UNIVERSITY OF WATERLOO
Faculty of Engineering
Department of Electrical and Computer Engineering

VIDEO BUFFERING SOLUTIONS FOR A CCD CAMERA

ABC, Inc.

Prepared by
Paul Roukema
ID XXXXXXXX
userid pwroukem
3A Computer Engineering

March 22, 2012

March 22, 2012

Professor Sachdev, Chair
Electrical and Computer Engineering,
University of Waterloo,
Waterloo, ON
N2L 3G1

Dear Professor Sachdev:

This report, entitled "Video Buffering Solutions for a CCD Camera", was prepared as my 3A Work Report for ABC, Inc.. This report is in fulfillment of the course WKRPT 400. The purpose of this report is to determine the optimal approach for video buffering in a CCD based camera system.

ABC, Inc. is a leading electronic design services firm, providing a full range of electronic product design services.

My functional manager while at ABC was XXXXXXXXX. I also worked for a number of project managers throughout the my time at ABC.

I wish to thank Mr. Simon Law for creating the uw-wkrpt Latex document class which was used to typeset this report.

I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm that this report has not been previously submitted for academic credit at this or any other academic institution.

Yours sincerely,

_____

Paul Roukema,
ID XXXXXXXX

Contributions

At ABC I worked on two different project teams; the first consisted of four people, while the second had about 14 at its peak. Although neither team was particularly large in an absolute sense, the second team was large compared to other project teams in the company.

Both teams I was involved with were engaged in design services projects. The first team's goal was to develop a demonstration high-definition camera system. Using existing hardware and development kits, the team was to produce a working demonstration camera system with HD-SDI (a professional video interface) output in 6 weeks. Some of the challenges in this process included the integration of a full image processing pipeline, including demosaicing and color correcting into an FPGA, as well as developing the required control software. The three biggest challenges were tuning the CCD sensor drive signals to reach 30 frames per second, integration of our video framebuffer and reverse engineering of an LVDS to HD-SDI converter board designed for a different system.

The second team I worked on had the goal of building a production grade version of the camera system built in the previous project.This involved designing and building a stack of image capture, processing and power supply boards, which duplicated the functionality of the previous system, while incorporating additional processing power, architectural changes and ruggedization, while shrinking the mechanical footprint of the system. Much of the challenges encountered in this project stemmed from integrating with the video processing chip being used. The team had no prior experience with the vendor's products and the documentation provided was not as clear or comprehensive as it could have been.

My tasks included a full range of hardware, software and FPGA development activities. During the first project I developed a several VHDL video path blocks, including a digital filter and a number of I/O modules. I also developed the firmware for configuring and operating the camera system. I performed the majority of the development work for the integration of a third-party LVDS to HD-SDI board, including reverse engineering the required startup sequence.

During the second project I performed the schematic design for both the primary sensor board, which provided drive signals and digitization of outputs for the CCD sensor, as well as the primary power supply board for the system. I then developed the control path driver software for the sensor board and the video ingest FPGA. Once hardware arrived I performed the bringup and testing for the sensor data and control paths up to the main processing IC, as well as debugging the high-speed interprocessor communication link in the system.

This report focuses on a system level design problem encountered during the first project.

In the system designed for that project, the input to the video pipeline operates in a different clock domain and and with different timing than the output. In order to convert between rates, some type of conversion is required. This report examines the tradeoffs between two options for implementing this conversion and makes a recommendation as to which option to implement.

Overall, the two project I worked on form part of the core business of the company. ABC's primary business is to complete projects such as these in a way that satisfies all parties involved. Both of these projects posed additional challenges due to the tight timelines and significant integration work required. My work was crucial to the successful and timely completion of both projects.

Summary

The purpose of this report is to determine the optimal approach for video buffering in a CCD based camera system. The scope of this report is to examine two solutions which meet the requirements of the system to determine which offer the best combination of low resource utilization, power consumption and complexity along with high flexibility.

The major points cover in this report are the system background, buffering block requirements, options, analysis, recommendations and conclusions. The system background is explored to provide context for the requirements and analysis. The overall requirements are then presented, along with two solutions which meet these requirements. The relevant resource utilization and power consumption metrics are then calculated, along an examination of the complexity and flexibility factors of the system.

The primary conclusion of this report is that the approach identified as FIFO based has the best resource utilization and power consumption of the two approaches analyzed. In addition this approach demonstrates considerably lower complexity. Finally this approach is determined to be not as flexible as the alternate approach, which is based on a traditional framebuffer.

The primary recommendation of this report is that the FIFO based approach be used, both due to lower complexity as well a lower power consumption and resource utilization. The complexity aspect is highlighted as key to this recommendation. A secondary recommendation suggests that in the case that the traditional framebuffer approach is used, the integration of the framebuffer be given a high priority and be teated as a critical path item.

Table of Contents

List of Figures

List of Tables

1 Introduction

ABC is a leading design services firm with offices in Waterloo and San Jose. ABC provides a variety of electronic product design services to a wide range of companies. The first project that I was involved in at ABC involved the rapid creation of a demonstration platform for a high definition camera system. Using existing products and and evaluation kits as building blocks, the team I was on created the system, including video processing and HD-SDI output over the course of a six week period.

Despite the fact that we were able to start from a known working set of hardware and processing components, integration still posed a significant challenge, both in terms of design effort and schedule. In particular, the complexity of the DDR2 memory controller and its interface to the framebuffer block posed significant challenges. Multiple issues stemming from a tool version change and some errors in the configuration of the framebuffer cost the team nearly a week in the tight schedule. Towards the end of this bring-up process, the team began to explore alternate options for replacing the framebuffer. One possible solution that was brought up was to use a large FIFO memory. Although initially mentioned in an almost joking context, due to the expectation that the memory requirements would vastly exceed the amount of free memory in the FPGA, some quick calculations showed it might be a practical solution. Although the framebuffer solution was fixed before the FIFO approach's configuration could be fully debugged, the module was written and simulated before work was discontinued. This report aims to analyze both the FIFO and framebuffer approaches in order to determine their overall suitability, both in general and for this project specifically.

1.1 Background

The project which provides the context for this report involved the creation of a standalone HD-SDI camera using a development kit for a Kodak KAI-02150 Interline CCD imager [1]. The desired output format is 1080p30 per SMPTE 274M [2]. The video was to be output over HD-SDI con formant to SMTPE 292M [3]. The development kit for the CCD included an Altera Cyclone IV FPGA, specifically an EP4CE75F23C6 device [4], connected to a 512 Mb DDR2 memory, Micron MT47H32M16HR-25E[5].

At a high level, the video path in this system is as follows. Light incident on the imager photodiodes is converted to electrons and stored. At the start of a frame, the stored charge is moved from the photodiodes on the imager to a set of CCDs, which allow the image to be scanned out. The data is then digitized by and AFE chip, and sent to the FPGA on a source-synchronous, 8:1 LVDS link. The FPGA receives this data and performs a number of processing operations, eventually resulting in a SMPTE compliant 1080p30, YCbCr 4:2:2

video signal suitable for HD-SDI transmission [2] [3]. This signal is then serialized using 7:1 LVDS and sent to an HD-SDI transmitter board, which produces a 1.5 Gbps HD-SDI signal.

A number of factors in the design of the video path require the presence of multiple clock domains and video image formats. In order to produce SMTPE compliant video, the output format must be 1920 by 1080 pixels of active video, contained within a frame with dimensions of 2200 by 1125 pixels. This data must be output with a pixel clock of 74.25 MHz. These requirements are not compatible with the readout area of the CCD or the clock rate required to reach 30 FPS readout from the CCD.

The CCD imager used in this project supports multi-tap readout, allowing the image area to be divided into top and bottom or left and right halves and clocked out simultaneously from multiple outputs in a converging manner. As an interline device, the imager is composed of three primary components. The first is the photodiodes used to capture the incident light. The accumulated charge is then transferred to a set of vertical CCDs, one per column of pixels, which allows the image to be shifted out vertically into a horizontal CCD. The horizontal CCD then shifts individual charge packets from each photodiode to the output amplifiers. To support the multi-tap readout modes, the horizontal CCD is duplicated to both the top and bottom of the imager, and each CCD is split into two independently driven segments, either top-bottom for the vertical CCDs or left-right for horizontal CCDs . In this system, the left-right split is used in order to achieve the required frame rate. As a result, the minumum readout area is 1013x1146 pixels per tap based on the CCD imager specifications. In addition, a number of non-readout clocks are required for each line of the image, as well as a per frame sequence to move the image from the photodiodes to the horizontal CCDs. The result is that the AFE outputs a frame with a first line of 1704 pixels, followed by 1146 lines of 1142 pixels for each tap as shown in Figure 1. The pixel clock is 40 MHz, which is the maximum supported by the imager and AFE.

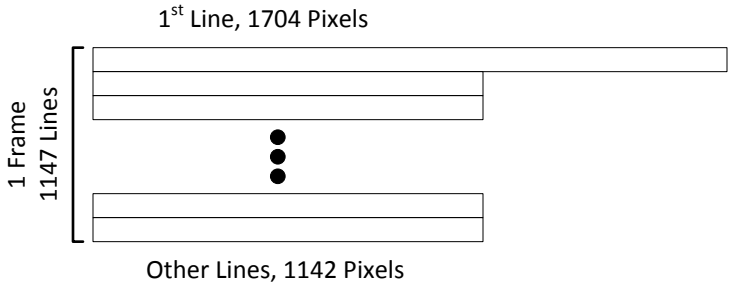

Figure 1: Video frame at AFE output

As a result of the different frame sizes and clock domains in use, some type of video buffering and retiming is required in order to bridge the video path. The two tap input is converted to a standard raster scan by a line buffer module at the input, which accepts the two converging inputs and outputs a raster order frame at two pixels in parallel per clock. The video processing blocks that ABC has already developed generally expect a single pixel per clock, so all other processing (demosaicing, color correction, white balance, etc.) must be performed after the buffering block has converted the video to a suitable single pixel per clock format. As a result, the video pipeline within the FPGA must follow the flow shown in Figure 2. Depending on the choice of buffering block, some additional blocks may be need to provide memory access, however these are essentially transparent to the video path itself.

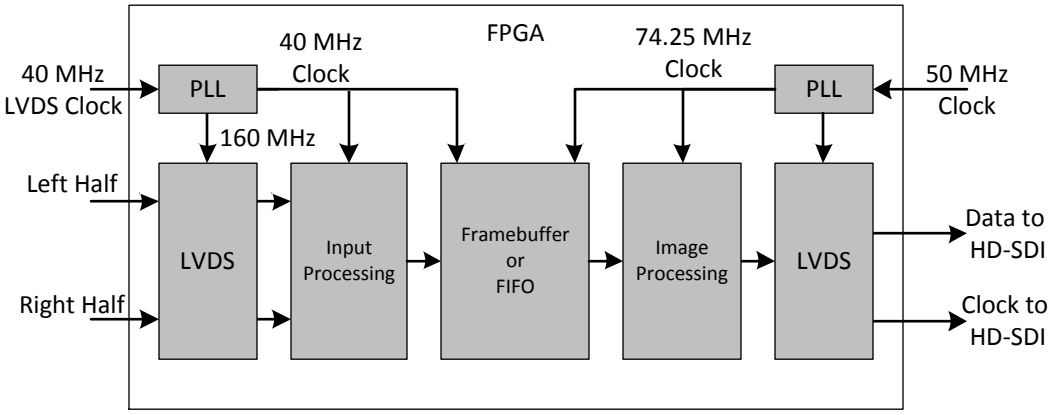

Figure 2: FPGA video path block diagram

1.2 Scope

The primary goal of this report is to examine the tradeoffs between two approaches to provide the video buffering, general requirements for which were identified in the previous section. Various metrics will be used to analyze the two approaches. These metrics will include both purely quantitative metrics and quantitative assessments of subjects such as flexibility and complexity.

The solutions to be evaluated are limited to two distinct options which were considered and implemented during the course of the actual project timeline. Although other options for buffering exist and some were considered during the project, they do not represent a significant architectural departure from the two options that will be examined. As a result, it is not expected that analysis of these options will produce significantly different outcomes.

Quantitative analysis will focus on two main aspects of the system. The first area of investigation is FPGA resource utilization. Certain undisclosed aspects of the project led to the potential for significant resource pressure in the implementation of the system. Since the FPGA on the development system is fixed, such issues can require significant rework if not managed properly from the beginning. The second area of investigation is power consumption. System power consumption can have significant architectural and cost related effects. Qualitatively, the implementation complexity of the solution, as well as its flexibility will be examined. The complexity of implementing a solution can be a major driver of risk, especially when dealing with short deadlines. The flexibility of a solution determines the potential for its reuse, which helps to amortize implementation costs over the long term.

2 Methodology

In order to evaluate each option, four distinct quantitative data points will be used. These will include the number of FPGA logic cells used by the option, the number of internal FPGA memory blocks, known as M9Ks used, the number of pins used and the estimated power consumption of the relevant blocks.

In order to collect the required data for analysis, compilation information for two variants of the project FPGA will be used. The only substantiative difference between these variants will be the video buffering and conversion approach used. The compilation reports will be used to derive information regarding the resource utilization of the FPGA. This information will both directly inform the analysis of resource utilization metrics and also indirectly assist in the calculation of estimated power consumption. Power consumption will be calculated using the Altera PowerPlay Early Power Estimator [6], which will allow the power consumption of just the blocks of interest to be derived. If external components or I/O are required for an option these power figures will also be calculated and added to internal FPGA dissipation.

Once the quantitative data points are collected, they will be combined using a computational decision making matrix to produce a single cost number for each potential solution. As all quantitative metrics being collect are already positively correlated to cost, no conversion of metrics is required. The data points will be normalized within the data set, then each solution's costs will be scaled and summed by weight. The resulting solution cost data will help to inform final conclusions and recommendations.

3 Requirements and Solutions

The requirements for any potential video buffering and conversion block are fairly simple. Any solution must convert the input frame, with an active area of 1920x1080 delivered in a frame with a first line of 3404 pixels (1702 clocks), and 1146 subsequent lines of 2284 pixels (1142 clocks) delivered at 40 Mhz with 2 pixels per clock into the output frame. The output frame must be compliant to the 1080p30 video format specified in SMPTE 274M system 7 [2] specifically an, active area of 1920x1080 in a frame of 2200x1125 delivered at 74.25 MHz. The details of both frames are shown in Figure 3. Both input and output pixels are 12 bits wide. There is some flexibility in the framing of the input data if required. In order to simplify the implementation of 2D spatial filters in the video pipeline, flexibility in the vertical front and back porch of the output frames is also required.

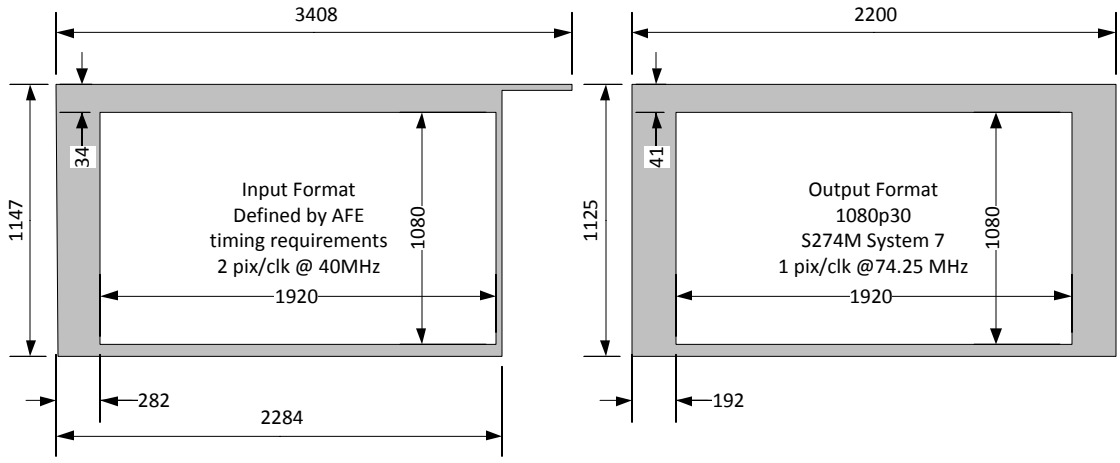

Figure 3: Video frame formats. Horizontal dimensions line pixels, vertical dimensions in lines

All solutions must also operate within the synchronization constraints of the system's video synchronization architecture. Specifically, in order to ensure that the entire video pipeline remains frame locked, while also ensuring that perfectly timed video is delivered to the HD-SDI transmitter, the system is synchronized through a master timing generator operating the 74.25 MHz domain. This timing generator counts out precisely $2200 * 1125 = 2475000$ clocks, generating a synchronization pulse with a falling edge one per period. This pulse is then transfered into the 40 MHz clock domain and sent to the AFE chip, where it resets the internal horizontal and vertical counters, starting the readout of a new frame. The sync signals from the AFE are then driven through the FPGA into the input of the video buffer. In

order to avoid jitter in the timing of the output frame, the output timing generator must be free running, but from the same 74.25 MHz clock responsible for the master timing generator. Slaving the output timing to the input timing will result in jitter due to the multiple clock crossings between the 74.25 MHz and 40 MHz domain. For clarity the synchronization architecture is also shown in Figure 4.

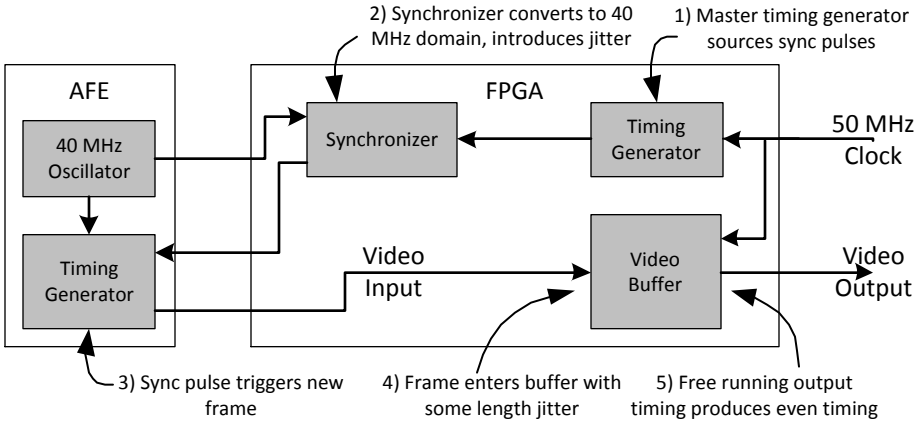

Figure 4: Video synchronization architecture

These two components describe the hard requirements that any potential solution must meet. Neither requirement is particularly difficult to design for and most well designed buffering solutions should accommodate them reasonably easily. The primary concerns in relation to these requirements would be the decoupling of the input and output frame timing and the reconfigurability of the input and output timing.

3.1 DDR2 Frame Buffer

The first potential solution to the video buffering requirements is a traditional frame buffer. A frame buffer operates by storing the data for one or more full frames in memory. This data can be updated as input data becomes available and read out as required for output. In order to prevent concurrent reading and writing of the same frame, which can result in visual artifacts such as tearing, at least two frames are typically stored. This allows one frame to be read while the other is written. Framebuffers are able to provide good decoupling between the input and output since the frame input and readout processes are almost entirely independent.

Based on 12-bit pixel data (expanded to 16-bits to simplify alignment) and storing just the active image data, the memory requirements can be calculated as $1920 * 1080 * 2\ bytes *$

$2\ frames = 8,294,400\ bytes$ or 8 MB. This greatly exceeds the amount of memory available with in the FPGA fabric, so an external memory is required. The memory bandwidth requirements are $1920 * 1080 * 2\ bytes * 2 * 30\ Frames/s = \ 238\ MB/s$. The development board being used features an on board DDR2 memory with a capacity of 64 MB, organized as 32 Mb deep by 16-bits wide, which provides sufficient space for the image data. The use of a DDR2 memory imposes two major constraints on the framebuffer. The first constraint is latency tolerance, since accessing a DDR2 memory has a relatively high latency, tens to low hundreds of cycles. The second constraint is access efficiency, since DDR2 memory requires long burst reads and writes to reach full bandwidth. Both of these constraints can be dealt with using FIFOs to buffer write data and prefetch reads.

Since no other elements of the FPGA design require the use of external memory, when determining the resource use of the framebuffer, the resources used by the memory controller must also be considered. In addition, the power consumption of the DDR2 memory itself must also be considered. The full calculation of the power consumption for a DDR2 memory and controller is quite complex and dependent on termination schemes as well as access patterns. In order to simplify the calculations, the Micron DDR2 System-Power Calculator [7] is used to calculate the approximate power consumption of the memory. Based on the memory bandwidth requirements a DDR2 clock rate of 150MHz is selected. The relatively low clock rate conserves power, while still exceeding the minimum specification of 125 MHz for most DDR2 devices and being easy to generate. At this speed the maximum theoretical transfer rate is $150\ MHz * 2 * 2\ Bytes = 600\ MB/s$, although this rate can never be attained in practice, due to refresh cycles and command latency. This theoretical number does however allow the percentage of time being spent in burst reads and writes to be calculated. Based in the calculations performed above, reads and writes each require a bandwidth of $238/2 = 119\ MB/s$, which requires data transfer on $119\ MB/s / 600\ MB/s \approx 20\%$ of clock cycles in each direction. Based on these numbers and the memory on the development board, an average power consumption of 187.8 mW can be calculated.

Table 1 shows the resulting resource usage and estimated power consumption for the framebuffer solution. As can be seen in the table, the majority of the power consumption and resource usage comes from the requirement for a DDR2 controller and the power consumption of the memory chip itself. The framebuffer consumes a comparatively small portion of the FPGA resources. This suggests that the resource requirements may be mitigated in scenarios where multiple buffers are required, given that only 40% of memory bandwidth is used, even with low memory speeds.

Table 1: Framebuffer resource usage and power consumption

Element	FPGA Resource Usage			Power Consumption [mW]
	Logic Cells	M9Ks	Pins	
Framebuffer	1142	4	0	
DDR2 Controller	5625	9	47	186
Memory Arbiter	227	0	0	
FPGA Total	6994	13	47	240
DDR3 Memory				188
Total	6994	13	47	428

3.2 Internal FIFO

The second potential solution to the video buffering requirements is a FIFO based approach. Given the synchronization requirements, in particular the precise frame locking of the input and output, as well as the free-running of the output timing generator, it can be easily shown that a FIFO capable of buffering no more than one frame can be used. This approach simply pushes all pixel from the active image area into a FIFO as they are received in raster order, then as active area pixels are required at the output pulls pixels out of the FIFO. With some synchronization logic to ensure that the relative timing of FIFO reads and writes results in correct alignment, the data is cleanly buffered between clock domains. This approach results in very simple logic to control the ingress and egress of data. In fact, with the modification of using two FIFOs, and swapping them between read and write roles on frame boundaries, the same logic also models the framebuffer described above.

Although the FIFO approach can easily be shown to require only a single frame of memory for buffering, this amount of data (4MB) still exceeds the capacity of the internal memory blocks in all but the largest FPGAs. Even given a device with sufficient capacity, using the internal memory for video buffering would not be economical and external memory would still be required. However, careful analysis of the video timing in this system can will show that only a small amount of memory is actually needed if certain timing changes are made.

Figure 2 shows a set of suitable modifications to the AFE timing and the resulting timing differences between input and output. Although the output timings do not match the previously discussed parameters, this is due to the way that the timing generation is handled and not due to any changes in final output format. Two key items have been modified in the timing of the AFE output in order to minimise the the required buffering. The most important item is that the number of clocks per line has been adjusted so that duration of

each line is closely matched to the output, this limits the accumulation of error over the course of the frame. The second change is to extend the first line in order to reduce the amount of mismatch when the first active pixel arrives. Because the input side is configured to have a slightly shorter line duration than the output the overall combination of these two effects is to reduce the amount of buffering required. The initial delta is not reduced further since some slack is needed due to latencies in the video pipeline and since the results of the delta shown are sufficiently low. The initial delta is just under 543 pixels, with the final delta growing to just under 3250 pixel, with the input always leading the output. The final delta of 3250 pixels determines the resulting minimum FIFO depth. The Altera FIFO generation wizard prefers power-of-two FIFO depths, so a 12 bit wide FIFO with a depth of 4096 entries can be used. Such a FIFO requires $12 * 4096 = 49152\ bits = 48\ Kb$ of ram. This is equivalent to 6 M9K blocks, which provide 8 Kb of memory when used in power-of-two widths, as would be used to create this FIFO.

Table 2: Timing analysis for FIFO based buffering

Metric	CCD Timing (40 MHz)			274M S7 (74.25 MHz)			Delta	
		Clocks	μs		Clocks	μs	μs	Pixels
1st Line Length	14000	14000	350	2200	2200	29.63		
Line Length	1184	1184	29.60	2200	2200	29.63		
First Active Line	34	53072	1326.8	45	99000	1333.3		
First Active Pixel	141	141	3.5	280	280	3.8		
First Active Clock		53213	1339.3		99280	1337.1	6.78	542.4
First Active Line	1113	1330608	33265.2	1124	2472800	33303.7		
First Active Pixel	1101	1101	27.5	2200	2200	29.6		
First Active Clock		1331709	33292.7		2465000	33333.3	40.6	3248.7

The overall resource consumption of the FIFO approach is quite low as can be seen in Table 3. Because it does not have to interface to an external memory controller the logic usage of the buffering block is reduced compared to the framebuffer. As calculated above, the internal memory usage is also quite low. Due to the optimization of the video timing only a small amount of buffer is required. This timing optimization is one of the main weaknesses of the design, since many video sources cannot be optimized in this way.

Table 3: FIFO resource usage and power consumption

Element	FPGA Resource Usage			Power Consumption [mW]
	Logic Cells	M9Ks	Pins	
FIFO Buffer	536	6	0	27
Total	536	6	0	27

4 Analysis

To allow the quantitative analysis to produce a single cost metric for each solution, the collected data points must be weighted and summed. In the case of this report, data will be weighted based on the assignation of a rating to each quantitative data point of a relative importance level. All items are weighted such that they have a reasonable level of influence on the final outcome.

For this project, the number of logic cells used is weighted at 30% of the total, due to the potential of this project to become resource constrained on logic. The number of M9K blocks is weighted at 40% since the other logic in the system puts more pressure on this resource, and because the very useful SignalTap debugging tool also puts pressure on this resource. In contrast, pin usage is only rated at 10% weight, since this is an existing board and the primary cost of pin usage in this case is verification and debug time if issues arise. Power consumption is rated at 20% weight, reflecting a desire to keep overall power consumption under certain limits so existing power adapters can be used.

Each input metric is assigned a final weight w_i based on the relative weights above and chosen so all weights sum to 100%. In order to scale the input data points, c_{ij}, proportional points p_{ij} are created according to $p_{ij} = c_{ij} \div max(c_{i1}, c_{i2}, \ldots, c_{in})$. Each proportional data point is then scaled be weight to provide a final metric, f_{ij},according to $f_{ij} = p_{ij} \cdot w_i$. These final weights are then summed to provide a total cost for each solution. The results of these calculations can be seen the the decision making matrix shown in Table 4.

Table 4: Decision Matrix

		Framebuffer			FIFO		
Factor	w_i	c_{i1}	p_{i1}	f_{i1}	c_{i2}	p_{i2}	f_{i2}
Logic Cells	30%	6994	1.00	0.30	536	0.08	0.02
M9Ks	40%	13	1.00	0.40	6	0.46	0.18
Pins	10%	47	1.00	0.10	0	0.00	0.00
Power [mW]	20%	428	1.00	0.20	27	0.06	0.01
Total	100%			1.00			0.22

As the table shows, the FIFO based solution outperforms the framebuffer approach on every metric, with a minimum margin of more than a factor of two. The final cost of the FIFO solution is less than one quarter that of the framebuffer approach. By all metrics amenable to quantitative analysis, the FIFO solution is superior to the framebuffer solution.

From a complexity standpoint, the framebuffer approach also looses out to the FIFO.

The requirement for integration and configuration of the DDR2 controller offers an immense area for problems to arise. Since the DDR2 controller is third party IP, the visibility into it for debug is also reduced and it is not possible to simply walk across the office and discuss any issues with the original designer of the block. The FIFO approach does have some complexity drawback however, as it depends on timing relationships between elements in the video pipeline to a much greater degree than the framebuffer. As a result more of the video path must be operating correctly before the FIFO can operate. Overall however, the complexity of the framebuffer is higher.

Flexibility is the one aspect where the framebuffer is substantially better than the FIFO. The FIFO depends quite heavily on the relationships between input and output timings, with little tolerance for errors and misconfiguration. This can be somewhat alleviated by using a deeper FIFO, but this approach will quickly run into resource limitations. The extent of the flexibility limitations of the FIFO approach is such that the existing test patter generator in ABC's IP library is difficult to use when driving the FIFO. When performing simulations, the full frame size must be used unless the developer is willing to recalculate the timings. This significantly slows the simulation process, increasing verification time. In contrast, the framebuffer is quite flexible about video timings, essentially requiring only matching frame periods and active image areas. This greatly increases potential for reuse, in fact if the framebuffer is used in this application it will be a reuse of an existing block.

5 Conclusions

From the analysis in the report body, it was concluded that the FIFO approach produces the lower resource utilization and power consumption of the two options considered. In all metrics considered, specifically logic cells, memory blocks and pins, the FIFO approach has significantly lower resource utilization. The power consumption of the FIFO approach is less than the framebuffer by more than an order of magnitude.

Another aspect that was examined was complexity. In this regard the FIFO approach was also superior, largely due to not requiring and external memory interface. The memory interface incurs a significant amount of integration complexity due to the presence of third party IP and off-chip interfaces. By comparison the FIFO system is self-contained and uses only a simple FIFO generator sub-block.

The final aspect examined was flexibility. This aspect shows the tradeoff of the FIFO approach, as it requires a narrow set of video timing parameters and relationships to work properly. The framebuffer is already adapted from a different project and can handle a broad range of video formats and timings.

6 Recommendations

The primary recommendation of this report is that the FIFO approach be used to implement the video buffering needed for this project. This recommendation is based on the reduced resource utilization of the FIFO approach, as well as its lower complexity. Given the tight timeline of the project, the reduced complexity in particular is a key advantage as it should reduce the risk of schedule slips due to unforeseen integration difficulties. Due to the simplicity of the hardware for the FIFO approach a from scratch implementation may take less time than the integration effort required for the framebuffer.

A secondary recommendation of this report is that if the framebuffer approach is selected for use, integration and test of the framebuffer on the target board be prioritized and allocated extra debug time. The complexity of the interfacing between the framebuffer core, the memory controller and the memory itself presents a significant integration risk. One potential way to reduce the risk may be to ensure that the same tool versions used for FPGA compilation are the same as those previously used for the framebuffer block.

Glossary

AFE - Analog Front End, a integrated circuit containing analog conditioning circuitry and an analog to digital converter.

DDR2 DRAM - A type of dynamic memory utilizing high-speed source-synchronous interfaces and transferring data on both clock edges.

FIFO - First-in, First-out, a type of queue where items exit in the order they are stored.

FPGA - Field Programmable Gate Array, a type of programmable logic device, typically a relatively large device based on look-up tables.

HD-SDI - High Definition Serial Digital Interface, a broadcast grade video interface running at 1.5 Gb/s.

LVDS - Low Voltage Differential Signaling, a signaling system allowing high speeds over differential pair electrical interfaces.

PLL - Phase Locked Loop, a type of clock conditioner and generator allowing multiple clocks with carefully controlled relationships to be generated.

SMPTE - Society of Motion Picture and Televison Engineeers.

VHDL - VHSIC Hardware Description Language.

References

[1] "KAI-02150 Image Sensor Datasheet," Eastman Kodak ISS, July 2011. [Online]. Available: http://www.kodak.com/global/plugins/acrobat/en/business/ISS/datasheet/interline/KAI-02150LongSpec.pdf

[2] *Television - 1920 x 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates*, SMPTE Std. ST 274M-2003, August 2003.

[3] *Television - Bit-Serial Digital Interface for High-Definition Television Systems*, SMPTE Std. ST 292M-1998, October 1998.

[4] "Cyclone IV Device Handbook," Altera Inc., November 2011. [Online]. Available: http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf

[5] "512Mb: x4, x8, x16 DDR2 SDRAM," Micron Technology Inc., July 2011. [Online]. Available: http://micron.com/parts/dram/ddr2sdram/~/media/Documents/Products/Data%20Sheet/DRAM/455512MbDDR2.ashx

[6] "Cyclone III, Cyclone IV and Cyclone V PowerPlay Early Power Estimator," Altera Inc., November 2011. [Online]. Available: http://www.altera.com/support/devices/estimator/cy3-estimator/cy3-power_estimator_download.html

[7] "DDR2 SDRAM System-Power Calculator," Micron Technology Inc., January 2010. [Online]. Available: http://micron.com/~/media/Documents/Products/Power%20Calculator/4298ddr2_power_calc_web.ashx