ANALOG DEVICES

AN-1435 アプリケーション・ノート

ADuCM3027/ADuCM3029 シリアル・ポートを使用した UART の実装

Sachin Dwivedi 執筆

はじめに

ADuCM3027/ADuCM3029 プロセッサで同期シリアル・ペリフェラル・ポート (SPORT) を使用すると、ソフトウェアのオーバヘッドを最小限に抑え、全二重、非同期ポートを実装してユニバーサル非同期レシーバー／トランスミッタ (UART) と通信できます。このアプリケーション・ノートでは、複数の標準ボーレートで完全な UART インターフェースを実装する方法を説明します。

SPORT の概要

SPORT インターフェースは、多数のシリアル・データ通信プロトコルをサポートします。さらに、SPORT は、多数の業界標準データ・コンバータ、コーデック、デジタル・シグナル・プロセッサ（DSP）などのプロセッサに対して接着剤を使用しないハードウェア・インターフェースを提供します。

SPORT インターフェースの主な特徴と機能

- 連続して実行するクロック
- MSB ファーストまたは LSB ファーストの 3 ビット～32 ビット長のシリアル・データ・ワード
- 2 つの同期送信および 2 つの同期受信データ信号
- サポートされるデータ・ストリームの合計を倍にするバッファ
- 構成可能なフレーム同期信号

SPORT インターフェースの詳細については、https://wiki.analog.com/resources/eval/sdp/sdp-b/peripherals/sport を参照してください。

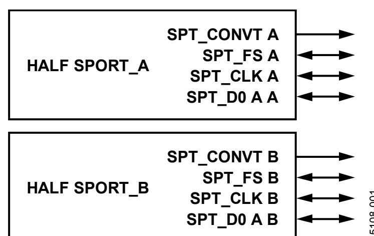

図 1. SPORT 信号

アナログ・デバイセズ社は、提供する情報が正確で信頼できるものであることを期していますが、その情報の利用に関して、あるいは利用によって生じる第三者の特許やその他の権利の侵害に関して一切の責任を負いません。また、アナログ・デバイセズ社の特許または特許の権利の使用を明示的または暗示的に許諾するものでもありません。仕様は、予告なく変更される場合があります。本紙記載の商標および登録商標は、それぞれの所有者の財産です。※日本語版資料は REVISION が古い場合があります。最新の内容については、英語版をご参照ください。

©2016 Analog Devices, Inc. All rights reserved.

Rev. 0

アナログ・デバイセズ株式会社

本　　社／〒105-6891 東京都港区海岸 1-16-1　ニューピア竹芝サウスタワービル
電話 03（5402）8200

大阪営業所／〒532-0003 大阪府大阪市淀川区宮原 3-5-36　新大阪トラストタワー
電話 06（6350）6868

目次

はじめに 1
SPORT の概要 1
改訂履歴 2
非同期通信 3
非同期 SPORT トランスミッタ 3
非同期 SPORT レシーバー 3
ハードウェアとソフトウェアの概要 4
ハードウェアの概要 4
ソフトウェアの概要 4
ドライバ関数のプロトタイプ 5
ソフトウェアのフローチャート 6
SPORT_A ブロック転送 6
SPORT_B ブロック受信 7
波形 8
SPORT_UART_Emulator のコード 9
SPORT_UART_Emulator.h 9
SPORT_UART_Emulator_Transmit.c 14
SPORT_UART_Emulator_Receive.c 14
まとめ 16

改訂履歴

4/2017−Revision 0: Initial Version

非同期通信

同期／非同期のシリアル通信の相違は、クロック信号とフレーム同期信号の有無にあります。同期シリアル・ポートには、クロック信号とオプションのフレーム同期信号があります。非同期ポートには、クロック信号とオプションのフレーム同期信号はありません。クロック信号がない非同期ポートでは、所定のデータ・レート（ビット・レート）で通信する必要があります。フレーム同期がない場合、ワード・フレーム情報はデータ・ストリームに組み込まれます。開始ビットは、転送の開始をマーキングします。停止ビットは、転送の完了をマーキングします。ワード長は、レシーバーとトランスミッタの間で事前に指定されます。

非同期 SPORT トランスミッタ

シリアル・ポートの送信側は、UART の対象ビット・レートと同じクロック・レートで、内部クロックの生成向けに構成する必要があります。この構成は、SPORT_A ブロックのクロック分周器レジスタ（SPORT_DIV_A）の CLKDIV ビットをセットして実行されます。

SPORT_DIV_A レジスタの CLKDIV ビット =

$$\frac{PCLK}{2 \times Baud\ Rate} - 1$$

PCLK はペリフェラル・クロック信号。

SPORT_A クロックは、シリアル・ポートを目的のビット・レートと同期するために使用されます。実際のクロック信号（SPORT_ACLK）には、何も接続しません。フレーム同期信号（SPORT_AFS）が内部で生成され、信号がフロートで維持されるように構成します。SPORT_A ブロックは、UART 転送をエミュレートするため、常に LSB ファーストで転送する必要があります。SPORT_CTL_A レジスタの SLEN フィールドにある SPORT_A ブロックで転送されるビット数をプログラムします。SPORT_NUMTRAN_A レジスタで転送されるワードの合計数をプログラムします。各ワード・サイズは、SLEN フィールドで指定します。

SPORT_A ブロックが UART デバイスに送信する SPORT 転送では、UART は常に最初の転送データを 0x00 として受信します。これは破棄可能で、SPORT_A ブロックに転送される正しいデータ・シーケンスが追従します。転送の開始時に（構成後）UART Rx ラインがアイドル・ハイ（ロジック 1）になり、SPORT データ・ラインがアイドル・ロー（ロジック 0）になるため、このシーケンスが発生します。UART は、このロジック 0 を開始ビットとして解釈し、ロジック 0 のフレーム全体を転送の開始時に受信します。

非同期 SPORT レシーバー

シリアル・ポートは、内部フレーム同期信号なしで新しい転送を開始するかどうか決定する必要があります。UART デバイスの転送ピンは、ADuCM3029/ADuCM3027 の SPORT_B ブロックのデータ・ライン・ピン（SPORT_BD0）とフレーム同期ピン（SPORT_BFS）に接続します。SPORT_B ブロックは、内部クロックの生成とアクティブ・ロー外部フレームの同期信号向けに構成されます。

SPORT は、入力ビット・ストリームとのフェーズ同期を確保しないので、入力の非同期データ・ストリームをオーバーサンプリングする必要があります。SPORT の受信クロックは、目的のボーレートの 3 倍に設定する必要があります。例えば、ADuCM3029/ADuCM3027 SPORT が UART デバイスと 9600 bps で通信する場合は、SPORT の受信クロックを 28,800 bps に設定する必要があります。この設定では、適切な分周比を計算するか、SPORT_B ブロックのクロック分周器レジスタ（SPORT_DIV_B）の CLKDIV ビットをプログラミングします。

SPORT_DIV_B レジスタの CLKDIV ビット =

$$\frac{PCLK}{2 \times 3 \times Baud\ Rate} - 1$$

アクティブ・ロー・フレームの同期信号（SPORT_BFS）は、内部生成クロック（SPORT_BCLK）のアクティブ・エッジでポーリングされます。UART パケットの下位レベルにある開始ビットにより SPORT_BFS 信号がアサートされると、SPORT_B ブロックは UART デバイスから転送されたワードの受信を開始し、パケットの N ビットすべてが受信されるまで SPORT_BFS ラインをチェックしません（N は SPORT_CTL_B レジスタの SLEN フィールドによってプログラムされます）。SPORT は、オーバーサンプリングされた開始ビットをフレームの同期に使用して、入力の非同期データ・ストリームの受信を開始します。

ハードウェアとソフトウェアの概要

ハードウェアの概要

図 2 に、ADuCM3029/ADuCM3027 SPORT と別のデバイスの基本的な UART ポートの送信（Tx）ピンと受信（Rx）ピンとの接続を示します。

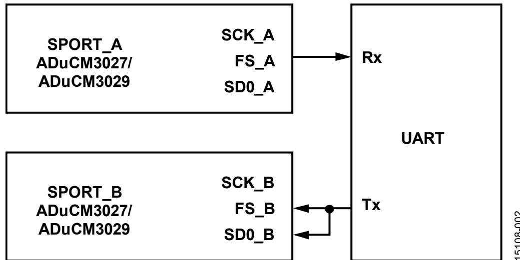

図 2. ADuCM3029/ADuCM3027 マイクロコントローラ・ユニット（MCU）から UART へのインターフェース

ソフトウェアの概要

SPORT から非同期で入出力するデータを管理するには、最低限のソフトウェアが必要です。SPORT の送信と受信用の C 関数については、Code for the SPORT_UART_Emulator セクションで説明します。このコードは、UART ホストと ADuCM3029/ADuCM3027 SPORT の間で双方向の転送に対して、複数のボーレートと転送回数で試験済みです。

非同期トランスミッタ（SPORT_A ブロック）

送信側では、転送する N ビットのデータを UART 伝送パケットにフォーマットする必要があります。UART デバイスを正しく受信できるように、開始ビットと停止ビットをワードに追加する必要があります。

データ形式の例は、次のようになります。

- 8-N-1 の転送形式（8 ビット・データ + 0 パリティ・ビット + 1 停止ビット）の場合、データ = 0xAA（b#1010 1010）
- 変更されたデータ = b# 1 10101010 0 1（1 停止ビット + 8 ビット・データ + 1 開始ビット + 1 停止ビット）

ワード全体が送信されると、SPORT_AD0 ラインは LSB の値を保持するので（LSB が最初に伝送された場合）、転送の最初に停止ビットを追加する必要があります。連続したバイト間で生成される信号でグリッチ（データの破損の原因になる）を防ぐため、UART Rx ラインはアイドル・ハイに設定する必要があります。

非同期 SPORT レシーバー（SPORT_B ブロック）

SPORT_B ブロックは、オーバーサンプリングされたデータを受信するので、受信側は送信側よりも複雑になります。8-N-1 の転送形式では、データは 3 の係数でオーバーサンプリングされるので、シリアル・ポートは 27 ビットを受信するよう、フレーム同期（SPORT_BFS）で計上される 3 つのサンプリング開始ビットを破棄するようにプログラムする必要があります。受信される 27 ビットは、UART デバイスが送信するパケット、8 ビットのデータ、1 停止ビットを表します（3 の係数でオーバーサンプリングされます）。

実際のデータは、ビット操作により、オーバーサンプリングされたデータから抽出されます。中間のビット、つまり正しい値は UART デバイスから送信された各ビットに対して、受信データの 3 ビット・シーケンスから抽出されます。抽出されたビットは、1 バイトのデータを形成するように組み立てられます。

DATA FORMAT: UART	START BIT	DATA BYTE = 8 BITS								STOP BIT
		LSB	1	2	3	4	5	6	MSB	
EQUIVALENT BIT PATTERN FOR SPORT0	000	xxx	yyy	xxx	yyy	xxx	yyy	xxx	yyy	111
	3 ZEROES	BYTE REPRESENTED BY 24 BITS								3 ONES

15108-003

図 3. UART フレームと SPORT 受信フレームで予想されるデータ形式

ドライバ関数のプロトタイプ

次の関数は、8 ビットの同期データで動作しますが、他のデータ幅をサポートするように簡単に変更できます。使用事例の C 関数の詳細については、SPORT_UART_Emulator のコードセクションで説明します。

SPORT_UART_Tx_Initialise 関数は、UART 転送エミュレーション向けに ADuCM3029/ADuCM3027 プロセッサの SPORT_A ブロックを構成してセットアップします。SPORT_A 内部クロックは PCLK から供給され、6.5 MHz に構成されます（デフォルト）。SPORT_CTL_A レジスタを使用して、希望する送信ボーレートを設定します。SPORT_IEN_A レジスタを使用して、送信するデータ・バッファが空であることを通知するための割込みを構成します。SPORT_A ブロックを有効にする前に、SPORT_NUMTRAN_A レジスタを使用して転送するワード数をプログラムできます。

SPORT_UART_Rx_Initialise 関数は、UART 受信エミュレーション向けに ADuCM3029/ADuCM3027 プロセッサの SPORT_B ブロックを構成してセットアップします。SPORT_B ブロックは 3 の係数で入力データ・ストリームをオーバーサンプリングするように構成されます。フレーム同期は、外部、ロー、アクティブ状態に対して構成されます。SPORT_IEN_B レジスタを使用して、受信するデータ・バッファがフルであることを通知するための割込みを構成します。また、SPORT_B ブロックを有効にする前に、SPORT_CTL_B レジスタの SLEN フィールドを次のように構成します。

3 ×（ワード・サイズ + 停止ビットの数）− 1

SPORT_UART_Tx_Transfer 関数は、バッファによってポイントされるロケーション内のデータを変更することで、UART 転送データの形式を作成します。その後、変更されたデータは、転送のために SPORT_A_TX レジスタに出力されます。この関数は、ビット・マスキングとシフト動作を使用します。

SPORT_UART_Rx_Transfer 関数は、SPORT_B_RX レジスタからオーバーサンプリングされたデータを受信します。ビット操作では、SPORT_B_RX データの各 3 ビット・シーケンス（UART デバイスが送信した 1 ビットごとに受信される 3 ビット）の中間ビットが抽出されます。抽出されたビットは、バイト・サイズのデータに組み立てられます。この関数は、実際に受信されたデータを表す、組み立てられたバイトを返します。次のサンプル・コードは、27 ビットの SPORT レジスタから 8 ビットの UART 形式でデータを抽出する方法を示します。

```
/* Receive data into the RX buffer */
temp = *pREG_SPORT0_RX_B;
  /* Extract the 8 bits from the 27 bits received
  */
  value = 0;
  value += ((temp >> 23) & (1 << 0));
  value += ((temp >> 19) & (1 << 1));
  value += ((temp >> 15) & (1 << 2));
  value += ((temp >> 11) & (1 << 3));
  value += ((temp >> 7) & (1 << 4));
  value += ((temp >> 3) & (1 << 5));
  value += ((temp >> 1) & (1 << 6));
  value += ((temp >> 5) & (1 << 7));
  /* Return the assembled byte */
  return value;
```

ソフトウェアのフローチャート

SPORT_A ブロック転送

SPORT_A ブロックは、UART Tx ポートをエミュレートします。SPORT_A ブロックを使用して UART Tx ポートをエミュレートするには、SPORT ブロックを転送向けに初期化してイネーブルにする必要があります。イネーブルの場合は、転送を保留されているデータがあるか、SPORT_STAT レジスタが確認されます。転送するデータがある場合は、保留中のデータから UART データ・パケットを作成し、SPORT 転送レジスタにデータ・パケットの書込みを実行します。

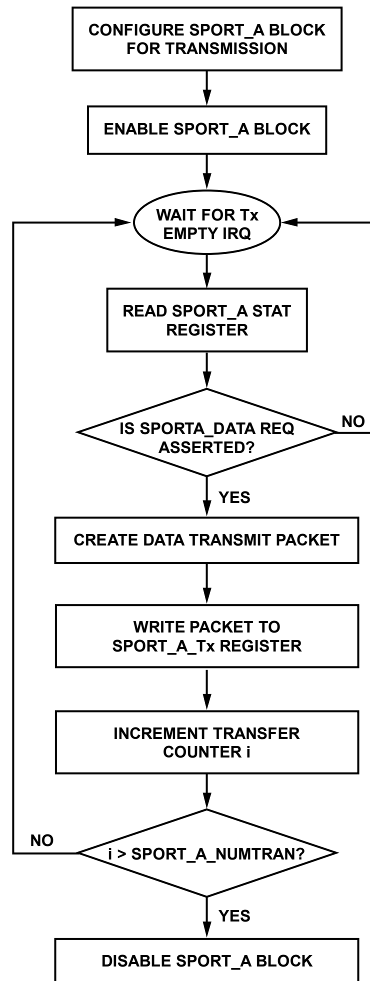

図 4. SPORT_A ブロック転送のフローチャート

SPORT_B ブロック受信

SPORT_B ブロックは、UART Rx ポートをエミュレートします。SPORT_B を UART Rx ポートとして使用するには、最初に SPORT_B ブロックを受信向けに初期化してイネーブルにする必要があります。SPORT がイネーブルの場合は、SPORT_B データ・レジスタに転送を保留されているデータがあるか、SPORT_STAT レジスタが確認されます。データがある場合は、そのデータを取得して 8 ビットの UART データを抽出します。

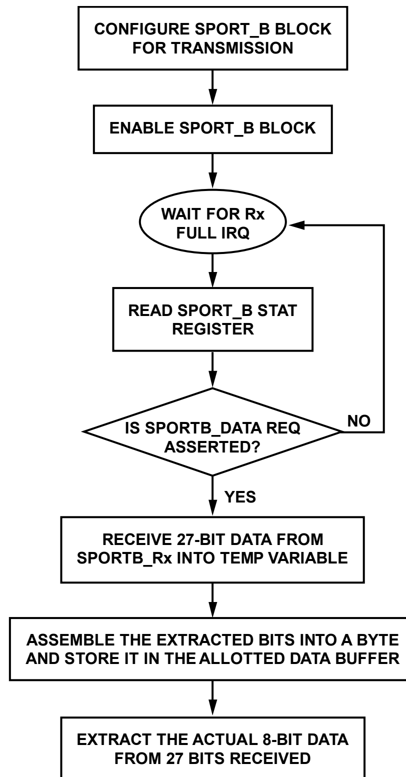

図 5. SPORT_B ブロック受信のフローチャート

波形

図 6 に、SPORT_A ブロックからの送信と UART デバイスでの受信の波形を示します。9600 bps で 8 ビット・データ（0x96）の単一フレーム、UART 転送エミュレーションに必要な追加のフォーマット・ビットが含まれます。

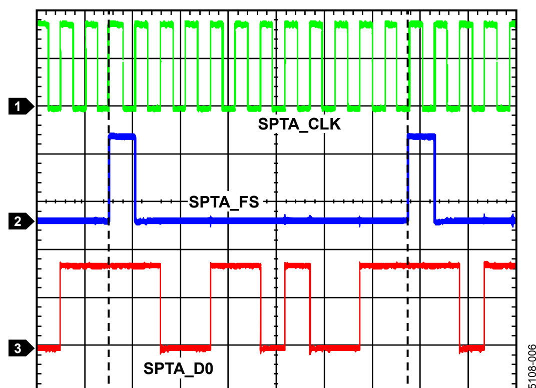

図 6. 1 つのフレームに対する SPORT_A ブロック転送と UART デバイスの受信

図 7 に、UART デバイスからの送信と SPORT_B での受信の波形を示します。9600 bps で 8 ビット・データ（0x96）（開始ビットと停止ビット）の単一フレーム、UART 受信エミュレーションに適した 3 倍の転送ボーレートでサンプリングされています。

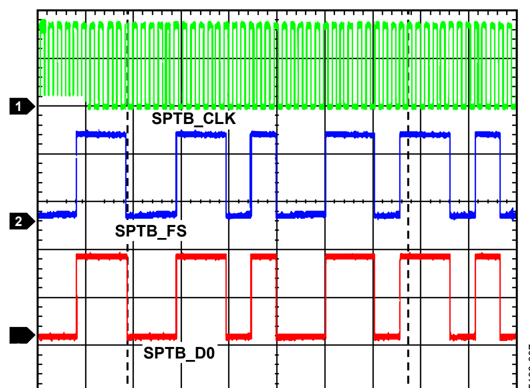

図 7. 単一フレームに対する UART デバイスの送信と SPORT_B ブロックの受信

図の赤いパターンは、単一フレームの転送を示します。

SPORT_UART_EMULATOR のコード

ここで示すコードは、次の使用事例のサンプルです。

- SPORT_A ブロックからの送信と UART による受信
- UART からの送信と SPORT_B ブロックによる受信

単一フレームの転送に使用されるデータ形式は、8-N-1 です（1 開始ビット、8 ビットのデータ、0 パリティ・ビット、1 停止ビット）。これらの事例では、PCLK = 26 MHz と複数のボーレートで試験を実施しています。

SPORT_UART_EMULATOR.H

```
/* SPORT Based UART Emulator Application */
/* SPORT_A emulates Transmission Side while SPORT_B emulates Reception Side */
/* Two Use Cases
       (a) Transmission from SPORT_A and Reception by UART
(b) Transmission from UART and Reception by SPORT_B */
/* Tested with PCLK = 26 MHz and Baud Rates - 9600bps, 19200bps, 38400bps, 57600bps,  115200bps,
230400bps */
/* Define the word_size and baud_rate for UART before proceeding */

#include "system.h"
#include "startup.h"
#include "stdint.h"

/* Definitions used for supporting both use cases */

#define SLEN_TX     (word_size + stopbits + paritybit + 1)
#define SLEN_RX     (3 * (word_size + stopbits + paritybit) - 1)
#define FSDIV_TX    (word_size + stopbits + paritybit + 2)
#define SYS_PCLK    26000000
#define TRAN_SIZE 3
#define baud_rate 9600
#define word_size 8
#define stopbits    1

/* Global Variables used for both use cases */

uint32_t temp;
uint8_t flag = 0;
uint8_t tbuf[TRAN_SIZE];
uint8_t rbuf[TRAN_SIZE];

int i=0; /* Transfer Loop Counter */
uint16_t res;

/* Definitions for Functions used for both use cases */
void Change_CLKDIV(int pVal, int hVal);
void SPORT_UART_Tx_Initialise();
void SPORT_UART_Rx_Initialise();
```

```
void SPORT_UART_Tx_Transfer(uint8_t *buf);
uint8_t SPORT_UART_Rx_Transfer();

/* Description: Function to change the PCLKDIV and HCLKDIV
   Input Parameters:       int pVal - Value of PCLK Divisor
                     int hVal - Value of HCLK Divisor
   Return:           void
*/
void Change_CLKDIV(int pVal, int hVal)
{
  uint32_t uiTemp;
  // Change PCLKDIVCNT
  uiTemp = *pREG_CLKG0_CLK_CTL1;
  uiTemp &= ~(BITM_CLKG_CLK_CTL1_PCLKDIVCNT);
  uiTemp |= (pVal << BITP_CLKG_CLK_CTL1_PCLKDIVCNT);
  *pREG_CLKG0_CLK_CTL1 = uiTemp;

  // Change HCLKDIVCNT
  uiTemp = *pREG_CLKG0_CLK_CTL1;
  uiTemp &= ~(BITM_CLKG_CLK_CTL1_HCLKDIVCNT);
  uiTemp |= (hVal << BITP_CLKG_CLK_CTL1_HCLKDIVCNT);
  *pREG_CLKG0_CLK_CTL1 = uiTemp;
}

/* Description: Function to initialize and configure the SPORT_A for UART Transmission Emulation
   Input Parameters:       void
   Return:           void
*/
void SPORT_UART_Tx_Initialise()
{
  float value;
  value = ((SYS_PCLK / (2 * baud_rate)) - 1);

  /* Configure the GPIO pins as alternate functions for SPORT_A_Tx */
  *pREG_GPIO2_CFG |= (1 << BITP_GPIO_CFG_PIN00) | (1 << BITP_GPIO_CFG_PIN02);
  *pREG_GPIO1_CFG |= (1<< BITP_GPIO_CFG_PIN15);
  *pREG_GPIO0_CFG |= (1<< BITP_GPIO_CFG_PIN12);
  *pREG_GPIO0_PE |= (1 << 12);

  /* Disable the SPORT_A_Tx before the configuration*/
  *pREG_SPORT0_CTL_A &= ~(1 << BITP_SPORT_CTL_A_SPEN);

  /* Configure CLk Divider */
  *pREG_SPORT0_DIV_A |=    ((uint16_t) value << BITP_SPORT_DIV_A_CLKDIV) |
  ((FSDIV_TX) << BITP_SPORT_DIV_A_FSDIV);

  /* Configure the Data interrupts and the Transfer Complete interrupts */
```

```
  *pREG_SPORT0_IEN_A |= (1<< BITP_SPORT_IEN_A_TF) | (1<< BITP_SPORT_IEN_A_DATA);

  /* Program Number of Transfers */
  *pREG_SPORT0_NUMTRAN_A = TRAN_SIZE;

  /* Write the CTL register */
  *pREG_SPORT0_CTL_A | = ((SLEN_TX) << BITP_SPORT_CTL_A_SLEN)
    | (1 << BITP_SPORT_CTL_A_ICLK)
    | (1 << BITP_SPORT_CTL_A_IFS)
    | (1<< BITP_SPORT_CTL_A_FSR)
    | (1 << BITP_SPORT_CTL_A_SPTRAN)
    | (1 << BITP_SPORT_CTL_A_LSBF);

  /* Enable SPORT_A */
  *pREG_SPORT0_CTL_A |= (1<< BITP_SPORT_CTL_A_SPEN);
}

/* Description: Function to initialize and configure the SPORT_B for UART Reception Emulation
Input Parmaeters:   void
Return:                     void
*/
void SPORT_UART_Rx_Initialise()
{
  float value;
  value = ((SYS_PCLK /(2 * 3 * baud_rate)) - 1);

  /* Configure the GPIO pins as alternate functions for SPORT_B_Rx */
  *pREG_GPIO0_CFG |= (2 << BITP_GPIO_CFG_PIN00)
    | (2 << BITP_GPIO_CFG_PIN01)
    | (2 << BITP_GPIO_CFG_PIN02)
    | (2 << BITP_GPIO_CFG_PIN03);

  /* Configure Clk Divider */
  *pREG_SPORT0_DIV_B |= ((uint16_t) value << BITP_SPORT_DIV_B_CLKDIV);

  /* Use external FS */
  /* Configure Data interrupts and Transfer Complete Interrupt */
  *pREG_SPORT0_IEN_B = (1<< BITP_SPORT_IEN_B_TF) | (1<< BITP_SPORT_IEN_B_DATA);

  /* Program Number of Transfers */
  *pREG_SPORT0_NUMTRAN_B = 2;

  /* Write to CTL register */
  *pREG_SPORT0_CTL_B | = ((SLEN_RX) << BITP_SPORT_CTL_B_SLEN)
    | (1 << BITP_SPORT_CTL_B_ICLK)
    | (1 << BITP_SPORT_CTL_B_FSR)
    | (1 << BITP_SPORT_CTL_B_LFS);
```

```
  /* Enable SPORT_B to receive data */
  *pREG_SPORT0_CTL_B |= (1<< BITP_SPORT_CTL_B_SPEN);
}

/* Decription: Function to transmit data from SPORT_A_TX register to UART Device after formatting
   Input Parameters:       uint8_t *buf - Value of the data to be transmitted
   Return :          void
*/
void SPORT_UART_Tx_Transfer(uint8_t *buf)
{
  uint16_t res;

  /* Place a start and a stop bit */
  uint16_t tx_mask, tx_startbits, tx_stopbits;

  /* Create Masks for transmitting the word
  Example: if word_size = 8
  tx_mask = b'11111111
  tx_startbits = b'01111111100
  tx_stopbits = b'10000000001
  */

  tx_mask = (1 << word_size) - 1;
  tx_startbits = tx_mask << 2;
  tx_stopbits = ((0x0C) << (word_size + paritybit)) | 1;

  /* Remove all the bits that won't be transmitted */
  (*buf) &= tx_mask;
  res = (*buf) << 2;       /* Make space for the start bit and previous stop bit */
  res &= tx_startbits;     /* Add the start bit */
  res |= tx_stopbits;      /* Add the stop bits */

  /* Put this value into the SPORTA_TX register */
  *pREG_SPORT0_TX_A = res;
}

/* Description: Function to receive data into SPORT_B_RX register from UART Device,
                extract the sampled bits and return the assembled data for storage.
   Input Parameters:       void
   Return:           uint8_t value - Assembled Received Data for storage
*/
uint8_t SPORT_UART_Rx_Transfer()
{
  /* Oversample by 3 and extract the middle bit of every transmittted bit */
  uint32_t value;
```

```
/* Get the received middle stop bit */
uint8_t rxd_stop;

/* Receive data into Rx Buffer */
temp = *pREG_SPORT0_RX_B;

/* Extract the 8 bits from the 27 bits received */
value = 0;

switch (word_size)
{
  case 8: value += ((temp >> 23) & (1 << 0));      // bit 0
    value += ((temp >> 19) & (1 << 1));      // bit 1
    value += ((temp >> 15) & (1 << 2));      // bit 2
    value += ((temp >> 11) & (1 << 3));      // bit 3
    value += ((temp >> 7) & (1 << 4));       // bit 4
    value += ((temp >> 3) & (1 << 5));       // bit 5
    value += ((temp << 1) & (1 << 6));       // bit 6
    value += ((temp << 5) & (1 << 7));       // bit 7
    break;
  case 7: value += ((temp >> 20) & (1 << 0));
    value += ((temp >> 16) & (1 << 1));
    value += ((temp >> 12) & (1 << 2));
    value += ((temp >> 8) & (1 << 3));
    value += ((temp >> 4) & (1 << 4));
    value += ((temp >> 0) & (1 << 5));
    value += ((temp << 4) & (1 << 6));
    break;
  case 6: value += ((temp >> 17) & (1 << 0));
    value += ((temp >> 13) & (1 << 1));
    value += ((temp >> 9) & (1 << 2));
    value += ((temp >> 5) & (1 << 3));
    value += ((temp >> 1) & (1 << 4));
    value += ((temp << 3) & (1 << 5));
    break;
  case 5: value += ((temp >> 14) & (1 << 0));
    value += ((temp >> 10) & (1 << 1));
    value += ((temp >> 6) & (1 << 2));
    value += ((temp >> 2) & (1 << 3));
    value += ((temp << 2) & (1 << 4));
}

*pREG_SPORT0_CTL_B &= ~(1<< BITP_SPORT_CTL_B_SPEN);
*pREG_SPORT0_NUMTRAN_B = 2;
*pREG_SPORT0_CTL_B |= (1<< BITP_SPORT_CTL_B_SPEN);
```

```
  return value;
}

/* Interrupt Handler Routine for SPORT_A_TX */
void SPORT0A_Int_Handler()
{
  if ((i < (TRAN_SIZE)) && (*pREG_SPORT0_STAT_A & BITM_SPORT_STAT_A_DATA))
  {
    SPORT_UART_Tx_Transfer(&tbuf[i++]);
  }
  if(i >= TRAN_SIZE)
  {
    *pREG_SPORT0_CTL_A &= ~(1<< BITP_SPORT_CTL_A_SPEN);
  }
}

/* Interrupt Handler Routine for SPORT_B_RX */
void SPORT0B_Int_Handler()
{
  if((i < TRAN_SIZE) && (*pREG_SPORT0_STAT_B & BITM_SPORT_STAT_B_DATA))
  {
    rbuf[i++] = SPORT_UART_Rx_Transfer();
  }
  if(i >= TRAN_SIZE)
  {
    *pREG_SPORT0_CTL_B &= ~(1<< BITP_SPORT_CTL_B_SPEN);
  }
}
```

SPORT_UART_EMULATOR_TRANSMIT.C

```
#include "SPORT_UART_Emulator.h"

/* Main Function for Use Case (a) Transmission from SPORT_A and Reception by UART */
int main()
{
  /* Change PCLK to 26 MHz */
  Change_CLKDIV(1, 1);

  /* Enable the NVIC IRQ ID for SPORT A handler */
  NVIC_EnableIRQ(SPORT_A_EVT_IRQn);

  /* Create Data pattern for transmit buffer */
  for (int i=0; i < TRAN_SIZE; i++)
  {
    tbuf[i] = 0x13 + (0x19 << (i % 5)) + (0x6D << (i % 3));
  }

  /* Configure the SPORT_A for use case */
```

```
  SPORT_UART_Tx_Initialise();

  while(1) {}
}
```

SPORT_UART_EMULATOR_RECEIVE.C

```
#include "SPORT_UART_Emulator.h"

/* Main Function for Use Case (b) Transmission from UART and Reception by SPORT_B */
int main()
{
  /* Change PCLK to 26 MHz */
  Change_CLKDIV(1, 1);

  /* Enable the NVIC IRQ ID for SPORTB_Rx handler */
  NVIC_EnableIRQ(SPORT_B_EVT_IRQn);

  /* Configure the SPORT_B for use case */
  SPORT_UART_Rx_Initialise();

  while(1) {}
      }
```

まとめ

このアプリケーション・ノートでは、ADuCM3029/ADuCM3027 プロセッサの SPORT 通信プロトコルを使用して、全二重 UART 通信をエミュレートする方法を説明します。この方法は標準の UART デバイスとのインターフェース接続に使用できます。

このアプリケーション・ノートに示す使用事例は、すべての標準ボーレートでコア・モードとダイレクト・メモリ・アクセス（DMA）モードで試験済みです。SPORT 転送サイクルでは最大 115,200 bps のボーレート、SPORT 受信サイクルでは最大 57,600 bps のボーレートで信頼性の高い結果が観察されています。双方向 5 ビット～8 ビットのデータ・サイズの転送で、適切な動作が可能か試験しています。