Chapter 3. The VGA, SD card & audio demo boards for Raspberry Pi Pico and Raspberry Pi Pico W

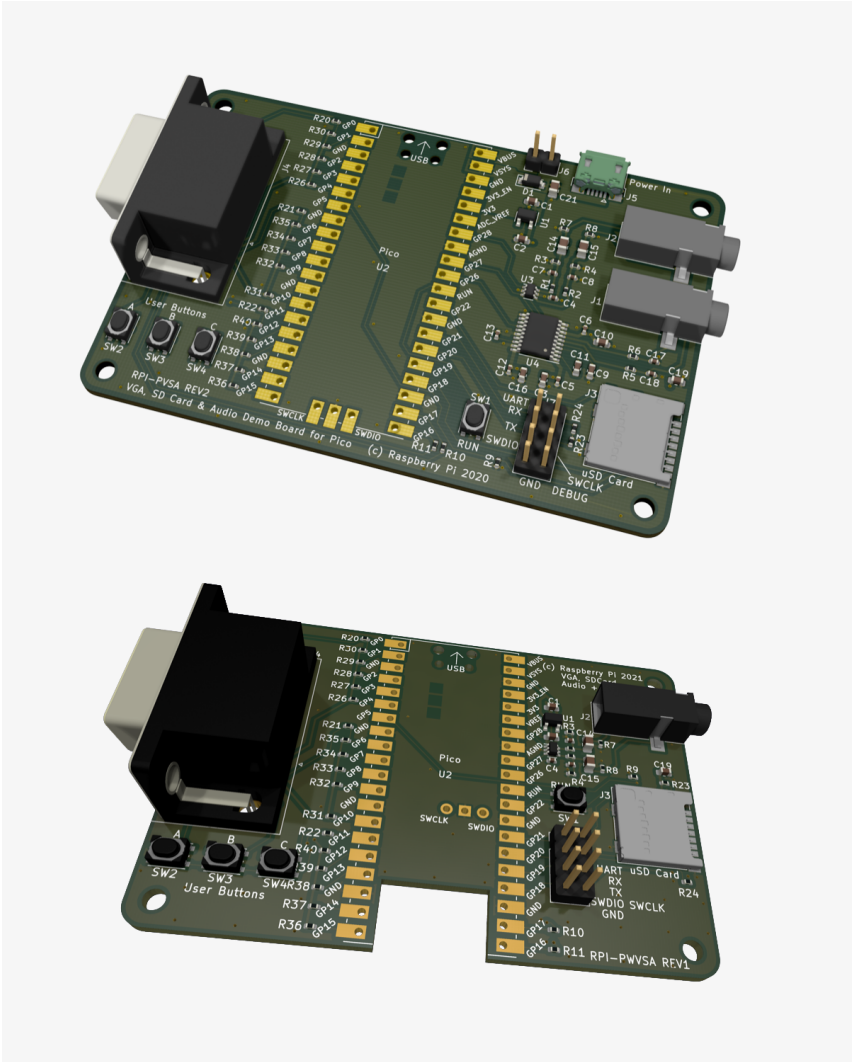


Figure 12. KiCad 3D rendering of the VGA, SD card & audio design example for Raspberry Pi Pico (top) and Raspberry Pi Pico W (bottom)

This example design is intended to serve two distinct purposes. Firstly, we show how we can design a PCB that incorporates *Raspberry Pi Pico or Raspberry Pi Pico W as a module*, used simply as a component on a larger design. Secondly, some of the more complex RP2040 applications require specific additional hardware in order to function correctly. This design provides some example designs for four of these applications, **VGA** video, **SD card** storage, and two flavours of audio output; **analogue PWM**, and **digital I2S** (Raspberry Pi Pico only). Experimental software using these features can be found at Pico Playground.

This design is built using Raspberry Pi Pico or Raspberry Pi Pico W, but as both provide direct access to the pins of RP2040, *much of the circuitry shown here would be equally applicable to designs based around RP2040 itself.*

Schematics and layout files are available for KiCad at https://datasheets.raspberrypi.com/rp2040/VGA-KiCAD.zip and https://datasheets.raspberrypi.com/rp2040/VGA-PicoW-KiCAD.zip. KiCad is a free, open-source suite of tools for designing PCBs and can be found at https://kicad.org/.

One of the key differences between designing with the Raspberry Pi Pico/Raspberry Pi Pico W and RP2040 is that not all

of the I/Os of RP2040 are available to be used on Raspberry Pi Pico and Raspberry Pi Pico W. This is because some of the I/Os are used for internal house-keeping (such as power supply control and monitoring, and an LED) or the wireless interface on Raspberry Pi Pico W, and are not exposed to the outside world. This introduces some challenges, particularly as our choice of application examples want more pins than are available. We believe we've thought of some cunning solutions to this, especially when you consider that we've also added three user buttons and a UART connection to the mix. We'll go through these constraints, and their solutions, in detail later.

Schematic, PCB layout and Raspberry Pi Pico/Raspberry Pi Pico W footprint files are provided in **KiCad** format, with similar design rules as the previous minimal design example in Chapter 2. This time around, whereas the minimal design example has two layers, with a 1mm thick PCB, we've opted for a four-layer, 1.6mm thick PCB. This is primarily because adding extra layers to our PCB means that we can now devote entire layers to power and ground. This is important in a number of ways. Firstly, it improves power-supply decoupling. With the addition of these two layers, we now have two large, parallel rectangles of copper; one connected to the power supply, the other to ground. These are then separated by a thin dielectric material (an insulating PCB layer sandwiched between the copper layers), which makes this a simple parallel plate decoupling capacitor. Secondly, and perhaps most importantly for this design, we now have a variety of low-impedance paths back to RP2040 where the quickly changing I/O return currents can flow back fast and unhindered, without creating current loops which can cause electromagnetic emissions. Another benefit of moving to four layers is that as there is now less of a gap between signal tracks on the top layer and the ground plane directly beneath, it is now much easier to create tracks of different characteristic impedances that may be required in your designs. In this case, we will want 75Ω tracks for the VGA colour signals as VGA is a 75Ω system, using 75Ω cables and 75Ω load termination in the monitor.

This design can be sub-divided into five sections: **power**, **VGA**, **SD card**, **audio**, and **Raspberry Pi Pico/Raspberry Pi Pico W itself**.

3.1. Power

3.1.1. Power input

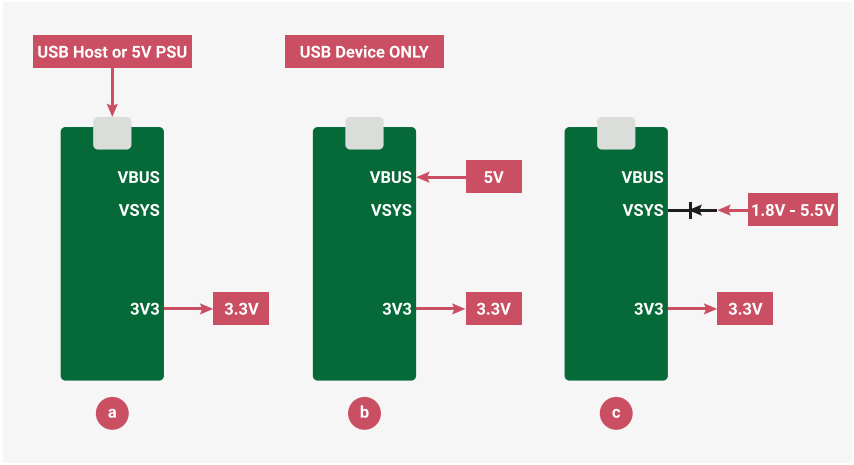


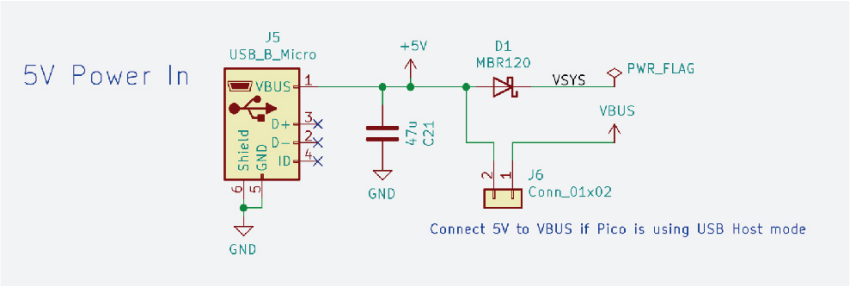
Figure 13. Recommended ways of powering Raspberry Pi Pico and Raspberry Pi Pico W

There are three main ways we can safely power Raspberry Pi Pico and Raspberry Pi Pico W, and the choice is entirely dependent on your application. We can either use the **micro USB connector** on the device itself (option (a) in Figure 13), or we can provide power to either the **VBUS pin** (option b), or the **VSYS pin** (option c).

NOTE

The **3.3V** pin is an output from Raspberry Pi Pico or Raspberry Pi Pico W and should not be connected to an external power source. It is intended to be used as an output to provide power to external circuits.

Figure 14. Section of schematic showing power input



The **VSYS pin** is the main system power supply on Raspberry Pi Pico and Raspberry Pi Pico W. From this supply, a 3.3V supply is generated and used to power RP2040; and also the 3.3V output pin which we can use to power circuits on our design.

The **VBUS pin** is connected to the VBUS of the micro USB connector. There is an onboard diode connecting VBUS to VSYS, which means that VBUS can be used to power VSYS, but not the other way around.

This design provides different options for providing the power, and the *choice of which one to use depends very much on your application*. The first thing to consider is if the USB functionality of Raspberry Pi Pico or Raspberry Pi Pico W will be used.

3.1.1.1. Not using USB

If we are not using USB, then we must provide power for Raspberry Pi Pico or Raspberry Pi Pico W. One way of doing this is to *provide power to Raspberry Pi Pico/Raspberry Pi Pico W from our board*, through Raspberry Pi Pico or Raspberry Pi Pico W's pins. See Figure 15 for an illustration of this. The preferred way of implementing this is to provide a voltage to the **VSYS pin** via a Schottky diode (Figure 14). The one-way nature of the diode ensures we don't encounter any problems if we also supply power to the VBUS pin (accidentally or deliberately). Raspberry Pi Pico and Raspberry Pi Pico W can take a voltage of between 1.8 and 5.5V, as they have an internal buck-boost regulator (one which can regulate the output to a higher or lower voltage than its input), but due to the fact we have an additional voltage regulator in our design (**U1**, more on this later), we need to make sure that VSYS is greater than 3.5V so that **U1** will operate correctly.

Alternatively, we could *provide power to the VBUS pin* (not to be confused with the VBUS connection on Raspberry Pi Pico or Raspberry Pi Pico W's USB connector), rather than the VSYS pin. This would internally power VSYS via the onboard diode, but we must be sure that we *do not connect another power supply to the USB connector* on Raspberry Pi Pico or Raspberry Pi Pico W.

On this design we use a micro USB connector (**J5** in Figure 14) to provide a 5V power input. This is then connected to VSYS via **D1**, which is an MBR120 Schottky diode that can carry up to 1A. There is also an optional jumper (**J6**) we could use if we need to power the VBUS pin, but as we are not using USB, this is unnecessary.

As a third alternative, we could attach a 5V supply to Raspberry Pi Pico or Raspberry Pi Pico W's USB connector, rather than our board's USB connector, similar to device mode discussed below and in Figure 16.

3.1.1.2. Using USB

If we are going to be using USB, then we need to know whether it will be in **host** or **device** mode.

3.1.1.2.1. Device mode

If we are using Raspberry Pi Pico or Raspberry Pi Pico W in device mode, then the host it is attached to will provide 5V on the VBUS pin of the USB connector, which in turn will internally provide VSYS with power (5V minus the drop across the onboard diode). This is everything we need, voltage-wise, we do not need to do anything extra on our design; but this power is only available when the USB host is attached. See Figure 16. If we need to be self-powering, i.e. not reliant on the incoming 5V from the USB host, then we need to provide our own power from the carrier board. Again, we can connect a 5V supply to the micro USB connector **J5**, so that we provide around 5V to the **VSYS pin** of Raspberry Pi Pico/Raspberry Pi Pico W. *Make sure jumper **J6** is open circuit*, as this could result in directly connecting two 5V supplies together. See Figure 15 for an illustration.

3.1.1.2.2. Host mode

If we are using USB host mode, then this time, Raspberry Pi Pico/Raspberry Pi Pico W needs to provide 5V to the VBUS pin of **its own** micro USB connector (not J5). This means that our carrier board design must supply the **VBUS pin** with 5V, as well as powering Raspberry Pi Pico/Raspberry Pi Pico W. We can do this on our design by simply connecting the micro USB connector (**J5** on the schematic) to a 5V supply, and also by fitting a jumper on **J6**, so that this 5V supply gets connected directly to the VBUS pin of Raspberry Pi Pico/Raspberry Pi Pico W. VSYS is supplied by a combination of the onboard diode on Raspberry Pi Pico and Raspberry Pi Pico W, as well as diode D1 on our design, which is perfectly safe.

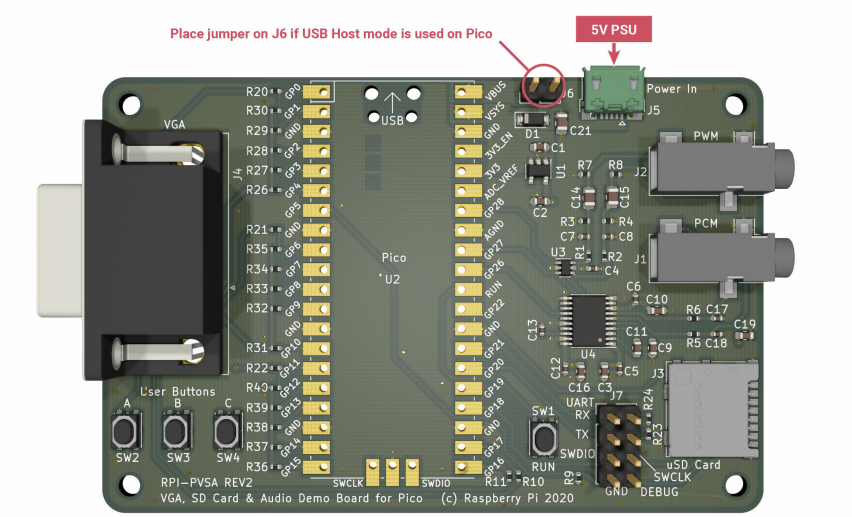


Figure 15. Powering the system using the USB power connector on the VGA, SD card & audio board

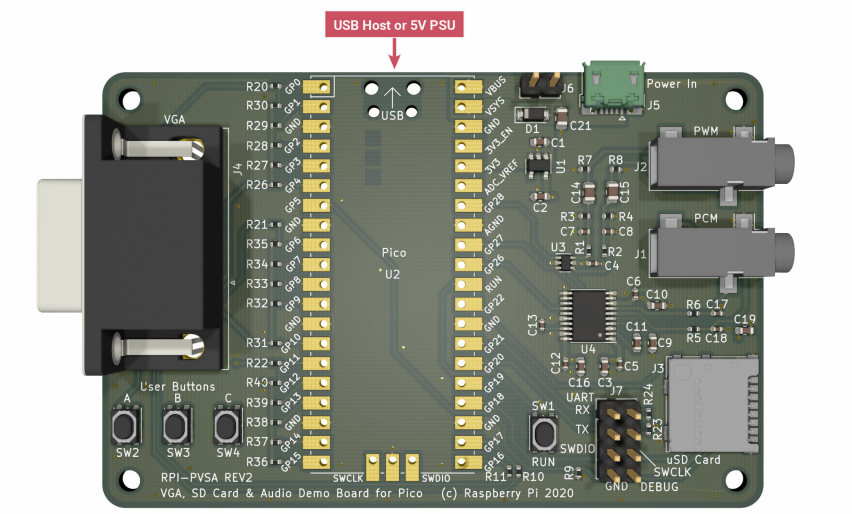


Figure 16. Powering the system using the USB connector on Raspberry Pi Pico/Raspberry Pi Pico W

3.1.2. Audio power supply

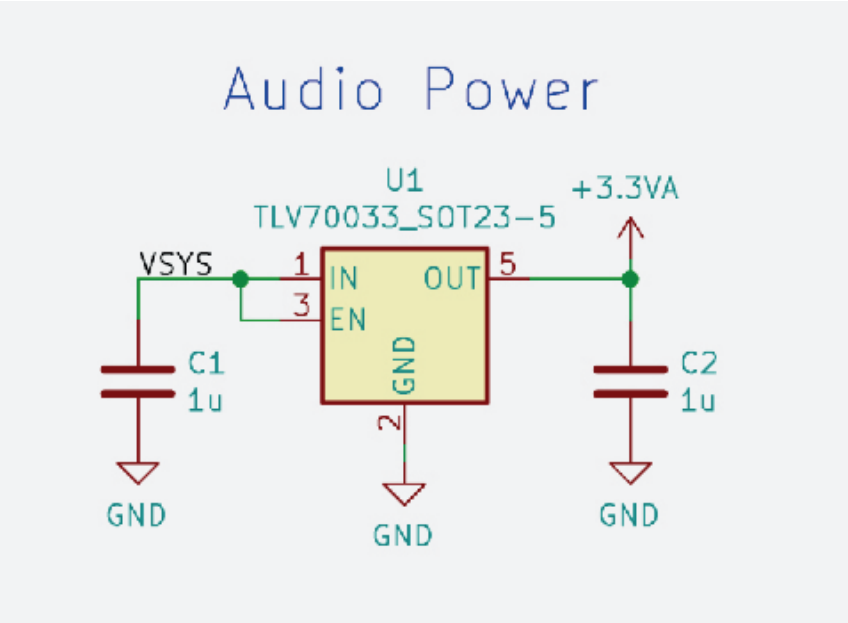


Figure 17. Schematic section showing an additional LDO used for powering the audio

In addition to providing power for Raspberry Pi Pico and Raspberry Pi Pico W, we have some circuits on this design which need suitable power supplies. Fortunately, they are all 3.3V, so we can simply use the 3.3V supply from Raspberry Pi Pico or Raspberry Pi Pico W itself. However, as we have some audio circuitry on this design, it's good to have a nice, clean power source, without all the digital switching noise, for the sensitive audio output sections. To this end, we've included a 3.3V linear voltage regulator (**U1** in Figure 17), specifically for the audio output, which is supplied by VSYS (which is always present, unlike VBUS). This device is a TLV70033, which is a low-dropout (LDO) regulator, with a fixed 3.3V output. This can supply 200mA, which is more than enough for the audio circuits used here. The datasheet for the TLV70033 tells us that we need 1µF capacitors on the input and output pins. We've used 0603 sized ones here (**C1** and **C2**).

NOTE

The switching regulator used on Raspberry Pi Pico and Raspberry Pi Pico W has two operating modes, depending on the amount of current running through it. In low-current mode (less than a few tens of mA), in order to increase its efficiency, it starts to run in power saving mode, which uses PFM (pulse frequency modulation). Ordinarily, this is a good thing, as it increases efficiency, reducing the power consumed at low loads. However, this comes at a price: namely a little more voltage ripple on the 3.3V supply. Most of the time this isn't a problem, but for noise-sensitive circuits, you might want to switch this power saving feature off, and return to the less efficient, but less noisy PWM (pulse width modulation) mode. Raspberry Pi Pico and Raspberry Pi Pico W allow us to do this by forcing the regulator to always use PWM mode, and we do this by setting GPIO23 on RP2040 high. In the VGA demo below, the effects of this noise can be seen if we look carefully at the VGA monitor; we can see slight variations of colour in the horizontal lines, as this supply noise is transferred directly to the DAC outputs. If we disable the PFM mode of the regulator, this magically goes away.

3.2. VGA video

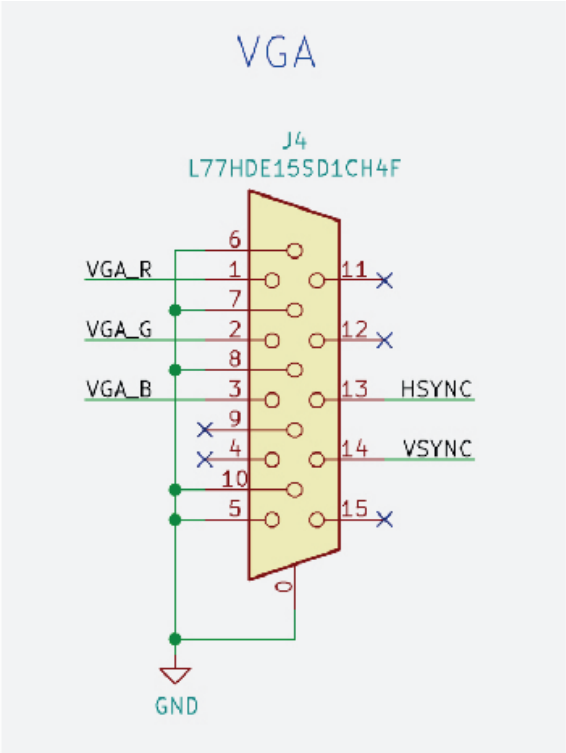


Figure 18. Schematic section showing the VGA video connector

The first application of RP2040 we're demonstrating is VGA analogue video output. This particular example uses the PIO (programmable I/O) on RP2040 to output a commonly used 16-bit RGB data format (RGB-565), and these digital outputs then need to be converted to three analogue output signals: one for each colour. RGB-565 uses five bits each for the red and blue channels, and six bits for the green. In addition to these 16 data bits, VGA monitors also require HSYNC and VSYNC signals for horizontal and vertical blanking timing. That brings us to a total of 18 pins that are needed. As we've mentioned before, pins are at a premium, and we want to use as few as possible so that we can cram more functionality into this design. To this end, we can free up a pin by limiting the green channel to five bits, which will make all the channels the same resolution, by removing the green LSB (least significant bit). It is still desirable for RP2040 to process RGB-565 format data, so PIO will still output six bits of green data to the GPIOs; but we can choose not to use the green LSB in the function select register of that particular GPIO, instead letting RP2040 use it for something else (in this case, the clock for the SD card). The VGA PIO software requires that all 16 bits of data need to be on contiguous (in unbroken, consecutive numerical order) GPIOs, with the sequence being red first, then green, then blue, with the LSB first in each case, which introduces a further design constraint. Raspberry Pi Pico and Raspberry Pi Pico W each have two contiguous rows of GPIOs available for our use: GPIOs 0 to 22, and 26 to 28. We therefore must place VGA data somewhere in 0 to 22, and it makes sense to start at one end or the other in order to make sure there are as many contiguous pins remaining for other functions as possible. We've chosen to use GPIO 0 to 15, which means that the green LSB is GPIO 5, and this is going to be used as SD_CLK. HSYNC and VSYNC can go on any GPIO, as long as they are next to each other. We've picked 16 and 17.

3.2.1. Resistor DAC

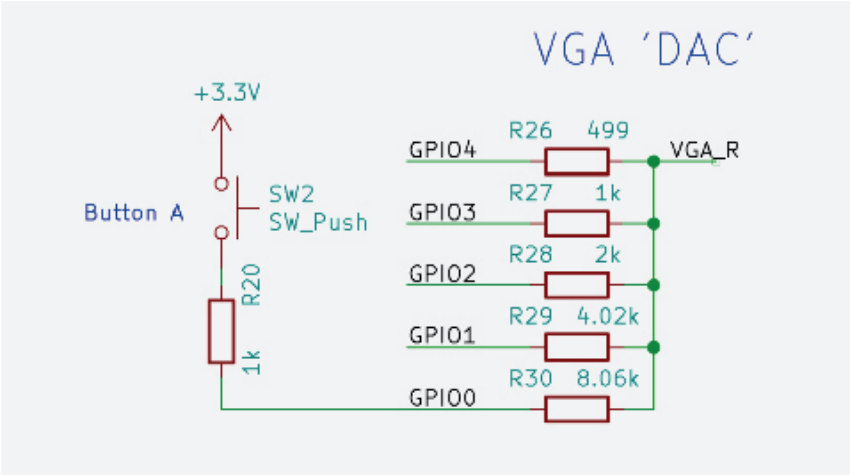


Figure 19. Schematic section showing the red channel of the VGA resistor DAC

The three colour channels on a VGA connector need to be analogue signals, varying from 0 to 0.7V. We therefore need to convert the digital, 3.3V outputs of RP2040 to an analogue signal. Dedicated video DACs (digital to analogue converters) can be used to do this, but a cheaper and simpler method is shown here. You can create a simple DAC using a group of resistors connected directly to the digital outputs. The values of the resistors are weighted to give different amounts of significance to each bit, in the ratio 1:2:4:8:16. It's not going to be as good as using a dedicated video DAC - one of the major drawbacks is that any voltage variation on the IOVDD supply of RP2040 is going to be present on the DAC output - but it's cheaper, considerably less complex, and a lot more fun. If we look at the red channel, net VGA_R on the schematic, we can see the red LSB (GPIO0) is connected to it through a 8.06kΩ resistor. The next bit (GPIO1) has (roughly) half this (4.02kΩ), the next has half again, and so on for the rest of the bits. Ideally, for the most linear DAC performance, we want exactly double the previous resistor value, but these are the nearest commonly available 1% values. This means each GPIO output bit can contribute twice as much current through its resistor than the previous bit, and all these individual current contributions are summed together at the output. The result of this is that if all the bits are high (3.3V), corresponding with the maximum digital value, we have all five resistors in parallel to 3.3V. Basic circuit theory tells us that this is $1/R_{parallel} = 1/499 + 1/1000 + 1/2000 + 1/4020 + 1/8060 = 0.00388$, so $R_{parallel}$ is 258Ω. If we have a monitor connected to this signal, then we will have a 75Ω resistor to ground inside the monitor (this isn't shown on this schematic). This creates a potential divider, with 3.3V connected to 258Ω, which in turn is then connected to 75Ω to ground in the monitor. This means we have a full-scale voltage of 3.3 × 75 / (258 + 75) = 0.74V, which is close enough to the target of 0.7V.

3.2.2. User buttons

The user buttons are not strictly part of the VGA, but because we've decided to add them (**SW2**, **SW3** and **SW4**, see Figure 19), connecting them to the LSBs of the red, green and blue channels, it's important to talk about them, and on their use in the software. We thought it was important to add a few buttons to this design, especially as VGA, SD card and audio give us a lot of potential applications that could use a button or two (e.g. video or music controls, etc). As we've already said, pins are at a premium, and we couldn't afford to dedicate a pin or two to something as frivolous as buttons, so we've come up with the idea of making the VGA LSBs multi-purpose, with a simple hack.

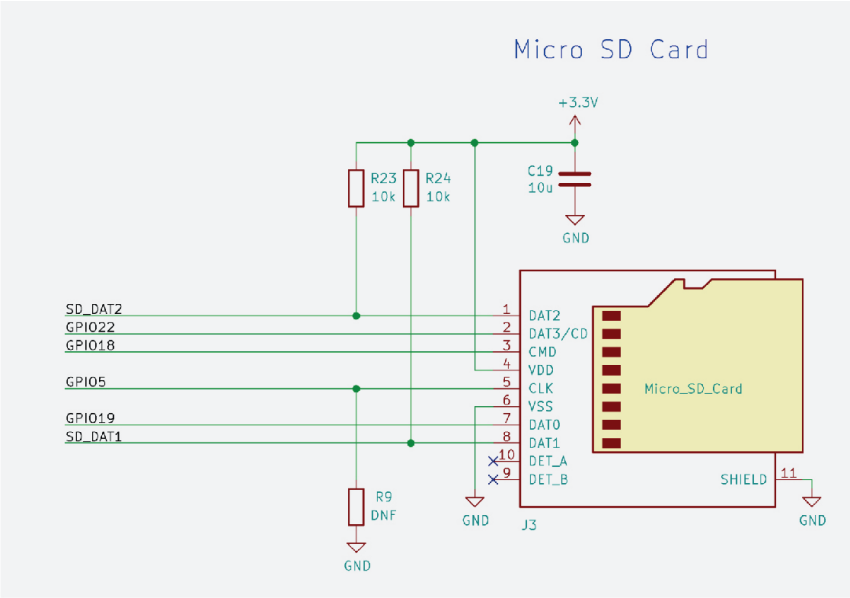
The basic idea is that the GPIO in question is used for VGA as usual during the active periods of video data, but during the video blanking periods, when the DAC levels are not as critical, we can flip the GPIO direction to an input, and then poll it, before flipping it back to an output for the next active video period. If button **SW2** is pressed, then GPIO0 will be connected to potential divider of a 1kΩ resistor (**R20**) to 3.3V, and 8.06kΩ (**R30**) to (worst-case) 0V. This means that GPIO0 will see at least 2.93V, and will therefore register as logic 1. If the button is not depressed, then GPIO4 will be 0.7V or lower, which will result in a logic 0. Of course, this relies on a monitor's 75Ω load resistor for the pull-down. If there is no monitor present, then the user could activate the GPIO's internal pull-down instead.

Obviously, if the button is pressed during active video transmission, then we might expect this to have an effect on the DAC signal level. However, as we are only interfering with the LSB, any effect would be minimal, but the introduction of the 1kΩ resistor (**R20**) in series with the button means that RP2040 will have little problem over-driving this, so the effect on the DAC signal will be minimal. The final point to note regarding the DAC is that, as we've previously

mentioned, the outputs should have 75Ω characteristic impedance. On this PCB, we've used a 1.6mm, four-layer board stack-up, with a gap of 0.36mm between the outer and inner layers. This means that track widths of 0.3mm gives us roughly 75Ω.

3.3. SD card

Figure 20. Schematic section showing the micro SD card connector



The second application we are demonstrating is using an SD card. This design has a micro SD card (**J3**), which has a 4-bit data bus, as well as a clock (CLK) and command (CMD) signal. We can access the SD card in either 4-bit mode, SPI, or 1-bit mode. The constraints our SD PIO software puts upon us is that the four data signals must be connected to contiguous GPIOs. The CLK and CMD signals can go anywhere. As the VGA signals have used up GPIOs 0 to 17, we are left with a contiguous block of five GPIOs, 18 to 22. We will use GPIO19 to 22 for the data bus, and GPIO18 for the CMD signal. For the CLK signal, we are going to use GPIO5, which is in the middle of the VGA signals. If you remember, GPIO5 was earmarked for the 6th, unused bit of the green VGA output. This GPIO can be repurposed by selecting a different function on the GPIO mux, so we are free to assign it to be SD CLK. Often, SD interfaces include pull-up and pull-down resistors on the PCB. This is to ensure that safe values are present at all times, especially when the I/Os are in an undefined state; but also because some of the SD signals are used as mode-select pins (e.g. SPI mode, 1-bit mode, etc). In this design, we are relying on RP2040 to set the GPIO pulls. We have added the option for a pull-down for the CLK signal (**R9**) should we find that in a particular application it is needed, as it is important the CLK input is in a valid state at all times. Having said all this, we have actually included pull-ups on bits 1 and 2 of the SD data bus (**R23 & R24**). This is because we haven't wired these SD card I/Os directly to Raspberry Pi Pico, and have instead connected them via jumper headers (pins 1 to 2 and 3 to 4 of **J7**), which means it's possible to remove the jumpers and still have valid levels on these I/Os. Obviously, if we want 4-bit operation, we must connect the jumpers first. The reason we've done this is, as has been already mentioned, the SD card can also be accessed using SPI or 1-bit mode, which means that if either of these methods are used, we can potentially repurpose data bits 1 and 2 for other uses. Which brings us to…

3.3.1. UART

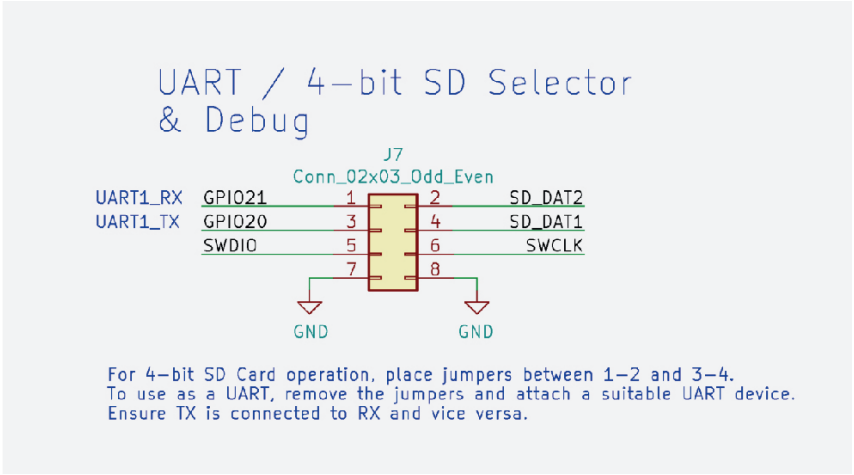


Figure 21. Schematic section showing the optional UART and SWD debug header

As alluded to above, if we use the SD card in 1-bit mode, or do not even use SD card at all, we are then free to use these pins for a UART, which is always a useful thing to have. To this end, we can simply connect a 3.3V compatible UART to pins 1 and 3 of **J7**, rather than the jumpers needed for 4-bit SD card operation. Nominally, GPIO21 is UART1_RX, and GPIO20 is UART1_TX if the dedicated hardware UART controllers are used, but if a PIO UART is implemented, then the TX and RX selection would be configurable.

3.3.2. Debug – SWD

J7 is also home to the SWDIO and SWCLK pins on this design. If Raspberry Pi Pico or Raspberry Pi Pico W has been attached to this PCB in such as way as to connect the debug pins, then they are made available on this header to connect a debugger to. Of course, a debugger could also be connected directly to the Raspberry Pi Pico/Raspberry Pi Pico W itself if this is more suitable.

3.4. Audio

This design demonstrates two different audio options that RP2040 can use, analogue PWM and digital PCM/I2S. However, as you might expect, we cannot afford to dedicate separate pins for each solution, so these two options use the same pins, and the choice of which is to be used is made in software. The circuitry for both audio options have been populated, as the option not being used shouldn't suffer any problems when driven in the wrong mode.

NOTE

We need to remember to connect the audio output device to the correct jack: **J1** for PCM and **J2** for PWM.

These outputs are intended to be used as a line-level driver, and connected to an amplifier's line-in input, but they should also work for headphones with higher impedances. The remaining GPIOs we have available are 26 to 28, and happily, this is all that is needed; two for PWM, and three for PCM.

3.4.1. PWM audio

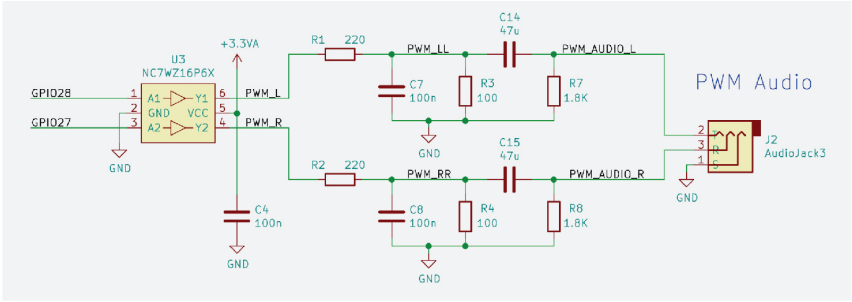


Figure 22. Schematic section showing the Analogue PWM audio circuit

The first method we are going to consider is the analogue PWM. This method is the same as is used on the Raspberry Pi 4 audio output jack, and we've borrowed the circuit from it. This works by taking the digital audio, and outputting it from two GPIO pins as digital PWM (pulse width modulation) signals, one for each of the stereo pair. These digital signals are then fed into a small logic buffer (**U3**). This is so that we can use our nice, clean, audio 3.3V supply we discussed earlier, so hopefully we won't get the digital noise from the main 3.3V supply on our audio signal. This buffered output, which is still a 3.3V digital signal, is then fed into an analogue filter, and the result is that we get an AC-coupled analogue signal in the audible frequency range, which we can then connect to an amp or headphones.

3.4.2. PCM/I2S audio

NOTE

I2S audio output isn't included in the VGA, SD card & audio design example for Raspberry Pi Pico W

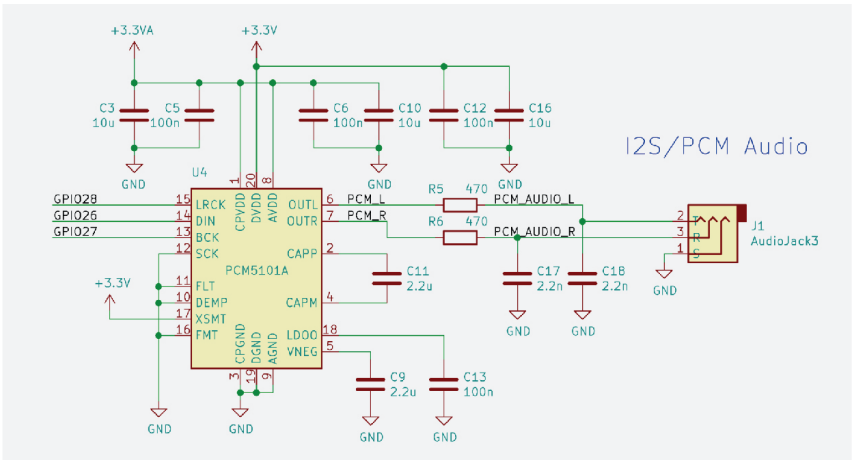


Figure 23. Schematic section showing the Digital I2S PCM audio circuit

The second audio option used here is digital PCM using I2S. This method takes digital audio in PCM (pulse code modulation) format, and sends it using the I2S protocol to an audio DAC, which in turn is connected to an audio output jack. In this design, we've chosen to use the PCM5101A audio DAC. GPIO27 is connected to the BCK input (bit clock), GPIO26 to DIN (serial data), and GPIO28 to LRCK (left or right clock). The rest of the circuitry surrounding the DAC is as per the typical application circuit in the PCM5101A datasheet.

3.5. Raspberry Pi Pico and Raspberry Pi Pico W