The DAN Frame Store

Ian Pratt

February 18, 1993

The DAN Frame Store is still currently in the design stage, though its architecture has been decided. It is hoped to have the PCB layouts ready for the beginning of April. Any information given here is liable to change, and to quote DEC "I make no commitment that this document accurately describes *any* device I may *ever* make."

The DAN Frame Store (DFS) is to be a device that supports an X server type display, but also supports the automatic DMA of video data from the DAN into the frame buffer. The DFS will support at least 64 such video streams. Each stream will be identified by its VCI, and will be DMAed into the 'window' to which it belongs. A number of pixel formats may be supported, different types identified by the VCI. The command stream for the X server will also by arriving on the same DAN interface. These cells will get passed to the CPU by virtue of having the 'loop back' bit set in their port controller route byte.

1 DAN Video Format

The DAN video format attempts to provide a single control structure that will support a number of different pixel formats. The format is designed for networks, and thus has a fairly high control overhead to deal with error recovery. A storage service may choose to filter this out.

The formats supported are all 'still frame' rather than 'motion vector'. This was done because it is not clear that formats such as MPEG are beneficial to transmission over networks. They produce an extremely bursty traffic profile, and have poor error recovery. They also require three or more full frame buffers at each end, which leads to expensive hardware and arguments about how 'real time' they are.

In the DAN format, images are broken down into a large number of 32x8 pixel tiles. 2D tiles were chosen over a line based format because they allow for a 2D compression scheme to be implemented, and are small enough in each dimension to allow pictures to be sent only in sizes that are integer multiples of tiles. The pixel data for each tile is packed into 48 byte cells with no per cell payload overhead. Each tile of data is preceeded by a single control cell. This contains amongst other things, the x,y co-ordinates of the tile within the video frame, a time stamp, and a clipping bit mask. Control cells are identified by having the AUU bit set, commonly known as *'the bit'*.

The x co-ordinate specifies which tile (as opposed to pixel) it is in the frame. The leftmost tile is tile 0. The y co-ordinate similarly specifies where in the grid the tile belongs. The tiles are numbered in TV scan line order i.e. zero is uppermost. The top bit of the 16 bit y field is used to indicate odd and even fields in an interlaced picture. Interlaced pictures

need to be dealt with differently by the frame store. When DMAing a tile to the frame buffer, it should skip alternate lines. On even fields it will add one full line to the DMA address before starting. The bit in the y co-ordinate should be set to indicate an even field. The frame store can be set to ignore interlace if low resolution pictures from interlaced sources are required.

The time stamp will be a 32bit value, constant over a frame (or field if interlaced), produced from a free running microsecond timer. The time stamp was chosen to be constant over a field to allow tiles to be easily identified as belonging to the same frame. A time stamp is better than a simple frame counter because it allows for variable frame rates, such as might be produced by a video storage server.

The clipping bit mask is a 32 byte field, one bit per pixel. It specifies whether the pixel will actually be written to the frame buffer during the DMA or dropped. A '1' causes the transfer to occur. The order of the bits is left to right then top to bottom. The least significant bit in every byte (bit 0) refers to the left most pixel (little endian bit ordering). This bit mask is intended to be used for dealing with overlapping windows, non rectangular windows, free form windows etc. The video stream can be piped through one or more DAN clipping nodes (small DSP processors), that will perform a logical AND operation on the bit mask field, each imposing there particular constraint.

Control cells also contain optional length and CRC fields. These are necessary for some compressed data pixel formats. Compressed data will be of variable length, and may not contain special 'end of tile' bytes hence the length field provision. The length recorded will be the pixel data payload, excluding the control cell. Another feature of some compression schemes is that corrupt data can cause the decompression engine to wedge horribly. A CRC field goes some way to prevent this. It is intended that uncompressed data formats will not use CRC's.

The position in the control cell of the CRC and length field was chosen to be in the same as in AAL-5. i.e. aligned to the end of the cell. This may mean that AAL-5 CRC generation hardware can be persuaded to produce these CRCs. Whether this will prove useful only time will tell.

Note that no information specifying the pixel format in use or the frame size etc. is needed. This is all implicit in the VCI.

Bytes	Name	Use
00h - 01h	X co-ord	X co-ordinate of tile in whole tile units.
02h - 03h	Y co-ord	Y co-ordinate and field indicator.
04h - 07h	Time Stamp	Frame / field time stamp in μs.
08h - 27h	Bit Mask	Clipping bit mask of the tile, '0' = mask.
28h - 29h	reserved	Reserved in AAL-5, spare if we need it.
2Ah - 2Bh	Length	length in bytes of the pixel data payload.
2Ch - 2Fh	CRC	CRC value for whole tile.

Table 1: Control Cell layout

2 Pixel Formats

As mentioned previously, pixel data will be packed into the 48 byte payload area. It is envisaged that a number of pixel formats may be supported :

- 24 bit RGB. The native format of the frame buffer, but costly in terms of bit rate required. Sixteen cells are required to contain the pixel data. This will be the first to actually be implemented.
- 8 bit RGB / Monochrome. This provides a low quality colour or monochrome image, and is easily implemented over the 24bit RGB hardware. Six cells of data are required, of which five and a third are used.
- 16 / 12 bit YC_rC_b. By separating the chrominance and luminance data, a fair quality image can be obtained quite cheaply. Since many video sources have YC as their native output, this may mean simpler ATM Cameras. A 3x3 matrix multiplier can be used to transform YC to RGB. 16 bit YC will require eleven data cells, 12 bit just eight.
- Compressed. Using a DCT based still image compression scheme such as JPEG, good quality pictures are obtainable at only one or two bits pre pixel. It is expected that the data for a tile will amount to only one or two cells. Thus, the control overhead / unused portion of the cell can amount to as much as 50% of the bandwidth.

3 DAN Audio Format

Although not an issue for the DFS, I have included my thoughts on a DAN Audio format. No decisions have been yet, but it is likely to be something along the lines of:

- Samples are gathered into 1 millisecond blocks. Each block is proceeded by a Control cell (marked by the AUU bit). The control cell will contain a microsecond time stamp, with the rest of the cell reserved for things like silence suppression etc. One millisecond blocks are sufficiently small such that they can be dropped without too much audio disruption, and do not introduce much latency.
- The samples will be placed into 48 byte cells with no per-cell overhead. All samples will be byte aligned (there will be no madness of trying to pack 14bit CD quality samples). Stereo sources will produce cells containing alternate left and right samples, left first. Since all sample formats are likely to be either 1, 2 or 4 bytes per sample (or sample pair in stereo), the issue of splitting samples across cell boundaries will not occur. Samples should be arranged in little endian (lsb first) order.
- As with video, no explicit sample frequency or mono / stereo indicator is required as this is implicit in the VCI.

4 Frame Store Specification

The feature list of the DAN Frame Store currently looks like this:

- ARM 610 CPU clocked at 20 MHz, Memory system at 10MHz.
- 8MB of workspace DRAM. Supports 100ns sequential accesses.
- 32KB of boot EPROM, byte to word expanded, 800ns access.
- 6MB of Triple Ported DRAM for the frame buffer, enough for two 1024x1024 24bit deep screens. Sequential accesses take 100ns given no clashes with DMA. One serial port is used by video output to provide the display. This implicitly refreshes the TPDRAM. The second port is used by the video input unit to DMA the pixel tiles into the frame buffer.
- 1MB of VRAM to provide an 8 bit pseudo colour overlay plane. A 256 entry colour look up table (CLT) is indexed to find the physical colour. Entries 0 and 1 are reserved to 'window in' to either the first or second 24 bit frame buffers.
- PIO controller with RS232 port and a couple of timers.
- A 64x64 two colour hardware cursor is provided by one of the Xilinx chips. The two colours may be set by programming the reserved 0 and 1 entries in the CLT. The cursor shape is set by writing a two bit (colour and mask) bitmap into some SRAM.
- Programmable display generator implemented on on Xilinx. The basic display will be 1024x864 pixels at 60Hz (The same as DEC Station 5000/25s). More esoteric formats avaliable will include one for the University of Cambridge 'Rainbow' 3D TV display. The video output bus is four pixels wide to get the necessary bandwidth and feeds an RGB DAC.
- 'Probably' ATM interface. All cells with the 'loop back' bit set will be destined for the CPU and will arrive at the 'Probably' ATM interface. Should bit 2 be set in the port controller route byte, a FIQ will be generated. This high priority interrupt cell is intended to be used to re-enter the boot EPROM, which will decode the cell as a form of GDB remote debugging protocol. This includes operations such as read/write register, read/write memory, jump to location. This protocol will also be used for booting purposes. Normal non 'FIQ' cells are queued in a 32 cell 'pseudo FIFO' implemented in dual ported SRAM. The TX queue is of similar size, and implements an 'infinite hardware retry' policy. Head of line blocking (which shouldn't occur on a *working* DAN anyway) can be detected by a watch thread running periodically, or a full TX queue.
- The Video De-Multiplexer (VDMux) arbitrates between the different pixel format decoders that may be connected (e.g. raw RGB, JPEG compressed etc). It contains the 'window state' for each of the video windows that are active. This includes X,Y screen position of the window, which of the two frame buffers to use, and the interlace mode. Up to 64 windows can be open at a time. The VDMux also contains state about the current tile for each window. This includes the X,Y offset of the tile in the window, the bit mask to use, and the number of pixels remaining to be DMAed, and the calculated DMA address.
- Video DMA engine schedules requests from the VDMux against processor and video out TPDRAM accesses. DMAs can not cross scan line boundaries, and occur two pixels at a time every 50ns. Presented with each pixel pair are two mask bits. The

DMA address is supplied by the VDMux. DMA of over a million pixels per frame should be possible, enough for 4 full size TV images at full frame rate. This should be in excess of what the DAN can source even at high compression ratios.

- Pixel decoders are connected to the ATM interface, one per format being used. Each decoder is associated with a particular 1st VCI byte. When it sees a non-loop back cell with the right 1st VCI byte, it accepts it. If it is a data cell (AUU bit not set) it starts converting it to 24bit RGB suitable for DMA. Control cells are stored and later passed to the VDMux unit when any data previously pending has been DMAed. The control cells need to be held back like this to ensure that any pending DMAs go to the place indicated by the previous control cell for that stream. The 2nd VCI byte is used to identify to which stream a cell belongs and is unique across all pixel decoder units (this is the byte the VDMux identifies streams by). If a need were felt to increase the DAN bandwidth into the frame store, each pixel decoder could use a different port on the switch fabric. The VDMux and DMA engine should be able to cope.

5 Construction

It is hoped that by using high density MACH PALs that the CPU, DRAM, 'Probably' interface, DMA engine and VDMux can be packed on to a single PCB. The TPDRAM and VRAM will be surface mounted onto another board, along with the video output circuitry, display generation and cursor hardware. Pixel decoder boards will plug in to a bus on the main CPU board. The method for connecting boards together will be the 'bed of nails' approach - hundreds of turned pin sockets with long pins.

6 Software

The DFS will run the lower level part of an X Server. Exactly where the partition between the DAN CPU Node and the DFS will lie is yet to be decided. It is likely that the X Server will be 'double headed' in that it will be both 8 and 24 bit. The 8 bit overlay appears to the CPU as 4 pixels per word, which makes blitter operations significantly quicker. Eight bit displays are thus preferable for all windows that are not going to make good use of 24 bit colour.

As mentioned previously, the EPROM will support GDB remote dubuging protocol, and this will be used for booting. This was chosen in preference to the bulky Wanda boot protocol. Initially, before the 'Probably' interface is commissioned, serial line booting may be used.

7 Finally

More detailed documentation in the form of 'Programmers Guides' are being written, and will appear in /homes/iap10/doc/DFS as parameters become finalised.