Part C Individual Project

Final Report

Name: Thomas Finch

ID Number: B417345

Programme: Electronic and Computer Systems Engineering

Module Code: 17ELC025

Supervisor: Vincent Dwyer

Project title: Graphics Output Device

Abstract:

This paper presents the design, development and implementation of a graphics output device that facilitates the navigation and viewing of Bitmap image files stored on an SD card.

During the design phase, the problem of retrieving and displaying image data from an SD card without access to a computer was identified. This was followed by preliminary research used to develop requirements and a technical specification. An emphasis was placed on the development of a product that could be constructed from low cost electronic components.

The development phase considered the overall system implementation, followed by the research and development of the various related sub systems. Specifically, memory storage interfacing, graphics display interfacing and the overall system architecture.

The final implementation phase considered the physical hardware and software construction of the system described during the development phase. In practice, this involved development of an elaborate microcontroller based system implemented on a breadboard with the appropriate control software developed in Assembly and C++.

Contents

1 Introduction

1.1 Purpose

The SD card [1] is the typical device for storing and sharing photos, yet the media within cannot be accessed without a computer or equivalent device. There is no purpose-built product that enables the sharing of photos directly from an SD card to a display available on the mass-market.

The purpose of the project was to create a low-cost, low-power device that could output photos to a television monitor or screen, from image data stored on a removable flash storage device compatible with consumer digital cameras.

1.2 Aim and objectives

Aim: The creation of a device that can output to a television or monitor, image data selected from a removable storage medium via user interaction using a graphical user interface [2].

Objective(s):

- The device must be constructed in such a way so as to have the capability to interface with removable storage media compatible with digital cameras.
- The device must be constructed in such a way so as to facilitate a mechanism by which a user can navigate between the image files on the removable storage media.
- The device must be constructed in such a way so as it facilitate a mechanism by which the a selected image file can be viewed on screen.
- The device must be constructed in such a way so as to maximise the accessibility of the potential user base.

1.3 Background

Following initial research into existing solutions, it was found that certain digital cameras featured the ability to output image data directly from their internal SD card to a television. The operational characteristics of these types of cameras is outlined below.

A digital camera will typically utilise a Secure Digital (SD) memory card as its removable flash based storage device. SD cards have a wide range of capacities enabling hundreds or thousands of photos to stored. To facilitate interfacing, SD cards support a serial communication protocol with a specific command-response structure enabling data to be written to and read from the flash memory.
To maximise the number of photos that can be stored on a card, a digital camera will typically store image files in a JPEG compressed lossy image format [3].
To manage the numerous image files, the digital camera will implement a file system architecture on the SD card. The FAT32 file system format [4] is most commonly implemented due to its widespread adoption and simple but robust implementation.

To facilitate a video output functionality, digital cameras will usually feature a HDMI [5] or AV [6] output with the appropriate video signal generation hardware.
However, with the decline in the adoption of analogue televisions in recent times, AV outputs

are no longer supported in the most recent models of digital cameras.
In conjunction to image output, the digital camera will also introduce a graphical menu overlay. This enables the user to select between and view a specific image file stored on the memory card.

2 System design

2.1 Technical requirements

Functional

- The device must support communication interfacing with a flash memory storage device commonly utilised in digital cameras.
- The device must facilitate the ability to distinguish between and read data from files stored on the removable storage media.
- The device must facilitate an image output interface compatible with a common display interface standard.
- The device must be able to interpret the data of image file formats commonly implemented in digital cameras.
- Using the image data file, the device then must then facilitate a mechanism by which the image can be outputted via the display interface.
- The device should enable, through user interaction, the navigation and output, via the display interface, of specific image files on the memory storage device.

Non-functional

- The device must be developed using readily available electronic components.
- The electronic components utilised must remain within the manufacturers rated operational tolerance range.
- The device must provide adequate heat dissipation for the internal electronic hardware at normal room temperatures.
- The device should minimise cost of manufacture by reducing both the value and number of components required.
- The device must have sufficiently low power usage so that it can be powered from a small battery source.
- The device should have sufficient technical performance so as to provide an acceptable user experience.

2.2 Specification

To manage scope boundaries and to establish the realistic outcome of the project, it was paramount that a clear product specification was developed.

With an emphasis on minimising the cost of manufacture, and in the interests of developing a system using readily available low cost electronic components, it was clear that an approach utilising a programmable logic device would not meet the requirements of the project.
As such, the clear alternative was the use of a low cost, low power readily available general purpose microcontroller, supported by additional electronic components to meet the requirements of the project. Specifically, an 8-bit AVR Atmega328P microcontroller [7] was selected. However, following the initial prototyping phases a software compatible Atmega324A microcontroller [8] was utilised due to the requirement of an increased pin count for I/O interfacing.

This brought with it some potential limitations in the system implementation. With the limited processing capabilities of a low cost microcontroller, it would not be feasible to support common digital display interface standards such as HDMI due to their relatively high processing requirements.
Therefore, a display standard with lower processing requirements but still with mainstream adoption would be required.
Although adoption is beginning to decline, a display standard that remains supported by the majority of televisions and monitors is the Video Graphics Array (VGA) display standard [9]. As such, support for the VGA standard was deemed the most appropriate to conform to the devices required functionality.

Having established the prevalent usage of SD cards in digital cameras, it was clear that the product should provide support for this storage media.
Furthermore, as the SD card format has such mainstream adoption almost any digital processing unit with digital I/O pins is capable of serially interfacing with it. Many microcontrollers also feature hardware accelerated communication modules facilitating the microprocessor to carry out additional functionality while serial interfacing is occurring.

Support was also required for a file system architecture used to manage the image files on the hardware media.
As digital cameras typically utilised the FAT32 file system support was still requirement, however it was unclear as to the feasibility of this goal.

As digital cameras typically store image files in a compressed JPEG format, this was the image format the system should support.
However, due to complexity in implementation, it was decided that this was not a primary goal, and as such, implementation would only be considered following the satisfaction of all other requirements. So it was decided that support would only initially be provided for the simpler device independent uncompressed Bitmap file format [10].

The final consideration of the overall system implementation was the user interfacing. As the full scope of the implementation had not been established in the technical requirements. It was decided that a simple navigation would be provided through the use of push buttons allowing the user to select between image files on the SD card.

A summary of the product specification can be found in the bullet points below.

- Support for Secure Digital (SD) cards via a low level serial communication interface
- Support for a FAT32 file system architecture to facilitate file interfacing and management

- Support for a Video Graphics Array (VGA) interface for image output on a television or monitor
- Support for reading and interpreting image data in Bitmap files
- Physical push button interface enabling users to select between image files on the SD card

2.3 System organisation

To manage the overall organisation of the device, an overview diagram highlighting each of the main technical constituents was created. This is illustrated in Figure 1 below.

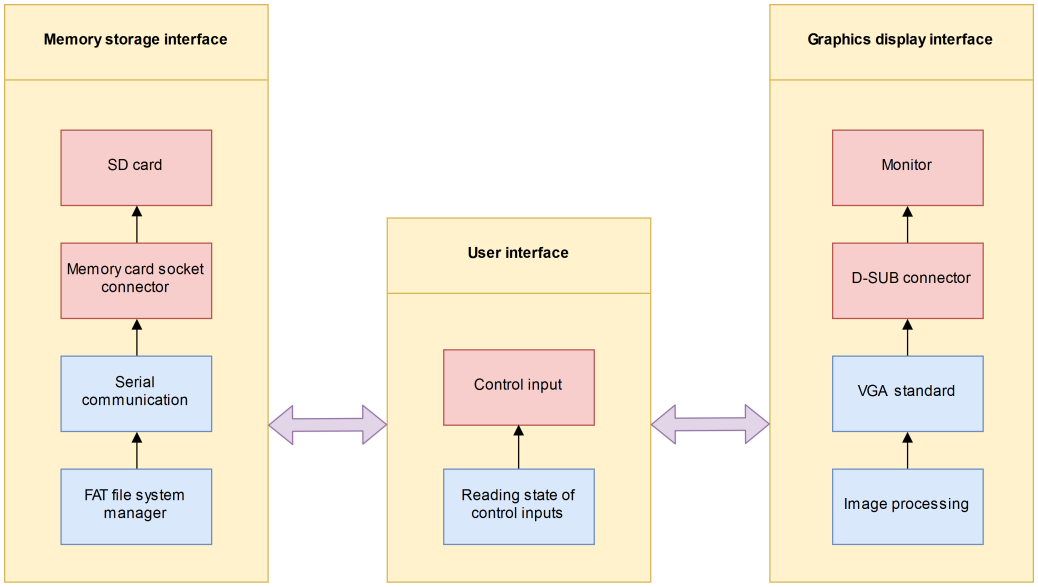

Figure 1: System overview diagram

The orange regions represent the highest level of abstraction in the overall design and the pink and blue regions represent the lower hardware and software levels respectfully.
Further detail regarding these technical constituents can be found in the appropriate sections within the report.

Although not explicitly indicated in the diagram a large aspect of the project was the infrastructure required to facilitate the intercommunication between the various sub systems. Detail regarding this has been provided in the "System architecture" section of the report.

3 Memory storage interfacing

The principle investigation point for this aspect of the project was the physical SD card interfacing and FAT file system interfacing.
Prior to physical implementation and prototyping, a significant amount of initial research was required. As such, the research and hardware and software development have been separated into their respective "Research" and "Design and Methodology" sections.

3.1 Research

3.1.1 Secure digital standard

The Secure Digital (SD) non-volatile memory card was introduced in 1999, created as part of a collaboration between SanDisk, Panasonic and Toshiba [11]. The following year, the companies set up the SD Association (SDA) a non profit organisation set up to create and promote the SD standards.
These standards enable manufacturers to easily implement an SD interfacing protocol in their products. As such, these standards are readily available from the SD association website [12] and can be found in the various provided specification documents.
Since its inception, there have been multiple revisions to the original standard, enabling SD card devices to feature significantly larger memory capacities and faster operational speeds.

3.1.2 SD memory card internal organisation

Internally an SD card consists of two main structures [13]:

- Memory core
- Card controller, with associated read-only registers

The memory core, composed of flash memory, is what dictates the maximum storage capacity of the card and it is where read write data is stored. The memory core is divided into 512 byte sectors or blocks and as such, typical interactions with the memory core will require the reading/writing to multiple sectors.
The card controller manages both the transactions with the memory core, but also enables interfacing via external devices. The various associated card controller registers contain relevant operating information about the SD card. However, in a practical implementation that enables reading and writing to the memory core, they are not required.
To facilitate external interfacing an SD card features nine external pin connections. A diagram illustrating these connections can be found in Figure 2 below.

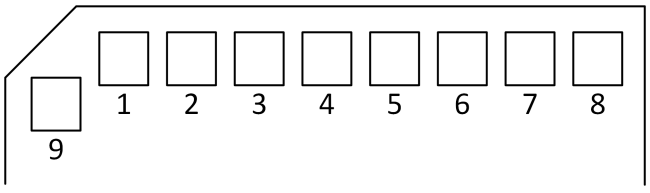

Figure 2: SD card external pin connections

A description of the functionality for each pin can be found in Table 1 below.

Pin	Pin Name	Description
1	CS	Chip select (active low)
2	DI	Data input
3	GND	Ground
4	VCC	Supply voltage (2.7-3.6V)
5	SCK	Serial clock
6	GND	Ground
7	DO	Data output
8	RSV	Reserved
9	RSV	Reserved

Table 1: SD card external pin descriptions [14]

3.1.3 Communication interface and software protocol

In order to interface with an SD card a specific serial communication interface and a higher level software protocol is required.
The default communication interface for interfacing with an SD card via an external device is the proprietary "SD mode" [15]. This enables very high baud rates in excess of 50 MBits/s. However, it is less well documented and requires a more complex implementation.

An alternative communication interface, and the form that was utilized in this project, was the "SPI mode" [16]. This is a byte orientated serial communication protocol that utilizes the Serial Peripheral Interface (SPI) bus. And although this interfacing mode only supports a subset of the full SD protocol and is limited by the maximum SPI frequency of the interfacing device. The main advantage is the ease in implementation, and as such enables a multitude of different devices including the AVR microcontroller utilised in this project, to interface with an SD card. Further information regarding the SPI bus protocol can be found at [Appendix 1].

The higher level software protocol is a command-response orientated message exchange protocol. Whereby, the interfacing device will send a valid command, requesting the SD card to perform a specific operation, and following this, the SD card, having recognised a valid command, will respond with a specific response message.
Each command has a corresponding response message. Although SD cards support up to 64 different commands, in typical operation, only a handful are required. Note, prior and during the command send operation, the chip select pin must remain active low. A table found in [Appendix 2] outlines the most commonly implemented command and response messages.

A diagram, outlining the packet structure for a typical command response message, can be found in Figure 3 below. Note, each command also includes Cyclic Redundancy Check (CRC) error detecting code, however this is an optional feature in SPI mode.

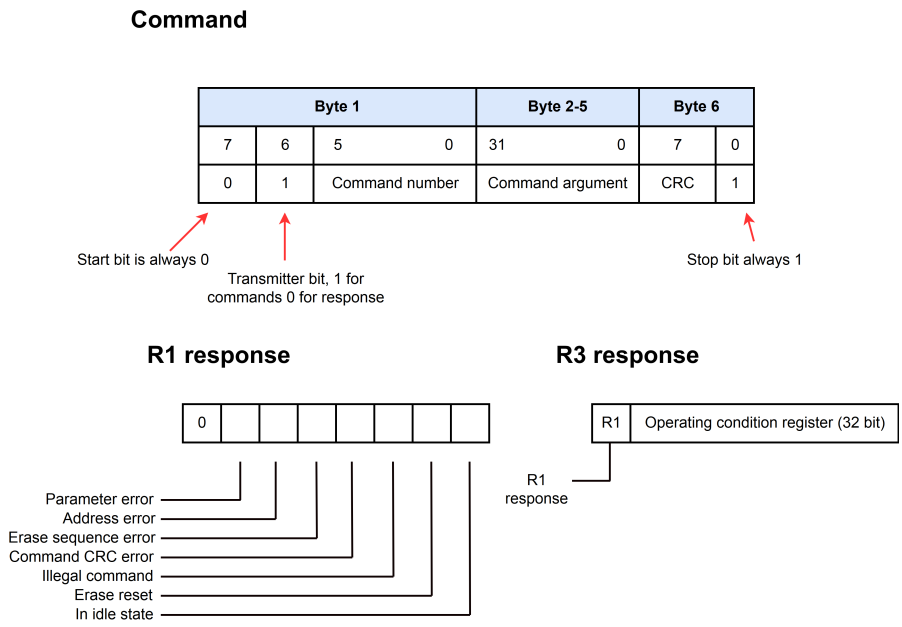

Figure 3: Command response packet structure [19]

As the SPI protocol is fully duplex and the serial clock is controlled by the interfacing master device, in order for the interfacing device to read the response messages, the device must continuously send data and check for a valid response from the SD card. The card will usually respond within 0-8 bytes, however, some operations may take longer and will respond with an R1 message followed by a busy flag [19].

In conjunction to the command response messages, during data transfer operations, such as read and write commands, data packets and responses are also exchanged. Figure 4 below outlines structure of these data packets.

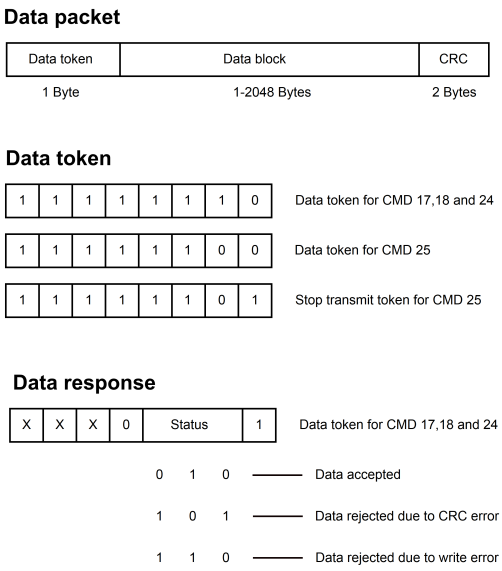

Figure 4: Data packet and response structure [19]

Further detail regarding their implementation can be found in the next sub section "Hardware and Software driver".

3.2 Design and methodology

3.2.1 Hardware and software driver

To facilitate file interfacing on the SD card, consideration of the low level hardware and software driver implementation was first required. The software driver for the SD interfacing was developed in C.

As previously established, the SPI protocol would be utilised to enable serial interfacing with the SD card.
Fortunately, the AVR microcontroller features a dedicated onboard SPI module. This meant that the protocol could be implemented in hardware, and therefore significantly reduce the overhead in the software implementation.
Control of this communication module, including attributes such as operational frequency and mode, were facilitated through the manipulation of specific control registers in the microcontroller. Detail regarding how this module can be figured is outlined the microcontrollers datasheet.

The next step was to consider how to begin reading and writing to the SD card using the SD commands sent via the SPI interface. It was found that before the SD card can accept accept read, write and various other commands, the SD card must first be set in the "ready" operational state [20].
To configure the card into this state, an initialisation sequence is required. This consists of a series of specific SD commands. Also, during initialisation, the operational freqeuency cannot exceed 400 KHz otherwise initialisation will fail. A description of these initialisation operations carried out using the microcontroller can be found outlined in the bullet points below.

- It was ensured that the SD card device was at an operating voltage of between 2.7 and 3.6 V.
- The chip select pin was deselected.
- Approximately 80 serial clock pulses were then generated by the Atmega microcontroller.
- The chip select pin was then selected.
- SD Command 0 (Software reset) was then sent to the SD card. This command requires a CRC value of "0x95" to be included as part of the command. The microcontroller then waited until an R1 response of "0x01" was received. This indicated the that card was in idle state.
- SD Command 8 (Check voltage range) was then sent to the device. This command is only applicable to SD card's following SDC revision 2 or later, it is however a requirement to initialise these card revisions. If the card is of an earlier revision, the card will simply respond with an R1 response of "0x05", indicating an illegal command. A card with a later SDC revision will respond with the 12 bit response R7, indicating whether the operational voltage is acceptable. In this software implementation the response received was simply ignored.
- SD Command ACM41 (Initiate initialization process) was then sent to the device. This command consisted of Command 55 (Leading ACMD41 Command) followed by Command 1 (Initiate initialization process). A few hundred milliseconds later an R1 response of "0x00" was then received, indicating that the card was ready to receive further commands.

A schematic illustrating the circuit diagram used during this initial prototyping phase can be found in Figure 5 below. In order to read and verify the response from the SD card, LED's connected to PORTD on the microcontroller were used to display the byte response message.

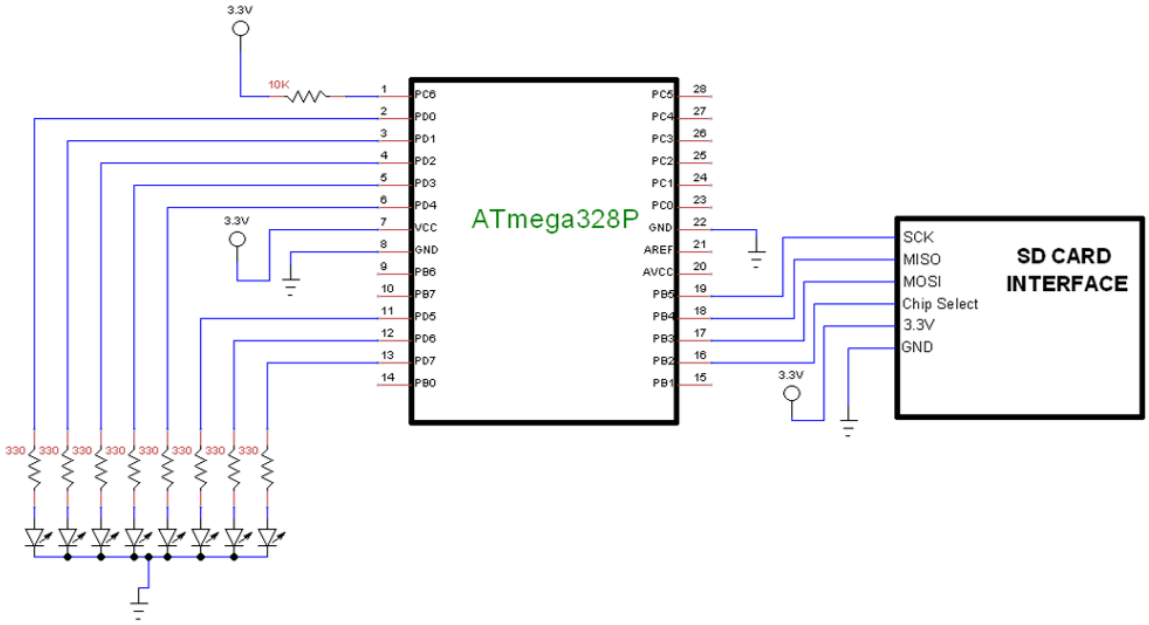

Figure 5: Circuit schematic of prototype circuit

As the microcontroller has a 1.8 V - 5.5 V voltage operating range it was also be powered by the same 3.3 V power source as the SD card, mitigating the requirement for a voltage level shifter. All data and commands sent to the SD card was done so via the MOSI connection and all responses were received by the microcontroller via the MISO connection.

In practice, the initialisation sequence proved challenging to perform successfully. This was because, although not specified in the protocol, a dummy byte was required before every command.

With the SD card in the "ready" operational state, the card was ready to accept read and write commands [19]. These commands, similarly to the initialisation sequence, required a specific series of SD commands.
Although the SD card facilitated a read and write multiple block operation, during this prototyping phase, only a single read and write block sequences were considered. A description of the read single block sequence is outlined below.

- The chip select pin was selected.
- Two dummy bytes of "0xFF" were then sent to the card
- SD Command 17 (Read a block) was then sent to the device. CRC bytes were also required, however the value did not matter. The microcontroller then waited until an R1 response of "0x00" was received.
- The Microcontroller then sent dummy bytes and checked response from SD card until the data token "0xFE" was received, which indicated the start of the data packet.
- All bytes of this data packet, including the CRC bytes, were then be read by the microcontroller.
- The chip select pin was then deselcted.

A description of the write single block sequence is outlined below.

- The chip select pin was selected.
- Two dummy bytes of "0xFF" were sent to the card
- SD Command 24 (Write to a block) was then sent to the card. CRC bytes were also required, however the value did not matter. Also included in the command was the address of a block in the SD card's memory core. The microcontroller then waited until an R1 response of "0x00" was received.
- The Microcontroller then sent a further two dummy bytes.
- A data packet, consisting of the data token, "0xFE", a 512 byte data block and a CRC value was then sent to the card. An R1 response of "0x00" was then be received from the card.
- The chip select pin was then deselected.

Utilising the circuit diagram above and these read and write command sequences, the functionality of the implementation was validated in the test described in Figure 6 below.

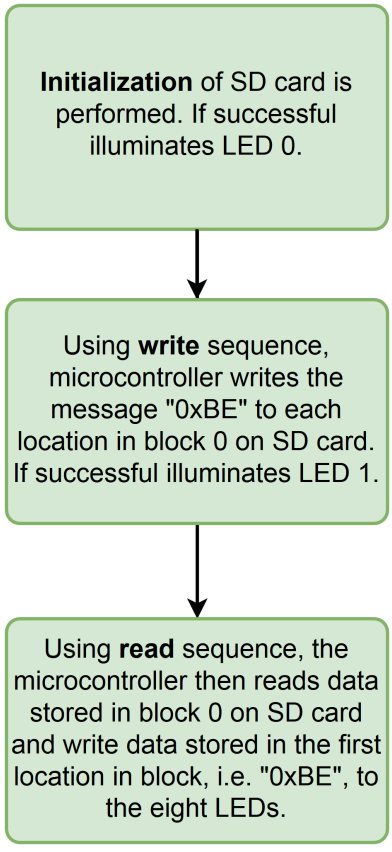

Figure 6: Test operation sequence

3.2.2 FAT interfacing

As previously mentioned, in its raw hardware structure, an SD card is divided into a sequence of 512 byte blocks. As such, files in excess of 512 bytes in size must be stored across multiple blocks. Predictably, this makes accessing and storing files a complex task. The complexity of which exponentially increase as more files are added.

To mitigate this, a file formatting system is employed. A commonly implemented file system architecture in flash memory storage devices is the File Allocation Table (FAT) [21] architecture. As previously established, SD card utilised in digital cameras are typically formatted in the FAT32 variant. FAT32 is typically utilised as it can support individual files that are up to 2 GB in size and storage sizes of up to 32 GB.

It was decided that, due to the time requirement of writing a software implementation to enable interaction with a FAT file system, a freely available software library would instead be utilised.

The most popular software package for implementing FAT interfacing in small embedded devices with limited memory is a software module known "PetitFs" [22].
This software, developed by a user known as "elmchan", provides a device independent application interface for interacting with a FAT file system. A summary of the interfaces functionality can be found in Table 2 below.

Function	Description
pf_mount	Mount a volume
pf_open	Open a file
pf_read	Read file
pf_write	Write file
pf_lseek	Move read/write pointer
pf_opendir	Open a directory
pf_readdir	Read a directory item

Table 2: Petit FatFS application interface

As the application interface is device independent, low level drivers for the specific platform must still be implemented.
However, these were simply the SD drivers that had been previously developed and as a result were simply ported into the software module.

4 Graphics display interfacing

The principle investigation point for this aspect of the project was the development of VGA driver.
Similarly to the "Memory storage interfacing", this section has been sub divided into "Research" and "Design and Methodology".

4.1 Research

4.1.1 VGA standard

The Video Graphics Array (VGA) standard is a raster picture display standard developed by IBM in 1987. The original standard supported up to 16 colours with a 640 x 480 screen resolution at a screen refresh rate of 60 Hz [9]. The standard utilizes progressive scanning to display each frame on screen. The frequency at which each scan line is refreshed or the vertical refresh frequency is defined as 31.47 KHz.
Physically, the protocol utilizes a 15 pin D-subminiature connector, shown in Figure 7 below.

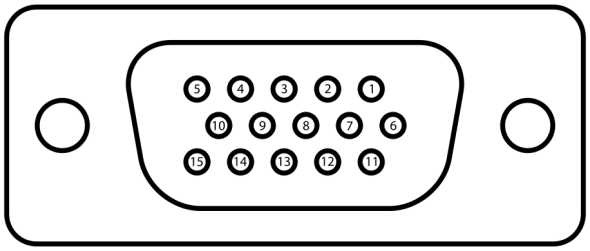

Figure 7: DE-15 VGA connector [23]

A description of the functionality of each pin can be seen below in Table 3 below.

Pin Number	Pin Name	Description
1	RED	Red video
2	GREEN	Green video
3	BLUE	Blue video
4	ID2/RES	formerly Monitor ID bit 2, reserved since E-DDC
5	GND	Ground (HSync)
6	RED RTN	Red return
7	GREEN RTN	Green return
8	BLUE RTN	Blue return
9	KEY/PWR	formerly key, now +5 V DC, powers EDID EEPROM chip on some monitors
10	GND	Ground (VSync, DDC)
11	ID0/RES	Reserved, unconnected
12	ID1/SDA	Reserved, unconnected
13	HSync	Horizontal sync, +5 V TTL, polarity negative
14	VSync	Vertical sync, +5 V TTL, polarity negative
15	ID3/SCL	formerly Monitor ID bit 3, I2C clock since DDC2

Table 3: VGA pin connections [24]

In the practical implementation of the interface, only five pins require active signals. These are the three analogue colour pins; red, green and blue, and the two digital pins; horizontal sync

and vertical sync. The other DDC and ID pins are utilised by the various digital protocols required for direct communication with the monitor. However, these are not required to output image data, and therefore, can be left unconnected.

In operation, the digital sync pins are used in the synchronisation of the video output on screen. The vertical sync, "active low", pulse is used to start a new frame and points to the start of the first scan line. The horizontal sync, "active low", pulse is used to point to the start of each new scan line.
The analogue pins, red, green and blue, are used to colour each pixel on screen. Any colour permutation can be represented by varying the intensity of each colour component in this palette. The intensity is varied by supplying an analogue voltage between 0 V and 0.7 V. The higher the voltage level, the greater the intensity of the colour.

4.1.2 Video signal generation

The timing diagram, Figure 8 below, describes the signal pulses required to generate a valid VGA signal.

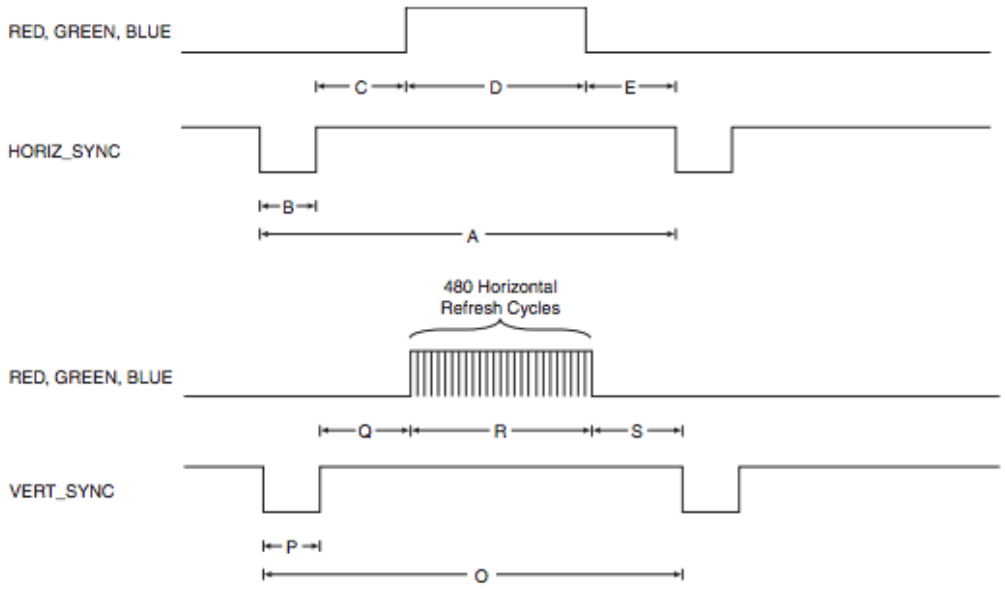

Figure 8: VGA waveform timing diagram [25]

As shown in the timing diagrams, each frame consists of various stages. The overlapping of the various stages can be clearly observed in Figure 9 below.

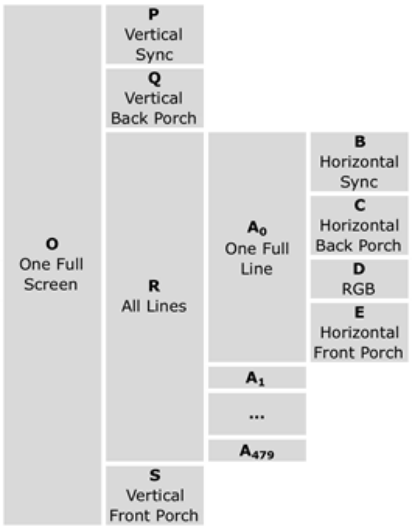

Figure 9: Summary of stages and stage overlap in the VGA standard [25]

Table 4 below specifies the precise timing values required for each of the stages considered in Figure 8 and Figure 9.

Stage	Parameter	Duration	Clock cycles	Lines
One Full Line	A	31.78 uS	800	-
Horizontal Sync	B	3.81 uS	96	-
Horizontal Back Porch	C	1.91 uS	48	-
RGB	D	25.42 uS	640	-
Horizontal Front Porch	E	0.64 uS	16	-
One Full Screen	O	16.68 mS	-	525
Vertical Sync	P	0.064 mS	-	2
Vertical Back Porch	Q	1.05 mS	-	33
All Lines	R	15.25 mS	-	480
Vertical Front Porch	S	0.32 mS	-	10

Table 4: Timing of stages within VGA standard [26]

As the VGA protocol utilizes progressive scanning, each scan line is refreshed top to bottom, pixel by pixel left to right. The frequency at which the pointer moves to the next pixel is known as the pixel clock frequency. The VGA standard defines the pixel clock frequency as 25.175 MHz. Based upon this frequency, the standard also defines the number of clock cycles (pixels) or scan lines per stage. As a result, these values can also be found in Table 4 above.

A description of the fundamental sequence of operations required to enable a valid VGA signal for a single frame is as follows [26]:

- *Vertical sync*, 2 lines. The horizontal and vertical sync are driven active low. The RGB pins are also driven low 0 V. At the end of the two lines horizontal and vertical sync are driven high 5 V.
- *Vertical back porch*, 33 lines. The vertical sync pin and RGB pins remain in there previous state. However, at the beginning of each new line, the horizontal sync pulse will be driven active low for 96 clock cycles before being driven high.
- *Visible line output*, 480 lines. Where each line consists of the following stages:

  - *Horizontal sync*, 96 pixels. The horizontal sync is driven active low. The vertical sync pin and RGB pins remain in their previous state. The horizontal sync is then driven high at the end of the stage.
  - *Horizontal back porch*, 48 pixels. All pins remain in there previous state.
  - *RGB output*, 640 pixels. This is the screen output region, where each of the pixels are visible on screen. As previously mentioned, the colour of each pixel is represented by varying the intensity of each component in the colour palette. At the end of the stage the RGB pins must be driven low 0 V.
  - *Horizontal front porch*, 16 pixels. All pins remain in their previous state.
- *Vertical front porch*, 10 lines. The vertical sync pin and RGB pins remain in there previous state. However, at the beginning of each new line, the horizontal sync pulse will be driven active low for 96 clock cycles before being driven high.

In implementation, the validity of a VGA output is unaffected by the order in which the stages of protocol occur.
Also, although the VGA standard does not specify it, a horizontal sync pulse is required on every line. Although not clearly indicated in Figure 8 and Figure 9, it is described in the sequence of operations above.
In practice it was found that without this, on some external monitors the video output on screen was unstable, but for the majority of monitors, no output would be displayed.

4.2 Design and Methodology

4.2.1 Timing adjustments

With a two stage single-level pipeline design and a maximum operational clock frequency of 20 MHz, even if the microcontroller could push a pixel out on every clock cycle, this would be more than 25% slower than the pixel clock frequency defined by the VGA standard.
To mitigate this, initially an overclock utilising a 25.175 MHz crystal oscillator was attempted. In practice the microcontroller appeared to remain stable, however in the interest of remaining within the manufacturers specified tolerances, it was decided that operational frequency must be limited to 20 MHz.

An alternative solution would therefore be required. Fortunately it was found that it was possible to produce valid VGA signals using devices with a lower pixel frequency than as specified by the VGA standard [28].
This had been realized by ensuring that the timings of the protocol were met by reducing the number of clock cycles per line accordingly. This meant that, although the pixel clock frequency was reduced, the vertical refresh frequency remained the same. As a result of this, the screen refresh rate of 60 Hz could be maintained.

As the vertical refresh rate and screen refresh rate remained constant, the number of lines per frame also remained the same. Consequently, the only signal generation characteristic that required adjustment was the number of pixels per stage.

As the number of pixels per stage was proportionate to the pixel clock frequency, the new number of pixels per stage could be calculated by multiplying the ratio, calculated in Equation

3, by the number of pixels for each stage of the VGA standard.

$$R = \frac{\text{Operating frequency of microcontroller}}{\text{Pixel clock frequency of the VGA standard}} \tag{1}$$

$$R = \frac{20MHz}{25.175Mhz} \tag{2}$$

$$R = 0.7944.. \tag{3}$$

Table 5 below shows the adjusted timings and pixels per stage for implementation. Although, the actual number of pixels required for each stage was a floating point value, the number was simply rounded up or down to the nearest whole number.

Stage	Parameter	Duration	Clock cycles	Lines
One Full Line	A	31.75 uS	635	-
Horizontal Sync	B	3.80 uS	76	-
Horizontal Back Porch	C	1.90 uS	38	-
RGB	D	25.40 uS	508	-
Horizontal Front Porch	E	0.65 uS	16	-
One Full Screen	O	16.67 mS	-	525
Vertical Sync	P	0.064 mS	-	2
Vertical Back Porch	Q	1.05 mS	-	33
All Lines	R	15.24 mS	-	480
Vertical Front Porch	S	0.32 mS	-	10

Table 5: Adjusted Timing of stages within VGA standard

As a result of this, although timing precision of the stages was reduced, in practice it proved negligible and remained sufficiently precise for external monitors to recognise the signal.

4.2.2 Video frame buffer

The next consideration of the VGA signal generation was the video frame buffer, the memory required to store the image data required for each frame.
As the microcontroller addresses byte sized memory locations, to simplify the implementation it was assumed that each colour pixel on screen would be represented by a single byte of memory. Therefore, knowing that the device was capable of outputting a maximum pixel screen resolution of 508 x 480, 243.84 KB of memory would be required to store the image data required for a single frame.

However, in order to update the colour of each visible pixel on screen, a minimum of one extra clock cycle between pixel output would be required to increment the memory address. As a result, the maximum feasible screen resolution would be 254 x 480 resulting in new frame buffer size of 121.92 KB. However, even with the reduced memory requirements, with the microcontrollers very limited on-board memory, it was apparent that an external memory solution would be required.

The external memory would be not only have to provide a large enough memory buffer, but also to have a small enough latency so as to not affect the fidelity of the image data on screen.

This meant the total propagation delay would need to be less than the 100 ns time period of the pixel clock frequency during the visible output stage. However, to ensure there would be no issues, only memory devices with a propagation delay in the region of 10-20 ns were considered.

Another requirement of the memory device was that it would need to be interfaced via a parallel rather than a serial interface. This would enable the microcontroller to output an 8-bit external memory address on one of its 8-bit ports in a single operation. A requirement necessary to ensure each pixel on screen could be updated as quickly as possible. In contrast, a serial interface would require multiple control signals, increasing the time required to output each pixel and therefore reducing the maximum horizontal resolution.

To further add to these requirements, the memory device would have to be of a through hole mounting type rather than surface mount. This was so as to minimize the difficulty in prototyping and implementation.

The only available memory chips that could be found meeting this criteria were asynchronous static RAM modules. The largest capacity of which was 32 KB. As such, the IDT IDT71256SA [27], a 32 K x 8-bit SRAM chip with a 15 ns read access time, was selected.

With a 32,768 byte memory buffer, assuming each pixel was represented by a single byte, the maximum attainable resolution would be 256 x 128.
Though, to optimize to the operational characteristics of the VGA signal generation, a resolution of 254 x 120 was selected. This enabled the maximum horizontal resolution to be employed, whilst ensuring the vertical resolution remained a factor of the number of visible lines on screen, reducing the complexity of the software implementation.

4.2.3 Visible pixel output

As previously considered, to change the colour of an individual pixel displayed on screen, there was a requirement for an analogue voltage between 0 V to 0.7 V to be applied on each of the RGB pins. By dividing this voltage range into a number of voltage steps, a maximum colour resolution can be introduced.
As such, having established in the video frame buffer that each pixel would be represented by a single byte in memory, a maximum of 256 different colour permutations could be represented.
In order to convert an 8-bit binary value into an analogue voltage, a digital to analogue (DAC) was also required.
As the Atmega does not feature an onboard DAC, an external DAC device was required. Though there were many possible options, it was decided that fastest and simplest method would be to create a passive resistor network. The passive R2R resistor network design selected, based on similar implementations, is illustrated in Figure 10 below.

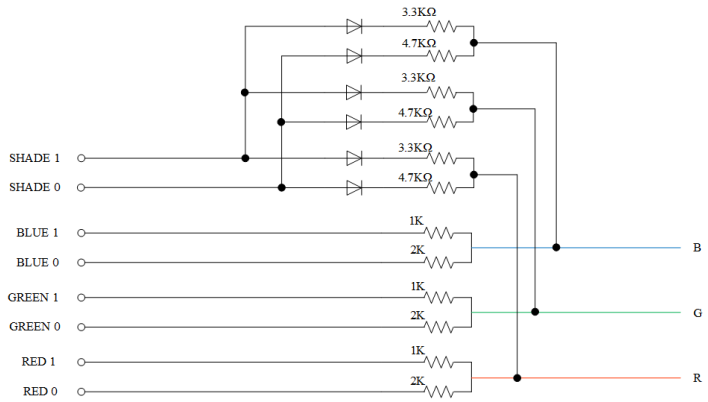

Figure 10: Passive resistor network DAC design [28]

Although in practice 8 bit RGB colour typically utilises "RRGGGBBB" it was decided that for

this design implementation a more colour accurate reproduction would be attempted through a more balanced "RRGGBBSS" colour space.
As each colour had two bits, a total of 4 different shades of red, green and blue could be represented. Therefore, utilising all 6 colour bits, a total of 64 colours could be represented. And, as there were also 2 shade bits, a further four intensity variations of those 64 colours could be represented. This therefore enabled a 256 colour palette.
The diodes were necessary in the design so that the individual 2 bit DACs did not feedback into each other, which otherwise would have resulted in strange colour combinations.

The selection of the resistors was also an important consideration, as it needed to be ensured that the output voltage to the RGB pins was not greater than 0.7 V. As the VGA RGB pins also feature 75 ohm terminating resistors, this was also accounted for in the resistor selection stage.
Through use of basic electronic theory, it was calculated that in this design implementation, the maximum output voltage on any one of the RGB pins would be 0.708V .

4.2.4 Hardware and software driver

To facilitate the video signal generation, both a hardware and software driver implementation was required. Therefore, to validate the various attributes relating to the signal generation, a number of prototype designs were created.

The first prototype implementation was created to verify the video signal generation driver and the passive DAC colour selector. The circuit schematic can be found in Figure 11 below.

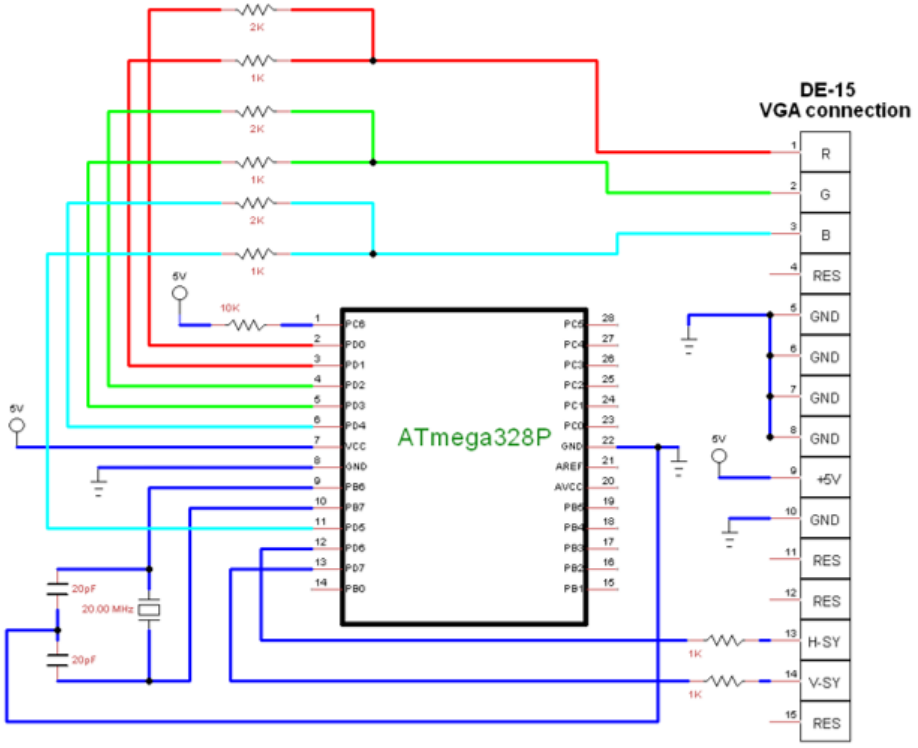

Figure 11: Atmega328P VGA driver circuit

As shown in the figure above, the Atmega328P utilised an external crystal to operate at the required 20 MHz clock frequency. Also, the DAC design was simplified to limit complexity in implementation. This resulted in a 16 colour palette.

In the physical prototyping, the circuit shown above was implemented on a solderless breadboard. Operating at up to 20 Mhz, it was initially thought that parasitic inductances and capacitances would have a significant effect on the circuit, however in practice this appeared not to have an effect. Also, to ensure the power to the IC remained stable, 0.1 uF decoupling capacitors were placed near to the microcontrollers power terminals.

In terms of the overall operation, the prototype performed the following functionality:

- Initially the microcontroller wrote four 6-bit colour values to four internal memory locations.
- The microcontroller then performed the video signal generation. And, during the RGB output pixel stage, the microcontroller read the colour data back from these four memory collections and outputted the value to PORTD connected to the DAC.
- The video signal generation then repeated indefinitely and the final result was a four colour image on screen.

The output generated during this prototyping stage can be found in [Appendix 4].

Due to the level of timing precision required for VGA signal generation, it was apparent that a granular level of software control was required. This meant any higher level languages such as C or C++ could not be utilized. The most appropriate approach to the software development was therefore to employ assembly code. This enabled the clock cycles in each subroutine to be calculated and then precisely tailored to the timing requirements of the video signal generation.

There were two possible approaches to the software implementation:

- Utilising a timer based interrupt service routine (ISR)
- Utilising a precisely timed assembly loop routine

Initially the timer based ISR based solution was attempted, this approach would enable additional functionality to be easily implemented in the code because the timer would ensure the video signal generation timings were precisely maintained.
However, this proved particularly difficult to implement and severely reduced the readability of the code. As such, the assembly routine was selected instead.

This approach essentially consisted of embedding the various precisely timed sub routines for each stage of the video signal generation within an infinite software loop.
Additional software functionality could then be embedded within stages of the signal generation, however it would need to be ensured that this did not affect the timings of the video signal generation.

The initial prototype design proved very challenging to get functioning correctly and added an almost two month delay to progress in the project. It was thought that this was as a result of lack of precision in the software implementation, and therefore, constant prototyping with slightly different VGA signal timings was carried out during this period. An overclock, as previously mentioned, was even attempted.
The issue however, as previously mentioned in the video signal generation theory, was that all the information regarding VGA signal generation neglected to clearly specify that a horizontal sync pulse was required on every line.
As such, during prototyping, even with attempted diagnosis using a logic analyser, the fault

could not be found.
A solution wasn't realised until an obscure article [31], outlining the exact parameters of the VGA protocol, was found.

Following this, a second physical prototype was created to verify the operational characteristics of the SRAM module. Primarily, this was to ensure that the video signal generation would be unaffected by the latency caused by adding the memory device. But also, it was to investigate how pixel data could be written to and read from the external memory module. The circuit schematic can be found in Figure 12 below.

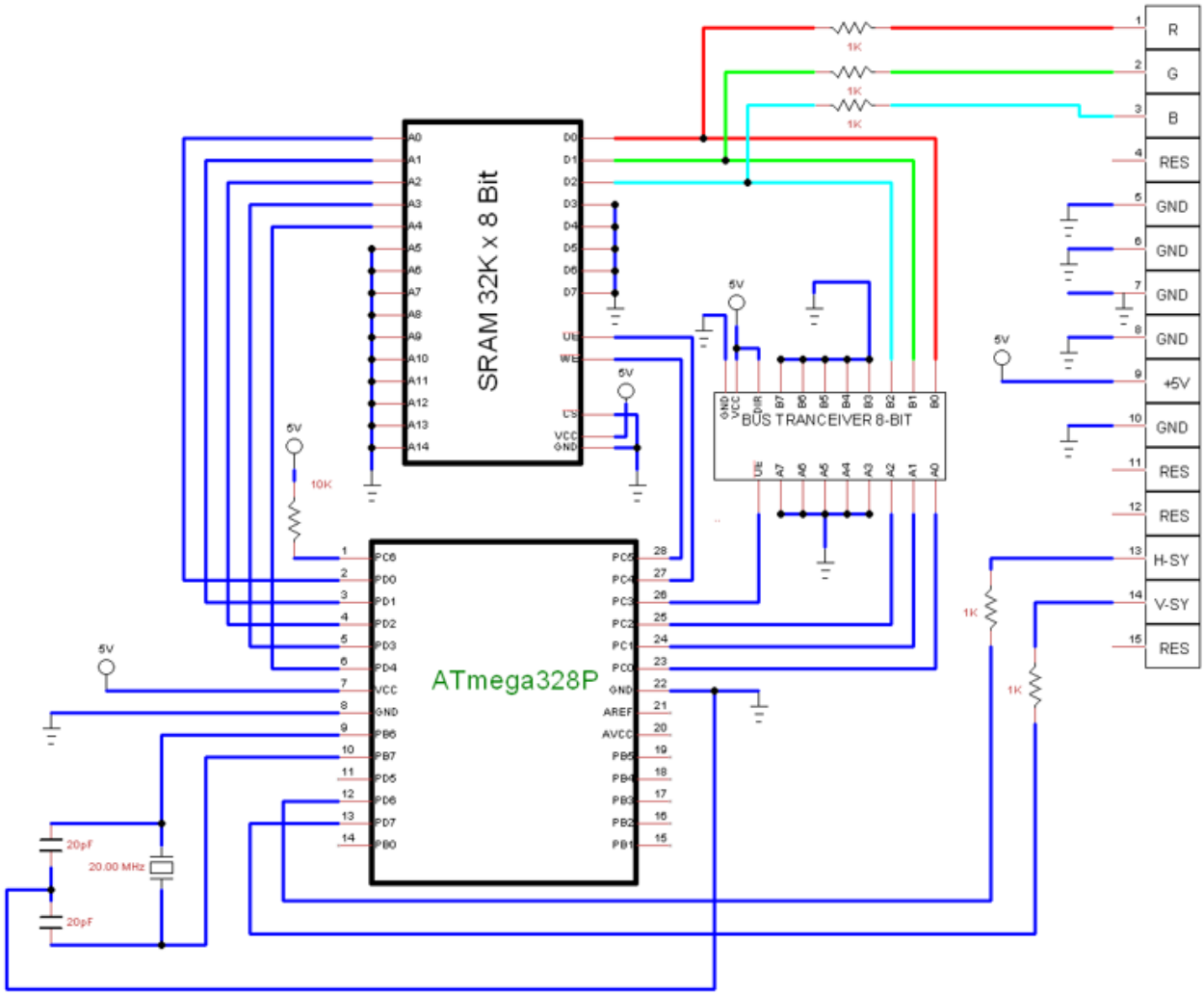

Figure 12: Atmega328P VGA driver circuit using SRAM

As can be seen in the diagram, the Atmega328P did not have a sufficient number of pins to connect all the address pins, data pins, write enable, output enable and chip select pins. As such only a 4 bit address and 3 bit data input was connected.
Also featured in the circuit schematic is an 8-bit tristate buffer, further detail regarding tristate buffer logic can be found in [Appendix 3]. The tristate buffer implemented during prototyping was a SN74ABT245BNE4 [29], this was due to its low cost and 3.5 ns propagation delay.

During prototyping it was found that if the microcontrollers pins were not disconnected during the RGB output stage, the pixels on screen would become distorted due to interference on the line.
As such, the tristate buffer enabled the microcontrollers pins to be effectively disconnected while data was being output from the SRAM module on screen.

Although the Atmega microcontroller does feature tristate behaviour on some of the I/O ports, it was found during testing, interference remained on the line distorting the image output.

The functionality of the prototype is described below:

- Initially, the microcontroller set the "output enable" pin on the tristate buffer and the "chip select" on the SRAM module active low.
- The microcontroller then iterated through addresses 0 to 31 on the SRAM module and at each address the colour green (0x02) was be written to the three bit data input bus.
- With the data written, the "output enable" pin on the tristate buffer logic and "chip select" on the SRAM module were then set high by the microcontroller.
- The microcontroller then generated the VGA video signal. However, prior to the RGB pixel output stage, the "chip select" on the SRAM module was driven active low. Then, during the RGB output pixel stage, the microcontroller iterated through each of the addresess in SRAM outputting the data at each address on screen.
- At the end of RGB stage the "chip select" pin was then driven high and the process repeated.

The software driver remained very similar to that of the previous prototype, however instead of reading from internal memory during the RGB pixel output stage, pixel data was read from the SRAM module.
In the physical prototyping, this implementation proved successful though, some minor distortions could be seen on screen. The exact cause was unclear, however it appeared to be as a result of some interference.

It was anticipated that to scale this solution to utilise all the memory locations required for the 254 x 120 resolution, a software compatible Atmega324A microcontroller would be implemented. This 40 pin device had sufficient I/O pins to to provide both a 15 bit address and an 8 bit data input. For efficient memory access during the visible line output stage, the microcontroller considered each pixel address as a lower and upper address.

Where, the lower 7 bits represented a row address between 0 and 119, and, the upper 8 bits represented a column address between 0 and 253.

During the horizontal back porch of the visible line output, the row address would be incremented by one every four lines. This would effectively reduce the vertical resolution from 480 to 120.
And as the row address was updated during the horizontal back porch, only the column address would need to be incremented and outputted during the RGB output stage. So, for each line, the column address would be incremented from 0 to 253.

Furthermore, as the Atmega microcontroller can update the state of an 8 pin port in a single clock cycle, the column address could be updated in a single cycle. Effectively enabling each pixel on screen to be updated every 2 clock cycles.

5 System architecture

Having considered the main hardware sub-systems in the project, the principle investigation point of this section was exploring how these are sub-systems were integrated to form the final system implementation.
In contrast to the previous to sections, the system architecture design and implementation did not feature a definitive research section, rather research was carried out during the design and development process.

5.1 Double buffering

Although during the "Hardware driver" sub section of "Graphics Display Interfacing" it was verified that data could be written to a video frame buffer and then outputted on screen, this process did not occur on a real time basis.
In practice, prior to the video signal generation, the image data was written to the memory buffer, then, during the video signal generation, the buffered image data was outputted on screen.

The main issue with this design implementation was that, before the memory buffer has been filled with the image data, nothing could be outputted on screen.
So, while the new frame is updated, the screen would "blank" for a period of time. The duration of which varied depending on the time required to update each new frame.

To mitigate this, a solution enabling a new frame be updated while the other frame was being outputted was required.
As it turns out, this is a major issue faced in computer graphics and as a result, a solution was found in a research paper titled "Simulink Model for Double Buffering" [30]. The paper described a practice known as "double buffering", whereby the image data for one frame can be output on screen whilst the other frame is being updated.
This is facilitated through the use of two memory buffers, each with a sufficient memory size to store a single frame.
So, during operation, when the second frame has finished being drawn, the video frame buffers are swapped, allowing the new frame to be outputted on-screen.
This process then repeats, enabling each frame to be smoothly updated on screen. A diagram illustrating this functionality can be seen in Figure 13 below.

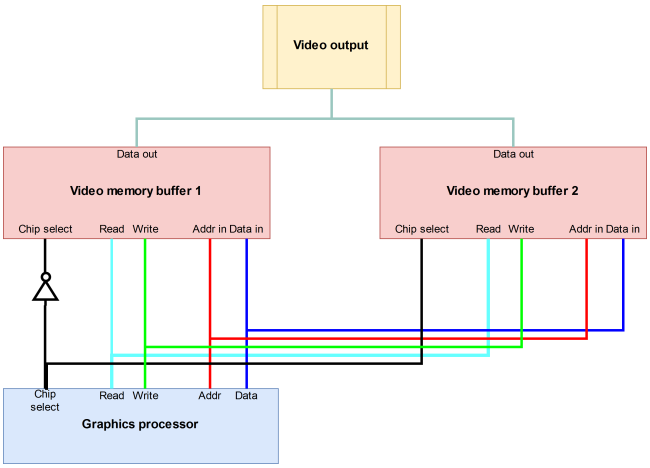

Figure 13: An implementation of a double buffered graphics solution

In functionality, the inverter logic component enables the graphics processor to select between either of the memory buffers
Thus, utilising the "Read" and "Write" pins, the graphics processor can write data to a buffer during one cycle and then read data from the other buffer on the next cycle.
Although in this example diagram, the reading and writing of a new frame cannot occur concurrently. In practice, graphics processors utilise highly specialised parallel compute units enabling this precise behaviour.

Therefore, it was decided that to implement this double buffer mechanism in the project and maintain the highest screen resolution, another 32 KB external SRAM memory chip would be utilised to act as the secondary memory buffer.

Although this would not allow frame draw and frame output to occur simultaneously, the operational characteristics of the VGA standard would enable the microcontroller to draw a new frame during the inactive porch regions of the video signal generation.
Then, when the new frame was completed, the video memory buffers could be swapped and the new frame output on screen.

However, due to the processor intensive task of generating the video signal, there would leave little processing time each frame refresh to update the other memory buffer. As such, it would take a rather long period of time to update each new frame.
It was clear that this would have a significantly negative effect on the performance of the system.

5.2 Multiprocessor design

To mitigate the increased work load requirements of the system, and, inspired by the parallel processing capabilities of modern graphics processing units, it was decided that a multiprocessor design would be implemented. This would be facilitated through the addition of a second AVR microcontroller, uC2.
This would enable the time critical video signal generation to be managed by uC1, while all the non time critical tasks could be offloaded to uC2. Furthermore, while all the software developed for uC1 needed to be implemented in assembly, object orientated C++ software could be developed for uC2, significantly reducing the time required for software development.

This multiprocessor design would also enable the double buffered memory mechanism to occur in parallel, enabling the frame draw and video signal generation to happen simultaneously. To achieve this, uC2 would draw the frame in one video frame buffer while uC1 outputted the data from the other frame buffer on screen.
Then, when uC2 had finished drawing its frame, the memory buffers would be swapped, and the process could continue.

As the length of time required to draw a new frame would vary, it was decided that uC1 would also be delegated the responsibility of managing the memory buffer swap.
This would enable uC1 to precisely control when the memory buffer swapped, thus enabling, the time critical requirements of the video signal generation to remain unaffected.

However, in order to implement this functionality, a communication interface between the microcontrollers was required.
This communication interface was required to facilitate the following functionality:

- uC2 would be able indicate to uC1 when the frame draw had completed and that the memory buffers could be swapped.

- uC1 would be able to indicate to uC2 that the memory buffer swap had occurred and that it could continue drawing the next frame.

Figure 14 below illustrates the overall multiprocessor design implementation.

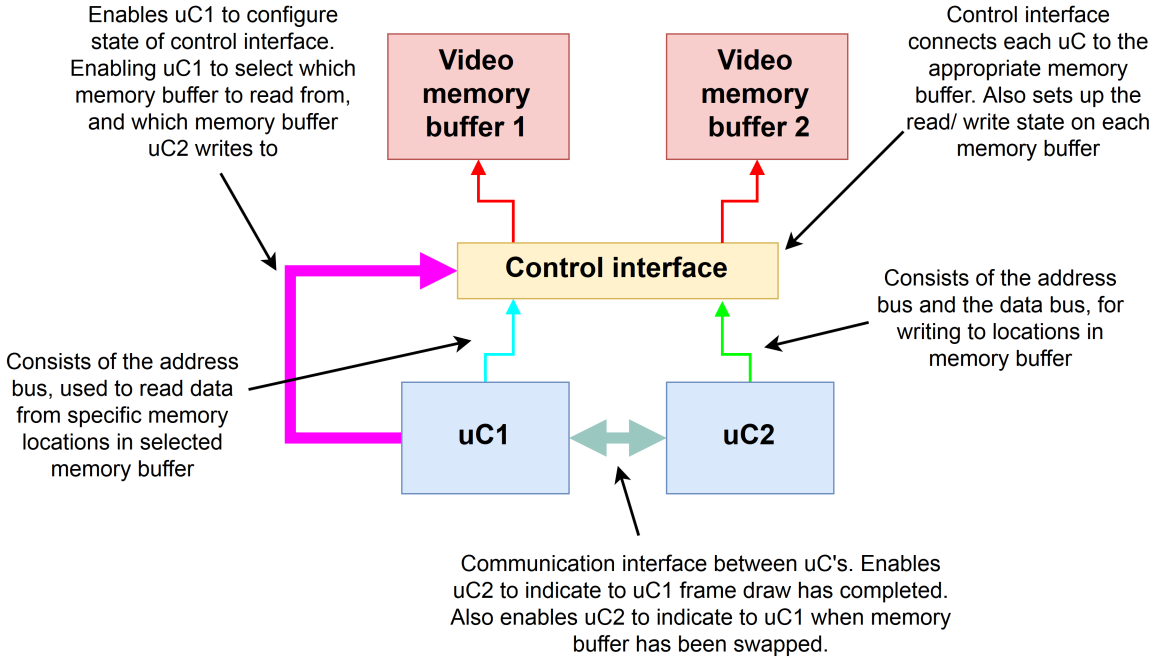

Figure 14: Multiprocessor double buffered memory configuration

As indicated in the diagram, a control interface would also be required to facilitate the memory buffer swap. Further details regarding this implementation can be found in the next sub section, "Signal flow control".
As the serial interfacing module on uC2 was already being used for SD card interfacing and only limited communication was required, a simple two line communication approach was implemented.
One line would enable uC2 to communicate with uC1 and the other would enable uC1 to communicate with uC2. Figure 15 below illustrates these two connections.

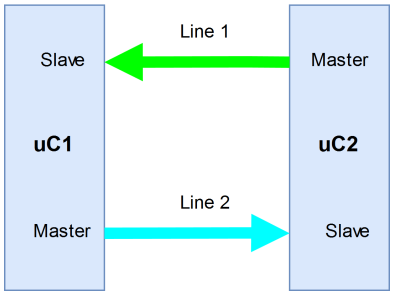

Figure 15: Two line master-slave communication interface

In their default inactive states, both lines are pulled high 5V. To indicate to uC1 the frame draw has completed, uC2 will pull line 1 down to active low.
Then, during the inactive vertical front porch region of the VGA signal generation, uC1 will check the input state of its pin connected to line 1.
If it detects the line is pulled active low it will then swap the memory buffers and pull line 2 down to active low for a short period.

This will indicate to uC1 that the memory buffer swap has occurred and that it can set line 1 back to its inactive state and continue drawing the next frame. Flow diagrams illustrating the parallel behaviour of both uC1 and uC2 can be found in Figure 16 below.

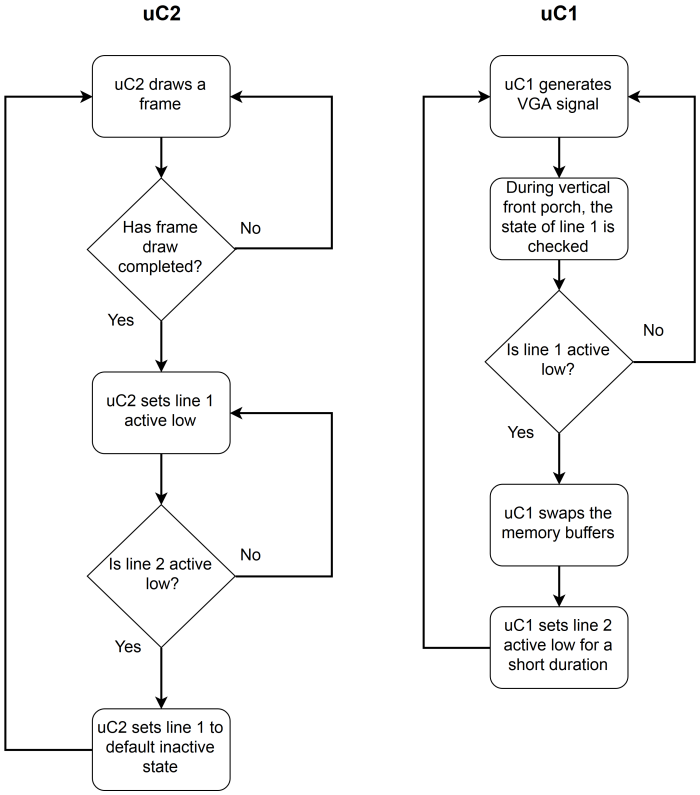

Figure 16: Parallel behaviour of uC1 and uC2 in the multiprocessor system

5.3 Signal flow control

As previously mentioned, a control system was required to enable uC1 to swap the connections on the memory buffers. The control system would need to simultaneously facilitate the following functionality:

- Enable uC1 to select which memory buffer to interact with.
- Enable uC1 to read from the selected memory buffer and output the data at specific memory locations to the DAC connected to the video output.
- Physically disconnect the other memory buffer from the DAC output, and, enable uC2 to write data to specific memory locations in the buffer.

As the pin connections on uC1 and uC2 required a physical re-routing, a hardware solution was required.
Also, as the memory buffer swap would occur during the vertical front porch region of the video signal generation, the actual connection re-routing needed to have a minimal time latency. This was so as to not affect the video signal generation timings.

It was decided that this would be employed through the use of the 8 bit SN74ABT245BNE4 tristate buffer gates utilised during the prototyping stage of the graphics display interfacing.

The overall system architecture, featuring these 8 bit tristate buffers for the signal flow control, can be found in Figure 17 below.

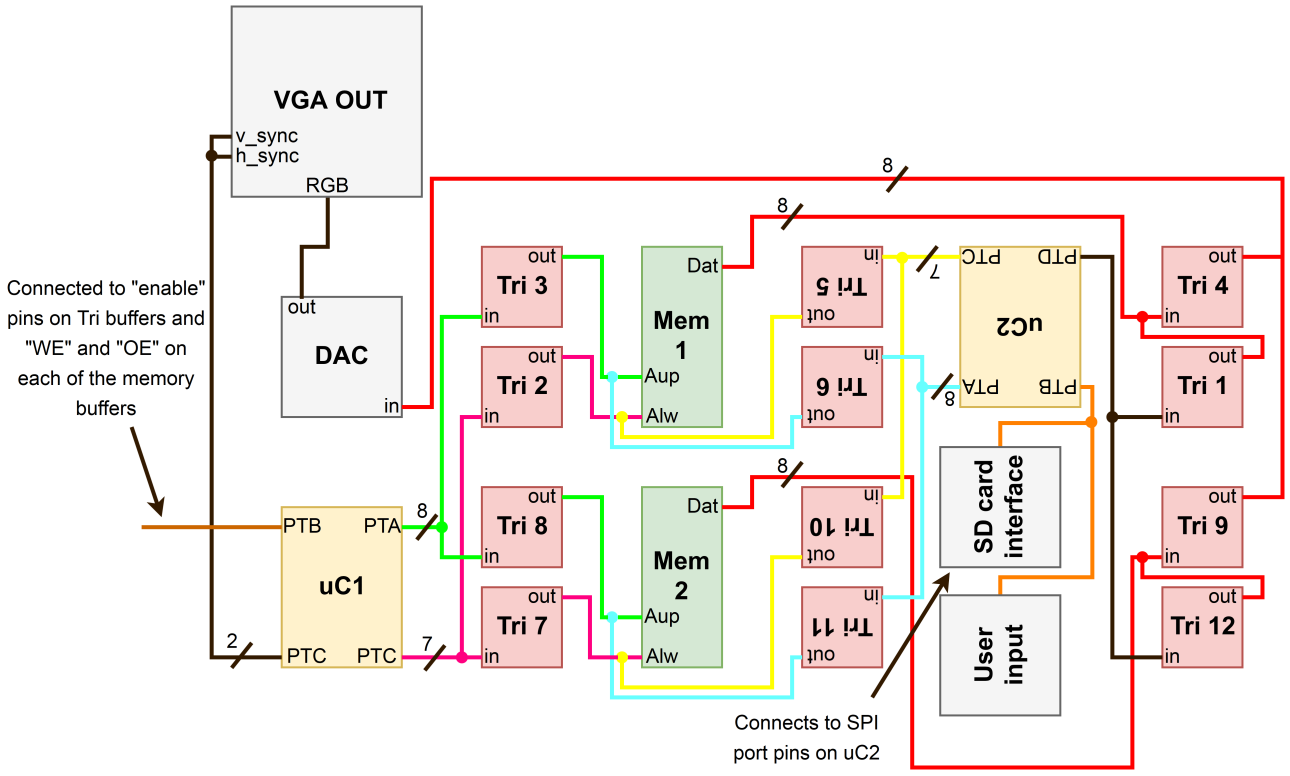

Figure 17: System diagram featuring tristate buffers

The bullet points below summarise the functionality for each of the tristate buffers.

- **Tri 1**: Connects all 8 pins of uC2 PORTD to 8 bit data input on memory buffer 1.
- **Tri 2**: Connects 7 pins of uC1 PORTC to the lower 7 bits of the 15 bit address input on memory buffer 1.
- **Tri 3**: Connects all 8 pins of uC1 PORTA to the upper 8 pins of the 15 bit address input on memory buffer 1.
- **Tri 4**: Connects the 8 bit data output from memory buffer 1 to the input of the 8 bit DAC connected to the physical VGA connector.
- **Tri 5**: Connects 7 pins of uC2 PORTC to the lower 7 bits of the 15 bit address input on memory buffer 1.
- **Tri 6**: Connects all 8 pins of uC2 PORTA to the upper 8 bits of the 15 bit address input on memory buffer 1.
- **Tri 7**: Connects 7 pins of uC1 PORTC to the lower 7 bits of the 15 bit address input on memory buffer 2.
- **Tri 8**: Connects all 8 pins of uC1 PORTA to the upper 8 bits of the 15 bit address input on memory buffer 2.
- **Tri 9**: Connects the 8 bit data output from memory buffer 2 to the input of the 8 bit DAC connected to the physical VGA connector.
- **Tri 10**: Connects 7 pins of uC2 PORTC to the lower 7 bits of the 15 bit address input on memory buffer 2.
- **Tri 11**: Connects all 8 pins of uC2 PORTA to the upper 8 bits of the 15 bit address input on memory buffer 2.

- **Tri 12**: Connects all 8 pins of uC2 PORTD to 8 bit data input on memory buffer 2.

For brevity, the enable pins for the tristate buffers and the write enable and output enable for the memory buffers are not illustrated in Figure 17.

Tables 6 and 7 below illustrate the required states of the tristate buffers to enable uC1 to read from memory buffer 1 or memory buffer 2 whilst enabling uC2 to write to the other memory buffer.

Connection	State
Tri 1 En	High
Tri 2 En	Active low
Tri 3 En	Active low
Tri 4 En	Active low (Toggle)
Tri 5 En	High
Tri 6 En	High
Tri 7 En	High
Tri 8 En	High
Tri 9 En	High (Toggle)
Tri 10 En	Active low
Tri 11 En	Active low
Tri 12 En	Active low
Mem 1 WE	High
Mem 1 OE	Active low
Mem 2 WE	Active low
Mem 2 OE	High

Table 6: uC1 reading from memory buffer 1 and uC2 writing to memory buffer 2

Connection	State
Tri 1 En	Active low
Tri 2 En	High
Tri 3 En	High
Tri 4 En	High (Toggle)
Tri 5 En	Active low
Tri 6 En	Active low
Tri 7 En	Active low
Tri 8 En	Active low
Tri 9 En	Active low (Toggle)
Tri 10 En	High
Tri 11 En	High
Tri 12 En	High
Mem 1 WE	Active low
Mem 1 OE	High
Mem 2 WE	High
Mem 2 OE	Active low

Table 7: uC1 reading from memory buffer 2 and uC2 writing to memory buffer 1

As clearly shown in the table, the states of the tristate buffers and memory buffers are the complement of each other.
This therefore enabled control through a single inverter connected to a single pin on uC1. Figure 18 below illustrates this in the implementation.

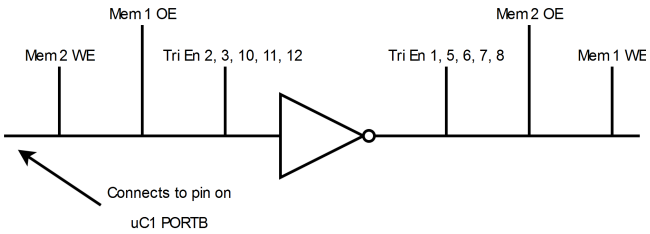

Figure 18: Inverter control connection

As the data output state of the memory buffer modules needed to be toggled on and off during the video signal generation, both tristate 4 and 9 required independent control.
As such, each of these pins were assigned an individual control pin on PORTB of uC1.

However, following the hardware implementation, a limitation in the design of signal flow control was noticed.
In operation, as uC1 would set and leave the "Write enable" pin enabled on the memory buffer connected to uC2. The data present on PORTD of uC2 would then be automatically written to the currently selected memory address.
And, as each address was split into a lower and upper section, uC2 would be unable to update the location simultaneously. So, as either the upper or lower section of the address had to be updated first, data would be written to this new location before the entire address had been updated.
Thus, this limitation meant that uC2 was unable to write to memory address locations in a random order.

In an attempt to mitigate this, the tristate behaviour of the PORTD pins on uC2 were utilised. In theory, this would enable the PORTD pins to be disconnected from the data input on the memory buffer.
However, it was found that interference remained present on connections meaning that random data was still being written to the locations in the memory buffer.

As a result of this limitation, it was concluded that the simplest solution would be to write to each memory location on a sequential line by line basis. Though, it was clear that this would add complications to the frame draw software implementation.

A photograph of the final circuit board implementation can be found at [Appendix 5].

6 Image processing

With the main hardware implementation established, the principle investigation point of this section was to to explore the mechanism by which Bitmap image files could parsed and outputted on screen.
In conjunction to this, the users interaction and the resulting graphical response was also considered.

6.1 Bitmap file reading

A Bitmap file is a device independent raster graphics format capable of storing two dimensional digital image files.
Structurally, a Bitmap image file consists of the following regions:

- **Bitmap header:** details the file type
- **DIB header (bitmap information header):** details specific attributes related to file including; dimensions, number of bits per pixel etc.
- **Image data:** holds the raw pixel map that provides the colour data for each pixel on screen.

At the time of writing, the FAT file system management software had not been sufficiently developed for implementation. As such, the Bitmap file could only be stored in the microcontrollers 32 KB program memory. However, there was only 12 KB of unused memory, as a portion of this memory was already occupied by program data.
Also, as a standard Bitmap image file features no image compression, with an 8 bit colour depth, the resolution of the image, excluding the header file information, was equal to the number of bytes of memory required for storage. Therefore, it was not be possible to store a Bitmap image with a resolution of 254 x 120 in the unused program memory.
As such, to both satisfy the limited memory requirements and limit the complexity of the image processing software, a Bitmap image file of resolution 127 x 60 was selected. Where, at this resolution, only 8 KB of storage was required. This resolution also enabled the image to be more easily scaled in both the horizontal and vertical axis, where, each pixel in the bitmap file would be represented by two pixels on screen in both axis.

However, as previously established, the data locations in the memory buffer could only be updated on a line by line sequential basis.
The software implementation would therefore need to first consider the pixel location in the memory buffer and then calculate which pixel this was mapped to in the Bitmap image file.
Furthermore, as the Bitmap image was stored in the secondary read only memory, the microcontroller would not be able to access the the Bitmap data directly.
As such, the required image data would first need to be copied into the microcontrollers internal R/W memory.

However, as the Bitmap image file was significantly larger than the 2 KB of internal RAM available to the AVR microcontroller, only a region of the Bitmap image would be accessible at any one time. To efficiently manage this in implementation, a 512 byte memory buffer region was set aside in the AVR's main memory.

This meant that the 8 KB Bitmap image file could be divided into 16 individual 512 byte buffers. Where, each 512 byte region of the Bitmap file could be read into the memory buffer

at any one time.
To access specific pixel data the buffer region was first calculated followed by an address offset in that buffer.

As the main limiting factor in updating a new frame on screen would be caused by the delay in accessing the Bitmap data from the secondary storage; the 512 byte buffer ensured that the Bitmap data access time was significantly improved in comparison to constant access of the secondary storage.

The final output of a 127 x 60 Bitmap image read from the devices program memory can be found in Figure 19 below.

Figure 19: 127 x 60 Apollo 11 moon landing image [32]

It is apparent from the image that the colours are incorrect, this is as a result of different colour space implementation in the image file when compared with the hardware implementation.

6.2 Text mode

To enable external interfacing from the user it was decided that a simple directional push button mechanism would be utilised. This would enable the user to navigate between and select image files.
To provide feedback to the user as to which file has been selected, a simply text based graphical user interface would be implemented.

In practice, this would configure the frame on screen as a equivalent character space, where each frame would be represented by characters. To implement this, the regions where text was not required would simply be represented by blank spaces.
This however, would not easily facilitate both graphics and characters to be displayed on screen simultaneously.

At the time of writing this is yet to be developed and implemented and as such no further development or prototyping has occurred.

7 Results and conclusion

The work carried out on the project so far has been summarised below:

- The aims, objectives and technical requirements of the project were established.
- Research and prototyping of SD cards and low-level interfacing was carried out.
- A device driver, in the form of both a hardware and C software implementation, was developed to enable interfacing with an SD card.
- Interfacing with a FAT32 file system architecture was researched.
- Research and prototyping into the VGA standard using a microcontroller implementation was carried out.
- A device driver, in the form of both a hardware and an Assembly software implementation, was developed to enable interfacing with a VGA capable monitor/television.
- An overall system architecture that integrated the various sub system implementations was developed and implemented on breadboard. This hardware and software system implementation consisted of a multiprocessor design with a double buffered memory solution and a sophisticated control network.
- A C++ software implementation enabling the device to parse and interpret Bitmap image data was developed. This also facilitated the image file to be displayed on screen through the VGA interface.
- A character based graphics mode using the VGA interface was researched.

The final physical implementation of the device can be found in Figure 20 below.

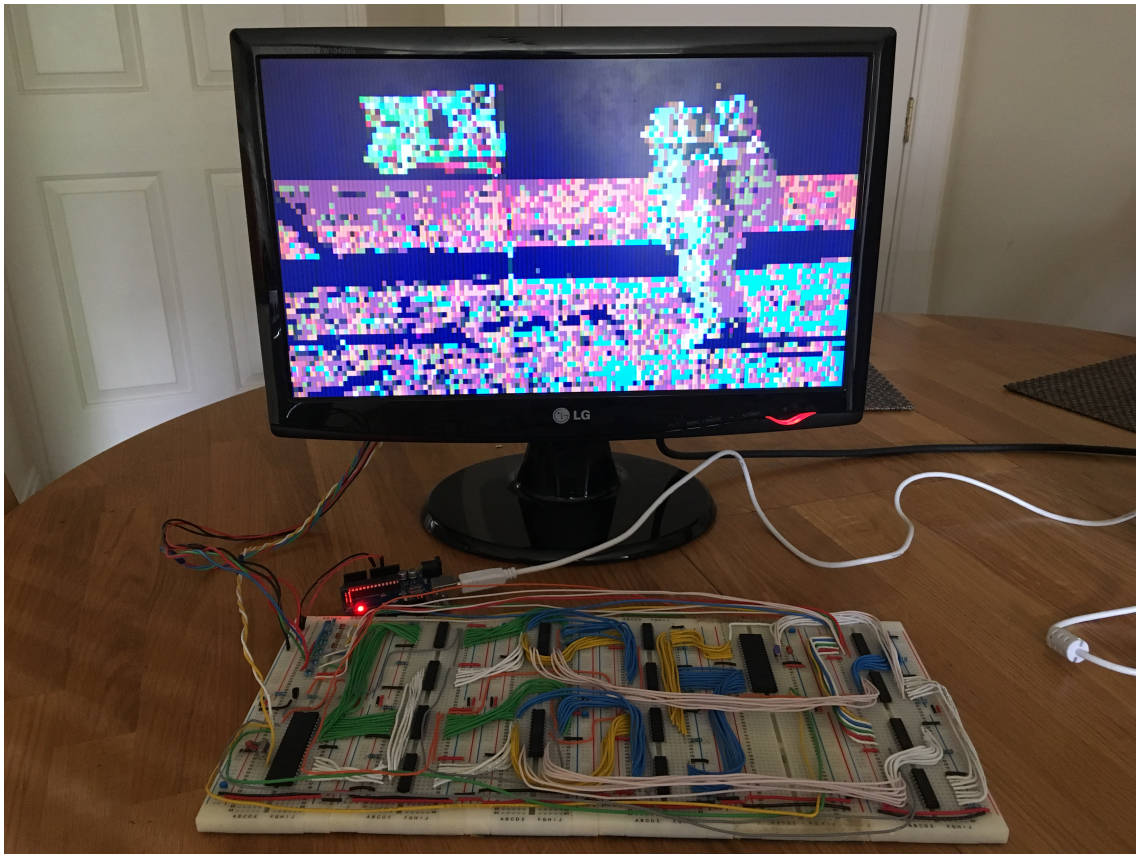

Figure 20: A photograph of the final device implementation

The devices capabilities, at the time of writing, has been summarised below:

- The device can interface with an SD card at a driver level through an SPI interface; enabling the device to read from and write to blocks within the SD cards flash memory.
- The device can generate a VGA video signal producing a frame on screen with a resolution of 254 x 120 and a 256-colour palette.
- The device can update a new frame on screen, flicker free, with the use of a double buffered multiprocessor design implementation.
- The device is capable of parsing, interpreting and scaling Bitmap image files which can then be displayed on screen through the VGA interface.

It is anticipated that for the Viva demonstration the device will also be capable of the following additional functionality:

- The device will be capable of interfacing with a FAT32 file system architecture implemented on an SD card. This will enable Bitmap image files stored on the SD card to be accessed and read.
- The device will also be capable of accepting user input through a simple push button interface, enabling the selection between and viewing of the Bitmap files on the SD card.
- The device will also feature a simple character based graphical user interface, enabling the user to receive feedback on their interactions through the push button interface.

Upon reflection of the capabilities of the final system implementation, the majority of the technical requirements have been sufficiently realised. Though, there is still further development required to satisfy all the requirements. It is anticipated that, as the most demanding aspects of the project have already been resolved, these further requirements can be realised for the Viva demonstration. Throughout the project, there has been several unanticipated delays slowing rate of progress. These stem partially from the unforeseen scope of the project, which resulted in significant complications in the final design implementation, but also from setbacks in the research and development phase.
In the projects inception, management of scope creep was attempted using a definitive specification, although complications in the final design implementation still arose. Most notably was the implementation of the double buffered memory solution, which required an elaborate control network to facilitate the desired functionality. This exponentially increased the complexity in the system architecture, which in turn drastically increased the time spent in the development phase. In hindsight, a solution that sacrificed some system performance in favour of a simpler implementation may have been a more appropriate solution.
During the research and development phase, the setback that had greatest impact on the progress of the project was the horizontal synchronisation issue previously discussed in the "Graphics display interfacing" section. The issue came as a result of a lack of clarity regarding the VGA standard. As such, this was unanticipated and in hindsight could not have been predicated or mitigated.

To summarise, though the project proved to be more of an academic exercise rather than a solution to an applicable real-world problem, the complexity of the implementation meant that a very broad range of creative electronic engineering solutions were exercised. This also highlighted the surprising capability of modern low-cost microcontrollers.

In terms of the future considerations, although not necessarily applicable to the intended purpose of the project, with minor modification the device could provide an ideal platform for

implementing high resolution animated graphics to otherwise limited microcontroller based projects.

References

[1] SD Standard Overview[Online]. Available:https://www.sdcard.org/developers/overview/ [Accessed: 26 Oct. 2017].

[2] Thomas Finch. "ELC025 Project; Interim Report". School of Electronic, Electrical and Systems Engineering., Loughborough University., Dec. 2017.

[3] Overview of JPEG[Online]. Available:https://jpeg.org/jpeg/[Accessed: 26 Oct. 2017].

[4] Paul Stoffregen. Understanding FAT32 Filesystems[Online]. Available:https://www.pjrc.com/tech/8051/ide/fat32.html[Accessed: 27 Oct. 2017].

[5] FAQ for HDMI 1.4[Online]. Available:https://www.hdmi.org/manufacturer/hdmi_1_4/hdmi_1_4_faq.aspx[Accessed: 27 Oct. 2017].

[6] Definition of: composite video[Online]. Available:https://www.pcmag.com/encyclopedia/term/40120/composite-video[Accessed: 27 Oct. 2017].

[7] Atmel. ATmega328/P Datasheet Complete [Online]. Available:http://ww1.microchip.com/downloads/en/ DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf[Accessed: 28 Oct. 2017].

[8] Atmel. ATmega324A Datasheet Complete[Online]. Available:http://ww1.microchip.com/downloads/en /DeviceDoc/ Atmel-42714-ATmega324A_Datasheet.pdf[Accessed: 15 Feb. 2018].

[9] Encyclopaedia Britannica. VGA Technology[Online]. Available:https://www.britannica.com/technology/VGA[Accessed: 29 Oct. 2017].

[10] Microsoft. Bitmap Storage[Online]. Available:https://msdn.microsoft.com/en-us/library/windows/desktop/dd183391(v=vs.85).aspx[Accessed: 03 Nov. 2017].

[11] dpreview staff. Three Giants to develop new "Secure Memory Card"[Online]. Available:https://www.dpreview.com/articles/6861681955/newmemory[Accessed: 04 Nov. 2017].

[12] Technical Committee SD Association. Simplified Specifications[Online]. Available:https://www.sdcard.org/downloads/pls /index.html[Accessed: 06 Nov. 2017].

[13] Interfacing SD Card with AVR Microcontroller[Online]. Available:https://www.engineersgarage.com/embedded/avr-microcontroller-projects/sd-card-interfacing-project[Accessed: 08 Nov. 2017].

[14] Cem Berik. Secure Digital (SD) card pinout[Online]. http://pinouts.ru/Memory/sdcard_pinout.shtml[Accessed: 09 Nov. 2017].

[15] Technical Committee SD association. *SD Specifications Part 1 Physical Layer Ver 6.0*. 2017. p 26.

[16] Technical Committee SD association. *SD Specifications Part 1 Physical Layer Ver 6.0*. 2017. p 209.

[17] Texas Instruments. Serial Peripheral Interface (SPI) User Guide[Online]. Available:http://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf[Accessed: 11 Nov. 2017]. SPRUGP2A. Mar. 2012.

[18] Corelis. SPI Tutorial[Online]. Available:https://www.corelis.com/education/tutorials/spi-tutorial/[Accessed: 14 Nov. 2017].

[19] Elm Chan. How to Use MMC/SDC. Available:http://elm-chan.org/docs/mmc/mmc_e.html[Accessed: 14 Nov. 2017].

[20] P. Ruiz-de-Clavijo, Enrique Ostúa, Manuel-J. Bellido, Jorge Juan, Julián Viejo, David Guerrero "Minimalistic SDHC-SPI hardware reader module for bootloader applications" *Microelectronics journal 67 32-37*. Jul. 2017.

[21] Bob Eager. A tutorial on the FAT file system[Online]. Available:http://www.tavi.co.uk/phobos/fat.html[Accessed: 25 Nov. 2017]. 2017.

[22] Elm Chan. Petit FAT File System Module[Online]. Available:http://elm-chan.org/fsw/ff/00index_p.html[Accessed: 25 Nov. 2017].

[23] Mobius. The Pinout of a VGA connector[Online]. Available:https://commons.wikimedia.org/wiki/File:DE15_Connector_Pinout.svg[Accessed: 08 Dec. 2017]. 2006.

[24] g4vii, Honza S, Sunil Desai. VGA pinout[Online]. Available:http://pinouts.ru/Video/VGA15_pinout.shtml[Accessed: 08 Dec. 2017].

[25] Ryan Fanelli, David Hartino., "Homemade VGA adapter"[Online]., Available:https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/raf225_dah322/raf225_dah322/index.html Dept. Elec & Comp Eng., Cornell Univ., 2012.

[26] SECONS Ltd. VGA Signal 640 x 480 @ 60 Hz Industry standard timing[Online]. Available:http://tinyvga.com/vga-timing/640x480@60Hz[Accessed: 14 Dec. 2017]. 2008.

[27] IDT. CMOS Static RAM IDT71256SA[Online]. Available:https://www.idt.com/document/dst/71256sa-datasheet[Accessed: 18 Dec. 2017]. Nov. 2014.

[28] VGA Video Generator[Online]. Available:http://www.lucidscience.com/pro-vga20video20generator-7.aspx[Accessed: 18 Dec. 2017]. 2011.

[29] Texas Instruments. SN54ABT245A Datasheet[Online]. Available:http://www.ti.com/lit/ds/symlink/sn74abt245b.pdf[Accessed: 11 Feb. 2018]. Apr. 2005.

[30] R.B.Sheeparamatti, B.G.Sheeparamatti, Manjula Bharamagoudar, Nayan Ambali "Simulink Model for Double Buffering"., Basaveshwar Engineering College., 1-4244-0136-4/06/$20.00., IEEE., 2006.

[31] VGA Connector Pin Information. Available:http://lateblt.tripod.com/bit74.txt[Accessed: 25 Feb. 2018].

[32] Joe O'Dea. AS11-40-5874. Available:https://www.hq.nasa.gov/alsj/a11/images11.html#5874[Accessed: 25 Mar. 2018] 2017.

Appendices

1 Serial Peripheral Interface Bus

The SPI bus [17] is a master-slave full duplex communication protocol. For a minimum implementation of a single master-slave configuration four connections are required, namely:

- Serial Clock (SCK)
- Master Output Slave Input (MOSI)
- Master Input Slave Output (MISO)
- Chip Select (CS or SS)

In order to include additional slave devices, each additional slave requires another chip select connection.
The master device controls both the serial clock and the chip select connected to all the devices and therefore also initiates all communication transactions.

There are four operation modes of SPI, 0 to 3. Each mode defines a unique clock polarity (CPOL) and clock phase (CPHA) on the master serial clock. The default operation mode for an SD card is operating mode 0, however operating mode 3 also works in the majority of applications. A table illustrating the configuration of each of these operating modes can be found in Table 8 below.

Operation Mode	Clock polarity (CPOL)	Clock Phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

Table 8: SPI operating modes

A timing diagram illustrating the behaviour of each of the different modes of operation can be found in Figure 21 below.

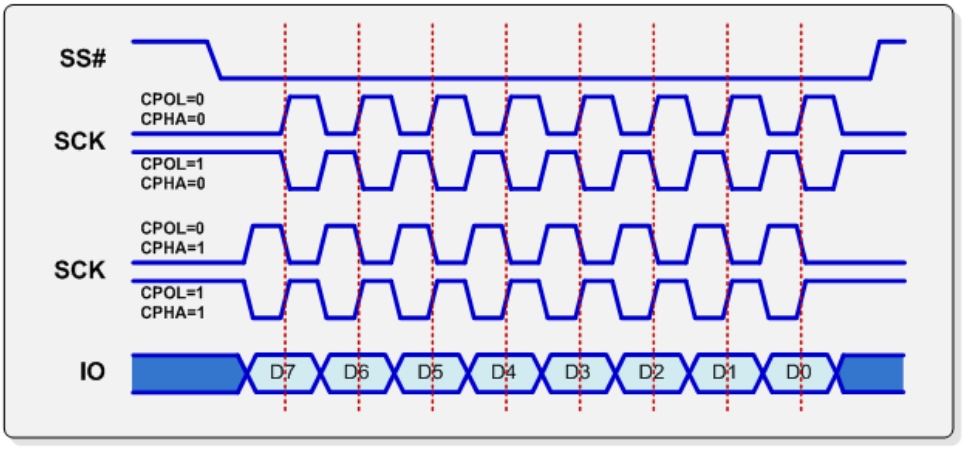

Figure 21: SPI bus timing of the different operating modes [18]

2 SD command response messages

Command Index	Argument	Response	Data	Abbreviation	Description
CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
ACMD41(*1)	*2	R1	No	APP_SEND_OP_COND	For only SDC. Initiate initialization process.
CMD8	*3	R7	No	SEND_IF_COND	For only SDC V2. Check voltage range.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD16	Block length[31:0]	R1	No	SET_BLOCKLEN	Change R/W block size.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Leading command of ACMD<n> command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.
*1:ACMD<n> means a command sequense of CMD55-CMD<n>. *2: Rsv(0)[31], HCS[30], Rsv(0)[29:0] *3: Rsv(0)[31:12], Supply Voltage(1)[11:8], Check Pattern(0xAA)[7:0]					

Figure 22: Common SD command response messages [19]

3 Tristate logic

A diagram and truth table for this logic gate can shown in Figure 23 and Table 9 below respectfully.

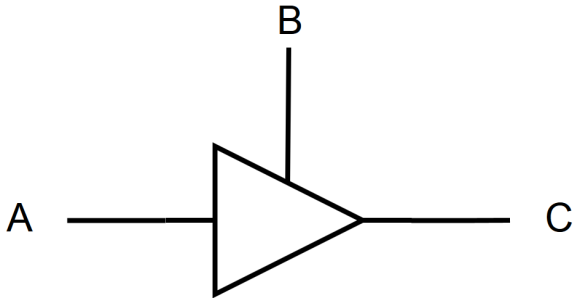

Figure 23: Tristate buffer gate

A	B	C
Low	Low	Z (High impedance)
High	Low	Z (High impedance)
Low	High	Low
High	High	High

Table 9: Tristate gate truth table

As shown in the truth table above, toggling the 'B' input 'high' or 'low' enables the output 'C' to be connected or disconnected from the 'A' input. This physical disconnection is achieved through the use of a third "high impedance" state.

4 VGA prototype output result

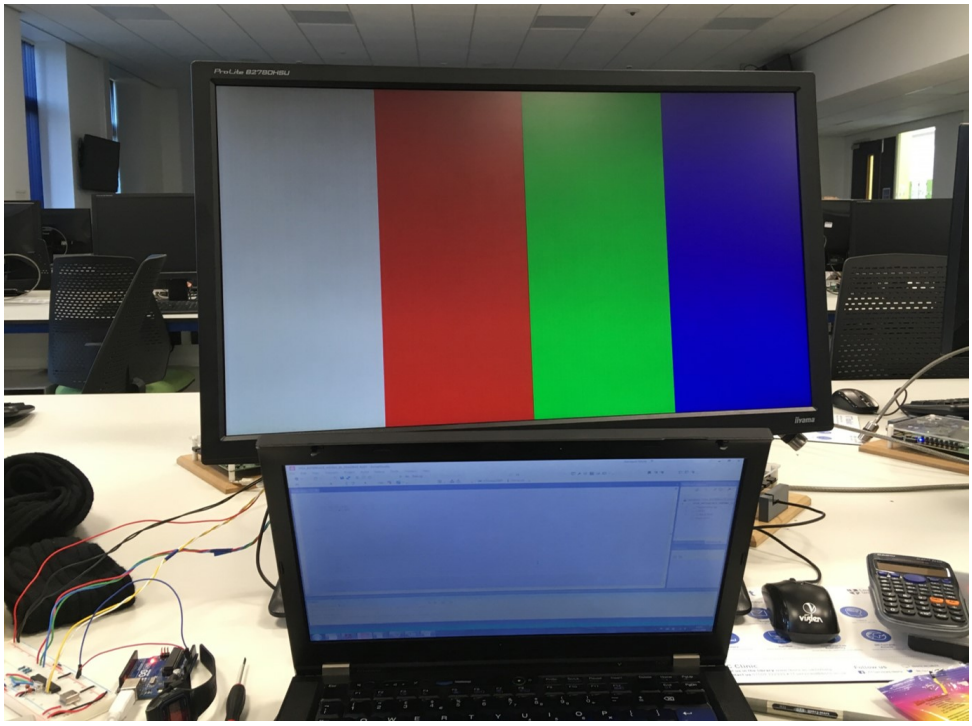

Figure 24: Photo of four colour output via VGA

5 Final circuit implementation

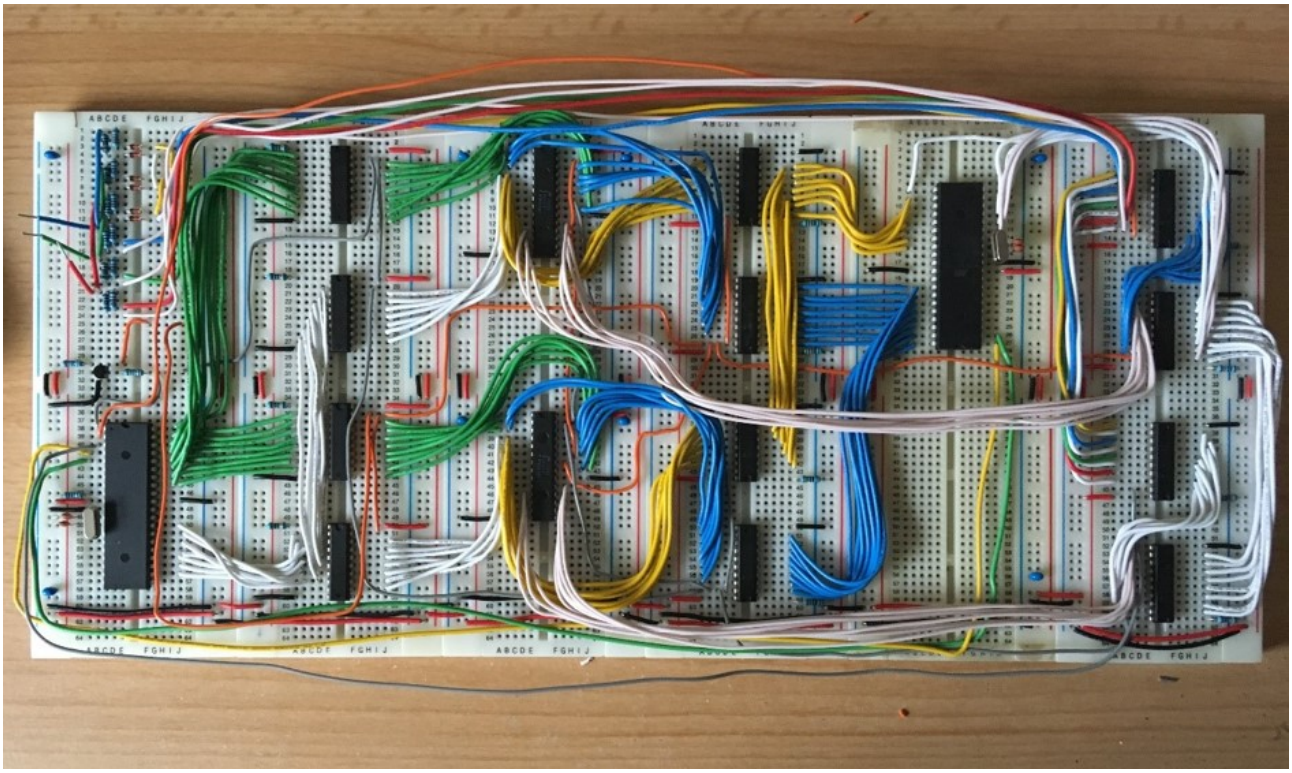

Figure 25: Final system design implemented on breadboard