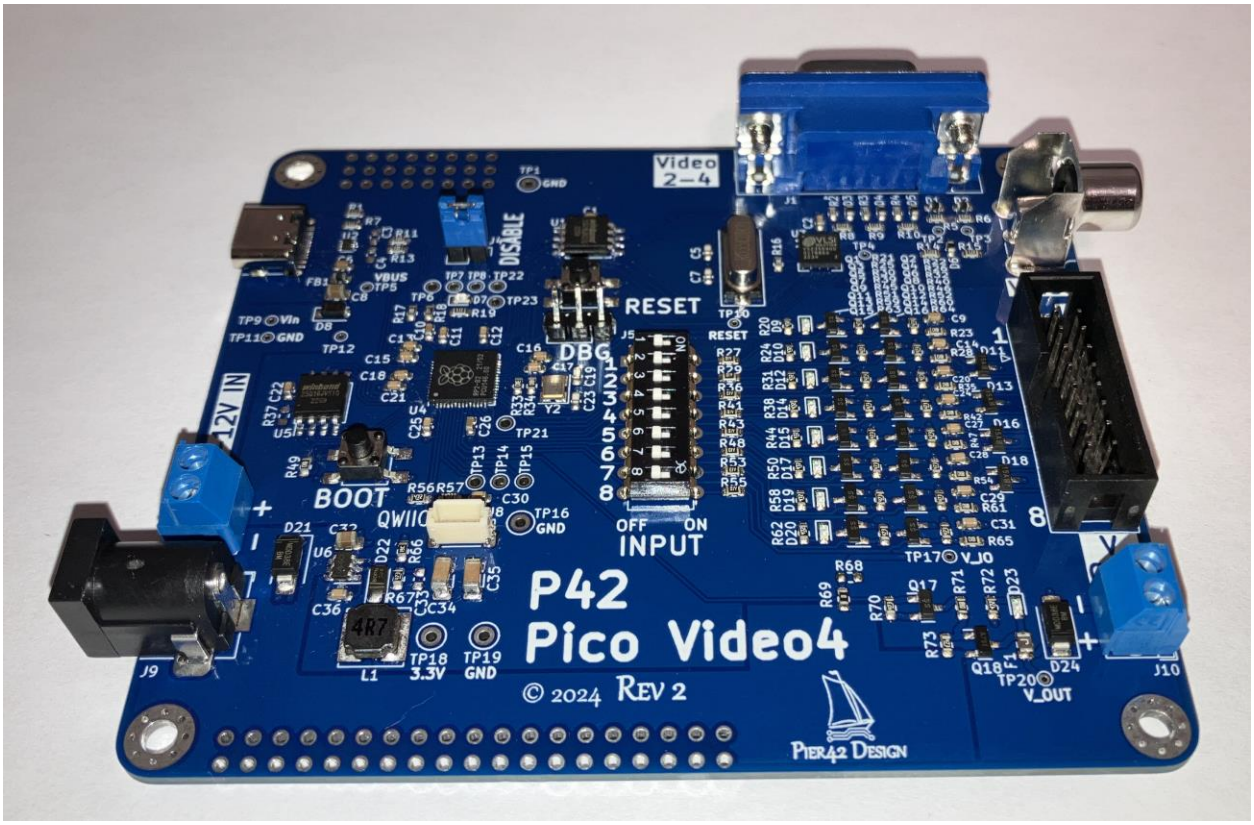


Pier42 Pico-Video4 Rev2.0

Designed by Wolfgang Friedrich

https://hackaday.io/project/196770-rpi-pico-video4

https://www.tindie.com/products/pier42/rpi-pico-rp2040-4x-video-palntsc-board/

Table of Contents

Introduction

The Pico Video4 Display is a stand-alone RaspberyPi pIco board.

This board provides up to 4 analog composite video display interfaces with integrated frame buffer memory accessible through SPI. The 4 video outputs are accessible through 1 RCA connector and 1 VGA DB15-HD connector that uses the red, green and blue channels for the composite signal.

PAL and NTSC output formats are supported; display resolutions range from 320x200 with 65536 colours up to 720x576 with reduced colour count. It can be ordered as NTSC version with a 3.579545 MHz crystal or as PAL version with 4.43618 MHz crystal. Currently 2 resolutions are implemented: NTSC 320x200 with 256 colours or PAL 300x240 with 256 colours. The heart of this design is the VLSI VS23S040 chip, which is able to output 4 composite video with resolutions from 320x200 in 65536 colours to 720x576 in 4 colours. The chip has a 1Mbit framebuffer per channel, unused memory can be used for graphics tiles, which can be copied into the image data by the internal fast memory block move hardware.

A 16 Mbit SPI FLASH memory is available on-board. It is pre-loaded with a character bitmap consisting of 94 characters (ASCII code 33-126), a vintage Amiga Boing Ball demo, and 9 images ready to display.

It has 8 inputs that are voltage compatible to the input voltage; 5 V when powered from USB-C and up to 12 V when powered from the barrel connector or screw terminal.

Specifications:

- Operating supply voltage 5.0 V – 12 V
- Raspberry Pi Pico with 16 Mbit Program Flash memory
- USB-C connector for 5 V power and communication/programming
- SWdebug header
- 4x Composite Video Output
- Maximum resolution 720x576 in 4 colours
- Implemented resolutions: NTSC 320x200 or 426x200 with 256 colours and PAL 300x240 or 500x240 with 256 colours
- Crystal: NTSC 3.579545 MHz or PAL 4.43618 MHz
- Communication interface: SPI up to 38 MHz
- Video Frame Buffer: 4x 1 Mbit = 4x 128 KByte
- Data Flash: 16 Mbit = 2 Mbyte
- QWIIC I2C connector vertical orientation
- 8 inputs, high voltage up to input voltage level 5-12 V, with LED indicator
- 8 dip switches to simulate the inputs
- LED and 2-pin jumper for status and control
- Power output, switched and fused with 500mA polyfuse
- TH prototyping area
- Size: 85mm x 100mm (3.3" x 3.9")

Hardware

Video channel mapping

The 4 video channels of the VS23S040 are mapped as following:

Video Channel	Hardware Mapping
0	RCA port
1	VGA blue
2	VGA green
3	VGA red

RPi Pico pin mapping

Function	RPI GPIO port	Alt. Function	Note
IN1Pin	13		
IN2Pin	12		
IN3Pin	11		
IN4Pin	10		
IN5Pin	9		
IN6Pin	8		
IN7Pin	7		
IN8Pin	6		
OUTPin	5		
OUTLED	27		
ENABLE	26		
TP6	29	UART0 RX/ADC3	
TP7	28	UART0 TX/ADC2	
TP8	25	UART1 RX	
TP13	2	I2C1 SDA	
TP14	3	I2C1 SCL	
TP15	4	UART1 TX	
GPIO15	15		Bottom side on PCB rev1
GPIO21	21		Bottom side on PCB rev1
GPIO22	22		Bottom side on PCB rev1
QWIIC_SCL	1	UART0 RX	J8 (pins 3 and 4 are swapped)
QWIIC_SDA	0	UART0 TX	J8 (pins 3 and 4 are swapped)

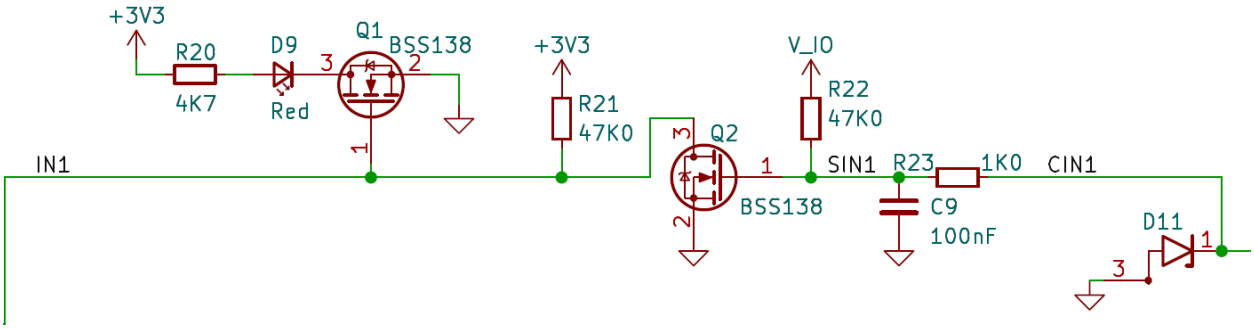
From the table GPIOs are through holes, wires can easily soldered to them. All adjacent holes are on a 2.54mm grid. TP 15, 21, 22 are just pads on the bottom side, wires could be soldered as well. J8 is a QWIIC connector, but I messed up the pinout, pins 3 and 4 are switched. Infos can be found here:
[Sparkfun QWIIC]

Most GPIOs have more interface functions or can also be used as PWM / SIO (single-cycle IO) / PIO (programmable IO). The datasheet tells all the details or here is a good summary:

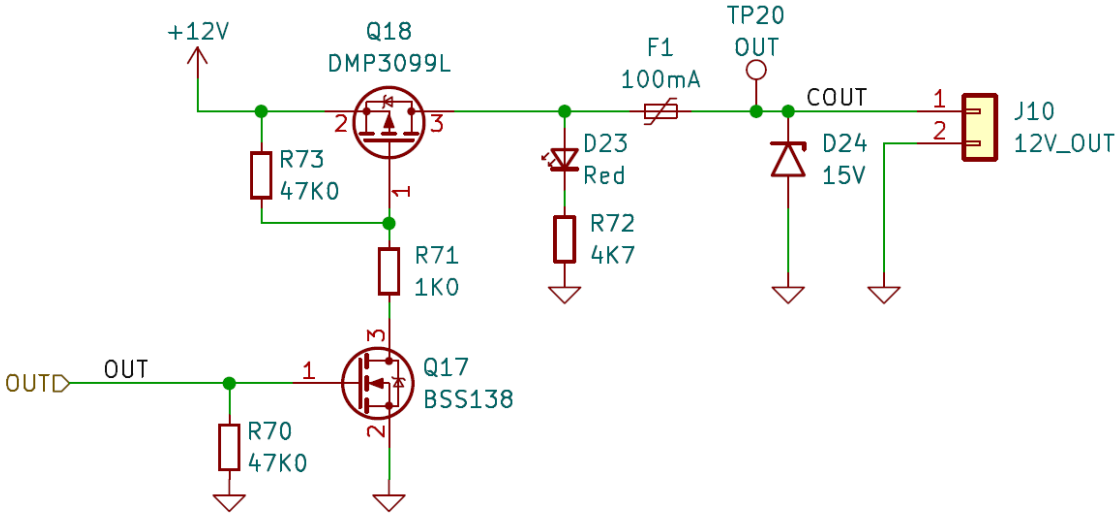
[GPIO Function Matrix]

IO circuits

Input circuit:



Output power circuit:



.

More heading

.

More headings

.

Flash Memory

Memory Map

By default the 16 Mibit (2 MiByte) Flash memory is used as follows:

Memory Bytes		Description
0 - 751	0x00000-0x002EF	94 letters each 8 byte
752 - 4095	0x002F0-0x00FFF	empty
4096 - 12287	0x01000-0x02FFF	8 frames 32x32 byte for BoingBall demo
12287 - 131071	0x03000-0x1FFFF	BMP image for picture demo or logo for Input demo
131072 - 262143	0x20000-0x3FFFF	Image #1
262143 - 393215	0x40000-0x5FFFF	Image #2
393216 - 524287	0x60000-0x7FFFF	Image #3
524288 - 655359	0x80000-0x9FFFF	Image #4
655786 - 786431	0xA0000-0xBFFFF	Image #5
786432 - 917503	0xC0000-0xDFFFF	Image #6
719504 - 1048575	0xE0000-0xFFFFF	Image #7
1048576 - 1179647	0x100000-0x11FFFF	Image #8
1179648 - 1310719	0x120000-0x13FFFF	Image buffer for colour conversion
1310719 - 2097151	0x140000-0x1FFFFF	free

Program FLASH with character bitmap

See https://hackaday.io/project/21097/instructions for details.

Here is am ASCII character table for the symbols programmed in Flash by default

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
33	21	!	57	39	9	81	51	Q	105	69	i
34	22	“	58	3A	:	82	52	R	106	6A	j
35	23	#	59	3B	;	83	53	S	107	6B	k
36	24	$	60	3C	<	84	54	T	108	6C	l
37	25	%	61	3D	=	85	55	U	109	6D	m
38	26	&	62	3E	>	86	56	V	110	6E	n
39	27	‘	63	3F	?	87	57	W	111	6F	o

40	28	(	64	40	@	88	58	X	112	70	p
41	29	0	65	41	A	89	59	Y	113	71	q
42	2A	*	66	42	B	90	5A	Z	114	72	r
43	2B	+	67	43	C	91	5B	[	115	73	s
44	2C	,	68	44	D	92	5C	\	116	74	t
45	2D	-	69	45	E	93	5D	]	117	75	u
46	2E	.	70	46	F	94	5E	^	118	76	v
47	2F	/	71	47	G	95	5F	_	119	77	w
48	30	0	72	48	H	96	60	`	120	78	x
49	31	1	73	49	I	97	61	a	121	79	y
50	32	2	74	4A	J	98	62	b	122	7A	z
51	33	3	75	4B	K	99	63	c	123	7B	{
52	34	4	76	4C	L	100	64	d	124	7C	|
53	35	5	77	4D	M	101	65	e	125	7D	}
54	36	6	78	4E	N	102	66	f	126	7E	~
55	37	7	79	4F	O	103	67	g			
56	38	8	80	50	P	104	68	h			

Arduino Integration

What to do for programming the board with Arduino IDE 2.3:

Follow:

https://arduino-pico.readthedocs.io/en/latest/install.html#installing-via-arduino-boards-manager

Software

Resolutions

Currently implemented are

- PAL 300x240 8bit
- PAL 500x240 8bit
- NTSC320x200 8bit
- NTSC426x200 8bit

How to select an implemented resolution, please see chapter “Config ()” on page 11.

Possible Resolutions are (copied from the VLSI VS23S040 datasheet):

Resolution	H	V	Pixels	Colors [1]	Bits per pixel	Memory bytes
NTSC YUV422 [2]	352	240	84480	65536	8+4	126720
MCGA	320	200	64000	65536	16	128000
CDG	300	216	64800	65536	16	129600
QVGA	320	240	76800	8192	13	124800
NTSC VCD	352	240	84480	4096	12	126720
PAL VCD	352	288	101376	1024	10	126720
NTSC noninterlaced	440	243	106920	512	9	120285
PAL noninterlaced	520	288	149760	128	7	131040
HVGA	480	320	153600	64	6	115200
EGA	640	350	224000	16	4	112000
VGA letterbox	640	400	256000	16	4	128000
NTSC Analog	440	486	213840	16	4	106920
NTSC SVCD	480	480	230400	16	4	115200
NTSC DVD	720	480	345600	8	3	129600
VGA	640	480	307200	8	3	115200
PAL Analog	520	576	299520	8	3	112320
PAL SVCD	480	576	276480	8	3	103680
PAL DVD	720	576	414720	4	2	103680

Implementation of the current resolutions was greatly supported by VLSI Solutions http://www.vlsi.fi/en/home.html as part of demo code for the VS23S010D-L chip.

For more information see:

http://www.vsdsp-forum.com/phpbb/viewforum.php?f=14

http://www.vlsi.fi/en/products/vs23s010.html

https://webstore.vlsi.fi/epages/vlsi.sf/en_GB/?ObjectID=13893093

Constants

The following constants are provided by the library. They are useful to make the code adapting to other resolutions.

Name	Description
XPIXELS	X size of the picture area
YPIXELS	Y size of the picture area

Color Palette

The colour palette is defined as part of the controller configuration. It is described in good detail by a series of forum posts on the VLSI website:

Nice color palettes for VS23S010 TV-Out

Understanding Protolines and Line Pointers

Also a handy table to pick colours by their 8 bit YUV value is located in chapter “8-Bit Default Palette”

P42_Pico_Video4 Functions

COM Port Terminal set to 19200 baud, 8N1, transmit CR only

```
w[1-8]Write Image into Buffer 1-8
w0    Write Logo Image into Memory 128x128 px^2
i[1-8]Colour convert and write Image into Buffer 1-8
i0    Colour convert and write Logo Image into Memory 128x128 px^2
s[0-8]Show Image[0-8] on screen
m[0-8]Show Memory Dump of Image[0-8] 'x':Exit 'anykey':next page
d[0-9]Delete memory section of Image[0-9]
c     Clear Screen
v     Show Version
?     This help command
```

Library Control Functions VS23S040.h/.cpp

P42Display ()

Board and SPI interface configuration.

Pin	Pin Nr	Direction	Default	Description
MVBLK		In	X	Ready signal after Block Move Command
nWP		Out	High	Write Protect for Flash Memory and Video Controller
nHOLD		Out	High	Hold signal for SPI interface
nMemSelect		Out	High	SPI Select for Flash Memory
nSlaveSelect		Out	High	SPI Select for Video Controller

Config ()

Full configuration of the video controller including protolines, picture lines and video resolution.

```
word Config( byte channel );
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
return value	word	Content of the Device ID register for the selected channel

Example: Configure channel 0

```
Result = P42Display.Config ( CH0 );
```

The resolution is set in the VS23S040D.h file by removing the comment of the desired resolution:

```
// *** Select Video Resolution here ***
#define NTSC320x200
//#define PAL300x240
```

Here NTSC320x200 is selected.

SPIReadRegister ()

Read an 8bit register value from the video controller.

```
byte SPIReadRegister (byte address, boolean debug);
```

Value	Size	Description
address	byte	Opcode of the video controller command, also called register address
debug	boolean	Define if the address and return value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

return value	byte	Result of the read command

Example: Read Manufacturer and Device ID register (8bit)

```
Result = P42Display.SPIReadRegister (ReadDeviceID, true);
```

⇨ Result will be 0x2B.

Debug output will be: "SPI address: 0x9F : 0x2B"

SPIReadRegister16 ()

Read a 16bit register value from the video controller.

```
word SPIReadRegister16 (byte address, boolean debug);
```

Value	Size	Description
address	byte	Opcode of the video controller command, also called register address
debug	boolean	Define if the address and return value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`
return value	byte	Result of the read command

Example: Read Manufacturer and Device ID register (16bit)

```
Result = P42Display.SPIReadRegister (ReadDeviceID, false);
```

⇨ Result will be 0x2B00.

No debug output.

SPIWriteRegister ()

Write an 8bit value to a register in the video controller.

```
void SPIWriteRegister (byte address, byte value, boolean debug);
```

Value	Size	Description
address	byte	Opcode of the video controller command, also called register address
value	byte	Register value to be written into the given register address
debug	boolean	Define if the address and register value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

Example: Write GPIO Control Register

```
P42Display.SPIWriteRegister( WriteGPIOControl, PIO4Dir | PIO4High, false );
```

SPIWriteRegister16 ()

Write a 16bit value to a register in the video controller.

```
void SPIWriteRegister16 (byte address, word value, boolean debug);
```

Value	Size	Description
address	byte	Opcode of the video controller command, also called register address
value	word	Register value to be written into the given register address
debug	boolean	Define if the address and register value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

Example: Write left limit of visible picture

```
SPIWriteRegister16 (WritePictureStart, STARTPIX-1, false );
```

SPIWriteRegister32 ()

Write a 32bit value to a register in the video controller.

```
void SPIWriteRegister32 (byte address, unsigned long value, boolean debug);
```

Value	Size	Description
address	byte	Opcode of the video controller command, also called register address
value	unsigned long	Register value to be written into the given register address
debug	boolean	Define if the address and register value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

Example: Write microcode

```
SPIWriteRegister32 (WriteProgram, ((OP4 << 24) | (OP3 << 16) | (OP2 << 8) | (OP1)), false );
```

SPIWriteRegister40 ()

Write a 40bit value to a register in the video controller. The 40bit value is split into 2x 16bit plus one 8bit parameter for a more intuitive and readable code. Only the 'Block Move Control 1' register is 40bit wide, so the parameters are conveniently named for the register only.

```
void SPIWriteRegister40 (byte address, word source, word target, byte control, boolean debug );
```

Value	Size	Description
address	byte	Opcode of the video controller command, also called register address
source	word	Source memory address for the block move command
target	word	Target memory address for the block move command
control	word	Control bits for block move and DAC output
debug	boolean	Define if the address and register value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

Example: Enable PAL Y lowpass filter

```
SPIWriteRegister40 (WriteBlockMoveControl1, 0x0000, 0x0000, BMVC_PYF, false );
```

SPIReadByte ()

Read an 8bit value from the SRAM video buffer memory in the video controller.

```
byte SPIReadByte (byte channel, unsigned long address);
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
address	unsigned long	video buffer memory address
return value	byte	Result of the read command

Example: Read address 0 from channel 1

```
Byte1 = SPIReadByte ( CH1, 0x00000000 );
```

SPIReadWord ()

Read a 16bit value from the SRAM video buffer memory in the video controller.

```
word SPIReadByte (byte channel, unsigned long address);
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
address	unsigned long	video buffer memory address
return value	word	Result of the read command

Example: Read address 0 from channel 2

```
Word1 = SPIReadWord ( CH2, 0x00000000 );
```

SPIWriteByte ()

Write an 8bit value to the SRAM video buffer memory in the video controller.

```
void SPIWriteByte (byte channel, unsigned long address, byte value, boolean debug );
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
address	unsigned long	video buffer memory address
value	byte	Data value to be written into the given memory address
debug	boolean	Define if the address and memory value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

Example: Write a YUV data value to a specific x,y coordinate to channel 3

```
SPIWriteByte ( CH3, PICLINE_BYTE_ADDRESS(y) + x, YUVdata, false);
```

SPIWriteWord ()

Write a 16bit value to the SRAM video buffer memory in the video controller.

```
void SPIWriteWord (byte channel, unsigned long address, word value, boolean debug );
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
address	unsigned long	video buffer memory address
value	word	Data value to be written into the given memory address
debug	boolean	Define if the address and memory value will be written to a serial debug port. The debug port needs to be configured in the setup() routine, e.g. `Serial.begin(115200);`

Example: Clear entire video buffer channel 0 memory (everything not only the picture data area!)

```
for ( i=0; i < 65536; i++)
       SPIWriteWord ( CH0, i, 0x0000, false);
```

Library Graphics functions VS23S040.h/.cpp

ClearScreen ()

Clear the video screen by filling the framebuffer memory with a given colour value. The colour can be picked from the default colour table in chapter “8-Bit Default Palette”.

```
void ClearScreen ( byte channel, byte colour );
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
colour	byte	YUV colour value picked from default palette

Example: Clear screen and set to a light blue background colour for channel 0.

```
P42Display.ClearScreen ( CH0, 0x5c );
```

FilledRectangle ()

Draw a filled rectangle into the video buffer. This function was re-used from the Arduino demo provided by VLSI. See here for details: http://www.vsdsp-forum.com/phpbb/viewtopic.php?f=14&t=2172

The colour can be picked from the default colour table in chapter “8-Bit Default Palette”.

```
void FilledRectangle (byte channel, u_int16 x1, u_int16 y1, u_int16 x2, u_int16 y2, u_int16 color);
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
x1	u_int16	x coordinate of top left corner of the rectangle
y1	u_int16	y coordinate of top left corner of the rectangle
x2	u_int16	x coordinate of bottom right corner of the rectangle
y2	u_int16	y coordinate of top bottom right of the rectangle
color	u_int16	YUV colour value picked from default palette. Only the lower 8 bit are used for colour information.

Example: Draw a 10 pixel by 10 pixel square in the top left corner of the screen in yellow colour on channel 0.

```
P42Display.FilledRectangle ( CH0, 0, 0, 9, 9, 0xBF );
```

SetRGBPixel ()

This is an experimental function and should not be used for now. Eventually it will perform a RGB-to-YUV conversion depending on the colour space of the given colourspace.

Draw a pixel on the screen at the given coordinates.

The colour is a 32 bit unsigned integer of the format 0x00RRGGBB representing a 24bit RGB value.

```
void SetRGBPixel (byte channel, word x, word y, unsigned long colour);
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
x	word	x coordinate of the pixel
y	word	y coordinate of the pixel
colour	unsigned long	32 bit unsigned integer of the format 0x00RRGGBB representing a 24bit RGB value

Example: Draw a yellow pixel at the coordinates on channel 0.

```
P42Display.SetRGBPixel ( CH0, 314, 159, 0x00FFFF00 );
```

SetYUVPixel ()

Draw a pixel on the screen at the given coordinates.

The colour can be picked from the default colour table in chapter “8-Bit Default Palette”.

```
void SetYUVPixel (byte channel, word x, word y, byte colour);
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
x	word	x coordinate of the pixel
y	word	y coordinate of the pixel

colour	byte	YUV colour value picked from default palette	

Example: Draw a green pixel at the coordinates on channel 0.

```
P42Display.SetYUVPixel ( CH0, 157, 079, 0x98 );
```

PrintChar ()

Print a character of the default character set, stored in SPI Flash, on the screen at the given coordinates. The character is always an 8x8 pixel area, even if the right most columns do not contain any positive pixels.

The default character set is described in chapter “Program FLASH with character bitmap”.

The colour can be picked from the default colour table in chapter “8-Bit Default Palette”.

```
void PrintChar (byte channel, char Letter, word x, word y, byte colour);
```

Value	Size	Description	
channel	byte	Channel # to be configured, can be from 0 to 3	
Letter	char	ASCII code of the character to print on screen	
x	word	x coordinate of the top left corner of the character	
y	word	y coordinate of the top left corner of the character	
colour	byte	YUV colour value picked from default palette	

Example: Draw a dark purple hashtag at the coordinates on channel 0.

```
P42Display.PrintChar ( CH0, '#', 0, 40, 0x23);
```

PrintString ()

Print a character string of the default character set, stored in SPI Flash, on the screen at the given coordinates. The characters are always an 8x8 pixel area (fixed width font), even if the right most columns do not contain any positive pixels.

The default character set is described in chapter “Program FLASH with character bitmap”.

The colour can be picked from the default colour table in chapter “8-Bit Default Palette”.

```
void PrintString (byte channel, char* Text, word x, word y, byte colour);
```

Value	Size	Description	
channel	byte	Channel # to be configured, can be from 0 to 3	
Text	char*	Pointer to the 1st character of the sting to print on screen	
x	word	x coordinate of the top left corner of the 1st character	
y	word	y coordinate of the top left corner of the1st character	
colour	byte	YUV colour value picked from default palette	

Example: Print a string at the coordinates in brown letters on channel 0.

```
P42Display.PrintChar ( CH0, 'Nasenbaer', 0, 40, 0xF4);
```

UARTDataToFlash ()

```
byte UARTDataToFlash ( u_int32 length, u_int32 mem_location);
```

Value	Size	Description
length	u_int32	Length of the data packet
mem_location	u_int32	Flash memory location of the image data

Example:

…

DisplayBMPFromFlash ()

Displays an image saved in the given Flash memory location. The pixel colour index is use as the video chip default palette index. No colour conversion is done at this stage.

Currently there is a limitation, that the x value must be a 32-bit boundary.

```
void DisplayBMPFromFlash (byte channel, u_int32 mem_location, u_int16 x, u_int16 y );
```

Value	Size	Description
channel	byte	Channel # to be configured, can be from 0 to 3
mem_location	u_int32	Flash memory location of the image data
x	word	x coordinate of the top left corner of the image (32-bit aligned)
y	word	y coordinate of the top left corner of the image

Example:

…

Library functions ImageFlashWrite.h/.cpp

f_SPImemdump ()

Print a number of bytes from the SPI Flash memory chip on the serial console.

```
void f_SPImemdump (unsigned long address, unsigned int bytes);
```

Value	Size	Description

Address	ulong	Start address of memory dump
Bytes	uint	Number of bytes to display

Example:

```
f_SPImemdump (0x3000, 32);
```

Output (typical header start of a BMP image file:

```
0x3000:  0x42 0x4D 0x0A 0x13 0x00 0x00 0x00 0x00 BM......;
0x3008:  0x00 0x00 0x36 0x04 0x00 0x00 0x28 0x00 ..6...(.;
0x3010:  0x00 0x00 0x34 0x00 0x00 0x00 0x49 0x00 ..4...I.;
0x3018:  0x00 0x00 0x01 0x00 0x08 0x00 0x00 0x00 ........;
```

f_DownloadImage ()

A BMP image is received over UART and written to the specified memory location. Currently only 256 colour BMPs make sense to write, because they are the only type that gets displayed correctly by the DisplayBMPFromFlash () routine.

The image needs to be sent in binary mode to not convert any characters into different sequences. TeraTerm4 send file option works really well, when Binary option is enabled.

```
void f_DownloadImage ( unsigned long memory_location );
```

Value	Size	Description
memory_location	ulong	Start address of memory to store the image data

Example:

…

f_DownloadConvertImage ()

A BMP image is received over UART, colour space converted and written to the specified memory location. The colour space of the original image is mapped to the 8-bit default palette shown in section xxx. Currently only 256 colour BMPs make sense to write, because they are the only type that gets displayed correctly by the DisplayBMPFromFlash () routine.

The image needs to be sent in binary mode to not convert any characters into different sequences. TeraTerm4 send file option works really well, when Binary option is enabled.

The colour conversion is basically a 3D distance optimization with all RGB values of the palette colours and the pixel colour to convert are mapped in a 3D space (R->x; G->y; B->z) and the shortest distance between pixel and respective palette colour are used as display colour.

```
void f_DownloadConvertImage ( unsigned long memory_location );
```

Value	Size	Description	
memory_location	ulong	Start address of memory to store the converted image data	

Example:

…

SPImemSectorErase ()

Erase a 256 byte memory in the Flash memory. Start address must be 256 byte boundary aligned.

```
void SPImemSectorErase ( unsigned long mem_addr );
```

Value	Size	Description	
mem_addr	ulong	Start address of memory to be erased (must be a 256 byte boundary)	

Example:

YUV Palette

Without a working RGB to YUV conversion yet, the easiest way is to pick the 8bit YUV colour value from the following default palette colour table:

8-Bit Default Palette

H\L	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x																
1x																
2x																
3x																
4x																
5x																
6x																
7x																
8x																
9x																
Ax																
Bx																
Cx																
Dx																
Ex																
Fx																

NTSC/PAL Color Conversion Tool

Unfortunately not available yet.

Video Signal Information

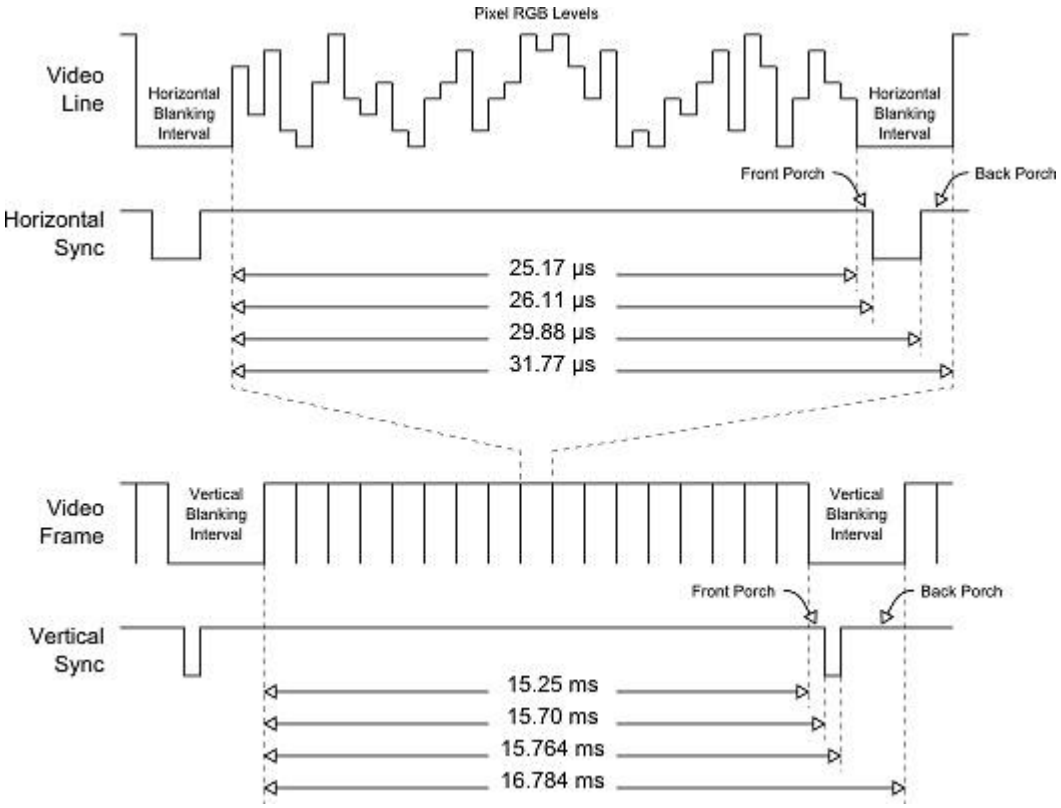
Timings for 640x480:
http://tinyvga.com/

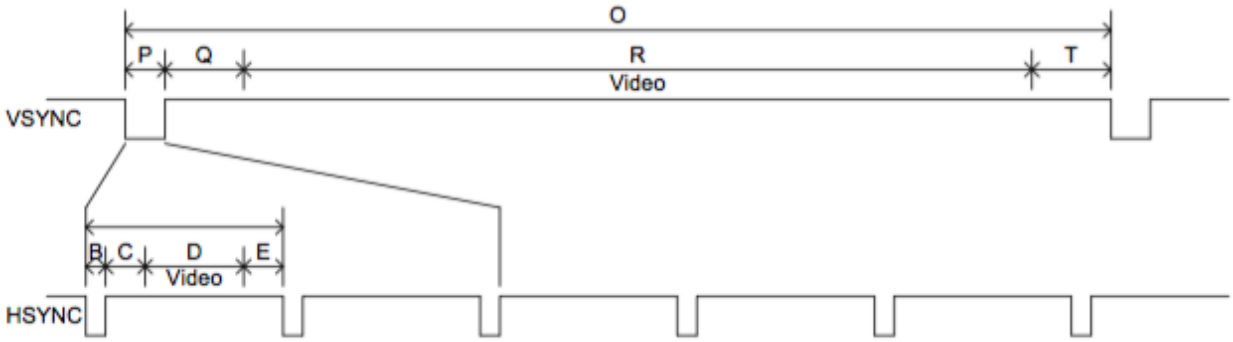
http://www.microvga.com/
Mit einem FPGA einen alten Laptop Schirm ansteuern
https://drive.google.com/file/d/1KpEgE7tbPQhqqmzTtySVD6Gch_TDvQic/view

https://hackaday.com/2015/10/15/spit-out-vga-with-non-programmable-logic-chips/
https://hackaday.io/project/9782-nes-zapper-video-synth-theremin/log/32271-vga-sync-generation
VGA controller in VHDL
http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf

No guarantee for the correctness of the websites listed here.

Frame Timing:





Clock	26.25	MHz	
HSYNC (Measured in Clocks)			
A	800	30.48	Us
B	96	3.66	uS
C	48	1.83	uS
D	640	24.38	uS
E	16	0.61	uS
VSYNC (Measured in HSYNC's)			
O	525	16.00	mS
P	2	0.06	mS
Q	33	1.01	mS
R	480	14.63	mS
S	10	0.30	mS

This is a living document. Any missing content will be added as appropriate.

Revision Control

Version	Data	Changes
1.0	22. May 2021	Initial Madman Chicken-scratch Manifesto
1.1	29. July 2024	Added GPIO content
2.0	17. Jan 2026	Board Version 2.0