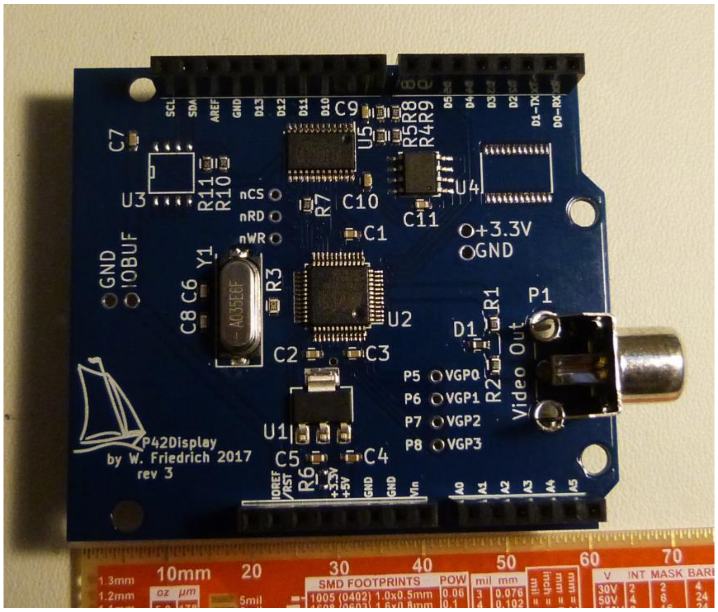


Arduino Video Display Shield Rev3

Designed by Wolfgang Friedrich

https://hackaday.io/project/21097-arduino-video-display-shield

https://www.tindie.com/products/Wolfi/arduino-video-display-shield/

Table of Contents

Introduction

The Video Display Shield is an expansion board for the Arduino Platform.

This board provides an analog composite video display interface with integrated frame buffer memory accessible through SPI. 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 3.579545MHz crystal or as PAL version with 4.43618MHz 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 VS23S010D-L chip, which is able to output composite video with resolutions from 320x200 in 65536 colours to 720x576 in 4 colours. The chip has a 1Mbit framebuffer, 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 2Mbit SPI FLASH memory is available on-board. It is pre-loaded with a character bitmap consisting of 94 characters (ASCII code 33-126). A SOIC-8 footprint for a I2C EEPROM is also on board, but not populated.

The board uses the Arduino IOREF voltage to translate between 3.3V on the shield side and the respective IO voltage on the Arduino side. So this shield works together with UNO, MEGA and also with any 3.3V system that uses the Arduino form factor and pinout, without modifications.

Specifications:

- Operating supply voltage 4.5V - 20V
- Board IO voltage (IOREF) 1.5V - 5.5V
- Composite Video Output
- Maximum resolution 720x576 in 4 colours
- Implemented resolutions: NTSC 320x200 with 256 colours and PAL 300x240 with 256 colours
- Crystal: NTSC 3.579545MHz or PAL 4.43618MHz
- Communication interface: SPI up to 38MHz
- Video Frame Buffer: 1Mbit = 128KByte
- Flash: 2Mbit = 256KByte
- EEPROM: up to 2Mbit (optional, not populated)
- IO connectors Arduino compatible
- Size: 61.3mm x 53.4mm (2.4" x 2.1")

Hardware

The Video Display Shield is designed for the Arduino Uno, but also runs on the Mega (tested) and should run with the Due (not tested).

A 2Mbit SPI FLASH memory is available, preloaded with character bitmap data.

Use different SPI chip select pins

If more SPI devices are used, the use of the slave select signals needs to be coordinated. This shield uses Arduino pins 10 for the controller chip select and 9 for the Flash chip select. Modifications by cutting traces and adding wires should only be performed when you know what you are doing.

Use with MEGA2560 or Due

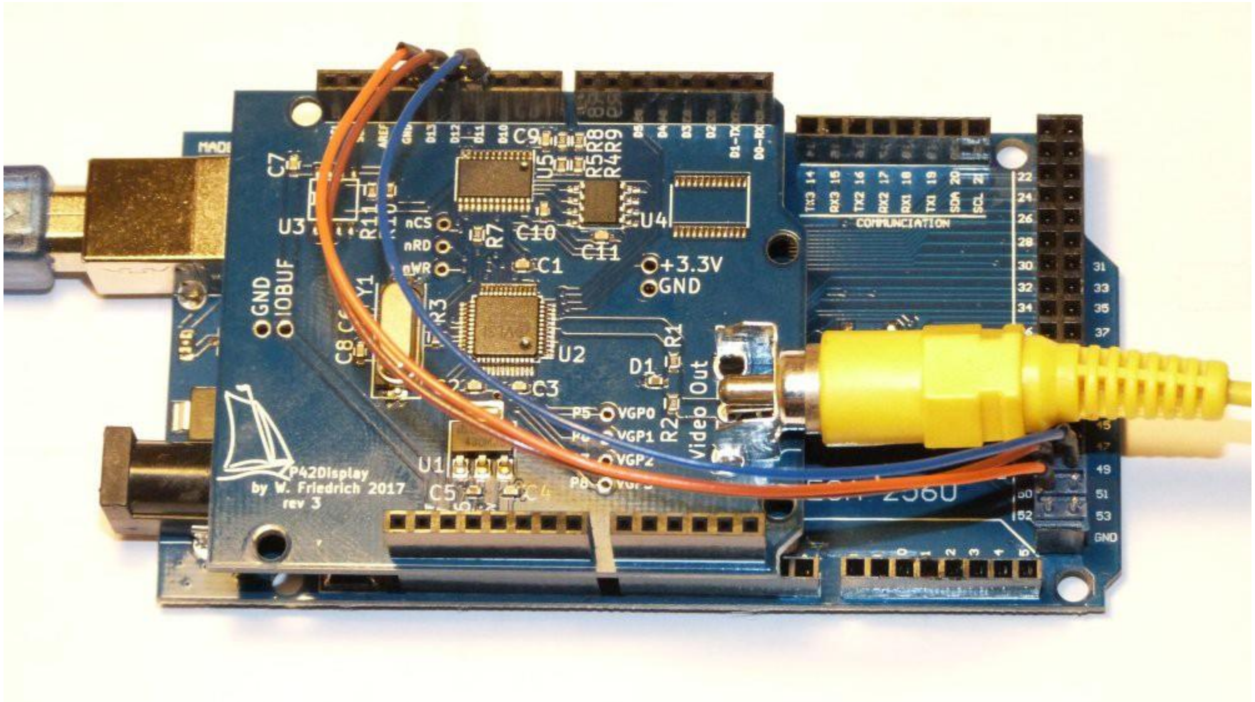
For the MEGA, connect the SPI communication signals through jumper wires to the shield.

SCK: Mega pin 52 -> Uno shield pin 13 - orange wire

MOSI: Mega pin 51 -> Uno shield pin 11 - blue wire

MISO: Mega pin 50 -> Uno shield pin 12 - brown wire

The slave select pin does not need to be hand-wired, as it is mapped as a normal GPIO in a normal Uno sketch. The picture for the pin numbers 50-51 on the Mega connector is misleading due to the parallax.



(The RCA connector is not in an ideal location for the wire jumpers. Here is a reason for another spin).

Make pins 11 to 13 on the Mega inputs or tri-stated outputs.

Add the define somewhere in the header file for the shield or at the beginning of the sketch

```
#define MEGA
```

Add the code to disable the pins that are used by Uno for SPI in the setup() function

```
#ifdef MEGA
pinMode(11, INPUT);
pinMode(12, INPUT);
pinMode(13, INPUT);
#endif
```

This is all what is needed to connect the MEGA to the Video Display shield.

Connection to the Due would be similar, using the SCK, MOSI and MISO signals from the SPI connector.

SPCK: Due SPI pin 3 -> Uno shield pin 13

MOSI: Due SPI pin 4 -> Uno shield pin 11

MISO: Due SPI pin 1 -> Uno shield pin 12

Unfortunately the SPI connector is hidden under the shield when it is plugged in. Additional hardware modifications to the Arduino Due or a special adapter would be required for a reliable connection.

Flash Memory

Memory Map

By default the 2Mibit (256KiByte) 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 - 262144	0x03000-0x3FFFF	empty

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	

Control Functions

Resolutions

Currently implemented are

PAL 300x240 8bit

NTSC320x200 8bit

How to select an implemented resolution, please see chapter “Config()” on page 8. Other resolutions will be available in the future.

Possible Resolutions are (copied from the VLSI VS23S010D-L 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”

P42Display()

Board and SPI interface configuration.

Pin	Pin Nr	Direction	Default	Description
MVBLK	6	In	X	Ready signal after Block Move Command
nWP	7	Out	High	Write Protect for Flash Memory and Video Controller
nHOLD	8	Out	High	Hold signal for SPI interface
nMemSelect	9	Out	High	SPI Select for Flash Memory
nSlaveSelect	10	Out	High	SPI Select for Video Controller

Config()

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

The resolution is set in the VS23S010D-L.h file by removing the comment of the desired resolution:

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

Here NTSC320x200 is selected.

Also the MEGA selection is done in the same file just below the resolution.

```
// *** Uncomment this line if target system is an Arduino MEGA ***
#define MEGA
```

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 (unsigned long address);
```

Value	Size	Description
address	unsigned long	video buffer memory address
return value	byte	Result of the read command

Example: Read address 0

```
Byte0 = SPIReadByte (0x00000000 );
```

SPIReadWord ()

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

```
word SPIReadByte (unsigned long address);
```

Value	Size	Description
address	unsigned long	video buffer memory address
return value	word	Result of the read command

Example: Read address 0

```
Word0 = SPIReadWord ( 0x00000000 );
```

SPIWriteByte ()

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

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

Value	Size	Description
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

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

SPIWriteWord ()

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

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

Value	Size	Description
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 memory (everything not only the picture data area!)

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

Graphics commands

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 colour );
```

Value	Size	Description
colour	byte	YUV colour value picked from default palette

Example: Clear screen and set to a light blue colour.

```
P42Display.ClearScreen ( 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 (u_int16 x1, u_int16 y1, u_int16 x2, u_int16 y2, u_int16 color);
```

Value	Size	Description
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.

```
P42Display.FilledRectangle ( 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 (word x, word y, unsigned long colour);
```

Value	Size	Description
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.

```
P42Display.SetRGBPixel ( 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 (word x, word y, byte colour);
```

Value	Size	Description
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.

```
P42Display.SetYUVPixel ( 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 (char Letter, word x, word y, byte colour);
```

Value	Size	Description
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.

```
P42Display.PrintChar ('#', 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 (char* Text, word x, word y, byte colour);
```

Value	Size	Description
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: Draw a brown string at the coordinates.

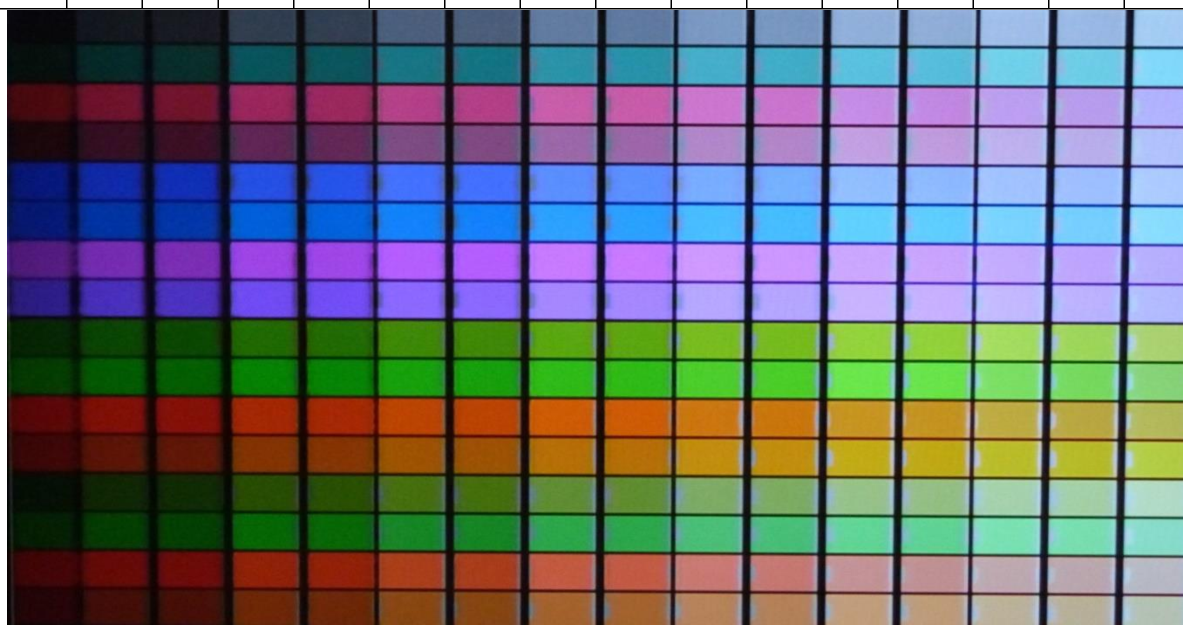
```
P42Display.PrintChar ('Nasenbaer', 0, 40, 0xF4);
```

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.

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

Revision Control

Version	Data	Changes
1.0	9. March 2018	Initial Version
1.1	26. March 2018	Added Flash Memory Map Fixed wrong connector pin description for Arduino Mega/Due use.