A dynamic hardware video processing platform

Ray Andraka

Andraka Consulting Group
16 Arcadia Dr, N. Kingstown RI 02852-1666

ABSTRACT

Image processing typically requires either a very low frame rate or a considerable amount of dedicated hardware to achieve satisfactory results. Further, custom algorithms often require development of the entire video capture and processing system. Reconfigurable logic can be used to realize a dynamically alterable image capture and processing system. Algorithm changes at the frame rate are made possible with high speed and partial reconfiguration. Partial reconfiguration permits common functions such as video timing and memory control to be kept in place while the custom processing algorithms are dynamically replaced or modified. The use of a common framework with application overlays also allows the designer to concentrate on the image processing algorithm instead of worrying about the background functions. The use of reconfigurable logic for image processing provides the flexibility of a general purpose DSP with the speed of dedicated hardware.

Keywords: FPGA, image processing, video, reconfigurable computer, dynamic hardware

1. INTRODUCTION

Image processing, simply by the sheer volume of data to be processed, is very demanding of processing power. Even some of the simplest algorithms are computationally intensive, often requiring several multiplies per pixel processed. The processing throughput requirement virtually eliminates the use of conventional DSP processor chips as the compute engine in most serious imaging systems. Until recently, the only alternatives were use a large array of DSP processors, or to design custom pipelined hardware.

The performance of a single general purpose DSP microprocessor is not usually adequate for video applications. Even with advances in DSP processor design such as on chip cache, RISC code, out of order execution, and branch prediction, these processors still suffer from the inherent serial nature of their instruction streams. Performance of DSP systems is frequently accelerated by using arrays of processors. This presents an often formidable software partitioning challenge. Fortunately in video systems, the partitioning is often easier than in other systems. The return realized by parallel processors is diminished as the number of processors is increased due to the overhead needed to distribute the processing. The high software and hardware costs, coding challenges, and limited performance combine to make this approach unattractive in many applications.

Systems based on hardware typically take advantage of the low level parallelism and control not available to DSPs to accelerate the processing. This avoids the problems associated with the serial instruction stream inherent to DSPs, but at the cost of custom hardware. Algorithms realized in hardware traditionally give up the flexibility available with software systems. Hardware algorithms also do not enjoy the maturity of development tools or the fast turn-arounds of the software solutions. In order to meet cost and performance goals, developers of hardware systems have been more or less forced to rely on custom silicon. Unfortunately, the high costs associated with ASIC developement puts hardware performance out of reach for most video systems.

The advent of Field Programmable Gate Arrays (FPGAs) has reduced the penalties of a hardware solution by improving flexibility, development time and costs. This brings the hardware solution to a whole class of applications that could not afford the traditional DSP approaches. Still, the prevailing use of FPGAs has been for fixed logic functions, and normally not as part of the signal processing data path. In many applications, the reconfigurability of SRAM based FPGAs can be exploited to realize significant savings in hardware, power and board space. In those cases, the device is configured with only the logic that is needed at a particular instant. When that logic is no longer needed, it can be replaced with new logic.

By continually repeating this process, a sort of "virtual hardware" machine is created. Devices that permit partial reconfiguration, such as National Semiconductor's Configurable Logic Array (CLAy) family, have the additional advantage of being able to quickly replace small portions of the circuit while the rest continues to operate. This paper presents an imaging system based in reconfigurable logic and designed to accept dynamically configured custom algorithm overlays.

2. BACKGROUND

The dynamic hardware video processing platform grew out of a need for a low cost, high performance video processing system for an entertainment application. The prototype system was implemented in a cluster of workstations and a video capture front end. It took 15 to 20 frame times to process each frame of captured video, so response to moving objects in the image was poor. The need for a hardware acceleration was obvious, but the cost looked prohibitive. About the same time, National Semiconductor completed their development of an evaluation board for their Field Configurable Multi-Chip Module (FCM). The FCM is an array of four modified CLAy configurable logic array devices tiled onto a single substrate, creating a nearly homogeneous array of 12,544 configurable logic cells. The evaluation board, dubbed the FCM*Fun*[tm] (Field Configurable Multi-Chip Module Function board) offered an easy and cost effective route to converting the algorithm to hardware. By using partial reconfiguration and breaking the algorithm into phases, the entire hardware processor could be realized inside a single FCM. The FCM*Fun*[tm], presented a working platform, with memory and a host interface, that could easily be expanded to meet the needs of this project.

At first cut, the design was to be a point design, optimized to the application. Early in the project, discussions of possible algorithm enhancements made possible by a hardware realization along with additional potential applications shifted the focus to a more flexible video processing platform. This shift in focus led to the concept of a foundation providing the basic resources such as memory control and video timing with dynamically overlaid applications. While this complicated the initial design, the benefit is starting to be seen as reduced time to develop new applications. The overlay approach also gave the benefit of providing more usable hardware to each application, since only the hardware for the active overlay needs to be in place.

3. GENERIC VIDEO PROCESSOR ARCHITECTURE

In order to develop the video processing system, it was first necessary to define a generic video processor architecture that could be used as the basis for a flexible platform. The definition of a generic video processor can be elusive. Nevertheless, there are common elements to nearly every system. These include a conversion of the video input to a form that can be processed, a frame buffer memory, host system interfaces and sometimes additional memory for intermediate results or data.

A system front end of some kind is normally required to convert the video to a processable form. This generally consists of an Analog to Digital Converter (ADC) and the associated analog stuff needed to properly condition the signal. Also, a mechanism for synchronizing the processing hardware to the timing of the video source is needed. Most raster video systems employ a protocol of synchronization pulses transmitted during the image retrace times. Recovery of the frame timing requires detection and sometimes separation of these pulses from the video signal. The conditioned pulses are used to synchronize a local timing generator to the video source timing. For this scheme to work, the locally generated timing has to be identical (within tolerances) to the source timing. The timing formats normally adhere to one of several dozen standards. A generic timing chain has to have the flexibility to generate timing for any timing scheme likely to be used.

Most video processing needs to access the image data in some order other than the one it is presented by the video source. To provide this access, the image data for each frame has to be collected in a frame buffer memory that is randomly accessible by the processor. That frame buffer has to include sufficient memory to store at least one, and preferably several image frames, plus the addressing and control logic to stream the image in and read it out. Most processes require more time than the vertical retrace time, so some sort of double buffering capability is also required. The double buffering allows one frame to be collected while the previous frame is processed.

Another common element is an interface for system setup, control and frequently processed data output. Additional memory is also handy for many applications that need storage for program scripts, intermediate results or buffered output. Some systems are used to modify an image rather than analyze one. Those systems need a video output as well. Figure 1 shows a block diagram of a generic video processor for a raster video system.

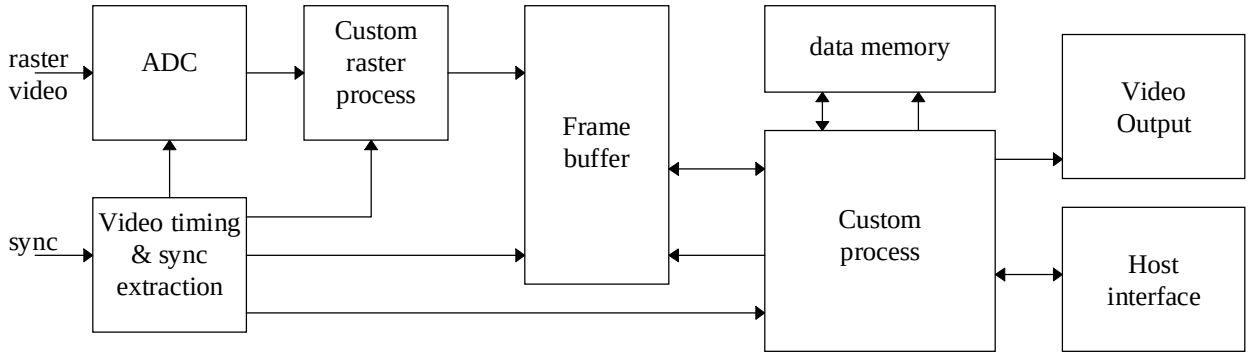

Figure 1. Generic video processor system

The system front end consisting of the Analog to Digital converter and its associated logic can be fairly generic. A color ADC with a reasonable sample rate (about 20 MHz) and the ability to set the upper and lower references for the ADC provide a great deal of flexibility in the front end. Other nice features are selection of sync source and polarity and flexible encoding formats (number of bits in conversion). A Brooktree BT254 triple 8 bit ADC provides all of these features, so it was selected as the heart of the system front end. Programmable current sources on the ADC are used to independently set the upper and lower reference level for each of the three ADC channels.

Often, it is possible to do some processing on the rasterized video. An example of this is color space conversion. Since there are a number of possible preprocessing algorithms that could be done on the incoming video stream. A truly generic system will have the hooks needed to put in a custom raster process.

Most algorithms will need to access the data in a different order than a raster sequence, so a frame buffer is required to collect the rasterized image. Once collected, the custom algorithm needs to have random access to the stored image, plus somewhere to store results. In order to permit simultaneous image capture and processing, some form of dual port or ping pong memory is needed for the frame buffer. In order to satisfy the concurrent processing and collection and to provide a result area large enough for a full frame, the frame buffer needs to hold three full frames worth of data. For algorithms that do not have a need to store frame sized results, a 2x frame buffer is sufficient. Using a maximum pixel rate of 20 MHz and a minimum frame rate of 30 Hz dictates an absolute maximum of 667K pixels in a frame. The overhead for retrace realistically cuts this figure to something less than 512K pixels. A 2 megabyte deep frame buffer will provide storage of 4 frames containing 512 lines each, and up to 1024 pixels per line. By using VRAM, the concurrent capture and processing is met, and the need for external de-rasterizing shift registers is eliminated. VRAM is a special dual-ported DRAM with a conventional DRAM interface plus a static shift register that can be transferred to or from the DRAM a line at a time. Obviously, the VRAM will need a controller to orchestrate refresh, shift transfers, and accesses by the custom process.

The flow of video data into the frame buffer has to be synchronized to the video source for coherent capture to occur. Normally, the timing from the video source is provided as synchronization pulses that occur at the beginning of each field and each line. These pulses are either mixed into one of the video channels or provided as a separate sync signal. The ADC has circuits in it for detection of the sync signal. However, once the signal is detected it must be used to lock the processor timing to the video source timing. The timing chain then produces the various control signals needed to regenerate the image, such as the start of the active video on a line. This timing can change for different systems depending upon the number of pixels in a line, the horizontal and vertical sweep frequencies and the number of lines in a field. For a generic video system, the timing must be easily adjustable to suit the immediate purpose. The pixel clock also needs to be adjustable over a wide range with a small resolution. Having the ability to pick the right pixel clock for an application demands a carefully designed phase lock loop (PLL) capable of operation over a wide range without component changes. The National Semiconductor CGS410 is one such PLL, as it was designed for pixel clock generation. The variability in timing formats between different systems requires a flexible timing chain. To provide this flexibility, the video timing should be implemented in reconfigurable logic and clocked by the PLL. Image jitter caused by sampling the sync pulse can be reduced significantly by oversampling.

The captured image is processed by the custom application. The exact process differs from application to application. The processor generally has to read data from the frame buffer, perform some algorithm, then return the result to another buffer. The access to the frame and result buffers is frequently nonlinear, requiring complex address and control sequences. The variability of the address and control structures dictates that these circuits must be part of the custom application overlay rather than a part of the generic platform logic.

The details of how the processor communicates its results to the outside world, either via a host system or through a dedicated video data port, varies from system to system. Nevertheless, the interfaces themselves tend to be defined more by the systems they connect to rather than by the processor. That being the case, it is possible to define generic interface blocks that act as an intermediary between the processor and the outside world.

This survey of a generic video processing system reveals a significant fraction of the system can be defined without specific knowledge of the application. It follows that these common blocks could be provided as a foundation to build a custom system on, providing a vehicle for rapid system development. Using reconfigurable logic for the custom portions of the processor permits the function of the system to be changed simply by reconfiguring. Partial reconfiguration allows the common circuits to remain in place and even operating while reconfiguration changes the processing algorithm. The total hardware required for a system can be reduced when the algorithm can be partitioned into time-exclusive subfunctions. Even in simpler systems, the approach of using a generic platform with application overlays gains the benefits of decreased development time, increased debug capability, and outstanding flexibility.

4. VIDEO PROCESSOR PLATFORM

4.1 Physical hardware

The video processing system was developed around the FCM*Fun*[tm] evaluation board. This board is a 16 bit ISA card, designed to plug into a PC. The board is host to an FCM reconfigurable multi-chip module. To support that module, the board contains a hardwired ISA interface (BT & BC in **Figure 2**) and an additional CLAy 31 logic array. The board also sports a 128K x 32 static data RAM (DM) and a configuration memory (CM) for storing logic configurations. The configuration memory will accept industry standard RAM or ROM The FCM*Fun*[tm] has four 130 pin expansion connectors that are connected directly to pins on the FCM, plus three headers for additional test points and data connections. The FCM*Fun*[tm] block diagram and board layout are shown in Figure 2 and Figure 3 respectively. The CLAy 31 provides the logic to build a sophisticated host interface and automated high speed reconfiguration for the FCM from multiple configurations stored in the CM memory. The FCM provides a rich reconfigurable environment for a complex custom processor. This single module provides the processor overlay area, host interface, data memory, and video output (via expansion connectors) for the generic video system shown in Figure 1.

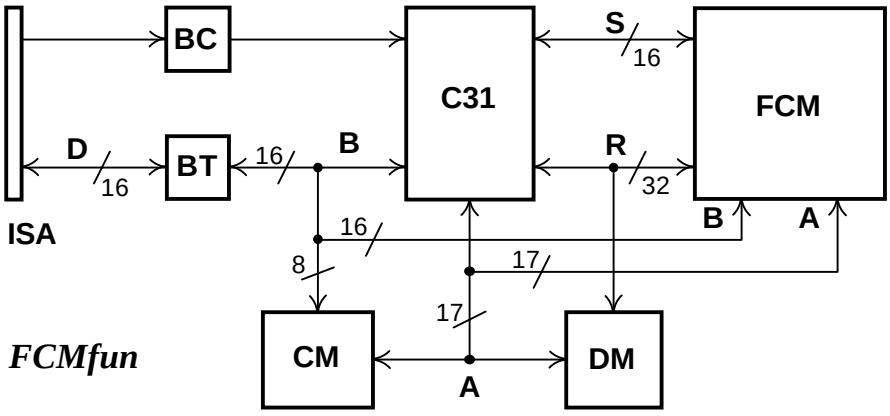

Figure 2. **FCM*Fun*[tm]** *block diagram. (Courtesy of National Semiconductor Corporation)*

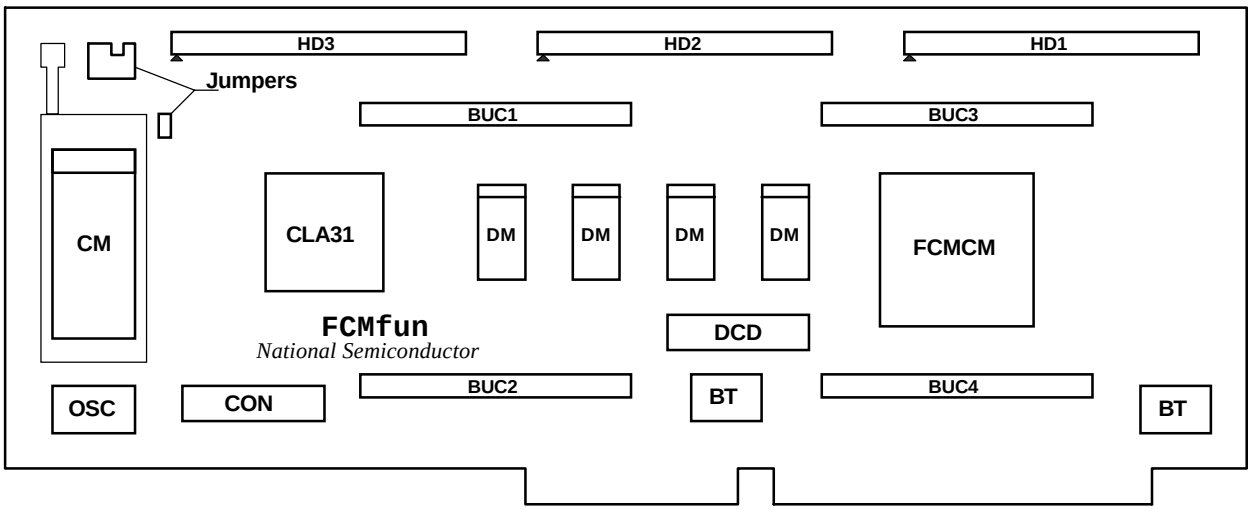

Figure 3. **FCM*Fun*[tm]** *board. (Courtesy of National Semiconductor Corporation)*

The remaining portions of the generic processor shown in **Figure 1**, are provided by logic on a daughtercard that attaches to the expansion connectors on the FCMFun board. The 3x5 inch daughter card provides a video analog-to-digital converter (ADC), frame buffer memory, and a pixel clock generator. The analog to digital converter is a Brooktree BT254 with its associated analog stuff (anti-alias filters and programmable reference amps). The converter's digital outputs are wired directly to the FCM so that in-line processing can be performed on the video data stream, and so that data format in the frame buffer can be redefined by configuration. The frame buffer consists of two totally independent banks of 1M x 16 VRAM to provide the maximum flexibility in its use. The VRAM controllers are in the FCM, so that their designs can be altered as the need arises and to minimize the parts count. A National Semiconductor CGS410 pixel clock generator PLL is used to generate the pixel clock using the 32 Mhz system clock as a reference. Each of these entities has independent dedicated connections directly to the FCM through the BUC3 and BUC4 expansion connectors (BUC is an acronymn for big ugly connector). The small physical size of the daughtercard and the desire to make the physical hardware as flexible as possible drove the decision to implement the memory controllers and interfaces to the other daughtercard components in the FCM's reconfigurable logic.

4.2 Gateware

The majority of the dynamic video processor platform is contained within the FCM reconfigurable logic array. This was primarily done to keep the hardware cost at a minimum and to provide the most flexibility. The initial development probably took longer than using dedicated hardware for functions like VRAM control and video timing. The time invested, however, resulted in low cost logic that is compact, simple to interface, robust, and most importantly redefinable. These circuits were carefully optimized for minimum area and maximum performance to keep the impact to the video system minimal. **Figure 4** shows the relationship of reconfigurable logic (contained within the FCM) to the rest of the video processor system. **Figure 5** shows the physical layout of the logic inside the FCM.

4.2.1 VRAM controllers

The VRAM controllers are realized in the FCM. Each independently accesses a 512 pixel by 1024 line by 16 bit VRAM. Each controller supports both single and fast page mode access and uses a RAS only refresh. These controllers also generate the special sequences needed by VRAM to transfer the shifted line data into the DRAM array. The controllers are clocked by the 32Mhz system clock. Single access cycles are four clocks, while fast page mode cycles are two clocks per word with an additional two clocks to set-up the transfer. By using both controllers, it is possible to get a pixel per clock cycle in burst transfers. Normally the two banks are set up so that odd pixels go into one, while the even pixels go into the other. This way, the controllers can be fed common address and control signals to access two adjacent pixels at the same time. This setup also simplifies the timing of the video transfer to the memory. The write transfer access copies one line of 512 pixels from the shift register to the DRAM array. By interleaving the banks, lines of up to 1024 pixels can be written with one transfer command issued at the end of a line. This eliminates otherwise tight timing constraints that would be needed to handle a mid-line transfer.

The controller includes arbitration of refresh, line transfer, and processor access requests. The interface to the application is a simple request and acknowledge scheme. This ensures the processor gets access in a timely manner while also keeping the memory refreshed and updated. If the contents of the VRAM need to be retained during reconfiguration, the VRAM controllers in the FCM must continue to be clocked to keep the memories refreshed. The video processor platform has been designed to permit clocking during reconfiguration so that image capture and memory refresh can occur while the algorithm is changed. The controller design includes interlocks to prevent spurious memory requests by a changing custom control circuit. If not blocked, such spurious requests could corrupt data stored in the VRAM.

Each controller occupies one six cell wide column of one tile in the FCM for the control and address generation. The data path interface logic occupies 8 cells per bit, arranged along adjacent edges in tiles B and C. This leaves about 80% of tiles B and C open for the data path portion of the application overlay.

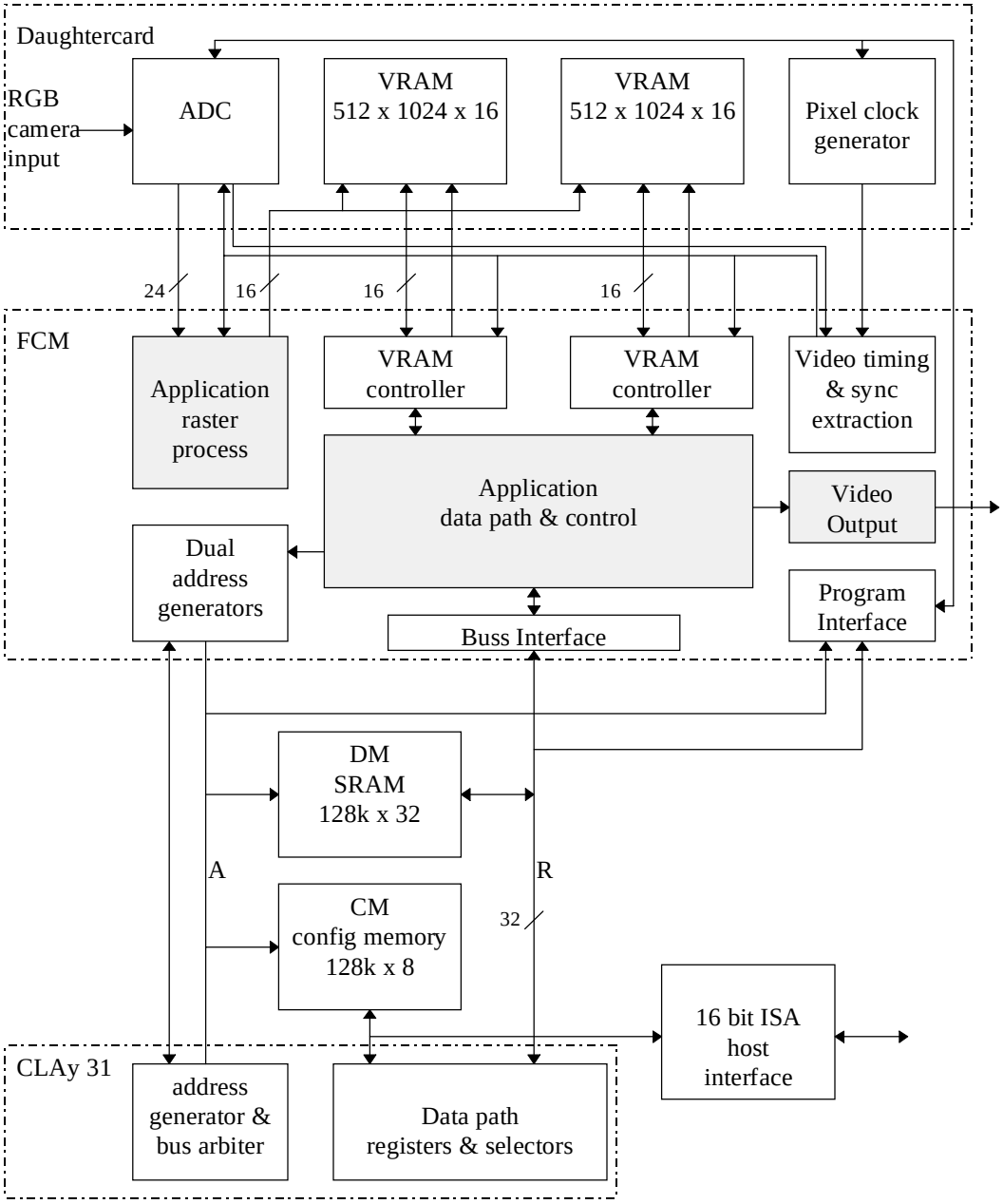

Figure 4. Dynamic hardware video processing platform. shaded areas indicate application overlays.

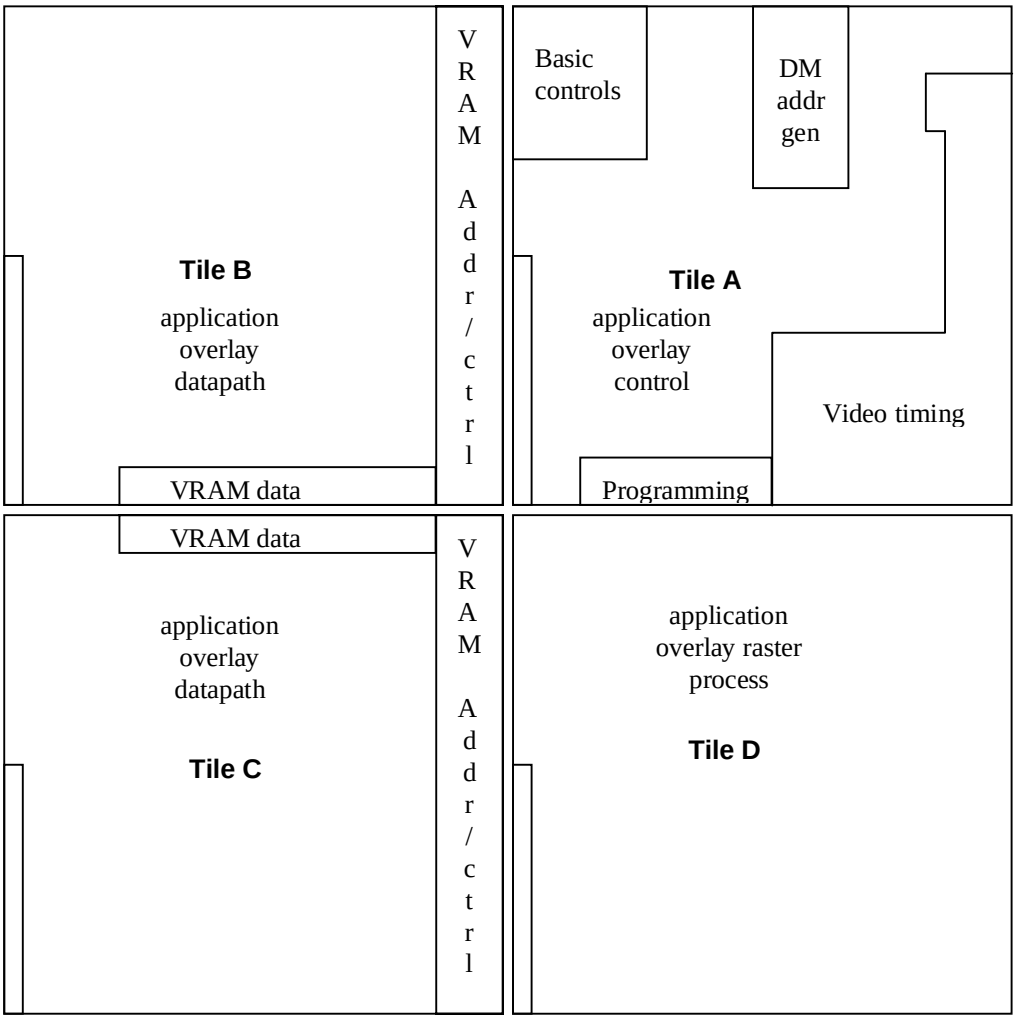

Figure 5. FCM physical layout. The unlabelled boxes on the lower left of each tile are the bus interfaces. Open space is free for application overlay use

4.2.2 Video timing

The video timing generation is also done within the FCM, also to save hardware costs and permit flexibility. The timing chain is set up as a counter whose outputs are compared to a small look up table. Each time the counter matches the look up, the address into the look up is incremented and the corresponding action is taken. Actions include such things as start and stop of the active video, opening and closing the noise gate, sampling for vertical sync, and the end of line reset. Figure 6 illustrates the video timing logic. This architecture permits the timing to be adjusted simply by changing the values in the look up and reprogramming the PLL on the daughter card. As it turns out, this arrangement also permits a higher clock rate than conventional decoding. The video timing circuit works with 4x pixel clock rates up to 91 Mhz. The video timing uses a 4x pixel clock to minimize the jitter associated with sampling the sync pulse, and to provide finer control of event timing in the horizontal line. The timing generator occupies roughly 15% of tile A in the FCM.

4.2.3 FCMFun bus interface

Compromises on data memory access had to be made to utilize the FCMFun board, since the host access and the data memory share the same bus. The FCMFun bus interface encompasses the buss data registers, application data memory (DM) address generators, and the address decoder for programming and application use. These are shown as separate blocks in **Figure 4**.

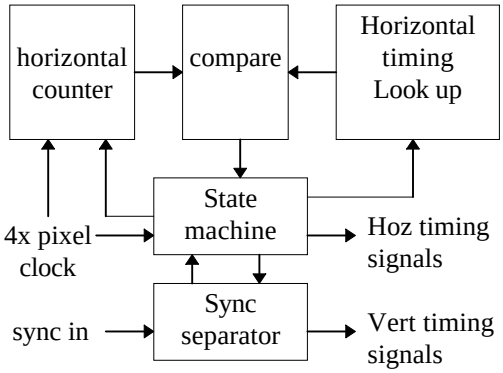

Figure 6. Video timing generator

The video processing platform includes a pair of address counters for application access to the data memory so that separate read and write pointers into the memory may be maintained. The one of the counters is a loadable up-counter, while the other is a loadable downcounter. This combination of counters has worked well for the applications built to date, most of which used part of the DM as a microcode program store and part for a results buffer. By starting one counter at the top of the data memory and working down while the other counter starts at the bottom of memory and counts up, maximal use of the memory is achieved. Of course, if a different addressing scheme is required, these counters can always be replaced by the application.

The FCM*Fun*[tm]'s R buss carries all data between the host interface, the data memory (DM), and the logic in the FCM. This buss also carries the FCM's configuration data during reconfiguration. The traffic on the buss is controlled with an arbiter in the CLAy 31. Bus arbitration is performed on the CLAy 31 to avoid excessive delays for host accesses. The FCM design includes interlocks to ensure spurious buss requests from changing control logic do not corrupt reconfiguration. When the application needs access to the data memory, it sends a request to the arbiter in the CLAy 31 and waits for a buss grant. The buss interface in the FCM consists of data buss registers to improve buss timing, a small buss controller, address decoding, and a simple interface to the programmable components on the daughtercard. The bus interface also provides 32bits of control and status. 5 of these bits are used to control and monitor parts of the generic platform. The remaining bits are available for use by application overlays. The buss interface, control and status registers occupy approximately 2% of each tile. The remaining interface logic resides in TILE A, leaving nearly 60% of that tile open for application overlays. That area is normally used for application specific control and timing logic.

4.2.4 Application overlay area

The application overlay area encompasses both the areas set aside for pre-capture (raster) and post-capture processing, as well as customized control circuits, and any video output logic needed for an application. The video processing platform is specifically designed to permit new applications to be overlaid while the system clock is operating. The application overlay area is loosely partitioned by tile so that raster processing is done in tile D, data path portions of the main process reside in tiles B and C, and application specific timing and control reside in tile A. This suggested partitioning reflects the layout of the interfaces in the VPP design. The ADC's 24 bit output and the 16 bit serial input to the VRAM is wired to tile D, where about 95% of the cells are unoccupied by the foundation logic. Tile D is intended for application specific raster operations, although it may also be used for control or post capture processing with inter-quad connections to tiles A or C. The VRAM address, control and data occupy edges in tiles B and C, making the large open areas in these tiles the ideal place for data path processing. Tile A contains timing and bus interface logic for the foundation. In this case it is logical to overlay the application specific controls adjacent to the generic controls. Each of the tiles have undedicated I/O pins available for application specific I/O. These pins are connected to the unused BUC1 and BUC2 expansion connectors.

5. APPLICATION OVERLAY DESIGN

The ability to overlay applications without having to design, layout, or route the common part of the design is the big advantage of the video processor platform. For complex algorithms that can be partitioned into time exclusive sub-functions, there is the additional advantage of reusing hardware.

The large area of unused reconfigurable logic available in the video processing platform is sufficient for a single overlay solution in many applications. In those cases, a single configuration is loaded at system initialization then left alone after that. In single overlay (static configuration) systems, the design process is not unlike a typical FPGA development. There is no need to consider logic states as a result of reconfiguration or connectivity of overlays, since the configuration is never changes. In these systems, the platform design can be viewed as a partially completed and pre-routed design. That design is simply expanded on to customize it to the desired application.

The application overlay concept encourages dynamically reconfiguring the system, in effect changing the hardware as a part of the algorithm. This technique can drastically reduce the total amount of hardware required. Designing a dynamic system is not without pitfalls however.

First, when doing a partial reconfiguration, care must be taken to make sure all the connections to both the overlay and the underlying design are properly made. Failure to connect an input is likely to cause improper operation and in certain extreme (rare) cases could even damage the device. Not only do the inputs need to be connected, but the in place control structures need to be appropriate to the overlay, and must be in the correct state for startup. If there is enough time available, replacing overlays using a reconfiguration of the entire tile avoids the interconnectivity problems. Reconfiguration of a cell with an identical function has no effect on the cell operation, so a full reconfiguration has the same effect as a partial, but has the added benefit of ensuring the expected underlying logic is in place. Of course, the reduced time to load an overlay using partial configuration is lost when doing a full configuration.

The second consideration is that newly overlaid logic will cause flip-flops fed by it to go into an unknown state unless specific safeguards are taken. The design of the video processing platform's foundation logic includes interlocks to prevent transient circuits corrupting the memories or configuration. Unfortunately, placing such interlocks into an overlay is considerably more difficult, as the order of the configuration becomes important. At a minimum, the design of each overlay should be analyzed for potential malfunction due to unknown states both during reconfiguration and upon completion of initialization. Very little work has been done to date concerning partial reconfiguration in an operating array. Common sense dictates all unprotected control circuits should be initialized to known values at the end of configuration. Where expansion I/O is used to connect to other circuits, care needs to be used to avoid bus contention and indeterminate controls to critical logic (like memory write enables).

Finally, the reconfiguration takes a considerable amount of time to perform. Depending on the size of the configuration and the source of the data (ISA bus vs local configuration memory), the overlay could take longer to load than the maximum refresh interval for the VRAM. If data needs to be retained in the VRAM during reconfiguration, the underlaying logic must continue to be clocked during reconfiguration, and may not be reset upon completion of the reconfiguration. The video processing platform is designed to be clocked during configuration. Any overlaid control logic that is intended to remain active after a subsequent reconfiguration should also have similar interlocks.

The application can access the frame buffer two pixels at a time, at two clocks (32Mhz) per access in burst mode. The algorithm control and datapath logic is also clocked at 32 Mhz. While this requires consideration to performance in the design and layout, the 30 ns cycle time is sufficient for most processing. Heavily arithmetic operations may need to use clock enables to work at the data rate (16 Mhz) instead of at the clock rate. Floorplanning of the design is necessary to ensure acceptable performance and to guarantee the overlay stays within the overlay area and connects efficiently to the foundation. Finally a thorough simulation and timing analysis is highly recommended to increase the likelihood of a successful integration of the overlay with the foundation. In systems with multiple overlays, extra attention should be paid to ensuring proper control and connectivity for all possible sequences of overlays.

The development of the video processing platform included the design of a number of static configuration overlays intended for system debug and test. These overlays included memory tests, pixel clock PLL and ADC exercisers, and a frame

grabber. The static applications avoid the dynamic reconfiguration issues and are sufficient to verify system operation. Later work went into development of the application overlays for the entertainment application the project began with. The first of these is a processor that reads a script from the data memory defining regions in the image. For each region, the processor finds the brightest pixel and puts its location, intensity and color into a results buffer. The processor uses the fast page mode VRAM access to permit processing of the entire frame within a single frame time. This overlay also includes a color space converter that operates on the raster data from the ADC. The color space converter transforms the RGB color space to the two dimensional color space needed by the processor. The color space converter as implemented is an 8 bit 3 x 2 pipelined matrix multiplier running at the pixel clock frequency. The color space converter design fits within a single tile (tile D) of the platform. The next application is an image histogram. This application uses counters and a pipelined decoder to create a histogram of each region specified by a script stored in the data memory. Upon the completion of each region, the histogram counters are dumped to a result buffer in the data memory. At the time of this writing, the bright pixel algorithm is being tested using dynamic overlays to load the frame buffer with test data. The histogram algorithm is currently in design.

6. CONCLUSIONS

Using a dynamically configured system for video processing can save significant amounts of hardware. The original target application, an entertainment system, takes advantage of the dynamic configuration to set up scripts for the main process. The performance of the hardware based system coupled with the ability to completely reconfigure the hardware during the frame retrace time opens the possibility of a very complex multi-function image processor with essentially the same per-unit hardware cost as a single function system. By having a working platform containing the common video system elements, the designer can concentrate on the implementation of his algorithm rather than on system level details. The flexibility of the reconfigurable logic also permits tweaking of the algorithm at a very low cost, and may even be usable for adaptive algorithms designed to mutate at the frame rate. The design time spent in the video processing platform foundation is beginning to yield benefits in the form of rapid product development for specific application overlays. As FPGAs mature, I expect reduced configuration times and improved tools to make reconfigurable systems even more attractive and easier to design.

7. ACKNOWLEDGMENTS

The development of the Dynamic Video Processor Platform was done under contract to National Semiconductor Corporation. The author wishes to thank Loren Carpenter, Al Brilliott, Tim Garverick, Harry Holt and Edson Gomersall for their technical and financial support throughout the development. CLAy and FCMFun are registered trademarks of National Semiconductor Corporation

8. REFERENCES

1. J. D. Hadley and B. L. Hutchings, “Designing a partially reconfigured system,” *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, John Schewel, Editor, Proc. SPIE 2607, pp 210-220, 1995.

2. J. G. Eldredge and B. L. Hutchings, “Run-time reconfiguration: a method for enhancing the functional density of SRAM based FPGAs,” *Journal of VLSI in signal processing*, Volume 12, 1996. Pages 67-86

3. J. D. Hadley and B. L. Hutchings, “Design methodologies for partially reconfigured systems,” *Proceedings of the IEEE workshop on FPGAs for Custom Computing Machines*, Peter Athanas and Kenneth L. Pocek, editors, pp. 78-84, Los Alamitos, California, April 1995. IEEE Computer Society, IEEE Computer Society Press.

4. P. Lysaught and J. Dunlop, “Dynamic reconfiguration of FPGAs,” *More FPGAs: Proceedings of the 1993 International workshop on field programmable logic and applications*, W. Moore and W. Luk, editors, pages 82-94, Oxford England, September 1993

5. C. R. Rupp, “FCMFun[tm] Reference Manual,” National Semiconductor, August 1995. Version 1.0