EyeLytics

Main/Baseline Profile HD H264 Encoder FPGA/ASIC IP

Combines low cost and high quality **April 20th, 2010**

EyeLytics' Main/Baseline Profile High Definition H264 Encoder IP core has been developed using an optimized heavily pipelined architecture. This architecture gives our encoder a 2x performance advantage over our competitors using a similar amount of logic. Our encoder is developed with the whole system in mind. This makes our interface system optimized and easy to use. The encoder is fully verified in the lab with a real life surveillance system and application.

IP Features:

- Supports image sizes from 176x144 up to 2000x2000 pixels.
- Multi-channel is supported by time sharing the encoder.
  - Each channel can have **different resolution, quality and frame rate**.
  - **Main Profile and Baseline Profile dynamically switchable** between channel
  - There is no limit on the number of channels sharing an encoder.
- Constant quality. Scene complexity does not degrade the image quality. Instead, the frame rate is adjusted to achieve the best image quality. This feature is very important in surveillance applications which often prefer a clear image over smooth motion.
- Variable bit-rate with peak limit
- Low latency. **Less than 32 lines of delay** from the end of video input frame to the end of encoded bitstream.
- High compression
  - CABAC / CAVLC
  - Deblock
  - all 13 Intra Luma modes
  - all 4 Intra Chroma modes
  - single slice operation
  - all 4 inter prediction macroblock partitions
  - all 4 inter prediction sub-macroblock partitions
  - skip macroblock
  - search window of 240x128 pixels
  - half pel and quarter pel search
- High performance
- Low Gate Count

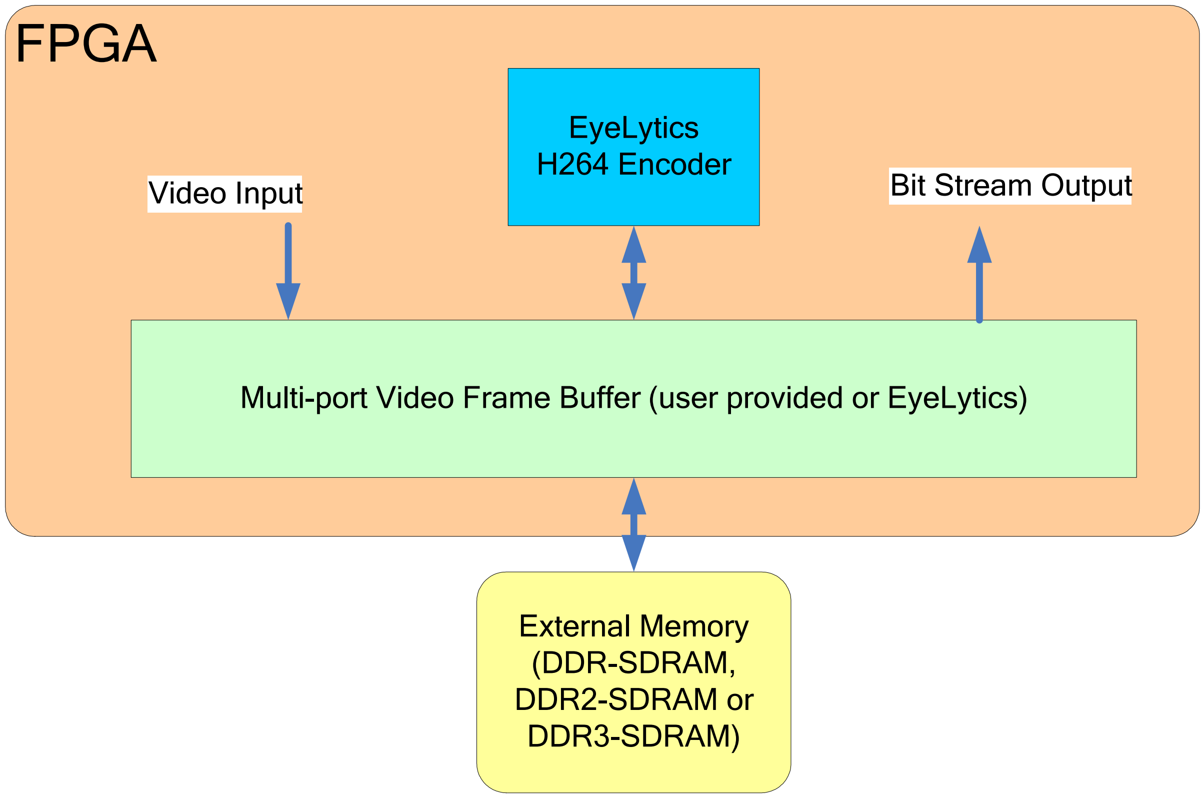
Tech.	**Altera CycloneIII -8 speed**	**Altera Stratix III -2 speed**	**0.13um standard cell**
Logic Count	~32K LC	~25K ALUT	~300K gates
Memory Count	~1.3Mbit ~191 M9K	~1.3Mbit ~154 M9K ~5 M144K	~1.3Mbit
DSP Count	5 DSP	5 DSP	N/A
Perform.	150MHz 720p30 1080p15	300MHz 720p60 1080p30	300MHz 720p60 1080p30

IP camera demo

Contact:

- sales@eyelytics.com
- www.eyelytics.com

1 Introduction


FPGA
EyeLytics H264 Encoder
Video Input
Bit Stream Output
Multi-port Video Frame Buffer (user provided or EyeLytics)
External Memory (DDR-SDRAM, DDR2-SDRAM or DDR3-SDRAM)

EyeLytics' Main/Baseline Profile High Definition H264 Encoder IP core is designed to receive video input and to send bit stream output thru a separate multi-port video frame buffer module. An EyeLytics multi-port video frame buffer IP core is available. Please contact sales@eyelytics.com. Besides the multi-port video frame buffer connections, the Encoder IP core only requires clocks, reset, some static signals potentially come from software programmable registers and a "StartFrame" pulse to start encode of an image frame. There are also status signals "FrameDone" and "BitStreamByteCount" to notify designers that the encode is finished and to report the byte count of the output bit stream.

2 Interface Signals

Signal Name	Direction	Bit width	Descriptions
System Signals			
clk	in		system clock
CabacClk	in		clock for the CABAC module. CABAC module runs slower than the rest of the system. To avoid slowing down the whole system by the CABAC module, CABAC module runs with this separate CabacClk. CabacClk should run at less than 2/3 of the system clock frequency. A high CabacClk frequency allows the encoder to support high bit rate applications. Not used in CAVLC mode
reset	in		system reset
Control Input Signals			
StartFrame	in		A single pulse of StartFrame starts the encoder engine
StartSeq	in		Indicates the current frame is the first frame of a video sequence. This signal should be valid when StartFrame is asserted.
CavlcMode	in		1: current frame enables using CAVLC 0: current frame enables using CABAC
PFrame	in		1: current frame is a P Frame 0: current frame is an I Frame *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
QP	in	5:0	Quantisation parameter. Setting it 0 to generate the highest quality video and higher bit rate. Setting it to larger values to generate lower quality video and lower bit rate. Valid range is 0 to 51. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
HSizeInMBMinusOne	in	6:0	Defines the width of the image. The value should be the width in pixel divided by 16 and minus one. For example, a SD image of 720x480 should have this input set to 720/16-1 = 44. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*

VSizeInMBMinusOne	in	6:0	Defines the height of the image. The value should be the height in pixel divided by 16 and minus one. For example, a SD image of 720x480 should have this input set to 480/16-1 = 29. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
PicSizeIn128ByteWords	in	13:0	Defines the amount of data in the image. The value should be the width in pixel times the height in pixel times 1.5 divided by 128. The 1.5 factor is to calculate the number of bytes in a 4:2:0 video. For example, a SD image of 720x480 should hav this input set to 720x480x1.5/128 = 4050. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
frame_num	in	3:0	This should be set to 0 for the first frame of a sequence. It should then be incremented by one for every frame in the same stream. When 15 is reached, the frame_num of the stream should wrap around and be 0 again. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
flush_frame	in		Set to 1 to flush the bitstream data at the end of encode. This should always be set to 1 for multi-stream encode. For single stream encode, user has the option to keep this 0 and only flushing the encoder at the very last frame of a sequence. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
BS_DmaAddr	in	24:0	Set the external memory address where the bit-stream should be written to. Each address location is a 32-bit word. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
IN_BaseAddr	in	24:0	Set the external memory address where the video input data should be read from. Each address location is a 32-bit word. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*

REF_BaseAddr	in	24:0	Set the external memory address where the video reference data should be read from. Each address location is a 32-bit word. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted..*
OUT_BaseAddr	in	24:0	Set the external memory address where the video decoded / reference data should be written to. Each address location is a 32-bit word. *This signal should be valid one cycle before StartFrame is asserted and remain static until FrameDone is asserted.*
Bitstream Output Signals			
BSValid	out		BSData is valid
BSData	out	7:0	Compressed bit stream output
Status Output Signals			
FrameDone	out		A signal indicates the encoder is ready for the start of the next encode.
BitStreamByteCount	out	23:0	This signal is incremented for every bit-stream byte sent out. When it reaches 0xFFFFFF, it will wrap around back to 0x000000.
Frame Buffer Interface Signals			
IN_FB_read_req	out		Video input frame buffer read request.
IN_FB_ready	in		Video input frame buffer read ready
IN_FB_address	out	24:0	Video input frame buffer address. Each address location is a 32-bit word.
IN_FB_rdata	in	63:0	Video input frame buffer read data
IN_FB_rdata_valid	in		Video input frame buffer read data valid
IN_FB_rdata_ren	out		Video input frame buffer read data read enable
ME_FB_read_req	out		Motion estimation frame buffer read request.
ME_FB_ready	in		Motion estimation frame buffer read ready
ME_FB_address	out	24:0	Motion estimation frame buffer address. Each address location is a 32-bit word.
ME_FB_rdata	in	63:0	Motion estimation frame buffer read data
ME_FB_rdata_valid	in		Motion estimation frame buffer read data valid
MCC_FB_read_req	out		Motion compensation chroma frame buffer read request.
MCC_FB_ready	in		Motion compensation chroma frame buffer read ready
MCC_FB_address	out	24:0	Motion compensation chroma frame buffer address. Each address location is a 32-bit word.
MCC_FB_rdata	in	63:0	Motion compensation chroma frame buffer read data
MCC_FB_rdata_valid	in		Motion compensation chroma frame buffer read

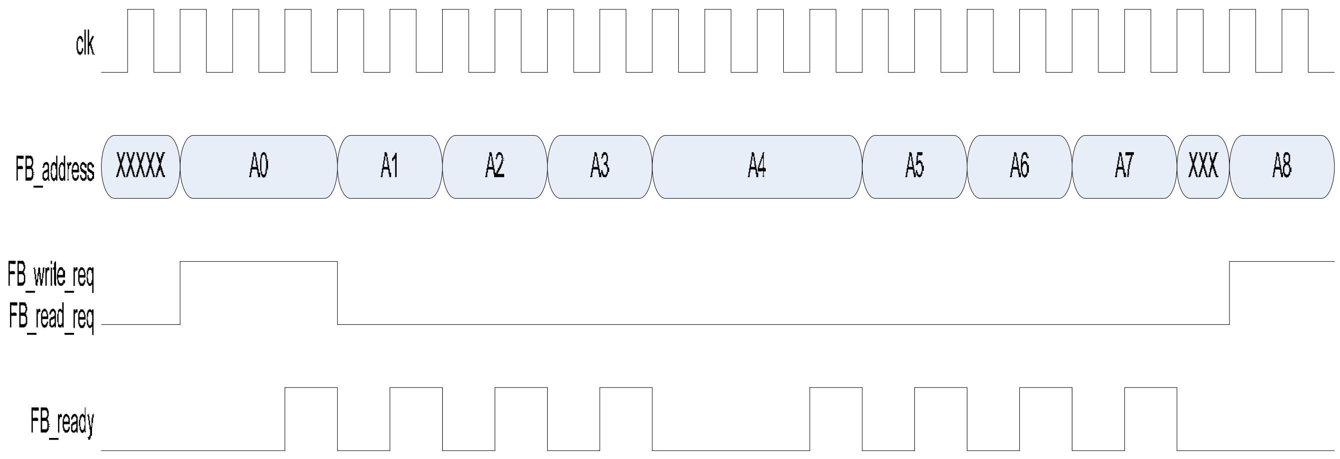
			data valid
MCC_FB_rdata_ren	out		Motion compensation chroma frame buffer read data read enable
DB_FB_write_req	out		Deblock frame buffer write request
DB_FB_ready	in		Deblock frame buffer read ready
DB_FB_address	out	24:0	Deblock frame buffer address. Each address location is a 32-bit word.
DB_FB_wdata	out	63:0	Deblock frame buffer write data
DB_FB_be	out	7:0	Deblock frame buffer write data byte enable
DB_FB_wdata_req	in		Deblock frame buffer write data request
BS_FB_write_req	out		Bit stream frame buffer write request
BS_FB_ready	in		Bit stream frame buffer read ready
BS_FB_address	out	24:0	Bit stream frame buffer address. Each address location is a 32-bit word.
BS_FB_wdata	out	63:0	Bit stream frame buffer write data
BS_FB_be	out	7:0	Bit stream frame buffer write data byte enable
BS_FB_wdata_wr	out		Bit stream frame buffer write data write
BS_FB_wdata_wen	in		Bit stream frame buffer write data write enable
BS_FB_wdata_pause	in		Bit stream frame buffer write data pause

3 External Frame Buffer Requirement

An external multi-port frame buffer is required to build the encoding system. The following sections describe the read / write operations.

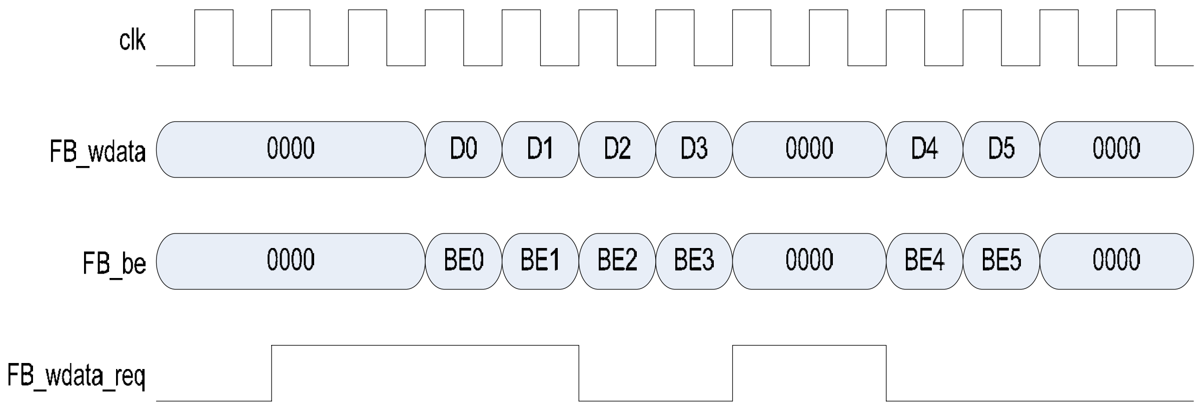
3.1 Frame Buffer Request Operation

1) Encoder sets the FB_address with the target frame buffer address location.
2) Encoder then asserts the FB_write_req signal or the FB_read_req signal to request frame buffer write operation or read operation respectively.
3) Each frame buffer request is for eight bursts of 16 bytes or for a total of 128 bytes. Each burst can have a different burst starting address. As a result, there are eight FB_ready pulses to latch eight starting addresses.
4) The first FB_ready signal generated by the frame buffer is used as both a grant signal as well as an address advance signal. FB_write_req or FB_read_req should be deasserted after the first FB_ready if there are no more requests afterward.
5) Encoder will change its address the next cycle after each FB_ready pulse.
6) FB_ready should never be asserted two consecutive cycles in a row.



3.2 Standard Frame Buffer Data Write Operation

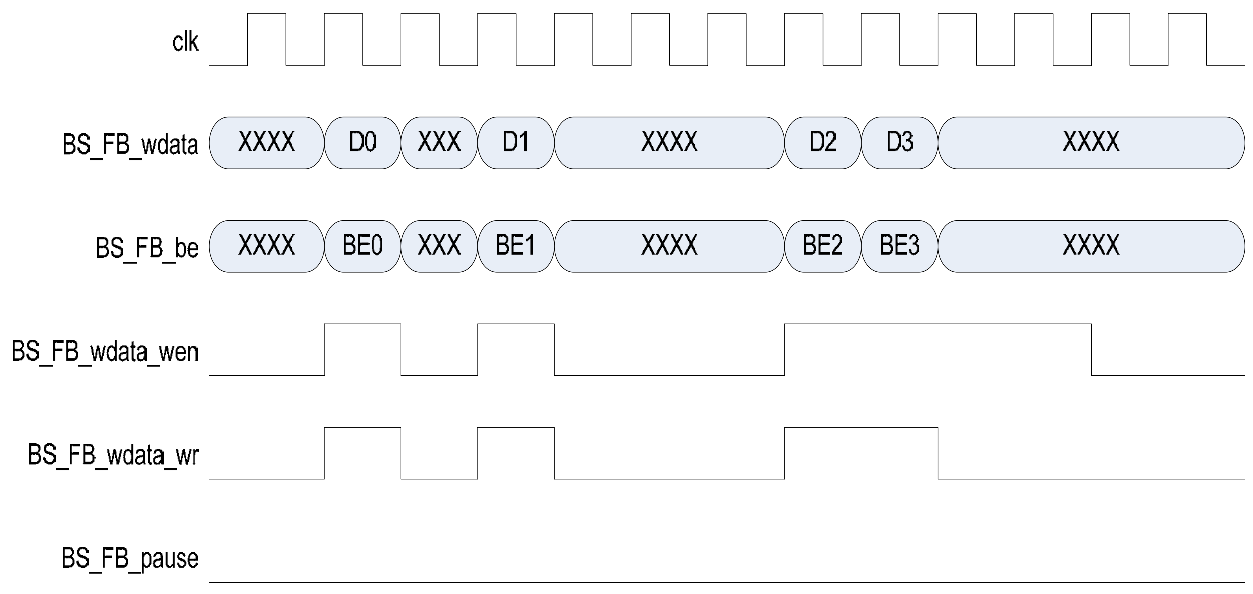
1) Frame buffer is expected to assert the FB_wdata_req signal one cycle for each 64-bit data. This means FB_wdata_req should be asserted for two cycles for each starting address to transfer 16 bytes.
2) Encoder presents valid data on FB_wdata and FB_be two cycles after each FB_wdata_req assertion.
3) Encoder also zeros out the FB_wdata and FB_be buses when there is no data being transferred. This is to make it easier for the frame buffer designer to meet timing. Simply use "or" logic to multiplex all the FB_wdata and FB_be buses together.



3.3 Bitstream Frame Buffer Data Write Operation

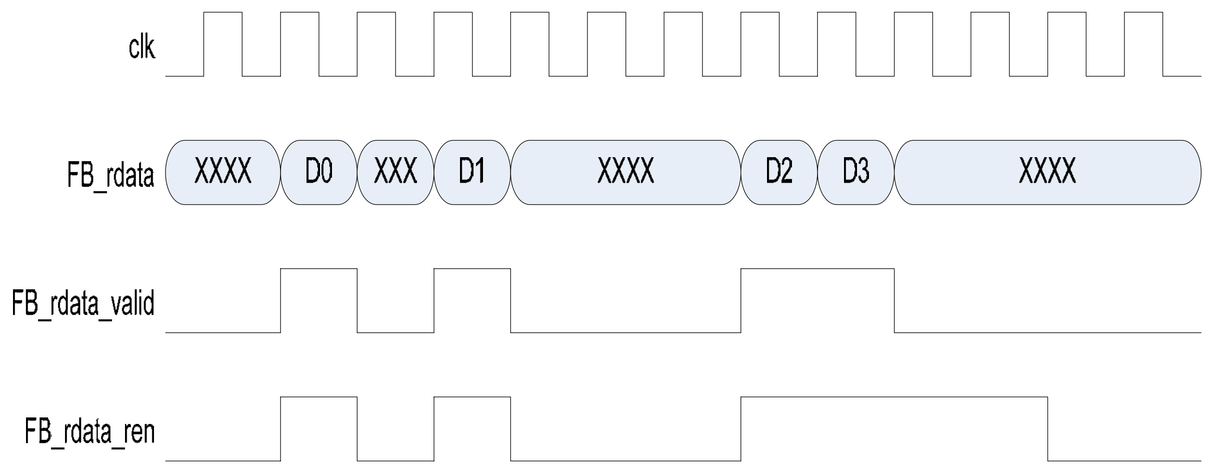
The Bitstream frame buffer interface is different from the standard frame buffer interface. This is done to reduce memory usage for the whole system. **Designers are highly recommended to use the fb_wrbuf_share module to convert the Bitstream interface to the standard frame buffer interface**. The fb_wrbuf_share module can be downloaded from www.eyelytics.com/download/download.html.

1) Frame buffer asserts BS_FB_wdata_wen signal to allow data transfer. If Frame buffer can accept data all the time, BS_FB_wdata_wen can be tied to high.
2) Encoder asserts BS_FB_wdata_wr signal to send data to Frame Buffer.
3) The BS_FB_wdata and BS_FB_be are valid only when both BS_FB_wdata_wen and BS_FB_wdata_wr are asserted.
4) Unlike the standard frame buffer write operation, BS_FB_wdata and BS_FB_be won't be zeroed out by the encoder.
5) Frame buffer is expected to have an internal buffer of 64 64-bit words. When this internal buffer is filled more than half full, the frame buffer interface is expected to assert the BS_FB_wdata_pause signal.



3.4 Frame Buffer Data Read Operation

1) Frame buffer is expected to assert the FB_rdata_valid signal one cycle for each 64-bit data. This means FB_rdata_valid should be asserted for two cycles for each starting address to transfer 16 bytes.
2) Frame buffer should also present the FB_rdata during the same cycle that FB_rdata_valid is asserted.
3) When FB_rdata_ren signal is present in the encoder interface, Frame Buffer should only assert FB_rdata_valid when FB_rdata_ren is asserted in the same cycle. This means Frame Buffer is required to have buffering for each port that uses FB_rdata_ren.



To simplify the user's design and to reduce memory usage, a fb_rdbuf_share module can be downloaded from www.eyelytics.com/download/download.html. This fb_rdbuf_share module allows multiple clients (e.g. IN and MCC of the encoder) to share a single buffer. This reduces amount of memory needed in the whole system.

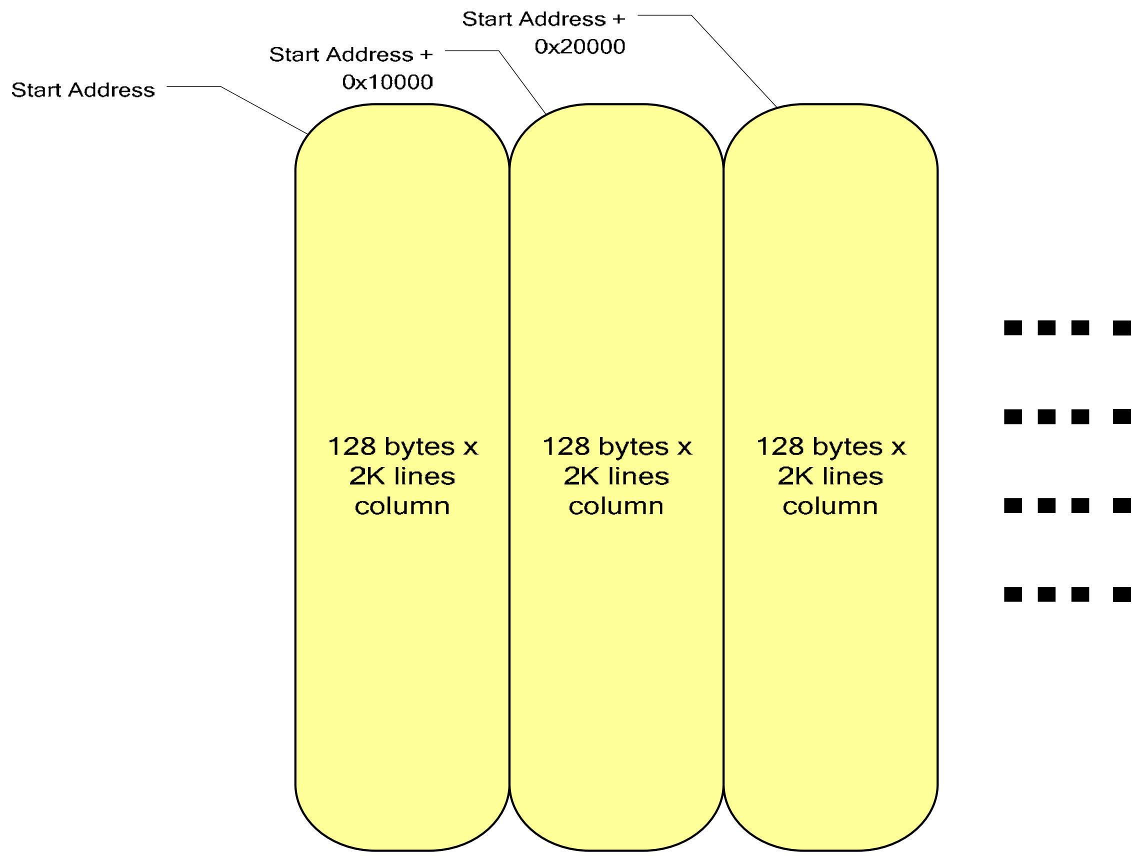
3.5 Frame Buffer Performance Recommendations

Frame buffer performance directly affects the encoder performance. As a result, the frame buffer designer should pay careful attention to the following:

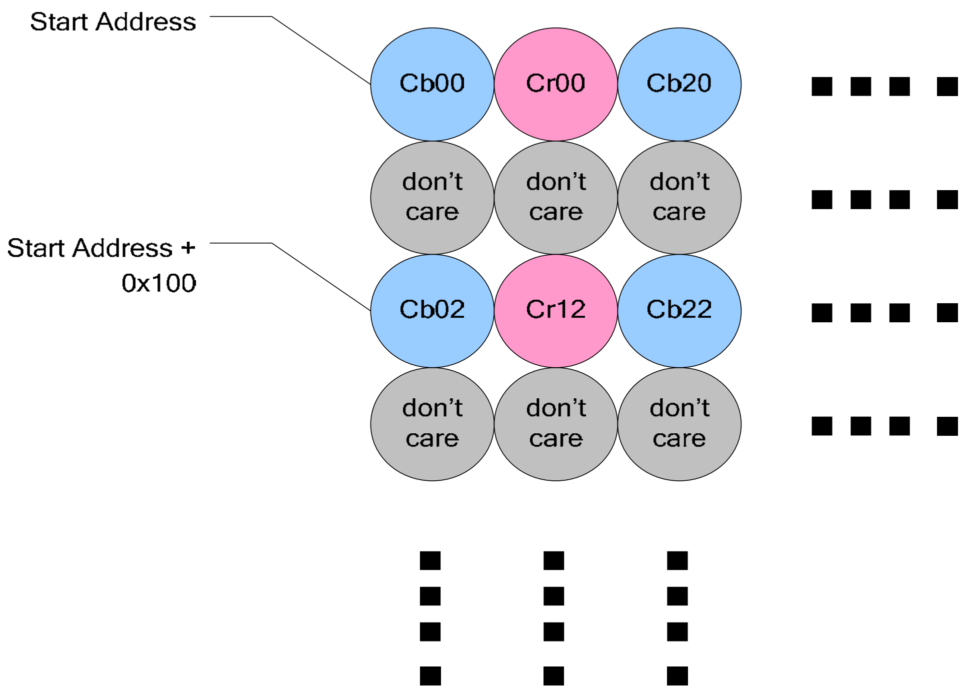
1) Allow multiple pending requests. This can be done by using a small FIFO to buffer up the requests.
2) Arbitration has to be as fair as possible.
3) Frame buffer should be optimized to do long bursts and random accesses. Low latency is not as important.
4) A well designed high efficiency Frame buffer with 32-bit 150MHz DDR2-SDRAM should have adequate memory bandwidth to support the encoder.

4 Video In Frame Buffer Storage Formats

Video input data is passed to the encoder through the frame buffer. Video input data should be stored inside the frame buffer with luma and chroma component separated. Chroma component should be stored 0x800000 bytes after the luma component counterpart. Both Luma and Chroma frames are partitioned into 128 byte columns. Each column has 2K lines. Each column is stored in frame buffer after the column from the left. If the frame has a horizontal size that is not multiple of 128 bytes or the frame has a vertical size that is less than 2K lines, the storage should be left unpacked. User can choose either not writing to those storage spaces, to fill them with any content, or to use the storage space for other applications.


Start Address
Start Address + 0x10000
Start Address + 0x20000
128 bytes x 2K lines column
128 bytes x 2K lines column
128 bytes x 2K lines column

Within each 128 bytes x 2K lines column, the Luma data should be stored in a linear raster scan order. However, the Chroma data is Cb/Cr component interleaved. Also, since the encoder uses 4:2:0 format, every other chroma line is not used.

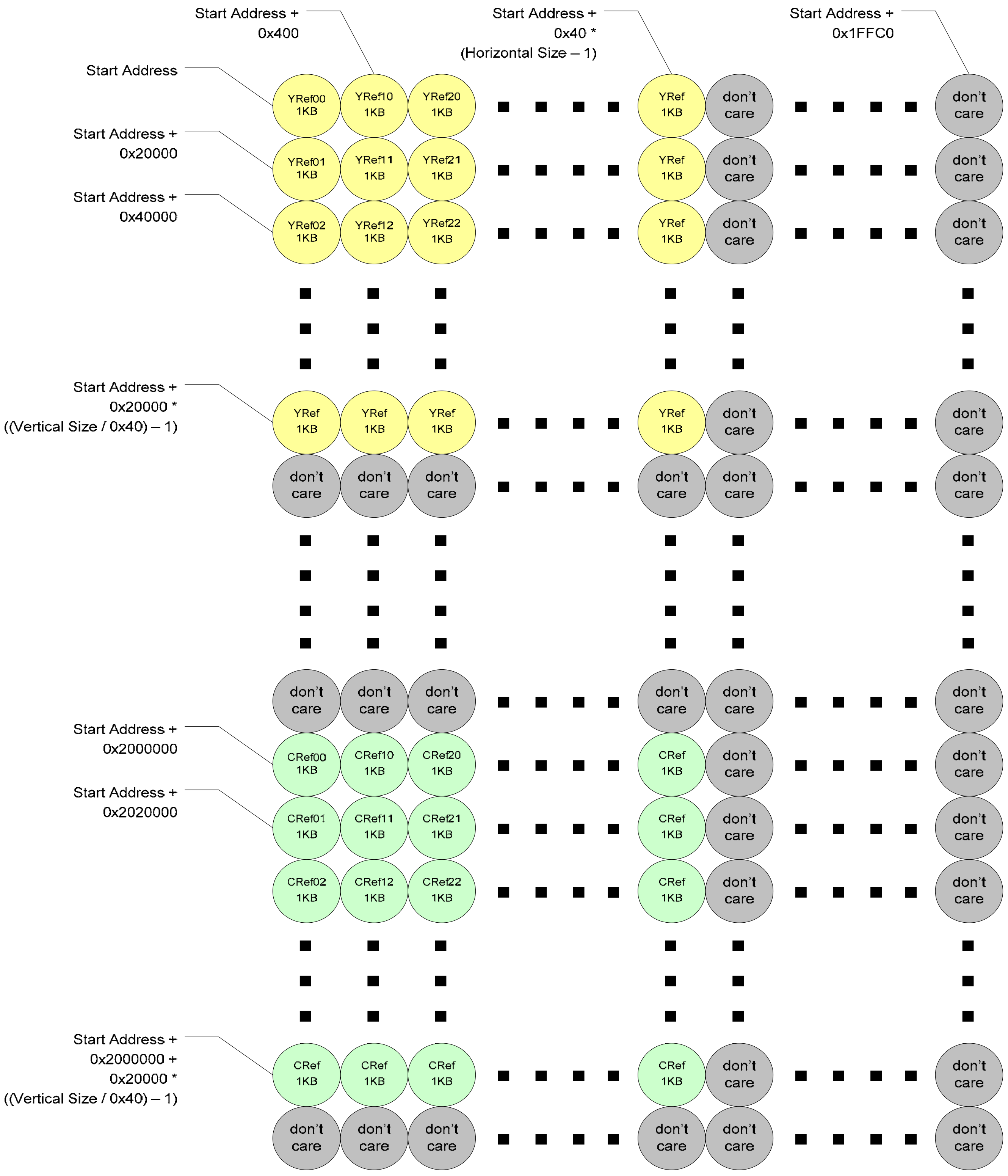


5 Reference Frame Frame Buffer Storage Requirement

When doing a P frame encode, the previous decoded output is used as the current encode reference frame. The decoded output is stored to the frame buffer using the DB_FB signals. The ME_FB signals read the reference frame to perform motion estimation. The MCC_FB signals read the reference frame to perform motion compensation.

The luma and chroma components of the reference frame are stored in separate locations. The luma frame is always stored with bit 23 of the FB_address and base_address set to zero. The chroma frame is always stored with bit 23 of the FB_address and base_address set to one. Since each address location is a 32-bit word, the chroma reference will be 0x2000000 byte away from the luma reference counter part.

Both luma and chroma reference data are built with 1Kbyte reference building blocks. Each 1K byte reference building block stores reference info for a 16x64 pixel region. The following shows how these reference building blocks are arranged in the Frame Buffer.


Start Address
Start Address + 0x400
Start Address + 0x40 * (Horizontal Size – 1)
Start Address + 0x1FFC0
Start Address + 0x20000
Start Address + 0x40000
Start Address + 0x20000 * ((Vertical Size / 0x40) – 1)
Start Address + 0x2000000
Start Address + 0x2020000
Start Address + 0x2000000 + 0x20000 * ((Vertical Size / 0x40) – 1)
YRef00 1KB
YRef10 1KB
YRef20 1KB
YRef01 1KB
YRef11 1KB
YRef21 1KB
YRef02 1KB
YRef12 1KB
YRef22 1KB
YRef 1KB
CRef00 1KB
CRef10 1KB
CRef20 1KB
CRef01 1KB
CRef11 1KB
CRef21 1KB
CRef02 1KB
CRef12 1KB
CRef22 1KB
CRef 1KB
don't care