IV.3.3 LOGIC DESIGN

The IGC is divided into the following modules:

(1) InputLatch: input pads, latches all inputs to the IGC and decodes register address
(2) Controller: handles the input handshaking and the movement of instructions through the IGC instruction pipeline
(3) Sequencer: the microcode sequencer, including the microcode store
(4) LoopCount: loop counters, used by Sequencer for execution loops
(5) PixMemAddr: generates the pixel-memory address outputs
(6) DirectReg: serializes 32-bit integer data for direct-to-ALU mode
(7) Serializer: bit-serializes the QEE coefficients, and produces synchronization signals
(8) OutputLatch: output logic and pads for the IGC

IV.3.3—1 InputLatch Module

InputLatch contains all of the input pads, as well as logic for conditioning and buffering the input signals for the other modules.

All of the IGC inputs, including the input data **D<0:31>HTPR**, the register address **T<0:2>LLPR**, the register write enable **WrEnLLPR**, the "go" signal **GoLLPR**, the condition inputs **ST1HLPR** and **ST2LLPR**, the reset signal **ResetHLPR**, and the test mode inputs **TestOEHLPR** and **TestPEHLPR**, are of type "latched on the rising edge of Ph", denoted by the **-LPR** suffix. Specifically, this means these signals must be stable in a window about the falling edge of Ph (see timing specifications); typically, these signals are expected to be generated off-chip, using an edge-triggered latch clocked on the rising edge of Ph. The inputs are latched on Ph2 in **InputLatch** (except for **TestOEHLPR** and **TestPEHLPR**, which are just buffered).

D<0:31>HTPR are bi-directional; they can also act as outputs in test mode (see below), but in normal operation they act as input signals with timing similar to the -LPR inputs. As inputs, **D<0:31>HTPR** are latched on Ph2 and again on the following Ph1; they are driven across the chip as **D<0:31<HSP2**.

WrEnLLPR and **T<0:2>LLPR** are latched on Ph2. On any clock cycle for which **WrEnLLPR** is asserted, the register address **T<0:2>LLPR** is decoded during Ph1, so

that one of the 8 pairs of register-load enables **IWord<H,L>SP2**, **PWord<H,L>SP2**, **AWord<H,L>SP2**, **BWord<H,L>SP2**, **CWord<H,L>SP2**, **DWord<H,L>SP2**, **EWord<H,L>SP2**, or **FWord<H,L>SP2** is asserted; these enable lines are driven accross the chip.

The 8 virtual input registers of the IGC are distributed among the other IGC modules; each module receives its input data by latching some subset of **D<0:31>HSP2** when the appropriate register-load enable signal, ***Word<H,L>SP2**, is asserted.

GoLLPR is simply latched on Ph2, and sent to the **Controller** module as **GoLSP1**.

ST1HLPR and **ST2LLPR** latched on Ph2 and sent to the **Sequencer** module as **ST1HSP1** and **ST2HSP1**.

TestOEHLPR is just buffered, and sent to **Sequencer** as **TestOE<H,L>**. **TestPEHLPR** is buffered and becomes **TestPE<H,L>**, the output enables for the bi-directional **D<0:31>HTPR** pads. When **TestPEHLPR** is asserted, **InputLatch** tri-states **D<0:31>HSP2**, and **D<0:31>HTPR** actively drive the data on **D<0:31>HSP2** off-chip. Normally, one cycle after **TestPEHLPR** goes hi, **TestOEHLPR** would be brought high, causing **Sequencer** to assert the current microcode word, **RdData<0:31>HSP1**, onto **D<0:31>HSP2**. Thus, the memory contents can be read via **D<0:31>HTPR**; the input device should tri-state these inputs when **TestPEHLPR** is asserted. This mode is intended only for testing; **TestPEHLPR** and **TestOEHLPR** should be grounded during normal operation.

IV.3.3—2 Controller Module

Controller is basically a maze of random logic and drivers which (1) supervise the passage of instructions through what is effectively a three-stage pipeline in the IGC, and (2) generate **ShiftHSP1**, which controls the **Serializer** module and becomes the IGC output **TrStHSPR**.

In the first stage of the instruction pipeline, an instruction is written to the virtual input registers, and the various fields of the input data words are latched into the various IGC modules by the ***Word<H,L>SP2** signals. Each of these signals specifies a write to the virtual input register indicated by the first letter of the signal name, i.e.:

IWord<H,L>SP2 specifies a write to the I register. For writes to the I and P registers, the various fields of the data words, including the starting microcode address for the instruction and the starting values for the pixel-memory address and execution loop counters, are just latched. Writes to the 6 coefficient registers cause the coefficient value to be latched into **Serializer**; the exponent is decoded and the decoded value is latched. After all appropriate registers have been loaded, the inputting device assertes **GoLLPR**; this causes **IBsyHSP1** and **CBsyHSP1** to be asserted, indicating that a new instruction is loaded and ready to be processed; the logical-OR of IBsy and CBsy, **BusyHBP1**, becomes the IGC output signal **BusyHSPR**, which signals the inputting device that no more data may be written into the IGC at this time.

An instruction passes to the second stage of the pipe when it is *posted:* the fields in the opcode are transferred to another set of latches and bit-serialization of the A,B,C,D,E,F coefficients begins (so that the coefficient bit-streams can begin being shifted into the QEE's of the EMC's); at this point, the instruction is said to be *pending*. Passing of the instruction from this first *input latching* stage to the second stage of the pipeline occurs in two stages, controlled by **PostI<H,L>SP2** and **PostC<H,L>SP2**. **PostI** is asserted when **IBsyHSP1** = 1, indicating that a new instruction has been latched into the IGC (the first stage of the pipeline is full), and **IPHSP1** = 0, indicating that there is no instruction *pending* (the second stage of the pipeline is empty). When **PostI** is asserted, the instruction is *posted*, meaning that the starting microcode address, the starting pixel-memory addresses, and the initial counts for the loop counters are passed to another set of latches. Also, **IPHSP1** is raised, indicating that the pending instruction is ready for execution in **Sequencer**, and preventing **PostI** from being asserted again. **PostC** is asserted when: **CBsyHSP1** = 1, indicating that the coefficients of the new instruction have not yet been posted; **CSBHSP1** = 0, indicating that the **Serializer** has (or is about to) finish serializing the coefficients of the previous instruction; and **Shift** will also be asserted, so that the last bit of the previous set of coefficients is not overwritten when the new set of coefficients is posted. When **PostC** is asserted, the coefficients are also "posted", and bit-serialization begins in **Serializer**. **PostC** is delayed by a half-cycle, to become **PostC<H,L>SP2**, and this SP2 form controls the **Serializer**. Neither **PostI** nor **PostC** should be asserted on successive cycles, hence **IBsy** and **CBsy** are each qualified by the previous cycle value of **PostI** and **PostC** to produce **NIHldHSP2** and **NCHldHSP2**.

An instruction enters the third and final stage of the pipe when **Sequencer** begins executing the microcode sequence for the instruction. This is indicated by the **Sequencer** asserting **NewLSP1** for a cycle. This indicates that the **Sequencer** has begun executing

microcode at the starting address specified in the opcode. The loop count and pixel-memory address fields from the opcode are also loaded into their respective counters. Serialization of the coefficients continues, if necessary. **IPHSP1** is lowered to indicate that the instruction no longer is *pending*, and the second stage of the pipe is empty, so that the next instruction may be posted. **IPHSP1** may be thought of as an RS flip-flop which is set by **PostI** and cleared by **NewLSP1**.

There must be some synchronization between the instruction which is pending, whose coefficients are being bit-serialized in **Serializer** and passed into the QEE's, and the previous instruction, which may still be executing in **Sequencer**. **Serializer** produces the signal **TRRHSP1**, which goes high when the LSB of the QEE result for an instruction has reached the pixel-ALU. At this point, if the previous instruction is still executing, operation of the QEE's must temporarily cease until the previous instruction finishes executing. This is done by allowing **TRRHSP1** to inhibit the shift enable signal **ShiftHSP1**. In the default *conditional* shift control mode, **ShiftHSP1** is asserted unless **TRRHSP1** is asserted. When the previous instruction is done, and the instruction passes to the third (executing) stage of the pipe, it executes a conditional branch to test for **TRRHSP1**, asserts the *tree force* shift control mode, to force **ShiftHSP1** over the "bump", and continues executing. If the instruction requires the QEE's be run at half speed, the *tree halt* shift control mode can be asserted to unconditionally de-assert **ShiftHSP1** on alternate cycles. The shift control control modes are summarized as follows:

Mode	RModeHSP1	Tree1HSP1	Tree0HSP1	TRRHSP1	ShiftHSP1
Reset	1	x	x	x	0
Tree Force	0	0	x	x	1
Conditional	0	1	0	0	1
Conditional	0	1	0	1	0
Tree Halt	0	1	1	x	0

ShiftHSP1 is delayed by an additonal half-cycle, to become **Shift<H,L>SP2** and this SP2 form is used to control the **Serializer**.

When the P register is written with **D31HTPR** = 1, the IGC enters RMode; the reset signal, **ResetHSP1**, also forces the IGC into RMode. RMode is used for reading and writing the microcode store. In RMode, **CSBHSP1** is disabled and **Shift** is

unconditionally asserted. This means that when a new instruction is loaded that **PostC** can occur immediately. Unconditionally asserting **Shift** also flushes out the various shifters in **Serializer**. Similarly, **Sequencer** always asserts **NewLSP1**, which means that **PostIHSP1** can also occur immediately, and so **IPHSP1** can only remain asserted for one cycle. Thus, in RMode, the IGC can never get hung, that is, **BusyHBP1** can never get stuck high, preventing more input. Furthermore, if the IGC becomes hung, putting it in RMode will always reset **Controller** to its quiescent state (**IBsyHSP1** = **CBsyHSP1** = **BusyHBP1** = **IPHSP1** = **PostCHSP1** = **PostIHSP1** = 0). However, since **BusyHBP1** is usually stuck high in a hung IGC, RMode must be entered either by asserting **ResetHLPR**, or by overriding the normal input protocol which requires that **BusyHSPR** be low.

IV.3.3—3 Sequencer Module

Sequencer is the heart of the IGC. It contains the **Memory** sub-module, with 416 (nominally 512) 32-bit words of microcode store, the **AddrGen** sub-module, which controls program flow and generates microcode addresses, and two latches which hold the starting microcode address for an instruction during the first and second (pending) stages of the instruction pipeline.

Sub-module **Memory** takes the pre-decoded address from **AddrGen** and reads or writes the corresponding microcode word in the memory array.

The microcode word is 32 bits wide, formatted as follows:

0	DirEn	enable for direct-to-ALU register (DirectReg module)
1	ACmp	becomes IGC output ACmp
2	AGtsS	becomes IGC output AGtsS
3	AGtsT	becomes IGC output AGtsT
4	CCmp	becomes IGC output CCmp
5	CGtsC	becomes IGC output CGtsC
6	BCmp	becomes IGC output BCmp
7	BGtsE	becomes IGC output BGtsE
8	BGtsM	becomes IGC output BGtsM
9	LdC	becomes IGC output LdC
10	LdE	becomes IGC output LdE
11	MWrt	becomes IGC output MWrt
12-14	PMAInstr<0:2>	pixel-memory address instruction (PixMemAddr)
15-16	Tree<0:1>	generates ShiftHSP1 and IGC output TrSt
17	Cnt2	count enable for LoopCounter 2
18	Cnt1	count enable for LoopCounter 1
19-27	BrAddr<0:8>	branch address (goes to sub-module AddrGen)
28-30	SeqInstr<0:2>	sequencer instruction (goes to sub-module AddrGen)
31	Done	Done signal

The last word in the microcode sequence which executes some instruction, as well as the microcode word at address 0 (the "idle" state), has the Done microcode bit set, and the branch field set to address 0. Thus when the instruction completes, control jumps either to the next instruction (if **IPHSP1** = 1), or to address 0, the Idle state.

The microcode memory is nominally 128 rows by 128 columns (only 104 rows in Version 5.0). The memory word-lines, **Wd<0:103>HMEnH**, are computed from the 7 MSB's of the address in pre-decoded form: **Hi<0:6>HSP2**, **Med<0:3>HSP2**, and **Lo<0:3>HSP2**. The 2 LSB's of the address in pre-decoded form, **RS<0:3>HSP2**, select one of 4 rows for each of the 32 microcode bits.

AddrGen generates addresses for **Memory** in *pre-decoded* form. On any microcycle, 3 possible addreses may be selected: (1) the *new* address, the starting address for the next instruction to be executed, (2) the *incremented* address, the address immediately following

the current one, and (3) the *branch* address, the address specified in the branch address field, **BrAddr<0:8>HSP1**, of the current microcode word. **BrAddr<0:8>HSP1** is pre-decoded to produce the signals **BrHi<0:7>HSP1**, **BrMed<0:3>HSP1**, **BrLo<0:3>HSP1**, and **BrRS<0:3>HSP1**. Similarly, **NewAddr<0:8>HSP1** are pre-decoded. The incremented address is generated from the current address in pre-decoded form. More or less simultaneously with the pre-decoding, during the first half or so of Ph1, the address source is selected, based on the sequencer instruction field of the current microcode word, **SeqInstr<0:2>HSP1**, and the condition codes: **TRRHSP1** from **Serializer**, the terminal counts **TC1HSP1** and **TC2HSP1** from **LoopCount**, and the IGC status inputs **ST1HSP1** and **ST2HSP1**. The address source is specified by **New<H,L>SP1** and **Branch<H,L>SP1**. The three possible addresses are multipexed to get the next address in two stages: first, the *new* or *incremented* address is selected according to whether **New<H,L>SP1** is asserted; next, either this address or the *branch* address is selected according to whether **Branch<H,L>SP1** is asserted.

NewLSP1 may be asserted when **DoneHSP1** is asserted, as a microcode word with the "Done" microcode bit set; this "done" word may be either the last word of the microcode sequence for the previous instruction, if the previous instruction is just finshing, or the microcode word at address 0, if **Sequencer** is in the Idle state. When **DoneHSP1** goes high, and if an instruction is pending (**IPHSP1** = 1), **NewLSP1** is asserted. **NewLSP1** is also unconditionally asserted if **RModeHSP1** is asserted.

Branch<H,L>SP1 is generated by a pre-charge/evaluate circuit. It is pre-charged during Ph2, and may be evaluated low during Ph1, in which case **Branch<H,L>SP1** is de-asserted. For each of the condition inputs **TC<1,2>HSP1**, **TC1LSP1**, **ST<1,2>HSP1**, and **TRRHSP1**, there is a pull-down path which is enabled if the a conditional branch sequencer instruction is selected and the corresponding condition input is asserted. (For conditional branches, the branch is taken if the condition is 0). There is a pull-down path for the unconditional increment sequencer instruction, and two paths that cause **BranchHSP1** to be asserted whenever **New<H,L>SP1** is asserted. Note that when the IGC is in RMode, the *new* address is always selected.

When **TestOE<H,L>** is asserted, **Sequencer** drives the **Memory** outputs **RdData<0:31>HSP1** onto **D<0:31>HSP2**. When **TestPEHLPR** is also asserted, **InputLatch** drives this data off-chip on **D<0:31>HTPR**. **TestOE** and **TestPE** are asserted only during chip testing; they is grounded during normal operation.

IV.3.3—4 LoopCount Module

LoopCount contains two 7-bit down counters with zero-detect, and **WordLatch**'s and **PostILatch**'s for storing the starting loop count values from the opcode and supplementary opcode during the first and pending stages of the instruction pipeline.

Loop Counter 1 is loaded from bits 23-29 of the I register (opcode), but the value is represented in excess-116 form; that is, the value in bits 23-29 is the desired count value plus 116. When **DoneHSP1** is asserted, the counter is loaded from the **PostILatch**. When **Cnt1LSP1** is asserted, the count value is decremented. **Cnt1LSP1** must not be asserted when **DoneHSP1** is asserted; the microcode assmbler *asmpp5* should check for this. The zero-detect circuits asserts **TC1HSP1** when the count value is 116 (to compensate for the offset in the loaded value).

Loop Counter 2 is loaded from bits 16-22 of the P register (supplementary opcode). The counter is very similar to **Loop Counter 1**, except that the input value is not offset, so the zero-dectect logic is slightly different. Counting is enabled by **TC2LSP1** and **TC2HSP1** is asserted when the count value is zero.

Each loop counter contains an 2-way multiplexing input half-latch driven on Ph1, a 7-bit add-one circuit, and an output half-latch driven on Ph2. The counter is decremented by adding one to the one's complement of the count value, and complementing the result. The zero-detector looks at the output of the Ph1 half-latch, and the value of the load and count control signals **DoneHSP1** and **CntLSP1**; based on these signals in computes whether or not the new count value will equal the offset, and if so **TCHSP1** is asserted.

Count enables for the two counters, **Cnt1LSP1** and **Cnt2LSP1**, are generated directly from the corresponding bits of the **Sequencer** microcode word. The two loopcounters assert **TC1HSP1** and **TC2HSP1**, respectively, when they contain all 0's.

By performing conditional branches on **TC1SHP1** and **TC2HSP1**, the **Sequencer** can execute simple loops.

The loop counters do not halt when they reach 0. The microcode writer must take care that a loop counter does not underflow before **TC*i*HSP1** is detected by a conditional branch.

IV.3.3—5 PixMemAddr Module

PixeMemAddr generates the pixel-memory address outputs of the IGC. For any IGC instruction, 3 sequences of pixel-memory addresses may be specified by starting address in fields of the opcode and supplementary opcode.

PixMemAddr contains 3 counters, for each of the 3 pixel-memory address sequences, 3 pairs of latches for storing the starting pixel-memory addresses for an instruction during the first and *pending* stages of the instruction pipeline, and logic to decode the pixel-memory address instruction field of the microcode word, **PMAInstr<0:2>HSP1**.

When an instruction is loaded into the IGC, the destination address field from the opcode, and the two source address fields from the supplementary opcode, are loaded into the WordLatch's.

When the instruction is posted, the addresses are simply passed to the PostILatch's.

Finally, when **DoneHSP2** is asserted, indicating that **Sequencer** is idle or that the previous instruction is ending, the starting addresses are loaded into the 3 address counters. As the instruction executes, the pixel-memory address instruction field of the microcode word specifies which of the 3 counters provides the address bits **Addr<0:7>HSP1**, and if that counter is to be post-incremented or post-decremented. The pixel-memory address instruction is decoded as follows:

Mnemonic	PMAInstr2	PMAInstr1	PMAInstr0	Action
aux	0	0	0	use Auxiliary
dst	0	0	1	use Destination (default)
dst+	0	1	0	use Destination, then increment
dst-	0	1	1	use Destination, then decrement
aux+	1	0	0	use Auxiliary, then increment
src	1	0	1	use Source
src+	1	1	0	use Source, then increment
src-	1	1	1	use Source, then decrement

Note that the Auxiliary address counter cannot be decremented, and that no mechanism

exits for *pre*-incrementing or *pre*-decrementing any of the address counters.

IV.3.3—6 DirectReg Module

DirectReg consists of two 32-bit latches and a 32-bit shift register. **WordLatch** is loaded on a write to the C register. The data is tranferred to **PostILatch** when the instruction is posted. When execution begins, the data is transferred to the shift register.

Whenever the microcode bit **DirEnHSP1** is asserted, the shift register shifts right one bit. The value in the LSB position of the shifter appears on **ALUDatHSP1** and is combined with the **Sequencer** output **ACmpHSP1** in **OutputLatch**. If the shifter is bumped more than 32 times, the value is sign-extended.

PostILatch also generates the **WrData<0:31>LSP1** inputs to **Sequencer**. When loading microcode, this data is the word of microcode to be written into the microcode store.

DirectReg is meant to be used by writing 32-bit integer data to the C register. This data can be (1) shifted as address or control data into an external device, such as the IGC Port Transmit Controller, via the IGC output **ALUDatHSPR**, (2) sent directly into the EMCs' pixel-ALU's via **ACmpHSPR** (see **OutputLatch**), (3) shifted into the EMC configuration register via **ACmpHSPR**, or (4) loaded into microcode store during IGC initialization.

IV.3.3—7 Serializer Module

The **Serializer** is the most complicated of the IGC modules. Its function may be summarized as follows:

(1) convert the floating point coefficients (A,B,C,D,E,F) supplied with an instruction into bit-serial 2's-complement fixed-point numbers with a specified number of fractional bits

(2) produce the signal **LSBHSP1** which becomes the IGC output **LSBHSPR**, and controls pipelining in the quadratic expression evaluators of the EMC's

(3) produce the signal **CSBHSP1**, which is high when **Serializer** is "busy", and which goes low when the coefficients from the next instruction may be posted.

(4) produce the signal **TRRHSP1**, which passes tokens marking the LSB and MSB of the QEE result, and is used to synchronize overlapped instructions

When a coefficient is loaded into the IGC in the first stage of the instruction pipeline, its mantissa is latched into one of the six **Serializer** sub-modules **CoefSer**, and its signbit is latched into one of the **CoefGate**'s. The exponent is decoded in sub-module **ExpDec**, and the one-hot output of **ExpDec** is also latched into **CoefSer.** The bits of the opcode that control handling of the coefficents are latched into sub-module **ILatch**, and the exponent field of the opcode is also decoded in **ExpDec** and the decoded exponent latched into sub-module **ITok**.

When **PostC<H,L>SP2** is asserted and the coefficients are posted, the mantissa is passed to leafcell **ManReg** (in **CoefSer**), the sign-bit is passed to leafcell **PostCLatch** (in **CoefGate**), and the decoded exponent is passed to the Token Shifter (in **CoefSer**). The decoded exponent field of the opcode is also passed to the Token Shifter in sub-module **ITok**). The out-of-range exponent decoder output, **Tok64LSP2**, is also passed to **CoefGate** and to **CSBLatch**.

At this point, bit-serialization of the coefficients begins. Each instance of **CoefSer** emits a bit-stream representing the corresponding coefficient value in fixed-point unsigned form, and emits a token on ***TokOutLSP2** the timing of which encodes the magnitude of the coefficient. In **CoefGate**, the bit-stream may be 2's-complemented if the sign is negative, and gated to 0 if the corresponding **ABEnHSP2** or **DEFEnHSP2** from **ILatch** is de-asserted or if the exponent was out-of-range (**Tok64LSP2** is asserted), to produce the **Serializer** output ***DatLSP2**. One bit is produced on ***DatLSP2** for every cycle on which the **Serializer** input **Shift<H,L>SP2** is asserted.

From the tokens on **<A-F>TokOutLSP2** and **ITokOutLSP2**, sub-module **CSBLatch** generates **CSBHSP1,** which goes high when the coefficients are posted, and low when the **Serializer** is done with one set of coefficients and a new set may be posted. It also generates the pipelining control signal for the QEE's, **LSBLSP2.**

Sub-module **TRRShifter** generates the signal **TRRHSP1,** which marks the LSB and MSB of the QEE result, and is used to synchronize the QEE operation with the pixel-ALU

and pixel-memory micro-instructions.

—ExpDec Sub-module

ExpDec, the exponent decoder, adds the exponent field of each coefficient, **D<23:30>HSP2**, to a "fractional bits" value, **FBits<0:4>HSP2**, and decodes the result into a "one-hot" signal, **Dec<0:63>LSP1**, and an "out-of-range" signal **Dec64LSP1**. It also contains the register for **FBits<0:4>HSP2**.

The 65 **ExpDec** outputs, **Dec<0:64>LSP1**, are generated from pre-decoded values. An 8-bit adder generates the value to be decoded, **Exp<0:7><H,L>SP2**. For adder outputs in the range i = 128_{10} through 191_{10}, **Dec(191-i)LSP1** is asserted. For adder outputs outside this range, **Dec64LSP1** is asserted; one of **Dec<0:63>LSP1** is also asserted, but its value ends up being ignored if **Dec64LSP1** is also asserted. The A inputs to the adder are the exponent field of the input data word, **D<23:30>HSP2**, and the B inputs are either the fractional bits value **FBits<0:4>HSP2**, or 0's, according to the value of **PWordLSP2**.

The value in the 5-bit register represents the number of fractional bits to be generated when floating-point QEE coefficients are converted to bit-serial fixed-point. The register is loaded by writing the P register with the desired value for number of fractional bits (FBITS) encoded as 129 + FBITS. For a value of FBITS in the legal range 0 - 30, the exponent field is 129 - 159, so D30 will be high; this causes **XWord<H,L>HSP2** to be asserted, and the FBITS register is loaded with the 5-bit value FBITS+1. **PWord** forces the B inputs to the adder to 0, so the adder output is just the exponent field of the P word, 129 + FBITS. So, for a value of FBITS in the legal range 0 to 30, one of the exponent decoder outputs **Dec(62-FBITS)LSP1** will be asserted. A write to the P register causes a value for FBITS to be latched whenever D30 = 1; however, any value for which D30 = 1 but which is outside the valid range for FBITS will cause logical failure of the IGC and must be avoided.

In normal operation, **XWord** is de-asserted and the B input to the adder is contents of the FBITS register, FBITS + 1. The A input to the adder is the exponent field of a coefficient, which is encoded in the form EXPONENT + 127, where EXPONENT is the actual numerical value of the exponent, according to the IEEE standard. The output of the adder is therefore EXPONENT + 128 + FBITS. This adder output is decoded so that exactly one of **Dec<0:63>LSP1** is asserted, and also **Dec64LSP1** is asserted if the value is out-of-range. For values of EXPONENT in the range -FBITS to 63-FBITS, the adder output is

128 to 191, so the decoder output **Dec(63-(EXPONENT+FBITS))LSP1** is asserted. If EXPONENT is outside this range, the adder output is outside the 128 - 191 range, and **Dec64LSP1** is asserted. This means that valid values for coefficients are with exponents in the range -FBITS to 63-FBITS. Any coefficient with an exponent outside this range, including the various "exceptions" defined in the IEEE standard, will cause the decoder output to be **Dec64LSP1** to be asserted, and the coefficient will effectively be 0 (see below). For very small coefficients this is the natural result, since for a number with exponent smaller than -FBITS the correct truncated fixed-point representation is in fact 0. However, care must be taken not to send coefficients above the legal range as non-sensical results will be obtained.

Operation of the exponent adder and decoder is summarized in Table 1.

actual numerical value of exponent	value of D<23:30>	value on decoder inputs Exp<0:7>	index i for which DeciLSP1 is asserted
LOADING FBITS REGISTER: PWordLSP2 = 0, XWordHSP2 = 1			
—	129 + 0	129	62
—	129 + 1	130	61
.	.	.	.
.	.	.	.
.	.	.	.
—	129 + 29	158	33
—	129 + 30	159	32
NORMAL MODE: PWordLSP2= 1, XWordHSP2 = 0			
< -FBITS	< 127-FBITS	< 128	64
-FBITS	127-FBITS	128	63
1-FBITS	128-FBITS	129	62
.	.	.	
.	.	.	
.	.	.	
62-FBITS	189-FBITS	190	1
63-FBITS	190-FBITS	191	0
>63-FBITS	> 190-FBITS	> 191	64

— ILatch Sub-Module

ILatch latches and decodes 4 bits of the opcode which determine how the QEE is to be handled for the instruction. When **IWord<H,L>SP2** is asserted, **D<17-19,22>HSP2** are latched. **D<19:18>HSP2** are called the QEEMode bits, and determine how the QEE is to be used for this instruction: 00 indicates that the instruction does not use the QEE's, 01 indicates that the QEE's are to be used in constant mode, 10 indicates linear mode, and

11 indicates 11 quadratic mode.

The output **IWCHSP2** is asserted if either **D18** or **D19** of the opcode is set, indicating that the instruction is one which uses the QEE's.

The output **ABEnHSP2** is asserted if the A and B coefficient bit-streams are used, that is, if the QEE Mode bits specify either linear or quadratic mode.

Similarly, the output **DEFEnHSP2** is asserted if the D, E, and F coefficient bit-streams are used, that is, if the QEE Mode bits specify quadratic mode.

The output **TRREnLSP2** is asserted if **D17** of the opcode (the TRREn bit, see below) is set, indicating that the instruction is one for which an LSB token should be generated on **TRRHSP1**.

The output **MBIHSP2** is asserted if **D22** of the opcode (the MBI bit, see below) is set, indicating that the instruction is one for which the *full* QEE result must be generated; that is, enough bits of the QEE result must be generated so that the sign-bit is generated at every EMC pixel-processor, rather than just a fixed number of bits being generated. **TRREnLSP2** and **MBIHSP2** are qualified by **IWCHSP2**; **D22** = 1, **D18** = **D19** = 0 is an illegal opcode.

These outputs are used by the other sub-modules of **Serializer**.

—CoefSer (Coefficient Serializer) Sub-Modules (6 on)

A block diagram of **CoefSer** is shown in Figure 2.3.x. There are 6 instances of this sub-module in **Serializer**, one for each of the quadratic coefficients A, B, C , D, E, and F. Each **CoefSer** contains: a 23-bit wide instance of leafcell **WordLatch**; a 23-bit latch, leafcell **ManReg**; a 65-bit wide latch, leafcell **TokLatch**; the Token Shifter, a parallel-loadable shift register 67 to 89 bits in length composed of instances of leafcell **TokStage**; and a pre-charge/evaluate bit-line 24-bits in length. The instances of **CoefSer** differ only in the length of the Token Shifters.

When each coefficient of an instruction is writen to the appropriate register, the 23-bit mantissa field is latched into **ManLatch**, and the decoded exponent, in the form of the

ExpDec outputs **Dec<0:64>LSP1**, is latched into **TokLatch**. When the coefficients are posted, the mantissa is transferred from **ManLatch** into **ManReg**. Simultaneously, the decoded exponent is transferred from **TokLatch** into the Token Shifter. Stages 0 to 63 of the Token Shifter are loaded with the outputs of **TokLatch** corresponding to **ExpDec** outputs **Dec<0:63>LSP1**, and the remaining stages are loaded with 0's. The one logical-high signal which is loaded is called a *token*. For each shift cycle (clock cycle for which **Shift<H,L>SP2** is asserted), this token is shifted one position to the left. When the token reaches the leftmost end of the Token Shifter, the sub-module output **TokOutLSP2** is asserted for one shift cycle.

After **ManReg** and the Token Shifter are loaded, bit-serialization of the coefficients begins. A precharge/evaluate bit-line is used to produce the coefficient bit. This line is precharged high on **Ph2**, and evaluated and latched on **Ph1**. The token in the Token Shifter is a pointer into **ManReg**; if the bit of **ManReg** opposite the token contains a one, the bit-line is evaluated low, and **BitHSP2** is asserted.

ManReg is 23 bits in length, and contains the actual mantissa field of the coefficient. The length of the Token Shifter varies for the various instances of **CoefSer**. Bits 0 to 39 of the Token Shifter, loaded from **Dec<0:39>LSP1** (via **TokLatch**), lie to the right of **ManReg**; the next 22 bits, bits 40 to 62, are aligned with **ManReg**, with bit 40 corresponding to the LSB of the mantissa; bit 63 corresponds to the understood leading 1 of the mantissa. The additional bits of the Token Shifter lie to the left of the MSB-end of **ManReg**, 3 for the C coefficient, 13 for A and B, and 25 for D, E, and F.

If the coefficient is in range, 0's and 1's are produced as the token is shifted to the left in the Token Shifter. When the token is shifted beyond the left-most bit of **ManReg**, only 0's are produced; the reflects the fact that sign-extension in a sign-magnitude representation is realized simply by adding trailing 0's. If the coefficient is in range, but is large enough that the exponent is greater than 24-FBITS, then the token is placed in the Token Shifter to the right of **ManReg**; thus the first few bits of the coefficient will be 0's. This reflects the fact that precision is lost when converting very large numbers to fixed-point.

When the coefficient is zero or out of range and the exponent decoder output **Dec64LSP1** is asserted, the coefficient if forced to zero in **CoefGate** (see below), so the location of the token in the Token Shifter is unimportant.

—CoefGate Sub-Modules (6 on)

Sub-module **CoefGate** performs two's compementing on the coefficient bit-stream if the sign is negative, introduces the separator bit required by the QEE's, gates the coefficient to zero if the exponent was out-of-range or if required by the QEEMode setting, and delays the bit-stream by one additional clock cycle.

When a coefficient register is loaded, the signbit of the coefficient is latched into leafcell **WordLatch**. When the coefficients are posted, the signbit is transferred to one instance of **PostCLatch**, and the other instance of **PostCLatch** is loaded with the logical-AND of the latched version **Tok64LSP2** of the out-of-range exponent decoder output, and one of the **ILatch** outputs **ABEnHSP2**, **DEFEnHSP2**, or 0 (depending on which instance of **CoefGate**).

The bit-stream from **CoefSer**, **BitHSP2**, is two's complemented using the following algorithm: "pass the bits unchanged up to and including the first 1 that is seen, then invert all the remaining bits". **SOneHSP2** is cleared when the coefficients are posted, and remains low until the first 1 is seen. If the sign-bit was 1, then **InvLSP2** is asserted when **SOneHSP2** goes high, and the remaining bits of the coefficient will be inverted in the exclusive-OR gate.

The bit-stream next passes through an instance of **TokStage**, which injects the 0 separator bit. Finally, it is AND-ed with the output of the nullify **PostCLatch**, and passes into an instance of **ShiftStage** which delays the bit-stream by one more clock cycle. This cycle of delay, plus the cycle introduced by the separator-bit injector, insure that the token on **LSBLSP2** preceeds the LSB's of the coefficients (on **<A-F>DatLSP2**) by 2 clock cycles, as required by the QEE's on the EMC chips.

—ITok Sub-Module

ITok contains just an instance of the 65-bit wide latch **TokLatch**, and a Token Shifter like the Token Shifters in the 6 **CoefSer** sub-modules. This I Token Shifter is 74 bits in length. Stages 0 to 64 are loaded with the **TokLatch** outputs corresponding to **Dec<0:64>LSP1**, and the 9 stages to the left of stage 64 are loaded with 0's. This is slightly different from the coefficient token shifters, which are loaded only with the TokLatch outputs corresponding to **Dec<0:63>LSP1**.

As with the coefficient token shifters, the token is shifted to the left whenever **Shift<H,L>SP2** is asserted, and the signal **ITokOutLSP2** is asserted for one shift cycle when the token reaches the leftmost end of the shifter.

The token in **ITok** does not act as a pointer, as with the coefficient token shifters pointing to the coefficient mantissas; it is just used to count the number of coefficient bits to be generated, as described below (sub-module **CSBLatch**).

—CSBLatch Sub-Module (Determining Length of Coefficient Bit-Streams)

CSBLatch generates the signal **CSBHSP1**; it goes high one shift cycle (a cycle during which **Shift<H,L>SP2** is asserted) after a set of coefficients is posted (when the LSB appears on **BitHSP2**), remains high while the set of coefficients is bit-serialized, and goes low again one clock cycle after the last bit of the coefficients to be generated appears on ***BitHSP2**. Thus, **CSBHSP1** stays high for the same number of shift cycles as the number of bits of the coefficients to be generated, not including the separator bit. When **CSBHSP1** goes low, **Controller** is enabled to assert **PostC<H,L>SP2** and post the coefficients of the next instruction; thus, the timing of **CSBHSP1** determines the number of bits generated in the bit-serial representation of the coefficients. The number of coefficient bits to be generated is necessarily the same for all six coefficients in any given coefficient set, and is identically equal to the number of bits of QEE result to be generated in the EMC's.

Several factors determine the number of bits that should be generated when the coefficients are bit-serialized (and therefore the number of bits of QEE result to be generated).

First, the mechanism used to allow pipelining of coefficient sets in the QEE's requires that the LSB tokens be at least 12 cycles apart, that is, one more than the longest 'sticky register' in the EMC's. Because of the 0 separator bit between coefficient bit-streams, this means that coefficient bit-streams must be at least 11 bits in length (including the fractional bits, if any). This means that **CSBHSP1** always goes high for at least 11 shift cycles, and always goes low for at least one shift cycle between coefficient sets.

Second, many IGC instructions may require that a certain number of bits of the QEE result (to the left of the radix point) be generated. For example, an instruction which loads the

QEE result into a segment of pixel-memory LEN bits in length requires that LEN bits of the integer part of the QEE result be generated, regardless of how small the magnitude of the coefficients and QEE result may be; however, this instruction would not require that any higher-order bits of the QEE result be generated, even if the coefficients and QEE results were very large.

Third, some IGC instructions require enough bits of the QEE result be generated to guarantee that the sign-bit is generated at every pixel-processor in every EMC. For example, the instruction to scan convert a polygon edge requires that the sign-bit of the QEE result be generated at every pixel, so that each pixel can evaluate whether it is inside or outside the polygon. For such instructions, the magnitude of the QEE results, and hence the number of bits which must be generated, depends on the magnitude of both the quadratic coefficients as well as the maximum values of the pixel coordinates x and y. Since x and y lie in the interval 0 - 2047 (x and y are 11-bit numbers), then the maximum possible length for the QEE result

$$Q(x,y) = Dx^2 + Exy + Fy^2 + Ax + By + C \qquad (1)$$

is given by

$$L_{TR} = MAX(l_D + 22, l_E + 22, l_F + 22, l_A + 10, l_B + 10, l_C) + 3 + 1 \qquad (2)$$

where l_A through l_F refer to the length of each coefficient, not including sign bit. This accounts for D, E, and F being multiplied by 22-bit numbers (x^2, xy or y^2), and A and B being multiplied by 10-bit numbers. Since 6 terms are added together, 3 bits must be added to account for overflow. Since 3 overflow bits allow for 8 terms added together, Ax and By may be allowed to 'count' twice, which is why 10 is added to l_A and l_B rather than 11. The additional 1 is added to allow for the sign-bit.

Additionally, many instructions require that the QEE result be extended to the sign-bit or to some minimum number of bits, whichever is larger.

Thus, the number of coefficient and QEE result bits to be generated depends on the absolute minimum of 11, the magnitudes of the coefficients, the magnitude of the exponent field of the opcode, and the bits of the opcode which determine how the coefficients are to be handled.

CSBHSP1 is generated as a 7-way logical-OR of 7 *busy* latches, whose outputs are

<A,B,C,D,E,F,I>BsyHSP1. When a set of coefficients is posted, one or more of these ***BsyHSP1** may be set. Any instruction which uses the QEE's (**IWCHSP2** is asserted) causes **IBsyHSP1** to be set. If the instruction requires the full QEE result (**MBIHSP2** is asserted), then the 6 busy latches for the coefficients may also be set. If the QEE mode is *constant*, then **ABEnHSP2** = **DEFEnHSP2** = 0 and only **CBsyHSP1** is set; if the QEE mode is *linear*, then **ABEnHSP2** = 1 and **DEFEnHSP2** = 0, so **ABsySHP1**, **BBsyHSP1**, and **CBsyHSP1** are set; and if the QEE mode is *quadratic*, all six **<A-F>BsyHSP1** are set. The setting of each of **<A-F>BsyHSP1** is qualified by whether that coefficient was in range, that is, if the corresponding latched out-of-range signal **Tok64LSP2** is not asserted.

Each of these busy signals is cleared when the token pops out of the corresponding Token Shifter, that is, when the corresponding signal ***TokLSP2** goes low for one cycle; once all of the ***BsyHSP1** signals have been cleared, **CSBHSP1** goes low. The number of shift cycles **CSBHSP1** stays high is equal to

$$TS_{length} - i + 1 \qquad (3)$$

where TS_{length} is the overall length of the limiting token shifter and i is the index of the stage where the token is latched when the coefficients are posted. For the 6 coefficient Token Shifters, i is the value 0 - 63 for which **Dec*i*LSP1** is asserted; if **Dec64LSP1** is asserted (the coefficient is out-of-range), then i is unimportant since the corresponding **Tok64LSP2** will be asserted and the **BsyHSP1** signal is never set. For the I Token Shifter, i is the value 0 - 64 for which **Dec*i*LSP1** is asserted, and the larger value 64 is used if **Dec64LSP1** is asserted (in this case both **Dec64LSP1** and one of the **Dec<0:63>LSP1** are asserted).

For instructions which require that a minimum number of bits of the integer part of the coefficients (FNI) be generated (independent of the coefficient magnitudes), the exponent field of the opcode should be set to FNI + 115. The exponent adder output is therefore FN + 116 (where FN = FNI + FBITS is the total number of bits of the QEE result to be formed including the fractional bits), so **Dec(75-FN)LSP1** is asserted. The I Token Shifter is 74 bit long, so Eq. 3 does in fact give FN coefficient bits generated. The maximum value for FNI is therefore 75-FBITS, since larger values are decoded as out-of-range. Since i <= 64 for the I Token Shifter, Eq. 3 also gives that at least 11 bits of the coefficients are generated, as required by the QEE's.

The other Token Shifters, for the 6 coefficients, are sized so that if **MBIHSP2** is set, the coefficients are sign-extended to the point that the sign-bit of the quadratic expression value at all pixel processors is sure to be generated. The portion of each coefficient Token Shifter to the left of **ManReg** is sized according to the corresponding term in Eq. 2, so that the token emerges on ***TokLSP2** when enough bits of QEE result have been generated to cover that coefficient's contribution to the MAX term in Eq. 2. From the Exponent Decoder section, we know that for in-range coefficients that the decoder output **DeciLSP1** is asserted, where i = 63-(EXPONENT+FBITS) and EXPONENT is the actual numerical value of the exponent. Since the length of a coefficient, not including signbit, is given by l = 1 + EXPONENT + FBITS, the decoder output **Dec(64 - l)SP1** is asserted. Thus i = 64 - l, so the number of coefficient bits for the limiting Token Shifter is TS_{length} - 63 + l. If D is the limiting coefficient, then TS_{length} is 89, so the number of coefficient bits is l_D + 26, which corresponds to the value of Eq. 2 if l_D + 22 dominates the MAX term. If a coefficient is out-of-range, the corresponding ***BsyHSP1** is never set, so that timing of the ***TokLSP2** is unimportant.

The timing for **CSBHSP1** is shown in Figure XXX.

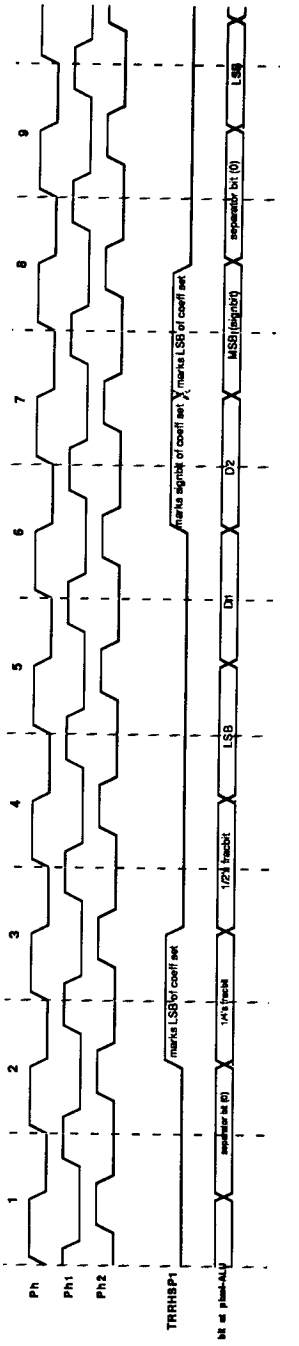
— Sub-Module TRRShifter

TRRShifter is a shift register which generates the output **TRRHSP1**, which goes to the branch control logic in **Sequencer**, and to the Shift control logic in **Controller**; this signal is used to synchronize the serialization of coefficients in **Serializer** and generation of the QEE results in the EMC's, with the execution of microcode in **Sequencer** which generates the pixel-ALU and pixel-memory micro-instructions for the EMC's.

For any instruction which uses the QEE, that is for which the QEE Mode bits of the opcode are not 00, and for which the TRREn bit of the opcode is set, **TRRHSP1** goes high 2 shift cycles prior to the first integer bit of the QEE result appearing at the 'tree' input to the pixel-ALU's; this is necessary for instructions which combine pixel-memory with the QEE result, so that the first bit of pixel-memory can be fetched in time to arrive at the pixel-ALU on the same microcycle on which the LSB of the QEE result arrives. **TRRHSP1** stays high for exactly one shift cycle. Since **TRRHSP1** may appear when the sequencer is executing the previous instruction, **Controller** may cause **ShiftHSP2** to be inhibited when **TRRHSP1** appears, essentially freezing the **Serializer** and QEE's at the point where the LSB of the integer part of the QEE result is at the pixel-ALU. When the microcode for the instruction begins executing, or if it was already executing, it tests for

TRRHSP1 and proceeds if or when **TRRHSP1** is asserted.

For an instruction which requires the sign-bit of the QEE result to be generated, that is for which the MBI bit of the opcode is set, **TRRHSP1** goes high one shift cycle prior to the sign-bit (or MSB) of the QEE result appearing at the 'tree' input to the pixel-ALU and again stays high for exactly one shift cycle.



Thus, for many instructions, two tokens will appear on **TRRHSP1**, and they overlap, although they may be on successive cycles. This is done in writing mi insuring that the FNI field is at least 1 for any instruction which uses the QEE's

TRRShifter contains a 31-bit wide instance of **TokLatch**, which latches **Dec<32:62>LSP1** when **XWord<H,L>SP1** is asserted (a new value for n fractional bits, FBITS, is loaded).

The shifter proper contains 31 loadable stages (each an instance of leafcell **Tol** 24 shift-only stages (each an instance of leafcell **ShiftStage**). When a set of cc posted for which the QEE is used and TRREn is asserted, **TRREnLSP2** is ass token is loaded into one of the 31 loadable stages (the position of the token con FBITS). For instructions which do not use the QEE's or do not have the TRRE bit set, **TRREnLSP2** is not asserted and no token is loaded into the shifter wh **PostC<H,L>SP2** is asserted.

The token emerges from the loadable section and is OR-ed with **MsInLSP2** fr module **CSBLatch. MsInLSP2** is asserted for one shift cycle when **CSBH** the high-to-low transition. This occurs when the last bit of the coefficients for a instruction is serialized. The one cycle pulse on **MsInLSP2** injects a token intc only portion of **TRRShifter**.

Eventually, the one (or possibly two) tokens appear at the right-hand end of the cause the one shift cycle pulses on **TRRHSP1**.

IV.3.3—8 OutputLatch Module

OutputLatch contains the output pads for the IGC as well as some logic. All c are of the same type, a simple unclocked output buffer.

The coefficient bit-stream outputs, **<A-F>DatHSPR**, and the pipeline control, **LSBHSPR**, are generated by delaying the corresponding outputs of **Serializer** by one and a half clock cycles.

The QEE shift control, **TrStHSPR**, is generated by delaying the **ShiftHSP2** output of **Controller** by two and a half cycles (NOTE: should be one and a half, but an extra cycle was added because the EMC is missing a latch for the **TrStHLPR** input).

The pixel-ALU control signals **AGtsTHSPR**, **AGtsSHSPR**, **BGtsMHSPR**, **BGtsEHSPR**, **BCmpHSPR**, and **CGtsCHSPR**, are generated by delaying the corresponding outputs of **Sequencer** by two clock cycles.

The pixel-ALU control **ACmpHSPR** is normally generated by delaying the corresponding **Sequencer** output. However, when the **Sequencer** output **DirEnHSP1** is asserted, enabling the "direct-to-ALU" mode, **ACmpHSPR** is driven by the output of **DirectReg**, **ALUDatHSP1**; the bit-stream from **DirectReg** is normally complemented, since the sense of the pixel-ALU output is opposite that of the ACmp EMC input (if the B and C pixel-ALU inputs are 0), however, it may be uncomplemented if the **Sequencer** output **ACmpHSP1** is asserted. In other words, if DirEnHSP1 = 1, then **ACmpHSPR** becomes the logical exclusive-NOR of **ACmpHSP1** and **ALUDatHSP1**.

The output of **DirectReg** is also available as the IGC output **ALUDatHSPR**.

The ALU controls **LdCHSPR**, **LdEHSPR**, **CCmpHSPR**, and the pixel-memory write enable **MWrtHSPR**, also are generated by delaying the corresponding output of **Sequencer** by two cycles. However, these outputs are forced to 0 when **RModeLSP2** is asserted. This prevents the state of the EMC's from being modified when the IGC is in RMode during an "on-the-fly" microcode load. LdC = 0 and LdE = 0 prevent the pixel-ALUs' Carry and Enable registers from being altered, MWrt = 0 prevents pixel-memory from being overwritten, and CCmp = 0 prevents the QEE Configuration registers from being altered.

The pixel-memory address outputs are generated in pairs, **AddrL<0:7>HSPR** and **AddrH<0:7>HSPR**. In normal operation, these signals are the same, and are generated by delaying the correspoding outputs of **PixMemAddr** by one cycle. However, when a redundant ALU code is given, and when ACmpHSP1 = 1, the signal **PairHSP1** is

asserted, which causes the signals **AddrH<0:7>HSPR** to be the complements of the corresponding **AddrL<0:7>HSPR**. This is useful is addressing the configuration register of a specific EMC, since the EMC configuration logic is only activated if its address inputs are all 0's.

The six external device strobe signals **ExtOp<0:5>HSPR** are generated using the redundant ALU codes. The redundant codes are indexed by

(AGtsTHSP1 * 4) + (AGtsSHSP1 * 2) + (ACmpHSP1).

ExtOp7 and ExtOp8 are not defined, since for these redundant codes **AGtsTHSP1** = **AGtsSHSP1** = 1, which causes the configuration register of an EMC to be activated, provided its address inputs **Addr<0:7>HSPR** are all 0's.

AGtsTHSP1	AGtsSHSP1	ACmpHSP1	ACTION
0	0	0	ExtOp0HSPR asserted, address lines same
0	0	1	ExtOp1HSPR asserted, address lines H,L paired
0	1	0	ExtOp2HSPR asserted, address lines same
0	1	1	ExtOp3HSPR asserted, address lines H,L paired
1	0	0	ExtOp4HSPR asserted, address lines same
1	0	1	ExtOp5HSPR asserted, address lines H,L paired
1	1	0	EMC config mode active, address lines same
1	1	1	EMC config mode active, address lines H,L paired