Low-Cost Video Interface White Paper

May 22, 2003

Contents

Introduction

The Low-Cost Video Interface platform is intended to show a cost-effective means to interface video decoders with ITU-R.656 interfaces (such as the TVP5145 and TVP5150) to the external memory interface (EMIF) of a low-cost processor, such as the TMS320C6204. The method used is to interface the video decoder to RAM connected to the processor's bus through glue logic. This glue logic is implemented in an FPGA (field programmable gate array). The platform delivers both the FPGA programming and the DSP software needed to successfully transfer data from the video decoders into the DSP, transform the data into RGB format (used for computer monitors), and transfer the data out of the DSP to a video DAC to be shown on a computer monitor.

This paper describes the design at a high level. (Details of the hardware and software design can be found in the "Low-Cost Video Interface Hardware Description" and "Low-Cost Video Interface Software Architecture" documents.) The intent is to give enough background to allow the platform to be used as a base for an actual video design. The paper also contains information how to the design can be extended or modified under certain conditions.

System Design Goals

The goal of the system was to provide a low-cost method to interface a video decoder to low-cost members of the C62x/C67x DSP families. It was important to make the interface applicable to as many members of the C62x and C67x families of processors as possible.

System Block Diagram

Figure 1 shows the block diagram for the system.

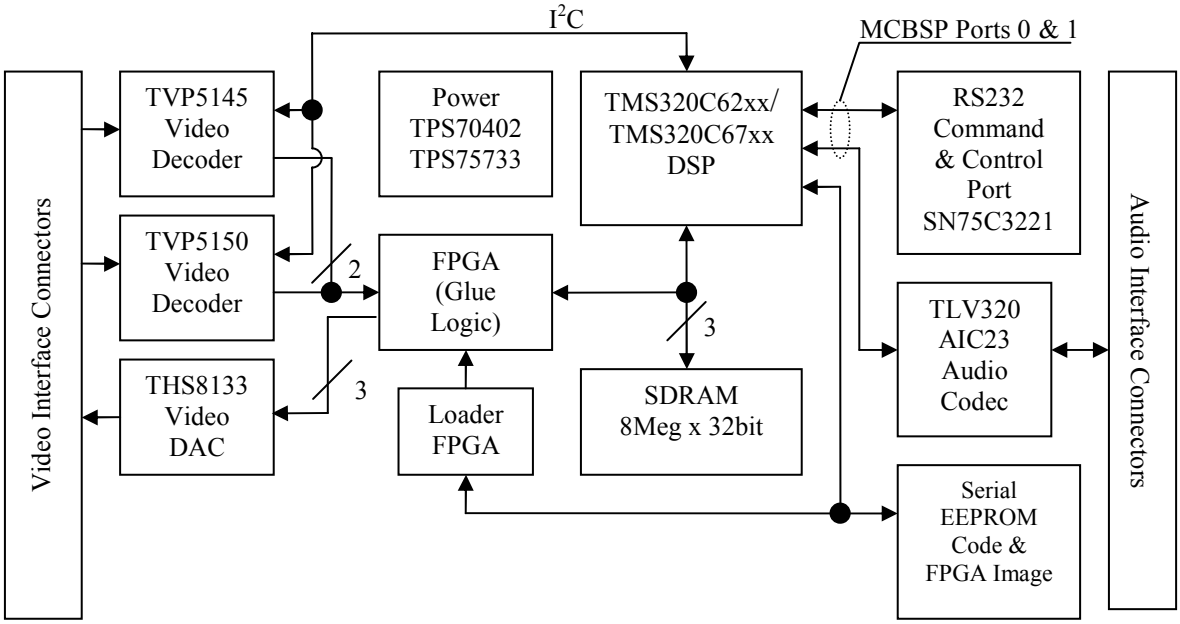


Figure 1. Block Diagram of Video Interface

The basic flow of the video data is from the video interface connectors into either the TVP5145 or TVP5150 video decoder. The digital data from the decoders are output to the FPGA, which places the data into the SDRAM. The DSP takes the decoded data (in YCbCr format), reads it into internal memory, and then transforms it into RGB format. The data is written out to the SDRAM. The FPGA takes the RGB data from the SDRAM and writes it to the THS8133B video DAC, where it can be seen on a computer monitor.

The basic flow of audio data is from the audio input through the TLV320AIC23 codec into the DSP through a McBSP port. The audio data is delayed to match the video processing delay, and then output through the McBSP to the codec to be output on speakers.

A more detailed description of the data flow is given in the "System Operation Overview" section of this paper.

The remainder of this section details the rationale for the processor, memory, and FPGA selections for this platform.

Processor Selection Considerations

The Low-Cost Video Interface platform was designed with consideration of alternate processor selections and, where possible, the design was made to support as many options as possible. Toward this end the following constraints were imposed:

- The host port / expansion bus / PCI bus was completely avoided as it varies or is not present in many members of the C6xxx family.
- The floating point capabilities of the C67xx family were avoided.
- The DMA was limited to 4 channels.
- The McBSP was limited to 2 ports.
- Only two timers were used.

FPGA Selection Considerations

For this design an Altera® ACEX® family FPGA was chosen, specifically the EP1K30.

The choice of an Altera® part over a Xilinx® part was influenced by a less complicated timing model. Other than this, part cost was the driving factor for choice.

The ACEX family of FPGA's was at the time the least expensive of Altera's SRAM based FPGA's that also included enough on board RAM to implement the necessary FIFO's for the video data.

The part used on this platform comes in several package configurations and speed grades. The one chosen is the least expensive package and slowest speed grade offering for this family. It is an inexpensive package partly because it is not a low profile device or BGA device. These factors will ultimately need to be re-evaluated for any design migrations.

Further, the code use on this design can be compiled using Altera's free web based design software eliminating the need for a full design suite of HDL tools and associated costs.

After the platform design was frozen, Altera released its Cyclone family of FPGA's that appear to be capable of replacing the ACEX part used in this design. This new family of FPGA is priced considerably lower than the one used on this platform and thus could provide another level of cost reduction based on this design. Note: No attempt has been made to implement or verify this option.

Memory Selection Considerations

Several memory options were available for use on this design and include SRAM, SDRAM, and SBSRAM (Synchronous Burst SRAM). As the platform was intended to showcase a low cost design, cost was the major driving factor in memory selection.

The choice matrix for the different types of memory is shown here

Category	SRAM	SDRAM	SBSRAM
Cost/Bit	Medium	Low	High
Speed	Medium	High	High
Capacity	Medium	Large	Medium
Interface	Asynchronous	Synchronous	Synchronous

Except for the complexity of a synchronous interface, SDRAM won in all categories and thus was chosen for implementation on this design.

Another memory requirement was storage for the programming image of the FPGA and the processor code.

Should the design require large amounts of processor code then Flash memory would be appropriate to hang on the processor bus. It is recommended to use a 32 bit wide Flash architecture to minimize any asymmetric loading of the bus.

Since this design required only a small code set for the processor, a small serial EEPROM was chosen instead. A boot loader stored in one of the FPGA memories was used to load the processor code into the internal processor RAMs.

System Operation Overview

The purpose of this platform is to specifically demonstrate the capability of the C6xxx processor family to deal with real time video input and output. To this end, an FPGA was used in which input and output FIFO's were created to manage the asynchronous nature of the processor bus and also to implement necessary translations of data required for maximization of bus bandwidth.

Video Decoders

There are two video decoders used on this platform. They are tied to a common bus through tri-state buffers which is managed by the processor through register bits within the FPGA.

The input to the FPGA requires a 4:2:2 YCbCr data format as described by the ITU-R.BT-601 standard. One exception to this is in the number of pixels per line, allowing for the so called square pixel formats to be used providing the ITU-R.BT-601 EAV and SAV codes are embedded in the video data.

Both 525 and 625 line systems are supported.

FPGA Video Input Translation

Because the processor bus bandwidth is maximized using full bus width accesses, the video input stream is translated to a slightly different representation from that of the ITU-R.BT-601 4:2:2 YCbCr standard.

The video data is received from the decoders in the following format:

Cb_0, Y_0, Cr_0, Y_1, Cb_2, Y_2, Cr_2, Y_3, Cb_4, Y_4, Cr_4, Y_5, …

It is translated to the following format for 8 bit data:

Cb_0, Cb_2, Cb_4, Cb_6, Y_0, Y_1, Y_2, Y_3, Cr_0, Cr_2, Cr_4, Cr_6, Y_4, Y_5, Y_6, Y_7,
Cb_8, Cb_{10}, Cb_{12}, Cb_{14}, Y_8, Y_9, Y_{10}, Y_{11}, Cr_8, Cr_{10}, Cr_{12}, Cr_{14}, Y_{12}, Y_{13}, Y_{14}, Y_{15}, …

and is translated to the following format for 10 bit data:

Cb_0, Cb_2, Cb_4, Y_0, Y_1, Y_2, Cr_0, Cr_2, Cr_4, Y_3, Y_4, Y_5,
Cb_6, Cb_8, Cb_{10}, Y_6, Y_7, Y_8, Cr_6, Cr_8, Cr_{10}, Y_9, Y_{10}, Y_{11}, ….

The 10-bit format is packed as 3 10-bit data word in one 32-bit word with 2 bits of padding in the MSB's of the word.

Thus in 8-bit mode, data is always represented as a group of 8 pixels; in 10-bit mode, it is always represented as a group of 6 pixels.

At the end of a line of video, the EAV sequence is written to the input FIFO, and the data is padded with undefined values to complete the 6- or 8-pixel grouping depending on whether the mode is set to 8 or 10 bits. Note: The SAV sequence is not written to the FIFO at the beginning of the line.

Video Input FIFO

The video input FIFO in the FPGA manages the difference between the synchronous video data rate and the asynchronous access rates of the processor.

Translated input video data is stored in a 32-bit by 256-word FIFO. Once 128 words have been stored in the FIFO, the processor bus is requested by the FPGA. After the processor relinquishes control of the memory bus, 128 words of data are burst out to the video buffers in the SDRAM.

Always bursting 128 words guarantees that the FIFO always makes accesses within a single page of the SDRAM, thus minimizing overhead on the bus.

Once a complete field has been received from the input video translator logic, the FIFO is filled with undefined data until a 128 word boundary is achieved. An additional 256 words of undefined data is then pushed into the FIFO to force all valid video data remaining in the FIFO to be flushed out to the SDRAM video buffers.

Finally, the odd/even field bit is set in the FPGA register, an interrupt is generated, signaling the processor that a complete field has been received, and, if the last field received was an even field, the buffer address is reset to the beginning of the buffer. The FIFO then waits for the next field of video data to start.

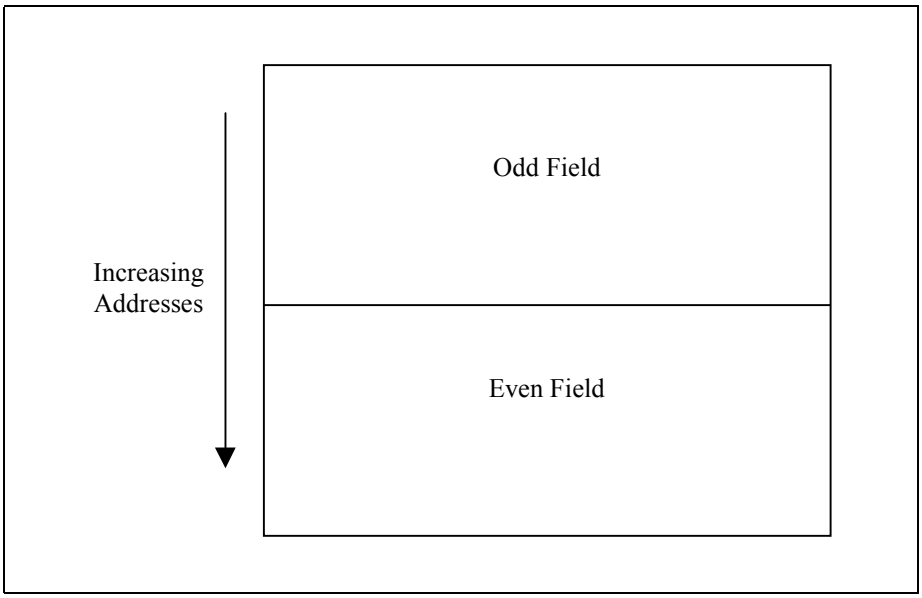
Video Input Buffer

The video input buffer is a continuous region of memory in the SDRAM capable of holding one entire frame of video (both even and odd frames).

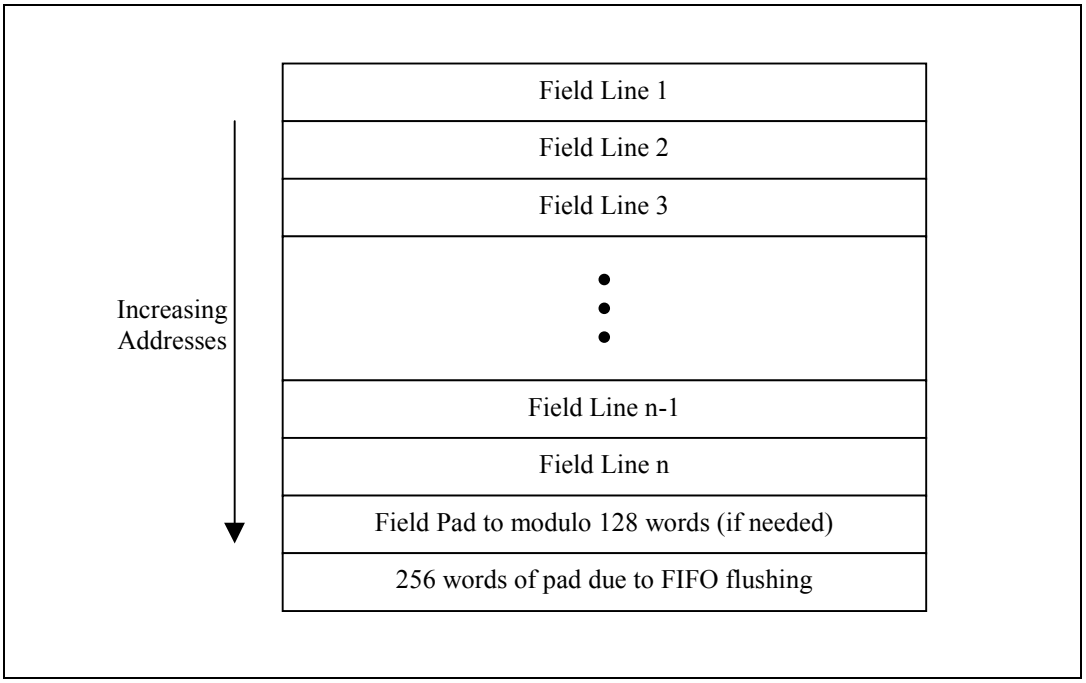
Each line of video data is stored in increasing addresses in memory.

The odd field is stored at the beginning of the input buffer and the even field is stored immediately after.

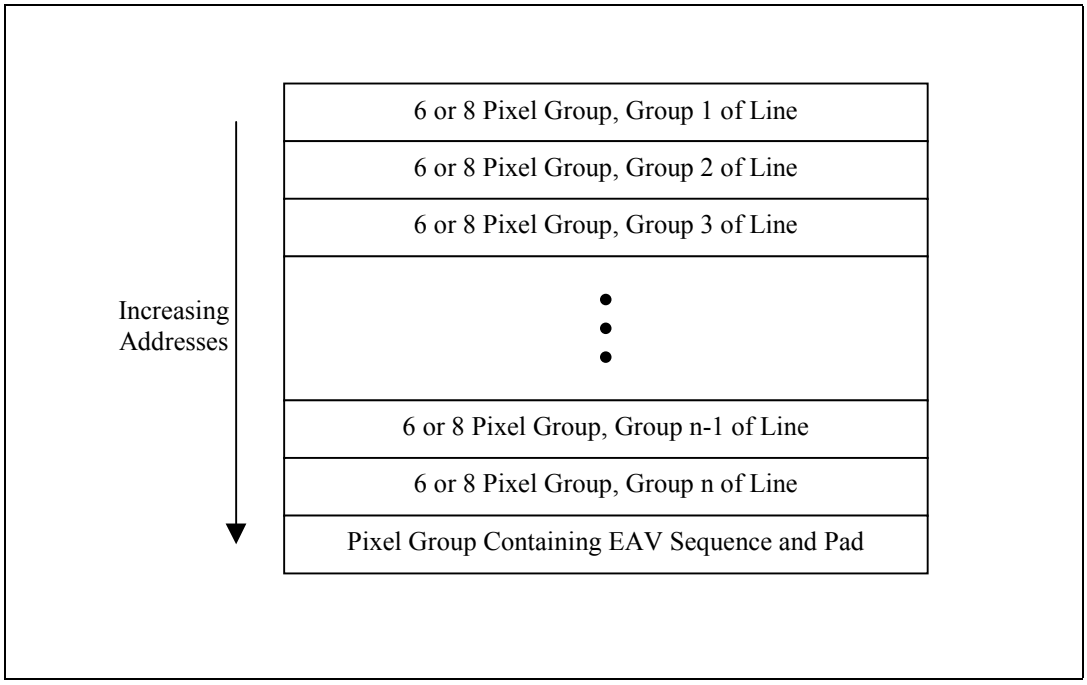
The various structures of the video input buffer are shown in the following diagrams.



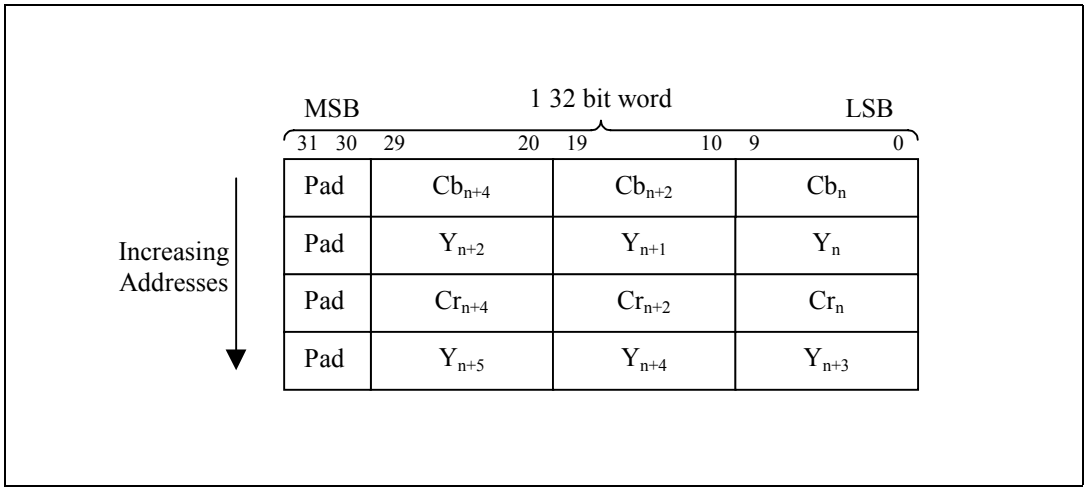
Video Input Buffer



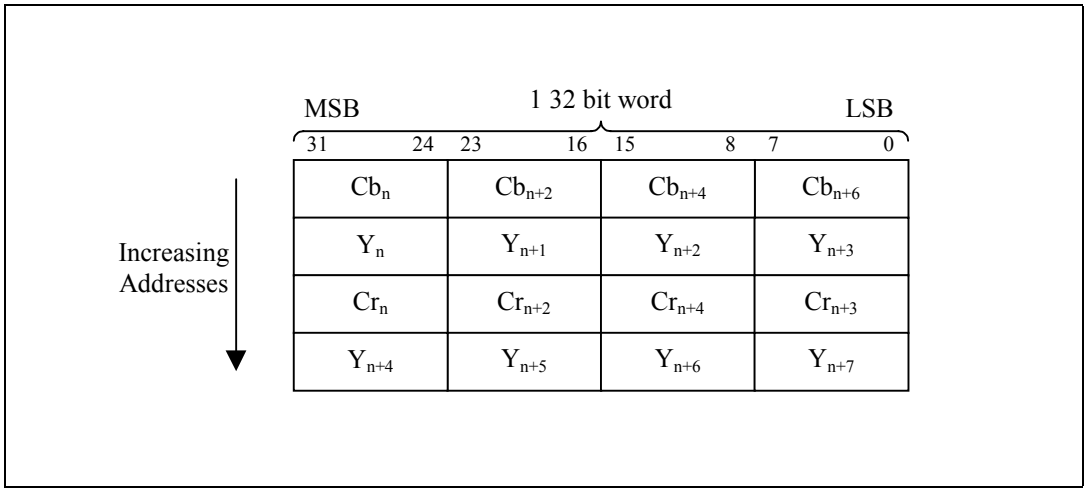
Field Structure within Video Input Buffer



Structure of one line of video in the Video Input Buffer

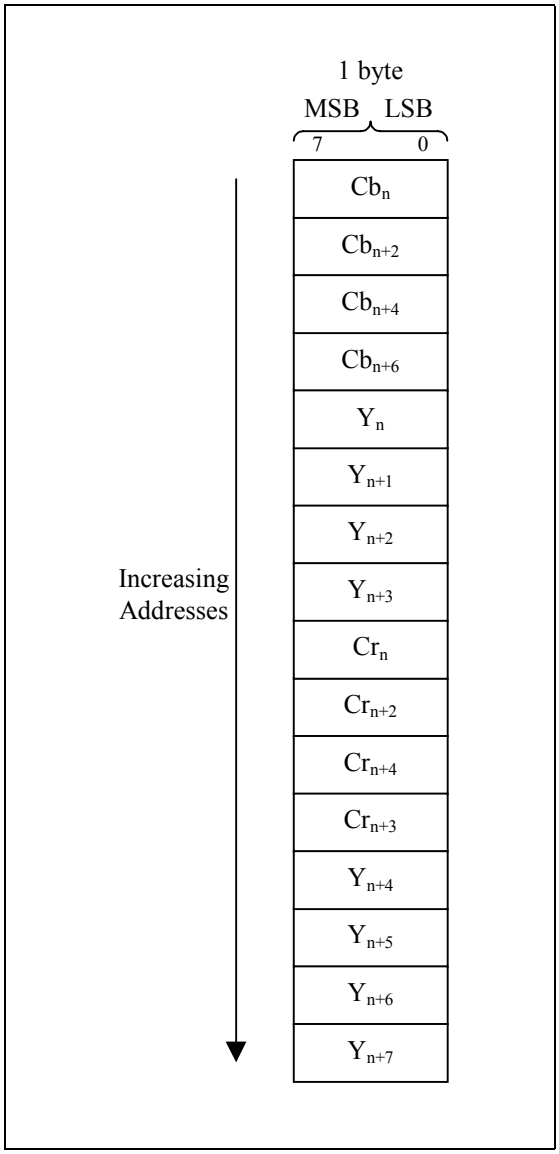


Structure of one 10-bit pixel group accessed as 32-bit words



Structure of one 8-bit pixel group accessed as 32-bit words

Note the byte ordering within the word is backwards. This is due to endianess issues. Since a majority of algorithms access 8 bit data as bytes rather than words, the order was modified to allow the order represented as bytes to be increasing as shown in the next figure.



Structure of one 8-bit pixel group accessed as 8-bit bytes

Since the number of lines per field and number of pixels per line is dependent solely on the SAV and EAV sequences, the input FIFO can handle any arbitrary sized video frame. It is the responsibility of the software to know these values.

The 4:2:2 YCbCr format decimates the color samples to half the rate of the luminance samples. Therefore, pixels in the format are always represented as pairs with one Cb and one Cr sample being shared by two Y samples.

Processor Action

After the video input FIFO interrupt is signaled, the processor reads the data from the most recent field, does whatever work is necessary on it, then writes it to the output video buffer for display (generally the previous frame is written).

Since data accesses are, by nature, required by the processor at the time of execution of the instruction, they have a high priority on the external bus. Data accesses are also, by definition, random on a generalized machine. These two aspects of data accesses can cause considerable overhead on the external memory bus thus reducing the available bandwidth to the system.

To avoid this overhead, the processor reads one line at a time from the external video input buffer and stores it in internal RAM via a DMA channel. Once the color space conversion has been completed, the processor then puts the data into the external video output buffer again via a DMA channel. This allows for accesses to the SDRAM in a highly sequential format which minimizes overhead of activate and deactivate cycles on the bus.

Video Output Buffer

The video output buffer is a continuous region of memory in the SDRAM capable of holding one entire frame of video.

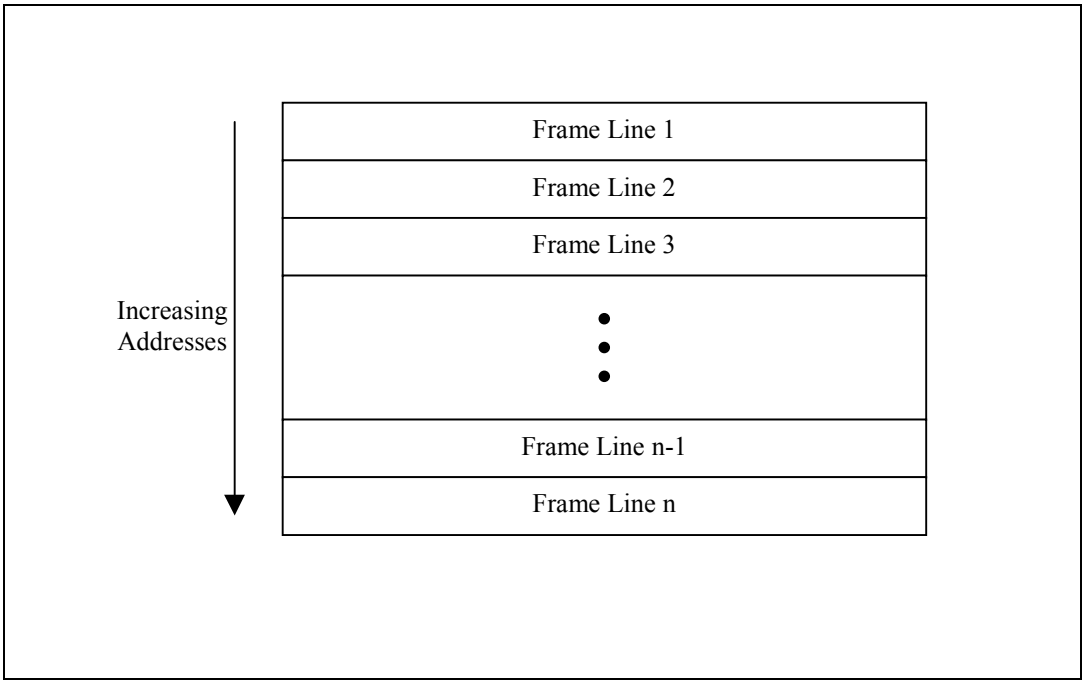
Each line of video data is stored in increasing addresses in memory.

The output video buffer is stored in a progressive format thus only one field is present.

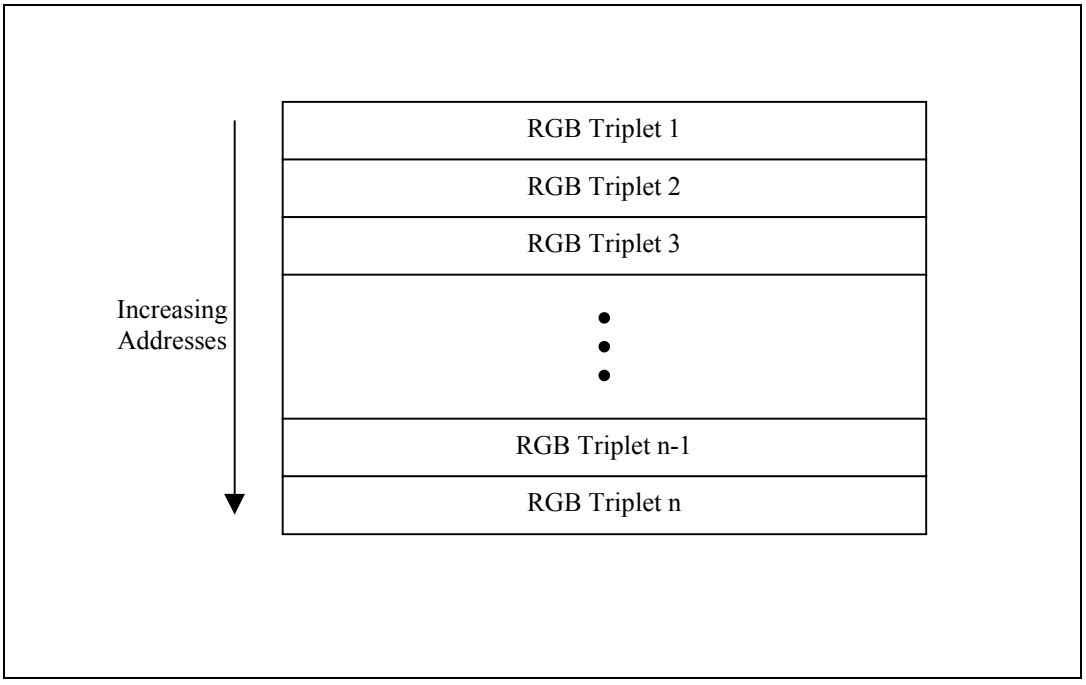
There are no additional data beyond active pixel RGB triplets required in the output buffer. Thus the data is a direct representation of the video displayed.

The video frame size is determined by constant values defined within the FPGA code. Changing to other frame sizes requires recompilation of the FPGA code. See the section on the FPGA register for more detail.

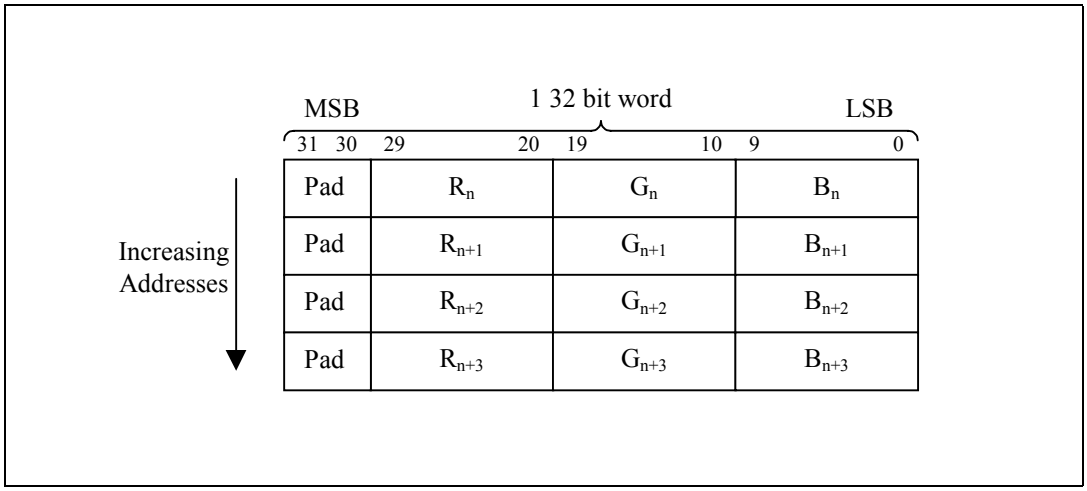
The various structures of the video output buffer are shown in the following diagrams.



Frame Structure within Video Output Buffer

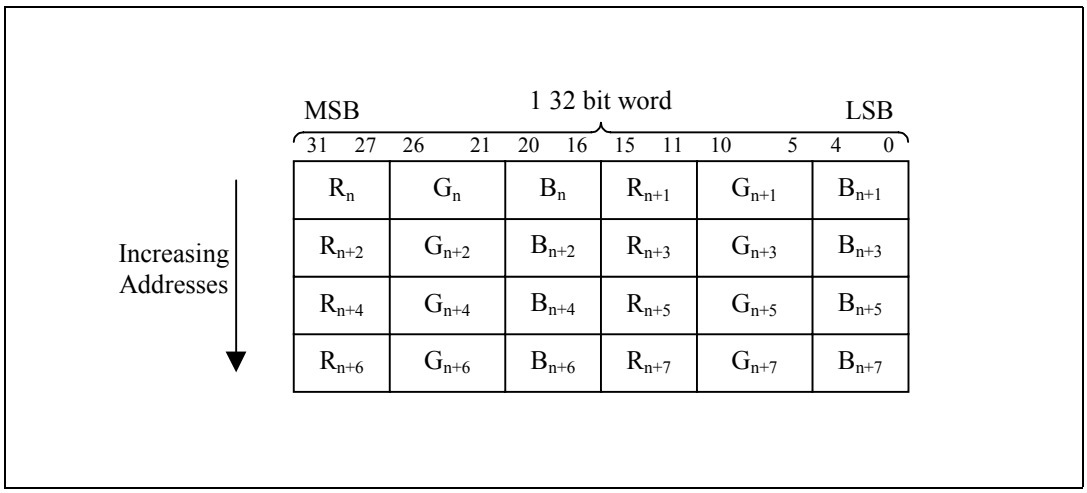


Line Structure within Video Output Buffer



Structure of 10-bit RGB triplets accessed as 32-bit words

Note: 8 bit formats would use the 10-bit format by shifting the 8-bit data up 2 bits and padding the least significant bits of each field.



Structure of 5:6:5 bit RGB triplets accessed as 32-bit words

Note the order is swapped once again for endianess issues of the 16-bit representations.

The processor does the interleaving of the fields as it writes the output lines to the buffer. This requires the output buffer to be somewhat synchronized with the output FIFO as glitches will occur in the video if the processor and FIFO addresses cross during display updating.

Video Output FIFO

The video output FIFO in the FPGA manages the difference between the synchronous video data rate and the asynchronous access rates of the processor.

RGB output video data is stored in a 32-bit by 256-word FIFO. Once 128 words have been read from the FIFO, the processor bus is requested by the FPGA. After the processor relinquishes control of the memory bus, 128 words of data are burst read from the video output buffer in the SDRAM.

Always bursting 128 words guarantees that the FIFO will always make accesses within a single page of the SDRAM thus minimizing overhead on the bus.

After a full frame of video has been output, the FIFO will contain extraneous data from beyond the end of the video output buffer.

Once a full frame has been output, flushing reads are implemented until the FIFO is on a 128-word boundary. The buffer pointer is then reset to point to the beginning of the video output buffer and 256 flushing reads of the FIFO are made to remove extraneous data from the previous frame and fill the FIFO with the beginning of the current frame.

After the flushing reads are complete, the FIFO is ready to output the new frame of video data.

FPGA Video Output Translation

At the beginning of a frame, each RGB triplet is read from the FIFO and separated into its component video values. Each component value is then presented to the Video DAC for display.

In between each output video frame, appropriate synchronization signals are generated depending on the type of output video being generated.

RGB Component Video Output

RGB component video output requires synchronization signals to be embedded within the video signals themselves. These blanking and synchronization signals are generated by toggling appropriate pins on the Video DAC. See the THS8133 specification sheet for details.

VGA Component Video Output

VGA output requires separate horizontal and vertical synchronization signals. These signals are generated using the same timing as those for embedded signals in RGB component video. The video signals, however, do not have synchronization information embedded in them as the VGA monitor does not recognize values below black but instead interprets this as video information.

Further, there is a single 100% white pixel generated at the beginning of every horizontal synchronization signal (including during vertical blanking and synchronization) to aid the VGA monitors Variable Gain Amplifier to track 0% and 100% signal levels. This will show up as a

white line on the right side of the monitor if the monitor is adjusted in a manner in which it can be seen.

Stereo Audio

The platform includes a stereo audio codec which is managed entirely by a timer and a DMA engine within the C6204. Audio signals correlated to the input video signal are digitized and stored in a circular buffer. The audio data is then routed back to the audio codec after being appropriately delayed to correlate with the output video signal.

Since the buffer is in memory accessible by the processor, there is full capability for the DSP to do audio processing as well as video processing on a complete set of audio/video signals.

Compatibility across Processors

The design goal was to make the interface applicable to as many members of the C62xx, C64xx, and C67xx families of processors as possible. To this end, the method of attachment was decided to be the memory interface.

There are other interfaces available to the designer depending on which processor from the C6xxx family is chosen but only the memory interface is reasonably common across all processors of the family.

For example, the C6204 processor used on this design has an extra external memory bus capable of interfacing with synchronous FIFO's. It is entirely conceivable to implement the FPGA in a manner which would mimic a synchronous FIFO, thus freeing up the memory interface entirely for code. However, this interface is not available on all members of the C6xxx family and for this reason was not chosen as the attachment point.

Members of the C64xx processor family contain more elaborate memory interfaces than those of the C67xx and C62xx families. These differences are significant but should not burden the designer who wishes to use them as the FPGA talks to the SDRAM memory directly through proper bus arbitration mechanisms.

Having the FPGA arbitrate the bus directly significantly lessens the burden of hardware compatibility between processors and this is why all members of the C6xxx processor family are available for implementation using this video interface scheme.

FPGA Implementation

The most important aspect of the platform, the video interface implemented in the FPGA, is compatible across the C62xx, C64xx, and C67xx families. To this end, only the EMIF (External Memory Interface) was used as it is the only (relatively) common interface to all processors in the C6xxx family.

The C6204 processor has one of the most limiting EMIF's and thus defines the minimum hardware requirements necessary for implementation. These limits are listed here.

1. The SDRAM must operate with a read latency of 2 clock cycles
2. The SDRAM mode is limited to burst lengths of 1

Upon initialization of the SDRAM, the C6204 writes a value of 0x30 into the mode register of the SDRAM. The FPGA code assumes this value in the mode register as well.

Since the C6204 writes only this value and no others, some optimizations of the memory bus on the circuit board were made to simplify layout issues. These optimizations can be seen on the Memory page of the schematic as pin swaps of data, address, bank, and DMQ pins. This

swapping maintains the integrity of the 0x30 write to the mode register as well as appropriate data and address transformations for row, column, and byte accesses.

These optimizations were for the sole purpose of layout simplification and are not necessary for proper operation of the system. ***Note that this optimization is not compatible across all C6xxx processors, as other members of the family can write different values into the mode register.***

Some processors in the C6xxx family are capable of 133MHz bus rates. The FPGA chosen is the slowest speed grade in its family (this helps minimize cost). Currently the FPGA code does not compile to a speed fast enough to operate at this data rate using this part. Test compilations targeting a faster grade FPGA did accomplish this timing requirement but were not tested on a circuit board.

The FPGA does not attempt to refresh the SDRAM memories so the processor refresh must be enabled or data will be lost.

Some processors in the C6xxx family have 64-bit and/or 16-bit data buses for the external memory interface. For those with a 64-bit bus, the FPGA should be capable of hooking directly to one half of the bus. This will of course not optimize bus accesses but will require no modifications to the FPGA.

If a 64- or 16-bit bus architecture is required, the FPGA code will require significant modification.

A 16-bit bus should be capable of transferring video at full rate on a 100MHz bus with around 65% utilization (according to measurements on the 32 bit bus). Thus a 16-bit implementation would not be out of the question for 8-bit data, but doubtful for 10-bit data.

Since the output of the video is progressive, the output video is a 50Hz/60Hz frame rate for a 25Hz/30Hz frame input video rate.

Processor Speed

Processors slower than the 200 MHz C6204 used on the platform may be limited in terms of the video bandwidth they can support. So, for instance, it may not be possible to support large display sizes without dropping frames.

DMA

One area where it was not possible to maintain compatibility across all processors was the DMA. Some members of the C6xxx family have an EDMA (enhanced DMA) peripheral block instead of a DMA block. Selecting one of these processors will require fairly significant re-writes of some sections of the code. The DMA is used for three functions on the Video Interface platform: UART interfacing, audio buffering, and video transfers. The related files which will require modification are: periph.c, audio.c, and video.c, as well as the configuration (.cdb) file. The

basic features in the DMA should be completely reproducible using an EDMA peripheral, so while some translation will be required, the functionality should be re-creatable without significant work. Reference 5, which was used as the foundation of the UART implemented on the platform, has example configurations for the EDMA which may prove useful. No attempt has been made to replicate the other functions (audio and video) using the EDMA.

Memory Configuration

Another area that may need addressing, depending on processor choice, is the memory configuration. To minimize external bus traffic, the program and data are kept internal to the DSP on the platform, which uses a good portion of both the 512 Kbits allocated to internal program space and the 512 Kbits of internal data space on the C6204. On processors with less internal memory, external program/data memory may be required and caching may need to be employed. For more sophisticated video algorithms, the internal memory limitations may also require the use of cached program and/or data. The subsequent increase of bus traffic may have an impact on the capability of the platform to perform real-time video transfers without dropping frames.

Features to Assist Demonstration and Evaluation

Some features have been added to the demonstration platform to assist in the evaluation of the video interface function. These features are described here.

The C6204 contains a McBSP interface that can be easily tied to a PC's UART through a level translator (e.g., the SN75LV4737A). A command interface has been developed that allows the video function to be controlled from a program on the host PC. Using this program, the video source and audio volume can be controlled.

Several LED's have been placed on the board to indicate when power is good on the unit, when UART communication is taking place, and when the processor is running normally.

Performance Measurements

The following sections describe various performance metrics associated with the Low-Cost Video Interface demonstration platform.

The information is intended to aid a designer in determining applicability of how components of this system may be applied to their own design.

Processor Utilization

The DSP code implemented on the low-cost interface platform utilizes approximately 36% of the available processing power when video is streaming through the system. If video is halted the background BIOS tasks take about 20%.

The video's 16% can be explained almost entirely by the conversion function, ycbcr422pl_to_rgb565(), which requires (46 + numPixels*3) clock cycles each time it is called (once for each line in this case). For a 720X480 display this requires 480*(46 + 720*3), or 1,058,880 clock cycles. With the 200 MHz TMS320C6204, this results in 5.3 ms per frame. With a 30 Hz frame rate, each frame takes 33.3 ms. 5.3 / 33.3 = 15.9%

The remaining 20% for the BIOS background task is primarily due to the high rate at which the BIOS loop is run. A 50 μs time period was selected for the BIOS timer primarily for the I^2C serial interface that it clocks. Because Timer1 is still available, one could separate the I^2C clock from the BIOS timer, significantly reduce the BIOS rate, and free up additional MIPS if needed.

Memory Utilization

There are three sections of memory accessible to the DSP on the video platform: internal program RAM, internal data RAM, and external SDRAM.

The internal RAM blocks are 0x10000 bytes each. Because the machine word size is 32-bits, this equates to 16384 instructions in the program space and 16384 words in the data space. The external SDRAM was originally sized to handle multiple frames of a large display; 32 MB are available in the two SDRAMs (most is unused).

The C6000 family allows the processor to operate in a cache mode which would allow executable code to be stored in the SDRAM if desired. The video platform, however, operates without cache and keeps the executable code in internal program space. 78% of the available program space is used by the platform in this configuration.

The internal data space allocated on the platform takes up 84% of what is available. By far the largest block of internal data is the audio buffer used to delay the audio track to sync with the video. One quarter of the internal data RAM is used for this purpose. Putting the audio buffer in SDRAM would free up internal RAM at the expense of more EMIF bus traffic.

The SDRAM contains an input buffer, an output buffer, and an image of both the DSP code and the FPGA which are used for programming EEPROM. Together, these use 25% of the SDRAM. Additionally, the video buffers are oversized; with an NTSC ITU-R BT.601 signal only 16% of the allocated video buffers are used (a 1600 X 1200 display would use almost 92%).

External Bus Utilization

The external memory bus bandwidth must be managed by the designer in order to ensure video data throughput is maintained.

Calculation of bus bandwidth is some what heuristic in nature due to required idle cycles between certain bus commands like precharge, activate, deactivate, and arbitration.

The processor refresh for SDRAM occupying the full memory range available to a chip enable signal runs approximately 1%. Smaller sized memories would require fewer refresh cycles per unit of time, reducing this overhead proportionately.

One frame of video requires 5 accesses of the memory to complete. First, the FPGA stores the decoded video data into memory; second, the DSP reads the data out of memory via a DMA channel; third, the DSP writes modified video data out to memory via another DMA channel; fourth, the FPGA reads the data from memory and routes it to the video DAC.

The fourth access actually transfers each pixel twice, as the output frame rate is twice that of the input frame rate. Writing each pixel twice is an artifact of displaying the output video buffer after each field is updated rather than only after a complete frame is updated. This method was chosen as a means to minimize temporal distortions.

It is because the fourth access makes two accesses that there are 5 equivalent accesses of memory for each complete frame.

A rough estimate of the number of accesses per frame can be given as:

$$A_f = \left(2\frac{L_f (P_l + 2) S_p}{S_w} + 3 L_f P_l W_{RGB} + 768 \right) \left[\frac{words}{frame} \right]$$

where A_f is the number of word accesses per frame, L_f is the number of lines per frame, P_l is the number of pixels per line, S_p is the average number of samples per YCbCr pixel, S_w is the number of YCbCr samples per 32 bit word, and W_{RGB} is the average number words per RGB pixel.

The number 2 represents the 2 accesses per word for input video, the 3 represents the 3 access per word for output video (one by the DSP and the other 2 by the double rate output), the 2 added to the number of pixels per line represents the additional 2 pixels per line which the EAV

sequence occupies on input, and the 768 represents the number of additional accesses made during buffer flushing for both input and output video buffers.

For 8-bit PAL/SECAM ITU input mode and 5:6:5 RGB output mode this results in:

$$A_f = 2\frac{L_f(P_l+2)S_p}{S_w} + 3L_f P_l W_{RGB} + 768 = 2\frac{576(720+2)2}{4} + 3\cdot 576\cdot 720\frac{1}{2} + 768 = 1038720\left[\frac{words}{frame}\right]$$

For 8 bit NTSC ITU input mode and 5:6:5 RGB output mode this results in:

$$A_f = 2\frac{L_f(P_l+2)S_p}{S_w} + 3L_f P_l W_{RGB} + 768 = 2\frac{507(720+2)2}{4} + 3\cdot 507\cdot 720\frac{1}{2} + 768 = 914382\left[\frac{words}{frame}\right]$$

The reason the lines per frame parameter, L_f, is 507 and not 480 is due to the TVP5145 decoder decoding more than the standard number of lines. There are 254 lines decoded for the odd field and 253 decoded in the even field.

From the above values, the amount of time on the bus can be estimated as:

$$T_f = A_f(A_t + B_o)\left[\frac{time}{frame}\right]$$

where T_f is the time per frame spent accessing the bus, A_t is the time for a single 32 bit word access, and B_o is the bus overhead time per burst access (this includes precharge, activate, deactivate, and bus arbitration).

For 8-bit PAL/SECAM ITU input mode and 5:6:5 RGB output mode this results in:

$$T_f = A_f(A_t + B_o) = 1038720\left(\frac{1}{100000000} + \frac{18}{128\cdot 100000000}\right) \approx 11.85\left[\frac{\text{ms}}{frame}\right]$$

For 8-bit NTSC ITU input mode and 5:6:5 RGB output mode this results in:

$$T_f = A_f(A_t + B_o) = 914382\left(\frac{1}{100000000} + \frac{18}{128\cdot 100000000}\right) \approx 10.43\left[\frac{\text{ms}}{frame}\right]$$

The bus overhead is an estimated average derived from the FPGA design implementation, SDRAM mode register settings, and refresh cycles.

The bandwidth usage of the bus can be calculated as:

$$\%BW_{bus} = 100\frac{T_f}{T_v}[BandWidth\%]$$

where T_v is the time period of a single video frame

This yields for 8-bit PAL/SECAM ITU input mode and 5:6:5 RGB output mode:

$$\%BW_{bus} = 100\frac{11.85}{40} = 29.6[BandWidth\%]$$

and for 8-bit NTSC ITU input mode and 5:6:5 RGB output mode:

$$\%BW_{bus} = 100\frac{10.43}{33.33} = 31.3[BandWidth\%]$$

The measured values were 30.6% for PAL and 30.5% for NTSC. These numbers correspond well with estimated values in these modes.

For 10-bit modes (the DSP software does not support these modes but the FPGA does) the calculations would be:

10-bit PAL/SECAM ITU input mode and 10:10:10 RGB output mode:

$$A_f = 2\frac{L_f(P_l+2)S_p}{S_w} + 3L_fP_lW_{RGB} + 768 = 2\frac{576(720+2)2}{3} + 3\cdot 576\cdot 720\cdot 1 + 768 = 1799424\left[\frac{words}{frame}\right]$$

$$T_f = A_f(A_t + B_o) = 1799424\left(\frac{1}{100000000} + \frac{18}{128\cdot 100000000}\right) \approx 20.52\left[\frac{\text{ms}}{frame}\right]$$

$$\%BW_{bus} = 100\frac{20.52}{40} = 51.3[BandWidth\%]$$

10-bit NTSC ITU input mode and 10:10:10 RGB output mode:

$$A_f = 2\frac{L_f(P_l+2)S_p}{S_w} + 3L_fP_lW_{RGB} + 768 = 2\frac{507(720+2)2}{3} + 3\cdot 507\cdot 720\cdot 1 + 768 = 1583960\left[\frac{words}{frame}\right]$$

$$T_f = A_f(A_t + B_o) = 1583960\left(\frac{1}{100000000} + \frac{18}{128\cdot 100000000}\right) \approx 18.01\left[\frac{\text{ms}}{frame}\right]$$

$$\%BW_{bus} = 100\frac{18.01}{33.33} = 54.0[BandWidth\%]$$

These results have not been measured and so are only estimates of expected bandwidth usage rates in these modes assuming software to support it.

Design Extensions

This portion of the paper considers how the design might be modified and extended to use less memory and integrate the video library routines available from TI.

Memory Reduction

The current implementation has two 4-Meg by 32-bit SDRAM memory components for a total of 32MB of external data space comprising all of the chip enable spaces 2 and 3. This was done primarily to show the system would work in other than just a minimal configuration.

Recognizing the memory components can be significant contributors to the cost of the entire system, one may wish to reduce the amount of memory on a product implementation as a cost reduction measure.

The current incarnation of the system only requires 2-Meg by 32-bit of external data space for the video buffers. This allows storage of single frames of video storage for frames as large as 1920x1080 (standard video format) or 1600x1200 (computer display). If the target implementation will never use frames of this size, further reduction in memory requirements can be achieved.

Reduction of the video frames below the current 2-Meg by 32-bit buffer size will require changing some constants in the FPGA code and recompiling it. This change is minor and documented in the FPGA code itself.

Software Integration with TI-supplied C6x Video Routines

TI supplies code to implement various functions related to video processing. For example, the conversion routine used in the platform, "ycbcr422pl_to_rgb565," is from this source. The video product being designed will determine how these functions are used. With a theoretical 1600 MIPS, the C6204 can perform the matrix YCbCr to RGB conversion on a pixel by pixel basis in real time. In addition to MIPS, memory may be a constraint for some video functions. For example, edge detection may require multiple complete frames to be stored for comparison. This would likely require external memory, and probably could not be done without dropping frames.

On the video platform the DSP's interface to the FPGA and the video data is through shared external SDRAM and an interrupt from the FPGA (tied to INT4). Upon completion of a video field (one half of an interlaced frame), the FPGA sends and interrupt to the DSP signaling the event. This is the cue for the DSP to begin processing. There are three buffers in the SDRAM defined for video data: two for the odd and even input fields, and one for a progressive RGB output. These are declared inputFrame[2] and outputFrame in the DSP code. They are located at 0x0200:0000 and 0x0210:0000 respectively.

For algorithms operating on entire frames the DSP probably needs to either halt the FPGA from overwriting these buffers or else make a shadow copy elsewhere in SDRAM for post-processing. If the required field processing time is less than one-half of the video frame rate, then the processing can be done in real time without missing frames, as is done in the conversion routine performed in the platform. If the required processing time is greater than the field time, the DSP will not be able to complete its work without missing frames.

Options for the Audio Sample Rate

The Low-Cost Video Interface platform accepts an audio channel, which it delays by a programmable number of samples in order to allow syncing of the audio and video channels. There are several reasonable options for sample rates on the audio stream. Typical audio standards are multiples of 48 kHz or 44.1 kHz. In reality, on the platform, the audio input and output are analog signals and the digitized data is not modified which makes the choice of sample rates somewhat arbitrary, as long as it is high enough to maintain adequate fidelity. If some signal processing was being performed on the audio data, an exact standard would likely be more critical. There are a couple reasonable options in this regard.

On the platform the ‘AIC23 is configured to run in “USB mode” which allows various audio sample rates related to both the 44.1kHz CD standard, and the 48 kHz standard. To get exact standard frequencies, the USB mode requires a 12 MHz input clock. On the platform they are a little off because the clock driving the AIC23 comes from a divided clock out of the DSP at 200 MHz / 16 = 12.5 MHz. This doesn't make any discernable difference in audio quality, but does differ from the standard. Exact frequencies could be generated by changing the DSP's clock input to be 48 MHz.

Conclusions

The Low-Cost Video Interface demonstration platform clearly demonstrates the capability for interfacing full frame rate video and audio to Texas Instruments' line of C6xxx processor family.

Further, the performance measurements indicate there is ample room for additional computational bandwidth both within the processor and on the memory bus. This opens the door for many video applications areas.

The ability to also incorporate stereo audio through the processor allows options such as audio source tracking and triggering to also be available to the system designer.

References

1. K. Jack, Video Demystified, 3rd edition, Elsevier Science, 2001.
2. Texas Instruments, "TVP5145PFP NTSC/PAL/SECAM/Component Digital Video Decoder with Macrovision™ Detection," SLES029A, May 2002.
3. Texas Instruments, "TVP5150 Ultra Low Power NTSC/PAL Video Decoder," March 2003.
4. Texas Instruments, "TMS320C62x Image/Video Processing Library Programmer's Reference," SPRU400A, April 2002.
5. Texas Instruments, "TMS320C6000 McBSP: UART Application Report," SPRA633A, August 2001.
6. Altera, "ACEX 1K Programmable Logic Device Family Data Sheet," September 2001.
7. Texas Instruments, "TMS320C6000 EMIF-to-External SDRAM Interface Application Report," SPRA433B, December 2001.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265