CHAPTER 1 CSL Pipeline

TABLE 1.1 Chapter Overview

1.1 Definitions
1.2 CSL Pipeline Overview
1.3 CSL Pipeline Concepts
1.4 CSL Pipeline Examples

1.1 Definitions

1.2 CSL Pipeline Overview

What is a pipeline ?

A digital pipeline divides a “job“ into many small jobs which are executed in parallel. A pipestage consists of combinational logic which processes the information, sequential devices to store the results processed by the combinational logic and control, valid and clock signals which are used to control propagation of the information from one pipestage to another. A pipeline is constructed from pipestages. Information is passed from one pipestage to another pipestage.

How are pipestages constructed ?

A digital pipeline is constructed from combinational and sequential logic (elements) and sequential elements. The sequential elements are typically controlled by a clock, a reset, and an enable. Each pipe stage consists of a combinational logic cone feeding a set of sequential elements. The outputs of the previous stages sequential elements drive the inputs of the next stages of combinational logic. Data is passed from one pipestage to the next where it is modified or passed through to the next pipestage. Data can be *forwarded* to pipestages past the next pipestage. Automatic detection of data leaks and data duplication in pipelines is a an important feature in a set or design checking tools set.

Pipestages are used to increase the operating frequency of digital devices by reducing the critical path between flip-flops. If the pipeline type has a valid bit then each pipestage must have a register/flip-flop that has a valid input and a valid output.

1.3 CSL Pipeline Concepts

This section describes what a pipeline is and how to build different types of pipelines using the CSL pipeline specifications.

Pipelining is a technique used in the making of fast CPUs. It allows for multiple instructions to be overlapped in execution. A pipeline is a series of connected data processing units such that the output of one unit drives the input of the next one. This can be illustrated with an analogy to an assembly line. An automobile assembly line for instance, consists of a certain number of steps or stages connected in series. Each step has a contribution to the building of a car (eg: install the engine, mount the seats, etc) and all operate in parallel with each other yet on a different car. In the pipeline different parts of different instructions are completed by different steps of the pipeline. These steps are called pipe stages or pipe segments[1].

1.3.1 Valid bit generation and usage

We will discuss now how a valid bit is generated and what it is used for. A valid bit indicates that the associated data in the pipe stage is valid (combinational logic when an event occurs). A valid bit is set to true when the event occurs. Otherwise the valid bit is set to false. The data and the valid bit are sent down the pipeline together every cycle. The valid bit can be true or false. Valid data announces its arrival when the valid bit is true. When a unit receives a data transaction and the valid bit is true then the unit will either process the data if the transaction is meant for the unit or pass the data through. Additional control signals and transaction/packet identification can be sent in parallel with the data and the valid bit to further qualify the data for down stream units. They can look at the valid bit and if it is true, they can look then at pipelined control, transaction id, and other signals to determine if the unit should process the transaction and where the transaction should be sent. For example, a transaction can have a destination tag which indicates where the packet should be sent in a pipeline that has a branch and two or more destinations.

Some pipelines are designed only to propagate the data when the valid bit is true, others are free-flowing and they propagate the data when the clock signal is enabled and others may be controlled by a downstream stall signal.
For example, when an upstream ALU unit that is computing a result, it needs to know when the received data is valid. An instruction decoder decodes instructions and sends pipelined control signals to the down stream units. One set of control signals is sent to a register file. The register file control signals can indicate that a read transaction should be executed by the register file. This read transaction means that valid data will appear at the output of the register file n cycles later, n being the latency through the register file. The register file can then generate a valid bit to associate with the read data which is generated by the register file every cycle. Only read data that is the result of a read transaction is valid. Valid read transactions turn on the valid bit. The equation for a valid bit generated by the register file is:

```
rf_valid = rd_en;
```

A register file can generate a valid bit which is associated with each read transaction.

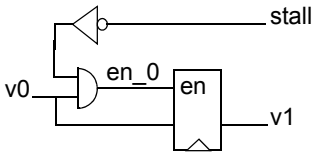
1. From Computer Organization and Design by Hennessy and Patterson

The register file sends the read data and the valid bit to an ALU every cycle.

1.3.2 Stalling upstream pipelines

Pipelines which are enabled by a stall from a down stream block and which have a valid bit need to qualify the valid bit with the inverse of stall to avoid sending duplicates to the down stream pipeline. The logic which generates the enable for the next pipestage uses the previous pipestages valid bit and the stall signal for the pipeline to generate the enable signal for the next pipestage. The valid bit must be qualified with the stall signal.

FIGURE 1.1 valid bit qualification



1.3.3 Pipeline types and lists of different pipelines

Pipelines can be:

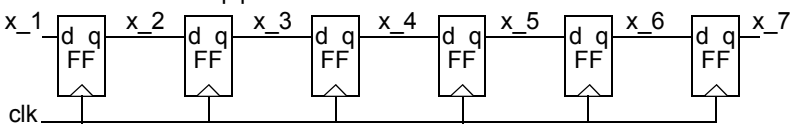
- Standard Pipeline
- Pipeline with valid bit
- Pipeline with stall and qualified valid bit for duplicate prevention

!!add the rest of the pipeline types here

1.3.3.1 Standard Pipeline

In a standard pipeline when the clock is enabled data flows through the pipeline.

FIGURE 1.2 Standard pipeline



Pipeline transactions with no stall or valid bit. The pipeline is free flowing. We show the transaction through the pipeline below(data flowing through the pipeline):

```
(turn into a table) and add all the pipestages in the figure (0 and 4)
xxxxx(pipe stage numbers)
```

TABLE 1.2

time	data 01234
t0	10000
t1	01000
t2	00100
t3	10010
t4	01001
t5	00100
t6	00010

1.3.3.2 Pipeline with valid bit

Data flows through the pipeline when the clock is enabled. The valid bit is used to indicate which pipelines contain valid data.

FIGURE 1.3 Pipeline with valid bit

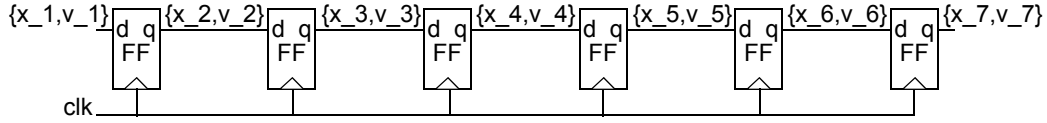


TABLE 1.3

time	pipestage 01234
t0	10000
t1	01000
t2	00100
t3	10010
t4	00100
t5	00010

1.3.3.3 Pipeline with stall and qualified valid bit for duplicate prevention

A valid bit in a pipeline is used to indicate that data is valid. The valid bit should not be duplicated. If the valid bit is used in the next pipe stage it needs to be qualified with the stall signal. The valid is qualified with the stall signal to turn off the valid bit when the stall is off. If the valid bit is not qualified

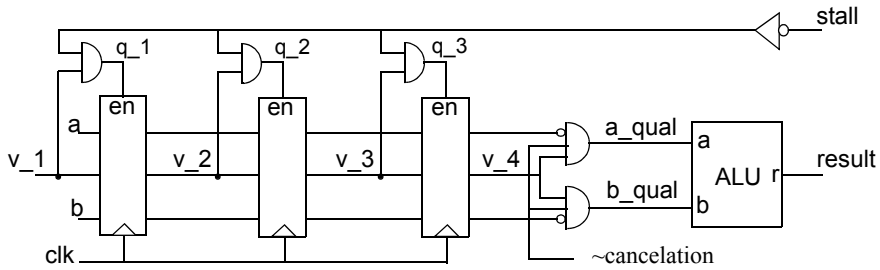
then when the valid bit is 1 and the stall is off and the down stream logic is not controlled by the stall signal as in Figure 1.4 then the transaction will be duplicated each cycle that the stall signal is on. Down stream logic uses the valid bit in stage 4

Pipeline operation

- Data flows when the inverted value of the stall signal is false. The flip flops are enabled.

The pipeline controls all of the parallel flip flops. This can be an internally or externally generated signal.

FIGURE 1.4 Pipeline bus with stall and qualified valid bit



The combinational logic in the alu is shut off when the data is invalid.

The valid bit can be used to zero out data that feeds combinational logic. In the example in Figure 1.4 the a and b signals are qualified with the valid bit to zero them out whenever the valid bit is zero. Data which is invalid does not cause the circuit to be activated.

The equation for the valid bit in pipestage n is:

v = v from pipestage n-1 register output and not stall from pipestage n+1and not global cancel where global cancel is a signal that can cancel all stall bits.

For example if a conditional branch is taken then if there are valid instructions in the previous pipestages they need to be cancelled.

Pipeline transactions with stall

TABLE 1.4

time	pipestage 01234	stall
t0	10000	0
t1	01000	0
t2	00100	0
t3	00100	1
t4	00100	1

t5	00010	0
t7	00001	1

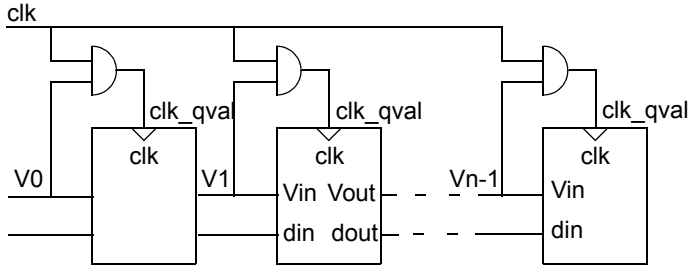
VERILOG CODE

```
always@(posedge clk) begin
   if (en)
      v1 <= v0;
      a1 <= a0;
      b1 <= b0;
 end
assign r1 = v1&a1 * v1&b1; // the logic in the multiplier is "acti-
vated" only when the data is valid
```

1.3.4 Micro pipelining

A regular pipeline has a single clock. Each pipestage in a regular pipeline is clocked at the same time. In micropipeline pipestages are turned on only when there is a valid transaction for that pipestage. The valid transaction controls the clock enable for that pipestage. Each subsequent pipestage can turn on the next pipestage. This is used for power management reasons.

FIGURE 1.5 Micro pipelining



Each pipeline turns on when it's needed. More than on cycles valid bit needs to be used to turn on the clock otherwise there will be a race between data and clock.

1.3.5 Potential problems related to the use of valid bits and stall logic

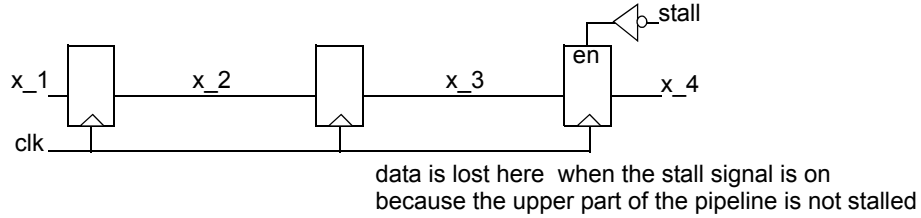
Pipelines which use valid bits and stall logic need to be designed correctly to avoid data loss and data duplication.

1.3.5.1 Register Leaks due to stall conditions

A register leak is defined as a condition where valid data in a pipeline is lost.. Data loss can occur for

when the lower part of the pipeline is stalled and the upper part of the pipeline is not stalled. An upstream register writes data on a downstream register which is disabled. Data flows in the upper part of the pipeline and it is not written into the register that is not enabled. The x_3 value is lost whenthe stall signal is on.

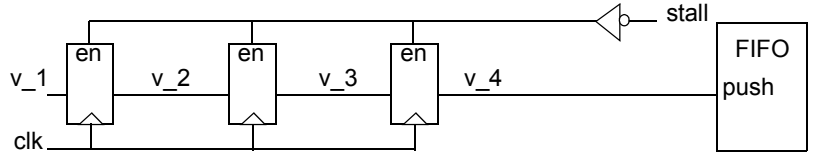
FIGURE 1.6 Register leak



1.3.5.2 Transaction duplication due to missing stall qualification of the valid signal

The example in Figure 1.7 on page 7 shows a case where the FIFO is written every cycle that v_4 is 1. The valid bit controls the push operation. If the v_4 valid bit is not qualified with the stall then during a stall when v_4 is 1 the FIFO will be written every cycle and the data will be duplicated.

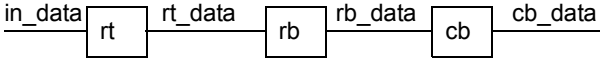
FIGURE 1.7 Duplicate transactions



1.3.6 Pipelines and units

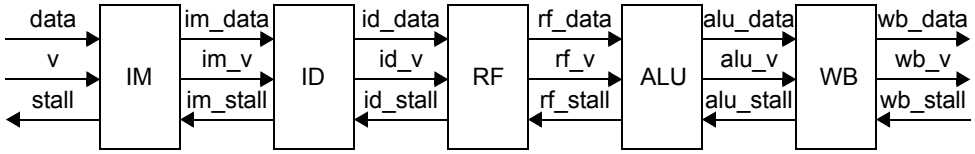
A pipeline is used to send data through a series of processing steps much like an automobile is assembled using an assembly line. These processing steps can be located in different units in a chip design. Each unit can modify or pass through the data in each transaction. The unit is controlled either by a hard wired controller or a programmable controller or by pipelined control bits.
Stall signals are typically sent from the downstream unit to the upstream unit. The stall input is then used to control all flip flops in the unit in parallel.

FIGURE 1.8 Simple pipeline through several units



A pipeline contains control, data, and valid bits. A pipeline can span many units. For example a simple pipeline may span the following units.

FIGURE 1.9 An example pipeline spanning 5 units




```
alu_stall = wb_stall;
rf_stall = alu_stall;
id_stall = rf_stall;
im_stall =id_stall;
```

1.3.7 Units with two or more stall inputs

Units with two or more stall inputs can have register leaks or transaction duplication in the pipeline due to incorrect stall logic implementations internal to the unit. The stall signals need to be OR'ed together and need to be connected to the correct pipestages.

FIGURE 1.10

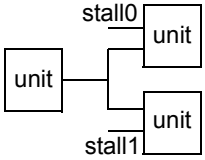
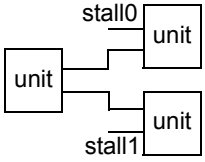


FIGURE 1.11



1.3.8 Pipelines with sequential feedback loops with branches and stalls

Pipelines which contain sequential feedback loops with branches thats are controlled by one or more stall signals also need to be designed carefully. One solution is to unroll the hardware sequential loop and to use branch and merge pipelining. The loop can only be replaced by a branching solution when the loop is not a SCC (Strongly connected component).

FIGURE 1.12 Sequential feedback loop

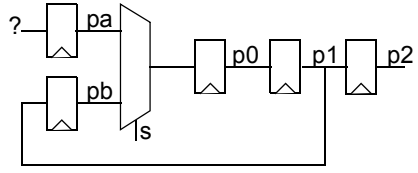
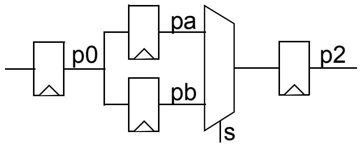


FIGURE 1.13 Branching solution



1.3.9 Pipeline Generator

Creating pipelines is useful when there are a number of signals which are sent or bundled together across different pipe stages.
Another case in which automatically generating pipestages is useful is when complex branching and merging occurrs in the pipelines. In some cases the design of the pipeline branches and merges is as or more complicated than the design of the logic in the pipeline stages. Complex branching and merging can lead to register leaks and duplicate data in the pipelines. Automatically generating the pipline branches can prevent register leaks and duplicates.

FIGURE 1.14 Data valid stall pipeline

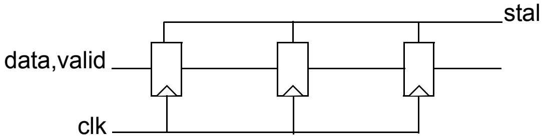
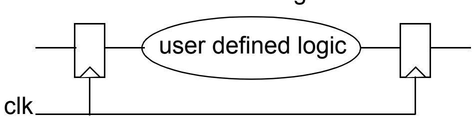


Figure 1.15 on page 9 shows 3 different Fip Flops which are not connected. This means that we have generated these different signals on the inputs and the outputs of the Flip Flops, which are actually 6 different sets of signals. The user inserts the combinational logic that would go between Flip Flops.

FIGURE 1.15 User defined logic is inserted into the pipelines



User defined logic can be inserted in pipeline stages.
//

1.3.9.1 Auto Pipeline Generator

FIGURE 1.16

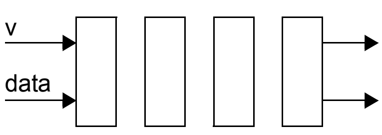


FIGURE 1.17 After auto_pipe_gen there are created connection between bloks, no stall

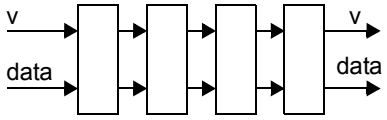
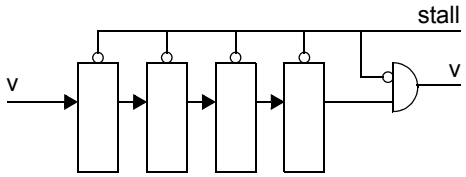


FIGURE 1.18 Auto_pipe_gen with stall



1.3.10 CSL Pipestage generator

The CSL pipestage generator assists the designer in building pipelines.

- Pipelines are automatically generated using the CSL pipeline commands.
- Logic is inserted into CSL pipelines using code inlining.
- Stalls are used to control the data flow down a pipeline.

Enables are used to move a state through the pipelines

1.3.11 Pipestage operations

The uspstream logic is the logic which writes to a pipestage. The downstream logic is the logic which is written to. A stall logic is required to prevent upstream logic from overwritting down stream state when there is a mismatch in processing speeds between the upstream and the downstream logic or when another down stream unit is stalled.

1.3.12 Pipeline latency match generator

1.3.13 Pipeline latency checker

The cslc is told what are the input clocks signal names. The cslc follows the clock trees to the input pins of the state elements, the connectivity between the ins and outs of each element is traced starting with the beginning of each pipeline which is traced from ther inputs to the chip.
The pipeline will measure the number (**n**) of pipestages between two specified signals and generate **n** pipe stages. The import signal to the **n** pipestages is specified. The clock signal and the enable signal for each stage are extracted from the original set of pipestages.

1.3.14 Pipeline branching

There are several cases where multiplexers are used to control the flow of valid bits in pipelines:

1. Mux inputs same pipestage
2. Data forwarding muxes with different inputs
3. Valids from different clock domains
4. Fork and Join

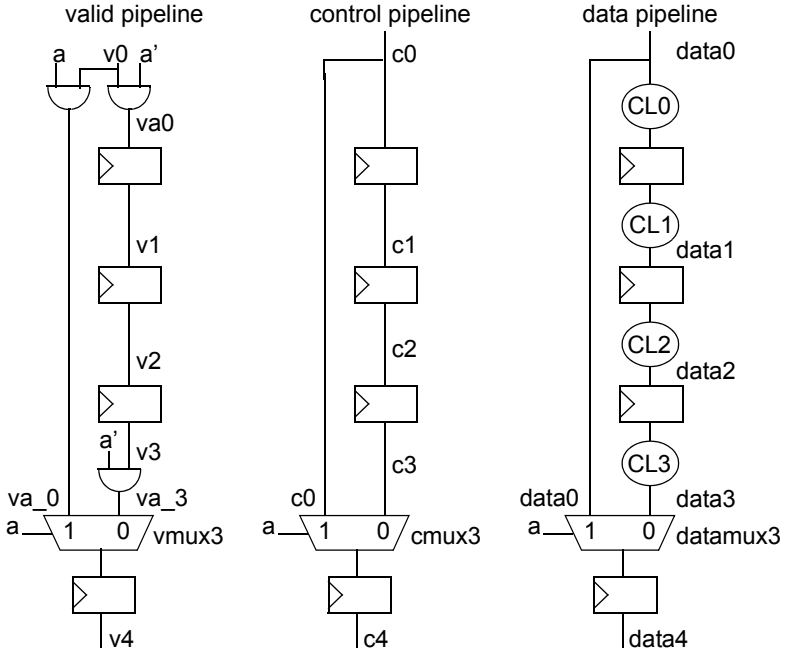
Valid bit pipelines should be constructed to match datapath pipelines. The problem here is to build a valid pipeline with the same mux structure to match the datapath mux structure.

1.3.15 Pipeline branching and data forwarding

Data forwarding is used to bypass logic, collapsing bubbles in pipelines, and to match variable pipeline delays in other pipelines.

For example, in Figure 1.19 on page 12 there is a valid pipeline, a control pipeline, and a data processing pipeline. All pipelines have the same branches controlled by the same mux selects. Only the valid bit is qualified with the mux select value for the branch. The three pipelines are coded separately so that the designer can see the valid, the control, and the data bits flowing through the pipeline.

FIGURE 1.19 Building parallel pipelines with matching bypass logic



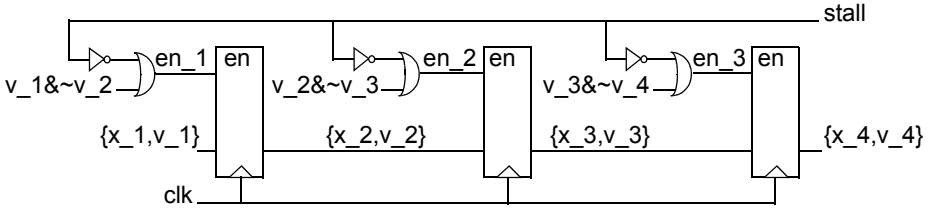
1.3.15.1 Pipeline with bubble collapsing

There are three different operations.

- Data flows when the enable signal is on.
- Data flows when the stall signal is off and the previous pipestage valid bit is on and the current pipestage valid bit is off.

This pipestages in the pipeline shown in Figure 1.20 flow if the stall signal is off and the data in the previous pipestage is valid or when the valid bit associated with a pipeline is 0 and the previous pipestage bit value is a 1. Note that logic cones could be present between the flip flops but they are not shown.

FIGURE 1.20 Example of a bubble collapsing and previous stage forwarding logic pipeline



Pipeline with bubble collapsing

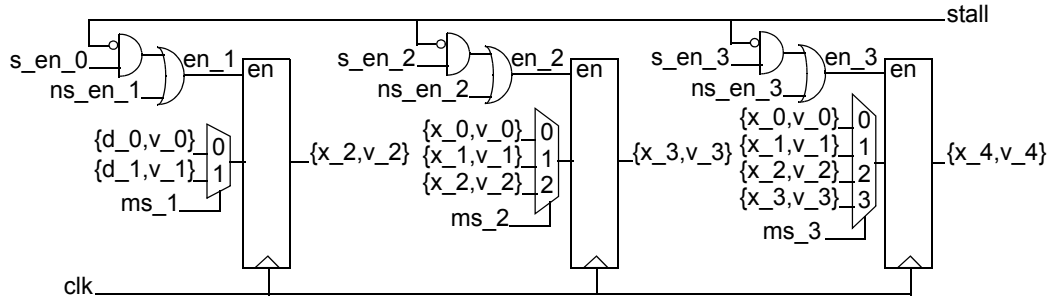
TABLE 1.5

Data	Stall
10000	0
01000	0
00100	0
10010	0
01001	1
00101	1
10100	0
01010	1
11011	1
01111	1
11111	1

1.3.15.2 Pipeline with bypass logic for buffering pipelined data

The individual pipestages in the pipeline shown in Figure 1.21 flow when the stall signal is off and the stall enable is true or the not stall enable is true. Note that the inputs to each pipestage are the outpus of all the previous pipestages. The output of a pipestage can jump over n pipestages if each of the n pipestage's valid bit is 0.
Each mux in each subsequent pipe stage can select the valid bit from ANY previous pipe stage based on the values of the valid bits in the current and previous pipe stages. It is a bypass mechanis

FIGURE 1.21 Example of pipeline bubble collapsing and previous stage forwarding logic



s_en = no stall enable
s_en_1 =
s_en_2 =
s_en_3 =

ns_en = no stall enable
ns_en_1 =
ns_en_2 =
ns_en_3 =

ms = mux select
ms_1 =
ms_2 =
ms_3 =

Pipeline with fowarding

TABLE 1.6

Valid	Stall
10000	0
00001	0
00000	0
10000	0
10001	1
00011	1
00001	0
00000	1
11011	1
01111	1
11111	1

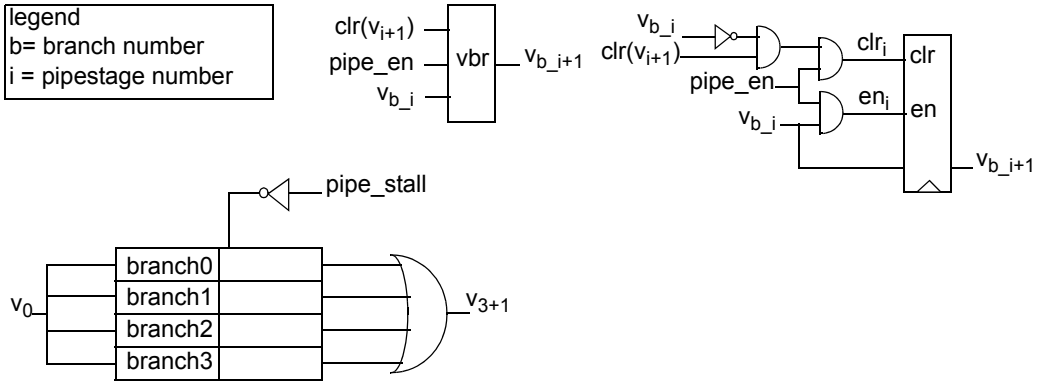
1.3.16 Pipeline branch and merge semaphores

A single operation launches multiple operations in different branches. Each branch can have its own latency. If the branches have different latencies then an upstream stall will be generated to prevent the next operation flowing down the pipe from entering the semaphore branch and merge until all operations in flight in the multiple branches are guarenteed to reach the semaphore and join a unit before a valid bit from the next operation. The branch operations can complete in non-deterministic times. When all operations are completed (all valid bit registers are 1), a single valid bit is generated for the next stage at the bottom of the pipeline.

The valid bit registers in i+1 hold the valid bit values until all single valid bits in pipeline branches have reached the output of each valid bit register. Note that each register is effectively a SR flip flop.

v_{i+1} is the AND of all valid bits from the merged pipelines. v_{i+1} is the clear signal for all valid bit registers. The clear signal wil only clear the valid bit registers if v_{b_i} is false.

FIGURE 1.22 Pipeline semaphore branch and merge example !!move to the end of concepts



1.3.17 Duplication of data in a pipeline with branches

Transactions can be duplicated in the pipeline branches because the valid bits which are written to one or more pipeline branches are not qualified with the pipeline branch enable.

<ADD>
Data eye

early/late arrival time
early/late skew
rise time/fall time
50% duty cycle

When working with pipelines we need a Valid Bit Pipeline to match datapath
The problem here is to build a valid pipeline with the same mux
structure to match the datapath mux structure.

There are several case
1. mux inputs same pipestage
2. data forwarding muxes with different inputs
3. valids from different clock domains
4. Fork and Join
A single operation generates multiple operations which all must complete before the operation is complete. The branch operations can complete in non-deterministic times. When all operations are completed then the single valid bit is generated for the next stage.

Register Leaks due to stall conditions
register leak due to two stall signals used in a pipeline.
The lower part of the pipeline is stalled and the upper part of the pipeline
is not stalled. Data flows in the upper part of the pipeline and is not
written into the register that is not enabled.

Extra transactions generated in the pipe because the valid bit not qualified with enable.

```
always @(posedge clk) begin
   valid0 <= valid;
   valid1 <= valid0;
end

// valid qualified with the enable from the previous stage

always @(posedge clk) begin
   valid0 <= valid & en;
   valid1 <= valid0 & en0;
end
```

</ADD>

1.3.17.1 Checking for required signals in a pipestage

TABLE 1.7

c	clock
v	valid
r	reset
e	enable
s	stall

TABLE 1.8

	c	r	e	s	v
input bit set	t	t	t	t	t
output bit set	x	x	x	x	t

Clock, valid, reset, enable and stall must be either t either x.
x = not applicable
t = true
The pin on the pipeline must be set in order to set the bits in the table.

Bit set for each pipestage - based on the methods which are called for each pipestage, you can set a bit.

Pipestage validity - check the pipestages using the bit set.
Sets the bits in the table if the pin on the pipeline is set.
x = not applicable
f = false
t = true

TABLE 1.9

	c	r	e	s	v
input bit set	t	f	t	f	t
output bit set	x	x	x	x	t

The two tables together:
bitset result = pipestage.bitset() & pipeline.bitset();
The truth table - pp:

TABLE 1.10

l	s	result
x	x	1
x	f	0
x	t	0
f	x	0
t	x	0
f	f	1
f	t	0
t	f	0
t	t	1

Release 2.0 Pipeline future:

- power efficient pipeline designs;
- Clock control on a per pipe stage basis can be controlled using the previous
- valid bits from the previous 2-3 pipe stages.

1.4 CSL Pipeline Examples

1.4.1 Register leak example

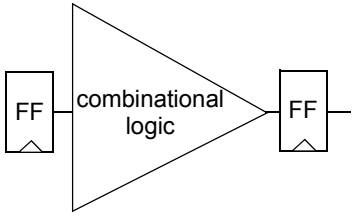
Two or more stall signals used in a pipeline

```
always @(posedge clk) begin
   if (stall0) valid0 <= valid;
   if (stall1)  valid1 <= valid0;
end
```

1.4.2 Pipeline timing issues

How logic delays determine the clock frequency.
The longest delay through a combinational logic cone in the pipeline determines the minimum clock period and the operating frequency of the pipeline.

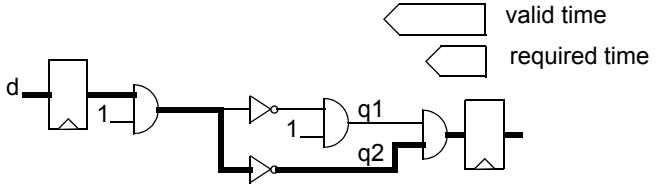
FIGURE 1.23 combinational logic cone



minimum delay between the flip flops to meet the setup and hold time
+ clock skew + clock jitter + clock to q delay

1.4.2.1 Shortest path determines the setup time

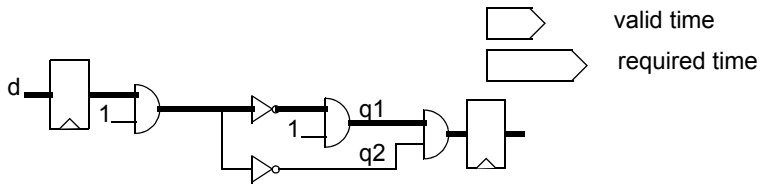
FIGURE 1.24 Shortest path example



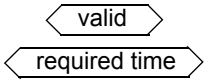
Logic gates induce delays in the propagation of data values through a circuit. In the example in Figure 1.24 the highlighted path which corresponds to q2 propagates faster than the path corresponding to q1 because q1 path has an extra AND gate.

1.4.2.2 Longest path determines the hold time

FIGURE 1.25 Longest path example



The extra AND gate before q1 in Figure 1.25 does not change the value of the input signal (since the other input is 1) but induces a delay. Thus q1 will take longer to be produced compared to q2



1.4.2.3 Clock skew

Clock skew is the maxiumum delay between the two clock signal end points. For example each flip flop is connected to a clock signal. The wire delays and the gate delays for the clock network cause different delays.

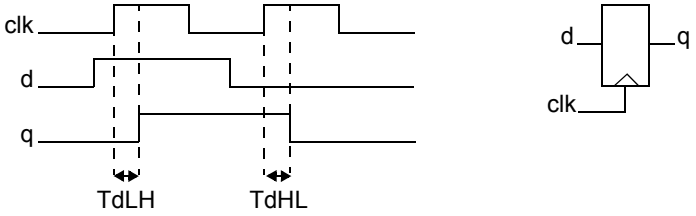
Problems with clock lines

- rise and fall times have to be equal
- duty cycles have to be equal
- clock lines cannot have glitches

1.4.2.4 Clock to Q delay

For a synchronous device (eg. a D flip-flop) the delay from clock to q determines when the output changes. figure 1.28 shows in a waveform example the delay that occurs from the time the clock changes to the time this change is reflected on the output.

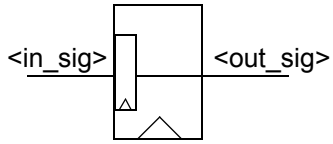
FIGURE 1.26



1.4.2.5 CSL pipelines

- pipeline create (<in_sig cout_sig><number of stages>) add N pipeline stages

FIGURE 1.27



- match latency between two signals
- pipeline create (<in_sig><out_sig><number of stages>);
- latency (<in_sig><out_sig>)
- the number of pipestages between stages is measured and a number is returned

Connections between modules

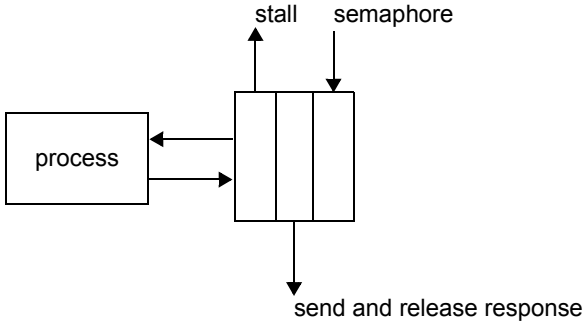
1.4.3 Hardware semaphore

1.4.4 Hardware semaphore pipelines

A semaphore is used to stop an operator.Is sent down a pipeline to the different units. Units read the packet containing the semaphore. If the semaphore applies to their unit then the send pipeline is stalled , the action correspoding to the semaphore is processed, the semaphore is released, and the valid transactions are sent to the downstream pipeline.

The semaphore transaction can be a bit in a packet, a word that is sent down a pipeline or a write to an address that sets a bit. A semaphore in a pipeline stops all units before the current unit processes the current transactions related to the semaphore and then releases the semaphore. Semaphorers are sent from one unit to another. The process is completed before sending the semaphore and the processed data to the next unit.

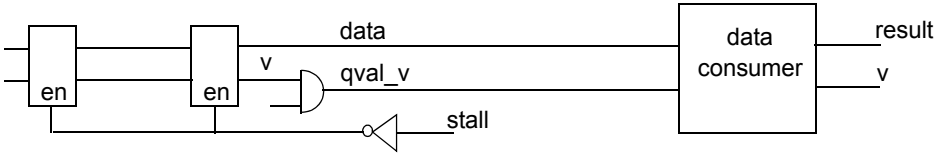
FIGURE 1.28 Hardware semaphore operation

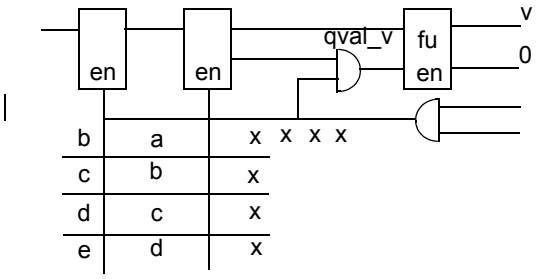


1.4.4.1 Pipelines & valid bits

Current ASIC designs can contain hundreds of pipeline stages. Each pipe stage can contain data and a valid bit. The valid bit is set when data is valid. Some designs use the valid bit to indicate to processing units to “process the data“. In other words data “announces“ its arrival to the consumer. “Out of band“ control signals can control bypass muxes which can be used to bypass processing units which may contain one or more pipestages. In effect the data is forwarded by “jumping” over a certain number of pipestages. This implies that there can be multiple paths through the data and valid pipelines. For power reasons datapath branches should have a qualifier based on the down stream mux selects. To prevent a single valid bit from propagating to multiple pipestages the valid bit in each branch needs to be qualified without its downstream mux select equation. A branch is taken if the set of the muxes selects the branch.The branch select equation minterm which select that branch is true.

FIGURE 1.29 Valid bit branch/merge





In order to stall multiple pipe stages a single signal or multiple signals or'ed together are connected to the registers in each each pipe stage. There are many issues associated with pipe lines and valid bits.
Pipeline split/merge-qualifying valid bits at the split point.
Selecting from different pipe stages at the merge point.
Stalling pipe stages at the merge point.
Register overwrite when inverted stall not used as enable.
Cancellation logic for multiple pipe stages-turning off the valid bit permanently-canceling the operation.Register copying-not using the inverted stall bit to qualify the valid bit.

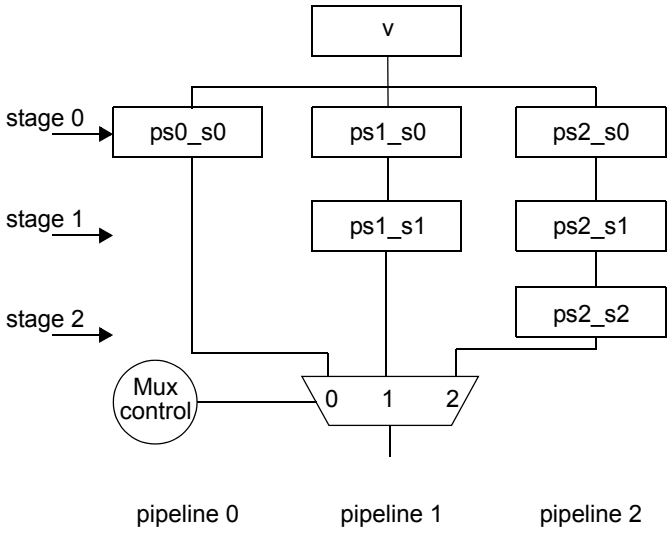
Yet they are fundamental to designing pipelines. In particular the valid bit is a communication mechanism between different units (read designers). Data announces its arrival to a new pipe stage or unit using a valid bit. When the valid bit is asserted the data at the out put of the pipe stage is valid and should be consumed by the next pipe stage. This means that if the pipe stage is stalled the valid bit should be "shut off" by qualifying it with the pipe stage n+1's stall signal. Otherwise the n pipe stage will be stalled and the valid bit will be turned on. This will result in copies of the valid bit and data in pipe stage n being transmitted to the down stream pipe stages.
The next problem designers face in digital pipelines is register overwrite. This occurs when the enable signal of a pipeline is not qualified with the n+1 pipe stage.

Pipelines may split and merge. A split occurs when a single pipeline splits into more than one branch.If the multiple branches will subsequently merge the designer needs to determine if the merge point can handle more than on valid pipeline input. Examples of pipelines that split and then merge which can only have one input valid include multimedia filters which use branches to implement different filter types, where the filter type is selected by a control pipeline which is in sync with the valid bit and data pipe stages. The control pipeline contains the operation to perform. In order to ensure that only one valid bit is received at the merge point several strategies may be used.

1. If there are the same number of pipe stages in the two pipeline branches then at the pipeline split point only the pipeline branch which has a valid operation will have the valid bit set. The pipeline that has an invalid operation will have its valid bit shut off. The implementation uses the operation specified in the control branch to qualify the valid bits in the split pipelines.

2. There may be a a different number of pipe stages in the pipeline branches. Different operations

may be in flight in the different branches. The merge point needs to select the branch with a valid bit and stall the other branches with valid bits at the merge point.
Another example of a split and merge pipeline is a micro processor pipeline. The microprocessor has different execution units. Only one execution unit will process the data from the register file. The valid bits in each pipe stage are mutually exclusive. Only one valid bit can be on in each pipe stage.
A processor should contain a valid bit for each pipe stage. The valid bit is used to qualify the operations using in that pipe stage. The valid bits can be temporarily turned off by the down stream stall bit to prevent register leaks (the output of a register is in effect disabled).
The valid bits can be zero'ed out if the the operations in multiple pipe stages should be invalidated due to a branch. This in effect flushes the pipeline in one cycle.



1.4.4.2 suffixes

All SE's are labeled with a valid phase suffix and pipestage
this can be used later to check if the SE's are correctly connected

CD RISC

FIGURE 1.30 Behavioral view

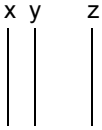


FIGURE 1.31 micro pipelining - each pipeline turns on when required.

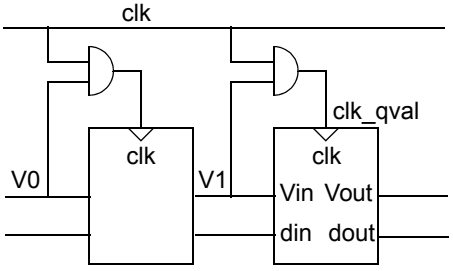
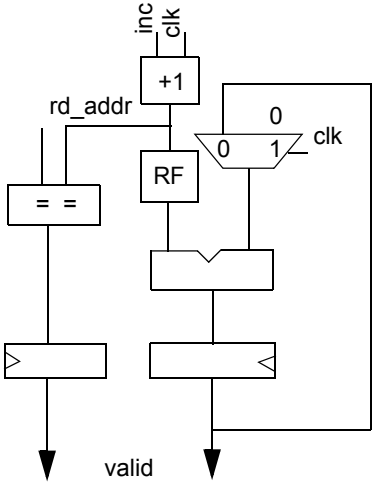
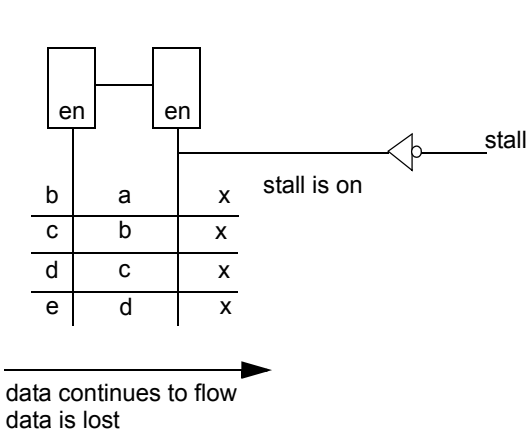


FIGURE 1.32 Loop implementation using an iterative approach



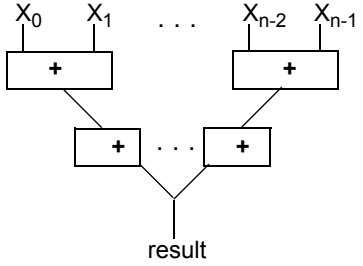
</ADD>
functional unit is driven by valid -so clock for qualified valid
csl_pl pl
csl_ps ps0, ps1, ps2
ps2.qualifysignal_with_enable(v);

Register Leak



To eliminate duplicate data from being sent down the pipeline the valid bit in the last stage in the pipeline is qualified with the enable bit.

FIGURE 1.33 Loop unrolling using parallel operations



dependency time