Network-on-Chip (NoC) architectures provide a good way of realizing efficient interconnections and largely alleviate the limitations of bus-based solutions.

NoC has emerged as a solution to problems exhibited by the shared bus communication approach in System-On-Chip (SoC) implementations. This includes the lack of scalability, clock skew, lack of support for concurrent communication, and power consumption.

Network-on-Chip communication is realized by packet, the communication requirement of this paradigm is affected by architecture parameters selection such as topology, mapping, routing, buffer size etc... Focusing on topology, there are 2 selections: regular topology and custom topology. These choices also affect hardware complexity and performance.

The challenges in this thesis are the designing of a prototype NoC with a real application including parallel execution and inserting Short Pass Link (SPL) to regular 2D-mesh topology NoC as an optimization technique and prototyping on FPGA to evaluate its performance and hardware complexity accurately. The optimization is executed based on our designed NoC named OASIS, JPEG encoder, which is selected as a target real application, is divided into 8 tasks to be mapped to each node. Then, design appropriate Network Interface (NI) for the application tasks, and improve the communication delay by employing the SPL insertion algorithm. Finally, I evaluated its communication performance improvement and accurate hardware utilization variance.

I prototyped the system in hardware and I evaluated its performance in terms of l-
tency and power using Dimension reversal transaction, Hotspot transaction, and a parallelized JPEG encoder. From the performance evaluation results, I concluded that the Dimension reversal execution time in ONoC with SPL decreased by 29.7%, when compared to the original base architecture, Hotspot execution time decreased by 16.9%, and JPEG encoder decreased by 43.7%. The area of these three test benches in ONoC with SPL increased under 5%, the Dimension reversal increased by 2.93%, the Hotspot increased by 2.60%, and the JPEG encoder increased by 4.28%. The power consumption slightly increased by 0.49% on average. The results indicate that the architecture is effective in balancing the power and performance of NoC design.
# Contents

Chapter 1 Introduction 6  
1.1 Background ............................................. 6  
1.2 Problems and Contributions .......................... 7  
1.3 Related Works .......................................... 9  
  1.3.1 Comparison between NoC and Bus ................. 9  
  1.3.2 FPGA Prototyping .................................. 11  
  1.3.3 Topology Optimizations ......................... 11  
1.4 Report Organization .................................. 12  

Chapter 2 OASIS Network-on-Chip Overview 13  
2.1 On Chip Interconnection .............................. 13  
  2.1.1 Topology and Routing ............................ 13  
  2.1.2 Switching ......................................... 14  
  2.1.3 Flow Control ..................................... 14  
  2.1.4 Arbiter ........................................... 14  
  2.1.5 Network Interface ................................. 15  
2.2 OASIS NoC Features .................................. 15  
2.3 OASIS Router Functions ............................... 16  

Chapter 3 Design Details 18  
3.1 Topology Design ....................................... 18  
3.2 OASIS Pipeline Design ............................... 20  
  3.2.1 Input port Design ................................ 21  
  3.2.2 Switch Allocator Design ......................... 22  
3.3 OASIS Arbiter Design ................................ 23  
  3.3.1 Crossbar Design .................................. 25  
3.4 Network Interface (NI) Design ....................... 26  
3.5 SPL Approach ......................................... 28  
  3.5.1 SPL Insertion Algorithm ......................... 29  
  3.5.2 Additional Ports Area Utilization ................. 31  
  3.5.3 SPL Modifications ................................ 32  
  3.5.4 Rewriting Codes for Modifications ................. 33  
3.6 Putting It All Together ............................... 35
Chapter 4 Evaluation Results

4.1 Environments and Parameters .................................................. 38
4.2 Dimension Reversal and Hotspot Simulation Results .................... 40
4.3 JPEG Encoder Simulation Results ............................................. 41
4.4 Hardware Design Analysis ....................................................... 42
4.5 Power Consumption Analysis ................................................... 43

Chapter 5 Conclusion .................................................................. 46

Chapter 6 OASIS Network-on-Chip with JPEG encoder Verilog-HDL code 54
Chapter 1

Introduction

1.1 Background

Current Systems-on-Chip (SoC) executes applications that demand extensive amounts of parallel processing. Network-On-Chip (NoC) [1], [2], [3] provides a good way of realizing interconnections on silicon and largely alleviates the limitations of bus-based solutions. Deep sub-micron processing technologies have enabled the implementation of new application-specific embedded architectures that integrate multiple software programmable processors and dedicated hardware components together onto a single chip. Recently, this kind of architecture has emerged as key design solutions for today’s non-electronic design problems, which are being driven by emerging applications in the areas of (1) wireless communication, (2) broadband/distributed networking, (3) distributed computing, and (4) multimedia computing. NoC is becoming an attractive option for solving bus-based systems problems. It is a scalable architectural platform with a huge potential to handle the increasing complexity and which can easily provide reconfigurability. In NoC architectures, processors are connected via a packet-switched communication network on a single chip. It is similar to the way that computers are connected to Internet. The packet-switched network routes information
between network clients (e.g. processors, memories, and custom logic devices). Figure 1.1 illustrates a typical NoC paradigm and point-to-point network. Packet switching supports asynchronous data transfer. It also provides extremely high bandwidth by distributing the propagation delay across multiple switches, effectively pipelining the packet transmission. In addition, NoC offers several promising features: First, it transmits packets instead of words. For that reason, dedicated address lines, like those in bus based systems, are not necessary since the destination addresses of packets are included in the packet. Second, transmission can be conducted in parallel if the network provides more than one transmission channel between a sender and a receiver. Thus, unlike bus-based Systems-on-Chip, NoC presents theoretical infinite scalability, facile IP core reusing, and higher level of parallelism.

![Figure 1.1: Typical paradigms (a) circuit switching, (b) packet switching model](image)

1.2 Problems and Contributions

We previously designed a NoC named OASIS which is an $n \times n$ mesh-topology. The NoC uses wormhole like switching, a First-Come-First-Served (FCFS) scheduler, and re-transmission flow control which is like ACK/NACK flow control. However the flow control, only source and destination core manage their flow control. OASIS NoC
Figure 1.2: Typical standard topology, (a) N x M mesh topology, (b) 3 stage Clos topology

is a regular NoC architecture based on a grid-topology [1].

In our previous work, we optimized OASIS and we called it OASIS-2. As an optimization, OASIS-2 employs an avoiding buffer overflow technique instead of OASIS re-transmission flow control (RFC) as flow control. The used avoiding buffer overflow technique efficiently prevents buffers from overflowing. Random Number Generator (RNG) transmission is used for simulation, and it achieves total transmission time is decreased by 19.5% with 4.38% extra area utilization in average.

OASIS-2 still has optimized spaces for target applications. I proposed Optimized OASIS NoC system (ONoC) which is a highly optimized for topology based on OASIS-2. OASIS-2 employed an \( n \times m \) mesh topology which is a simple layout perspective and the local interconnections between resources and switches are independent of the size of the network [6]. NoC topology has 2 choices; one is a standard such as Grid-Topology (Mesh), torus, star, and clos. Figure 1.2 illustrates standard topologies. These architectures are not concerned for dedicated applications. Each processing element (PE) is connected uniformly without consideration of the network load balance and spots where increasing traffic is concentrated which may cause the packet drops
or stall transactions, and then degrades the overall performance significantly. Some researchers attempted to solve these problems by increasing buffer size and the wires’ width at the expense of the area utilization and power consumption. The other topology choice is custom topology, whose router cannot re-use, and the routing algorithm is difficult to implement. The routers have identical components (numbers of port, physical wires), so the design time is longer than standard. Figure 1.3 illustrates a typical design of a custom topology. The significant increased design complexity of the NoC-based system causes unacceptable simulation time with the traditional simulation methods [7], [8]. The NoC architecture researches include a lot of trade-offs such as: topology, routing, switching, scheduling, flow control, buffer size, packet size, and any optimization techniques. It is difficult to analyze in only high-level simulations. Therefore NoC prototype is an essential research for evaluating the performance of the NoC architecture under real applications [10].

This report explores the algorithm to structure a partial custom topology, which is based on a standard 2D mesh topology and inserts short pass link (SPL) for long range communications and the high frequency communication in whole data communication. These two elements can cause performance decline significantly. Then, I prototyped NoC and measured the performance of some traffics and real application and hardware complexity.

1.3 Related Works

1.3.1 Comparison between NoC and Bus

NoC is becoming an attractive option for solving bus-based systems problems. Evaluation comparisons are presented between NoC and shared-bus AMBA [9]. The
AMBA protocol is an open standard, on-chip bus specification. A single-layer central multiplexer configuration with pipelined transaction is used in this work. The NoC employs mesh-based topology with deterministic XY routing with shortest path mapping between communicating cores. Single-flit-packet wormhole routing, and handshake flow control. For comparison, MPEG-2 video decoder traffic in cycle-accurate realistic simulation environment is used as real application. Supported by analytical and simulation results, it has been shown that the NoC reduces decoding time by a factor of 2.46 on average compared to AMBA [5]. Some researchers compare and contrast the Network-on-Chip (NoC) with Point-to-Point (P2P) and bus-based communication architectures. The performance of the NoC-based implementation is very close to that of the P2P for the same application. Moreover, the scalability analysis based on duplicating the bottleneck module in the MPEG-2 design shows that the performance of the NoC design scales as well as the P2P, while the bus-based implementation scales much more poorly [4].
1.3.2 FPGA Prototyping

The practical NoC interconnections’ fast and accurate performance evaluation and design survey are the critical issue. Some researchers present a flexible HW/SW emulation environment implemented on an FPGA that is suitable to explore, evaluate, and compare customized NoC [7]. However, they used Traffic Generator (TG), and did not use a real application. An implementation on FPGA and explained how to implement NoC with multi cores in details are explained [8], but its execution time measurement was in terms of assembly language instructions executed. These two papers mentioned how to deeply implement on FPGA and no parameter trade-offs.

1.3.3 Topology Optimizations

The IPs are mapped onto a standard topology or customized for a given application. Standard topology can reduce the design time but it probably includes idling ports, whole customized topology needs additional design time but it can enhance the performance. There is a design time and the performance trade-off. An approach that automatically selects the best topology for a given application and produces a mapping of cores onto that topology tool, named SUNMAP. The evaluation is executed with TG, and xpipes compiler [20] is used for hardware is presented in [19]. Fully customized topologies are shown in [21], which exploits the detailed understanding of the communication workload for optimization purpose, and they implemented proposed algorithms in a C++ package. These topology optimizations are mostly evaluated in software level.

In this report, the implementation on FPGA, and simulation under JPEG encoder as a real application is done. Moreover, the hardware complexity and execution time eval-
uation show the comparison with NoC architecture optimizations by inserting Short Pass Link (SPL).

1.4 Report Organization

This report presents the background, motivation, and some related works are introduced in chapter 1. Chapter 2 explains OASIS Network-on-Chip overview which is our previously designed NoC. Chapter 3 explains system architecture of the design in detail and Short Pass Link insertion algorithm as an optimization technique, which can reduce the execution time unless large increasing area. Chapter 4 explains evaluation results. Target device is Altera StratixIII and measurements of the execution time and hardware complexity results in details are shown. Last chapter concludes this report.
Chapter 2

OASIS Network-on-Chip Overview

2.1 On Chip Interconnection

The NoC interconnection paradigm is characterized by its topology, routing, switching, flow control, and arbiter. There are various trade-offs between hardware-cost and performance, and design-time and performance. So designers need to take care and deeply understand about all design choices.

2.1.1 Topology and Routing

As I explained in Chap.1, Topology is commonly chosen from two types, standard and customized. The routing methods are selected depending on the topology [11], [12]: standard topology (e.g. Mesh, Torus, Star, and etc.) can easily implement routing mechanism because each router sends the same calculation or comparison, but customized topology is necessary to design specific routing mechanism, so the design time may be longer than standard [17], [18]. Figure 2.1 illustrates 3 x 3 Network size mesh topology.
2.1.2 Switching

In packet switching, how the packets are passed between the routing nodes is known as switching. The switching strategy influences the buffering in the routing nodes and how resources are used [13]. In the switching strategies, Store and Forwarding (SF), Wormhole Switching (WS), and Virtual Cut Through (VCT) can be chosen. These switching methods transmit flits (flow control units), which are parts of individual packets.

2.1.3 Flow Control

Flits must be transmitted in a way that there are no dropped flits, or with some kind of resend protocol. Low power consumption, and calculation time, are not dependent on situation of the network and are ideal for flow control. ON/OFF, Credit-based, Handshaking, and ACK/NACK are the commonly used control flows used in NoC [16].

2.1.4 Arbiter

Mesh topology, inputs and outputs of routers mostly have five directions: north, east, south, west, and local. It is acceptable for routers to have multiple inputs, but
only one input stream is processed and routed. To solve this competition when only one input can access to the same output, designer can use virtual channel switch, where each port of the switch has multiple parallel buffers [15]. But, this causes a significantly hardware area utilization rise. Another solution is arbiter [14], it grants inputs which transmit for the same output stream.

2.1.5 Network Interface

The network interface (NI) has two functions, one is in the transmitter side, which divides a message data into payloads, generates flits and then transmits flits to the network. The other one is the receiver side, which assembles the incoming flits to generate the data message that will be delivered to the attached PE. Figure 2.2 illustrates brief sample NI implementation in NoC.

![Figure 2.2: NoC with Network Interface](image)

2.2 OASIS NoC Features

We previously designed a basic Network-on-Chip named OASIS [22]. The system is a n × m mesh-topology and uses wormhole switching, a First-Come-First-Served (FCFS) scheduler, and a novel re-transmission flow control (RFC). The FCFS scheduling is a simple algorithm and was carefully designed to be suitable for implementation in hardware. The RFC scheme is implemented in the processing elements (PEs) them-
selves; source nodes send checksum and the PEs of destination nodes confirm the correctness. If flits have been corrupted or dropped, they must be retransmitted. In this situation, new flits are resent from source PEs. Using this technique, OASIS can correctly transmit data even in noisy environments. The major drawback of OASIS is in its FCFS scheduler algorithm. The FCFS has unbalanced network utilization when looking at overall performance, because some transactions are served by the scheduler and other transactions are always stalled. This approach causes unbalanced network utilization and increases the latency because the retransmission technique must re-send flits if errors occur. In our previous work, we optimized OASIS and we called it OASIS-2. The system employs stall-go for avoiding buffer overflow, and matrix arbiter for scheduler. It also supports wormhole-like switching and virtual-cut-through forwarding method. The switching method which is chosen in a given instance depends on the level of packet fragmentation. Each router has input buffers which can store up to four flits. When a packet is divided into more than four flits, OASIS-2 chooses wormhole switching. When packets are divided into less than four flits, the system chooses virtual cut through. In other words, when buffer the size is greater than or equal to the number of flits, virtual cut through is used, but when buffer size is less than or equal to the number of flits, wormhole switching is employed.

2.3 OASIS Router Functions

Our early designed OASIS router provides extremely high bandwidth by distributing the propagation delay across multiple switches, thus pipelining the packet transmission. Our three-stage pipelined router architecture uses a speculative strategy based on a simple look-ahead routing, where each flit additionally carries one hot encoded
next-port identifier used by the downstream routers, in providing routing adaptation. Each router compares the current address and destination address to select the output port direction. OASIS router main functions are divided into routing calculation, arbitration & flow control, and data transmission. The first stage where; input flits are stored in the input buffer. When the buffers are almost full it, sends a signal to upstream neighbor router. On the second stage, routing calculation is done by using the stored flits’ information. The flits’ stored module sends Request signal to the arbitration module, and then the arbiter selects the winner to access the output, and sends Grant signal to the stored input module, this system based on least recently served scheme. The flow control is employed to avoid dropping flits, and it uses a state machine to manage the signals coming from downstream neighbor router about its input module status and also the state router’s output data. Finally, the third stage which transmits flits that have included updated routing information and fixed payload to the adequate output direction. The router supports pipelined routing, so all calculations and comparisons can execute in parallel.
Chapter 3

Design Details

In this Chapter, I explain OASIS architecture and router design with Verilog-HDL codes such as buffer, routing calculation, arbiter, flow control and crossbar. Then, As a topology optimization, SPL insertion algorithm and its modifications is explained.

3.1 Topology Design

![4x4 mesh topology](image)

Figure 3.1: 4x4 mesh topology
Figure 3.1 illustrates OASIS topology design, and it is described by Code 3.1. \( i \) indicates 5 in-out ports. Figure 3.2 illustrates one router, \( i = 0 \) is Local port, \( i = 1 \)
is "North" port, $i = 2$ is "East" port, $i = 3$ is "South" port and $i = 4$ is "West" port. OASIS employ $n \times m$ Mesh topology, code 3.1 indicates routers’ connection. Parameter $X_{\text{WIDTH}}$ and $Y_{\text{WIDTH}}$ means network size, when $i == 1$, the each router’s north input port receive data from south port of (current x, current y+1) router. ($y_{\text{pos}}==Y_{\text{WIDTH}}$) indicates the router position is north edge, so there are no input data from north port. All ports connections can be written same methods.

### 3.2 OASIS Pipeline Design

![A router block diagram](image)

Our designed router is a three-stage pipelined router architecture. In this chapter, we will explain the three stages: first stage includes "Buffer design". Second stage includes "Routing", Avoiding buffer overflow and "Scheduling", and the third stage includes "Crossbar". Figure 3.3 illustrates the router, left five modules are input port modules implementing buffers, routing scheme. Another module is the sw alloc module implementing scheduler and flow control, and finally the crossbar module implementing the crossbar scheme.
3.2.1 Input port Design

Code 3.2: Code for managing FIFO

```verilog
always @(posedge clk) begin
  if (!reset) begin //If out of reset
    if (enqueue) begin //Write a flit to the buffer
      fifo[tail_ptr] <= data_in;
      tail_ptr <= tail_ptr + 1;
    end
    if (dequeue) begin //Read a flit from the buffer
      head_ptr <= head_ptr + 1;
    end
  end
  //nearly full signal = stop_out,
  if (((tail_ptr + FULL_LVL[LOG2D-1:0] + 1'b1)==head_ptr) && enqueue && !dequeue)
    stop_out <= 1'b1;
  end
  if (((tail_ptr + FULL_LVL[LOG2D-1:0])==(head_ptr+1'b1)) && !enqueue && dequeue)
    begin
      stop_out <= 1'b1;
  end
  if ((tail_ptr + FULL_LVL[LOG2D-1:0])== head_ptr) begin
    if ((enqueue && !dequeue) || (!enqueue && dequeue)) begin
      stop_out <= 1'b0;
    end
  end
  else ...
end
```

The router has 5 independent input ports. Input port has two functions, "Buffering" and "Routing calculation". "Buffering" is written by Code 3.2, this module manage that FIFO header and tail calculation(2-9 lines), and stop_out signal is calculated (11-22 lines) for upstream router’s flow control. The signal is used in stop.go.v which designs a finite state machine. "Routing calculation" calculates look ahead XY routing, line 2-8 calculates next router address which is used routing calculation (11-18 lines).

Code 3.3: XY routing code

```verilog
//assign next addresses
if (nextport == `EAST) next_xaddr = xaddr + 1'b1;
else if (nextport == `WEST) next_xaddr = xaddr - 1'b1;
else next_xaddr = xaddr;
if (nextport == `NORTH) next_yaddr = yaddr + 1'b1;
else if (nextport == `SOUTH) next_yaddr = yaddr - 1'b1;
else next_yaddr = yaddr;
//evaluate next port
if (next_xaddr == xdest) begin
  if (next_yaddr == ydest) route = `SELF;
```
else if(next_yaddr < ydest) route = 'NORTH;
else route = 'SOUTH;
end else begin
  if (next_xaddr < xdest) route = 'EAST;
  else route = 'WEST;
end

3.2.2 Switch Allocator Design

"Switch Allocator" includes a scheduler "Matrix Arbiter" and "Stop Go" which is a flow control for avoiding packet overflow. Code 3.4 is a part of Matrix Arbiter. 1-15 lines generate grant i, and 17-23 lines calculate next state of all matrix elements, finally 25-30 updates states. This arbiter is shown in Fig refMatrixArbiter. Each row of the matrix means competitive inputs, and has priorities. After the highest priority input is served, the priority will be changed to lowest by inversing one’s row and column. For example, Fig. 3.4 (a) shows an example of how this matrix works: input 1 has the highest priority, after it is served (b), the first row and column values are inverse and input 1 changed to lowest priority. The highest priority input port means the row has all 1, so the arbiter gives "grant" for that.

Code 3.4: Matrix Arbiter code
3.3 OASIS Arbiter Design

As flow control, OASIS employs stop go technique. The used avoiding buffer overflow technique efficiently prevents buffer from overflowing. Data transfer is controlled by signals indicating buffer condition. In the absence of stall-go function, receiver cores need to judge whether there are dropped packets or not. If there are, the transmitter must resend the dropped packets using a receiving request signal from master cores. In addition, stall go scheme incurs communication blocking, but at the same time it can considerably reduce latency. Figure 3.6 (a) illustrates the state machine of this approach, and (b) shows the input FIFO situation of nearly full signal output. Code 3.5 shows how to design the above state machine. State "Go" indicates the receiving FIFO can store more than two flits. State "Sent" means that it can store one flit. State "Stop" means that it cannot store any more flits. Figure 3.5 illustrates stall go block diagram. There is no state transition from "Stop" to "Sent", because once the state
transition “Stop” happens, priority of this port is altered to low in sending router.

Figure 3.5: Stall go block diagram

Figure 3.6: (a) state machine design for stall go, (b) statement of nearly full signal output

Code 3.5: Code for stall go state machine

```verilog
always @(posedge clk) begin
  if (!reset) begin
    if ((state=="GO" && stop_in && data_sent)
      state <= 'SENT1;
      if (stop_in && !data_sent)
        state <= 'GO;
      if (!stop_in && data_sent)
        state <= 'STOP;
    end
  end else
    state <= 'GO;
  end
  if ((state=="STOP" && stop_in) // stop_in = nearly_full
    state <= 'GO;
  end else
    state <= 'GO;
  end
  assign blocked = !((state=="STOP") && !stop_in) || ((state=="SENT1") && !stop_in && data_sent);
```

24
3.3.1 Crossbar Design

"Crossbar" transmits flits to neighbor routers, OASIS crossbar code is written in Code 3.6. line 1-6 is crossbar.v which makes mux_outs relying on the numbers of output ports, line 8-31 is mux_out.v which transmits flits to adequate direction by cntrl.

![Crossbar Diagram](image)

**Figure 3.7: Arbiter control signals**

Code 3.6: Code for Crossbar

```verbatim
//crossbar.v
generate
  for (i=0;i<NOUT;i=i+1) begin:output_loop
    mux_out #(NIN, WIDTH) cbar_mux(.cntrl(cntrl_reg[NIN*(i+1)-1:NIN*i]), .data_in(data_in), .data_out(data_out[WIDTH*(i+1)-1:WIDTH*i]));
  endgenerate
//mux_out
generate
  //loop over each bit of data
  for (i=0;i<WIDTH;i=i+1) begin:bit_loop
    assign data_out[i] = mux(cntrl, data_bits[i]);
  //loop over each input channel
   for (j=0;j<n_in;j=j+1) begin:input_loop
    assign data_bits[i][j] = data_in[WIDTH*j+i];
  end
end
function mux;
  input [n_in-1:0] cntrl;
  input [n_in-1:0] data_in;
  integer i;
begin
  mux = 0;
  for (i=0; i<n_in; i=i+1) begin
    if(cntrl[i] == 1'b1) mux = data_in[i];
  end
endfunction // mux
```
3.4 Network Interface (NI) Design

First, before starting with the Network Interface design, we should set the flit structure. A message, which is the actual data of each processing element (PE), is divided into some payloads to generate flits, and the payloads are a part of flit structure [33]. The NI has two functions, one is the transmitter NI which divides a message data into...
payloads, generates flits with routing information and then transmits flits to the network. The other one is the receiver NI which assembles the incoming flits to generate the data message that will be delivered to the attached PE. In case where the receiver NI receives flits from different PEs, flits need to have source address information as tag. Figure 3.8 illustrates a sample of designed transmitter NI block diagram which divides input data into three flits, and the controller transmits those flits in three clock cycles and if a stall signal is received, it stops transmitting. Figure 3.9 presents the receiver NI block diagram which assembles three input data to PE data and sends at the same time. The NI designs are aimed for the Dimension reversal, Hotspot and JPEG Encoder application. Dimension reversal and Hotspot has same the NI design, JPEG encoder transmitted data size is 9 bits, 25 bits and 39 bits, as illustrated in Fig.3.11, four communication types are proposed. In this paper, we simplify the NI design and fit to
largest payload size, so each payload size is assigned to 39 bits, Fig 3.10 shows flits’ structure. However, if we reconfigure the flit structure and a part of NI design (which relies on the flit size), the NI will be able to be used for other target applications.

3.5 SPL Approach

In Mesh topologies designs, there is a possibility that some transactions may need larger latency or a lot of hops, and then it may become a bottleneck. To solve these problems, Short Pass Link (SPL) is proposed. Figure 3.12 illustrate a simple example of SPL. An ONoC router has 5 direction’s in-out ports (Local, North, East, South and West), but in order to realize SPL, an additional sixth port is needed for the Extra port. However, an additional port increases the area occupation and power consumption. Therefore, the SPL is inserted only between specific communications which include the possibilities to become a bottleneck among the whole communication patterns. If a lot of SPL are inserted without moderation, this may consume not only large area but also high power consumption. In order to avoid these problems, an available SPL budget should be carefully set.

Figure 3.12: Sample of Shot Pass Link
3.5.1 SPL Insertion Algorithm

![Diagram of SPL insertion algorithm]

Figure 3.13: SPL insert algorithm

This algorithm selects the communications which need SPL. First, we should decide the available SPL resource budget so we can prevent the increasing area utilization and power consumption, which are the drawback of SPL addition, Chap. 3.5.2 shows the SPL insertion influence in details. Next, we calculate and get the communication costs for all communications patterns with their communication frequencies and distances. Depending on these calculations, the SPL is inserted to the highest cost communication. Simulation and comparison result, compare updated design and pre-updated design and check if there are possibilities for the SPLs insertion to reduce the performance. After adding the SPL, the algorithm goes back around all communication cost calculations until the available SPL budgets are exhausted.
S: available resource
i: target sender router
j: target receiver router

$f_{ij}$: target communication frequency

$C_{ij}$: target communication total cost

\[
f_{ij} = \max \frac{V_{ij}}{\sum_p \sum_{p \neq q} V_{pq}}
\]  \hspace{1cm} (3.1)

\[
d_M(i, j) = |i_x - j_x| + |i_y - j_y|
\]  \hspace{1cm} (3.2)

\[
C_{ij} = f_{ij} \times d_M(i, j)
\]  \hspace{1cm} (3.3)

Figure 3.13 illustrates the SPL Insert Algorithm. Formula 1 is used to measure the communication frequency by calculating the whole communication with all neighbor nodes transaction volume, and the target communication for the whole neighbor node usability volume which means $p$ is always neighbor of $q$. The whole communication volume is expressed by $\sum_p \sum_{p \neq q} V_{pq}$; where $p$ indicates the sender node, $q$ indicates the receiver node, $p$ and $q$ are always neighbors. Then, The target communication frequency is expressed by $V_{ij}$; where $i$ indicates the sender node, and $j$ indicates the receiver node. To calculate distance of communications, Manhattan distance is employed (Formula 2), the address of $i$ is expressed by $(i_x, i_y)$, and the address of $j$ is expressed by $(j_x, j_y)$. Finally, the cost calculation is computed (Formula 3) using values from (1) and (2).
3.5.2 Additional Ports Area Utilization

Table 3.1: 5-ports OASIS Router Area Utilization

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Input port</th>
<th>Switch Allocator</th>
<th>Crossbar</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUTs</td>
<td>1-port</td>
<td>71(7.4%)</td>
<td>300(31%)</td>
<td>310(32.1%)</td>
</tr>
<tr>
<td></td>
<td>5-ports</td>
<td>355(36.8%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td>1-port</td>
<td>72(15.2%)</td>
<td>90(18.9%)</td>
<td>25(5.3%)</td>
</tr>
<tr>
<td></td>
<td>5-ports</td>
<td>360(75.8%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: 6-ports OASIS Router Area Utilization

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Input port</th>
<th>Switch Allocator</th>
<th>Crossbar</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUTs</td>
<td>1-port</td>
<td>75(5.8%)</td>
<td>469(36.5%)</td>
<td>366(28.5%)</td>
</tr>
<tr>
<td></td>
<td>6-ports</td>
<td>450(35%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Registers</td>
<td>1-port</td>
<td>99(13.3%)</td>
<td>144(19.4%)</td>
<td>36(4.8%)</td>
</tr>
<tr>
<td></td>
<td>6-ports</td>
<td>594(79.8%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The logic structure of Stratix III devices delivers unprecedented performance and logic efficiency. This logic structure is built from basic logic units known as adaptive logic modules (ALMs). Each ALM contains a variety of (look-up table) LUT-based resources, two full adders, carry-chain segments, two flipflops, and many additional logic enhancements that can be flexibly divided into two adaptive LUTs (ALUTs) [29]. Quartus II compilation report shows logic utilization which includes combinational ALUTs, memory ALUTs and dedicated logic registers. In my design, combinational ALUTs and dedicated logic registers are mostly used. Table 3.1 shows a 5-ports router area utilization and table 3.2 shows a 6-ports router area utilization. The 6 ports router’s ALUTs utilization increases 33.2% and registers utilization increase 56.6%. OASIS has identical routers for all nodes, so if somewhere that needs additional port by SPLs, it significantly increases area utilization. If designers insert a lot of SPLs, the area utilization significantly increase, so it is recommended to consider design restrictions.
3.5.3 SPL Modifications

![Figure 3.14: Extra-Port insertion](image)

The proposed low latency, Optimized OASIS NoC system (ONoC) is a highly optimized for target application based on OASIS-2. The main ONoC modifications are: an in-out port addition of a router, the modification of the network’s connections design and the flit structure. An initial ONoC router has 5 in-out ports for Local, North, East, South and West directions. To configure SPL commutation, it is essential to add the Extra-Port for the routers hosting the SPL. Figure 3.14 shows the Extra-Port for both sender and receiver node. Other routers keep having the same number of 5 ports to prevent increasing the area utilization for extra port uselessly. Another necessary modification is the flit structure. ONoC has 5 bits dedicated to define the next port direction. So it’s necessary to extend the next port field from 5 bits to 6 bits for the SPL as an additional direction, so the whole flit also needs to extend 1 bit. The final modification is the network connections, between two routers in Mesh topology which are connected by SPL.

Figure 3.15, 3.16 and 3.17 illustrates each application mapping with SPL links, we assume resource budget is 5% of original area utilization, then following the algorithm,
Dimension reversal and Hotspot budgets are exhausted at two SPLs, so only two SPLs will be implemented. JPEG encoder is also executed, then SPL selects three SPLs. Additionally, each node which needs to add SPL addressed edge of the topology, it means unused port can be used for SPL. For example, dimension reversal (3,0) east and south in-out ports are free, so SPL input selects east, and output port also selects south port. Now, we must modify the program directly to be performed manually.

3.5.4 Rewriting Codes for Modifications

In this sub-chapter, code modifications are explained. As I explained previous chapter, NoC architecture is necessary to modify for SPLs. Code 3.7 shows modifi-
cation for SPLs insertion. I modified the loop function for Mesh topology, because it is necessary to simplify the connection of all routers (line 1-4). Then, designers can insert SPLs. for example, line 15-17 shows the address (0,3) north output connects to (1,0) south input port, the address (1,0) south port has no connections, so the south port can use for a SPL without adding extra ports. Another, modification is routing calculation. OASIS router employs look ahead XY routing, so the routing calculates next router’s output direction. The SPL is inserted from source node to destination node directly, so flits transmission by SPL always the next port is local port. The line 6-13 shows routing calculation for (0,3) to (1,0) transmission.

Code 3.7: Code Modification for NoC architecture

```vhdl
//y loop
for (y_pos=0; y_pos<Y_WIDTH; y_pos=y_pos+1) begin:y_loop2
  //x loop
  for (x_pos=0; x_pos<X_WIDTH; x_pos=x_pos+1) begin:x_loop2
    ...
    ///(1,0)
    if (x_pos == 1) begin
      if (y_pos == 0) begin
        //tile interface of router, x_pos = 1, y_pos = 0, i = 0
        ...
        //north edge of router, x_pos = 1, y_pos = 0, i = 1
        ...
        //east edge of router, x_pos = 1, y_pos = 0, i = 2
        ...
        //south edge of router, x_pos = 1, y_pos = 0, i = 3
        assign net_data_in [x_pos][y_pos][`WIDTH*(3+1)-1:`WIDTH*3] = net_data_out [0][3][`WIDTH*3-1:`WIDTH*1];
        assign net_stop_in [x_pos][y_pos][3] = net_stop_out [0][3][1];
      end
      else next_yaddr = yaddr + 1'b1;
    end
    ...
  end
end
```

Code 3.8: Code Modification for look a head routing

```vhdl
if(nextport == 'WEST) begin
  next_xaddr = xaddr - 1'b1;
end else if (nextport == 'EAST) next_xaddr = xaddr + 1'b1;
else next_xaddr = xaddr;
```

```vhdl
if(nextport == 'NORTH) begin
  if((xaddr==0&yaddr==3)&&(xdest==1&ydest==0)) begin
    next_xaddr = 1;
    next_yaddr = 0;
  end
  else next_yaddr = yaddr + 1'b1;
```

34
### 3.6 Putting It All Together

![Diagram of the top module of designed circuit with file names](image)

Figure 3.18: The top module of designed circuit with file names

Figure 3.18 illustrates whole system architecture with file names, and Figure 3.19 shows a router system architecture. Figure 3.20 shows ONoC with JPEG encoder file hierarchy.

Top module of ONoC and ONoC with JPEG encoders are attached as Appendix.
Figure 3.19: Router top module with file names
Figure 3.20: ONoC with JPEG encoder file hierarchy
Chapter 4
Evaluation Results

The proposed low latency, Optimized OASIS NoC system (ONoC) is a highly optimized version with SPL insert algorithm of our earlier designed architecture (OASIS-2). In this chapter, I would like to explain the comparison OASIS-2 and ONoC with SPL algorithm at the hardware and execution time.

As explained in Chap. 3.5, the target applications are Dimension Reversal, Hotspot and JPEG encoder. Each application is designed in Verilog-HDL.

4.1 Environments and Parameters

The proposed architecture is designed in Verilog-HDL. Synthesis with industrial CAD tools (Altera Quartus II ver. 11.0), and simulated with ModelSim-Altera 6.6. The target device is Altera Stratix III EP3SL150F1152C2. Table 4.1 shows design and simulation parameters for Dimension reversal test case, input data is continuously injected, both transmitter and receiver nodes are assigned (6 nodes), and each transmitter node sends one thousand flits, and receivers verifies receiving flits. Table 4.2 shows design and simulation parameters for Hotspot test case. The both transmitter and receiver nodes are equal to 8 nodes and continuously injecting flits. Each transmitter sends one thousand flits, and receivers verify received flits. The Dimension reversal and Hotspot
transmit the random number traffic. There are no calculations at PEs, so clock is set to only one type. Table 4.3 shows JPEG encoder design and simulation parameters. Input RGB data is inputted into the RGB data module. Input RGB data ratio is $96 \times 96$ (27,906 bytes). The input to the module is a 24-bit data bus with 8 bits each for the Red pixel, Green pixel, and Blue pixel. JPEG encoder whose specifications strict the maximum clock speed, and each PE waits receiving input data, so there are two types of clock cycles are defined clock for JPEG encoder and network interconnection. Our algorithm (Fig.3.13) starts from budget selection, the budget is assumed to 5% of extra hardware of original design’s.

Table 4.1: Dimension reversal simulation environment

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OASIS-2</th>
<th>ONoC with SPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology</td>
<td>Mesh</td>
<td>Mesh with SPL</td>
</tr>
<tr>
<td>Budget</td>
<td>0</td>
<td>5%</td>
</tr>
<tr>
<td>Flow control</td>
<td>ACK/NACK(at PE)</td>
<td>Stall go</td>
</tr>
<tr>
<td>Routing Algorithm</td>
<td>XY-routing</td>
<td>XY-routing</td>
</tr>
<tr>
<td>Switching Method</td>
<td>Wormhole</td>
<td>Wormhole</td>
</tr>
<tr>
<td>Flit Size</td>
<td>28 bit</td>
<td>28 bit</td>
</tr>
<tr>
<td>Buffer Depth</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.2: Hotspot simulation environment

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OASIS-2</th>
<th>ONoC with SPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology</td>
<td>Mesh</td>
<td>Mesh with SPL</td>
</tr>
<tr>
<td>Budget</td>
<td>0</td>
<td>5%</td>
</tr>
<tr>
<td>Flow control</td>
<td>ACK/NACK(at PE)</td>
<td>Stall go</td>
</tr>
<tr>
<td>Routing Algorithm</td>
<td>XY-routing</td>
<td>XY-routing</td>
</tr>
<tr>
<td>Switching Method</td>
<td>Wormhole</td>
<td>Wormhole</td>
</tr>
<tr>
<td>Flit Size</td>
<td>28 bit</td>
<td>28 bit</td>
</tr>
<tr>
<td>Buffer Depth</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Table 4.3: JPEG encoder simulation environment

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OASIS-2</th>
<th>ONoC with SPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology</td>
<td>Mesh</td>
<td>Mesh with SPL</td>
</tr>
<tr>
<td>Budget</td>
<td>0</td>
<td>5%</td>
</tr>
<tr>
<td>Flow control</td>
<td>ACK/NACK(at PE)</td>
<td>Stall go</td>
</tr>
<tr>
<td>Routing Algorithm</td>
<td>XY-routing</td>
<td>XY-routing</td>
</tr>
<tr>
<td>Switching Method</td>
<td>Wormhole</td>
<td>Wormhole</td>
</tr>
<tr>
<td>Flit Size</td>
<td>55 bit</td>
<td>55 bit</td>
</tr>
<tr>
<td>Buffer Depth</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

4.2 Dimension Reversal and Hotspot Simulation Results

In this section, the simulation results will be explained. The number of SPL depends on the budget selection. The budget is assumed to 5% extra hardware of original design’s one, Table 4.4 shows under 5% additional area, and finally the number of SPL is decided to two. Hotspot’s budget (Table 4.5) also assumed to 5% and finally SPL is set to two. Figure 4.1 shows the comparison results between the base architecture (OASIS-2) and the proposed ONoC with SPLs. The execution time includes packetization and depacketization time. The throughput means the average transaction flits in a cycle. Figure 4.1 shows execution time and throughput of original OASIS-2, ONoC-SPL1, and ONoC-SPL2. The throughput increased by 49.6 % and the execution time reduced to 29.7 % at ONoC SPL3. Figure 4.2 shows Hotspot simulation result, and the throughput increased by 24.8 %, and the execution time reduced to 16.9 % at ONoC SPL2.

Table 4.4: Hardware complexity: Dimension reversal

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OASIS-2</th>
<th>ONoC-SPL1</th>
<th>ONoC-SPL2</th>
<th>ONoC-SPL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (ALUTs)</td>
<td>9,008 (8%)</td>
<td>9,119 (8%)</td>
<td>9,272 (8%)</td>
<td>9,605 (8%)</td>
</tr>
<tr>
<td>Speed (MHz)</td>
<td>182.78</td>
<td>169.58</td>
<td>171.53</td>
<td>167.28</td>
</tr>
<tr>
<td>Power consumption (mW)</td>
<td>660.61</td>
<td>663.12</td>
<td>662.54</td>
<td>661.46</td>
</tr>
<tr>
<td>Additional hardware</td>
<td>0</td>
<td>1.23%</td>
<td>2.93%</td>
<td>6.65%</td>
</tr>
</tbody>
</table>
Figure 4.1: Dimension reversal simulation result

Table 4.5: Hardware complexity: Hotspot

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OASIS-2</th>
<th>ONoC-SPL1</th>
<th>ONoC-SPL2</th>
<th>ONoC-SPL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (ALUTs)</td>
<td>10,041 (9%)</td>
<td>10,130 (9%)</td>
<td>10,302 (9%)</td>
<td>10,911 (10%)</td>
</tr>
<tr>
<td>Speed (MHz)</td>
<td>187.62</td>
<td>180.41</td>
<td>178.83</td>
<td>169.78</td>
</tr>
<tr>
<td>Power consumption (mW)</td>
<td>666.01</td>
<td>667.29</td>
<td>667.33</td>
<td>669.20</td>
</tr>
<tr>
<td>Additional hardware</td>
<td>0</td>
<td>0.89%</td>
<td>2.60%</td>
<td>8.66%</td>
</tr>
</tbody>
</table>

4.3 JPEG Encoder Simulation Results

JPEG encoder simulation result will be explained here. The SPL budget is assumed to 5% of original OASIS-2, and the number of SPL is selected as three from Table 4.6. Figure 4.3 shows the throughput, the execution time, and latency. The latency indicates network interconnection time for one JPEG encoder clock. The throughput increases by 22.6%, the latency decreased by 41.7% and the execution time decreased by 43.7% at ONoC SPL3. Figure 4.4 shows the floorplan generated for the JPEG encoder NoC in the Altera Stratix III EP3SL150F1152C2 device.
4.4 Hardware Design Analysis

The hardware complexity analysis will be explained in this section, Altera Quartus II report, which shows the area utilization, clock speed, and this tool provides Power Analysis tools to estimate power consumption [34]. TimeQuest Timing Analyzer is also supported by the tool and it shows FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock [35]. Target device has 113,600 ALUTs, each table shows area utilization ratio based on the number of ALUTs. The area of these three test benches in ONoC which has SPL increased under

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OASIS-2</th>
<th>ONoC-SPL1</th>
<th>ONoC-SPL2</th>
<th>ONoC-SPL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (ALUTs)</td>
<td>28,401(25%)</td>
<td>28,733 (25%)</td>
<td>29,138 (26%)</td>
<td>29,616 (26%)</td>
</tr>
<tr>
<td>Speed (MHz)</td>
<td>193.8</td>
<td>173.13</td>
<td>176.03</td>
<td>171.79</td>
</tr>
<tr>
<td>Power consumption (mW)</td>
<td>815.90</td>
<td>815.84</td>
<td>823.32</td>
<td>823.92</td>
</tr>
<tr>
<td>Additional hardware</td>
<td>0</td>
<td>1.17%</td>
<td>2.59 %</td>
<td>4.28%</td>
</tr>
</tbody>
</table>
5%, the Dimension reversal increased by 2.93%, the Hotspot increased by 2.60%, the JPEG encoder increased by 4.28%.

4.5 Power Consumption Analysis

To analyze power consumption, the Altera Quartus II PowerPlay Power Analysis accurately estimates our design power consumption. These three simulations are analyzed with ModelSim-Altera gate-level simulation. The power analysis tool, Signal activity and static probability information derive from a Verilog Value Change Dump File [34]. Table 4.4, 4.5, and 4.6 also show power consumptions. Dimension reversal ONoC SPL2 increased by 1.93 mW (0.29%), Hotspot ONoC SPL1 increased by 1.32 mW (0.20%), and JPEG encoder ONoC SPL3 increased by 8.02 mW (0.98%).
Every result shows increasing power consumption a little. The power consumption is estimated by dynamic power dissipation, static power dissipation, and I/O power dissipation. SPLs insertion affects dynamic and static power dissipations, both parameters increase slightly.
Figure 4.4: Chip floor plan for JPEG encoder system implementation
Chapter 5

Conclusion

In this report, I presented architecture, design, and evaluation of a Network-on-Chip architecture which utilizes several advanced optimization techniques. The software NoC simulation tools are not enough to evaluate accurate a NoC parameters’ effects and trade-offs, it is not accurate hardware complexity and performance, because it uses many assumptions and approximations. So, I prototyped the system in hardware and evaluated its performance in terms of execution time, area utilization, and power using Dimension reversal transaction, Hotspot transaction, and a parallelized JPEG encoder.

OASIS NoC employs mesh topology which is no careful consideration for given applications, but designing customized topology takes long time and it may occupy large area utilizations, so I proposed SPL insertion algorithm to reduce latency which directly affect performance. The algorithm selects which communications are needed to insert a SPL, and the number of SPLs is decided by depending on hardware budgets. From the performance evaluation results, I conclude that the Dimension reversal execution time in ONoC decreased by 29.7%, when compared to the original base architecture, and Hotspot execution time decreased by 16.9%, and JPEG encoder decreased
by 43.7% The area of these three test benches in ONoC which has SPL increased under 5%, the Dimension reversal increased by 2.93%, the Hotspot increased by 2.60%, the JPEG encoder increased by 4.28%, and the power consumption slightly increased by 0.49% on average. The results indicate that the architecture is effective in balancing the power and performance of NoC design.
References


Chapter 6

OASIS Network-on-Chip with JPEG encoder Verilog-HDL code

The developed OASIS Network-on-Chip with JPEG encoder is implemented in the Verilog-HDL code. The codes are attached in this appendix.

CODE DESCRIPTION
Code 6.1: Integration JPEG tasks and NoC

// This module integrates all "JPEG tasks" nodes modules
// and NoC top module "network"

'timescale 1ns / 100ps

'define n00 54:0
'define n10 109:55
'define n20 164:110
'define n30 219:165
'define n01 274:220
'define n11 329:275
'define n21 384:330
'define n31 439:385

'define packet 55//packet size
'define ns 8//network size X x Y

module t4x2_top(clk ,rst ,n_clk ,
    //(0,0) input
    end_of_file_signal ,
    enable ,
    data_in ,
    // (1,1) output
    // Net_flit_out ,
    // Net_flit_in ,
    JPEG_bitstream ,
    data_ready ,
    end_of_file_bitstream_count ,
    eof_data_partial_ready
);

input clk ,rst;//clk for JPEG process
input n_clk;//network clock
input end_of_file_signal;
input enable;
input [23:0] data_in;

output [31:0] JPEG_bitstream;
output data_ready;
output [4:0] end_of_file_bitstream_count;
output eof_data_partial_ready;

//integration wire
//output
wire [`ns*`packet -1:0] Net_flit_out;

//Network input from NIs
// output
wire [`ns*`packet -1:0] Net_flit_in;

//STALL GO
reg [`ns -1:0] stop_in;
wire [`ns -1:0] stop_out;

node_01 n01(.enable(enable),
    .data_in(data_in),
    .flit(Net_flit_in[`n01]),
); //RGB data input (0,1)

node_00 n00(
    .clk(clk),
    .n_clk(n_clk),
    .rst(rst),

.flit_in(Net_flit_out[\'n00]),
.fr.it_out(Net_flit_in[\'n00])
}; //RGBtoYCrCb //((0), (0)

node_10 n10{
  .n_clk(n_clk),
  .clk(clk),
  .rst(rst),
  .flit_in(Net_flit_out[\'n00]),
  .flit(Net_flit_in[\'n00])
}; //CR DCT Quantization Huffman

node_11 n11{
  .n_clk(n_clk),
  .clk(clk),
  .rst(rst),
  .flit(Net_flit_out[\'n11]),
  .flit_out(Net_flit_in[\'n11])
}; //CB Quantization Huffman

node_21 n21{
  .n_clk(n_clk),
  .clk(clk),
  .rst(rst),
  .flit(Net_flit_out[\'n21]),
  .flit_out(Net_flit_in[\'n21])
}; //Y Quantization Huffman

node_20 n20{
  .n_clk(n_clk),
  .clk(clk),
  .rst(rst),
  .flit(Net_flit_out[\'n20]),
  .flit_out(Net_flit_in[\'n20])
}; //FIFO to synchronize

node_30 n30{
  .clk(clk),
  .n_clk(n_clk),
  .rst(rst),
  .flit_in(Net_flit_out[\'n30]),
  .end_of_file_signal(end_of_file_signal),
  .end_of_file_bitstream_count(end_of_file_bitstream_count),
  .data_ready(data_ready),
  .eof_data_partial_ready(eof_data_partial_ready)
}; // check FF code

node_31 n31{
  .rst(rst), .clk(clk),
  .flit_in(Net_flit_out[\'n31]),
  .JPEG_bitstream(JPEG_bitstream),
  .data_ready(data_ready),
  .end_of_file_bitstream_count(end_of_file_bitstream_count),
  .eof_data_partial_ready(eof_data_partial_ready)
};//(1,1) JPEG output

network network(.clk(n_clk),
  .reset(rst),
  .data_in(Net_flit_in),
  .data_out(Net_flit_out),
  .stop_in(stop_in),
  .stop_out(stop_out)
);

assign Net_flit_in[\'n31] = 55'h0;
always @(posedge clk or posedge rst)
begin
if(rst)begin
  stop_in <= 0;
end

assign Net_flit_in[\'n31] = 55'h0;
always @(posedge clk or posedge rst)
begin
if(rst)begin
  stop_in <= 0;
end

Code 6.2: Network on Chip top module

```verilog
/*
 * Top level network - Instantiates Mesh with SPLs
 */

`include "defines.v"

module network(clk, reset,
    data_in, data_out,
    stop_in, stop_out);

//network size
parameter X_WIDTH = 4;
parameter Y_WIDTH = 2;

//router parameters
parameter NOUT = 5;
parameter FIFO_FULL_LVL = 2;
parameter FIFO_DEPTH = 4;
parameter FIFO_LOG2D = 2;

input clk, reset;

input [X_WIDTH*Y_WIDTH -1:0] stop_in;
output [X_WIDTH*Y_WIDTH -1:0] stop_out;

//inout for original
input [X_WIDTH*Y_WIDTH*`WIDTH -1:0] data_in;
output [X_WIDTH*Y_WIDTH*`WIDTH -1:0] data_out;

wire [(`WIDTH*NOUT)-1:0] net_data_out [X_WIDTH -1:0][Y_WIDTH -1:0];
wire [(`WIDTH*NOUT)-1:0] net_data_in [X_WIDTH -1:0][Y_WIDTH -1:0];
wire [NOUT -1:0] net_stop_out [X_WIDTH -1:0][Y_WIDTH -1:0];
wire [NOUT -1:0] net_stop_in [X_WIDTH -1:0][Y_WIDTH -1:0];

genvar i, x_pos, y_pos;

// Original
wire [(`WIDTH*NOUT)-1:0] net_data_out [X_WIDTH-1:0][Y_WIDTH-1:0];
wire [(`WIDTH*NOUT)-1:0] net_data_in [X_WIDTH-1:0][Y_WIDTH-1:0];
wire [NOUT-1:0] net_stop_out [X_WIDTH-1:0][Y_WIDTH-1:0];
wire [NOUT-1:0] net_stop_in [X_WIDTH-1:0][Y_WIDTH-1:0];

genvar i, x_pos, y_pos;

// i indicates port, 0:local, 1:north, 2:east, 3:south, 4:west

generate
    for (y_pos=0; y_pos<Y_WIDTH; y_pos=y_pos+1) begin:y_loop
        for (x_pos=0; x_pos<X_WIDTH; x_pos=x_pos+1) begin:x_loop
            router #(NOUT, FIFO_DEPTH, FIFO_LOG2D, FIFO_FULL_LVL)
                rtr2(.clk(clk), .reset(reset),
                    .data_in(net_data_in[x_pos][y_pos]),
                    .data_out(net_data_out[x_pos][y_pos]),
                    .stop_in(net_stop_in[x_pos][y_pos]),
```
end
close
endgenerate

generate
//y loop
for (y_pos=0; y_pos<Y_WIDTH; y_pos=y_pos+1) begin:
y_loop2
//x loop
for (x_pos=0; x_pos<X_WIDTH; x_pos=x_pos+1) begin:
x_loop2

//set up inter-router connections with correct boundary conditions
if (x_pos == 0) begin
if (y_pos == 0) begin
//tile interface of router, x_pos = 0, y_pos = 0, i = 0 //
assign net_data_in[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH*0] = data_in[(WIDTH* X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH*y_pos)+1];
assign data_out[(WIDTH*X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH* y_pos)+(WIDTH*x_pos)] = net_data_out[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH *0];
assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];

//north edge of router, x_pos = 0, y_pos = 0, i = 1 //
assign net_data_in[x_pos][y_pos][WIDTH*(1+1)-1:WIDTH*1] = net_data_out[x_pos +1][y_pos][WIDTH*(3+1)-1:WIDTH*3];
assign net_stop_in[x_pos][y_pos][1] = net_stop_out[x_pos+1][y_pos][3];
//east edge of router, x_pos = 0, y_pos = 0, i = 2
assign net_data_in[x_pos][y_pos][WIDTH*(2+1)-1:WIDTH*2] = net_data_out[x_pos +1][y_pos][WIDTH*(4+1)-1:WIDTH*4];
assign net_stop_in[x_pos][y_pos][2] = net_stop_out[x_pos+1][y_pos][4];
//south edge of router, x_pos = 0, y_pos = 0, i = 3
assign net_data_in[x_pos][y_pos][WIDTH*(3+1)-1:WIDTH*3] = 0;
assign net_stop_in[x_pos][y_pos][3] = 1'b1;
//Insertion SPL from (0,0) to (2,2), west -> extra
assign net_data_in[x_pos][y_pos][WIDTH*(4+1)-1:WIDTH*4] = 0;
assign net_stop_in[x_pos][y_pos][4] = 1;
end
end
if (x_pos == 1) begin
if (y_pos == 0) begin
//tile interface of router, x_pos = 1, y_pos = 0, i = 0 //
assign net_data_in[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH*0] = data_in[(WIDTH* X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH*y_pos)+1];
assign data_out[(WIDTH*X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH* y_pos)+(WIDTH*x_pos)] = net_data_out[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH *0];
assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];

//north edge of router, x_pos = 1, y_pos = 0, i = 1
assign net_data_in[x_pos][y_pos][WIDTH*(1+1)-1:WIDTH*1] = net_data_out[x_pos ][y_pos+1][WIDTH*(3+1)-1:WIDTH*3];
assign net_stop_in[x_pos][y_pos][1] = net_stop_out[x_pos+1][y_pos][3];
assign net_stop_in [x_pos][y_pos][1] = net_stop_out[x_pos][y_pos+1][3];

//east edge of router, x_pos = 1, y_pos = 0, i = 2
assign net_data_in [x_pos][y_pos][`WIDTH*(2+1)-1:`WIDTH*2] = net_data_out[x_pos +1][y_pos][`WIDTH*(4+1)-1:`WIDTH*4];
assign net_stop_in [x_pos][y_pos][2] = net_stop_out[x_pos+1][y_pos][4];

//south edge of router, x_pos = 1, y_pos = 0, i = 3
assign net_data_in [x_pos][y_pos][`WIDTH*(3+1)-1:`WIDTH*3] = 0;
assign net_stop_in [x_pos][y_pos][3] = 1'b1;

//west edge of router, x_pos = 1, y_pos = 0, i = 4
assign net_data_in [x_pos][y_pos][`WIDTH*(4+1)-1:`WIDTH*4] = net_data_out[x_pos -1][y_pos][`WIDTH*(2+1)-1:`WIDTH*2];
assign net_stop_in [x_pos][y_pos][4] = net_stop_out[x_pos-1][y_pos][2];

end

end

if (x_pos == 2) begin
if (y_pos == 0) begin

//tile interface of router, x_pos = 2, y_pos = 0, i = 0
assign net_data_in[x_pos][y_pos][`WIDTH*(0+1)-1:`WIDTH*0] = data_in[(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*(x_pos+1))-1:(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*x_pos)];
assign data_out[(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*(x_pos+1))-1:(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*x_pos)] = net_data_out[x_pos][y_pos][`WIDTH*(0+1)-1:`WIDTH*0];

assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];

//north edge of router, x_pos = 2, y_pos = 0, i = 1
assign net_data_in [x_pos][y_pos][`WIDTH*(1+1)-1:`WIDTH*1] = net_data_out[x_pos +1][y_pos][`WIDTH*(3+1)-1:`WIDTH*3];
assign net_stop_in [x_pos][y_pos][1] = net_stop_out[x_pos+1][y_pos][3];

//east edge of router, x_pos = 2, y_pos = 0, i = 2
assign net_data_in [x_pos][y_pos][`WIDTH*(2+1)-1:`WIDTH*2] = net_data_out[x_pos +1][y_pos][`WIDTH*(4+1)-1:`WIDTH*4];
assign net_stop_in [x_pos][y_pos][2] = net_stop_out[x_pos+1][y_pos][4];

//south edge of router, x_pos = 2, y_pos = 0, i = 3
//modified for SPL connected with (1,1) north port(i=1)
assign net_data_in [x_pos][y_pos][`WIDTH*(3+1)-1:`WIDTH*3] = net_data_out [x_pos +1][y_pos][`WIDTH*(1+1)-1:`WIDTH*1];
assign net_stop_in [x_pos][y_pos][3] = net_stop_out[x_pos+1][y_pos][1];

// west edge of router, x_pos = 2, y_pos = 0, i = 4
assign net_data_in [x_pos][y_pos][`WIDTH*(4+1)-1:`WIDTH*4] = net_data_out[x_pos -1][y_pos][`WIDTH*(2+1)-1:`WIDTH*2];
assign net_stop_in [x_pos][y_pos][4] = net_stop_out[x_pos-1][y_pos][2];
end

end

if (x_pos == 3) begin
if (y_pos == 0) begin

//tile interface of router, x_pos = 3, y_pos = 0, i = 0
assign net_data_in[x_pos][y_pos][`WIDTH*(0+1)-1:`WIDTH*0] = data_in[(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*(x_pos+1))-1:(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*x_pos)];
assign data_out[(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*(x_pos+1))-1:(`WIDTH*X_WIDTH*y_pos)+(`WIDTH*x_pos)] = net_data_out[x_pos][y_pos][`WIDTH*(0+1)-1:`WIDTH*0];

assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];
//north edge of router, x_pos = 3, y_pos = 0, i = 1
assign net_data_in [x_pos][y_pos][WIDTH*(1+1)-1:WIDTH*1] = net_data_out[x_pos][y_pos+1][WIDTH*(3+1)-1:WIDTH*3];
assign net_stop_in [x_pos][y_pos][1] = net_stop_out[x_pos][y_pos+1][3];

//east edge of router, x_pos = 3, y_pos = 0, i = 2
assign net_data_in [x_pos][y_pos][WIDTH*(2+1)-1:WIDTH*2] = 0;
assign net_stop_in [x_pos][y_pos][2] = 1'b1;

//south edge of router, x_pos = 3, y_pos = 0, i = 3
assign net_data_in [x_pos][y_pos][WIDTH*(3+1)-1:WIDTH*3] = 0;
assign net_stop_in [x_pos][y_pos][3] = 1'b1;

//west edge of router, x_pos = 3, y_pos = 0, i = 4
assign net_data_in [x_pos][y_pos][WIDTH*(4+1)-1:WIDTH*4] = net_data_out[x_pos-1][y_pos][WIDTH*(2+1)-1:WIDTH*2];
assign net_stop_in [x_pos][y_pos][4] = net_stop_out[x_pos-1][y_pos][2];
end

if (x_pos == 0 )begin
if (y_pos == 1)begin
//tile interface of router, x_pos = 0, y_pos = 1, i = 0
assign net_data_in[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH*0] = data_in[(WIDTH*X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH*y_pos)+(WIDTH*x_pos)];
assign data_out[(WIDTH*X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH*y_pos)+(WIDTH*x_pos)] = net_data_out[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH*0];
assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];

//north edge of router, x_pos = 0, y_pos = 1, i = 1
assign net_data_in [x_pos][y_pos][WIDTH*(1+1)-1:WIDTH*1] = 0;
assign net_stop_in [x_pos][y_pos][1] = 1'b1;

//east edge of router, x_pos = 0, y_pos = 1, i = 2
assign net_data_in [x_pos][y_pos][WIDTH*(2+1)-1:WIDTH*2] = net_data_out[x_pos+1][y_pos][WIDTH*(4+1)-1:WIDTH*4];
assign net_stop_in [x_pos][y_pos][2] = net_stop_out[x_pos+1][y_pos][4];

//south edge of router, x_pos = 0, y_pos = 1, i = 3
assign net_data_in [x_pos][y_pos][WIDTH*(3+1)-1:WIDTH*3] = net_data_out[x_pos-1][y_pos][WIDTH*(1+1)-1:WIDTH*1];
assign net_stop_in [x_pos][y_pos][3] = net_stop_out[x_pos-1][y_pos][1];

//west edge of router, x_pos = 0, y_pos = 1, i = 4
assign net_data_in [x_pos][y_pos][WIDTH*(4+1)-1:WIDTH*4] = 0;
assign net_stop_in [x_pos][y_pos][4] = 1'b1;
end
end
if (x_pos == 1 )begin
if (y_pos == 1)begin
//tile interface of router, x_pos = 1, y_pos = 1, i = 0
assign net_data_in[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH*0] = data_in[(WIDTH*X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH*y_pos)+(WIDTH*x_pos)];
assign data_out[(WIDTH*X_WIDTH*y_pos)+(WIDTH*(x_pos+1))-1:(WIDTH*X_WIDTH*y_pos)+(WIDTH*x_pos)] = net_data_out[x_pos][y_pos][WIDTH*(0+1)-1:WIDTH*0];
assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];

//north edge of router, x_pos = 1, y_pos = 1, i = 1
assign net_data_in [x_pos][y_pos][WIDTH*(1+1)-1:WIDTH*1] = 0;
assign net_stop_in [x_pos][y_pos][1] = 1'b1;

//east edge of router, x_pos = 1, y_pos = 1, i = 2
assign net_data_in [x_pos][y_pos][WIDTH*(2+1)-1:WIDTH*2] = net_data_out[x_pos][y_pos+1][WIDTH*(3+1)-1:WIDTH*3];
assign net_stop_in [x_pos][y_pos][2] = net_stop_out[x_pos][y_pos+1][3];

//south edge of router, x_pos = 1, y_pos = 1, i = 3
assign net_data_in [x_pos][y_pos][WIDTH*(3+1)-1:WIDTH*3] = net_data_out[x_pos][y_pos-1][WIDTH*(1+1)-1:WIDTH*1];
assign net_stop_in [x_pos][y_pos][3] = net_stop_out[x_pos][y_pos-1][1];

//west edge of router, x_pos = 1, y_pos = 1, i = 4
assign net_data_in [x_pos][y_pos][WIDTH*(4+1)-1:WIDTH*4] = 0;
assign net_stop_in [x_pos][y_pos][4] = 1'b1;
end
end

//north edge of router, x_pos = 1, y_pos = 1, i = 1
//modified for SPL, it connects with (0,0) west port (i=4)
assign net_data_in [x_pos][y_pos]['WIDTH*(1+1)-1:’WIDTH*1] = net_data_out [0][0]['WIDTH*(3+1)-1:’WIDTH*3];
assign net_stop_in [x_pos][y_pos][1] = net_stop_out [0][0][4];
// assign net_data_in [x_pos][y_pos]['WIDTH*(1+1)-1:’WIDTH*1] = 0;
// assign net_stop_in [x_pos][y_pos][1] = 1'b1;

//east edge of router, x_pos = 1, y_pos = 1, i = 2
assign net_data_in [x_pos][y_pos]['WIDTH*(2+1)-1:’WIDTH*2] = net_data_out [x_pos +1][y_pos]['WIDTH*(4+1)-1:’WIDTH*4];
assign net_stop_in [x_pos][y_pos][2] = net_stop_out [x_pos+1][y_pos][4];
//south edge of router, x_pos = 1, y_pos = 1, i = 3
assign net_data_in [x_pos][y_pos]['WIDTH*(3+1)-1:’WIDTH*3] = net_data_out [x_pos -1][y_pos-1]['WIDTH*(1+1)-1:’WIDTH*1];
assign net_stop_in [x_pos][y_pos][3] = net_stop_out [x_pos-1][y_pos-1][1];
//west edge of router, x_pos = 1, y_pos = 1, i = 4
assign net_data_in [x_pos][y_pos]['WIDTH*(4+1)-1:’WIDTH*4] = net_data_out [x_pos -1][y_pos-1]['WIDTH*(2+1)-1:’WIDTH*2];
assign net_stop_in [x_pos][y_pos][4] = net_stop_out [x_pos-1][y_pos][2];
end

///(0,0)south port (3) -> (2,1) north port
if (x_pos == 2 )begin
if (y_pos == 1)begin
//tile interface of router, x_pos = 2, y_pos = 1, i = 0
assign net_data_in [x_pos][y_pos]['WIDTH*(0+1)-1:’WIDTH*0] = data_in[‘WIDTH*X_WIDTH*y_pos]+‘WIDTH*(x_pos+1)-1:’WIDTH*X_WIDTH*x_pos];
assign data_out[‘WIDTH*X_WIDTH*y_pos]+‘WIDTH*(x_pos+1)-1:’WIDTH*X_WIDTH*x_pos] = net_data_out [x_pos][y_pos]['WIDTH*(0+1)-1:’WIDTH*0];
assign net_stop_in [x_pos][y_pos][0] = stop_in[‘WIDTH*X_WIDTH*y_pos]+x_pos];
assign stop_out[‘WIDTH*X_WIDTH*y_pos]+x_pos] = net_stop_out [x_pos][y_pos][0];
//north edge of router, x_pos = 2, y_pos = 1, i = 1
//modified for SPL
assign net_data_in [x_pos][y_pos]['WIDTH*(1+1)-1:’WIDTH*1] = net_data_out [0][0]['WIDTH*(3+1)-1:’WIDTH*3];
assign net_stop_in [x_pos][y_pos][1] = net_stop_out [0][0][3];
// assign net_data_in [x_pos][y_pos]['WIDTH*(1+1)-1:’WIDTH*1] = 0;
// assign net_stop_in [x_pos][y_pos][1] = 1'b1;

//east edge of router, x_pos = 2, y_pos = 1, i = 2
assign net_data_in [x_pos][y_pos]['WIDTH*(2+1)-1:’WIDTH*2] = net_data_out [x_pos +1][y_pos]['WIDTH*(4+1)-1:’WIDTH*4];
assign net_stop_in [x_pos][y_pos][2] = net_stop_out [x_pos+1][y_pos][4];
//south edge of router, x_pos = 2, y_pos = 1, i = 3
assign net_data_in [x_pos][y_pos]['WIDTH*(3+1)-1:’WIDTH*3] = net_data_out [x_pos -1][y_pos-1]['WIDTH*(1+1)-1:’WIDTH*1];
assign net_stop_in [x_pos][y_pos][3] = net_stop_out [x_pos-1][y_pos-1][1];
//west edge of router, x_pos = 2, y_pos = 1, i = 4
assign net_data_in [x_pos][y_pos]['WIDTH*(4+1)-1:’WIDTH*4] = net_data_out [x_pos -1][y_pos-1]['WIDTH*(2+1)-1:’WIDTH*2];
assign net_stop_in [x_pos][y_pos][4] = net_stop_out [x_pos-1][y_pos][2];
end

if (x_pos == 3 )begin
if (y_pos == 1)begin
//tile interface of router, x_pos = 2, y_pos = 1, i = 0

assign net_data_in[x_pos][y_pos][`WIDTH*(0+1)-1:`WIDTH*0] = data_in[('WIDTH*X_WIDTH*y_pos)+('WIDTH*(x_pos+1))-1:('WIDTH*X_WIDTH*y_pos)+('WIDTH*x_pos)];
assign data_out[('WIDTH*X_WIDTH*y_pos)+('WIDTH*(x_pos+1))-1:('WIDTH*X_WIDTH*y_pos)+('WIDTH*x_pos)] = net_data_out[x_pos][y_pos][`WIDTH*(0+1)-1:`WIDTH*0];
assign net_stop_in[x_pos][y_pos][0] = stop_in[(X_WIDTH*y_pos)+x_pos];
assign stop_out[(X_WIDTH*y_pos)+x_pos] = net_stop_out[x_pos][y_pos][0];

//north edge of router, x_pos = 2, y_pos = 1, i = 1
assign net_data_in[x_pos][y_pos][`WIDTH*(1+1)-1:`WIDTH*1] = 0;
assign net_stop_in[x_pos][y_pos][1] = 1'b1;

//east edge of router, x_pos = 2, y_pos = 1, i = 2
assign net_data_in[x_pos][y_pos][`WIDTH*(2+1)-1:`WIDTH*2] = 0;
assign net_stop_in[x_pos][y_pos][2] = 1'b1;

//south edge of router, x_pos = 2, y_pos = 1, i = 3
assign net_data_in[x_pos][y_pos][`WIDTH*(3+1)-1:`WIDTH*3] = net_data_out[x_pos][y_pos-1][`WIDTH*(1+1)-1:`WIDTH*1];
assign net_stop_in[x_pos][y_pos][3] = net_stop_out[x_pos][y_pos-1][1];

//west edge of router, x_pos = 2, y_pos = 1, i = 4
assign net_data_in[x_pos][y_pos][`WIDTH*(4+1)-1:`WIDTH*4] = net_data_out[x_pos-1][y_pos][`WIDTH*(2+1)-1:`WIDTH*2];
assign net_stop_in[x_pos][y_pos][4] = net_stop_out[x_pos-1][y_pos][2];
end
end // block: x_loop
end // block: y_loop
generate
endgenerate
endmodule // network