ASIC-VRAM interface requirements (TPC version)

M.J. LeVine
May 8, 1997
Version 1.5
file: daq.star.bnl.gov:~levine/DAQ_IMPLEMENTATION/A-V-TPC.mif

1.0 Background

There are two versions of this document, one for the TPC receiver cards, and one for the SVT receiver cards. They differ only in the number of pads (TPC) or anodes (SVT) , in the number of timebins per pad/anode, and in the number of cluster pointer pairs to be read out for each pad/anode.

There are 6 ASICs on a mezzanine card. These ASICs are organized in 2 banks; each bank feeds 4 512KB VRAMs. This structure suggests 2 controllers which can be virtually independent of one another.

The output data port of the ASIC is 8 bits wide, with 2 control signals, V_ACK and V_REQ which form the handshake with the controller (see ASIC specification). There are 2 modes in which the ASIC generates data via its sequential output port:

1. During acquisition each ASIC outputs one 8-bit ADC value every 154 nsec. There is a 16-deep FIFO at the output to provide some elasticity. However, the VRAM controller is obliged to accept one data byte from each ASIC every 140 nsec, on average. If this is not observed, incoming data will be lost.
2. Following acquisition (signalled by END_ACQ), the controller must empty the cluster RAM on the ASIC via the same output port. The cluster RAM outputs 4K 2-byte pointers using the same port and the same handshake signals. Here, there is no fixed time budget, since the data are stored on the ASIC. However, the data must be dealt with as quickly as possible, since this is a performance issue.

1.1 Modes of operation

During acquisition of an event, ADC data flow through the ASIC to the VRAM (see **1.2** below). At the completion of acquisition, each ASIC asserts its END_ACQ signal. When all ASICS associated with a controller have asserted END_ACQ, the controller should enter cluster dump mode (see **1.3)**, during which it transfers the cluster pointers, stored internally on each ASIC, to the VRAMs. It changes the mode of operation of the ASICs by pulsing CL_DUMP to each of them.

The VRAM is large enough to accomodate several events (12). Before acquisition begins for any event, the i960 must signal the controller which event buffer is to be used for the next event. This is done via a register in the controller. This buffer number is used during both ADC acquisition and cluster dump to set the appropriate address bits.

The VRAM selected for the mezzanine board is the IBM 025161, which is a 512kB part which consists of two 256kB VRAMs with separate byte enables on the random access side, but with a *single 16-bit sequential port*.. This adds a small constraint on the sequencing of data into the VRAM: two values must be present on adjacent ASICs before the data can be clocked into the VRAM 2-byte-wide port.

The IBM025160 has two byte enables, $\overline{\mathrm{LCE}}$ and $\overline{\mathrm{UCE}}$. In this application they are used to select one-half of a VRAM's memory. Both bytes are to be wired together to the same byte lane on the i960 bus, and as a result, $\overline{\mathrm{LCE}}$ and $\overline{\mathrm{UCE}}$ must **never be asserted at the same time**!

1.2 ADC acquisition mode

The data arrive from each ASIC in a time order which corresponds to the ADC byte for every TPC pad for one time sample, followed by every TPC pad for the next time sample, etc. From the i960 point of view, this order is not acceptable. In order to make the data ordering appropriate for the code running on the i960, two steps have been taken:

1. Data from each ASIC is sequenced into the *same* VRAM, then a dummy entry is entered into this same VRAM, until 64 ADC values have been entered for all 3 ASICs (plus the dummy ASIC). Then the data stream is sequenced to the 2nd VRAM, etc. All of the ADC values corresponding to a single timebin are then found in a single byte lane of the VRAM. This then makes a sequence of 4 ADC values corresponding to adjacent time samples for a given pad appear to the i960 as part of a 4-byte object.
2. The ADC values are stored in the VRAM array in a sequence where the most rapidly varying index is ASIC number [0..3], with the next most rapidly varying index being the pad number. The i960 code would like to see packed arrays of all the data corresponding to a single ASIC. In order to accomplish this, the address lines asserted by the i960 are scrambled before they are presented to the VRAM.

The sequence required is given below. In the following tables, the notation **ASIC A | ASIC B** means that ASIC A provides the lower 8 bits and ASIC B provides the upper 8 bits of the 16-bit SAM input. The notation **dummy** means that the contents of the upper 8 bits is meaningless.

TABLE 1. ADC data sample sequence

pad number	timebin	sequence	source	destination
0	0	0	ASIC A | ASIC B	VRAM 1
		1	ASIC C | dummy	
1		2	ASIC A | ASIC B	VRAM 1
		3	ASIC C | dummy	
. . .				
63		126	ASIC A | ASIC B	VRAM 1
		127	ASIC C | dummy	
0	1	128	ASIC A | ASIC B	VRAM 2
		129	ASIC C | dummy	
1		130	ASIC A | ASIC B	VRAM 2
		131	ASIC C | dummy	
. . .				
63		254	ASIC A | ASIC B	VRAM 2
		255	ASIC C | dummy	
0	2	256	ASIC A | ASIC B	VRAM 3
		257	ASIC C | dummy	
1		258	ASIC A | ASIC B	VRAM 3
		259	ASIC C | dummy	
. . .				
63		382	ASIC A | ASIC B	VRAM 3
		383	ASIC C | dummy	

1.3 CPP data

The cluster pointers emerge from the ASIC’s exit FIFO as 2 pairs of bytes, corresponding to the beginning and end pointers, each of which emerges as the low byte followed by the high byte. These, too, must be assembled into the VRAM array so that a longword on the i960 bus appears as the sequence {begin low, begin high, end low, end high.} In addition, the memory lines need to be scrambled as for the ADC data so that all of the pointers for a given ASIC appear to be a compact array to the i960. Note that the scrambling of i960

TABLE 2. CPP data sample sequence

sequence	pad number	cluster number	data source	destination
1	0	0	ASIC A begin LOW|ASIC B begin LOW	VRAM 1
2	0	0	ASIC A begin HIGH | ASIC B begin HIGH	VRAM 2
		0	ASIC C begin LOW | dummy	VRAM 1

TABLE 2. CPP data sample sequence

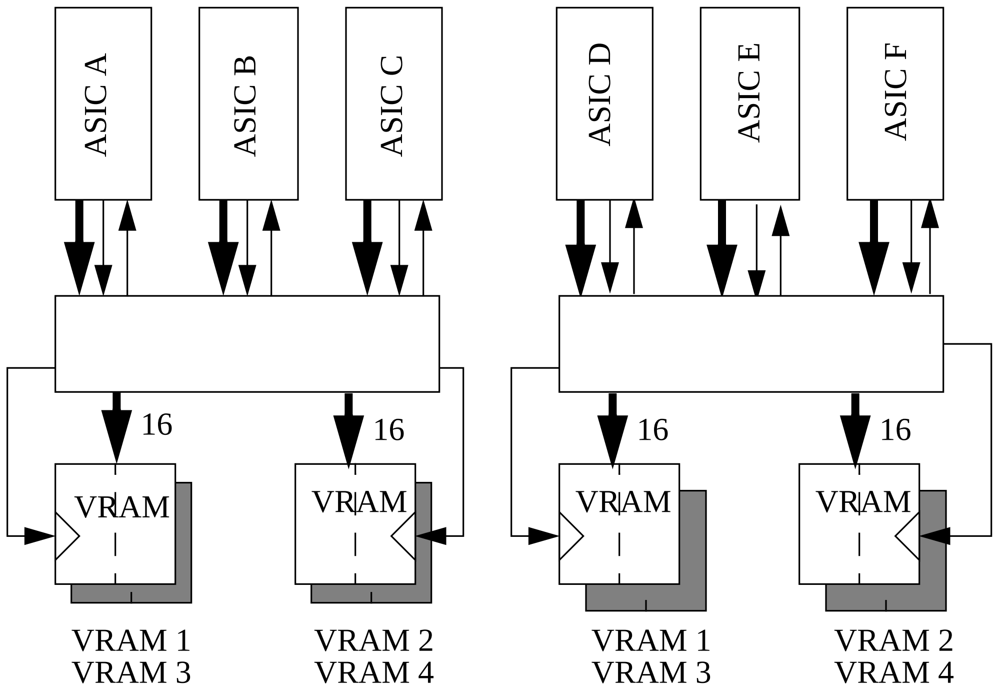
sequence	pad number	cluster number	data source	destination
3	0	0	ASIC A end LOW | ASIC B end LOW	VRAM 3
		0	ASIC C begin HIGH | dummy	VRAM 2
4	0	0	ASIC A end HIGH | ASIC B end HIGH	VRAM 4
		0	ASIC C end LOW | dummy	VRAM 3
5	0	1	ASIC A begin LOW | ASIC B begin LOW	VRAM 1
		0	ASIC C end HIGH | dummy	VRAM 4
6	0	1	ASIC A begin HIGH | ASIC B begin HIGH	VRAM 2
		1	ASIC C begin LOW | dummy	VRAM 1
7	0	1	ASIC A end LOW | ASIC B end LOW	VRAM 3
		1	ASIC C begin HIGH | dummy	VRAM 2
8	0	1	ASIC A end HIGH | ASIC B end HIGH	VRAM 4
		1	ASIC C end LOW | dummy	VRAM 3
9	0	2	ASIC A begin LOW | ASIC B begin LOW	VRAM 1
		1	ASIC C end HIGH | dummy	VRAM 4
10	0	2	ASIC A begin HIGH | ASIC B begin HIGH	VRAM 2
		1	ASIC C begin LOW | dummy	VRAM 1
11	0	2	ASIC A end LOW | ASIC B end LOW	VRAM 3
		1	ASIC C begin HIGH | dummy	VRAM 2
12	0	2	ASIC A end HIGH | ASIC B end HIGH	VRAM 4
		1	ASIC C end LOW | dummy	VRAM 3
	. . .	. . .		
125	0	31	ASIC A begin LOW | ASIC B begin LOW	VRAM 1
		30	ASIC C end HIGH | dummy	VRAM 4
126	0	31	ASIC A begin HIGH | ASIC B begin HIGH	VRAM 2
		30	ASIC C begin LOW | dummy	VRAM 1
127	0	31	ASIC A end LOW | ASIC B end LOW	VRAM 3
		30	ASIC C begin HIGH | dummy	VRAM 2
128	0	31	ASIC A end HIGH | ASIC B end HIGH	VRAM 4
		30	ASIC C end LOW | dummy	VRAM 3
	. . .	. . .		
8065	63	0	ASIC A begin LOW | ASIC B begin LOW	VRAM 1
	62	31	ASIC C end HIGH | dummy	VRAM 4
	. . .	. . .		
8192	63	31	ASIC A end HIGH | ASIC B end HIGH	VRAM 4
		31	ASIC C end LOW | dummy	VRAM 3
8193	63	31	ASIC C end HIGH | dummy	VRAM 4

address lines is not identical for ADC and CPP data; address lines A[21..20] are used to distinguish between the ADC and CPP accesses (see below).

Looking at Table 2, sequence number 1 and 8193 are unlike the remainder of the steps; they represent startup and shutdown stages of the process. The remainder of the table is just a repetition of steps 9-12 (shaded in the table).

Performance is an issue for the CPP transfer. But each of the objects shown in Table 2 which form a 16-bit quantity is available simultaneously. So each sequence in the table can in principle be performed in the 25 nsec required to access the next item in the ASIC's CPP stream. If necessary, pipelining of this sequence should be utilized to achieve this speed.

The ASIC design guarantees that, following the first CPP byte, all successive CPP data will be available at intervals of 22 nsec. If these data are required at intervals longer than 22 nsec/byte, it is not necessary to wait on the V_REQ signal.



For the ADC data (A[21..20] < 3) the i960 address lines have the following meaning:

• buffer number (0 -11):	A[21..18]
• ASIC (0-5):	A[17..15]
• pad number (0-63):	A[14..9]
• timebin (0-511):	A[8..2], BE[3..0]

At the VRAM, these functions have been mapped differently to the VRAM byte enables and address bits:

• buffer number (0-11):	VA[17..14]
• ASIC (0-5):	VA[0], $\overline{\text{LCE}}$, $\overline{\text{UCE}}$
• pad number (0-63):	VA[6..1]
• timebin (0-511):	VA[13..7], $\overline{\text{LCE}}$, $\overline{\text{UCE}}$

Specifically, the VRAM byte enables **for the ADC data** are given by:

```
LCE 1 = !(BE0 &  !A17 & !A15) left controller
UCE 1 = !(BE0 &  !A17 &  A15)
LCE 2 = !(BE1 &  !A17 & !A15)
UCE 2 = !(BE1 &  !A17 &  A15)
LCE 3 = !(BE2 &  !A17 & !A15)
UCE 3 = !(BE2 &  !A17 &  A15)
LCE 4 = !(BE3 &  !A17 & !A15)
UCE 4 = !(BE3 &  !A17 &  A15)

LCE 1 = !(BE0 &  A17 & !A15)  right controller
UCE 1 = !(BE0 &  A17 &  A15)
LCE 2 = !(BE1 &  A17 & !A15)
UCE 2 = !(BE1 &  A17 &  A15)
LCE 3 = !(BE2 &  A17 & !A15)
UCE 3 = !(BE2 &  A17 &  A15)
LCE 4 = !(BE3 &  A17 & !A15)
UCE 4 = !(BE3 &  A17 &  A15)
```

The VRAM address lines **for the ADC data** are related to the i960 address lines as follows:

```
VA[0]      = A[16]
VA[6..1]   = A[14..9]
VA[13..7]  = A[8..2]
VA[17..14] = A[21..18]
```

For the CPP data (A[21..20] = 3), the i960 address lines have the following meaning:

• buffer number (0-11):	A[19..16]
• ASIC number (0-5):	A[15..13]
• pad number (0-63):	A[12..7]
• cluster number (0-31):	A[6..2]

The VRAM byte enables **for the CPP data** are given by:

```
LCE 1 = !(BE0 & !A15 & !A13)  left controller
UCE 1 = !(BE0 & !A15 &  A13)
LCE 2 = !(BE1 & !A15 & !A13)
```

```
UCE 2 = !(BE1 & !A15 &  A13)
LCE 3 = !(BE2 & !A15 & !A13)
UCE 3 = !(BE2 & !A15 &  A13)
LCE 4 = !(BE3 & !A15 & !A13)
UCE 4 = !(BE3 & !A15 &  A13)

LCE 1 = !(BE0 & A15 & !A13)   right controller
UCE 1 = !(BE0 & A15 &  A13)
LCE 2 = !(BE1 & A15 & !A13)
UCE 2 = !(BE1 & A15 &  A13)
LCE 3 = !(BE2 & A15 & !A13)
UCE 3 = !(BE2 & A15 &  A13)
LCE 4 = !(BE3 & A15 & !A13)
UCE 4 = !(BE3 & A15 &  A13)
```

The VRAM address lines **for the CPP data** are related to the i960 address lines as follows:

```
VA[0]      = A[14]
VA[5..1]   = A[6..2]
VA[11..6]  = A[12..7]
VA[15..12] = A[19..16]
```

2.0 Functional requirements

The following paragraphs roughly describe the areas which have to be covered by this design.

2.1 Initialization

The VRAM contains a mask (WPBM) which is used to enable the SAM data on a per-bit basis as it is transferred from the SAM to the DRAM array. This mask register powers up as all ‘0’ s, effectively disabling any writes from the SAM to the DRAM. The register must be initialized to all ‘1’ s using a dedicated sequencing of the chip’s control lines.

This initialization is the shared responsibility of the software and the controller hardware. The software is obliged to write to a special address with 32 bits of ‘1’s on the data bus. A write to this address will cause the controller to sequence the VRAM’s control lines to write the data present on the data lines into the chip’s WPB mask. [Note that this may be easier to implement as a pair of addresses corresponding to the 2 sets of chip enables on the IBM chips.]

2.2 Refresh

Refresh for the VRAM must be provided.

2.3 Arbitration

There are three requesters for the VRAM random-access cycle:

- transfer from the SAM to the VRAM memory
- refresh cycles
- i960 bus accesses

Arbitration must be provided with priority in the order listed.

2.4 Filling from ASIC

Data must be assembled from the various ASIC sequential ports to the SAM port of the VRAMs. When a SAM is full, its contents must be transferred to the appropriate row of the VRAM before the next ASIC data are serviced.

Data must be entered in the sequence shown in Tables 1 and 2.

2.5 i960 bus access

Addresses presented by the i960 bus master (either i960 or PLX) need to be scrambled in one of two ways as described above.

For performance reasons, burst mode must be accomodated on the i960 bus for processor accesses. Burst mode does not have to be provided for the PLX bridge. Thus a maximum of 4 longword accesses must be accomodated.

This is complicated by the address scrambling necessary for the ADC and CPP data. The scrambling results in 4 successive words in i960 address space not being contiguous in the VRAM.

Taking the case of the ADC data, 4 consecutive longwords on the i960 bus have A[3..2]=00,01,10,11. But these addesses result in VA[8..7]=00,01,... Note that these accesses still fall within a row of the same byte of the same VRAM chip, so that burst access can be accomodated with reasonable efficiency.

For the case of the CPP data, successive cluster words as seen from the i960 bus correspond to VRAM address bits VA[2..1]. Again, these fall within a row, and can be accessed without extra delay.

2.6 ADC/CPP transition

The VRAM controller treats incoming data as ADC data until it receives an END_ACQ from each of its constituent ASICs. It then transitions to CPP mode. When it has completed emptying the ASICs of CPP data, which it must determine based on its internal counters, it must signal the CPU that an event is ready. It does this by asserting an interrupt. This interrupt is to remain asserted (level) until it is reset by the CPU.

The CPU resets the asserted interrupt by writing to a register interface built into the controller, which has the following effects:

1) The controller deasserts the interrupt signal.

2) The controller resets any internal logic (e.g., counters) to make it ready for a new event.

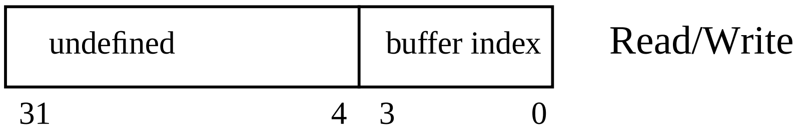
3) The controller distributes a CLEAR signal to each of its constituent ASICs.

The controller must also have a CLEAR input which is generated at the fiber optic interface when an ‘abort event’ token is received. Upon receipt of this CLEAR, the controller must execute numbers 2) and 3) in the above paragraph.

2.7 Register interface

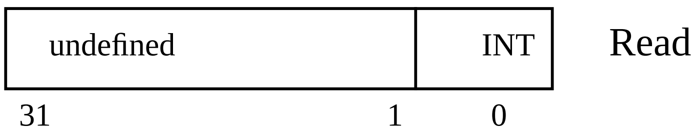
The following registers must be implemented in the controller, accessible to the i960 processor bus. In all cases the registers must appear as 32-bit objects, even if the contents are byte-wide or even less. This means that the data should be in the least significant byte lane, and the registers should fall on 4-byte addresses.

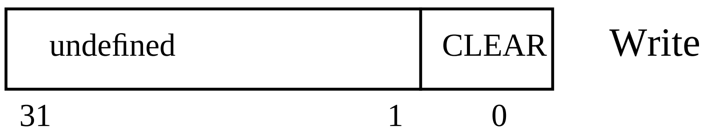
2.7.1 Buffer producer index (BPI) $XXXXXX00



Bits BPI [3:0] are used by the controller to determine the number of the buffer into which the next event should be stored. The software must guarantee that the proper buffer number is written into the controller before the event starts, and that the buffer register will not be modified while an event is in progress.

2.7.2 Control Status register (CSR) $XXXXXX04

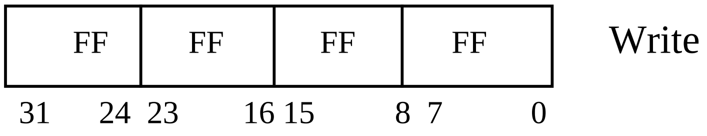




The INT bit, when =1, signifies that the controller is presently asserting an interrupt.

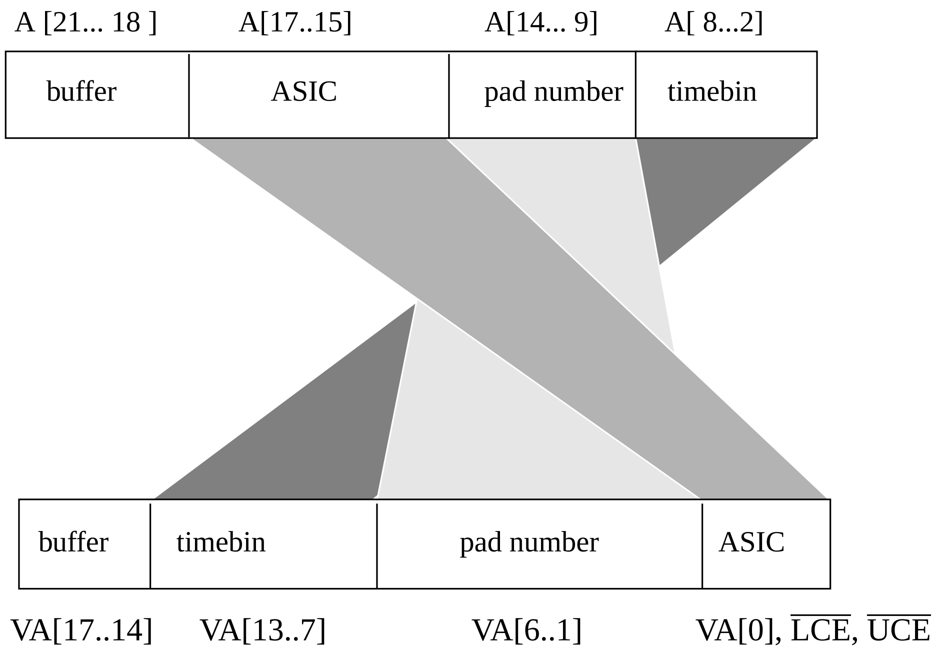
Writing a ‘1’ to the CLEAR bit causes the controller to deassert its interrupt, together with the side effects listed in an earlier paragraph. Writing a ‘0’ to this bit has no effect.

2.7.3 WPBM register $XXXXXX08



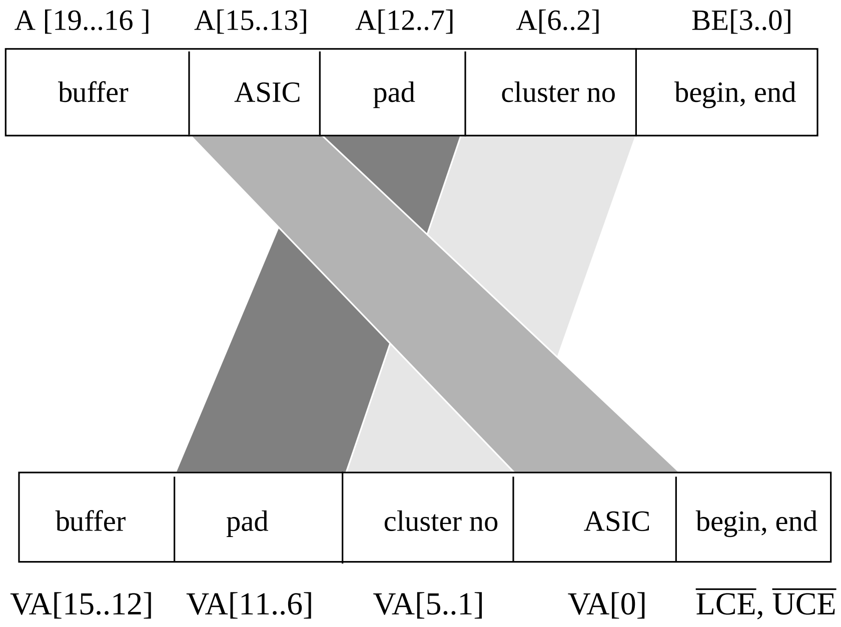
A write to this register causes the controller to sequence the VRAM control lines to write the data present on the data lines into the VRAM’s WPB mask. Note that the data have no effect on the controller; the data enter directly into the data ports of the affected VRAM chips without passing through the controller.

i960 view of the ADC data

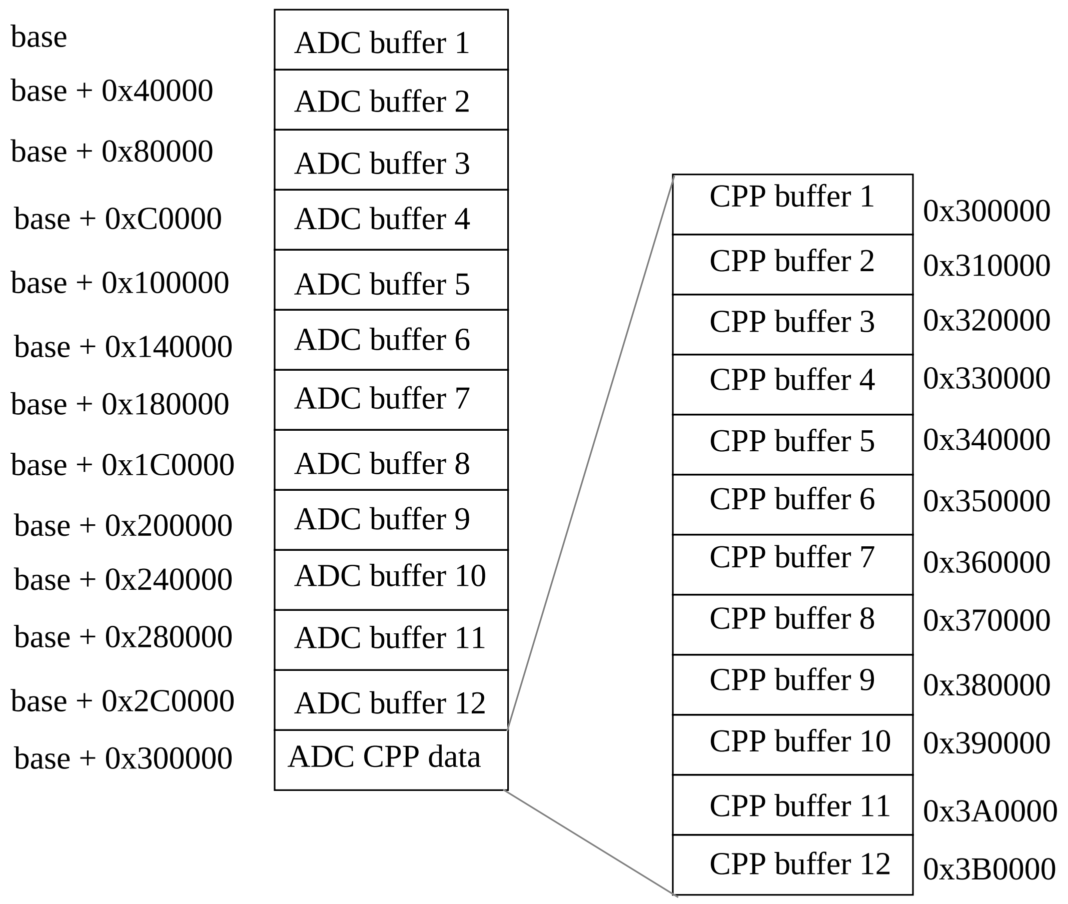


ADC data seen from the VRAMs

i960 view of the CPP data



CPP data seen from the VRAMs



layout of buffers in i960 address space