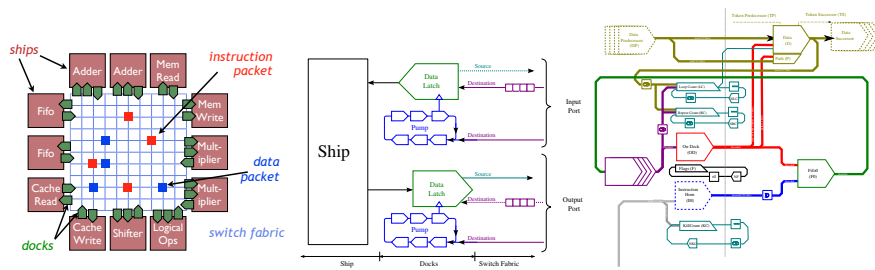


The FleetTwo Dock

January 15, 2008

Abstract



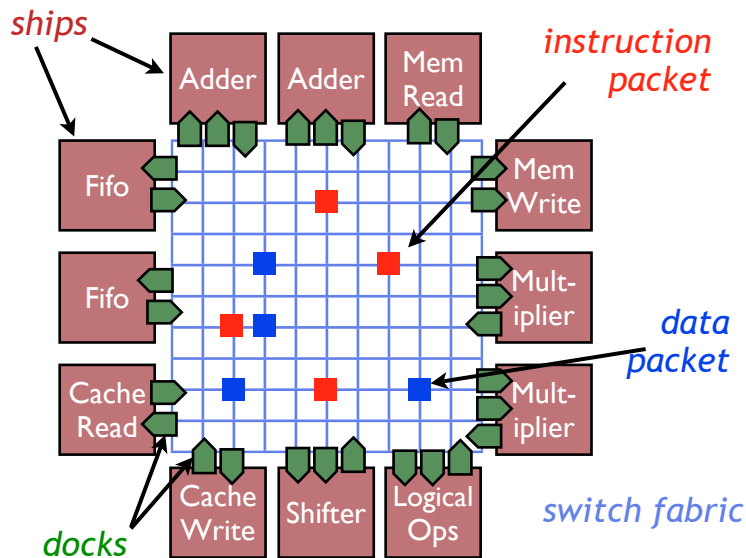
1 Overview of Fleet

A Fleet processor consists of a *switch fabric* with several functional units called *ships* connected to it. At each connection between a ship and the switch fabric lies a programmable element known as the *dock*.

A *path* specifies a route through the switch fabric from a particular *source* to a particular *destination*. The combination of a path and a single word *payload* is called a *packet*. The switch fabric carries packets from their sources to their destinations. Each dock has two destinations: one for *instructions* and one for *data*. A Fleet is programmed by depositing packets into the switch fabric; these packets' paths lead them to the instruction destinations of the docks.

When a packet arrives at the instruction destination of a dock, it is enqueued for execution. Before the instruction executes, it may cause the dock to wait for a packet to arrive at the dock's data destination or for a value to be presented by the ship. It may present a data value to the ship or transmit it for transmission to some other destination.

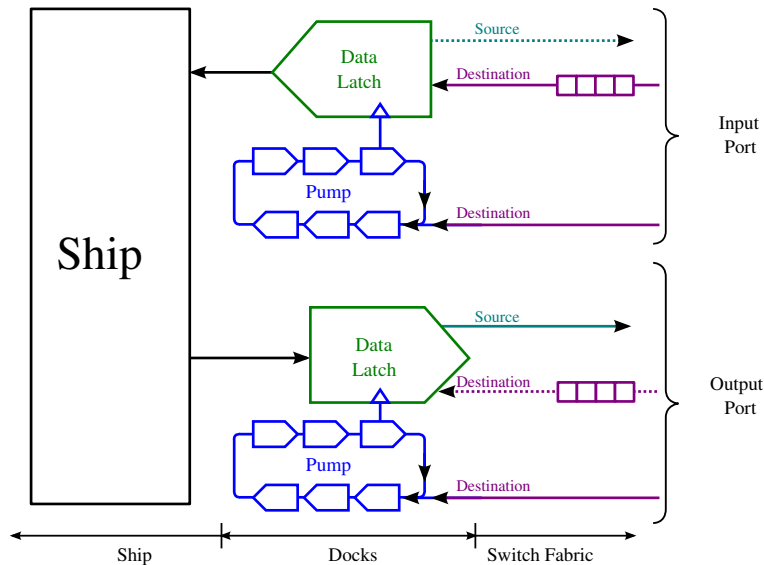
When an instruction sends a packet into the switch fabric, it may specify that the payload of the packet is irrelevant. Such packets are known as *tokens*, and consume less energy than data packets. From a programmer's perspective, a token packet is indistinguishable from a data packet with a unknown payload.



Overview of a Fleet processor

2 The Ship-Switch Fabric Interface

The diagram below represents a *programmer's* conceptual view of the interface between ships and the switch fabric. Actual implementation circuitry may differ substantially. Sources and destinations that can send and receive only tokens – not data items – are drawn as dashed lines.



The interface between the switch fabric and the ship

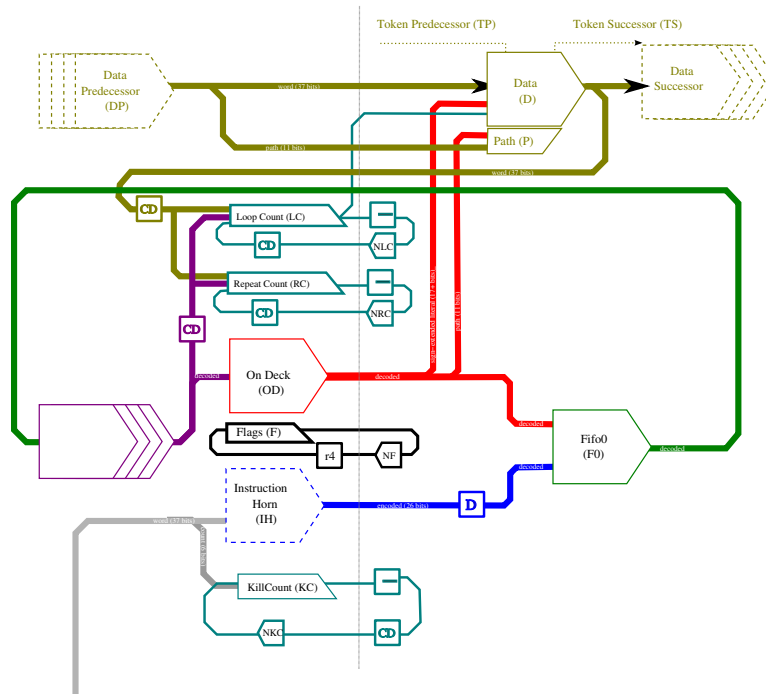
The term *port* refers to an interface to the ship, the *dock* connecting it to the switch fabric, and the corresponding sources and destinations on the switch fabric.

Each dock consists of a *data latch*, which is as wide as a single machine word and a *pump*, which is a circular fifo of instruction-width latches. The values in the instruction fifo control the data latch.

Note that the pump in each dock has a destination of its own; this is the *instruction destination* mentioned in the previous section. Note that unlike all other destinations, there is no buffering fifo guarding this one. The size of these fifos are exposed to the software programmer so she can avoid deadlock.

3 The FleetTwo Pump

The diagram below shows the datapath for the FleetTwo pump circuitry. The square box marked D on the output from the IH latch is the instruction decoder, which decodes word-width instructions into a set of control signals suitable for operating the pump. The boxes marked CD are carry detectors. These detect zero values in the count and also generate the partial differences used in the decrement operation.



The pump datapath

The latches of primary interest here are:

- IH: Instruction Horn (leaf node; may be shared)
- F0: Fifo Stage 0 (first fifo stage)
- OD: On Deck
- F: Flags, NF: Next Flags
- P: Path (the path to use for outbound data/tokens)
- D: Data
- DP: Data Predecessor (ship for output ports, switch fabric for input ports)
- DS: Data Successor (switch fabric for output ports, ship for input ports)

- RC: Repeat Count, NRC: Next Repeat Count
- LC: Loop Count, NLC: Next Loop Count
- KC: Kill Count, NKC: Next Kill Count

Each instruction that executes causes the latches of the pump to fire in two phases, denoted as the “left phase” and the “right phase”. In the diagram, the left phase latches are those to the left of the vertical line down the center, and the right phase latches are to the right. Therefore each instruction execution requires two GasP pipeline stages to complete.

3.1 Flags

The pump has four flags: A, B, S, Z. Of these four, only the first two may be modified directly by instructions.

- The A and B flags are general-purpose flags which may be set and cleared by the programmer.
- The S flag, known as the *summary* flag, is generated from the value currently in latch D. The function which generates this flag is ship-specific, but the default choice is the most significant bit of the value in the latch unless stated otherwise. Essentially, this function indicates how to interpret a word as a boolean; this interpretation is ship-specific.
- The Z flag, known as the *zero* flag, is set whenever the value in the loop counter (LC) is zero. This flag can be used to perform certain operations (such as sending a completion token) only on the last iteration of a loop.

Many instruction fields are specified as two-bit *predicates*. These fields contain one of four values, indicating if an action should be taken unconditionally or conditionally on one of the A or B flags:

- 00: if A is set
- 10: if B is set
- 01: if Z is set (LC=0)
- 11: always

4 Instructions

In order to cause an instruction to execute, the programmer must first cause that instruction word to arrive in the data latch of some output dock. For example, this might be the “data read” output dock of the memory access ship or the output of a fifo ship. Once an instruction has arrived at this output dock, it is *dispatched* by sending it to the *instruction port* of the dock at which it is to execute.

Each instruction is 26 bits long, which makes it possible for an instruction and an 11-bit path to fit in a single word of memory. This path is the path from the *dispatching* dock to the *executing* dock.



Note: the instruction encodings below are simply “something to shoot at” and a sanity check to make sure we haven’t overrun our bit budget. The final instruction encodings will probably be different.

All instructions other than *kill*, *massacre*, *clog*, and *unclog* have the following format:



The abbreviation SK stands for *Successor Killable*; if this bit is cleared, then a kill will never prevent the following instruction from executing if the current instruction was executed. This functionality is implemented by giving the pump two states: *killable* and *not killable*. Executing an instruction with the SK bit set puts the pump in the *killable* state; executing an instruction with the SK bit cleared takes it out of this state.

The abbreviation DL stands for *Decrement Loop*; if this bit is set, the loop counter decrements. Once an instruction has finished executing (including repeating, if applicable), the instruction will reloop if the loop count (LC) value was greater than zero *prior to decrementing*.

The abbreviation P stands for *predicate*; this is a two-bit code that indicates if the instruction should be executed or ignored. If an instruction is ignored, it might still reloop.

4.1 ReLooping and RePeating

Instructions which do not repeat have no effect on the repeat counter (except for those that explicitly set it).

An instruction will repeat if **any** of the following are true:

1. The instruction is a send instruction with the ∞ bit set
2. The instruction is a repeating instruction (as shown below) and the repeat counter is greater than zero.

An instruction reloops if **all** of the following are true:

1. The instruction is not a send with the ∞ bit set.
2. The repeat counter has reached zero.
3. The instruction is a “relooping instruction” as shown below.
4. The loop counter is greater than zero. This check takes into account the result of a `literal` instruction, but does not take into account the effect of the DL bit.

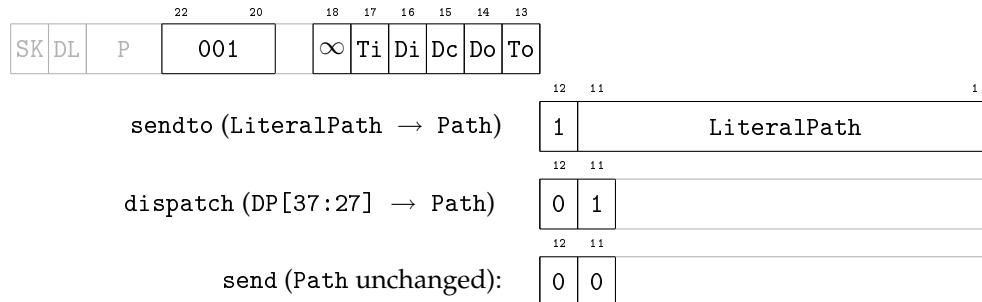
	Repeat	ReLoop
<code>send</code>	Y	Y
<code>literal</code>	N	Y
<code>flags</code>	N	Y
<code>repeat</code>	N	Y
<code>loop</code>	N	Y ^a
<code>takeLoopCounter</code>	N	Y
<code>clog</code>	N	N
<code>unclog</code>	n/a	n/a
<code>kill</code>	n/a	n/a
<code>massacre</code>	n/a	n/a

^anote, however, that the decision to reloop or not is based on the value in the loop counter *before* execution of the `loop` instruction

Table 1:

Important: if an instruction is *not* to reloop (according to the rules above), it must execute even if the head of the instruction fifo is occupied (full).

4.2 send (variants: sendto, dispatch)



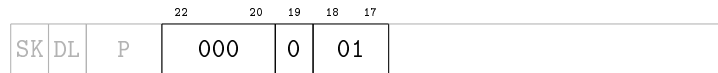
- ∞ - standing instruction
- Ti - Token Input: wait for the token predecessor to be full and drain it.
- Di - Data Input: wait for the data predecessor to be full and drain it.
- Dc - Data Capture: pulse the data latch.
- Do - Data Output: fill the data successor.
- To - Token Output: fill the token successor.

The F0, DS, and TS stages must all be empty in order for an instruction to execute.

If the ∞ bit is *not* set, the repeat count (RC) is latched with $\max(\text{RC}-1, 0)$, and the instruction retires if $\text{RC}=0$ prior to decrementing. Otherwise, the instruction repeats.

4.5 repeat

This instruction loads the repeat counter with either a literal or the contents of the data register.



from data latch:

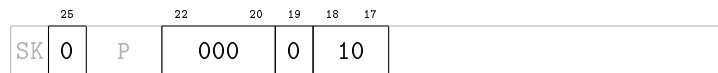
0	
---	--

from literal:

1	Literal
---	---------

4.6 loop

This instruction loads the loop counter with either a literal or the contents of the data register.



from data latch:

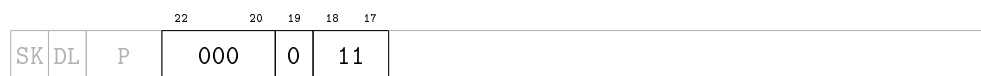
0	
---	--

from literal:

1	Literal
---	---------

Note that the bit normally marked DL (“decrement loop counter”) **must** be set to 0 in this case.

4.7 takeLoopCounter



The P field is a predicate; if it does not hold, the instruction is ignored (but may reloop). This instruction copies the value in the loop counter LC into the least significant bits of the data latch and leaves all other bits of the data latch unchanged.

4.8 kill



When a `kill` instruction reaches IH, it will wait there for the pump to be in the *killable* state and for the OD stage to be full. When this occurs, the instruction at OD is retired and the count field of the `kill` instruction is decremented. If the `kill` instruction's count was 0 before decrementing, the `kill` instruction is retired. The programmer is assured that a `kill` instruction with a count of n will kill $n + 1$ *consecutive* instructions.

Note that a counting kill is capable of "killing through" instructions whose predecessor has the SK ("successor killable") bit not set.

4.9 massacre



When a `massacre` instruction reaches IH, it will wait there for the pump to be in the *killable* state and for the OD stage to be full. When this occurs, all instructions in the instruction fifo are retired.

4.10 clog



When a `clog` instruction reaches OD, it remains there and no more instructions will be executed until an `unclog` is performed.

4.11 unclog



When an `unclog` instruction reaches IH, it will wait there until a `clog` instruction is at OD. When this occurs, both instructions retire.

Note that issuing an `unclog` instruction to a dock which is not clogged and whose instruction fifo contains no `clog` instructions will cause the dock to deadlock.

