Proposed Datapath for the Approximated Affine Motion Estimation architecture

In this report a first, basic version of the Block diagram for the Approximated Affine Motion Estimation (AME) Datapath is presented. Notice that no optimizations and no pipeline stages are present: the scheme is drawn assuming **infinite resources** and **zero time delay**.

Sommario

0. High Level Block Diagram

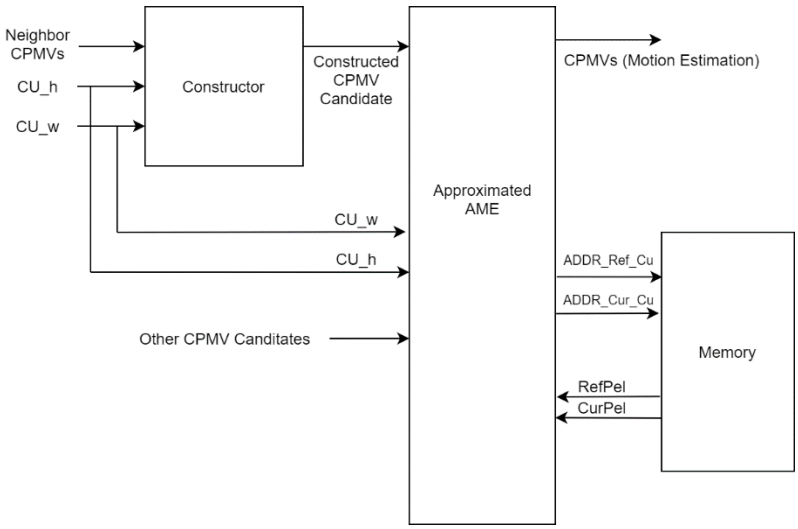

Figure 1. High Level Block Diagram

The structure is made by a "Constructor", which computes a triplet of Control Point Motion Vectors (CPMVs) candidates and the "Approximated Affine Motion Estimation (AME)" block, which computes the CPMVs that work as Motion estimation for the Coding Unit (CU) to be encoded. This system retrieves the pixels of the Current CU and the Reference Frame simply from a Memory. Notice that

- **Cu_h** and **CU_w** are width and height of the Coding Unit to be encoded. They can assume just three values due to the VVC standard constraints and design choices: (16,32,64).
- **Refpel** and **Curpel** are the pixel luminance values from the Reference Frame and Current CU that must be compared to perform the motion estimation. They can range in [0,255].
- **Other CPMV Candidates** are CPMVs that are not built by the constructor but are collected in other simple ways by the VTM (VVC Test Model reference software) as explained in [1].

In the following sections each block is shown and explained in detail.

1. Constructor

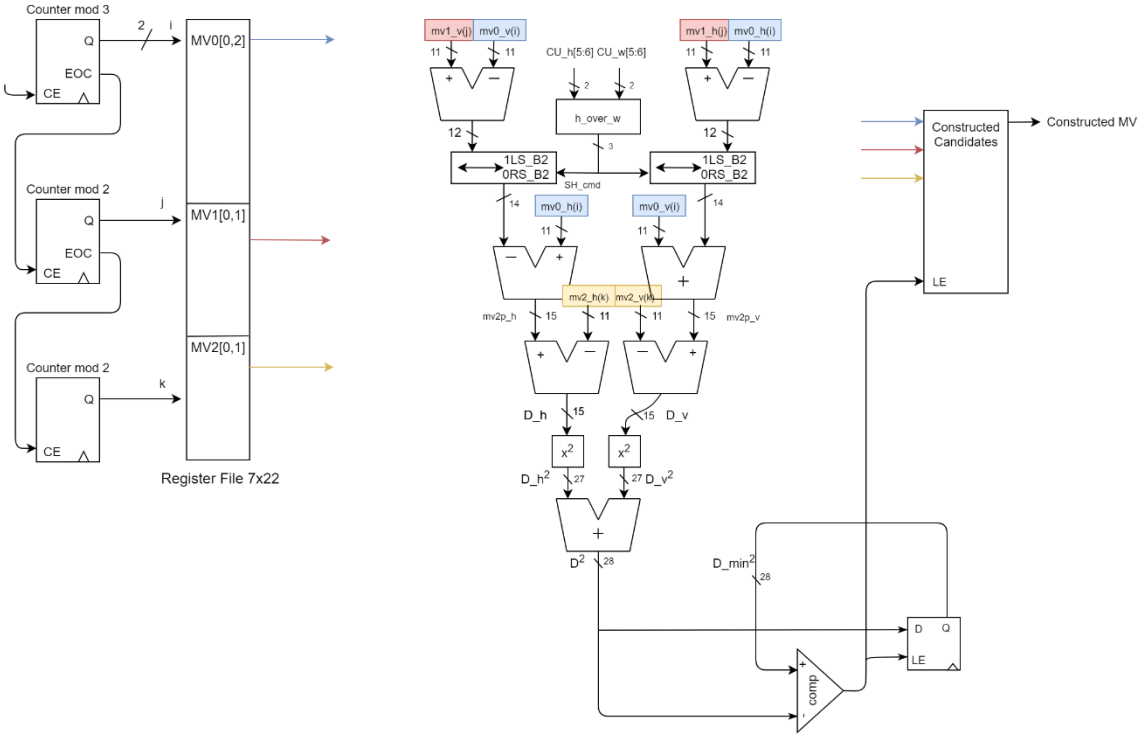

Figure 2. Constructor Datapath

(Notice that the connections between the Register File and the processing elements are not explicitly drawn, they are implicitly shown by the colours.)

This architecture is responsible of computing the Constructed Candidates following the algorithm described in section II-B of (2), except for the refinement process that here is absent. The parallelism is calculated according to the range of values that each variable can assume.

There are two operators which are important to be described in detail.

1.1. The *h_over_w* block

The output of the first subtractor, $(mv_{1,(v,h)} - mv_{0,(v,h)})$ must be multiplied by $\frac{CU\_h}{CU\_w}$ but, instead of using a divider to preform this latter division, we exploit the fact that the only values that these two variables can assume are (16,32,64). With this in mind, I propose the following scheme

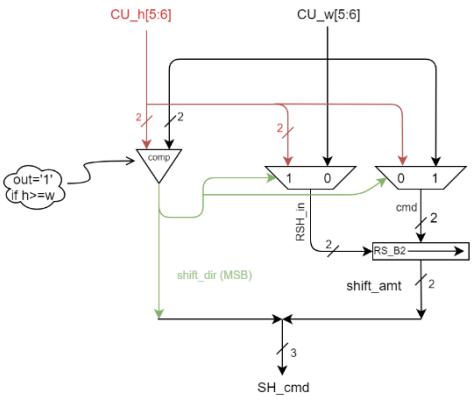

Figure 3. Simplified divider proposed Datapath (h_over_w)

Since the possible values for the operands are only three, just the two MSBs are sufficient to compute the result, which is expressed in a form useful for the following block (*1LS_B2,0RS_B2*), which computes the product between $(mv_{1,(v,h)} - mv_{0,(v,h)})$ and $\frac{CU\_h}{CU\_w}$.

In fact, the operation $\frac{CU\_h}{CU\_w}$ has just six possible results, which are $2^{\pm i}$ with $(i = 0,1,2)$. This means that the cited multiplication consists of just a shift towards the left or the right dependently on the fact that $CU\_h > CU\_w$ or vice versa, respectively. That is why the result is expressed by one bit for the direction of the shift (1: Left, 0: Right) and other two bits (*shift_amt*) for the amount of the shift operations to be done. The following table shows the truth table for this operator.

CU_h	CU_w	CU_h[4:6]	Cu_w[4:6]	shift_dir (1LS;0RS)	shift_amt
16	16	001	001	1	00
16	32	001	010	0	01
16	64	001	100	0	10
32	16	010	001	1	01
32	32	010	010	1	00
32	64	010	100	0	01
64	16	100	001	1	10
64	32	100	010	1	01
64	64	100	100	1	00

Truth table for the simplified divider. The result is expressed in a form that is convenient for the following block to compute its result

Brief description of an internal block:

- **RS_B2:** Is a barrel shifter able to perform a Right shift of 0,1,2 positions on the input data *RSH_in* depending on the command *cmd*.

1.2. 1LS_B2,0RS_B2

This block is a barrel shifter capable of performing a Left or Right Shift of 0,1, or 2 positions depending on the command *SH_cmd* provided by the *h_over_w* block. The reason behind the need of this block has been explained in the previous section.

Notice that, when performing a Right Shift, the fractional part of the result is discarded. This corresponds to perform a truncation on the number. This choice is done in order to reduce the parallelism without penalties on the constructed result (the results are the same with respect to the software model).

2. Approximate AME (hardware accelerator)

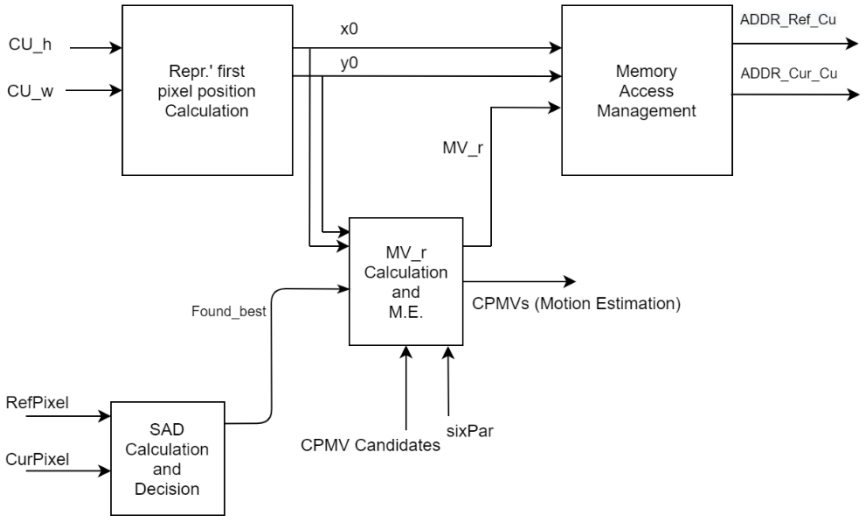

Figure 4. Approximated AME High level bock diagram

The block responsible for the Motion Estimation is partitioned in four sub-blocks (each one with a specific task) which are is shown in the following sections.

2.1. Representatives' first pixel position calculator

The Motion Estimation is done by taking the candidates CPMVs which produce the minimum Sum of Absolute Differences (SAD) between the Reference frame and the motion compensated CU. Actually, the motion compensation is not applied to the whole CU, but only to some 4x4 pixels block, called **representative blocks**. These blocks are located in strategical positions of the CU, to compute their position this procedure is followed (with reference to figure 5)

1. Divide the CU in 16x16 blocks (highlighted **in blue** in the figure)
2. Take the 4x4 blocks (highlighted **in red**) on the corner of each 16x16 block
3. These 4x4 blocks are the representative blocks

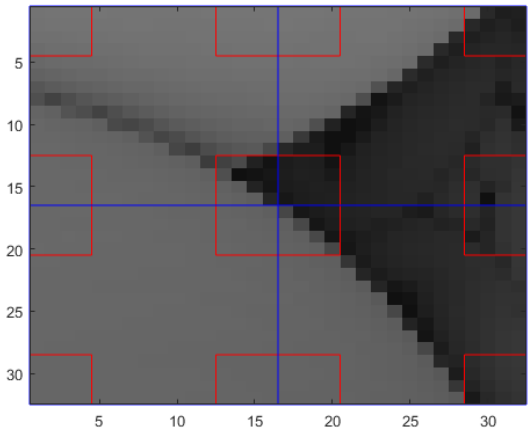

Figure 5. An example of 32x32 CU with the reference blocks highlighted in red

The block shown in this section calculates the position of the first pixel in each representative block (indicated as $(x0, y0)$), which is essential to:

- Compute the Motion Vector relative to that block
- Compute the Memory Address to read the luminance values from memory, corresponding to both the CU and the Reference frame.

In the following figure the datapath of this block is shown

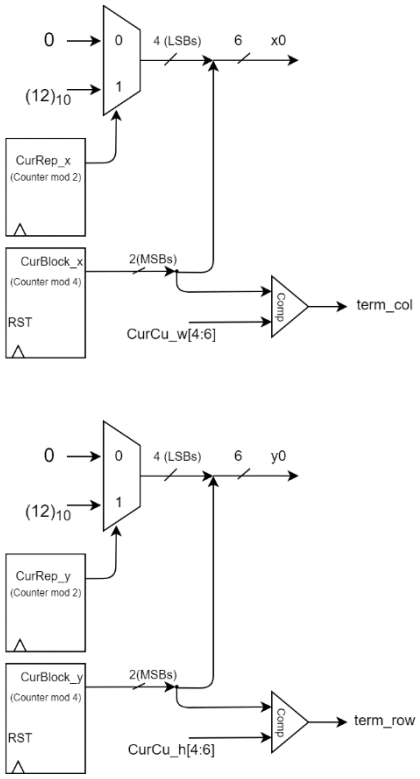

Figure 6. First pixel position calculator

2.2. MVr Calculation and Motion Estimation

Here the candidates CPMV are stored in a Register File and used to compute the motion vector relative to each 4x4 block (MVr) following the model described in sections II-A and III-A of (2). This is done in order to perform the motion compensation and compute the SAD.

This block is also responsible of carrying out the result of the estimation. A flag coming from the SAD calculator tells if the current candidate is the one associated to the minimum SAD value, in the case this is true the current candidate is copied from the input register file into the output register, drawn in the lower part of the diagram.

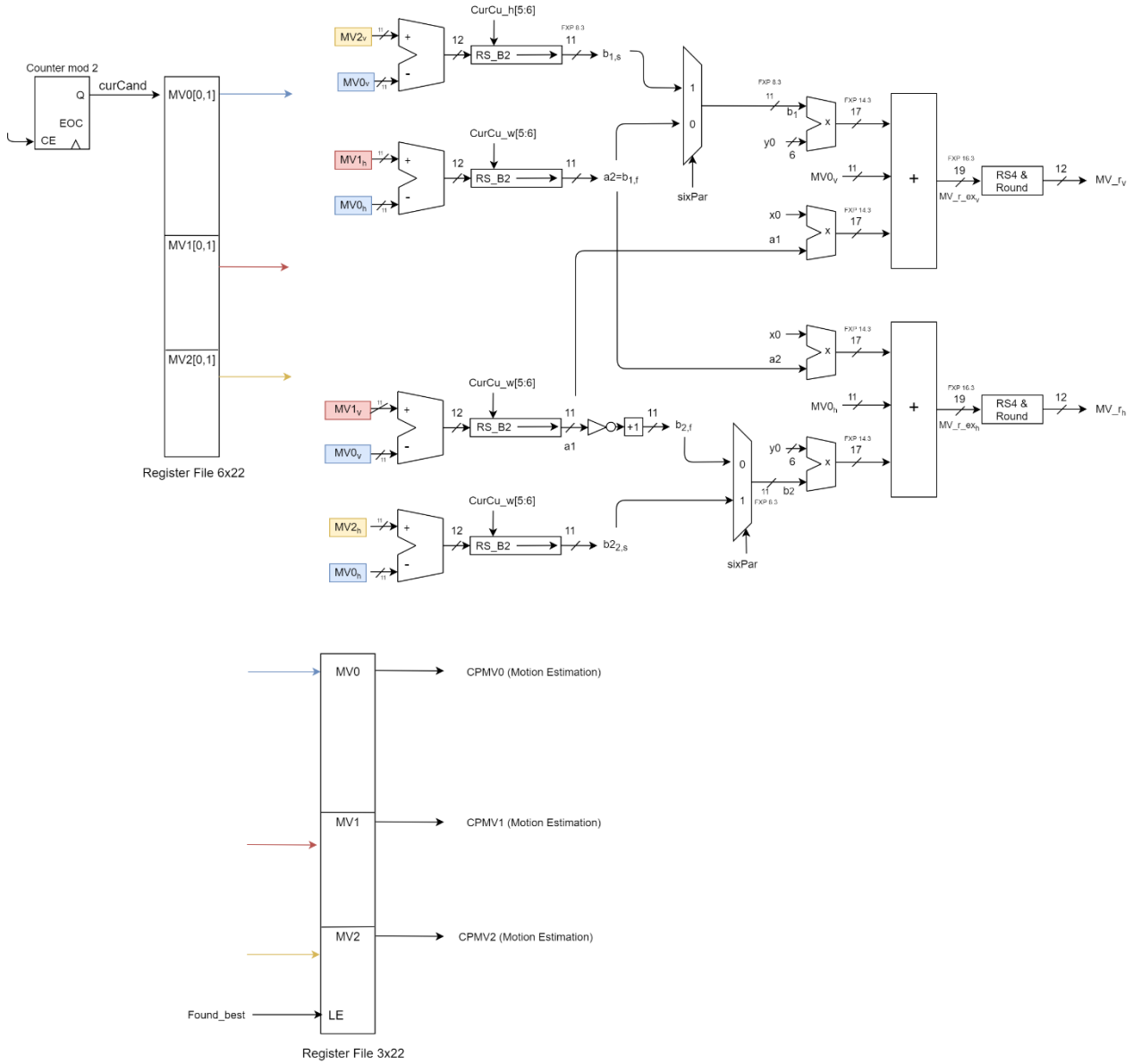

Figure 7. MVr Calculation and Motion Estimation

A brief description of some blocks and signals:

- **RS_B2** is a barrel shifter capable of performing a right shift of 0,1, or 2 positions depending on the command input. This allows us to perform the division by *h* or *w* without using a divider. Notice that the result is represented on Fixed Point with 3 bits of fractional part. This choice is done to keep the same results with the software model without increasing too much the internal parallelism.
- **RS4&Round** is a block which performs the *Round to nearest integer* approximation and a Right shift of 4 bits. This is done because the MVs in the VVC are expressed with a 1/16-pixel accuracy.

- **sixPar** is a flag informing the accelerator if the 6-parameter model is currently in use. This allows to adapt the calculation of the motion vector accordingly.

  Notice that the connections between the Register File and the processing elements are not explicitly drawn, they are implicitly shown by the colours.

2.3. Memory Access Manager

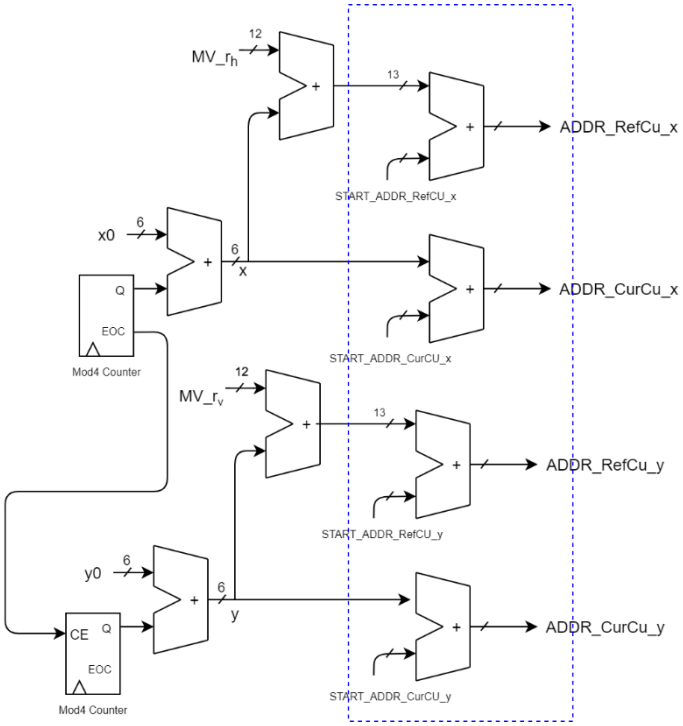

Figure 8. Memory Address Manager - Datapath

The block in section 2.1 computes only the position of the first pixel in a 4x4 block, but dependently on the way the memory is organized, it may be required to compute the position of all the pixels in order to read all the luminance values. This block simply addresses all the pixels in a 4x4 block, adding the motion vector to the position of the pixels in the reference frame to perform the motion compensation.

The last column of adders highlighted in the blue dotted line is optional (depending on the way the memory is organized), it adds the starting address of each data structure to the relative address computed by the previous elements.

2.4. SAD calculation and Decision

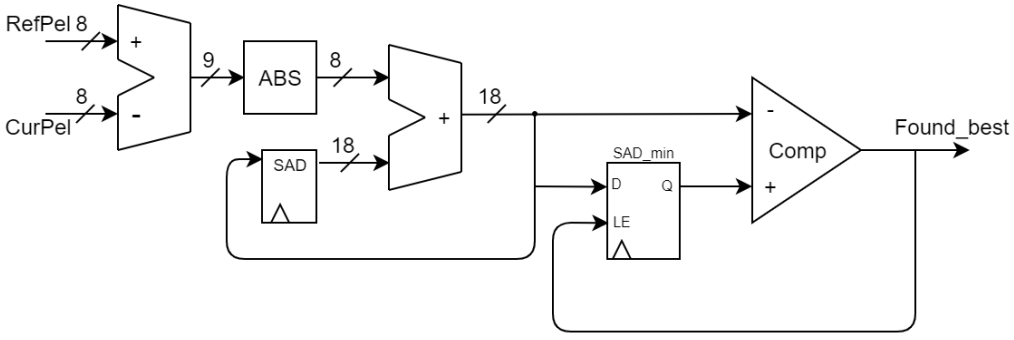

Figure 9. SAD calculation and Decision - Datapath

This block computes the SAD for each of the two candidates and asserts the flag *Found_best* when the SAD for the current candidate is the minimum one. This flag is fed into the block shown in section 2.2.

2.5. Global View

To have a better idea of how the blocks are connected, in the following figure a detailed global view of the Approximated AME datapath is shown.

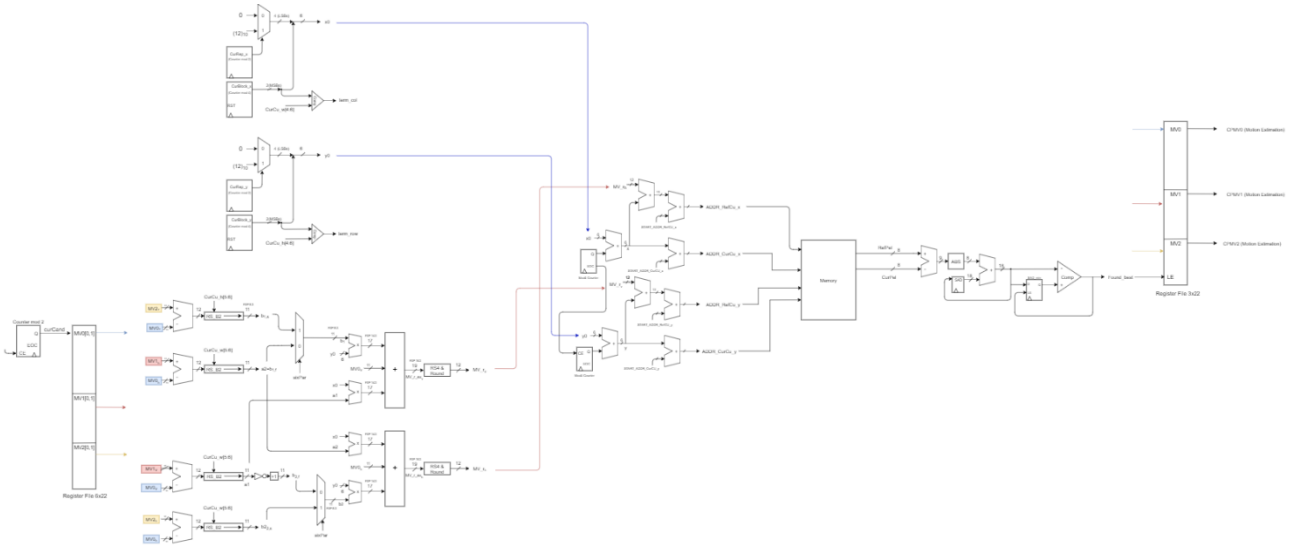

Figure 10. Detailed Global View of the AME Datapath

References

[1] ISO/IEC JTC 1/SC 29/WG 11, "Test Model 8 for Versatile Video Coding (VTM 8)", 2020-01-17

[2] K. Zhang *et al.*, "An Improved Framework of Affine Motion Compensation in Video Coding," in IEEE Transactions on Image Processing, vol. 28, no. 3, pp. 1456-1469, March 2019