TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

DaVinci HDTV Using THS8200

Neal Frager and Juan Gonzales
Digital Customer Applications Team
Texas Instruments

DaVinci HDTV Using THS8200

1. Digital Video
2. Hardware Overview
3. Software Overview
4. HD Video Demo

DaVinci HDTV Using THS8200

- Outline
  - Digital Video
  - HDTV Standards
  - DM6446 VPBE HD Capabilities
  - THS8200 HD Video Encoder
  - Software Overview
  - HD Video Decoding Demonstration

Neal / Juan

Digital Video

- Digital Video: 25 min
  - Video Window / Memory Buffer
  - Resolution / Bits Per Pixel / Refresh Rate
  - Interlaced / Progressive
  - Standards: YCbCr (HDTV) / RGB (PC) / BT.656 (SDTV)
  - Video Timing Diagram
  - JG: YCbCr and RGB are pixel formats, BT.656 is a video protocol for SDTV which includes YCbCr.

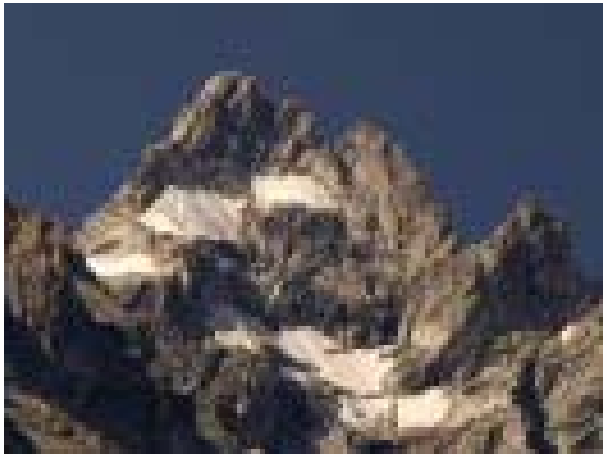
Digital Video

1. Resolution
2. Bits Per Pixel (BPP)
3. Video Pixels In Memory
4. Refresh Rate
5. Progressive / Interlaced
6. RGB and YCbCr Pixel Formats
7. HDTV Standards

Resolution

1491 x 991 pixels
303 KB

358 x 238 pixels
23 KB



149 x 99 pixels
6 KB

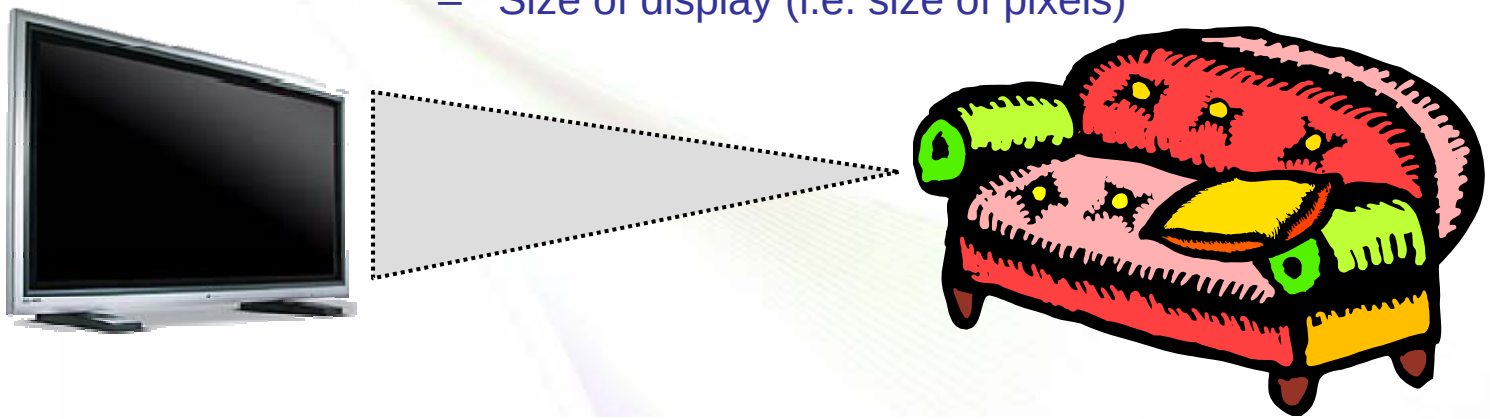
- Definitions
  - Pixel: (short for *picture element*) represents each point of information in a picture
  - Resolution: describes the number of pixels horizontally and vertically

What resolution should you use?

- Printers or displays with fixed dot (or pixel) densities
- Conforming to a specific standard (NTSC, PAL, etc.)
- Data rate, processing rate, storage capacity

Resolution

- Perceived resolution is a function of:
  - Actual resolution
  - Distance
  - Size of display (i.e. size of pixels)



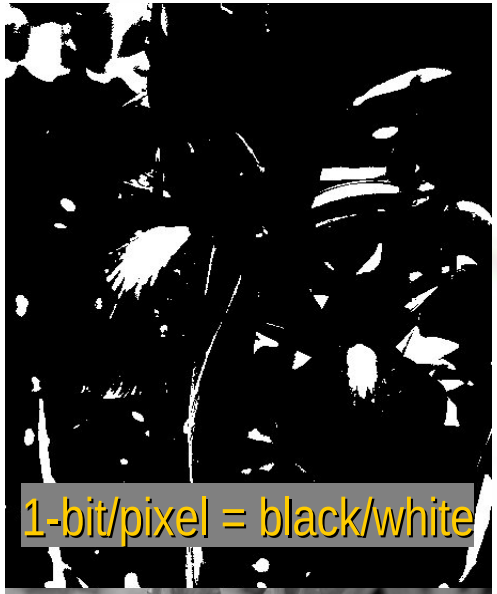
Standard (analog) TVs with 4:3 aspect ratio

Screen size	Suggested viewing distance
27"	6.75 ft
32"	8 feet
36"	9 feet
40"	10 feet
45"	11.25 feet
50"	12.5 feet

HDTV-capable TVs with 4:3 or 16:9 aspect ratio

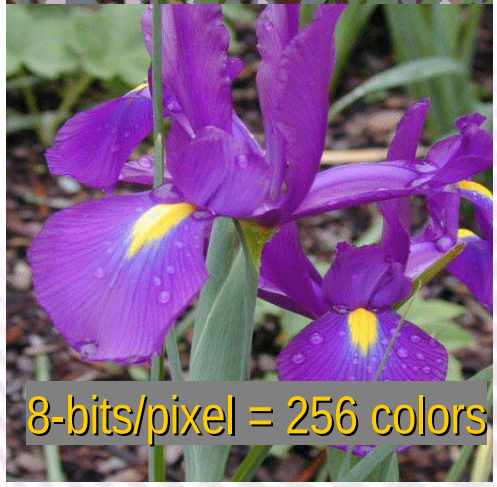
Screen size	Suggested viewing distance
30"	6.25 feet
35"	7.3 feet
40"	8.3 feet
45"	9.4 feet
50"	10.4 feet
55"	11.5 feet

Bits Per Pixel (BPP)



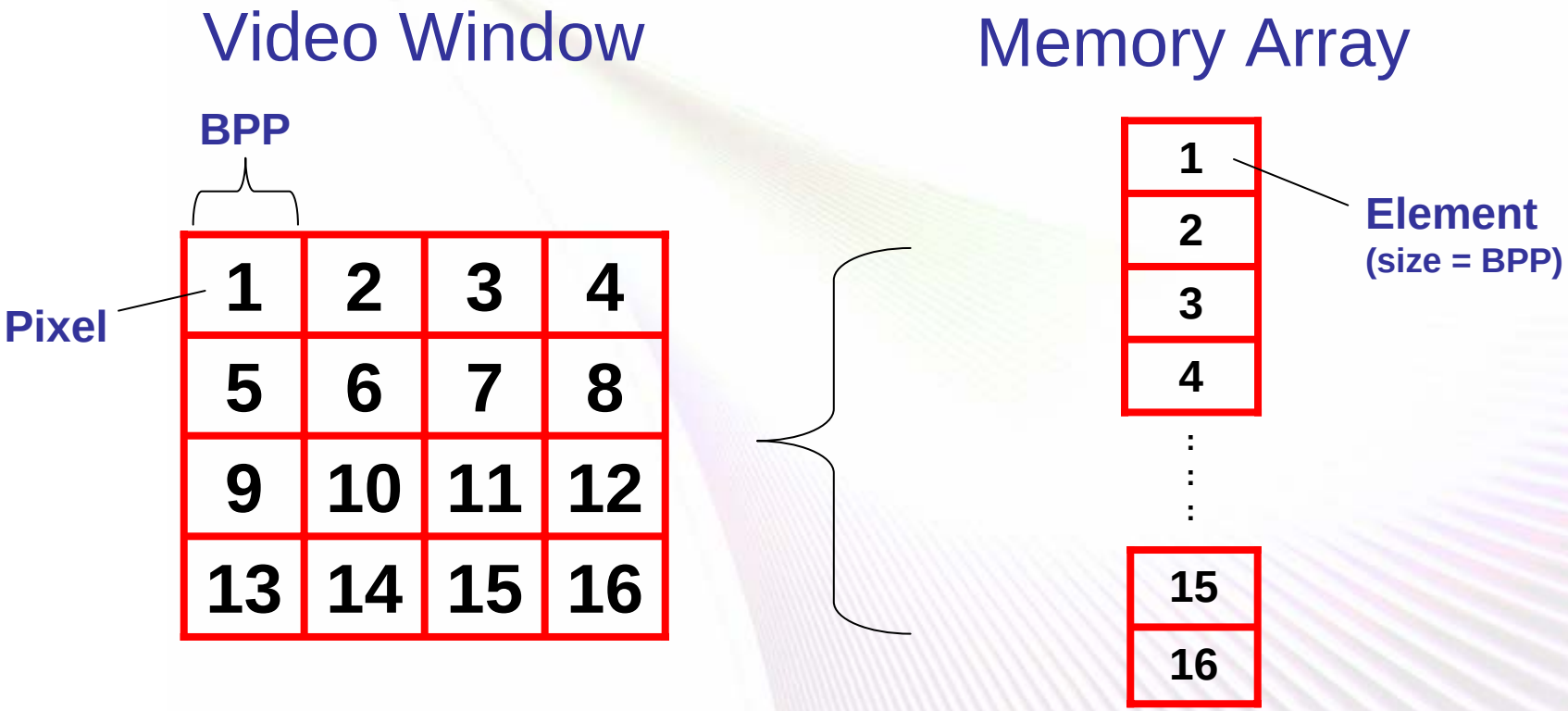
24-bits/pixel = 16 million colors

8-bits Red
8-bits Green
8-bits Blue



Minds in Motion

Video Pixels in Memory



Refresh Rate

5 frames/sec **15 frames/sec**

Minds in Motion

Progressive vs. Interlaced

15 frames/sec progressive

30 fields/sec interlaced

Minds in Motion

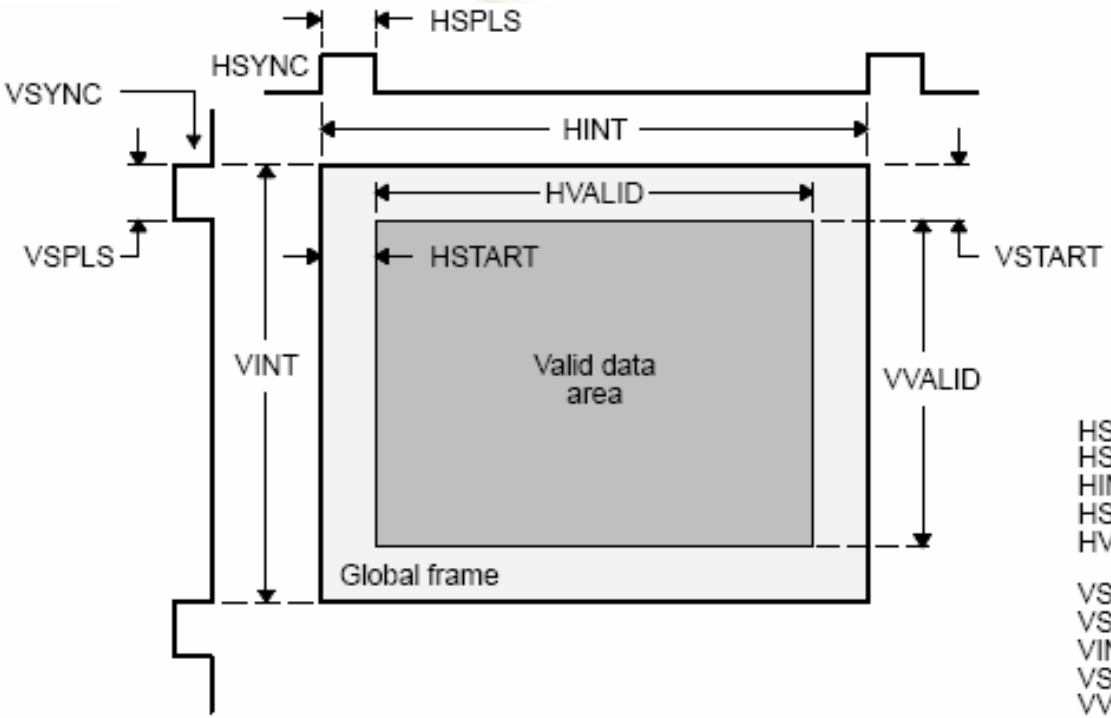
Interlacing Side Effects

- While interlacing can reduce the data rate without reducing the image resolution, it does produce some side effects

- Tomato is moving right in this example
- See how the image is distorted due to movement between the two fields …
- This is called "combing" (or "feathering")

- Similarly the skier seems to flicker (called "line twitter")
- Worse yet, very thin image elements (such as the ski pole) that are only one line wide seem to disappear and reappear (exacerbated by vertical movement)

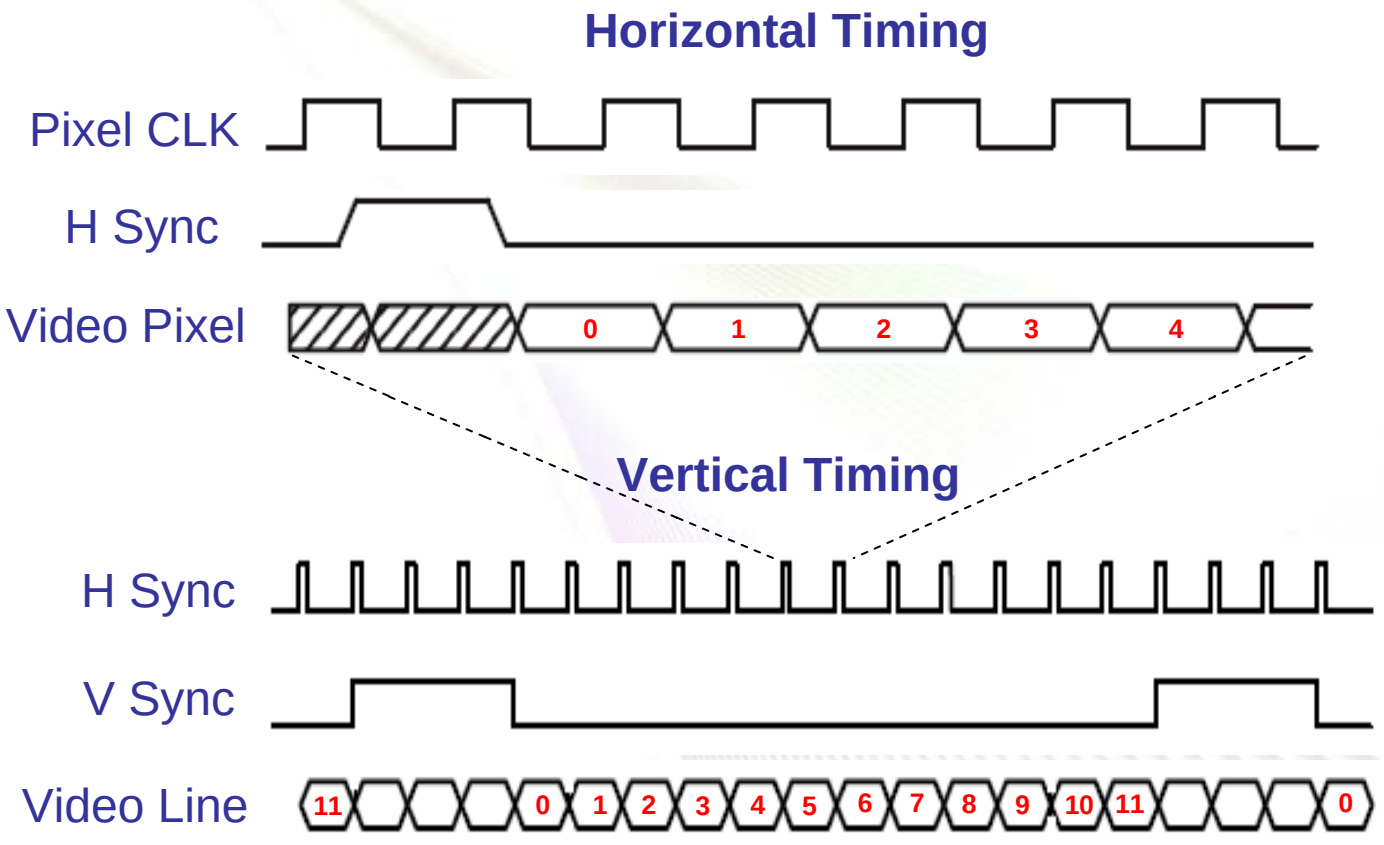
Progressive Video Timing



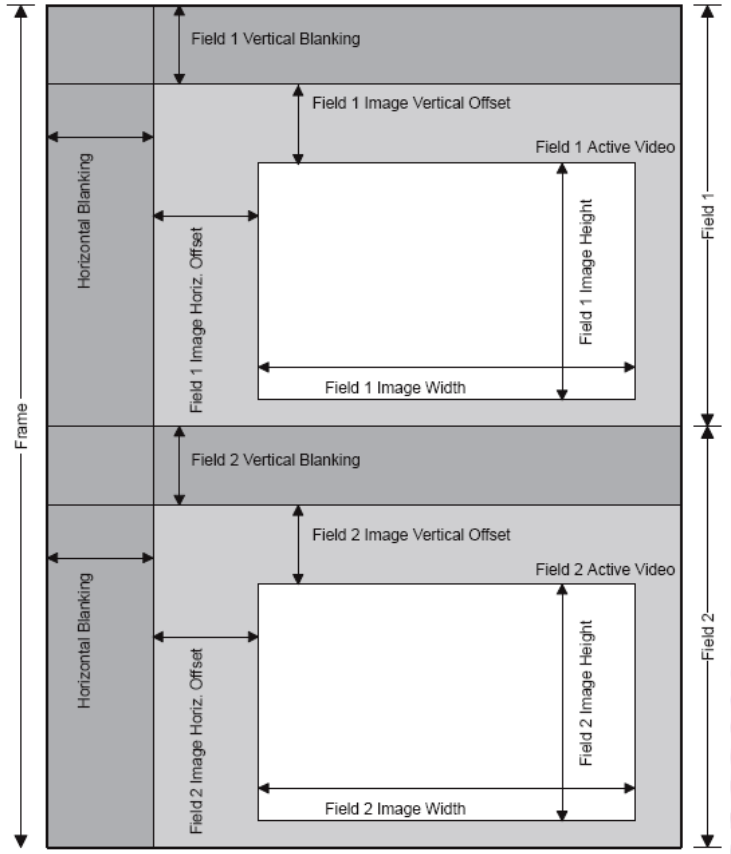
HSPLS – Horizontal sync width
HSYNC – Horizontal sync
HINT – Horizontal interval between HD
HSTART – Horizontal start of valid data
HVALID – Horizontal valid area

VSPLS – Vertical sync width
VSYNC – Vertical sync
VINT – Vertical interval between HD
VSTART – Vertical start of valid data
VVALID – Vertical valid area

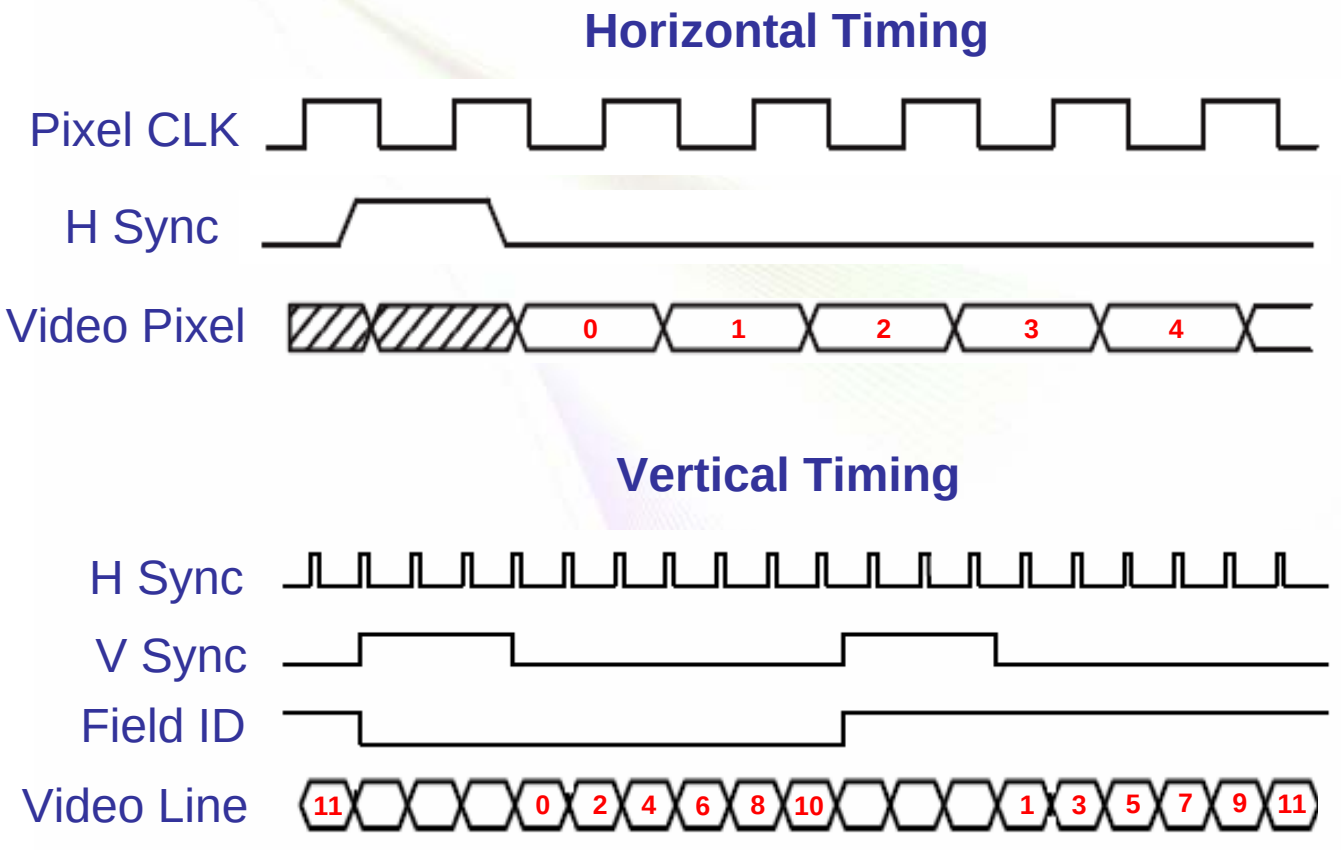
Progressive Video Timing



Interlaced Video Timing

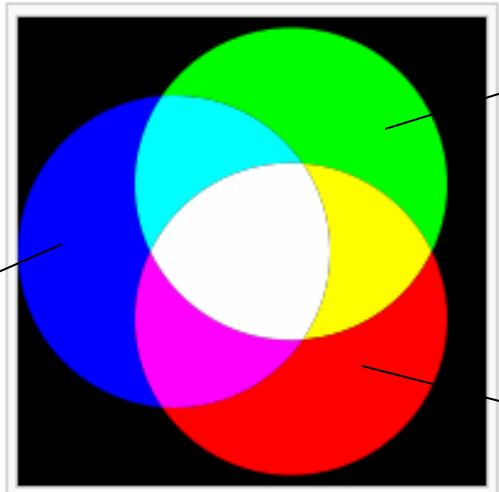


Interlaced Video Timing

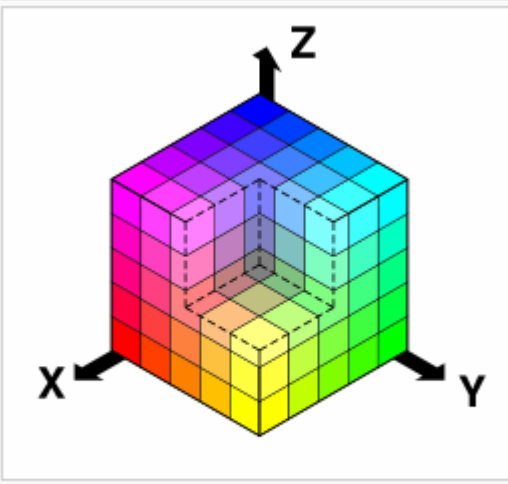


RGB Pixel Format

Pixels Split into Color Components



A representation of additive color mixing

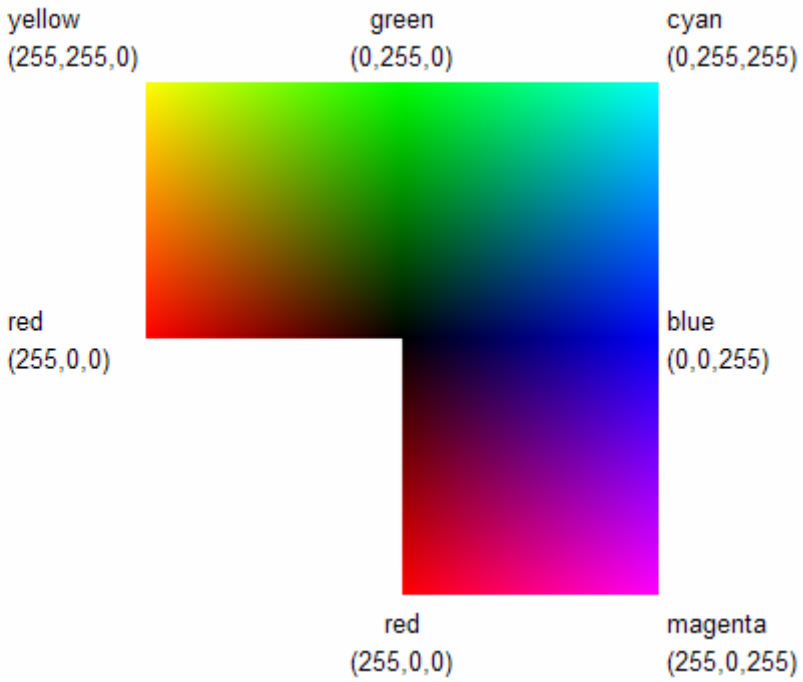


The RGB color model mapped to a cube (with cut-away shown)

RGB Pixel Format

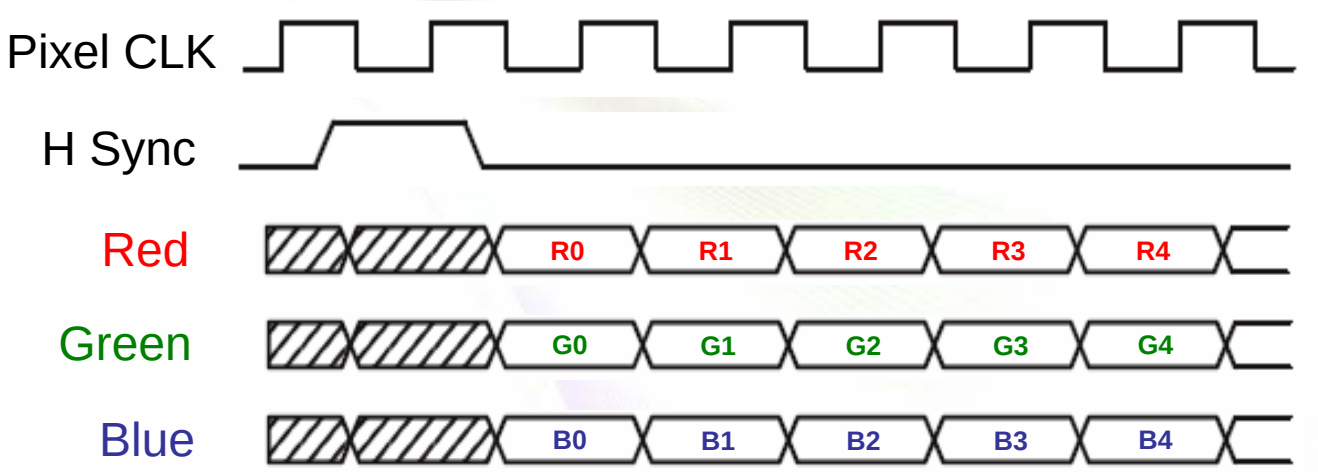
True Color 24-bit BPP Format (RGB888)

- (0, 0, 0) is black
- (255, 255, 255) is white
- (255, 0, 0) is red
- (0, 255, 0) is green
- (0, 0, 255) is blue
- (255, 255, 0) is yellow
- (0, 255, 255) is cyan
- (255, 0, 255) is magenta

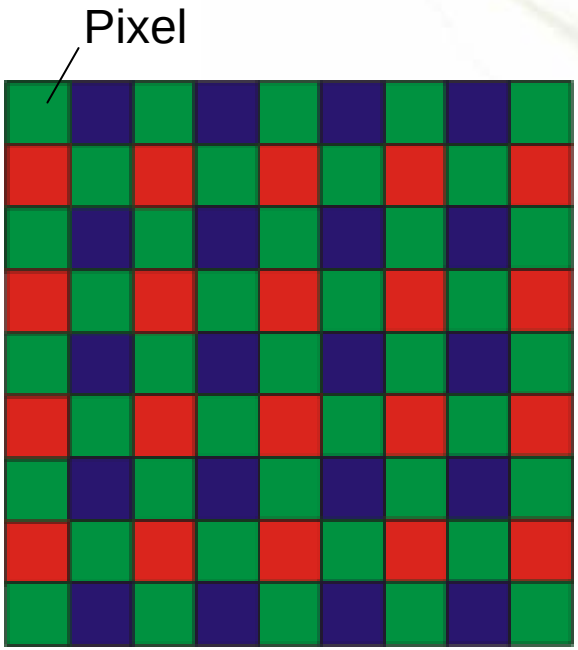


Color Can Be Reduced
- 18-bit: RGB666
- 16-bit: RGB565
- 8-bit: RGB332

RGB Pixel Timing

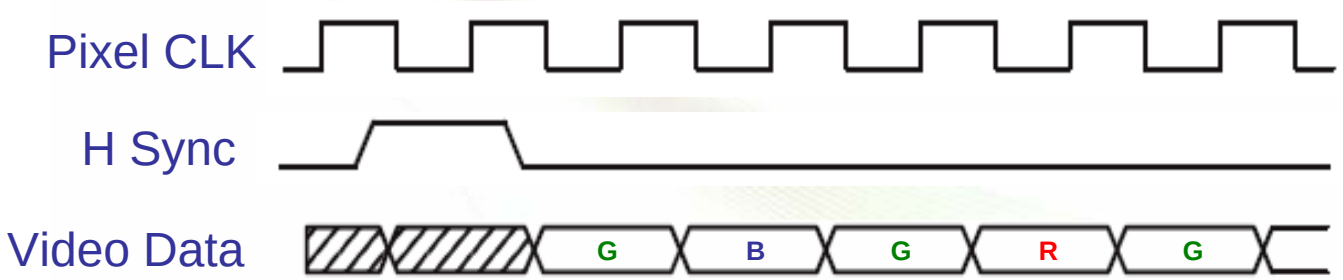


RGB Bayer Pattern Protocol



- Used By CCD and CMOS Sensors
- Each Pixel Contains Single Color Component
- Demosaic Algorithms to Generate RGB pixels
- Double Green Data for Human Eye Sensitivity

RGB Bayer Pattern Timing



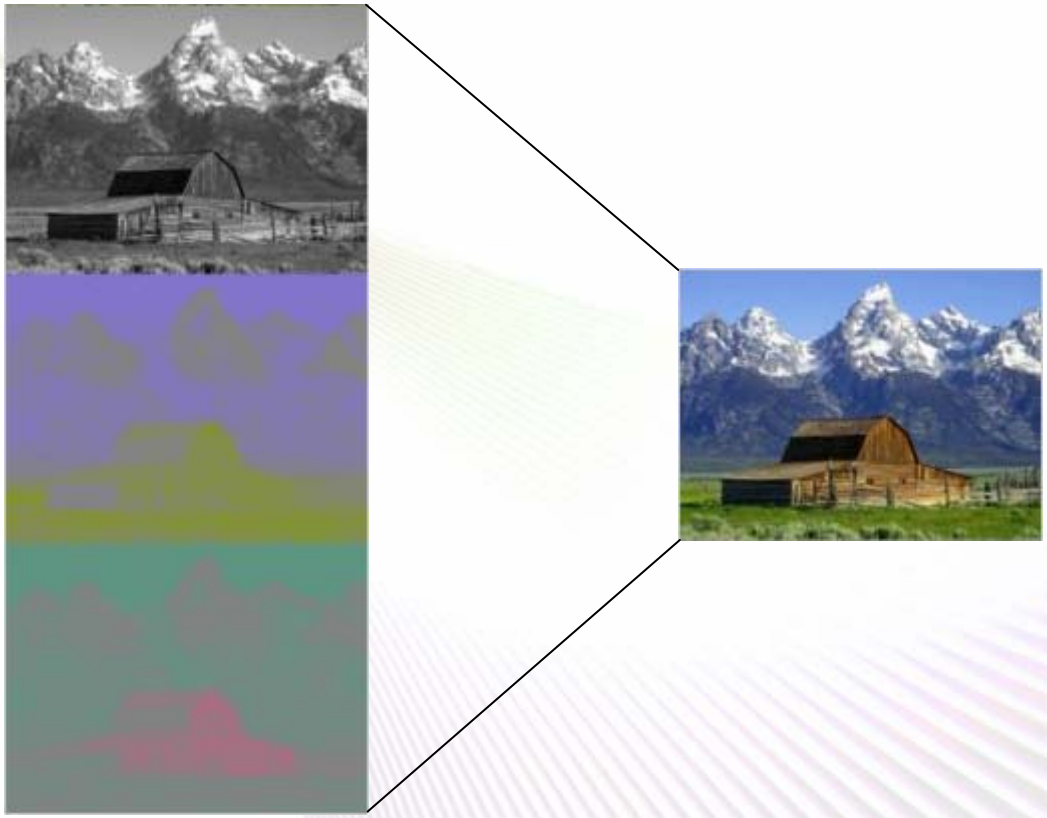
Only Single Data Bus to Drive!

YCbCr Pixel format

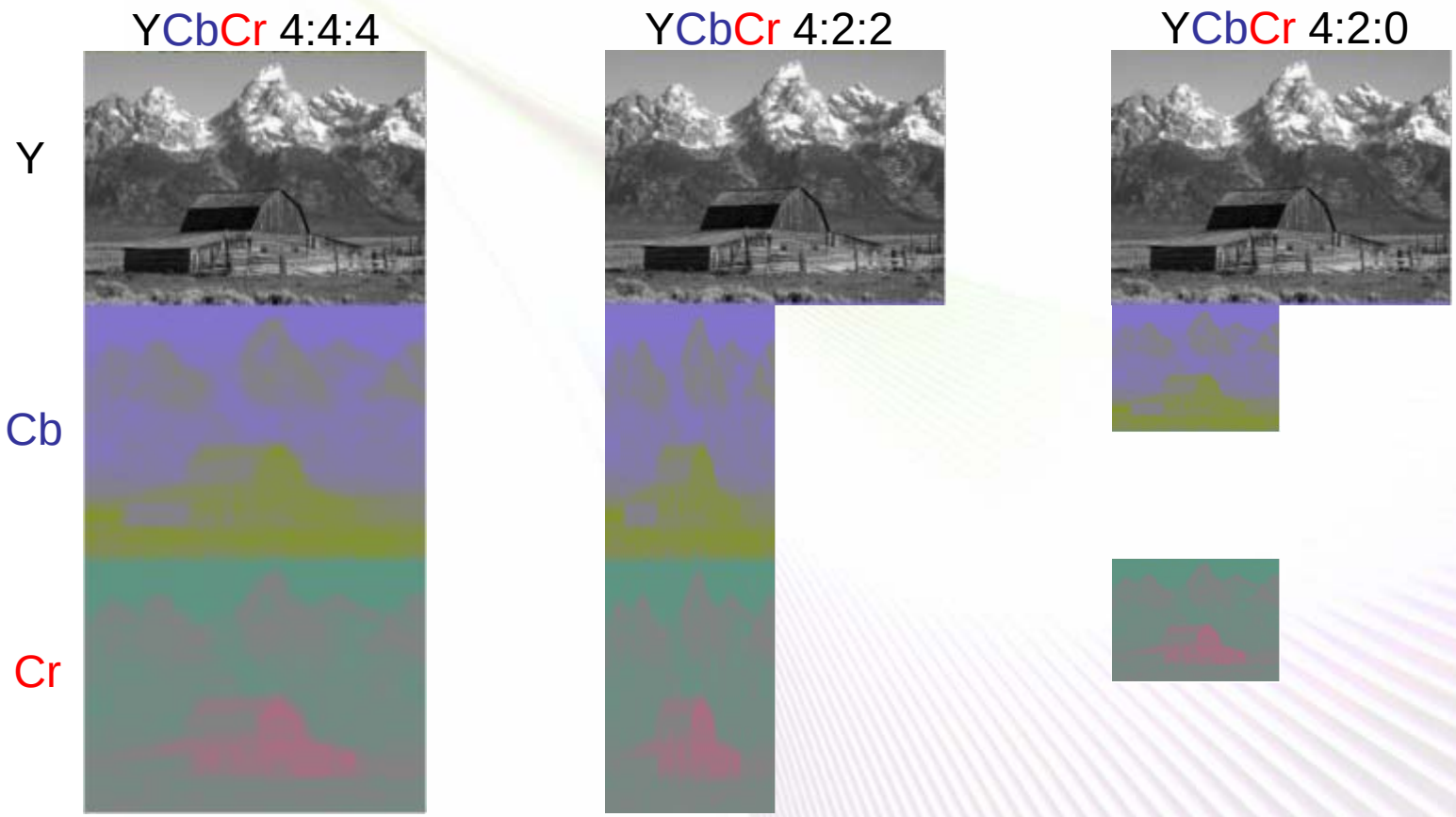
Luma (Y)

Chroma Blue (Cb)

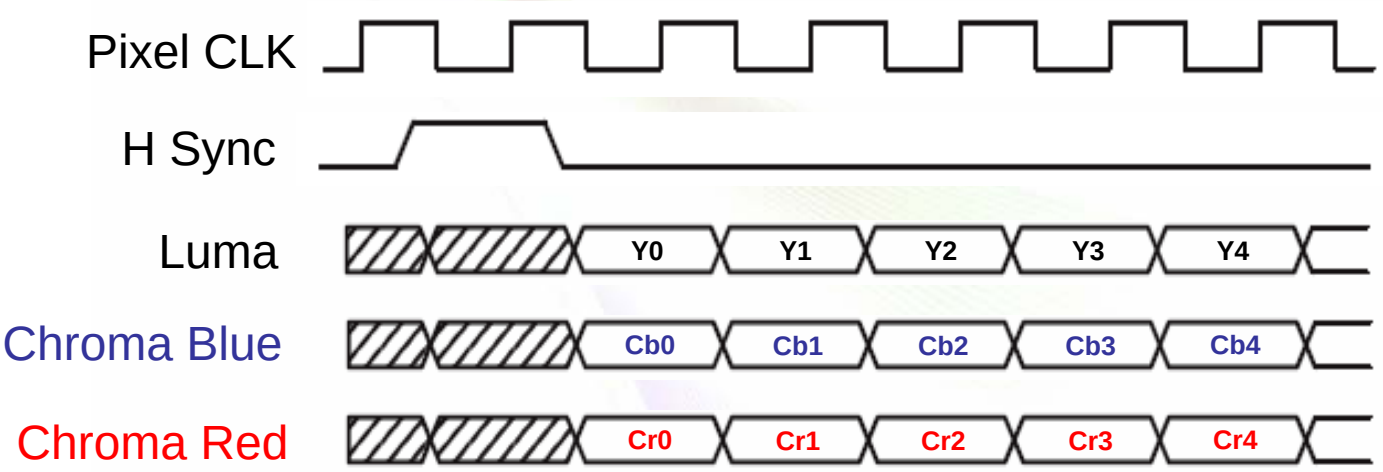
Chroma Red (Cr)



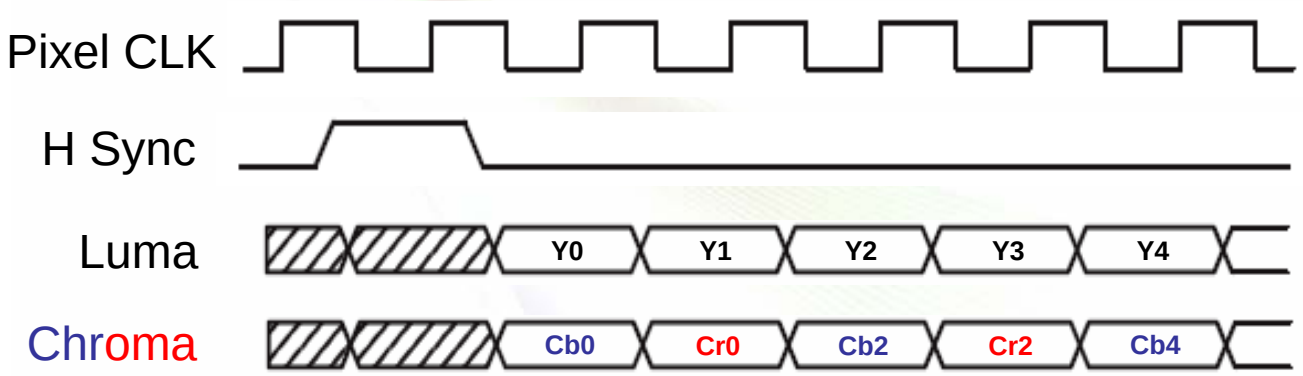
YCbCr Pixel Ratios



YCbCr 4:4:4 Pixel Timing

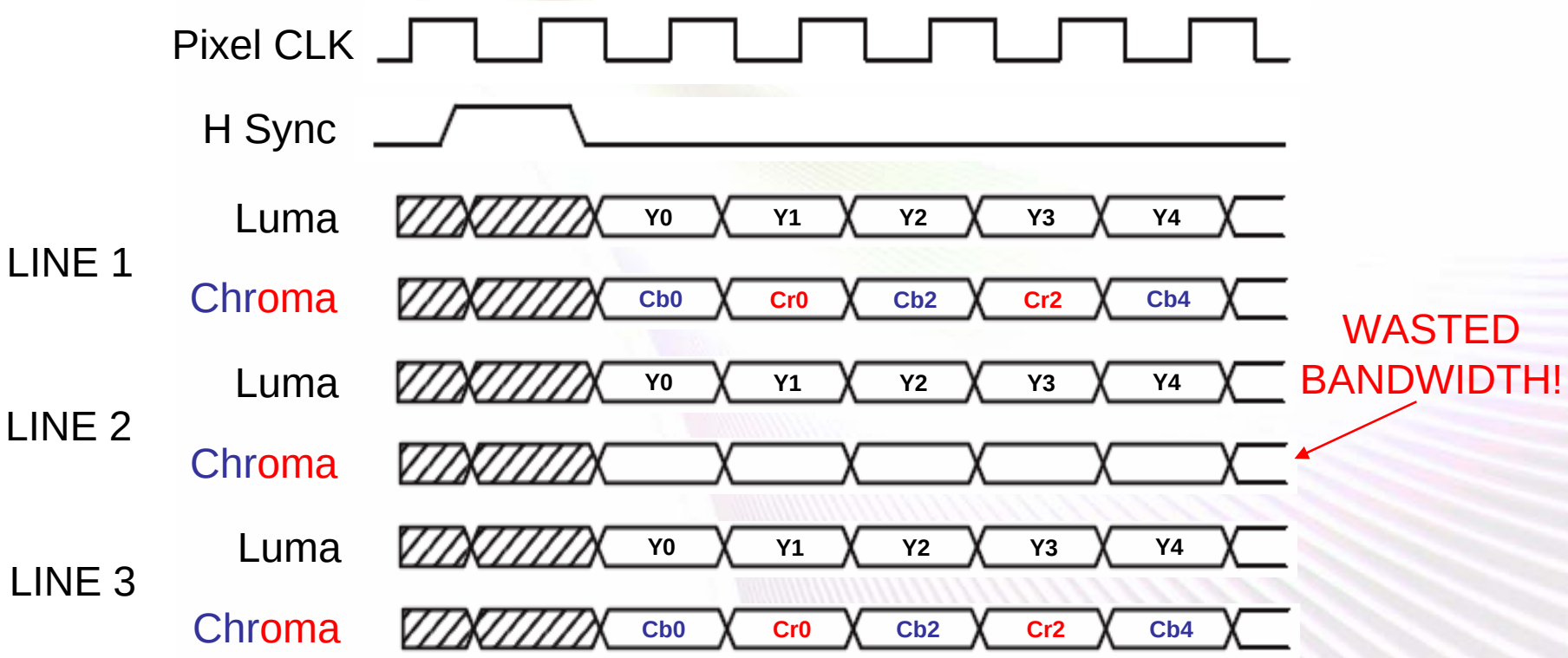


YCbCr 4:2:2 Pixel Timing



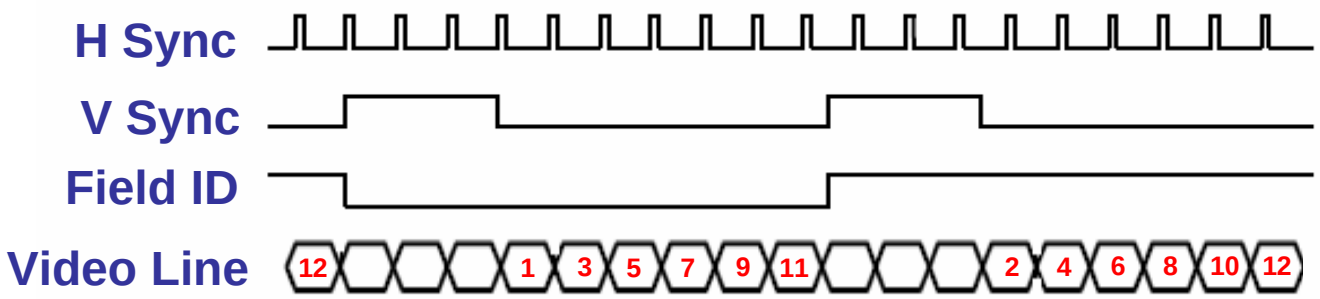
Only 2 Data Busses Instead of 3!

YCbCr 4:2:0 Pixel Timing



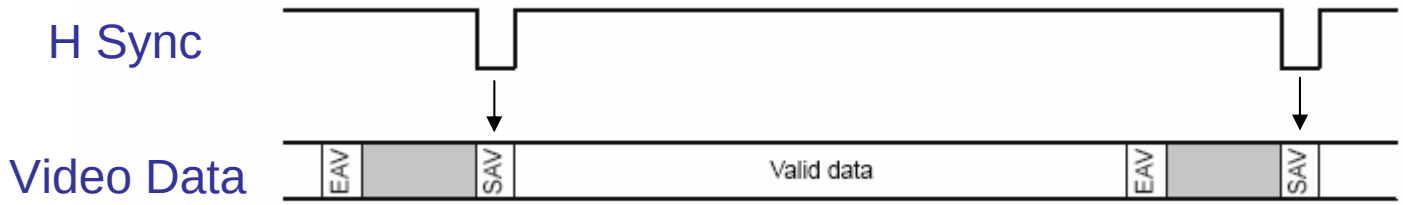
BT/ITU.656 Protocol

External Sync Model



BT/ITU.656 Protocol

SYNC SIGNALS CAN BE EMBEDDED IN DATA!



SAV = Start of Active Video
EAV = End of Active Video

BT/ITU.656 Protocol

Video Timing Reference Codes for SAV and EAV

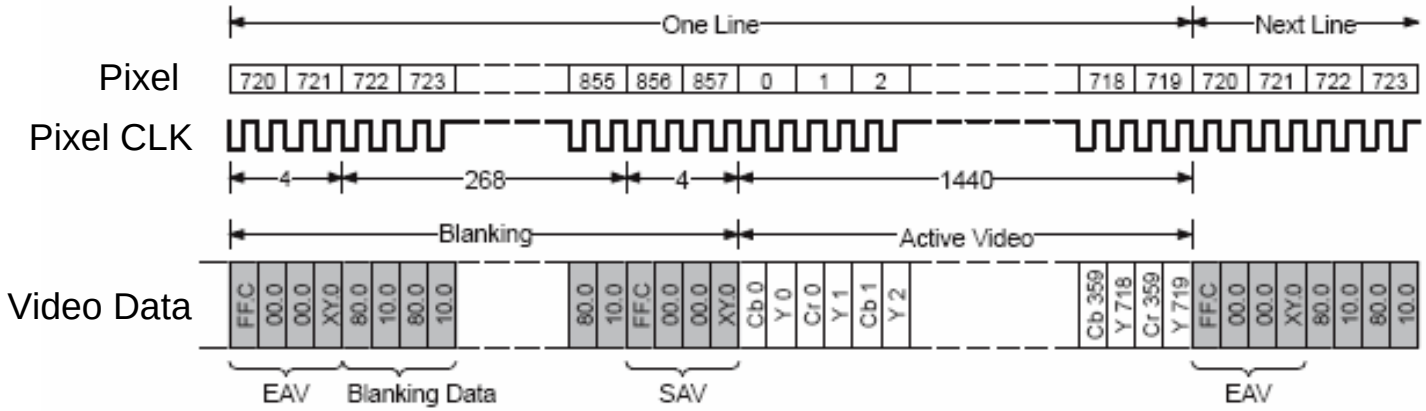
Data Bit Number	First Word (FF)	Second Word (00)	Third Word (00)	Fourth Word (XY)
9 (MSB)	1	0	0	1
8	1	0	0	F
7	1	0	0	V
6	1	0	0	H
5	1	0	0	P3
4	1	0	0	P2
3	1	0	0	P1
2	1	0	0	P0
1	1	0	0	0
0	1	0	0	0

F, V, H Signal Descriptions

Signal	Value	Command
F	0	Field 1
	1	Field 2
V	0	0
	1	Vertical blank
H	0	SAV
	1	EAV

BT/ITU.656 Protocol

NTSC 720x480 Example



Only CLK and Data Bus Needed!

HDTV Standards

- HDTV Standards – 5 min
  - 720p
  - 1080i

Common Video Standards

Standard	Resolution	Refresh Rate	I/P	Data Rate
NTSC	720 x 480	60 fields/sec	I	10M pix/sec
PAL	720 x 576	50 fields/sec	I	10M pix/sec
4CIF	704 x 576	30 frames/sec	P	12M pix/sec
CIF	352 x 288	30 frames/sec	P	3M pix/sec
QCIF	176 x 144	30 frames/sec	P	760K pix/sec
720P	1280x720	60 frames/sec	P	55 M pix/sec
1080I	1920x1080	60 fields/sec	I	62 M pix/sec

SDTV Standard NTSC

- Resolution = 720 x 480
- Bits Per Pixel = Typically 16
- Refresh Rate = 60 Fields Per Second (30 Frames)
- Video Format = Interlaced
- Pixel Format = Typically YCbCr 4:2:2

- Data Transmit Requirement:

  16BPP * (720x480) * 30fps = **165.888 Mbps**

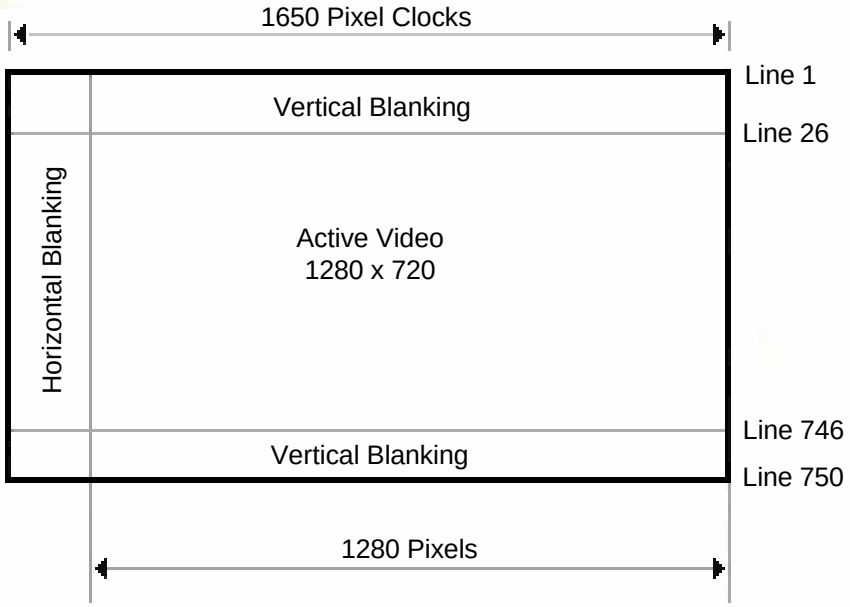
HDTV Standard 720p

- Resolution = 1280 x 720
- Bits Per Pixel = Typically 16
- Refresh Rate = 60 Frames Per Second
- Video Format = Progressive
- Pixel Format = Typically YCbCr 4:2:2

- Data Transmit Requirement:

  16BPP * (1280x720) * 60fps = **884.736 Mbps!!!**

720p Frame Standard



1650 Pixel CLKs * 750 Lines * 60 fps = 74.25 MHz Pixel Clock

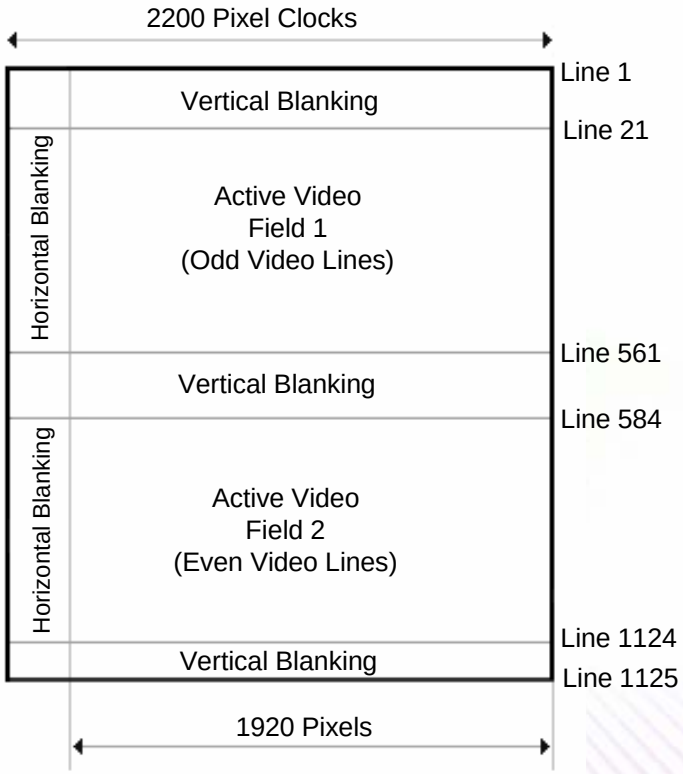
HDTV Standard 1080i

- Resolution = 1920 x 1080
- Bits Per Pixel = Typically 16
- Refresh Rate = 60 Fields Per Second (30 Frames)
- Video Format = Interlaced
- Pixel Format = Typically YCbCr 4:2:2

- Data Transmit Requirement:

  16BPP * (1920x1080) * 30fps = **995.328 Mbps!!!**

1080i Frame Standard



2200 Pixel CLKs * 1125 Lines * 30 fps = 74.25 MHz Pixel Clock

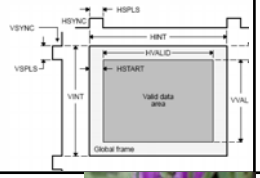
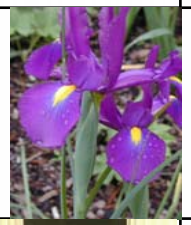
DM6446

- DM6446 VPBE HD Capabilities – 10 min
  - Pixel Clock Options
  - Features
  - Output interface

DaVinci HDTV Using THS8200

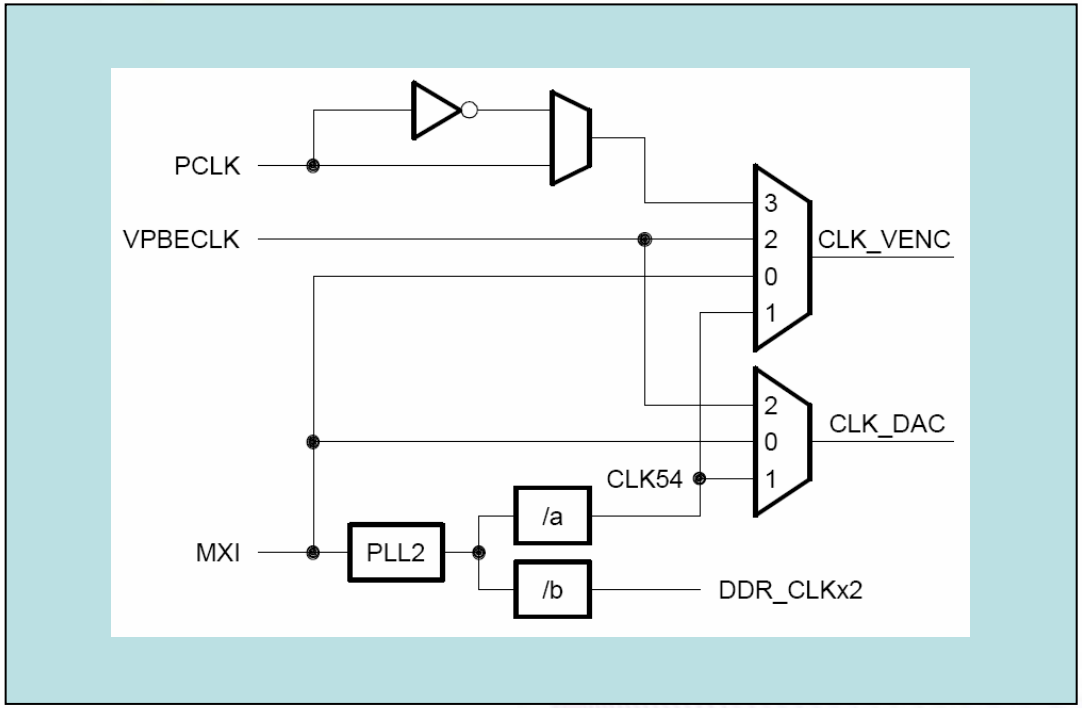
1. Digital Video
2. Hardware Overview
3. Software Overview
4. HD Video Demo

Hardware Overview

Feature	DaVinci VPBE	THS8200
Maximum Resolution 	4096 x 2048 in all 4 Windows	8192 x 2048
Timing 	Master/Slave Progressive/Interlaced	Master/Slave Progressive/Interlaced
Pixel formats 	8-bit YCbCr, 16-bit YCbCr, RGB888, RGB565	Flexible 10/20/30-bit YCbCr and RGB
Maximum Pixel clock 	75 MHz	205 MHz

Minds in Motion

DaVinci VPBE Pixel Clock



MUXSEL	CLK54	CLK_VENC	CLK_DAC
0	Off	27	27
1	54	54	54
2	Off	VPBECLK	VPBECLK
3	Off	PCLK	Off

DaVinci VPBE Output Formats

Pin Name	YCC16	YCC8 / BT.656	RGB888	RGB666
HSYNC	HSYNC	HSYNC	HSYNC	HSYNC
VSYNC	VSYNC	VSYNC	VSYNC	VSYNC
VCLK	VCLK	VCLK	VCLK	VCLK
VPBECLK	VPBECLK	VPBECLK	VPBECLK	VPBECLK
YOUT7	Y7	Y7,Cb7,Cr7	R7	R7
YOUT6	Y6	Y6,Cb6,Cr6	R6	R6
YOUT5	Y5	Y5,Cb5,Cr5	R5	R5
YOUT4	Y4	Y4,Cb4,Cr4	R4	R4
YOUT3	Y3	Y3,Cb3,Cr3	R3	R3
YOUT2	Y2	Y2,Cb2,Cr2	G7	G7
YOUT1	Y1	Y1,Cb1,Cr1	G6	G6
YOUT0	Y0	Y0,Cb0,Cr0	G5	G5
COUT7	C7	LCD_AC	G4	G4
COUT6	C6	LCD_OE	G3	G3
COUT5	C5	BRIGHT	G2	G2
COUT4	C4	PWM	B7	B7
COUT3	C3	CSYNC	B6	B6
COUT2	C2	-	B5	B5
COUT1	C1	-	B4	B4
COUT0	C0	-	B3	B3
GIO46/PWM1	-	-	R2	R2
GIO47/PWM2	-	-	B2	B2
GIO0	-	-	LCD_OE	LCD_OE
GIO2	-	-	G0	
GIO3	LCD_FIELD	LCD_FIELD	B0	LCD_FIELD
GIO4	-	-	R0	
GIO5	-	-	G1	
GIO6	-	-	B1	
GIO38	-	-	R1	

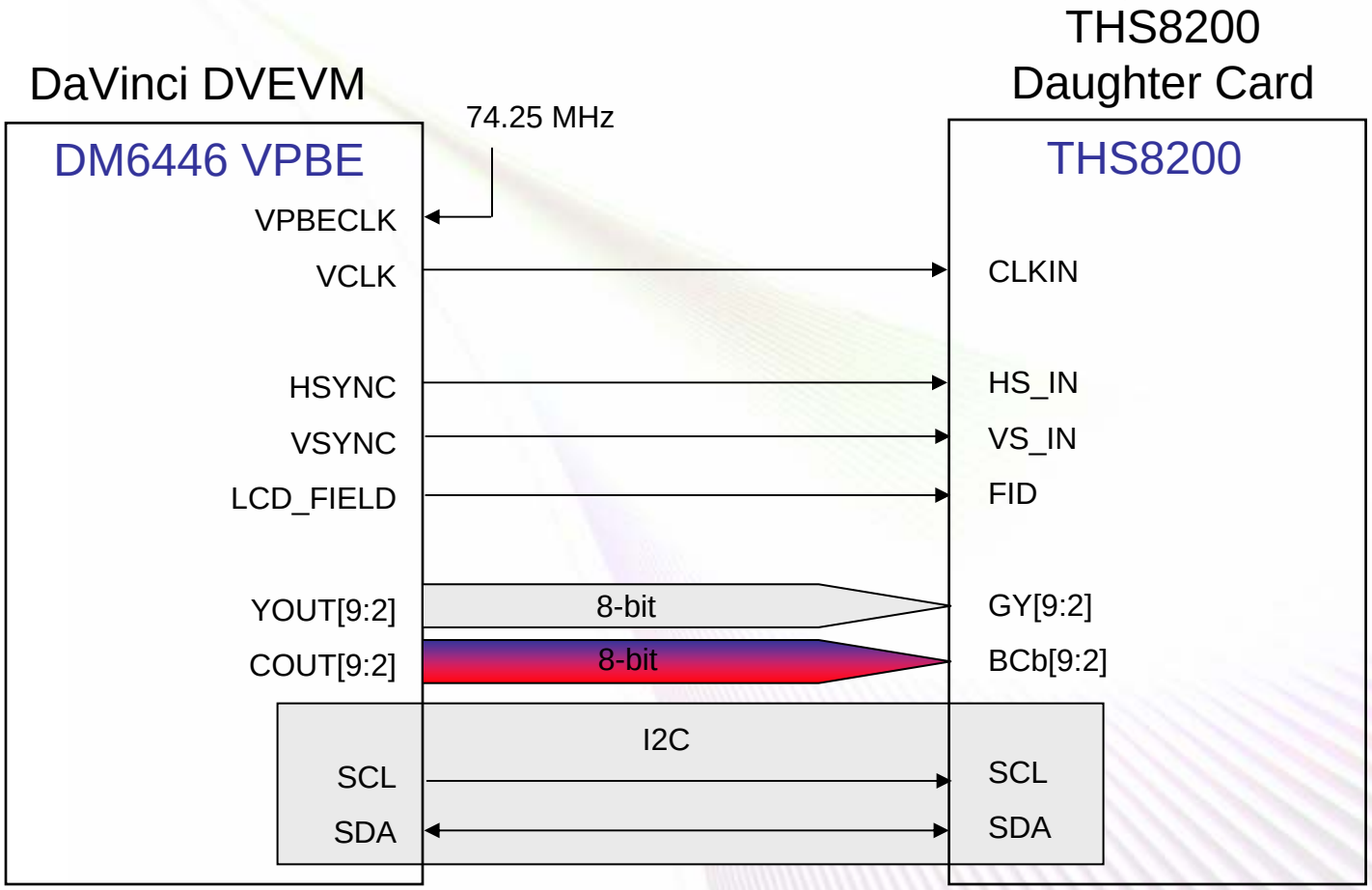
THS8200 30-bit Input

GY[9:0]/[9:2]	G0/Y0	G1/Y1	G2/Y2	G3/Y3	G4/Y4	G5/Y5	G6/Y6	G7/Y7
BCb[9:0]/[9:2]	B0/Cb0	B1/Cb1	B2/Cb2	B3/Cb3	B4/Cb4	B5/Cb5	B6/Cb6	B7/Cb7
RCr[9:0]/[9:2]	R0/Cr0	R1/Cr1	R2/Cr2	R3/Cr3	R4/Cr4	R5/Cr5	R6/Cr6	R7/Cr7

THS8200 Standard Inputs

	INPUT INTERFACE					TIMING CONTROL		SYNCHRONIZATION	
	30-BIT	20-BIT	10-BIT(1)	16-BIT	15-BIT	EMBEDDED TIMING	DEDICATED TIMING	MASTER	SLAVE
[PRESET] HDTV-SMPTE296M progressive (720P)	X (4:4:4)	X (4:2:2)				X			X
[PRESET] HDTV-SMPTE274M progressive (1080P)	X (4:4:4)	X (4:2:2)				X			X
[PRESET] HDTV-SMPTE274M progressive (1080I)	X (4:4:4)	X (4:2:2)				X			X
[GENERIC] HDTV	X (4:4:4)	X (4:2:2)				X	X		X
[PRESET] SDTV-ITU.1358 (525P)			X(2)			X			X
[PRESET] SDTV-ITU-R.BT470 (525I)			X(3)			X			X
[PRESET] SDTV-ITU-R.BT470 (625I)			X(3)			X			X
[GENERIC] SDTV	X (4:4:4)	X (4:2:2)				X	X		X
[PRESET] VESA	X(4)			X(4)	X(4)		X	X	X

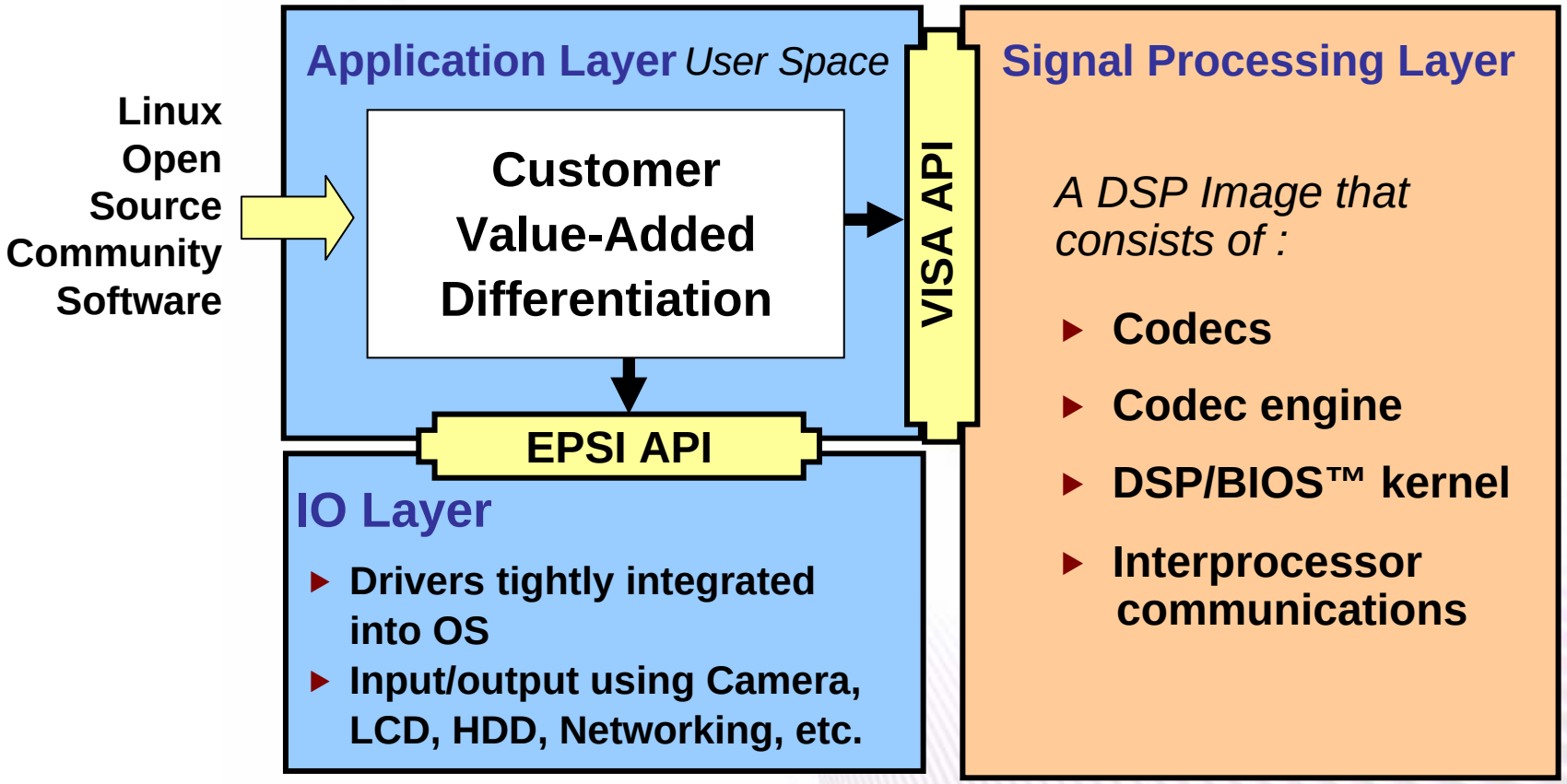
Hardware Block Diagram



DaVinci HDTV Using THS8200

1. Digital Video
2. Hardware Overview
3. Software Overview
4. HD Video Demo

Software Overview



- Software partitioning reflect the underlying hardware: DSP, ARM user and ARM kernel space

Master Thread Key Activity

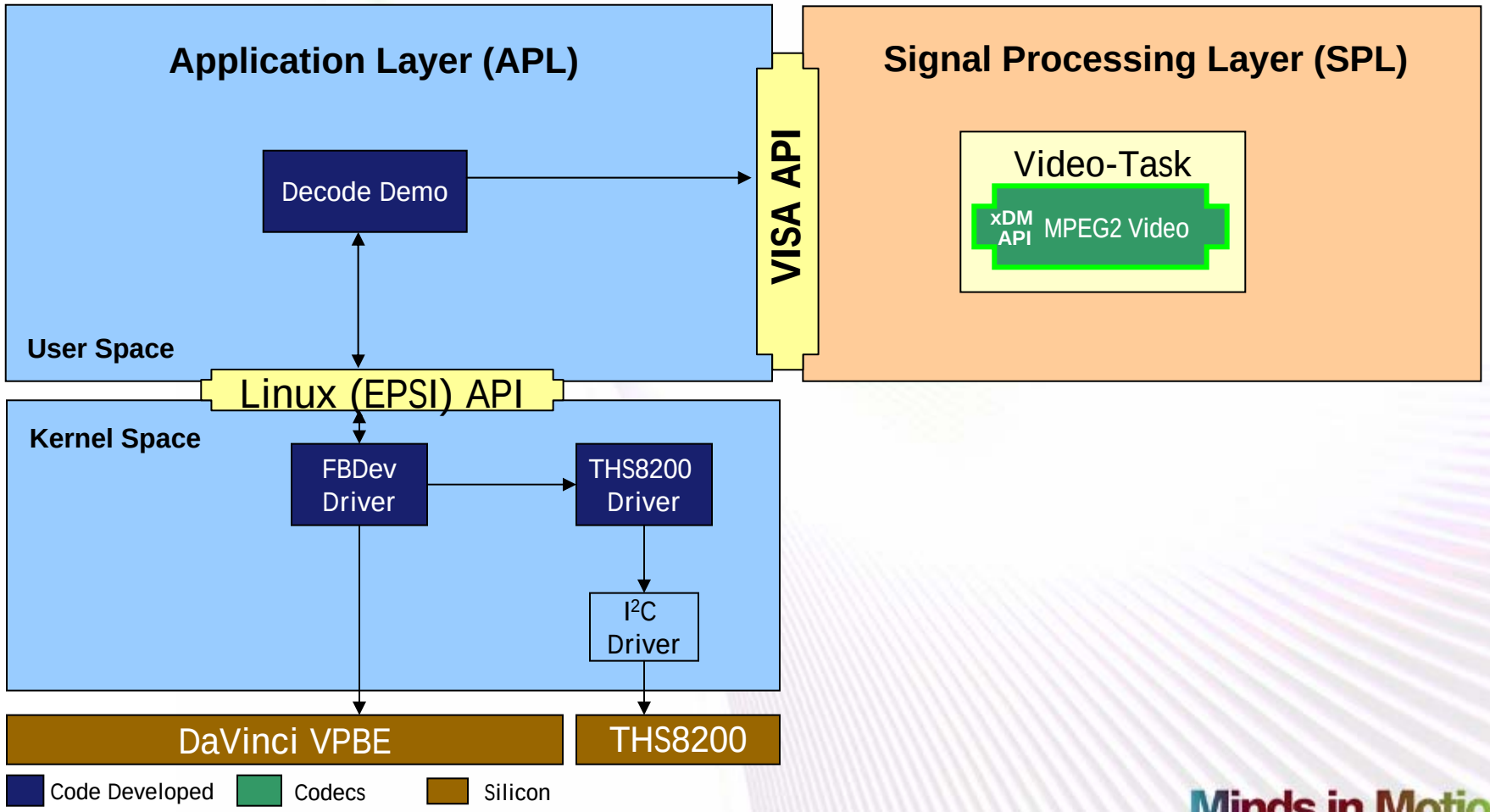
```
ifilefd = open("./fname", O_RDONLY);
odevfd = open("/dev/fb/x", O_WRONLY);
ioctl(odevfd, CMD, &args);
myCE = Engine_open("dec_disp", myCEAttrs);
myVD = VIDDEC_create(myCE,"viddec",params);

while( doPlayVideo == 1 ) {
        read(ifilefd, &rd, sizeof(rd));
        VIDDEC_control(myVD, ...);
        VIDDEC_process(myVD, ...);
        write(odevfd, &wd, sizeof(wd));
}
 close(odevfd);
 close(ifilefd);
 VIDDEC_delete(myVD);
 Engine_close(myCE);
```

// Create Phase
// get input file
// get output device
// initialize IO devices
// prepare VISA environment
// prepare to use video decoder

// Process phase
// read/swap buffer with input file
// option: perform VISA std algo ctrl
// run algo with new buffer
// pass results to output device

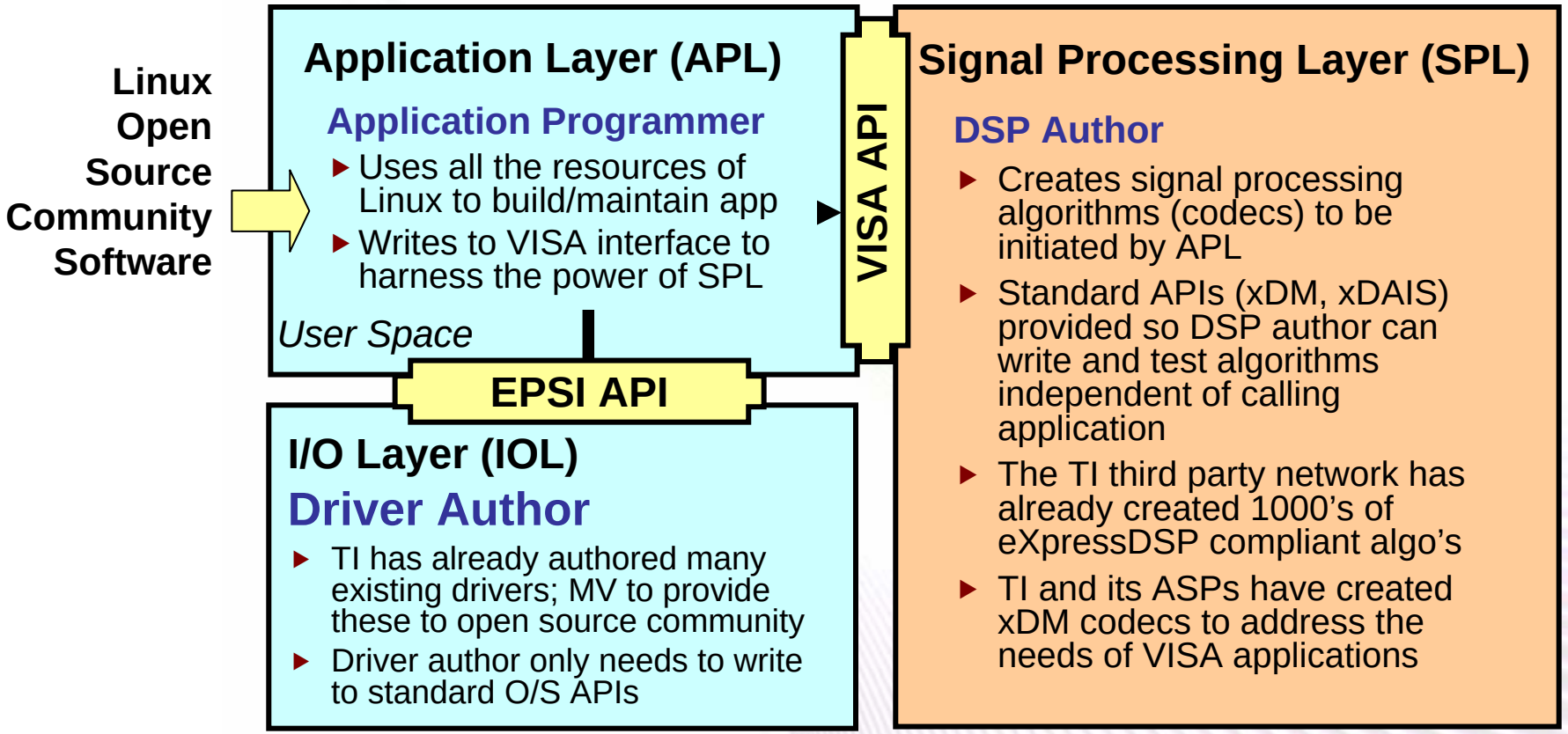
// Delete phase
// return IO devices back to OS
// algo RAM back to heap
// close VISA framework

Software Overview



Minds in Motion

Software Roles



- **Three primary developer roles exist: Application, DSP, and Driver authors**
- A fourth role – System Integrator – this role is simplified by the clearly defined APIs.

Linux Frame Buffer Driver (FBDev)

1. FBDev Driver Overview
2. Original FBDev Driver Block Diagram
3. FBDev Driver with HD Support
4. Run-time Configuration Switching

Linux Driver API

- There are 5 system calls normally supported by character devices such as the Frame Buffer Linux Display driver (FBDev).
  - open(), close(), read(), write(), and ioctl()
- Secondary system calls: mmap() and munmap()

Linux Driver API

open(char *pathname, int flags)

- open or create a file
- *pathname* is the device file to be opened
- *flags* are the actions performed on open
  - may be O_RDWR, O_RDONLY, or O_WRONLY
- open returns a positive integer the file descriptor used for all further accesses, or '-1' on error.
- on error *errno* will be set to indicate the cause.

Linux Driver API

close(int fd)

- close the file associated with file descriptor *fd*
- close returns ‘0’ on success, or ‘-1’ on error
- *errno* will be set to indicate the cause of the error

Linux Driver API

read(int fd, void *buf, size_t count)

- read a maximum of *count* bytes from the file associated with *fd* into buffer *buf*
- read returns the number of bytes read or on error '-1'
- on error, *errno* will be set to indicate the cause.

Linux Driver API

write(int fd, void *buf, size_t count)

- write *count* bytes from buffer *buf* to the file referenced by *fd*
- return the number of bytes written, or '-1' if there was a write error
- on error, *errno* will be set to indicate the cause.

Linux Driver API

ioctl(int fd, int request, …)

- ioctl performs the device specific control functions on the special file *fd*
- *request* is the device specific command
- the third argument is commonly a pointer to a command specific data structure, technically it may be of any form, and is often denoted simply as *arg*
- return ‘-1’ on error, or ‘0’ for success
- on error, *errno* will be set to indicate the cause.

Linux Driver API

mmap(void *start, size_t length, int prot,
int flags, int fd, off_t offset)

- map *length* bytes starting at *offset* from the file *fd*, preferably starting at *start*
- the actual start of the mapping is returned by mmap()
- *prot* and *flags* are described in the main page
- mmap returns a pointer to the mapped memory or '-1' on error
- on error, *errno* will be set to indicate the cause.

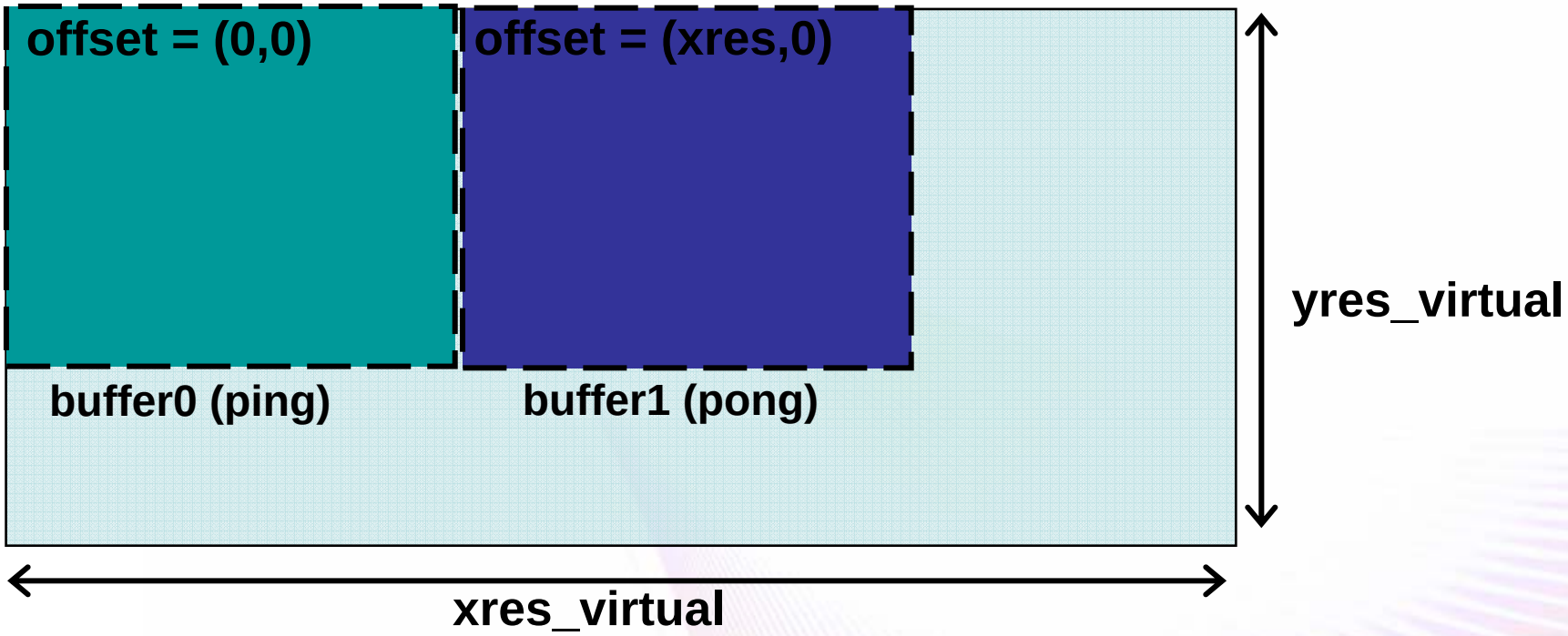
munmap(void *start, size_t length)

- munmap deletes the mapping starting at *start*
- returns '0' for success, '-1' on error
- on error, *errno* will be set to indicate the cause.

FBDev Driver Overview: User APIs

- Main file operations
  - Framebuffer device node: /dev/fb/x
    - open() & close()
    - read() & write()
    - ioctl()
      - FBIOGET_VSCREENINFO
      - FBIOPUT_VSCREENINFO
      - FBIOGET_FSCREENINFO
      - FBIOPAN_DISPLAY
      - FBIO_WAITFORVSYNC
      - FBIO_SETATTRIBUTE
      - FBIO_SETPOSX
      - FBIO_SETPOSY

FBDev Driver Overview



- Use FBIOPUT_VSCREENINFO to set BPP, Window size/position
- Use FBIOPAN_DISPLAY to ping-pong
- Use FBIO_WAITFORSYNC to block until current buffer completes

FBDev Driver Overview: User APIs

- FBIOPUT_VSCREENINFO
  - Called when user passes a structure to set the parameters such as:
    - X and Y resolution,
    - margins on the top, bottom, left and right
    - Bits per pixel mode and size and offset of color components in 1 pixel data
  - Checks that the parameters are acceptable
  - Changes register settings to affect the change
- FBIOGET_VSCREENINFO
  - Copies out to user the current fb_var_screeninfo structure
- FBIOGET_FSCREENINFO
  - Returns the “fixed” information about the framebuffer such as the start and length of framebuffer memory, length of a line, flags for hardware accelerations, if any.

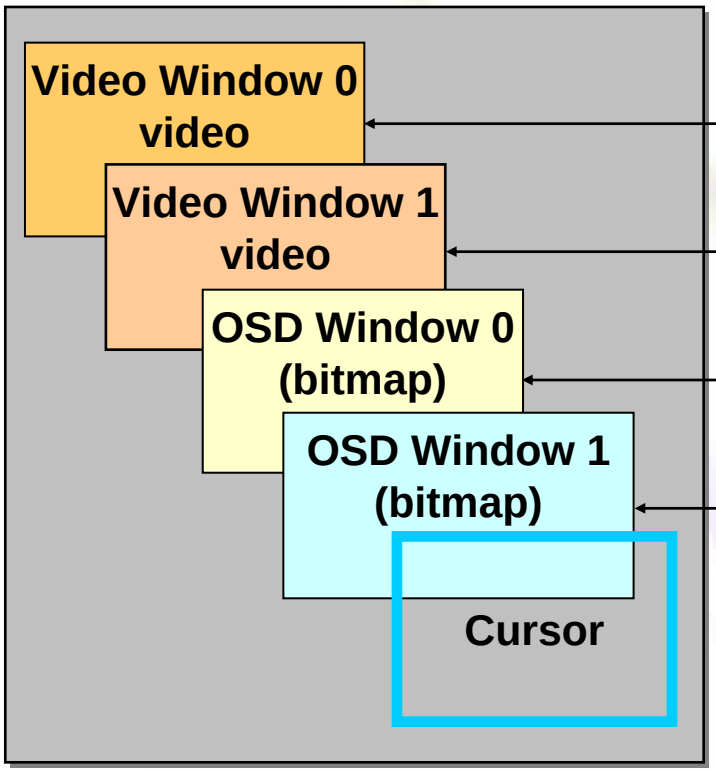
FBDev Driver Overview: User APIs

- FBIOPAN_DISPLAY
  - To view a particular portion of the framebuffer memory
  - Can be used for double/triple buffering
    - By allocating virtual framebuffer size twice/thrice the required (screen) size and viewing two halves alternately
- FBIO_WAITFORVSYNC
  - Call made by applications to block until the next vsync – guarantees that a frame has been displayed.
- FBIO_SETATTRIBUTE
  - Ioctl for setting the attribute blending levels
- FBIO_SETPOSX ioctl
  - To set the X position of the window
- FBIO_SETPOSY ioctl
  - To set the Y position of the window

DaVinci FBDev Driver Overview

- Features
  - Display Image On Screen
    - RGB16 on the OSD0 window (/dev/fb/0)
    - YCbCr 4:2:2 on the VID0 window (/dev/fb/2)
    - YCbCr 4:2:2 on the VID1 window (/dev/fb/3)
    - Control OSD0 blending via the Attribute window (/dev/fb/2)
    - Using read/write or mmap
    - Supports NTSC / PAL Resolution Only
  - Set / Query Parameters
    - Window size
    - Bits-per-pixel
    - OSD window 0 size/position
    - Video window 1 size/position

DaVinci FBDev Driver Overview



/dev/fb/1 YCbCr 4:2:2

/dev/fb/3 YCbCr 4:2:2

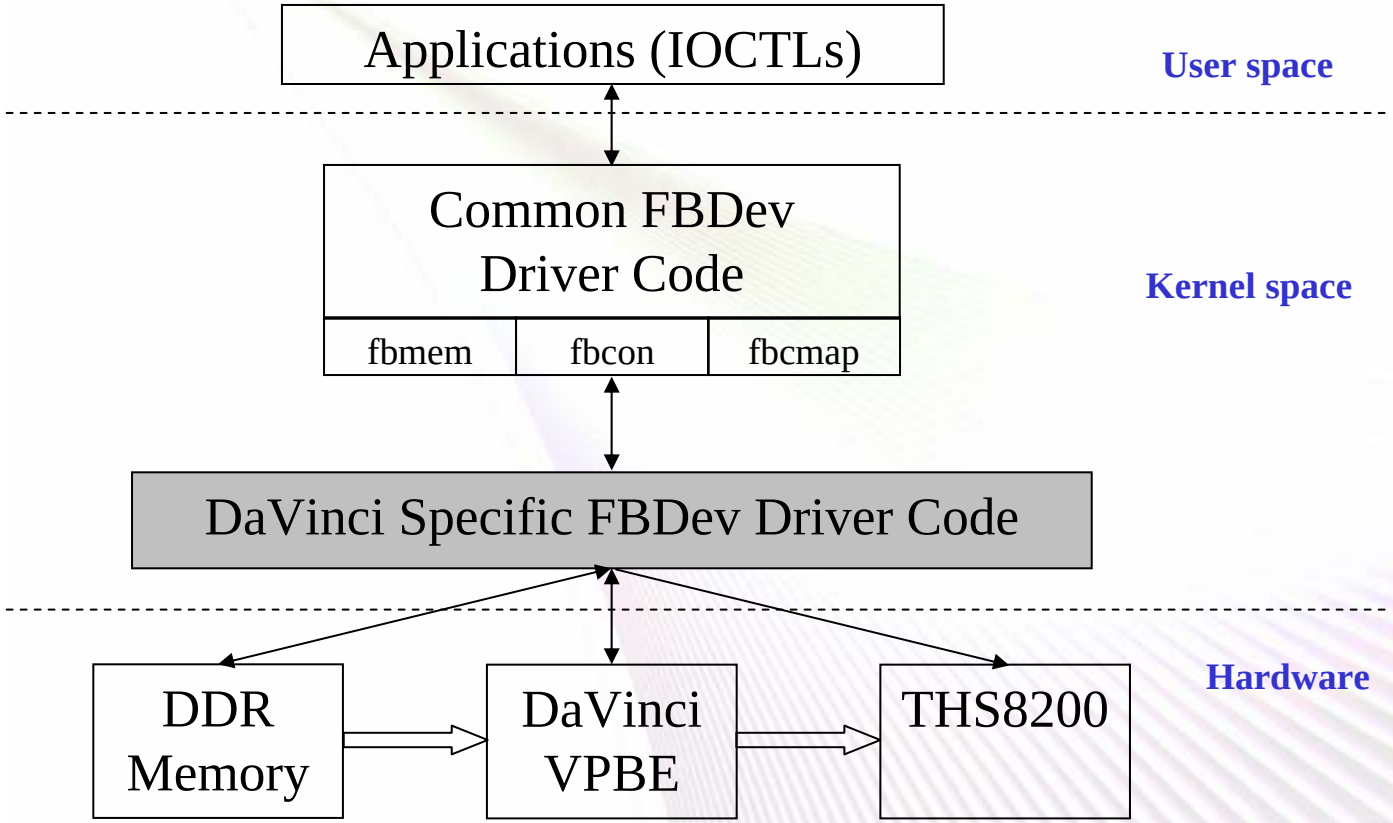
/dev/fb/0 RGB16

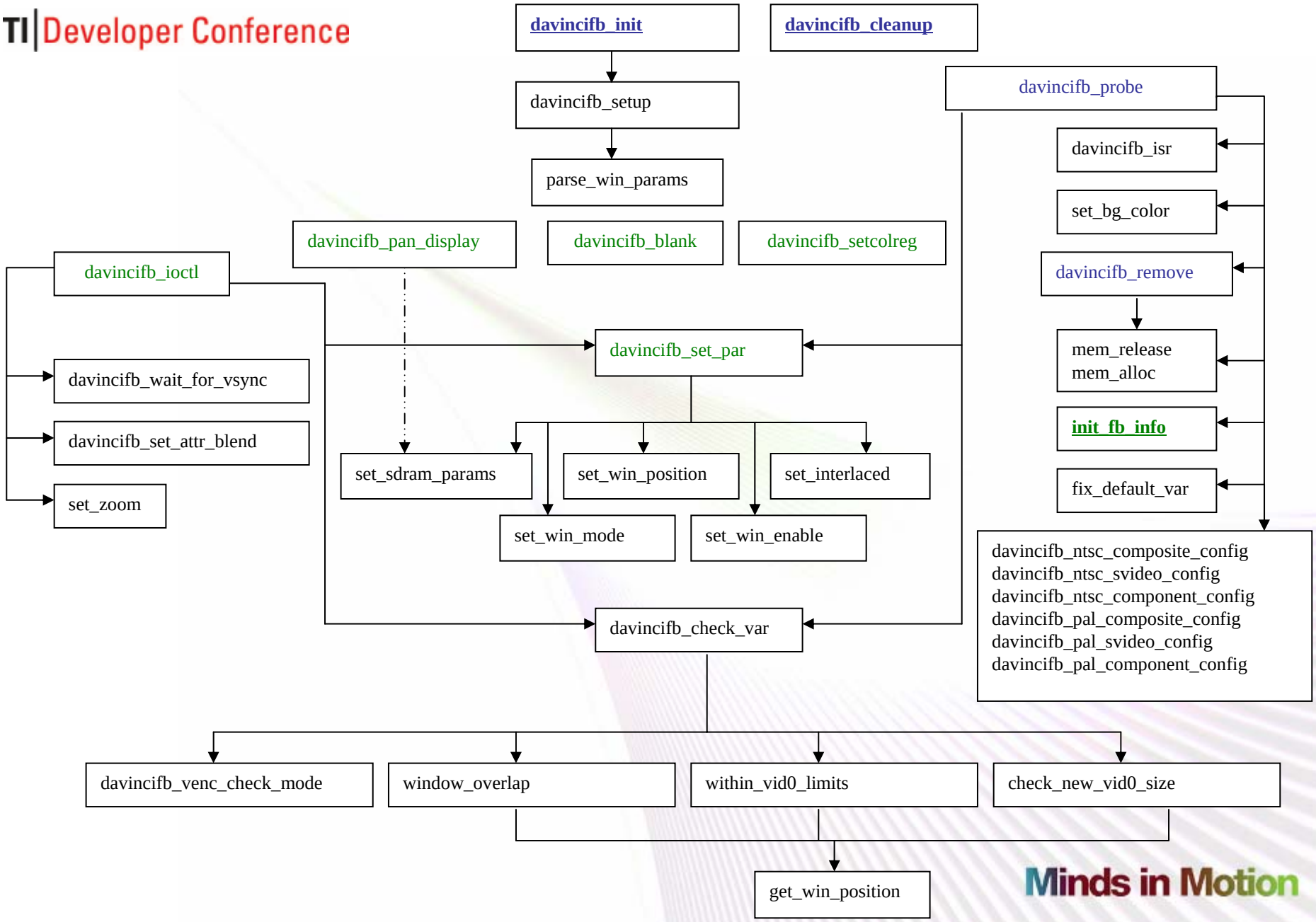
/dev/fb/2 attribute (blending)

FBDev Driver Overview: source files

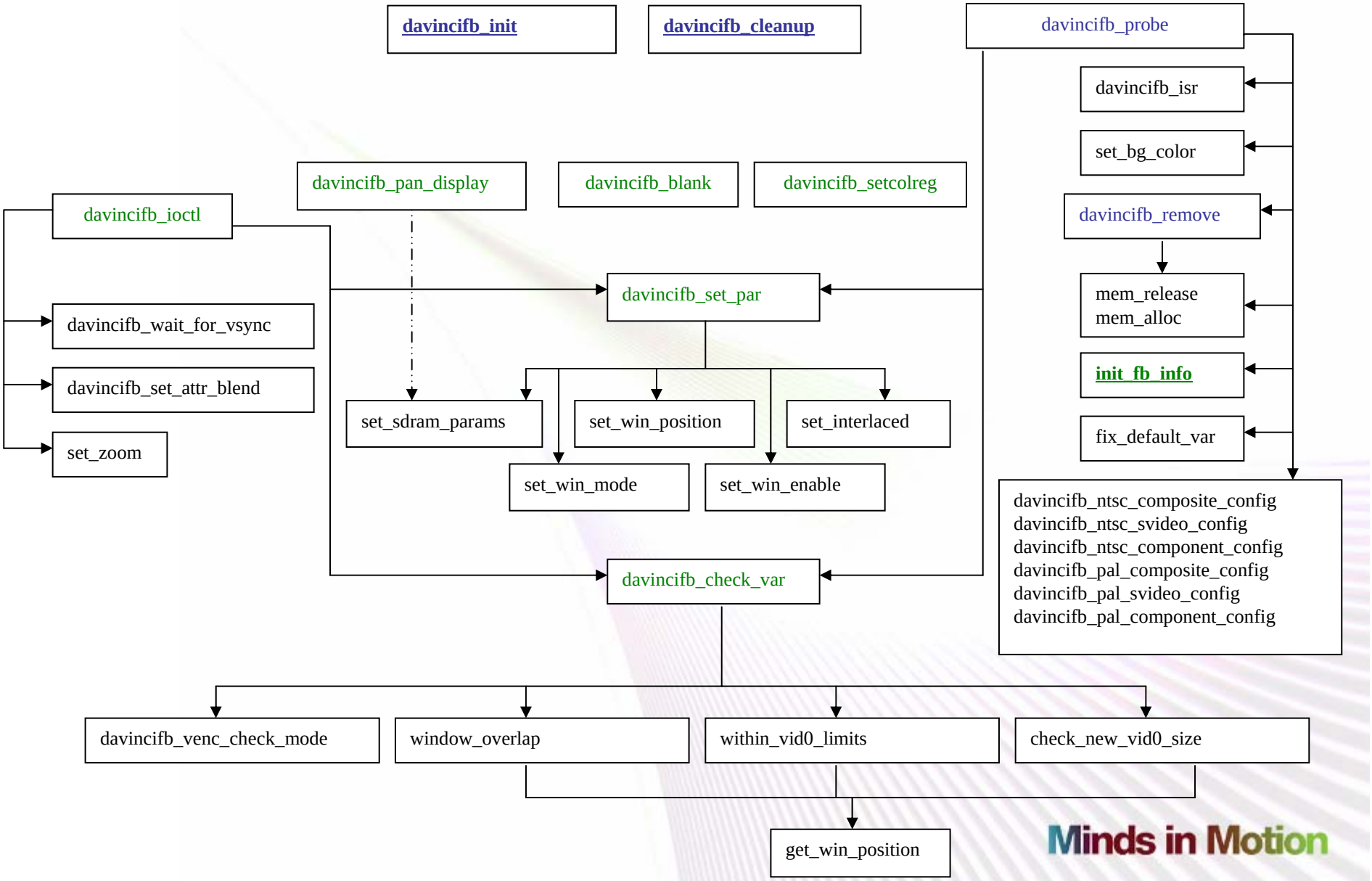
- DaVinci Specific FBDev Driver Files
  - drivers/video/davincifb.c
  - drivers/video/davincifb.h
- Other Files
  - include/linux/fb.h
  - drivers/video/fbmem.c
  - drivers/video/fbcon.c
  - drivers/video/fonts.c

FBDev Driver Overview
Software Architecture

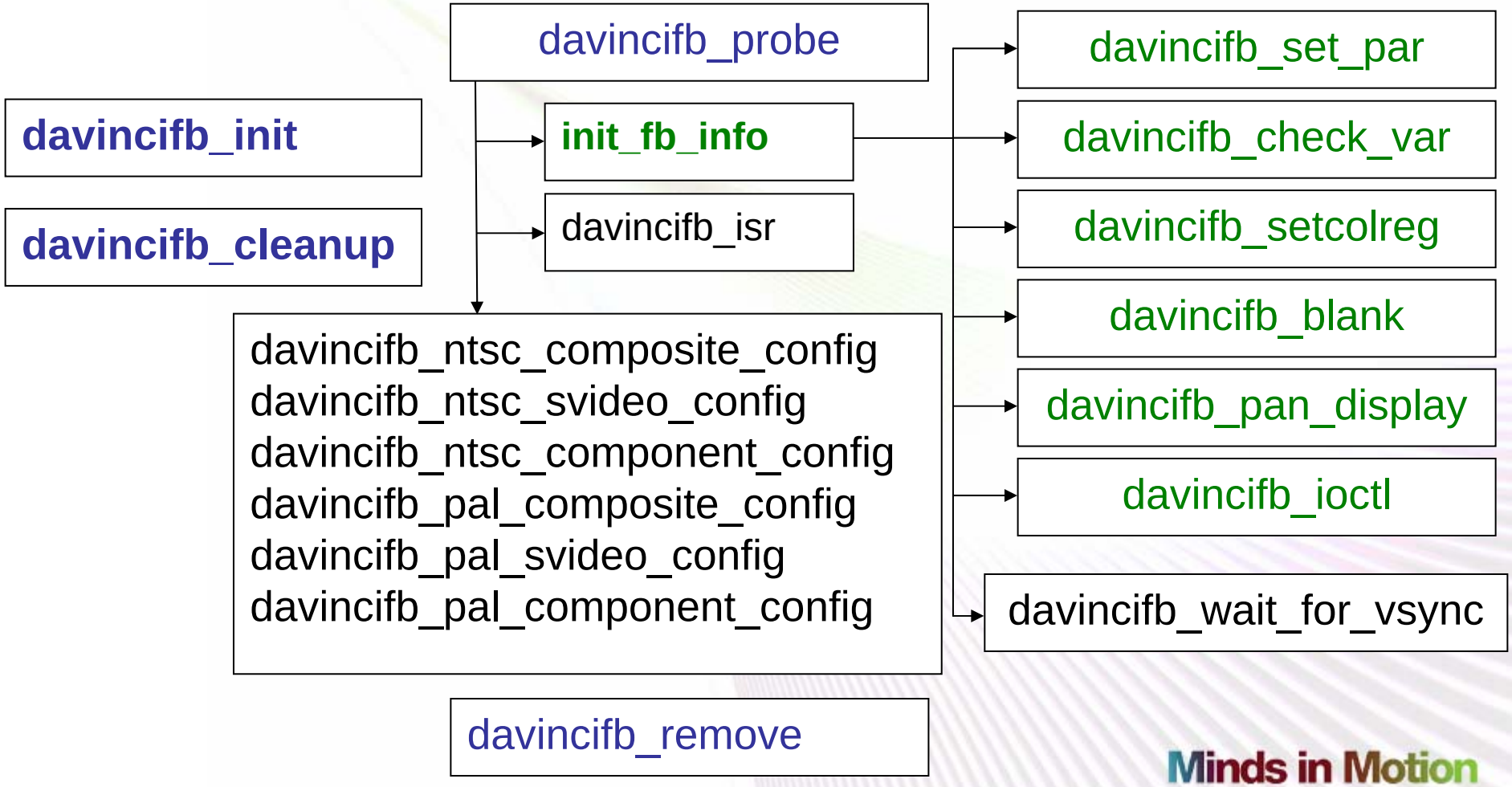



davincifb_init
davincifb_cleanup
davincifb_setup
davincifb_probe
parse_win_params
davincifb_isr
set_bg_color
davincifb_pan_display
davincifb_blank
davincifb_setcolreg
davincifb_ioctl
davincifb_remove
davincifb_set_par
mem_release
mem_alloc
davincifb_wait_for_vsync
init_fb_info
davincifb_set_attr_blend
fix_default_var
set_sdram_params
set_win_position
set_interlaced
set_zoom
set_win_mode
set_win_enable
davincifb_ntsc_composite_config
davincifb_ntsc_svideo_config
davincifb_ntsc_component_config
davincifb_pal_composite_config
davincifb_pal_svideo_config
davincifb_pal_component_config
davincifb_check_var
davincifb_venc_check_mode
window_overlap
within_vid0_limits
check_new_vid0_size
get_win_position

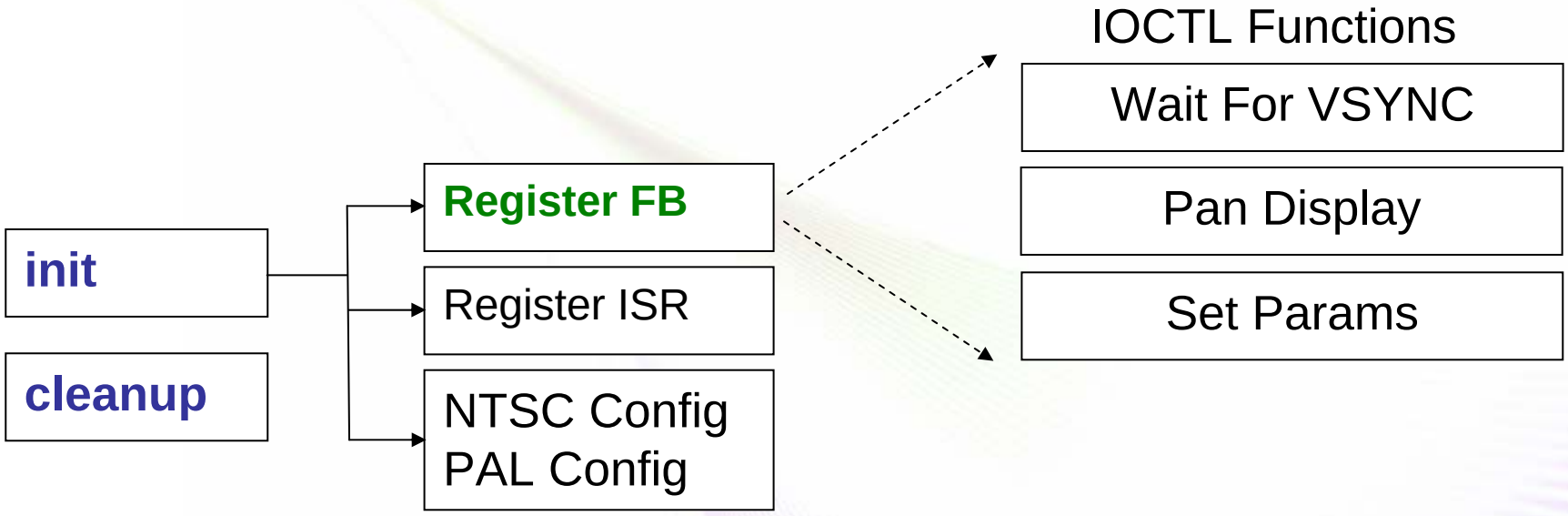
Original FBDev Driver



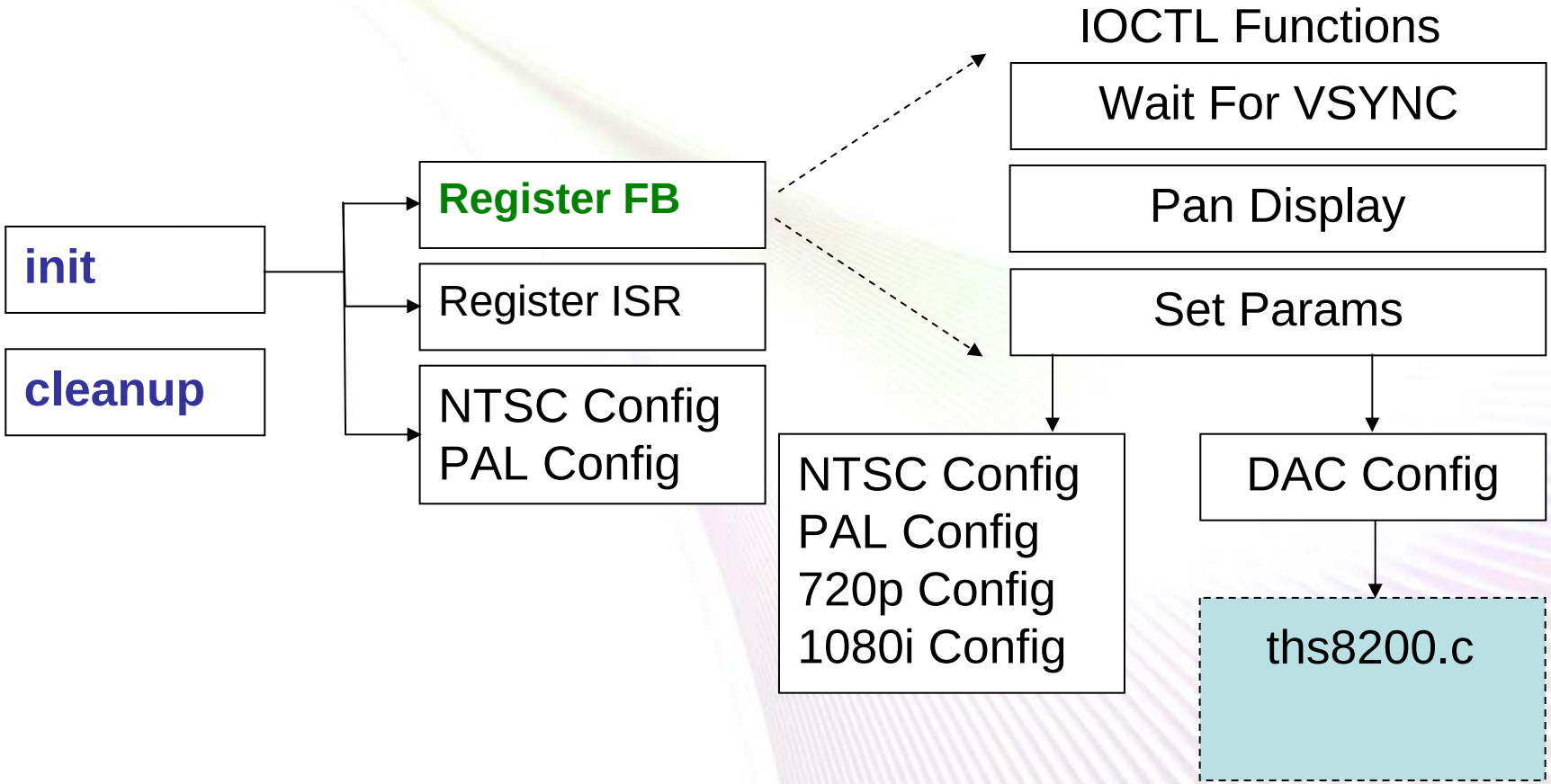
Original DaVinci FBDev Driver



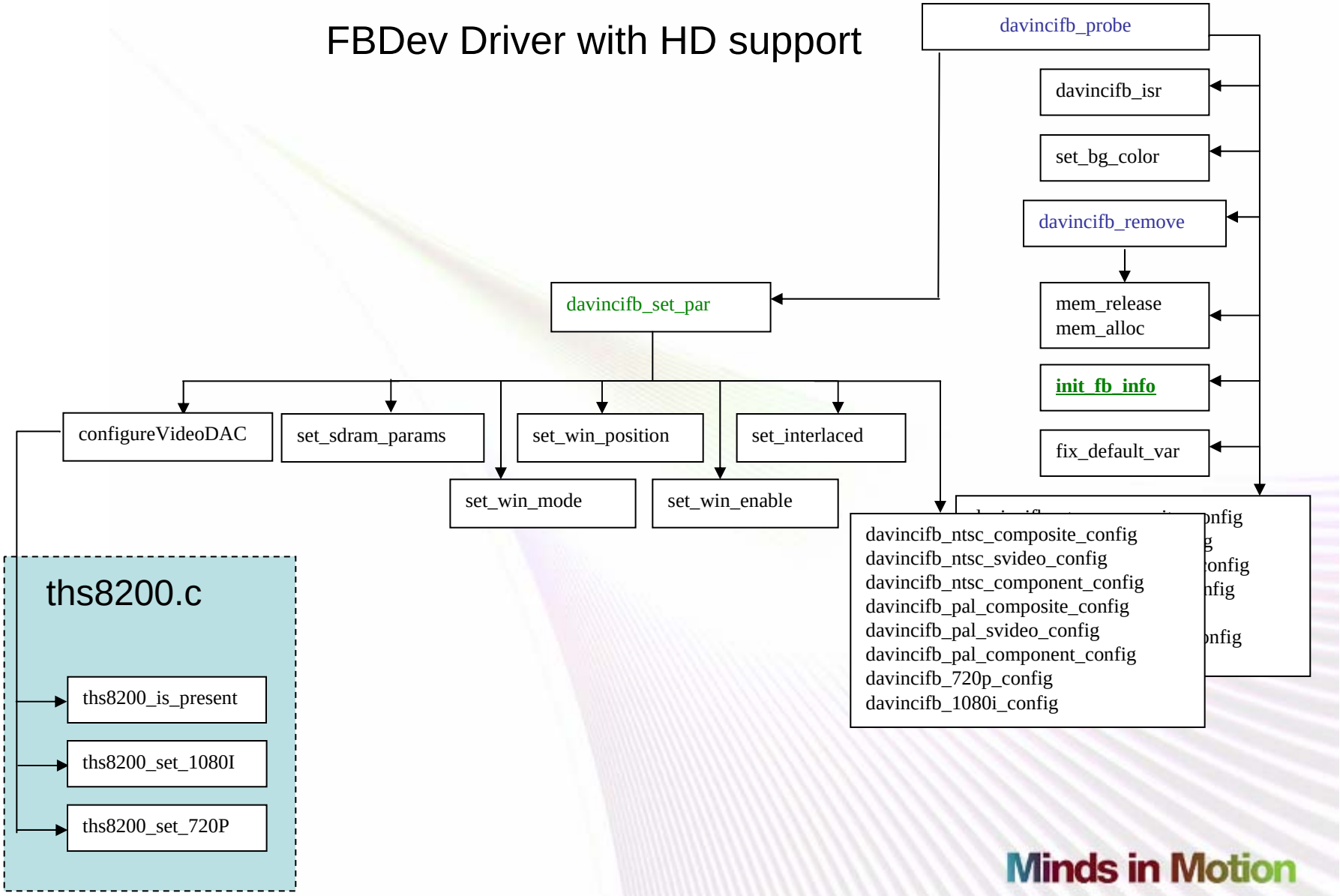
Original DaVinci FBDev Driver



FBDev Driver with HD Support



FBDev Driver with HD support



Run-time Configuration Switching

- Modifications to DVEVM Decode demo
  - Bigger buffers to handle HD resolutions
  - Resolution config via '-r' command line option
    - ./decode_hd -l -r 720p -v 720p.m2v
    - ./decode_hd -l -r 1080i -v 1080i.m2v

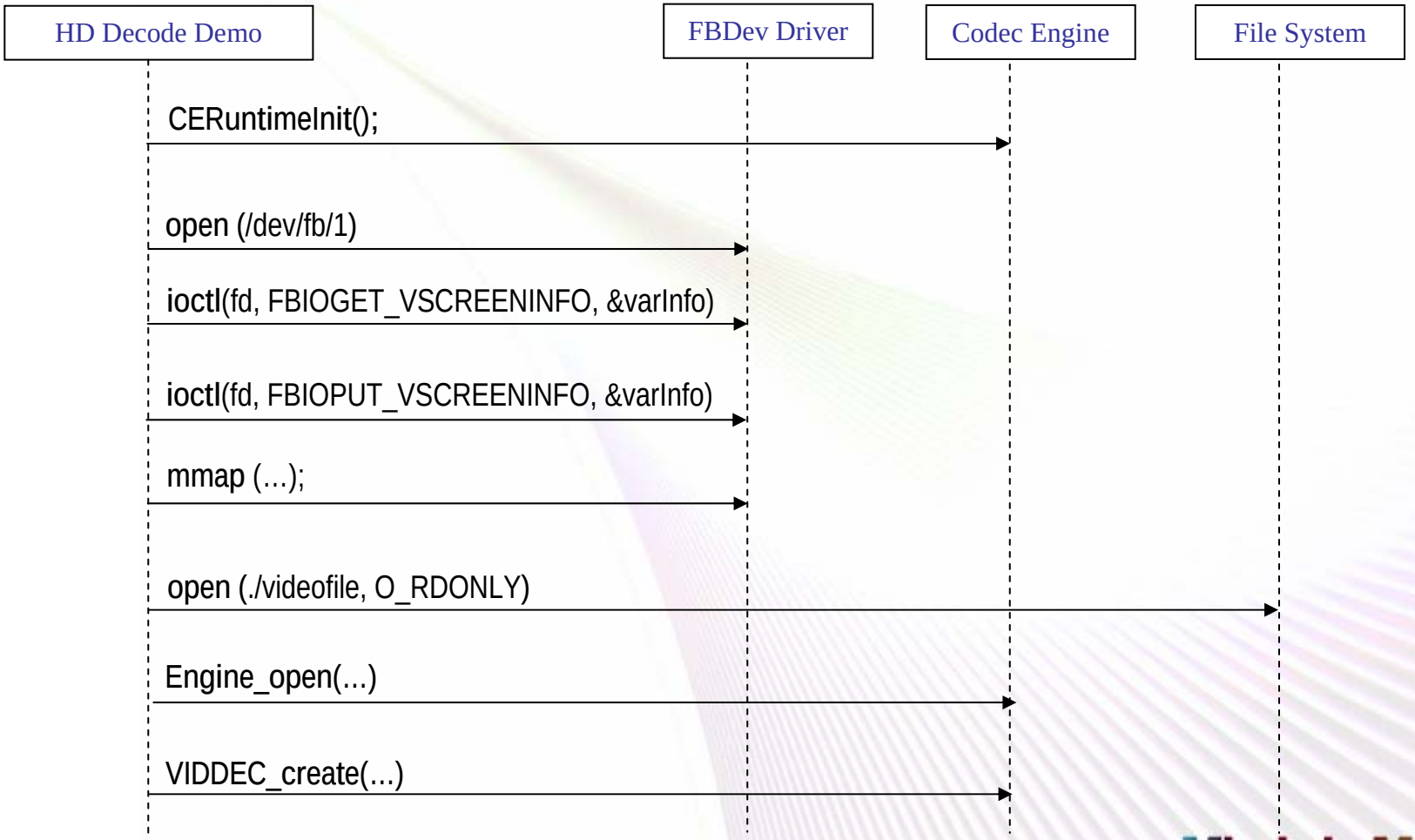
DaVinci HDTV Using THS8200

1. Digital Video
2. Hardware Overview
3. Software Overview
4. HD Video Demo

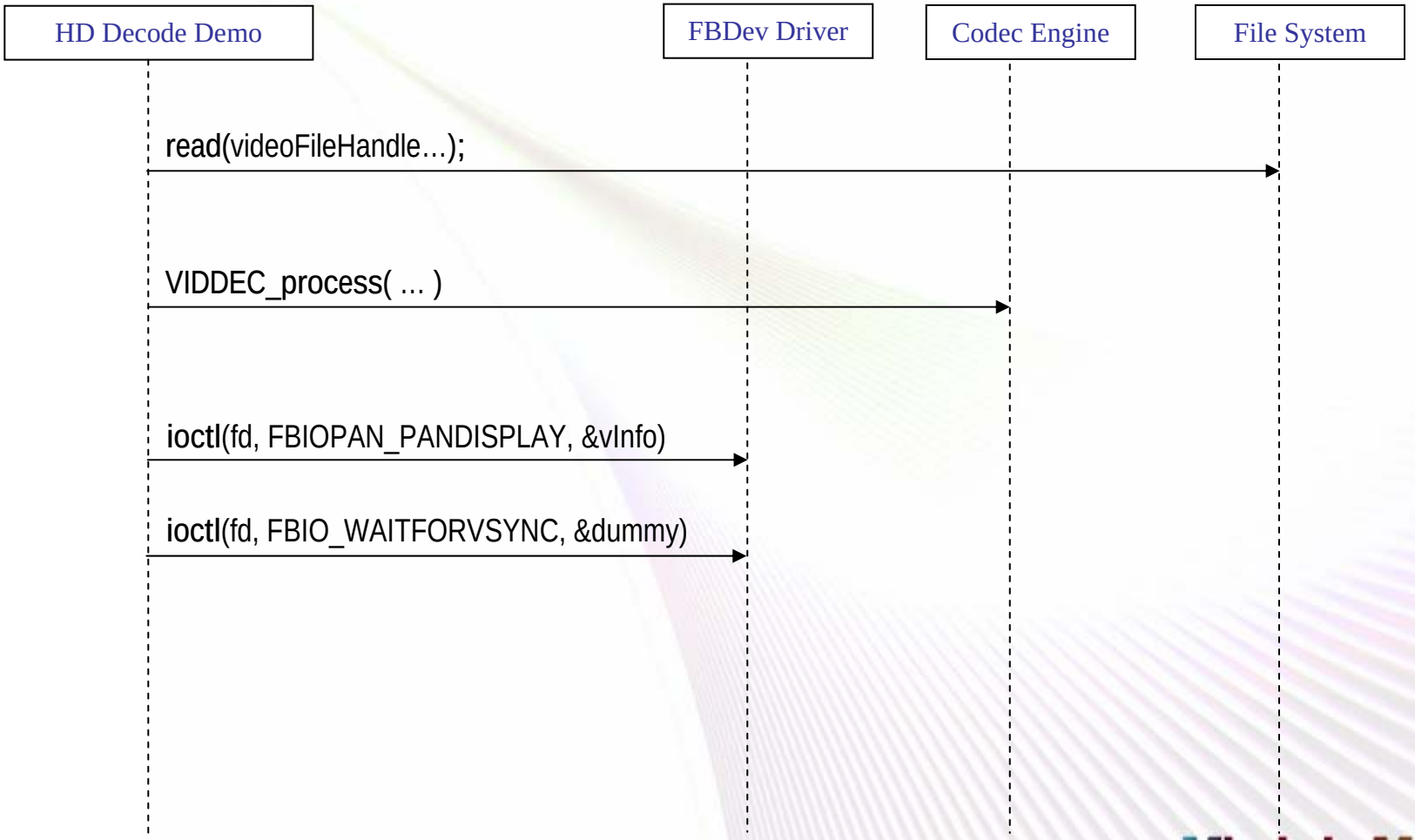
HD Video Demo

- HD Video Decode Demo – 15 min
  - System Block Diagram
  - Software Block Diagram
  - Demo

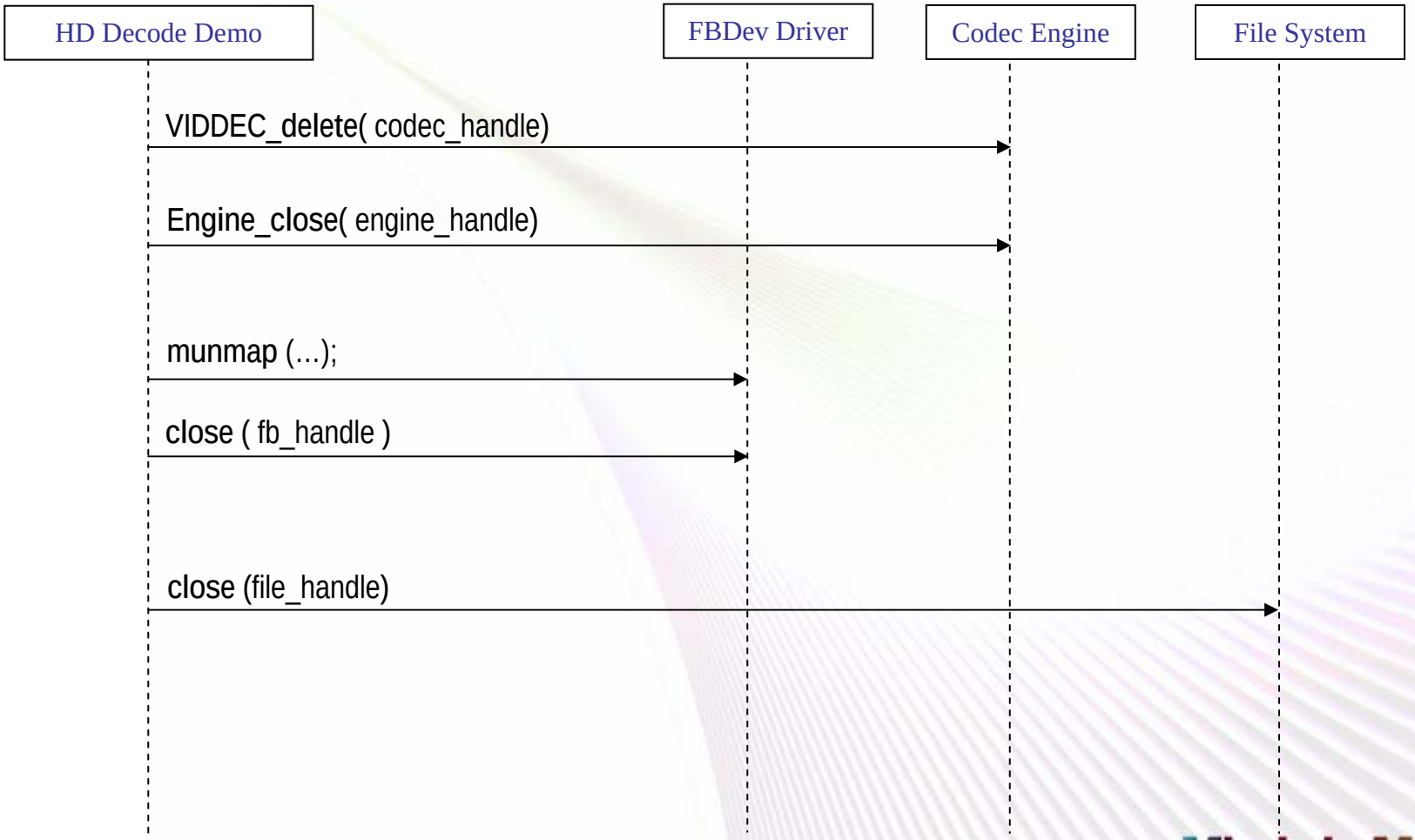
HD Decode Demo: Init Phase



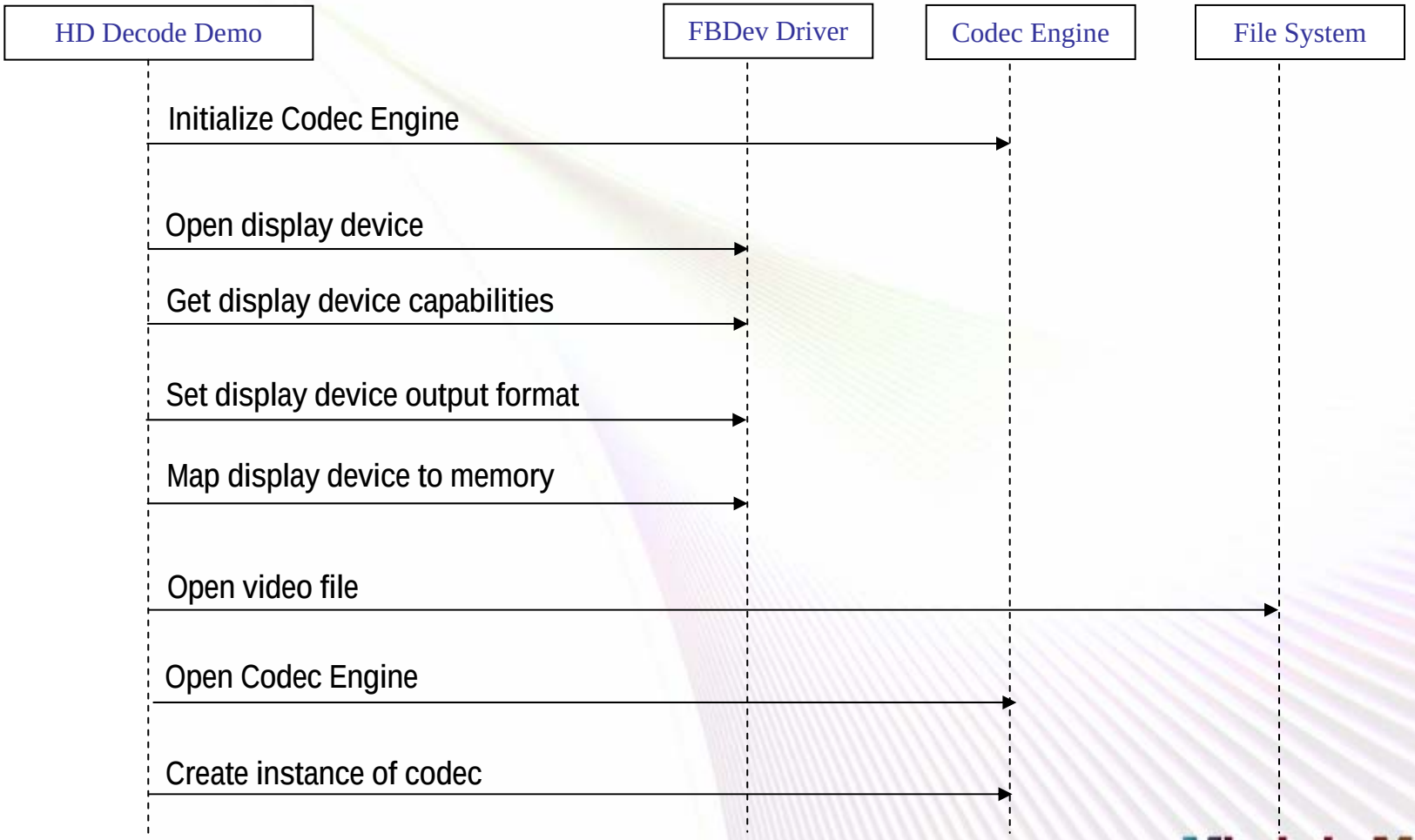
HD Decode Demo: Process Phase



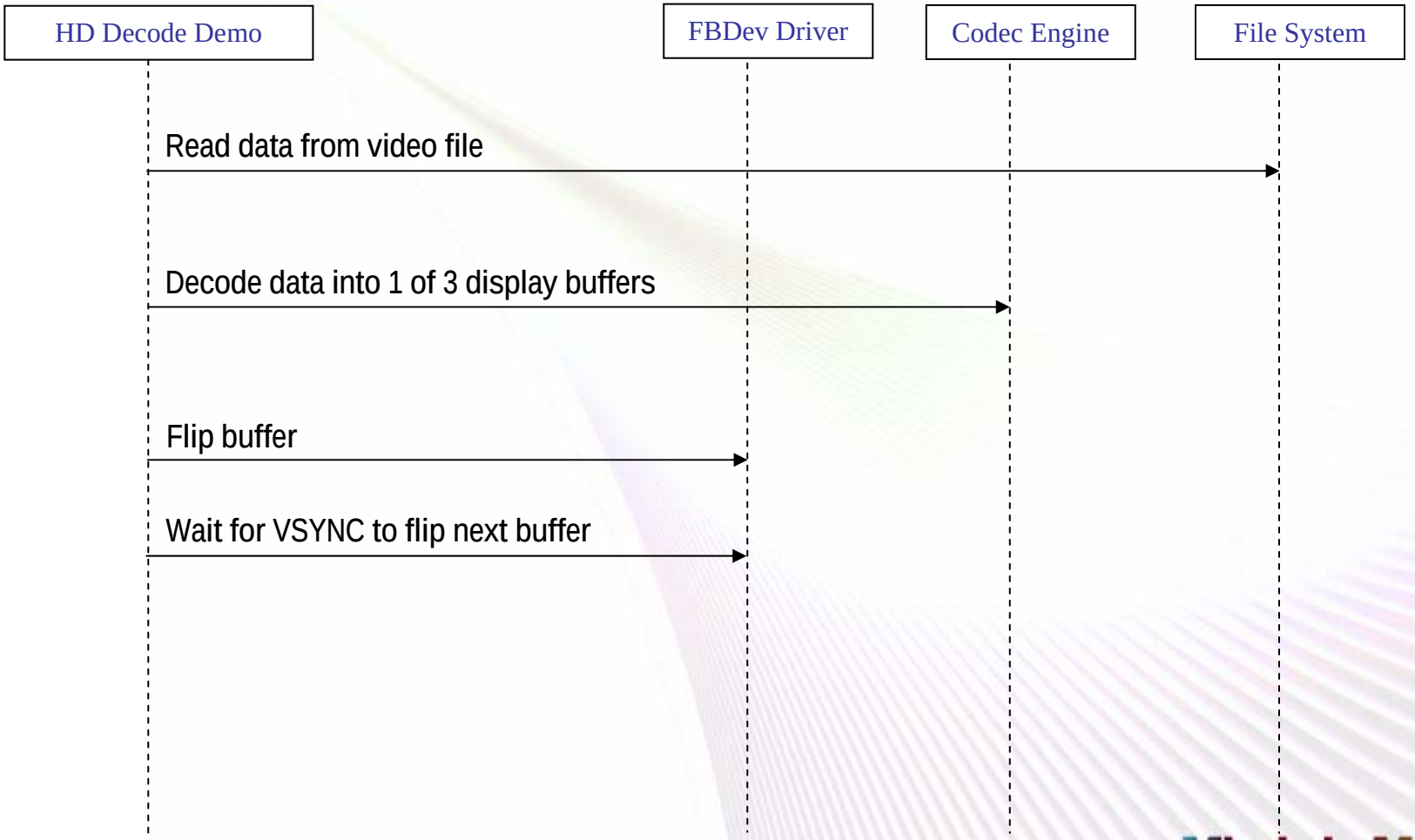
HD Decode Demo: Delete Phase



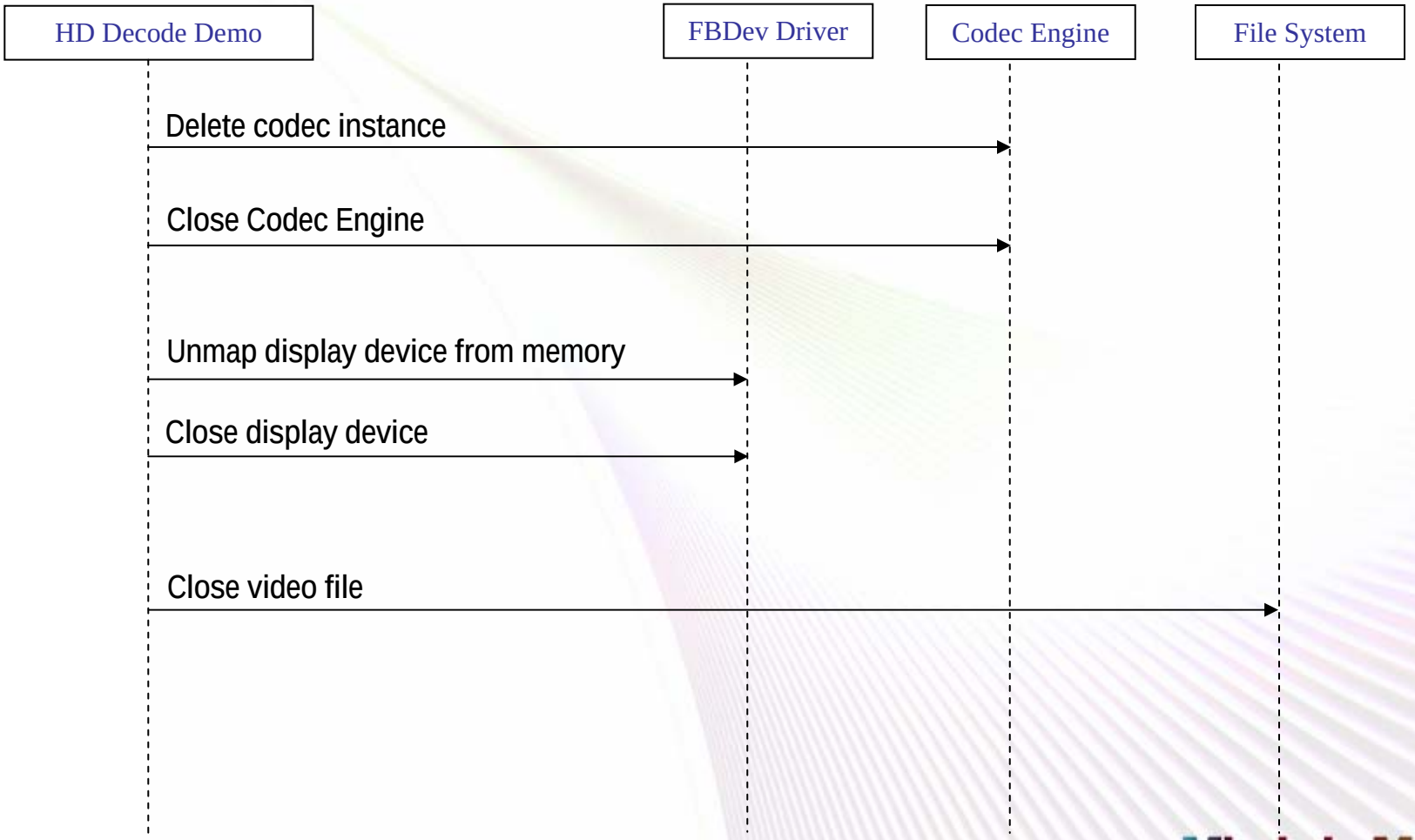
HD Decode Demo: Init Phase



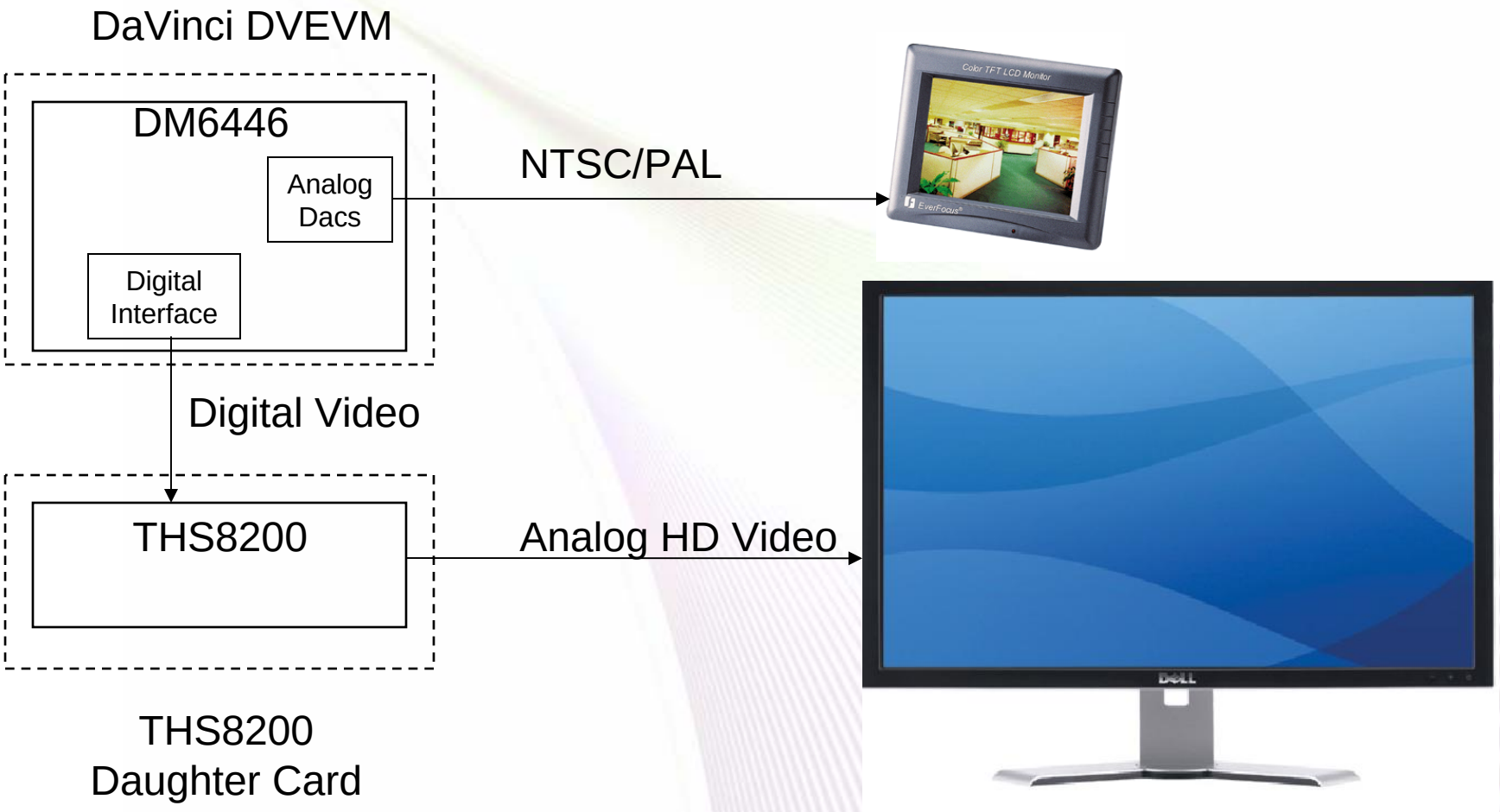
HD Decode Demo: Process Phase



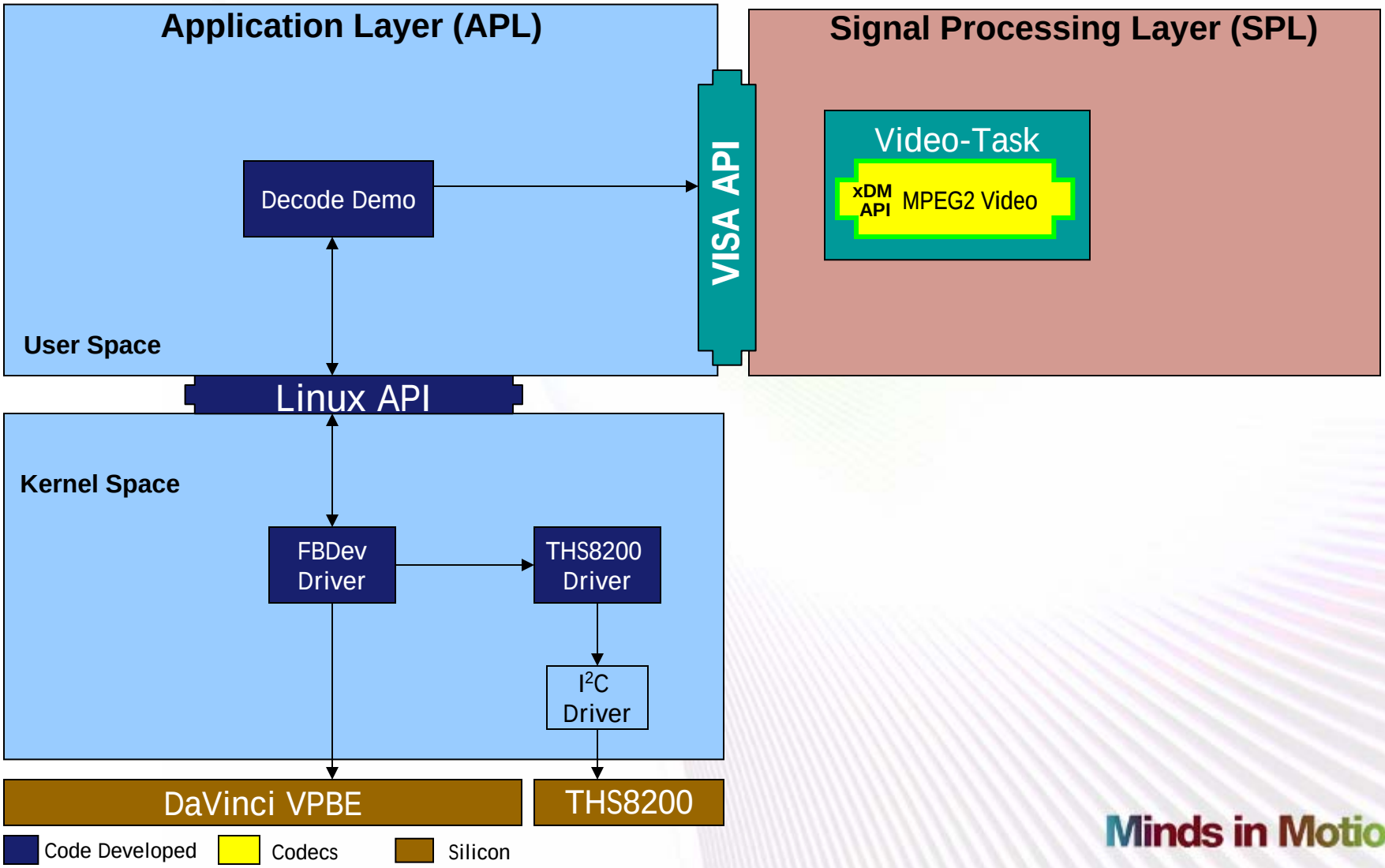
HD Decode Demo: Delete Phase



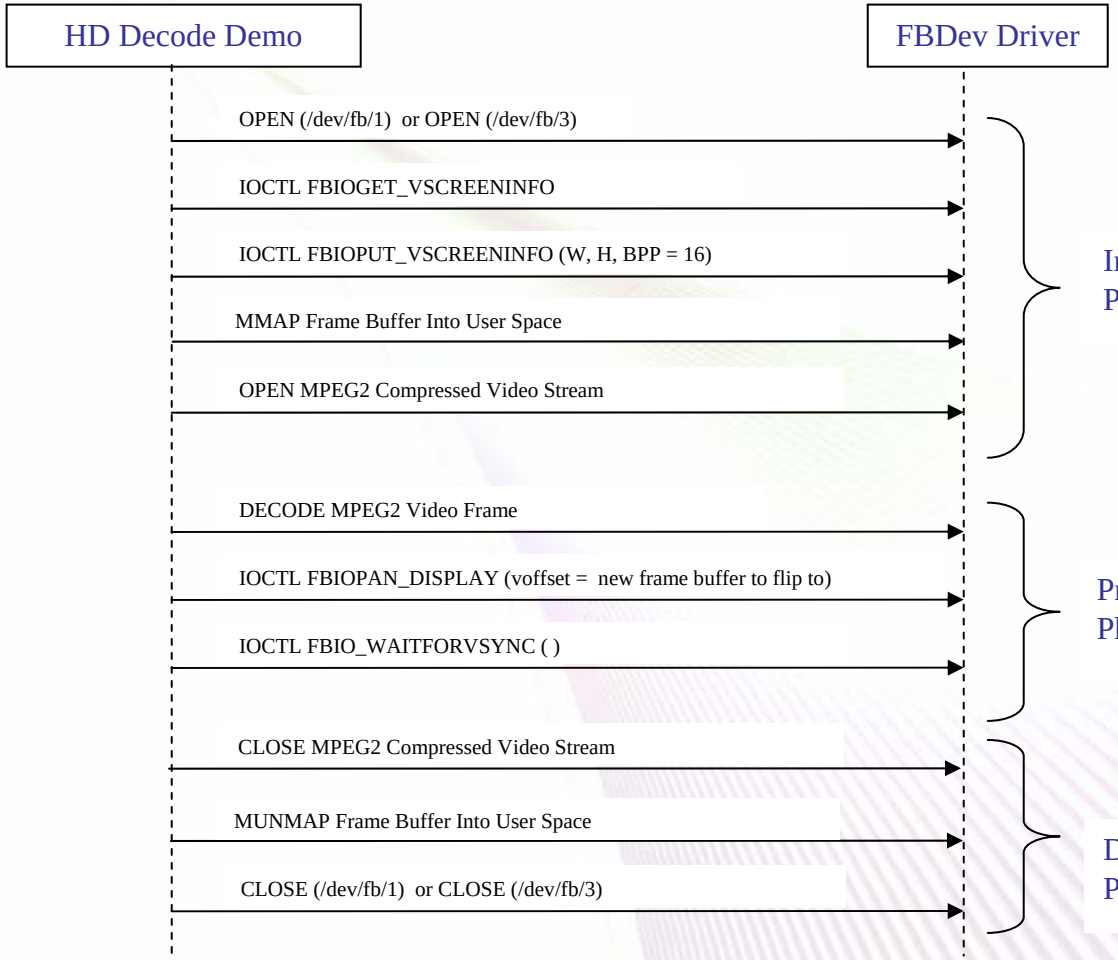
HD Decode Demo: Block Diagram



Software Overview



HD Decode Demo



TI Developer Conference
March 7-9, 2007 • Dallas, TX

DaVinci HDTV using THS8200

Neal Frager and Juan Gonzales
Digital Costumer Applications Team
Texas Instruments

Minds in Motion