```
+---------------+
! d i g i t a l !   I N T E R O F F I C E   M E M O R A N D U M
+---------------+
```

TO:	Dolphin Project List
DATE:	16 Aug 78
FROM:	Alan Kotok
DEPT:	LCEG
DTN #:	231-6381
LOC:	MR1-2/E47
FILE:	SPDI

SUBJ: Shared Pages Data Integrity (SPDI) Box Specification

Dolphin SPDI Box Functional Specification Revision 1

1.0 INTRODUCTION

The Shared Pages Data Integrity (SPDI) Box is an optional unit which plugs into the Dolphin Bus. Its function is to insure data integrity between the caches of a multi-processor system, for which it is required. It works in conjunction with the Mboxes of the various processors to prevent stale data from accumulating in caches. The overall scheme is described in a memo from Kotok entitled "Shared Pages in Multiprocessor Dolphin Systems". The latest revision of that memo at this writing was 2, dated 7 Aug 78.

2.0 TECHNOLOGY

The SPDI Box is built with Macro Cell Arrays. It should require no RAMs, and uses little more than the standard bus interface chipset. It should fit on a fraction of a board.

3.0 FUNCTIONAL DESCRIPTION

The SPDI Box sits on the Dolphin Bus and recognizes bus requests for memory having address bit 9 a one. It will also recognize requests in I/O address space for its internal registers and status. Its most commonly used function is with "write" requests to memory. The processors will detect "shared writable" pages, and tag them with a one in bit 9 of their addresses. These pages actually live in physical memory in the corresponding physical page with bit 9 a zero.

A "write" request to an address with bit 9 a one is acknowleged by the SPDI box if it is not busy. The SPDI box will then re-issue a write request, deleting bit 9, and sending along the data word(s) received. (At this time no use is known for other than single word requests.) It then issues "cache zapper" bus requests to all other active processors in the system. These are bus cycles which request the processor to kill any copies of the word(s) it may have in its cache. The SPDI box will retry these "zappers" until accepted, or a timeout occurs. The positive acknowledgment by a processor of a zapper bus request is sufficient for the SPDI box, and implies that the processor will zap its cache before issuing any further bus memory requests. The SPDI box remains busy until all zappers have been acknowleged.

CPU MUST DO AGREE TO NO MORE CACHE REF BEFORE ACKNOWLEDGING ZAP TO SPDI

An internal control register in I/O space identifies the active processors, and the requesting processor identifies itself in the ID field of each request.

The only "read" request handled is an "interlock read". Processors will suppress bit 9 from the address on other reads. Upon receipt of an "interlock read", the SPDI box re-issues it with bit 9 suppressed, in the name of the requesting processor.

This arranges that the data will return directly to the requesting processor. The SPDI box will acknowlege the requesting processor immediately, unless it is busy, and will keep trying the memory request until accepted or a timeout occurs. The SPDI box will then enter a "selective busy" state, in which it appears busy to all memory requests except those from the processor which issued the "interlock read". The SPDI Box will remain in this state until the receipt of a "write release" request, which will be handled in the same fashion as described for "write" requests.

THERE SHOULD BE A TIMEOUT ON THE WRITE RELEASE

4.0 APPLICABILITY

Only one SPDI box may be active in the system at one time. This is necessary to insure that data written under one "critical sequence" interlock will be seen by a subsequent processor seizing that interlock. For reliability reasons, it seems desirable to have a "hot standby" unit. For this reason, it may be desirable to combine the functionality defined herein into the Dolphin Bus Repeater, since the most common multiprocessor configuration (2 processors) has exactly 2 bus repeaters.

5.0 COST

It is currently estimated (by John Allen) that the SPDI box will take 17 or 18 MCAs, and 1/2 board. If it is combined with the bus repeater, this drops to 6 or 7 MCAs.

6.0 OPEN ISSUES

Should the SPDI box contain some sort of interprocessor communication facility?

```
+---------------+
! d i g i t a l !   I N T E R O F F I C E   M E M O R A N D U M
+---------------+
```

TO: Dolphin Project List
Jim Fleming
Peter Hurley
Ron McLean

DATE: 07 Aug 78
FROM: Alan Kotok
DEPT: LCEG
DTN #: 231-6381
LOC: MR1-2/E47
FILE: SHARED3

****REVISION 2****

SUBJ: Shared Pages in Multiprocessor Dolphin Systems

This revision (2) corrects some problems discovered in revision 1 in the area of avoinding a deadly embrace, and assuring data modified under an interlock will be propagated before another processor can seize the same interlock.

This memo is to clarify and refine the scheme for handling shared writable pages in multiprocessor Dolphin systems which was the subject of the John Allen memo of 16 Jun 78 (file: MBOX4). It also represents a revision and improvement on the scheme described in my memo of 18 Jul 78 (file: SHARED). This new scheme includes suggestions made by Mike Newman and a brainstorming session in my office with Bosack, Guglielmi, Allen, Lewine, McLean and Murphy.

The problem to be solved is that of shared writable pages. These are defined as pages which have at least one processor which has written the page, and at least one other processor making any references to the page. This state is to be carefully distinguished from the state where many processors have the privilege of writing, but none has yet done so, or the case where only one processor has yet exercised the option of referencing the page, even though several have the privilege.

It is a goal of the system to insure that words written by one processor will be seen by the other processors after a "short" delay, even though for speed reasons, these pages wish to be cached by all processors. Unless an interlock instruction (such as AOSE) is used, no guarantee can be made that simultaneous updating of a word will work, or that one processor will "instantaneously" see the results of another's writing. It is, however, a goal that shared data updated by one processor which is in control of an interlock will be seen only in updated form by a second processor when that processor gains possession of the interlock.

To arrange that data integrity is maintained by the hardware (and microcode), it is first necessary to detect that a page has become shared writable. This is done with the help of several bits in the Core Status Table (CST) for the page. The bits in the CST are one for each processor in the system, currently a maximum of 4; along with the existing "modified" bit. Each "processor" bit is a one if that processor has made any reference to the page. A bit of the physical addressing space is used in conjunction with an optional "Shared Pages Data Integrity" box, which goes on the Dolphin Bus. The function of this bit (bit 9) will be explained below.

Each time a change or update is made to a processor's page table cache, or a page table cache entry is encountered with its CST update bit on, a reference is made to the CST. This reference is an interlocked read-write sequence. A physical page which is already shared-writable is one in which more than one processor reference bit is set, and the "modified" bit is set. In this case, the microcode would

1. Or-in bit 9 into the physical address stored in the page table cache for this entry.

2. Or-in a one bit into its processor use bit of the CST word, and write it back.

There are two cases in which a page needs to be changed to shared writable status. The first is that the CST word as read had the "modified" bit set, and exactly one use bit set, but not the bit for the processor doing the update. This is the case of a page which has been written by another processor, and not yet shared. In this case, the following actions are taken:

1. During the CST update, no modification is made to the CST word.

2. A page fault is taken to the software.

3. The monitor goes thru a CST update interlock and looks to see if the CST "age" field is set to a combination called "transition state", (which is detectable by microcode to block references to the page).

   If so, someone else got here first, and no further action is necessary. Release the CST interlock and skip to step 5.

   If not, set the transition state, write back the CST word, and release the interlock.

4. A request is issued to the processor whose reference bit was on in the CST. This request causes that processor to do the following items:

   1. Issue a cache sweep to validate core on the subject physical page.

   2. Issue a new instruction which scans the page table cache, setting the CST update bit on all entries pointing to the object physical page.

why not just invalidate the entry to force a refill

3. Upon completion of both these items, doing a CST update, changing the "age" field to the current time, and oring-in the processor use bit of the processor which initiated this request.

5. Go off and do something else until a check of the CST word for the object page returns to legal state.

When the processor detecting the above transition is able to resume the process causing the fault, correct data for the object page will all be in main memory, and a mechanism described below will insure continued integrity. Since no successful reference was made to the object page before the above sequence was executed, the first retry of the failed reference will cause a CST update reference to be made, where it will be determined that the page is now shared writable.

A third processor tripping over the same problem as above would either have its microcode CST update find the page in "transition state", and just wait for it to become unblocked, or might just catch the page after the second processor detected the transition and before it set the "transition state". In that case, the third processor will hit the CST update interlock mentioned above, and will either take over the transition action, or yield to the second processor, and wait.

If a processor attempts a CST update and finds that it is the first to attempt to write a page that at least one other processor is reading, the transition to shared writable is much simpler. The microcode simply ors-in the processor reference bit and the modified bit, and writes back the CST word. It then sets bit 9 in its page table cache entry, just as if it had discovered that the page was shared writable before it came along.

notify 1st CPU?

This works because any copies of words from this page in any cache must be correct, since no one had modified them before, and any subsequent modifications will be handled by the "shared pages data integrity" box. Processors not having write privileges on this page, or not yet having written the page do not set bit 9 of their address, and do not set "written" in their page table cache. This distinction allows pages to have writable access from some processes and not from others.

How does Proc with Read only privilege notice that 2nd processor modified data?

The effect of setting bit 9 of a physical address is to put the page in "writethrough" cache status. This means that each time a word in such a page is written, it will be both written into the cache, and sent to memory. Although during ordinary read operations, bit 9 is suppressed from the address sent to the memories, it is included during write operations and the read portions of interlock reads. This is where the "shared pages data integrity" (SPDI) box comes in.

Ordinary memory is limited to 2 to the 26th words (67 Million words). A reference above this limit, (address bit 9 on), refers to the SPDI box. In the case of a "write" operation, the SPDI box if not busy will accept it and re-issue it to the same address, less bit 9. It will then issue a "cache zapper" request to each other active processor in the system for the word (or words) in question. The acceptance by the processor of the cache zapper bus transaction indicates that the receiving processor will delete these words from its cache. Thus, any write in a shared page will soon assure that no old copy resides in any cache.

In the case of an interlock read operation, the SPDI box, if not busy, will accept it, and re-issue it to the same address, less bit 9. It will be issued in the name of the originating processor, so the result will go directly to that processor. The SPDI box will remain busy to all but write release operations. (It is not clear whether this is a "interlock busy" or ordinary "busy" response.) These will be treated the same as write operations explained above.

Explicit interlock instructions (like AOSE) are handled as follows. If an interlock instruction is encountered by the EBOX, it will first issue an interlock-read to the MBOX. This operation implies a write test by the MBOX, since writing privileges are necessary to do this type of instruction. The MBOX will only execute an honest memory interlock function if the page is marked "writethrough" in the page table cache by virtue of bit 9 being one in the physical address, or if the page is marked uncached. Otherwise, if the page is marked "written" already, the read will be done normally from the cache (or memory if necessary). If the page is either marked "writable but not written", or is not in the page table cache, the normal paging algorithm will be clanked, which will cause, in the normal course of events, an interlocked update of the CST. Thus, the first processor attempting to do an interlock on a given word either finds the page already shared, in which case the memory interlock is done, or finds that the page is not shared, or will execute the algorithms above if it determines that this is the transition to shared writable status.

If the page was not shared, then the interlock function is irrelevant. The next processor to attempt to use this interlock will trip over the transition to sharing. A processor which detects the transition to sharing puts the page into shared status before it can read the word, and will find a good copy in memory.

Since the interlock read goes through the SPDI box, the following deadly embrace is avoided: Processor 1 has the interlock and is attempting to release it with a SETOM. Processor 2 is testing the interlock. If processor 2 could do the interlock read around the SPDI box, it could lock-up the memory interlock when the SPDI box was attempting to do the SETOM write. The SPDI box would wait patiently for the interlock to free-up, which it would never do, since the interlock write is camping at the gate for the SPDI box, which is busy.

In order to assure that data modified under the control of an interlock will be seen in correct form by a subsequent owner of the interlock, only one SPDI box can be active in a system. Otherwise, the last data write may still be rattling around the buses after the interlock is released and reseized.

Bus repeaters must be prepared to save write requests which have been rejected for interlock reasons, and let other requests go by, retrying the interlocks occasionally. The space in the bus repeater buffer used by an interlock request must be saved until the write release comes by, or else a deadly embrace similar to the one described above regarding the SPDI box may occur. Thus the bus repeater may be able to accept only write release operations at certain times.

I currently believe that this scheme hangs together. The loose ends are:

1. How the messages to flush page table caches and sweep data caches are handled

2. How KI paging makes use of all this, since a CST is needed.

3. What kind of busy responses to send from the SPDI box.

I solicit your comments.

```
+---------------+
! d i g i t a l !   I N T E R O F F I C E   M E M O R A N D U M
+---------------+
```

TO:	Dolphin Project List	DATE:	20 Jul 78
	Jim Fleming	FROM:	Alan Kotok
	Peter Hurley	DEPT:	LCEG
	Ron McLean	DTN #:	231-6381
		LOC:	MR1-2/E47
	****REVISION 1****	FILE:	SHARED2

SUBJ: Shared Pages in Multiprocessor Dolphin Systems

This memo is to clarify and refine the scheme for handling shared writable pages in multiprocessor Dolphin systems which was the subject of the John Allen memo of 16 Jun 78 (file: MBOX4). It also represents a revision and improvement on the scheme described in my memo of 18 Jul 78 (file: SHARED). This new scheme includes suggestions made by Mike Newman and a brainstorming session in my office with Bosack, Guglielmi, Allen, Lewine, McLean and Murphy.

The problem to be solved is that of shared writable pages. These are defined as pages which have at least one processor which has written the page, and at least one other processor making any references to the page. This state is to be carefully distinguished from the state where many processors have the privilege of writing, but none has yet done so, or the case where only one processor has yet exercised the option of referencing the page, even though several have the privilege.

It is a goal of the system to insure that words written by one processor will be seen by the other processors after a "short" delay, even though for speed reasons, these pages wish to be cached by all processors. Unless an interlock instruction (such as AOSE) is used, no guarantee can be made that simultaneous updating of a word will work, or that one processor will "instantaneously" see the results of another's writing.

To arrange that data integrity is maintained by the hardware (and microcode), it is first necessary to detect that a page has become shared writable. This is done with the help of several bits in the Core Status Table (CST) for the page. The bits in the CST are one for each processor in the system, currently a maximum of 4; along with the existing "modified" bit. Each "processor" bit is a one if that processor has made any reference to the page. A bit of the physical addressing space is used in conjunction with an optional "Shared Pages Data Integrity" box, which goes on the Dolphin Bus. The function of this bit (bit 9) will be explained below.

Each time a change or update is made to a processor's page table cache, or a page table cache entry is encountered with its CST update bit on, a reference is made to the CST. This reference is an interlocked read-write sequence. A physical page which is already shared-writable is one in which more than one processor reference bit is set, and the "modified" bit is set. In this case, the microcode would

1. Or-in bit 9 into the physical address stored in the page table cache for this entry.

2. Or-in a one bit into its processor use bit of the CST word, and write it back.

There are two cases in which a page needs to be changed to shared writable status. The first is that the CST word as read had the "modified" bit set, and exactly one use bit set, but not the bit for the processor doing the update. This is the case of a page which has been written by another processor, and not yet shared. In this case, the following actions are taken:

1. During the CST update, no modification is made to the CST word.

2. A page fault is taken to the software.

3. The monitor goes thru a CST update interlock and looks to see if the CST "age" field is set to a combination called "transition state", (which is detectable by microcode to block references to the page).

   If so, someone else got here first, and no further action is necessary. Release the CST interlock and skip to step 5.

   If not, set the transition state, write back the CST word, and release the interlock.

4. A request is issued to the processor whose reference bit was on in the CST. This request causes that processor to do the following items:

   1. Issue a cache sweep to validate core on the subject physical page.

   2. Issue a new instruction which scans the page table cache, setting the CST update bit on all entries pointing to the object physical page.

   3. Upon completion of both these items, doing a CST update, changing the "age" field to the current time, and oring-in the processor use bit of the processor which initiated this request.

5. Go off and do something else until a check of the CST word for the object page returns to legal state.

When the processor detecting the above transition is able to resume the process causing the fault, correct data for the object page will all be in main memory, and a mechanism described below will insure continued integrity. Since no successful reference was made to the object page before the above sequence was executed, the first retry of the failed reference will cause a CST update reference to be made, where it will be determined that the page is now shared writable.

A third processor tripping over the same problem as above would either have its microcode CST update find the page in "transition state", and just wait for it to become unblocked, or might just catch the page after the second processor detected the transition and before it set the "transition state". In that case, the third processor will hit the CST update interlock mentioned above, and will either take over the transition action, or yield to the second processor, and wait.

If a processor attempts a CST update and finds that it is the first to attempt to write a page that at least one other processor is reading, the transition to shared writable is much simpler. The microcode simply ors-in the processor reference bit and the modified bit, and writes back the CST word. It then sets bit 9 in its page table cache entry, just as if it had discovered that the page was shared writable before it came along.

This works because any copies of words from this page in any cache must be correct, since no one had modified them before, and any subsequent modifications will be handled by the "shared pages data integrity" box. Processors not having write privileges on this page, or not yet having written the page do not set bit 9 of their address, and do not set "written" in their page table cache. This distinction allows pages to have writable access from some processes and not from others.

The effect of setting bit 9 of a physical address is to put the page in "writethrough" cache status. This means that each time a word in such a page is written, it will be both written into the cache, and sent to memory. Although during read operations, bit 9 is suppressed from the address sent to the memories, it is included during write operations. This is where the "shared pages data integrity" (SPDI) box comes in.

Ordinary memory is limited to 2 to the 26th words (67 Million words). A write reference above this limit, (address bit 9 on), refers to the SPDI box. It will accept the write operation, if not busy, and re-issue it to the same address, less bit 9. It will then issue a "cache zapper" request to each other active processor in the system for the word (or words) in question. The acknowlegement by the processor of the cache zapper bus transaction indicates that the receiving processor will delete these words from its cache. Thus, any write in a shared page will soon assure that no old copy resides in any cache.

Explicit interlock instructions (like AOSE) are handled as follows. If an interlock instruction is encountered by the EBOX, it will first issue an interlock-read to the MBOX. This operation implies a write test by the MBOX, since writing privileges are necessary to do this type of instruction. The MBOX will only execute an honest memory interlock function if the page is marked "writethrough" in the page table cache by virtue of bit 9 being one in the physical address, or if the page is marked uncached. Otherwise, if the page is marked "written" already, the read will be done normally from the cache (or memory if necessary). If the page is either marked "writable but not written", or is not in the page table cache, the normal paging algorithm will be clanked, which will cause, in the normal course of events, an interlocked update of the CST. Thus, the first processor attempting to do an interlock on a given word either finds the page already shared, in which case the memory interlock is done, or finds that the page is not shared, or will execute the algorithms above if it determines that this is the transition to shared writable status.

If the page was not shared, then the interlock function is irrelevant. The next processor to attempt to use this interlock will trip over the transition to sharing. A processor which detects the transition to sharing puts the page into shared status before it can read the word, and will find a good copy in memory.

I currently believe that this scheme hangs together. The loose ends are:

1. How the messages to flush page table caches and sweep data caches are handled
2. The location and number of SPDI boxes
3. How KI paging makes use of all this, since a CST is needed.

I solicit your comments.

0B504E7B

```
+---------------+
! d i g i t a l !   I N T E R O F F I C E   M E M O R A N D U M
+---------------+
```

TO:	Dolphin Project List Jim Fleming Peter Hurley	DATE:	18 Jul 78
		FROM:	Alan Kotok
		DEPT:	LCEG
		DTN #:	231-6381
		LOC:	MR1-2/E47
		FILE:	SHARED

SUBJ: Shared Pages in Multiprocessor Dolphin Systems

This memo is to clarify and refine the scheme for handling shared writable pages in multiprocessor Dolphin systems which was the subject of the John Allen memo of 16 Jun 78 (file: MBOX4).

The problem to be solved is that of shared writable pages. These are defined as pages which have at least one processor which has written the page, and at least one other processor making any references to the page. This state is to be carefully distinguished from the state where many processors have the privilege of writing, but none has yet done so, or the case where only one processor has yet exercised the option of referencing the page, even though several have the privilege.

It is a goal of the system to insure that words written by one processor will be seen by the other processors after a "short" delay, even though for speed reasons, these pages wish to be cached by all processors. Unless an interlock instruction (such as AOSE) is used, no guarantee can be made that simultaneous updating of a word will work, or that one processor will "instantaneously" see the results of another's writing.

To arrange that data integrity is maintained by the hardware (and microcode), it is first necessary to detect that a page has become shared writable. This is done with the help of a bit in the physical address of the page, and several bits in the Core Status Table (CST) for the page. A one in bit 9 of the physical page number defines a page which is shared writable. The MBOX of the Dolphin will make use of this bit, as will an optional "Shared Pages Data Integrity" box, which goes on the Dolphin bus. The bits in the CST are one for each processor in the system, currently a maximum of 4; along with the existing "written" bit. Each "processor" bit is a one if that processor has made any reference to the page.

When a processor brings the page pointer into its page table cache, an interlocked reference is always made to the CST. If bit 9 of the physical page number is a zero, and either the "written bit is on or this processor is attempting to set the written bit, and any other processor reference bits are on, then the page is transitioning to the shared writable state. If bit 9 of the physical address was a one, the page is already shared writable.

In all other cases, the processor or's in its reference bit, and written bit, if appropriate, and writes back the CST word. Bit 9 of the bus physical address is forced to zero by the MBOX on all read references.

When the microcode of a processor detects the transition to a shared writable page, it causes a page fault interrupt to the monitor. If the cause of the transition is that this processor is attempting to reference a page which has been written by another processor, the following actions must be taken:

1. During the CST update, the reference bit for this processor is ored in.

2. A one in bit 9 of the physical page address must be ored into the SPT entry pointing to this page.

3. In general, only one other processor reference bit will have been found to be on in the CST when this processor's bit was ored in. It is, however, possible for a third processor to find both the original reference bit and the second processor's reference bit on. In the case where more than one other reference bit is found on, the following step need not be taken, since it will be done by the first processor to detect the transition. The step is to send a message (by yet undefined means) to the other processor, which has written the page, to flush its page table cache, and do a cache sweep on the page, validating core. It is necessary to sweep the entire page table, since arbitrary virtual pages can point to the same physical page.

4. The processor receiving the "flush" message must confirm the completion of the sweep to the asking processor, but need not stop its own processing during this time.

5. The asking processor must always flush its page table cache, and wait for the confirmation of the sweep, if asked for.

The procedure to follow when a processor discovers that it is the first to write an already shared-for-reading page is as follows:

1. During the CST update, ones are ored into both the processor reference bit and the written bit.

2. The old copy of the CST entry would be saved by the microcode to allow the page fault software to distinguish this case from the previous one.

3. A one is ored into bit 9 of the physical page address in the SPT.

4. Messages are sent to all processors having "one" in their reference bits telling them to flush their page table caches.

5. This processor must flush its page table cache.

6. No cache sweeps are necessary since all data in both core and caches are correct.

7. Processing in this processor can proceed when all the page table flushes are confirmed.

8. Another processor detecting this state more-or-less simultaneously cannot in fact do so, since the oring-in of the "written" bit is done by an interlock operation. It would detect the situation illustrated above, where it is the third (or fourth) processor to attempt reading of an already written page.

The effect of setting bit 9 of a physical address is to put the page in "writethrough" cache status. This means that each time a word in such a page is written, it will be both written into the cache, and sent to memory. Although during read operations, bit 9 is suppressed from the address sent to the memories, it is included during write operations. This is where the "shared pages data integrity" (SPDI) box comes in.

Ordinary memory is limited to 2 to the 26th words (67 Million words). A write reference above this limit, (address bit 9 on), refers to the SPDI box. It will accept the write operation, if not busy, and re-issue it to the same address, less bit 9. It will then issue a "cache zapper" request to each other active processor in the system for the word (or words) in question. The acknowlegement by the processor of the cache zapper bus transaction indicates that the receiving processor will delete these words from its cache. Thus, any write in a shared page will soon assure that no old copy resides in any cache.

Explicit interlock instructions (like AOSE) are handled as follows. If an interlock instruction is encountered by the EBOX, it will first issue an interlock-read to the MBOX. This operation implies a write test by the MBOX, since writing privileges are necessary to do this type of instruction. The MBOX will only execute an honest memory interlock function if the page is marked "writethrough" in the page table cache by virtue of bit 9 being one in the physical address, or if the page is marked uncached. Otherwise, if the page is marked "written" already, the read will be done normally from the cache (or memory if necessary). If the page is either marked "writable but not written", or is not in the page table cache, the normal paging algorithm will be elanked, which will cause, in the normal course of events, an interlocked update of the CST. Thus, the first processor attempting to do an interlock on a given word either finds the page already shared, in which case the memory interlock is done, or finds that the page is not shared, or will execute the algorithms above if it determines that this is the transition to shared writable status.

If the page was not shared, then the interlock function is irrelevant. The next processor to attempt to use this interlock will trip over the transition to sharing. A processor which detects the transition to sharing puts the page into shared status before it can read the word, and will find a good copy in memory.

I currently believe that this scheme hangs together, with the loose ends being how the messages to flush page table caches and sweep data caches are handled, and the location and number of SPDI boxes. I solicit your comments.