2.4 Pixel-planes 4 Chip Design

Principal Author: John Poulton

Microelectronic Systems Laboratory

The University of North Carolina at Chapel Hill
Department of Computer Science
New West Hall 035 A
Chapel Hill, N.C. 27514

Table of Contents

2.4. Pixel-planes 4 Chip Design

Introduction

This document describes the design of the enhanced memory chips which form the heart of the Pixel-planes Smart Frame Buffer (Section 2.2). The sections of the document describe:

(1) The external interface to the chip, listing signal names for the two major functions--image generation and video scanout.

(2) Modular decomposition of the chip into five top-level design modules is described. Signals input to and output from each module are listed, and a brief English-language description of each module presented.

(3) The five top-level design modules in the chip are described in more detail. Logic schematics are provided for each module at each level of decomposition.

(4) The actual implementation of the five top-level modules is defined to the transistor level, and leaf-cell designs are fully described, both geometrically and functionally.

2.4.1. External Interface Signals

External connections to the chip are conveniently partitioned into two categories: signals that belong to the **image generation** processing circuits and signals that are associated with **video scan-out**. Table 2.4.1 lists the external interface signals by category.

There are 49 signal pins listed in table 2.4.1. There may be more than one **Vdd** and more than one **GND** pin on any given version of the Pixel-planes 4 memory chip. In nMOS versions, there is likely to be one **VHot** pin for each of the two clocks.

Table 2.4.1: External interface signal for Pixel-planes 4.

Image Generation	
Ph	The image-generation clock
ADatHLPR	"A" data stream to multiplier
BDatHLPR	"B" data stream to multiplier
CDatHLPR	"C" data stream to multiplier
LSBHLPR	LSB control for multiplier pipelining
TrStHLPR	Tree-step control
AGtsTHLPR **AGtsSHLPR** **ACmpHLPR**	Controls for a-port of ALU
BGtsMHLPR **BGtsEHLPR** **BCmpHLPR**	Controls for b-port of ALU
CGtsCHLPR **CCmpHLPR** **CNewHLPR**	Controls for c-port and carry register of ALU
LdEnHLPR	Enable Register load
MWrt<0:1>HLPR	Read/write control for pixel memory
Add<0:6>HLPR	7 pixel memory addresses
Video Scan-Out	
Px	The video scan-out clock
VDat<0:7>HEX1	Video data outputs
BAdd<0:3>HLXR	4 byte address lines for video scan-out selector
PAdd<0:5>HLXR	6 pixel address lines for video scan-out selector
ShLdHLXR	Shadow-Register unload control
ScShHLXR	Scan-path shift enable
ScCnt<0:1>HLXR	2 scan-path function controls
ScInHLXR	Scan-path input
ScOutHSXR	Scan-path output
Power Supplies	
Vdd (vdd)	+5 volt power supply
GND (vss)	Power supply and signal return
Subst	Substrate connection (nMOS version only)
VHot	+8 volt hot-clock power supply (nMOS version only)

2.4.2. Top-Level Design Modules

This section outlines the chip organization in terms of five top-level design modules:

(1) Chip I/O Interface

(2) Multiplier
(3) ALU Array
(4) Memory System
(5) Scan-Path Controller

Figure 2.4.1 shows the five top-level design modules of the enhanced memory chip, and indicates all inter-module signals.

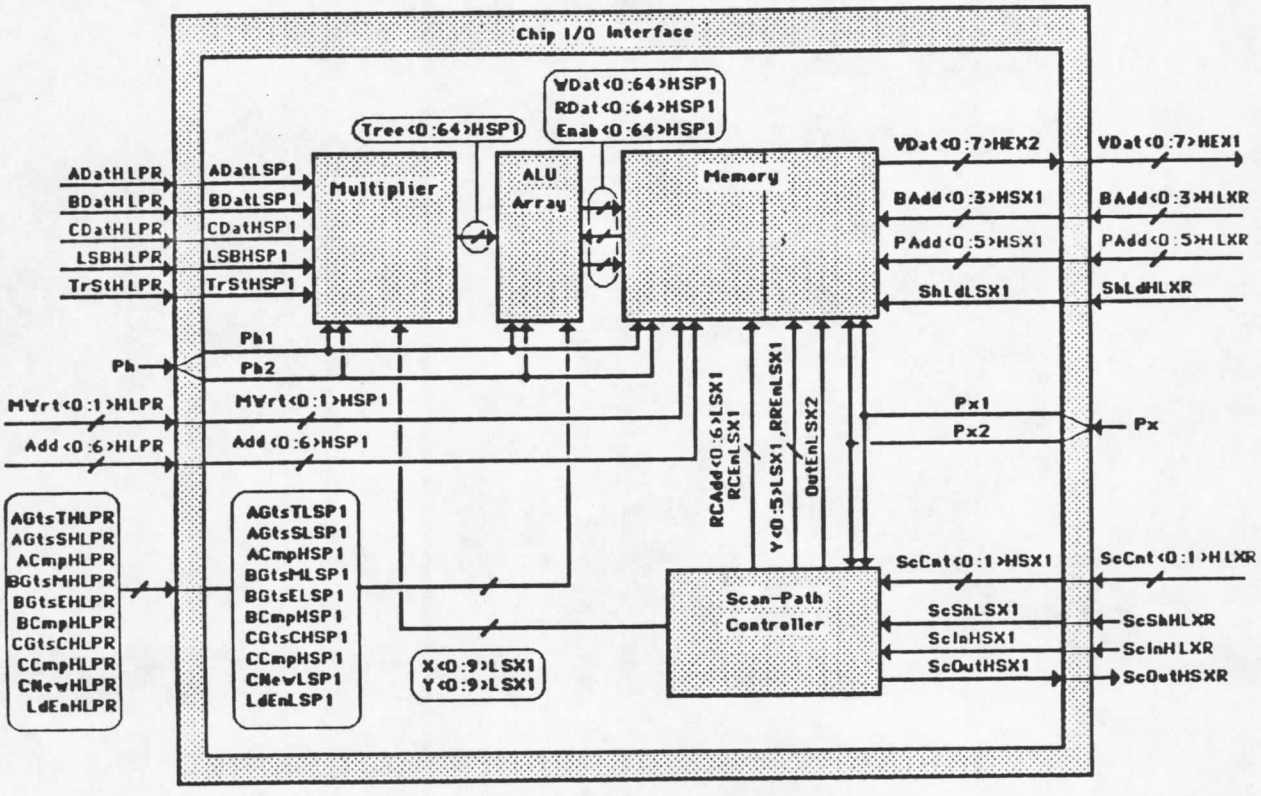


Figure 2.4.1: Top-level design modules and inter-module signals.

2.4.2.1. Chip I/O Interface

This module translates external bus signals to internal chip signals, modifying timing and voltage levels appropriately. The module also converts the external single-phase clocks **Ph** and **Px** to the internal 2-phase, non-overlapping form required by the chip's internal circuitry. Referring to Figure 2.4.1, all signals entering the chip from the left are associated with image generation and are type **LPR**; the chip I/O Interface converts these to type **SP1**. Signals entering and leaving on the right are associated with the scan-path and with video scan-out. The I/O Interface converts type **LXR** signals entering the chip to type **SX1**. The scan-path output **ScOutHSX1** is converted to type **SXR** for output to the system. The video data outputs are precharge-evaluate signals, type **EX1**.

(2) A mechanism that models a read-only port orthogonal to the 64-bit wide image generation port. This mechanism allows a selected byte of a selected row to be output from the chip onto the video data bus via the signals **VDat<0:7>H**. A byte consists of bits from 8 adjacent columns; one of 9 bytes is selected for output by the address **BAdd<0:3>H**. The selected byte is taken from a row (pixel) selected by the address **PAdd<0:5>H**. One of the 65 rows is redundant, selected if **PAdd<0:5>** equals **Y<0:5>** AND **RREnL** is asserted. **Y<0:5>L** and **RREnL** are loaded into the Scan-Path Controller at system configuration time--see below. Memory read/write operations cause the data in the selected pixel to be copied into a 'shadow register' inside the Memory System. This register is double-buffered, and the master can be loaded into the slave by asserting the control **SdhLdHLXR** (Shadow Load). The video outputs **VDat<0:7>H** are enabled only if the signal **OutEnH** is asserted; the signal is generated by the Scan-Path Controller, as described in the next subsection.

2.4.2.5. Scan-Path Controller

In the Pixel-planes Frame Buffer, a serially scanned data path connects all of the enhanced memory chips in the Buffer in a daisy-chained fashion. This serial path is used for four functions, selected by the signals **ScCnt<0:1>**. The Scan-Path Controller controls each chip's access to this scan path. The four functions that can be selected are:

(1) **Test**. The scan path input **ScInH** is shorted to the output **ScOutH** to provide a confidence test for the scan path.

(2) **Alive**. The scan path is connected through each chip's Alive register, providing a way to scan in (and out) a pattern of 1's and 0's. A chip whose Alive register contains a 0 is disabled for output to the video data bus; chips whose Alive registers contain 1's are enabled.

(3) **Configuration**. The scan path is connected through each chip's Configuration register, a 29-bit register that contains the 10-bit x- and 4-bit y-address of the chip's pixel column, the 6-bit address of the redundant row, the 7-bit address of the redundant column, and a two bits containing the redundant row and column Enables. The outputs of this register are brought out to the Multiplier on signals **X<0:9>LSX1** and **Y<0:9>LSX1**, and to the Memory System on signals **Y<0:5>LSX1, RCAdd<0:6>LSX1, RREnLSX1** and **RCEnLSX1**. A pattern of 1's and 0's can be shifted through the system at initialization time; this pattern represents these addresses in bit-serial form, LSB first, as follows: **RCEn, RREn, RCAdd0,... RCAdd6, Y0,... Y9, X0,... X9**. These bits are active-HI on the scan path; the active-LO versions are used internally for convenience in logic design.

(4) **Scan-Out**. The scan path is connected through each chip's Global-Token register. During video scan-out, those chips whose Global-Token registers contain 1's AND whose Alive registers contain 1's are enabled for output to the video data bus, signaled by asserting the signal **OutEnH**. This signal is transmitted to the Memory System from the Scan-Path Controller to enable outputs.

Shifting data or tokens on the scan-path is enabled when **ScSh** is asserted; data and tokens are saved when this control is not asserted.

2.4.3. Gate-Level Module Description

This section outlines in considerable detail the gate-level implementation of each of the five top-level design modules in the Pixel-planes 4 enhanced memory chip together with an English-language description of the function of the module.

The functional specification of each module in the chip is completely specified in terms of an executable computer program written in the C programming language; this code is presented in section 2.4.4. The C-language functional description is also a functional simulator for the chip; it can be combined with functional blocks that describe the board level parts of the Pixel-planes Frame Buffer (Section 2.2), to model a complete working system that generates simulated images. The system functional simulator is fully documented in Section 2.7.

2.4.3.1. Chip I/O Interface

Figure 2.4.2 is a logic diagram of the ChipIO module.

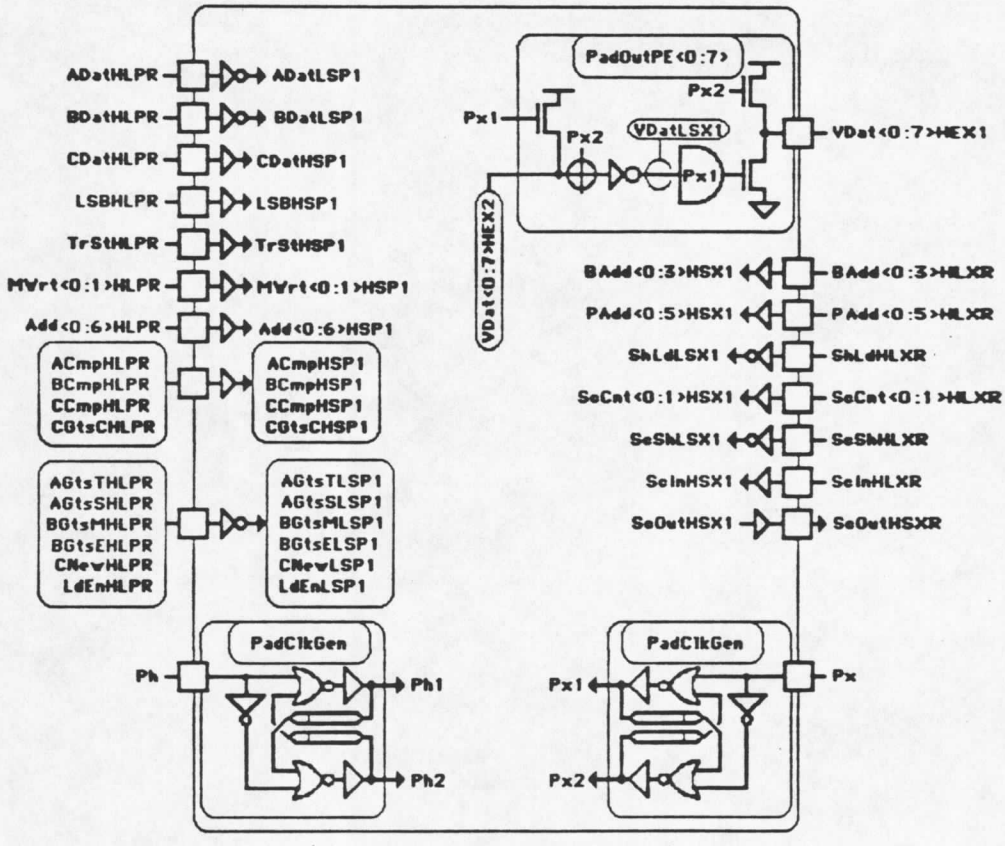


Figure 2.4.2: Chip I/O module logic diagram.

The circuit blocks for the clock generators and for the various types of Input/Output pads are discussed in more detail in Section 2.4.5.

2.4.3.2. Multiplier

Figure 2.4.3 shows the second-level modular decomposition of the Multiplier module.

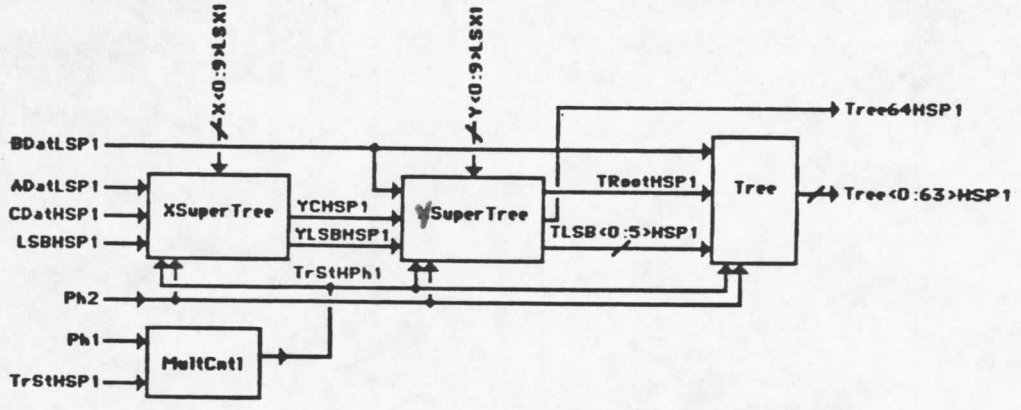


Figure 2.4.3: Second-level modular decomposition of Multiplier.

The modules XSuperTree and YSuperTree are simple serial/parallel multiplier, each having 10 stages. Each of these multipliers is supplied with registers that allow operations in the super-tree and tree to be 'overlapped', as described below.

XSuperTree accepts the **A** and **C** data streams input to the chip and produces the output **YCHSP1** (the 'C' input to the YSuperTree multiplier). This output has the mathematical form $Ax + C$, where x is the x-address of the chip's pixel column. The value of x is passed to the module from the Scan Path Controller by means of the signals **X<0:9>HSX1**. XSuperTree also accepts the chip input signal **LSBHSP1**, which controls the mechanism that allows overlapping operations to occur in the multiplier. The module produces, in turn, the signal **YLSBHSP1** and passes it to the YSuperTree as its overlap-control input.

YSuperTree takes the inputs **BDatLSP1, YCHSP1,** and **Y<0:9>HSX1, and produces the outputs TRootHSP1** and **Tree64HSP1**. **TRoot**, the input to the Tree multiplier's root node, has the form $Ax + By + C$, where $Ax + C$ is that calculated by XSuperTree and By is the value transmitted to the chip on **B** multiplied by the y-address of the chip's pixel column, the four MSB's **Y<6:9>HSP1**. The output **Tree64HSP1** is the value of $Ax + By + C$ for the full 10 bits of **Y**, corresponding to the y-address of the redundant pixel. YSuperTree also produces six signals **TLSB<0:5>HSP1** that control instruction overlapping in the Tree multiplier.

Tree accepts data **TRoot** and **BDatLSP1**, producing the 64 outputs **Tree<0:63>HSP1** for each of the 64 pixels in the chip's column.

The module MultCntl simply generates the qualified clock **TrStHPh1** from **TrStHSP1**.

XSuperTree

Figure 2.4.4 shows the third-level modular decomposition of the module XSuperTree. This module consists mainly of 10 instantiations of the module STStage, a serial multiplier stage with LSB shift register. The figure shows the 10 instantiations abutted, with light lines indicating the boundary between instances. Note that instance #0 is at the right end of the array of instances, so that stage N's multiplicand input **McLSP1** comes from the corresponding x-address input

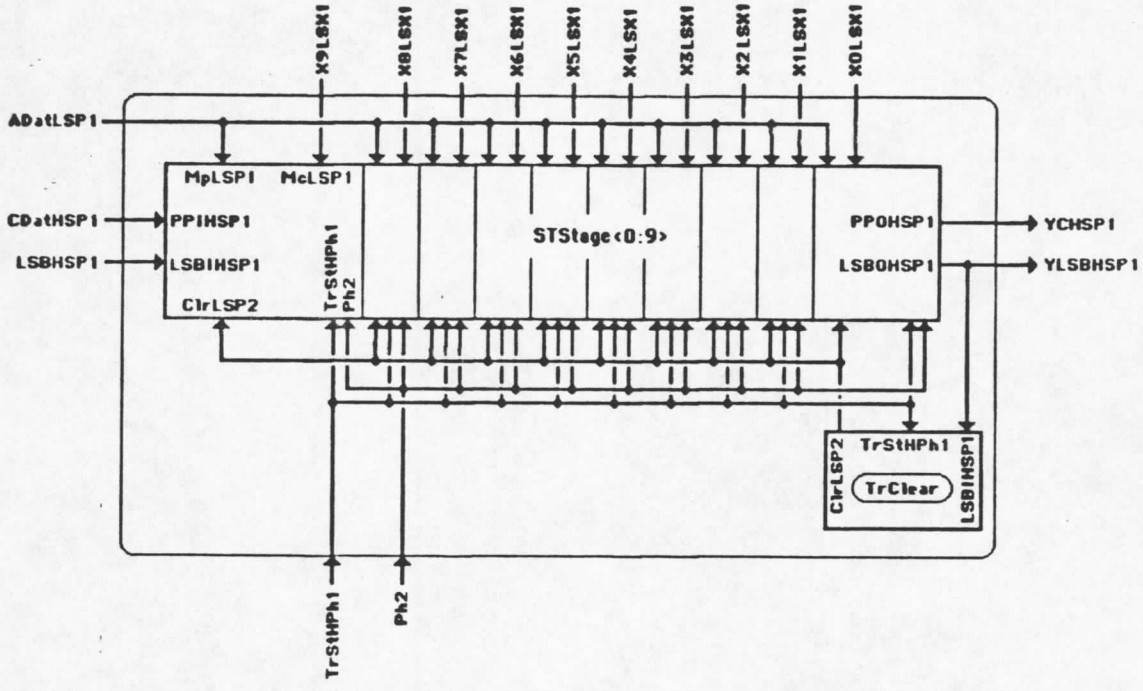


Figure 2.4.4: Third-level modular decomposition of XSuperTree.

X<*N*>HSX1. Abuttment should be taken as connecting the output signals **PPOHSP1, LSBOHSP1** to the inputs **PPIHSP1, LSBIHSP1** of the succeeding stage. The data input **ADatLSP1** is applied to all stages at the **MpLSP1** (multiplier) input.

The module TrClear latches the **LSBOHSP1** of the last STStage on **Ph1** and produces the signal **ClrLSP2**. Timing for the LSB control mechanism is described below.

YSuperTree

The YSuperTree module is essentially identical to the XSuperTree module, with the addition of a number of outputs. Figure 2.4.5 shows the logic diagram of this module.

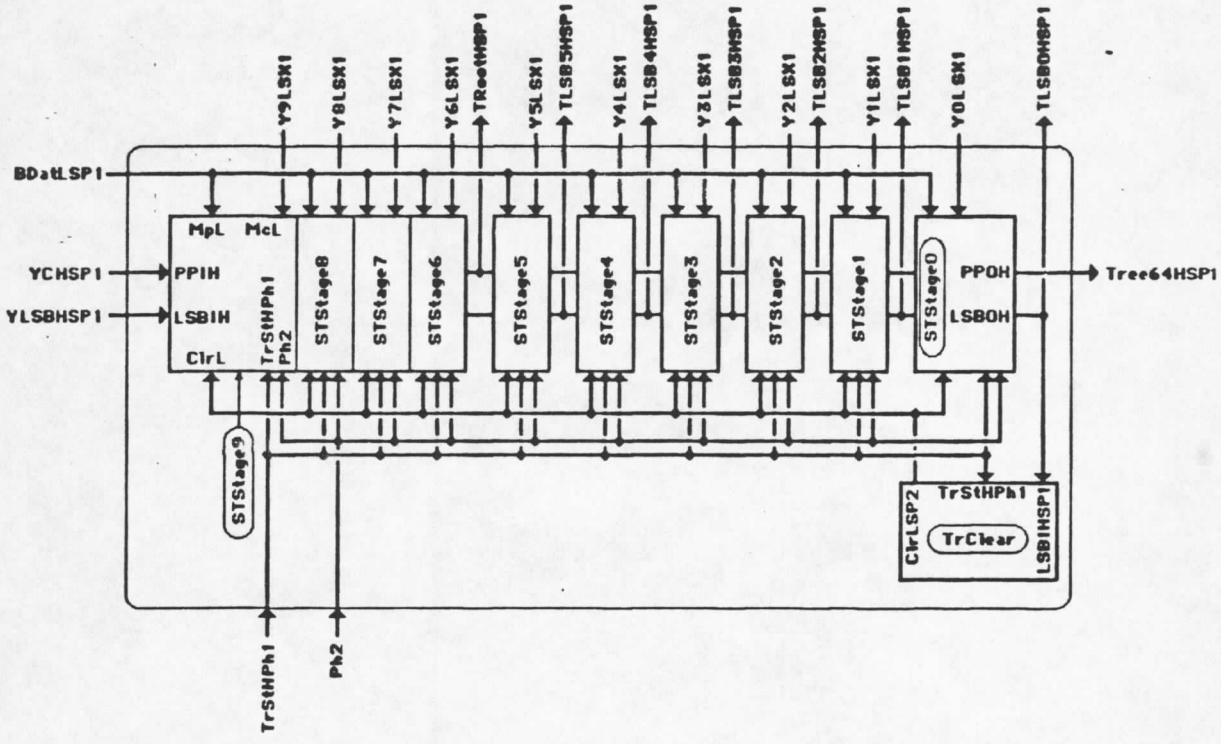


Figure 2.4.5 Third-level modular decomposition of YSuperTree

The module receives as data inputs the signals **YCHSP1** (from the XSuperTree) as the constant input, **BDatLSP1** as multiplicand, and **Y<0:9>HSX1** as the 10 bits of the multiplier. Six LSB control outputs **TLSB<0:5>** are brought out from instances 0 through 5 (respectively) of STStage. These outputs enable overlapping of instructions in the Tree multiplier. STStage6's **PPOHSP1** output is brought out of the YSuperTree module to serve as the root input to the tree, **TRootHSP1**. The multiplier inputs **MpLSP1** in the YSuperTree come from the chip's B data input **BDatLSP1**. The 'C' input to the YSuperTree must have the mathematical value $Ax + C$, carried to the module on the signal **YCHSP1**. The module gets its LSB control input from XSuperTree on **YLSBHSP1**.

Tree

Figure 2.4.6 shows the logic for the Tree module.

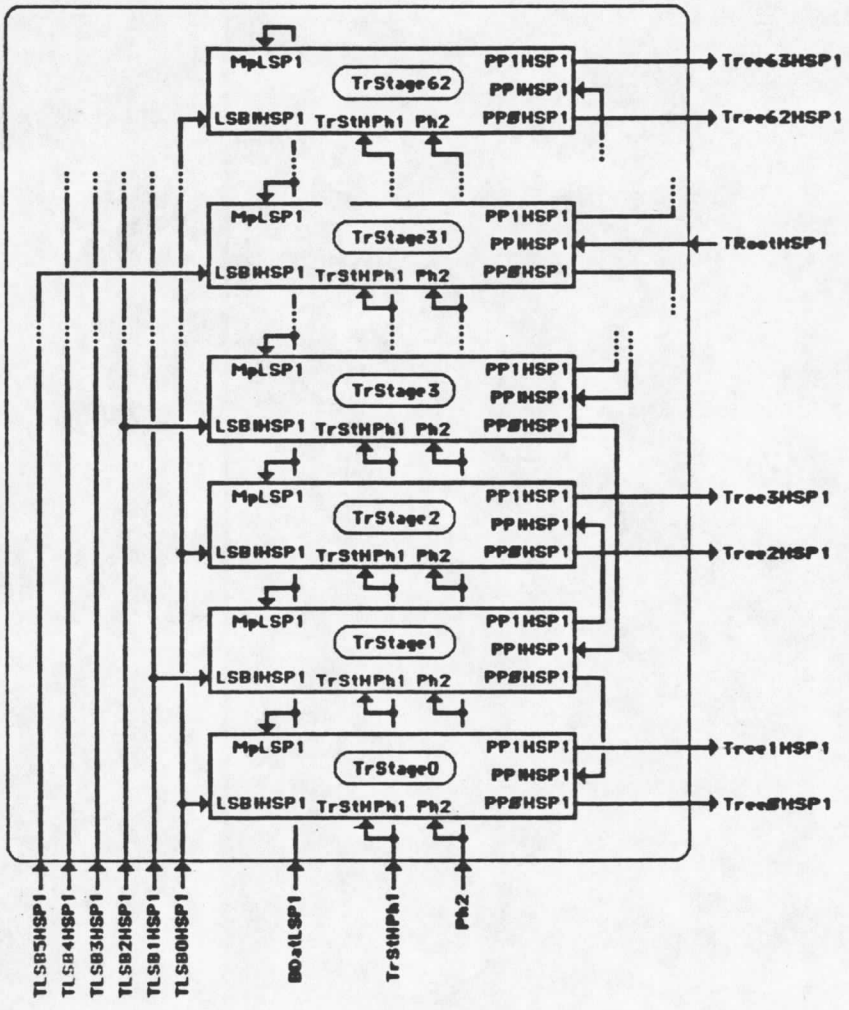


Figure 2.4.6: Third-level modular decomposition of Tree.

The module consists of 63 instantiations of the submodule TrStage, with the data inputs **PP1HSP1** and outputs **PP0HSP1, PP1HSP1** connected in a binary tree The 'flat tree' wiring is evident on the right-hand side of the Figure. All even-numbered instances are leaf-nodes on the tree, generating pairs of signals **Tree<*N*>HSP1** and **Tree<*N+1*>HSP1.**

Leaf Cells

The leaf cells of the multiplier are now defined. The first of these is the control module MultCntl that produces the qualified clock **TrStHPh1**. The circuit for this module is shown in Figure 2.4.7.

Figure 2.4.7: Logic for MultCntl module.

Two leaf cells, STStage and TrClear, are used in XSuperTree and YSuperTree. The multiplier stage STSTage's logic schematic is shown in Figure 2.4.8. This module is a serial/parallel multiplier that accepts a partial-product input **PPIHSP1** and adds a one-bit product of **MpLSP1** (the multiplier) and **McLSP1** (the multiplicand) to form the output partial product **PPOHSP**. The stage produces and stores a local one-bit carry **Cry**.

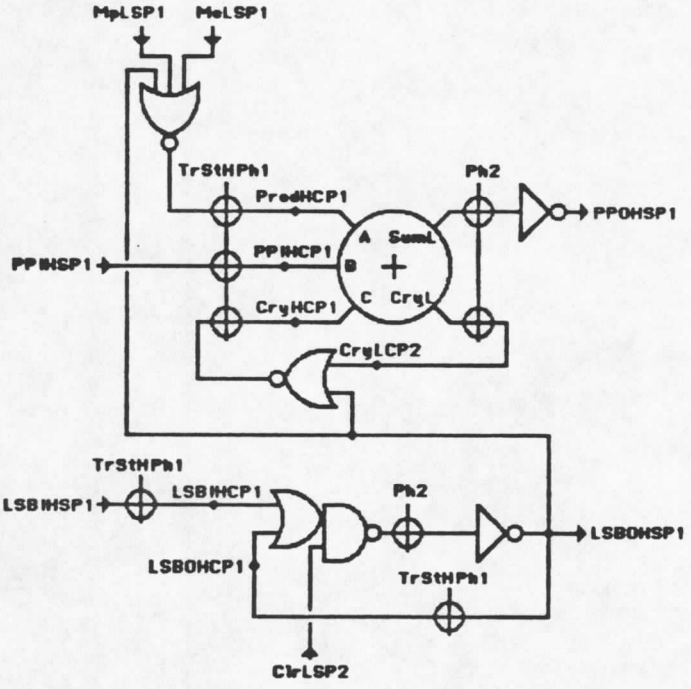


Figure 2.4.8: Logic schematic for STStage module.

A special shift-register structure in the stage allows XSuperTree and YSuperTree to carry out overlapped operations (a new summand can be shifted into the partial-product chain while the MSB's of the old result are being computed just ahead of it). This register is 'sticky' in that, when a logic-1 is received at **LSBHSP1**, on the following cycle **LSBOHSP1**=1 is brought back to the first stage to lock this state. **LSBOHSP1** remains HI until the control **ClrLSP2** is asserted. Multiplier timing is discussed further below.

The TrClear module is shown schematically in Figure 2.4.9.

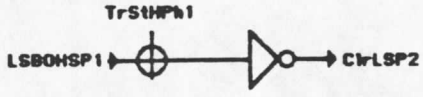


Figure 2.4.9: Logic for TrClear module.

It latches the output **LSBOHSP1** of the final STStage of XSuperTree (and YSuperTree) on **Ph1** and produces the signal **ClrLSP2**.

The Tree module is built entirely of instances of TrStage. The logic schematic for this module is shown in Figure 2.4.10.

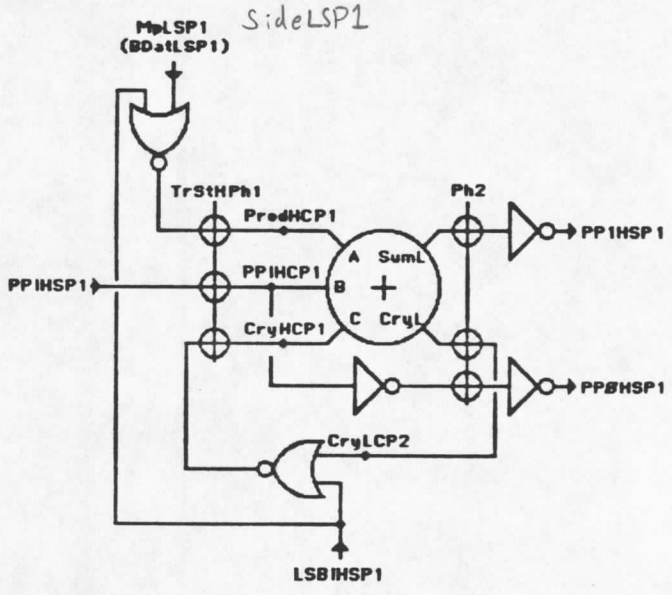


Figure 2.4.10: Logic schematic for TrStage module.

This module closely resembles the STStage module with two exceptions.

(1) There is no LSB control register (this control is taken from the last six stage of YSuperTree).

(2) There is no multiplicand input--the stage computes both partial products for the two possible values of the multiplicand, producing the two outputs **PP1HSP1** (multiplicand=1) and **PP0HSP1** (multiplicand=0).

Multiplier Timing

Figure 2.4.11 shows the timing for control and data in the XSuperTree module. The input **CDatHSP1** is an N-bit serial representation of the C coefficient, and **ADatLSP1** is an M-bit serial representation of the A coefficient. N and M are defined by

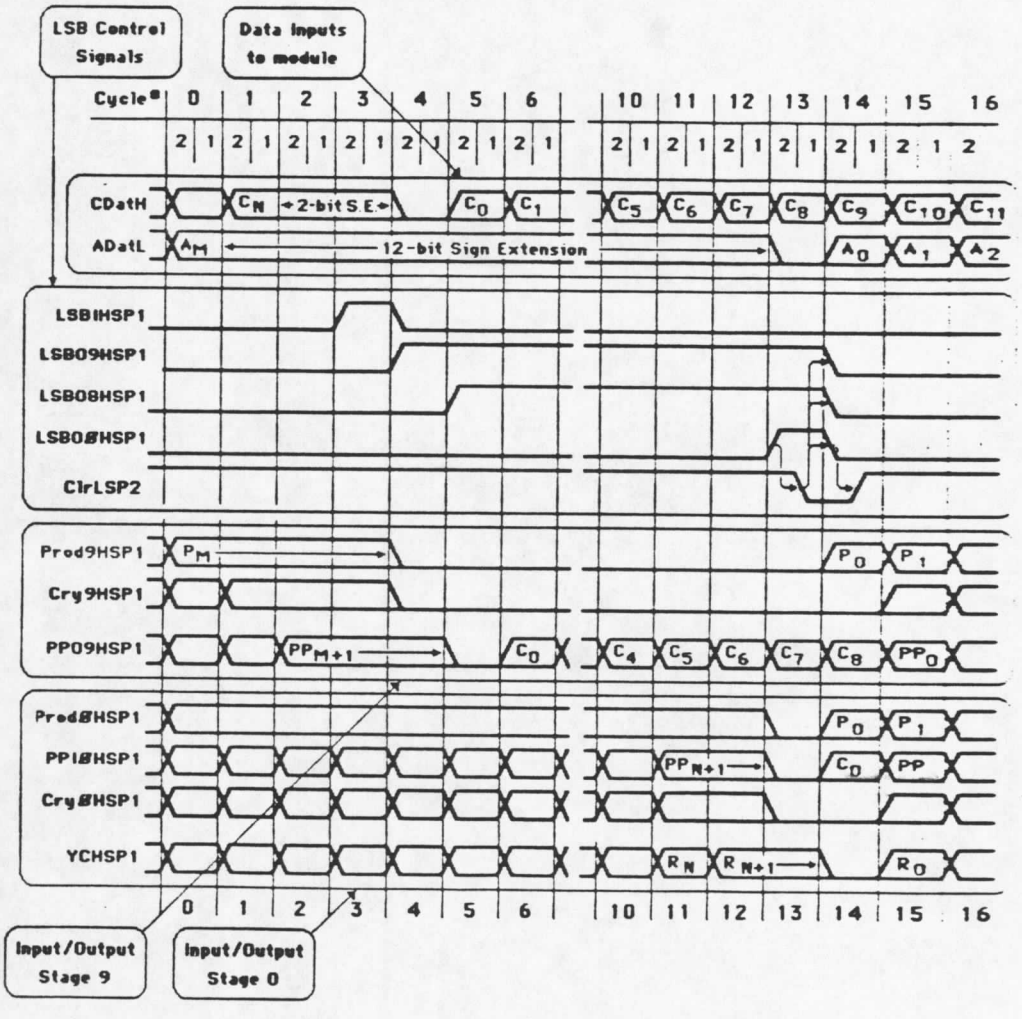


Figure 2.4.11: Timing for XSuperTree multiplier module.

$$N = \mathbf{Max}(\text{Length}(A)+10,\ \text{Length}(B)+10,\text{Length}(C))$$

$$M = N + 10$$

In this discussion, the Nth bit of C and the Mth bit of A are the sign bits in the two's complement representation of each number. To form the correct product $Ax + C$, C must be sign-extended by two bits, while A is sign-extended by 12 bits. The first sign extension bit of C is needed because the result $Ax + C$ may in general overflow, becoming $N + 1$ bits long. The second sign extension is required because $Ax + C$ becomes the 'C' input for the YSuperTree module where it is added to another N-bit long number; again overflow may occur, resulting in an $(N + 2)$-bit number. Both A and C are followed by a 'separator bit', which must be 0. The separator bit is required to allow the LSB control mechanism to clear the carry in stage 0 before each operation. It must be 0 in order that numbers without fractional parts can be handled correctly in the ALU–see section 2.3 for further details about number handling and exceptions.

LSB Control Timing

The LSB control mechanism allows successive Multiplier operations to be carried out with only a single 'separator bit' time wasted. Without this mechanism, roughly 20 bit times would be wasted between operations to clear the carry registers in each stage. Each stage of the Multiplier contains a special LSB shift register that is 'sticky' in the state that outputs a logic-1 from the register. The register's main function is to generate the signal **LSBO<**n**>**HSP1 at each stage n. This signal forces the **ProdH** and **CryH** inputs to the stage's adder LO, so that, on the next cycle

(1) the **PPOHSP1** output of the stage will be the same as the **PPIHSP1** input on the previous cycle (the stage becomes a shifter), and

(2) The **CryHCP1** input to the stage's 'carry register' (the loop connecting **CryL** of the adder around to **C**) will contain a logic-0.

The LSB registers, once set, can be cleared only by the module-global signal **ClrLSP2**.

Referring to Figure 2.4.11, on cycle #3 the LSB control sequence is initiated by asserting **LSBIHSP1** for one cycle. This causes the **LSBO9HSP1** output of stage 9's LSB shifter to go high at the beginning of cycle #4. The LSB shifters are chained, so on cycle #5 stage 8's LSB output **LSBO8HSP1** goes HI, on cycle #6 stage 7's output goes HI, and so forth. Finally on cycle #13, stage 0's output **LSBO0HSP1** goes HI. This signal is input to the TrClear module where it is latched on **Ph1** and produces the signal **ClrLSP2**. On **Ph1** of cycle #13, **ClrLSP2** goes active (LO) causing **LSBO**'s to go LO on cycle #14, completing the LSB control sequence. Causality arrows on the Figure show how these events are linked.

Data Timing

The LSB control sequence is initiated on the micro-cycle just before the separator bit is sent to the C-input of the XSuperTree, cycle #3 in Figure 2.4.11. On cycle #4, stage 9's LSB control register outputs a logic-1, clearing the carry and forcing the **Prod** input LO. On cycle #5, stage 9 therefor passes the LO separator bit through unscathed, and, on succeeding cycles passes the LSB's of C through. Succeeding stages are 'disabled' by the cascading **LSBO** signals on successive cycles, just ahead of the separator bit. At stage 0, the separator bit and the LSB signal arrive together on cycle #13. The separator bit is passed through stage 0 on cycle #14. This cycle begins the next calculation: C_0 is input to stage 0, having been passed through the previous stages, and arrives simultaneously with the LSB of the next A. On cycle #15, the first bit of the result R_0 is output from the module.

Figure 2.4.12 shows the timing for the YSuperTree, which is nearly identical to that in the XSuperTree module. The signal **LSBO0HSP1** output by the XSuperTree becomes the LSB control input for the YSuperTree, **YLSBHSP1**. The 'C' input to the YSuperTree is the result output from the XSuperTree, **YCHSP1**. Note that the timing relation between the YLSBI assertion and the separator bit in YC is the same as that between LSBI and C in Figure 2.4.11.

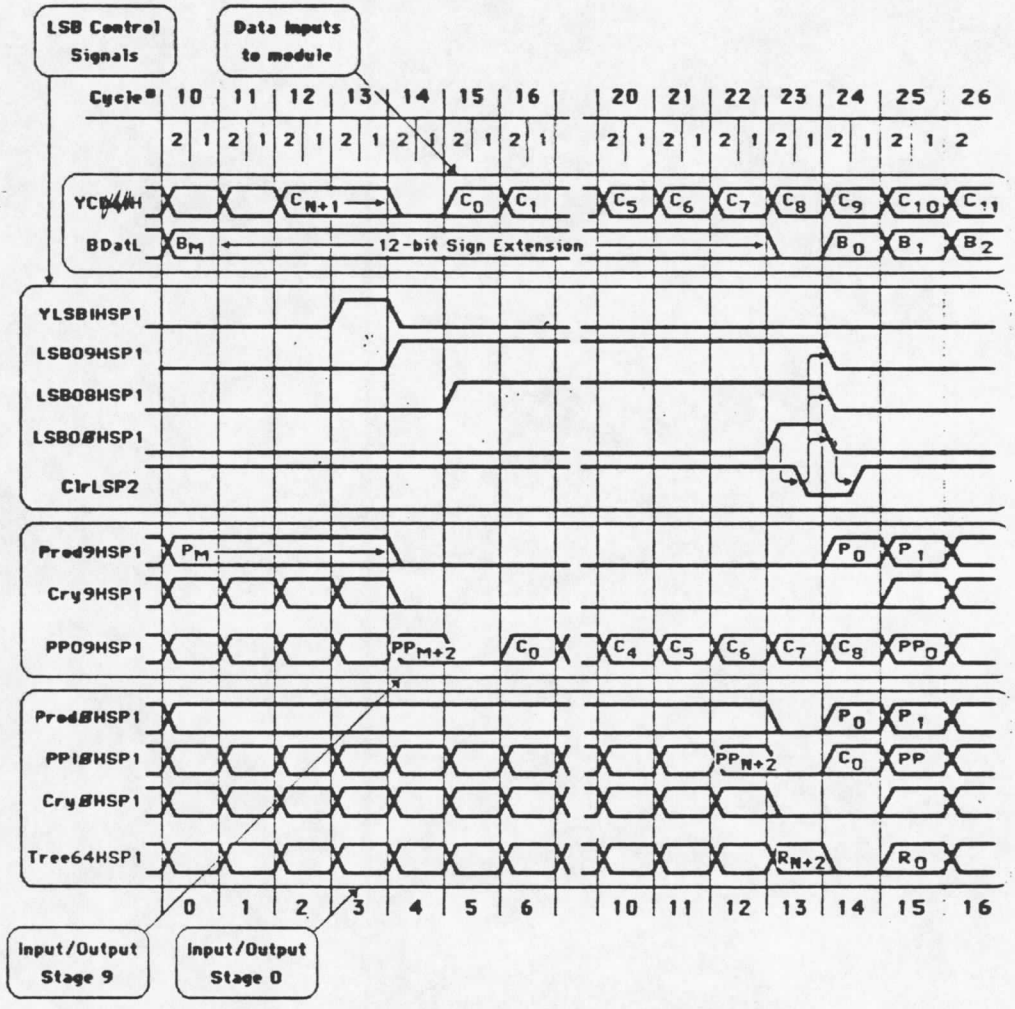


Figure 2.4.12 Timing in YSuperTree multiplier module.

The YC data stream is now $N + 1$ bits long with one sign extension, as described above. The B data stream has its MSB aligned with the $(N-1)$th bit of YC.

2.4.3.3. ALU Array

This section describes the logic-level implementation of the ALU Array module in the Pixel-planes chip. Figure 2.4.13 shows the second-level modular decomposition of this module.

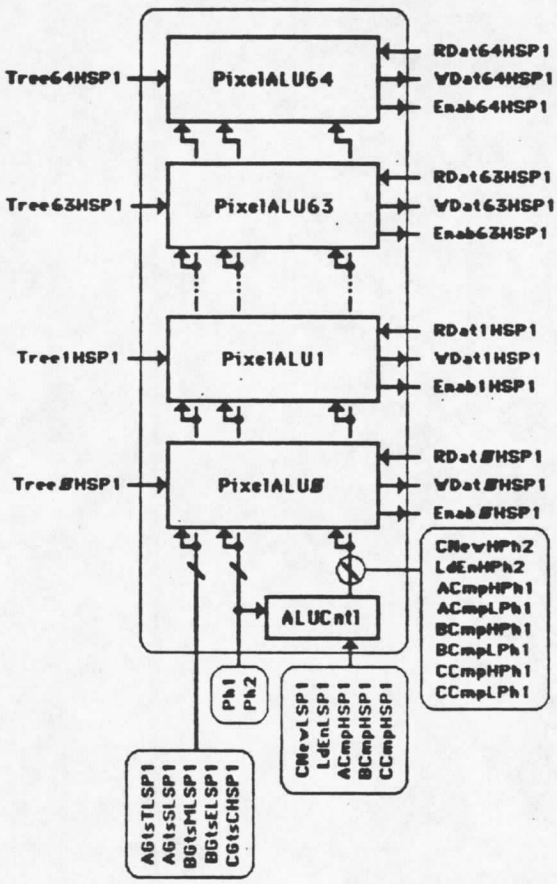


Figure 2.4.13: Logic schematic of ALUArray module.

The module consists of 65 identical instantiations of PixelALU, each of which accept a one-bit data stream from the Multiplier, **Tree*N*HSP1** and a one-bit data stream from the Memory System, **RDat*N*HSP1** and produce the data stream **WDat*N*HSP1** and the control **Enab*N*HSP1** output to the Memory System. ALU function in each instance of PixelALU is selected by the controls **AGtsTLSP1, AGtsSLSP1, BGtsMLSP1, BGtsELSP1, CGtsCHSP1**, and the qualified clocks **ACmpHPh1, ACmpLPh1, BCmpHPh1, BCmpLPh1, CCmpHPh1, CCmpLPh1, CNewHPh2, LdEnHPh2** generated by the ALUCntl module.

The logic for the ALUCntl module is shown in Figure 2.4.14.

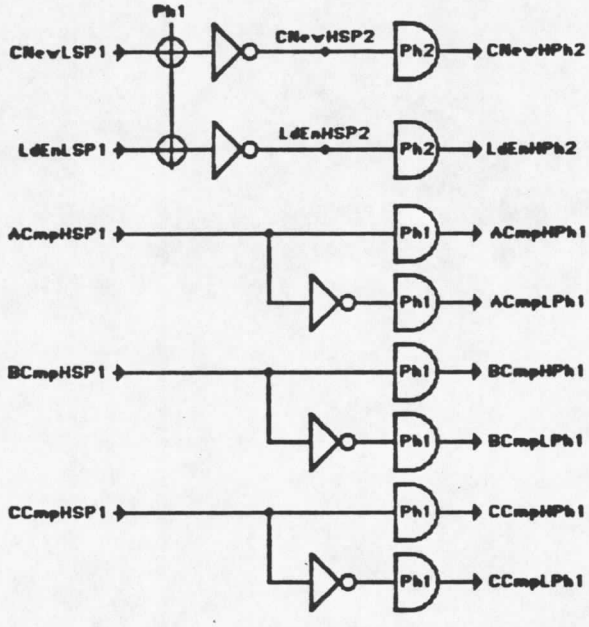


Figure 2.4.14: Schematic of ALUCntl module.

The controls **CNewLSP1** and **LdEnLSP1** are latched into the module on **Ph1**, producing the signals **CNewHSP2, LdEnHSP2**. These in turn qualify the **Ph2** clock to produce **CNewHPh2** and **LdEnHPh2**. The three signals **ACmpHSP1, BCmpHSP1, CCmpHSP1** generate directly the three qualified clocks **ACmpHPh1, BCmpHPh1, CCmpHPh1** and are inverted to generate **ACmpLPh1, BCmpLPh1, CCmpLPh1**.

Figure 2.4.15 shows the logic for the Pixel ALU module.

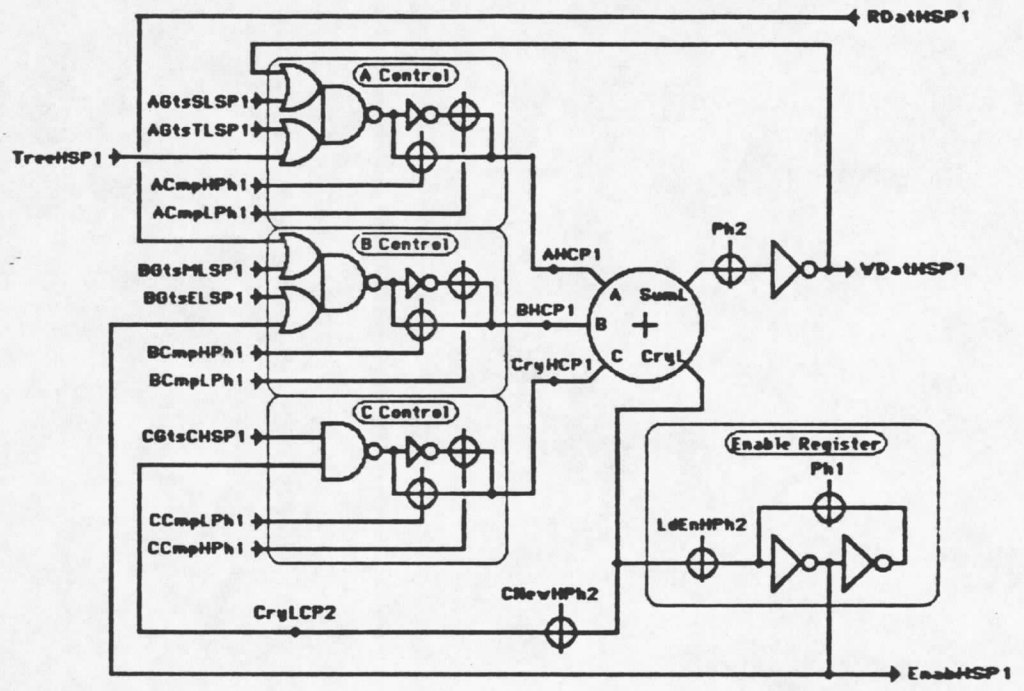


Figure 2.4.15: Schematic of Pixel ALU module.

The module consists basically of a one-bit full adder with carry register, together with a loadable Enable register. Each of the three data input ports of the adder is controlled by a selector/complementer that allows various data sources to be directed to the port, and also allows this data to be complemented or forced to a particular logic value.

The A and B selector/complementers are identical. For the A-port, the data source **TreeHSP1** can be selected by asserting **AGtsTLSP1** ('A gets Tree') or the source **WDatHSP1** can be selected by asserting **AGtsSLSP1** ('A gets Sum'). For the B-port, the sources **RDatHSP1** and **EnabHSP1** are selected using **BGtsMLSP1** ('B gets Memory read data') and **BGtsELSP1** ('B gets Enable'). Data to the ports can be complemented by asserting the control **ACmpHSP1** (or **BCmpHSP1**) which result in generating the qualified clock **ACmpHPh1** (**BCmpHPh1**); otherwise the complementary clocks **ACmpLPh1** and **BCmpLPh1** are generated. Either port can be forced HI (or LO when the complement control is asserted) when neither data source is selected.

The C-port control differs by having only one data source, namely the carry output of the adder. When **CNewLSP1** is asserted, the next **Ph2** causes the node **CryLCP2** to take on the value of the carry output of the adder. When **CNewLSP1** is not asserted, this node is isolated and stores the previous carry value. When **CGtsCHSP1** ('C gets Carry') is asserted, the value of the carry stored on **CryLCP2** is passed into the C-port of the adder. Data can be complemented by asserting **CCmpHSP1** (resulting in the qualified clock **CCmpHPh1**); otherwise **CCmpLPh1** is generated. The C-port can be forced HI or LO by de-asserting **CGtsCHSP1**, just as the A- and B-ports can.

The Enable register is loaded from the carry output port of the adder when **LdEnLSP1** is asserted--on the next **Ph2**, this control causes the qualified clock **LdEnHPh2** to be generated,

Note that from the micro-programmer's point of view, all of the ALU controls are active-HI.

loading the register. The register is refreshed on **Ph1**, but since the data output is never changed during refresh, it is typed **SP1**.

The ALU control signals are usefully grouped in two categories: register controls (**CNew** and **LdEn**) and logic function selection (**AGtsT, AGtsS, ACmp, BGtsM, BGtsE, BCmp, CGtsC, CCmp**). The eight logic function selectors are conveniently divided into three fields associated with each of the three input ports to the adder. Table 2.4.2 shows how each of the three fields operates on its associated port; the table entries for each control function assume active-HI controls (as they would appear from the micro-programmer's point of view), so, for example, an entry that shows AGtsT=1 would imply that **AGtsTLSP1**=0.

Table 2.4.2: Logic function selection in Pixel ALU.

A Field			
AGtsT	**AGtsS**	**ACmp**	**Function**
0	0	0	A=1
0	0	1	A=0
0	1	0	A=Sum
0	1	1	A=!Sum
1	0	0	A=Tree
1	0	1	A=!Tree
1	1	0	A=Sum•Tree
1	1	1	A=!(Sum•Tree)

B Field			
BGtsM	**BGtsE**	**BCmp**	**Function**
0	0	0	B=1
0	0	1	B=0
0	1	0	B=Enab
0	1	1	B=!Enab
1	0	0	B=Mem
1	0	1	B=!Mem
1	1	0	B=Mem•Enab
1	1	1	B=!(Mem•Enab)

C Field		
CGtsC	**CCmp**	**Function**
0	0	C=1
0	1	C=0
1	0	C=Cry
1	1	C=!Cry

The ALU logic function selectors were chosen in such a way that the full one-bit add function, required for various graphic algorithms, can also provide a variety of other Boolean functions. Table 2.4.3 shows the truth table for a full adder.

Table 2.4.3: Truth table for full adder.

A	B	C	Sum	Carry
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Examination of this table shows that the Carry output provides the AND and OR logic functions for the A and B inputs when the C-input is forced HI and LO, respectively. The Sum output provides the XOR and NXOR functions of A and B.

Use of the various logic functions in the ALU is discussed more completely in document 2.3.

2.4.3.4. Memory System

Figure 2.4.16 shows the second-level modular decomposition of the Memory System module in Pixel-planes 4.

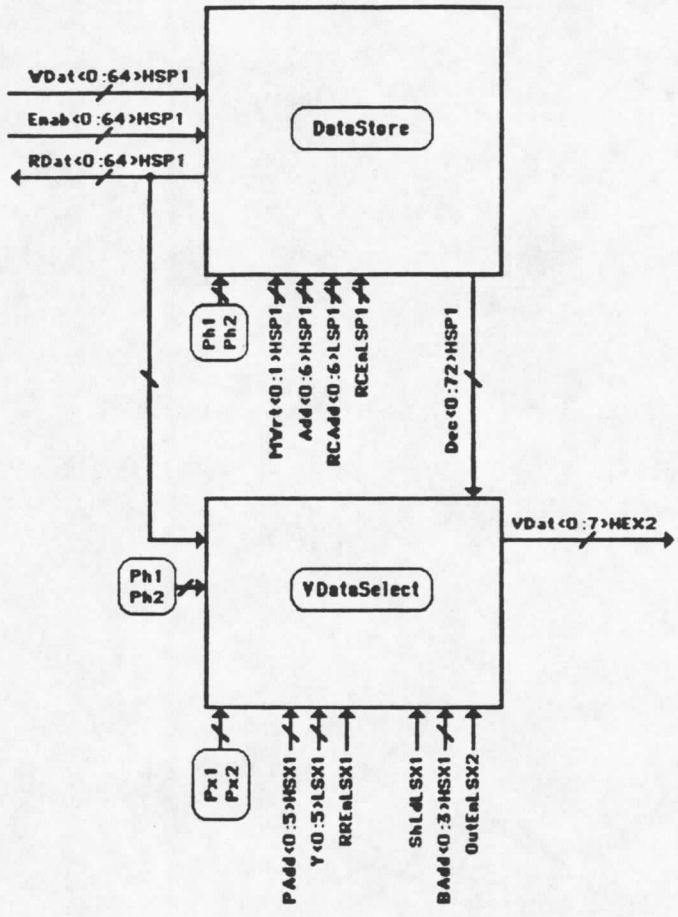


Figure 2.4.16: Second-level modular decomposition of Memory System

The Memory System consists of two sub-modules, DataStore and VDataSelect.

The DataStore module, shown in Figure 2.4.17, contains 65 instantiations of the PixelMemory module, each of which contains the memory and memory-control circuits for a pixel.

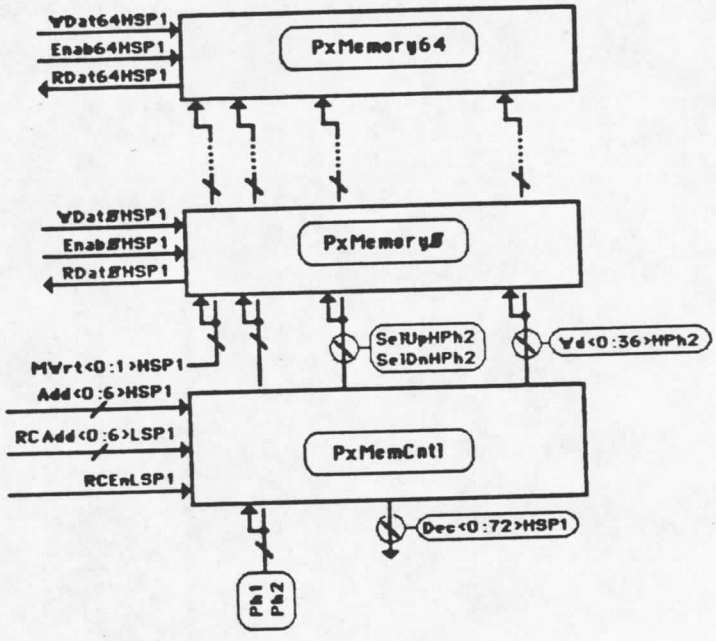


Figure 2.4.17: Third-level modular decomposition of DataStore module.

These sub-modules are controled by PxMemCntl, which contains the column address decoders and redundant column select.

VDataSelect is shown in Figure 2.4.18.

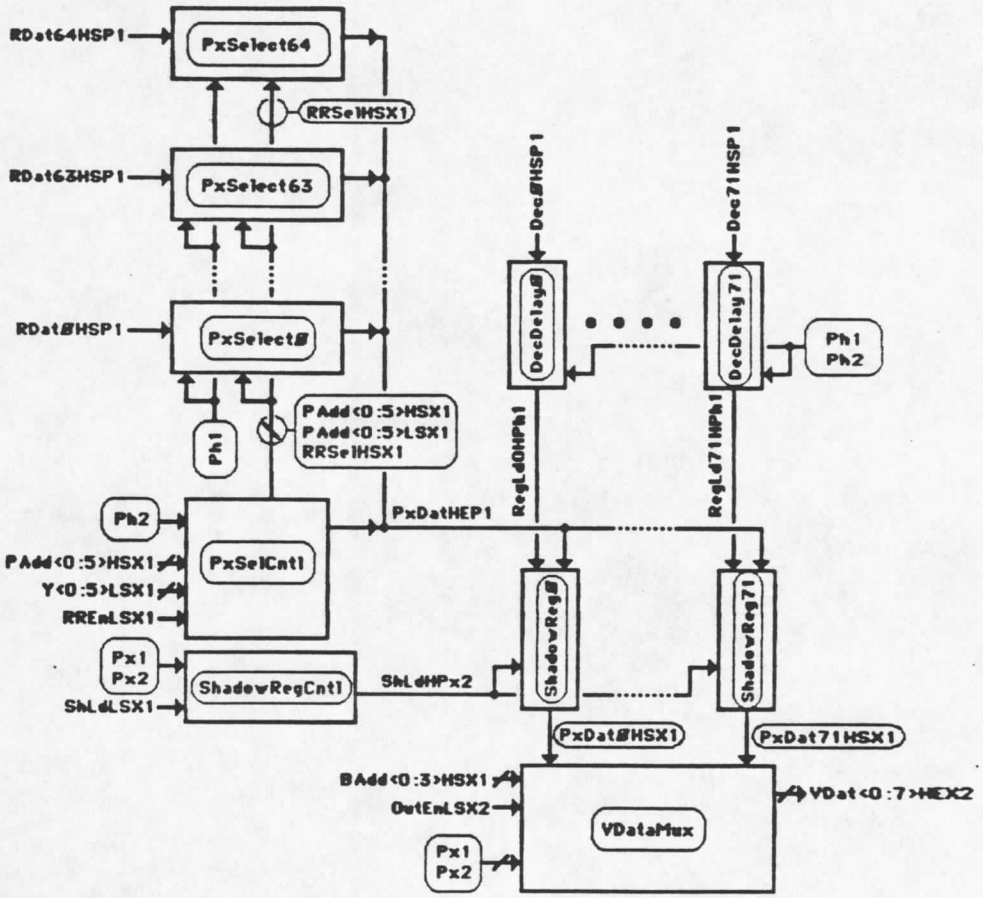


Figure 2.4.18: Third-level modular decomposition of VDataSelect.

The module allows the Video Controller (Section 2.5) to select a pixel whose data will be used to refresh the display screen.

VDataSelect contains three main parts:

(1) A pixel selector, consisting of 65 PxSelect modules and the PxSelCntl controller, decodes the pixel-select address **PAdd<0:5>HSX1**, together with the redundant-row address/control **Y<0:5>LSX1,RREnLSX1**, selecting one of the 65 signals **RDatHSP1** for output to the one-bit bus **PxDatHEP1**.

(2) An ensemble of registers ShadowReg<0:71> receive a copy of the currently selected pixel as follows: On every memory operation, the one-hot address of the selected bit is passed to the ShadowReg ensemble through the ensemble of one-cycle delays DecDelay. The selected ShadowReg is loaded with the value passed to it on **PxDatHEP1**. In this way the currently-selected pixel memory is copied, bit by bit, into the shadow register.

(3) VDataMux decodes the byte address **BAdd<0:3>HSX1** and selects one of the 9 bytes of data stored in the ShadowReg (**PxDat0HSX1...PxDat7HSX1, PxDat8HSX1...PxDat15HSX1**, etc.). If the chip is enabled for output (**OutEnLSX2** is asserted), the module puts this data onto the bus **VDat<0:7>HEX2**.

Memory System Leaf Cells

The leaf-cells of the PixelMemory module are shown in Figure 2.4.19.

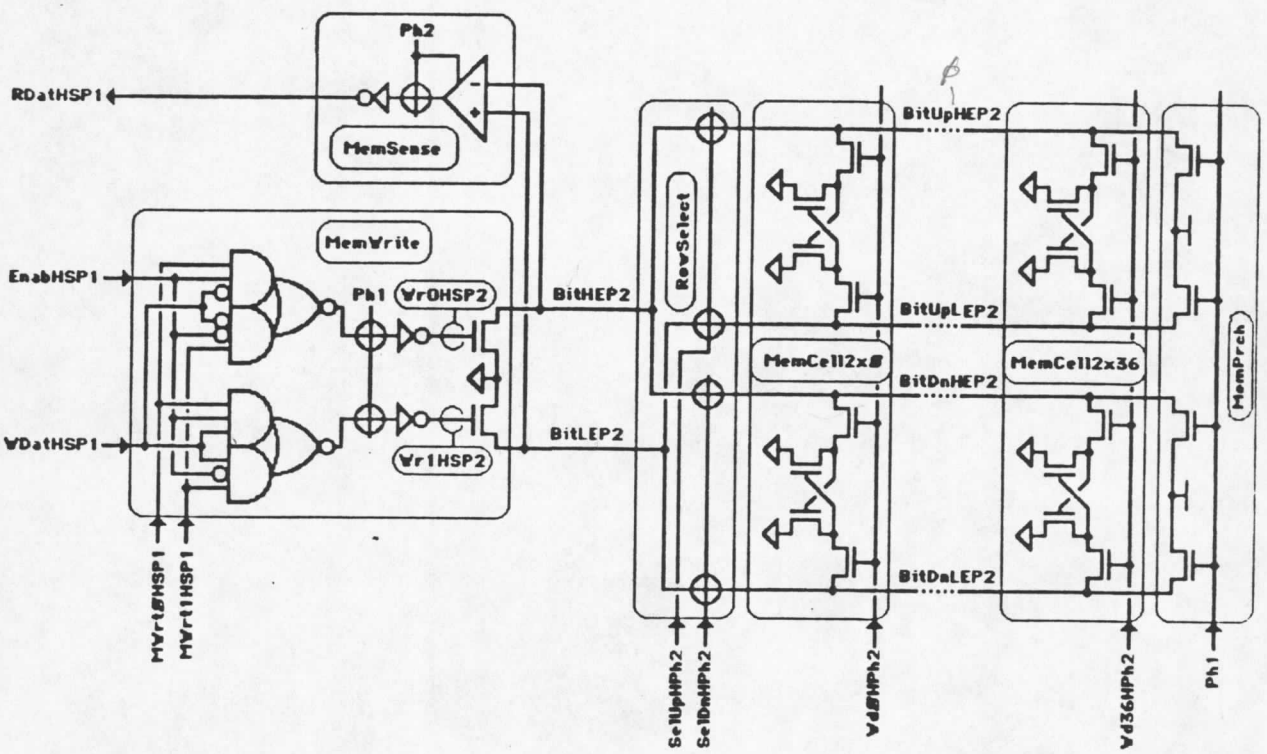


Figure 2.4.19: Schematic of PxMemory module.

PixelMemory contains 74 bits of memory organized as two rows by 37 columns, together with precharge/read/write/ control circuits and a row selector. The module is built up of five submodules:

(1) An array of 37 cells MemCell2x contain the 74 memory bits for the pixel (two bits are in the redundant column). These memory cells are implemented as 4-transistor dynamic RAM's, with symmetric bit lines, **BitUpHEP2,BitUpLEP2** and **BitDnHEP2,BitDnLEP2**, and column select (word) lines **Wd0HPh2**. The word lines are an ensemble of one-hot signals that select one cell in each row for read/write access.

(2) MemPrch causes both pairs of bit lines to be precharged HI during **Ph1**.

(3) RowSelect connects one of the rows to read/write control circuits during **Ph2**; the one-hot signal pair **SelUpHPh2,SelDnHPh2** enables the selection. The selected row is connected to the bit-lines **BitHEP2,BitLEP2** from MemWrite and MemSense modules. The de-

selected row performs a read operation (and gets refreshed) during **Ph2**.

(4) MemWrite provides write-access to the selected memory bit. The module logically combines the memory control signals **MWrt<0:1>HSP1** and the pixel enable **EnabHSP1** to perform the following functions:

MWrt1	MWrt0	Function:
0	0	Read
0	1	Write if Enab
1	0	Write if !Enab
1	1	Write always

To implement these functions, the module generates two signals **Wr0HSP2,Wr1HSP2** that cause one or neither bit line to be pulled LO during **Ph2**. These two signals are logically related to the controls by

$$Wr0 = (!WDat \bullet Enab \bullet MWrt0) + (!WDat \bullet !Enab \bullet MWrt1)$$
$$Wr1 = (WDat \bullet Enab \bullet MWrt0) + (WDat \bullet !Enab \bullet MWrt1)$$

(5) MemSense detects the voltage difference across the bit lines during **Ph2** and latches the value read, producing the read data **RDatHSP1**. Note that **RDat** is valid following memory write operations as well as reads.

Figure 2.4.20 shows the schematic of the PxMemCntl module.

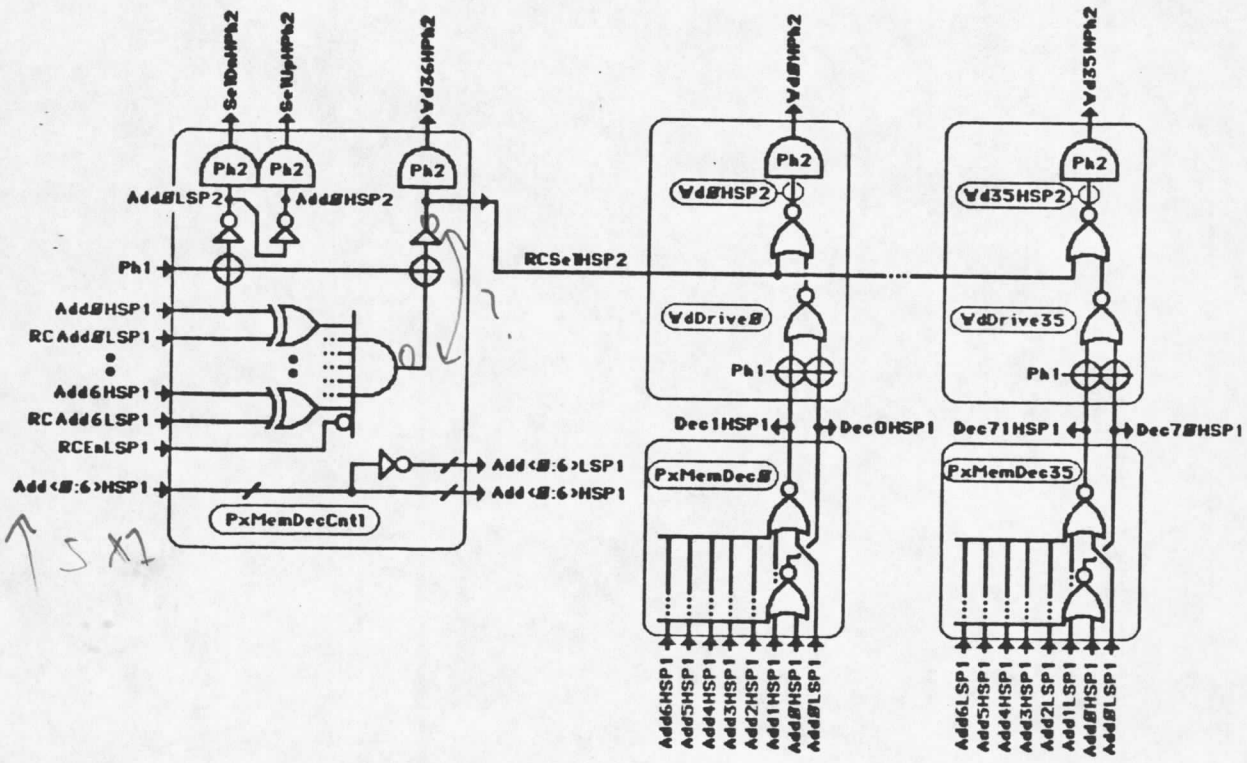


Figure 2.4.20: Schematic of sub-modules of PxeMemCntl module.

This module generates the word-control lines **Wd<0:36>HPh2** and the row selects **SelUpHPh2,SelDnHPh2** for the PixelMemory modules, and also produces the signals **Dec<0:71>HSP1** for enabling the shadow registers. PxMemCntl is built out of three basic submodules:

(1) PxMemDecCntl passes the pixel-bit address **Add<0:6>HSP1** through to the column-select decoders, providing the complements **Add<0:6>LSP1**. It also compares this address bit-by-bit with the address of the redundant column, **RCAdd<0:6>LSP1**. If the redundant column is enabled by **RCEnLSP1**=0, the signals **RCSelHSP2** is asserted; this in turn causes the word-line for the redundant column, **Wd36HPh2**, to be asserted. PxMemDecCntl also produces the signal **SelUpHPh2** when **Add0HSP1**=1 and **SelDnHPh2** when **Add0HSP1**=0 (even addresses select the lower row of each row pair in PixelMemory, while odd addresses select the upper row).

(2) The WdDrive modules produce all column-select signals except the one for the redundant column. Since memory is organized in row pairs, the outputs of two decoders are OR'd in WdDrive, so that **Wd*N*HPh2** is asserted in either **Dec*(2N)*HSP1** or **Dec*(2N+1)*HSP1** is asserted. If the redundant row is selected, **RCSelHSP2** is HI, and all WdDrive modules are disabled.

(3) The modules PxMemDec<0:35> decode the seven addresses **Add<0:6>**, producing **Dec<0:71>HSP1**. These modules are organized in pairs corresponding to the pairing of memory cells in PixelMemory.

Figure 2.4.21 shows the leaf cells of the shadow-register and video data multiplexer in VDataSelect.

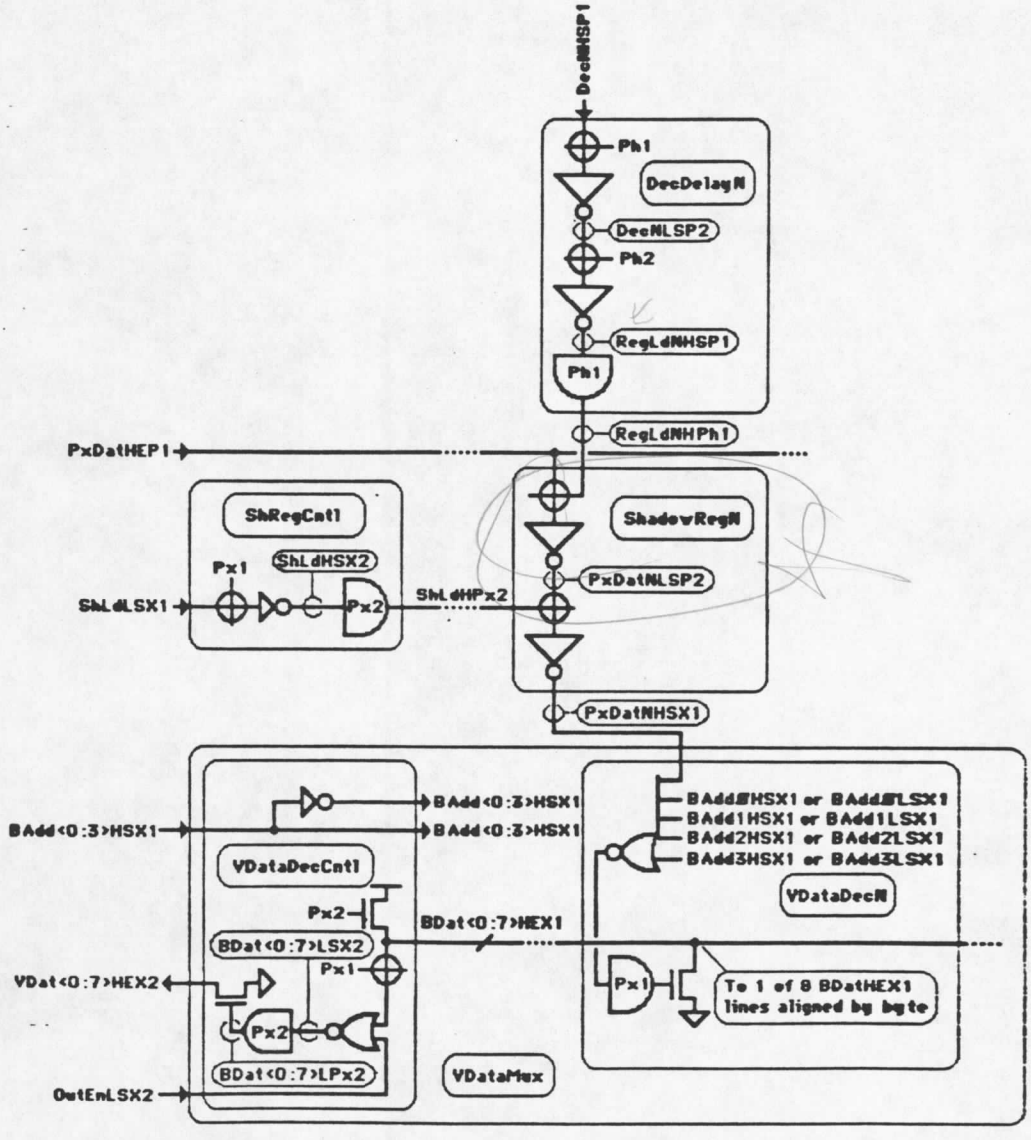


Figure 2.4.21: Schematic of ShadowReg, VDataMux, and associated circuits.

The shadow-register mechanism is contained in three leaf-cell modules:

(1) DecDelay delays the arrival of the one-hot signal **DecNHSP1** for one clock cycle to match the delay through the pixel data selector from **RDat** to **PxDat**. When **RegLdHSP1** is HI, **RegLdHPh1** is asserted, causing the data on **PxDatHEP1** to be loaded into the master half of the shadow register.

(2) ShadowRegCntl latches the control signal **ShLdLSX1** and produces the qualified clock **ShLdHPx2**.

(3) ShadowReg's master, which, after all bits of pixel memory have been 'visited', contains a copy of the pixel memory data, can be unloaded into the slave by asserting **ShLdHPx2**. Note that there is a clear violation of signal typing at the input to the slave, where a type-

SP2 signal is latched on the **Px** clock; in general this violation will lead to synchronization failure in the register. Timing for the VDataMux module is discussed below; the discussion describes the conditions under which synchronization failure in the ShadowReg will not affect the video data.

The VDataMux (video data multiplexer) consists of two leaf-cell modules:

(1) The VDataDecCntl module passes the byte address **BAdd<0:3>HSX1** through to the data decoders, providing the complements **BAdd<0:3>LSX1**. The module also contains the on-chip video data bus access circuits. The module data bus **BDat<0:7>HEX1** is precharged by VDataDecCntl on **Px2**, and its value is latched during the evaluation phase **Px1**. If enabled for output (**OutEnLSX2**=0), the module evaluates the on-chip video data bus **VDat<0:7>HEX2**, passing **BDat** onto **VDat**. If not enabled, the **VDat** bus wires are left HI.

(2) The modules VDataDec select a byte-wide subset of the **PxDat***N*HSX1, connecting them to the internal bus **BDat<0:7>HEX1. These modules are constructed so that**

BDat<0:7> gets	when **BAdd3..BAdd0=**
PxDat<0:7>	0000
PxDat<8:15>	0001
PxDat<16:23>	0010
PxDat<24:31>	0011
PxDat<32:39>	0100
PxDat<40:47>	0101
PxDat<48:55>	0110
PxDat<56:63>	0111
PxDat<64:71>	1000

Figure 2.4.22 shows the leaf cells of the pixel selector mechanism.

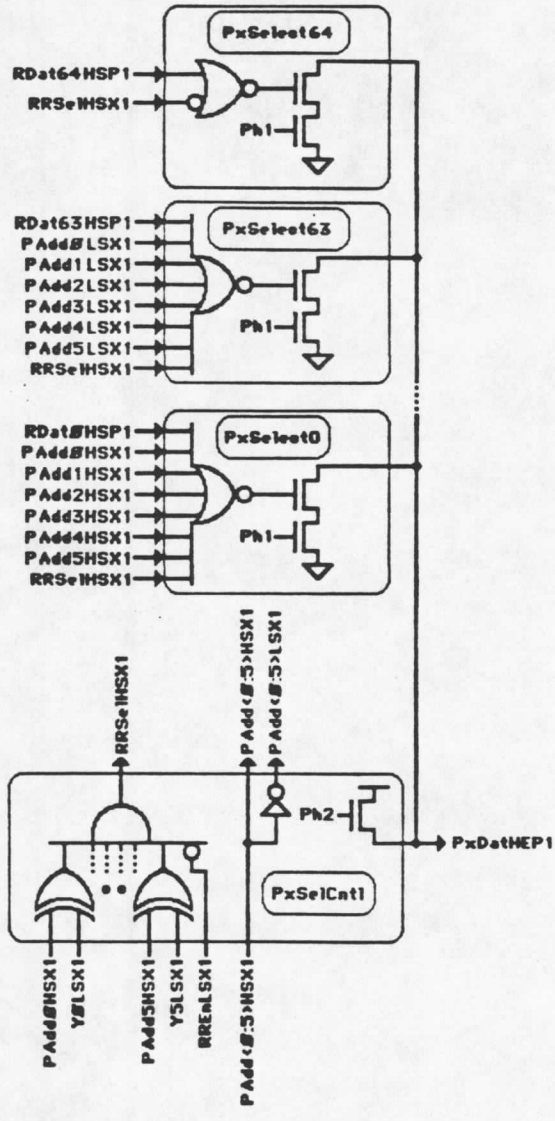


Figure 2.4.22: Schematic of PxSelect and PxSelCntl submodules of YDataSelect

The submodule PxSelCntl passes the pixel-select address **PAdd<0:5>HSX1** through to the PxSelect modules, and produces the complements **PAdd<0:5>LSX1**. The module also compares the pixel-select address with the address of the redundant row **Y<:5>LSX1** (active-LO for implementation reasons). When the two are equal AND the redundant row enable RREnLSX1 is asserted, PxSelCntl asserts the redundant row select **RRSelHSX1**. This submodule also contains the precharge circuit for the pixel data bus **PxDatHEP1**.

The PxSelect modules decode **PAdd** so that the selected pixel's data is transferred to the **PxDatHEP1** bus. If **RRSelHSX1** is asserted, PxSelect<0:63> are disabled and PxSelect64 (corresponding to the redundant row) is enabled and puts its data onto the **PxDat** bus.

Memory Timing

Figure 2.4.23 shows the timing for a typical PixelMemory module in DataStore (see also Figure 2.4.19).

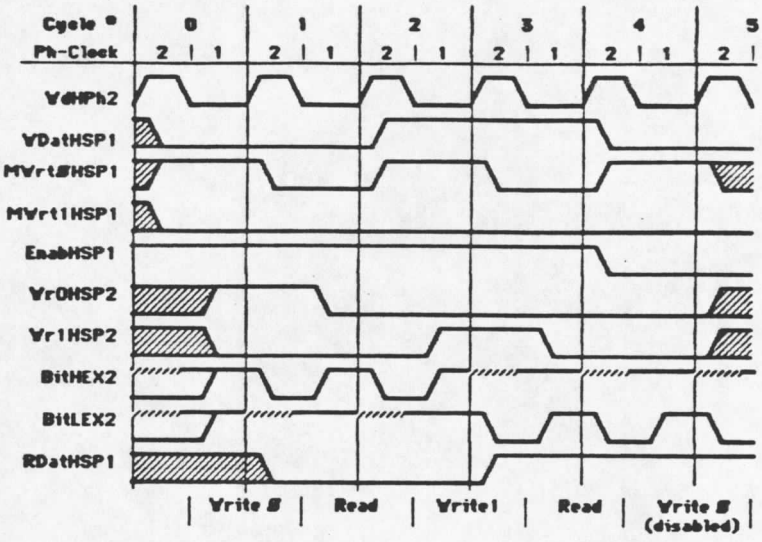


Figure 2.4.23: Timing in DataStore module.

The figure shows a sequence of cycles in which one particular memory bit is accessed on each successive cycle. First a '0' is written to the bit, then read back; then a '1' is written to the bit, then read back; finally at attempt is made to write a '0' to the bit with **EnabHSP1**=LO, resulting in another read.

At the beginning of cycle #0, **WDat** goes LO and **MWrt1,MWrt0**=01 (Write in Enabled), initiating the write-0 sequence. On **Ph1** of cycle #0, the signals **Wr1HSP2,Wr0HSP2** go to 01, and the memory bit lines **BitHEP2,BitLEP2** are precharged HI. The write is completed during **Ph2** of cycle #1, where the MemWrite circuit pulls **BitHEP2** LO. The **RDatHSP1** output also becomes valid here, reflecting the data written to memory.

At the beginning of cycle #1, **MWrt1,MWrt0** goes to 00, initiating a read sequence. On **Ph1** of cycle #1, **Wr1,Wr0** go LO, and the bit-lines are again precharged. The read is completed during the first half of cycle #2; the memory cell pulls **BitHEP2** LO, and the data is transferred to **RDatHSP1**.

Cycle #2 initiates another write operation, this time with **WDatHSP1**=1. The write-1 operation is completed during the beginning of cycle #3, with the new data appearing simultaneously in the memory cell and on the **RDat** output.

Cycle #3-cycle #4 reads back the '1' just written and initiates another write-0 operation. However, at the beginning of cycle #4, **EnabHSP1** goes LO, so that cycle #4-cycle #5 performs another read operation, retrieving the previously-written '1' again.

Figure 2.4.24 shows the timing in the VDataSelect module and in the decoder of DataStore.

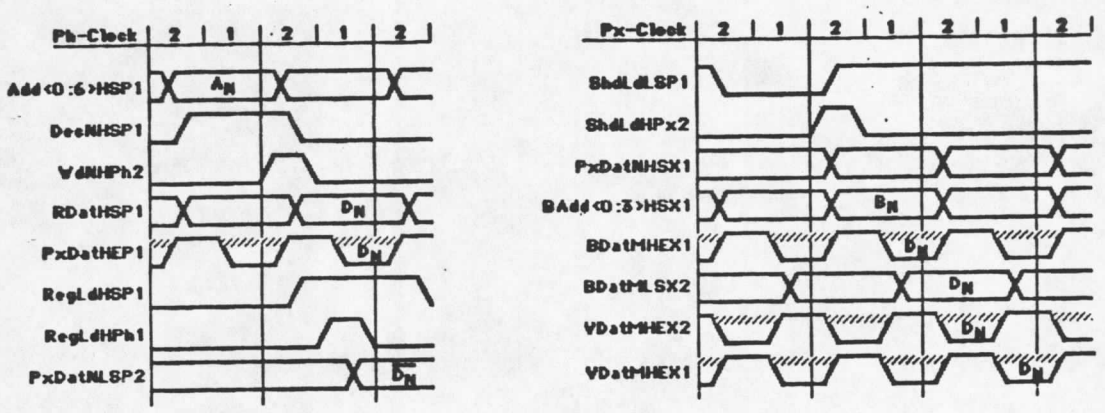
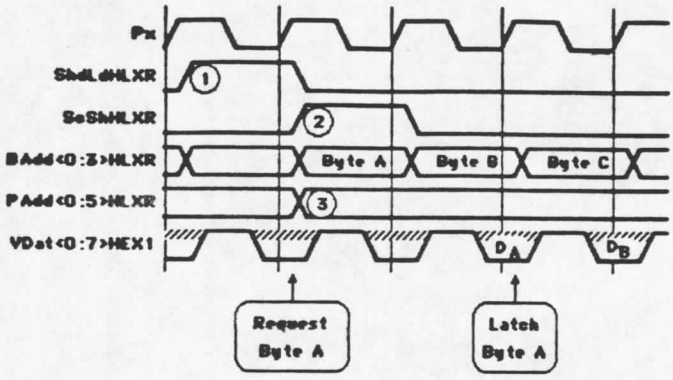


Figure 2.4.24: Timing in VDataSelect.

The left-hand timing diagram shows address *N* arriving on **Add<0:6>HSP1**, causing **Dec*N*HSP1** to be asserted. This in turn enables **Wd*N*HPh2** to toggle on the next **Ph2**. The corresponding memory bit is accessed during this **Ph2** half-cycle, and the **RDatHSP1** for the selected row takes on the value read from or written to memory, DN. The enabled PxSelect puts this value onto **PxDatHEP1** on the next **Ph1**. During this same interval, **Dec*N*HSP1** travels through DecDelay, arriving at **RegLd*N*HSP1 one-half cycle ahead of the pixel data DN. This in turn enables RegLd*N*HPh1** to toggle, loading the value on **PxDatHEP1** into the ShadowReg*N*, and updating **PxDat*N*LSP2**.

The right-hand timing diagram shows the train of events required to unload the shadow register and pass the video data out of the chip. On the first cycle, **ShLd** is asserted and latched into ShadowRegCntl. **PAdd<0:5>HSX1** must remain stable during the unload to prevent shadow-register master from being trashed by a faulty write--see below. On the next cycle, **ShLdHPx2** is asserted, unloading the register and updating all of the **PxDat*N*HSX1** outputs. The byte-address **BAdd<0:3>HSX1** may change on this cycle to request new video data. The data selected is transferred onto the module video data bus **BDat<0:7>HEX1** on the next **Px1** and latched into the VDataDecCntl module. On the next **Px2** (third cycle) the data is transferred onto the chip's internal video data bus **VDat<0:7>HEX2**, and latched into the output pads PadOutPE. Finally, the data is output on **Ph1** of the third cycle onto the external data wires **VDat<0:7>HEX1**.

Figure 2.4.25 shows the timing of video data requests from the point of view of the chip I/O pins.



Notes:
(1) Load Shadow Register at least 1 cycle before requesting data.
(2) Shift in Global Token on or before requesting data.
(3) Change PAdd at least 1 cycle later than asserting ShdLd and on or before requesting data.

Figure 2.4.25: External bus timing for reading video data.

The notes on the figure are self-explanatory; there is a 2-cycle delay between data requests (changing **BAdd<0:3>HLXR**) and the rising edge where the requested can be latched.

2.4.3.5. Scan-Path Controller

Figure 2.4.26 shows the second-level modular decomposition of the ScanPathCntl module.

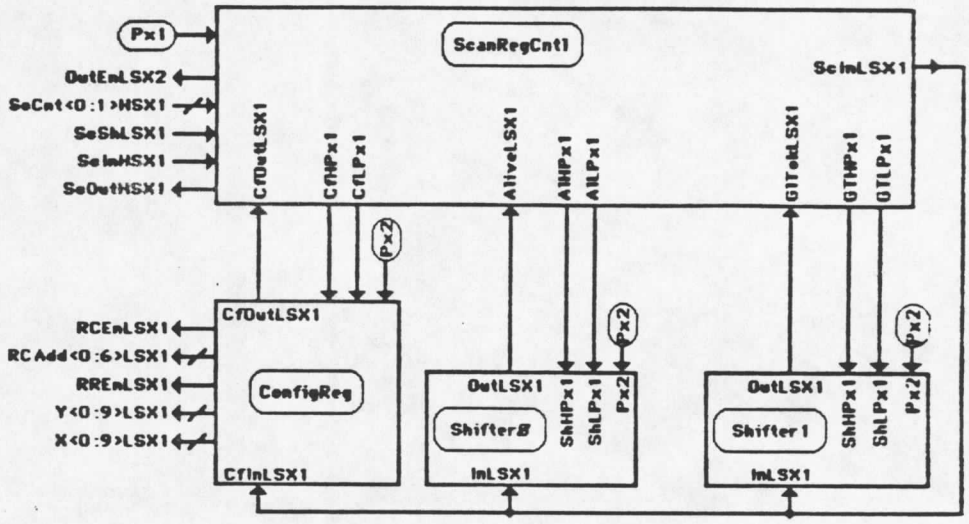


Figure 2.4.26: Second-level modular decomposition of ScanPathCntl.

The module has four submodules:

(1) ScanRegCntl, which decodes the control inputs **ScCnt<0:1>HSX1** and **ScShLSX1** and, together with the value of **Alive**, generates the qualified clocks that control the registers in ScanPathCntl.

(2) ConfigReg, a collection of 29 Shifter registers that store the values of the chip's X- and Y-address **X<0:9>LSX1,Y<0:9>LSX1**, the address of the redundant column **RCAdd<0:6>LSX1**, and the redundant row and column enables **RREnLSX1,RCEnLSX1**.

(3) Two instantiations of Shifter that implement the Alive and Global-Token registers.

The module's function is determined by the three external controls and the state of the internal Alive register according to Table 2.4.4:

Table 2.4.4: Scan-Path Controller functions.

ScCnt1	ScCnt0	Alive	ScSh	Function:
0	0	x	1	Select Alive register
0	1	1	1	Select Config register
1	0	1	1	Select Global Token register
1	1	x	x	Short ScIn to ScOut
0	1	0	x	Short ScIn to ScOut
1	0	0	x	Short ScIn to ScOut
0	0	x	0	Save internal state
0	1	1	0	Save internal state
1	0	1	0	Save internal state

Figure 2.4.27 shows the third-level decomposition of the ConfigReg module.

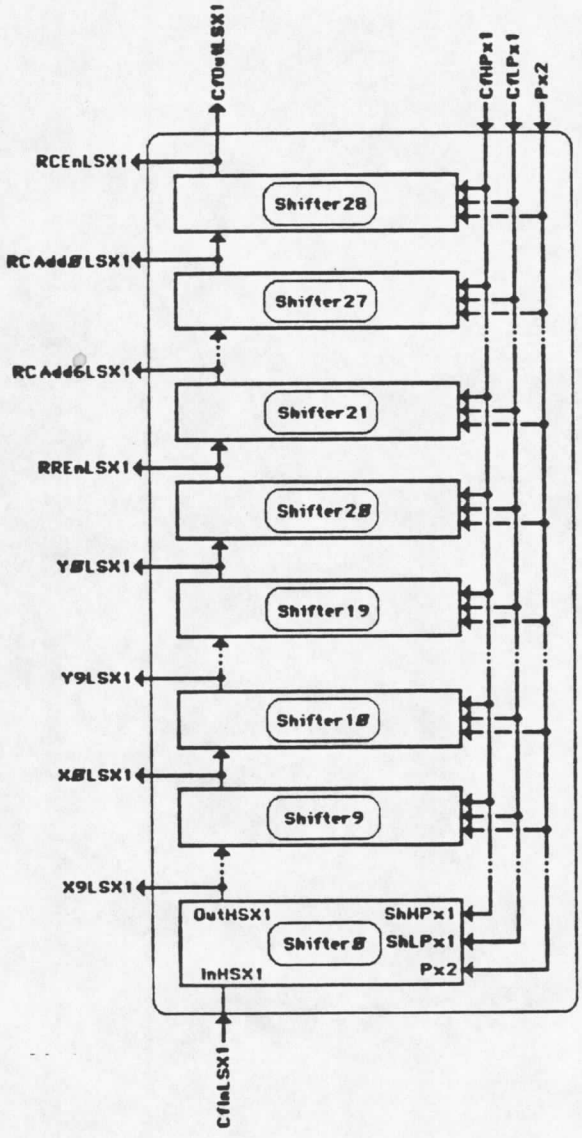


Figure 2.4.27: Schematic of ConfigReg module.

This module is simply 29 instantiations of Shifter with control inputs **CfHPx1**, which, when asserted, causes data to shifted into **CfInLSX1** and simultaneously out of **CfOutLSX1**, and **CfLPx1** which refreshes the state of the registers.

Figure 2.4.28 shows the internals of the ScanRegCntl module.

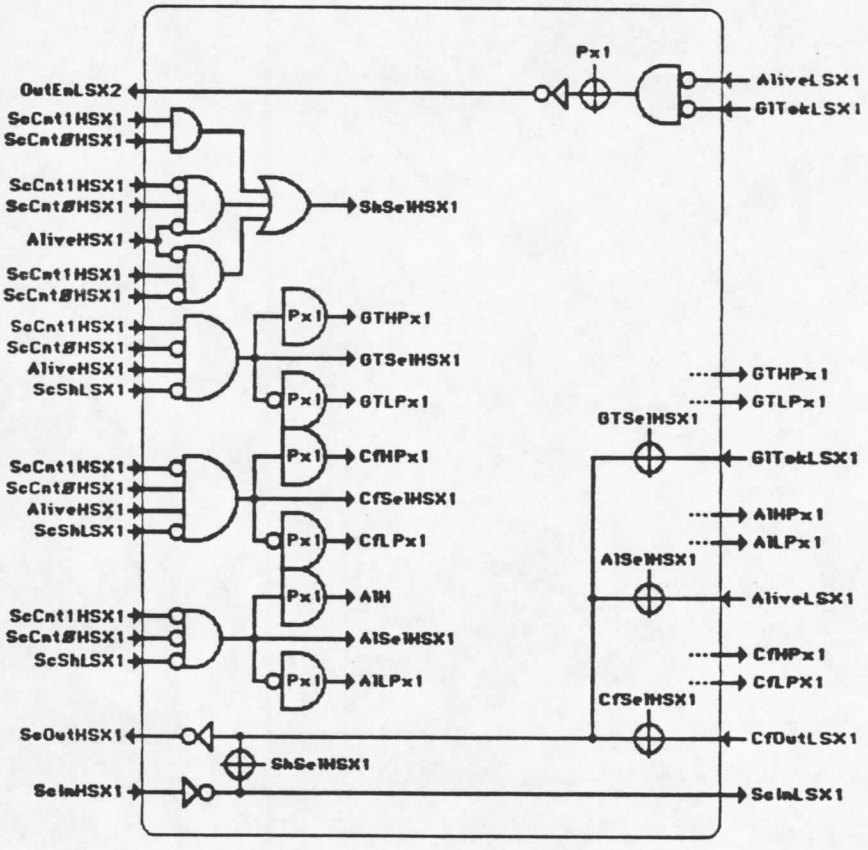


Figure 2.4.28: Schematic of ScanRegCntl module.

The module has three functions:

(1) The generation of the four decoded control signals **ShSelHSX1** (Short-circuit Select), **GTSelHSX1** (Global-Token register Select), **CfSelHSX1** (Configuration register Select), **AlSelHSX1** (Alive register Select), and associated qualified clocks.

(2) Selection of a destination for **ScInHSX1** (the scan-path input) and a source for the **ScOutHSX1** output.

(3) Generation of the **OutEnLSX2** (Output Enable) signal from **Alive** and **GlTok**.

The four selects are generated (referring to Table 2.4.4) by the following logic equations:

Sh = (ScCnt1 • ScCnt0) + (!Alive • !ScCnt1 • ScCnt0) + (!Alive • ScCnt1 • !ScCnt0)
GT = ScCnt1 • !ScCnt0 • Alive • ScSh
Cf = !ScCnt1 • ScCnt0 • Alive • ScSh
Al = !ScCnt1 • !ScCnt0 • ScSh

The selectors **ShSelHSX1, GTSelHSX1, CfSelHSX1, AlSelHSX1** are mutually exclusive, and drive a four-to-one multiplexer that selects the internal path from **ScInHSX1** to **ScOutHSX1**. Note that the **ScIn** and **ScOut** signals are active-LO inside the ScanPathCntl module for implementation convenience.

The signal **OutEnLSX2**, passed to the Memory module to enable module/chip video output, should be asserted only when

(1) the module/chip is Alive, and

(2) the Global Token is on the module/chip.

The active-LO versions of these signals are ANDed on the module and latched on **Px1** to satisfy the timing requirements.

Figure 2.4.29 shows the module Shifter.

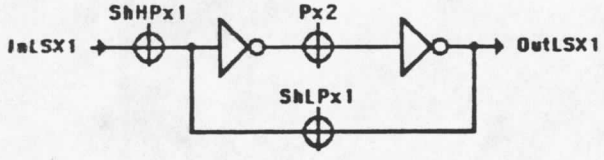
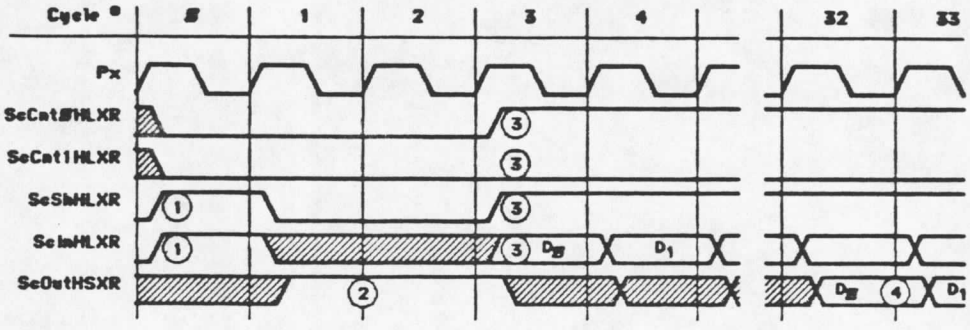


Figure 2.4.29: Schematic of Shifter module.

This module is a simple shifter with state-save. Shifting is enabled when **ShHPx1** is asserted; the mutually exclusive **ShLPx1** causes the latch to refresh its contents.

Scan-Path Timing

Figure 2.4.30 shows the external bus timing associated with the ScanPathCntl.



Notes:

(1) Input a '1' to the Alive register. Register selection and data input can be settled on or before cycle that enables shifting.

(2) Alive register output appears 1 cycle later, can be latched 2 cycles after input. Global Token register timing is identical. In multi-module chips, delay is (N+1) cycles where N is number of modules.

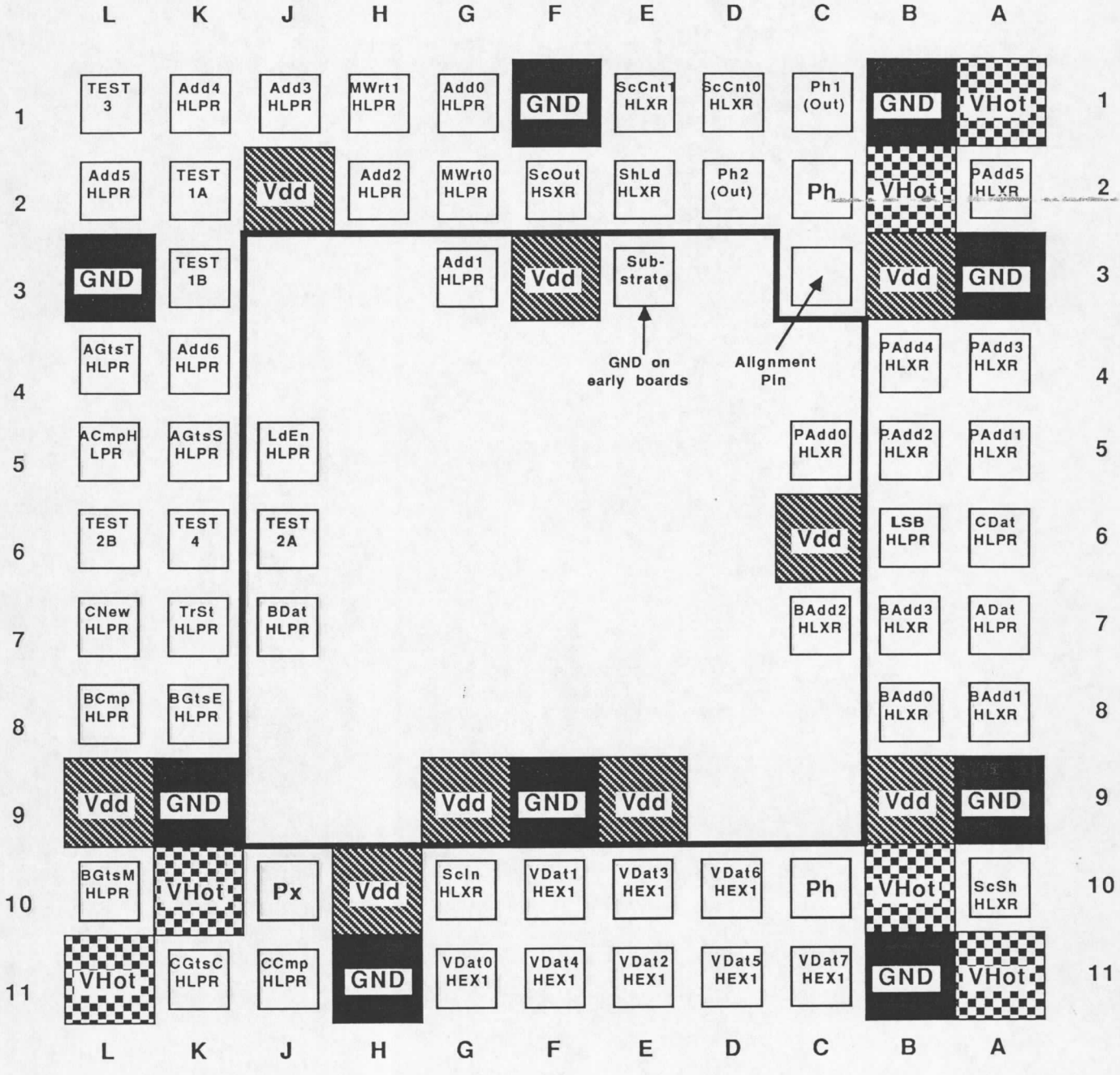
(3) Input data to Configuration register. Register selection and data input can be settled on or before cycle that enables shifting. Data output on this cycle is previously-stored contents of LSB of Configuration register.

(4) Configuration register outputs appear 29 cycles later, can be latched 30 cycles later. In multi-module chips, delay is (29N+1) cycles, where N is number of modules.

Figure 2.4.30: Timing for external interface to Scan-Path Controller.

Cycle #0 of the figure shows selection of the Alive register on a chip by asserting **ScCnt1HLXR,ScCnt0HLXR**=00. Simultaneously (or later) the scan-path input **ScInHLXR** gets the data to be loaded (a '1' in this case) and shifting is enabled by asserting **ScShHLXR**. The data appears at the scan-path output **ScOutHSXR** during the next cycle and may be latched at the beginning of cycle #2. For chips that contain more than one module, this two-cycle delay becomes an (N+1)-cycle delay, where N is the number of modules on the chip. On cycle #1, **ScShHLXR** is de-asserted (and the value on **ScInHLXR** is ignored); the output **ScOutHSXR** maintains its value until shifting re-commences.

Cycle #3 in the Figure shows the initiation of a scan-in/out sequence for the Configuration register. When **ScCnt1,ScCnt0** goes to 01, the Configuration register is selected; simultaneously (or later) shifting can be re-enabled by asserting **ScSh**. Assertion of **ScSh** causes the **ScOut** output to take on the value stored in the LSB of the Configuration register; for the next 29 cycles, the register data appears at the **ScOut** output in LSB-first order. Finally, on cycle #32, the first data bit inserted on cycle #3 appears out the **ScOut** output—it may be latched at the rising edge of cycle #33. For one-module chips, the total delay through the Configuration register is 30 cycles; for an N-module chip, the delay is ((29*N)+1) cycles.



Pinout for Pixel-planes 4.1 and 4.2

TOP VIEW