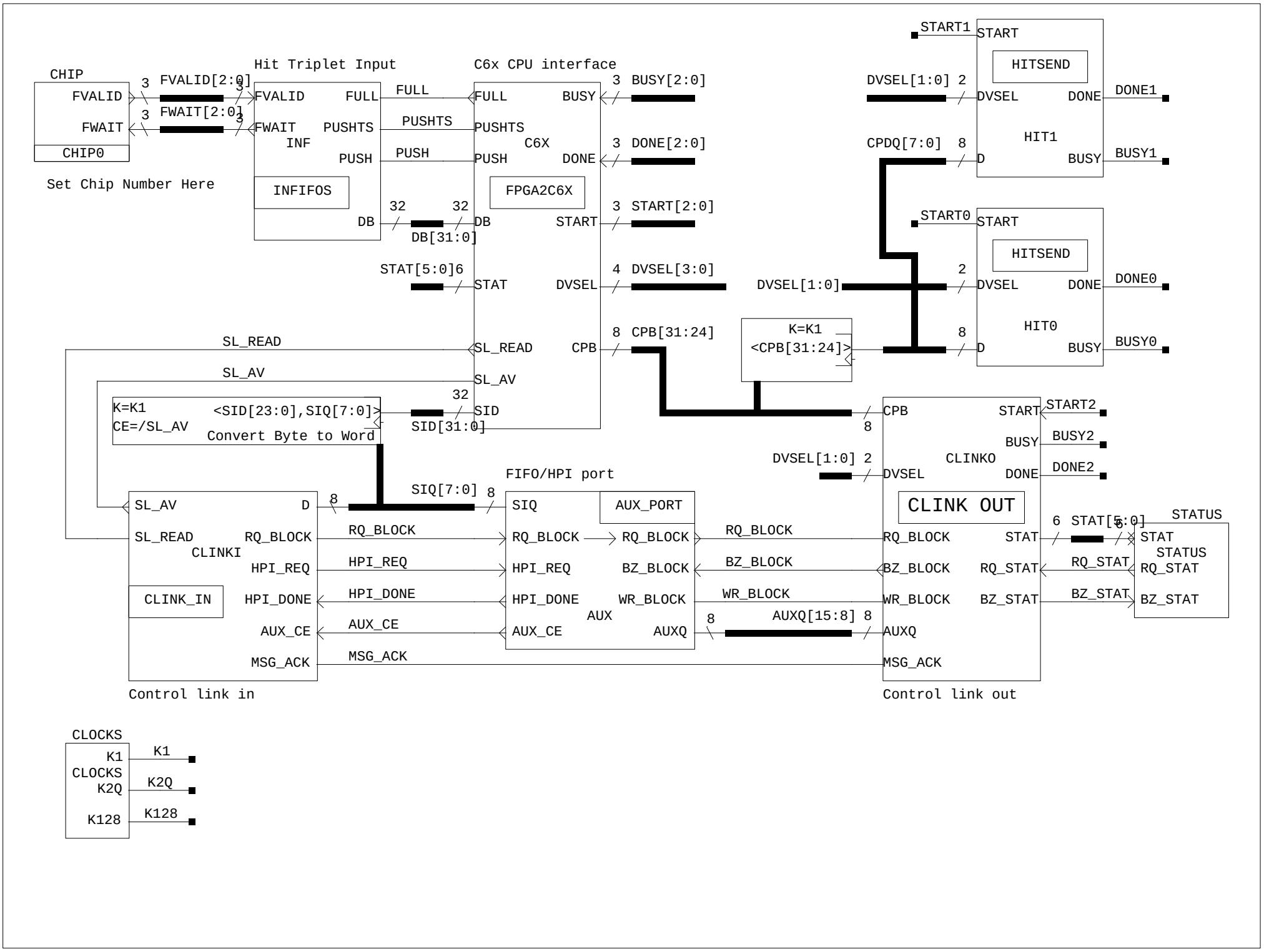
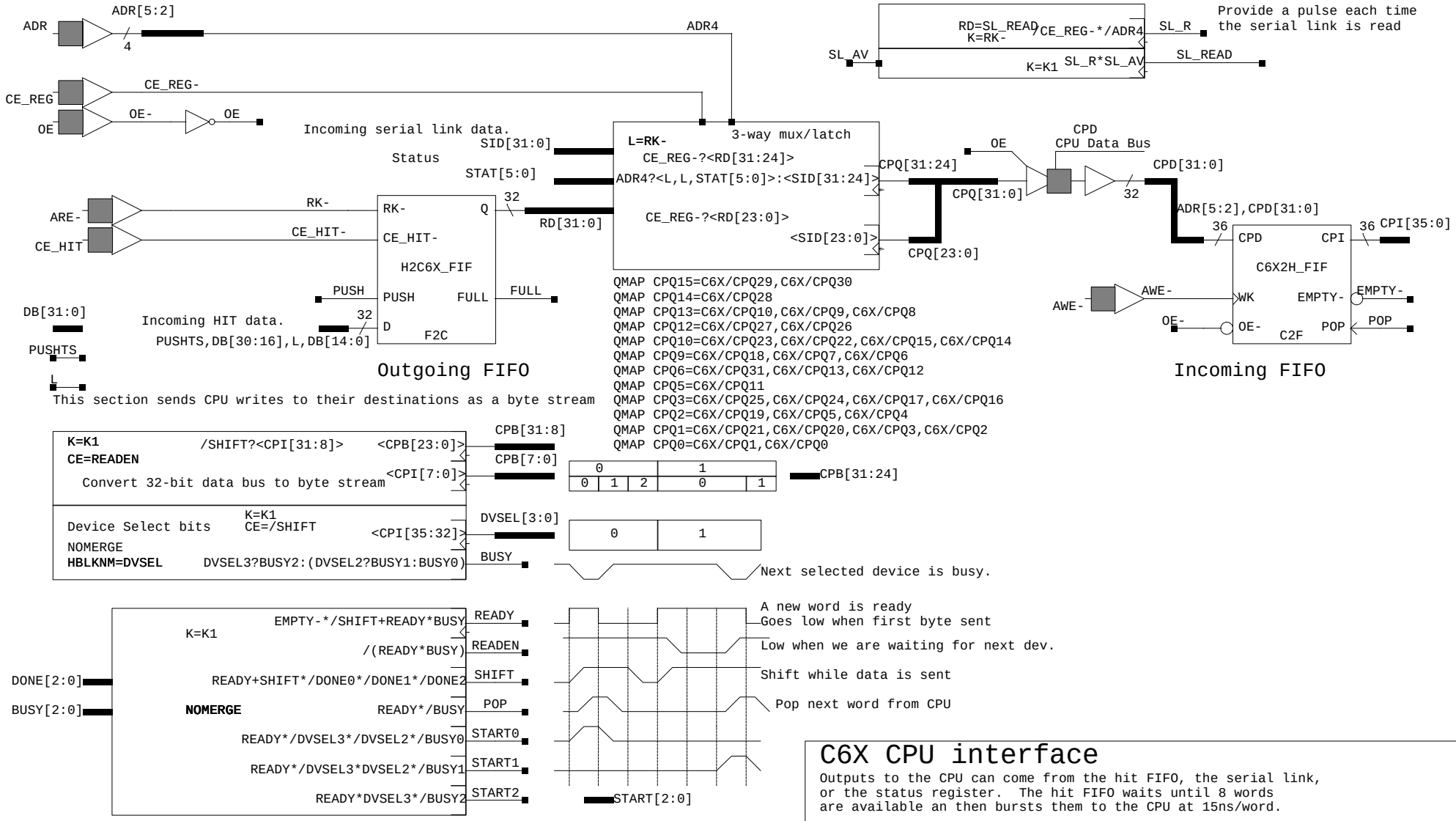
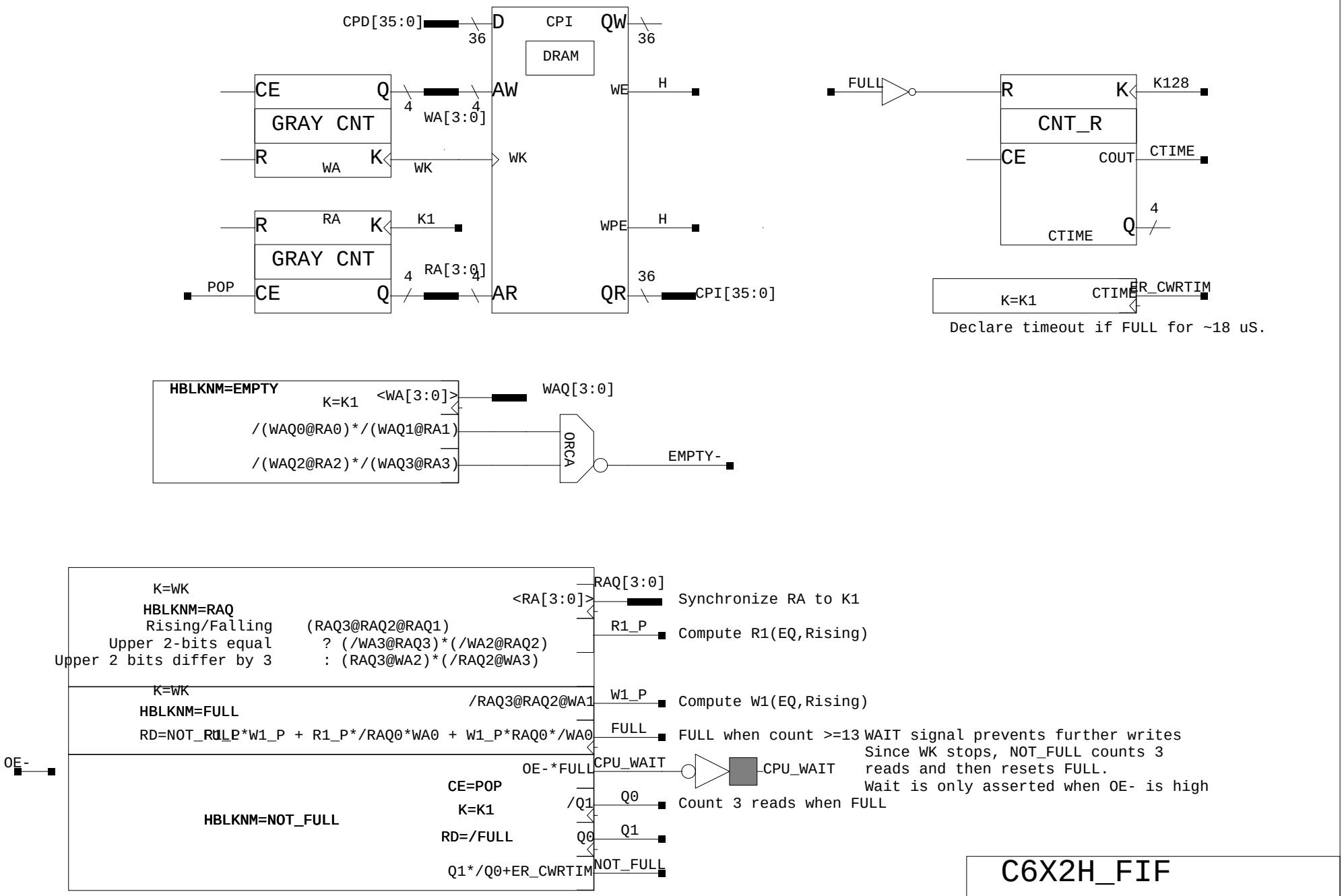

CHIP
FVALID
FWAIT
CHIP0
Set Chip Number Here
3
FVALID[2:0]
FWAIT[2:0]
Hit Triplet Input
FVALID
FWAIT
FULL
PUSHTS
PUSH
INF
INFIFOS
DB
32
DB[31:0]
FULL
PUSHTS
PUSH
C6x CPU interface
C6X
FPGA2C6X
BUSY
DONE
START
DVSEL
CPB
BUSY[2:0]
DONE[2:0]
START[2:0]
DVSEL[3:0]
CPB[31:24]
STAT[5:0]6
STAT
SL_READ
SL_AV
SID
SID[31:0]
K=K1
CE=/SL_AV
<SID[23:0],SIQ[7:0]>
Convert Byte to Word
SIQ[7:0]
8
FIFO/HPI port
SIQ
AUX_PORT
RQ_BLOCK
BZ_BLOCK
WR_BLOCK
AUX
AUXQ
AUXQ[15:8]
HPI_REQ
HPI_DONE
AUX_CE
MSG_ACK
CLINKI
CLINK_IN
Control link in
K=K1
<CPB[31:24]>
START1
START0
HITSEND
HIT1
HIT0
DVSEL[1:0]
CPDQ[7:0]
D
DONE1
BUSY1
DONE0
BUSY0
CLINKO
CLINK OUT
STAT
RQ_STAT
BZ_STAT
STATUS
START2
BUSY2
DONE2
Control link out
CLOCKS
K1
K2Q
K128



C6X CPU interface

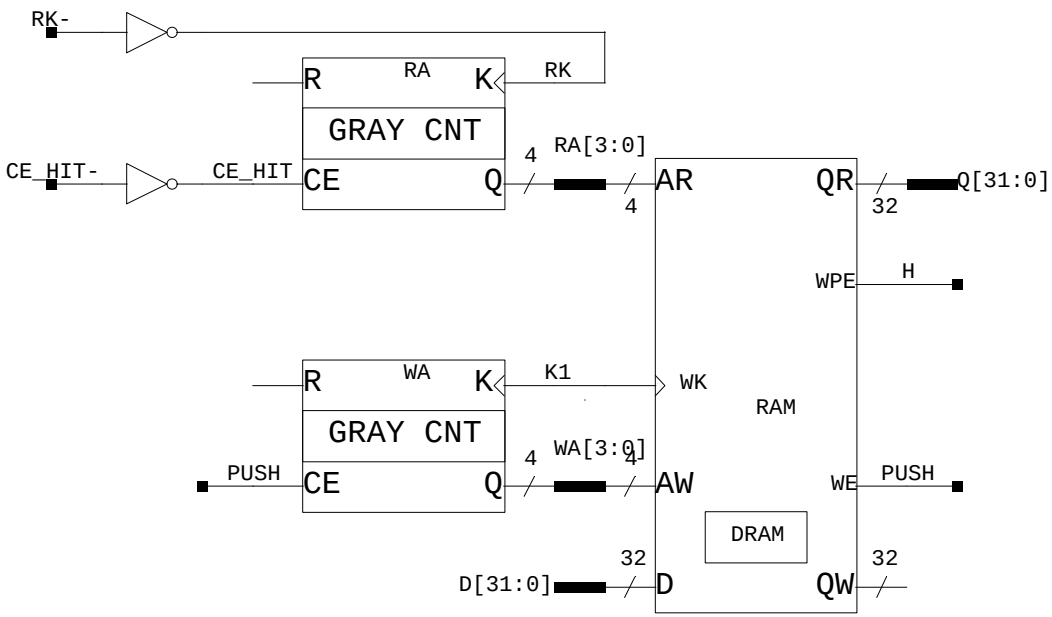
Outputs to the CPU can come from the hit FIFO, the serial link, or the status register. The hit FIFO waits until 8 words are available an then bursts them to the CPU at 15ns/word.

The CPU writes data to the input FIFO, where it gets byte-serialized and sent to the hit-out ports, or to the serial link.



C6X2H_FIF

CPU Writes through this FIFO



K-Maps for 4 cases

		R[1:0] Rising (R3=R2)				R[1:0] Falling (R3@R2)			
	W[1:0] / R[1:0]	00	01	11	10	00	01	11	10
R[3:2]=W[3:2]	00	0	1	2	3	0	15	14	13
	01	15	0	1	2	1	0	15	14
	11	14	15	0	1	2	1	0	15
	10	13	14	15	0	3	2	1	0
W[3:2]-R[3:2]=3	00	15	14	13	12	9	10	11	12
	01	14	13	12	11	10	11	12	13
	11	13	12	11	10	11	12	13	14
	10	12	11	10	9	12	13	14	15

EQR	R1_P= R1	W1_P=/W1
EQF	R1_P=/R1	W1_P= W1
E1R	R1_P=/R1	W1_P=/W1
E1F	R1_P= R1	W1_P= W1

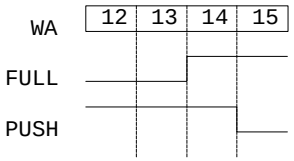
K=K1
HBLKNM=RAQ
Rising/Falling (RAQ3@RAQ2@RAQ1)
Upper 2-bits equal ? (/WA3@RAQ3)*(/WA2@RAQ2)
Upper 2 bits differ by 3 : (RAQ3@WA2)*(/RAQ2@WA3)

RAQ[3:0] <RA[3:0]> — Synchronize RA to K1

R1_P — Compute R1(EQ,Rising)

K=K1
HBLKNM=FULL
/RAQ3@RAQ2@WA1 — W1_P — Compute W1(EQ,Rising)

R1_P*W1_P + R1_P*/RAQ0*WA0 + W1_P*RAQ0*/WA0 — FULL — FULL when count >=13



K=K1

(RA3@RA2)+RA1+RA0 — SENDING — True for 7 transfers.

/SENDING+REQ_EN*/START — REQ_EN — Synchronize Sending to K1

REQ_EN*SENDING — START — Pulse at the start of a new block

/START*(DMA_REQ+HALF@WA3) — DMA_REQ — Request next DMA

START@HALF — HALF — Which half is next to transfer.

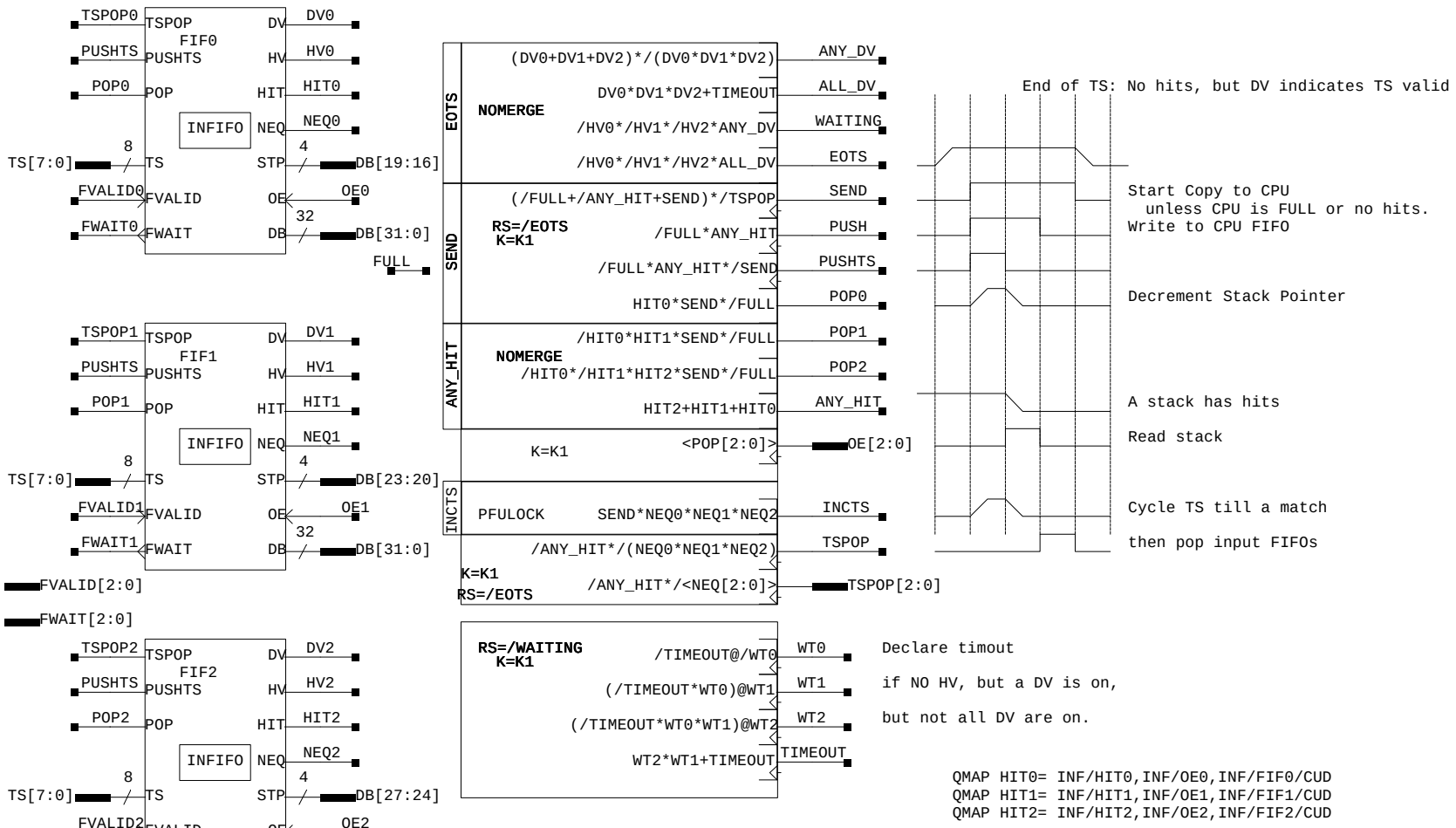
Note that DMA_REQ for the next block can happen while the current block is being sent.

H2C6X_FIFO

Generates read pointer and write pointer using gray code counters.

Asserts FULL when W-R >= 13

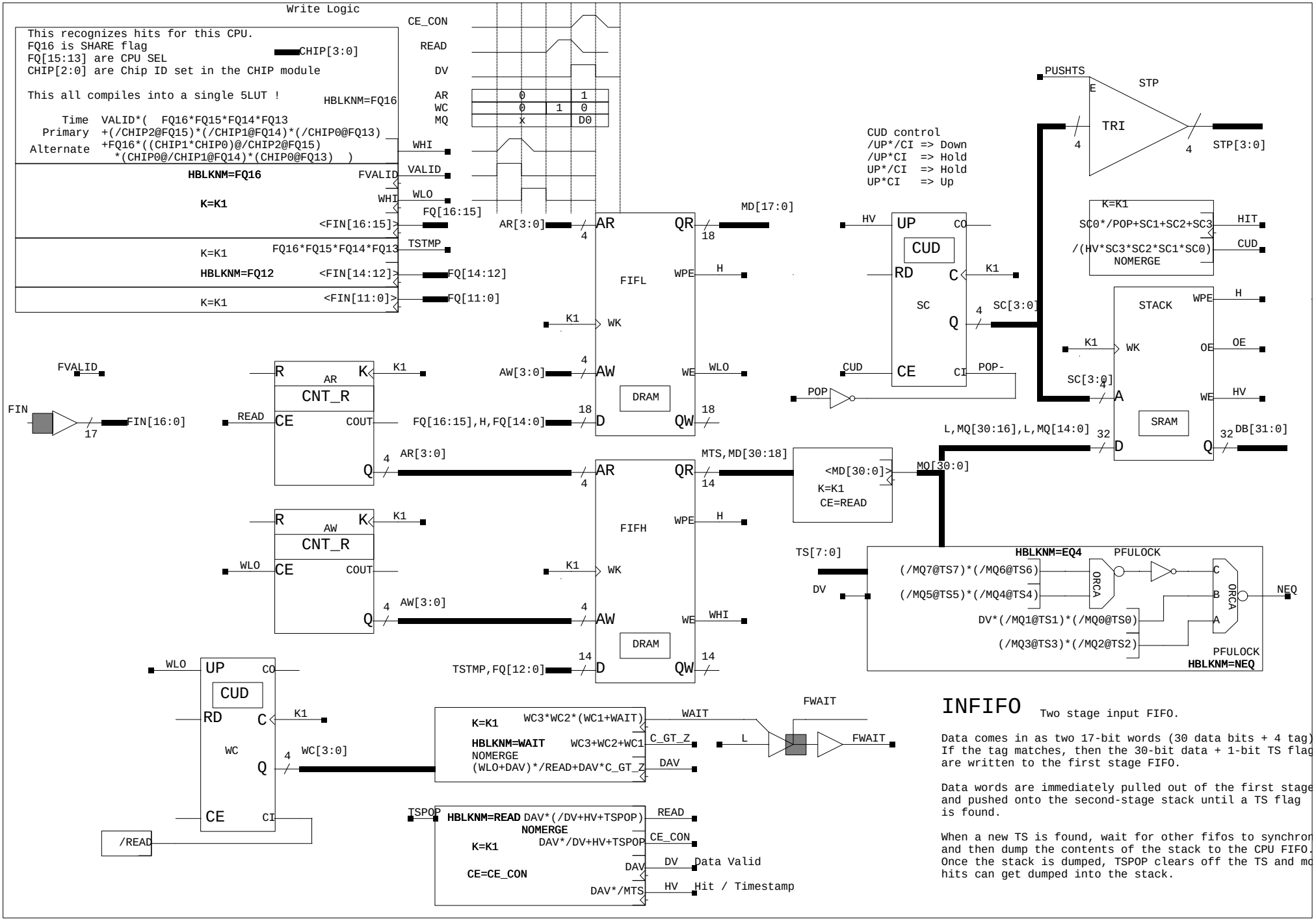
Burst data to CPU in 8-word blocks



INFIFOS

Merge data from 3 FIFOS.

Each FIFO fills its stack till it encounters a TS marker.
When all three have found a new TS (or a timeout occurs),
write the current TS and hit counts to the CPU FIFO.
Then copy each stack to the CPU FIFO.
At the same time, increment the local TS counter till
it matches any one of the timestamps at the top of the
input FIFOS.
POP all FIFOs that match and begin filling the stacks again.



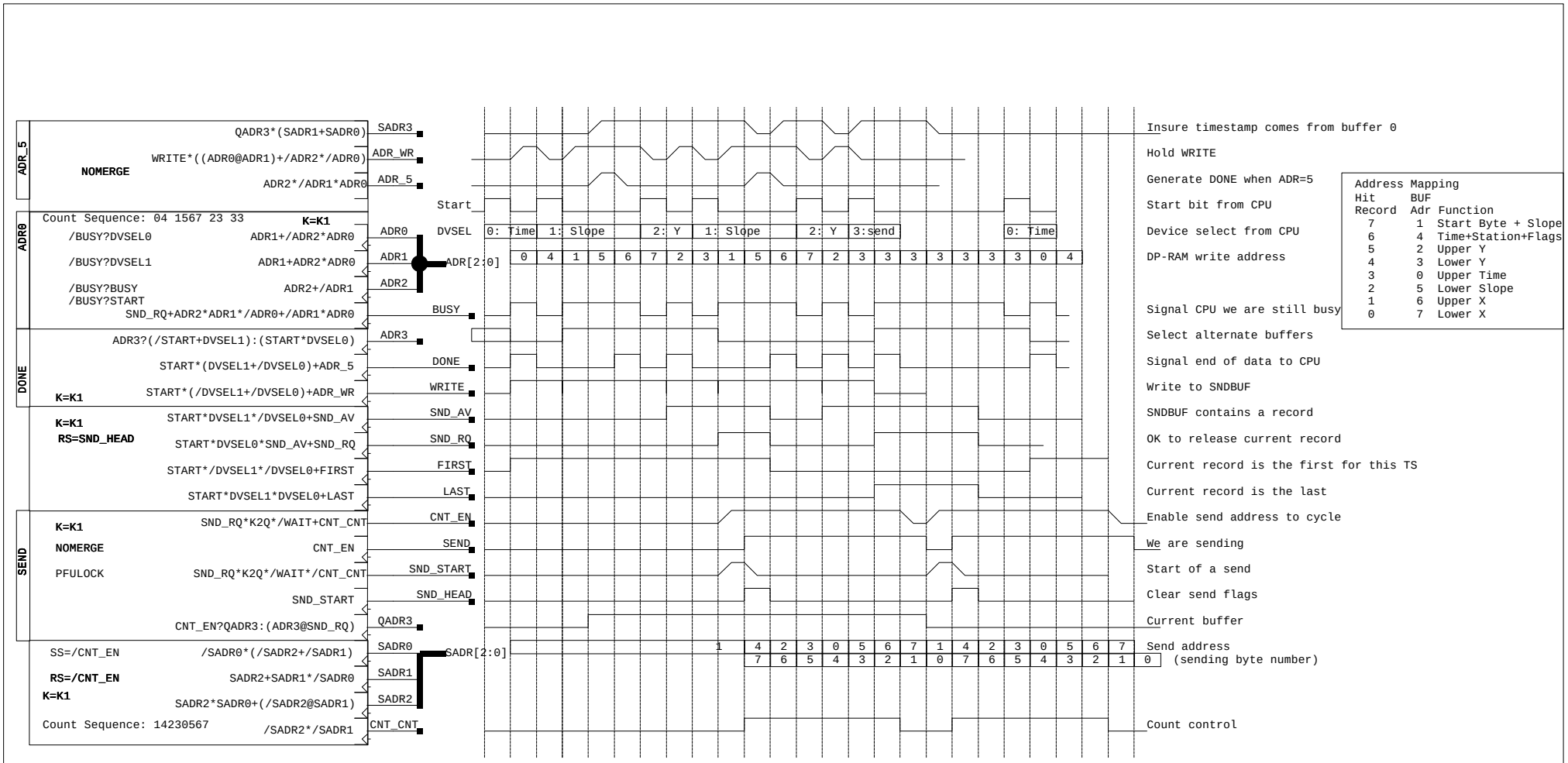
INFIFO

Two stage input FIFO.

Data comes in as two 17-bit words (30 data bits + 4 tag)
If the tag matches, then the 30-bit data + 1-bit TS flag
are written to the first stage FIFO.

Data words are immediately pulled out of the first stage
and pushed onto the second-stage stack until a TS flag
is found.

When a new TS is found, wait for other fifos to synchron
and then dump the contents of the stack to the CPU FIFO.
Once the stack is dumped, TSPOP clears off the TS and mo
hits can get dumped into the stack.

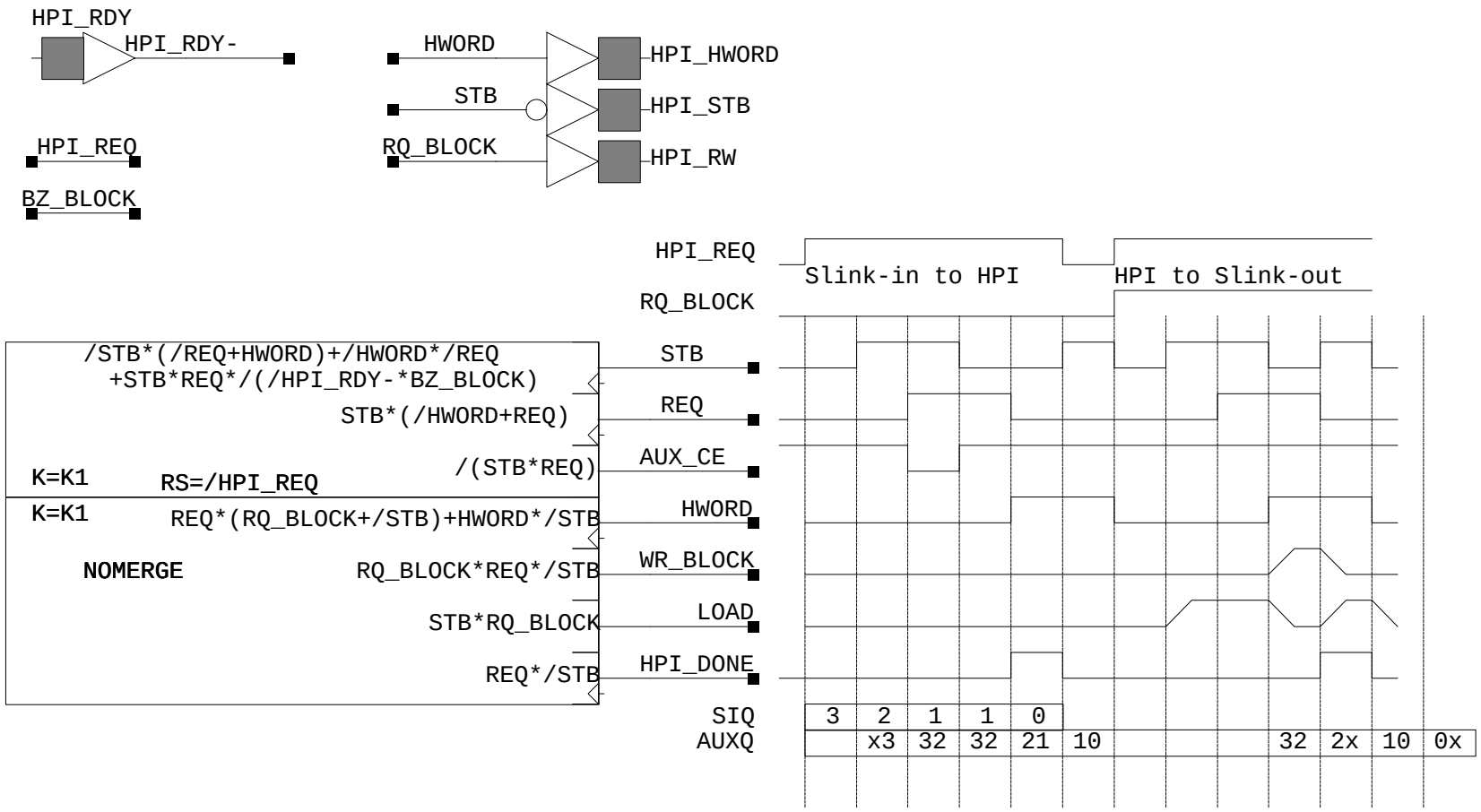


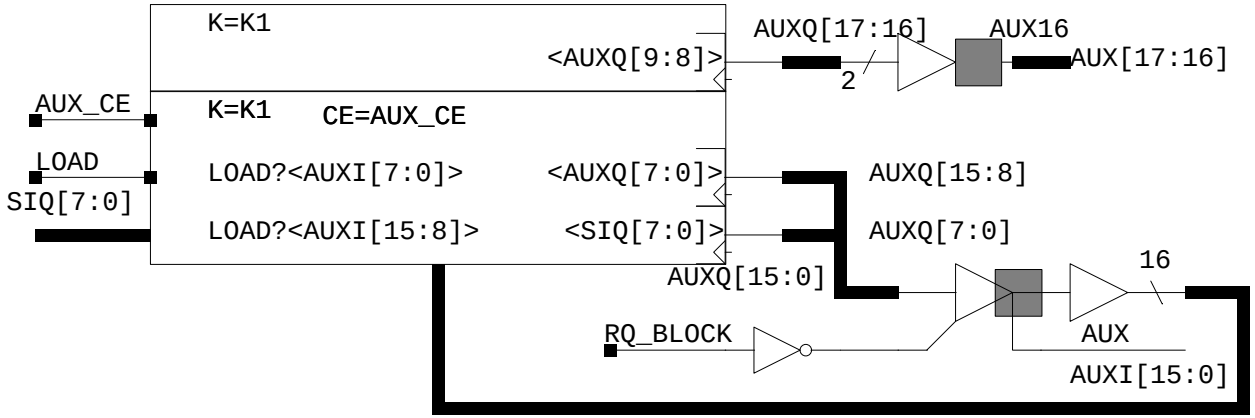
Hit Sender

The CPU sends hits by writing words to 4 registers:

- 0: Time 10 bit timestamp + 3 station + 3 flags
- 1: Slope 13-bit slope (If pending hit, send it)
- 2: Y 16-bit Y coordinate
- 3: End End of timestamp, send pending hit

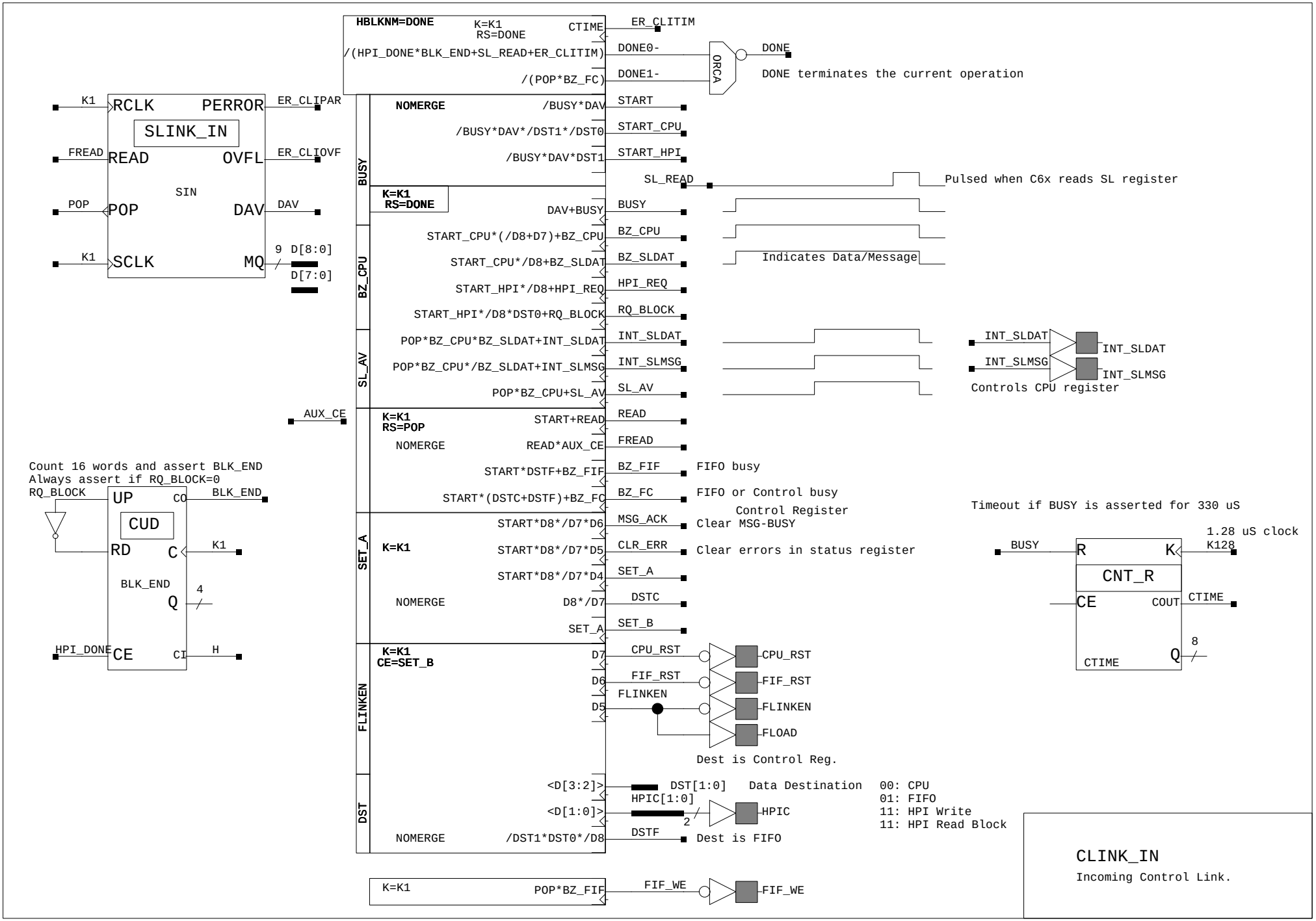
The registers get written as a stream of bytes.
An entire hit record is assembled in SNDBUF, then sent when the CPU begins writing the next.
This pipelining allows the First/Last bits to be set properly in the outgoing byte stream.





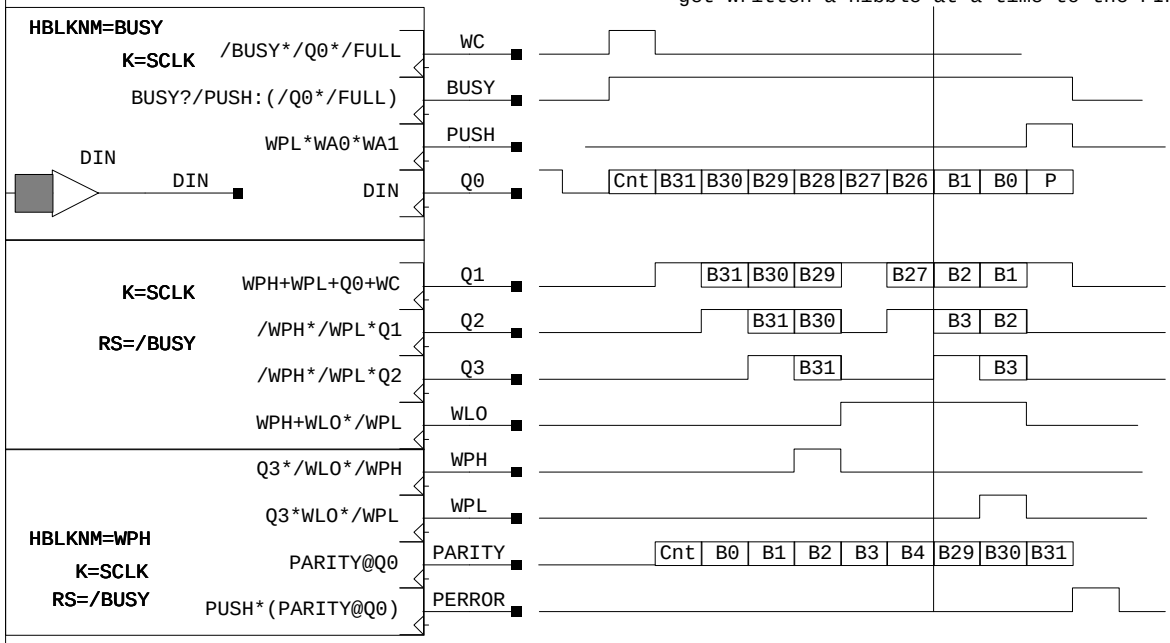
AUX _PORT

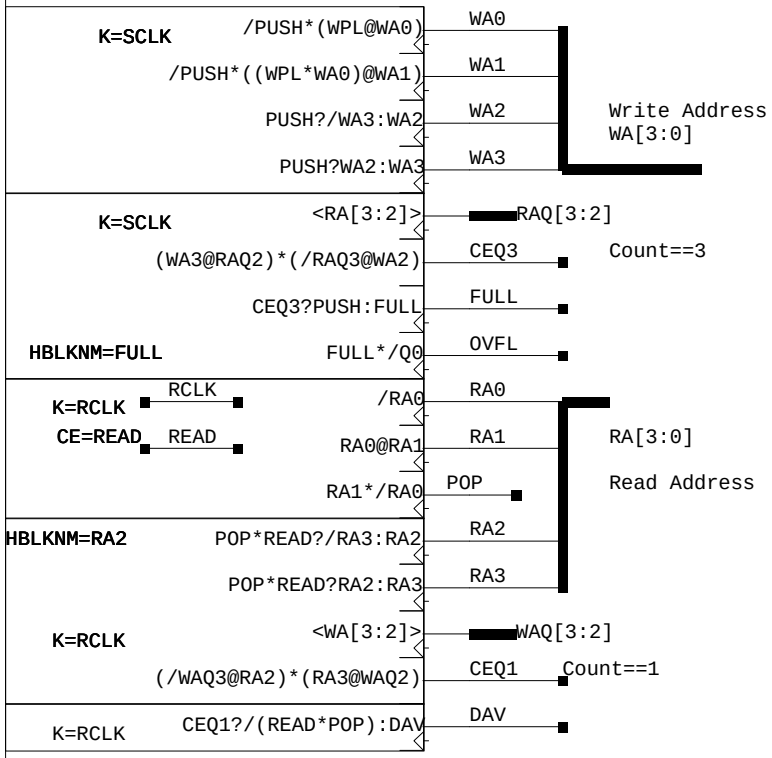
Transfers data between the Control link and HPI/FIFO


HBLKNM=DONE
K=K1
RS=DONE
CTIME
ER_CLITIM
/(HPI_DONE*BLK_END+SL_READ+ER_CLITIM)
DONE0-
ORCA
DONE
/(POP*BZ_FC)
DONE1-
DONE terminates the current operation
K1
RCLK
PERROR
ER_CLIPAR
SLINK_IN
FREAD
READ
OVFL
ER_CLIOVF
SIN
POP
POP
DAV
DAV
K1
SCLK
MQ
9
D[8:0]
D[7:0]
BUSY
NOMERGE
/BUSY*DAV
START
/BUSY*DAV*/DST1*/DST0
START_CPU
/BUSY*DAV*DST1
START_HPI
SL_READ
Pulsed when C6x reads SL register
K=K1
RS=DONE
DAV+BUSY
BUSY
BZ_CPU
START_CPU*(/D8+D7)+BZ_CPU
BZ_CPU
START_CPU*/D8+BZ_SLDAT
BZ_SLDAT
Indicates Data/Message
START_HPI*/D8+HPI_REQ
HPI_REQ
START_HPI*/D8*DST0+RQ_BLOCK
RQ_BLOCK
SL_AV
POP*BZ_CPU*BZ_SLDAT+INT_SLDAT
INT_SLDAT
INT_SLDAT
INT_SLDAT
POP*BZ_CPU*/BZ_SLDAT+INT_SLMSG
INT_SLMSG
INT_SLMSG
INT_SLMSG
POP*BZ_CPU+SL_AV
SL_AV
Controls CPU register
AUX_CE
K=K1
RS=POP
START+READ
READ
NOMERGE
READ*AUX_CE
FREAD
Count 16 words and assert BLK_END
Always assert if RQ_BLOCK=0
RQ_BLOCK
UP
CO
BLK_END
START*DSTF+BZ_FIF
BZ_FIF
FIFO busy
START*(DSTC+DSTF)+BZ_FC
BZ_FC
FIFO or Control busy
Control Register
CUD
START*D8*/D7*D6
MSG_ACK
Clear MSG-BUSY
Timeout if BUSY is asserted for 330 uS
1.28 uS clock
K128
RD
C
K1
K=K1
START*D8*/D7*D5
CLR_ERR
Clear errors in status register
BUSY
R
K
CNT_R
BLK_END
START*D8*/D7*D4
SET_A
4
Q
NOMERGE
D8*/D7
DSTC
CE
COUT
CTIME
SET_A
SET_B
HPI_DONE
CE
CI
H
K=K1
CE=SET_B
D7
CPU_RST
CPU_RST
8
CTIME
Q
D6
FIF_RST
FIF_RST
FLINKEN
D5
FLINKEN
FLINKEN
FLOAD
Dest is Control Reg.
<D[3:2]>
DST[1:0]
Data Destination
00: CPU
01: FIFO
11: HPI Write
11: HPI Read Block
HPIC[1:0]
<D[1:0]>
2
HPIC
NOMERGE
/DST1*DST0*/D8
DSTF
Dest is FIFO
K=K1
POP*BZ_FIF
FIF_WE
FIF_WE
CLINK_IN
Incoming Control Link.

Serial Section

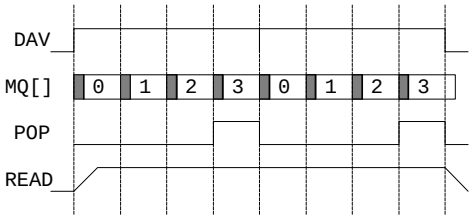
Bits come in on shift register and
get written a nibble at a time to the FIFO



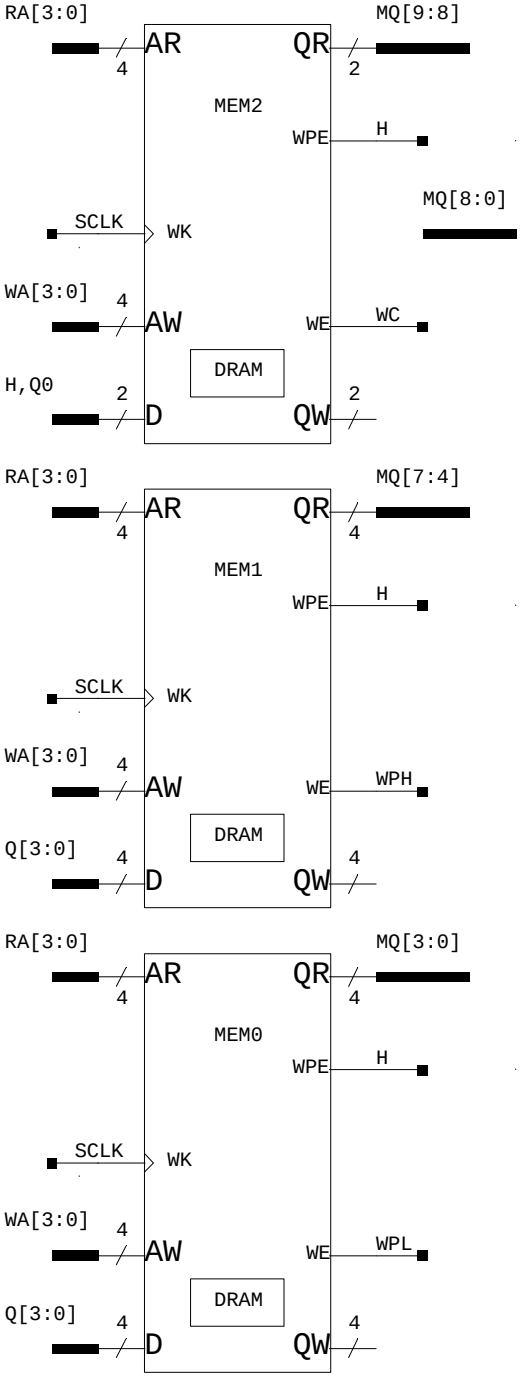


Fifo Section

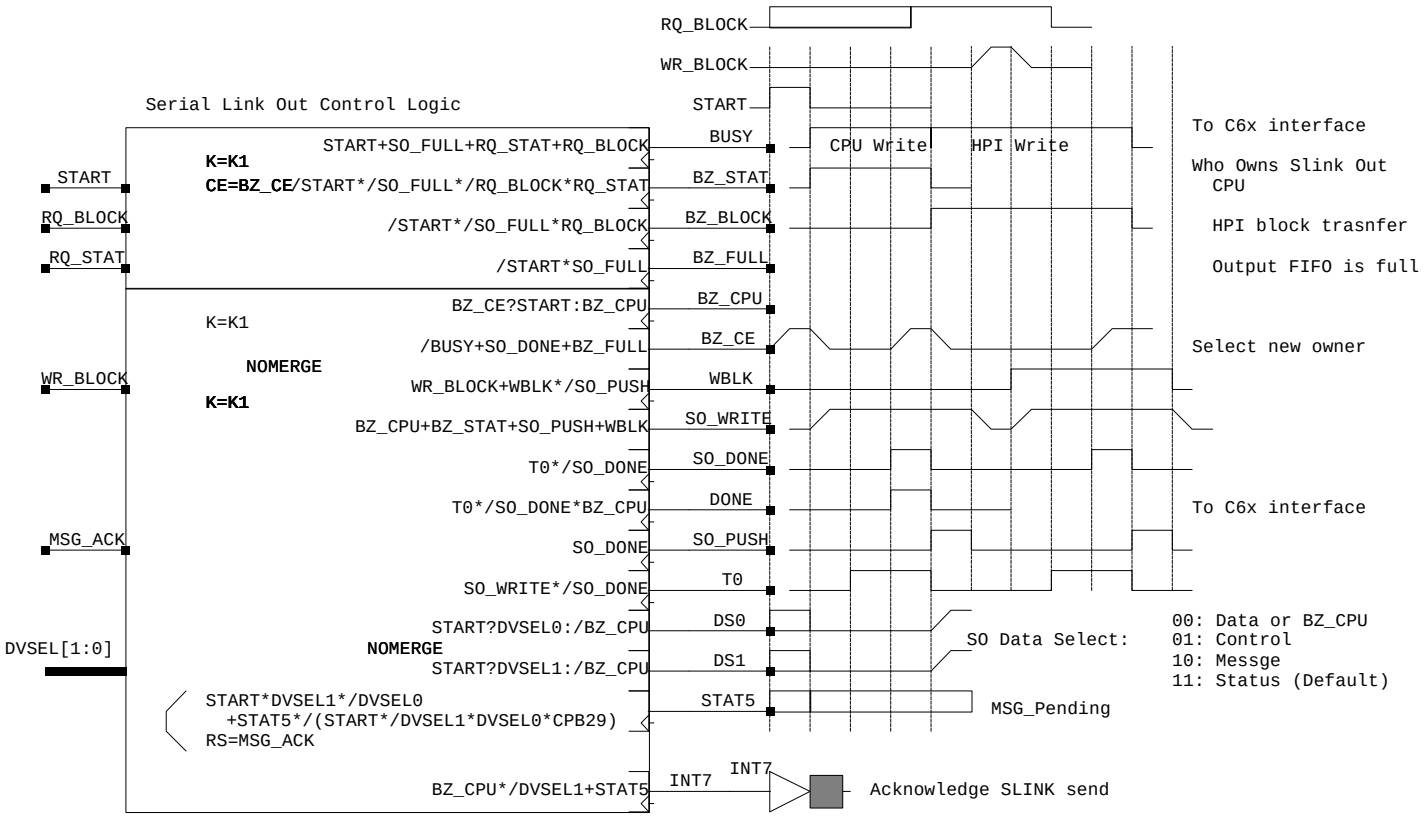
Stores four 4-byte words.
Writes (WPL) will cycle through WA[1:0]
PUSH will cycle through WA[3:2]
Reads (RE) will cycle through RA[3:0]
Upper two bits of address are gray code

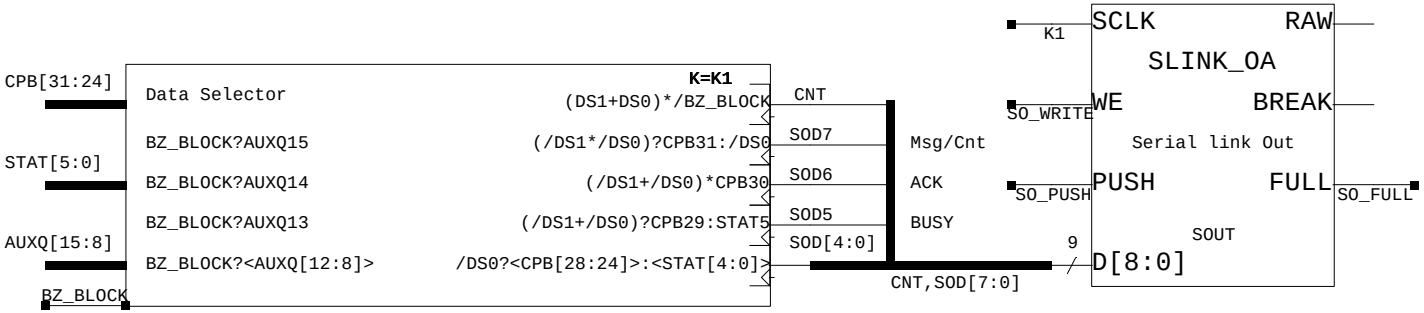


DAV indicates current data is valid
READ will cycle to next byte
POP will be asserted on the 3rd READ
Data is valid (Tco + Tpfu) after clock



```
QMAP @P/BZ_CPU=@P/BZ_CPU,@P/INT7,@P/SO_DONE,DONE2
QMAP @P/BZ_E=@P/SO_WRITE,@P/WBLK,@P/BZ_CE,@P/T0,@P/SO_PUSH
QMAP STAT5=STAT5,@P/DS0,@P/DS1
```



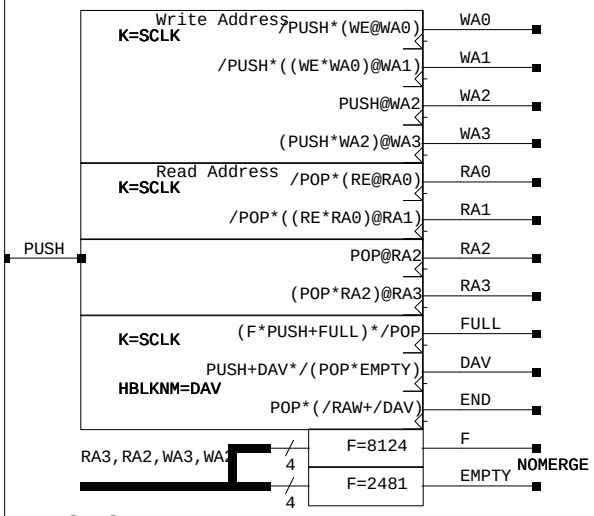


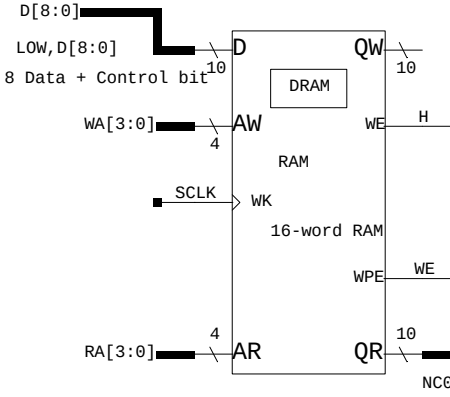
Control Link Output

Send control link words from CPU HPI, or status.

Fifo Section

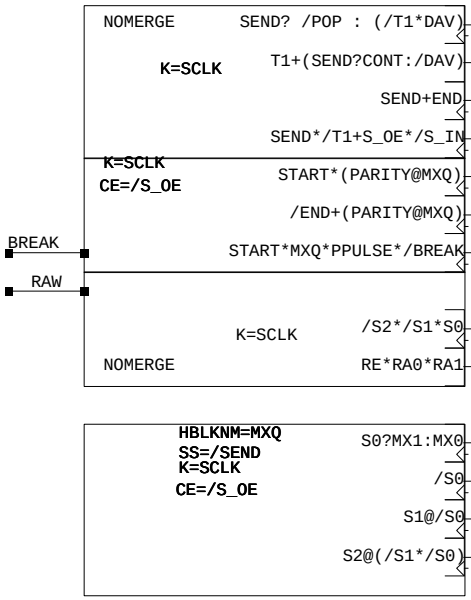
Uses 16-byte memory to store four 4-byte words.
Writes (WE) will cycle through WA[1:0]
PUSH will cycle through WA[3:2]
Reads (RE) will cycle through RA[1:0]
POP will cycle through RA[3:2]

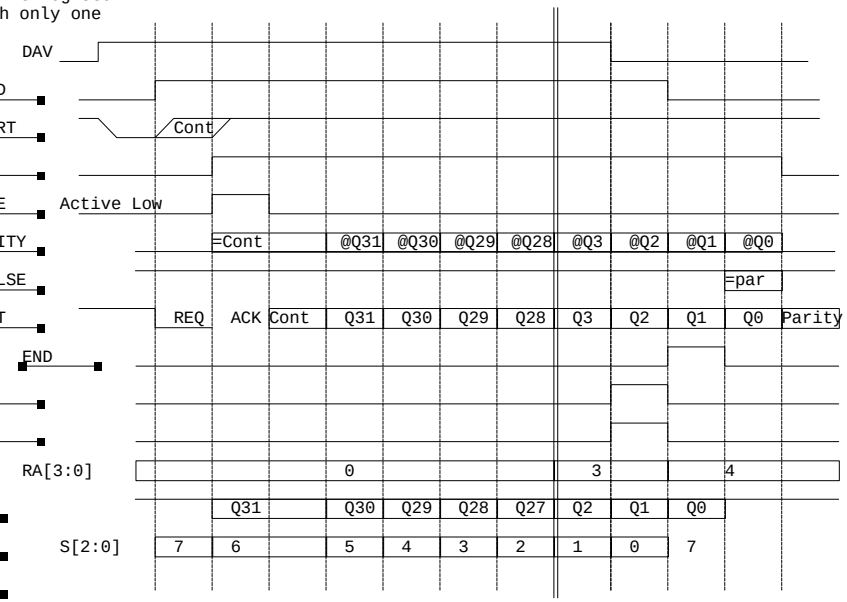




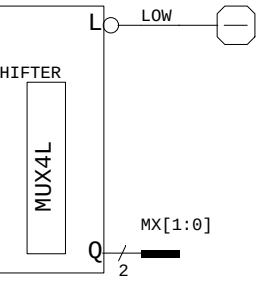
Serial Section

Sends 32-bit words with Start, Control, and Parity bits
Uses a 8-to-1 Mux instead of a shift register.
If RAW, send continuous bytes with only one
Start+Control, and no parity.

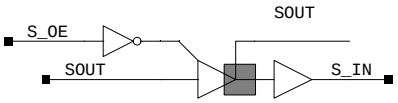




QMAP @P/START=@P/START,@P/SEND,@P/T1,@P/S_OE
QMAP @P/POP=@P/POP,@P/RA0,@P/RA1
QMAP @P/RA2=@P/RA2,@P/RA3,@P/F,@P/EMPTY

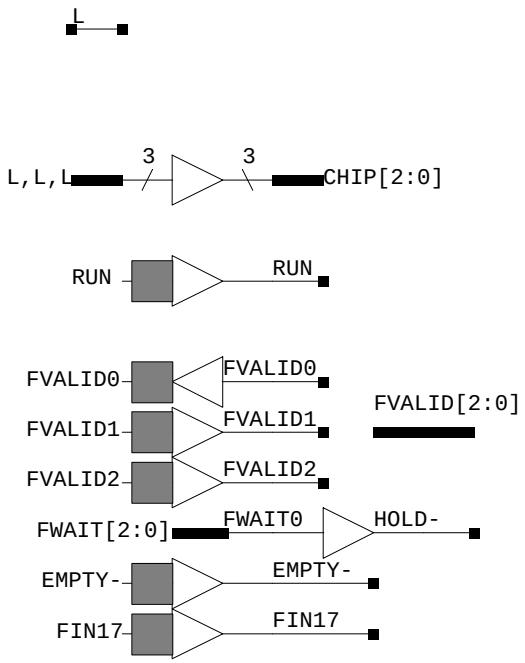


Assert a low start bit and then wait for the receiver to respond with a high ACK bit.
Incoming S_IN is synchronized in S_OE



SLINK_OA

Serial link out with FIFO and Arbiter

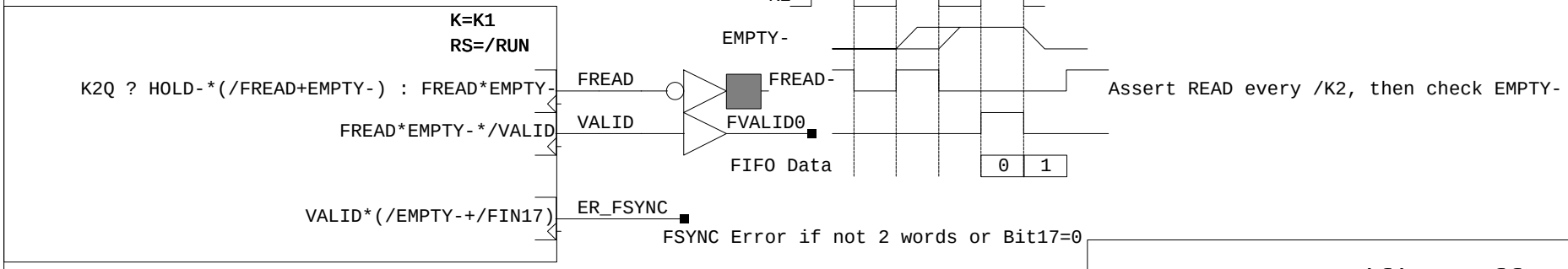


Set Chip number 0 to 7

RUN is driven by system monitor chip

CHIP 0-2: FVALID[n] is driven here
Others: FVALID[*] are received from PAD

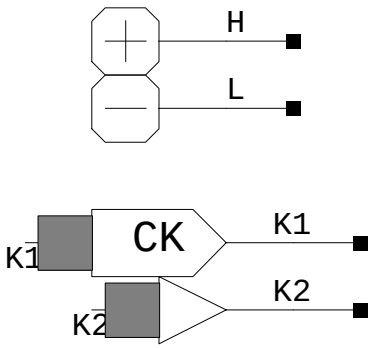
CHIP 0-2: HOLD comes from FWAIT input from INFIFO
Others: Hold is not used



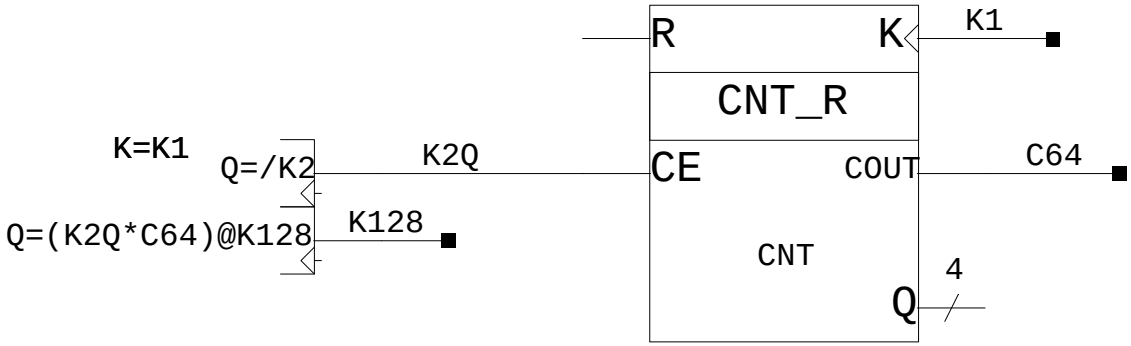
CHIP specific stuff

This contains all of the logic to make
8 different versions of this chip

Chips 0-2 control FIFOs 0-2
Chips 3-7 do not control any FIFOs

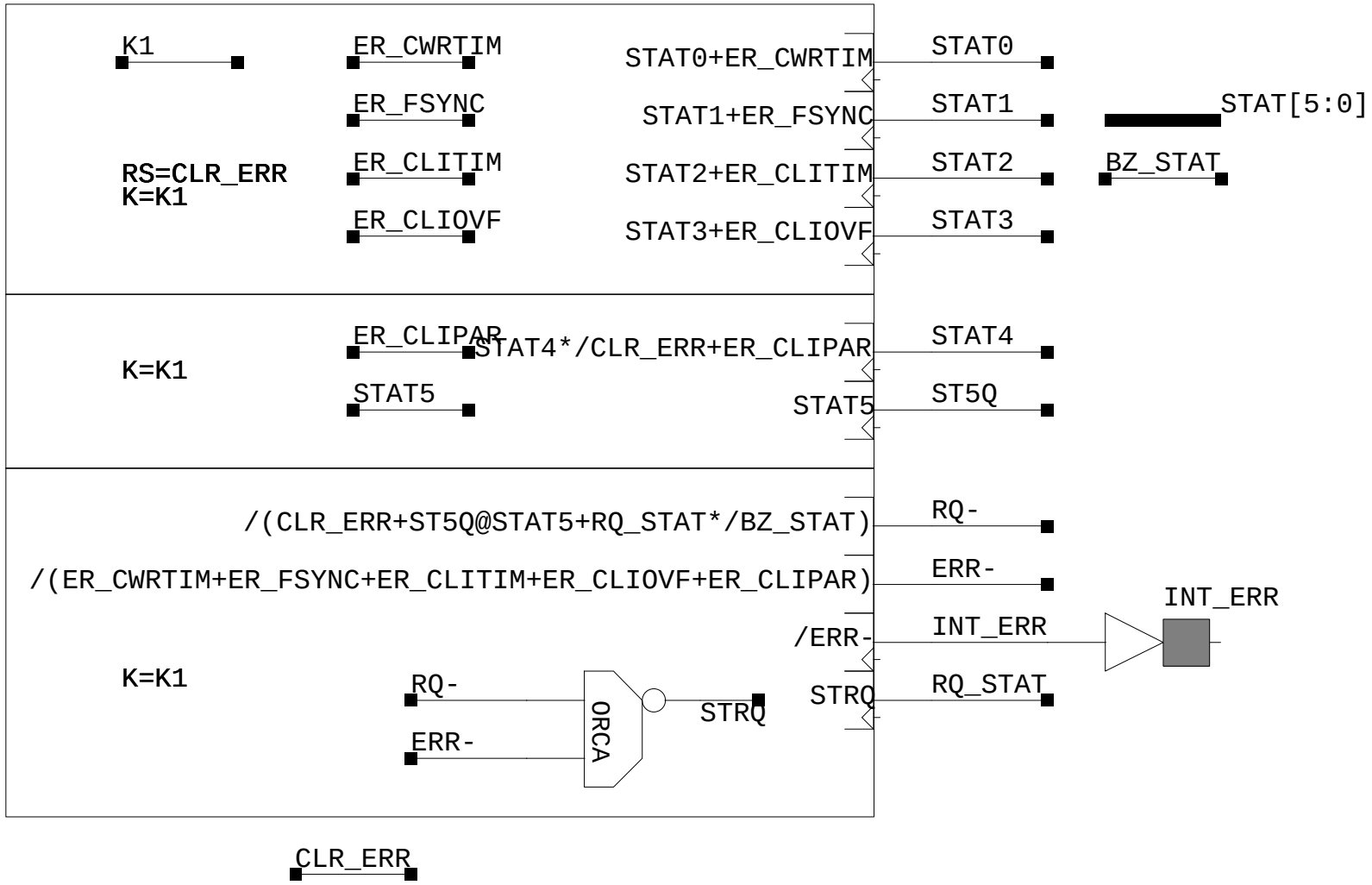


K2 is delayed to insure setup/hold



CLOCKS
Generate Global clocks

K1	20ns	From system clock
K2Q	40ns	System clock and synchronized to K1
K128	1,280ns	Used by timeout timers.


K1
ER_CWRTIM
ER_FSYNC
ER_CLITIM
ER_CLIOVF
RS=CLR_ERR
K=K1
STAT0+ER_CWRTIM
STAT1+ER_FSYNC
STAT2+ER_CLITIM
STAT3+ER_CLIOVF
STAT0
STAT1
STAT2
STAT3
STAT[5:0]
BZ_STAT
K=K1
ER_CLIPAR
STAT4*/CLR_ERR+ER_CLIPAR
STAT5
STAT5
STAT4
ST5Q
/(CLR_ERR+ST5Q@STAT5+RQ_STAT*/BZ_STAT)
/(ER_CWRTIM+ER_FSYNC+ER_CLITIM+ER_CLIOVF+ER_CLIPAR)
/ERR-
STRQ
RQ-
ERR-
INT_ERR
RQ_STAT
INT_ERR
K=K1
RQ-
ERR-
ORCA
STRQ
CLR_ERR

STATUS