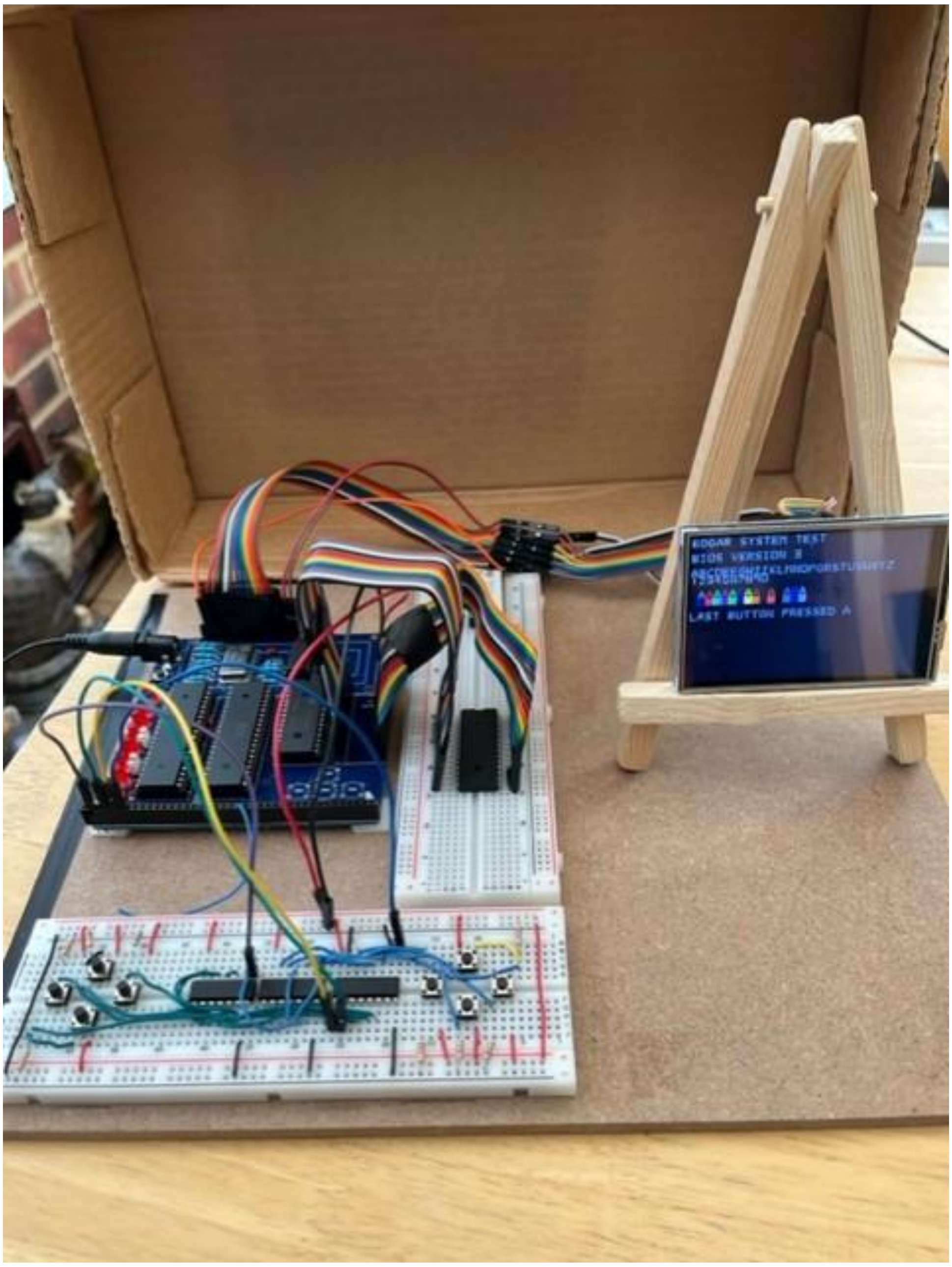


Edgar 6502

Technical Manual

Version 3.0

Contents

Acknowledgements

Much of this document is the integration of work from others, I wish to acknowledge:

- Ben Eater – for detailed design / YouTube videos which introduced me to this topic.
- Forums at 6502.org – for help with getting the original 6502 working.
- 8-bit Guy on Youtube – for sharing the idea of a tile based game which inspired *Catch Clemo.*
- “Tebl” from GitHub – whose PCB I adapted to suit this project.
- My patient wife who has put up with the “joys” of 6502 construction!

Hardware Overview

The goal of this project was to write a computer game from scratch. The computer was inspired by the video series by Ben Eater [1]. The first iteration was built following the videos and using one of Ben Eater's 6502 kits. The base setup is therefore a Ben Eater 6502 computer with 6522 VIA on $6000+. The aim was to make minor modifications to this setup in order to support development of a computer game.

Following construction of the initial computer it was modified by increasing the clock frequency using an 8MHz crystal oscillator which is a drop in replacement for the 1MHz oscillator included in the kit.

A project to write my first game was completed on a TFT screen readily available [2], with a resolution of 320 x 280 pixels. This was then upgraded and the code modified to work with a similar screen of much higher resolution. The 480 x 320 pixel TFT [3] is connected on B0-B7 on port B, RD,WR,RS,CS on port A A7-A4. For controlling the screen 4 push buttons were used on port A A3-A0 of the VIA in addition to 1 interrupt. This was then upgraded to an 8 button controller (modelled on an NES controller [4]) using a 8 bit parallel in serial out shift register – this is connected on port A A2-A0 of the VIA with the addition of 1 interrupt.

To complete the construction of the game a simple BIOS was written to control the screen and read the controller. The setup was christened "EDGAR" in honour of the sentient computer from the film Electric Dreams, as often it seemed to have a mind of its own!

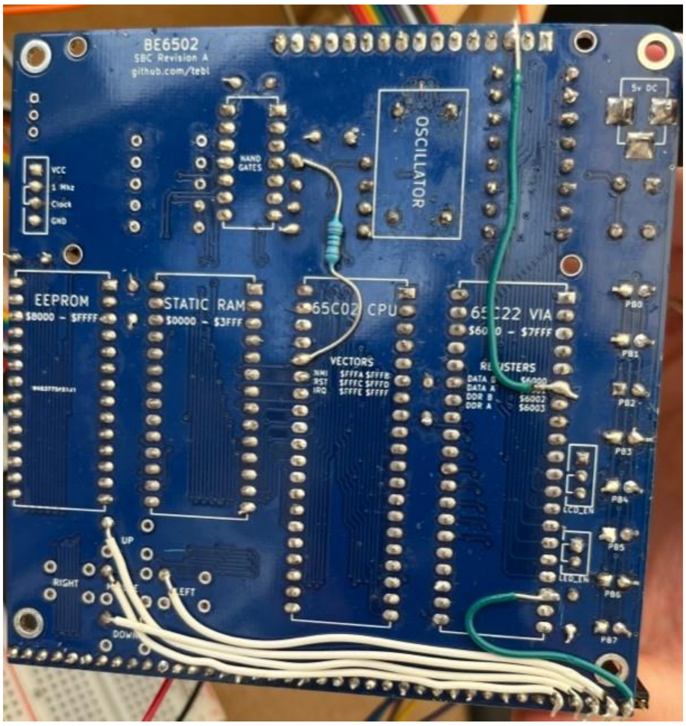
Further development is now ongoing to produce data storage and to enable this a second VIA. A more fully feature game is planned using much larger levels. Once complete it is intended to design a bespoke PCB with these functions integrated.

PCB Expansion – (Not Required for Screen or Controller)

Although the breadboard computer worked well, it was not particularly robust and so a PCB version was made. To avoid the overhead of PCB design, an existing design was employed [5]. The drawback with this design was that it was tailored towards using an LCD screen as was used in Ben Eater's original videos. However, it did have output headers for an external clock, the full address bus, the full data bus and some of the outputs of the VIA. In order to gain full access to all of the outputs of the VIA and make the design even more expandable the following modifications were made:

- Pull-up resistor from BE pin of 6502 to correct an error in the original design
- Changed top header with jumper wire for A4 replacing variable resistance contrast control for LCD screen
- Jumpers from where switches would be for A0-A3 to bottom header on pins 35-38
- Jumper for interrupt from VIA to bottom header on pin 39
- Removed pull high resistors (1k) from A0-A5
- Jumpers in place of 28C256 EEPROM to allow easier programming iterations

The modifications to the PCB are as shown below:



Memory Map – High Level

Address	Contents	Binary (A15,A14,A13,A12)
0000	0000 – 00FF – Zero page 0100 – 01FF – Stack 0200 – 02FF – BIOS RAM (0.25k) 0300 – 0FFF – User Ram (3.25k)	0000
1000	1000 – 1FFF – User Ram (4k)	0001
2000	2000 – 2FFF – User Ram (4k)	0010
3000	3000 – 3FFF – Program Ram (4k)	0011
4000	4000 – 4FFF – Program Ram (4k)	0100
5000	5000 – 5FFF – Program Ram (4k)	0101
6000	6000 – 6FFF – Versatile Interface Adapter 1	0110
7000	7000 – 7FFF – Versatile Interface Adapter 2 **(to be implemented)**	0111
8000	8000 – 8FFF – Program ROM (4k)	1000
9000	9000 – 9FFF – Program ROM (4k)	1001
A000	A000 – AFFF – Program ROM (4K)	1010
B000	B000 – BFFF – Program ROM (4k)	1011
C000	C000 – CFFF – Character map & display constants (4k)	1100
D000	D000 – DFFF – File loader (4K) **(to be implemented)**	1101
E000	E000 – EFFF – BIOS (4k)	1110
F000	F000 – FFF9 – BIOS (4k) FFFA – FFFF – IRQ, Reset, NMI	1111

BIOS Subroutines

Function name:	**Delay_ms**
Location:	
Purpose:	Function to delay for 1ms - 8MHz processor
Registers used:	x, y
Functions called:	None
Preparation work:	None

Function name:	**Get_Cursor_H_Coordinate**
Location:	
Purpose:	Function to get H coordinate from cursor reference (row number) – sets TFT_Cursor_H and TFT_Cursor_H_MSB
Registers used:	a, x
Functions called:	None
Preparation work:	Set H position: *lda #TFT_CURSOR_H_NUM* *sta TFT_CURSOR_H_ROW*

Function name:	**Get_Cursor_W_Coordinate**
Location:	
Purpose:	Function to get W coordinate from cursor reference (column number) - sets TFT_Cursor_W and TFT_Cursor_W_MSB
Registers used:	a, x
Functions called:	None
Preparation work:	Set W position: *lda #TFT_CURSOR_W_NUM* *sta TFT_CURSOR_W_COL*

Function name:	**Read_Controller**
Location:	
Purpose:	Function to read the controller input into the CONTROLLER_STATUS 0 - not pressed, 1 - pressed
Registers used:	a
Functions called:	None
Preparation work:	None

Function name:	**TFT_Draw_Char**
Location:	
Purpose:	Function to output a character to the screen
Registers used:	a, x, y
Functions called:	*Get_Cursor_H_Coordinate* *Get_Cursor_W_Coordinate* *TFT_Write_Com* *TFT_Write_Data*
Preparation work:	Set cursor position: *lda #TFT_CURSOR_H_NUM* *sta TFT_CURSOR_H_ROW* *lda #TFT_CURSOR_W_NUM* *sta TFT_CURSOR_W_COL* Set the character to print: *lda #1* *sta TFT_BYTE_OFFSET* Set foreground colour: *lda #2* *sta TFT_PIXEL_COLOUR* Set background colour: *lda #0* *sta TFT_PIXEL_COLOUR_BG*

Function name:	**TFT_Fill_Data**
Location:	
Purpose:	Function to fill the screen with one specific colour for faster clearing / drawing rectangles – currently only works for black
Registers used:	a, x, y
Functions called:	*TFT_Write_Com* *TFT_Write_Data*
Preparation work:	Set pixel colour to black: *lda #$00* *sta TFT_PIXEL_COLOUR*

Function name:	**TFT_Init**
Location:	
Purpose:	Initialise the TFT screen ready to write data
Registers used:	a, x, y
Functions called:	*Delay_ms* *TFT_Write_Com* *TFT_Write_Data*
Preparation work:	Setup port A for output: *lda #%11110000* *sta DDRA* Setup port B for output: *lda #%11111111* *sta DDRB* Set all control pins to high: *lda #%11110000* *sta PORTA*

Function name:	**TFT_Next_Char**
Location:	
Purpose:	Function to move the cursor on one character
Registers used:	a
Functions called:	*TFT_Next_Line* (if required)
Preparation work:	None

Function name:	**TFT_Next_Line**
Location:	
Purpose:	Function to move the cursor on one line
Registers used:	a
Functions called:	None
Preparation work:	None

Function name:	**TFT_Print_String**
Location:	
Purpose:	Function to print a string - slower than printing each char individually but more flexible
Registers used:	a, x, y
Functions called:	*Get_Cursor_H_Coordinate* *Get_Cursor_W_Coordinate* *TFT_Draw_Char* *TFT_Next_Char* *TFT_Write_Com* *TFT_Write_Data*
Preparation work:	Set cursor position: *lda #TFT_CURSOR_H_NUM* *sta TFT_CURSOR_H_ROW* *lda #TFT_CURSOR_W_NUM* *sta TFT_CURSOR_W_COL* Store string length: *lda #$1e* *sta STRING_LENGTH* Store memory location of string: *lda #$00* *sta STRING_LSB* *lda #$a0* *sta STRING_MSB* Set foreground colour: *lda #2* *sta TFT_PIXEL_COLOUR* Set background colour: *lda #0* *sta TFT_PIXEL_COLOUR_BG*

Function name:	**TFT_Write_Com**
Location:	
Purpose:	Function to write a command to screen bus
Registers used:	a
Functions called:	None
Preparation work:	Load command into a register: *lda #$c2*

Function name:	**TFT_Write_Data**
Location:	
Purpose:	Function to write data to screen bus
Registers used:	a
Functions called:	None
Preparation work:	Load data into a register: *lda #$28*

Screen Setup

Physical connection of the screen to the VIA is the same for both 320x280 and 480x320 screens. The following table outlines the connections necessary which are achieved in my build via the use of Dupont cables.

VIA Pin	Screen Pin	Purpose
6 A4	LCD_CS	LCD control pins
7 A5	LCD_RS	LCD control pins
8 A6	LCD_WR	LCD control pins
9 A7	LCD_RD	LCD control pins
10 B0	LCD_D0	LCD data pins
11 B1	LCD_D1	LCD data pins
12 B2	LCD_D2	LCD data pins
13 B3	LCD_D3	LCD data pins
14 B4	LCD_D4	LCD data pins
15 B5	LCD_D5	LCD data pins
16 B6	LCD_D6	LCD data pins
17 B7	LCD_D7	LCD data pins
(+5V)	LCD_RST	Used to reset screen
(+5V)	5V	Power for screen
(Ground)	Ground	Ground for screen

Before writing to the screen the screen must be initialised, the routine *TFT_Init* achieves this using parameters reverse engineered from the manufacturers by observation of supplied C++/Arduino code. The initialisation routines for the two screens require different parameters which may be seen in *Catch Clemo V2* for the 320x280 screen and *BIOS V2, BIOS V3, Catch Clemo V4* and *Catch Clemo V5* for the 480 x 320 screen.

The BIOS functions for writing characters to the screen operate in the following grid, the character to write to is set via *TFT_CURSOR_H_ROW* and *TFT_CURSOR_W_COL* to set the cursor position. As each row/column reference is a two byte number they are numbered in multiples of 2 as shown in the following table.

	30 Columns Wide - 480 Pixels																													
20 Rows High – 320 Pixels	00	02	04	06	08	0a	0c	0e	10	12	14	16	18	1a	1c	1e	20	22	24	26	28	2a	2c	2e	30	32	34	36	38	3a
	02																													
	04																													
	06																													
	08																													
	0a																													
	0c																													
	0e																													
	10																													
	12																													
	14																													
	16																													
	18																													
	1a																													
	1c																													
	1e																													
	20																													
	22																													
	24																													
	26																													

An individual character can be written to the screen as shown in the following example for the character A in location (20,10):

```
lda #20                 ;move cursor to new position
sta TFT_CURSOR_H_ROW
lda #40
sta TFT_CURSOR_W_COL
lda #1
sta TFT_BYTE_OFFSET
jsr TFT_Draw_Char
```

To aid display of adjacent characters three string handling routines are provided:

- *TFT_Next_Char* – to move the cursor to the next tile to the right, and to the next line if necessary.
- *TFT_Next_Line* – to move the cursor to the left most position on the next line.
- *TFT_Print_String* – to print a string to the screen, using the two functions above.

Whilst not written into a bespoke function it is possible to write to individual pixels / a custom area as follows (note 0,0 is bottom left of the screen):

Set area to fill

```
lda #$2a                ;Column address set - note screen turned on its side so x is y and vice versa!
jsr TFT_Write_Com
lda TFT_CURSOR_H_MSB    ;write x1
jsr TFT_Write_Data
lda TFT_CURSOR_H
jsr TFT_Write_Data
lda TFT_CURSOR_H_MSB_END    ;write x2 MSB first then LSB
jsr TFT_Write_Data
lda TFT_CURSOR_H_END
jsr TFT_Write_Data
lda #$2b                ;page address set
jsr TFT_Write_Com
```

```
lda TFT_CURSOR_W_MSB        ;write y1
jsr TFT_Write_Data
lda TFT_CURSOR_W
jsr TFT_Write_Data
lda TFT_CURSOR_W_MSB_END    ;write y2 MSB first then LSB
jsr TFT_Write_Data
lda TFT_CURSOR_W_END
jsr TFT_Write_Data
lda #$2c                    ;memory write - set memory ready to receive data
jsr TFT_Write_Com
```

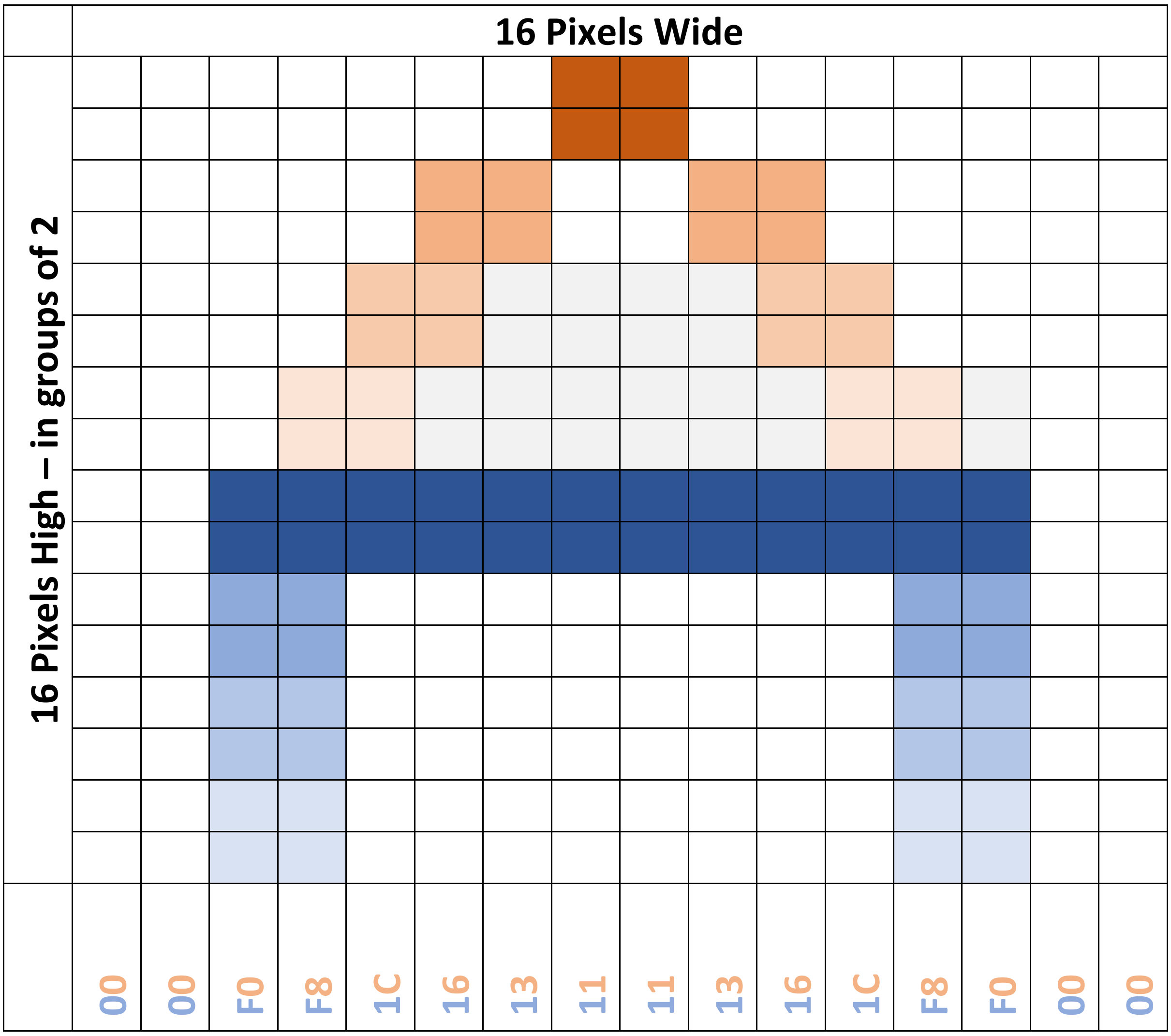
Write out colour to one pixel (repeat this for each pixel)

```
ldx TFT_PIXEL_COLOUR        ;store colour for one pixel
lda black,x                 ;load relevant nibble
;write out first nibble
sta PORTB                   ;output char onto bus
lda #%10100010              ;set tft_wr low (TFT_CS & TFT_WR)
sta PORTA
lda #%11100010              ;put back into previous state (TFT_CS)
sta PORTA
inx                         ;move to next nibble
lda black,x                 ;load relevant nibble
;write out second nibble
sta PORTB                   ;output char onto bus
lda #%10100010              ;set tft_wr low (TFT_CS & TFT_WR)
sta PORTA
lda #%11100010              ;put back into previous state (TFT_CS)
sta PORTA
```

Character Setup

For speed of output and to reduce data storage each character, whilst being represented on the screen as 16x16 pixels, is limited on the vertical axis to groups of two pixels giving an effective resolution of 16x8 pixels.

Each column is therefore represented as an 8 bit number in hexadecimal as in the below example:



Each character has two colours, a foreground colour and a back ground colour. These are set via the variables *TFT_PIXEL_COLOUR* or *TFT_PIXEL_COLOUR_BG*. Further details are in the colour specification section which follows.

For example:

lda #$02 *;set foreground pixel colour to white*

sta TFT_PIXEL_COLOUR

lda #00 ;set background pixel colour to black

sta TFT_PIXEL_COLOUR_BG

The BIOS character set is stored in program ROM from address C000. It is possible to define custom characters within a program and then divert the character printing routines to this new character set. In order to do this the variables *CURRENT_MAP_LSB* and *CURRENT_MAP_MSB* must contain the memory address of the custom characters.

For example, to set the address to a character map at $a100:

lda #$00

sta CURRENT_MAP_LSB

lda #$a1

sta CURRENT_MAP_MSB

Character Map

Character	Decimal	Hexadecimal
Null	0	00
A	1	01
B	2	02
C	3	03
D	4	04
E	5	05
F	6	06
G	7	07
H	8	08
I	9	09
J	10	0A
K	11	0B
L	12	0C
M	13	0D
N	14	0E
O	15	0F
P	16	10
Q	17	11
R	18	12
S	19	13
T	20	14
U	21	15
V	22	16
W	23	17
X	24	18
Y	25	19
Z	26	1A
SPACE	27	1B
1	28	1C
2	29	1D
3	30	1E
4	31	1F
5	32	20
6	33	21
7	34	22
8	35	23
9	36	24
0	37	25

Colour Setup

There are 17 default colours within the system. As each number is two bytes the colours are numbered in terms of the byte offset from the first colour as follows:

Colour	Decimal	Hexadecimal
Black	0	00
White	2	02
Red	4	04
Cyan	6	06
Magenta	8	08
Green	10	0a
Blue	12	0c
Yellow	14	0e
Orange	16	10
Brown	18	12
Light Red	20	14
Dark Grey	22	16
Light Grey	24	18
Light Green	26	1a
Light Blue	28	1c
Very Light Grey	30	1e
Light Brown	32	20

To control the foreground or background colour set the variables *TFT_PIXEL_COLOUR* or *TFT_PIXEL_COLOUR_BG*.

For example:

lda #$02 *;set foreground pixel colour to white*

sta TFT_PIXEL_COLOUR

lda #00 *;set background pixel colour to black*

sta TFT_PIXEL_COLOUR_BG

It is possible to define 16 bit custom colours to use with all of the routines for drawing characters. To achieve this the colour values must be appended to the end of the colour table at the end of the character map at C000. To maintain compatibility with other EDGAR programs the built in colours should remain as is, but this is not critical. Colours should be stored little endian first as otherwise they get flipped when outputting. For example:

RED = $00f8

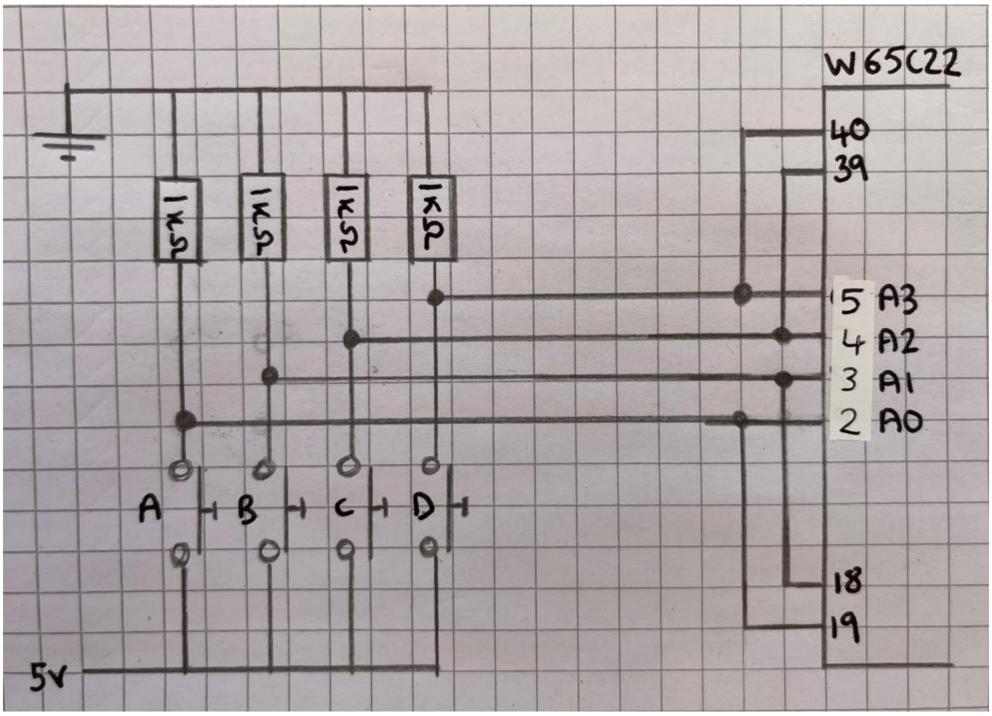
Controller Design

There are three acceptable controller configurations depending on the preference of the user and the availability of equipment.

4 Button (with no logic)

This is the simplest setup but requires the most connections to the VIA. This setup works with *BIOS V1*, *BIOS V2, Catch Clemo V2* and *Catch Clemo V4*.

Requires: 4 x push buttons, 4 x 1k resistors, connecting wires.



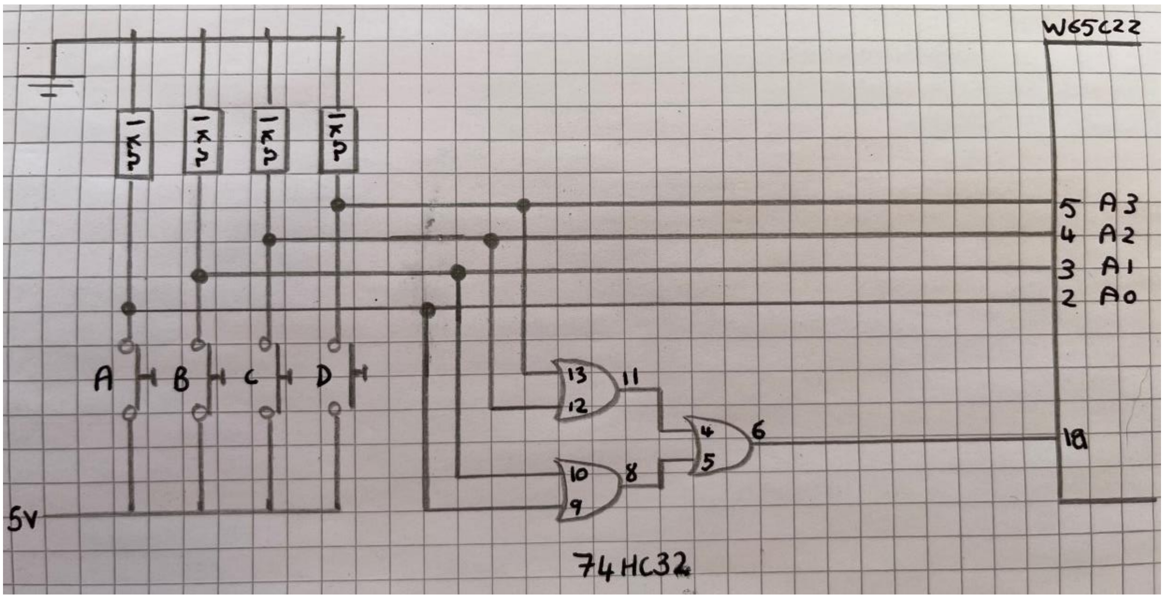
1. Connect resistors to ground
2. Connect push button to resistor and +5V
3. Connect from each button to respective pins on VIA

When a button is pressed it pulls the interrupt and the data pin of the VIA high triggering the interrupt to read which button was pressed.

4 Button (with logic to reduce the number of interrupts required)

Through using logic gates this setup reduces the number of connections to the VIA. This setup works with *BIOS V1*, *BIOS V2, Catch Clemo V2* and *Catch Clemo V4*.

Requires: 4 x push buttons, 4 x 1k resistors, 74HC32 OR gate, connecting wires



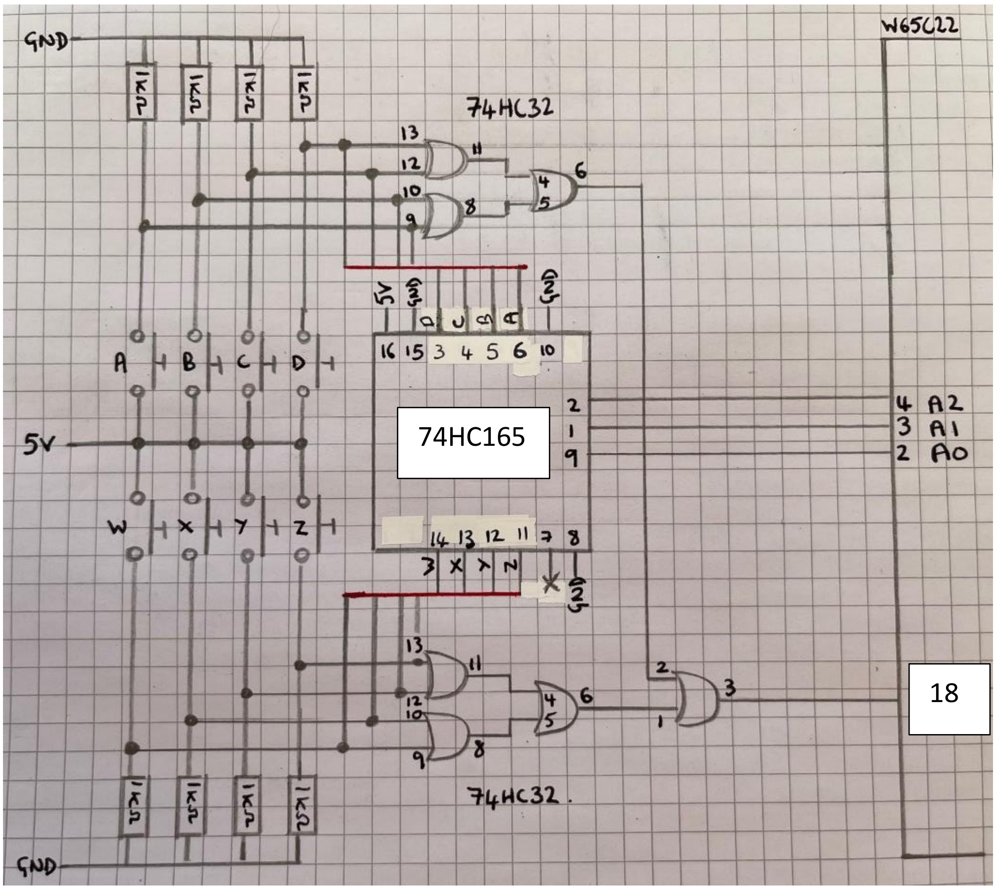
1. Connect resistors to ground
2. Connect push button to resistor and +5V
3. Connect from each button to respective pin on VIA
4. Connect from each button to respective pin on 74HC32
5. Connect the 74HC32 to the VIA

When a button is pressed it pulls the interrupt (through the OR gates) and the data pin of the VIA high triggering the interrupt to read which button was pressed.

8 Button (with logic and shift register)

This more complex setup reduces the number of connections to the VIA and increases the number of input buttons via the use of a shift register. This setup works with *BIOS V3,* and *Catch Clemo V5*.

Requires: 8 x push buttons, 8 x 1k resistors, 2 x 74HC32 OR gate, 1 x 74HC165 Shift register, connecting wires



1. Connect resistors to ground
2. Connect push button to resistor and +5V
3. Connect from each button to respective pin on 74HC165
4. Connect 74HC165 to the VIA
5. Connect from each button to respective pin on the 74HC32s
6. Connect the 74HC32s to the VIA

When a button is pressed it pulls the interrupt (through the OR gates) and the data pin of the VIA high triggering the interrupt. The interrupt routine then uses *Read_Controller* to read from the shift register, returning the button which was pressed into the variable *CONTROLLER_STATUS*.

Memory Map – Low Level

Address (Hexadecimal)	EDGAR Variable Name	Usage
0000-007f		Not reserved
0080	*CHAR_MAP_LSB*	Character map location in memory (built in) least significant byte
0081	*CHAR_MAP_MSB*	Character map location in memory (built in) most significant byte
0082	*CURRENT_MAP_LSB*	Current character map location in memory least significant byte (to allow character map switching for custom character maps)
0083	*CURRENT_MAP_MSB*	Current character map location in memory most significant byte (to allow character map switching for custom character maps)
0084	*LEVEL_LSB*	Custom character map location in memory least significant byte
0085	*LEVEL_MSB*	Custom character map location in memory most significant byte
0086	*STRING_LSB*	String location in memory least significant byte
0087	*STRING_MSB*	String location in memory most significant byte
0088 – 00ff		Not reserved
0100 – 01ff		6502 stack
0200	*COUNTER_I*	program counter which can be used independently of x,y registers
0201	*COUNTER_J*	program counter which can be used independently of x,y registers
0202	*TFT_H_COUNTER*	location of TFT height counter for drawing
0203	*TFT_W_COUNTER*	location of TFT width counter for drawing
0204	*TFT_HALF_COUNTER_H*	counter for faster TFT drawing counts half the screen for H
0205	*TFT_HALF_COUNTER_W*	counter for faster TFT drawing counts half the screen for W
0206	*TFT_PIXEL_COLOUR*	location of offset of current pixel colour foreground (an increment from black)

0207	*TFT_PIXEL_COLOUR_BG*	location of offset of current pixel colour background (increment from black)
0208	*TFT_CURSOR_H*	Location in pixels of cursor in y - LSB
0209	*TFT_CURSOR_H_MSB*	location in pixels of cursor in y – MSB
020a	*TFT_CURSOR_H_MSB_END*	location in pixels of cursor end in y - required for row 4
020b	*TFT_CURSOR_H_END*	Location in pixels of cursor end in y (top right of character)
020c	*TFT_CURSOR_H_ROW*	location of cursor in number of rows
020d	*TFT_CURSOR_W*	location of cursor in x in pixels - LSB
020e	*TFT_CURSOR_W_MSB*	location of cursor in x in pixels - MSB
020f	*TFT_CURSOR_W_MSB_END*	location of cursor end in x in pixels - MSB - required for column 16
0210	*TFT_CURSOR_W_END*	location of cursor end in x in pixels (top right of character)
0211	*TFT_CURSOR_W_COL*	location of cursor in number of columns
0212	*TFT_BYTE*	current byte from character map
0213	*TFT_BYTE_OFFSET*	offset to current character from start of character map - (x by 16 to get actual offset)
0214	*STRING_LENGTH*	length of a string for printing
0215	*CONTROLLER_STATUS*	Status of all 8 controller buttons bit 0 - A, bit 1 - B, bit 2 - C, bit 3 - D, bit 4 - W, bit 5 - X, bit 6 - Y, bit 7 – Z
0216-02ff		Reserved for BIOS use
0300-2fff		User RAM
3000-5fff		User RAM / Loaded program RAM
6000	*PORTB*	Port B of VIA 1
6001	*PORTA*	Port A of VIA 1
6002	*DDRB*	Data direction register for port B of VIA 1
6003	*DDRA*	Data direction register for port A of VIA 1
6004-600b		Reserved for VIA 1
600c	*PCR*	Peripheral control register for VIA 1, controls positive or negative edges for interrupts
600d	*IFR*	Interrupt flag register
600e	*IER*	Interrupt enable register

600f-6fff		Reserved for VIA 1
7000-7fff		Reserved for VIA 2 **(to be implemented)**
8000-bfff		Program ROM
C000 – CFFF		Character map & display constants
D000 – DFFF		File loader **(to be implemented)**
E000 – FFF9		BIOS
FFFA – FFFF		IRQ, Reset, NMI

Schematic

Basic computer as Ben Eater 6502 [1] but with the connections to the VIA as outlined in the screen setup and controller setup sections above.

References

[1] Build a 6502 computer | Ben Eater

[2] Elegoo El-SM-004 Arduino UNO R3 2.8 "TFT touch screen with SD card socket, blue : Amazon.co.uk: Computers & Accessories

[3] Dealikee 3.5 inch TFT Touch Screen Module with SD Card Socket Compatible for Arduino Uno Mega 2560 Board SC3A-1 : Amazon.co.uk: Business, Industry & Science

[4] NES Controller Interface with an Arduino UNO - Projects (allaboutcircuits.com)

[5] GitHub - tebl/BE6502-Build-a-65c02-computer: A PCB being made while watching Ben Eaters "Build a 6502 computer" video series. Includes the computer itself, a standalone slow clock and an Arduino Mega shield for the bus monitor sketch..