AN11351

Implementing a UART using SGPIO on LPC4300

Rev. 1 — 15 April 2013 Application note

Document information

Info	Content
Keywords	LPC4300, SGPIO, UART
Abstract	This application note describes how to implement UART transmission and reception function via SGPIO on NXP's LPC43xx. The UART implemented in this application note operates in half and full duplex mode with 8 data bits, no parity, 1 stop bit, no flow control. The supported baud rates are 9600bps/19200bps/38400bps/57600bps/115200bps

Revision history

Rev	Date	Description
1	20130415	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

NXP's LPC43xx ARM Cortex-M4 based microcontrollers for embedded applications include an ARM Cortex-M0 coprocessor, up to 1 MB of flash, up to 264 kB of SRAM, and advanced configurable peripherals such as the Serial General Purpose I/O (SGPIO) interface. The LPC43xx devices operate at CPU frequencies of up to 204 MHz.

This application note describes how to implement UART transmission and reception functions (hereafter called TX and RX) via SGPIO on the LPC43xx with UART operations in half and full duplex mode. Following sections are included in this application note:

- SGPIO peripheral introduction
- Development and test environment
- Driver implementation
- Demonstration and test result

The UART protocol implemented is: 8 data bits, no parity, 1 stop bit, no flow control. The supported baud rates include 9600bps, 19200bps, 38400bps, 57600bps and 115200bps.

The sample software is tested on Keil's MCB4300 evaluation board with UART0; TX and RX pins can be configured via SGPIO output and input pins respectively.

2. SGPIO peripheral introduction

SGPIO is a new peripheral available on the LPC43XX devices from NXP.

SGPIO offers standard GPIO functionality enhanced with features to accelerate serial data stream processing. The enhanced features are realized with slices.

There are 16 SGPIO slices and 16 SGPIO IO pins. A slice is a section of hardware that handles the data processing when sending or receiving serial data.

A slice consists of a 32-bit FIFO that is used to clock data in or out, a data shadow register, and a clock divider to generate a clock for the slice. Slices also have several interrupt capabilities. Please refer to user manual UM10503 for details on the interrupt functionality of a slice. The basic operation of one slice is shown in Fig 1.

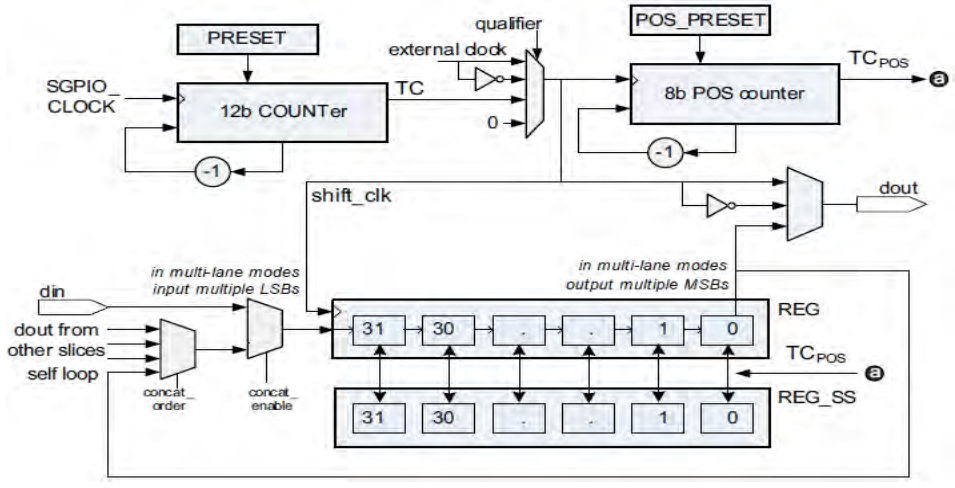


Fig 1. Basic operation of one slice

3. Development environment

3.1 Hardware

This project uses a Keil MCB4300 evaluation board with UART for quick hardware set up and software development.

The TX/RX pins (P2_0/P2_1) of UART0 on the MCB4300 board are configured as an SGPIO function. Jumpers J16 and J13 should be selected for UART0.

A UART cable connects UART0 on MCB4300 to the COM port on a PC.

Keil ULINK2 is the debugger used in this development.

The connection diagram of the target board is shown in Fig 2.

Fig 2. Connection diagram

3.2 Software

Keil uVision4 (V4.5) is used for the development IDE.

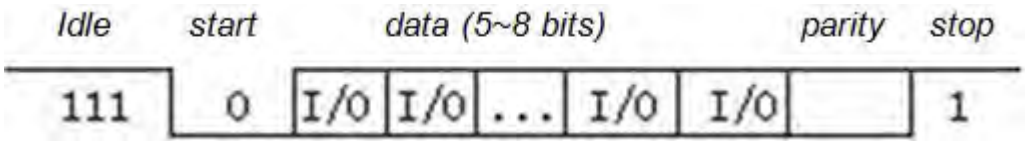
Putty is the recommended serial communication software on the PC, however, any other terminal program will work as well.

4. Driver implementation

The SGPIO peripheral emulates the UART protocol to send and receive data via SGPIO. It includes:

- SGPIO clock configuration for providing the required baud rate for UART communications
- SGPIO pin initialization and slice configuration for UART TX and RX function
- Emulate sending and receiving of data frames conforming to UART protocol via SGPIO
- Implement sending and receiving multiple bytes of data in UART half and full duplex mode

A waveform showing the UART protocol is shown in Fig 3:



The implemented protocol is: start bit + data (8 bits) + no parity + stop bit

Fig 3. UART waveform

The SGPIO slice and pin mapping for the UART emulation is shown in Fig 4.

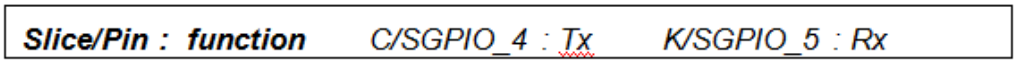


Fig 4. UART slice and pin mapping

The following steps describe how to implement the driver.

The driver implementation is contained in files *sgpio_uart.c* and *sgpio_uart.h*.

4.1 SGPIO clock configuration

To ensure optimum performance, the SGPIO peripheral input clock (SGPIO_CLK) should be the same as system clock. It is configured to generate the shift clock and swap clock for serial data communication.

The shift clock frequency is equal to UART baud rate which samples once for each bit; the swap clock is designed as the cycle time of a frame.

Table 1. SGPIO clock control setting

Register	Slice C	Slice K	Function description
SLICE_MUX_CFG	0<<2 | 0<<6	0<<2 | 0<<6	(1) Use clock internally generated by COUNTER (2) Shift 1 bit per clock
SGPIO_MUX_CFG	0<<5	0<<5	Qualifier enable for clock

For additional details please refer to function *SGPIO_UART_setclk()* in sgpio_uart.c file.

4.2 SGPIO initialization

SGPIO needs to be initialized for UART TX and RX to function. The pin for TX should be configured as an SGPIO output slice and all of the data bits in data and shadow registers should be “1” to set the UART to idle mode (see Table 2).

Table 2. SGPIO TX initialization setting

Register	Slice C	Function description
OUT_MUX_CFG	0 | 0<<4	(1) 1-bit mode (2) state set by GPIO_OEREG
GPIO_OEREG	1<<2	Enable output for TX
REG/ REG_SS	0xFFFFFFFF	Keep idle state before sending valid data

For RX, besides setup of input mode, it should be ready to match the start bit of UART RX. The setting of SGPIO is shown in Table 3:

Table 3. SGPIO RX initialization setting

Register	Slice K	Function description
GPIO_OEREG	0<<2	Enable input for RX
SLICE_MUX_CFG	1<<0	Match data mode
REG	0xFFFFFFFF	To match the data in REG_SS
REG_SS	0x1FFFFFFF	To sample 3 times for start bit

Refer to the related source code for more details regarding initialization.

4.3 Send data frame

According to the UART protocol (1 start bit, 8 data bits, no parity, 1 stop bit), emulation of sending the serial data frame is shown in Fig 5:

```
tmp = 0x00 | (Data<<1) | (0x01<<9);//10b = start bit(0) + data + stop bit 1
LPC_SGPIO->REG_SS[SGPIO_slice] = tmp;
```

Fig 5. Send data frame

The data frame will be swapped with the shadow register when the previous frame has been shifted out completely.

4.4 Receive data frame

Reception of the start bit uses the pattern match functionality of a slice. When the start bit is received, the SGPIO is configured to capture the serial data that follows. First, the pattern match is switched off to capture data, then the swap clock is started and a capture interrupt is enabled for frame data reception. For implementation details please refer to the function SGPIO_UART_Rx_Capture().

After all bits of a data frame are captured via the SGPIO interface, the 8-bit data is extracted by software:

```
tmp = LPC_SGPIO->REG_SS[SGPIO_slice];
shift_bit = (FrameLen-1)*OVERSAMPLING_NUM;
tmp = tmp>>(32-shift_bit);
```

Fig 6. Extract 8 bit data

4.4.1 Oversampling

To improve the reliability of received data, each bit is sampled 3 times. The oversampling clock should be configured as described in Section 4.1. When using oversampling for the RX signal, extra lines of code should be added to restore each data bit before returning the received data. The way to restore a bit is to compare the three sampled values to each other and then choose the one that occurs two or more times. For more detail, please refer to the function *SGPIO_UART_ReceiveByte()* in sgpio_uart.c.

The OVER_SAMPLING and OVERSAMPLING_NUM for oversampling are defined in *sgpio_uart.c*.

```
#define OVER_SAMPLING   1
#if OVER_SAMPLING
  #define OVERSAMPLING_NUM  3 //sample 3 times for one bit
#else
  #define OVERSAMPLING_NUM  1
#endif
```

Fig 7. Define oversampling function

4.5 Driver APIs

When a user wants to implement an application program using the drivers directly, the following driver API functions can be called.

Table 4. API introductions

Prototype	Input	Return	Description
SGPIO_UART_setclk	**SGPIO_Txslice:** SGPIO slice for UART TX **SGPIO_Rxslice:** SGPIO slice for UART RX **UART_ConfigStruct :** Pointer to a UART_CFG_Type structure that contains the configuration information for UART	SUCCESS or ERROR	Set the SGPIO UART clock according to the specified parameters in the UART_ConfigStruct.
SGPIO_UART_Tx_Init	**SGPIO_TxPin:** SGPIO pin for UART TX **SGPIO_slice:** SGPIO slice for UART TX	None	Initialize the SGPIO pin as UART TX
SGPIO_UART_Rx_Init	**SGPIO_RxPin:** SGPIO pin for UART RX **SGPIO_slice:** SGPIO slice for UART RX	None	Initialize the SGPIO pin as UART RX
SGPIO_UART_Setmode	**SGPIO_slice:** SGPIO slice for UART TX/RX **NewState:** New State of transfer	None	Start/Stop transfer (TX/RX) on SPGIO slice

Prototype	Input	Return	Description
	function, should be: - ENABLE: start TX/RX - DISABLE: stop TX/RX		
SGPIO_UART_Enter_SendIdle	**SGPIO_slice:** SGPIO slice for UART TX	None	Enter idle state in TX
SGPIO_UART_SendByte	**SGPIO_slice:** SGPIO slice for UART TX **Data:** Data to transmit (must be 8-bit long)	None	Transmit a single byte via SGPIO UART
SGPIO_UART_ReceiveByte	**SGPIO_slice:** SGPIO slice for UART RX	8 bit Data received	Receive a single byte via SGPIO UART

Generally, an application program will need to send and receive multiple bytes of data. This project provides example of sending and receiving multiple byte data in half duplex and full duplex mode in sgpio_uart.c.

5. Demonstration

5.1 Environment

This demonstration is tested on the Keil MCB4300 evaluation board with the UART configured as detailed above.

5.2 Demonstration

To demonstrate the UART communication via SGPIO, this example is designed to operate in both half duplex and full duplex mode.

In half duplex mode, ASCII code is received and sent back to the PC COM port one direction at a time. The simple functions to send and receive multi bytes of data are implemented in block mode. Please refer to SGPIO_UART_Send() and SGPIO_UART_Receive() in sgpio_uart.c.

In full duplex mode, ASCII code is received and sent back to the PC COM port simultaneously. The simple functions to send and receive multiple bytes of data are implemented in non-block mode. Please refer to SGPIO_UART_Send() and SGPIO_UART_Receive() in sgpio_uart.c.

In this application note, baud rates of 9600bps, 19200bps, 38400bps, 57600bps and 115200bps are supported configurations.

The project in the example can run in internal RAM and internal flash (for LPC43xx parts with on-chip flash).

5.3 Test result

Open Putty on the PC and configure the serial line to: 9600bps baud, 8 bits data, no parity, 1 stop bit, no flow control.

Connect the UART cable between UART0 on the board and COM on the PC after preparing the test environment described in Section 3. Connect the Keil ULINK2 between the target board and PC. Open the project *SGPIO_Uart.uvproj* and start ‘debug’ after building OK in internal RAM mode.

The following is printed on the screen when running the demo:

```
 Hello! NXP Semiconductors
 SGPIO UART demo in internal RAM
(MCU lpc43xx - ARM Cortex-M4)
<Press key>:
 [1]-> half duplex mode
 [2]-> full duplex mode
```

Fig 8. UART output via SGPIO

Press ‘1’ on the keyboard of PC to enter half duplex test mode, then press any number or letter keys, such as ‘1’,’2’,’3’,’4’,’a’,’b’,’c’,’d’,’e’,’f’,’g’ which then will be printed:

```
Waiting for Rx... Press any key
1234abcdefg
```

Fig 9. UART communication in half duplex mode

Press ESC to exit half duplex mode and wait for a new mode selection:

```
Waiting for Rx... Press any key
1234abcdefg
<Press key>:
 [1]-> half duplex mode
 [2]-> full duplex mode
```

Fig 10. Operation mode selection

Press ‘2’ to enter full duplex mode. It will keep sending serial data and will not exit until any key is pressed (should be letter or number key). Below is an example print out on the Putty terminal.

```
<Press key>:
 [1]-> half duplex mode
 [2]-> full duplex mode
Sending... Press any key to receive data while sending
Sending... Press any key to receive data while sending
Sending... Pres 'k' s any key to receive data while sending

<Press key>:
 [1]-> half duplex mode
 [2]-> full duplex mode
```

Fig 11. UART communicate in full duplex mode

The ‘k’ is received after pressing it at any time while continuously sending “Sending... Press any key to receive data while sending”, and the sending is continued to end a whole line. This indicates reception and transmission can be done at the same time.

5.4 Performance

As mentioned above, only the following UART protocol is supported in this application note:

Start bit + 8 data bits + no parity + stop bit + no flow control

The following baud rates can be supported: 9600bps, 19200bps, 38400bps, 57600bps and 115200bps.

Communication at 115200bps baud rate was tested in both half and full duplex mode using oversampling and the results were good.

6. Conclusion

This application note describes how to implement half and full duplex UART transmission and reception using the SGPIO peripheral on the NXP LPC43xx microcontroller.

Although the demonstration code is simple, it is a good reference solution to implement one or more UART channels via SGPIO in a user's application.

7. References

[1] *LPC43xx User Manual UM10503,* Rev. 1.6, NXP Semiconductors, 25 January 2013.

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 15 April 2013
Document identifier: AN11351