Technical notes on using Analog Devices products and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Implementing UART Using ADuCM302x Serial Ports

Contributed by Sachin Dwivedi

Rev 1 – May 16, 2016

Introduction

Using the synchronous serial ports (SPORTs) on the ADuCM302x processor, it is possible to implement a full-duplex asynchronous port to communicate with the UARTs with minimal software overhead. This EE-note describes how to implement a full UART interface for multiple standard baud rates.

Asynchronous Communication

The difference between synchronous and asynchronous serial communication is the presence of a clock signal and frame sync signal. A synchronous serial port has a clock signal and an optional frame sync signal. An asynchronous port does not have clock and frame sync signals. In the absence of a clock signal, the asynchronous ports must communicate at a predetermined data rate (bit rate). In the absence of a frame sync, the word framing information is embedded in the data stream. A start-bit marks the beginning of a transmission. A stop-bit marks the completion of a transmission. The word length is predetermined between the receiver and transmitter.

Asynchronous SPORT Transmitter

The transmit side of the serial port must be configured for internal clock generation with a clock rate equal to the desired bit rate of the UART. This is done by setting the CLKDIV bit in the clock divider register (SPORT_DIV_A) for the SPORT_A block.

$$SPORT\_DIV\_A.CLKDIV = \frac{PCLK}{2 * Baud\ Rate} - 1$$

This clock is only used to synchronize the serial port to the desired bit rate. The actual clock signal (SPORT_ACLK) does not connect to anything. Configure the Frame Sync signal (SPORT_AFS) to be internally generated and leave the signal floating. The SPORT_A block must always transmit LSB first to emulate UART transmission. Program the number of bits to be transmitted by the SPORT_A block in the SLEN field of the SPORT_CTL_A register. Program the total number of words to be transferred in the SPORT_NUMTRAN_A register, with each word size decided by the SLEN field.

In case of SPORT transmission, where the SPORT_A block transmits to a UART device, the UART always receives the first transfer as 0x00, which can be discarded, followed by a correct sequence of data transmitted by the SPORT_A block. This is because at the start of transfer (after configuration), the UART Rx line is idle-high (Logic 1) and SPORT data line is idle-low (Logic 0). The UART interprets this Logic 0 as a start-bit and receives an entire frame of Logic 0 at the beginning.

Asynchronous SPORT Receiver

The serial port must determine where a new transmission begins without the presence of an internal frame sync signal. The transmit pin of the UART device connects to the Data line pin (SPORT_BD0) and Frame Sync pin (SPORT_BFS) on the SPORT_B of the ADuCM302x processor. The SPORT_B block is configured for internal clock generation and active low external frame sync signal. As the SPORT cannot guarantee any phase synchronization with the incoming bit stream, it is necessary to oversample the incoming asynchronous data stream. The receive clock on the SPORT must be set to three times the desired baud rate. For example, if the ADuCM302x SPORT communicates with the UART device at 9,600 bps, the receive clock on the SPORT must be set to 28,800 bps. This is done by calculating the appropriate divisor and programming the CLKDIV bit in the clock divider register (SPORT_DIV_B) for the SPORT_B block.

$$SPORT\_DIV\_B.CLKDIV = \frac{PCLK}{2 * 3 * Baud\ Rate} - 1$$

The active low Frame Sync signal (SPORT_BFS) is polled on the active edge of the internally generated clock (SPORT_BCLK). When the SPORT_BFS signal is asserted due to the low-level start bits of the UART packet, the SPORT_B block starts receiving the word transmitted from the UART device, and does not check the SPORT_BFS line until all N bits of the packet are received (N is programmed by the SLEN field in the SPORT_CTL_B register). SPORT uses the oversampled start-bit as a frame sync to kick off the reception of the incoming asynchronous data stream.

Hardware Overview

Figure 1 shows the connection between the ADuCM302x SPORTs and TX (transmit) and RX (receive) pins of a basic UART port on another device.

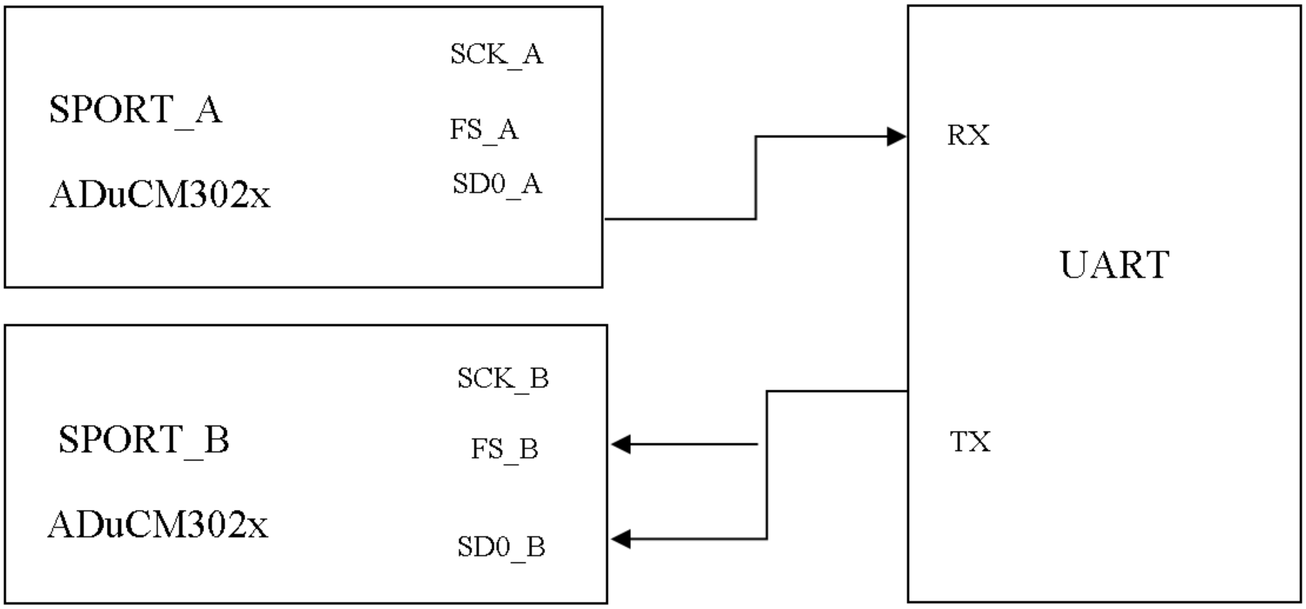


Figure 1. ADuCM302x MCU to UART Interface

Software Overview

The software required to manage the asynchronous data moving in and out of the SPORT is minimal. The C functions for the SPORT transmission and reception are detailed in the Code for SPORT_UART_Emulator section. The code has been tested for multiple baud rates and multiple number of transfers between the UART Host and ADuCM302x SPORTs for both transfer directions.

Asynchronous SPORT Transmitter (SPORT_A Block)

On the transmit side, the N-bit data to be transmitted must be formatted into a UART transmission packet. A start-bit and stop-bit must be added to the word for correct reception by the UART device.

An example of the data format is shown below.

For an 8-N-1 transmission format (8-bit data + 0 parity bit + 1 stop-bit) and data = 0xAA (b#1010 1010)

Modified Data = b# 1 10101010 0 1 (1 stop-bit + 8-bit data + 1 start-bit + 1 stop-bit)

A stop-bit must be appended at the beginning as the SPORT_AD0 line retains the value of the LSB (if LSB is transmitted first) when a complete word is sent. The UART Rx line must be set to idle high to avoid glitches in the generated signal between consecutive bytes, leading to corrupt data.

Asynchronous SPORT Receiver (SPORT_B Block)

The receive side is more complex than transmit side, as the SPORT_B block receives an oversampled data. For an 8-N-1 transfer format, as the data is over-sampled by a factor of 3, the serial port must be programmed to receive 27 bits, thereby discarding the 3 sampled start-bits, which is accounted for in the frame sync (SPORT_BFS). The 27 bits received represent the packet transmitted by the UART device, 8 bits of data, and 1 stop-bit (oversampled by a factor of 3).

The actual data is then extracted from the oversampled data by bit manipulation operations. The middle bit, which is considered to be the correct value, is extracted from the 3-bit sequence in the received data, for every transmitted bit from the UART device. The extracted bits are assembled together to form a byte of data.

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 zeros	Byte represented by 24 bits								3 ones

Figure 2. Expected Data Formats for UART Frame and SPORT Receive Frame

Driver Function Prototypes

The following functions are designed to work on an 8-bit asynchronous data, but can easily be changed to support other data widths. The C functions for the use case are detailed in the Code for SPORT_UART_Emulator section.

```
void   SPORT_UART_Tx_Initialise ( )
```

The function is used to configure and setup the SPORT_A block on the ADuCM302x processor for UART Transmission Emulation. The SPORT_A internal Clock is derived from the PCLK, which is configured to 6.5 MHz (default). The desired baud rate is set for the transmission, along with configuration using the SPORT_CTL_A register. Interrupt for Transmit Data buffer empty is configured using the SPORT_IEN_A register. The Number of words to be transferred are programmed using the SPORT_NUMTRAN_A register, before enabling the SPORT_A block.

```
void   SPORT_UART_Rx_Initialise ( )
```

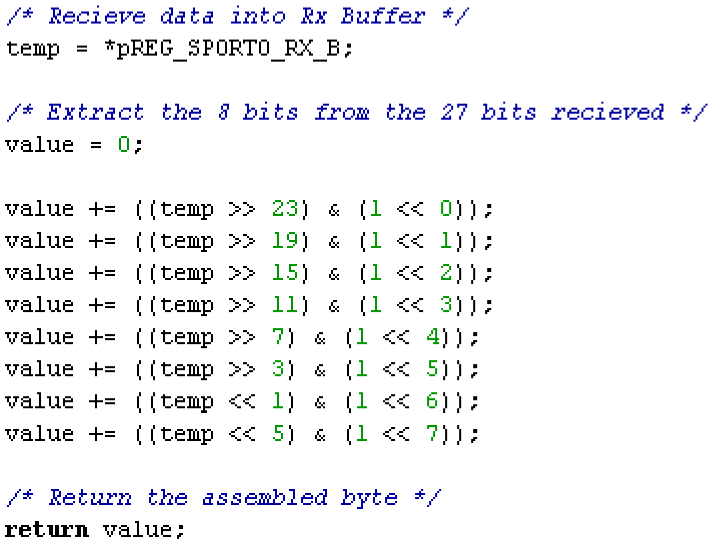
The function is used to configure and setup the SPORT_B block on the ADuCM302x processor for UART Reception Emulation. The SPORT_B block is configured to oversample the incoming data stream by a factor of 3. The Frame sync is configured for external low active nature. Interrupt for Receive Data buffer full is configured using the SPORT_IEN_B register, and SLEN field of the SPORT_CTL_B register is configured to (3 * (word size + No.of stop bit) – 1), before enabling the SPORT_B block.

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

The function creates the UART transmission data format by modifying the data in location pointed to by buf. The modified data is then put into the SPORT_A_TX register for transmission. The function uses bit masking and shifting operations.

```
uint8_t SPORT_UART_Rx_Transfer ( )
```

The function receives the oversampled data from the SPORT_B_RX register. A bit manipulation operation is used to extract the middle bits of every 3-bit sequence of the SPORT_B_RX data (3 bits received for every 1 bit transmitted by the UART device). The extracted bits are assembled into a byte sized data. The function returns the assembled byte, representing the actual received data.



```
/* Recieve data into Rx Buffer */
temp = *pREG_SPORT0_RX_B;

/* Extract the 8 bits from the 27 bits recieved */
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;
```

Software Flow Diagrams

SPORT_A Block Transmission

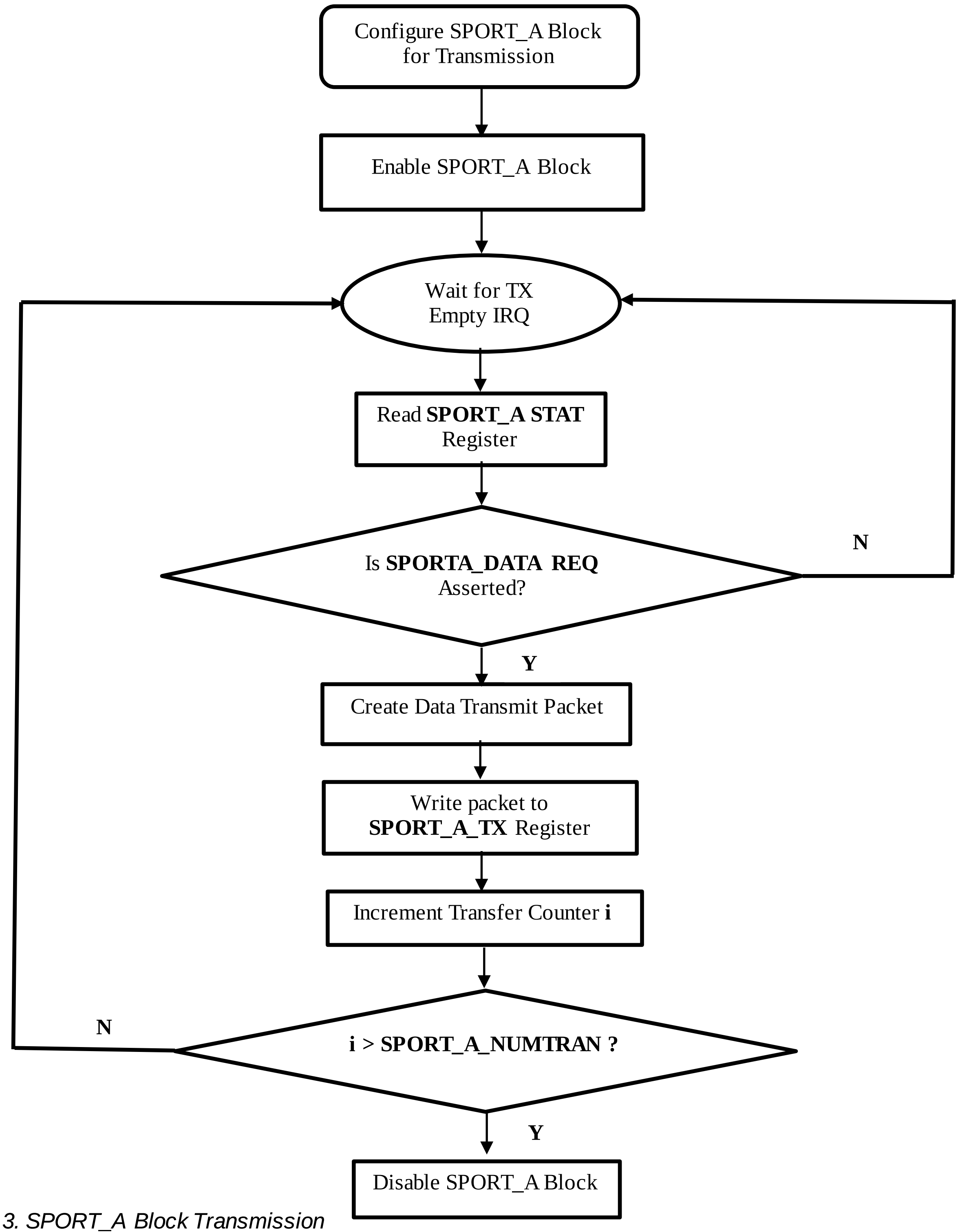

Configure SPORT_A Block for Transmission
Enable SPORT_A Block
Wait for TX Empty IRQ
Read **SPORT_A STAT** Register
Is **SPORTA_DATA REQ** Asserted?
N
Y
Create Data Transmit Packet
Write packet to **SPORT_A_TX** Register
Increment Transfer Counter **i**
i > SPORT_A_NUMTRAN ?
N
Y
Disable SPORT_A Block

Figure 3. SPORT_A Block Transmission

SPORT_B Block Reception

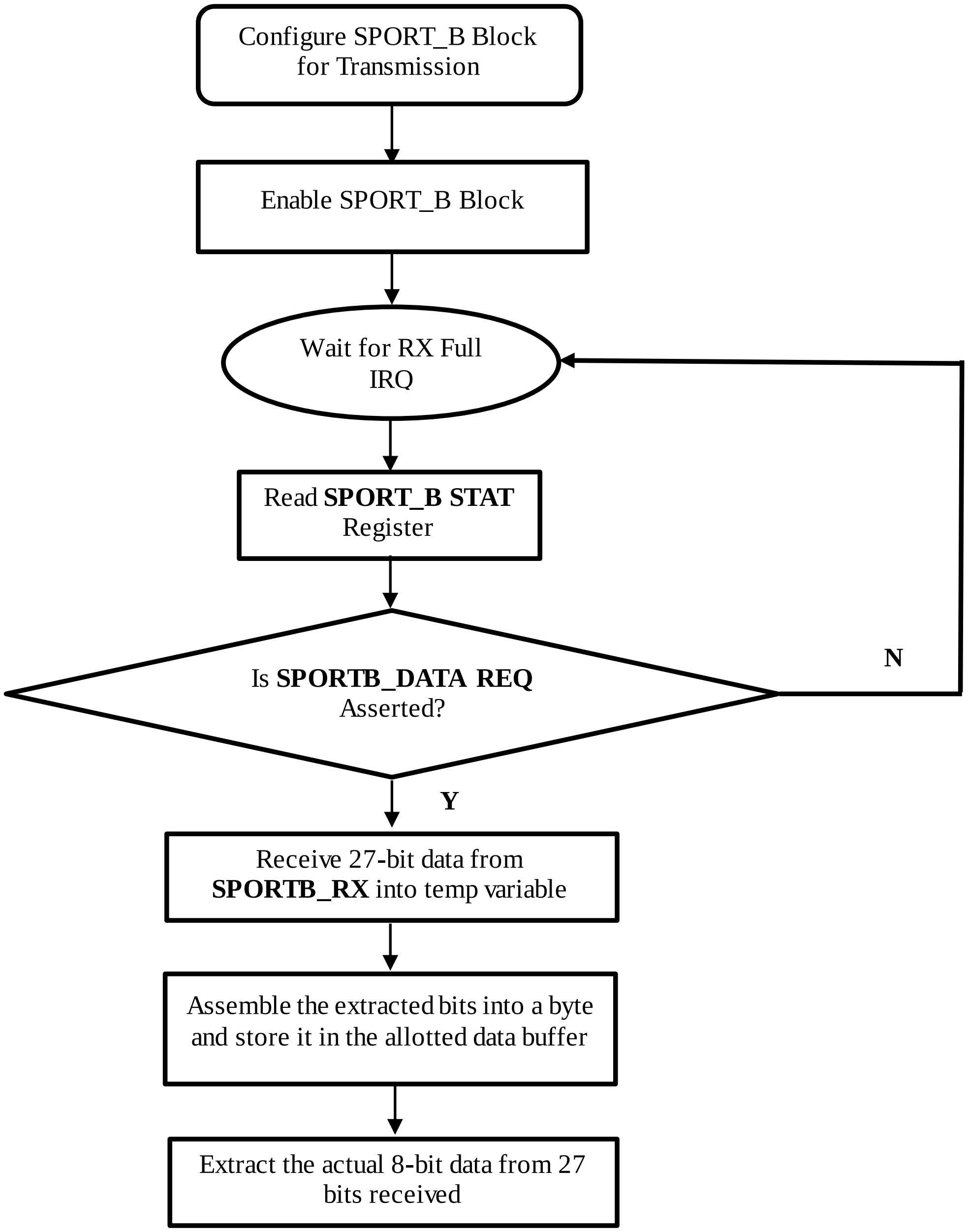

Configure SPORT_B Block for Transmission
Enable SPORT_B Block
Wait for RX Full IRQ
Read SPORT_B STAT Register
Is SPORTB_DATA REQ Asserted?
N
Y
Receive 27-bit data from SPORTB_RX into temp variable
Assemble the extracted bits into a byte and store it in the allotted data buffer
Extract the actual 8-bit data from 27 bits received

Figure 4. SPORT_B Block Reception

Waveforms

Figure 5 shows the waveforms for transmission from the SPORT_A block and reception on the UART device at 9600 bps, for a single frame of 8-bit data (0x96) and additional formatting bits required for UART transmission emulation.

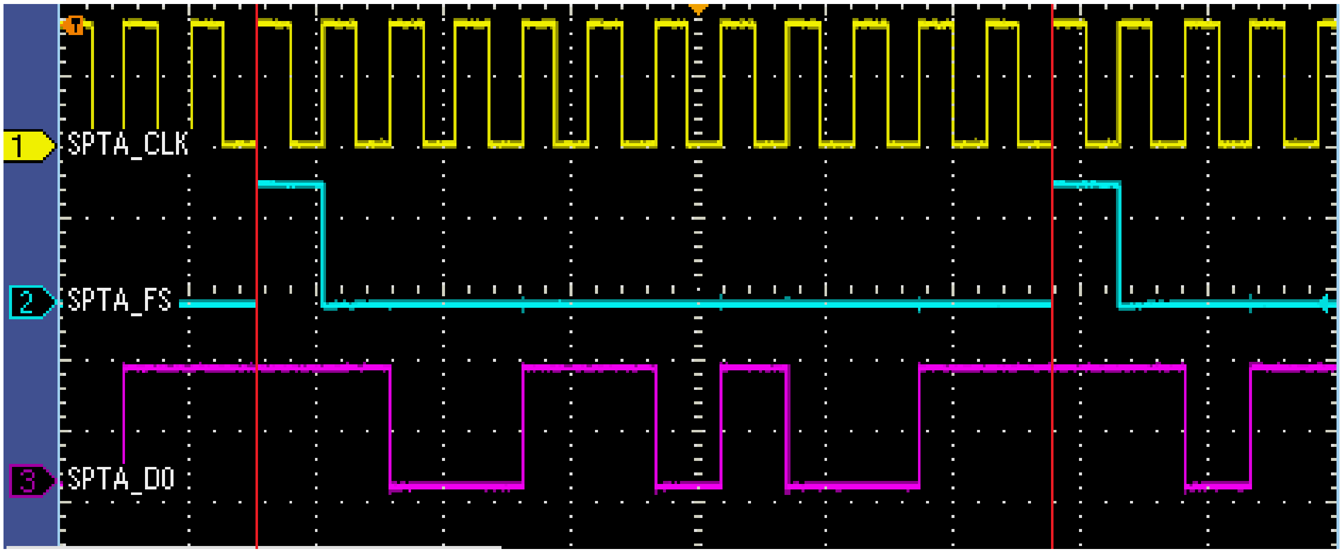


Figure 5. SPORT_A Block Transmission and UART Device Reception for Single Frame

Figure 6 shows the waveforms for transmission from the UART device and reception on the SPORT_B block at 9600 bps, for a single frame of 8 bit data (0x96) (with a start-bit and a stop-bit), sampled at 3 times the transmission baud rate for proper emulation of UART reception.

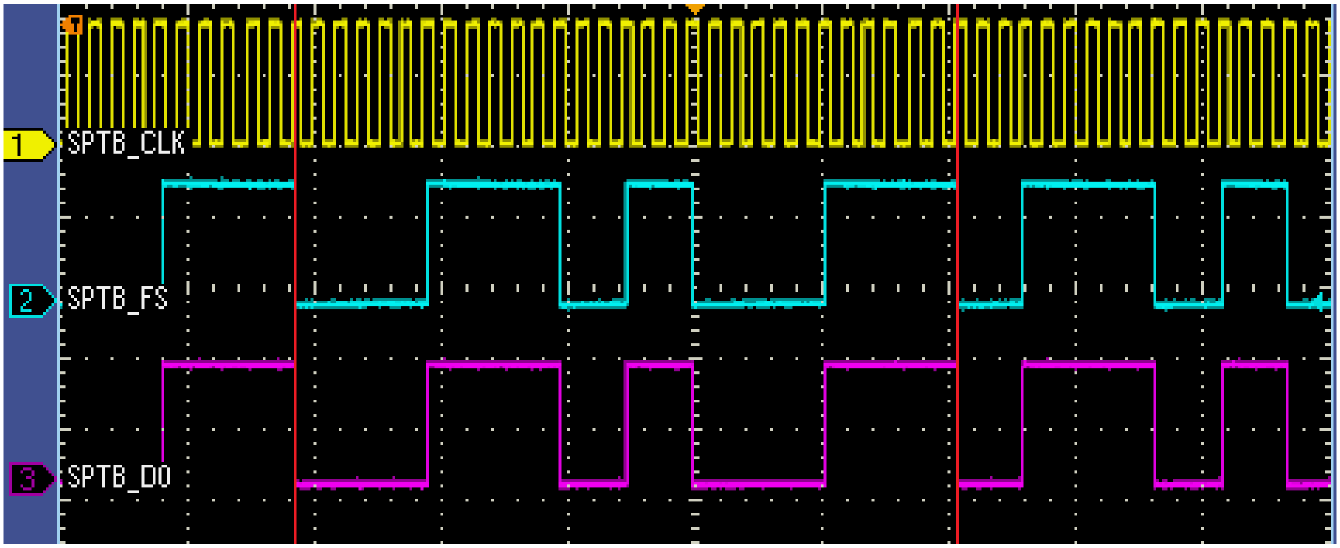


Figure 6. UART Device Transmission and SPORT_B Block Reception for Single Frame

The red cursor lines in the figures indicate a single frame of transfer.

Code for SPORT_UART_Emulator

The code listing provides an example for the following use cases:

(a) Transmission from the SPORT_A block and Reception by UART

(b) Transmission from UART and Reception by the SPORT_B block

The data format used for a single frame of transfer is 8-N-1 (1start-bit, 8 bits of data, 0 parity bits and 1 stop-bit). These cases have been tested at PCLK = 26 MHz and multiple baud rates.

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 recieve 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;
  /* Recieve data into Rx Buffer */
  temp = *pREG_SPORT0_RX_B;

  /* Extract the 8 bits from the 27 bits recieved */
  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)
  {

  }

}
```

Conclusion

The EE-Note describes how to use the SPORT communication protocol on the ADuCM302x processor to emulate a full-duplex UART communication, which can be then used to interface with any standard UART device.

The use case presented has been extensively tested in Core and DMA modes for all standard baud rates. Reliable results have been observed for baud rates up to 115200 bps on SPORT transmission cycle and up to 57600 bps on SPORT reception cycle. Data size ranging from 5– 8 bits for transfers in both directions has been tested for proper operation.

References

[1] *ADuCM302x Mixed-Signal Control Processor with ARM Cortex-M3 and Low-Power Management Hardware Reference Manual*. Revision 0.3, November 2015. Analog Devices, Inc.

[2] *ADuCM3027/ADuCM3029 Ultra Low-Power ARM Cortex-M3 MCU with Integrated Power Management Datasheet.* Rev PrF, February 2016. Analog Devices, Inc.

[3] *Implementing a Glueless UART Using The SHARC® DSP SPORT (EE-191).* Rev 1, May 2003. Analog Devices, Inc.

Document History

Revision	**Description**
Rev 1 – May 16, 2016 *by Sachin Dwivedi*	Initial Release