Control Two ADV601's with a Single ADSP 2185

1. Introduction

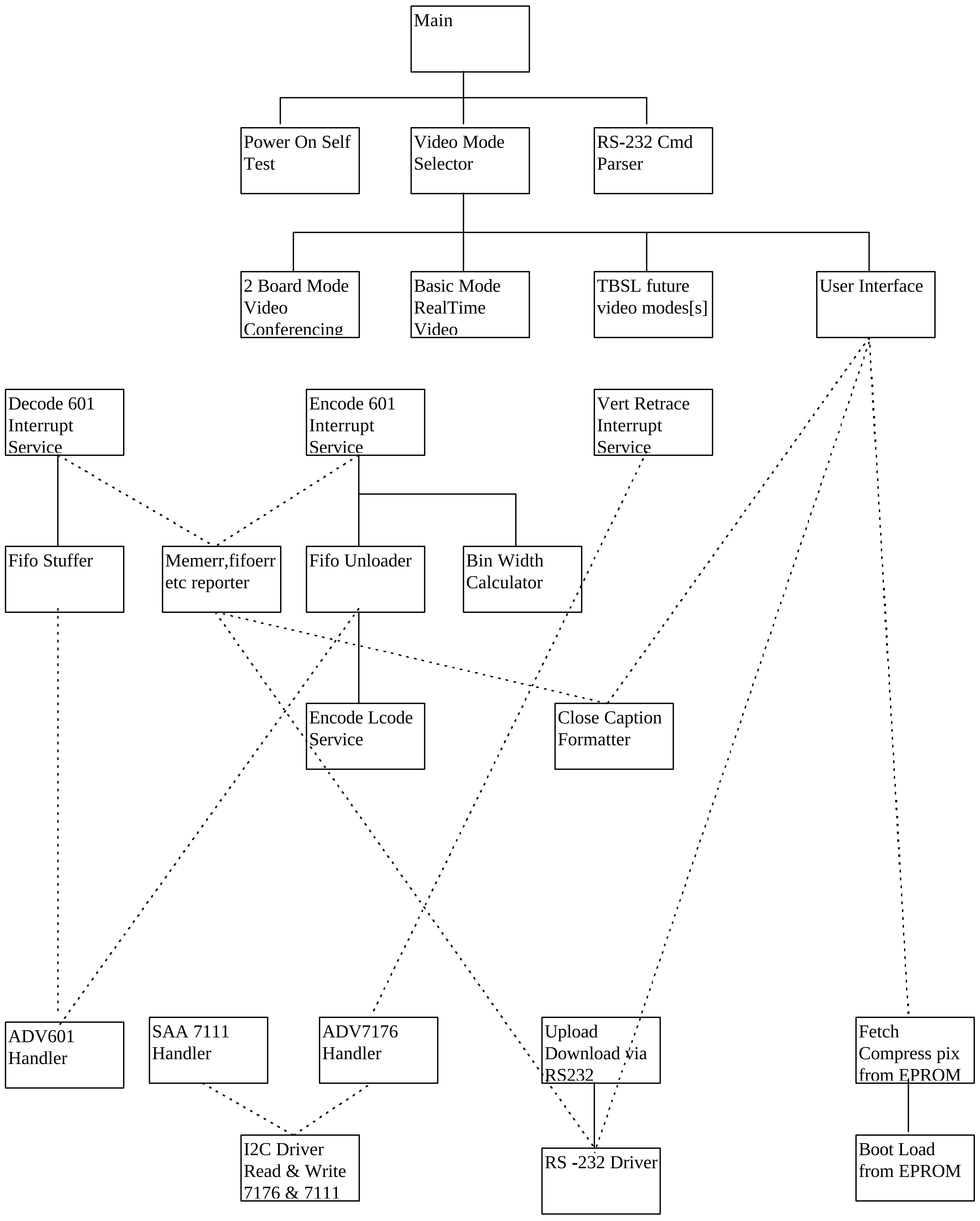
This application note describes the DSP software written to operation the Videopipe ADV601LC evaluation board. After reading this application note (and studying the source code) you will be able to

1. Modify the program to enchance performance or fix bugs.
2. Reuse software modules in other ADV601 designs
3. Use the existing program as a software test bed to exercise your own ADV601 code.
4. Give new functionality to the Videopipe board such as video conferencing or surveillance camera video compression.
5. Use the Videopipe board as a software testbed for new ADV601 applications.

This application note was written shortly after the program was completed and the basic theory of operation was still clear in the mind of the program's author. As time passes expect the code to be modified for all the usual reasons, and this application note probably will not be re written. In other words, this note decribes the program as of early fall 1997. Future versions of the program code may differ from what is described herein.

The organization of this document follows a popular outline for a Software Requirements Specification. It seemed to be a natural order for a "how the code works" article. You can consider it to be a rewrite of the specification after the job is done. This documents tells you how the code really works after the debug sessions, whereas a specification is a design goal laying out how you hope things will work before you actually get them working.

Figure 1.1 shows the module hierarchy of the Video Pipe Software. Each module is separately assembled and resides in its own source file. Solid lines represent flow of control down wards. Dashed lines show calls to low level driver modules which can be called from more than one higher level module. Drivers called from both foreground (interrupt) and background (main) must guard against re entrancy problems. In all cases, modules will reenter successfully if they preserve the state of the Data Address Generator (DAG) registers and the interrupt mask register, and don't modify local data memory variables. The 2185 has a wealth of registers permitting computations entirely in registers.


Main
Power On Self Test
Video Mode Selector
RS-232 Cmd Parser
2 Board Mode Video Conferencing
Basic Mode RealTime Video
TBSL future video modes[s]
User Interface
Decode 601 Interrupt Service
Encode 601 Interrupt Service
Vert Retrace Interrupt Service
Fifo Stuffer
Memerr,fifoerr etc reporter
Fifo Unloader
Bin Width Calculator
Encode Lcode Service
Close Caption Formatter Service
ADV601 Handler
SAA 7111 Handler
ADV7176 Handler
Upload Download via RS232
Fetch Compress pix from EPROM
I2C Driver Read & Write 7176 & 7111
RS -232 Driver
Boot Load from EPROM

2. Documents

Documents are listed here as a bibliography.

2.1 Analog Devices ADV7176 Data Sheet Rev 0 1996

2.2 Analog Devices ADV601 Data Sheet Rev 0 1997

2.3 Phillips SAA7111 Data Sheet

2.4 Phillips Desk Top Video Data book IC22 1995

2.5 Analog Devices ADSP-2185 Data Sheet Rev 0 1997

2.6 ADSP-2100 Family User's Manual 3rd Edition 9/95

2.7 ADSP-2100 Family Assembler Tools and Simulator Manual 2nd Edition 11/94

3. Platform

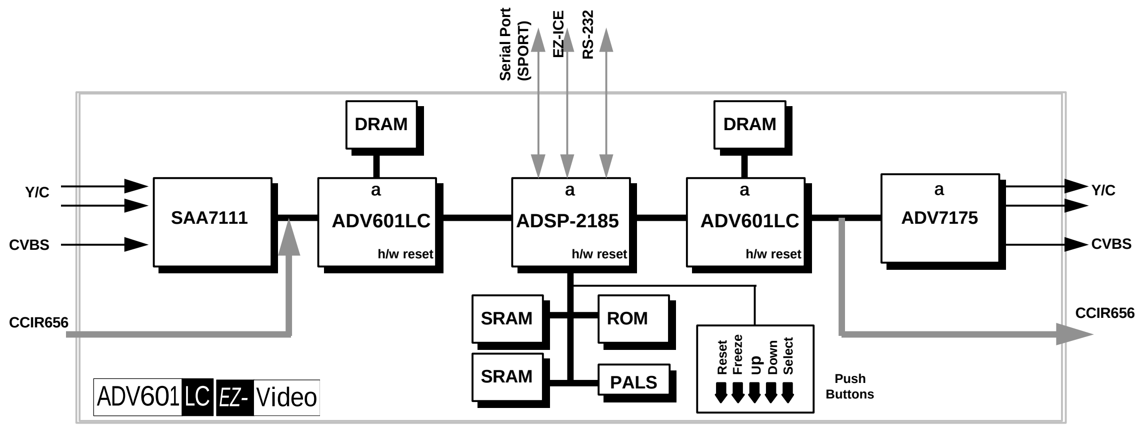
3.1 Software and Tools

The Videopipe board was designed to look like the EZKIT lite ADSP 2181 development board so that the Ezkit software tool set would work. The entire project was done in DSP assembler without the use the C compiler. You can obtain necessary tools by purchasing the EZ-Kit Lite board. The Videopipe board supports the EZ-ICE in circuit emulator which permits code loading without burning proms, breakpoints, examination of registers and memory, as well as single step tracing. EZ-ICE was invaluable during the program development phase.

3.2 Hardware

As you can see from the block diagram, the Videopipe board is a pair of ADV601's connected "back-to-back". Video is compressed in the encode ADV601. The compressed video is buffered into the Video Ram and then decompressed and played back by the decode ADV601. The main task of the DSP software is to keep the compressed video moving from the encode ADV601 into the Video Ram and from the Video Ram out to the Decode ADV601. The DSP serves as a sophisticated DMA controller for the compressed video.

3.3 Hardware Block Diagram



3.3.1 Video Ram

Video Ram is interfaced to the DSP as External Data memory starting at address 0 and running up to 1FFF hex (8K). The DSP accesses the video memory as 16 bit words. Two memory accesses are required to transfer a 32bit long word to the ADV601 compressed data registers (which are 32 bits wide). The Videopipe board is equipped with 256 Kbytes (128 K words) of Video ram. The DSP has only 13 address pins, limiting it's address reach to 8K. The additional address bits (A13 thru A15 and the two ram chip enables) are taken from page bits in the card control register. To cross a page boundary, the software must rewrite the page bits in the card control register. The software aligns data structures (headers and compressed video) in video ram to minimize the amount of page turning required. It also takes care to call routine VRAMPAGE to set the page bits prior to accessing the video ram.

The video ram is interfaced directly to the ADV601 chips. The memory can transfer between the encode ADV601, or the decode ADV601. The type of transfer is controlled by the transfer mode bits in the card control register. There are four ADV601 to Video Ram transfer modes , plus a DSP to Video Ram a DSP to encode ADV601 and a DSP to decode ADV601 mode, for a total of 7 different "memory" transfer modes.

Transfer Mode Name	Transfer Mode Code	Required Wait States	Address Range
E601 to Video Ram	0	0	0-8K
Video Ram to D601	1	0	0-8K
DSP to/from Video Ram	2	0	0-8K
DSP to/from E601	3	4	Don't Care
DSP to/from D601	4	4	Don't Care
Video Ram to E601	5	0	0-8K
D601 to Video Ram	6	0	0-8K

.

Prior to accessing the Video Rams or the ADV601's from the DSP it is necessary to set the desired mode code into the card control register. It is also necessary to execute the DSP instruction DMOVLAY = 2 to cause DSP data memory accesses in the lower 8K to access the external Video Ram instead of the 2185 on chip data memory. The compressed data registers and the Video ram as so fast that no wait states are required for proper hardware operation. Access to the other ADV601 registers (register data, register address and interrupt mask and status require 4 wait states for reliable operation. The ADSP 2185 lacks a "wait request pin" so it is necessary to program the 2185 on chip Wait State Control Register for the necessary number of wait states. The Videopipe program programs 4 wait states (defined by symbol CRUISE_SPEED) into the Wait State Control Register.. There are just two places in the program (reading the encode ADV601 and writing the decode ADV601) where the wait state generator is re programmed for zero wait states (defined by symbol WARP_SPEED).

The DSP moves data between the ADV601's and video memory with READ instructions, such as AX0 = DM(I1,M1), never write instructions. The transfer mode controls the direction of data flow. Either the ADV601 or the Video Ram chip is enabled to drive the data bus, and the DSP drives the address bus to control the location of the transfer. DSP write instructions will cause the DSP to drive the data bus which would contend (fight) with the ADV601 or Video Ram drivers. The data moved also appears in the DSP's register (in this case AX0) so a program can see the video data as it goes thru with out the bother of resetting the transfer mode to read it back from video ram.

3.3.2 Interrupts

3.3.2.1 Decode 601 HIRQ on

3.3.2.2 7176 Vertical sync on

3.3.2.3 Encode 601 HIRQ on

3.3.2.4 Others?

3.3.3 Control Ports

3.3.3.1 Video Ram page register

3.3.3.2 Control Port ?

3.3.4 Memory Map

4. External Interface

4.1 Auto Sense/Switch video input.

The Videopipe supports two different video input connectors (S-VHS and Composite) and can handle both NTSC (North American) and PAL (European) video formats. After power up the software looks into the SAA7111 video decoder chip and automatically sets the board to match the applied video. If no video is present at the input, the board displays a "splash screen" (a still image of a pretty girl).

4.2 The PushButtons

Button	**Button Description**

Select Button (S3)	Selects which parameters are controlled by the **up button** and the **down button.** *Press Select 0 time(s), Up/Down controls Bit Rate in Mbps* *Press Select 1 time(s), Up/Down controls Field/sec* *Press Select 2 time(s), Up/Down controls Image Resolution* *Press Select 3 time(s), Up/Down controls Image Cropping Region* *Press Select 4 time(s), Up/Down controls Special Effect* Press Select 5 time(s), Up/Down *controls Bit Rate in Mbps etc....*
Up Button (S1)	Moves Up list of parameters (see lists to right)
Down Button (S2)	Moves Down list of parameters (see lists to right)
Freeze Button (S4)	Freezes current Image on the TV monitor for close visual analysis
Reset Button (S5)	Hardware and Software Reset

4.3 Function Parameters

Select Button	**Up Button Down Button**	**Reset Value**
Select Button pushed 0 times to control **Bit Rate in Mbps**	• 44.2E+6 • 33.6E+6 • 19.7E+6 • 14.1E+6 • 9.7E+6 • 9.2E+6 • 8.7E+6 • 8.2E+6 • 7.7E+6 • 7.2E+6 • 6.7E+6 • 6.1E+6 • 4.1E+6 • 2.6E+6 • 2.0E+6 *All Values Mbps*	44.2E+6
Select Button pushed 1 time to control **Field/sec**	• 60/50 • 30/25 • 15/12.5 • 10 • 5 • 1 *All Values Fields/sec*	60/50
Select Button pushed 2 times to control **Image Resolution**	• CCIR601 • CIF • QCIF • QQCIF	CCIR601
Select Button pushed 3 times to **Image Cropping Region**	• 720 x 486/576 • 708 x 480 • 360 x 240/288 • 180 x 120/144 • Pix-in-Pix	720 x 486/576
Select Button pushed 4 times to control **Special Effect**	• None • Edge 4 • B&W • LEDs do Motion Detection	None

Select Button pushed 5, 10, ... times loops to the top of column		

5. EZ-Video Hardware

5.1

6. Timing

7. Input Processing

8. Processing

8.1 main (main.dsp)

8.1.1 Inputs

8.1.2 Outputs

8.1.3 Processing

Main is responsible for setting hardware registers with global impact such as ICNTL (programs interrupt hardware) and System Control Register (configures I/O pins), and IMASK. It then runs CPU_TEST (borrowed from the EZ-KIT Lite monitor) . CPU_TEST must come first since it checks that bits initialized by hardware reset are actually coming up as specified in the 2185 documentation. Then DAC_CBAR is called to put up the ADV7176 internal color bar display. If color bars appears in the TV monitor, the CPU is running code, the I2C bus is working, the 7111 is generating VCLK and the ADV7176 is running.
As a safety first measure, main initializes the Data Address Generator (DAG) registers to reasonable values. Subroutines using the DAGs should set all the registers they use, but just in case we set them here too.
SCAN_KBD is called once to sense "reset with any key down" which signals CARD_TEST to go into loop forever diagnostics mode. The Data Memory Wait register is programmed with enough wait states for dependable access to the ADV601 registers in external IO space. CARD_TEST then checks things on the Videopipe card, as opposed to things inside the 2185 CPU. The BANNER loop sends a cheerful greeting message out the RS232 serial port, and the USRVEC loop initializes the serial port program downloader borrowed from the EZ KIT Lite Monitor. Then the 7111 video decoder chip, the 7175 encoder chip and the close captioning software are started. VRAM_INIT sets up the video buffer headers and loads the splash screen into video memory from the PROM , just as if it had been captured by the ADV601. Then the playback start flag is set to cause the playback ADV601 to start up on the vertical retrace interrupt. Playback will show the splash screen until (and if) we start capturing input video.

MAIN's next task it to auto sense the input video source and type. Source can be analog or digital video. Analog video can be composite or S-video, (different input connectors) and NSTC or PAL (either connector can furnish either format). Main first checks for digital video. He tri states the 7111 data bus (the DIG_VID argument to ADC_INIT handles that) and waits for CAPTURE_ISR to bump REC_NFIELDS. If REC_NFIELDS starts counting up with the Video ADC tri stated, digital video from J12 must be feeding the capture ADV601. It cannot be analog video because the analog to digital converter is tri-stated (off). After enough passes thru CHECK_DIGITAL, MAIN knows there is no digital video present, so he starts calling SIGNAL_SEEK. Signal seek will try both analog input connectors, and automatically select the one with video one it, so the user can plug into either analog input and have the card display it. If video is detected, SIGNAL_SEEK reports the format, PAL or NTSC. MAIN then calls CAPTURE_GO to set the capture ADV601 to the detected video format, DAC_INIT to do the same thing to the Video decoder (7175) and sets PLAYBACK_START to re initialize the playback ADV601. The re initialzation is necessary if the input is PAL. MAIN starts every chip off set for NTSC. If SIGNAL_SEEK reports the video is actually PAL, every chip needs to be re initialized. If its NTSC, well, reinitialization never hurts any chip. If no input video of any kind is detected, then MAIN has to open the horizontal phase locked loop in the 7111 video decoder to stabilise the frequency of VCLK. This is necessary to get good color. MAIN also kills the RTCO color corrector signal from the 7111 decoder to the 7175 encoder and re initializes the the 7175 telling him that VCLK is a few hundred hertz off the correct frequency. DAC_INIT will make a small correction to the color burst frequency to compensate for the small error in VCLK.

The auto sense process is controlled by state variable VIDEO_SIGNAL. It runs once from start up. Once a final state (SPLASH_SCREEN, NSTC, PAL, or DIG_VID) is reached, the search process stops. The user must press the reset button to restart it. For instance, if the board is powered up with analog composite video connected, then unplugging the video source will make the output TV monitor go dark, which is the expected reaction. It will stay dark until the user plugs the video back in. To switch to another video source the user must press the reset button. State NO_VIDEO is the initial (searching) state. If after a search, no input video is found, then the state goes to SPLASH_SCREEN. Digital video is assumed to be NTSC since at this writing there is no way for the card to tell the difference between NTSC and PAL digital video. Later versions of the card or firmware may be more clever.

MAIN checks watch dog timers (PLAYBACK_DOG and CAPTURE_DOG). Should one of these grow large, an ADV601 has stopped and needs to be reinitialized. They are bumped by TIMER_TIC who runs off the internal real time clock interrupt, and zeroed by the ADV601 fifo service routines. MAIN only checks CAPTURE_DOG if input video has been detected. If there is no video, the ADV601 won't assert the FIFO_SERVICE_REQUEST interrupt and the watchdog will keep growing. MAIN has to pass control to CHKBUTTON (looks at the user buttons) TEXT_LEGEND (composes alphanumeric messages for the TV monitor and CCAPTION_BYTES_OUT (moves the message to the TV two bytes per frame).

TEXT_LEGEND is part of MAIN at this writing, but by rights it belongs in the CCAPTION module, and may get moved there some day. TEXT_LEGEND drives alphanumeric data onto the TV monitor screen using the Line 21 Close Captioning system on NTSC TV sets. This technique does not work on PAL TV receivers. The on screen text tells the user which parameter (compression ratio, field rate, or resolution) the UP and DOWN buttons are adjusting. Two lines , a title and a value are displayed. The TV monitor is set so the value can be rewritten without rewriting the title line (PAINTON mode). Closed Captions are written to the TV set 2 ASCII bytes per frame. Each two byte transmission requires a time consuming I2C message be sent to the 7175 video encoder just after vertical retrace on field 1. The I2c transmission is done in background so that ADV601 FIFO service request interrupts can "punch thru" the I2C driver to avoid loss of video. The ADV7175 transmits the two bytes on the first line out of vertical retrace (Line 21). If the DSP changes the two bytes at the same time the ADV7175 is transmitting them, the TV set will receive a scrambled transmission and display garbage. TEXT_LEGEND creates the byte string for the TV set and places it in buffer CLOSE_CAPTION and sets variable CAP_LEN to indicate message length. He then waits for TWO_BYTES_OUT to empty the buffer and decrement CAP_LEN to zero. When this occurs, TEXT_LEGEND knows the buffer is available to send the next message.

When CHKBUTTON senses the MODE button pressed, changing the mode, he passes a pointer to a text string to identify the mode. TEXT_LEGEND keeps the pointer in variable TITLE_STRING. After writing the title to the TV set, he zero's TITLE_STRING as a signal that the title need not be rewritten again. TEXT_LEGEND is a state machine driven by variable PINGPONG. (Originally he had only two states, hence the name) . The state machine goes thru states KICKIT (an initialize message) , TITLE (send title string) VARS (send variable string) POPPIT (needed only in POPON caption mode) and then IDLE.

8.2 self test (cputest.dsp, cardtest.dsp & ramtest.dsp)

8.2.1 Inputs

None

8.2.2 Outputs

selfTstRst	0 = Pass, Non-Zero = fail

8.2.3 Processing

The CPUTEST module is called first thing after power on reset. This code is borrowed from the EZKIT-Lite monitor and modified lightly. It starts off ("test1") by reading the various memory mapped control registers and checking that they contain the "hardware reset value". If the caller has modified any of the registers then CPUtest will fail. "Test2" walks various bit patterns thru all the DSP's internal registers. "Test3" checks the DO loop stack by nesting do loops four deep. "Test4" checks reads and writes to program memory. Array pm_user (4096 words) is allocated for test purposes only. Should program memory run short, this array could be reduced in size, or the test eliminated completely. This module used to check data memory, video ram and the audio codec in the EZKIT, but it no longer does so. Cputest checks only things inside the ADSP2185 chip itself. This particular diagnostic does not seem crucial to product success. Videopipe has never experienced a cputest failure. It was included in the PROM more as a card design verification than anything else.

Module CARDTEST is a general purpose "diagnostic runner". Array test_list contains the entrance addresses to each individual diagnostic. The main loop of CARDTEST, walks down test_list, running each diagnostic in the list and displaying PASS or FAIL via the RS-232 serial port after the diagnostic returns control to the main loop. More diagnostics can be added to test_list if symbol NTEST is edited to increase the size of test_list to make room for the additional diagnostics. CARDTEST can operate in "press_on_regardless" mode, "loop on error" mode, or "burn_in" mode. Videopipe is shipped in "loop-on-error" mode. Any failed diagnostic is looped forever, permitting a technician to scope out the circuitry. When looping on error, the color bar display will stay up because CARDTEST never returns to MAIN and so the splash screen (pretty girl in war paint) never gets displayed on the TV.

CARDTEST actually needs two pieces of information to run a diagnostic properly. He of course needs the entry address of the diagnostic itself (where to call it at). He also needs the address of a text string to output via the RS-232 serial port to inform the user WHICH diagnostic is passing or failing. Instead of having two lists (test_list and name_string_list) and keeping both lists correct, CARDTEST expects each diagnostic to have TWO entry points, one entry to run the test and a second entry to return a pointer to the diagnostic's name string in AX0. By convention, the second "retrieve the pointer" entrance is always 2 addresses LOWER than the main "run the diagnostic" entrance. Array test_list contains the "run the diagnostic" entrance, and CARDTEST computes the "retrieve the pointer" entrance by subtraction.

Module RAMTEST checks the 2185's internal data memory and then does the external 256Kb video memory. The video ram interface, mapping and page registers are described in the "Hardware - Video

Ram" section (above). In a nutshell, the bottom half of data memory space can be mapped out to a page of Video ram. Diagnostic dmtest maps the lower half of datamemory INSIDE the 2185 and runs the page test module. This test ought to pass, since it it checking RAM inside the DSP. If it fails, it is most likely a programming error in the page_test code. Diagnostic vramtest maps the lower half of data memory OUTSIDE to the video ram chips, and runs page_test once on each page of video memory. Diagnostic pagereg insures that all NPAGEs of video ram are separate and distinct, by writing a different number into the same location of all the pages and then reading it back and ensuring the all NPAGES are different. Failure in pagereg means just that, the hardware page register is failing to select all the pages of video memory.

Do the processor, the 7176,then the video ram buffer, the ADV601's and the SAA7111 last.
7176 Test. Set the chip to output NTSC color bars in master mode. Use the ADV7176 handler's dac_cbar function to do this.
CPU test. Take the EZKIT code as is
Video Ram buffer. Do address and address complemented from the CPU
ADV601's Do the bin width register test and then the fifo test on both chips.
SAA7111. Enable the color bar output of the 7176 back into the SAA7111. Check for horizontal PLL lockup and NSTC video reported. Then observe the video inputs (Composite and S-video). If sync is detected, report PAL or NTSC. If PAL detected, switch the 7176 over to PAL color bars. As a short cut, ignore PAL for the first go round.
Go thru the chip handlers to access the chips.
Do loop on error. If a test fails, autometically loop on it til it passes to permit technicians to scope out the problem.

8.3 command parser

8.3.1 Inputs

8.3.2 Outputs

8.3.3 Processing

Take this module from the EZKIT as is.

8.4 User Interface

8.4.1 Inputs

8.4.2 Outputs

Target Compression Ratio
Target Frame Rate
Sharpen/Soften
Chroma Intensity
Field Double
Resolution
Bit Error Rate

8.4.3 Processing

Read the buttons on the vertical retrace (60 Hz) to debounce them. Follow a wrist watch setting scheme. Make the results availible to the system. Give user feedback via Close Caption and RS232 (both together)

8.5 Encoder Interrupt handler Encisr.dsp

8.5.1 Inputs

8.5.2 Outputs

8.5.3 Processing

Read Int mask & status register. Process ALL interrupt conditions in this order,
LCODE (call Lcode handler)
STATS_RDY (call binwidth calclulator)
FIFO_SRQ (call Fifo Unloader)
MEM_ERR, FIFO_ERR,FIFO_STOP,CCIR_656ERR, (Call error reporter to complain)
Permit Encode LCODE interrupt to punch thru the fifo Unloader

8.5.4 I/O

Ready for new address flag. Mode modules wait for this flag to set before writing new field info, and clearing the flag.

8.5.5 Processing

Upon LCODE interrupt do the following
Output field size.
Reset Fifo Unloader's new field buffer address and page register setting.
Shut down fifo unloader.
Set the "ready for new address" flag to let folk know that they can give us the next buffer address any old time now.

8.6 Decoder Interupt handler

8.6.1 Inputs

8.6.2 Outputs

8.6.3 Processing

8.6.3.1 Main interrupt entrance (capture_isr)

Control passes to capture_isr whenever the encode (raw video to compressed video) ADV601 asserts Host Interrupt (/HIRQ). This ADV601 is interfaced to IRQL0. Only Lcode (IRQ2) and the decode ADV601 (IRQl) have higher priority. The service routine first reads the ADV601 interrupt mask and status register. This read will clear the interrupt. The program then services all the interrupts that are set, and loops back to read the interrupt mask and status register again. Normally it will read zero, (no ADV601 interrupts assserted) and the program can exit. Occasionally a second interrupt condition will occur (say FIFO_SRQ asserting while the STATS_RDY code is executing.) in which case we service it too. To prvent a lockout in the event of hardware failure. we have the STUCK_HOT counter to force a return after a reasonable number of services. The three error conditions (CCIR_ERR, FIFO_ERR and MEM_ERR are counted for debug purposes, but only MEM_ERR causes a chip reset via a call to encode_init. CCIR_ERR and FIFO_ERR have no corrective action as of this writing.

STATS_RDY is serviced by moving the contents of array BW_RECIP_BW into the binwidth and reciprocal bin width registers of the ADV601. Module SERVO and RCURVE running in the background are responsible for refreshing BW_RECIP_BW as required. As of this writing the statistics registers are not read, but they may well be in future versions.

FIFO_SRQ is serviced by moving the contents of the FIFO into video ram. DRAIN_FIFO is the bottom of the important loop which moves one 16 bit word per machine instruction. The loop runs until the counter is exhausted or the LCODE interrupt is asserted. Prior to loop entry, the LCODE interrupt is enabled. Should LCODE occur, the loop will be interrupted and control will pass to LCODE_ISR. LCODE can only happen when we are reading the FIFO, it is not a truly asynchonous interrupt that can happen at any time. Upon exit from the loop, the routine checks MY_EOF_FLAG which will be true only if LCODE occured, in which case end of field processing (chk_late_lcode and capture_switch_buf) are called.

8.6.3.2 Lcode Interrupt entrance (lcode_isr)

Normally LCODE sets somewhere in the middle of unloading the FIFO. In the normal case all that is required is to force the DRAIN_FIFO loop to terminate. the OWNCNTR = 1 statement handles this, and setting the flag MY_EOF_FLAG is the only processing required. Then the RTI statement carries control back to the DRAIN_FIFO loop which then exits. Two special cases need be considered. Suppose LCODE is active before the DRAIN_FIFO loop starts executing? Or suppose the last word in the FIFO is also the last word of the field? If LCODE is hot before DRAIN_FIFO starts looping, then we must do a long word read in the LCODE routine to clear the LCODE interrupt. Otherwise LCODE stays hot and we cannot ever get out of the service routine. If the routine does RTI with LCODE hot, we come right back in at the top on the very next machine instruction. The only way to clear LCODE is to read a long word out of the ADV601 FIFO. The other case causes LCODE to go sometime between the end of the DRAIN_FIFO loop and the instruction that masks off LCODE. Again, corrective action is to read one long word out of the FIFO.

8.6.3.3 (check_late_lcode)

Sometimes interrupt latency causes the DRAIN_FIFO loop to read one to many long words out of the FIFO. This causes the Start of Field code to end as the LAST word of the prior field rather than the first word of the next field. CHECK_LATE_LCODE reads backwards from the end of field buffer looking for a start of field code. Should it find one, it moves it to array LATE_STUFF and puts the numbe of late words into BACKUP.

8.6.3.4 (fix_late_lcode)

After CHECK_LATE_LCODE finds a field header (or more) and puts it into LATE_STUFF, this routine copies LATE_STUFF into video ram in the proper location (start of next field compressed data. When it does this, it makes necessary corrections to the DRAIN_FIFO loop counter so that the FIFO read will ALWAYS end of a page boundary.

8.6.3.5 (capture_switch_buf)

8.6.3.6 (header_begin)

8.6.3.7 (header_finish)

8.6.3.8 (vram_init)

8.6.3.9 Startup routine (capture_go)

8.7 Fifo Stuffer

8.7.1 Inputs

New field buffer address
New field buffer page register setting.
New field size

8.7.2 I/O

Ready for new field flag. Mode modules wait for this flag to set before writing new field info and clearing the flag..

8.7.3 Processing

Upon FIFO_SRQ interupt, stuff one fifo load into the compressed data port. On the last fifo_load of the field, stop stuffing right at the last word. of the field. After stuffing the last word of the field, read the new field description (addrees, page reg & size, and set the ready for new field flag indicating that the supervisory program (basic mode or 2board mode) can pass the next field information.

8.8 Vertical Retrace Interrupt Handler

8.8.1 Inputs

Pointer to Close caption string to display

8.8.2 I/O

Ready for new closed caption string. Caller waits for flag to set and then passes in a new caption, and clears the ready flag.

8.8.3 Processing

Primary purpose is to load two close caption bytes into the 7176 on each FRAME (not field). Secondary purpose is to scan and debounce the user interface keys at 30 hz. Set the ready for new CC string when the last byte of the previous one is done.

8.9 Bin Width Calculator (servo.dsp)

8.9.1 Inputs

capture_siz	Last compressed field size	Unsigned 16 bit integer	Long words per field

compress_tgt	Target compressed field size.	Unsigned 16 bit integer	Long words per field

8.9.2 Outputs

scrunch	Bin width register settings	7FFF H = Max field size	8000 H = Min field size

8.9.3 Processing

The servo module compares the size of the last compressed field with the field size target and computes an output variable (scrunch) for the rcurve module that specifies the amount of compression to apply to the next field. A standard Proportional-Integral-Dirivative servo loop is coded. If the actual field size is LESS than target field size, output variable scrunch will increase (go toward 7FFF hex). If the actual field size is GREATER than target field size, scrunch will decrease (go toward 8000 Hex). The multiplier is used to scale actual and target field size to prevent overflow in two's complement arithmetic. Field size is expressed in long words per field which can range up to 65536 in a 16 bit unsigned variable which exceeds the range of 16 bit signed variables (32767). The servo error signal is computed by subtracting actual from target field size.

The dirivative of the error is computed by subtracting the newly computed error from the previous value of error. A more sophisticated program might use the ADV601 statistics to compute an error dirivative. This approach would cancel the one field time transportation lag suffered going thru the ADV601, and permit the system to anticipate and react more rapidly to video scene changes.

The error integral is computed by repeatedly adding the servo error signal into the error accumulator. Hardware saturation is used to prevent the error integral from rolling over in the event of a large un cancelable error. Normally, the error integral goes into the feedback signal to force the field size toward the desired size. This reduces the error to zero and the error accumulator stabilizes. If you ask for more compression that the system can do, then the error doesn't go to zero, and the error accumulator will keep growing. It is important to avoid rollover which will cause violent system oscillations. For that matter, all arithmetic in the servo loop must NOT rollover, but saturate at either plus or minus full scale.

Scrunch, the final output variable is computed by simply adding the error, the error dirivative and the error integral together. Each term is scaled by a gain term before summation. The gains are hand tuned for "best" system performance.

The servo module is taken from an earlier evaluation project which is documented in Analog Devices Application Note AN 524, available from the Analog Literature Center. This code performs somewhat better than the earlier version, due to some tuning over time. However the discussion on servo loop tuning and theory of operation given in AN 524 is still relavent to this code. Basically, set the intergral and dirivative gains to zero and apply as much proportional gain as possible until the system oscillates. Back off the proportional gain until the system is stable until all circumstances and types of video. Then increase the integral gain until the system transient response deteriorates. Finally apply dirivative gain to improve the transient response.

8.10 Bin Width Calculator (rcurve.dsp)

8.10.1 Inputs

Scrunch	Desired compression. 8000H is max, 7fff = min
sharp_control	Controls video sharpness by zeroing out higher frequency sub bands.

8.10.2 Outputs

bw_recip_bw	84 element array containing the 42 reciprocal bin widths and 42 reciprocal binwidths.

8.10.3 Processing

This module contains 127 different reciprocal bin width curves in array rbw. File rbw128.dat initializes this array, permitting curve changes by merely relinking the program. Input scrunch is scaled down to the range 0-127, and then used to select which reciprocal bin width curve to output. The do loop mask_off moves the selected 42 element reciprocal bin width curve out of array rbw into output array bw_recip_bw. The write pointer (I3) increments by TWO, to leave room to insert the necessary bin width values inbetween the reciprocals. Array bw_recip_bw is organized to match the reciprocal bin width and bin width register mapping of the ADV601. The encisr routine will do a block write of the entire bw_recip_bw array into the chip starting at recip bin width register.

Resolution is controlled by selectively zeroing out the recip bin width values of sub bands. Resolution 0 (sharp_control = 0) keeps ALL the sub bands. Resolution 1 zeros out the top sub band, resolution 2 zero's out the top two sub bands and so on. As more and more sub bands are zeroed out, the image gets softer and softer. In practice, the top sub band often contains no signal energy, merely noise, so a better image is obtained if no bits are expended encoding it. Only very high quality video contains useful signal in the top subband. The other resolutions, zero out more and more sub bands, softening the image a good deal. It is an interesting demostration of the properties of the wavelet transform, but probably not of great use in the real world.

The recip do loop computes the bin widths, from the reciprocal bin width by division. The loop guards against division by zero and assigns a bin width of zero if the reciprocal bin width is zero. The divide code is taken (comments and all) from the 21xx DSP library and inserted in line.

8.11 Close Captioning Formatter

8.11.1 Inputs

ASCII string to format
Rollup vs Popon captioning

8.11.2 Outputs

ASCII string with proper close captioning control bytes, at the begging and end..

8.11.3 Processing

Aligned so both bytes of control codes are on the same scan line. Do this by padding odd length strings to even with a trailing space.

8.12 ADV7176 hander

8.12.1 Functions provided

8.12.1.1 dac_init (Pal or NTSC, Video present/Not present)

Programs ALL ADV7176 registers, whether they need it or not. The PAL/NTSC switch is obvious. Dac_init also needs to know if the SA7111 is locked onto real video, or is coasting with the PLL unlocked in order to set the color subcarrier frequency properly.

8.12.1.2 ccaption (Write Close Capion byte pair)

8.12.1.3 dac_cbars (PAL or NTSC)

Calls the dac_init routine first and then sets the internal color bar bit in mode register 1.

8.12.2 Outputs

8.13 SAA7111 Handler

8.13.1 Functions provided

adc_init   (S-video, Composite or 7176 feedback)

8.13.2 Outputs

Horizontal PLL locked
Pal or NTSC

8.13.3 Processing

8.14 i2c bus driver

8.14.1 Inputs

Pointer to register data to transmit or recieve
pointer to (address of) First register to read or write.
CHIP ID code (7111 vs 7176)
Number of registers to move

8.14.2 Functions Provided

rite_i2c_chip
read_i2c_chip

8.14.3 Outputs

Success or Failure

8.14.4 Processing

Transfers byte wide data to and from the I2C bus. Does a “bus start”,   then sends the chip id and the first register address to the bus. Then it loops doing read or write of the bus for the specifiec number of registers. Transmission is always terminted with an I2C bus stop. Checks for acknowledge from the target chip after each byte transfer.

8.15 RS-232 driver

8.15.1 Inputs

8.15.2 Outputs

8.15.3 Processing

Take this module as is from the EZ-Kit

8.16 Fetch pix from prom

8.16.1 Inputs

Which picture
address in video ram to write the picture to.

8.16.2 Outputs

Size of picture

8.16.3 Processing

8.17

9. Output Processing

10. Data Dictionary

10.1 Video Ram Pointers

Pointers to addresses in video ram require some special processing. Since the video ram is paged, a video ram access implies resetting the video ram page bits, of which there are four. The entire video ram is 256 K bytes. However the DSP accesses video ram by words (there is no byte addressing in the 2185) so the maximum address is the 128K. However the video is always accessed as a 32 bit long word, thus making the maximum address 65K, which fits within a single 16 bit ADSP2185 machine word. Threfore all pointers to video ram, including the Next_field and Last_field pointers in the video buffer field headers, can be thought of a containing the 32 bit address of the desired piece of video. Such a pointer can be manipulated with ordinary arithmetic. For instance to advance a pointer from the beginning of field to the end of field, add the size of the field to the pointer. Naturally it is important to express the field size as the number of 32 bit long words in the field, not the number of 16 bit words in the field.

To use a video pointer put the four most significant bits into the page register. Left shift the 12 least signifcant bits once, and put them into a Data Address Generator Register. Subroutine OFF_SEG will separate the page bits and left shift the “offset” bits, returning both parts of the pointer in registers convenient for a call to VRAMPAGE, which then changes the page bits in the card control register.

10.2 The video buffer

10.3 Video Clips stored in PROM

10.4

11.