AN-1435
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com/cn

使用ADuCM3027/ADuCM3029串行端口实现UART

作者：Sachin Dwivedi

简介

使用ADuCM3027/ADuCM3029处理器上的同步串行端口(SPORT)可以实现全双工异步端口，以最小的软件开销与通用异步接收品/发送器(UART)通信。本应用笔记介绍如何实现适用于多个标准波特率的全双工UART接口。

SPORT概述

SPORT接口不仅支持各种串行数据通信协议，而且还能与诸多工业标准数据转换器、编解码器以及其他处理器（包括数字信号处理器(DSP)）实现无缝硬件接口。

SPORT接口的主要特性和功能包括

- 持续运行的时钟
- 串行数据字长3位到32位，MSB或LSB优先
- 两个同步发送数据信号和两个同步接收数据信号
- 支持总数据流两倍的缓冲区
- 可配置帧同步信号

有关SPORT接口的更多信息，请参见https://wiki.analog.com/esources/eval/sdp/sdp-b/peripherals/sport。

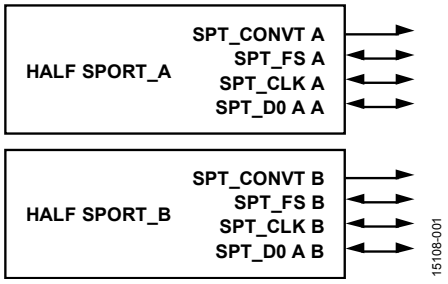

图1. SPORT信号

目录

修订历史

2017年4月—修订版0：初始版

异步通信

同步串行通信与异步串行通信的差别在于是否存在时钟信号和帧同步信号。同步串行端口具有一个时钟信号和一个可选的帧同步信号。异步端口没有时钟信号和帧同步信号。没有时钟信号时，异步端口必须以预定的数据速率（比特率）进行通信。没有帧同步信号时，字帧信息嵌入在数据流中。起始位标记传输开始。停止位标记传输结束。字长在接收器和发送器之间预先确定。

异步SPORT发送器

对于时钟速率等于UART所需比特率的内部生成时钟，必须配置串行端口的发送端。在时钟分频器寄存器(SPORT_DIV_A)中将CLKDIV位置位，便可对SPORT_A模块进行这一配置。

$$SPORT\_DIV\_A寄存器中的\ CLKDIV\ 位 = \frac{PCLK}{2\times 波特率} - 1$$

其中，*PCLK*是外设时钟信号。

SPORT_A时钟仅用于将串行端口同步到所需的比特率。实际时钟信号(SPORT_ACLK)不连接任何端口。将帧同步信号(SPORT_AFS)配置为内部生成，并使信号悬空。SPORT_A模块必须始终优先发送LSB以仿真UART发送。在SPORT_CTL_A寄存器SLEN字段编程设定SPORT_A模块要发送的位数。在SPORT_NUMTRAN_A寄存器中编程设定要传输的总字数，其中每个字的大小通过SLEN字段确定。

在SPORT_A模块向UART器件发送数据的SPORT发送中，UART始终接收首个传输数据作为0x00，这一数据是可丢弃的，后跟SPORT_A模块发送的正确数据序列。发生该序列的原因在于，开始发送时（配置完成后），UART Rx线路处于空闲高电平（逻辑1），SPORT数据线路处于空闲低电平（逻辑0）。UART在传输开始时将这一逻辑0解读为起始位并接收逻辑0的整个帧。

异步SPORT接收器

在没有内部帧同步信号时，串行端口必须确定新数据发送的开始位置。UART器件的发送引脚与ADuCM3029/ADuCM3027的SPORT_B模块上的数据线路引脚(SPORT_BD0)和帧同步引脚(SPORT_BFS)相连。SPORT_B模块配置用于内部生成的时钟和低电平有效外部帧同步信号。

由于SPORT无法保证与输入位流的任何相位同步，因此有必要对输入异步数据流进行过采样。SPORT上的接收时钟必须设置为所需波特率的三倍。例如，如果ADuCM3029/ADuCM3027 SPORT与UART器件的通信速率为9600 bps，那么SPORT接收时钟的波特率必须设置为28,800 bps。计算相应的除数并在时钟分频器寄存器(SPORT_DIV_B)中编程设定CLKDIV位即可为SPORT_B模块进行此设置。

$$SPORT\_DIV\_B寄存器中的\ CLKDIV\ 位 = \frac{PCLK}{3\times 2\times 波特率} - 1$$

低电平有效帧同步信号(SPORT_BFS)在内部生成时钟(SPORT_BCLK)的有效沿上进行轮询。如果UART数据包的低电平起始位导致SPORT_BFS信号置位，SPORT_B模块开始接收UART器件发送的字，并且在接收完UART数据包的所有N位（N由SPORT_CTL_B寄存器中的SLEN字段设定）之前不会检查SPORT_B线路。SPORT将过采样的起始位用作帧同步位以开始接收输入的异步数据流。

硬件和软件概述

硬件概述

图2显示ADuCM3029/ADuCM3027 SPORT与另一器件上基本UART端口的发送(Tx)引脚和接收(Rx)引脚的连接。

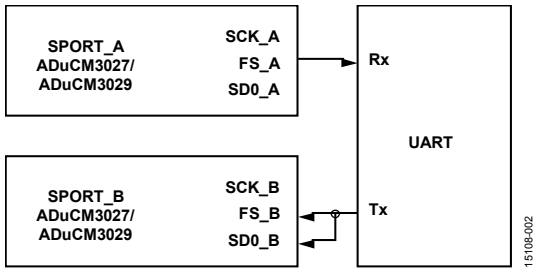

图2. ADuCM3029/ADuCM3027微控制器单元(MCU)与UART接口的连接

软件概述

管理进出 SPORT 的异步数据所需的软件最少。在“SPORT_UART_Emulator 专用代码”一节中对 SPORT 发送和接收专用的 C 函数进行了介绍。该代码针对 UART 主机与 ADuCM3029/ADuCM3027 SPORT 在两个传输方向上的多个波特率和多次传输进行测试。

异步SPORT发送器（SPORT_A模块）

在发送端，必须将待发送的 N 位数据格式化成一个 UART 发送包。在字中必须加入一个起始位和一个停止位，以便 UART 器件能够正确接收。

数据格式的示例如下：

- 对于 8-N-1 发送格式（8 位数据 + 0 奇偶校验位 + 1 个停止位），数据 = 0xAA (b#1010 1010)。
- 修改后的数据 = b# 1 10101010 0 1（1 个停止位 + 8 位数据 + 1 个起始位 + 1 个停止位）。

开始发送时必须附加一个停止位，这是因为在发送一个完整字时，SPORT_AD0 线路会保留 LSB 的值（如果 LSB 被优先发送）。必须将 UART Rx 线路设置为空闲高电平才能避免连续字节之间生成的信号中产生会导致数据损坏的毛刺。

异步SPORT接收器（SPORT_B模块）

接收端相比发送端更为复杂，因为 SPORT_B 模块接收的是过采样数据。对于 8-N-1 传输格式，由于数据的过采样系数为 3，因此必须对串行端口进行编程将其设定为接收 27 位，从而可以丢弃三个采样的起始位。在进行帧同步(SPORT_BFS)时需要考虑到这一点。接收到的 27 位代表 UART 器件发送的数据包以及 8 个数据位和 1 个停止位(过采样系数为 3)。

随后，通过位操作运算将实际数据从过采样数据中提取出来。UART 器件每发送一位，就会从已接收数据的 3 位序列中提取出值正确的中间位。提取出来的位会被汇编成一个字节的数据。

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

图3. 预期的UART帧和SPORT接收帧的数据格式

驱动程序函数原型

下列函数适用于 8 位异步数据，但也可以轻松更改为支持其他数据宽度。在“SPORT_UART_Emulator 专用代码”一节中对该用例专用的 C 函数进行了详细介绍。

SPORT_UART_Tx_Initialise 函数用于在ADuCM3029/ADuCM3027处理器上配置并部署 SPORT_A 模块以便进行 UART 发送仿真。SPORT_A 内部时钟来自时钟频率配置为 6.5 MHz（默认值）的 PCLK。发送所需的波特率已设定，另外还使用 SPORT_CTL_A 寄存器对发送进行了配置。对发送数据缓冲区为空时产生的中断信号使用 SPORT_IEN_A 寄存器进行配置。在使能 SPORT_A 模块之前，使用 SPORT_NUMTRAN_A 寄存器对要传输的字数进行编程设定。

SPORT_UART_Tx_Initialise 函数用于在ADuCM3029/ADuCM3027处理器上配置并部署 SPORT_B 模块以便进行 UART 接收仿真。SPORT_B 模块配置为对输入数据流进行过采样（过采样系数为 3）。帧同步配置用于外部低电平有效状态。对接收数据缓冲区已满时产生的中断信号使用 SPORT_IEN_B 寄存器进行配置。另外，在使能 SPORT_B 模块之前，对 SPORT_CTL_B 寄存器的 SLEN 字段进行配置，具体情况如下：

$$3 \times (\textit{字大小} + \textit{停止位数}) - 1$$

SPORT_UART_Tx_Transfer 函数通过修改缓冲区中位置指向的数据来创建 UART 发送数据格式。修改后的数据随后在 SPORT_A_TX 寄存器上输出用于发送。该函数使用位掩码和位移位运算。

SPORT_UART_Rx_Transfer 函数接收来自 SPORT_B_RX 寄存器的过采样数据。位操作运算提取 SPORT_B_RX 数据每个 3 位序列的中间位（UART 器件每发送 1 位会接收到 3 位）。提取出来的位被汇编成按字节计算大小的数据。该函数会返回表示实际已接收数据的汇编字节。下列代码示例显示如何从 27 位 SPORT 寄存器中提取数据并将其转换为 8 位 UART 数据：

```
/* 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;
```

软件流程图

SPORT_A模块发送

SPORT_A 模块仿真 UART Tx 端口。要使用 SPORT_A 模块仿真 UART Tx 端口，必须对 SPORT 模块进行发送初始化并将其使能。使能后，如果有待处理数据要发送，则检查 SPORT_STAT 寄存器。如果有数据要发送，则根据待处理数据创建 UART 数据包，然后将该数据包写入 SPORT 发送寄存器。

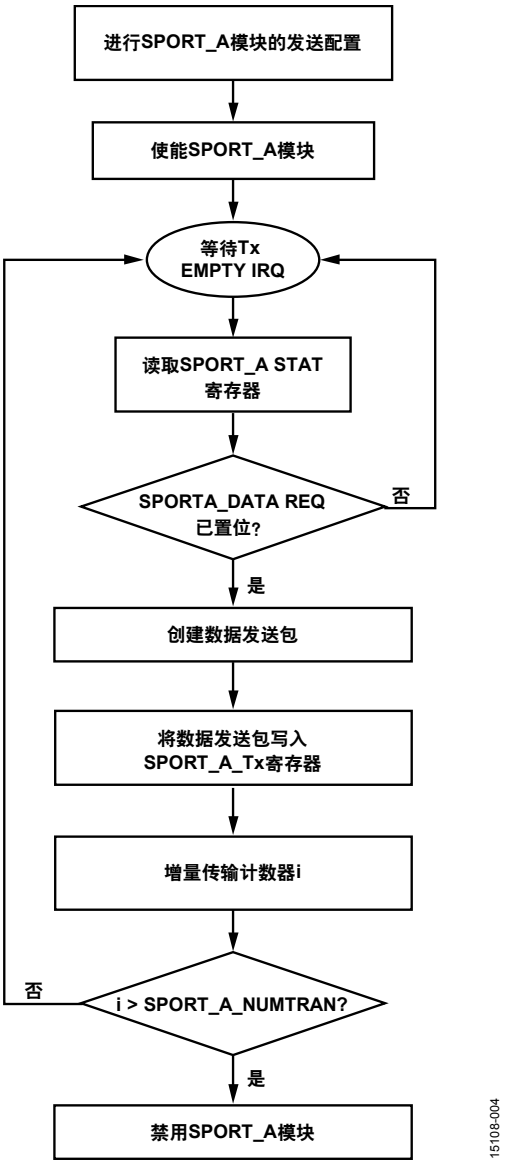

图4. SPORT_A模块发送流程图

SPORT_B模块接收

SPORT_B模块仿真UART Rx端口。要将SPORT_B用作UART Rx端口，必须首先对SPORT_B模块进行接收初始化并将其使能。SPORT 使能后，在 SPORT_B 数据寄存器中检查 SPORT_STAT 寄存器是否存在待处理数据。如果存在数据，则检索该数据并提取 8 位 UART 数据。

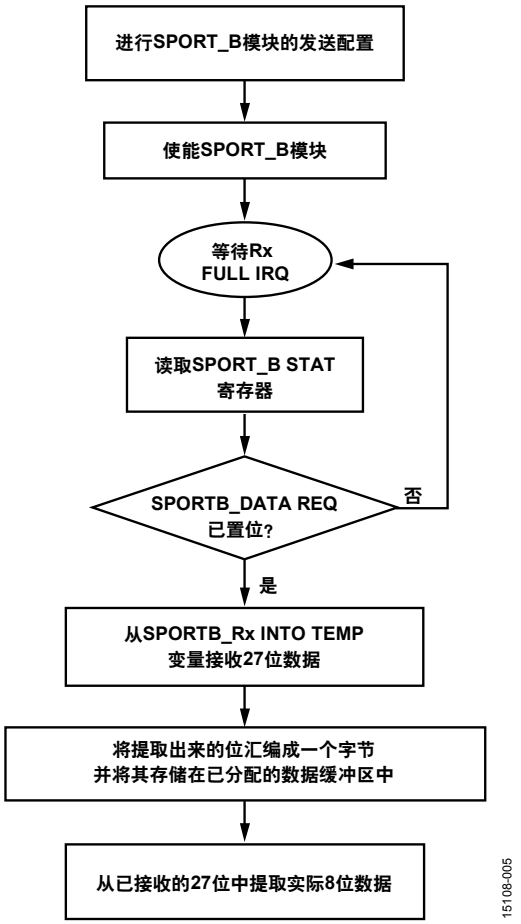

图5. SPORT_B模块接收流程图

波形

图 6 显示 SPORT_A 模块的发送波形和 UART 器件的接收波形（比特率：9600 bps，单帧 8 位数据(0x96)）以及 UART 发送仿真所需的其他格式化位。

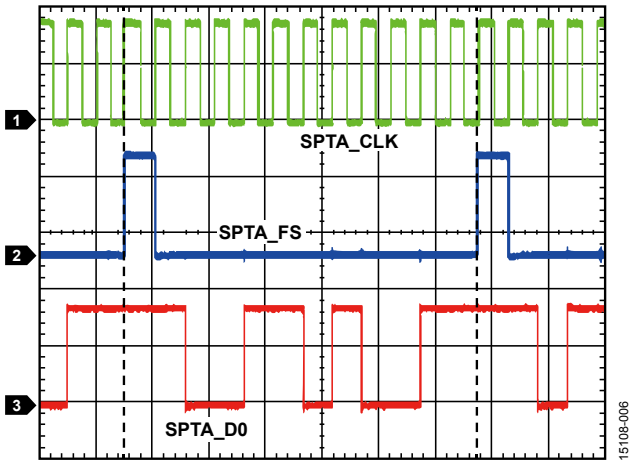

图6. SPORT_A模块发送和UART器件接收（单帧）

图 7 显示 UART 器件的发送波形和 SPORT_B 模块的接收波形（比特率：9600 bps，单帧 8 位数据(0x96)，带一个起始位和一个停止位，采样速率是发送波特率的 3 倍，可正确进行 UART 接收仿真）。

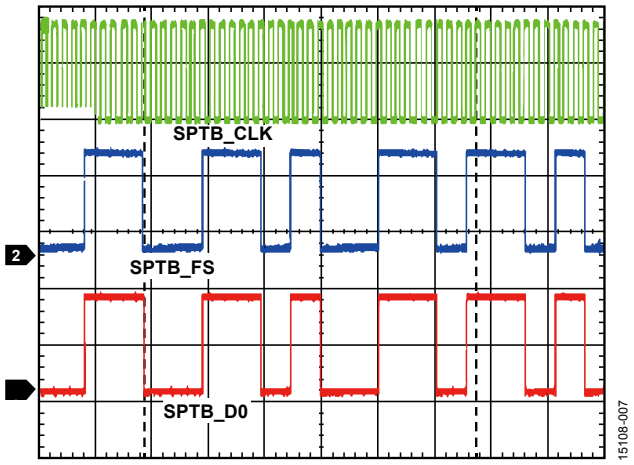

图7. UART器件发送和SPORT_B模块接收（单帧）

这些图中的红色走线表示单帧传输。

SPORT_UART_EMULATOR专用代码

本节中所示的代码为典型代码，适用于下列情形：

- SPORT_A 模块发送和 UART 接收
- UART 发送和 SPORT_B 模块接收

单帧传输使用的数据格式为 8-N-1（1 个起始位、8 位数据、0 奇偶校验位和 1 个停止位）采用 PCLK = 26 MHz 和多个波特率对这些情形进行了测试。

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) {}
      }
```

结语

本应用笔记介绍如何使用ADuCM3029/ADuCM3027处理器的SPORT通信协议来仿真全双工UART通信，以便与任何标准的UART器件实现接口。

本应用笔记所述用例在内核模式和直接存储器访问(DMA)模式下针对所有标准波特率进行了测试。根据可靠的测试结果，SPORT发送周期的波特率高达115,200 bps，SPORT接收周期的波特率高达57,600 bps。经过测试，两个方向上传输5到8位数据大小可用于正确运算。