimum slices utilization of 771 and a total estimated power consumption about 915 mW.

RTL schematic of the design is presented in Figure 6 and power summery results are described in Figure 5.

4. Conclusion

Modified (i-SLIP) scheduling algorithm has been investigated in order to implement our scheduler in VHDL. In fact, i-SLIP algorithm is:

• Simple to implement in hardware

• Fair, Starvation free, with throughput about 100%

• An algorithm which reduces the average burst length and Converges in *O* (*logN*).

We noticed that the slowest timing path through the scheduler passes through the programmable priority encoder logic of the grant/accept arbiters. Which means that optimized scheduler should be obtained for optimized arbiter.

References

[1] Young-Keun, P., Young-Keun, L. (2001). Parallel iterative matching-based cell scheduling algorithm for high-performance ATM switches Consumer Electronics, *IEEE Transactions*, 47, 134-137.

[2] Lee, T. T., Liew, S. Y. (2002). Parallel Routing Algorithms in Benes–Clos Networks, *IEEE Trans. Commun.*, 50 (11) 1841–1847.

[3] Jonathan Chao, H., Bin Liu. (2007). High Performance Switches and Routers, John Wiley & Sons, 27 avr., p. 232-234.

[4] Pindor, A. (1994). Experiences with implementing PIM (Parallel Iterative Methods) package on KSR1, *In*: Supercomputing Symposium' 94, Toronto, June.

[5] Lee, T. T., Soung, C., Liew. (2010). Principles of Broadband Switching and Networking, John Wiley & Sons, p. 58-63.

[6] Mckeown, N. W. (1995). Scheduling algorithms for input-queued cell switches, doctoral thesis. UMI order no. GAX96-02658, university of California: Berkeley, p. 20-52.

[7] Pape, J. (2006). Implementation of an On-chip Interconnect Using the i-SLIP Scheduling Algorithm: *Intermediate Report – Specification and Timeline*, p. 64-71.

[8] Cavendish, D., Lajolo, M., Liu, H. (2002). On the evaluation of fairness for input queue switches, *IEEE International Conference on Communications*. ICC 2002, V. 2.

[9] Varghese, G. (2005). Network Algorithmics: an interdisciplinary approach to designing fast networked devices, Morgan Kaufmann.

[10] McKeown, N. (1999). The iSLIP scheduling algorithm for input-queued switches, *IEEE/ACM Transactions On Networking*, 7 (2) 188–201.

[11] Sivaram, R., Stunkel, C. B., Panda, D. K. (2002). HIPIQS: A high-performance switch architecture using input queuing, *IEEE Trans. Parallel Distrib. Syst.*, 13 (3) 275–289, March.

[12] Cavendish, D., Goudreau, M., Ishii, A. (2001). On the fairness of scheduling algorithms for input-queued switches, *In*: Proc. 17th Int'l Teletraffic Congress, Brazil, 4, 829–841, December.