RP2040 を使用したハードウエア設計

RP2040 マイクロコントローラーを使用したボードと製品のビルド

奥付

作成日: 2023-03-22
バージョン: 0880191-clean

SDK について

本文書中の「SDK」は、当社の Raspberry Pi Pico SDK を意味します。SDK の詳細については、「Raspberry Pi Pico C/C++ SDK」で参照することができます。本文書に記載されているソースコードは、 3 条項 BSD ライセンスに含まれています。

法的免責事項

この「法的免責事項」の日本語訳は、情報提供のみを目的としていることにご注意ください。以下に記載されている「法的免責事項」の英語原文のみが法的な拘束力を持ちます。

随時変更される Raspberry Pi 製品の技術データおよび信頼性データ (データシートを含む) (以降は「資料」と表記) は、Raspberry Pi Ltd (以降は「RPL」と表記) によって「現状のまま」提供され、商品性および特定の用途への適合性の黙示的保証を含みますがこれに限定されず、いかなる明示的または黙示的な保証も否定します。いかなる場合も、適用される法律が許容する最大限の範囲において、原因の如何または法的責任の根拠にかかわらず、本資料の使用により何らかの形で生じる契約義務、厳格責任、(過失または故意の) 不法行為の直接的、間接的、付随的、特別、例示的、または結果的損害 (代替商品またはサービスの調達、用途、データ、または利益の喪失、または事業の中断を含みますがこれに限定されません) について、RPL は当該損害の発生の可能性を通知されていた場合を含め、責任を負わないものとします。

RPL は、本資料の内容または本資料に記載されている製品に対し、いつでも予告なしに拡張、改善、修正またはその他の変更を加える権利を留保するものとします。

本資料は、適切なレベルの設計知識を持つ熟練したユーザーを対象としています。ユーザーは、本資料の選択と使用、および本資料に記載されている製品の適用について、単独で責任を負うものとします。ユーザーは、本資料の使用により生じるすべての責任、費用、損害またはその他の損失について RPL を補償し、損害を与えないことに同意するものとします。

RPL は、Raspberry Pi 製品に関連してのみ本資料の使用をユーザーに許諾します。それ以外の本資料の使用は禁止されます。その他の RPL の知的財産権またはその他の第三者の知的財産権に対するライセンスは付与されません。

危険性の高い活動: Raspberry Pi 製品は、核施設、航空機の航行システムまたは通信システム、航空管制、兵器システム、セーフティクリティカルなアプリケーション (生命維持装置などの医療機器を含む) の運用など、製品の故障が死、人身障害、深刻な物理的または環境的損害に直接つながるような、フェイルセーフ性能を必要とする危険な環境での使用 (これ以降は「危険性の高い活動」と表記) を意図して設計または製造されていません。RPL は、危険性の高い活動に対する明示的または黙示的な適合性の保証を明確に否定し、危険性の高い活動における Raspberry Pi 製品の使用または適用について一切の責任を負いません。

Raspberry Pi 製品は、RPL の 標準規約に従って提供されます。RPL による本資料の提供は、RPL の 標準規約 (標準規約に記載されている免責事項および保証を含むがこれに限定されません) を拡張または変更するものではありません。

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME (“RESOURCES”) ARE PROVIDED BY RASPBERRY PI LTD (“RPL”) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage (“High Risk Activities”). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL’s Standard Terms. RPL’s provision of the RESOURCES does not expand or otherwise modify RPL’s Standard Terms including but not limited to the disclaimers and warranties expressed in them.

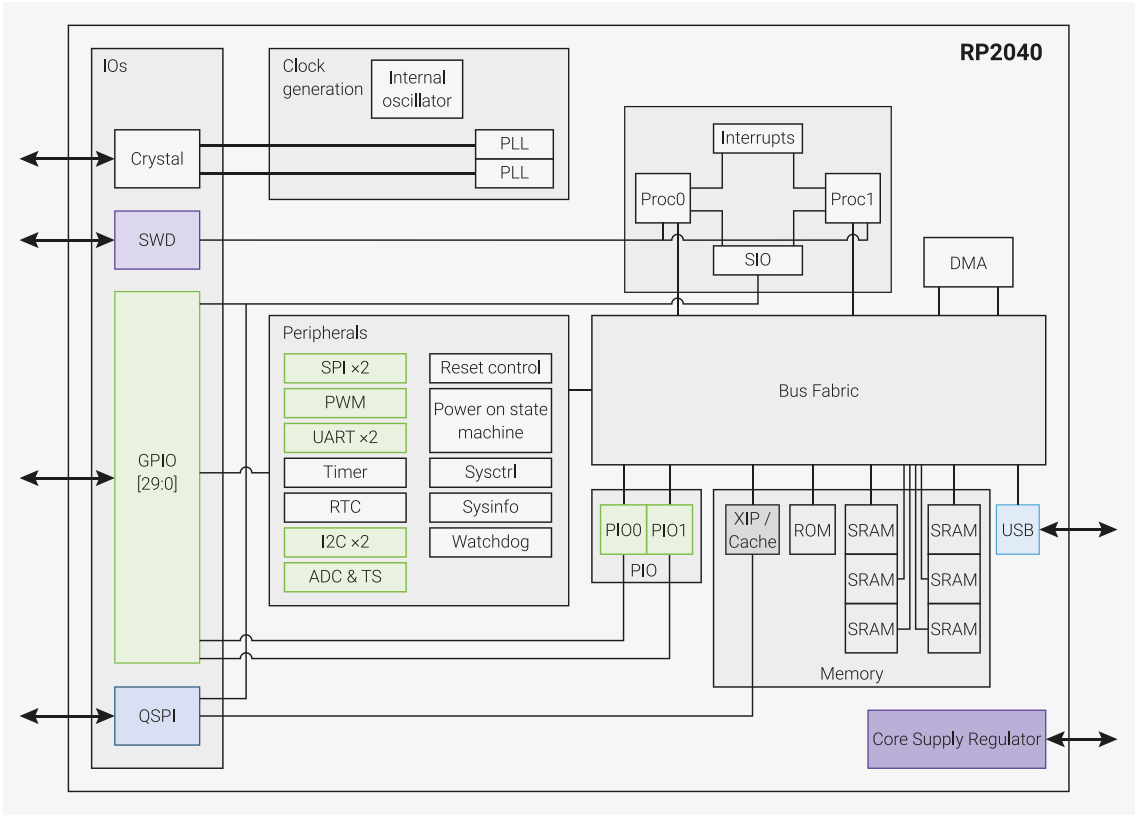
目次

第章 1. RP2040 について

RP2040は、柔軟なデジタルインターフェイスが搭載された低コストで高性能なマイクロコントローラーデバイスです。主な特長は以下のとおりです。

- デュアル Cortex M0+ プロセッサー (最大 133MHz)
- 264kB の内蔵 SRAM (バンク数: 6)
- マルチファンクション GPIO (30 ピン)
- 6 つの SPI フラッシュ専用 IO (XIP 対応)
- 周辺機器専用ハードウェア
- 周辺機器拡張用プログラマブル IO
- 内部温度センサー搭載 4 チャンネル ADC、0.5 MSa/s、12 ビット変換
- USB 1.1 ホスト/デバイス

図 1. RP2040チップのシステム概要図



専用の SPI、DSPI、QSPI インターフェイスを使用して、外部メモリーからコードを直接実行することができます。サイズの小さなキャッシュにより、標準的なアプリケーションのパフォーマンスが向上します。

SWD インターフェイスを使用してデバッグを行うことができます。

内蔵 SRAM は、コードやデータを格納するバンク内に配置されます。専用の AHB バスファブリック接続経由で内蔵 SRAM にアクセスされるため、バスマスターはブロックされることなく個別のバススレーブにアクセスすることができます。

DMA バスマスターにより、反復的なデータ転送タスクをプロセッサーからオフロードすることができます。

GPIO ピンは、直接起動することも、各種の専用ロジック関数から起動することもできます。

周辺機器専用 IP により、SPI、I2C、UART などの固定機能が提供されます。

柔軟な構成が可能な PIO コントローラーを使用して、さまざまな IO 機能を実行することができます。

PHY が内蔵されたシンプルな USB コントローラーにより、ソフトウェアの制御下で FS/LS ホストまたは FS/LS デバイスに接続することができます。

4 つの GPIO で、ADC 入力付きのパッケージピンが共有されます。

2 つの PLL により、USB または ADC の 48MHz 固定クロックと、最大 133MHz の柔軟なシステムクロックが稼働します。

内蔵の電圧レギュレーターによってコア電圧が供給されるため、完成品で供給する必要があるのは IO 電圧だけになります。

第章 2. 最小設計例

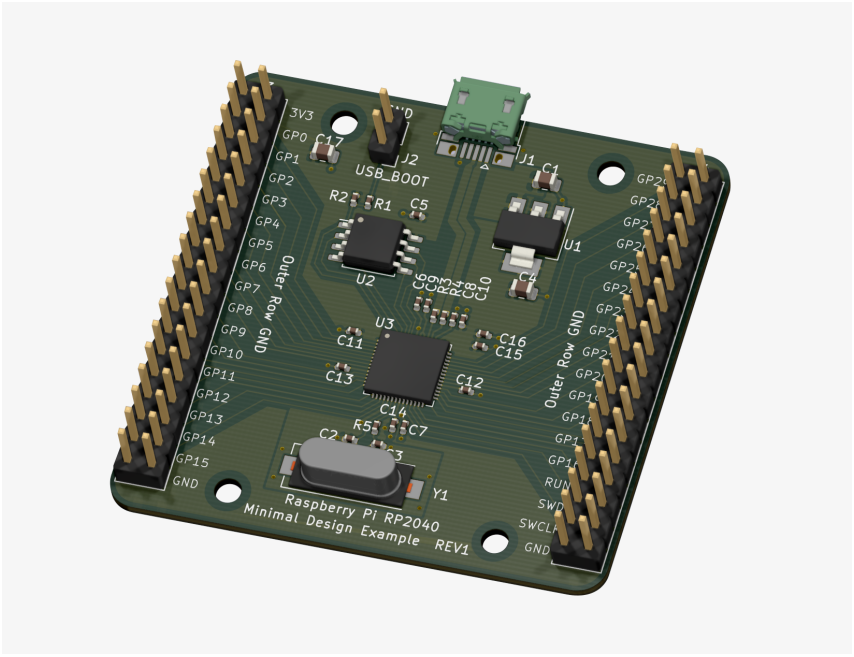


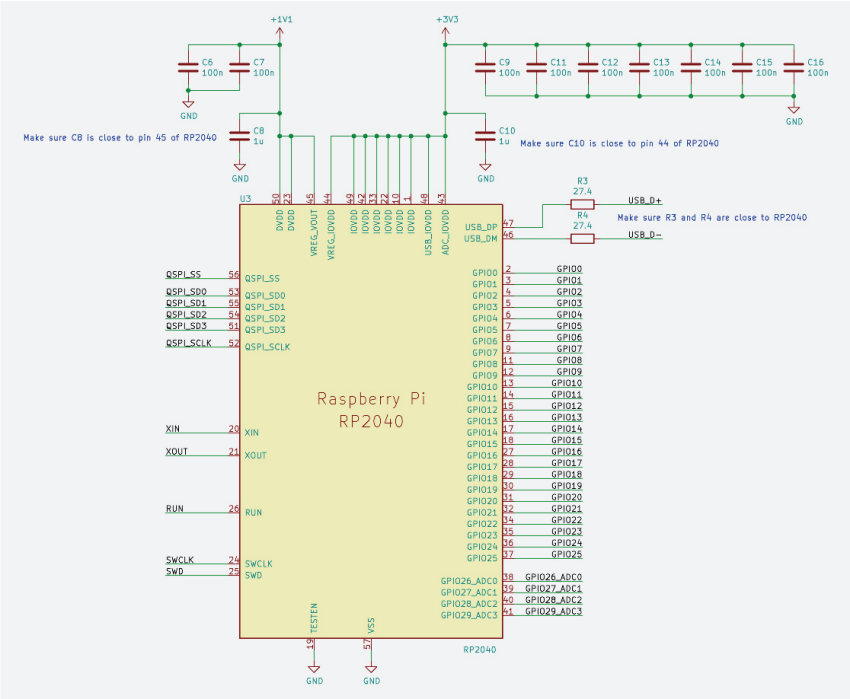
図 2. KiCad 3D レンダリングの最小設計例

この最小設計例の目的は、RP2040 ベースの PCB 設計の開始方法を示すことです。この設計例は、コードの実行に必要な最小限の回路で構成されています。KiCad 用の回路図とレイアウトファイルは、 https://datasheets.raspberrypi.com/rp2040/Minimal-KiCAD.zip で入手することができます。KiCad は、PCB を設計するための無料のオープンソースツールスイートです。 https://kicad.org/ で入手することができます。

このサンプル PCB は、銅層が 2 層になっていて、上面にのみ部品が配置されているため、簡単に組み立てることができます。また、小型の SMD (Surface Mount Device: 表面実装部品) が使用されています。比較的大きな最小トラック幅、クリアランス、穴径により、多くの PCB サプライヤーが低コストで簡単にこの設計を製造することができます。この PCB の名目上の厚さは 1mm ですが、もっと厚い PCB を製造することもできます。たとええば、1.6mm というのが一般的な厚さですが、この厚さの場合、USB の特性インピーダンスに関する問題が発生する可能性があります (これについては後述します)。

このような設計では、手作業によるはんだ付けを簡単に行うことができる大型の部品を使用するのが効果的だと思うかもしれませんが、RP2040 は、ピッチが小さく (ピン間のスペースは 0.4mm)、 56 ピン、7x7mm の QFN (Quad Flat No-leads) パッケージになっています。そのため、手作業ではんだ付けを行う場合は、高度な技術と経験が必要になります。こうした理由により、機械を使用して PCB の製造を行うことをお勧めします。ただし、QFN パッケージのはんだ付けを手作業で適切に行うことができる技術がある場合は、0402 コンデンサーなどの小型 SMD 部品を使用してもかまいません。

図 3. RP2040 接続の回路図



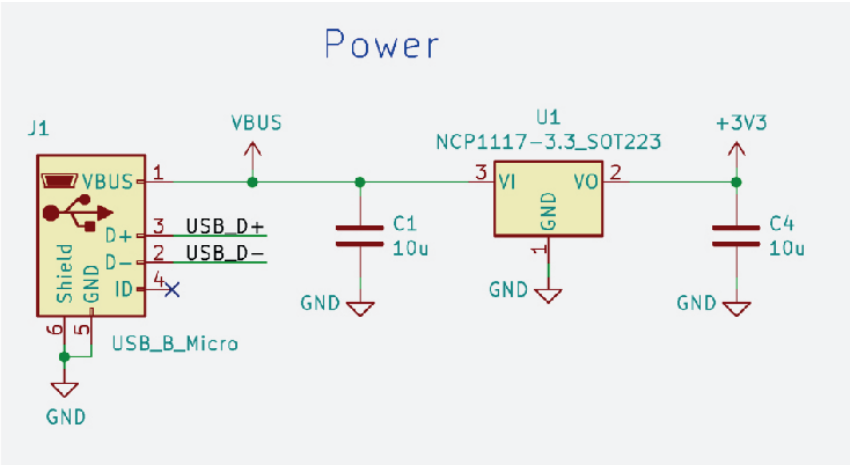
この設計は、電源、フラッシュストレージ、水晶発振器、IO (入力/出力) という 4 つの主要な要素から構成されています。これらの要素について、順に説明します。

2.1. 電源

RP2040 の最小設計の場合、3.3V (IO 用) と 1.1V (チップのデジタルコア用) という 2 つの電圧供給が必要になります。 ただし、RP2040 の内臓 LDO (Low Dropout Voltage Regulator: 低ドロップアウト電圧レギュレーター) により、3.3V が 1.1V に変換されるため、1.1V の電圧供給について気にする必要はありません。

2.1.1. 入力電源

図 4. 入力電源の回路図

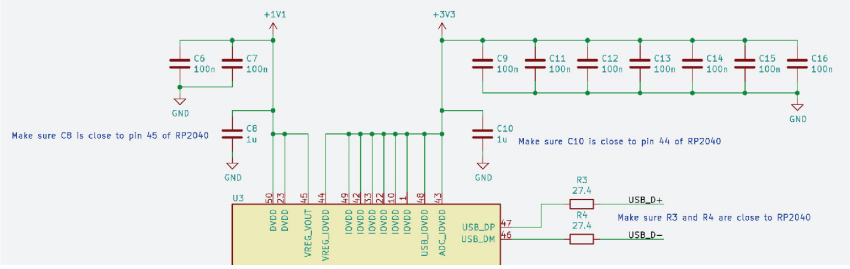


この設計の入力電源は、Micro-USB コネクター (図 4 の J1) の 5V VBUS ピンから供給されます。これは、電子機器に電源を供給する場合の一般的な方法ですが、RP2040 には USB 機能が搭載されているため、このコネクターのデータピン

に配線します。 この設計で必要な電圧は 3.3V だけであるため、USB の 5V 電源を下げる必要があります。ここでは、2 番目の外部 LDO 電圧レギュレーターを使用します。NCP1117 (上図の U1) の出力は、3.3V に固定されています。最大 1A の電流が供給されるため、ほとんどの設計に対応することができます。 NCP1117 のデータシートを確認すると、入力側に 10µF コンデンサーを配置し、出力側にも同じコンデンサーを配置する必要があることがわかります (上図の C1 と C4)。

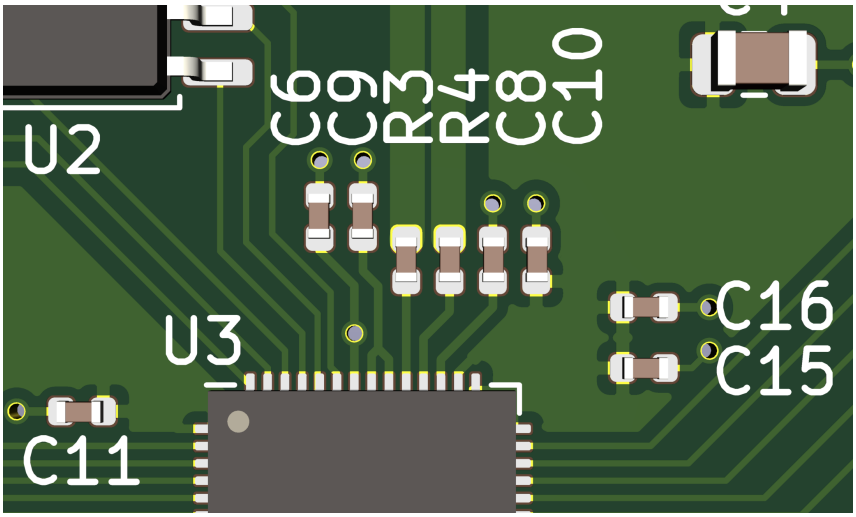
2.1.2. バイパスコンデンサー

図 5. RP2040 の入力電源、電圧レギュレーター、バイパスコンデンサーの回路図



RP2040 の電源設計を行う場合、バイパスコンデンサーも必要になります。バイパスコンデンサーには、2 つの基本的な機能があります。1 つは、電源ノイズを除去する機能で、もう 1 つは、RP2040 内部の回路を即時に使用するためのローカル電荷を供給する機能です。これにより、必要な電流が急激に増えた場合であっても、近接する部品の電圧レベルが過剰に低下するのを防ぐことができます。 そのため、電源ピンの近くにバイパスコンデンサーを配置することが重要です。通常、各電源ピンで *100nF のコンデンサー*を使用することを推奨していますが、例外もあります。

図 6. RP2040 の配線とバイパスのレイアウト



最初に、すべてのチップピンをデバイスから離して配線するための十分なスペースを確保するため、使用するバイパスコンデンサーの数を絞り込む必要があります。この設計では、デバイス上のスペースにあまり余裕がないため、RP2040 のピン 48 とピン 49 で同じコンデンサーを共有しています (図 6 と図 5 の C9)。スペースの問題は、小さな部品や、上下両面に部品が配置された 4 層 PCB など、コストのかかる高度な技術を使用すれば解決することができます。この設計では、使用するバイパスコンデンサーの数を減らし、最適な距離よりも少しだけピンから離れた位置にコンデンサーを配置することにより、複雑さとコストを低減しています。過剰な電圧ノイズによって最小電圧を下回る可能性がありますが、この設計には、最大動作速度を制限する効果があります。ほとんどの場合、この電圧と動作速度のトレードオフが問題になることはありません。

次に、内蔵電圧レギュレーターの要件について説明します。

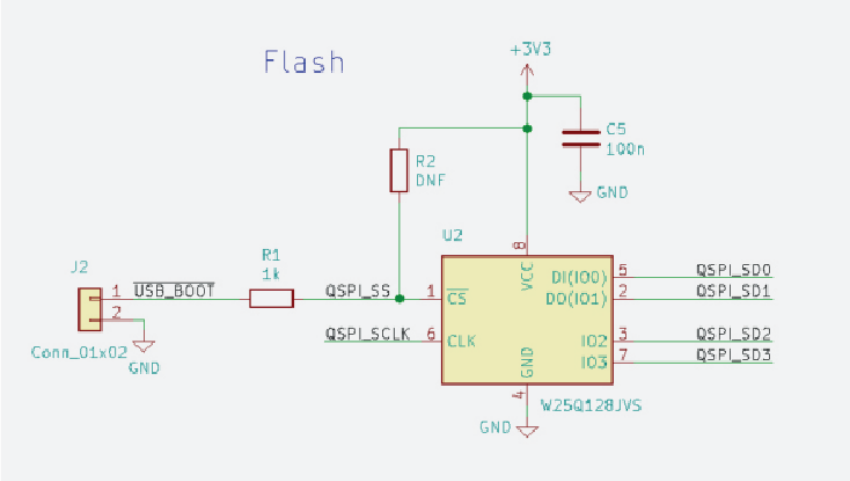
2.1.3. 内蔵電圧レギュレーター

内臓電圧レギュレーターの 3.3V 入力から 1.1V の電圧が供給されます。この設計では、単純に VREG_OUT ピンを DVDD ピンに接続します。内臓電圧レギュレーターには、バイパスコンデンサーに関する特殊な要件がいくつかあります。1.1V を安定して供給するために、入力 (VREG_IN) と出力 (VREG_OUT) の近くに 1µF のコンデンサーを配置する必要があります。また、内臓電圧レギュレーターには、これらのコンデンサーの ESR (Equivalent Series Resistance: 等価直列抵抗) の量に関する制約がありますが、ほとんどの場合、サイズの小さなセラミックチップコンデンサーを使用することにより、この要件を満たすことができます。この設計のコンデンサー C8 とコンデンサー C10 (図 5 を参照) では、0402 サイズのセラミックコンデンサーが使用されています。

オンチップ電圧レギュレーターの詳細については、「On-Chip Voltage Regulator」を参照してください。

2.2. フラッシュストレージ

図 7. フラッシュメモリーと USB_BOOT 回路の回路図



RP2040 を起動して実行するためのプログラムコードを格納するには、フラッシュメモリー (クワッド SPI フラッシュメモリー) を使用する必要があります。この設計では、128Mbit チップ (16M バイト) の W25Q128JVS デバイス (図 7 の U2) を使用しています。これは、RP2040 でサポートされている最大サイズのメモリーです。それほど多くのストレージを必要としないアプリケーションを開発する場合は、容量が少ない安価なメモリーを使用してもかまいません。

フラッシュデバイスの選択方法については、「SSI」を参照してください。

この設計で使用するデータバスは、周波数が非常に高く、使用頻度も高いため、短いワイヤを使用して、RP2040 の QSPI ピンをフラッシュメモリーに直接接続する必要があります。これにより、シグナルインテグリティを確保して、周辺回路のクロストークを低減させることができます。クロストークとは、特定の回路網の信号により、隣接する回路で不要な電圧が発生することを指します。これにより、エラーが発生する場合があります。

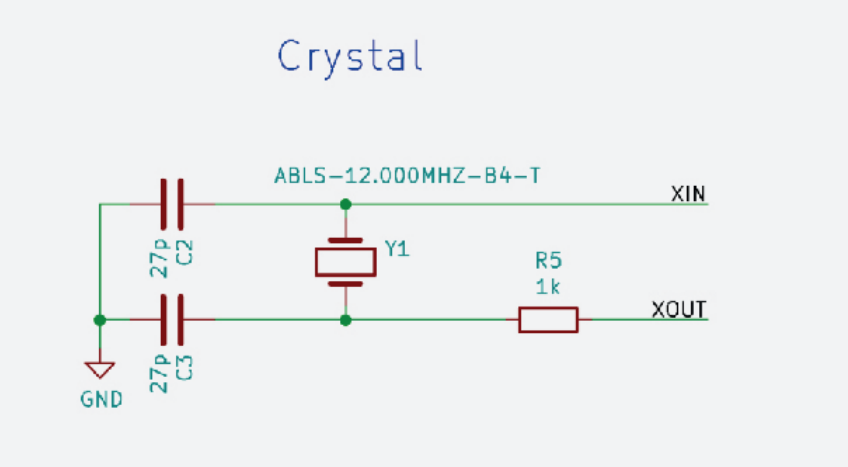
QSPI_SS 信号は、特殊な信号です。この信号はフラッシュに直接接続されていますが、2 つの抵抗器も接続されています。1 つ目の抵抗器 (上図の R2) は、3.3V 電源に対するプルアップ抵抗器です。フラッシュメモリーを使用するには、デバイスの電源投入時に、チップ選択入力の電圧が 3.3V 電源供給ピンと同じ電圧になっている必要があります。これらの電圧が異なっている場合は、フラッシュメモリーが正しく機能しません。RP2040 の電源を入れると、QSPI_SS ピンが自動的にプルアップピンになりますが (これがデフォルトの動作です)、その際に、QSPI_SS ピンの状態が保証されない時間が発生します。プルアップ抵抗器を追加することにより、この問題を解決することができます。上図の R2 には「DNF」と記載されていますが、これは「Do not Fit」という意味で、このフラッシュデバイスでは、外部のプルアップ抵抗器は必要ないということを表しています。ただし、別のフラッシュデバイスを使用する場合は、R2 の位置に 10kΩ の抵抗器を配置しなければならないことがあるため、このように記載しています。 2 つ目の抵抗器 (上図の R1)

は、1kΩ の抵抗器です。「USB_BOOT」と記載されたヘッダー (上図の J2) に接続されています。これは、QSPI_SS ピンが「ブートストラップ」として使用されているためです。RP2040 は、ブートシーケンスの実行中に IO の値を確認します。その値が論理値 0 である場合、RP2040 が BOOTSEL モードに戻り、RP2040 が USB マスストレージデバイスになります。この状態で、コードを直接 USB デバイスにコピーすることができます。 J2 のピンの間にジャンパーワイヤを配置する場合は、QSPI_SS ピンを接地します。その後、RUN ピンを切り替えるなどの方法でデバイスをリセットすると、RP2040 が BOOTSEL モードで再起動します。その際、フラッシュメモリーの内容が実行されることはありません。

信号に影響を与える可能性のある銅線が長くならないように、R1 と R2 の両方をフラッシュチップの近くに配置する必要があります。

2.3. 水晶発振器

図 8. 水晶発振器と負荷コンデンサーの回路図



RP2040 には独自の内部発振器が組み込まれているため、実際には、外部のクロックソースは必要ありません。ただし、この内部発振器の周波数はチップごとに異なり、供給電源の電圧や温度の違いによっても周波数が変化するため、安定した外部の周波数源を使用することをお勧めします。正確な周波数を必要とするアプリケーションの場合、外部の周波数源を使用する必要があります (USB など)。

外部の周波数源を供給するには、*CMOS 出力 (3.3V の矩形波) を持つクロックソース*を XIN ピンに渡すか、XIN と XOUT を *12MHz の水晶発振器*で接続します。水晶は比較的安価で精度も高いため、この設計では水晶発振器を使用します。

この設計で使用する水晶発振器は、ABLS-12.000MHZ-B4-T です (図 8 の Y1)。これは、一般的に使用されている 12MHz の水晶発振器で、公差は 30ppm です。この水晶発振器で、ほとんどの用途に対応することができます。この水晶発振器の最大 ESR は 50Ω、負荷容量は *18pF*です。付属部品の選択する際に、これらの数値が関係してきます。

適切な周波数で水晶発振器を機能させるための負荷容量は、各メーカーによって指定されています。この設計の目標負荷容量は 18pF です。同じ値の 2 つのコンデンサーを、水晶発振器の両側に 1 つずつ配置して接地すると (上図の C2 と C3)、この負荷容量になります。水晶発振器から見た場合、これらのコンデンサーは、水晶発振器の 2 つの端子の間で直列に接続された状態になります。これらのコンデンサーの組み合わせを基本的な回路理論で考えた場合、(C2*C3)/(C2+C3) として表すことができます。C2 と C3 の値は同じであるため、*単純に *C2/2 として表すことができます。この設計では、27pF のコンデンサーを使用しているため、直列接続の値は 13.5pF になります。この意図的な負荷容量だけでなく、PCB のトラックと、RP2040 の XIN ピンと XOUT ピンから発生する意図しない追加の負荷容量も考慮する必要があります。ここでは、この追加の負荷容量を 5pF と仮定します。この負荷容量は、C2 と C3 で並列的に発生するため、意図的な負荷容量にこの負荷容量を単純に加算すると 18.5pF になります。これは、上述した 18pF とい

う目標値に近い負荷容量になっています。

次に、水晶発振器の最大 ESR (等価直列抵抗) を考慮する必要があります。この設計では、ESR の最大値が 50Ω のデバイスを選択しましたが、この値は、1kΩ の直列抵抗器 (R5) とともに使用した場合に、水晶発振器の過剰使用による損傷を防ぐのに適した値です。別の水晶発振器を使用する場合は、この値を調整しなければならないことがあります。

2.4. IO

2.4.1. USB

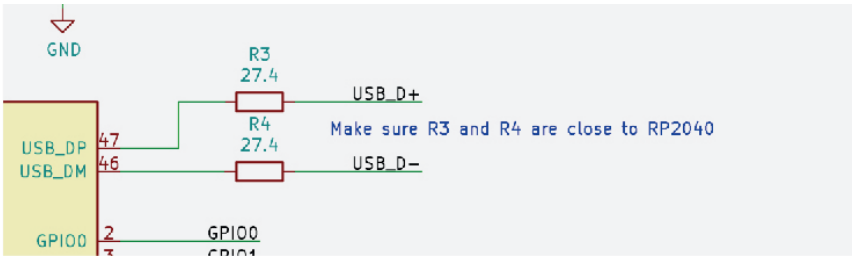


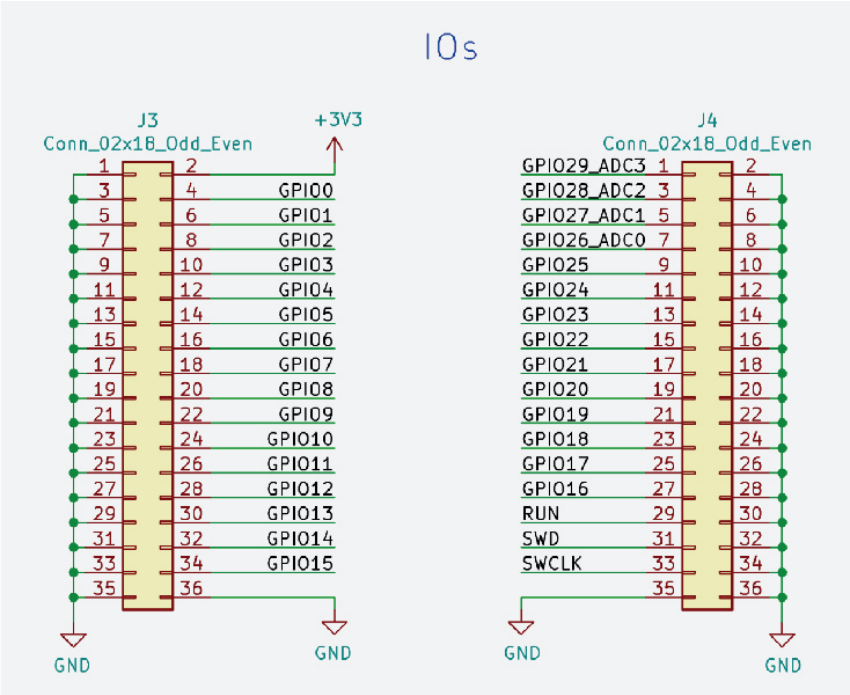
図 9. RP2040 の USB 端子と直列終端の回路図

RP2040 には、フルスピード (FS) とロースピード (LS) という 2 種類の USB で使用する 2 つのピンがあります。使用するソフトウェアに応じて、いずれかの USB が「ホスト」または「デバイス」として機能します。これまでに説明したように、RP2040 は USB マスストレージデバイスとして起動することもできるため、これらのピンを USB コネクター (図 4 の J1) に接続します。RP2040 の USB_DP ピンと USB_DM ピンは IO に組み込まれているため、USB の速度 (FS/LS) とモード (ホストモード/デバイスモード) を示すための追加のプルアップピンやプルダウンピンは必要ありません。ただし、USB のインピーダンス仕様の要件を満たすために、これらの IO では、27Ω の直列終端抵抗器 (図 9 の R3 と R4) をチップの近くに配置する必要があります。

RP2040 は、フルスピードデータレート (12M ビット/秒) に設定されていますが、転送経路 (チップとコネクターをつなぐ銅トラック) の特性インピーダンスが、USB 仕様の 90Ω (差動測定) に近い値になっているかどうかを確認する必要があります。この設計のように、厚さが 1mm のボードで、USB_DP と USB_DM で 0.8mm 幅のトラックを使用し、これらのトラック間に 0.15mm の間隔を設定すると、差動特性インピーダンスの値が約 90Ω になります。これにより、シグナルインテグリティを低下させる電圧の反射が最小限に抑えられるため、信号が転送経路をスムーズに通過するようになります。 転送経路を適切に機能させるには、転送経路の下には何も配置しないようにする必要があります。接地銅の部分が、トラックの全長にわたって途切れることなく広がっていることを確認してください。この設計では、最下部の銅層のほぼ全体を接地し、USB トラックが接地部以外を通過しないような構造になっています。厚さが 1mm を超える PCB を使用する場合は、2 つの方法があります。1 つは、USB の転送経路を改良して、トラックと地面との距離を小さくする方法です (この方法は、物理的に不可能かもしれません)。もう 1 つは、こうしたこととは別に、最善の措置を講じるという方法です。フルスピード USB は柔軟性が高い設計になっているため、さまざまな方法が考えられます。各種の用途で使用できますが、USB の規格に準拠するのが難しい場合があります。

2.4.2. IO ヘッダー

図 10. 2.54mm の IO ヘッダーの回路図



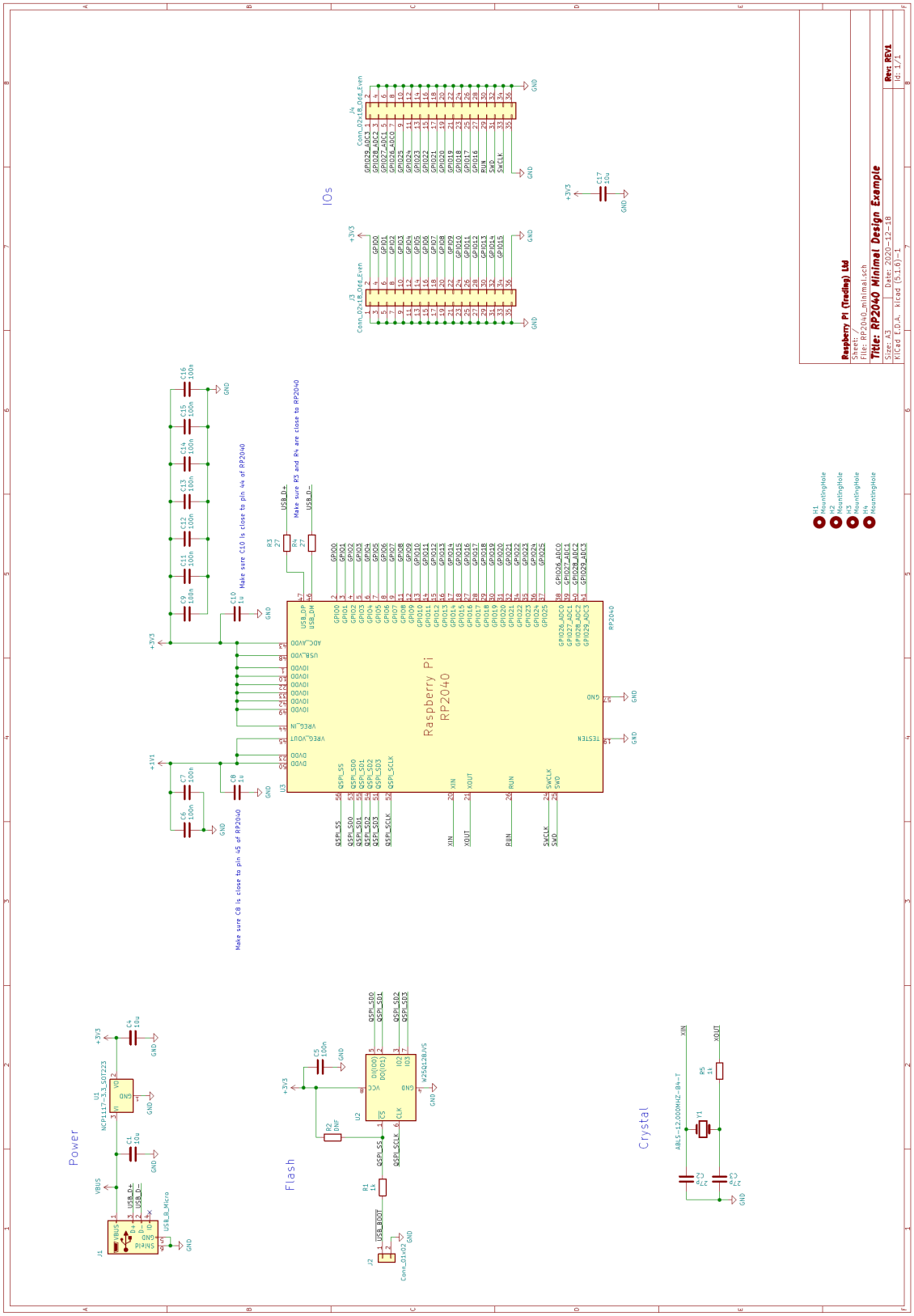
これまでに説明した USB コネクターのほかに、1 組の 2x18 ウェイ、2.54mm ヘッダー (図 10 の J3 と J4) がボードの両側に 1 つずつ配置され、これらのヘッダーに残りの IO が接続されています。これは、特定の用途を想定していない汎用的な設計であるため、IO はユーザーが自由に接続できるようになっています。各ヘッダーの内側のピン列が IO で、外側のピン列はすべて接地されています。IO コネクターでは、多くのピンを接地することをお勧めします。多くのピンを接地することにより、低インピーダンスの接地状態を維持し、IO 接続を通過する電流のリターン経路を十分に確保することができます。高速で切り替わる信号のリターン電流が、回路内で長いループ状の経路を通過すると電磁波障害が発生しますが、この障害を最小限に抑えるには、多くのピンを接地することが重要になります。

どちらのヘッダーも、同じ 2.54mm のグリッド上に配置されているため、このボードを別のボード (ブレッドボードなど) に接続する場合も、簡単に接続することができます。2×18 ウェイの代わりに 1 列の 18 ウェイヘッダーを取り付け、外側のピン列の接地を省略すると、さらに簡単にブレッドボードに接続することができます。

2.5. 回路図

完全な回路図を以下に示します。これまでに説明したように、設計ファイルは KiCad 形式で提供されています。

図 11. 最小設計のボードの完全な回路図



2.6. サポートされているフラッシュチップ

ブート ROM は、最初のフラッシュプローブシーケンスを使用して、フラッシュから第 2 ステージを抽出しますが、こ

のフラッシュプローブシーケンスでは、`03h` シリアルリードコマンド、24 ビットアドレス、約 1MHz のシリアルクロックが使用されます。クロック極性とクロック位相の 4 つの組み合わせを繰り返し循環させることにより、有効な第 2 ステージの CRC32 チェックサムが検索されます。

その後、同じ `03h` シリアルリードコマンドを使用して、第 2 ステージで自由に XIP (eXecute-in-Place) を設定できるため、RP2040 で、03h シリアルリードに対応した 24 ビットアドレスの`任意のチップ` (ほとんどの 25 シリーズフラッシュデバイスが対象になります) を使用して、キャッシュされたフラッシュの XIP を実行することができます。SDK には、CPOL=0 CPHA=0 用のサンプルの第 2 ステージが用意されています (https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/boot_stage2/boot2_generic_03h.S)。ブート ROM のルーチンを使用したフラッシュプログラミングを行う場合は、デバイス上で以下のコマンドを実行する必要があります。

- `02h`: 256 バイトページプログラム用のコマンド
- `05h`: ステータスレジスタの読み取りコマンド
- `06h`: 書き込み有効ラッチの設定コマンド
- `20h`: 4kB セクターの消去コマンド

RP2040 では、各種のデュアル SPI アクセスモードとデュアル QSPI アクセスモードがサポートされています。たとえば https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/boot_stage2/boot2_w25q080.S により、Winbond W25Q シリーズのデバイスがクアッド IO 連続読み取りモード用に設定されます。この場合、RP2040 からクアッド IO アドレスが送信され (コマンドプレフィックスは送信されません)、フラッシュからクアッド IO データが返信されます。

フラッシュ XIP モードの場合、Winbond 連続読み取りモードと同様に、フラッシュデバイスが標準的なシリアルコマンドに応答しなくなるため、注意する必要があります。フラッシュデバイスに電源が供給されていない状態で RP2040 をリセットすると、問題が発生する可能性があります。これは、ブート ROM のフラッシュプローブシーケンスにフラッシュが応答しなくなるためです。`03h` シリアルリードコマンドが実行される前に、以下に示す固定シーケンスがブート ROM によって実行されます。これは、各種のフラッシュデバイスで XIP を中止するための最善のシーケンスです。

- `CSn=1`, `IO[3:0]=4'b0000` (プルダウン経由で競合を回避)、x32 クロックを発行
- `CSn=0`, `IO[3:0]=4'b1111` (プルアップ経由で競合を回避)、x32 クロックを発行
- `CSn=1`
- `CSn=0`, `MOSI=1'b1` (Low-Z 駆動、その他すべての IO は Hi-Z 駆動)、x16 クロックを発行

このシーケンスに応答しない連続読み取りモードのデバイスの場合、各転送データにシリアルコマンドをプレフィックスとして付加する必要があります。これを行わなかった場合、内部リセット後に RP2040 の状態を復元できなくなります。

2.7. PCB を製作する

第章 2 は、最小設計の例です。意図的に銅を 2 層にし、上面だけに部品が配置されています。低コストで PCB を製作できるように、設計に関する制限を緩和しています。この設計は、Eurocircuits 社 (https://www.eurocircuits.com/) の標準的な PCB で動作することが確認されていますが、別の PCB メーカーが製造した PCB でも、ほとんど問題なく動作することが予想されます。

第章 3. Raspberry Pi Pico の VGA、SD カード、オーディオボード

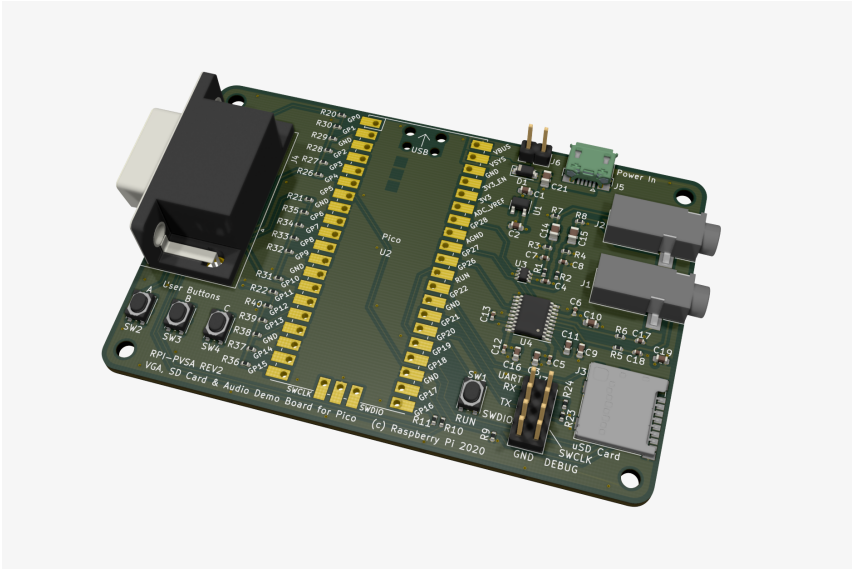


図 12. VGA SD カードの KiCad 3D レンダリングと Raspberry Pi Pico に対するオーディオ設計例

この設計例には、2 つの目的があります。1 つ目の目的は、Raspberry Pi Pico がモデルとして組み込まれた PCB の設計方法について説明するということです。この PCB は、大規模な設計における 1 つの部品として使用されます。 2 つ目の目的は、RP2040 の複雑なアプリケーションを正しく機能させるには、特定のハードウェアを追加する必要があるということを示すことです。ここでは、VGA ビデオ、SD カード*ストレージ、*アナログ PWM オーディオ出力、デジタル I2S オーディオ出力という 4 つのアプリケーションの設計例を紹介します。これらの機能を使用するサンプルソフトウェアは、 Pico Playground に公開されています。

この設計は Raspberry Pi Pico を使用することを前提としていますが、Raspberry Pi Pico から RP2040 のピンに直接アクセスできるため、この章に記載されている回路の多くは、RP2040 をベースとした設計でも同様に使用することができます。

KiCad 用の回路図とレイアウトファイルは、 https://datasheets.raspberrypi.com/rp2040/VGA-KiCAD.zip で入手することができます。KiCad は、PCB を設計するための無料のオープンソースツールスイートです。 https://kicad.org/ で入手することができます。

Raspberry Pi Pico と RP2040 で設計する場合の重要な違いの 1 つは、RP2040 のすべての IO を Raspberry Pi Pico で使用できるわけではないということです。これは、いくつかの IO が内部のハウスキーピング処理で使用され (電源の制御、電源の監視、LED の管理など)、外部の処理では使用されないためです。ここで設計するサンプルアプリケーションでは、Raspberry Pi Pico で使用可能なピンよりも多くのピンが必要になるため、いくつかのピンが使用できないというのは難しい問題になります。しかし、解決策はあります。3 つのユーザーボタンと UART 接続を追加するという方法です。これについては、後で詳しく説明します。

回路図、PCB のレイアウト、Raspberry Pi Pico のフットプリントファイルは KiCad 形式で提供され、第章 2で説明した最小設計例と同様の設計ルールが適用されます。前章の最小設計例では、厚さが 1mm で 2層構造になっている PCB を使用しましたが、ここでは、厚さが 1.6mm で 4層構造になっている PCB を使用します。層の数を増やすことにより、すべての層を電源と接地に割り当てることができるようになります。これは、いくつかの点において重要な意味を持ちます。1 つ目は、電源のバイパス性能が向上するということです。4 層構造にすることにより、並行する 2 つの大きな長方形の銅トラックを配置し、一方を電源に接続し、もう一方を接地することができます。これらの銅トラックを薄い

誘電体 (銅層に挟まれた絶縁 PCB 層) で分離することにより、シンプルな構造の並列プレート型バイパスコンデンサーが完成します。 2 つ目は、RP2040 に戻る複数の低インピーダンス経路で、電磁放射の原因となる電流ループが発生することなく、急激に変化する IO リターン電流がスムーズに流れるようになるということです。これが、この設計における最も重要な点です。 3 つ目は、最上層の信号トラックとその直下の接地プレーンとの間のギャップが少なくなるため、設計上必要な複数の特性インピーダンスのトラックを非常に簡単に作成できるということです。この設計の VGA は 75Ω のシステムであるため、75Ω のケーブルと 75Ω の負荷終端をモニター内で使用して、VGA カラー信号用の 75Ω のトラックを作成する必要があります。

この設計は、電源、VGA、SD カード、オーディオ、Raspberry Pi Pico 本体 という 5 つのセクションに分割することができます。

3.1. 電源

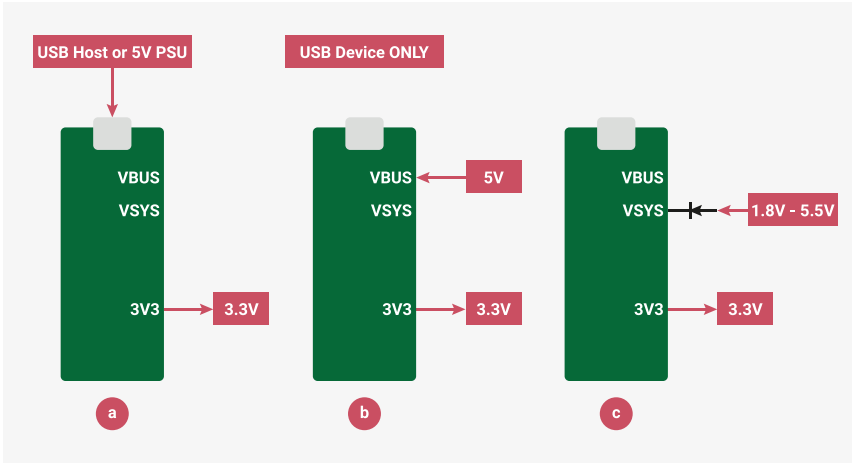


図 13. Raspberry Pi Pico の推奨電源投入方法

Raspberry Pi Pico に安全に電力を供給する方法は 3 つありますが、どの方法にするかについては、用途に合わせて自由に選択することができます。Raspberry Pi Pico に電源を供給するには、Micro USB コネクター*を使用するか (図 13 の a)、*VBUS ピン を使用するか (b)、VSYS ピン を使用します (c)。

注記

3V3 ピンは Raspberry Pi Pico からの出力であるため、外部電源に接続しないでください。このピンは、外部回路に電力を供給するための出力として設計されています。

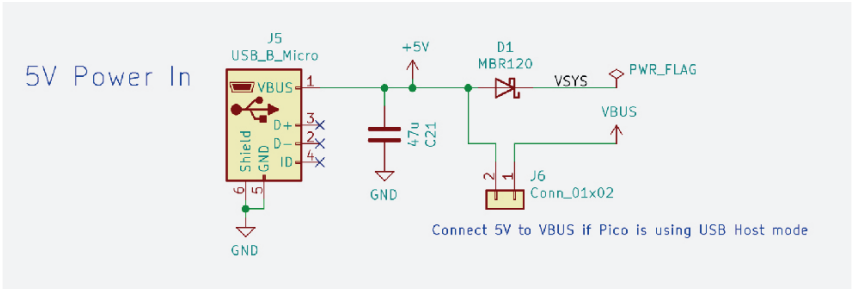


図 14. 入力電源の回路図

*VSYS ピン*は、Raspberry Pi Pico のメインシステム電源供給ピンです。Raspberry Pi Pico は、この電源供給ピンから 3.3V 電源を生成します。生成された電源は、RP2040 の電源として使用されます。Raspberry Pi Pico の 3V3 出力ピンを使用して、この設計の回路に電源を供給することができます。

*VBUS ピン*は、Raspberry Pi Pico の Micro USB コネクターの VBUS に接続されています。ボードには、VBUS と VSYS

を接続するダイオードが配置されているため、VBUS から VSYS に対して電源を供給することができます。ただし、VSYS から VBUS に対して電源を供給することはできません。

この設計では、いくつかの方法で電力を供給できますが、どの方法にするかについては、用途によって異なります。最初に、Raspberry Pi Pico の USB 機能を使用するかどうかを検討する必要があります。

USB を使用しない場合

USB を使用しない場合は、Raspberry Pi Pico に電力を供給する必要があります。これを行うための第 1 の方法として、Raspberry Pi Pico のピンを使用して、ボードから Raspberry Pi Pico に電力を供給します (図 15 を参照)。その際、ショットキーバリアダイオードを使用して VSYS ピン に電圧を供給することをお勧めします (図 14)。ダイオードの一方向性により、Raspberry Pi Pico の VBUS ピンにも電力を供給しても問題ありません。Raspberry Pi Pico には、降圧レギュレーター (出力電圧を入力電圧とは違う値に設定できるレギュレーター) が内蔵されているため、電圧の範囲を 1.8～5.5V の範囲で変化させることができます。ただし、この設計には電圧レギュレーターが追加されているため (図 17 の U1、詳細については後述)、U1 が正しく動作するように、VSYS の電圧を 3.5V よりも高くする必要があります。

第 2 の方法として、VSYS ピンではなく、Raspberry Pi Pico の VBUS ピン (Raspberry Pi Pico の USB コネクターの VBUS 接続と混同しないでください) に電源を供給することもできます。この場合、Raspberry Pi Pico のオンボードダイオードを使用して内部的に VSYS に電力を供給することになりますが、Raspberry Pi Pico の USB コネクターに別の電源を接続しないようにする必要があります。

この設計では、Micro USB コネクター (図 14 の J5) を使用して、5V の入力電源を供給します。このコネクターは、最大 1A を供給する MBR120 ショットキーバリアダイオード (上図の D1) 経由で VSYS に接続されています。VBUS ピンに電力を供給する必要がある場合は、オプションのジャンパー (上図の J6) を使用しますが、USB を使用しない場合は必要ありません。

第 3 の方法として、ボードの USB コネクターではなく、Raspberry Pi Pico の USB コネクターに 5V 電源を接続することもできます (これ以降の「デバイスモード」セクションの説明と図 16 を参照)。

USB を使用する場合

USB を使用する場合は、*ホスト*モードと*デバイス*モードのどちらで使用するかを判断する必要があります。

デバイスモード

Raspberry Pi Pico をデバイスモードで使用する場合、接続されたホストにより、USB コネクターの VBUS ピンに 5V の電圧が供給され、次にその VBUS ピンから VSYS に対して、内部的に電圧が供給されます (5V の電圧からボード上のダイオードのドロップアウト電圧を減算した電圧)。この設計の場合、電圧に関する要件はこれですべてですが、USB ホストが接続されている場合にのみ電力が供給されることに注意する必要があります。図 16 を参照してください。 ホストモードの USB から供給される 5V 電力を使用せずに、ボード本体で電力を供給する場合は、キャリアボードから電力を供給する必要があります。その場合は、5V 電源を Micro USB コネクター (上図の J5) に接続し、Raspberry Pi Pico の VSYS ピン*に約 5V の電力を供給します。その際、2 つの 5V 電源が直接接続される可能性があるため、ジャンパー (上図の *J6) がオープン回路になっていることを確認する必要があります。図 15 を参照してください。

ホストモード

USB をホストモードで使用する場合は、Raspberry Pi Pico の Micro USB コネクター (上図の J5 ではないことに注意) の VBUS ピンに 5V の電力を供給する必要があります。*そのため、キャリアボードを設計する場合は、Raspberry Pi Pico の *VBUS ピン*に 5V の電力を供給し、Raspberry Pi Pico にも電力を供給する必要があります。この設計でこれを行うには、Micro USB コネクター (上図の *J5) を 5V 電源に接続し、上図の J6 の位置にジャンパーを取り付けて、この 5V 電源を Raspberry Pi Pico の VBUS ピンに直接接続します。VSYS は、Raspberry Pi Pico ボードのダイオードと、この設計のダイオード (上図の D1) との組み合わせによって供給されますが、安全性にはまったく問題はありません。

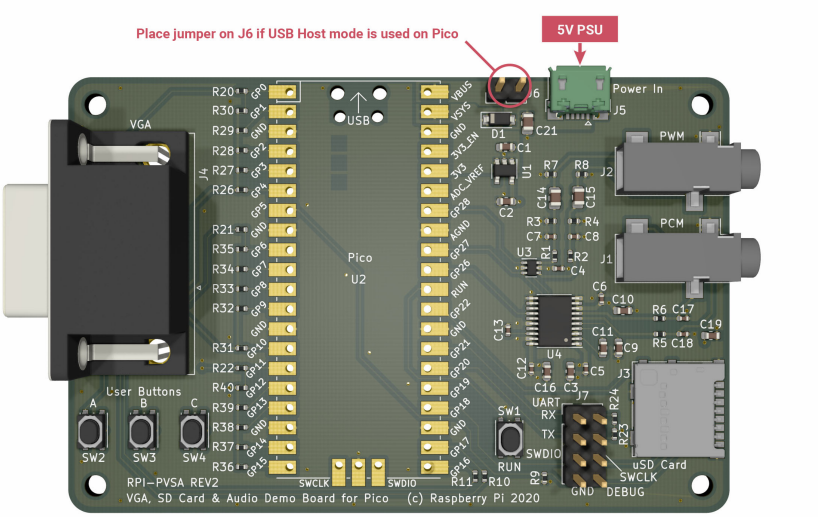


図 15. VGA、SD カード、オーディオボードの USB 電源コネクターを使用してシステムに電力を供給する

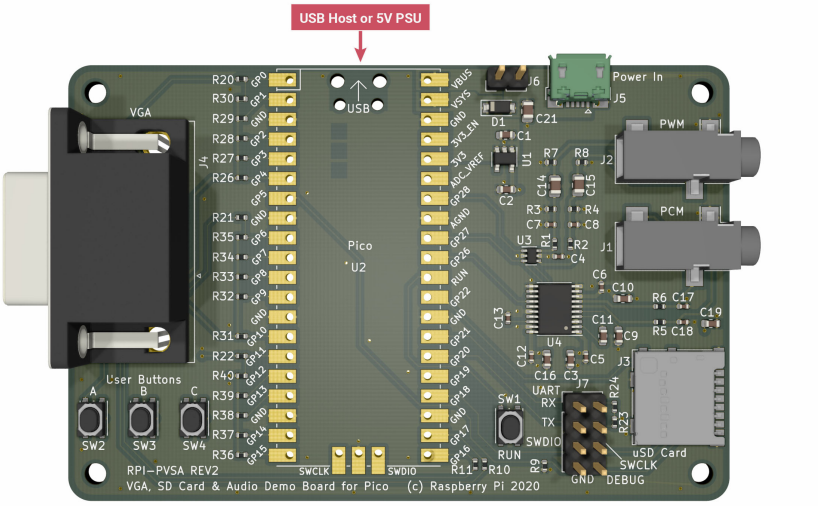


図 16. Raspberry Pi Pico の USB コネクターを使用して電力を供給する

3.1.1. 自動給電

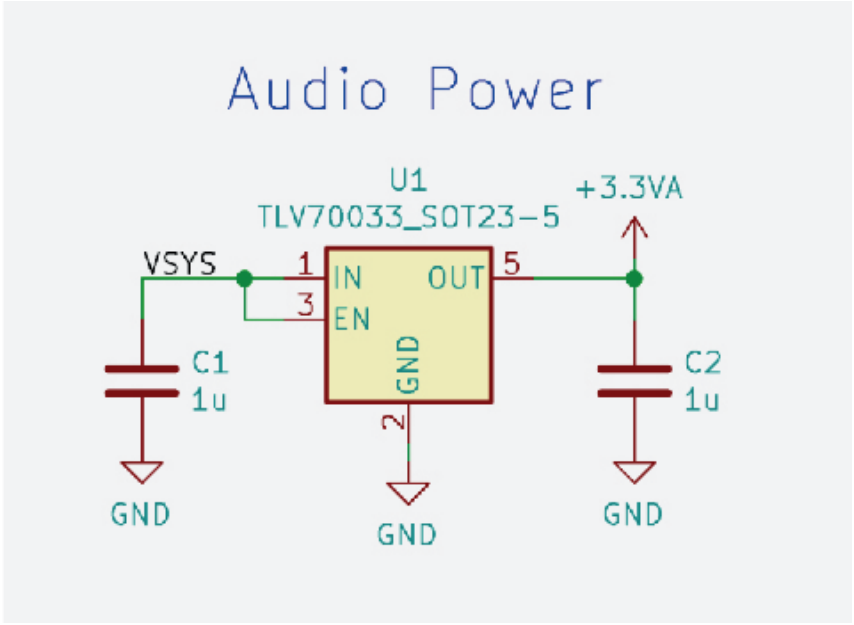


図 17. オーディオ電源として使用される追加の LDO の回路図

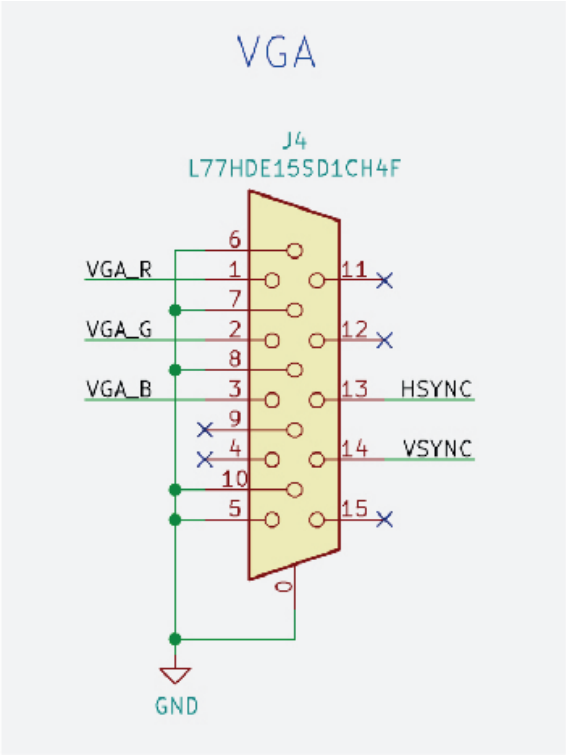
この設計では、Raspberry Pi Pico に電源を供給するだけでなく、適切な電源供給を行うための回路が必要になります。これらの電源はすべて 3.3V であるため、Raspberry Pi Pico で 3V3 を使用するだけで済みます。ただし、この設計にはオーディオ回路があるため、オーディオ出力部では、デジタルスイッチングノイズが発生しないクリーンな電源を使用する必要があります。そのためこの設計では、3.3V のリニア電圧レギュレーター (図 17 の U1) をオーディオ出力用として使用します。このレギュレーターの電力は、VSYS によって供給されます (VBUS とは異なり、VSYS は必ずボード上に搭載されています)。このレギュレーター (TLV70033) は、出力が 3.3V に固定されている LDO (Low Drop-Out: 低ドロップアウト) レギュレーターです。このレギュレーターが供給する電流は 200mA ですが、この設計で使用するオーディオ回路には十分な容量です。TLV70033 のデータシートには、入力ピンと出力ピンで 1µF のコンデンサーが必要になると記載されています。この設計では、0603 サイズのコンデンサーを使用しています (上図の C1 と C2)。

注記

Raspberry Pi Pico で使用されているスイッチングレギュレーターには、2 つの動作モードがあります。レギュレーターを流れる電流の量により、これらのモードが切り替わります。低電流モード (数十 mA 以下) の場合、処理効率を上げる目的で、PFM (Pulse Frequency Modulation: パルス周波数変調) による省電力機能を使用してレギュレーターが動作します。負荷が低い場合、Raspberry Pi Pico の消費電力が低下し、処理効率が上がります。ただし、3V3 電源の電圧リップルが多少大きくなります。ほとんどの場合、これが問題になることはありませんが、ノイズに敏感な回路では、この省電力機能をオフにして (処理効率は下がりますが)、ノイズの少ない PWM (Pulse Width Modulation: パルス幅変調) モードに切り替えることをお勧めします。Raspberry Pi Pico でこれを行うには、常に PWM モードを使用するようにレギュレーターを設定し、RP2040 の GPIO 23 で高い値を設定します。VGA モニターを注意して確認すると、電源ノイズが DAC 出力に直接送信され、水平線の色がわずかに変化するのがわかります。レギュレーターの PFM モードを無効にすると、この問題が解決します。

3.2. VGA ビデオ

図 18. VGA ビデオコネクターの回路図



RP2040 の最初のアプリケーションとして、VGA アナログビデオの出力アプリケーションについて説明します。ここでは、RP2040 の PIO (Programmable IO: プログラマブル IO) を使用して、一般的な 16 ビット RGB データ形式 (RGB-565) を出力します。このデジタル出力を、3 つのアナログ出力信号に変換する必要があります (1 つの色について 1 つの信号)。RGB-565 は、赤のチャンネルと青のチャンネルで 5 ビットを使用し、緑のチャンネルで 6 ビットを使用します (全体として 16 ビットのデータになります)。VGA モニターでは、この 16 ビットデータだけでなく、水平/垂直ブランキングタイミング用の HSYNC 信号と VSYNC 信号も必要になります。そのため、必要なピンは合計で 18 本になります。これまでに説明したようにピンの数は限られているため、できるだけ少ないピンで、より多くの機能を使用できる設計にする必要があります。そのため、緑の LSB (Lowest Significant Bit: 最低位ビット) を削除して、緑のチャンネルのビット数を 5 ビットに制限します。これにより、すべてのチャンネルが同じ解像度になるため、1 ビット分のピンが解放されます。RP2040 では、RGB-565 形式のデータを処理することが望ましいため、PIO は 6 ビットの緑データを GPIO に出力しますが、出力先の GPIO の機能選択レジスターではなく RP2040 で緑の LSB を使用することができます (ここでは、SD カードのクロックに対して緑の LSB を使用します)。 この設計には、使用できるピンの数のほかにも制約があります。それは、VGA PIO ソフトウェアで、連続する GPIO にすべての 16 ビットデータを送信しなければならないということです。その際、赤、緑、青の順に LSB を送信します。 Raspberry Pi Pico には、2 つの連続した GPIO グループがあります。1 つは GPIO 0 ～ 22 で、もう 1 つは GPIO 26 ～ 28 です。VGA データは、0 ～ 22 までのいずれかの GPIO に格納する必要がありますが、その他の機能用としてできるだけ多くの連続したピンを空けておくために、最初は先端または終端の GPIO にデータを格納するのが効率的な方法です。ここでは、GPIO 0 ～ 15 を使用するため、緑の LSB は GPIO 5 になり、これが SD_CLK として使用されます。 HSYNC と VSYNC については、隣接していればどの GPIO でもかまいません。ここでは、GPIO 16 と 17 を使用します。

3.2.1. 抵抗 DAC

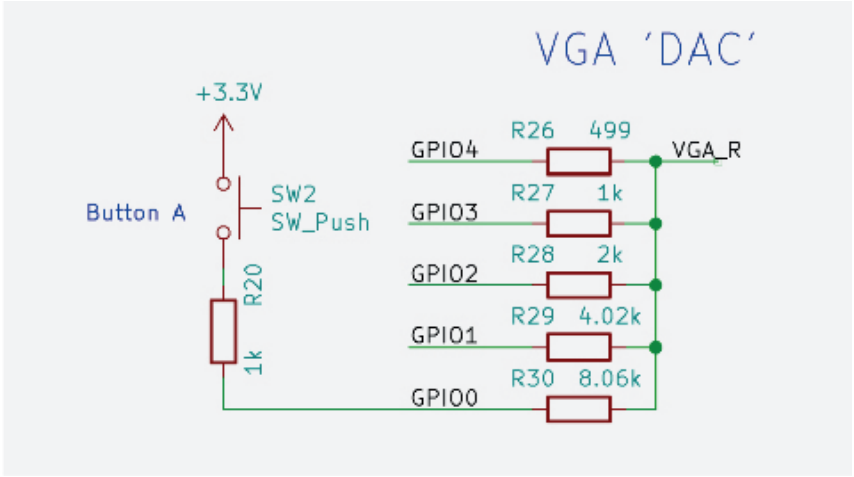


図 19. VGA 抵抗 DAC の赤のチャンネルの回路図

VGA コネクターの 3 つのカラーチャンネルは、0 ～ 0.7V のアナログ信号でなければなりません。そのため、RP2040 のデジタル 3.3V 出力をアナログ信号に変換する必要があります。専用のビデオ DAC (Digital to Analogue Converters: デジタル - アナログ変換器) を使用してアナログ信号に変換することもできますが、ここでは、より安価でシンプルな方法を紹介します。デジタル出力に直接接続された一連の抵抗器を使用して、シンプルな DAC を作成することができます。抵抗器の値は、1:2:4:8:16 の割合になっています。各ビットの重要度を設定するために、これらの値には重みが付けられています。この方法は、専用のビデオ DAC を使用する場合と比べて、変換時の品質が下がります。この方法の主な欠点として、RP2040 の IOVDD 電源の電圧変動が DAC 出力に影響するという点が挙げられます。しかし、コストが低く、構造もシンプルで、作業が楽しいというメリットがあります。 赤のチャンネル (上図の VGA_R) を見ると、赤の LSB (GPIO 0) が 8.06kΩ の抵抗器を経由して赤のチャンネルに接続されていることがわかります。次のビット (GPIO 1) の抵抗値は GPIO 0 の約半分 (4.02kΩ) になり、その次のビットはさらにその半分になります。残りのビットについても、同じように抵抗値が半分になっていきます。直線的な DAC 性能を実現するには、抵抗値を正確に 2 倍ずつ増やしていくことが理想的ですが、通常は、誤差が 1% の範囲に収まっていれば問題ありません (この設計でもそうなっています)。抵抗値を 2 倍ずつ増やしていくと、各 GPIO の出力ビットで、前のビットの 2 倍の電流が抵抗器を通過し、これらの電流がすべて合計されて出力されることになります。 その結果、最大デジタル値に対応するすべてのビットが 3.3V になり、5 つの抵抗器がすべて 3.3V 電源に対して並列接続されることになります。これを基本的な回路理論で表すと、次のような式になります: 1/Rparallel = 1/499 + 1/1000 + 1/2000 + 1/4020 + 1/8060 = 0.00388 (Rparallel の値は 258Ω になります)。 この信号にモニターを接続すると、そのモニターの内部で 75Ω の抵抗器が接地されることになります (上記の回路図には記載されていません)。これによって電位差が発生し、3.3V が 258Ω に接続され、次に 75Ω に接続されて、モニター内で接地されます。この場合のフルスケール電圧は、3.3 * 75 / (258 + 75) = 0.74V となります。これは、目標値の 0.7V に十分に近い値です。

3.2.2. ユーザーボタン

厳密には、ユーザーボタンは VGA の一部ではありませんが、この設計ではユーザーボタンを追加します (図 19 の SW2)。このボタンは、赤のチャンネル、緑のチャンネル、青のチャンネルの LSB に接続されます。このボタンをソフトウェア内でどのように使用するのかを理解することが重要です。VGA、SD カード、オーディオボードにより、1 つまたは 2 つのボタンを使用するさまざまなアプリケーション (ビデオや音楽用のコントロールなど) を開発できるため、この設計でボタンを追加してその仕組みを理解することが重要になります。これまでに説明したように、使用できるピンの数は制限されていて、ボタンのような重要度の低いものにピンを割り当てる余裕はないため、この設計では、VGA の LSB を複数の目的で使用することにします。

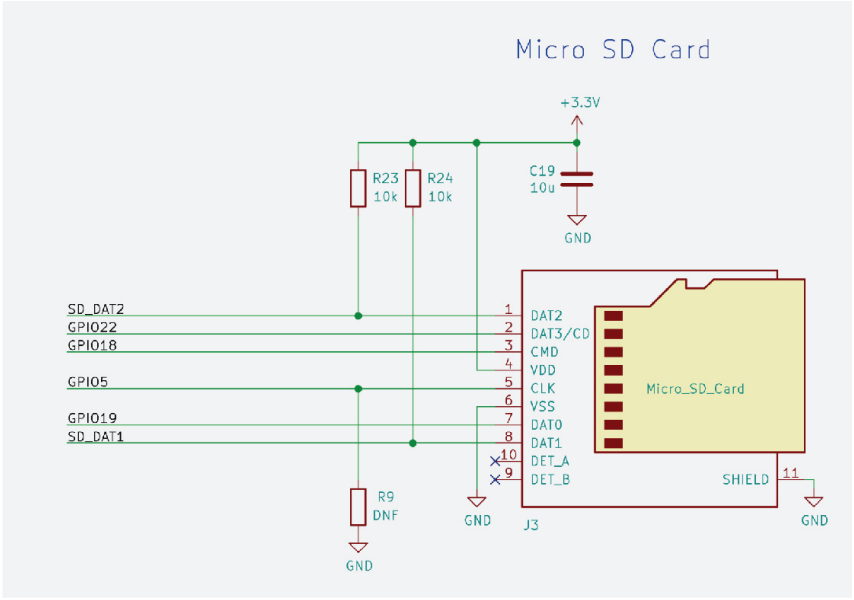
この場合、「ビデオデータがアクティブな状態になっている場合は、GPIO を通常どおりに VGA で使用し、ビデオデータがブランクになっていて、DAC レベルの重要度が低い場合は、GPIO を入力方向に反転させてポーリングを実行し、次のビデオデータがアクティブになったときに、GPIO を出力方向に反転させる」というのが基本的な考え方になりま

す。SW2 ボタンを押すと、GPIO 0 が 1kΩ の抵抗器の分圧回路 (上図の R20) を経由して 3.3V に接続され、8.06kΩ の抵抗器 (上図の R30) が 0V に接続されます (最悪の場合)。この場合、GPIO 0 には 2.93V 以上の電圧が供給されます。これが、ロジック 1 として登録されます。SW2 ボタンを押さなかった場合、GPIO 4 には 0.7V 以下の電圧が供給されます。これが、ロジック 0 として登録されます。プルダウン抵抗の場合、この動作はモニターの 75Ω の負荷抵抗器によって異なります。モニターを使用しない場合は、代わりに GPIO の内部プルダウン抵抗をアクティブにすることができます。

アクティブなビデオデータの転送中にボタンを押した場合は、DAC の信号レベルが影響を受けることが予想されます。ただし、LSB が干渉を受けるだけであるため、DAC 信号に対する影響は最小限になります。1kΩ の抵抗器 (上図の R20) が SW2 ボタンに直列接続されているため、データの転送中にボタンを押しても RP2040 ではほとんど問題はなく、DAC 信号に対する影響も最小限になります。 DAC に関する最後の注意点は、これまでに説明したように、出力側に 75Ω の特性インピーダンスを持たせる必要があるということです。この PCB は、厚さが 1.6mm で 4 層構造になっていて、外層と内層のギャップは 0.36mm です。そのため、トラックの幅を 0.3mm にすると、抵抗値が約 75Ω になります。

3.3. SD カード

図 20. Micro SD カードコネクターの回路図

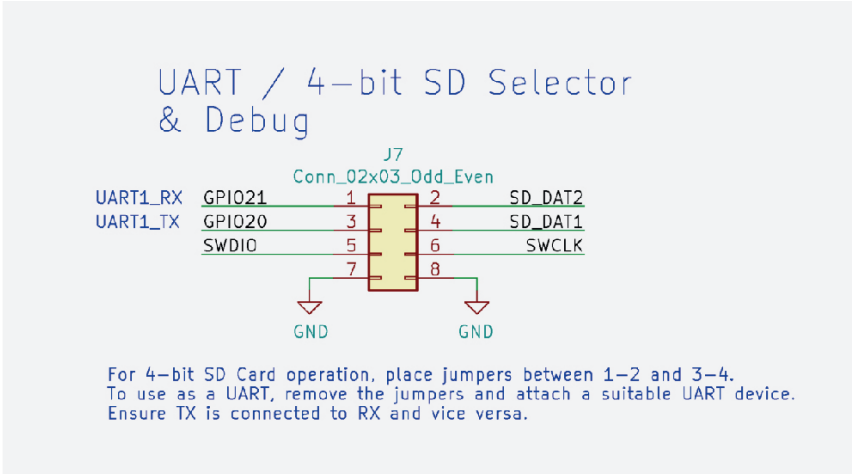


2 つ目のアプリケーションとして、SD カードを使用するアプリケーションについて説明します。この設計には、4 ビットのデータバス、クロック信号 (CLK)、コマンド信号 (CMD) を持つ Micro SD カード (上図の J3) が配置されています。4 ビットモード、SPI モード、1 ビットモードのいずれかを使用して、この SD カードにアクセスすることができます。SD PIO ソフトウェアの制約事項として、4 つのデータ信号を連続した GPIO に接続する必要があります。CLK 信号と CMD 信号については、送信先の制約はありません。GPIO 0 ～ 17 は VGA 信号に割り当てられていますが、18 ～ 22 という連続した 5 つの GPIO が未割り当てとして残っているため、ここでは、GPIO 19 ～ 22 をデータバスに、GPIO 18 を CMD 信号に割り当てることにします。CLK 信号については、VGA 信号の経路内にある GPIO 5 を割り当てます。GPIO 5 は、緑の VGA 出力の 6 番目の未使用ビット用として残されていたピンです。GPIO mux で別の機能を選択すると、この GPIO を別の目的で使用できるようになります。この GPIO は、SD CLK に自由に割り当てることができます。 多くの場合、PCB 上の SD インターフェイスには、プルアップ抵抗とプルダウン抵抗が配置されています。その理由は、常に安全な値を確保し (特に IO が定義されていない場合)、SD 信号の一部をモード選択ピンとして使用するためです (SPI モードや 1 ビットモードなど)。この設計では、RP2040 を使用して、GPIO のプルアップ抵抗とプルダウン抵抗を設定します。CLK 入力が常に正しい状態になっていることが重要であるため、特定の用途で CLK 信号が必要になる場合を考慮して、CLK 信号のプルダウン用オプションを追加しています (上図の R9)。実際には、SD データバスのビ

ット 1 とビット 2 にプルアップ抵抗が配置されています (上図の R23 と R24)。その理由は、SD カードの IO が Raspberry Pi Pico に直接配線されておらず、代わりにジャンパーヘッダー (上図の*J3*のピン 1、2、3、4) 経由で接続されているためです。そのため、ジャンパーを取り外しても、SD カードの IO で適切なレベルを確保することができます。4 ビットを動作させる場合は、最初にジャンパーを接続する必要があります。これまでに説明したように、SPI モードまたは 1 ビットモードで SD カードにアクセスできるため、いずれかのモードを使用して、データビット 1 と 2 を別の目的で再利用できる可能性があります。

3.3.1. UART

図 21. オプションの UART と SWD デバッグヘッダーの回路図



SD カードを 1 ビットモードで使用する場合や、SD カードをまったく使用しない場合は、その分のピンを UART で自由に使用することができます。その場合、4 ビットの SD カードを動作させるために必要なジャンパーではなく、上図の J7 のピン 1 とピン 3 に 3.3V 対応の UART を接続するだけで済みます。専用のハードウェア UART コントローラーを使用する場合、通常は GPIO 21 が UART1_RX になり、GPIO 20 が UART1_TX になりますが、PIO UART を実装する場合は、TX と RX を選択して設定することができます。

3.3.2. デバッグ - SWD

J7 は、この設計の SWDIO ピンと SWCLK ピンの実装場所でもあります。デバッグピンを接続するための手段としてこの PCB が Raspberry Pi Pico に接続されている場合、ヘッダー上のデバッグピンを使用してデバッガーを接続することができます。デバッガーを Raspberry Pi Pico 本体に直接接続することもできます。

3.4. オーディオ

この設計では、RP2040 に対応した 2 つのオーディオオプションであるアナログ PWM とデジタル PCM/I2S を使用します。ただし、それぞれのソリューションに個別のピンを割り当てる余裕はないため、この 2 つのオプションでは同じピンを使用します。どちらのオプションを使用するかについては、ソフトウェアによって決定されます。使用しないオプションを間違ったモードで実行しても問題が発生することはないため、この設計には、両方のオーディオオプションの回路が実装されています。

注記

オーディオ出力機器を正しいジャックに接続する必要があります。PCM は J1 に接続し、PWM は J2 に接続します。

オーディオ用の出力は、アンプのラインイン入力に接続して、ラインレベルドライバーとして使用しますが、インピーダンスが高いヘッドフォンでも使用することができます。 この時点で残っている GPIO は 26 ～ 28 ですが、これだけのピンがあれば、PWM と PCM に対応することができます (PWM を使用する場合は 2 本、PCM を使用する場合は 3 本のピンが必要になります)。

3.4.1. PWM オーディオ

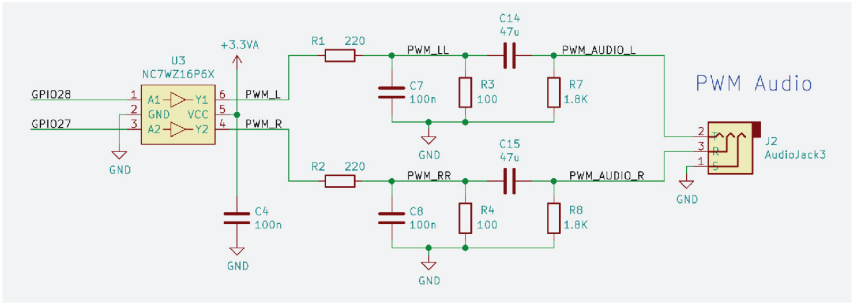


図 22. アナログ PWM オーディオ回路の回路図

最初に、アナログ PWM 方式について説明します。この方法は、Raspberry Pi 4 のオーディオ出力端子で使用されている方法です。そのため、回路も同じものになっています。デジタルオーディオが 2 つの GPIO ピンからデジタル PWM (パルス幅変調) 信号として出力されます。ステレオペアごとに、デジタル PWM 信号が 1 つ出力されます。このデジタル信号は，小型のロジックバッファ (上図の U3) に送信されます。これは、クリーンなオーディオ用 3.3V 電源を使用するためです。正しく設計すれば、3.3V 主電源のデジタルノイズがオーディオ信号に混入することはありません。バッファリングされた出力は、3.3V のデジタル信号としてアナログフィルターに送信されます。これにより、可聴周波数帯域の交流結合アナログ信号が生成され、この信号をアンプやヘッドフォンに接続できるようになります。

3.4.2. PCM/I2S オーディオ

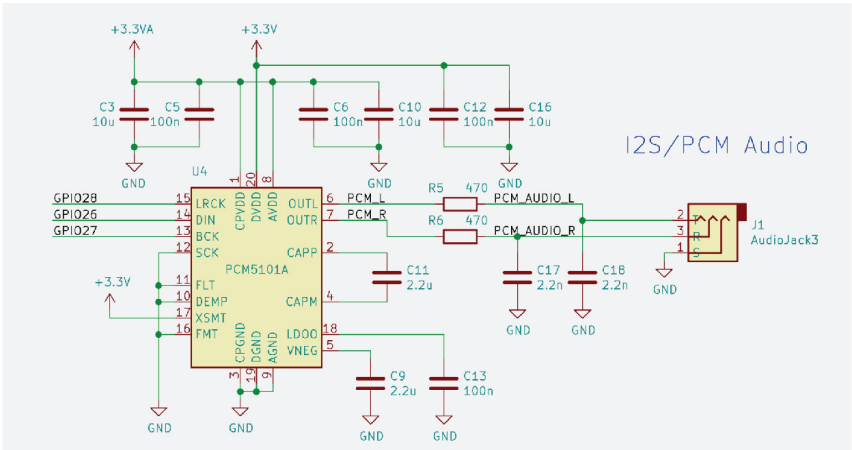


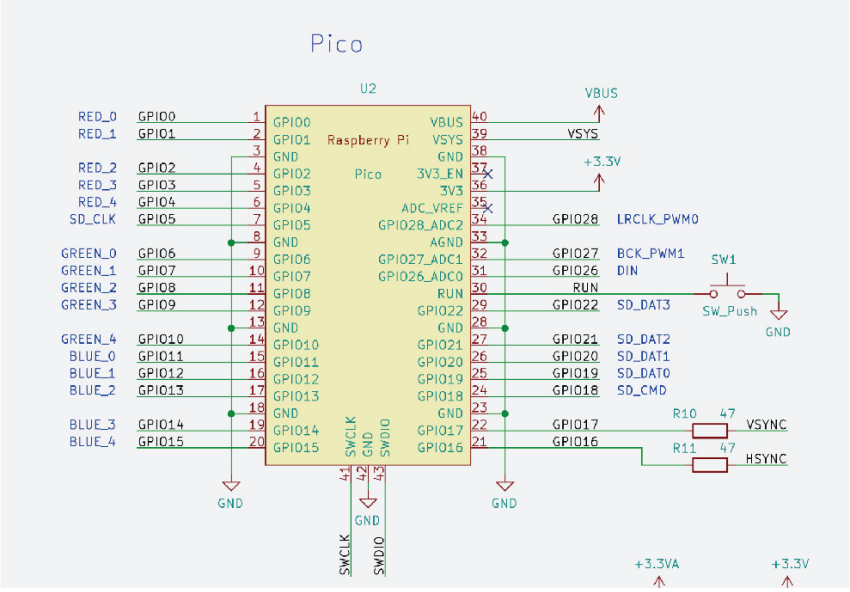
図 23. デジタル I2S PCM オーディオ回路の回路図

2 つ目のオーディオオプションは、I2S を使用するデジタル PCM です。これは、PCM (Pulse Code Modulation: パルス符号変調) 形式のデジタル音声を I2S プロトコルでオーディオ DAC に送信し、オーディオ出力端子に接続するという方法です。この設計では、PCM5101A オーディオ DAC を使用します。GPIO 27 は BCK 入力 (ビットクロック)、GPIO 26 は

DIN (シリアルデータ)、GPIO 28 は LRCK (左クロックまたは右クロック) に接続されています。DAC の周囲に配置された残りの回路は、PCM5101A データシートの標準的なアプリケーション回路に準拠しています。

3.5. Raspberry Pi Pico

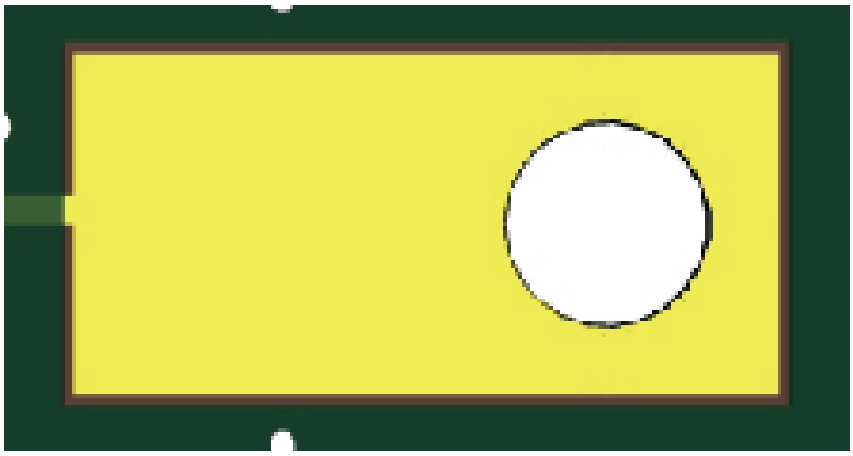
図 24. Raspberry Pi Pico に対する接続の回路図



この設計の最後の構成要素は、Raspberry Pi Pico 本体です。これまでに、Raspberry Pi Pico の接続の大部分について説明しましたが、まだ説明していないピンがいくつかあります。電源ピンの接続方法についてはすでに説明しましたが (VBUS 入力、VSYS 入力、3V3 出力)、GND についてはまだ説明していません。ノイズと EMC エミッションを最小限に抑えるため、ボード上の*すべて*の GND ピンを接地する必要があります (低インピーダンスの接地プレーンに接続するのが理想的です)Raspberry Pi Pico には、低ノイズ ADC のリターンパスとして使用するための AGND ピンがありますが、このアプリケーションでは ADC を使用しないため、このピンを通常の GND に接続するだけで済みます。 また、Raspberry Pi Pico のオンボード 3.3V 電源の代わりに、クリーンで安定した電源を供給するためのオプションの ADC_VREF ピンもありますが、このアプリケーションでは ADC を使用しないため、このピンはフローティングピンとして残しておきます。 Raspberry Pi Pico の RUN ピンは、RP2040 の reset_n ピン (アクティブロー) です。このピンは、Raspberry Pi Pico で High にプルアップされていますが (RP2040 が稼働している状態)、プッシュボタン (上図の SW1) を追加すると、このピンを Low にプルダウンして RP2040 をリセットすることができます。 Raspberry Pi Pico の最後のピンは 3V3_EN ピンです。このピンにより、Raspberry Pi Pico の 3.3V 電源の供給が制御されます。このボードでは、この電源を無効にする必要がなく、Raspberry Pi Pico 本体にプルアップ機能が実装されているため、RUN ピンをフローティングピンとして残しておくことができます。

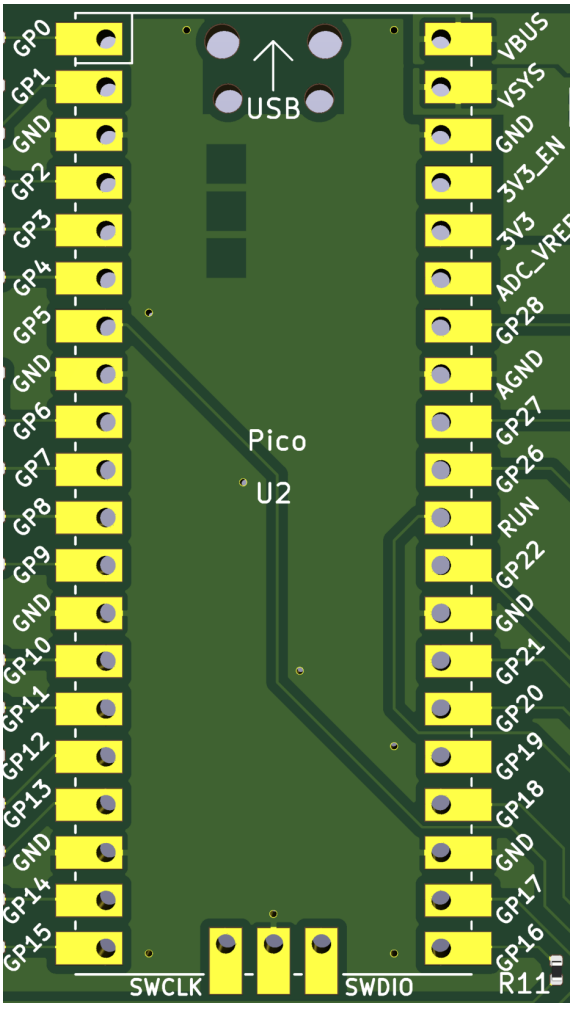
最後に、Raspberry Pi Pico 本体の取り付け方法について説明します。これが、この設計における最も重要な部分です。この設計では、2 つの方法があります。

図 25. 表面実装とスルーホールパッドの拡大図



Raspberry Pi Pico の各ピンは、2 つの方法ではんだ付けを行うことができます。スルーホールを使用して 0.1 インチのヘッダーをはんだ付けすることも、PCB 上で直接ピンをはんだ付けすることもできます。Raspberry Pi Pico にはキャステレーションエッジ (ピンがボードの端まで伸びていて、そこから PCB 本体の端まで伸びているエッジ) があるため、PCB 上で直接はんだ付けを行うことができます。

図 26. Raspberry Pi Pico の表面実装とスルーホールフットプリント



この設計の CAD フットプリントにより、両方の取付方法に対応することができます。Raspberry Pi Pico をこの設計に直接はんだ付けすることも、0.1 インチヘッダー (実際には、0.1 インチヘッダーとソケットの組み合わせ) を使用して 2 つの PCB を一緒に接続することもできます。

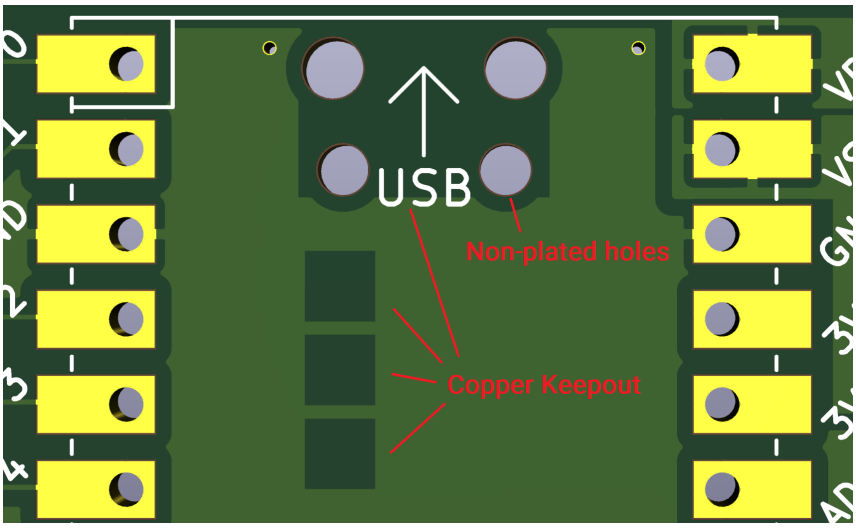


図 27. Raspberry Pi Pico の USB コネクター下部の穴と銅のキープアウト領域、テストポイント

このフットプリントのもう 1 つの特徴は、ボードの上部 (Raspberry Pi Pico の Micro USB コネクターの真下) にある 4 つのドリル穴です (図 27 を参照)。これは、Raspberry Pi Pico がキャリア PCB に対して水平になるようにするための穴です。USB コネクターを Raspberry Pi Pico に固定するためのスルーホール金具が、ボードからわずかに突き出していることがありますが、そうした場合にこれらの穴を使用します。これらの穴を使用して、余分な金属部分やはんだを安全に処理することができます。これらの穴のほかにも、銅層のキープアウト領域があることがわかります。Raspberry Pi Pico の下部にはテストポイントがありますが、これらのテストポイントは、製造テストで使用される銅層の露出部分です。銅層のキープアウト領域は、これらのテストポイントに対応しています。PCB の表面にはソルダーレジスト (PCB の表面にある緑色の絶縁膜) が残っているため、厳密には銅層のキープアウト領域は必要ありませんが、銅層のキープアウト領域を確保することにより、不慮の事故や PCB の加工不良でテストポイントがショートする可能性がほぼゼロになります。ただし、これが該当するのは、Raspberry Pi Pico をボードに直接はんだ付けした場合だけです。ヘッダーを使用して Raspberry Pi Pico を取り付ける場合は、銅層のキープアウト領域と USB 用の穴は必要ありません。

注記

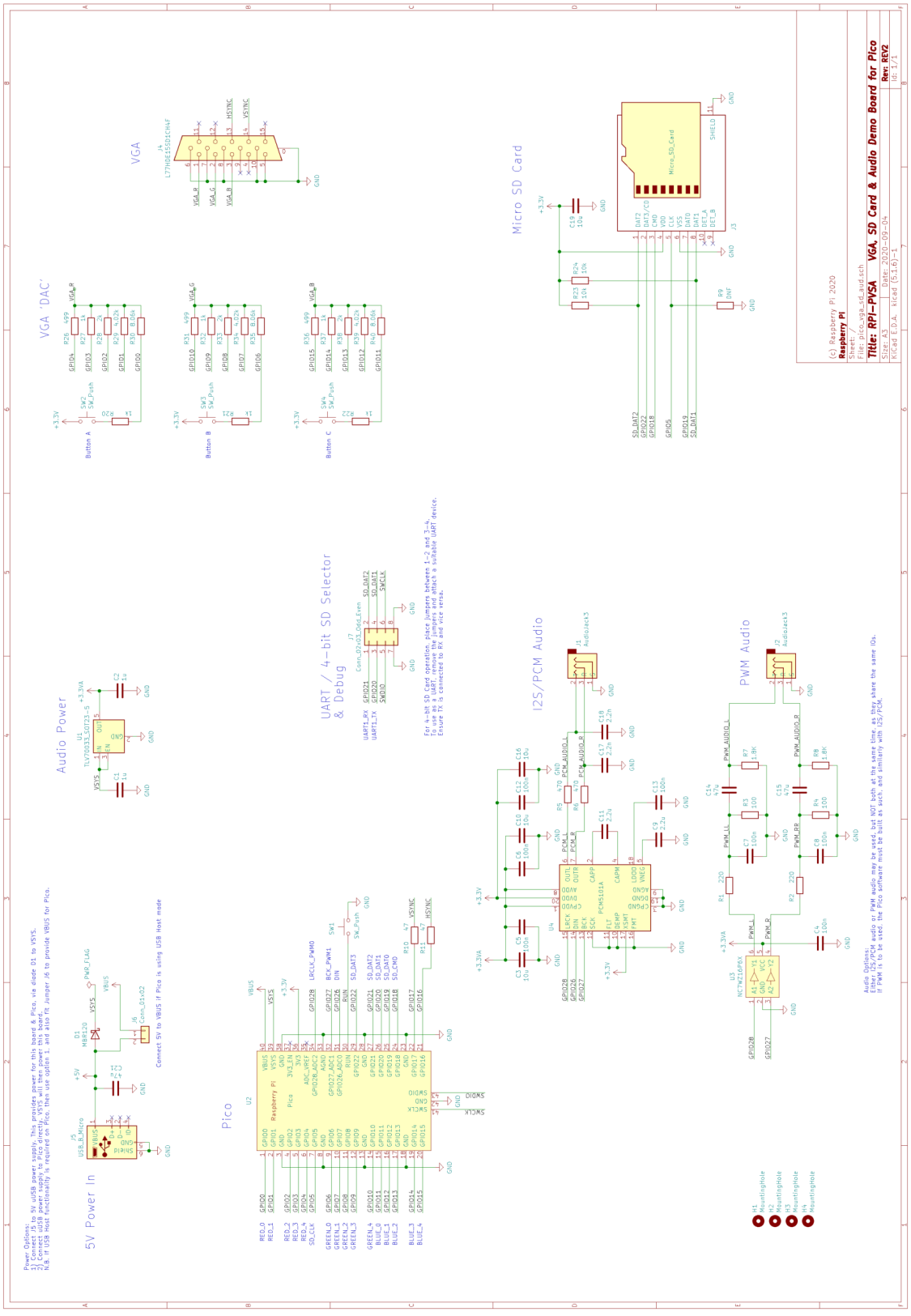
KiCad には、キープアウト層はありません。dwgs.user 層でキープアウト領域を表示することをお勧めします。その場合、PCB のレイアウト上で銅層を手動で削除する必要があります。

ここで、コンポーネントのキープアウトについて説明します。Raspberry Pi Pico をボードに直接はんだ付けする場合は、フットプリント全体でコンポーネントのキープアウト領域を確保する必要があります。ソケットやヘッダーを使用して Raspberry Pi Pico を取り付ける場合は、コンポーネントを自由に配置することができます (ただし、ヘッダーやソケットから適切な距離を確保する必要があります)。配置するコンポーネントの高さが、ソケットやヘッダーで確保する必要がある距離を超えないようにしてください。

3.6. 回路図

最後に、この設計の回路図を紹介します。本書の最初で説明したように、実際の KiCad 回路図のファイルを参照することができます。ここで紹介する回路図は最新のものではない場合があるため、実際の回路図を確認することをお勧めします。

図 28. 完全な回路図


VGA
VGA 'DAC'
Micro SD Card
Audio Power
UART / 4-bit SD Selector & Debug
I2S/PCM Audio
PWM Audio
5V Power In
Pico
Title: RPi-PVSA VGA, SD Card & Audio Demo Board for Pico
Rev: REV2
(c) Raspberry Pi 2020
Raspberry Pi

付録 A: レスキューデバッグポートを使用する

概要

RP2040 の レスキューデバッグポート (レスキュー DP) を使用すると、正しくないコードをフラッシュ内でプログラミングした場合に、チップを以前の状態にリセットすることができます。たとえば、システムクロックをオフにするコードをプログラミングすると、プロセッサーのデバッグポートにアクセスできなくなりますが、その状態でもレスキュー DP は機能します。これは、レスキュー DP のクロックの動作源が SWD インターフェイスの `SWCLK` であるためです。

Raspberry Pi Pico で BOOTSEL ボタンを使用すると、フラッシュ内でコードを実行する代わりに、チップを強制的に BOOTSEL モードにすることができます。レスキュー DP は、RP2040 をベースとした、BOOTSEL ボタンのない設計で使用します。

注記

SWD の詳しい設定方法については、「Raspberry Pi Pico をセットアップしよう」を参照してください。

OpenOCD からレスキュー DP を起動する

OpenOCD の RP2040 ポートにより、以下に示す 2 つのターゲットが提供されます。

- `rp2040.cfg`
- `rp2040-rescue.cfg`

`rp2040-rescue.cfg` は、`0xf` という ID を持つレスキュー DP に接続されます。

このレスキュー DP を使用するには、`rp2040-rescue` 設定を使用して OpenOCD を起動します。

```
$ openocd -f interface/raspberrypi-swd.cfg -f target/rp2040-rescue.cfg
...
Warn : gdb services need one or more targets defined
Now attach a debugger to your RP2040 and load some code
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections

Ctrl + C
```

次に、通常の `rp2040` 設定を使用して OpenOCD を起動します。

```
$ openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg
```

レスキュー DP によってチップが再起動したかどうかを確認するには、`VREG_AND_POR.CHIP_RESET` レジスターが次の値になっているかどうかを確認します: `0x40064008`.このレジスタのビット 20 は、`HAD_PSM_RESTART` ビットです。

別のターミナルで OpenOCD の Telnet ポートに接続し、`mdw` (memory display word) を使用して `CHIP_RESET` レジスターを

読み取ります。レスキュー DP によってチップが再起動された場合、このレジスターの値が `0x00100000` になります (ビット 20 が設定されます)。

```
$ telnet 127.0.0.1 4444
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Open On-Chip Debugger
> mdw  0x40064008
0x40064008: 00100000
```

これで、コードを読み込むための準備が整いました (「Raspberry Pi Pico をセットアップしよう」の「Use GDB and OpenOCD to debug Hello World」を参照)。

付録 B: 本文書のリリース履歴

表 1. 本文書のリリース履歴

リリース	日付	説明
1.0	2021 年 1 月 21 日	• 初回リリース
1.1	2021 年 1 月 26 日	• マイナーな修正 • ADC での DMA の使用に関する情報を追加 • M0+ レジスタと SIO CPUID レジスタに関する説明を追加 • タイマーについてより詳しい説明を追加 • Windows および macOS のビルド手順を更新 • 書籍名を変更、出力 PDF のサイズを最適化
1.2	2021 年 2 月 1 日	• マイナーな修正 • PIO ドキュメントのマイナーな改善 • 不足していた TIMER2 レジスタと TIMER3 レジスタを DMA に追加 • UART で MicroPython REPL を取得する方法を追加 • C SDK の V1.0.1 リリースに伴う情報を追加
1.3	2021 年 2 月 23 日	• マイナーな修正 • フォントを変更 • RP2040 のシンク/ソース制限に関するドキュメントを追加 • SWD ドキュメントのメジャーな改善 • MicroPython のビルド手順を更新 • MicroPython UART のサンプルコードを追加 • Thonny の手順を更新 • プロジェクト生成機能の手順を更新 • FAQ ドキュメントを追加 • 正誤表 E7、E8、E9 を追加
1.3.1	2021 年 3 月 5 日	• マイナーな修正 • C SDK の V1.1.0 リリースに伴う情報を追加 • MicroPython UART のサンプルを改善 • ピンの配置図を改善

リリース	日付	説明
1.4	2021 年 4 月 7 日	• マイナーな修正 • 正誤表 E10 を追加 • Github から C SDK を更新する方法に関する注意事項を追加 • C SDK の V1.1.2 リリースに伴う情報を追加
1.4.1	2021 年 4 月 13 日	• マイナーな修正 • ドキュメントに記載されているすべてのソースコードが 3 条項 BSD ライセンスに含まれているという説明を追加
1.5	2021 年 6 月 7 日	• マイナーな更新と修正 • FAQ を更新 • SDK のリリース履歴を追加 • C SDK の V1.2.0 リリースに伴う情報を追加
1.6	2021 年 6 月 23 日	• マイナーな更新と修正 • ADC 情報を更新 • 正誤表 E11 を追加
1.6.1	2021 年 9 月 30 日	• マイナーな更新と修正 • B2 リリースに関する情報を追加 • B2 リリースの正誤表を更新
1.7	2021 年 11 月 3 日	• マイナーな更新と修正 • 一部のレジスタのアクセスの種類と説明を修正 • コア 1 の起動シーケンスに関する情報を追加 • SDK の「パニック」処理に関する説明を追加 • picotool ドキュメントを更新 • *付録 A:アプリに関するメモ*付録 (「Raspberry Pi Pico C/C++ SDK」) に例を追加 • C SDK の V1.3.0 リリースに伴う情報を追加
1.7.1	2021 年 11 月 4 日	• マイナーな更新と修正 • USB ダブルバッファリングのドキュメントを改善 • Picoprobe ブランチを変更 • ドキュメントへのリンクを更新

最新リリースは、 https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040-JP.pdf で参照することができます。

Raspberry Pi Ltd