INTELLIGENT AND EFFICIENT EMBEDDED VIDEO MEMORY DESIGN IN THE ERA OF BIG DATA

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Jonathon David Edstrom

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

April 2017

Fargo, North Dakota

North Dakota State University
Graduate School

Title

INTELLIGENT AND EFFICIENT EMBEDDED VIDEO MEMORY DESIGN IN THE ERA OF BIG DATA

By

Jonathon David Edstrom

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Na Gong
Chair

Jinhui Wang

Zhibin Lin

Approved:

4/5/17	Scott C. Smith
Date	Department Chair

ABSTRACT

The growing popularity of smart phones, tablets, and other mobile devices has created an exponential demand for video applications in today’s society. Mobile device users want their devices to achieve a long battery lifetime while also being able to display high quality video. When mobile devices display video, the embedded video memory in the device consumes a large amount of power. Therefore, in this thesis, we present multiple novel, power-quality tradeoff techniques for enabling good quality video output, while simultaneously enabling power consumption reduction in order to maximize the lifetime of the battery. The described techniques are designed using minimal area overhead and are compared against recent, related works by researchers in the area of low power memory design.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Na Gong, for her time, guidance, and knowledge in completing all the works presented in this thesis. I would also like to thank my other committee members, Dr. Jinhui Wang and Dr. Zhibin Lin for their feedback and assistance in presentations, papers and other advice during my Masters study. I am thankful for all the help provided by the other members of my lab, especially Dongliang Chen and Yifu Gong. Their help with preparing experiments, calculating simulation results, and verifying designs was paramount in completing the necessary work for conferences, journals, and graduation.

I am grateful to Dr. Mark McCourt and the Department of Psychology at NDSU for allowing us to use their facility for psychophysical tests, without their help the process for obtaining results would have taken much longer and been much more difficult.

Last, I would like to recognize and thank the National Science Foundation and ND EPSCoR for the financial support from that made this research possible.

DEDICATION

To my parents, Fred and Terri, who have always been there to support me in everything I do.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CMOS	Complementary Metal-Oxide-Semiconductor
SRAM	Static Random Access Memory
ECC	Error-Correcting Code
LSB	Least Significant Bit
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience
VQA	Video Quality Assessment
HVS	Human Visual System
MSB	Most Significant Bit
I^2C	Inter-Integrated Circuit
ADC	Analog-to-Digital Converter
D-DASH	Data-Driven Adaptable SRAM Hardware
MB	Megabyte
MSE	Mean Squared Error
SSIM	Structural Similarity
RBL	Read Bit-Line
V_{dd}	Supply Voltage
RWL	Read Word-Line
NMOS	N-Type Metal-Oxide-Semiconductor
GND	Ground
DPSR	Data Pattern Self-Recovery
RDF	Random Dopant Fluctuation
MUX	Multiplexer
POST	Power-On Self-Test

CHAPTER 1. INTRODUCTION

The research and development of energy-efficient memory for general use or application specific designs has recently become of great interest [1]. There is now a need for new devices that are capable of saving power intrinsically, while maintaining a robust, minimal failure operation [2]. The detailed description of multiple techniques to allow for these savings, including partially disabling circuitry or minimizing supply voltage to increase power-efficiency and the use of self-correction techniques to mitigate errors and provide a high quality output, will be introduced in the different chapters within this thesis.

Video applications have been shown to inherently possess error resiliency to some extent [3]. This error resiliency enables those types of applications to be redesigned using approximate computing methods for power savings. The majority of the memory designs described in this thesis deal with video applications in particular; but these designs can be adapted for use in devices that are a part of other application types as well. The different memories we have designed use CMOS technology, but the methodologies used to incorporate power savings and data correction may prove useful in future technologies as well.

In order to evaluate the effectiveness of video memories, we perform simulations to measure critical performance parameters such as power efficiency, video quality, and area overhead. The relationship between these three metrics is shown in Figure 1. By constraining two of the three metrics and then improving the third, then comparing against recent research, we are able to clearly show the improvements of our work against the state of the art. The following chapters describe the entire design process, for three separate works, including comparison against the latest research.

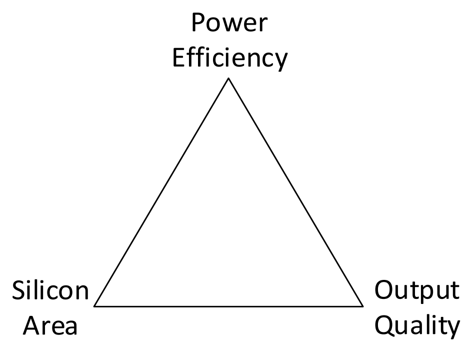


Figure 1. Trade-off triangle relationship of hardware evaluation metrics

The second chapter will describe the system implementation of a memory design capable of dynamically adjusting the amount of circuitry it will enable. The use of a luminance sensor that can detect the amount of light present within the devices environment will be used for changing which mode of operation the memory will be in. Previous works [4, 5, 6, 7] have shown the importance of different environmental scenarios and how they influence the user as they are viewing video on mobile devices. Based on the knowledge that the environmental surroundings impact the user’s perception, an intelligent bit truncation technique is proposed. This technique involves disabling part of the memory circuitry based on the amount of luminance in a present scenario. Using this technique, it is possible to reduce the power consumption of the memory by a maximum of 32.6% while still maintaining good quality video output [8].

The third chapter introduces the use of a data-mining based pattern discovery technique used to discover meaningful patterns in video data. Other low power SRAM designs have been developed for mobile video applications. 6T/8T and 8T/10T hybrid SRAM structures for optimizing power efficient mobile video streaming are presented in [9] and [10], respectively. A heterogeneous sizing scheme is presented in [11] to reduce the failure of conventional 6T bit-cells. In [12], a two-port SRAM using majority logic and data reordering is presented to minimize power consumption. The major issues with these works is their large area overhead and inability to adjust their power-quality tradeoff, since it is set during the design of the SRAM. Recently in [3], a voltage-scaled SRAM that can dynamically tradeoff between power and video quality through the use of a write assist technique and ECC. The encoder and decoder circuits used for implementing these techniques results in large computational complexity and silicon area. We use these previous works for comparison to our presented data-mining enabled design technique in the discussion chapter. The presented design for this memory exploits the use of discovered patterns in order to maximize power savings up to 43.7% with negligible area overhead [13].

The fourth chapter discusses data-pattern mining techniques that are used to discover rules that allow for self-correction functionality. Recently, new low power memory designs that make use of the inevitability of faults at scaled voltages, versus completely avoiding the faults, have gained popularity. A shifting technique is presented in [14] that always stores the LSBs in faulty cells in order to minimize output quality degradation. In [15], a squeezing technique is presented that compresses zeros to store them in less memory space, allowing the memory to avoid memory failures at low supply voltages. These works produce good quality results when implemented, but at the cost of complex operations and high area overhead. We use these previous works for comparison to our presented self-correction technique in the discussion chapter. When our self-correction technique is applied to a voltage scaled, video-based memory, the hardware can gain error resiliency, allowing it to output good quality video at near threshold voltages [16].

All information, tables, and figures in chapters 2, 3, and 4 are either directly taken or adapted, with permission to re-use, from [8], [13], and [16], respectively. The final chapter discusses the comparison of these techniques with state of the art designs in other recent works.

CHAPTER 2. LUMINANCE-ADAPTIVE SMART VIDEO STORAGE SYSTEM[1]

When designing mobile devices with modern advances in technology, two important aspects to keep in mind are the user's experience when using the device and the duration that the battery will be able to power the device. This section describes the use of environmental lighting in combination with adjustable video output in order to verify the potential for power savings when designing a novel memory architecture for storing video data. We examine the influence of ambient lighting on viewing experience for users, specifically the introduction of noise at different levels and the influence this has on the subjective video quality users perceive. The sensitivity of the human visual system is less when higher amounts of ambient luminance is present, therefore the amount of noise in higher luminance scenarios is increased in order to improve energy efficiency. We describe the system design, testing and verification process in detail. The final design simulation shows potential for 32.6% power savings with good subjective quality of experiences.

Quality of Experience Factors

The use of mobile devices for watching videos has greatly increased in recent years. In fact, according to a study by Cisco, between the years 2012 and 2014 there was a 400% increase in mobile video views [17]. With this in mind, mobile video is becoming one of the most important sources of information, and will continue to grow as time goes on. Now that people are using their devices extensively for viewing mobile videos, the focus of this particular section is to describe a way to supply mobile device users with good quality video while also maintaining a long battery life.

[1] The material in this chapter was co-authored by Jonathon Edstrom and Dongliang Chen. Jonathon Edstrom held the primary responsibilities of building and programming the system, collecting data, and verifying results. Dongliang Chen provided the presented SRAM hardware design with simulation results based on the concept verified by the system implementation.

User Experience

First, we perform an investigation of the quality that determines the user’s experience while using mobile devices. The typical metric used for analyzing the quality of video is the PSNR. The PSNR is a simple and widely used calculation that has a clear meaning to the experimenter, but research has shown that the PSNR may not fully encompass the complete QoE based on environmental conditions that are present in the surroundings of the mobile device’s user. This has been verified by psychophysical experiments that show PSNR might not be the best metric for calculating the quality of videos, especially in viewing conditions that have varying levels of environmental factors such ambient luminance [4, 5, 6, 7]. Xue *et al.* found a similar lack of correlation between VQA models and the true quality perceived by users and because of this, they conducted multiple subjective tests in order to test how the environmental factors played an influence in the output quality of video being displayed by mobile devices. The VQA models typically included some objective metric, such as PSNR, and did not take the contextual influences into account [4].

Battery Life

Second, we examined the battery life of mobile devices while viewing video. The memory included on mobile devices used for storing video data is one part of the system that consumes the most power [10]. Reducing the energy consumption of this power hungry component can potentially lead to high gains in battery life. By extending the total amount of energy available on a single charge of the battery included with the mobile device, user’s will be able to continue to watch mobile videos or use their devices for other functionality for longer, thus this is an important aspect of the research proposed in this design.

Impact of Luminance on Quality of Experience

Three factors that affect the human perception of videos: display size and viewing distance, user movements, and ambient luminance. While the QoE of the user can be noticeably influenced by all these factors [4], this section focuses on the ambient luminance as one specific type of environmental context. The main objective is to examine the luminance of the user's surroundings while a mobile device is in use to view mobile videos, and to use this knowledge to reduce power consumption of the mobile device while maintaining good video quality. By using multiple scenarios with varying levels of luminance as a test environment, it is possible to introduce the design and system implementation of a smart video memory that has the capability of dynamically changing video output quality using bit truncation.

There are a number of influential factors regarding luminance involved with mobile devices. The luminance level of the mobile device's display, the surrounding ambient luminance of the environment, and the reflection off the screen will all have an impact on how the user's perceives video images on the device. Previous works, such as [18], define models that describe how brightness corresponds to perception within the HVS. We can use these models to adapt the quality of displayed videos on mobile devices with the use of ambient luminance levels to optimize power efficiency.

Video Format, Processing, and Conversion

Raw YUV 4:2:0 format video were used for all testing and verification of the system design. In this format, the luminance or luma (Y) component data has a byte of data for each pixel and the chrominance or chroma components have one byte per four pixels for each chroma component (U and V) as can be seen in Figure 2.

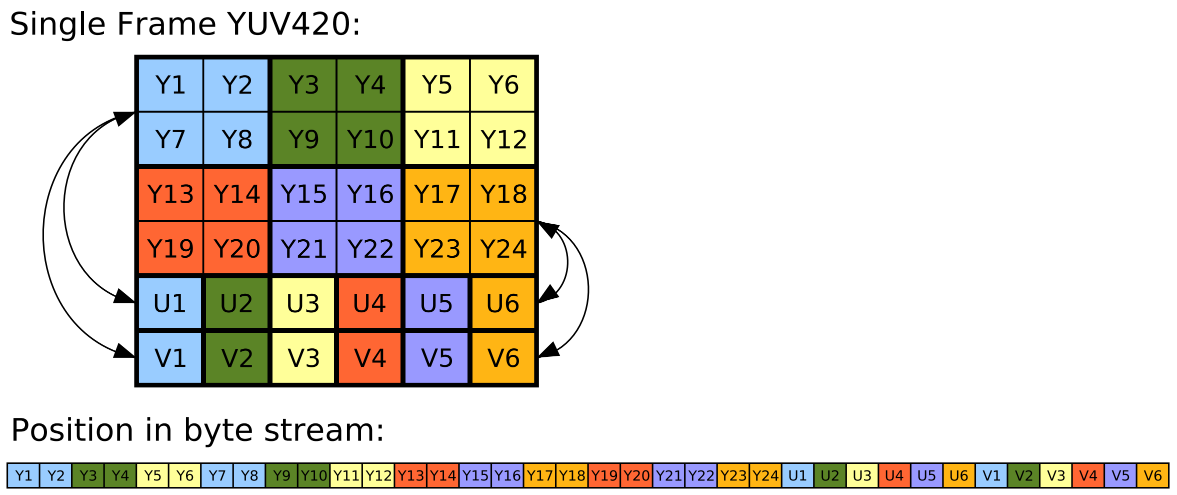


Figure 2. YUV 4:2:0 frame components and corresponding byte stream[2]

For a better understanding of how the data is organized in the memory an example, full color image, is displayed in Figure 3 with a visualization of how each component is stored sequentially in memory. Similar to the byte stream shown in Figure 1, for any single YUV 4:2:0 frame, the luma bytes are first stored, then the two chroma components bytes are stored contiguously afterwards in the memory. This separation of each component when stored in memory is illustrated in Figure 3.

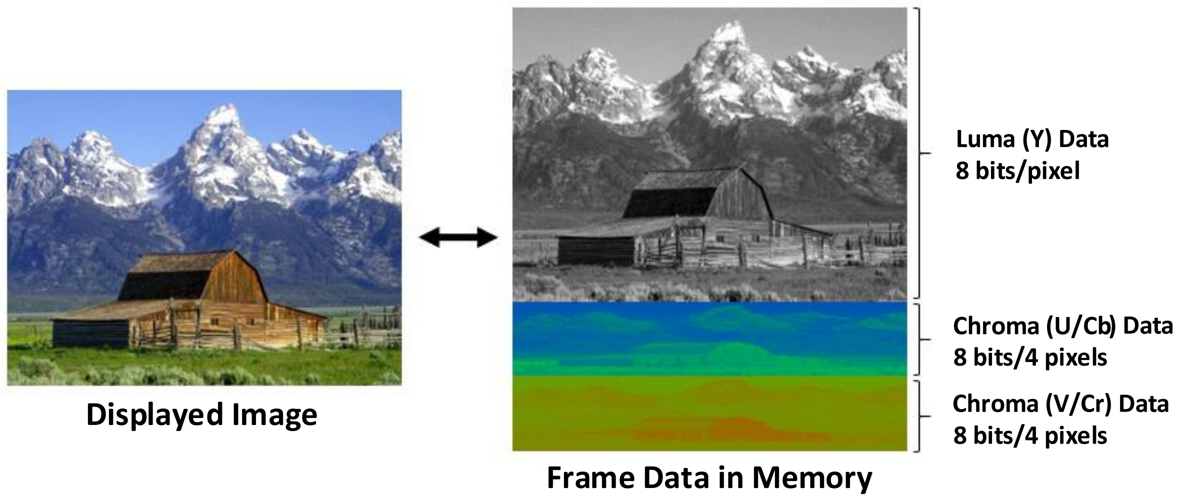


Figure 3. In-memory byte organization for a YUV 4:2:0 frame

[2] Public domain image obtained from: https://en.wikipedia.org/wiki/File:Yuv420.svg

The proposed method is to truncate more of the LSBs when there is a higher level of ambient luminance present within the mobile device user's surroundings. The truncation of LSBs causes less degradation in output video quality in comparison to truncating the MSBs. For the system implementation bit masking was used in order to achieve a working truncation process. This bit truncation process is discussed in detail, with example code, in section 2.3.2.

Perception of the Output Video by the User

The HVS dynamically adjusts the amount of luminance it takes in based on a logarithmic scale. The two photoreceptors that are responsible for this type of adaptation are called rods and cones and are located within the retina of each eye. Rods sense environmental brightness and respond more to dark and moderate light levels, calculated to be approximately between 10^{-6} and 10^{+2} cd/m^2. Cones on the other hand sense color in the environment and respond to dim to bright light levels, calculated to be approximately between 10^{-1} to 10^{+8} cd/m^2. LCD devices, such as the mobile devices in question, typically display videos in the luminance range of 10^{-2} to 10^{+2} cd/m^2, which is within the range of both retinal rods and cones [18, 19].

Environmental luminance directly influences the ability of the HVS to sense changes in contrast. When high levels of illumination are present and glare is introduced into a scenario where a user is viewing a LCD screen, the HVS loses contrast sensitivity as described in [4]. A recent work [20] shows losses in contrast sensitivity of 7% and 15% by user's viewing LCD devices in overcast and bright luminance conditions, respectively. Therefore, under these conditions, users may perceive unchanged video quality, even though the video has been degraded through the bit truncation technique. This useful noise-tolerance ability of the human eyes is useful when exploited to create a power-efficient mobile video storage implementation.

Determining the Number of Truncated Bits

In order to determine an optimal number of bits to truncate in different levels of ambient luminance, a Luminance Contrast model was developed based upon the RGB luminance. RGB Luminance (L_{RGB}) can be expressed as:

$$L_{RGB} = 0.3R + 0.59G + 0.11B \qquad (1)$$

The Luminance Contrast (Contrast) is then defined as:

$$Contrast = Diff_{Luminance} / Ave_{Luminance} \qquad (2)$$

The L_{RGB} is the RGB luminance, $Diff_{Luminance}$ and $Ave_{Luminance}$ are the luminance difference and average luminance, respectively. The *Akiyo* and *Foreman* benchmarks taken from [21] were used to calculate the luminance contrast. The results from these calculations are listed in Table 1.

Table 1. Calculated luminance contrast of benchmark videos

Benchmarks		*Akiyo*			*Foreman*		
Contexts	# Truncated Bits	$Ave_{Luminance}$	$Diff_{Luminance}$	*Contrast*	$Ave_{Luminance}$	$Diff_{Luminance}$	*Contrast*
Dark	0	90	1207849	13420	170	1428896	8405
	3	87	1229762	14135	166	1432373	8628
	4	82	1249489	15237	161	1418935	8813
	5	73	1269937	17396	150	1462527	9750
Overcast	0	70	837859	11969	150	1003382	6689
	3	67	861422	12857	146	1005861	6889
	4	62	876261	14133	141	997266	7072
	5	53	888169	16757	130	1027670	7905
Sunlight	0	60	239310	3988	140	286762	2048
	3	57	243927	4279	136	288654	2122
	4	52	247075	4751	131	284979	2175
	5	43	251374	5845	120	292400	2436

The results displayed in Table 1 show that the average luminance decreases by 20 and 30 lux in the overcast and sunlight scenarios, respectively. The luminance contrast also decreases by a large factor when larger levels of sunlight are introduced. The human eyes will notice less of the video distortion in higher levels of luminance, but as more bits are truncated from the video data, the more noticeable this contrast caused by distortion will be to the user.

From our analysis based on the RGB Luminance and Luminance Contrast models, it was possible to determine the optimal number of bits to truncate. The number of bits to truncate for each scenario are listed in Table 2 [22]. In order to verify our truncation process, we conducted a subjective test for different number of truncated bits in three viewing contexts: dark, overcast, and sunlight. 15 participants between the age of 18 and 30 with normal vision were asked to watch videos from each context. Once the participant finished viewing a sample, they were asked to respond whether they could tell if there was a quality difference between the samples shown. If they responded that there was no noticeable difference between sample videos at different qualities, the videos were considered to have achieved the same quality. Based on the user feedback received, it was possible to verify that the number of bits to truncate in each context scenario are indeed a good fit for good video output quality while also benefitting from power savings based on the truncation of video data.

Table 2. Luminance scenario definitions

Scenario	**Dark**	**Overcast**	**Sunlight**
Luminance (lux)	0-1000	1000-10000	10000+
Data Bits	00	01	10
Luma Truncation	xxxxxxxx	xxxxx000	xxxx0000

System Design and Implementation

This section provides a detailed explanation of the hardware and software implementation of the proposed smart mobile video storage system. This system consists of two separate sub-systems. The first sub-system is for receiving the ambient luminance of the surrounding environment and outputting context data bits to the second system. The second sub-system then uses this information to truncate bits of data from each pixel to be displayed in order to emulate the bit truncation process and then outputs the bit truncated frame to an LCD display.

Embedded Hardware Setup

The hardware for the first sub-system of the design consists of an Adafruit TSL2561 lux sensor breakout board that is connected to an Arduino Leonardo as input through the use of an I^2C communication protocol. The lux sensor uses an onboard ADC to send the analog data digitally to the Arduino development board. The Arduino has two digital pins defined as output that it feeds into the second sub-system to tell it which environmental scenario is present. The corresponding data bits can be seen in Table 2.

The other sub-system is a Raspberry Pi 2 development board that handles the majority of the computational processes. The two digital outputs of the Arduino board are mapped into two input pins on the Raspberry Pi 2 in order to define how many bits to truncate based on present luminance scenario. Both sub-systems and their interconnections can be seen in Figure 4.

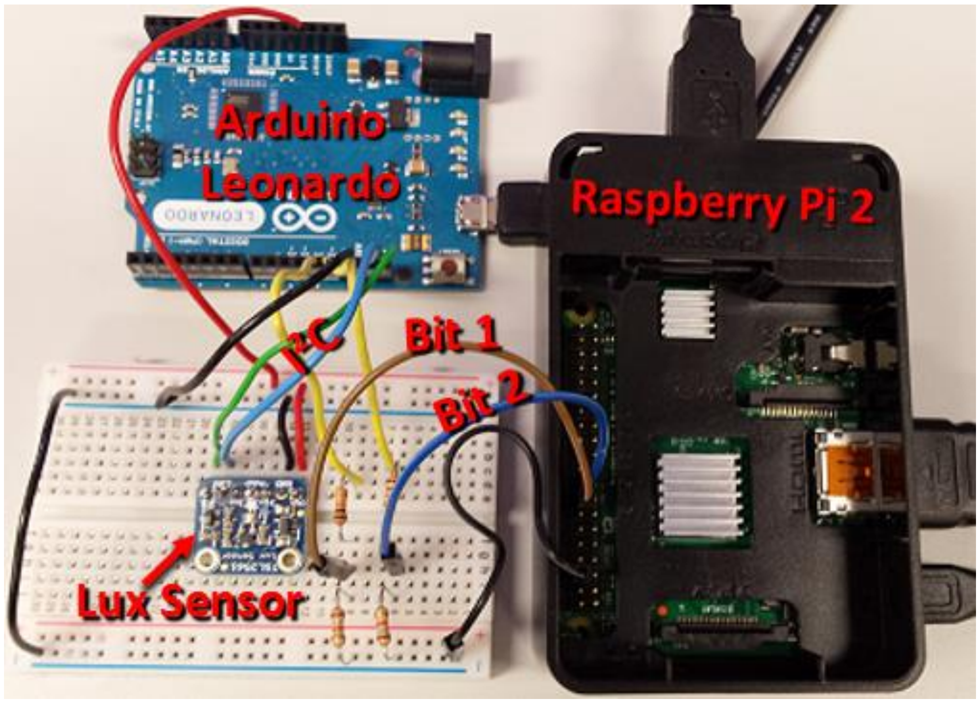


Figure 4. Hardware connections between both sub-systems

Embedded Software Utilization

The Arduino Leonardo embedded board reads the luminance value from the lux sensor, calculates which luminance scenario it is based on the analog reading and sends the corresponding data bits using two I/O pins connected to wires. These two wires are constantly being updated with the high or low voltages that represent the two bits that are being transmitted to the Raspberry Pi 2 board. A flowchart displaying this process can be found in Figure 5.

Video data is stored on a micro SD card that is plugged into the Raspberry Pi 2. For each frame in the video data, in the YUV 4:2:0 format, the Raspberry Pi 2 receives the luminance scenario, calculates the amount of bits to be truncated and truncates that amount of LSBs from every pixel's luma data in the frame. After this truncation process finishes, the frame is sent to the display via the HDMI output on the Raspberry Pi 2. The Raspberry Pi 2 repeats this process until there are no frames remaining in the video data. This process can be seen in Figure 5.

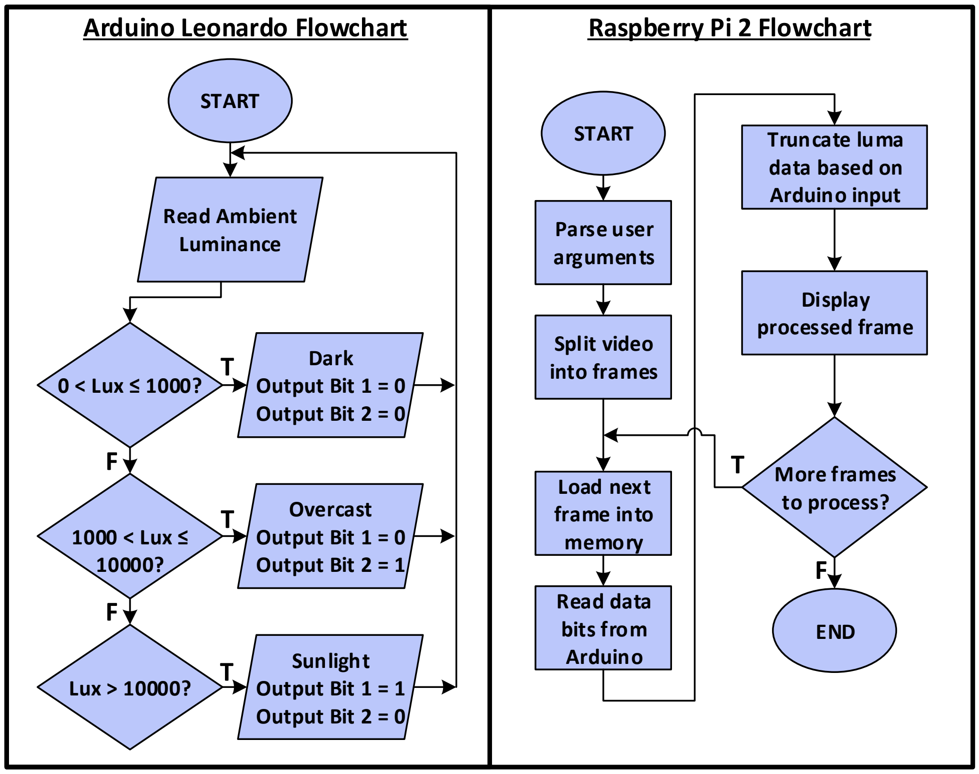

Arduino Leonardo Flowchart
START
Read Ambient Luminance
0 < Lux ≤ 1000?
T
Dark Output Bit 1 = 0 Output Bit 2 = 0
F
1000 < Lux ≤ 10000?
T
Overcast Output Bit 1 = 0 Output Bit 2 = 1
F
Lux > 10000?
T
Sunlight Output Bit 1 = 1 Output Bit 2 = 0
Raspberry Pi 2 Flowchart
START
Parse user arguments
Split video into frames
Load next frame into memory
Read data bits from Arduino
Truncate luma data based on Arduino input
Display processed frame
More frames to process?
T
F
END

Figure 5. Arduino Leonardo and Raspberry Pi 2 code flowcharts

Design Results

The finished system is able to play full length raw YUV 4:2:0 formatted video at any resolution, larger resolutions will not have as high of a frame rate since the bit truncation process is completed using software. When this design is implemented using hardware it will be able to achieve much faster frame rates for higher resolution videos. Sample output for zero (dark scenario), three (overcast scenario), and four (sunlight scenario) truncated bit frames can be seen in Figure 6.

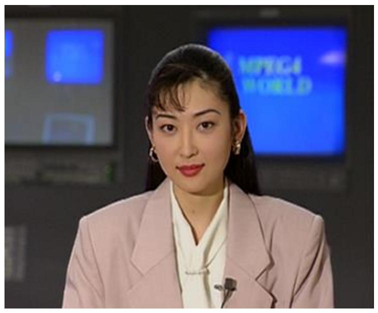
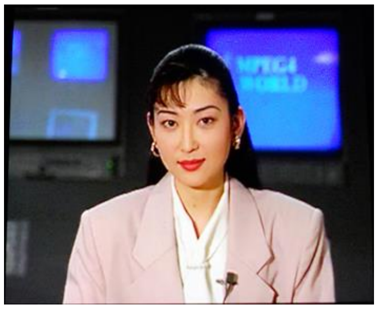
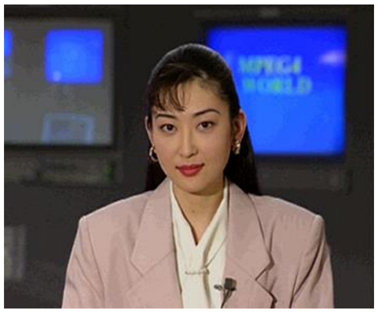
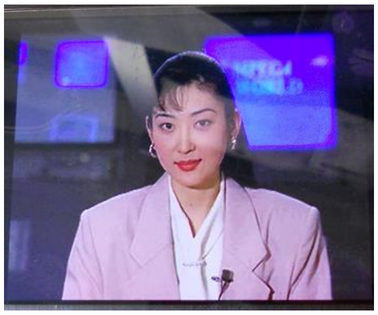
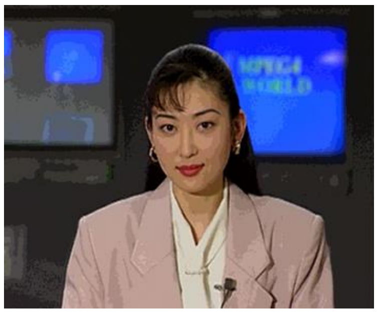
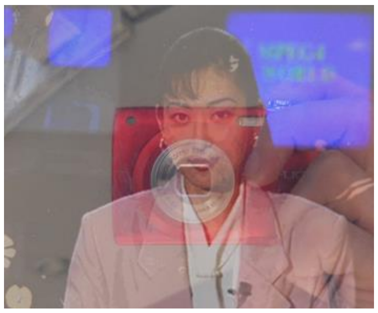
Image Type / Bits Truncated	Software Screenshot of LCD Screen (Print Screen)	Real World Screen Image (External Camera Image)
0		
3		
4		

Figure 6. Sample video output images from print screen and external camera

Hardware Design and Simulation

In order to evaluate the power efficiency of the proposed system implementation for the memory design, a memory chip was created that would follow the same procedure for truncating bits to save power. This memory chip can be seen in Figure 7. The simulation results of the 45nm CMOS technology design showed significant power reduction, reaching at most 32.6% power savings with negligible area overhead. The results from the hardware simulation can be seen in Table 3 [22].

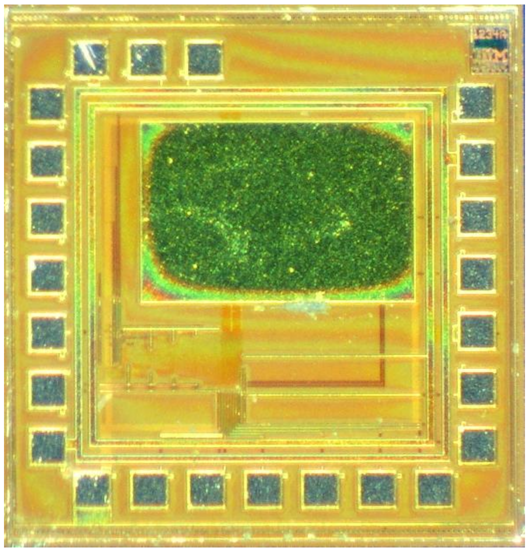


Figure 7. Physical memory chip image

Table 3. Memory power savings based on hardware simulation

Context	***Dark***	***Overcast***	***Sunlight***
Luma Truncation	xxxxxxxx	xxxxx000	xxxx0000
Write Power	2.88E-08	2.00E-08	1.88E-08
Read Power	1.11E-06	8.36E-07	7.49E-07
Power Savings	0%	24.8%	32.6%

CHAPTER 3. DATA-DRIVEN LOW-COST ON-CHIP MEMORY WITH ADAPTIVE POWER-QUALITY TRADE-OFF FOR MOBILE VIDEO STREAMING[3]

Video based applications can potentially drain the battery of a mobile device very quickly due to the high energy constraints of video hardware. The video decoding hardware includes frequent accesses to on-chip memory that can consume as much as 30% of the system power consumption and can occupy 65% or more of the entire video decoder area [23, 24]. A low-cost Data-Driven power efficient Adaptable SRAM Hardware (D-DASH) design with the capability for dynamic power-quality tradeoff for mobile video applications is presented. The proposed design is discovered through the use of advanced data-mining techniques that reveal useful data relationships that will be incorporated into the finalized hardware design for maximal power savings. The D-DASH design is broken down into three separate schemes, each giving its own power-quality tradeoff. The simulated hardware design allows for up to 43.7% power savings with a negligible area overhead of 0.06%. A detailed description of the data-mining techniques and their relationship with the resulting hardware design are presented in this chapter.

Mobile Video Data-Pattern Analysis

Mobile video application processes display the potential to relate application-level video data to its hardware-level equivalent. Three common characteristics that could assist in useful data relationships in the hardware-level design process are [25]: (1) inputs: the video data is noisy and redundant; (2) outputs: the videos on mobile devices are generated for humans and minor variations cannot be discerned by humans' eyes; and (3) computational patterns: statistical computations during the video decoding process potentially results in specific data patterns,

[3] The material in this chapter was co-authored by Jonathon Edstrom and Dongliang Chen. Jonathon Edstrom was in charge of all pattern discovery, data analysis, and video quality metric and simulation results. Dongliang Chen provided the presented SRAM hardware design with power simulation results based on the discovered patterns.

which can contribute to low-power hardware design. Hardware designers have the issue of being able to recognize these patterns due to the large volume of video data that is being processed by the mobile system. In order to alleviate the boundary introduced by this large data obstacle, association data-mining techniques are introduced in order to explore the characteristics of storing the video data in memory.

Data-Mining Assisted Video Analysis

In current devices, video frames are stored using the raw YUV format. This format includes one luma (Y) component, which contains the brightness information and two chroma (Cb and Cr) components, which contain the blue-difference and red-difference color information. A typical YUV 4:2:0 format frame of video data with resolution 352×288 is displayed in Figure 8. Every pixel within the frame contains 8 bits of luma data and 8 bits of subsampled chroma data. The data for this frame is stored in on-chip memory as binary bits and with access to these binary values we can perform association data-mining in order to discover bit-level patterns within the data.

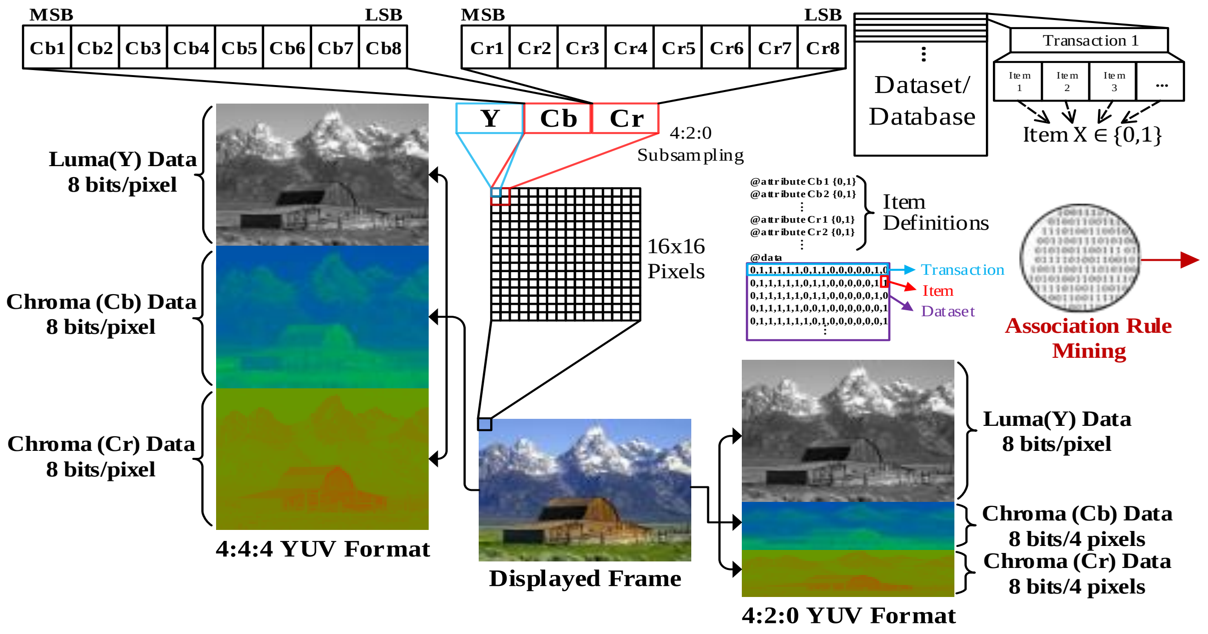


Figure 8. Data-mining assisted video data analysis

Table 4. Discovered data-mining association rules

Association Rules	***Support (%)***	***Confidence (%)***
Cr1=0 Cr2=1 → Cr3=1	38.341	94.050
Cr1=0 Cr2=1 Cr3=1 → Cr4=1	29.120	75.950
Cr1=0 Cr2=1 Cr3=1 → Cr5=1	28.270	73.731
Cr1=0 Cr2=1 Cr3=1 → Cb2=0	28.132	73.373
Cr1=0 Cr2=1 Cr3=1 → Cb3=0	25.450	66.377
Cr1=1 Cr2=0 → Cr3=0	57.153	98.798
Cr1=1 Cr2=0 Cr3=0 → Cr4=0	54.707	95.720
Cr1=1 Cr2=0 Cr3=0 Cr4=0 → Cr5=0	42.593	77.857
Cr1=1 Cr2=0 Cr3=0 Cr4=0 Cr5=0 → Cr6=0	30.338	71.229
Cr1=1 Cr2=0 Cr3=0 Cr4=0 Cr5=0 → Cr7=0	25.485	59.833
Cr1=1 Cr2=0 Cr3=0 → Cb1=0	50.019	87.517
Cr1=1 Cr2=0 Cr3=0 Cb1=0 → Cb2=1	45.947	91.860
Cr1=1 Cr2=0 Cr3=0 Cb1=0 Cb2=1 → Cb3=1	44.241	96.286
Cr1=1 Cr2=0 Cr3=0 Cb1=0 Cb2=1 Cb3=1 → Cb4=1	33.909	76.646

Association rule mining, which was originally introduced in 1993, is used to discover relationships that may not be immediately apparent between different variables in large datasets or databases [26]. Datasets in terms of association rule mining are constructed of many transactions with a constant number of items. Each item in a transaction can be a binary attribute, 0 or 1, that describes whether or not that particular item is present within a transaction or not. An example of this organization of the dataset including the transactions and their respective items can be seen in Figure 8. Once the data-mining algorithm has iterated through the data, discovering frequent occurrences of items, a list of rules are finalized and output. Each association rule that is output from this process is an implication of the form X → Y. X and Y are both either individual items, or a set of disjoint items. All rules also have related statistics, including, but not limited to, the confidence and support. These values are important in informing the user of how useful a given rule is, and also helps to organize the rules based on how often they occur in the data. The support value for an individual, or set of items, is the proportion of all transactions in a given dataset that contain that specific individual, or set of items. The confidence is the proportion of transactions that contain X which also contain Y, also known as the conditional probability, P(X|Y).

Using 12 videos from [21] and 4 videos from [27], association data-mining is used for analysis. The total size of the videos used in the test data set is 415.6 MB. The 12 videos from [21] were merged into a single .yuv file that contained a total of 3470 frames and 352×288 video resolution. Code was written to extract binary bit values for the luma and chroma data in the .yuv file and format it into .arff format for data analysis. After the .arff file is created and populated, Weka [28] was used in order to association rule mine the data using the apriori algorithm. Both subsampled chroma (i.e. YUV 4:2:0) and non-subsampled chroma (i.e. YUV 4:4:4) were investigated in order to see how much impact the compression has on the video format. The non-subsampled chroma has eight bits of each chroma component (Cb and Cr), the subsampled format is the same except that those bits are shared among 4 pixels, resulting in ¼ of the total chroma bits.

The most interesting pattern that was discovered from the results of mining the video data was the strong association between the MSB of the Cr chroma component to other bit values. Figure 9 displays that if Cr1 (i.e. the Cr chroma component's MSB) is equal to 1 (0), then the remaining bits in that Cr byte have a larger probability to be equal to the inverted value of the MSB, 0 (1). A similar pattern arises in the Cb bits, other than Cb1, most Cb bits have a higher probability to be 0 (1) when Cr1 equals 0 (1). Table 5 displays the probabilities of all possible chroma bit values based on the value of Cr1. Based on these results, a power-quality adaption theme for a flexible D-DASH design is now implementable.

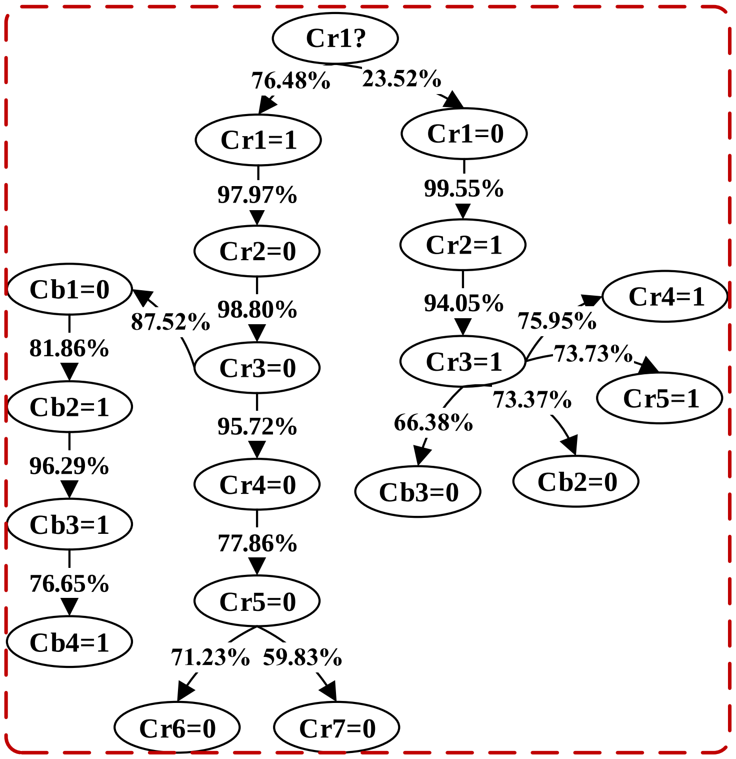


Figure 9. Rules diagram for chroma bit probability based on value of Cr1

Table 5. Data patterns for D-DASH design

Values	***Bit 2***	***Bit 3***	***Bit 4***	***Bit 5***	***Bit 6***	***Bit 7***	***Bit 8***
Cr1=0, CrX=1	99.9%	92.2%	89.7%	71.8%	71.4%	64.3%	56.3%
Cr1=1, CrX=0	99.2%	97.8%	83.3%	67.5%	59.8%	57.3%	53.2%
Cr1=0, CbX=0	60.5%	60.4%	64.0%	56.6%	52.8%	48.6%	50.5%
Cr1=1, CbX=1	86.6%	82.9%	55.3%	51.4%	54.8%	49.5%	50.4%

Video Quality Metrics

PSNR is used widely [6, 9, 3] by researchers to evaluate video quality which is defined as [29]:

$$PSNR = 10\log_{10}\left(\frac{255^2}{MSE}\right) \tag{3}$$

The MSE in the PSNR equation is the mean squared error between the original video (Org) and degraded video (Deg), and is expressed as:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[Org(i,j) - Deg(i,j)]^2 \tag{4}$$

The PSNR metric is well known and has been widely used in the past, however, it has been shown that it does not necessarily capture the true human perception of the video since it only takes the amount of error into account, not necessarily the influence the errors have on the image that is being displayed to the user [30]. Due to this fact, the SSIM metric was developed to predict the perceived image quality. It combines separate calculations for luminance, contrast and structural changes together, and is expressed as [30]:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5}$$

The luminance comparison function *l(x,y)*, is a function of the mean intensities, μ_x and μ_y, the contrast comparison function *c(x,y)*, is a function of the standard deviations, σ_x and σ_y, and the structural comparison function *s(x,y)*, is a function of the correlation between *x* and *y*, σ_{xy}. Setting $\alpha = \beta = \gamma = 1$ in the original equation results in the second equation. C_1 (C_2) is a constant that is included to avoid instabilities when the sum of the means (standard deviations) squared is equal to values near zero.

In the analysis of videos when applying D-DASH methodologies, both PSNR and SSIM are used to evaluate the output video quality. The reduction in quality can be calculated using these metrics with the following equation:

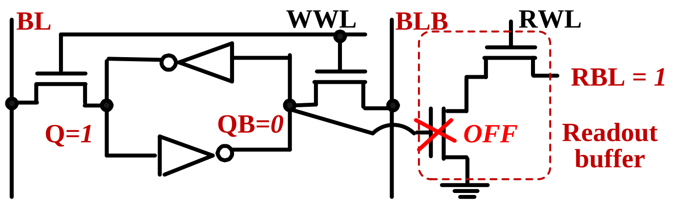
$$PSNR(SSIM)\ \%\ Reduction = \left(1 - \left(\frac{Output\ PSNR\ (SSIM)}{Original\ PSNR\ (SSIM)}\right)\right) \times 100\% \tag{6}$$

Proposed D-DASH Design

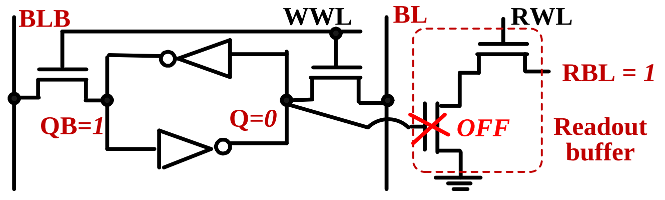
Based on the data patterns discovered in the previous section, three separate design schemes are presented (D-DASH-I to D-DASH-III). These schemes provide a system of run-time dynamic, power-quality trade-off for mobile video streaming devices.

D-DASH-I

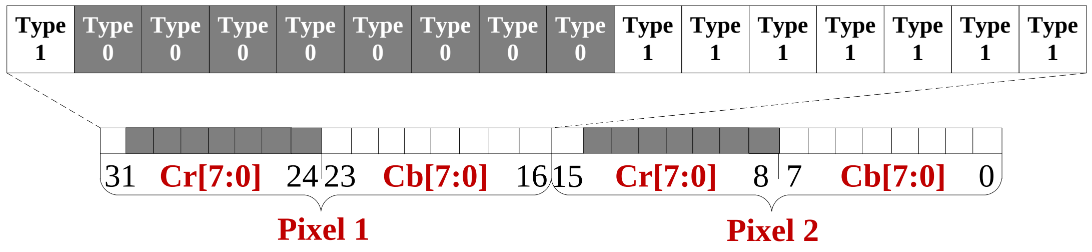
This scheme provides a zero cost, power efficient storage that uses data aware, low power readout buffer connections that are based on the data patterns obtained previously. The connections for the readout buffers are displayed in Figure 10. In conventional SRAM, a readout buffer consists of two NMOS transistors that are used to access the stored value by connecting to the reversed storage node (QB), as shown in Figure 10 (a) [31].



(a) Low-power type-1 SRAM bitcell to read *1*



(b) Low-power type-0 SRAM bitcell to read *0*



(c) A word in D-DASH-I

Figure 10. Data-aware D-DASH bit-cells types and organization in a word-line

During the memory read process, the RBL is first pre-charged to the supply voltage, V_{dd}, before the RWL is asserted. If a ‘1’ is stored in the SRAM cell and the RWL is asserted, RBL will remain at the voltage $V_{dd,}$ as the bottom NMOS will be turned off and there will be no switching activity in the RBL, enabling a low-power reading process. If a ‘0’ is stored in the SRAM cell, the RBL will be discharged to GND, resulting in large power consumption. The discharging activity during the readout process contributes significant power consumption in mobile video memory [9, 3, 12]. Based on this, data aware, low power readout buffer connections are proposed based on the previously discovered data patterns. The traditional connection, as shown in Figure 10 (a), are referred to as type-1 bit-cells. Alternatively, a type-0 bit-cell is presented to achieve low-power reading ‘0’ process, by connecting readout buffers to Q, as shown in Figure 10 (b). Note that, as compared to the conventional SRAM bit-cell (type-1), the type-0 bit-cell does not cause any silicon area overhead.

The chroma data patterns for each pixel display over a 70% probability that Cr will be 10000000_2 and Cb data will be 01111111_2, where both values are in binary. Based on these patterns, type-0 and type-1 bit-cells are applied to the memory design in order to store and load ‘0’ and ‘1’ values, respectively. Switching activity will be significantly reduced with this design scheme, reducing the power consumption of the hardware without area overhead. Figure 10 (c) displays the structure of a single word-line in the D-DASH-I scheme. An optimal combination of type-0 and type-1 bit-cells are used in order to enable zero overhead, power efficient, mobile video on-chip storage.

D-DASH-II

Additional power savings are introduced to the D-DASH-I design scheme in D-DASH-II by adding an additional control hardware to exploit the Cr and Cb binary patterns for further power savings. The MSB of the Cr data has a strong association to the value of many of the remaining bits as shown by the identified patterns. From this, a write circuit and a read circuit are added to the SRAM, as can be seen in Figure 11, in order to maximize the read bit-line power savings. In the write circuit, the MSB of the Cr data is used to determine whether to invert the input data or not, and use a flag-bit scheme similar to [31] in order to indicate if the data is flipped or not. A ‘0’ value indicates the data is inverted while a ‘1’ indicates it is not inverted. The design in [31] uses an extra bit-cell to store the flip bit, resulting in a 7% area overhead while D-DASH-II uses the LSB of the chroma data to store the flag bit. Table 6 shows that using the Cr LSB induces the least degradation (0.044% PSNR reduction and 0.058% SSIM reduction). With this design scheme, D-DASH-II enables more power savings as compared to D-DASH-I with negligible area overhead (+0.06%).

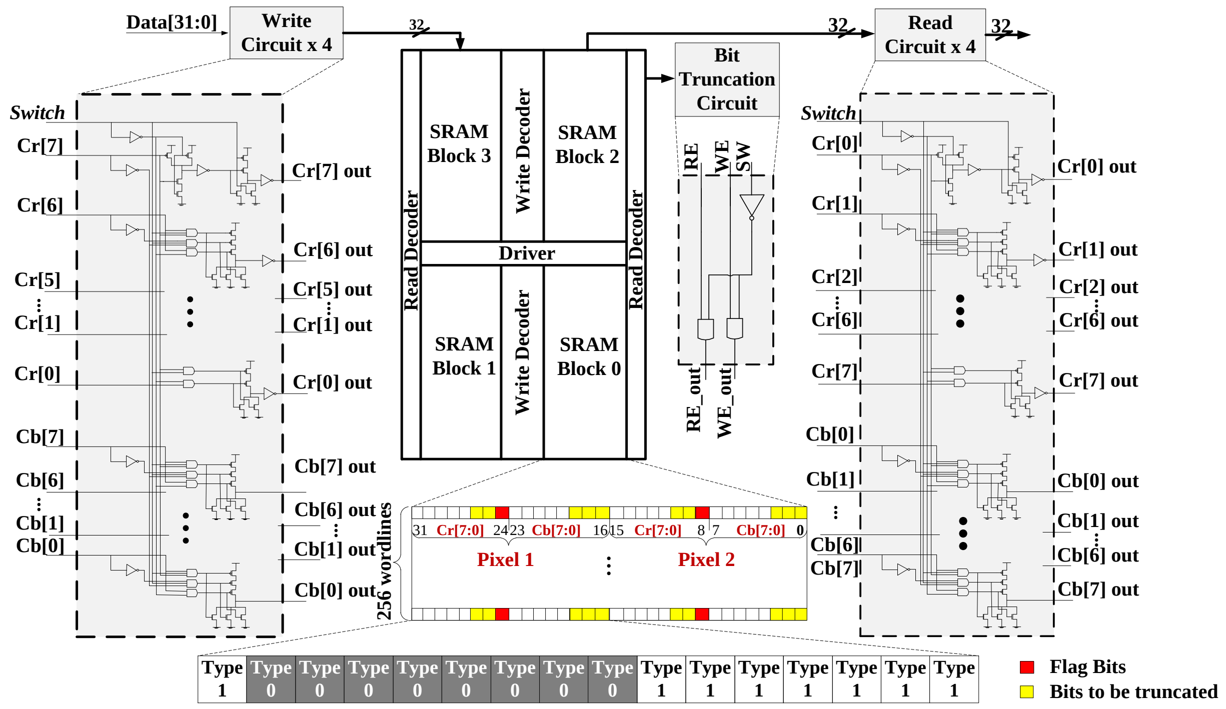


Figure 11. D-DASH with real-time adjustment between three schemes

D-DASH-III

In order to meet the power efficiency requirements of low power video applications, a third design scheme, D-DASH-III, is introduced. This scheme maximizes the power saving capabilities of the SRAM design by adopting all parts of the previous two designs and then adding bit-truncation to increase power savings by reducing bit switching. The bit truncation technique has been widely used in video memory design [22]. In order to determine how many bits should be dropped using the truncation technique we apply truncation at varying number of truncated bits, those results are listed in Table 6.

Table 6. PSNR and SSIM Calculations

# of drop bits	*PSNR*	*SSIM*	*PSNR % Reduction*	*SSIM % Reduction*
0 (Original)	36.868	0.934367	0.000%	0.000%
0 (Flag Bit)	36.852	0.933826	0.044%	0.058%
1	36.848	0.933213	0.054%	0.124%
3	36.641	0.929893	0.616%	0.479%
5	35.544	0.922561	3.592%	1.264%
7	32.593	0.910895	11.595%	2.512%
9	27.874	0.895294	24.395%	4.182%
11	21.506	0.862433	41.667%	7.699%
13	14.851	0.756258	59.718%	19.062%
15	10.782	0.623662	70.756%	33.253%

When the number of truncated bits becomes larger than 7, the PSNR and SSIM reduction are significant (PSNR reduction ≥ 24.395% and SSIM reduction ≥ 2.512%), indicating large video quality degradation. The allowed number of bits for bit truncation is therefore either 5 or 7. In order to further evaluate the video quality the sign_irene video benchmark, containing blue and red colors that would be directly influenced by Cr and Cb truncation [32], is analyzed using both 5 and 7 truncated bits, as can be seen in Figure 12. This shows that the truncation of 5 LSB bits (2 Cr LSBs and 3 Cb LSBs) is an optimized tradeoff between power savings and video quality. Figure 11 shows the proposed control circuitry to enable the bit truncation technique. The control bit SW is connected to the write enable (WE) and read enable (RE) of the word-line. When the value of the control bit, SW, is ‘0’, all of the 32 bit-cells connected to the word-line work as traditional bit-cells; when SW is ‘1’, the D-DASH-III design scheme is enabled and the outputs (WE_out and RE_out) will disable the write enable and read enable of the truncated bit-cells (the yellow bit-cells shown in Figure 11). This process will enable the bit truncation technique in order to achieve additional power savings beyond the savings from D-DASH-I and D-DASH-II.

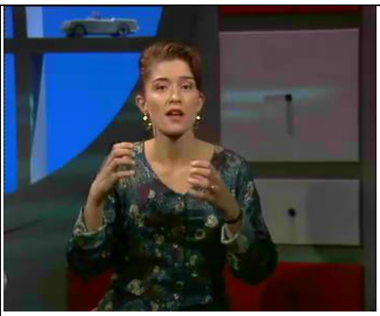
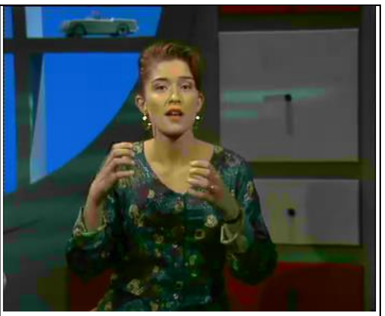
		
PSNR: 38.980559	PSNR: 37.047126	PSNR: 33.689856
SSIM: 0.942476	SSIM: 0.927677	SSIM: 0.912546
Original	5 Truncated Bits	7 Truncated Bits

Figure 12. Output of sign_irene benchmark based on the bit truncation technique

Simulation Results

In order to evaluate the effectiveness of the proposed technique, a 32kb SRAM is implemented using a high-performance 45-nm FreePDK CMOS process to meet the multi-megahertz performance requirement of today’s mobile video decoders.

Performance

The performance of the proposed D-DASH memory design is first analyzed for read and write delay times. Both read and write delays are found to be approximately 0.15 ns, which allows for successfully implementation of this memory in high-quality video format devices such as applications that can deliver 8K Ultra HD video [33].

Layout

The area overhead is analyzed for all three D-DASH schemes. The layout for D-DASH can be seen in Figure 13. D-DASH-I has no additional area overhead; D-DASH-II has a negligible area overhead of 0.64%; D-DASH-III, after careful layout design, with integrated control circuits and read decoder has equal area overhead to D-DASH-II.

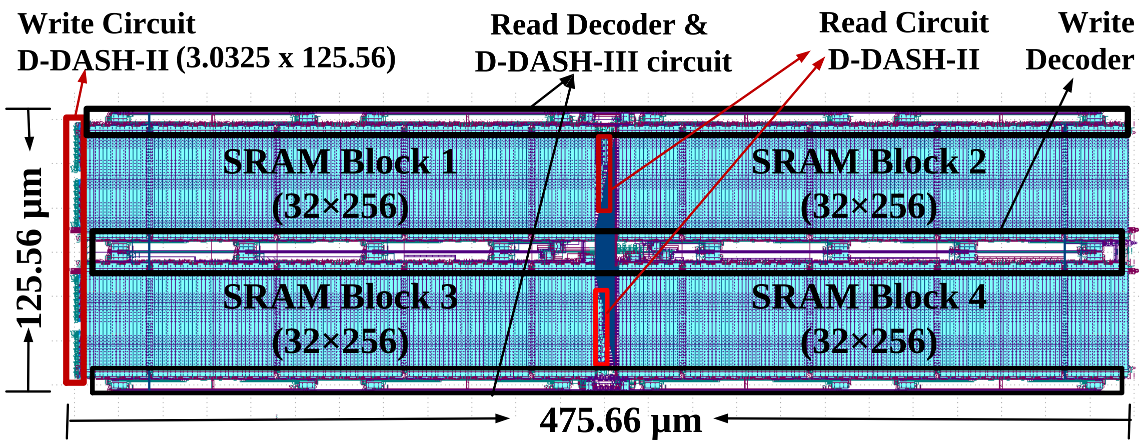


Figure 13. Layout of D-DASH memory design

Output Quality

Multiple videos are used to verify the output quality based on the proposed SRAM schemes. Figure 14 shows three specific video benchmarks with their calculated PSNR and SSIM metrics. D-DASH-I and D-DASH-II can deliver good video output quality and D-DASH-III results in negligible video degradation to achieve optimal power efficiency.

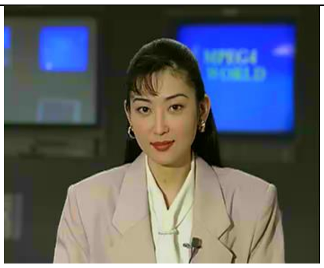
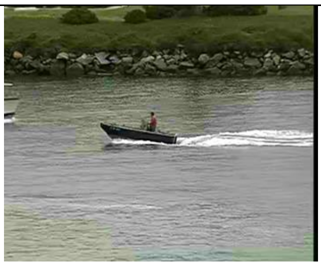
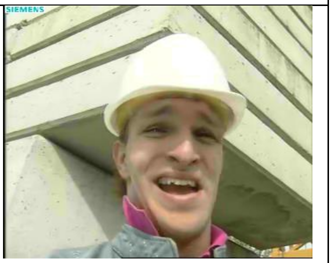
D-DASH	**Akiyo**	**Coastguard**	**Foreman**
Original / D-DASH-I	PSNR: 41.250858 SSIM: 0.961865	PSNR: 35.638277 SSIM: 0.919692	PSNR: 37.355170 SSIM: 0.923900
D-DASH-II	PSNR: 41.194586 SSIM: 0.961432	PSNR: 35.638277 SSIM: 0.919196	PSNR: 37.335870 SSIM: 0.923268
D-DASH-III	 PSNR: 37.971173 SSIM: 0.950497	 PSNR: 34.781284 SSIM: 0.914910	 PSNR: 35.829443 SSIM: 0.913802

Figure 14. D-DASH video output

Power Savings

To evaluate the power efficiency of D-DASH, a model to determine the read bitline (RBL) power consumption of mobile video memory is defined as:

$$P_r = \sum_{k=0}^{16} \sum_{i=0,1} [F_k(i) \cdot P_{rkt}(i) \cdot Z(k)] \qquad (7)$$

P_r is the power consumption of the read operation; *k* is the bit number; *t* is the SRAM type; *i* is the value stored in SRAM; *F(i)* indicates the probabilities of a bit to be 0 or 1, which is shown in Table 7; *Z(i)* indicates if the bit will be truncated (if truncated, *Z(i)* will be 0, if not truncated, *Z(i)* will be 1).

Table 7. Probability of each bit being 0 or 1

Bit-cell type	**Bit**	**D-DASH-I**		**D-DASH-II**	
		0	**1**	**0**	**1**
1	**Cr1**	0.2352	0.7648	0	1
0	**Cr2**	0.7588	0.2412	0.9940	0.0060
0	**Cr3**	0.7663	0.2337	0.9649	0.0351
0	**Cr4**	0.6615	0.3385	0.8482	0.1518
0	**Cr5**	0.5826	0.4174	0.6853	0.3147
0	**Cr6**	0.5247	0.4753	0.6254	0.3746
0	**Cr7**	0.5221	0.4779	0.5892	0.4108
0	**Cr8**	0.5097	0.4903	0.5097	0.4903
0	**Cb1**	0.7493	0.2507	0.8331	0.1669
1	**Cb2**	0.2456	0.7544	0.1958	0.8042
1	**Cb3**	0.2732	0.7268	0.2240	0.7760
1	**Cb4**	0.4923	0.5077	0.4265	0.5735
1	**Cb5**	0.5046	0.4954	0.4735	0.5265
1	**Cb6**	0.4702	0.5298	0.4568	0.5432
1	**Cb7**	0.5011	0.4989	0.5069	0.4931
1	**Cb8**	0.4986	0.5014	0.4956	0.5044

Table 8 displays the read power consumption for the two types of D-DASH bit-cells. The probability for each bit being either ‘0’ or ‘1’ is also extracted from the 12 benchmark videos and the results are listed in Table 7. The bits marked in grey are the bits that are truncated in D-DASH-III.

Table 8. RBL power consumption for different types of SRAM

SRAM type	Read 1	Read 0
Power of Type-1	3.22e-8	5.80e-8
Power of Type-0	5.88e-8	3.13e-8

Table 9 concludes the power savings of the proposed technique versus the traditional SRAM design. D-DASH enables power savings from 7.82% (D-DASH-I) to 43.07% (D-DASH-III).

Table 9. Power savings of the different D-DASH schemes

SRAM Designs	Power (W)	Power saving
Conventional SRAM	7.29E-07	-
D-DASH-I	6.72E-07	7.82%
D-DASH-II	6.33E-07	13.17%
D-DASH-III	4.15E-07	43.07%

CHAPTER 4. DATA-PATTERN ENABLED SELF-RECOVERY MULTIMEDIA STORAGE SYSTEM FOR NEAR-THRESHOLD COMPUTING[4]

In mobile video devices, intensive computations with frequent accesses to embedded memory consume large amounts of power and limit the battery life of such devices. In previous works, low-voltage memory designs have allowed for power consumption reduction, but at the cost of area overhead and complexity. This work describes a low-cost self-recovery video storage system created by discovering meaningful data patterns within the mobile video data. Data mining techniques allow for the discovery of both vertical and horizontal patterns hidden within the data. Based on the optimal patterns discovered using this technique, we present a novel DPSR SRAM design that achieves efficient near-threshold voltage computing while delivering good output video quality. Using advanced data mining techniques, we investigate the use of meaningful data patterns to enable self-recovery of the memory design. A 45nm 32kb SRAM is designed that delivers good video quality at near-threshold voltage (0.5 V) with negligible area overhead (3.97%).

Memory Failure Analsysis at Near-Threshold Voltage

Previous research has shown maximum computing efficiency is obtainable when a circuit is operating at near-threshold voltage [15]. However, at our target near-threshold voltage of 0.5V, SRAM failures become more severe due to the increased process variation. In particular, RDF effects lead to threshold voltage variation and SRAM cell failures [34]. Current manufacturing technologies describe the failure probability of a single SRAM cell to be between the range of 10^{-3} and 10^{-2}, depending on the area of the bit-cell in question [15, 35]. The

[4] The material in this chapter was co-authored by Jonathon Edstrom and Dongliang Chen. Jonathon Edstrom was in charge of all pattern discovery, data analysis, and video quality metric and simulation results. Dongliang Chen provided the presented SRAM hardware design with power simulation results based on the discovered patterns.

minimum sized SRAM has a maximum failure rate of 10^{-2} and bit-cells larger than the minimum sized version have lower failure rates. 58% area overhead over the minimized sized bit-cell will reduce the failure rate from 10^{-2} to 10^{-3} [35]. In the analysis used for this work, we consider both the minimum-sized SRAM with failure rate 10^{-2} and the upsized SRAM with failure rate 10^{-3}. Further optimization of the failure rate is possible using the priority-based sizing technique [36].

In order to get a better understanding of SRAM failure characteristics, we created error maps for a 256-word × 64-bit SRAM at both 10^{-2} and 10^{-3} failure rates. We assume that all memory faults have equal probability to occur in any given bit-cell position by using a uniform distribution when injecting faults to the memory error maps. Figure 15 displays the resulting error maps. Using 10^9 trials, we also examine the probability that multiple faults occur within the same 32-bit word-line and the results are listed in Table 10. Based on these results it is apparent that there are a low number of faulty bits in each word-line. Therefore, if a memory fault occurs in a given word-line the SRAM may use other bits in the same word to perform self-recovery if meaningful bit-level data-patterns exist.

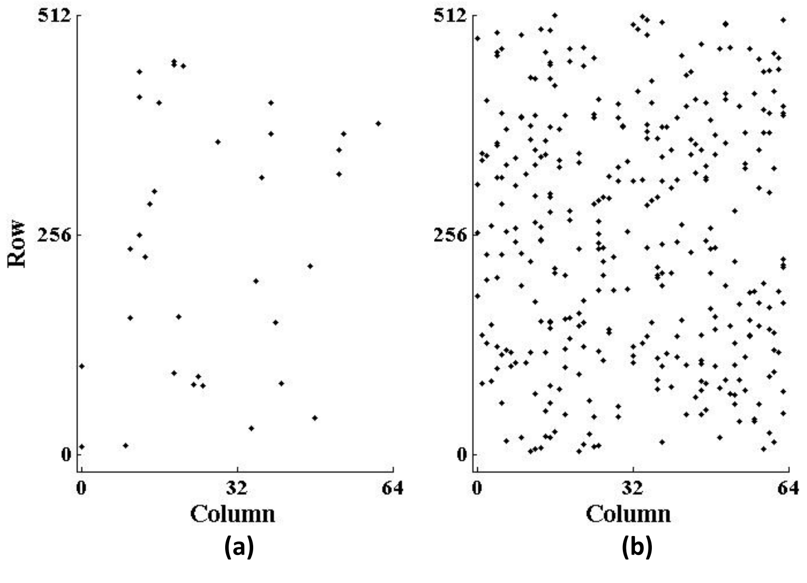


Figure 15. Error maps in the SRAM array at 0.5V
(a) Failure rate: 10^{-3} (0.001) (b) Failure rate: 10^{-2} (0.01)

Table 10. Fault probability in a 32-bit SRAM word

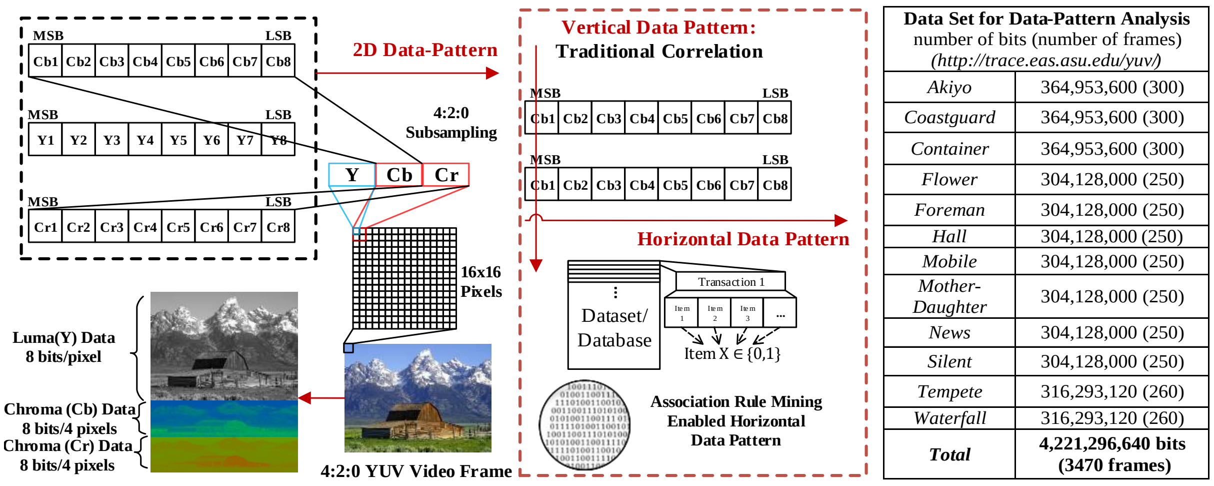
Number of faults per wordline	**SRAM failure rate: 10^{-3} (0.001)**	**SRAM failure rate: 10^{-2} (0.01)**
0	96.8523477%	72.7279953%
1	3.0992274%	23.2812509%
2	0.0479198%	3.6012385%
3	0.0005023%	0.3611914%
4	0.0000028%	0.0267011%
5	0%	0.0015432%
6	0%	0.0000756%
7	0%	0.000004%

Data Pattern Investigation for Self-Recovery

This section presents the methodology to discover data-patterns hidden within video data to enable reliable self-recovery from faults. Specifically, we propose a new two-dimensional data pattern approach to explore both horizontal and vertical data characteristics in order to find the optimal data patterns for applying self-correction techniques.

Rule Mining Enabled Horizontal Association

The typical format for storing and processing mobile video is the YUV format. The YUV format includes one luma (Y) component, which contains the brightness information of the image and two chroma components, which contain the blue-difference (Cb), and red-difference (Cr) color information. Figure 16 displays a typical frame of video data that could be stored in an embedded memory using YUV 4:2:0, 352×288 resolution video as an example. As depicted, each pixel has 8-bits of luma data and 8-bits of subsampled chroma data. Since video data I stored in on-chip memory as binary bits, we utilize an association data mining technique to identify horizontal bit-level data patterns.


MSB
LSB
Cb1 Cb2 Cb3 Cb4 Cb5 Cb6 Cb7 Cb8
Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8
Cr1 Cr2 Cr3 Cr4 Cr5 Cr6 Cr7 Cr8
2D Data-Pattern
4:2:0 Subsampling
Y Cb Cr
16x16 Pixels
Luma(Y) Data 8 bits/pixel
Chroma (Cb) Data 8 bits/4 pixels
Chroma (Cr) Data 8 bits/4 pixels
4:2:0 YUV Video Frame
Vertical Data Pattern: Traditional Correlation
Horizontal Data Pattern
Dataset/ Database
Transaction 1
Item 1
Item 2
Item 3
Item X ∈ {0,1}
Association Rule Mining Enabled Horizontal Data Pattern

Data Set for Data-Pattern Analysis number of bits (number of frames) *(http://trace.eas.asu.edu/yuv/)*	
Akiyo	364,953,600 (300)
Coastguard	364,953,600 (300)
Container	364,953,600 (300)
Flower	304,128,000 (250)
Foreman	304,128,000 (250)
Hall	304,128,000 (250)
Mobile	304,128,000 (250)
Mother-Daughter	304,128,000 (250)
News	304,128,000 (250)
Silent	304,128,000 (250)
Tempete	316,293,120 (260)
Waterfall	316,293,120 (260)
Total	**4,221,296,640 bits (3470 frames)**

Figure 16. 2D data-pattern enabled data self-correction

To enable association data mining for pattern discovery, we use 12 different video benchmarks in order to build a dataset [21]. The total video data size is about 528 MBs based on 3470 YUV frames. Figure 16 shows the breakdown of each video size in bits. We define each bit of video data as an individual item in our data-mining technique and we used Weka [28] to perform the well-known association rule-mining algorithm called Apriori on our constructed large dataset. Table 11 lists the horizontal data patterns we discovered for the chroma data in our video dataset.

Table 11. Discovered horizontal data-pattern association rules

Association Rules	**Confidence**	**Support**	**Confidence × Support**
Cb2=0 → Cb1=1	0.947547	0.232298	0.220113273
Cb2=1 → Cb1=0	0.975574	0.736405	0.718417571
Cb1=0 → Cb2=1	0.982838	0.736405	0.723766817
Cb1=1 → Cb2=0	0.926464	0.232298	0.215215734
Cr2=1 → Cr1=0	0.975149	0.235168	0.22932384
Cr2=0 → Cr1=1	0.999963	0.758811	0.758782924
Cr1=1 → Cr2=0	0.992164	0.758811	0.752864957
Cr1=0 → Cr2=1	0.999881	0.235168	0.235140015
Cr1=1 → Cr3=0	0.978025	0.747997	0.731559766
Cr1=0 → Cr3=1	0.922269	0.216914	0.200053058

Rule Mining Enabled Vertical Correlation

Many researchers have studied vertical data correlation characteristics of multimedia applications, such as video, in the past [10, 37]. These works indicate the MSBs of pixel data have strong correlation with the neighboring pixels and that the switching probability is very low. As listed in Table 12, the vertical correlation probability of the MSB in neighboring pixels is over 93%, while there is a reduction to 53% for the LSB.

Table 12. Vertical correlation probabilities

Cb Bit: 1 (MSB)	93.786775%	Cr Bit: 1 (MSB)	93.775505%
Cb Bit: 2	92.865158%	Cr Bit: 2	93.584600%
Cb Bit: 3	90.774607%	Cr Bit: 3	92.335457%
Cb Bit: 4	85.450795%	Cr Bit: 4	88.349737%
Cb Bit: 5	77.947842%	Cr Bit: 5	81.559365%
Cb Bit: 6	69.304415%	Cr Bit: 6	73.180250%
Cb Bit: 7	59.986183%	Cr Bit: 7	63.496048%
Cb Bit: 8 (LSB)	53.245386%	Cr Bit: 8 (LSB)	55.157592%

Optimal Data Patterns for Self-Recovery

In order to select an optimal data pattern from the discovered horizontal association data patterns and vertical correlation probabilities, we define the Weighted Confidence, a metric based on the support and confidence values of a particular rule as follows:

$$\textit{Weighted Confidence} = \textit{Confidence(Rule)} \times \textit{Support(Rule)} + \textit{Confidence(Complement Rule)} \times \textit{Support(Complement Rule)} \quad (8)$$

For example, the Weighted Confidence of the association rule $\overline{Cr1} \rightarrow Cr2$ would be:

$$\textit{Weighted Confidence}(\overline{Cr1} \rightarrow Cr2) = \textit{Confidence}(Cr1 = 0 \rightarrow Cr2 = 1) \times \textit{Support}(Cr1 = 0 \rightarrow Cr2 = 1) +$$

$$\textit{Confidence}(Cr1 = 1 \rightarrow Cr2 = 0) \times \textit{Support}(Cr1 = 1 \rightarrow Cr2 = 0)$$

$$= 0.999881 \times 0.235168 + 0.992164 \times 0.758811 = 0.988004972 \quad (9)$$

We then use this value as the metric for the horizontal data patterns and compare to the sum of the vertical correlation probabilities for ‘0’ and ‘1’ non-switching, which we label as the correlation value as follows:

$$Correlation = Confidence(Bit_{previous} = 0 \rightarrow Bit_{current} = 0) + Confidence(Bit_{previous} = 1 \rightarrow Bit_{current} = 1) \quad (10)$$

$Bit_{previous}$ and $Bit_{current}$ represent the video data bits in the same position of two neighboring pixels. As an example, the correlation of Cr2 would be calculated as follows:

$$Correlation(Cr2) = Confidence(Cr2_{previous} = 0 \rightarrow Cr2_{current} = 0) + Confidence(Cr2_{previous} = 1 \rightarrow Cr2_{current} = 1)$$

$$= 0.20908658 + 0.72675942 = 0.935846 \quad (11)$$

Based on the weighted confidence and correlation calculations we obtain the optimal bit-level data patterns with high prediction accuracy to enable self-recovery, as listed in Table 13.

Table 13. Optimal data patterns for enabling self-recovery

Cb bits	**Optimal Data Patterns**	**Correct Prediction (%)**	**Cr bits**	**Optimal Data Patterns**	**Correct Prediction (%)**
Cb1	Association $(\overline{Cb2} \rightarrow Cb1)$	93.853084	Cr1	Association $(\overline{Cr2} \rightarrow Cr1)$	98.810676
Cb2	Association $(\overline{Cb1} \rightarrow Cb2)$	83.898255	Cr2	Association $(\overline{Cr1} \rightarrow Cr2)$	98.800497
Cb3	Correlation $(Cb3_{previous})$	90.774607	Cr3	Association $(\overline{Cb1} \rightarrow Cr3)$	93.161282
Cb4	Correlation $(Cb4_{previous})$	85.450795	Cr4	Correlation $(Cr4_{previous})$	88.349737
Cb5	Correlation $(Cb5_{previous})$	77.947842	Cr5	Correlation $(Cr5_{previous})$	81.559365
Cb6	Correlation $(Cb6_{previous})$	69.304415	Cr6	Correlation $(Cr6_{previous})$	73.180250
Cb7	Correlation $(Cb7_{previous})$	59.986183	Cr7	Correlation $(Cr7_{previous})$	63.496048
Cb8	Correlation $(Cb8_{previous})$	53.245386	Cr8	Correlation $(Cr8_{previous})$	55.157592

Our previous calculations for horizontal weighted confidence and vertical correlation is based on the chroma data (Cr and Cb) in the video dataset but could be extended to the luma (Y) data. Our analysis shows that the luma bits have a more random switching trend and has less association with the other bits in the same word-line and the optimal data patterns are all based on the vertical correlation.

Recovery Failure Caused by Double Faults in Data Patterns

If two faults occur simultaneously in the same word-line and both are associated with the same data pattern it may cause recovery failure. We consider this and calculate the recovery failure rate for both horizontal associations and vertical correlations, which are listed in Table 14. The results show that DPSR has good reliability in these edge cases with extremely low self-recovery failure (less than 0.2%).

Table 14. DPSR recovery failure rate

Double Fault Type	**SRAM Failure Rate: 10^{-3}**	**SRAM Failure Rate: 10^{-2}**
Correlation Fault	0.0010899%	0.1077362%
Association Fault	0.0005957%	0.0587964%
DPSR Failure	0.0016856%	0.1665326%

DPSR Hardware Implementation

This section presents a simple circuit-level DPSR scheme with low implementation overhead. Figure 17 shows the array architecture of the proposed DPSR, where the total array size is 32kb and there are four blocks with 256 words × 32 bits. Applying a hierarchical readout bit-line scheme (local RBL and global RBL) reduces the access time. The self-recovery logic of DPSR can implemented by simply connecting MUXs to RBLs of the conventional SRAM design. As shown in Figure 17, each global bit-line (gbl) is connected to a MUX, which is controlled by the received fault positions. If a fault is present, the optimal data pattern enables self-recovery.

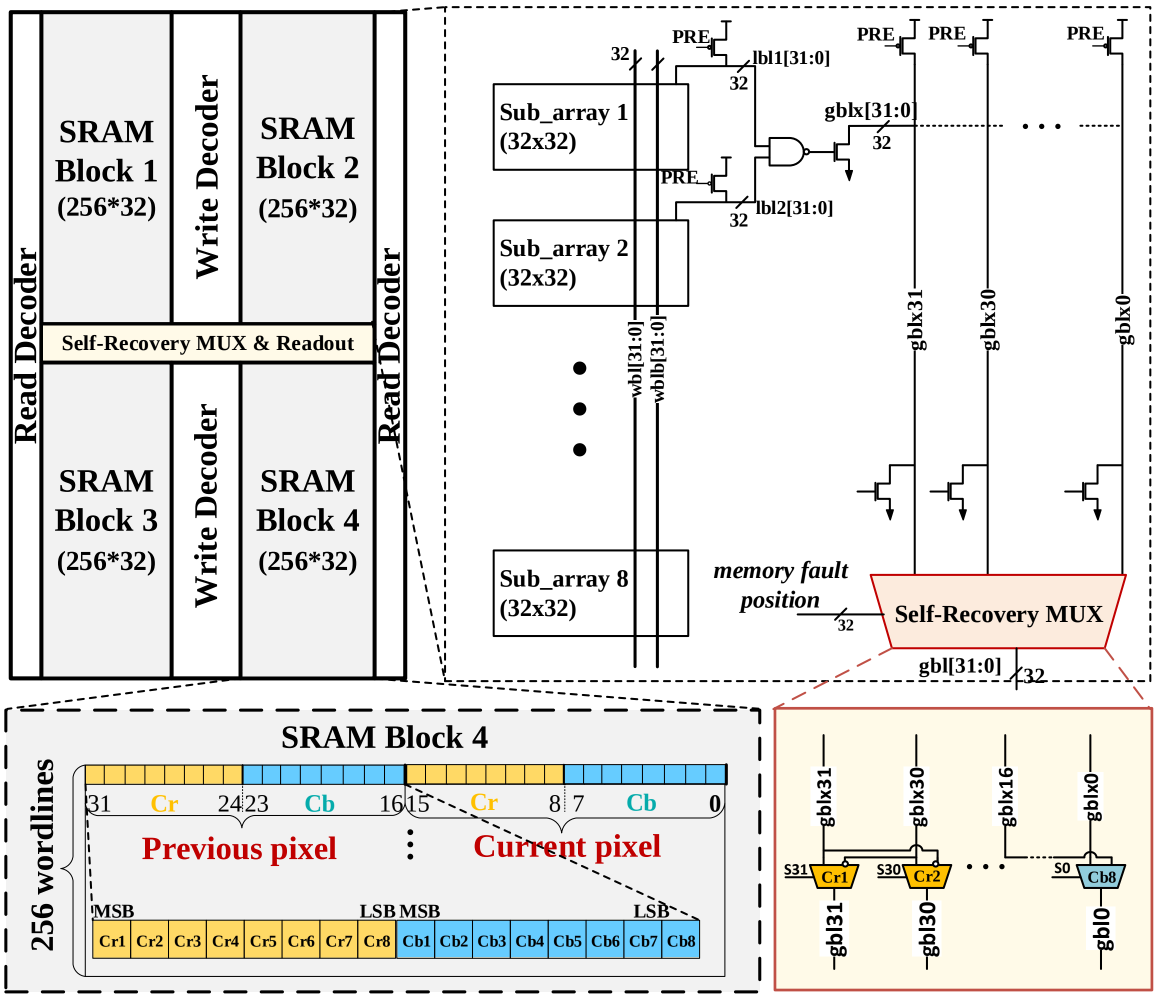


Figure 17. Proposed DPSR hardware design

Similar to other existing fault position aware mitigation techniques, DPSR receives pre-determined locations of the faulty bits, usually executed either during post-fabrication testing or during POST [15, 38, 39]. These processes can track the temporal degradation caused by memory failures, such as the aging effect.

There is a significant reduction in implementation complexity in this DPSR design as compared to existing techniques. For example, the shifting scheme presented in [15] needs to calculate the shift values based on the received fault positions and then performs shifting to store

LSBs in the identified faulty bit-cells. The evaluation that follows displays that DPSR also achieves a smaller area overhead, while still delivering good output video quality at near threshold voltage.

Evaluation Methodology and Results

To evaluate how effective our proposed technique is, we implement a 32kb SRAM using a high-performance 45-nm FreePDK CMOS process in order to meet the multi-megahertz performance requirement of today's mobile video decoders.

Performance

Due to the added MUXs, the read access time of DPSR increases from 0.27 ns to 0.31 ns, which is still capable of delivering high-quality video to application formats such as 8K Ultra HD [40].

Layout

Embedded SRAMs usually occupy a large portion of the area in a video chip, and therefore the area cost of embedded SRAM is an important design concern. Figure 18 shows the layout of the DPSR hardware design. The added self-recovery logic MUXs occupy an area of 18.79 µm × 43.47 µm, resulting in 3.97% area overhead. The self-recovery logic is added to RBLs and increasing the number of words in the memory is beneficial in reducing the area overhead.

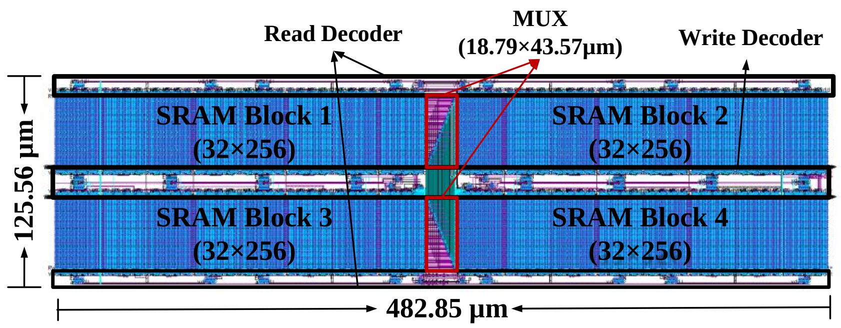


Figure 18. Proposed DPSR layout

Video Output Quality

For verification, we use a separate dataset than before, including three videos from [21] and five videos from [27]. We use the well-known PSNR [10] metric to evaluate the video quality. Researchers have shown that the PSNR with 30 dB or higher for a video is considered acceptable [15]. Table 15 compares the PSNR values using different techniques for failure rates of 10^{-2} and 10^{-3}. This comparison shows that our DPSR design has good recovery precision and can deliver good video quality with PSNR over 35 dB, even with the minimum sized SRAM. Figure 19 shows four of the videos from our verification dataset with failure rate 10^{-2}. As shown by these video output frames, DPSR achieves good video output quality at near-threshold voltage.

Table 15. PSNR video quality comparison

Videos	**Original decoded video**	**Conventional: 10^{-3}**	**DPSR: 10^{-3}**	**Conventional: 10^{-2}**	**DPSR: 10^{-2}**	**Ref.** [15]**: 10^{-3}**	**Ref.** [15]**: 10^{-2}**
akiyo	41.255457	37.135575	**41.073712**	29.297978	**38.342717**	41.253541	41.233291
bus	35.718543	34.120347	**35.703515**	28.636326	**35.663232**	35.707868	35.702169
city	36.80394	34.848668	**36.798971**	28.830418	**36.766432**	36.803067	36.79606
coastguard	35.669704	34.094057	**35.667145**	28.62693	**35.656755**	35.669025	35.663243
crew	37.145444	35.064527	**37.117851**	28.883566	**36.914495**	37.144506	37.136948
football	36.50373	34.655993	**36.462596**	28.778963	**36.290119**	36.502938	36.496437
foreman	37.214678	35.10653	**37.208593**	28.89477	**37.171422**	37.213679	37.205619
sign_irene	38.980559	36.109661	**38.86677**	29.114405	**37.530671**	38.979026	38.966286

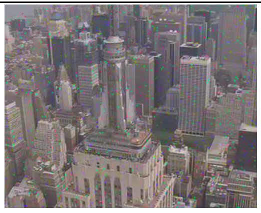
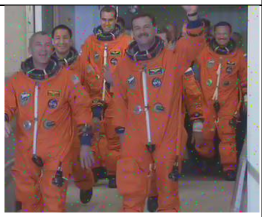
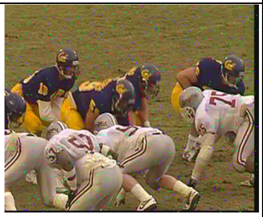
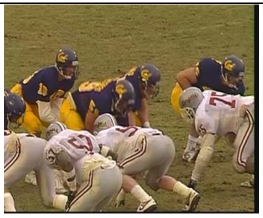
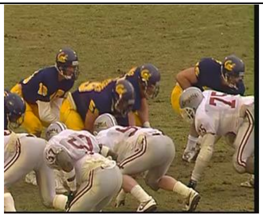
Videos	Original Video	Conventional: 10^{-2}	DPSR: 10^{-2}	Shift [15]: 10^{-2}
city	***PSNR*: 36.803940**	 ***PSNR*: 28.830418**	***PSNR*: 36.766432**	***PSNR*: 36.796060**
crew	***PSNR*: 37.145444**	 ***PSNR*: 28.883566**	***PSNR*: 36.914495**	***PSNR*: 37.136948**
football	***PSNR*: 36.503730**	 ***PSNR*: 28.778963**	 ***PSNR*: 36.290119**	 ***PSNR*: 36.496437**

Figure 19. Video output comparison of the different techniques

CHAPTER 5. DISCUSSION AND SUMMARY

This chapter gives a brief comparison of our memory designs based on power efficiency, video output quality, and silicon area overhead and summarizes our developed techniques.

Comparison of D-DASH to Prior Works

Table 16 compares our D-DASH [13] design schemes against state of the art techniques from recent works [3, 9, 10, 12]. D-DASH has the lowest implementation cost at only 0.06% silicon overhead and includes dynamic power-quality tradeoff. D-DASH-I and D-DASH-II exhibit the best video quality, except for [12], which is realized with large area overhead (~14%). D-DASH-III demonstrates the highest power efficiency, other than [10], which requires bit-cell array modification, resulting in as high as 52% area overhead.

Table 16. D-DASH comparison with prior works on low power SRAM

	TVLSI'08 [12]	**TCASVT'11** [9]	**TCASII'12** [10]	**JSCC'15** [3]	**D-DASH** [13]		
					D-DASH-I	**D-DASH-II**	**D-DASH-III**
video specific characteristics	correlation of MSB	contribution of MSB and LSB	different contribution of MSB and LSB	contribution of MSB and LSB	data-mining assisted video data patterns exploration		
dynamic adaption	No	No	No	Yes	Yes		
low-power technique	data filpping	6T+8T bitcells	8T+10T bitcells	ECC	data - aware bitline connection	data- aware bitline connection and data flipping	data-aware bitline connection, data flipping and bit truncation
bitcell array modification	Yes	Yes	additional word line	No	No	No	No
additional hardware needed	majority logic and data flipping block	single-ended 6T, peripheral circuitries	No	ECC encoder and decoder, write assisted circuit	No	control circuit consists simple gates	control circuit consists simple gates
power penalty for extra bits	Yes	No	No	No	No	No	No
readout power	-14%	-32%	-95%	-28%	**-7.82%**	**-13.17%**	**-43.07%**
video quality	good	acceptable	acceptable	acceptable	**good**	**good**	**acceptable**
area	+14%	+11.64%	+52%	+1.5%	**0%**	**+0.06%**	**+0.06%**
technology	90nm	90nm	45nm	28nm	45nm	45nm	45nm

Comparison of DPSR to Prior Works

Table 17 displays a comparison of the performance of our DPSR [16] memory design against other recent state of the art techniques. Using data-pattern enabled self-recovery, DPSR has the lowest implementation cost (3.97%) and has reliable operation at near-threshold voltage, allowing for maximum energy efficiency. DPSR also delivers the best video quality output, except for [14], which is realized with large area overhead (~14%).

Table 17. DPSR comparison with prior works on low power SRAM

	TCASI'12 [11]	DAC'15 [14]	TC'16 [15]	**DPSR** [16]
fault-position awareness	No	Yes	Yes	**Yes**
Low-power techniques	bit-cell Sizing	data-shifting	data-squeezing	**data-pattern enabled self-recovery**
bit-cell modified	Yes	No	No	**No**
near-threshold operation	No (0.9V)	Yes (-)	Yes (0.5 V)	**Yes (0.5V)**
additional logic needed	No	LUTs and shifter	Rearrangement logic and tag array, comparator, Mux	**MUX**
performance overhead	-	-	extra clock (for decompression)	**0.04 ns**
video quality	acceptable	good	-	**good**
area overhead	11-65%	14%	6.3%	**3.97%**

Summary and Future Work

Video is everywhere today. However, due to the large data size and intensive computation, video applications require frequent memory access and consume a large amount of power, limiting battery life and frustrating mobile users. In this thesis, we have introduced data-awareness and viewer-awareness to achieve better-informed and more efficient intelligent video memory design. Both D-DASH and DPSR memory design schemes, with an overview of their makeup, are compared against recent works by other researchers working in the area of low power memory design techniques. We plan to extend the proposed intelligent storage system to other data-intensive applications such as the synaptic storage used in online learning systems.

REFERENCES

[1] E. Terzioglu, S. S. Yoon, C. Jung, R. Chaba, V. Boynapalli, M. Abu-Rahma, J. Wang, G. Nallapati, A. Thean, C. Chidambaram, M. Han, G. Yeap and M. Sani, "Low Power Embedded Memory Design - Process to System Level Considerations," in *IEEE International Conference on IC Design & Technology (ICICDT)*, Kaohsiung, May 2011.

[2] A. Pathak, D. Sachan, H. Peta and M. Goswami, "A Modified SRAM Based Low Power Memory Design," in *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems (VLSID)*, Kolkata, Jan. 2016.

[3] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester and M. Alioto, "SRAM for Error-Tolerant Applications With Dynamic Energy-Quality Management in 28 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 5, pp. 1310-1323, May 2015.

[4] J. Xue and C.-W. Chen, "A Study on Perception of Mobile Video with Surrounding Contextual Influences," in *Proceedings of the 4th International Workshop on Quality of Multimedia Experience (QoMEX)*, Melbourne, Austrailia, Jul. 2012.

[5] J. Xue and C.-W. Chen, "Mobile JND: Environmental adapted perceptual model and mobile video quality enhancement," in *MMSys '12 Proceedings of the 3rd Multimedia Systems Conference*, Chapel Hill, Feb. 2012.

[6] J. Xue and C.-W. Chen, "Mobile Video Perception: New Insights and Adaptation Strategies," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 3, pp. 390-401, Jun. 2014.

[7] J. Xue and C.-W. Chen, "Towards Viewing Quality Optimized Video Adaptation," in *International Conference on Multimedia and Expo (ICME)*, Barcelona, Jul. 2011.

[8] J. Edstrom, D. Chen, J. Wang, H. Gu, E. A. Vazquez, M. E. McCourt and N. Gong, "Luminance-Adaptive Smart Video Storage System," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, 2016.

[9] I. J. Chang, D. Mohapatra and K. Roy, "A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 2, pp. 101-112, Feb. 2011.

[10] N. Gong, S. Jiang, A. Challapalli, S. Fernandes and R. Sridhar, "Ultra-Low Voltage Split-Data-Aware Embedded SRAM for Mobile Video Applications," *IEEE Transactions on Circuits and Systems II*, vol. 59, no. 12, pp. 883-887, Dec. 2012.

[11] J. Kwon, I. J. Chang, I. Lee, H. Park and J. Park, "Heterogeneous SRAM Cell Sizing for Low-Power H.264 Applications," *IEEE Transactions on Circuits and Systems I,* vol. 59, no. 10, pp. 2275-2284, Oct. 2012.

[12] H. Fujiwara, K. Nii, J. Miyakoshi, Y. Murachi, Y. Morita, H. Kawaguchi and M. Yoshimoto, "A Two-Port SRAM for Real-Time Video Processor Saving 53% of Bitline Power with Majority Logic and Data-Bit Reordering," in *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, Tegernsee, Oct. 2006.

[13] D. Chen, J. Edstrom, X. Chen, J. Wei, J. Wang and N. Gong, "Data-Driven Low-Cost On-Chip Memory with Adaptive Power-Quality Trade-off for Mobile Video Streaming," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design (ISLPED '16)*, San Francisco, 2016.

[14] S. Ganapathy, G. Karakonstantis, A. Teman and A. Burg, "Mitigating the Impact of Faults in Unreliable Memories for Error-Resilient Applications," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, San Francisco, Jun. 2015.

[15] A. Ferrerón, D. Suárez-Gracia, J. Alastruey-Benedé, T. Monreal-Arnal and P. Ibáñez, "Concertina: Squeezing in Cache Content to Operate at Near-Threshold Voltage," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 755-769, Mar. 2016.

[16] N. Gong, J. Edstrom, D. Chen and J. Wang, "Data-Pattern Enabled Self-Recovery Multimedia Storage System for Near-Threshold Computing," in *IEEE 34th International Conference on Computer Design (ICCD)*, Phoenix, 2016.

[17] G. Jarboe, "Tablets and Smartphone Video Views Up by 400% on 2012," Tubular Insights, 9 December 2014. [Online]. Available: http://tubularinsights.com/mobile-video-views-increase-400-per-cent/. [Accessed 14 March 2017].

[18] H. Kobiki and M. Baba, "Preserving Perceived Brightness of Displayed Image Over Different Illumination Conditions," in *Proceedings of 2010 IEEE 17th International Conference on Image Processing (ICIP)*, Hong Kong, Sep. 2010.

[19] M. A. Finkelstein and D. C. Hood, Handbook of Perception & Human Performance, New York: John Wiley and Sons, 1986.

[20] Y. J. Kim, "An Automatic Image Enhancement Method Adaptive to the Surround Luminance Variation for Small Sized Mobile Transmissive LCD," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 3, pp. 1161-1166, 2010.

[21] "YUV Video Sequences," [Online]. Available: http://trace.eas.asu.edu/yuv/. [Accessed 20 March 2017].

[22] D. Chen, X. Wang, J. Wang and N. Gong, "VCAS: Viewing Context Aware Power-Efficient Mobile Video Embedded Memory," in *Proceedings of the 28th IEEE International SoC Conference (SoCC '15)*, Beijing, 2015.

[23] M. A. Hoque, M. Siekkinen and J. K. Nurminen, "Energy Efficient Multimedia Streaming to Mobile Devices — A Survey," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 1, pp. 579-597, Nov. 2012.

[24] Y. Benmoussa, J. Boukhobza, E. Senn and D. Benazzouz, "Energy Consumption Modeling of H.264/AVC Video Decoding for GPP and DSP," in *2013 Euromicro Conference on Digital System Design (DSD)*, Santander, Sep. 2013.

[25] S. Venkataramani, S. T. Chakradhar, K. Roy and A. Raghunathan, "Approximate Computing and the Quest for Computing Efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, Jun. 2015.

[26] R. Agrawal, T. Imieliński and S. Arun, "Mining Association Rules Between Sets of Items in Large Databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C., May 1993.

[27] "Xiph.org Video Test Media," [Online]. Available: http://media.xiph.org/video/derf/. [Accessed 20 March 2017].

[28] "Weka," [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed 20 March 2017].

[29] A. Hore and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," in *2010 20th International Conference on Pattern Recognition (ICPR)*, Istanbul, Aug. 2010.

[30] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, Apr. 2004.

[31] H. Fujiwara, K. Nii, H. Noguchi, J. Miyakoshi, Y. Murachi, Y. Morita, H. Kawaguchi and M. Yoshimoto, "Novel Video Memory Reduces 45% of Bitline Power Using Majority Logic and Data-Bit Reordering," *IEEE Transcations on Very Large Scale Integration (VLSI) Systems,* vol. 16, no. 6, pp. 620-627, Jun. 2008.

[32] W. Yueh, M. Cho and S. Mukhopadhyay, "Perceptual Quality Preserving SRAM Architecture for Color Motion Pictures," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, Mar. 2013.

[33] D. Zhou, S. Wang, H. Sun, J. Zhou, J. Zhu, Y. Zhao, J. Zhou, S. Zhang, S. Kimura, T. Yoshimura and S. Goto, "14.7 A 4Gpixel/s 8/10b H.265/HEVC Video Decoder Chip for

8K Ultra HD applications," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, Feb. 2016.

[34] N. Gong, S. Jiang, A. Challapalli, M. Panesar and R. Sridhar, "Variation-and-Aging Aware Low Power embedded SRAM for Multimedia Applications," in *2012 IEEE International SOC Conference (SOCC)*, Niagara Falls, Sep. 2012.

[35] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper and N. S. Kim, "Minimizing Total Area of Low-Voltage SRAM Arrays through Joint Optimization of Cell Size, Redundancy, and ECC," in *2010 IEEE International Conference on Computer Design (ICCD)*, Amsterdam, Oct. 2010.

[36] S. A. Pourbakhsh, X. Chen, D. Chen, X. Wang, N. Gong and J. Wang, "Sizing-Priority Based Low-Power Embedded Memory for Mobile Video Applications," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, Mar. 2016.

[37] H. Noguchi, Y. Iguchi, H. Fujiwara, Y. Morita, K. Nii, H. Kawaguchi and M. Yoshimoto, "A 10T Non-Precharge Two-Port SRAM for 74% Power Reduction in Video Processing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Porto Alegre, Mar. 2007.

[38] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson and S.-L. Lu, "Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, San Jose, Jun. 2011.

[39] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu and D. Srivastava, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846-852, Apr. 2007.

[40] D. Zhou, S. Wang, H. Sun, J. Zhou, J. Zhu, Y. Zhao, J. Zhou, S. Zhang, S. Kimura, T. Yoshimura and S. Goto, "A 4Gpixel/s 8/10b H.265/HEVC Video Decoder Chip for 8K Ultra HD Applications," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, Feb. 2016.

APPENDIX A. SMART VIDEO STORAGE SYSTEM ARDUINO CODE

```
/*
 * Lux Sensor Arduino Code
 * Jonathon Edstrom - 2015
 * Read analog luminance through lux sensor and send data bits as output through I/O
 * Department: NDSU ECE Graduate Research
 * Project: Luminance-Adaptive Smart Video Storage System
 */

#include "SPI.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>

Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);

void setup() {
  Serial.begin(9600); // for debugging over serial communication

  pinMode(12, OUTPUT); // set pin 12 to output
  pinMode(13, OUTPUT); // set pin 13 to output

  tsl.enableAutoRange(true);
  tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);

  // Initialize the sensor
  if(!tsl.begin())
  {
    // Problem with connection, fix it and restart
    Serial.println("No TSL2561 detected... ");
    Serial.println("Check your wiring / I2C ADDR!")
    while(1);
  }
}

void loop(void) {
  sensors_event_t event;
  tsl.getEvent(&event); // Get a new sensor event

  // Display results (light is measured in lux)
  if (event.light > 0)
  {
    digitalWrite(12, LOW);  // Data bit 1 = 0
    digitalWrite(13, LOW);  // Data bit 2 = 0
  }
  else if (event.light > 1000 && event.light < 10000)
  {
    digitalWrite(12, LOW);  // Data bit 1 = 0
    digitalWrite(13, HIGH); // Data bit 2 = 1
  }
  else if (event.light > 10000)
  {
    digitalWrite(12, HIGH); // Data bit 1 = 1
    digitalWrite(13, LOW);  // Data bit 2 = 0
  }
  else // If = 0 lux, the sensor is probably saturated
  {
    Serial.println("Sensor overload");
  }
  delay(250); // 250ms delay
}
```

APPENDIX B. SMART VIDEO STORAGE SYSTEM RASPBERRY PI 2 CODE

```
/*
 * YUV420p Luminance Frame Truncation Program
 * Jonathon Edstrom - 2015
 * Truncates LSBs from luminance portion of YUV 4:2:0 frames
 * based on present environmental lighting conditions and outputs
 * them to the display in real time
 * Department: NDSU ECE Graduate Research
 * Project: Luminance-Adaptive Smart Video Storage System
 */

// includes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <time.h>
#include <wiringPi.h>

// instantiate globals
FILE * fileptr; // pointer for YUV data input/output files
unsigned char * buffer; // pointer for YUV data allocated memory
unsigned char * reset; // pointer value to reset the buffer to the beginning
unsigned char * tempholder; // holds starting value address between conversions
int filelen; // length of the input file (total bytes)
int xres; // width of the YUV video
int yres; // height of the YUV video
int bit1; // first bit of truncation info
int bit2; // second bit of truncation info
int lumsize; // luminance bytes per frame
int chromsize; // chromiance bytes per frame
int singlechromasize; // size of Cr or Cb in bytes per frame
unsigned char temp; // temporary holder for buffer byte
char * fbp = 0; // frame buffer pointer
struct fb_var_screeninfo vinfo; // screen info
struct fb_fix_screeninfo finfo; // screen info

// helper function to truncate bits on YUV frames
char truncate_bits( char byte )
{
    int truncbits = ( ( bit1 << 1 ) | bit2 );
    switch( truncbits ) {
        case 0:
            return byte; // don't truncate anything
            break;
        case 1:
            return ( byte & 248 ); // bitwise AND with 11111000 (truncate 3 bits)
            break;
        case 2:
            return ( byte & 240 ); // bitwise AND with 11110000 (truncate 4 bits)
            break;
        default:
            printf( "The number of bits to truncate is not valid. Exiting...\n" );
            exit( EXIT_FAILURE );
    }
}

// clamps value to 0-255 range
```

```
unsigned char clamp( float value )
{
    if( value < 0 )
    {
        return 0;
    }
    else if( value > 255 )
    {
        return 255;
    }
    else
    {
        return ( unsigned char ) value;
    }
}

// helper function for RGB pixel plotting
// credit given to: raspberrycompote.blogspot.com/2013/03/low-level-graphics-on-
// raspberry-pi-part_8.html
void put_pixel_RGB( int x, int y, int r, int g, int b )
{
    // calculate the pixel's byte offset inside the buffer
    // note: x * 3 as every pixel is 3 consecutive bytes
    int pix_offset = x * 3 + y * finfo.line_length;

    // approximately the same as 'fbp[pix_offset] = value'
    *((unsigned char*)(fbp + pix_offset)) = b;
    *((unsigned char*)(fbp + pix_offset + 1)) = g;
    *((unsigned char*)(fbp + pix_offset + 2)) = r;
}

// helper function for drawing to the frame buffer
void draw( unsigned char * rgbdata )
{
    int x, y, r, g, b;

    unsigned char * data = rgbdata;

    for( y = 0; y < yres; y++ )
    {
        for( x = 0; x < xres; x++ )
        {
            r = *data; data++;
            g = *data; data++;
            b = *data; data++;
            put_pixel_RGB( x, y, r, g, b );
        }
    }
}

// converts yuv420 frames to rgb888 frames for output to display
// equations found at: https://en.wikipedia.org/wiki/YUV
unsigned char * yuv420p_to_rgb( unsigned char * yuvframe )
{
    unsigned char * rgb = ( unsigned char * ) calloc((lumsize * 3), sizeof(unsigned
char));

    unsigned char * ydata = yuvframe;
    unsigned char * udata = yuvframe + lumsize;
    unsigned char * vdata = udata + singlechromasize;

    float b,g,r;
    int x,y;
    unsigned char * ptr = rgb;
    for (y = 0; y < yres; y++)
    {
        for (x = 0; x < xres; x++)
        {
            int yy = ydata[(y * xres) + x];
            int uu = udata[((y / 2) * (xres / 2)) + (x / 2)];
            int vv = vdata[((y / 2) * (xres / 2)) + (x / 2)];

            int c = yy - 16, d = uu - 128, e = vv - 128;
            r = ( 298 * c + 409 * e + 128 ) >> 8;
            g = ( 298 * c - 100 * d - 208 * e + 128 ) >> 8;
            b = ( 298 * c + 516 * d + 128 ) >> 8;

            unsigned char rval = clamp(r);
            unsigned char gval = clamp(g);
            unsigned char bval = clamp(b);

            *ptr++ = rval;
            *ptr++ = gval;
            *ptr++ = bval;
        }
    }

    return rgb;
}

// application entry point
int main( int argc, char * argv[] )
{
    wiringPiSetupGpio(); // Set up GPIO using BCM standard pin numbers
    pinMode( 23, INPUT ); // GPIO 23 set to INPUT
    pinMode( 24, INPUT ); // GPIO 24 set to INPUT

    if( argc != 3 ) // argc should be 3 for correct execution
    {
        // print usage assuming argv[0] is the program name
        printf( "usage: %s xres yres\n", argv[0] );
        exit( EXIT_FAILURE );
    }
    else // correct number of arguments
    {
        // initialize globals
        xres = atoi( argv[1] );
        yres = atoi( argv[2] );

        // calculate Y (luminance) and UV (chromiance) byte component sizes
        lumsize = xres*yres;
        chromsize = lumsize/2;
        singlechromasize = chromsize/2;
        tempholder = ( unsigned char * ) malloc( ( lumsize * 3 ) + 1 );

        // turn off terminal cursor blink
        FILE * cursorfile = fopen( "/sys/class/graphics/fbcon/cursor_blink", "w+" );
        fprintf( cursorfile, "%c", '0' );
        fclose( cursorfile );

        // framebuffer setup
        int fbfd = 0;
        struct fb_var_screeninfo orig_vinfo;
        long int screensize = 0;

        // open the framebuffer for reading/writing
```

```
fbfd = open( "/dev/fb0", O_RDWR );
if( !fbfd )
{
    printf( "Error: cannot open framebuffer device.\n" );
    exit( EXIT_FAILURE );
}

// get variable screen information
if( ioctl( fbfd, FBIOGET_VSCREENINFO, &vinfo ) )
{
    printf( "Error reading variable information.\n" );
    exit( EXIT_FAILURE );
}

// store screen information for reset (copy vinfo to vinfo_orig)
memcpy( &orig_vinfo, &vinfo, sizeof( struct fb_var_screeninfo ) );

// change variable screen information - force 24 bit and resolution
vinfo.bits_per_pixel = 24;
vinfo.xres = xres;
vinfo.yres = yres;
vinfo.xres_virtual = vinfo.xres;
vinfo.yres_virtual = vinfo.yres;
if( ioctl( fbfd, FBIOPUT_VSCREENINFO, &vinfo) )
{
    printf( "Error setting variable information.\n" );
    exit( EXIT_FAILURE );
}

// get fixed screen information
if( ioctl( fbfd, FBIOGET_FSCREENINFO, &finfo ))
{
    printf( "Error reading fixed information.\n" );
    exit( EXIT_FAILURE );
}

// map framebuffer to user memory
screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;
fbp = (char *) mmap(0, screensize, PROT_READ|PROT_WRITE, MAP_SHARED, fbfd, 0);
if( ( int ) fbp == -1 )
{
    printf( "Failed to mmap.\n" );
    exit( EXIT_FAILURE );
}

// create frame filename string
char framenum[4];
int framecount = 1;
sprintf( framenum, "%d", framecount++ ); // put frame number into string
char name[20] = "frame";
strcat( name, framenum );
strcat( name, ".yuv" );

// assume name is the filename to open
// open file using "rb" = read binary file access mode
fileptr = fopen( name, "rb" );

// if fopen returns a NULL pointer it failed to open the file
if( fileptr == NULL )
{
    printf( "Could not open file. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
```

```
int flag = 1;
do
{
    bit1 = digitalRead( 23 ); // read bit 1 from GPIO 23
    bit2 = digitalRead( 24 ); // read bit 2 from GPIO 24

    // file opened successful -> allocate memory buffer space
    fseek( fileptr, 0, SEEK_END ); // jump to end of file
    filelen = ftell( fileptr ); // get current byte offset in file
    rewind( fileptr ); // jump to beginning of file

    buffer = ( char * ) malloc( filelen + 1 ); // enough memory for file + \0
    reset = buffer;
    if( buffer == NULL )
    {
        printf( "Failed to allocate memory. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }

    fread( buffer, 1, filelen, fileptr ); // read file into memory

    buffer = reset; // reset buffer pointer address to beginning
    int index = 0; // current array index
    int count = 0; // keeps track of luminance byte index
    while( index < filelen ) // loop until EOF
    {
        while( count < lumsize ) // loop until end of frame's luminance data
        {
            temp = *buffer; // store current byte
            *buffer = truncate_bits( temp ); // truncate
            if( index < filelen )
            {
                // increment buffer address, index and count variables
                buffer++;
                index++;
                count++;
            }
        }

        count = 0; // reset luminance byte index
        if( index < filelen )
        {
            buffer += chromsize; // adjust ptr addr to next set of luma bytes
            index += chromsize; // adjust index to next set of luminance bytes
        }
    }

    buffer = reset; // reset buffer pointer to beginning of data
    tempholder = yuv420p_to_rgb( buffer ); // convert to RGB888 data
    draw( tempholder ); // draw the converted data to the screen

    // create next filename string
    sprintf( framenum, "%d", framecount++ ); // put frame number into string
    strcpy( name, "frame" );
    strcat( name, framenum );
    strcat( name, ".yuv" );

    // assume name is the filename to open
    // open file using "rb" = read binary file access mode
    fileptr = fopen( name, "rb" );

    // keep opening frames until there aren't any frames left
```

```
            if( fileptr == NULL )
            {
                flag = 0;
            }
        } while( flag );

        // cleanup
        munmap( fbp, screensize );
        if( ioctl( fbfd, FBIOPUT_VSCREENINFO, &orig_vinfo ) )
        {
            printf( "Error re-setting variable information.\n" );
        }
        close( fbfd );

        buffer = reset; // reset buffer pointer address to free memory
        free( buffer ); // deallocate memory block

    }

    return 0; // program success
}
```

APPENDIX C. D-DASH YUV VIDEO DATA TO ARFF CONVERSION CODE

```
/*
    YUV to ARFF Data C Program
    Jonathon Edstrom - 2015
    Converts Raw YUV data to ARFF data
    Department: NDSU ECE Graduate Research
    Project: Data-Driven Low-Cost On-Chip Memory
             With Adaptive Power-Quality Trade-off
             For Mobile Video Streaming
*/

// includes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

// instantiate globals
FILE * fileptr; // ptr for YUV data input/output files
unsigned char *buffer, *cbPtr, *crPtr; // ptr for YUV data allocated memory
unsigned char *reset; // pointer value to reset the buffer to the beginning
unsigned int filelen; // length of the input file (total bytes)
unsigned int xres; // width of the YUV video
unsigned int yres; // height of the YUV video
unsigned int lumsize; // luminance bytes per frame
unsigned int totallumbytes; // total amount of luminance bytes to process
unsigned int chromsize; // chroma bytes per frame
unsigned int singlechromsize; // chroma bytes for either Cb or Cr per frame
unsigned char cbVal, crVal; // temporary holders for buffer bytes
unsigned int framecount; // the number of frames in the video
unsigned int frames; // the number of frames to analyse
int header = -1; // boolean value to include or disinclude the .arff header
unsigned int format; // indicates 420 or 444 format specified by user input

// function to write bit value data to output file
void writeDataCSV(char cbByte, char crByte, char *str )
{
    int count;
    unsigned char value;
    for( count = 0; count < 16; count++ )
    {
        if(count < 8) // Cb Data
        {
            // bitwise AND with one-hot byte values masking out each bit
            value = ( cbByte & pow(2, 7-count) );
            if( value > 0 )
            {
                strcat( str, "1,");
            }
            else
            {
                strcat( str, "0,");
            }
        }
        else if() // Cr Data
        {
            // bitwise AND with one-hot byte values masking out each bit
            value = ( crByte & pow(2, 15-count) );
            if( value > 0 )
            {
                strcat( str, "1,");
            }
            else
            {
                strcat( str, "0,");
            }
        }
    }
}

// application entry point
int main(int argc, char * argv[])
{
    printf( "YUV to ARFF Dataset Program (J.E. 2015)\n" );

    if( argc != 7 ) // argc should be 7 for correct execution
    {
        // print argv[0] as program name w/ the following usage hint to user
        printf ( "usage: %s filename xres yres frames header format\n", argv[0] );
    }
    else // correct number of arguments
    {
        printf( "Setting things up...\n" );

        // initialize globals
        xres = atoi( argv[2] );
        yres = atoi( argv[3] );
        frames = atoi( argv[4] );
        header = atoi( argv[5] );
        if( header != 0 && header != 1 )
        {
            printf( "header value must be 0 (false) or 1 (true). Exiting.\n" );
            exit( EXIT_FAILURE );
        }
        format = atoi( argv[6] );
        if( format != 420 && format != 444 )
        {
            printf( "format value must be 420 or 444. Exiting.\n" );
            exit( EXIT_FAILURE );
        }

        // create output file name string
        int len = strlen( argv[1] ); // get length of input file name
        char filename[len];
        strcpy( filename, argv[1] ); // get input file name
        filename[len-4] = '\0'; // chop off the ".yuv" extension
        len = len + 25; // add correct amount for output file naming
        char outputstr[len];
        strcpy( outputstr, filename );
        strcat( outputstr, "_data.arff" );

        // assume argv[1] is the file name to open
        // open file using "rb" = read binary file access mode
        fileptr = fopen( argv[1], "rb" );

        // if fopen returns a NULL pointer it failed to open the file
        if( fileptr == NULL )
        {
            printf( "Could not open file. Exiting program.\n" );
            exit( EXIT_FAILURE );
        }
        else // file opened successful -> allocate memory buffer space
        {
```

```
printf( "YUV input file opened successfully!\n" );
fseek( fileptr, 0, SEEK_END ); // jump to end of file
filelen = ftell( fileptr ); // get current byte offset in file

// calculate total frames
if( format == 420 )
{
    framecount = ( 2 * filelen ) / ( xres * yres * 3 );
}
else if( format == 444 )
{
    framecount = ( filelen ) / ( xres * yres * 3 );
}

if( frames < 1 || frames > framecount )
{
    printf( "Frames value must be >= 1 and < total frames. Exiting.\n" );
    exit( EXIT_FAILURE );
}

// calculate Y (luminance) and UV (chrominance) byte component sizes
lumsize = xres*yres;
totallumbytes = lumsize*framecount;
if( format == 420 )
{
    chromsize = lumsize/2;
    singlechromsize = lumsize/4;
}
else if( format == 444 )
{
    chromsize = lumsize*2;
    singlechromsize = lumsize;
}

printf( "YUV filesize (bytes): %d\n# Frames: %d\n", filelen, framecount );
rewind( fileptr ); // jump to beginning of file

// enough memory for file + \0 (EOF)
buffer = ( char * ) malloc( filelen + 1 );
reset = buffer;
if( buffer == NULL )
{
    printf( "Failed to allocate memory. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
else
{
    printf( "Memory allocated successfully!\n" );
}

fread( buffer, 1, filelen, fileptr ); // read file into memory

buffer = reset; // reset buffer pointer address to beginning

// open file using "w" = write file access mode
fileptr = fopen( outputstr, "w" );

// if fopen returns NULL pointer it failed to open the file
if( fileptr == NULL )
{
    printf( "Could not write to .arff output file. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
```

```
else // file opened successful -> write ARFF header
{
    printf( "Created .arff file!\n" );
    fputs( "", fileptr );
    if( header == 1 )
    {
        fputs( "@relation yuvdata\n\n", fileptr );
        fputs( "@attribute Cb1 {0,1}\n", fileptr );
        fputs( "@attribute Cb2 {0,1}\n", fileptr );
        fputs( "@attribute Cb3 {0,1}\n", fileptr );
        fputs( "@attribute Cb4 {0,1}\n", fileptr );
        fputs( "@attribute Cb5 {0,1}\n", fileptr );
        fputs( "@attribute Cb6 {0,1}\n", fileptr );
        fputs( "@attribute Cb7 {0,1}\n", fileptr );
        fputs( "@attribute Cb8 {0,1}\n", fileptr );
        fputs( "@attribute Cr1 {0,1}\n", fileptr );
        fputs( "@attribute Cr2 {0,1}\n", fileptr );
        fputs( "@attribute Cr3 {0,1}\n", fileptr );
        fputs( "@attribute Cr4 {0,1}\n", fileptr );
        fputs( "@attribute Cr5 {0,1}\n", fileptr );
        fputs( "@attribute Cr6 {0,1}\n", fileptr );
        fputs( "@attribute Cr7 {0,1}\n", fileptr );
        fputs( "@attribute Cr8 {0,1}\n\n", fileptr );
        fputs( "@data\n", fileptr );
    }
    fclose( fileptr ); // close the file
}

// open file using "a" = append file access mode
fileptr = fopen( outputstr, "a" );

// if fopen returns NULL pointer it failed to open the file
if( fileptr == NULL )
{
    printf( "Could not write to .arff output file. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
else // file opened successful -> write YUV ARFF data
{
    printf( "Appending bit values to .arff file...\n" );

    buffer = reset; // set luma pointer to beginning of file
    unsigned int index = 0; // used for detecting EOF
    unsigned int count = 0; // used for moving to next set of luma data

    // calculate amount of bytes to analyze
    unsigned int length = ( filelen / framecount ) * frames;
    while( index < length )
    {
        cbPtr = buffer + lumsize; // point to beginning of next Cb
        crPtr = cbPtr + singlechromsize; // point to beginning of next Cr

        while( count < singlechromsize )
        {
            cbVal = *cbPtr;
            crVal = *crPtr;

            char *str = malloc (sizeof (char) * 100);

            writeDataCSV( cbVal, crVal, str );
            fputs( str, fileptr );

            free( str );
```

```
                              if( index < filelen )
                              {
                                  count++;
                              }

                              cbPtr++;
                              crPtr++;
                          }
                          count = 0; // reset index place holder
                          if( index < filelen )
                          {
                              // move buffer to next set of luma bytes (start of next frame)
                              buffer = buffer + lumsize + chromsize;
                              index = index + lumsize + chromsize;
                          }
                      }

                      fclose( fileptr ); // close the file
                  }

                  printf( "Freeing up allocated memory\n" );
                  buffer = reset; // reset buffer pointer address to free memory
                  free( buffer ); // deallocate memory block

                  printf( "Data was successfully output!\n" );
              }
          }
          return 0; // program success
      }
```

APPENDIX D. DPSR ERROR MAPPING MATLAB CODE

```
% SRAM Array Error Mapping MATLAB Program
% Plots a memory error map based on a uniformly distributed 2x2 matrix
% Code by: Jonathon Edstrom (4/26/2016)
function errorMapping( rows, columns, failureRate )
    % get uniformly distributed random 2x2 matrix of decimals in 0.0-1.0
    sramArray = rand( rows, columns );
    % set up the plot for the particular rows x columns size
    figure
    if( columns > 32 && columns <= 128 )
        set(gca,'XTick',0:32:columns);
        xAdjustment = 1;
    elseif( columns > 128 )
        set(gca,'XTick',0:256:columns);
        xAdjustment = 4;
    else
        set(gca,'XTick',0:1:columns-1);
        xAdjustment = 0.25;
    end
    if( rows > 32 && rows <= 128 )
        set(gca,'YTick',0:32:rows);
        yAdjustment = 10;
        markerSize = 8;
    elseif( rows > 128 )
        set(gca,'YTick',0:256:rows);
        yAdjustment = 20;
        markerSize = 4;
    else
        set(gca,'YTick',0:1:rows-1);
        yAdjustment = 1;
        markerSize = 16;
    end
    axis([-xAdjustment,columns,-yAdjustment,rows])
    title('Error map in SRAM array (errors are uniformly distrubuted)');
    xlabel('Column');
    ylabel('Row');
    hold on
    % open a progress bar for feedback to user
    progress = waitbar(0,'?/? (0%) Complete...');
    % calculate where the errors are based on the failureRate
    for i = 1:rows
       % calculate percentage completed and update progress bar
       percent = ( i / rows ) * 100.0;
       waitbar( i/rows, progress, sprintf( '%d/%d (%d%%) Complete...',
                 i, rows, uint8(percent) ) )
       for j = 1:columns
          if( sramArray( i, j ) <= failureRate )
              % plot point if random value is within the failureRate range
              plot(j-1,i-1,'.','MarkerSize',markerSize,'Color','k')
              % display which bit as text if plot is small enough
              if( rows <= 64 )
                text(j-1,i-1,sprintf(' Bit %d',j-1))
              end

          end

       end

    end
    close(progress); % close the progress bar
end
```

APPENDIX E. DPSR WORD LINE FAULTS PROBABILITY CODE

```c
/*
    Word Line Faults Monte Carlo Simulation Program
    Jonathon Edstrom - 2016
    Calculates the probability of faults that occur in a
    single word line using Monte Carlo methods based on failure rate
    Department: NDSU ECE Graduate Research
    Project: Data-Pattern Enabled Self-Recovery Multimedia
             Storage System for Near-Threshold Computing
*/

// includes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// function definitions
unsigned char injectFault( double errorRate );

int main(int argc, char * argv[])
{
    unsigned int wordLineBitSize; // size of the word line in bytes
    unsigned long long int numOfTrials; // # of trials to calculate probability
    double failureRate; // the failure rate of a single bit cell
    unsigned int seedValue; // used to seed the random number generator

    if( argc != 4 && argc != 5 ) // argc should be 4 or 5 for correct execution
    {
        // print the program name with the following usage hint to user
        printf ( "usage: %s wordLineBitSize numOfTrials failureRate
                 seedValue[optional]\n", argv[0] );
    }
    else // correct number of arguments
    {
        wordLineBitSize = atoi( argv[1] );
        numOfTrials = atoi( argv[2] );
        failureRate = atof( argv[3] );
        if( argc == 5 )
        {
            seedValue = atoi( argv[4] ); // seed to user specified argument value
        }
        else
        {
            srand( ( unsigned ) time( NULL ) ); // seed to random number using time
        }

        // check arguments are valid inputs
        if( wordLineBitSize <= 0 || numOfTrials <= 0 || failureRate < 0.0 ||
            failureRate > 1.0 )
        {
            printf( "Argument error, wordLineBitSize and numOfTrials must be positive
                     integers, failure rate must be a decimal between 0.0 and 1.0
                     inclusive. Now exiting.\n" );
            exit( EXIT_FAILURE );
        }

        unsigned long long int faultCount[wordLineBitSize+1]; // 0 to n faults count
        unsigned long long int i; // loop variable
        unsigned int j, k; // loop variables
        unsigned int faultsInWordLine = 0; // total faults in word line
        unsigned char wordLine[wordLineBitSize]; // wordline w/ 'wordLineBitSize' bits
        unsigned long long int specificFault = 0; // rule/correlation double faults
        unsigned long long int associationFault = 0; // association rule double faults
        unsigned long long int correlationFault = 0; // correlation double faults

        // initialize faultCount array values to zero
        for( j = 0; j < wordLineBitSize+1; j++ )
        {
            faultCount[j] = 0;
        }

        // run Monte Carlo simulation for numOfTrials argument
        for( i = 0; i < numOfTrials; i++ )
        {
            // reset total fault count
            faultsInWordLine = 0;

            // inject faults into wordLine at given failure rate
            for( j = 0; j < wordLineBitSize; j++ )
            {
                wordLine[j] = injectFault( failureRate );
                if( wordLine[j] == 1 )
                {
                    faultsInWordLine++;
                }
            }

            // association rule fault check
            // Cb1 & Cb2
            if( ( wordLine[0] == 1 && wordLine[1] == 1 ) || ( wordLine[16] == 1 &&
                  wordLine[17] == 1 ) )
            {
                specificFault++;
                associationFault++;
                //printf( "Association Failure: Cb1 & Cb2 Both Failed!\n" );
            }
            // Cr1 & Cr2
            if( ( wordLine[8] == 1 && wordLine[9] == 1 ) || ( wordLine[24] == 1 &&
                  wordLine[25] == 1 ) )
            {
                specificFault++;
                associationFault++;
                //printf( "Association Failure: Cr1 & Cr2 Both Failed!\n" );
            }
            // Cr1 & Cr3
            if( ( wordLine[8] == 1 && wordLine[10] == 1 ) || ( wordLine[24] == 1 &&
                  wordLine[26] == 1 ) )
            {
                specificFault++;
                associationFault++;
                //printf( "Association Failure: Cr1 & Cr3 Both Failed!\n" );
            }

            // Cb correlation fault check
            for( k = 2; k < 8; k++ )
            {
                if( wordLine[k] == 1 && wordLine[k+16] == 1 )
                {
                    specificFault++;
                    correlationFault++;
                }
            }
```

```
            // Cr correlation fault check
            for( k = 11; k < 16; k++ )
            {
                if( wordLine[k] == 1 && wordLine[k+16] == 1 )
                {
                    specificFault++;
                    correlationFault++;
                }
            }

            faultCount[faultsInWordLine]++;
        }

        printf( "\n" );

        // print out general amount of fault counts
        for( j = 0; j < wordLineBitSize+1; j++ )
        {
            if( faultCount[j] > 0 )
            {
                printf( "%u Fault(s): %llu\n", j, faultCount[j] );
            }
        }

        // print specific results
        printf( "\nCorrelation Double Fault: %llu\nAssociation Rule Double Fault:
                 %llu\nTotal Double Faults: %llu\n\n", correlationFault,
                 associationFault, specificFault );
    }

    return 0; // program success
}

// function that will calculate if a fault should exist based on failure rate
unsigned char injectFault( double errorRate )
{
    double randomNumber;

    // Don't divide by 0
    if( errorRate == 0.0 )
    {
        randomNumber = 1.0;
    }
    else
    {
        double error = ( 1 / errorRate );
        unsigned long long int errorVal = ( unsigned long long int ) error;
        randomNumber = rand() % ( errorVal + 1 );
    }

    if( randomNumber >= 1.0 )
    {
        // no fault
        return 0;
    }
    else
    {
        // fault
        return 1;
    }
}
```

APPENDIX F. DPSR ASSOCIATION RULE AND CORRELATION CODE

```
/*
    Chroma Bit Association Rule or Correlation Program
    Jonathon Edstrom - 2016
    Apply associations or correlation at a specified bit cell failure rate
    Department: NDSU ECE Graduate Research
    Project: Data-Pattern Enabled Self-Recovery Multimedia
             Storage System for Near-Threshold Computing
*/

// includes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

// instantiate globals
FILE *fileptr; // pointer for YUV data input/output files
unsigned char *buffer, *origBuffer; // pointer for YUV data allocated memory
unsigned char *reset, *origReset; // ptr value to reset the buffer
unsigned long long int filelen; // length of the input file (total bytes)
unsigned long long int xres; // width of the YUV video
unsigned long long int yres; // height of the YUV video
unsigned long long int lumsize; // luminance bytes per frame
unsigned long long int chromsize; // chrominance bytes per frame
unsigned long long int singlechromsize; // size of one chrominance component
unsigned char cb,cbComp,cr,crComp; // temporary holder for buffer byte
unsigned long long int framecount; // the number of frames in the video
long double errorRate; // decimal that sets how often a bit has a fault
long double randomNumber; // stores random numbers for inputting faults
unsigned int cborcr, bit, corrorrule, seedValue; // user input options

// function definitions
unsigned char getBitValue( unsigned char byte, int bitNum );
unsigned char injectFault();

// application entry point
int main(int argc, char * argv[])
{
    printf( "Chroma Bit Association Rule or Correlation Program (J.E.
             2016)\n" );

    if( argc != 8 ) // argc should be 8 for correct execution
    {
        // print program name with the following usage hint to user
        printf ( "usage: %s filename xres yres corr(0)-or-rule(1)-or-
                  noCorrection(2) failureRate cb(0)-or-cr(1) bitNum(MSB=1-
                  >LSB=8)\n", argv[0] );
    }
    else // correct number of arguments
    {
        printf( "Setting things up...\n" );

        // initialize globals
        xres = atoi( argv[2] );
        yres = atoi( argv[3] );
        corrorrule = atoi( argv[4] );
        errorRate = atof( argv[5] );
        cborcr = atoi( argv[6] );
```

```
bit = atoi( argv[7] );

// check bounds on arguments
if( cborcr < 0 || cborcr > 1 )
{
    printf( "%s is an invalid input. cb(0)-or-cr(1) argument must be
             0 or 1. Exiting program.\n", argv[4] );
    exit( EXIT_FAILURE );
}
if( bit < 1 || bit > 8 )
{
    printf( "%s is an invalid input. bitNum(MSB=1->LSB=8) argument
             must be an integer between 1 and 8 inclusive. Exiting
             program.\n", argv[5] );
    exit( EXIT_FAILURE );
}
if( corrorrule < 0 || corrorrule > 2 )
{
    printf( "%s is an invalid input. corr(0)-or-rule(1)-or-
              noCorrection(2) argument must be 0, 1, or 2. Exiting
              program.\n", argv[6] );
    exit( EXIT_FAILURE );
}
if( errorRate < 0.0 || errorRate > 1.0 )
{
    printf( "%s is an invalid input. failureRate argument must be a
             decimal value between 0.0 and 1.0 inclusive. Exiting
             program.\n", argv[7] );
    exit( EXIT_FAILURE );
}

// set up random number generator
if( cborcr == 0 )
{
    seedValue = bit;
}
else if( cborcr == 1 )
{
    seedValue = bit+8;
}
srand(seedValue); // seed to user specified argument value

// create output file name string
int len = strlen( argv[1] ); // get length of input file name
char filename[len];
strcpy( filename, argv[1] ); // get input file name

// assume argv[1] is the filename to open
// open file using "rb" = read binary file access mode
fileptr = fopen( argv[1], "rb" );

// if fopen returns a NULL pointer it failed to open the file
if( fileptr == NULL )
{
    printf( "Could not open file: \"%s\". Exiting program.\n",
            argv[1] );
    exit( EXIT_FAILURE );
}
else // file opened successful -> allocate memory buffer space
{
    printf( "YUV file: \"%s\" opened successfully!\n", argv[1] );
    fseek( fileptr, 0, SEEK_END ); // jump to end of file
    filelen = ftell( fileptr ); // get current byte offset in file
```

```
framecount = ( 2 * filelen ) / ( xres * yres * 3 );

// calculate Y (luma) and UV (chroma) byte component sizes
lumsize = xres*yres;
chromsize = lumsize/2;
singlechromsize = chromsize/2;

printf( "Size of YUV file in bytes: %llu\nNumber of frames in
         video: %llu\n", filelen, framecount );
rewind( fileptr ); // jump to beginning of file

// enough memory for file + \0 (EOF)
buffer = ( char * ) malloc( filelen + 1 );

// enough memory for file + \0 (EOF)
origBuffer = ( char * ) malloc( filelen + 1 );

reset = buffer;
origReset = origBuffer;
if( buffer == NULL || origBuffer == NULL )
{
    printf( "Failed to allocate memory. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
else
{
    printf( "Memory allocated successfully!\n" );
}

fread( buffer, 1, filelen, fileptr ); // read file into memory
rewind( fileptr ); // jump to beginning of file
fread( origBuffer, 1, filelen, fileptr ); // read file to memory

buffer = reset; // reset buffer pointer address to beginning
origBuffer = origReset; // reset buffer ptr address to beginning

int index = 0;
int count = 0;
while( index < filelen )
{
    buffer += lumsize; // skip luma data & jump to Cb data
    origBuffer += lumsize;
    index += lumsize; // inc. index by luma frame size (bytes)
    srand(seedValue); // seed each frame to same fault positions

    while( count < singlechromsize )
    {
        cbComp = *origBuffer;
        cb = *( origBuffer + 1 );
        crComp = *( origBuffer + singlechromsize );
        cr = *( origBuffer + singlechromsize + 1 );

        unsigned char faultExists;
        faultExists = injectFault();
        unsigned char replacementBit;

        // Cb
        if( cborcr == 0 )
        {
            // correlation
            if( corrorrule == 0 )
            {
                if( faultExists == 1 )
                {
                    replacementBit = getBitValue( cbComp, bit );
                    *( buffer + 1 ) = ( cb & ( 0 << ( 8 - bit ) ) );
                    *( buffer + 1 ) = ( cb | ( replacementBit << ( 8 - bit
                                        ) ) );
                }
            }
            // association rule
            else if( corrorrule == 1 )
            {
                if( faultExists == 1 )
                {
                    // Cb1
                    if( bit == 1 )
                    {
                        replacementBit = getBitValue( cb, 2 );
                        replacementBit = ~replacementBit;
                        *( buffer + 1 ) = ( cb & ( 0 << ( 8 - bit ) ) );
                        *( buffer + 1 ) = ( cb | ( replacementBit << ( 8 -
                                            bit ) ) );
                    }
                    // Cb2
                    else if( bit == 2 )
                    {
                        replacementBit = getBitValue( cb, 1 );
                        replacementBit = ~replacementBit;
                        *( buffer + 1 ) = ( cb & ( 0 << ( 8 - bit ) ) );
                        *( buffer + 1 ) = ( cb | ( replacementBit << ( 8 -
                                            bit ) ) );
                    }
                    // default to correlation otherwise
                    else
                    {
                        replacementBit = getBitValue( cbComp, bit );
                        *( buffer + 1 ) = ( cb & ( 0 << ( 8 - bit ) ) );
                        *( buffer + 1 ) = ( cb | ( replacementBit << ( 8 -
                                            bit ) ) );
                    }
                }
            }
            // no correction -> inject error
            else
            {
                if( faultExists == 1 )
                {
                    replacementBit = getBitValue( cb, bit );
                    replacementBit = ~replacementBit;
                    *( buffer + 1 ) = ( cb & ( 0 << ( 8 - bit ) ) );
                    *( buffer + 1 ) = ( cb | ( replacementBit << ( 8 - bit
                                        ) ) );
                }
            }
        }
        // Cr
        else
        {
            // correlation
            if( corrorrule == 0 )
            {
                if( faultExists == 1 )
                {
                    replacementBit = getBitValue( crComp, bit );
                    *( buffer + singlechromsize + 1 ) = ( cr & ( 0 << ( 8
                                                               - bit ) ) );
                    *( buffer + singlechromsize + 1 ) = ( cr | (
                                        replacementBit << ( 8 - bit ) ) );
                }
            }
            // association rule
            else if( corrorrule == 1 )
            {
                if( faultExists == 1 )
                {
                    // Cr1
                    if( bit == 1 )
                    {
                        replacementBit = getBitValue( cr, 2 );
                        replacementBit = ~replacementBit;
                        *( buffer + singlechromsize + 1 ) = ( cr & ( 0 <<
                                                            ( 8 - bit ) ) );
                        *( buffer + singlechromsize + 1 ) = ( cr | (
                                        replacementBit << ( 8 - bit ) ) );
                    }
                    // Cr2
                    else if( bit == 2 )
                    {
                        replacementBit = getBitValue( cr, 1 );
                        replacementBit = ~replacementBit;
                        *( buffer + singlechromsize + 1 ) = ( cr & ( 0 <<
                                                            ( 8 - bit ) ) );
                        *( buffer + singlechromsize + 1 ) = ( cr | (
                                        replacementBit << ( 8 - bit ) ) );
                    }
                    // Cr3
                    else if( bit == 3 )
                    {
                        replacementBit = getBitValue( cr, 1 );
                        replacementBit = ~replacementBit;
                        *( buffer + singlechromsize + 1 ) = ( cr & ( 0 <<
                                                            ( 8 - bit ) ) );
                        *( buffer + singlechromsize + 1 ) = ( cr | (
                                        replacementBit << ( 8 - bit ) ) );
                    }
                    // default to correlation otherwise
                    else
                    {
                        replacementBit = getBitValue( crComp, bit );
                        *( buffer + singlechromsize + 1 ) = ( cr & ( 0 <<
                                                            ( 8 - bit ) ) );
                        *( buffer + singlechromsize + 1 ) = ( cr | (
                                        replacementBit << ( 8 - bit ) ) );
                    }
                }
            }
            // no correction -> inject error
            else
            {
                if( faultExists == 1 )
                {
                    replacementBit = getBitValue( cr, bit );
                    replacementBit = ~replacementBit;
                    *( buffer + singlechromsize + 1 ) = ( cr & ( 0 << ( 8
                                                                   - bit ) ) );
                    *( buffer + singlechromsize + 1 ) = ( cr | (
                                        replacementBit << ( 8 - bit ) ) );
```

```
                            }
                        }
                    }

                    if( index < filelen )
                    {
                        buffer++;
                        origBuffer++;
                        index++;
                        count++;
                    }
                }
                count = 0; // reset index place holder
                buffer += singlechromsize; // move to next frame
                origBuffer += singlechromsize; // move to next frame
                index += singlechromsize; // move to next frame
            }

            buffer = reset; // reset buffer pointer address
            origBuffer = origReset; // reset buffer pointer address

            // open file using "w+b" = write/update binary file access mode
            fileptr = fopen( filename, "w+b" );

            // if fopen returns NULL pointer it failed to open the file
            if( fileptr == NULL )
            {
                printf( "Could not write output file. Exiting program.\n" );
                exit( EXIT_FAILURE );
            }
            else // file opened successful -> write back YUV data
            {
                printf("Created file: \"%s\", writing data...\n", filename);
                fwrite( buffer, 1, filelen, fileptr ); // write data to file
                fclose( fileptr ); // close the file
            }

            printf( "Freeing up allocated memory\n" );
            buffer = reset; // reset buffer pointer address to free memory
            free( buffer ); // deallocate memory block
            free( origBuffer ); // deallocate memory block

            printf( "Data was successfully output!\n" );
        }
    }

    return 0; // program success
}

// function to check value of the bit specified for a given byte
unsigned char getBitValue( unsigned char byte, int bitNum )
{
    // instantiate local variables
    unsigned char value;

    // bitwise AND with one-hot byte value
    value = ( byte & pow(2, 7-bitNum) );
    if( value > 0 )
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

// function that will calculate if a fault should exist based on failure rate
unsigned char injectFault()
{
    // Don't divide by 0
    if( errorRate == 0.0 )
    {
        randomNumber = 1.0; // no error
    }
    else
    {
        long double error = ( 1 / errorRate );
        unsigned long long int errorVal = ( unsigned long long int ) error;
        randomNumber = rand() % ( errorVal + 1 );
    }

    if( randomNumber >= 1.0 )
    {
        // no fault
        return 0;
    }
    else
    {
        // fault
        return 1;
    }
}
```