# The Multimedia Video Processor (MVP)


## Graham Short, Texas Instruments


## 1. History

In 1987 engineers from Texas Instruments started work on the specification of a device which would become the next generation in its TMS320 Digital Signal Processor and TMS340 Graphics System Processor families, building on the successes and lessons learnt from each. It was to be targeted at a number of graphics applications, an example of which are:

- X-terminals and PC/workstations where the TMS340 had made considerable inroads, but now 3D graphics put increased demand on the hardware.
- Video Conferencing where real-time video and audio compression and decompression for the then draft standard of H.320.
- Document Image Processing for *digital copiers* which scan, enhance, compress and transmit over modem or network.
- Multimedia applications, such as JPEG and MPEG compression and decompression where some of the TMS320 DSP family had been used.

The analysis of the processing requirements for these applications can be daunting. Table 1 details one such application, that of the Px64 standard, which is the video portion of the H.320 Video Conferencing standard. This is the requirement to compress and decompress an image at 30 frames-per-second.

The table shows that roughly 1.2 billion operations per second are required to perform this application. The total does not include pre- or post-processing, audio compression and decompression, any formatting required for data transmission or other system functions. Note that several of the functions in the table require 100 million operations per second which is comparable to the speed of today's RISC microprocessors. It became clear that, in order to develop a single device which would provide this order of processing capability, multiple processors would be needed on the chip. Clock speed alone could not provide this level of processing.

| **Function** | **MOPS (% of total)** |
|---|---|
| Motion estimation | 608 (51.0%) |
| Code mode decisions | 40 (3.4%) |
| Loop filtering (encode and decode) | 110 (9.2%) |
| Pixel difference | 18 (1.5%) |
| DCT (encode) | 74 (6.2%) |
| Inverse DCT (encode and decode) | 192 (16.1%) |
| Threshold/quantization/zig-zag scan | 50 (4.2%) |
| Bit stream encode | 17 (1.4%) |
| Reconstruction (encode and decode) | 62 (16.1%) |
| Bit stream decode and inverse quantization | 22 (1.8%) |
| **Total** | **1.193 MOPS** |

**Table 1 Typical processing requirements for CCITT H.320 (Px64) video compression and decompression**

In our experience in developing the TMS340 GSP family we knew that graphics processors require special hardware, such as barrel shifters, colour expanders and splittable arithmetic logic units. In the TMS340 however, instructions to draw lines and perform BitBlts were executed in microcode. This meant that the software programmers could only interface to the hardware in the manner allowed by the microcode. Generally this meant that graphics algorithms took more cycles in their inner loop than they might otherwise have had, if the programmer could access the hardware directly. Opening up the hardware, making it accessible to the programmer, was seen as a key requirement for the new design.

One aspect of the TMS340 that continued into the new design was that of programmability. Many fixed function ICs were being designed which contained for example, dedicated Discrete Cosine Transformation (DCT) hardware, geared towards the JPEG and MPEG standards. This approach was rejected in favour of providing hardware which could perform DCT and then be reused for a variety of other algorithms. Table 1 shows that a DCT is only 20% of the task of Px64 standard. Silicon area is too

precious to switch off part of it when you are in the other 80% of the algorithm, particularly when you have the sort of processing requirements placed upon you as shown in the table.

## 2. Architecture

Figure 1 shows a block diagram of the TMS320C80 or MVP which began sampling at the end of 1993. It integrates five powerful fully-programmable processors, a sophisticated DMA controller, 50K bytes of SRAM and a video timing controller. Four of the five processors are identical advanced digital signal processors (ADSPs) that have special-purpose hardware for graphics processing algorithms. Each ADSP is capable of performing many RISC-equivalent operations in a cycle, as we shall see. The fifth processor is a 32-bit RISC CPU called the Master Processor (MP). It includes an IEEE 754-compatible floating point unit (FPU). All five processors can be programmed in C or assembly language.

In addition to the processors, the MVP has a Transfer Controller (TC) which is an intelligent DMA controller. One of the first concerns in defining the architecture was how to keep the processors from having to wait for data. The solution was to have some on-chip memory dedicated to instruction and data, serviced for the processors by the TC. It receives requests from the processors to obtain *packets* of data from external memory for processing. Many imaging and graphics algorithms such as convolution, DCT, and FFT require multiple accesses within a group of pixels. Pulling these pixels on chip once and keeping any intermediate results on-chip, greatly reduces the number of accesses to external memory, which would easily become a bottleneck. After processing, data is moved by the TC back off-chip under the direction of the processors.

Rather than dedicate specific memory to specific processors, the MVP contains a sophisticated and, in silicon area terms, *large* crossbar switching network. This allows access by every processor to every data RAM (the processors do not normally read and write to the RAMs used as caches). This unlimited access to on-chip data greatly improves efficiency as there is no need for the processors to send large quantities of data to one-another. If its on-chip, its available to all processors.

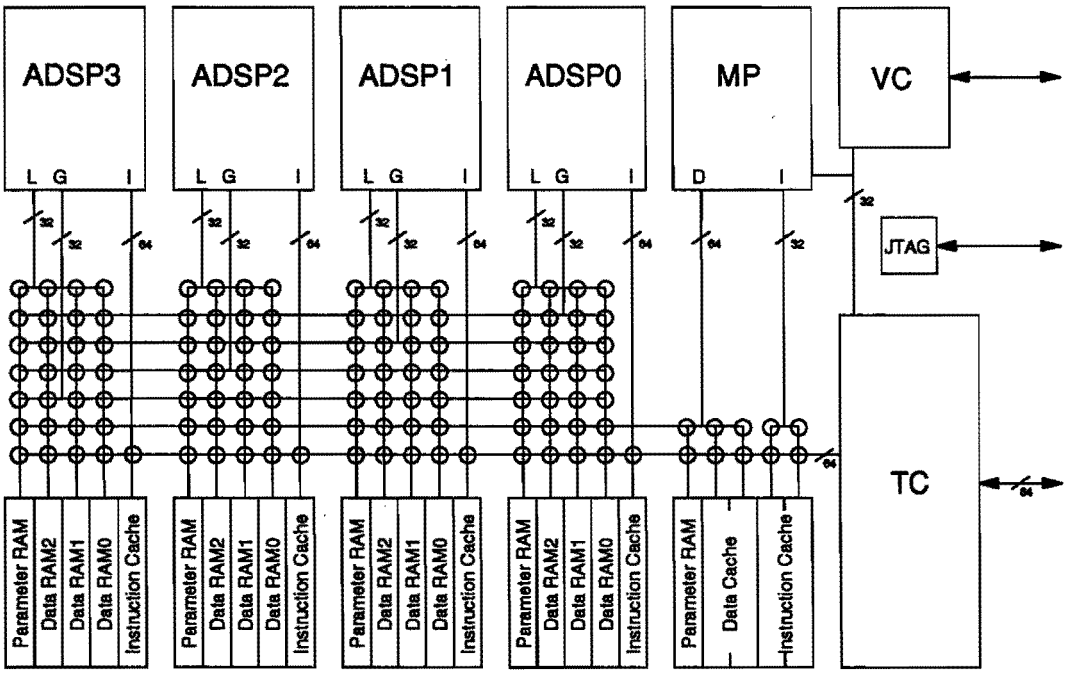


**Figure 1. Block Diagram of the TMS320C80 Multimedia Video Processor (MVP)**

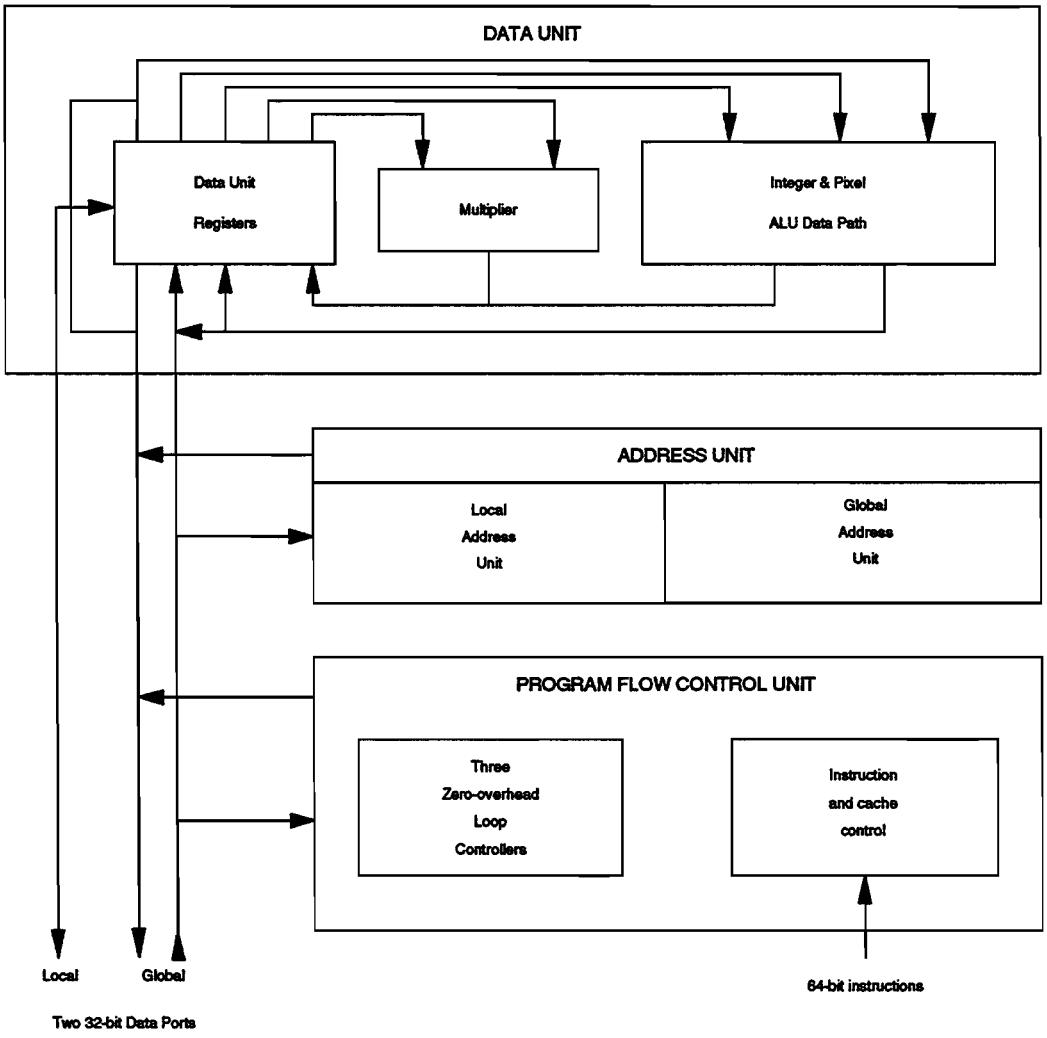


**Figure 2. Block Diagram of the ADSP**

## 3. Advanced Digital Signal Processor

The Advanced Digital Signal Processor (ADSP) is the key component to the MVP architecture's ability to provide the necessary processing power of its target applications. The justification for the *advanced* in its name is that whereas traditional DSPs performed well at the multiply-intensive signal processing, they did not do well at the bit-field intensive requirements necessary in pixel manipulations and entropy encoding and decoding. A high level block diagram of the ADSP is shown in Figure 2.

The ADSPs can each perform in excess of 10 RISC-like operations in each cycle. In order to specify the multiple parallel operations, a wide instruction word of 64-bits is used. The instruction has fields that independently control the multiplier, ALU, and the two Address Units. All instructions nominally execute in a single cycle. The instructions are pipelined in a 3 cycle pipe. A new instruction can be started every cycle unless there is pipe stall condition caused by either a cache miss or contention in accessing the on-chip RAM. As can be seen in the figure, the ADSP consists of three main operating units: the Data Unit, the Address Unit and the Program Flow Control Unit, which will be explained in turn.

Each ADSP has 44 user registers which can be the source or destination of ALU or memory operations. Although most registers can be used for general purpose manipulations, some also have specific roles. The Address Units have address and index registers used to access on-chip RAM. The Program Flow Control Units have loop control registers used for zero-overhead branches.

The Data Unit shown in Figure 2 is the main processing engine for the ADSP. It contains a multiplier and a 32-bit ALU, both of which are *splittable*. In graphics operations it is common to want to manipulate *pixels* rather than *words*. Thus to be able to split the 32-bit ALU so that it can perform 4 8-bit operations in parallel, can provide an immediate 4x performance improvement.

Likewise the multiplier, which can perform one 16 x 16 multiply in its standard configuration, can be split to perform two 8 x 8 bit multiplies in parallel (both multiplies occur in a single cycle). The multiplier has a rounding option which was incorporated to maintain the specified accuracy for the video compression standards.

Figure 3 shows some more detail of the Data Unit and the special hardware that was included to provide the bit-field manipulation that was mentioned earlier. This includes the following:-

- A barrel rotator that is capable of rotating the bits in a 32-bit number from 0 (no rotation) to 31 bits.
- A mask generator which takes an unsigned 5-bit input (n) and produces a corresponding 32-bit output ($2^n$ - 1).
- An expander that replicates from 1, 2 or 4 bits, 32, 16 or 8 times. This is useful amongst other things for monochrome to colour pixel expansion.
- A bit detector which determines the bit position of the leftmost one, rightmost one, leftmost bit change or rightmost bit change within a 32-bit word.
- A 3-input ALU that performs Boolean (bit-wise) and arithmetic combinations of the three inputs. In addition to performing strictly Boolean or arithmetic functions, a combination mode is also available, that enables the third operand to mask one or both sources of an arithmetic function in one pass through the ALU (i.e. one cycle).

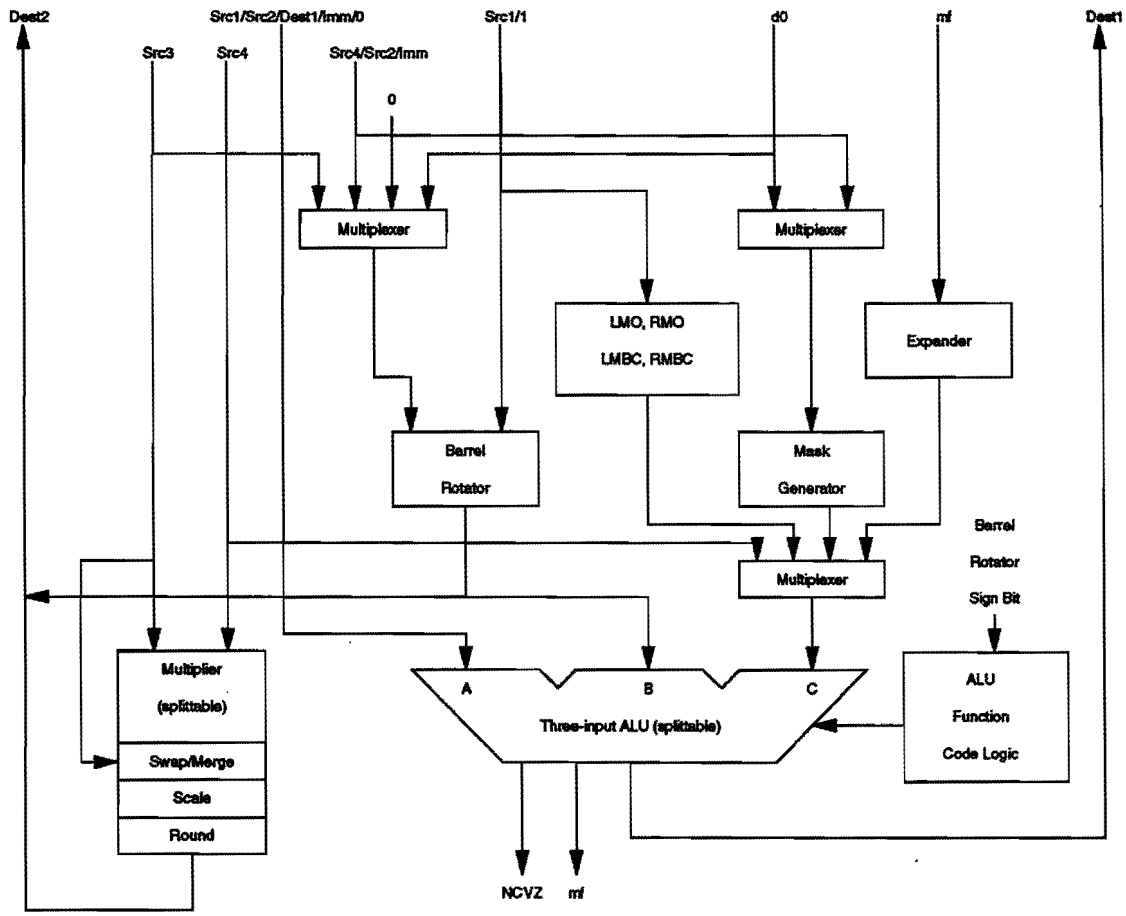


**Figure 3. Diagram of the ADSP Data Unit**

A single three-input ALU function can perform what could take multiple passes through a two-input ALU, which is why it is not a simple matter to quantify how many RISC-like operations the ADSP can perform in a single cycle. The instruction format allows the programmer direct control on the flow through the ALU datapath and which of the special hardware (if any) is used as source to the ALU.

The second important block in the ADSP shown in Figure 2 is the Address Unit. There are two (nearly) identical Address Units in each ADSP. One is called the Global Unit, the other the Local Unit. The only difference between them is that the Local Unit can only access memory in the RAMs immediately below the ADSP (called the local RAM) when two parallel memory accesses are performed. Together the two units can perform 2 memory operations in each cycle. Each memory operation is a load or a store that can be specified totally independent of the Data Unit operation. Each Address Unit has 6 address registers for specifying the address for the memory operation. An immediate value or one of 3 index registers can be added/subtracted to the address value and the result of the address computation can optionally modify the original address register value, to facilitate stepping through a memory array.

The source of a store or the destination of a load can be any of the 44 ADSP registers. Either or both Address Units can perform a data operation in place of a memory transfer. In these cases the result of the address calculation is written to the destination register in place of fetching data from memory. This again is a powerful aid in speeding up algorithms, since when the Address Units are not needed to perform memory operations, the programmer can use these units to perform data operations in parallel with those going on in the Data Unit.

```
diff                .set   d7          ; Contains 4 8-bit Pixel Differences
CurrPixel           .set   d6          ; Contains 4 8-bit Current Pixels
PrevPixel           .set   d5          ; Contains 4 8-bit Previous Pixels
SumABS              .set   d4          ; 4 running byte sums of absolute differences
OnesCountIndex      .set   x8          ; Contains carries from pixel differences
NumCount            .set   x9          ; Number of carries, got from table look-up
CurrentPixels       .set   a0          ; Pointer to current pixels
PreviousPixels      .set   a1          ; Pointer to previous pixels
CountTable          .set   a8          ; Pointer to 1s count table
CarryCount          .set   a9          ; Running sum of 1s count

ls0 = start_loop                                    ; initialize loop start address
le0 = end_loop                                      ; initialize loop end address
lc0 = <NUMBER OF PIXELS TO PROCESS>                 ; initialize loop count
 <OTHER SET-UP CODE>
start_loop:
 diff =mzc CurrPixel - PrevPixel                    ; split ALU, 4 subtracts
  || OnesCountIndex = mf                            ; move from prior calc...
                                                    ; ... in one Address Unit
  || CurrPixel = *CurrentPixels++                   ; load next set of 4 pixels...
                                                    ; ...in other Address Unit
 SumABS =mc (SumABS+diff) & mf | (SumABS-diff) & ~mf; split ALU, 4 absolute adds
  || a15 = *(CarryCount += NumCount)                ; Modify CarryCount from prev...
                                                    ; ...NumCount in one Unit
  || PrevPixel = *PreviousPixels++                  ; load next set of 4 pixels...
                                                    ; ... in one Address Unit
 diff =mzc CurrPixel - PrevPixel                    ; software pipeline
  || NumCount =b *(CountTable + OnesCountIndex)     ; Determine NumCount using...
                                                    ; ...LUT in one Address Unit
  || CurrPixel = *CurrentPixels++                   ; software pipeline
end_loop:
 SumABS =mc (SumABS+diff) & mf | (SumABS-diff) & ~mf; software pipeline
  || PrevPixel = *PreviousPixels++                  ; software pipeline
```

**Figure 4. Example ADSP inner loop assembly code**

Memory operations in the Address Unit, and ALU and multiplier operations in the Data Unit can all be specified as conditional, so that the write to the destination register is not performed unless the condition (which can be any one of 16 combinations of the 4 condition codes NCZV) is true. A mode is also available which allows the specification of a conditional source where the N condition code can select between two registers for a read operation.

The third main unit in the ADSP is the Program Flow Control (PFC) Unit. It controls the fetching, decoding of instructions and the instruction pipeline. Because each ADSP instruction can do many operations in parallel, key inner loops often require very few instructions. Thus the PFC contains three zero-overhead loop controllers. Each controller has a set of registers that specifies the starting and ending addresses and the current and initial loop counts. Once these registers have been initialized, branches controlled by them have zero overhead.

An example of ADSP code is shown in Figure 4. This code forms the inner loop for motion estimation algorithms in standards such as Px64. Part of the code looks as though it is duplicated, but it is in fact software pipelining, where one part of the algorithm is occurring at the same time as the previous iteration for another part of the algorithm. The four instructions in this inner loop produce a running count and absolute sum of differences from 8 8-bit pixels. Although there isn't space here to go into the details, hopefully the comments in the code should give a feel for what is going on. The intent is to show that in one instruction, a complicated 3-input ALU operation can occur on 4 independent pixels in the Data Unit, while another two memory or data operations are occurring in the Address Unit and loop and instruction control is occurring in the PFC Unit.

# 4. Master Processor

The RISC Master Processor (MP) is shown in Figure 5. The MP is aimed at general purpose high-level language programming. Additionally due to its integrated floating point unit, the MP will perform operations requiring higher precision and floating point values.

The MP Integer Unit has a 32-bit instruction word that performs integer register to register or load/store instructions, nominally in one cycle. Like the ADSP, it has a 3 cycle pipeline, allowing a new instruction to be loaded every cycle, providing a stall condition has not occurred. The 32-bit integer ALU operates on two sources and one destination. One source operand may be an immediate value. The MP ALU, like the ADSP, has a barrel shifter, mask generator and left/right-most one detects to improve its bit-field manipulation capabilities.

The MP Floating Point Unit is IEEE-754 compatible. Hardware support for the floating point unit consists of a full double-precision floating point ALU and a 32-bit single precision floating point multiply unit. The floating point hardware is pipelined and the floating point multiply unit is supported with microcode to provide single and double-precision operations.

Either a double-precision floating point ALU or a single-precision floating point multiply can be started in each cycle. A special set of *vector* instructions are also available, aimed at matrix style operations. These instructions initiate a multiply, add/subtract and a 32- or 64-bit load or store with auto-increment addressing every cycle.

The MP has 31 32-bit registers that are common to both the integer and floating point units. The registers are scoreboarded for floating point results and memory load operations. The scoreboard allows the MP to continue execution and only stall its pipe if an instruction tries to use a register prior to being available. Common with other RISC architectures, there is an additional 32-bit R0 dummy register that always reads as 0 and discards writes.

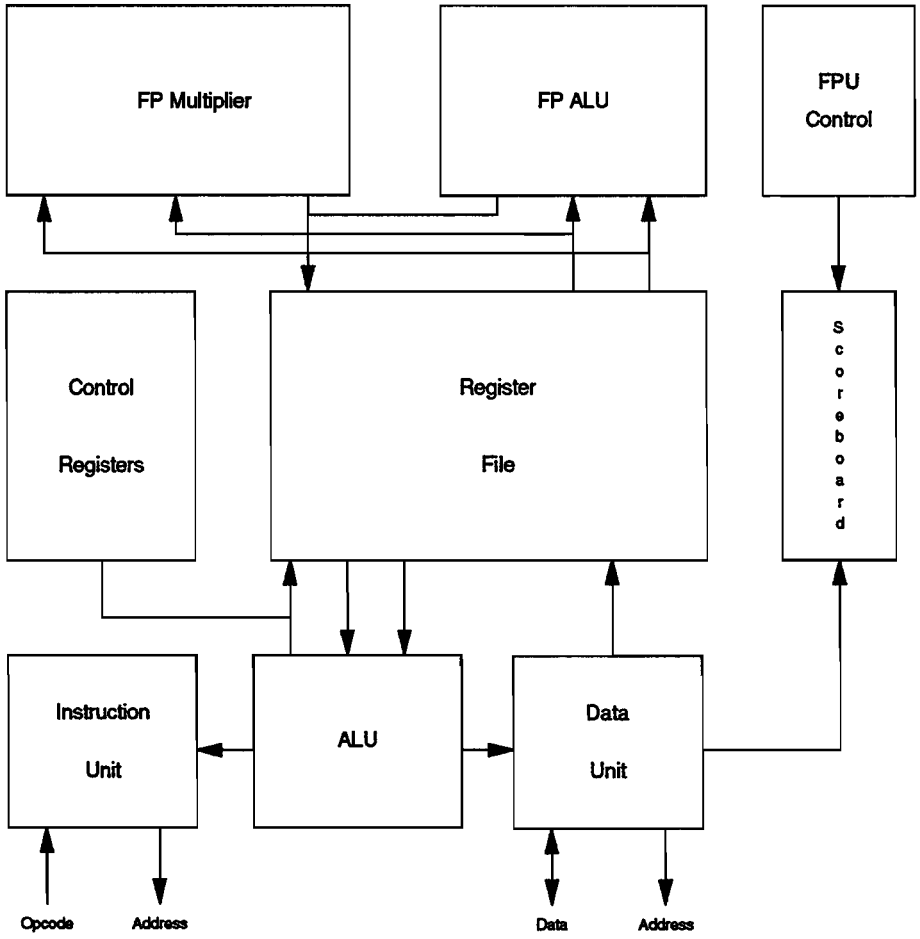


**Figure 5. Block diagram of the Master Processor (MP)**

## 5. Transfer Controller

The Transfer Controller (TC) is the data provider for the ADSPs and the MP. The processors supply cache miss and data movement requests to the TC. These are queued, prioritized and then serviced. It interfaces through the external pins to the external memory, transferring data directly to the on-chip RAMs.

Data transfers are specifically requested by the processors in the form of Packet Transfers (PTs). A PT is a transfer of blocks of data from a source to a destination. The source and destination can be either on-chip or off-chip. The source and destination transfers are completely independent, they can have different dimensions or use different transfer modes. There are two basic types of packet transfers: Dimensioned Transfers and Guided Transfers.

An example of a Dimensioned Transfer is shown in Figure 6. It shows a packet consisting of two patches of three lines each consisting of 512 adjacent 8-bit pixels. This might be needed for example if two ADSPs were going to perform a 3 x 3 convolution with each ADSP working on one of the patches of lines.

The first patch PQR might represent data to be transferred into ADSP0s data RAM and the second patch STU represents data to be transferred into ADSP1's data RAM. A dimensioned packet request is defined by a series of parameters placed in a fixed format.

The parameters for the example in Figure 6 would be:-

1. **A count**, the number *of contiguous bytes in a line = 512
2. **B count**, the number of steps to form a patch = 2 (number of lines - 1)
3. **C count**, the number of patch steps in a packet = 1 (number of patches - 1)
4. **Start Address**, linear address of start of line P
5. **B pitch**, the linear pitch of 2nd dimension (difference of P and Q)
6. **C pitch**, the linear pitch of 3rd dimension (difference of P and S)

Even though a transfer may be to the RAM associated with a particular processor, it is not necessary for that processor to make the request. These parameters are placed in a fixed format, normally in the parameter RAM of the processor requesting the PT. The start address of the parameters is loaded into a special location in the processors parameter RAM and a special register bit is set to inform the TC that there is a new packet transfer to perform.

As well as the parameters mentioned above, the PT parameter structure contains a *next entry address*. Once a TC completes a packet it (optionally) informs the invoking processor, then providing the special *end-of-list* flag is not set, moves to the *next entry address* and starts processing the next PT. In this way a linked list of PTs can be set up ahead of time and performed sequentially.

The second type of PTs is the Guided Transfer PTs. In these the start address and optionally size of the patches in a PT come from a *guide* table in on-chip memory, rather than being calculated solely from the initial parameters of the PT. Figure 7 illustrates one type of Guided Transfer. There are several variations, for instance the start address of the next patch can be calculated, either with respect to the start of the last patch, or from a fixed base address.

The TC can transfer data between source and destination using different types of PT; e.g. the source could be a complicated variable guided patch, and the destination could be dimensioned. This is very useful for packing off-chip frame buffer memory regions into the on-chip RAM, and greatly improves the efficiency of the work of the other processors.

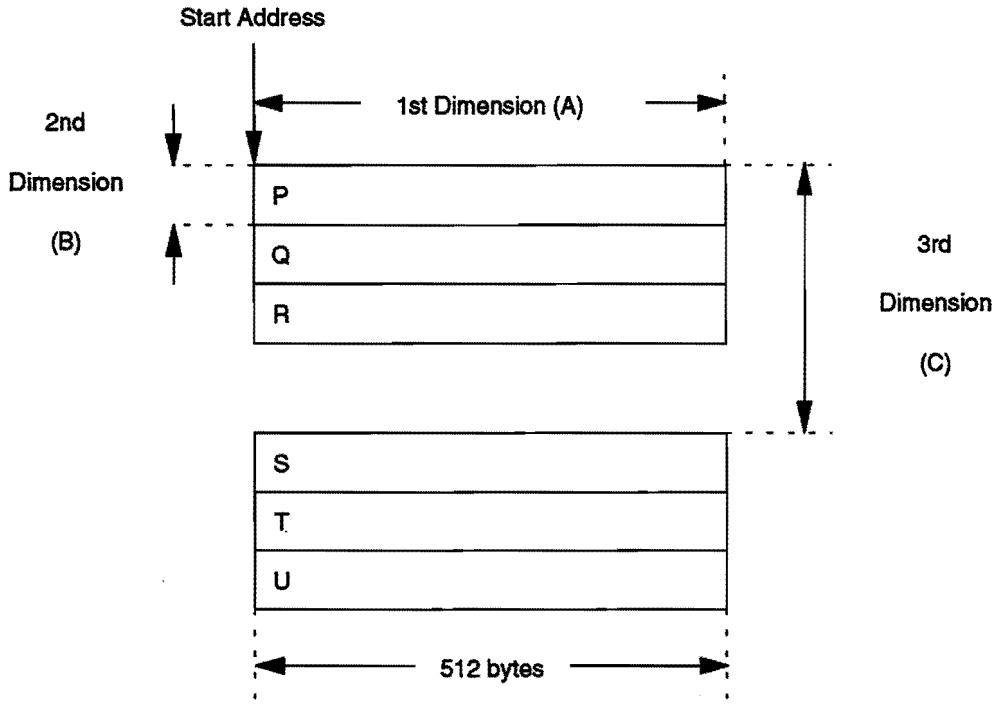


**Figure 6. Dimensioned Packet Transfer.**

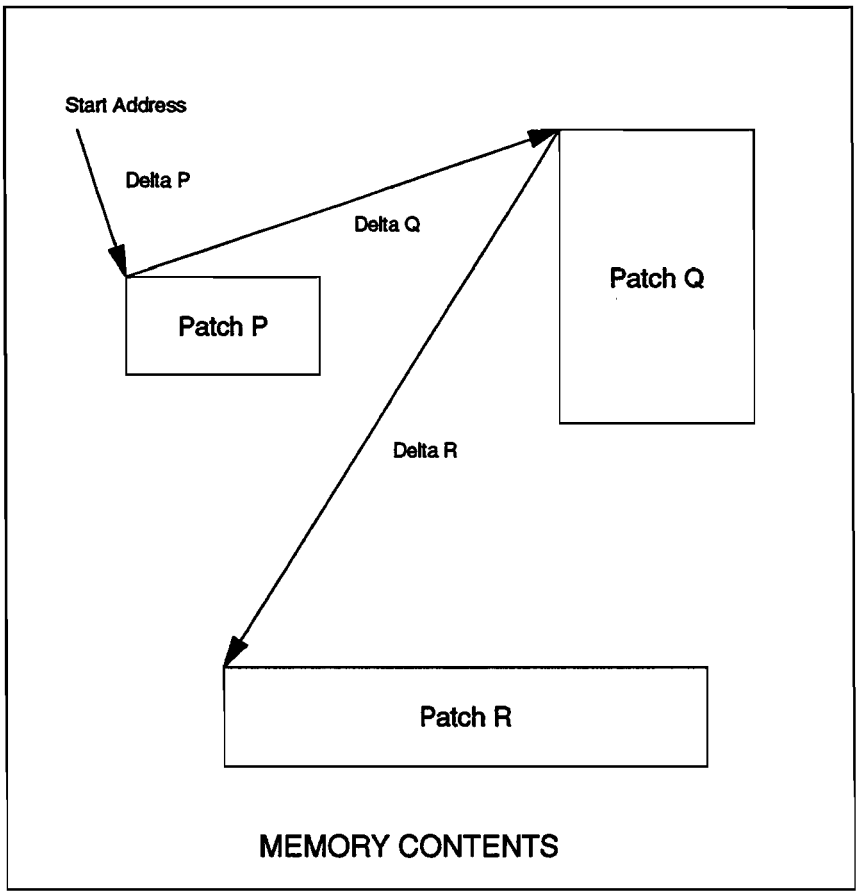


**Figure 7. Variable-Patch Delta Guided Transfer**

## 6. On-chip RAM and Crossbar

The 50K bytes of on-chip memory is physically separated into 25 2K byte RAMs. Each RAM can be accessed 8, 16, 32 or 64 bits at a time. Having a large number of individual RAMs enables many memory accesses to be performed in parallel by the processors. Each ADSP has one RAM (2K bytes) for an instruction cache, whereas the MP has two (4K bytes). These are loaded by the TC on recognition of a cache miss from the instruction control hardware in the processors. In addition, the MP has two RAMs (4K bytes) for a data cache which is also serviced by the TC. Each processor has one parameter RAM (2K bytes), which is primarily for general use, except that part of it is reserved for specific purposes, such as interrupt vectors and Packet Transfer data. The remaining 12 Data RAMs are completely free for general use.

Nominally the Data RAMs are split into four, three for each ADSP. This is only with respect to operations when both Local and Global Address Units are in use, since these share the Global Port for access to all the remaining Data RAMs. When both units are being used, the Local Address Unit can only access the Data and Parameter RAMs that are associated with its ADSP or its access will be stalled until the Global Unit is free.

The accesses into a RAM can switch between reads and writes on a cycle-by-cycle basis. Access to a RAM can also switch between processors on a cycle-by-cycle basis. Switching between processors is facilitated by a connection network called a crossbar. The crossbar allows multiple processors to access many RAMs in the same cycle, though only one processor can access an individual RAM in a single cycle. Each ADSP has two 32-bit

access ports to the crossbar via its Address Units. Both the MP and TC have one 64-bit access port.

The crossbar operation is pipelined. An access can start on every cycle, but each access occurs over two cycles. In the first stage of the pipe, the MSBs of the address that determine which RAM to access are sent out on the crossbar by the requesting processor (or TC). Several processors may request access to the same RAM during a single cycle, but the crossbar logic associated with each RAM determines which processor is granted access to the RAM for that cycle. In the second pipeline stage the processor that was granted access to a RAM sends the LSBs of the address over the crossbar to the RAM to select which bytes are to be read/written, and the data is transferred.

If two processors attempt to access the same memory in the same cycle, the crossbar arbitrates between them. When an ADSP is denied access, its instruction pipe is stalled until access is granted. The MP is slightly different since its registers are scoreboarded. The MP's instruction pipe will not be affected by having to wait for a load operation to complete, until an instruction is fetched which requires access to a register which is the destination of the uncompleted load.

The crossbar uses a combination of priority and round robin mechanisms to perform this arbitration as shown in Figure 8. The most urgent requests are the instruction cache servicing and some timing critical memory operations such as those scheduled by the Video Controller.

When a Packet Transfer is initiated, it is given either *urgent*, *high* or *low* priority. This controls whether the TC's access to the on-chip's RAM to service the transfer should override the MP and ADSP's data accesses.

Crossbar-shared memory is a very flexible multiprocessor memory architecture, since it puts the fewest restrictions on where data needs to be loaded. Since the crossbar involves nearly 1000 address and data lines to be connected between the processors and the memory it is clear that this type of network only becomes practicable when the processors are integrated onto the one device.

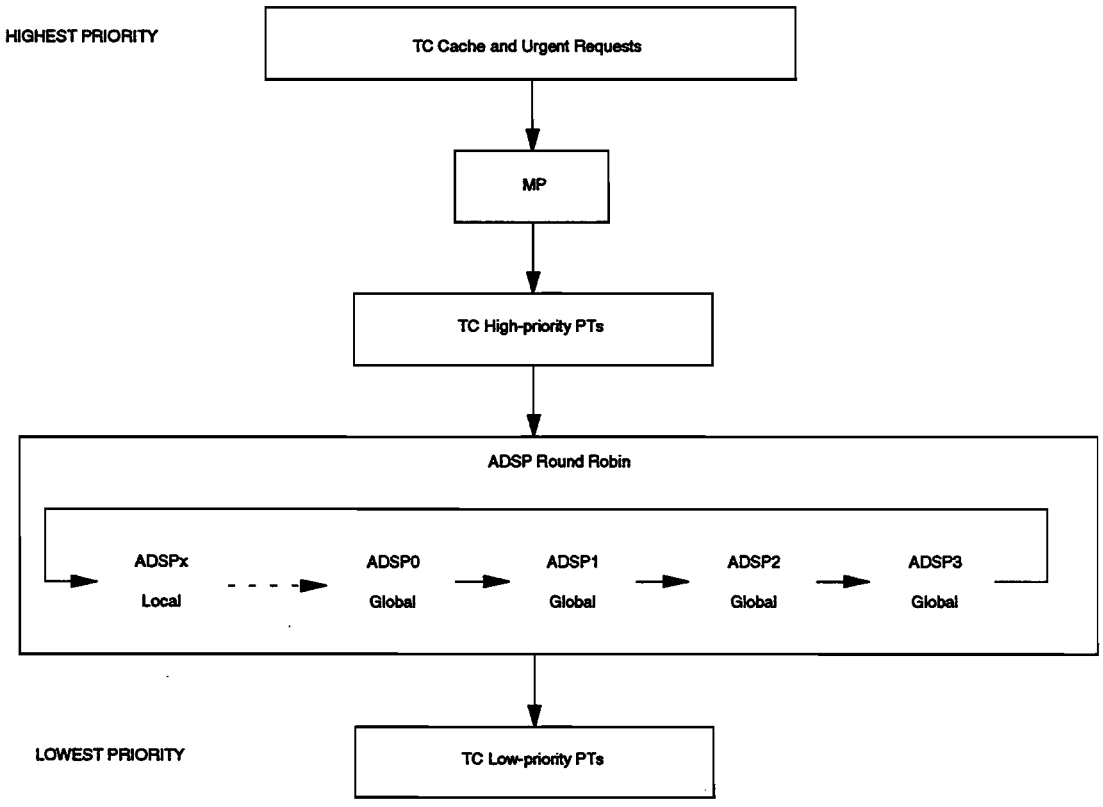


**Figure 8. Crossbar Prioritization Scheme**

## 7. Video Controller

The Video Controller (VC) is the interface between the MVP and the image capture and display systems. The VC, shown in Figure 9, has two independent Frame Timers. Each Frame Timer has its own input clock (FCLK) which operates asynchronously with respect to the rest of the MVP logic. Each timer can generate timing pulses to control a display or capture device.

In addition the VC contains a Serial Register Transfer Controller* which generates SRT requests to the TC to transfer data into and out of VRAM shift registers. The Frame Timers indicate to the SRT controller when an SRT needs to be performed and it generates the required addresses for the TC.

## 8. Summary

Through the use of parallel processing, the MVP puts a new level of programmability and performance on a single integrated circuit. Not only does the MVP integrate 5 processors onto a single device, each processor can execute many operations in parallel. Running at 50MHz clock speed, it is capable of over 2 billion operations per second. It can move 2.4 Gigabytes of data and 1.8 Gigabytes of instructions within the chip, plus 400 Megabytes of data to off-chip memory, each second. This dramatic improvement of single chip performance will make a whole new range of applications possible.

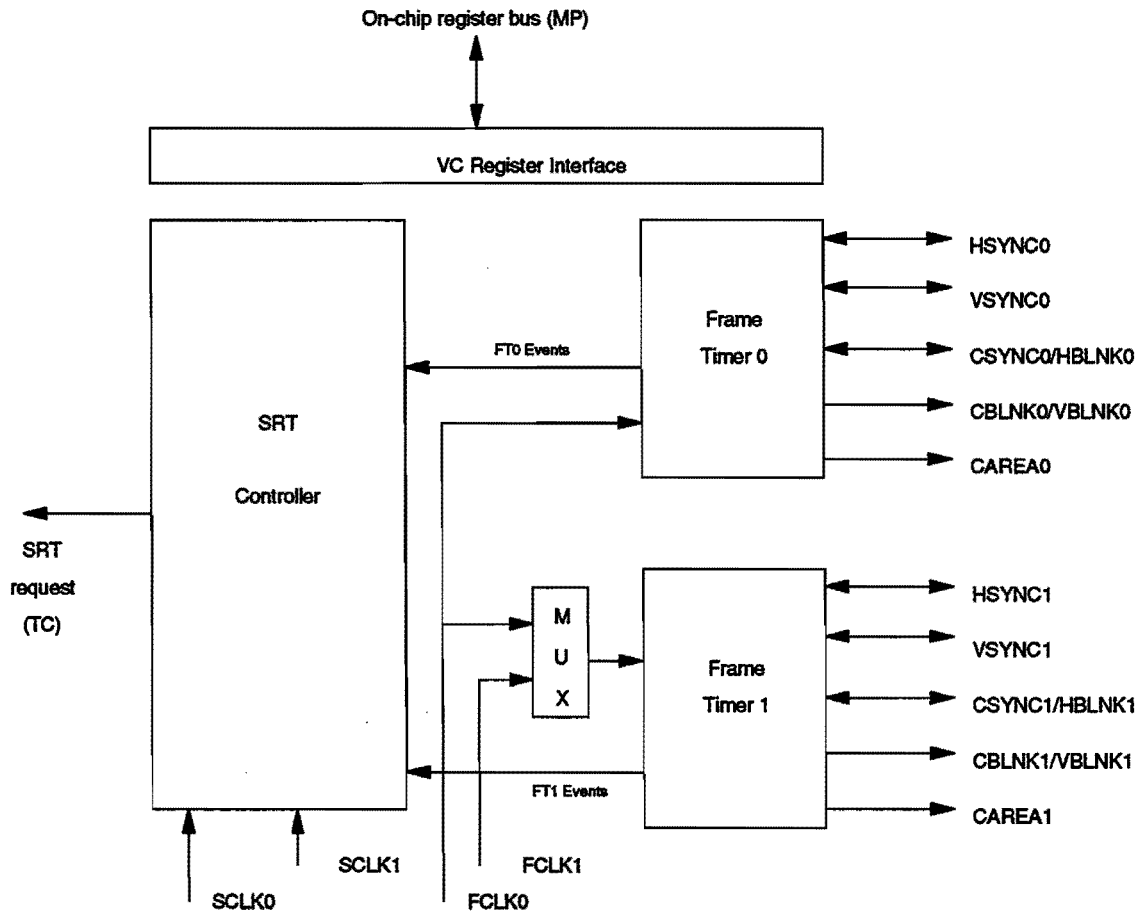


**Figure 9. Video Controller Block Diagram**