Vol 57, No 1—March 2023

ANALOG DEVICES
想像を超える可能性を
AHEAD OF WHAT'S POSSIBLE™

AnalogDialogue

MCUのSPIドライバを最適化し、ADCとの間で高速データ転送を実現する

著者：Denny Wang、アプリケーション・エンジニア
Sally Tseng、アプリケーション・エンジニア

概要

低消費電力のIoT（Internet of Things）アプリケーションやエッジ／クラウド・コンピューティングが普及するにつれ、データ転送を高速化する必要性がより高まっています。図1に、ワイヤレス・センシング・システムの例を示しました。このシステムでは、分解能が24ビットのA/Dコンバータ（ADC）を使用することで、高精度のデータ・アクイジションを実現します。この場合、マイクロコントローラ・ユニット（MCU）の処理能力については1つの課題があります。それは、ADCの高速シリアル・インターフェースを利用できるだけの余裕があるのかどうかというものです。

本稿では、ADCからの出力データをMCUに転送する際、高速SPI（Serial Peripheral Interface）を使用するケースを考えます。そのためには、データのトランザクションを実現するためのSPIドライバを設計しなければなりません。そのプロセスでは、ドライバを最適化する方法として様々な方法を検討することになるでしょう。また、ADCとMCUについては適切な構成（コンフィギュレーション）を採用する必要があります。本稿では、まずそうした事柄について詳しく説明します。その上で、SPIとDMA（Direct Memory Access）によってデータのトランザクションを実現するためのサンプル・コードを紹介します。更に、最適化を施した同一のドライバを異なるMCU「ADuCM4050」と「MAX32660」に適用し、分解能が24ビットのADC「AD7768-1」のスループットがどのように高速化されるのか検証します。

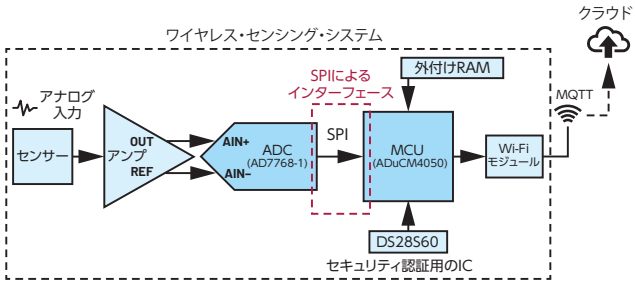

図1. ワイヤレス・センシング・システムの例

はじめに

まずは、MCU製品とSPIドライバの一般的な関係について説明します。

汎用のSPIドライバ

MCUのベンダーは、各製品向けに汎用のSPIドライバやAPI（Application Programming Interface）のサンプル・コードを提供しています。それらを使えば、ほとんどのアプリケーションに対応することができます。様々な構成を行ったり、ステートメントに関する決定を下したりすることが可能になるということです。しかし、ADCを使用するデータ・アクイジション・システムなど、特定の条件下ではそれらは有効に機能するとは限りません。汎用のSPIドライバでは、ADCの最高速度（最高スループット）に対応できない可能性があるのです。その理由としては、汎用のドライバには多様な構成に対応するための実装が含まれていることが挙げられます。実際のアプリケーションでは使用しない構成に関するコードが存在すると、余計なオーバーヘッドが生じ、遅延時間が発生します。

```
void adi_spi_MasterSubmitBuffer (struct hDevice, struct *buffer)
{
    if(…)                          ┐
        *p = xxx;                  │
    else if(…)                     │
        *p = xxx << xxx;           ├ 構成
        for (…)                    │
                                   │
    :                              ┘

    start_spi_transaction();
}
```

図2. 汎用APIに含まれる構成用のコード

何が問題になるのか？

ここでは、SPIを使用してMCUとADCの間でデータをやり取りするケースを考えます。その場合、MCUはADCの出力データを取得するために、SPIのメインとして機能することになります。MCU製品を選択する際には、消費電力が少ないことと、高速性能が得られることを条件として考慮することになるでしょう。ただ、アナログ・デバイセズの汎用SPIドライバをベースとしてデータのトランザクションを実現する場合には問題が生じる可能性があります。先述したように、ADCとMCUとのやり取りに必要のないコマンドが原因となって、データ転送速度が低下してしまうかもしれないからです。ADCの潜在的な能力を引き出し、最大のデータ転送速度を得るにはどうすればよいのでしょうか。これについて確認するために、筆者らはいくつかの実験を行うことにしました。具体的には、ADuCM4050とAD7768-1を組み合わせて解決策を検討することにしました。AD7768-1のデフォルトのフィルタ使用した場合、最大出力データ・レート（ODR：Output Data Rate）は256kHzです。しかし、ADuCM4050と組み合わせた場合、それが8kHzに制限されてしまいます。ODRを高めるための解決策としては、不要なコマンドの削除やDMAコントローラの利用といった方法が考えられます。以下では、それらの案について検討することにします。

SPIのメインとなるMCU

ADuCM4050は超低消費電力のMCUであり、メインのクロック・レートは26MHzです。プロセッサ・コアとしてはArm® Cortex®-M4Fを採用しており、3つのSPIを搭載しています。各SPIには、DMAコントローラとのインターフェースに使用する2つのDMAチャンネル（受信用と送信用）が組み込まれています。DMAコントローラとDMAチャンネルは、メモリとペリフェラルの間でデータをやり取りする手段を提供します。これはデータ転送の効率的な手法であり、コアを解放して他のタスクを処理することが可能になります。

SPIのノードとなるADC

AD7768-1は、低消費電力で高性能のシグマ・デルタ（ΣΔ）型ADCです。ODRと消費電力は、アプリケーションの要件を満たすように調整することができます。表1に示すように、デシメーション比と消費電力に関連する動作モードの組み合わせによってODRが決まります。

表1. 動作モード、デシメーション比、ODRの関係

動作モード	デシメーション比	ODR
高速（MCLK/2）	×32	256kHz
高速（MCLK/2）	×64	128kHz
中間（MCLK/4）	×32	128kHz
中間（MCLK/4）	×64	64kHz
低消費電力（MCLK/16）	×32	32kHz
低消費電力（MCLK/16）	×64	16kHz

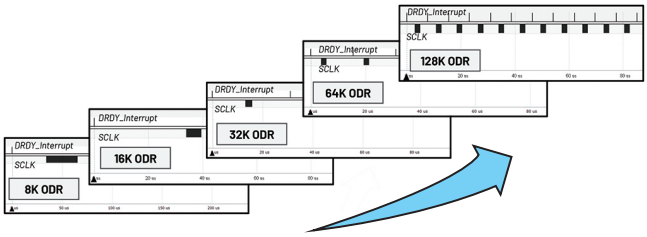

図3. ODRとDRDY/SCLKの関係

AD7768-1は連続読み出しモードを備えています。これも重要な機能です。ADCの出力データは0x6Cレジスタに格納されます。一般に、ADCのレジスタに対してデータの読み出し／書き込みを行う際には、事前にアドレスを指定する処理が必要になります。それに対し、連続読み出しモードでは、各データ・レディ信号の後に0x6Cレジスタからデータを直接読み出すことができます。ADCからの出力データは、アナログ入力電圧に対応する24ビットのコードになります（表2）。

表2. 入力電圧と出力コードの関係（理想値）

説明	アナログ入力電圧	デジタル出力コード
［正のフル・スイング］ − 1LSB	+4.095999512V	0x7FFFFF
［ミッドスケール］ ＋ 1LSB	+488nV	0x000001
［ミッドスケール］	0V	0x000000
［ミッドスケール］ − 1LSB	−488nV	0xFFFFFF
［負のフル・スイング］ − 1LSB	−4.095999512V	0x800001
［負のフル・スイング］	−4.096V	0x800000

データをやり取りするための接続

ADuCM4050とAD7768-1の間でデータをやり取りするためには、図4のようにピンを接続します。

MCUのGPIO28ピンからADCのRST_1ピンに対しては、リセット信号が送られます。データ・レディ信号は、ADCのDRDY_1ピンからMCUのGPIO27ピンに送信されます。残りのピンは、一般的なSPIの構成として接続しています。MCUがメイン、ADCがノードです。ADCのSDI_1ピンは、MCUからレジスタに対する読み出し／書き込みのコマンドを受け取ります。DOUT-1ピンは出力データをMCUに送信します。

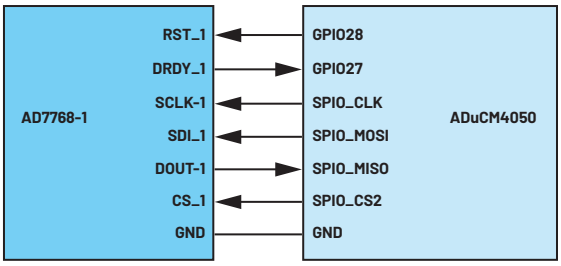

図4. AD7768-1とADuCM4050の接続。
SPIを使用するように構成しています。

データのトランザクションの実現

続いて、データのトランザクションがどのように実現されるのか詳しく説明します。

割り込みデータのトランザクション

連続したデータのトランザクションを実現するには、GPIO27ピン（DRDY_1ピンに接続）を使って割り込みトリガの授受を行います。ADCがデータ・レディ信号をGPIO27ピンに送信すると、MCUはデータのトランザクションに関するコマンドを含むコールバック関数を実行します。図5に示すように、データの取得の処理は、割り込みAと割り込みBの間の期間に実行する必要があります。

アナログ・デバイセズが提供する汎用SPIドライバを使用すれば、ADCとMCUの間のデータのトランザクションを簡単に実現することができます。但し、ADCのODRは、ドライバ内の冗長なコマンドによって8kHzに制限されます。この処理を高速化するには、コードを必要最小限なレベルまで簡素化しなければなりません。以下では、DMAを利用するデータのトランザクションの実現方法を2つ紹介します。1つはベーシック・モードのDMAトランザクション、もう1つはピンポン・モードのDMAトランザクションです。

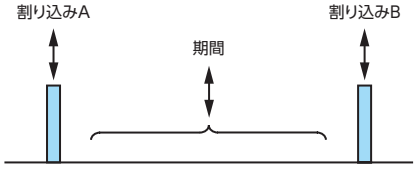

図5. 2つの割り込みの間の期間

ベーシック・モードのDMAトランザクション

各DMAトランザクションを利用するためには、事前にSPIとDMAの設定を行う必要があります（図6）。SPIの構成にはSPI_CTLを使用します。アナログ・デバイセズの汎用SPIドライバからSPI_CTL = 0x280fという設定が得られます。SPI_CNTは転送バイト数を表します。各DMAトランザクションでは、16ビットという固定ビット数のデータしか送信できません。したがって、SPI_CNTは2の倍数に設定する必要があります。この例では、SPI_CNT = 4とすることで、ADCの24ビットの出力データに対応します。SPI_DMAレジスタは、SPIのDMAをイネーブルに設定するために使用します。SPI_DMA = 0x5とすることで、受信DMAのリクエストがイネーブルに設定されます。また、pADI_DMA0->EN_SET = (1<<5)によって5番目のDMAチャンネルであるSPI0 RXをイネーブルにします。

```
//SPI settings
pADI_SPI0->CTL = 0x280f;
pADI_SPI0->CNT = (uint16_t)4;
pADI_SPI0->DMA = 0x5;

//DMA settings
pADI_DMA0->EN_SET = (1<<5);
pPrimaryCCD[5].DMASRCEND = (uint32_t)&pADI_SPI0->RX;
pPrimaryCCD[5].DMADSTEND(uint32_t)Current_address;
pPrimaryCCD[5].DMACDC = 0x4D000011;

//Dummy read
pADI_SPI0->RX;

//Data destination address maintenance
Current address = Current_address + 4;
```

図6. ベーシック・モードの
DMAトランザクションに対応するコード

各DMAチャンネルには、表3に示すDMAの構造体のレジスタが設けられています。なお、ソース・アドレスの末尾（SPI0のRx）については、Rx FIFO（First In, First Out）がレジスタからデータを自動的にプッシュするので、インクリメントの操作を行う必要はありません。一方、デスティネーション・アドレスの末尾は、アナログ・デバイセズの汎用SPIドライバに従い、[デスティネーション・アドレス] + SPI_CNT - 2という関数によって計算されます。

表3. DMAの構造体のレジスタ

名称	説明
SRC_END_PTR	ソース・エンド・ポインタ
DST_END_PTR	デスティネーション・エンド・ポインタ
CHNL_CFG	制御データの構成

ここで、現在のアドレスは、内部の配列バッファのアドレスです。DMAの制御データの設定には、次のようなものが含まれています。すなわち、ソース・データのサイズ、ソース・アドレスのインクリメント、デスティネーション・アドレスのインクリメント、残りの転送数、DMA制御モードの設定などです。0x4D000011という値によって、表4に示す構成に設定できます。

表4. 制御データの設定値0x4D00011に対するDMAの設定

レジスタ	説明	値
DST_INC	デスティネーション・アドレスのインクリメント	2バイト
SRC_INC	ソース・アドレスのインクリメント	0
SRC_SIZE	ソース・アドレスのインクリメント	2バイト
N_minus_1	[現在のDMAサイクル] − 1の総転送数	1（N = 2）
Cycle_ctrl	DMAサイクルの動作モード	ベーシック・モード

ダミーの読み出しコマンドSPI_SPI0 -> RXにより、SCLKのクロックがスタートします。そして、出力データはMISOラインを介してADCからMCUへ送信されます。MOSIラインでも、無視できるレベルのデータ転送が行われます。Rx FIFOが満杯になると、DMAリクエストが生成され、DMAコントローラが起動します。その結果、DMAのソース（SPI0のRx FIFO）からDMAのデスティネーション（内部の配列バッファ）にデータが転送されます。TcリクエストはSPI_DMA = 0x3によって生成されることに注意してください。

最後に、次の4バイトのデータを送信するために、現在のデスティネーション・アドレスに4を加算します。それにより、デスティネーション・アドレスを維持します。

また、SPI0のDMAチャンネルのpADI_DMA0->DSTADDR_CLRとpADI_DMA0->RMSK_CLRは、最初の割り込みが発生する前にmain関数内で設定する必要があります。この点には注意してください。前者のレジスタは、DMA Channel Destination Address Decrement Enable Clearです。これは、インクリメント・モードにおける各DMA転送の後にデスティネーション・アドレスをシフトするように設定します（インクリメント・モードにおいてのみデスティネーション・アドレスの計算機能が動作します）。後者のレジスタは、DMA Channel Request Mask Clearです。これは、チャンネルのDMAリクエストのステータスをクリアします。

図7（a）に、ベーシック・モードのDMAトランザクションのタイミングを示しました。各タイム・スロットは、それぞれDRDY信号、SPI/DMAの設定、DMAデータのトランザクションを表しています。CPUのアイドル時間をより有効に活用するためには、DMAコントローラがデータ転送の処理を行っている間にCPUにタスクを割り当てることが望ましいと言えます。

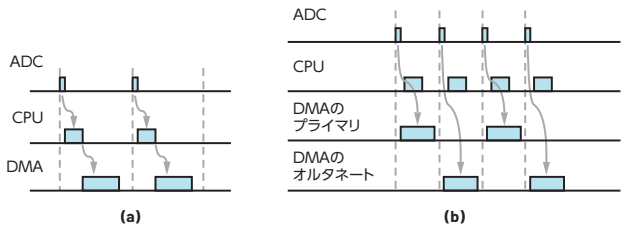

図7. ベーシック・モード（a）とピンポン・モード（b）のタイミング図

ピンポン・モードのDMAトランザクション

ダミーの読み出しコマンドが実行された後、DMAコントローラはデータのトランザクションを開始します。また、MCUのCPUはタスクのないアイドル状態になります。CPUとDMAコントローラを同時に動作させることができれば、タスクは直列ではなく並列に処理されることになります。つまり、DMAの構成（CPUによる）とDMAデータのトランザクション（DMAコントローラによる）を同時に実行できます。このアイデアを具現化するには、DMAコントローラにいわゆるピンポン・モードが必要になります。ピンポン・モードには、プライマリとオルタネートという2組のDMA構造体が組み込まれます。DMAコントローラは、DMAのリクエストのたびに、それら2つの構造体の切り替えを自動的に実行します。初期値が0に設定されている変数pは、DMAの構造体がプライマリ（p = 0）なのかオルタネート（p = 1）なのかを表します。p = 0の場合、ダミーの読み出しコマンドによってDMAのプライマリ・データのトランザクションが開始されます。同時に、オルタネートの構造体に値を設定すると、次の割り込みサイクルにはその構造体が対応することになります。p = 1の場合、プライマリの構造体とオルタネートの構造体の役割が切り替わります。DMAのトランザクションの最中にDMAの構造体を変更する処理は、ベーシック・モードにおけるプライマリの構造体においてのみ失敗する可能性があります。ピンポン・モードを使用することで、オルタネートの構造体にCPUがアクセスして書き込みが行えるようになります。同時に、DMAコントローラによってプライマリの構造体の読み出しが行えるようになります。なお、その逆の処理も可能です。図7（b）に示すように、DMAデータのトランザクションでは、最後のサイクルでDMAの構造体の構成が行われます。そのため、トランザクションはADCからMCUへDRDY信号が送信されるとすぐに実行されます。CPUとDMAは、一方が他方の処理を待つことなく同時に動作するようになります。その結果、トータルの動作時間が大幅に短くなり、ADCのODRを高める余地が得られるはずです。

割り込みハンドラの最適化

連続するデータ・レディ信号の間の期間には、コールバック関数内のコマンドを実行する時間だけでなく、アナログ・デバイセズのGPIO割り込みハンドラに含まれるコマンドの実行時間も含まれます。

MCUが起動した際、CPUはスタートアップ・ファイル（startup.s）を実行します。GPIOの割り込みハンドラをはじめとするイベント・ハンドラについては、すべてこのファイル内で定義されています。GPIOの割り込みがトリガされると、割り込みハンドラ関数（アナログ・デバイセズのGPIOドライバに含まれるGPIO_A_INT_HANDLERとGPIO_B_INT_HANDLER）が実行されます。一般的な割り込みハンドラ関数を使用した場合、CPUがすべてのGPIOピンを探索してトリガされているピンが見いだされます。続いて、その割り込みのステータスをクリアし、登録済みのコールバック関数を実行します。本稿で例にとるADC/MCUベースのアプリケーションでは、DRDY信号が唯一の割り込み信号となります。そこで、処理を高速化するためにこの関数を簡素化することを考えます。想定可能な解決策としては、スタートアップ・ファイルのリターゲット、元の割り込みハンドラの修正などが挙げられます。ここで、リターゲットというのは、割り込みハンドラを独自に定義し、スタートアップ・ファイル内の元のハンドラを置き換えるという意味です。

一方、割り込みハンドラの修正には、独自に定義したGPIOドライバが必要です。ここでは、割り込みハンドラを修正する方法を採用し、図8のように関数を修正することにしました。この関数では、DRDY信号に接続されたピンの割り込みステータスだけをクリアし、コールバック関数に直接進みます。なお、元のGPIOドライバについては、ビルド処理のターゲットに含まれるインクルードのボックスのチェックを外すことによってブロックする必要があります。

```
DCD   GPIO_A_Int_Handler_Denny  ; GPIO Int A          /* 9  */
DCD   GPIO_B_Int_Handler        ; GPIO Int B          /* 10 */
DCD   GPIO_Tmr0_Int_Handler     ; GP Timer 0          /* 11 */
DCD   GPIO_Tmr1_Int_Handler     ; GP Timer 1          /* 12 */
```

図8. ネスト型のベクタ割り込みコントローラ
(NVIC：Nested Vectored Interrupt Controller)

最適化による速度の改善効果

ここまでに説明した最適化の効果を確認するために、実機による評価を行いました。ここでは、24ビットのADCが出力する200個のデータを読み取るケースを例にとります。SPIのビット・レートは13MHzに設定されています。DRDY信号のピンとSCLKをオシロスコープに接続することにより、DRDY信号からSPIデータのトランザクション（DMAのトランザクションも）が開始するまでの時間を観測します。それにより、本稿で紹介した方法による速度の改善効果を定量化することができます。ここでは、DRDY信号からSCLK信号の開始までの時間をΔtと呼ぶことにします。SPIのビット・レートが13MHzである場合、Δtの測定値は以下のようになりました。

- (a) ベーシック・モードのDMAでは、Δtが3.754マイクロ秒
- (b) ピンポン・モードのDMAでは、Δtが2.834マイクロ秒
- (c) 割り込みハンドラを最適化したピンポン・モードのDMAでは、Δtは1.694マイクロ秒

上記の（a）、（b）の方法を採用すれば64kHzのODRに対応できます。それに対し、（c）の方法では128kHzのODRに対応することが可能です。（c）の方法では、Δtが短く、SCLK信号を早く終了させられるからです。SCLK信号（すなわち、データのトランザクション）がT/2（TはADCから出力される現在のデータの周期）よりも前に終了する場合、ODRを何倍かに高めることができます。アナログ・デバイセズの汎用SPIドライバを使用した場合にはODRが8kHzなので、非常に大きな高速化が実現されることになります。

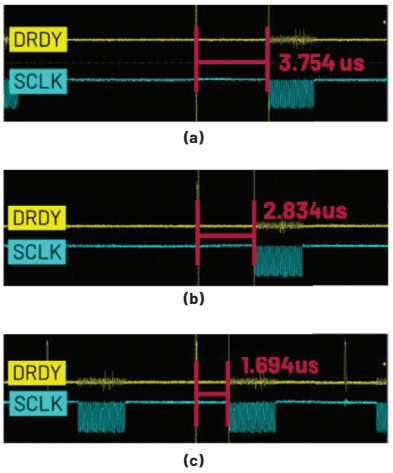

図9. Δtの測定結果。（a）はベーシック・モード、（b）はピンポン・モード、（c）は最適化済みの割り込みハンドラを適用したピンポン・モードを使用した場合の結果を表しています。

MAX32660とAD7768-1の組み合わせ

MCUとして、ADuCM4050の代わりにメインのクロック・レートが96MHzのMAX32660を使用した場合、どのような結果になるでしょうか。ここでは、割り込みハンドラの最適化を適用し、割り込みデータのトランザクションを使用して実験を行いました。この割り込み設定では、DMAの機能を使用することなく256kHzのODRを達成することができます（図10）。

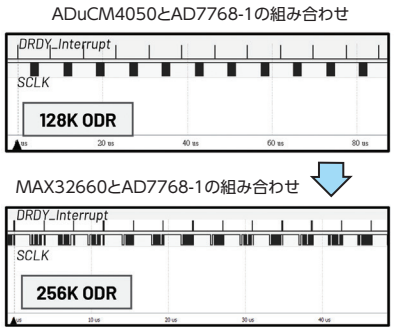

図10. MAX32660を使用した場合のODR（DMAは不使用）。ADuCM4050を使用した場合と比較しています。

まとめ

本稿では、ADCとしてAD7768-1、MCUとしてADuCM4050/MAX32660を使用した場合に、SPIを介したデータのトランザクションを高速化する方法を紹介しました。目標とする速度を達成するためには、アナログ・デバイセズの汎用SPIドライバを使用しつつ、冗長なコマンドを削除してデータのトランザクションを実行するべきです。また、DMAコントローラを使用してCPUコアを解放することも、連続したデータのトランザクションの高速化につながります。加えて、ピンポン・モードのDMAでは、適切なスケジューリングを行うことによってDMAの構成にかかる時間を節約できます。DMAによる高速化に加え、割り込みピンを直接指定することで、割り込みハンドラを最適化することも可能です。SPIのビット・レートが13MHzの場合、最高性能としてADCのODRを128kSPSまで高められます。

謝辞

本稿の執筆にあたっては多くのサポートと協力を得ました。ハードウェアに関する知識、ソフトウェアに関するサポート、デバッグに関するアドバイスなどを提供してくれたCharles Lee、メンターとして技術的なサポートを行ってくれたWilliam Chen、技術的な経験の多くを共有してくれたFrank Changに感謝します。

表5. SPIの高速化を図った結果。
ADuCM4050、MAX32660を使用した場合の結果をまとめています。

	ADuCM4050 (MCU)				MAX32660 (MCU)
データのトランザクション	最適化していない割り込み	ベーシック・モードのDMA	ピンポン・モードのDMA	最適化した割り込み	最適化した割り込み
バスの種類	SPI	SPI	SPI	SPI	SPI
メインのクロック・レート	26MHz	26MHz	26MHz	26MHz	96MHz
SPIのクロック・レート	13MHz	13MHz	13MHz	13MHz	20MHz
DRDYとSCLKの間隔	6.34マイクロ秒	3.754マイクロ秒	2.834マイクロ秒	1.694マイクロ秒	1.464マイクロ秒
ODR	8kSPS	32kSPS	64kSPS	128kSPS	256kSPS

著者について

Denny Wangは、アナログ・デバイセズ（台湾）のアプリケーション・エンジニアです。NCGプログラムで1年半を過ごしました。入社は2021年で、現在はシステム・ソリューション、データのセキュリティ／認証、MQTTワイヤレス伝送に関するサポートを担っています。北京大学でIC工学の修士号を取得。在学中は、半導体の製造プロセス、BLDC/PMSMモータの制御、アナログ回路図のマイグレーションに取り組んでいました。

著者について

Sally Tsengは、アナログ・デバイセズ（台湾）のアプリケーション・エンジニア（インターン）です。台湾大学で電気工学を専攻し、主にアナログ回路について研究しています。インターンシップとして、組み込みシステムにも取り組んでいます。

AD5703-0-3/23