Flow Control

How we allocate a resource
- Allocate buffers
- Allocate output ports

Store and Forward

Serial in
Serial out

route pipeline: 00, 01, 11
unic traversal: 00, 11

Assume 4 ports/packet

Store packet completely in input queue before forwarding
VIRTUAL CUT THROUGH

PART 1

S10 A1 A2 L0 L1 R0 A1 A2 D
S20 A1 A2 L0 L1 R0 A1 A2 D
S30 A1 A2 L0 L1 R0 A1 A2 D
S40 A1 A2 L0 L1 R0 A1 A2 D

STILL ARBITRATE FOR FULL PACKET'S WORTH OF BUFFERING, BUT DO NOT WAIT FOR WHOLE PACKET BEFORE STARTING TO SEND PACKETS

IN THIS COURSE, ASSUME WE ALWAYS USE VIRTUAL CUT THROUGH

CHANNEL BUFFER FLOW CONTROL PROBLEM

\[ A \rightarrow Q_s \rightarrow R \rightarrow Q_r \rightarrow D \]

* ASSUME SINGLE PACKET PACKETS

SENDER \hspace{1cm} CHANNEL \hspace{1cm} RECEIVER

\[ t_c = 2 \]

HOW DOES SENDER KNOW WHEN IT CAN SEND A PACKET TO RECEIVER? HOW DOES IT KNOW THERE IS ROOM IN THE RECEIVER'S QUEUE?

WE WILL EXAMINE TWO SCHEMES:

1. ON/OFF FLOW CONTROL (IE. USING STALL SIGNAL)
2. CREDIT-BASED FLOW CONTROL
On/off w/ Combinational Stall Signal

Set stall = 1 when recv cannot recv packet from queue

Stall D on this cycle
Unstall D on this cycle

Able to get full throughput except for 4 cycle stall in D stage

How deep do queues need to be to get full throughput?

1 element!

Not really queues - more like pipeline registers

What if t_e = 3? Still only needs one element of timing, but stall signal becomes more critical.
On/off + Partial Comb Stall Signal

Can only stall A, B cannot stall C

Maybe cannot stall R register because multiple bits in flight on one at once (wave fronting) or overhead for stalling R to high

Pict

0  A B C D
1  A B C D
2  A B C D
3  A B C D
4  A B C D
5  A B C D
6  A B C D
7  A B C D

Cannot stop Pict 3 in stage C!

Qs  R  Qn

Cannot stop Pict 3!

How deep is Qn to maintain full through put?

2 Elements

What if Qn is only one element?
WHAT IF $t_c = 3$?

Qs  $\rightarrow$  Q1  $\rightarrow$  Q2

0  $\rightarrow$  1  $\rightarrow$  2

3  $\rightarrow$  4  $\rightarrow$  5

6  $\rightarrow$  7

CANNOT STOP PACKETS 3 or 4

COMBINATIONAL STALL SIGNAL

Needs 3 elements of buffering.

But is a combinational stall signal realistic if $t_c = 3$?
ON/OFF w/ PIPELINED STALL SIGNAL

Stall signal easily on critical path pipeline it?

PILT

<table>
<thead>
<tr>
<th>0</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>J</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>D</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>7</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Stall signal takes 1 cycle to stall A, B, C stages
Queue acts as fixed buffering

How deep is Qn to maintain full throughput?

2 Elements

What if Qn is only one element?
We can also explicitly show the pipeline region for the stall signal.
ON / OFF / PARTIAL PIPELINES STALL SIGNAL

Can only stall A, B
cannot stall C
and stall signal
is pipelined

Cannot stop Pkt 3, 4
these packets "sick"
into receiver queue

How deep is Q_n to
maintain full throughput?
3 elements

What if Q_n is only
one element?
What if $t_c = 3$?

- Needs 6 elements of buffering
- In general, needs $2 \times t_c$ elements of buffering
- Can need a few extra elements of buffering depending on scheduling
- Buffers are poorly utilized. Have to send stall signal as soon as D stalls even if saw nothing to send!
APPLYING FLOW CONTROL CONCEPTS TO PROCESSORS

2 CYCLE, UNPIPELINED I $

op A
  fetch
  op B
  fetch
  op C
  fetch
  op D
  fetch

STALL DUE TO DATA HAZARD

ASSUMES YOU CAN STALL INSTRUCTION CACHE
NOT POSSIBLE WITH VALID REQ AND VAL IRP

2 CYCLE, PIPELINED I $

op A
  fetch
  op B
  fetch
  op C
  fetch
  op D
  fetch

WRITE SKID BUFFER!

* CONSIDER MOVING TO A DECOUPLED FRONT END OF FETCH QUEUE
Flow Control for Output Ports

Each requester sets corresponding request signal high if need served resource. Arbitrator sets a single grant signal high indicating which requester won arbitration.

Grant + Hold arbitrator allows requester to "hold on" to served resource until finished.

**Fixed Priority Arbitrator**

![Diagram of fixed priority arbitrator with priority levels and "kill" signal]

- 0 was highest priority
- 3 was lowest priority

**Round Robin Arbitrator**

![Diagram of round robin arbitrator with priority bits and grant decisions]

Winner is lowest priority on next cycle.
FAIRNESS

- Weak Fairness: every request eventually served
- Strong Fairness: requests served equally often

LOCAL VS. GLOBAL FAIRNESS

![Diagram of traffic flow]

1 unit of traffic

Round-Robin Arbiters

What percentage of the output bandwidth comes from each input terminal?

0: 0.125
1: 0.125
2: 0.25
3: 0.5

No global strong fairness, even though each round robin arbitrator has local strong fairness.
Router Block Diagram

**STANDARD VIRTUAL CUT THROUGH 3-STAGE ROUTER**

**ROUTE COMPUTATION (RC)**
- Simple combinational logic
- Duplicate per input port to avoid structural hazard

**SWITCH ALLOCATION (SA)**
- Two 2-input arbiters, one per output port
- Grant + hold, hold after head quit until tail quit

**SWITCH TRAVERSAL (ST)**
- Cross the crossbar and write output buffer

**Plate 1**

<table>
<thead>
<tr>
<th>M</th>
<th>0 LI RCSA ST (0 LI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3</td>
<td>0 LI &amp; SA ST (0 LI)</td>
</tr>
<tr>
<td>D</td>
<td>0 LI &amp; SA ST (0 LI)</td>
</tr>
<tr>
<td>T</td>
<td>0 LI &amp; SA ST (0 LI)</td>
</tr>
</tbody>
</table>

- Only header quit does route computation
- Body/tail quits cannot bypass header quit
- Wait in input queue
STALLS DUE TO CONTENTION

ROUTE LOOK AHEAD

STANDARD PIPELINE OPTIMIZATION OF doing TWO STEPS IN PARALLEL TO REDUCE TRANSACTION LAG/DEADLINE

PREDICT ROUTE FOR NEXT ROUTER OVLAP NEXT RC WITH CURRENT SA