アナログ・デバイセズに寄せられた珍問／難問集 Issue 172

MCUのSPIを使用し、非標準的なSPIを備えるADCにアクセスする

著者：Steven Xie、プロダクト・アプリケーション・エンジニア

質問：

任意のMCUが備えるSPIを使用し、非標準的なSPIを備えるA/Dコンバータにアクセスすることは可能でしょうか。

回答：

可能です。但し、少し工夫が必要になるでしょう。

はじめに

多くの場合、高い精度を誇る最新型のA/Dコンバータ（ADC）であれば、SPI（Serial Peripheral Interface）など、何らかのシリアル・インターフェースを備えているでしょう。その目的は、MCU（マイクロコントローラ・ユニット）、DSP（デジタル・シグナル・プロセッサ）、FPGA（フィールド・プログラマブル・ゲート・アレイ）などのコントローラとの間で通信が行えるようにすることです。それにより、各コントローラから、ADCの内部レジスタに対する読み書きや、A/D変換後のコードの読み出しを行うことが可能になります。SPIを採用すれば、プリント回路基板の配線を簡素化することができます。また、SPIは、パラレル・インターフェースよりもクロック・レートが高いこともあり、より広範な用途で使用されるようになっています。加えて、標準的なSPIを使えば、ADCとコントローラの間の接続も容易に行えます。

最新型のADCの中には、標準的なSPIを備えているものがあります。しかし、スループットを高めるために、ノードとして機能する非標準的な3線/4線式のSPIを備えているものも少なくありません。例えば、アナログ・デバイセズが提供するADCベースの製品ファミリ「AD7616」、「AD7606」、「AD7606B」は、シリアル・モードにおけるスループットを高めるために、2本または4本のSDOラインを備えています。一方、「AD7768」、「AD7779」、「AD7134」の各製品ファミリは、複数のSDOラインを備えており、SPIのメインとして機能します。ただ、ADCの構成とコードの読み出しを行うためにMCUのSPIを設計する作業を難しく感じる人は少なくないようです。

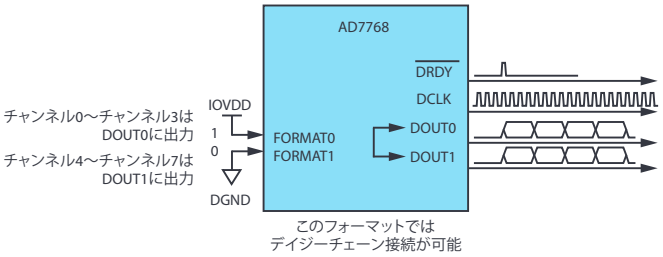


図1. AD7768の使用例。2つのデータ出力ピンを備えるSPIのメインとして構成しています（14001-193）。

MCUとADCを標準的なSPIで接続する

SPIは、同期式／全二重のメイン-ノード型インターフェースです。メインまたはノードからのデータの同期は、クロックの立上がりエッジまたは立下がりエッジによって実現します。また、メインとノードは、同時にデータを送信することが可能です。図2に、MCUが備える一般的な4線式SPIを使ってADCと接続する例を示しました。

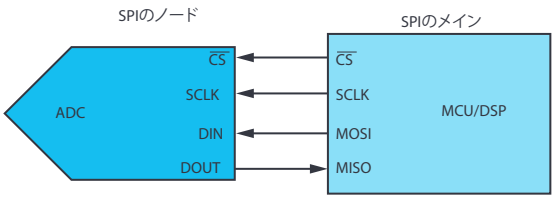


図2. MCUの標準的なSPIを使ってADCと接続する方法。ADCをノードとして使用する場合の例です。

SPIによる通信を開始するには、コントローラからクロック信号を送信し、$\overline{CS}$信号をイネーブルにすることによってADCを選択する必要があります。通常、$\overline{CS}$信号はアクティブ・ローです。SPIは全二重インターフェースなので、コントローラとADCはそれぞれMOSI/DINラインとMISO/DOUTラインを使って同時にデータを送信することができます。ユーザは、コントローラのSPIにおいてクロックの立上がりエッジと立下がりエッジのうちどちらでデータをサンプリング／シフトするかを自由に選択することが可能です。

メインとノードの間で信頼性の高いデジタル・インターフェースを実現するには、MCUのタイミング仕様とADCのタイミング仕様の両方に従う必要があります（図3）。

MCUのSPIとADCのシリアル・インターフェースに標準的なSPIのタイミング・モードが用意されていれば、プリント基板配線の設計やファームウェアの開発は難しくありません。しかし、新型のADCの中には、標準的なSPIのタイミング仕様に従わないシリアル・インターフェース・ポートを備えているものがあります。例えば、AD7768は、非標準的なタイミング仕様に即したSPIポート（シリアル・ポート）を備えています（図4）。このようなポートを介して、MCUやDSPからデータを読み出すのは、不可能であるように感じられます。

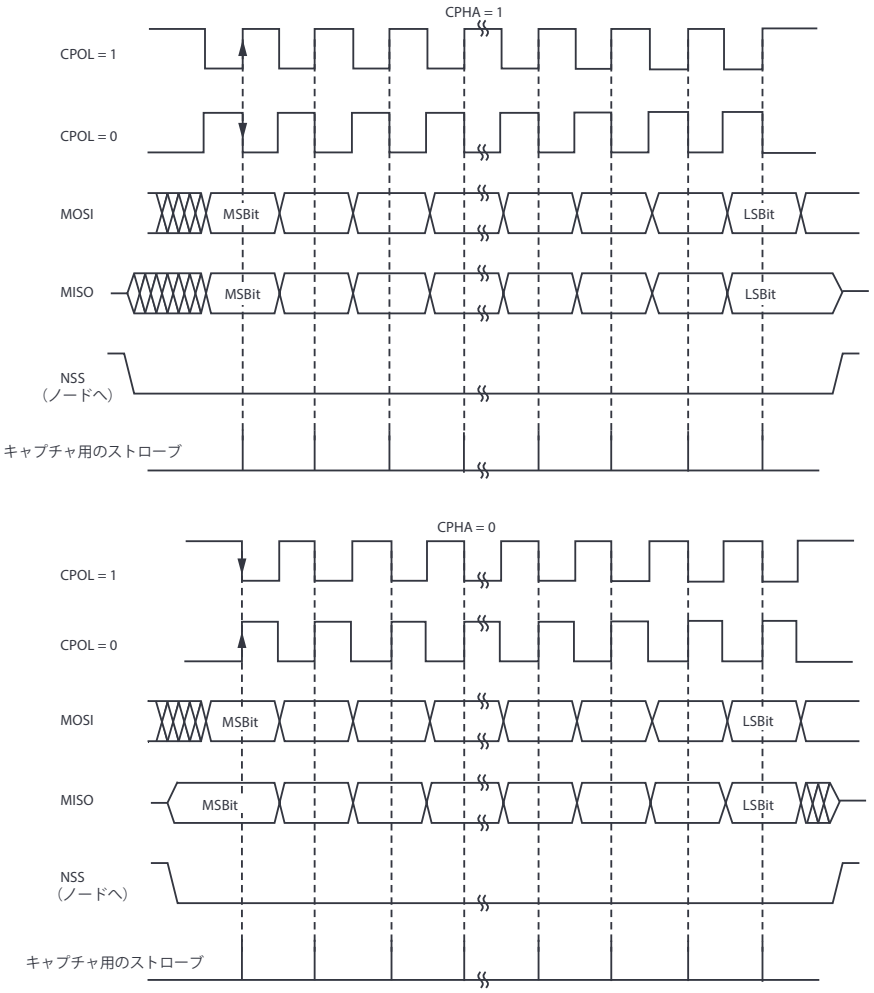


図3. SPIにおけるデータ・クロックのタイミング仕様の例

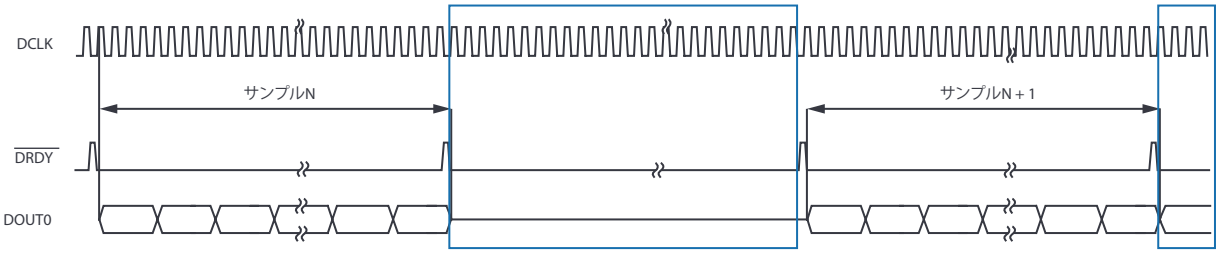


図4. AD7768のFORMATxピンを1xとして設定した場合のタイミング・チャート。
DOUT0だけにデータが出力されます。

本稿では、MCUの標準的なSPIを使用し、非標準的なSPIポートを備えるADCとのインターフェースを確立する方法について解説します。具体的には、シリアル・インターフェースによってADCの出力コードを読み出す方法として、以下の4つのソリューションを紹介します。

- ソリューション1：MCUをSPIのノード、ADCをSPIのメインとし、1本のDOUTラインによって接続する
- ソリューション2：MCUをSPIのノード、ADCをSPIのメインとし、2本のDOUTラインによって接続する
- ソリューション3：MCUをSPIのノード、ADCをSPIのメインとし、DMAを介して接続する
- ソリューション4：MCUをSPIのメイン兼ノードとし、2本のDOUTライン上のデータを読み出す

1つのDOUTラインを使用し、STM32F429のSPIによってAD7768の出力コードを読み出す

図4に示すように、AD7768のFORMATxピンを1xまたは10に設定した場合、チャンネル0～7からのデータは、DOUT0だけに出力されます。標準モードにおいて、AD7768や「AD7768-4」はメインとして動作し、MCU/DSP/FPGAに対してデータを出力します。AD7768/AD7768-4は、データ、データ・クロック（DCLK）、立下がりエッジを使用するフレーミング信号（$\overline{\text{DRDY}}$）をノードとなるデバイスに供給します。

STMicroelectronicsのMCU製品ファミリ「STM32Fxxx」は、広範なアプリケーションで使用されています。同ファミリの製品は、複数のSPIポートを備えており、標準的なSPIのタイミング・モードでそれらをメインまたはノードとして機能させることができます。以下のセクションで紹介する方法は、8/16/32ビット・フレームに対応する他のMCUにも適用できます。

AD7768とAD7768-4は、それぞれ8チャンネル／4チャンネルの同時サンプリングが可能なシグマ・デルタ（ΣΔ）型ADCです。チャンネルごとにΣΔ変調器とデジタル・フィルタを備えており、AC信号とDC信号の同期サンプリングを実現できます。標準的な性能として、±2ppmの積分非直線性（INL）、±50μVのオフセット誤差、±30ppmのゲイン誤差を達成し、110.8kHzの最大入力帯域幅で108dBのダイナミック・レンジを得ることが可能です。AD7768/AD7768-4では、入力帯域幅、出力データ・レート、消費電力のうちいずれかを優先することができます。3つのパワー・モードのうち1つを選択し、目標とするノイズと消費電力を達成できるよう最適化を図ることが可能です。このような柔軟性を備えていることから、AD7768/AD7768-4は、低消費電力のDC測定モジュールまたは高性能なAC測定モジュール向けの再利用が可能なプラットフォームとして使用することができます。しかし、残念ながら、AD7768は標準的なSPIのタイミング・モードに対応しておらず、シリアル・インターフェースのメインとして動作します。そのため、一般的には、コントローラとしてFPGA/CPLDを使用する必要があります。

ここで、STMicroelectronicsのMCU「STM32F429」用のディスカバリ・キット「32F429IDISCOVERY」と、AD7768の評価用ボード「EVAL-AD7768FMCZ」を接続する例を示します。両者のインターフェースは、図5に示すような接続で実現します。この構成では、AD7768の全8チャンネルからDOUT0だけにデータが出力されます。

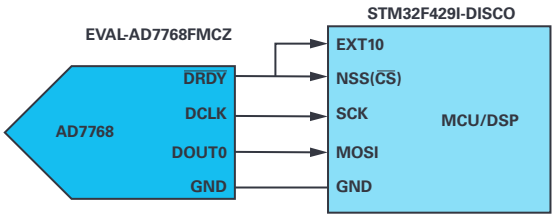


図5. 32F429IDISCOVERYとEVAL-AD7768FMCZを接続する例。AD7768はDOUT0だけにデータを出力します。DOUT0は、STM32F429のSPIに接続します。

この場合、解決すべき問題として、以下の事柄が挙げられます。

- AD7768はSPIのメインとして動作するので、STM32F429IのSPIはノードとして構成しなければなりません。
- $\overline{\text{DRDY}}$は、標準的な$\overline{\text{CS}}$とは異なり、DCLKの1サイクル分しかハイになりません。
- DCLKは連続的に出力され、$\overline{\text{DRDY}}$は全チャンネルのデータ出力が完了したときにローになります。

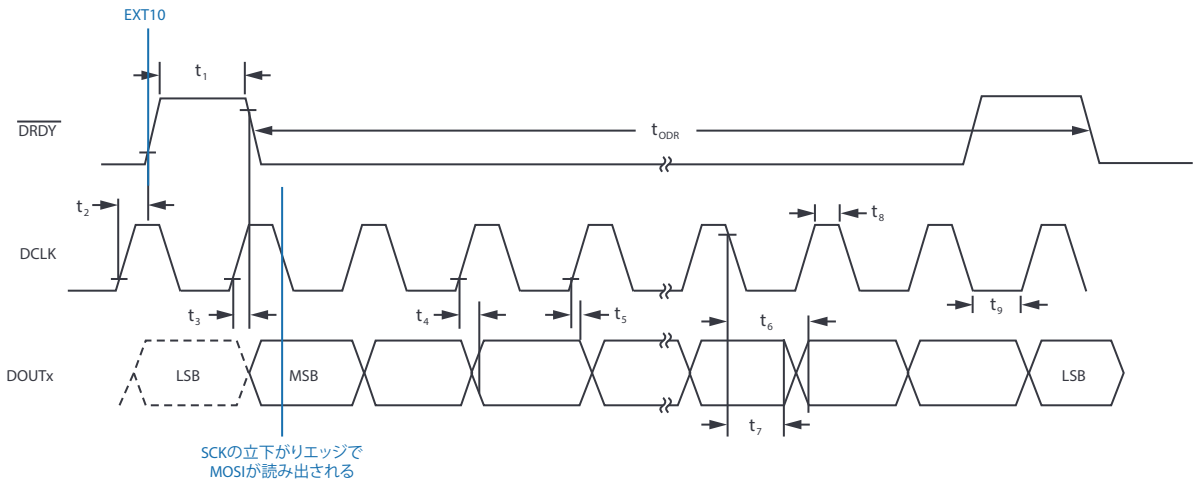


図6. AD7768におけるデータ・ビットの読み出しタイミング

ソリューション1：MCUをSPIのノード、ADCをSPIのメインとし、1本のDOUTラインによって接続する

以下、先述した4つのソリューションについて順に説明します。まず、ソリューション1は、以下のようなものになります。

- STM32F429の1つのSPIポート（例えばSPI4）を、DCLKを使ってMOSIのデータ・ビットを受信するノードとして構成します。
- AD7768の$\overline{\text{DRDY}}$をSTM32F429のEXTI0（外部割込み入力）ピンとNSS（SPIの$\overline{\text{CS}}$）ピンに接続します。$\overline{\text{DRDY}}$の立上がりエッジでEXTI0のハンドラ・ルーチンがトリガされ、SPIのノードがイネーブルになります。そして、$\overline{\text{DRDY}}$がローになって最初のDCLKの立下がりエッジから、データの受信が始まります（図6）。このタイミング設計は非常に重要です。
- チャンネル0～7のすべてのデータを受信したら、余分なデータ（無効データ）の読み出しを防ぐために、SPIをディスエーブルにする必要があります。$\overline{\text{DRDY}}$によってSPIのノードの$\overline{\text{CS}}$はローになり、DCLKは継続的にトグルしているからです。

MCUのファームウェアを開発する際の注意点

割込みモードにおいてソフトウェアを使用することにより、DCLKは最大4MHzで動作し、8kSPSの出力データ・レート（ODR）を得ることができます（図7）。

```
/*## Configure the SPI4 peripheral ###*/
Spi4Handle.Instance                      = SPI4;//use STM32F429 SPI4
Spi4Handle.Init.Direction                = SPI_DIRECTION_2LINES_RXONLY;
Spi4Handle.Init.CLKPhase                 = SPI_PHASE_1EDGE;//read at DCLK falling edge
Spi4Handle.Init.CLKPolarity              = SPI_POLARITY_HIGH;//read at DCLK falling edge
Spi4Handle.Init.DataSize                 = SPI_DATASIZE_8BIT;//or 16BIT
Spi4Handle.Init.NSS                      = SPI_NSS_HARD_INPUT;//make /CS low active
Spi4Handle.Init.Mode                     = SPI_MODE_SLAVE;//MCU SPI4 as SPI Slave
/*## Enable EXTI0 and SPI4 to Receive AD7768 Data bits ###*/
// clear EXTI0 IT flag prior to enable external interrupt 0 !!!
__HAL_GPIO_EXTI_CLEAR_IT(KEY_BUTTON_PIN);
HAL_NVIC_EnableIRQ(EXTI0_IRQn);
// wait for EXTI0 interrupt (/DRDY rising edge) to prepare for reading last conversion data
if (EXTI0_Flag == SET)
{
          EXTI0_Flag = RESET;//clear /DRDY rising edge flag variable
          // throw out the last byte/word captured in the previous ODR cycle !!!
          Rx_temp = *(__IO uint8_t *)&Spi4Handle.Instance->DR;
          __HAL_SPI_ENABLE(&Spi4Handle);
          // SPI4_CNVByteNum is the total data byte number to read in one conversion cycle
          while (SPI4_ByteCount < SPI4_CNVByteNum)
          {
                    // Check the RXNE flag
                    if (__HAL_SPI_GET_FLAG(&Spi4Handle, SPI_FLAG_RXNE))//
                    {
                              // transfer the received data from DR register to memory
                              SPI_RxBuffer[RxBuf_Idn] = *(__IO uint8_t *)&Spi4Handle.Instance->DR;
                              RxBuf_Idn++;
                              SPI4_ByteCount++;
                    }
          }
          // disable SPI4 to prevent read in extra data after all channel codes finished due to /DRDY
          is low active and DCLK continuously pulses
          __HAL_SPI_DISABLE(&Spi4Handle);
          SPI4_CNVCount++;
          RxBuf_Idn = SPI4_CNVCount * SPI4_CNVByteNum;
          SPI4_ByteCount = 0;
}//end of if (EXTI0_Flag == SET)
else
{//*** other software jobs ***//}
/*## handles External 0 interrupt request ###*/
// EXTI0 rising edge triggered to leave more response time for going into EXTI0_IRQHandler !!!
void EXTI0_IRQHandler(void)
{
          if(__HAL_GPIO_EXTI_GET_IT(EXTI0) != RESET)
          {
                    // enable SPI4 as soon as possible, and make sure before the first DCLK falling edge
                    after /DRDY falling !!!
                    __HAL_SPI_ENABLE(&Spi4Handle);
                    __HAL_GPIO_EXTI_CLEAR_IT(EXTI0);
                    EXTI0_Flag = SET;
          }
}
```

図7. SPI4のペリフェラルの構成

ソフトウェアでは、割込みハンドラを呼び出し、DCLKの1.5サイクル（375ナノ秒）以内にSPIを起動する必要があります。MCUがDCLKの立上がりエッジでデータを読み出すことができれば、更にDCLKの半サイクル分の余裕が生まれます。そのため、ソフトウェアによる割込みルーチンの呼び出しは、より容易になります。しかし、DCLKの立上がりエッジのt5では、DOUTxが最小で-3ナノ秒無効なので（IOVDDが1.8Vの場合は-4ナノ秒）、DOUTxの伝搬遅延（> |t5| + ［MCUのホールド時間］）を、プリント基板の配線またはバッファによって追加する必要があります。

ソリューション2：MCUをSPIのノード、ADCをSPIのメインとし、2本のDOUTラインによって接続する

ソリューション1では、8チャンネルすべてのデータをDOUT0だけに出力していました。そのため、データの読み出しが制約となり、ADCのスループットは8kSPSに抑えられていました。図1に示したように、チャンネル0～3をDOUT0に出力し、チャンネル4～7をDOUT1に出力すれば、データの転送時間を短縮することができます。その場合、シリアル・インターフェースの各配線は図8のように接続します。このような改良を施すことで、4MHzのDCLKにより、ODRを最大で16kSPSまで容易に高めることができます。

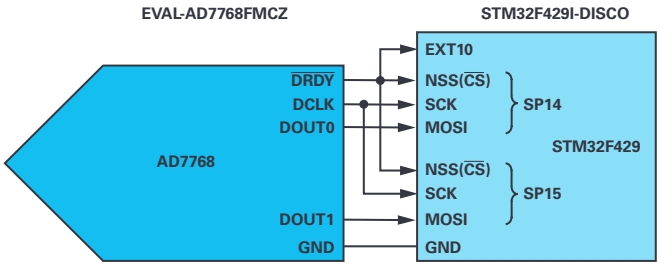


図8. ソリューション2を適用する場合の接続。AD7768はDOUT0とDOUT1にデータを出力します。DOUT0とDOUT1はSTM32F429のSPIに接続します。

ファームウェアにおいては、割込みモードではなくポーリング・モードを使用します（図9）。それにより、トリガとなるDRDYの立上がりエッジから、SPIがイネーブルになってデータが受信されるまでの遅延時間を短縮することができます。その結果、8MHzのDCLKによって、32kSPSのODRを達成できます。

```
/*## EXTI0 in Polling Mode and SPI4 & SPI5 to Receive AD7768 Data bits on DOUT0 and DOUT1 ###*/
// Polling for EXTI0 (/DRDY) rising edge to start MCU SPI ports
while (__HAL_GPIO_EXTI_GET_IT(EXTI0) != SET);
{
          __HAL_SPI_ENABLE(&Spi4Handle);
          __HAL_SPI_ENABLE(&Spi5Handle);
          __HAL_GPIO_EXTI_CLEAR_IT(EXTI0);
}
// throw out the last byte/word captured in the previous ODR cycle !!!
Rx_temp = *(__IO uint8_t *)&Spi4Handle.Instance->DR;
Rx_temp = *(__IO uint8_t *)&Spi5Handle.Instance->DR;
while (SPI4_ByteCount < SPI4_CNVByteNum)// total data byte number to read in one conversion cycle
{
          if (__HAL_SPI_GET_FLAG(&Spi5Handle, SPI_FLAG_RXNE))//
          {
                    SPI_RxBuffer[RxBuf_Idn] = *(__IO uint8_t *)&Spi4Handle.Instance->DR;
                    SPI_RxBuffer[RxBuf_Idn+1] = *(__IO uint8_t *)&Spi5Handle.Instance->DR;
                    RxBuf_Idn++;
                    SPI4_ByteCount += 2;
          }
}
__HAL_SPI_DISABLE(&Spi4Handle);
__HAL_SPI_DISABLE(&Spi5Handle);
```

図9. ファームウェアの例。ポーリング・モードのEXTI0とSPI4/SPI5を使用して、AD7768のDOUT0/DOUT1から出力されるデータを受信します。

ソリューション3：MCUをSPIのノード、ADCをSPIのメインとし、DMAを介して接続する

DMA（Direct Memory Access）は、ペリフェラルとメモリの間、またはメモリとメモリの間のデータ転送を高速化したい場合に用いられます。MCUの本体部分を動作させることなく、迅速にデータを転送できます。それにより、MCUのリソースを継続的に他の処理に振り分けることが可能になります。MCUのSPIをノードとして動作させ、DMAを介してデータを受信する方法については、デザイン・ノートをご覧ください。ファームウェアの例を図10に示しておきます。

ソリューション4：MCUをSPIのメイン兼ノードとし、2本のDOUTライン上のデータを読み出す

高い精度を実現する最新ADCの中には、高いスループットが得られたり、マルチチャンネルに対応したりすることを特徴とするものがあります。そうした製品は、シリアル・モードにおけるコードの読み出しを高速化するために、2/4/8本のSDOラインを持つSPIポートを備えています。複数のSPIポートを備えるMCUであれば、それらのSPIポートを同時に使用してデータを高速に読み出すことができます（図11）。

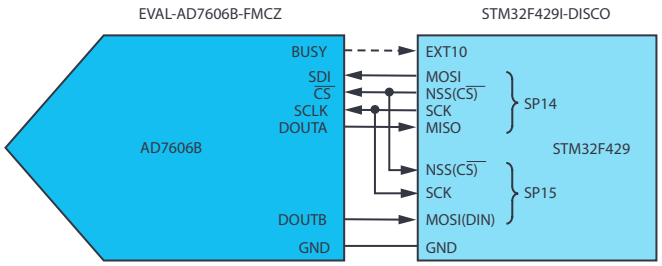


図11. EVAL-AD7606B-FMCZとSTM32F429I-DISCOの接続。MCUのSPIをメイン・モード／ノード・モードで使用し、DOUTAとDOUTBからのデータを受信します。

```
/*## EXTI0 in Polling Mode and SPI4 DMA to Receive AD7768 Data bits on DOUT0 ###*/
// Polling for EXTI0 (/DRDY) rising edge to start MCU SPI ports
while (EXTI0_Flag != SET);// wait for EXTI0 interrupt (/DRDY rising edge)
EXTI0_Flag = RESET;// clear flag variable
// throw out the last byte/word captured in the previous ODR cycle !!!
Rx_temp = *(__IO uint8_t *)&Spi4Handle.Instance->DR;
Spi4Handle.hdmarx->Instance->NDTR = SPI4_CNVByteNum;// set data number to read
Spi4Handle.hdmarx->Instance->PAR = (uint32_t)&(Spi4Handle.Instance->DR);// source address
Spi4Handle.hdmarx->Instance->M0AR = (uint32_t)(SPI_RxBuffer+RxBuf_Idn); // target address
//*** clear event flags corresponding to the stream in DMA_LISR or DMA_HISR register ***//
((DMA_Base_Registers *)Spi4Handle.hdmarx->StreamBaseAddress)->IFCR = 0x3FU << Spi4Handle.hdmarx->StreamIndex;
__HAL_DMA_ENABLE(Spi4Handle.hdmarx);
while ((Spi4Handle.hdmarx->Instance->CR & DMA_SxCR_EN) == SET) // hardware cleared
{;} // ADC data received in the target memory buffer
SPI4_CNVCount++;
RxBuf_Idn = SPI4_CNVCount * SPI4_CNVByteNum;
```

図10. DMAを使用する場合のファームウェアのコード例。
ポーリング・モードのEXTI0とSPI4のDMAを使用して、AD7768のDOUT0から出力されるデータを受信します。

図12は、32F429IDISCOVERYのSPI4をSPIのメイン、SPI5をSPIのノードとして使用し、AD7606Bの評価用ボード「EVAL-AD7606B-FMCZ」上にあるDOUTA/DOUTBからデータを受信する場合のコード例です。

AD7606Bは、分解能が16ビットで8チャンネルの同時サンプリングが可能なデータ・アクイジション・システム製品です。各チャンネルは、アナログ入力のクランプ保護機能、PGA（プログラマブル・ゲイン・アンプ）、ローパス・フィルタ、16ビットの逐次比較型（SAR）ADCを備えています。また、柔軟性の高いデジタル・フィルタ、2.5V／低ドリフトの高精度リファレンス、ADCを駆動するためのリファレンス・バッファ、柔軟性の高いパラレル／シリアル・インターフェースも内蔵しています。5Vの単電源で動作し、±10V、±5V、±2.5Vの真のバイポーラ入力範囲に対応します。そして、各チャンネルでは800kSPSのスループットを実現できます。

図13は、240kSPSで動作するAD7606Bの各信号をモニタした結果です。デジタル・インターフェースのうち、BUSY、SCLK、DOUTA、DOUTBをキャプチャしています。

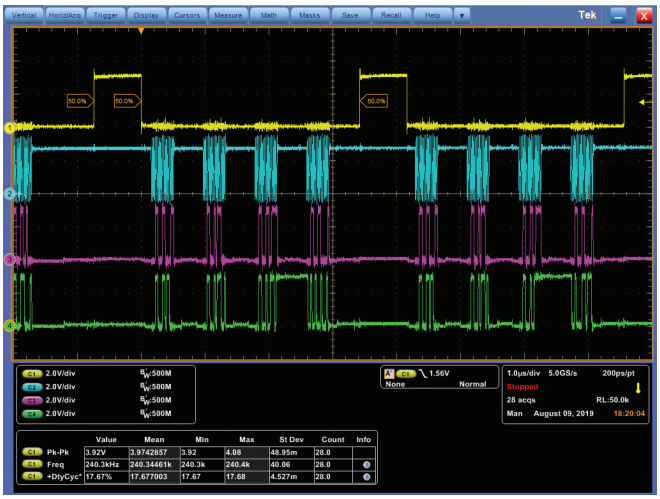


図13. AD7606Bの各信号。BUSY、SCLK、DOUTA、DOUTBのデータをオシロスコープでキャプチャしました。

```
/*## Configure the SPI4 as Master and SPI5 as Slave ###*/
Spi4Handle.Init.Direction      = SPI_DIRECTION_2LINES;
Spi4Handle.Init.CLKPhase       = SPI_PHASE_1EDGE;//read at DCLK falling edge
Spi4Handle.Init.CLKPolarity    = SPI_POLARITY_HIGH;//read at DCLK falling edge
Spi4Handle.Init.DataSize       = SPI_DATASIZE_16BIT;
Spi4Handle.Init.NSS            = SPI_NSS_SOFT;// NSS pin is configured as GPIO output for /CS
Spi4Handle.Init.Mode           = SPI_MODE_MASTER;// SPI4 as SPI Master
Spi5Handle.Init.Direction      = SPI_DIRECTION_2LINES_RXONLY;// only receive data
Spi5Handle.Init.NSS            = SPI_NSS_HARD_INPUT;
Spi5Handle.Init.Mode           = SPI_MODE_SLAVE;// SPI5 as SPI Slave
/*## Enable SPI4 as Master and SPI5 as Slave to Receive AD7606B Codes ###*/
__HAL_SPI_ENABLE(&Spi4Handle);
__HAL_SPI_ENABLE(&Spi5Handle);
while (SPI4_CNVCount < SPI4_CNVNum)
{
          CLR_CNV();
          SET_CNV();//AD7606B conversion start
          // wait for conversion finish, BUSY goes from high to low. Polling or interrupt mode
          while (BUSY == SET) {;}
          while (SPI4_WordCount < SPI4_CNVWordNum)// code number to read per conversion cycle
          {
                    CLR_CS();
                    *(__IO uint8_t *)&Spi4Handle.Instance->DR = 0;
                    while (__HAL_SPI_GET_FLAG(&Spi4Handle, SPI_FLAG_RXNE) != SET);
                    Delay_xus(1);// need half SCLK cycle delay for slow SCLK rate < 10MHz
                    SET_CS();
                    SPI_RxBuffer[RxBuf_Idn] = *(__IO uint16_t *)&Spi4Handle.Instance->DR;
                    SPI_RxBuffer[RxBuf_Idn+ADCSDO1_WordIdn] = *(__IO uint16_t \\
                    *)&Spi5Handle.Instance->DR;
                    RxBuf_Idn++;
                    SPI4_WordCount += 2;
          }
          SPI4_CNVCount++;
          RxBuf_Idn = SPI4_CNVCount * SPI4_CNVWordNum;
          SPI4_WordCount = 0;
}//while (SPI4_CNVCount < SPI4_CNVNum)
__HAL_SPI_DISABLE(&Spi4Handle);
__HAL_SPI_DISABLE(&Spi5Handle);
```

図12. SPI4をメイン、SPI5をノードとして構成する場合のコード例

まとめ

本稿では、MCUのSPIを使用して、非標準的なSPIを備えるADCにアクセスする方法を紹介しました。本稿で説明した内容をそのまま適用するか、多少の調整を加えることにより、SPIのメインとして動作するADCや、スループットを高めるために複数のDOUTラインを備えるADCのSPIを制御することができます。

謝辞

STM32F429IDISCOVERYキットの立ち上げやファームウェアのデバッグについて助言してくれたMika Jiang、Yao Zhaoの両アプリケーション・エンジニアに感謝します。

参考資料

Piyu Dhaker「SPIの基本を学ぶ」Analog Dialogue 52-09、2018年9月

「RM0090 Reference Manual: STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 Advanced ARM®-Based 32-Bit MCUs（RM0090リファレンス・マニュアル：ARM®ベースの高度な32ビットMCU「STM32F405/415」、「STM32F407/417」、「STM32F427/437」、「STM32F429/439」）」STMicroelectronics、2019年2月

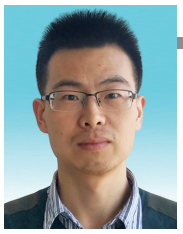
「STM32F427xx Data Sheet（STM32F427xx データシート）」STMicroelectronics、2018年1月

「UM1670 User Manual: Discovery Kit with STM32F429ZI MCU（UM1670 ユーザー・マニュアル：STM32F429ZI用のディスカバリ・キット）」STMicroelectronics、2017年9月

Miguel Usach「AN-1248 アプリケーション・ノート：SPIインターフェース）」Analog Devices、2015年9月

著者について

Steven Xie（steven.xie@analog.com）は、アナログ・デバイセズのプロダクト・アプリケーション・エンジニアです。2011年3月から北京支社の中国デザイン・センターに勤務。中国全土を対象として、SAR ADC製品の技術サポートを担当しています。それ以前は、無線通信基地局向けのハードウェア設計に4年間携わっていました。2007年に北京航空航天大学で通信/情報システムに関する修士号を取得しています。



Steven Xie