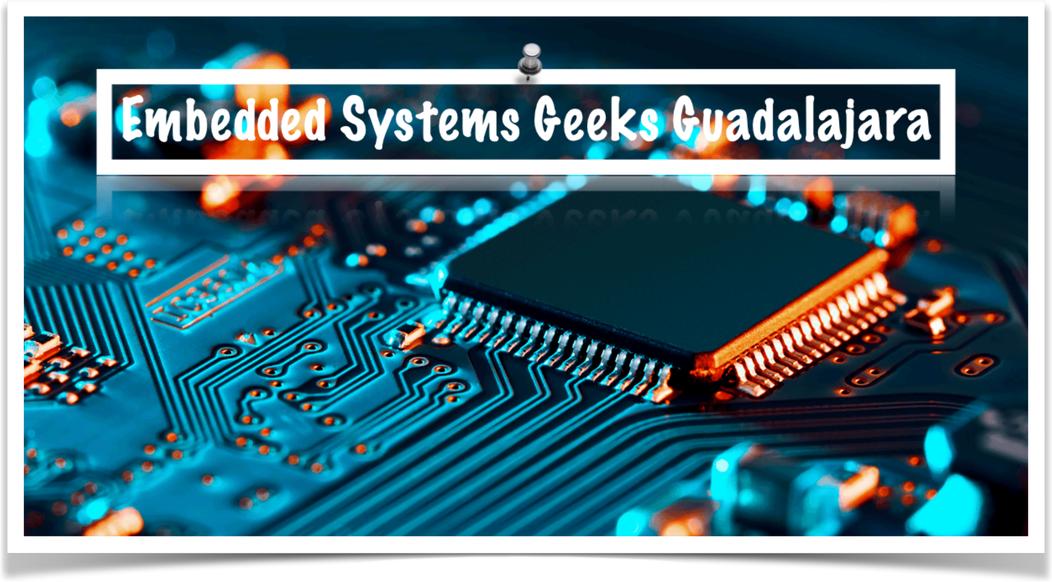
Example: A Pico VGA Driver

Dr Frank Zeyda - 25 April 2024 @ Zoolatech



Frank Zeyda

Dipl.-Inform. BSc PhD PgCLTHE

Independent Consultant
Safety-Critical Systems

https://www.linkedin.com/in/frank-zeyda/

frank.zeyda@gmail.com

Research Interests:

- Semantic Foundations / Unifying Theories (UTP)
- Formal Methods and Verification
- Rigorous Digital Engineering
- Automated Theorem Proving (Isabelle/HOL)

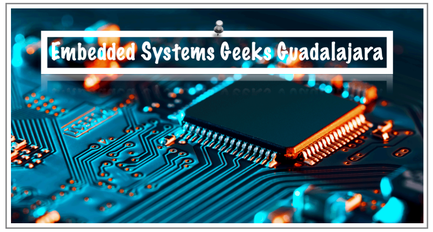
Academic Involvement (2001 to 2018):

- PhD from Teesside University (UK) in 2007.
- Research Associate/Fellow at the University of York (UK).
- Senior Lecturer at Teesside University and the University of York (UK).
- Lead RA on several EPSRC-funded and EC-funded research projects.
- Most publications are listed on: **https://dblp.org/pid/23/6727.html**

Industrial Involvement (1999 to 2000 & 2018 to 2022):

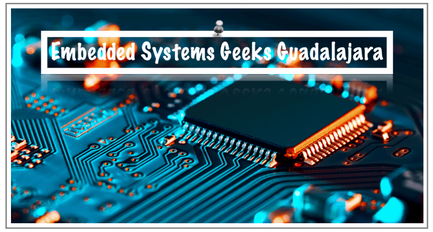
- Siemens Mobility / Transportation Systems (Germany)
- Verified Systems International GmbH (Germany)
- Galois, Inc. (U.S.) (consultant on industrial R&D projects)

Overview

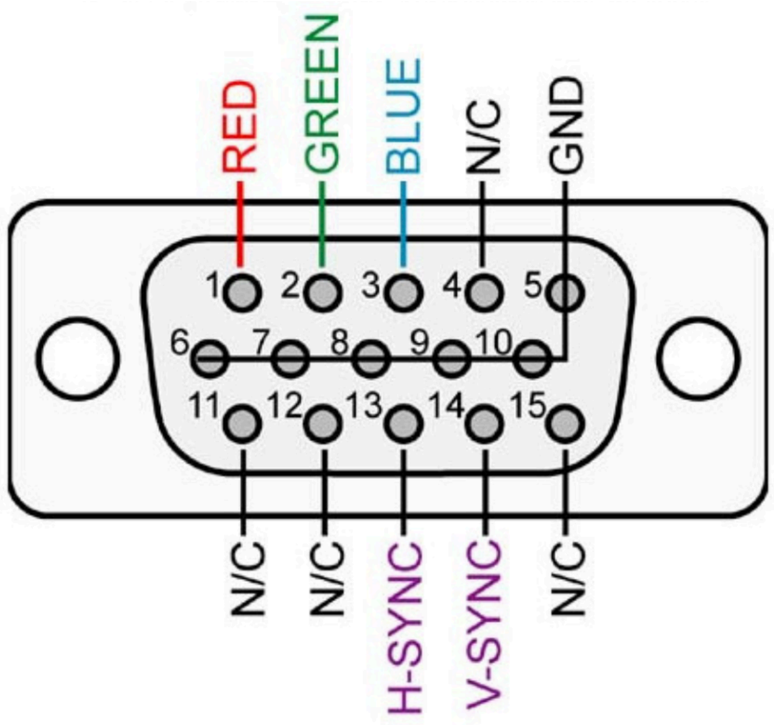


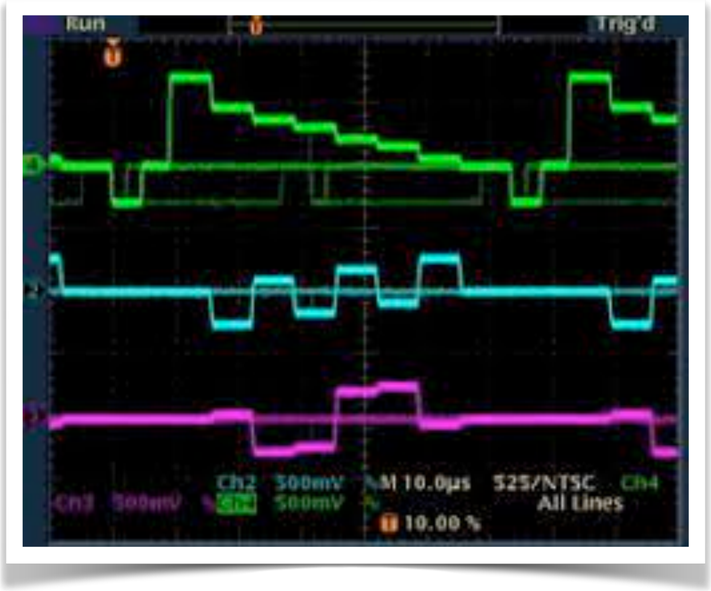
- Show how the Raspberry Pico can be used to generate a **VGA video signal**.
- Learn something about VGA and video signals in general.
- Understand how various *features* of the Pico, like **PIO**, **DMA**, **interrupts** and inter-processor communication (**SIO**) work together.
- Demonstrate VGA output based on a library that I implemented about five year ago and nearly forgotten about to learn about the RP2040 …
- The objectives is **not** to go through all the code line-by-line but just give a general impression and idea what is possible with the Pico and RP2040.
- I will make the code available on the ESG GitHub page ASAP.

VGA Connector

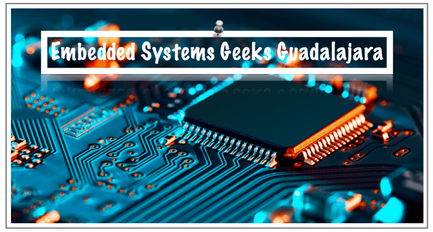


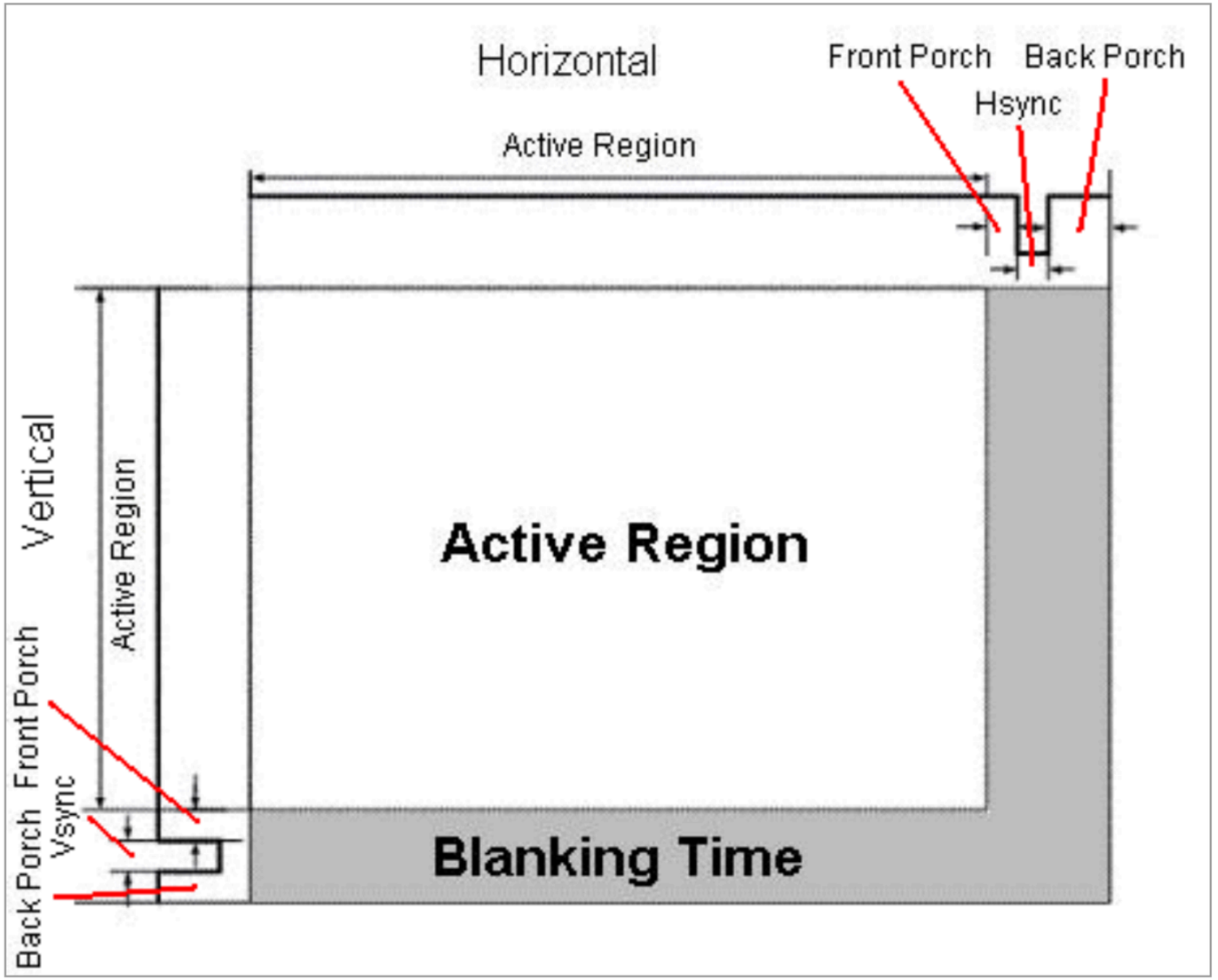
- VGA uses analogue transmission of color information (as **voltages**).





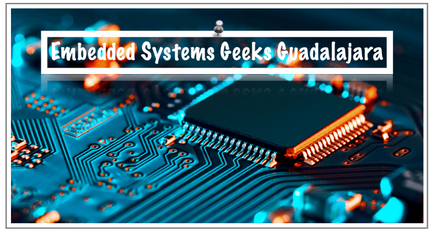
VGA Signal in a Nutshell



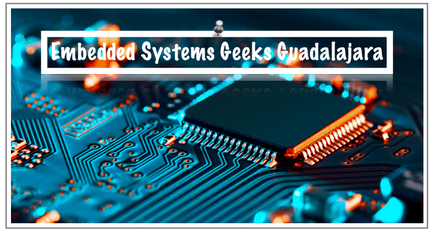


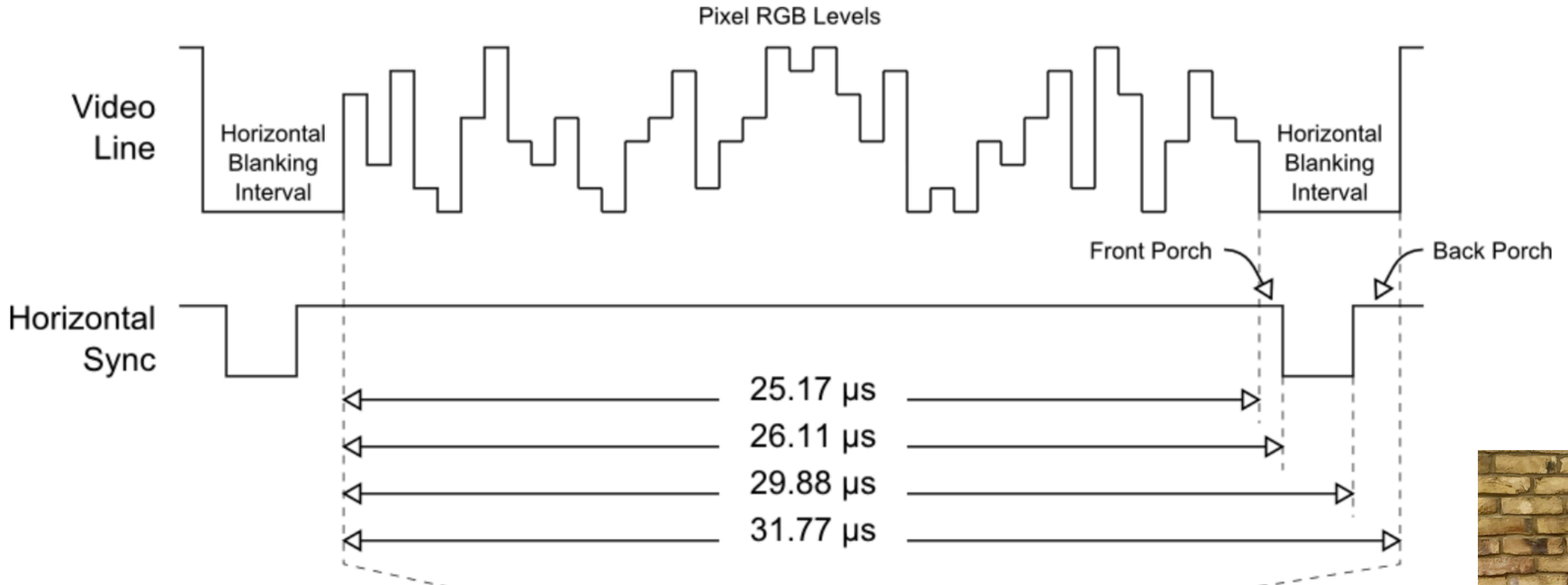
- Timing of **HSYNC** and **VSYNC** pulses determine the mode and resolution.
- No color signal is emitted during the horizontal and vertical blank (front and back porch).
- Color is produced during the active region, via the R, G and B wires.
- Pixel clock for 640x480@60 Hz is **25.175 MHz**.

VGA Formats with Timing

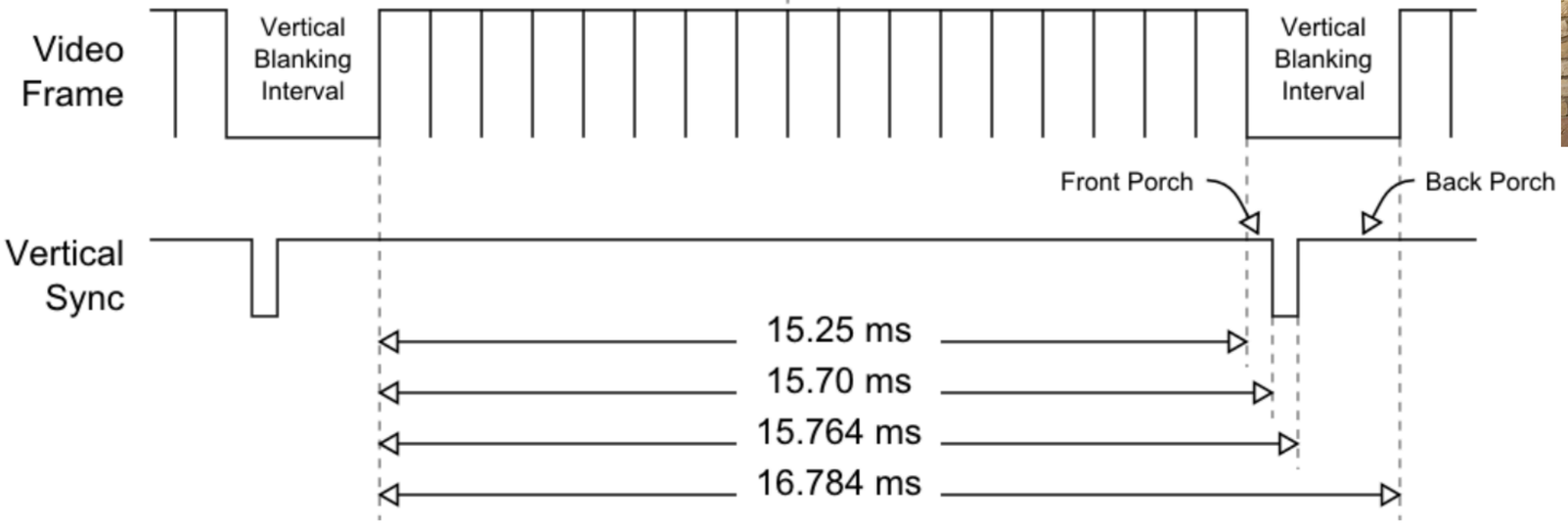


Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

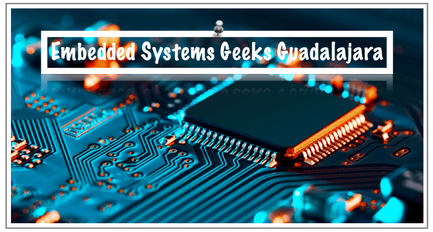




Its all about timing …

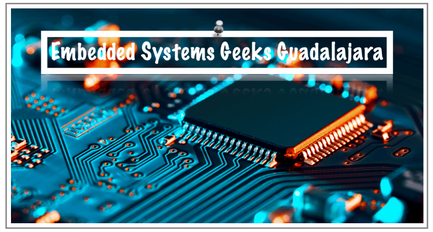


Pico Driver Design



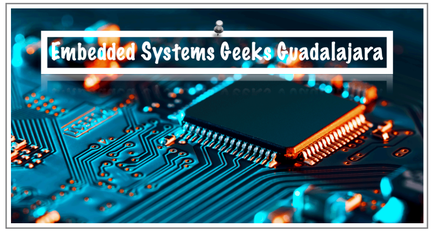
- For precise timing …
  - We use the **PIO**s to generate the **HSYNC**, **VSYNC** and **COLOR** signals.
  - Resource requirements: 2 **SM**s, 12 PIO instructions, interrupts 0, 1, 4.
  - One SM generates both the HSYNC and VSYNC signals.
  - Another SM churns out the pixel data at the right frequency.
- But wait! You said VGA used analog transmission (voltages)?
  - Does the RP2040 have analogue outputs? (**DAC**)

Pico Driver Design

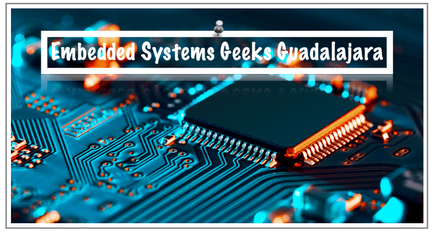


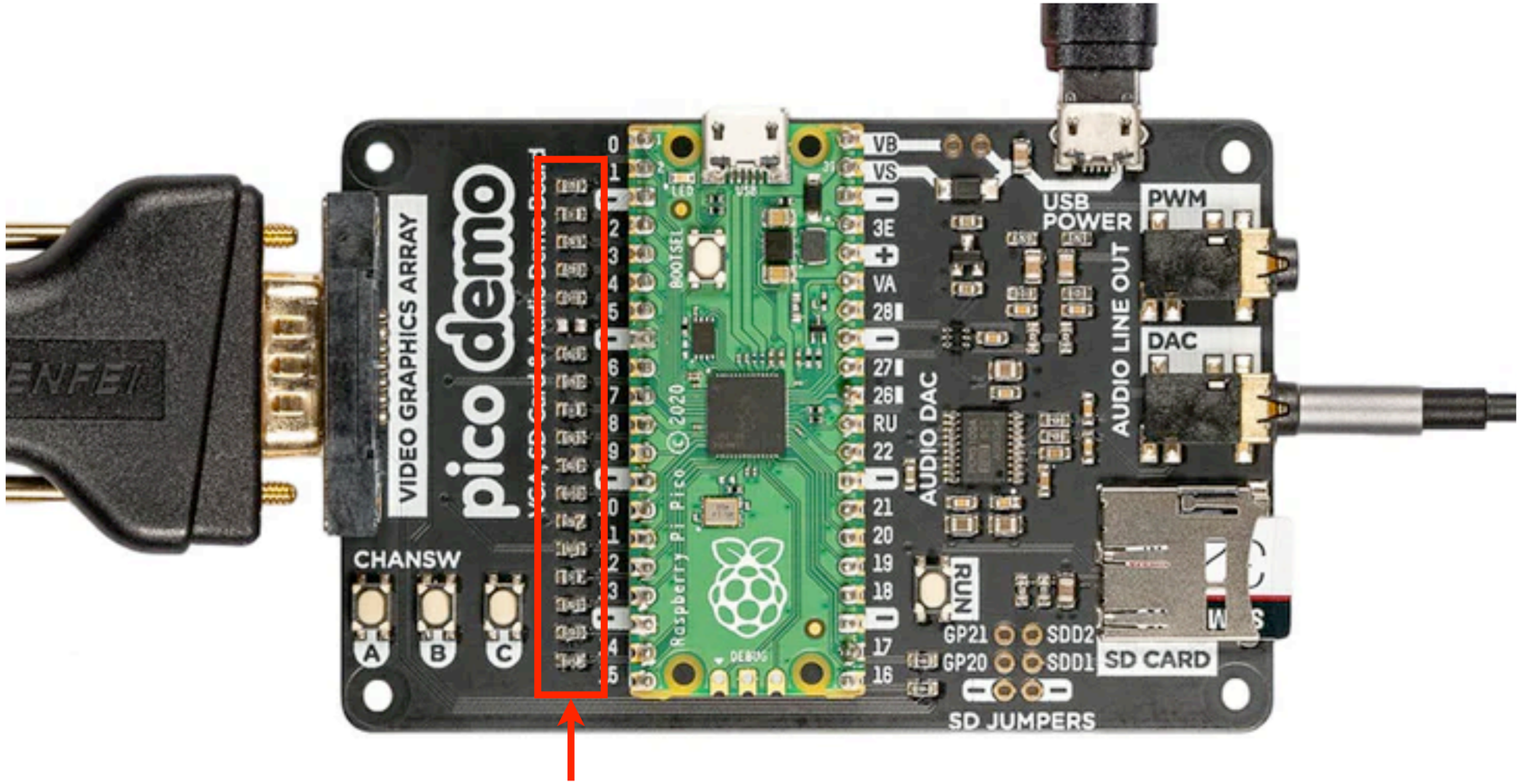
- For precise timing …
  - We use the **PIO**s to generate the **HSYNC**, **VSYNC** and **COLOR** signals.
  - Resource requirements: 2 **SM**s, 12 PIO instructions, interrupts 0, 1, 4.
  - One SM generates both the HSYNC and VSYNC signals.
  - Another SM churns out the pixel data at the right frequency.
- But wait! You said VGA used analog transmission (voltages)?
  - Does the RP2040 have analogue outputs? (**DAC**)
  - Sadly **not**, so we require an external **digital-to-analog** converter (**DAC**).

Pimoroni Pico VGA Base



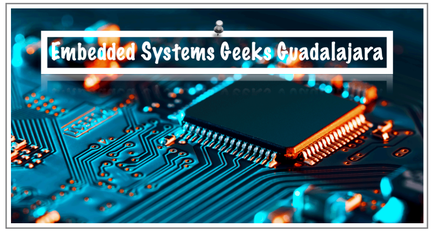
Pimoroni Pico VGA Base





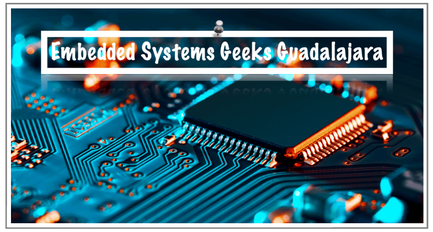
Resistor Ladder DAC

PIO Instruction Set



Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	0	0	0	Delay/side-set					Condition			Address				
WAIT	0	0	1	Delay/side-set					Pol	Source		Index				
IN	0	1	0	Delay/side-set					Source			Bit count				
OUT	0	1	1	Delay/side-set					Destination			Bit count				
PUSH	1	0	0	Delay/side-set					0	IfF	Blk	0	0	0	0	0
PULL	1	0	0	Delay/side-set					1	IfE	Blk	0	0	0	0	0
MOV	1	0	1	Delay/side-set					Destination			Op		Source		
IRQ	1	1	0	Delay/side-set					0	Clr	Wait	Index				
SET	1	1	1	Delay/side-set					Destination			Data				

PIO State Machine for COLOR



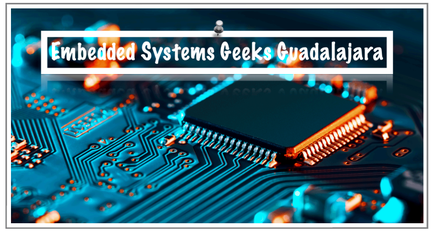

```
.wrap

; NOTE: The VGA driver may insert a one-cycle delay into the
; "wait irq 4" instruction below in case the user requests a
; phase-shift of the pixel output by 180° wrt the pixel clock.

.program vga_pixel ; requires 5 instructions
.wrap_target
  mov pins, null  ; set pixel output to zero (black)
  pull block      ; wait for arrival of pixel data via DMA
public wait_irq:  ; synchronise pixel output with the start
wait irq 4        ; of the subsequent active video scanline
loop:
  out pins, 16    ; churn out color bits for one pixel
  jmp !osre, loop ; continue for as long as the FIFO is fed
.wrap
```

AUTOLOOP

PIO State Machine for COLOR



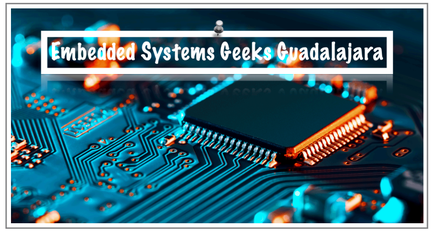

```
.wrap

; NOTE: The VGA driver may insert a one-cycle delay into the
; "wait irq 4" instruction below in case the user requests a
; phase-shift of the pixel output by 180° wrt the pixel clock.

.program vga_pixel
.wrap_target
  mov pins, null  ; set pixel output to zero (black)
  pull block      ; wait for arrival of pixel data via DMA
public wait_irq:  ; synchronise pixel output with the start
  wait irq 4      ; of the subsequent active video scanline
loop:
  out pins, 16    ; churn out color bits for one pixel
  jmp !osre, loop ; continue for as long as the FIFO is fed
.wrap
```

Triggered by SYNC State Machine

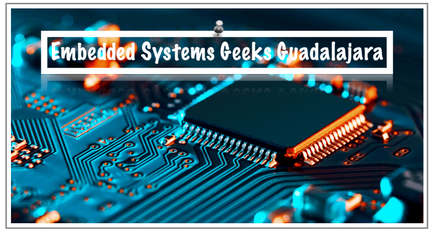
PIO State Machine for SYNC

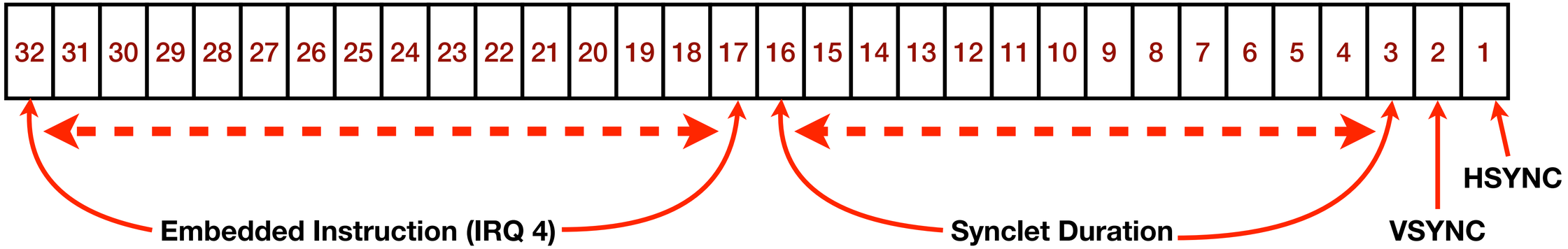



```
; The state machines must run at *twice* the frequency of the pixel clock.

.program vga_sync ; requires 7 instructions
.side_set 1       ; the side-set pin us used for the pixel clock
.wrap_target
public loop:
  out pins, 2     side 0 ; update HSYNC & VSYNC (OSR bits 0..1)
  out x, 14       side 1 ; fetch phase duration (OSR bits 2..15)
delay:
  nop             side 0
  jmp x--, delay  side 1 ; delay for x+1 pixel-clock cycles
  nop             side 0
  out exec, 16    side 1 ; execute embedded instruction (OSR bits 16..31)
; <embedded insn> side 0 ; either an "irq N" or "nop", both with "side 0"
  irq clear 4     side 1 ; for fault-tolerance, withdraw any interrupt 4
                         ; immediately e.g. if the vga_pixel SM has not
                         ; taken it (i.e. due to failure to pull from DMA)
                         ; besides, delay one cycle to be in sync with the
                         ; receiving "wait irq 4" instruction in vga_pixel
.wrap
```

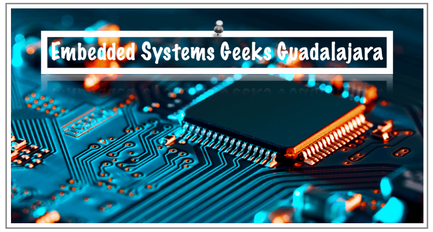
Notion of a “Synclet”





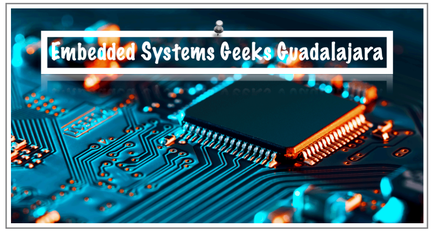

```
/* Note that the unit of duration is pixel clocks. */
static inline synclet_t vga_synclet(bool hsync_level,
                                    bool vsync_level,
                                    int duration /* must be in 4..16387 */,
                                    int irq      /* pass a -1 for none */) {
  assert(INRANGE(duration, 4, (1 << 14) - 1 + 4));
  return (hsync_level ? 0x1 : 0x0) << 0 |
         (vsync_level ? 0x1 : 0x0) << 1 |
         ((duration - 4) & 0x3FFF) << 2 |
         (INRANGE(irq, 0, 7) ? IRQ(irq) : NOP) << 16;
}
```

Synclet Generation



- How are **synclets** generated?
  - ➡ We use DMA channels for that purpose.
  - ➡ Nifty solution that DMA automatically reprograms itself.
  - ➡ Only things left for CPU:
    - Interrupt 0 at the start of vertical blank and to prepare
    - Interrupt 1 to prepare first two scantiness ahead of time
- For scanline generation: “race the beam”
  - ➡ Use a dedicated CPU core just to generate scanlines …
  - ➡ Double buffering leaves about **32 µs** per scanline (~ 40 ns per pixel)

Pesky Details

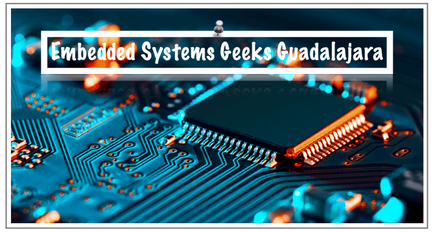


I am going to spare you those …

- Setting up the automatic reprogramming of DMA is not trivial.
- There are other issues with clock signals not being uniform.
  - ➡ Can result in blurry and/or low-quality video output.
  - ➡ Finding good **frequency divider** settings and the right level of **overclocking** proved challenging in its own right.
- Pico SDK turned out to be incomplete …
  - ➡ Had to implement my own library to use the **systick** counter.
- **Ambition**: clean design that showcases various features of the Pico.

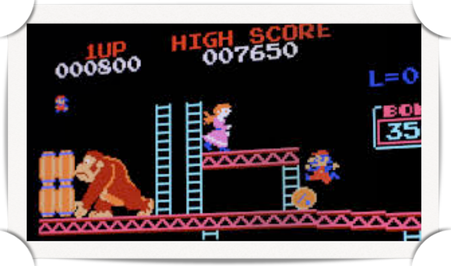
Demonstration

State of the Work

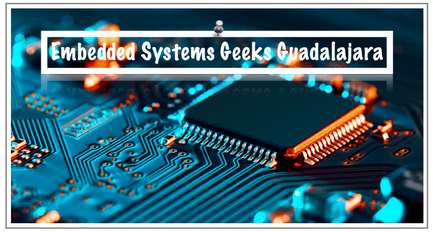


- The code is obscure in places and not easy to follow, despite comments.
  - ➡ Note happy about that but the developer (thankfully) does not need to understand it to use the driver.
- The **synclet** approach is **cool**, I have not seen anyone using this terminology before and doing it quite this way …
  - ➡ Naturally supports various VGA modes.
  - ➡ Fairly economic on resources (PIO SMs, DMA and IRQs).

- ***Scanline generation*** for text, sprites and images, etc., not implemented so far. Please feel free to play around with this (will make code public).
- Initial motivation: port **Donkey Kong** from MAME to the Pico.

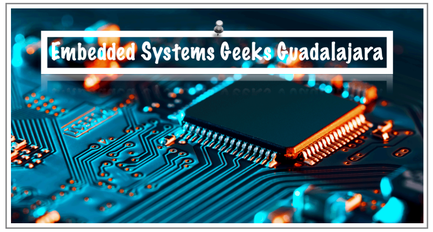


Conclusion



- Implementing a **VGA driver** from scratch taught me a lot about the features of the Pico and RP2040.
- Generally, I am amazed what can be done with a $4 USD chip.
  - Some developers seem to have ported Mario NES games …
- Most technical reviews and side-by-side comparisons, e.g., on YouTube are bullshit; think more about how to creatively use features of a chip instead of being hung up on performance metrics.
- In order to solve *tricky* problems, you need to dig **deeper**: don't always expect solutions being available on the web and/or on a silver plate.
- Pico does support *CircuitPython* but in order to **get the most** out of it, using C/C++ (perhaps Rust) seems inevitable. But for simple and **non-time-critical** control tasks, CircuitPython may well be sufficient.

Thanks for Listening!



Time for questions and comments.