Project Report

Of

JPEG Decoder

And

8-color Photo Viewer

Class: Embedded System Design

By Prof. Stephen Edwards (sedwards@cs.columbia.edu)

GROUP MEMBERS of GROUP 2 (in TA's list):

<ktchiao@yahoo.com> <Steve Chiao>

<seantang.mal@gmail.com> <Sean Tang>

<steelyang@gmail.com> <Steel Yang>

1. Proposal

TITLE: “JPEG Decoder and 8-color Photo Viewer”

SUMMARY: “We will build a photo viewer which can display 640*480 photos in baseline JPEG format or in 8-color format”

DESCRIPTION:

1.1 This JPEG decoder can download, decode and display baseline JPEG (out of 14 JPEG formats) pictures, and 8-color photo viewer can download 640*480 images in 8-color format from PC to Digilent external SRAM and display them on the 8-color monitor..

1.2 C-code programming part I: JPEG Decoder

- 1.2.1 Input image: byte-by-byte input through RS232 to external SRAM and convert from text characters to byte numbers.
- 1.2.2 Input defragmentation: frame/marker decoding and DQ/DH table setup.
- 1.2.3 VLD: Variable length decoding
- 1.2.4 ZZ: ZigZag scan
- 1.2.5 DQ: De-quantization
- 1.2.6 IDCT: Integer inverse discrete cosine transformation. (Optional – hardwired IDCT module)
- 1.2.7 Colorspace conversion: From YCbCr to RGB colorspace
- 1.2.8 Reordering: Re-order the bitstream in a line-by-line fashion
- 1.2.9 Color display mapping: map 24-bit RGB colorspace down to 8-color display in C
- 1.2.10 Output image: store decompressed 8-color image data into external memory
- 1.2.11 Display image: display image data from external memory to 8-color monitor.

1.3 C-code programming part II: 8-color Photo Viewer, from PC to 8-color monitor through Digilent board.

1.4 VHDL programming part: we need to create a new PCORE called video_emc_controller in VHDL format including 4 major functions:

1.4.1 Video controller: It can display the 640*480 image onto 8-color monitor, so basically it informs video buffer to get next image data to display when current data in its shift register has been used/shifted out, and video controller automatically load in next image data assuming video buffer has gotten the correct one already at the right timing.

1.4.2 Video buffer: Video buffer is the bridge for video controller to get image data from external SRAM, so when video controller asks for data, video buffer activates an OPB read cycle playing as an OPB master device. When it gets the correct data back, it stores the data in its own buffer waiting video controller to get them.

1.4.3 Mode master select: Since for external memory/SRAM being an OPB slave device, it may either have an OPB master access from external OPB bus entering into video_emc_controller, or have an internal OPB master access from video buffer, mode master select is used to separate two different master access. It's done by an internal mode register to tell the difference. Mode register is also mapped and saved into external SRAM in address 0xc00fffff (so it's like a write-through in cache). In mode register, lowest 3 bits are used to indicate video display (internally) or host SRAM access (externally), video display mode is the default mode, and we have next lowest 3 bits as force mode to force it go into host SRAM access mode for JPEG image downloading or JPEG image decoding.

1.4.4 External memory controller (EMC): External memory controller is an OPB slave device for OPB master to access (read or write), so it needs to behave as an OPB slave device, and being able to read & write SRAM in the right timing as well..

1.5 HW/SW system integration: Correctly integrate video_emc_controller PCORE into uBlaze system, and get correct C code running. (Professor will help this part due to tight project schedule.)

1.6 (Optional) Add selectable photo effects (C-code programming efforts) and/or add hardware accelerators for instruction/data intensive parts (VHDL programming efforts) if we have extra time.

RESULTS: (MILESTONE with DEMO)

MILESTONE 1: 8-color Photo Viewer – 8-color image is downloaded from PC to Digilent external SRAM through RS232, and display the resulting image onto 8-color monitor.

MILESTONE 2: JPEG Decoder with pre-stored image – Decode pre-stored small JPEG image (in BRAM), save de-compressed 8-color image into Digilent external SRAM, and display it onto 8-color monitor.

MILESTONE 3: JPEG Decoder with input-from-PC image – First input JPEG image from PC to Digilent external SRAM through RS232 interface, then de-compress it and save it back to external SRAM, and then finally display it onto 8-color monitor.

2. Task breakdown

2.1 C-code programming part I: JPEG Decoder

No	Name	Description
T2.1.1	Input image (1.2.1)	1. Input image byte-by-byte through RS232 to uBlaze. 2. Image in text characters is converted to image in byte number, and saved into external SRAM on digilent board. 3. Another optional way is to make image pre-saved into external SRAM when downloading the system code. (After receiving EOF character or special last 2-byte pattern called MARKER in JPEG, disable RS232-to-external-memory path & activate JPEG decoder C-code automatically.)
T2.1.2	JPEG decoder C-code porting (1.2.2 – 1.2.8)	1. Port JPEG decoder C-code to uBlaze system so LIB, FILE I/O, & malloc have to be modified or removed. 2. LIB – multipliers & other operations. 3. FILE I/O – fi & fo related 4. Memory mapping – Allocated for intermediate data storage. 5. Hardwired IDCT I/O (optional if extra time exists) – connect to OPB with internal input/output registers 2*64 bytes, & special handler to import and export block.
T2.1.3	Color display mapping (1.2.9)	Map 24-bit RGB colorspace down to 8-color display in C-code.
T2.1.4	Output image (1.2.10)	Store decompressed 8-color image data into external memory.
T2.1.5	Display image (1.2.11)	Display image data from external memory to 8-color monitor. (Enable RS232-to-external-memory path for next JPEG image file download.)
T2.1.6	VHDL IDCT (Optional)	Port IDCT with internal input/output registers 2*64 bytes, & special handler to import and export block into VHDL hardwired module.

2.2 C-code programming part I: 8-color Photo Viewer

No	Name	Description
T2.2.1	Input image (1.2.1)	1. Input image byte-by-byte through RS232 to uBlaze. 2. Image in text characters is converted to image in byte number, and saved into external SRAM on digilent board.
T2.2.2	Display image (1.2.11)	Display image data from external memory to 8-color monitor. (Enable RS232-to-external-memory path for next JPEG image file download.)

2.3 VHDL programming part: we need to create a new PCORE called video_emc_controller in VHDL format

No	Name	Description
T2.3.1	Hardware functional definition and partition	1. Itemize all functions we need: video controller to display images, video buffer being an OPB master device as well, mode master select (we decide not to implement an arbiter because we don't really need it, as long as we can switch the mode in between 2 masters), and an external SRAM controller to read and write SRAM. 2. Also we need to understand external SRAM address mapping since it will cause bugs if we can't properly assign the correct address (it did happen many times). For our case, we use 0xC0000003 as our SRAM base address, the last hex 0x'3' can be ignored as 0x'0' because we force byte enable all '1's in our hardware. We also assign first 480*80 words (32 bits) in SRAM as video buffer space for video display, and all the following words in SRAM are free for host to use for JPEG image storage or other purposes. 3. two external SRAM (256k*16) is combined into one SRAM (256*32), so for host to use it, byte-to-word, and word-to-byte conversion should be done properly. For video controller, it matches its shift register width well.
T2.3.2	Video controller	1. Video controller is to scan the 640*480 pixel screen in a RASPER fashion. Shift register in it is 32-bit long, matching one word in SRAM. 4-bit per pixel is defined to use, first 3 MSB bits are red bit, green bit, and blue bit. Last bit is unused in order to match address alignment problem. So 8 pixels output per shift register is done. It also means 480*80-word video buffer is needed for the whole video screen. 2. Video controller runs in pixel_clk domain, half speed compared to opb_clk domain, so when it signals "video buffer update" right after signal LoadNShift, video buffer have enough time (up to around 16 opb_clk cycles) to load in the new pixel word from external SRAM. 3. Video controller needs to count the correct offset address for the next pixel word, and that offset address is word-based, not byte-based by default convention.
T2.3.2	Video buffer	1. Video buffer separate two clock domain worlds, one world in pixel_clk domain to accept new pixel word load-in command from video controller, and the other world in opb_clk domain to play as an OPB master device to read in the new pixel word from SRAM controller acting as OPB slave device.

		2. Control signal passed from video controller is a cross-clock-domain signal, so double-latch to make metal stable is needed before in use in opb_clk domain for data integrity reason. 3. Video buffer needs to calculate out correct OPB read address, it means byte-based base address adds word-based offset address should be taken care properly in VHDL code.
T2.3.2	Mode master select	1. Mode select is stored in a control register in mode master select, and also in SRAM location 0xC00FFFFF. Why do that? Because when host to write this control register, it uses OPB bus, in order for us to give back the correct ACK in OPB protocol, we make this control data to save into SRAM in reality to make OPB protocol kept. Only one word is traded off out of 256k words, which we don't care. 2. In mode master select, we also screen the read/write request is true or false by checking its header (as identifier), first 12 bits of R/W address is 0xC00 or not, because in our MHS definitions for HIGH address is much larger than 1M bytes, and to prevent OPB master choose it but with wrong address. 3. Mode control register includes force mode because we make video display default, when host need to access SRAM, force mode is used to do that. Actually the mode control can be done in 1 bit, but since we have already coded it in VHDL, we just use it.
T2.3.2	External memory controller	1. External memory is an OPB slave device on one side to respond to OPB master request, and on the other side it needs to conduct SRAM read and write correctly considering SRAM limitations. 2. We keep MEM_CEN low to let SRAM R/W quickest, but with more power consumption. (In our case, timing is not that critical though because our clock cycle is 20nS wide.) 3. SRAM read is done using MEM_OEN and MEM_WEN to control, 3-cycle FSM stage is used to match SRAM write to simplify logic design, actually it can read in less than 3 cycles easily. 4. SRAM write is done using 1-cycle MEM_WEN to keep timing safe, so 3-cycle FSM stage is used (excluding idle state). 5. Read out data is only valid for one cycle when OPB_ACK = '1' to match OPB protocol.

2.4 MISC Notes we've considered and resolved

Note 0 – Final result is subject to change which means it may differ from what we state below here.

Note 1 – DI/OA/B[15:0] of external SRAM are bi-directional, so we should declare special tri-state I/O pads for them in Xilinx Platform Studio. Actually we don't need to do that. Another way to work around is to reference what Xilinx IP does, they declare three internal pins (pin_I, pin_O, & pin_T) of tristate IO pad in port list of related PCORE module, and declare them in system.mhs, and then Platform Studio will match them to a tristate IO pad automatically.

Note 2 – If external memory/SRAM controller (EMC) could be accessed by uBlaze and video controller at the same time, an EMC arbiter for that is needed. But like we stated above, actually we don't need that, the only thing we need is a mode switch if we don't need video display while decoding and downloading JPEG image.

Note 3 – Write effecient C codes to help debugging. Professor demonstrated some good tips for debugging.

Note 4 – A way to avoid EMC arbiter is to enable video controller AFTER JPEG decoding, so we need to have a special way/procedure to disable/enable RS232-to-external-memory path (for image file download), disable/enable uBlaze-to-external-memory path (for JPEG decoding), and disable/enable external-memory-to-video-display path (for video display) sequentially.

For example: (All control signals are active-high)

main_control[2:0] = { enable RS232-to-external-memory path (for download),
enable uBlaze-to-external-memory path (for decoding),
enable external-memory-to-video-display path (for display) }

main program {.

P0: (for display)
Initalize system,
Pre-loaded RGB=000 (black) to video memory in external SRAM,
⇨ main_control[2:0] = 001

⇨ Enable external-memory-to-video-display path (for display)

P1: (for download)
⇨ main_control[2:0] = 001
⇨ Enable external-memory-to-video-display path (for display)
if RS232 interrupt and main_control[2:0] = 001 and first 5-byte character pattern is ”start” (typed in Hiper)
⇨ main_control[2:0] = 100
⇨ Enable RS232-to-external-memory path (for download)
procedure { download image from RS232 to external SRAM }
Put JPEG image file into Hyper,
else if 6-byte pattern “theend” (typed in Hiper) is met,
⇨ image_byte_counter backs off 6 bytes, main_control[2:0] = 010
⇨ Enable uBlaze-to-external-memory path (for decoding)

P2: (for decoding)
⇨ image_byte_counter backs off 6 bytes, main_control[2:0] = 010
⇨ Enable uBlaze-to-external-memory path (for decoding)
procedure { JPEG decoding }
if JPEG decoding is done (notified by some special handler),
⇨ main_control[2:0] = 001
⇨ Enable external-memory-to-video-display path (for display)

P3: (for display)
Go back to P1

} // end of main program

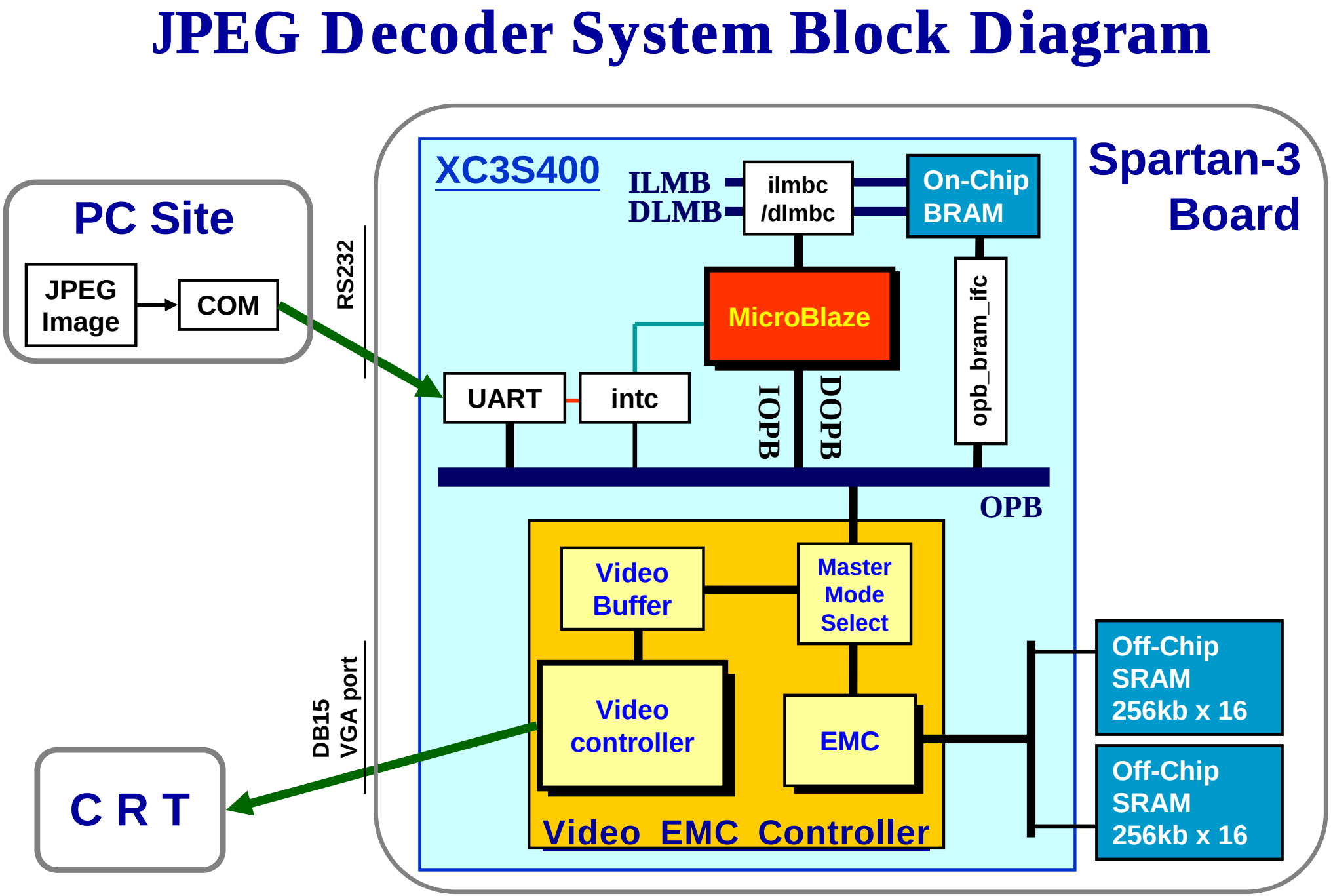
Comment : Arbiter turns into a switch,
switch to uBlaze if main_control[2:0] = 100 or 010
switch to Video display if main_control[2:0] = 001

3. Schedule breakdown (uncompleted)

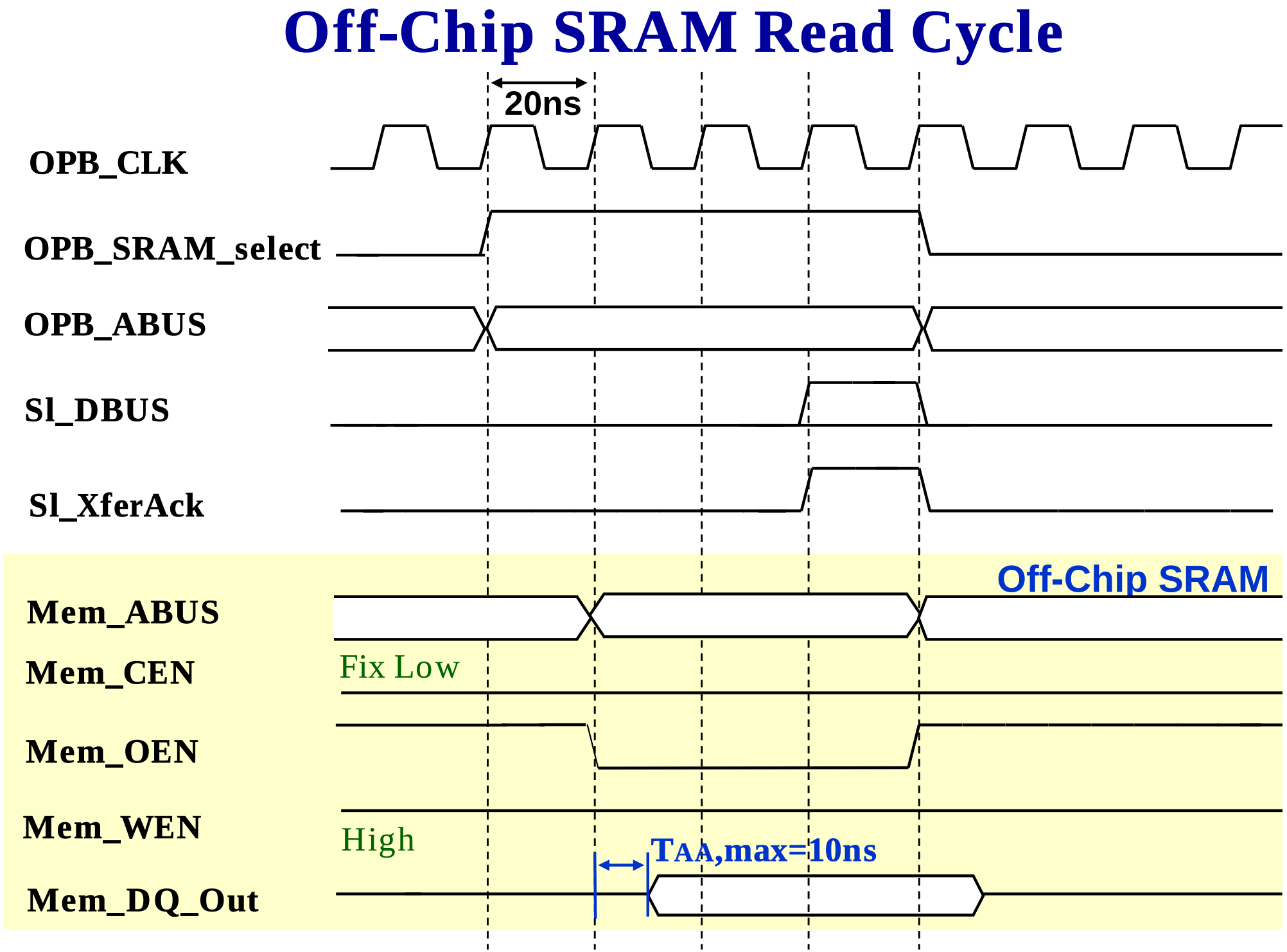
Project schedule breakdown
(Uncompleted because actually all three of us have spent much more time than stated below whenever we can do it remotely or on site in lab)
(Project due day has been re-scheduled to 2005/8/26 (Friday), so we did our best up to that day.)

Date	Sean	Steel	Steve	Note
2005/8/19 (Friday)	Morning (RS) Afternoon (OS) Night (OS)	Morning (RS) Afternoon (OS) Night (OS)	Morning (RS) Afternoon (OS) Night (OS)	OS means on-site in NCTU lab or other near places.
2005/8/20 (Saturday)	Morning (OFF) Afternoon (OFF) Night (RS)	Morning (RS) Afternoon (OFF) Night (RS)	Morning (OFF) Afternoon (OFF) Night (RS)	RS mean remote-site but doing project
2005/8/21 (Sunday)	Morning (OS) Afternoon (OS) Night (OS)	Morning (OS) Afternoon (OS) Night (OS)	Morning (OS) Afternoon (OS) Night (OS)	Off means unavailable at that time
2005/8/22 (Monday)	Morning (RS) Afternoon (OS) Night (OS)	Morning (RS) Afternoon (OS) Night (OS)	Morning (RS) Afternoon (OS) Night (OS)	
2005/8/23 (Tuesday)	Morning (RS) Afternoon (OS) Night (OFF)	Morning (RS) Afternoon (OS) Night (OS)	Morning (RS) Afternoon (OS) Night (OFF)	
2005/8/24 (Wednesday)	Morning (OS) Afternoon (OS) Night (OS)	Morning (OS) Afternoon (OS) Night (OS)	Morning (OS) Afternoon (OS) Night (OS)	

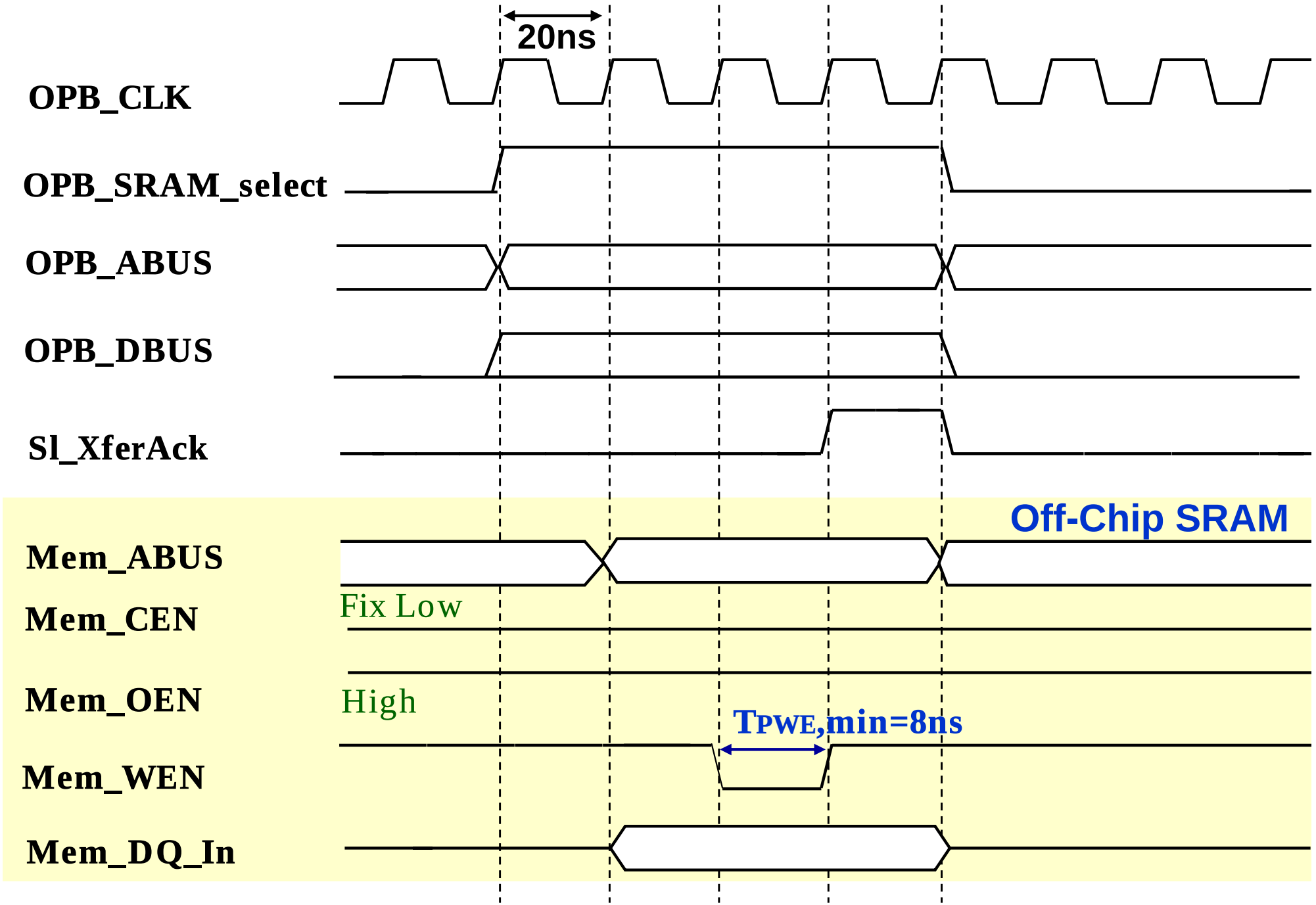
4. Block diagram of system architecture



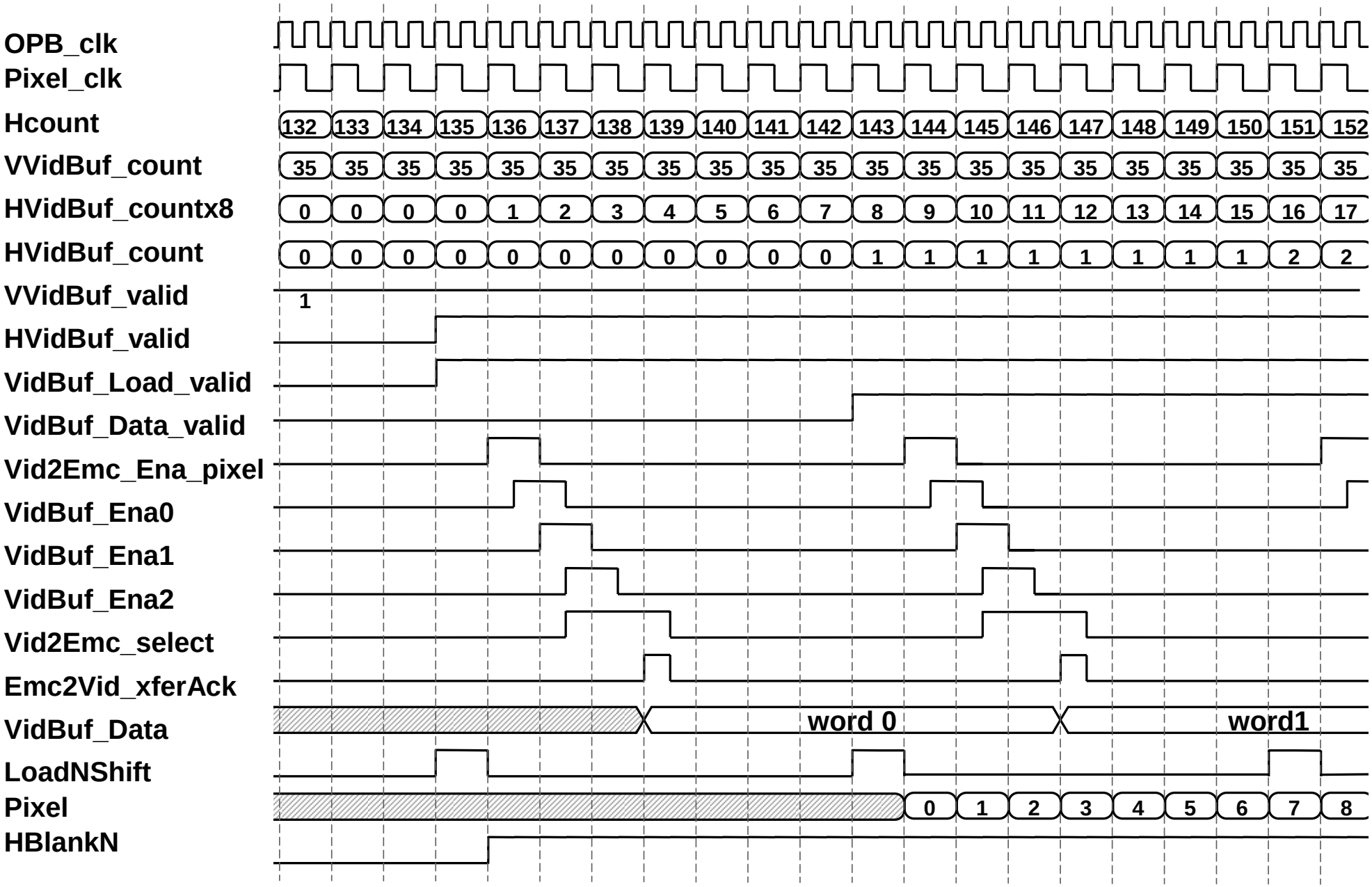
5. Timing diagram of video_emc_controller



Off-Chip SRAM Write Cycle



Start-of-Line Detail



End-of-Line Detail