Belegarbeit für das Modul Embedded Systems

Fachbereich 1
Studiengang IKT Master

VGA Videoausgabe mit dem rp2040

Student
Sidney Göhler
544131

Lehrbeauftragter
Raymond Horn

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Mittwoch, 27. September 2023

Einleitung

Für das Modul Embedded Systems an der Hochschule für Technik und Wirtschaft entwickeln die Studenten ein eingebettetes System, welches eine Videoausgabe auf einem Monitaor erzeugen soll.

Hierfür gibt es verschiedene Videoausgabe Standarts, wobei diese häufig mittlerweile nur eine grobe Leitlinie darstellen. Bekannte, aber teilweise überholte Standarts sind z.B. PAL, NTSC, XGA und VGA.

Moderne Standarts wie z.B. HD und WQXGA bieten dabei meistens mindestens die doppelte Pixeldichte, sowie einen umfangreicheren Farbraum.

Das Ende der Fahnenstange sind derzeit Videoausgabe Formate mit über 130Mio. Pixeln (16K), wobei es hier derzeit noch keine einheitlichen Bezeichnungen gibt.

Mit diesem Projekt möchte ich zum Einstieg in die Thematik eine möglichst kostengünstige und wenig komplexe Lösung entwickeln.

Hierfür eignet sich besonders der VGA Videostandart, da dieser historisch bedingt relativ tolerant ist was die *timing* Werte angeht und generell bis Heute von vielen Monitoren unterstützt wird.

Als Microcontroller verwende ich den rp2040 welcher in den letzten Jahren von der Raspberry Pi Foundation als erster eigenst entwickelter Microcontroller vorgestellt wurde. Der rp2040 basiert auf einem ARM Cortex-M0+, welcher neben den zwei Prozessorkernen auch einige Perephirie mit sich bringt.

Der Cortex-M0+ ist in einer Von-Neumann-Architektur gebaut worde, also mit einem gemeinsamen Daten- und Befehlsspeicher, was den Vorteil von streng sequentiellen Programmabläufen hat, mit dem Nachteil, dass der Datenbus meist einen Flaschenhals bildet, da sowohl Daten als auch Befehle über den selben Bus aus dem selben Speicher geladen werden.

Kompensiert wird dies durch die recht hohe Taktfrequenz von 133MHz im nicht-übertakteten Zustand und zusätzliche Koprozessoren, wie z.B. der DMA Controller oder die rp2040 spezifischen Programmable Input/Output (PIO) *state machines*.

Mit 264kB SRAM steht aber leider nicht wirklich viel Speicher zur Verfügung, wewegen bei einer gewünschten Farbauflösung von 8-bit keine echten 640x480 Bidinhalte gespeichert werden können, das allein für ein Bild schon ca. 245kB nötig wären.

Demzufolge werden die Bildinhalte mit einer niedrigeren Auflösung berechnet und für die Bildschirmausgabe auf 640x480 Pixel hochskaliert.

Contents

Grundlagen

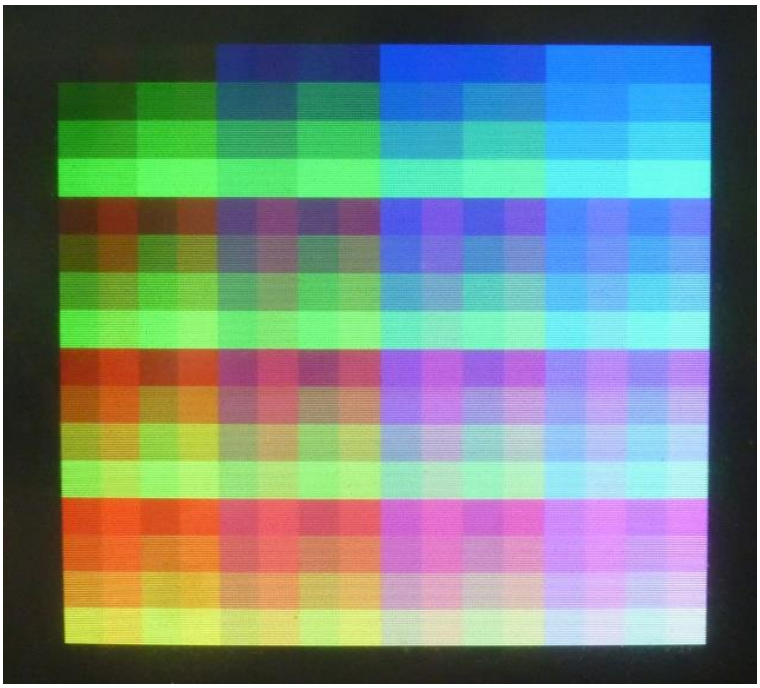
VGA

Video Graphics Array, kurz VGA, ist ein älterer analoger Videostandard, welcher weit verbreitet war, um Videoausgabegeräte und Monitore zu verbinden, bevor digitale Videoschnittstellen wie HDMI und DisplayPort aufkamen. VGA wurde in den späten 1980er Jahren von IBM entwickelt und wurde zum de-facto-Standard für die Übertragung von Bildinformationen.

VGA verwendet analoge Signale, um Videoinformationen zu übertragen, was bedeutet, dass das Videosignal kontinuierlich ist und im Spannungsniveau variiert, um verschiedene Farben und Helligkeitsstufen darzustellen. Dies steht im Gegensatz zu digitalen Übetragungsverfahren wie bspw. DVI oder HDMI, welche ausschließich diskrete Werte verwenden.

Diese Variierung der Spannungspegel geschiet über einWiederstandsnetzwerk, welches effektiv somit als DAC fungiert.

Es wird ausschließlich das RGB farbmodell verwendet, wobei die Anzahl der verfügbaren Farben daraus resultiert, wie viele diskrete Spannungspegel je Farbe über das Wiederstandsnetzwerk realisiert werden können.

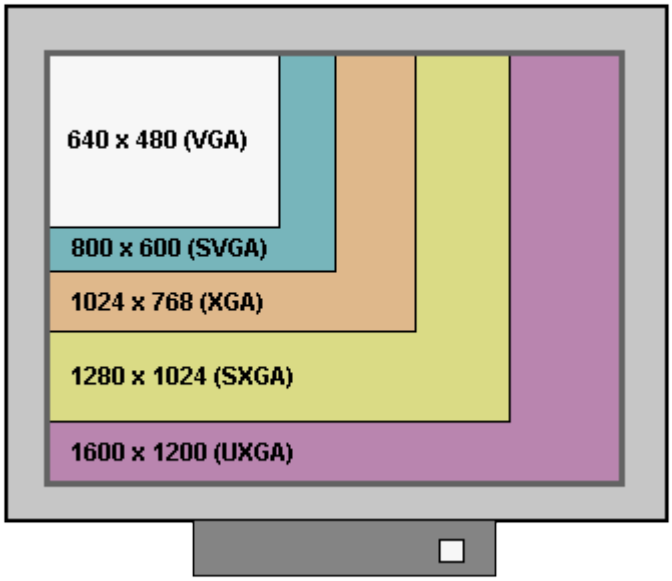


Farbpallette RRRGGGBB

Der maximale Helligkeitspegel ist dabei mit ~0,7V erreicht.

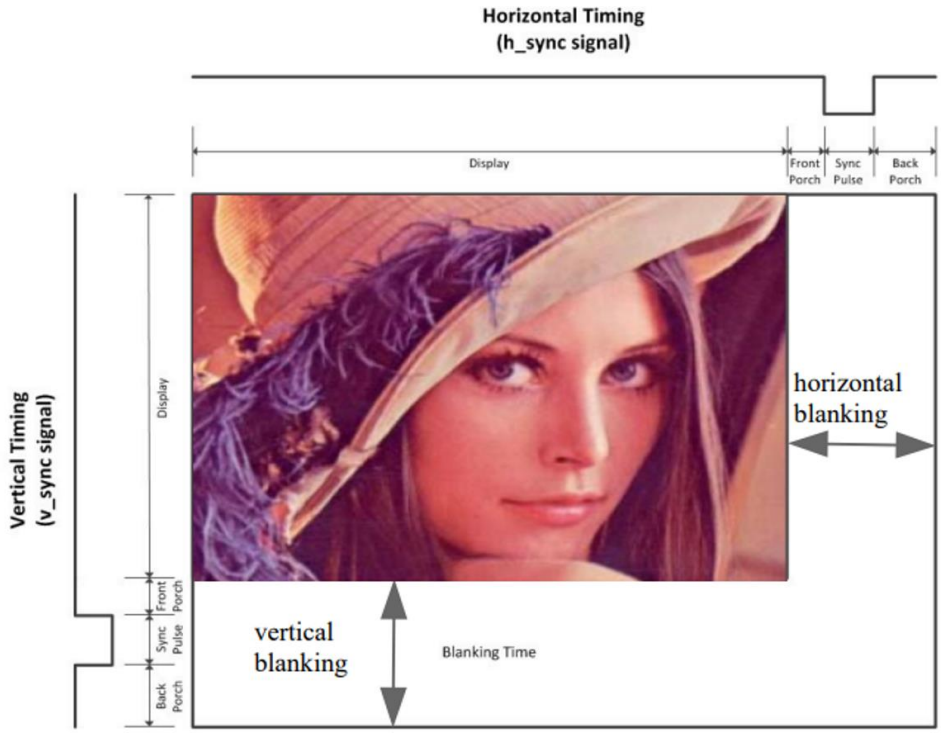
Der Abschlußwiderstand im Monitor hilft, mögliche Reflexionen durch die unangepasste Senke zur Quelle zu reduzieren und beträgt typischerweise 75 Ohm.

VGA unterstützt verschiedene Bildschirmauflösungen und Bildwiederholfrequenzen, wobei häufige VGA-Auflösungen 640x480 (VGA), 800x600 (SVGA) und 1024x768 (XGA) sind. Die Bildwiederholfrequenz liegt normalerweise zwischen 60 Hz und 75 Hz.



Neben den RGB-Signalen enthält VGA auch zwei Synchronisationssignale: horizontale Synchronisation (HSync) und vertikale Synchronisation (VSync). Diese beiden Signale dienen dazu, den Scanvorgang des Displays zu synchronisieren.

HSync gibt den Beginn jeder horizontalen Zeile an, während VSync den Beginn eines neuen Bildes signalisiert.



Interrupts

Unter einem Interrupt Request (IRQ) versteht man eine kurzfristige Unterbrechung der regulären Programmausführung, um einen in der Regel kurzen, aber häufig auch zeitkritischen Vorgang, abzuarbeiten.

Sie sind ein grundlegender Bestandteil in der Programmierung und insbesondere in der Programmierung von Microcontrollern, da sie es der häufig begrenzt leistungsfähigen CPU ermöglichen, auf Ereignisse zu reagieren, ohne ständig darauf zu warten oder sie zu Überwachen (polling). Dies führt häufig auch zu einem stark reduziertem Energiebedarf des Systems, da die Hardware bspw. im laufenden Betrieb nur dann rechnen muss, wenn ein Interrupt vorliegt.

Der rp2040 implementiert mehrere Arten von Interrupts:

- **Systick-Interrupt:** Da der rp2040 auf dem ARM Cortex-M0+ Kern basiert, kennt er auch diese Art von Interrupt. Ein Systick ist im Prinzip ein im Hintergrund laufender Timer, welcher in periodischen Abständen einen Interrupt auslösen kann, wenn der Microcontroller entsprechend Konfiguriert ist.

- **Peripherie-Interrupts:** Der rp2040 verfügt über zahlreiche Peripheriegeräte wie GPIO-Pins, UART, SPI, I2C und mehr. Diese Peripheriegeräte können Interrupts generieren, wenn bestimmte Ereignisse eintreten, wie das Empfangen von Daten an einem UART-Port oder eine Änderung an einem GPIO.

Anzumerken ist noch, das Interrupts in der Regel in einer gesonderten Routine/Funktion verarbeitet, auch Interrupt Service Routine (ISR) oder Interrupt Handler genannt. Besonders im Bereich der eingebetteten Systeme findet man häufiger Software, dessen Funktionalität, fast ausschließlich durch ISRs abgebildet wird, die Hauptroutine ist dabei im Extremfall nur noch für die Initialisierung zuständig.

Pulsweitenmodulation

Pulsweitenmodulation (PWM) ist ein Modulationsverfahren, welches in der Digitaltechnik verwendet wird, um analoge Signale mit einem digitalen Gerät wie dem rp2040-Mikrocontroller zu erzeugen.

Generell wird dabei eine elektrische Spannung mit einer konstanten Frequenz alterniert, wobei das zeitliche Verhältnis zwischen den beiden Spannungen als Tastgrad bezeichnet wird. Diese Tastverhältnis bestimmt dabei die durchschnittliche Ausgangspannung.

Der rp2040 hat insgesammt 16 kontrollierbare PWM Ausgänge, unterteilt in 8 sog. *Slices* mit jeweils 2 Kanälen. Durch die Doppelbelegung mancher PWM Kanäle auf 2 GPIO Pins, können prizipiell alle 30 GPIO Pins mit dem PWM Modul angesteuert werden.

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B		

DMA

DMA steht für Direct Memory Access und beizeichnet zunächst einmal den direkten Zugriff auf den Speicher ohne Umwege über die CPU. Der DMA Controller ist somit eine spezialisierte Hardwarekomponente, oder Subsystem in einem Computer, welche eine effizienten Datenübertragung zwischen Peripheriegerät und Speicher herstellt, wobei der Controller häufig mit mehreren Kanälen arbeitet, um seperate Datenübetragungen zu verwalten.

Schlussendlich ist das Ziel, die CPU soweit zu entlasten, dass sie andere Aufgaben bearbeiten kann.

Der DMA-Controller des RP2040 verfügt über 16 Kanäle, von denen jeder eine separate Datenübertragung verwalten kann. Diese Kanäle können vom Programmierer konfiguriert werden, um verschiedene Anforderungen an Datenübertragungen gleichzeitig zu bewältigen.

Es müssen unter anderem Start- und Zieladresse, die Datenwortlänge, sowie die Art und Weise wie der Kanal getriggert werden soll.

Schlussendlich ist es Möglich mehrere DMA Kanäle zu verketten, um bspw. den einen DMA Kanal nach der Datenübertragung neu zu konfigurieren. Diese Verkettung ermöglicht komplexe Programme, welche nicht von der CPU gesteuert werden.

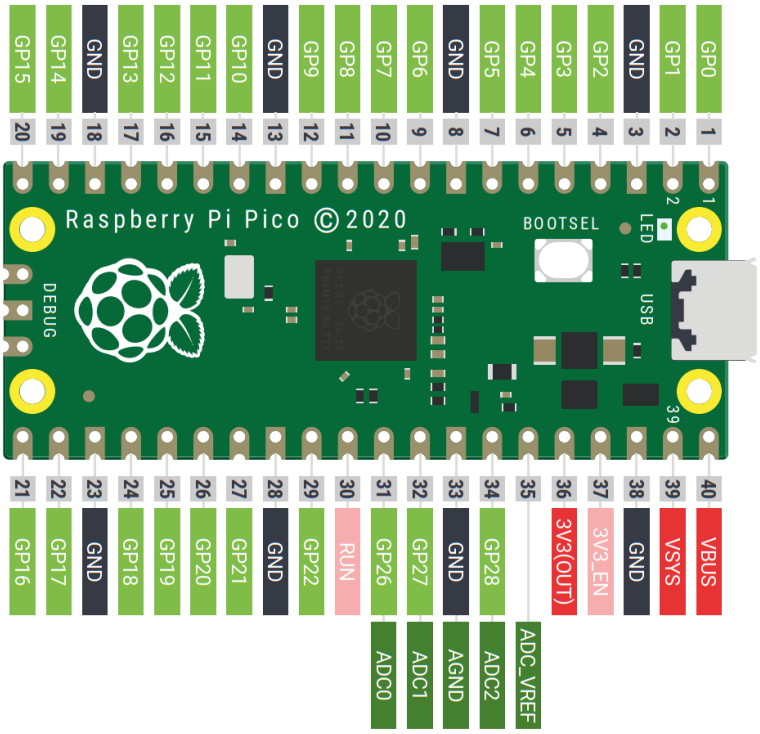
PIO

Der Programable I/O Koprozessor (PIO) stellt eine sehr flexible Schnittstelle zwischen den GPIOs und dem restlichen rp2040 dar. Es handelt sich dabei um einen rp2040 spezifische Hardwareblock, welcher eine Vielzahl an Protokollen und Übertragungsstandarts implementieren kann.

- **Parallele Verarbeitung:** Der rp2040 verfügt über zwei PIO-Zustandsmaschinen, welche Anweisungen parallel zur CPU. Diese Zustandsmaschinen sind so konzipiert, dass sie verschiedene I/O-Aufgaben ohne direktes Eingreifen des Hauptprozessors verarbeiten können, wobei sie unabhängig voneinander arbeiten.
- **Befehlssatz:** Der PIO Koprozessor verfügt über seinen eigenen Befehlssatz, der sich vom Befehlssatz des ARM Cortex-M0+ unterscheidet. Dieser Befehlssatz ist optimiert für die Durchführung von I/O-Operationen, einschließlich GPIO-Manipulation, SPI, I2C, UART und VGA!
- **Ausführen von Anweisungen:** Die beiden PIO instanzen verfügen jeweils über 4 unabhängige Zustandsmschinen, wobei anzumerken ist, dass diese sie zwar unabhängig voneinaner rechnen können, sie sich jedoch einen Befehlsspeicher mit maximal 32 Befehlen teilen.
  Die Syntax der eigens entwickelten Assembler nahen Sprache, ermöglicht aber teilweise, mehrere Funktionen mit nur einem Befehl zu bewirken.
- **Datenübertragung:** Der PIO-Coprozessor kann Daten von verschiedenen Peripheriegeräten und GPIO-Pins lesen und schreiben, wobei gleichzeitig jede Zustandsmaschine mit 2 FIFO Queues ausgestattet sind, wodurch eine synchronisation mit der restlichen internen Hardware, ermöglicht wird.

Praktische Umsetzung

Pin Out



Pinout für die VGA Bildausgabe durch PWM Taktung und Bit-Banging:

- GPIO0: uart tx
- GPIO1: uart rx
- GPIO2: VSYNC
- GPIO4: HSYNC
- GPIO6-8: ROT
- GPIO9-10: GRÜN
- GPIO11-13: BLAU
- GPIO 16: BUTTON 1
- GPIO 17: BUTTON 2

VGA Bildausgabe durch PWM Taktung und Bit-Banging

Aufgrund der Tastsache, dass der VGA Standart zu einer Zeit aufkam, als die meisten VGA Monitore noch eine Kathodenstrahlröhre besaßen, waren die Totzeiten, in denen kein Signal auf den Bildschirm projeziert wurde, länger als die Laufzeiten moderner Digitaltechnik es ermöglichen.

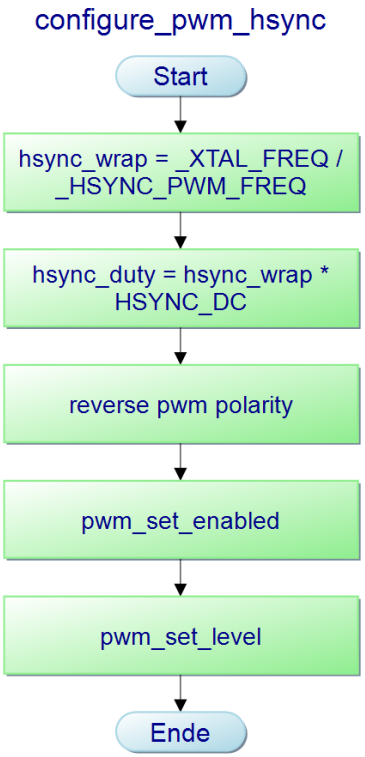
Dies hat in der Theorie den Vorteil, das unser Programm die Clockcycle der beiden Sync Signale nicht unbedingt Perfekt realisieren muss, damit ein Bild auf dem Monitor erscheint.

Zunächst wollen wir uns um die SYNC Singale kümmern. Eine Möglichkeit wäre es, einen Timer einzurichten, welcher alle 40ns einen Interrupt auslöst und eine Zähler Variable hochzählt. Ab dem Wert 519 würden wir einen GPIO, welchen wir vorher auf 1 gesetzt hatten, nun auf 0 setzen. Nach zwei weiteren Interrupts würden wir den GPIO wieder auf 1 setzen und einen weiteren Zähler für den VSYNC hochzählen.

Das Problem an dem Ansatz ist leider, dass die kürzeste Zeit, die ein Timer im rp2040 laufen darf, 1ms beträgt. Somit ist dieser Ansatz vom Tisch.

Ein weitere Ansatz, ist einen PWM Block im rp2040 so zu Kofigurieren, dass der PWM Block mit einer Trägerfrequenz von 31250Hz läuft und dabei eine Duty Cycle von $\left(\frac{96}{800}\right) * 100 = 12\%$ erzeugt.

Um den PWM Block so zu Konfigurieren, wird zunächst der *wrap* bestimmt, welcher dem Zähler im PWM Block mitteilt, ab welchem Wert er sich zurücksetzen soll. Darauf ergiebt sich effektiv der Takt.



Der *wrap* berechnet sich durch den Quotienten aus System Takt und gewünschte PWM Trägerfrequenz. Für sehr langsame Taktfrequenzen muss zusätzlich noch ein Clock divider Parameter gesetzt werden.

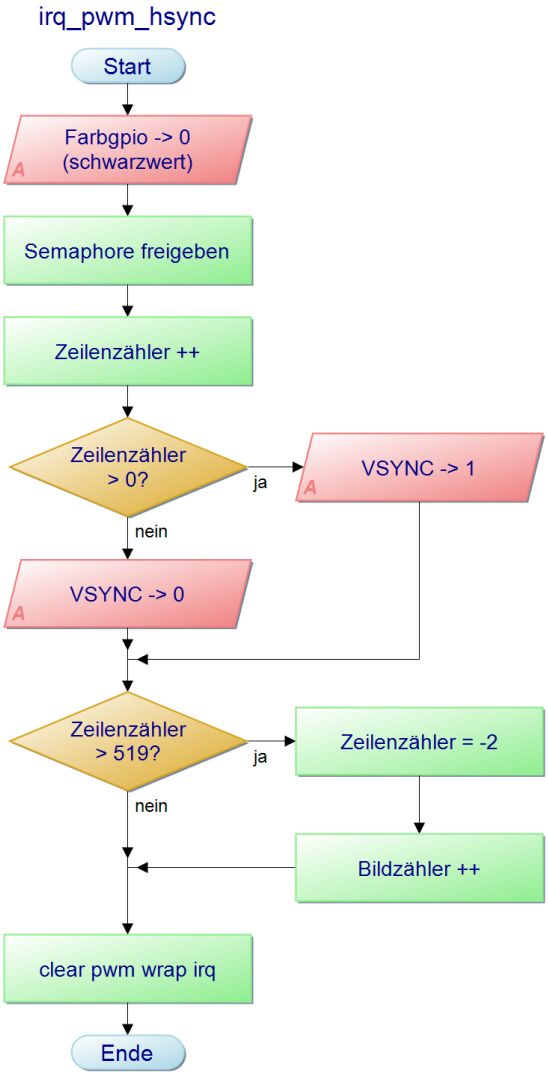
Der duty cycly Parameter ergiebt sich aus dem Produkt vom *wrap* und der gewünschten duty cycly in der Form $(1 - duty\ cycly)$

Da wir die meiste Zeit eine 1 und nur einen kurzen burst 0 für das HSYNC benötigen, drehen wir die polarity um.

Nachdem der PWM Block Initialisiert wurde, setzten wir einmalig den level (duty cycle).

Der Interrupt Handler für den PWM Block setzt zunächst alle Farbbits auf 0, da der VGA Standart in diesem Zeitraum eine Schwarzwertanpassung vorsieht.

Sollten die Farbbits in diesem Zeitraum nicht auf 0 gesetzt werden, ist auf dem Monitor im Nachinein, kein Bild zu sehen.



Im Hauptprogramm wird in der endlosschleife zur synchronisation auf die Freigabe einer Semaphore gewartet, welche im nächste Schritt des Interrupt handlers gegebn wird.

Nachfolgend wird der Zeilenzähler hochgezählt, welcher das VSYNC Signal im Prinzip simuliert.

Hat dieser Zähler den Wert 0 überschritten wird das VSYNC Signal auf 1 gesetzt.

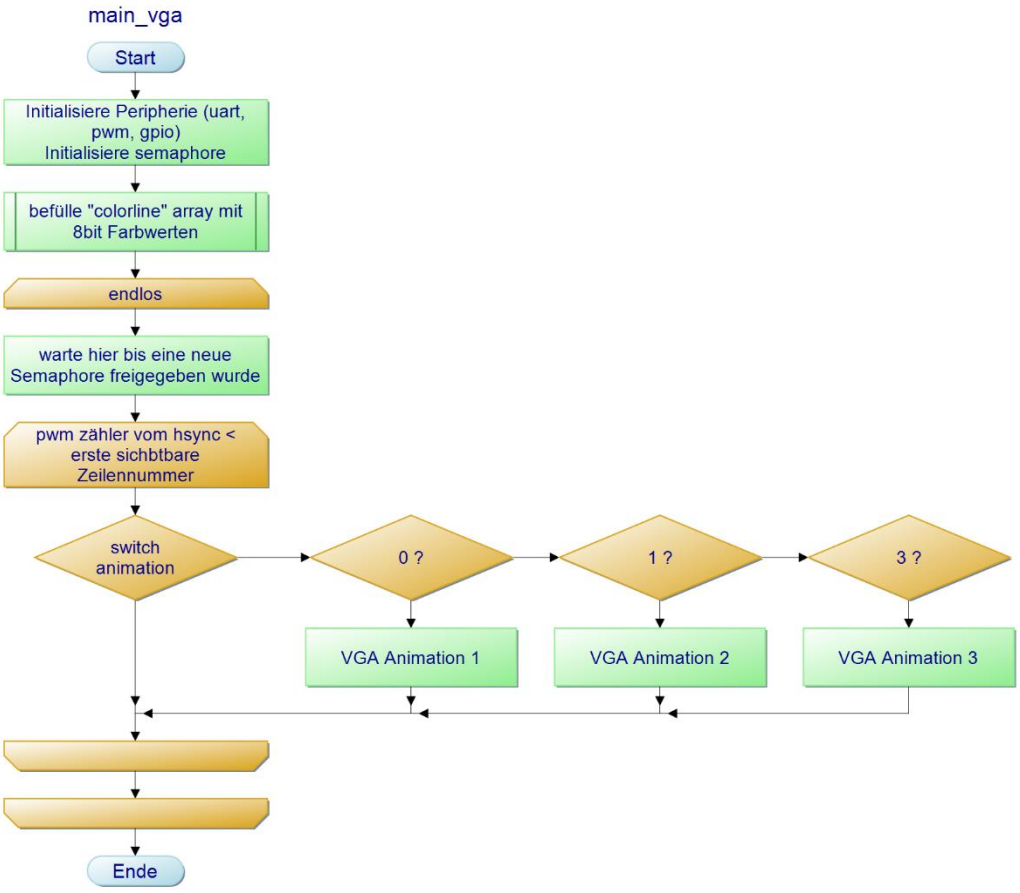
Haben wir beim VSYNC Zähler das Ende erreicht setzen wir diesen zurück.

Wir starten im Prinzip zuerst mit dem Puls, gefolgt von der Front Porch, Back Porch und abschließend Display Time.

An der Stelle an der abgefragt wird, ob der Zeilenzähler größer 0 ist nutzen wir die Tatsache aus, dass wir ein Statusbit in der CPU nutzen können und somit eine Verzweigung sparen können.

Generell bietet es sich an möglichst auf Verzweigungen im Interrupt handler zu verzichten, da diese häufig zu einer vielzahl an Sprüngen im Stack führen kann, was schlicht unnötig viel Zeit kosten kann.

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48



Im Hauptprogramm initialisieren wir zunächst die Peripherie, worunter uart zum debuggen, der PWM Block für das HSYNC und generell die GPIOS zählen.

Wie am Anfang gezeigt, sind zwei Taster am rp2040 angeschlossen, diese Taster zählen zwei Variablen hoch, welche für die Animationen genutzt werden.

Nach der Initialisierung wird eine Matrix mit Pixelinformationen gefüllt, welche für eine Animation genutzt wird.

In der Endlosschleife wird zunächst zur Synchronisation auf die Freigabe einer Semaphore gewartet. Wie erwähnt wir diese mit jeder Zeile freigegeben.

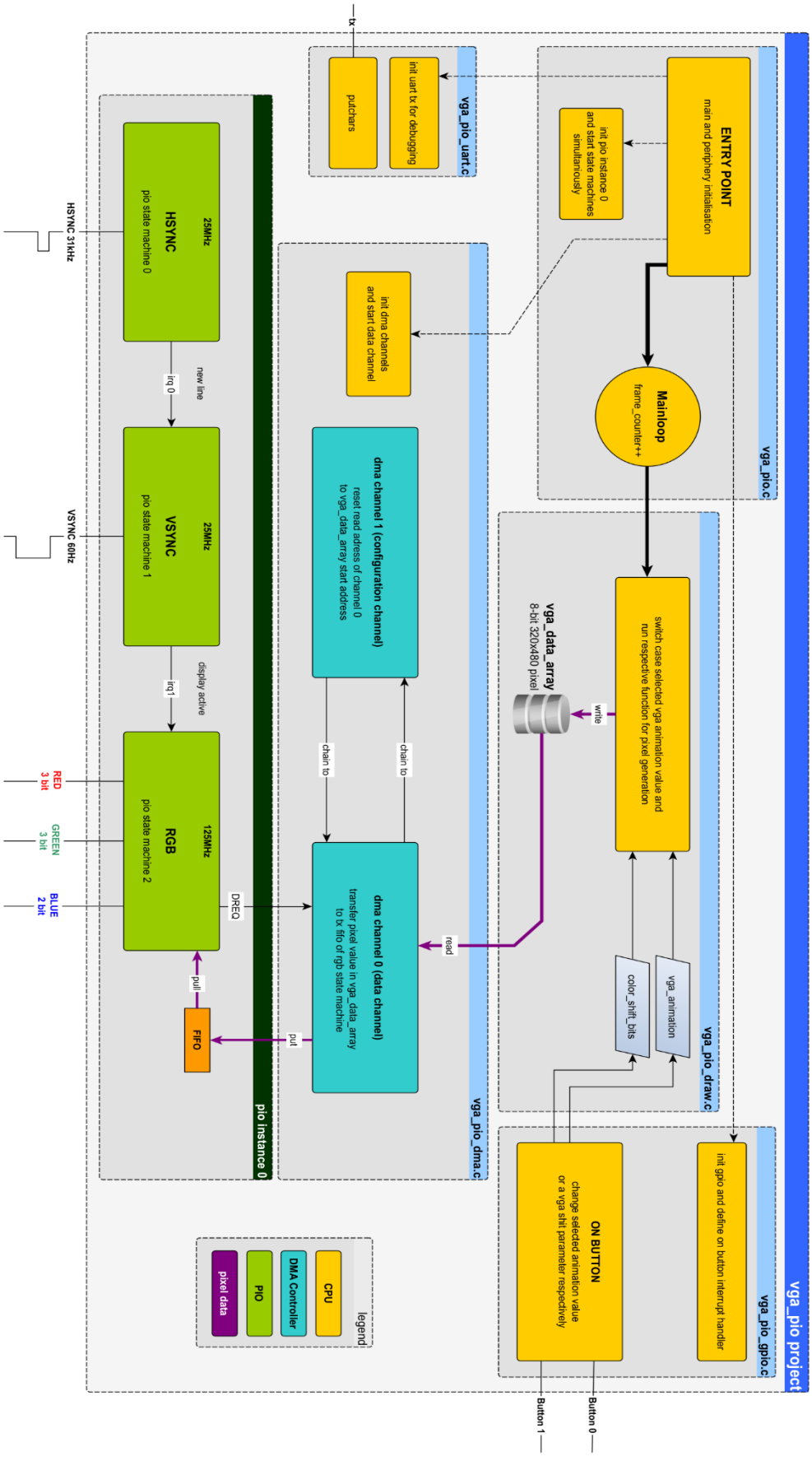
Aus Timinggründen wird ca. 3.8µs (96 Clock cylces oder ~600 Zählschritte) gewartet um nicht zu früh ein Bildsignal an die GPIOs zu senden.

Code

Der komplette Source-Code ist hier zu finden:

https://github.com/j4yj03/embedded_systems_rp_pico/tree/main/embedded/vga_pwm

VGA Bildausgabe mit PIO Zustandsmaschinen und DMA



Man hat bei der Variante mit dem PWM Block zwar gemerkt, dass es gehen kann, doch war es teilweise recht mühsam die einzelnen Komponenten so zu timen, dass ein sauberes VGA Bild auf dem Monitor zu sehen war, allein das ändern eines einzigen Datentypes konnte schon teilweise Probleme bereiten. Schlussendlich hat es aber geklappt.

Die zweite Variante macht sich den DMA Controller und ein PIO Inztanz zu nutze, um die CPU so zu entlasten, dass sie sich ausschließlich auf das berechnen der Pixeldaten konzentrieren kann.

Das wirklich besondere am rp2040 sind die Zustandmsachienen, welche sich mit den PIO Inztanzen programmieren lassen.

Während der Initialisierung der Zustandsmschienen werden diese mit den timing Werten beladen, da sie intern nur mit 5-bit rechnen. Dennoch ist es ihnen Möglich schlussendlich 8-bit in einem Takzyklus auf die GPIOs zu shiften.

Um mit den Timing Werten aus der oberen Tabelle arbeiten zu können, müssen die HSYNC und VSYNC Zustandsmaschienen mit 25MHz getaktet werden, gleichzeitig synchronisieren sie sich aber auch zusätzlich über gesetzte Interrupt Flags.

Die Zustandsmaschiene, welche für die RGB Ausgabe zuständig ist, wartet, bis der VSYNC Zähler seinen Interrupt gesetzt hat und shiftet anschließend 8 bits je Pixel aus dem FIFO Buffer and die GPIOs. Der FIFO Buffer wird konstant vom DMA Controller befüllt, wobei der zuständige DMA Kanal auf ein *Data Request Signal* von der RGB Zunstandsmaschiene wartet.

Die Pixeldaten entspringen alle aus einer 480x320 8-bit Array, welche die gesammten Pixeldaten beinhaltet. Die Startadresse dieses Arrays wurde während der Initialisierung gespeichert und mithilfe eines zweiten DMA Kanals an den ersten übermittelt, dieser nutzt diese Startadresse um erneut die Pixeldaten sukzessiv in den FIFO Buffer zu schreiben.

Nach der Initialisierung ist die CPU nur noch damit beschäftigt, das Pixeldaten Array zu befüllen und einen eventuellen Button Interrupt zu behandeln.

Die Buttons sind wie in der erster Variante dafür Zuständig, die Animation zu wechseln, bzw. einen Parameter zu setzen, welche die Farbbits während der Animation nach recht shiftet, was den Effekt hat, dass jede Animation, bei dies nutzt, grobflächiger wird, da eine shitft n nach rechts eine division durch 2^n

Zur Folge hat, was wiederum den Wertebereich mehr in den „sichtbaren Bereich“ bringt, wodurch mehr Flächen entstehen.

Im Grunde habe ich versucht, die selber Algorithmen für die Bildgeneration zu verwenden wie in der ersten Variante.

Da ich nicht direkt auf den Zählerwert der HSYNC Zustandsmaschiene zugreifen konnte, habe ich andere Zählervariablen genutzt, was teilweise zu ähnlichen Ergebnissen geführt hat.

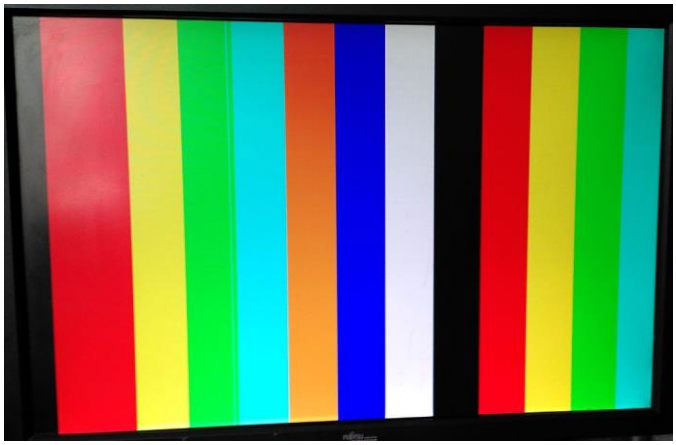
Eventuell war es aber auch der trägheit der ersten Variante geschuldet, dass dort andere, teilweise interessantere Animationen zum Vorschein kamen.

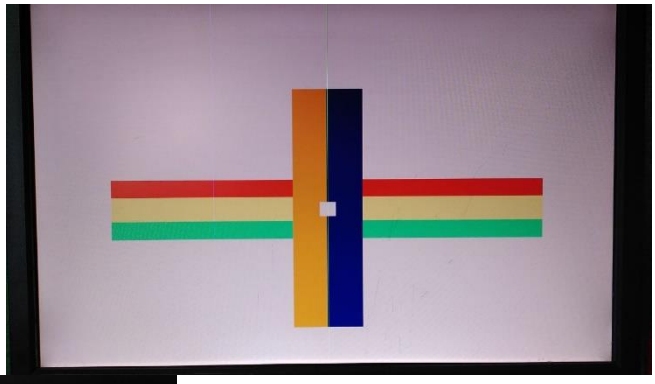
Code

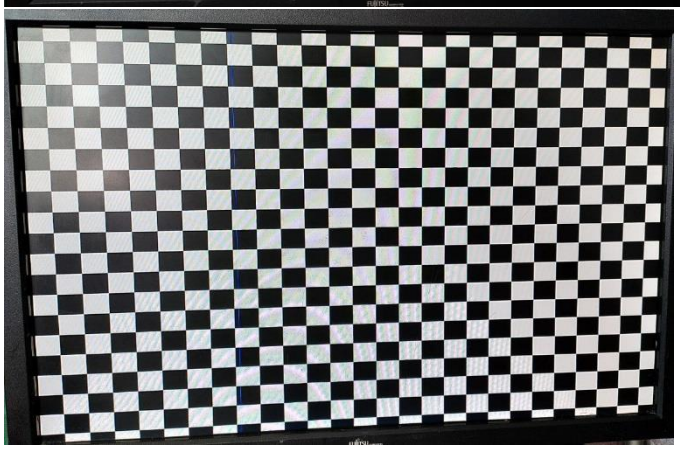
Der komplette Source-Code ist hier zu finden:

https://github.com/j4yj03/embedded_systems_rp_pico/tree/main/embedded/vga_pio

Galerie







Fazit & Ausblick

Schlussendlich kann ich nur sagen, dass dieses Projekt wirklich sehr interessant war. Besonders interessant war es beide Varianten, vergleichen zu können.

Besonders die erste Variante war etwas kniffelig in den Griff zu bekommen, was sicherlich auch daran lag, dass ich die meiste Zeit nur ein Analoges Oszilloskop zur Verfügung hatte, welches mir nur grob sagen konnte, wie meine Signale aussehen.

Persönlich fand ich auch die Erfahrung mit dem rp2040 interessant, da ich zuvor nur mit PIC Microcontrollern von Microship gearbeitet hatte. Mir hat dabei der Weg den die Pi Foundation schlägt gefallen, das man entweder die Möglichkeit hat, High- oder Low-Level in C zu Programmieren, oder High-Level in Python. Ich habe mitbekommen, dass es sogar einen Arduino Kernel gibt.

Ich denke ich werde dieses Projekt noch in irgendeiner Form weiter führen.

Pico DVI
https://github.com/Wren6991/PicoDVI/tree/master/software

Realtime Audio FFT to VGA Display
https://vanhunteradams.com/Pico/VGA/FFT.html

RP2040 Doom
https://github.com/kilograham/rp2040-doom/tree/rp2040

Quellen

Bibliography

Adams, Hunter. [Online] https://vanhunteradams.com/Pico/VGA/VGA.html.

Foundation, Pi. [Online] https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf.

—. [Online] https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf.

multiple. [Online] https://en.wikipedia.org/wiki/Video_Graphics_Array.

—. [Online] https://de.wikipedia.org/wiki/RP2040.