DRAFT: Do not distribute

MIT Concurrent VLSI Architecture Memo ??

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

The Architecture of the MAP Synchronization Stage

Draft Version

Steve Keckler

February 15, 1995

Abstract

This document describes the microarchitecture of the synchronization stage of the cluster pipeline in the MAP chip.

1 Overview

The synchronization stage of the pipeline is responsible for selecting valid instructions to issue to the execution stage of pipeline. Figure 1 shows a high level view of the synchronization stage. The *Instruction Buffer* structure has a slot for an instruction from each V-Thread. Instructions wait in the instruction buffer until they issue; they may wait for pending data dependencies or for pipeline or other resources to become available. Bypass signals are used to update operand fields of instructions already in the instruction buffer. The instruction buffer provides to the *resource checker* which indicate which instructions have all of their data dependencies satisfied, and what resources they need to use.

The resource checker compares the resources available (e.g. integer pipeline or send buffer), to the resources required by the instructions. It forwards those that are available to the arbiter. In addition, the resource checker has inputs from the cluster control module which holds the thread state. These inputs indicate thread priority, which threads are enabled to run, and which threads threads have been waiting to issue for a long time. The thread state controller contains stall cycle counter and the counter threshold registers used to prevent starvation and deadlock. Threads of lower priority are masked off by the resource checker and are not forwarded to the arbiter.

The arbiter performs a round-robin arbitration among all threads forwarded to it by the resource checker to select one to issue. The *tSel* lines indicate which thread is selected. This value is used locally to write to the execute register and forwarded to the instruction fetch stage, register read stage, and thread controller for use on the next cycle. Two late kill signals may arrive which invalidate the thread select. The *kill* signal is asserted by the C-Switch indicating that promised data will not be delivered. The ifKill signal is asserted by the if the instruction fetch unit cannot deliver a promised instruction from the instruction cache. If either kill signal is asserted matches the thread chosen by the thread selector, then the pipeline is bubbled and tSel is ignored. The signals tSel, kill, and ifKill are distributed throughout the pipeline so that kills may occur locally.

If the instruction contains a send, *thread selector* initiates arbitration for the send buffer. The *sTid* register keeps track of a send operation that is waiting to find out if it has won arbitration among other send operations for access to the send buffer. This is discussed further in Section 7.

1.1 Criteria for issue

An instruction may only be issued if the following conditions are satisfied:

- The H-Thread is in the RUN state
- If the operations belongs to a user thread, user thread issue is enabled
- All operands are valid
- For SEND operations:

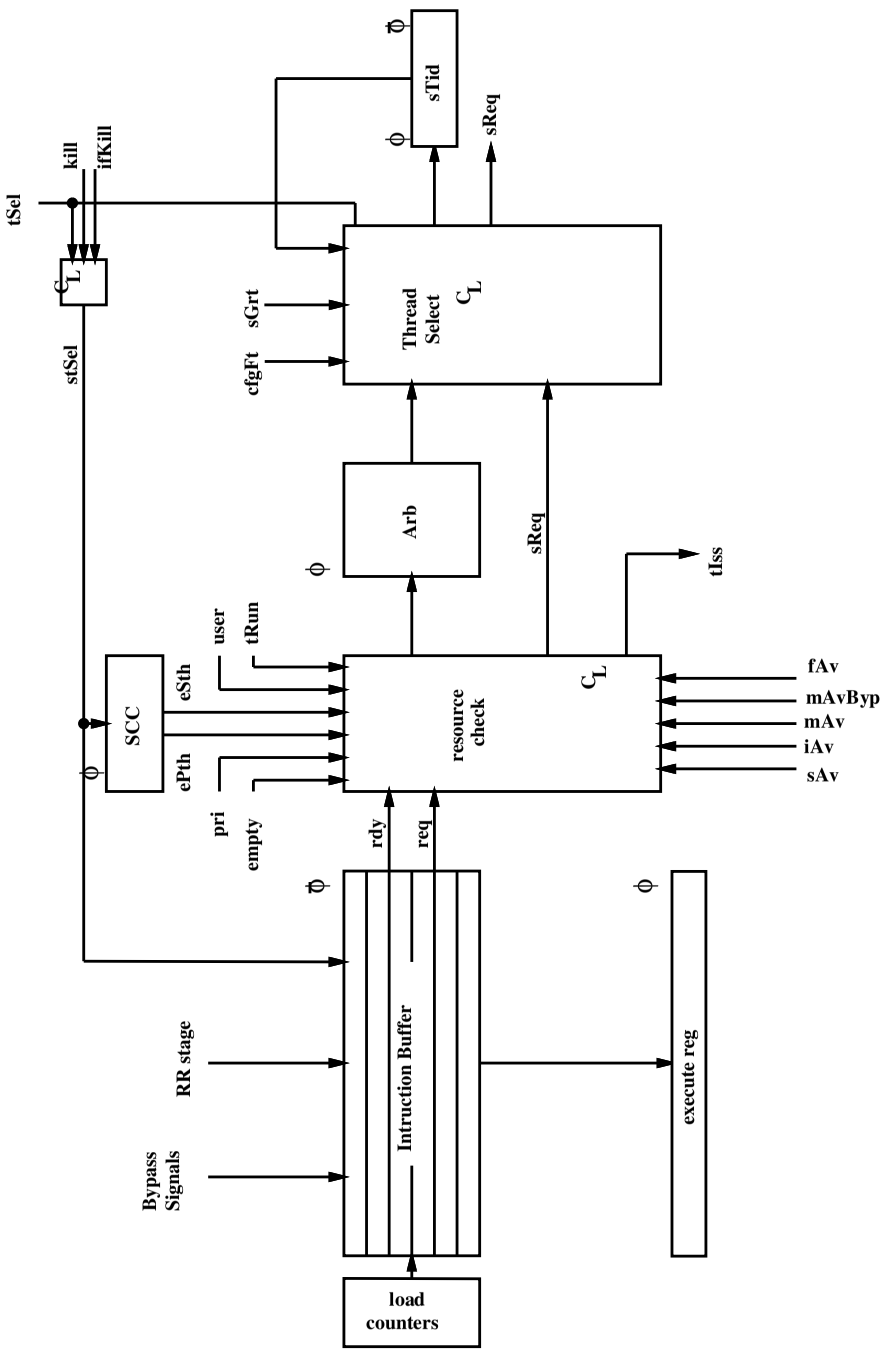

Figure 1: SZ stage top level diagram

  - Specified message composition registers are valid
  - Outgoing message buffering is available
  - The operation wins arbitration for netout
- For LOAD operations
  - Memory barrier counter $<$ max
- For MBAR operation
  - Memory barrier counter = 0
- Pipeline resources for all of the non-NOP operations are available

1.2 Outline

This document describes the following:

- Synchronization stage interface signals
- Instruction buffer design
- Thread state machine
- Resource requirements
- Arbitration policy
- Thread selection
- Timing

Area and power estimates are in progress.

2 Interface Signals

- Cluster Control
  - vAct[0:5] - activate V-Thread
  - vDact[0:5] - disactivate V-Thread
  - hAct[0:5] - activate V-Thread
  - hDact[0:5] - disactivate V-Thread

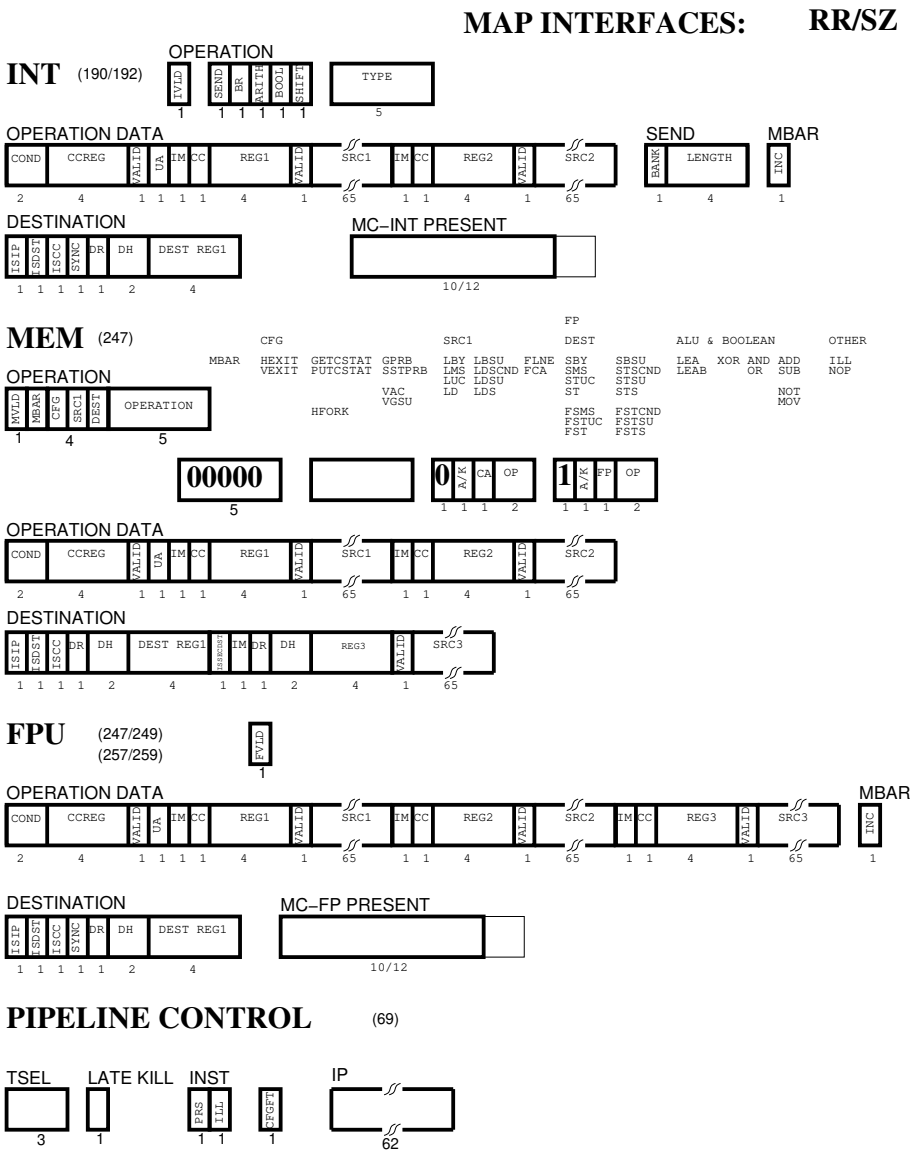

Figure 2: The RR-SZ Interface

MAP INTERFACES: SZ/EX

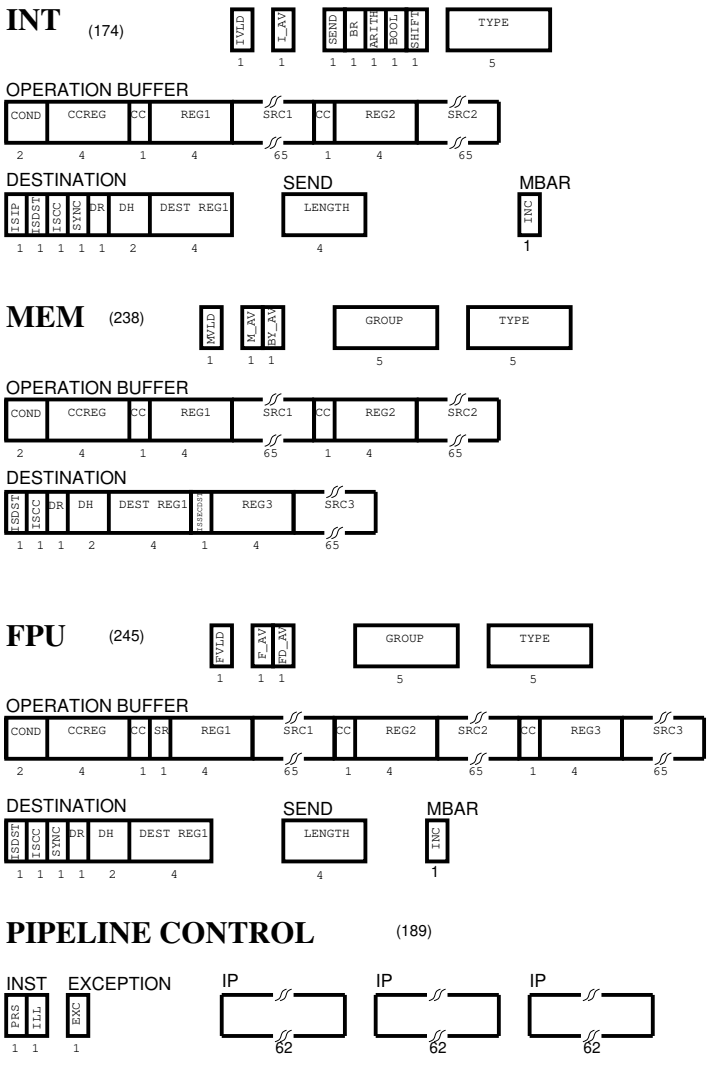

Figure 3: The SZ-EX Interface

  - tIss[0:5] - wired-or line, global to all clusters, which is released by a cluster when the stall cycle counter (SCC) exceeds its threshold. When this line is high, the global controller generates a thread timeout event.
  - exc - indicates that an exception is taking place and no user threads may issue. An exception handler unsets this when it has completed.
  - user - indicates that user threads are allowed to issue instructions.

- RR stage: The RR stage decodes the operations and reads the source operand registers before passing the data to the SZ stage. Figure 2 shows the data in this interface.[1]

- EX stage: The SZ stage places the instruction selected for issue in the EX register for execution during the subsequent cycle. This data includes all of the information required to execute the instruction, such as operations, operands, destinations, etc. Figure 3 show the data in this interface.[2]

- IFETCH

  - empty[0:5] - (input) indicates that an attempt to issue an instruction was made but failed due to insufficient number of operands in ifetch buffer. This indicates that tSel must be asserted later to get the requested instruction into the pipeline.
  - tRun[0:5] - (output) indicates that the corresponding threads are running.
  - tSel[0:5] - (output) indicates a request to issue an instruction down the pipeline from the IFETCH unit.
  - ifKill - (input) one line from the I-Fetch unit which indicates that the next instruction from the thread indicated by tSel on the previous cycle was not delivered. If the next instruction for this thread was selected by the SZ stage, the pipeline is bubbled.

- Bypassed values There are in excess of 389 bypass wires for the entire synchronization stage including thread and register specifiers.

  - CC Registers (72 wires for bypass for IIB, MIB, FIB):
    * local
      · data: 4 per hthread (need not be bypassed to SZ stage)
      · valid: 4 per thread
    * global
    * data: 8 per thread (need not be bypassed to SZ stage)
    * valid: 8 per thread

[1] This figure is replicated from Andrew Chang's register read stage design document.
[2] This figure is replicated from Andrew Chang's execute stage design document.

- Register Validation (36 wires for bypass for IIB, MIB, FIB if we allow automatic invalidation between register files in same cluster)
  - Validation from event queues for event threads (1)
  - Invalid from M-EX (3:tid + 4:rid)
  - Valid from CSW (3:tid + 4:rid)
  - Invalid from F-EX (3:tid + 4:rid)
  - Valid from F-EX (3:tid + 4:rid)
  - invalid from CSW late kill (3:tid + 4:rid)
- Register Data (219 for IIB, 292 for MIB, 146 for FIB):
  - I-EX (3:tid + 4:rid + 66:data)
  - M-EX (3:tid + 4:rid + 66:data)
  - F-EX (3:tid + 4:rid + 66:data)
  - CSW (3:tid + 4:rid + 66:data)

Thus the totals are:

- IIB: 327 bypass wires
- MIB: 400 bypass wires
- FIB: 254 bypass wires

- Pipeline signals
  - CFGFT - (input) configuration space feed through. This suspends arbitration and allows a configurations space register read request to go directly from the register read stage to the execute stage.
  - I_AV - (input) integer unit is available to accept an operation. This is false when the integer unit has lost C-Switch arbitration.
  - M_AV - (input) memory unit is available to accept an operation. This is false when the memory unit has lost M-Switch arbitration.
  - M_AVBYP - (input) memory unit is available to accept a bypass operation. This is false when the bypass path of the TLB miss handling memory unit has lost M-Switch arbitration.
  - F_AV - (input) floating point unit is available to accept an operation. This is false if the FP unit is executing a long latency operations (such a FDIV) or if the FP unit has lost C-Switch arbitration.
  - KILL[0:5] - (input) kill signal from the C-Switch to squash instruction issue for a selected thread if promised data cannot be delivered. This is one-hot encoded.
- Send resource arbitration

- S0_AV - (input) priority 0 send buffer can accept a message
- S0_REQ - (output) request for access priority 0 send buffer
- S0_GRT - (input) grant line for access to priority 0 send buffer
- S1_AV - (input) priority 0 send buffer can accept a message
- S1_REQ - (output) request for access priority 0 send buffer
- S1_GRT - (input) grant line for access to priority 0 send buffer

3 Instruction Buffer

3.1 Overview

The instruction buffer is distributed across the 3 function units in the cluster. All of the updates to the state in the instruction buffer takes place there, as does local checking of operation validity. The operation valid signals are combined at a central location to generate the instruction valid signals for the thread state machine. The instruction's IP is also kept at a central location so that it may be passed to the execution stage and used when generating error values (actually, we might need the IP for all operations since invalid operations–nops–do not use pipeline resources). In addition, two wires to signal that send arbitration needs to be performed are provided for the thread selector. All of the fields of the instruction buffer are bit interleaved. Figure 4 shows a diagram of the instruction buffer module.

3.2 Bypass

Figure 5 shows a simple instruction buffer data cell from the IIB, with bypass paths. The register cell has a single read and write port. All possible writers must be multiplexed in. The logic at the bottom of the diagram shows the control signals for the data cell. In addition to data cells, valid and control cells are required. The valid cells hold the status of the data and is used to determine if all of the operands are valid. Thus the valid bits need not be routed to the EX stage, but additional local logic is required (a local AND of all valid signals) to determine if the operation is valid. The control cells hold the operation specification (opcode, operand specifiers, etc.) and are used to drive the data and valid cell control lines.

For the IIB we need the following cells:

- cond: 1 input
- ccreg: 1 input
- valid: 12 inputs (1 for each CC reg)
- ua, im, cc, reg1: 1 input

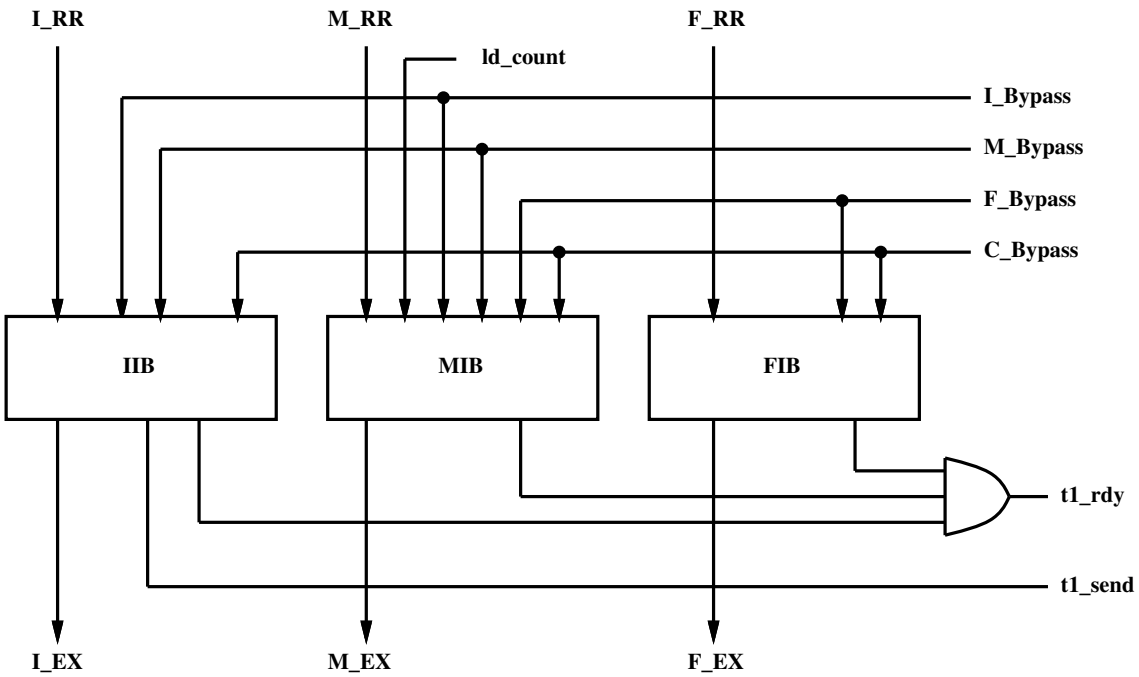

Figure 4: Instruction buffer.

- valid1: (12:ccreg, 1:eq, 1:m-ex, 2:csw) - 17 inputs
- src1: 4 inputs (times 66)
- im, cc, reg2: 1 input
- valid2: (12:ccreg, 1:eq, 1:m-ex, 2:csw) - 17 inputs
- src2: 4 inputs (times 66)
- bank, length, mbar, dest: 1 input
- mc-int present: 5 inputs

The control portion for each data field will have the same number of inputs as the field itself.

3.3 Validity Check

The validity check is essentially a giant AND of all operand valid bits. This includes all of the source registers, condition registers, message composition registers for send operations, and a check on the memory barrier counter for load and MBAR operations. For MBAR

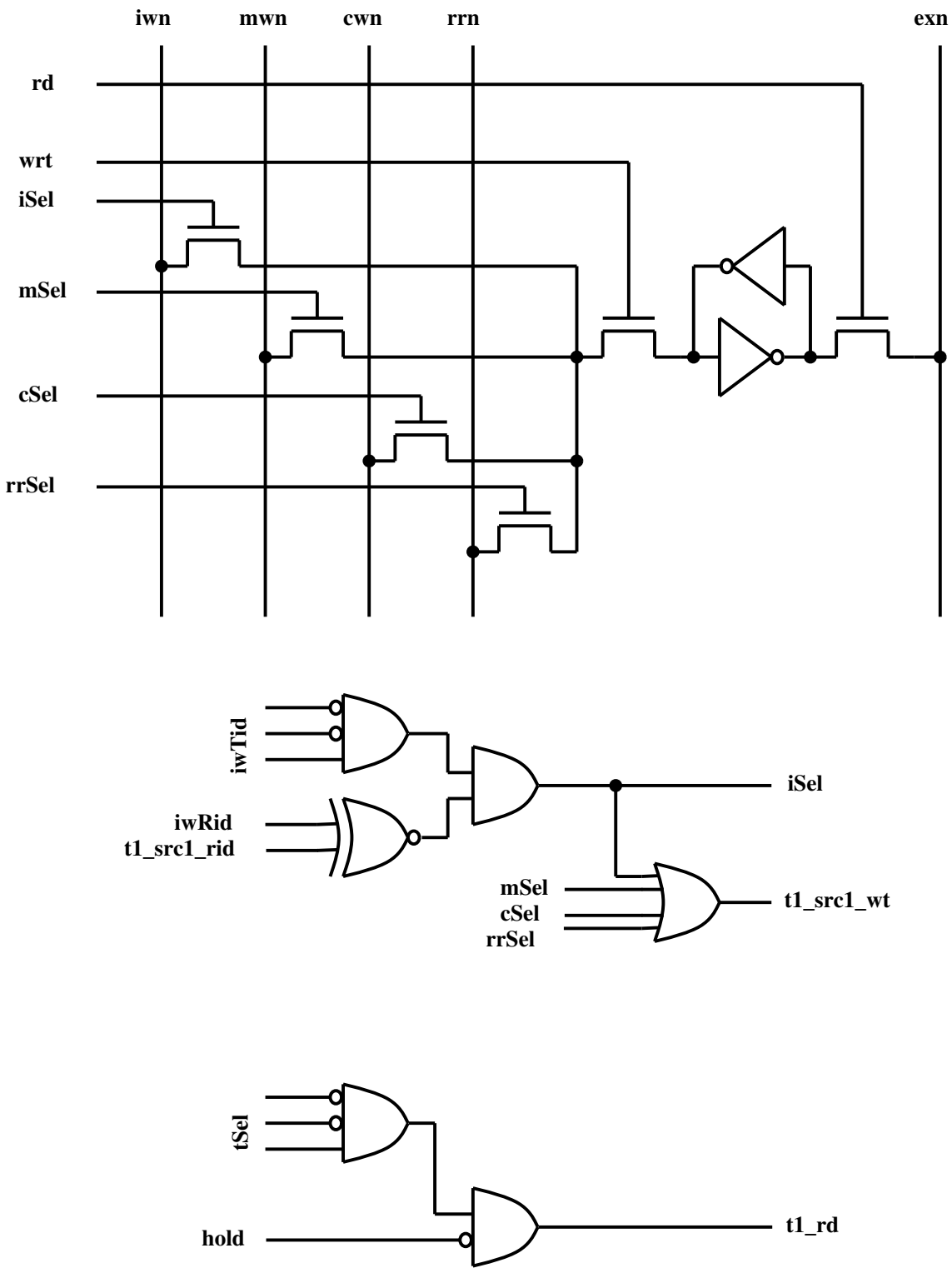

Figure 5: Instruction buffer cell and update logic.

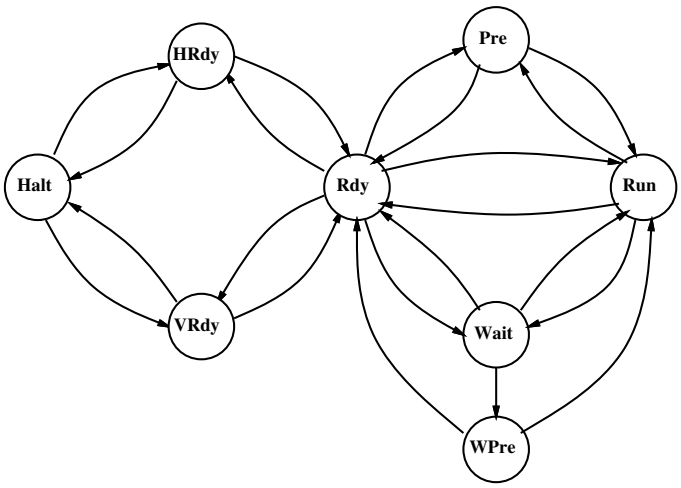

Figure 6: H-Thread States

to issue the memory barrier counter must be zero. For a load to issue the memory barrier counter must be less than the max value. This logic will be placed under the wires in the instruction buffer. Additional wire tracks will be required to concentrate all of the individual valid signals.

4 Thread State Machine

A thread may be in one of the following states:

- Halt - H-Thread is not enabled to run. The thread is in this state if the H-Thread is vacant or if the V-Thread is halted
- V-Act - The V-Thread is active but the H-Thread is not allowed to issue instructions.
- H-Act - The H-Thread will be allowed to issue instructions as soon as its V-Thread becomes active.
- Pnd - (pending) Thread may execute operations but not all operands are valid
- Rdy - Thread may execute operations and all operands are valid
- Pre - Thread has higher priority to issue for a single cycle.
- Wait - Thread is waiting for IF unit to deliver another instruction down the pipe.
- WPre - Thread is has a higher priority for a single cycle to eliminate a bubble in the top half of the pipe.

State	Inputs	Next State	Outputs
Halt	*hAct*	HRdy	*tRun*
	vAct	VRdy	
HRdy	*vAct*	Rdy	*tRun*, *pClear*
	hDact	Halt	
VRdy	*hAct*	Rdy	*tRun*, *pClear*
	vDact	Halt	
Rdy	*Ops Valid*, *tSel*	Rdy	*tRun*, *arb*, *sccReset*
	Ops Invalid	Pnd	*tRun*
	Ops Valid, *pth*	Pre	*tRun*, *TPre*
	empty	Wait	*tRun*, *sccReset*
	vDact	HRdy	
	hDact	VRdy	
Pnd	*Ops Valid*	Rdy	*tRun*, *Arb*, *sccReset*
	Ops Invalid	Pnd	*tRun*
	Ops Invalid, *sth*	Pnd	$\overline{tIss}$
	empty	Wait	*tRun*, *sccReset*
	vDact	HRdy	
	hDact	VRdy	
Pre	*tSel*, *Ops Valid*	Rdy	*tRun*, *sccReset*
	tSel, *Ops Invalid*	Pnd	*tRun*, *sccReset*
	vDact	HRdy	
	hDact	VRdy	
Wait	$\overline{empty}$	Wait	*tRun*, *sccReset*
	$\overline{empty}$, *tSel*, *Ops Valid*	Rdy	*tRun*
	$\overline{empty}$, *tSel*, *Ops Invalid*	Pnd	*tRun*
	$\overline{empty}$, *pth*	WPre	*tRun*
	empty, *sth*	Wait	$\overline{tIss}$
	vDact	HRdy	
	hDact	VRdy	
WPre	*tSel*, *Ops Valid*	Rdy	*tRun*
	tSel, *Ops Invalid*	Pnd	*tRun*
	vDact	HRdy	
	hDact	VRdy	

Table 1: Thread State Table

Figure 6 shows the transition diagram and Table 1 shows the state transition table for these states. A thread moves from *Halt* to *Rdy* when its V-Thread and H-Thread active signals are asserted. The thread only returns to Halt when the V-Thread and H-Thread are disactivated. If the user threads are prohibited from issue (such as during an exception), the thread remains in the running state but the resource checking logic will prevent instruction issue.

A thread moves to the *Wait* state when the IFETCH unit raises the empty bit for the thread, indicating that a prior tSel has been unable to be satisfied. While empty is true, the thread will not issue instructions. When empty becomes false, the thread remains in the wait state but becomes eligible for instruction issue (and has, by definition, all of its data dependencies satisfied). A thread may move from the Wait state to the WPre state when empty is no longer asserted and the counter exceeds the PTH. When in the Wait state, if empty is true, then the counter may trigger a STH event. When empty becomes false, the SCC is reset and later compared with the PTH.

When the thread is selected for issue, no operation is issued to the EX stage, but tSel is asserted. tSel tells the IFETCH and RR units to deliver an operation. Locally, a HOLD signal tells the instruction buffer to keep the current state and throw away whatever the RR stage gives it. The thread then moves back to the PND or RDY state. All state transitions are latched at the end of phi1 (rising clock).

In addition, the thread state machine includes the stall cycle counter (SCC), and two threshold registers: the preempt threshold register (PTH) and the stall threshold register (STH). The SCC is incremented while the thread is in the PND or RDY states and tSel is not asserted. The SCC is cleared whenever tSel is asserted or when the thread transitions from PND to RDY. When the SCC exceeds the STH and the thread is in the PND state, tIss is deasserted, which may signal an event in the global control unit. When the SCC exceeds the PTH and the thread is in the RDY state, the thread transitions to the PRE state, gaining higher priority until it issues.

5 Resource Check

The resource checking logic is responsible for filtering out those threads whose resources are not available, thus preventing them from arbitrating for issue. In addition, the priority of threads is checked, and only threads of the highest equal priority are allowed to arbitrate. Figure 7 shows the inputs and output to the resource control module.

5.1 Thread Prioritization

The synchronization stage selects based on a software controlled priority bit. One bit is provided for each user thread; a single bit controls both user and exception threads. The priority bit may be set through configuration space.

The partial order on priorities is shown in Figure 8. If all user threads are are at the same priority then there is no scheduling bias among them. However, user threads at *U1*

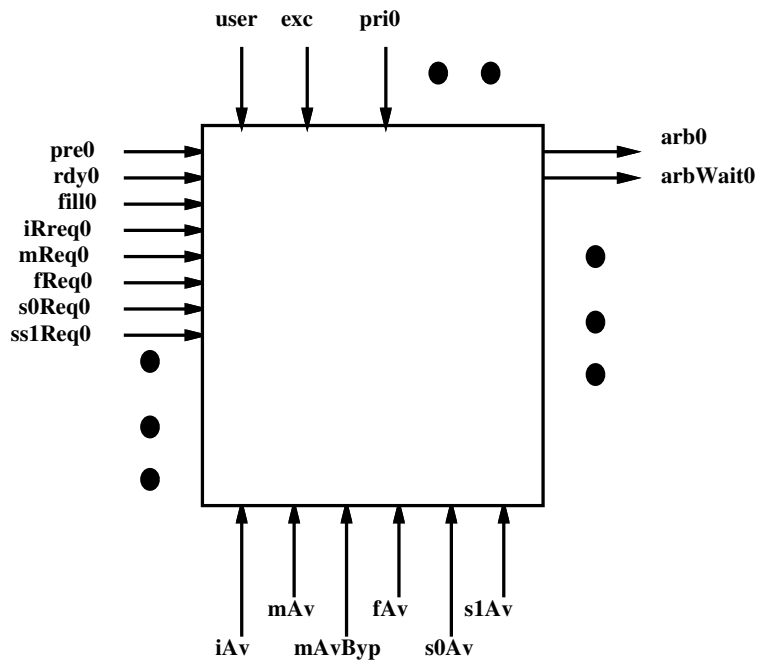

Figure 7: Inputs and Outputs for the resource control module.

will be granted issue slots over threads at *U0*, given that resources are available for both.

The event thread may set its priority to *S1*, in which it has the highest priority, always issuing if it's resources are available. When the event thread is at *S0*, it is equivalent in priority to the highest priority user thread *U0* or *U1*. The exception thread is always equivalent in priority to a user thread (say *U1*). Thus the event thread may choose to be at the same or higher priority.

Under certain situations, a thread may be ready to issue (all of its operands are present), but is starved due to other threads gaining issue slots and consuming pipeline resources. Livelock can occur with three threads: two threads repeatedly consume resources, while the third loses out every cycle. In order to guarantee no starvation, a preempt threshold (PTH) register is used for each thread. When a thread becomes ready to issue, the stall cycle counter (SCC) is set to zero, and is subsequently incremented every time the thread is denied issue. If the SCC exceeds the PTH, then the thread enters the *preempt* state. When any thread is in the preempt state, threads in the run state are prohibited from issue, thus guaranteeing that the preempt thread will issue an instruction. As soon as the preempt thread issues an instruction, it returns to the run state.

5.2 Resource Checking Protocol

The gross functionality of the resource control module is described below:

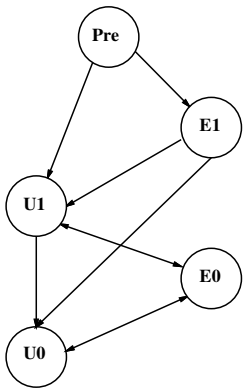

Figure 8: Thread Priority

- Always, all threads of equal highest priority with RDY and FILL are released to ArbWait.
- If no threads have PRE, then those threads of equal highest priority with RDY (U0 and E0, U1 and E0, or E1) and resources available are released to ArbAll.
- If any thread has RDY and PRE, then priority is ignored and only those threads with PRE and resources available and released to ArbAll. If no PRE threads have resources available, then no thread is released to ArbAll.

6 Arbitration

Arbitration in each of the arbiters (ArbAll and ArbWait) is performed using a simple round robin scheme, with a rotating token and possibly using lookahead. The exact design of the arbiters will influenced heavily by the C-Switch and M-Switch designs. If necessary the ArbWait arbiter could use a fixed priority scheme.

The second arbiter (ArbWait) is used to fill bubbles in the pipeline in the event a late kill signal arrives and aborts the thread that won arbitration for the execution stage. Instead of bubbling the entire pipeline, this opportunity can be used to fill a bubble in the top part of the pipeline incurred at a previous time.

7 Thread Selection

The thread selection module must be extremely simple. It makes the final decision as to whether tSel is to be asserted and for what thread. If a late kill comes in, the selector can switch in the ArbWait winner instead of the ArbAll winner. In addition, it arbitrates for the send resources and hold the send request for issue on the next cycle.

7.1 Send Resources

In order for a send operations to arbitrate for issue, space in the right priority send buffer must be available (sAv). If a send instruction wins arbitration in ArbAll, it must then arbitrate for the connection to the send buffer with send instructions from other clusters. The send instruction's identifier is placed in the sTid for use on the next cycle. In addition, sReq is asserted to arbitrate for send resources during the next cycle. At this point, the ArbWait winner, if any, may assert tSel.

On the next cycle, the entire arbitration procedure is repeated and the tread selection logic must choose between the thread in sTid, and the ArbAll and ArbWait winners. If sGrt is asserted, the send operation is issued. Otherwise, the send is returned to the arbitration pool (sTid) is nullified, and the winner from ArbAll is issued. If that operation is late killed, then the ArbWait winner asserts tSel.

7.2 Late Kill

Two late kill signals may affect the decision of the thread selector:

kill: The kill signal comes from the C-Switch and indicates that the data promised on this cycle will not be delivered. The thread selector compares this signal with the thread id of the ArbAll winner. If it is the same, then the ArbWait winner asserts tSel. The aborted operation must arbitrate again on a later cycle. The late may mark an operand invalid in the instruction buffer, rendering a previously ready operation pending.

ifKill: The ifKill signal arrives from the IF stage and indicates that the instruction that issued on the previous cycle has caused an instruction cache fault. Thus it must not issue an operation on this cycle. If ifKill is asserted, then if lastTsel matches the ArbAll winner then the ArbWait winner asserts tSel. Otherwise, the ArbAll winner issues.

8 Timing

Figure 9 is a timing diagram of the synchronization stage, showing the critical timing paths.

9 Area Estimate

10 Power Consumption

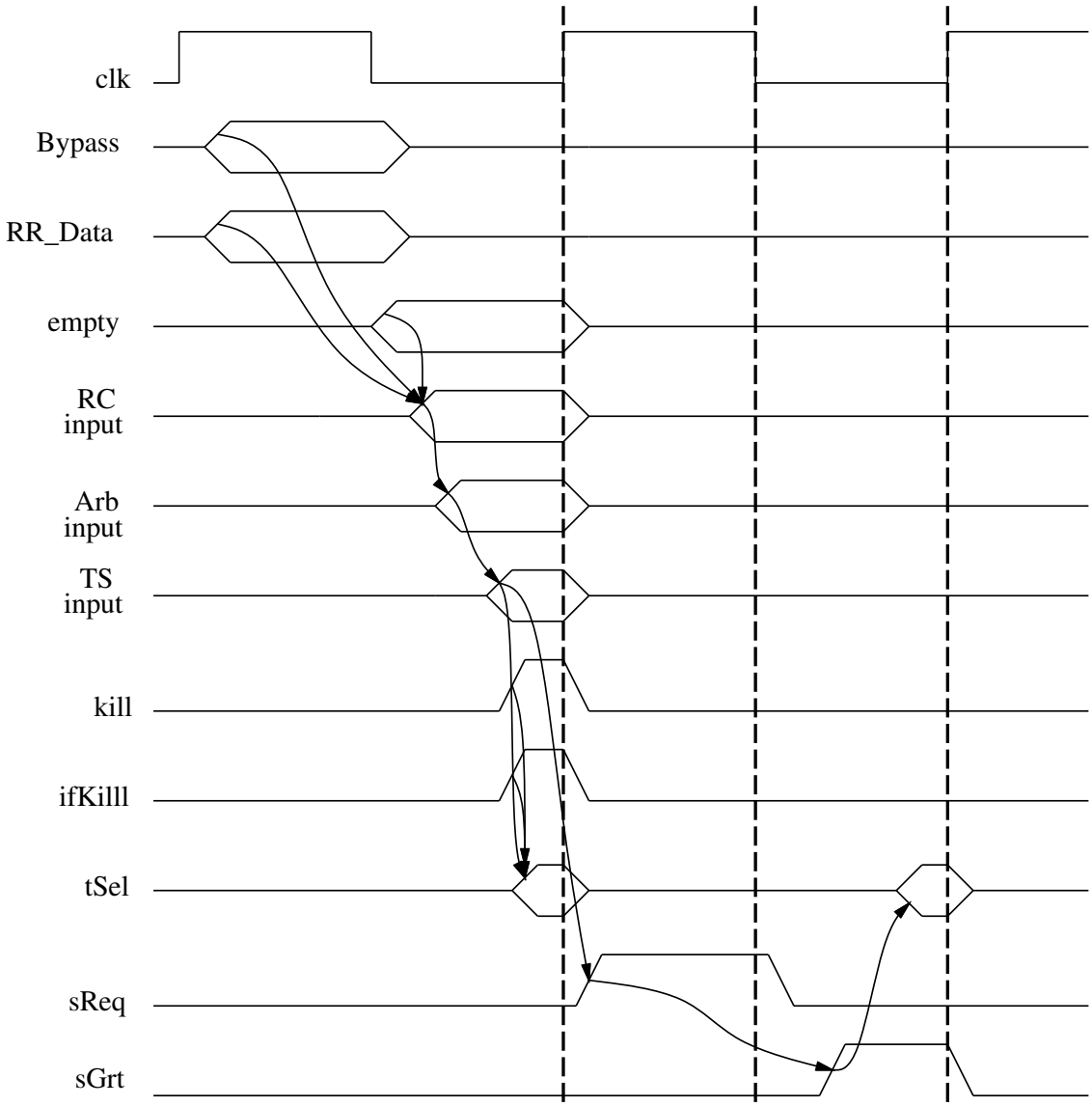

Figure 9: SZ stage timing