Using a 9-bit Software UART with Stellaris® Microcontrollers

ABSTRACT

Extend the functionality of the standard hardware UART available on Stellaris® microcontrollers by using the 9-bit UART add-on. This practical software solution provides a way to distinguish between an address or data character. The code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPMA032.

Contents

1 Introduction

The Universal Asynchronous Receiver Transmitter (UART) is a widely used serial communications peripheral used in many applications for connection to legacy devices and is present on all Stellaris® microcontrollers. Some systems require the additional functionality of a 9-bit UART, with automatic address detection. The 9-bit software UART uses a ninth bit to differentiate between a data or address character. The auto address detection allows the user to program a specific address that is used to determine if the received data is supposed to be processed or discarded. The 9-bit UART was written using the standard UART DriverLib API. The 9-bit UART uses the same function structure as the standard UART, but adds the NB_ prefix specifically for 9-bit UART calls. This application note describes the 9-bit UART software add-on in detail.

2 9-Bit UART Initialization

There are five initialization steps for the 9-bit UART:

1. Enable UART and GPIO peripherals.
2. Configure the GPIO pins for UART function.
3. Set up the UART hardware.
4. Enable the UART Rx interrupt.
5. Customize the user programmable address.

2.1 GPIO

The GPIO setup is not included in the 9-bit UART API functions and must be done explicitly by the user. The following steps are necessary to enable the UART and GPIO peripherals:

1. Enable the appropriate GPIO port for the UART0 or UART1 Tx and Rx pins.
2. Configure the UART0 or UART1 GPIO pins for UART operation.
   a. Configure the GPIO pins for standard push-pull operation.
   b. Set the pins to be peripheral controlled.
3. If your part has pin muxing, set the pin mux for your device.

Note: See the corresponding microcontroller data sheet for your device to find the correct UART GPIO port and pins.

2.2 UART

The 9-bit UART API supports the use of only one UART port. You can use either UART0 or UART1, but not both ports.

Note: For 9-bit UART operation, you must only use the DriverLib function calls that have the designated NB_ prefix.

There are two functions used to configure and initialize the 9-bit UART:

- `NB_UARTConfigSetExpClk()`

  Sets up baud-rate, number of data-bits, and number of stop-bits.

- `NB_UARTEnable()`

  Enables the UART hardware and the UART receive interrupt.

2.3 UART Interrupt

For proper operation of the 9-bit UART, enable the UART Rx interrupt using the `NB_UARTEnable()` function which enables the UART hardware interrupts on the nested vector interrupt controller (NVIC), and also enables the receive interrupt on the UART hardware. You must enable the processor interrupts and explicitly add the “NB_UARTIntHandler” interrupt handler to the interrupt vector table.

The UART interrupt can be configured to call a user function. If the `NB_USER_INT_HANDLER` macro is defined, then the `NB_UserIntHandler()` function will be called in the UART interrupt handler.

Note: The `NB_UserIntHandler()` is called in an interrupt context. Do not do time-consuming operations in this function.

2.4 Programmable Address

Use the following functions to set the 9-bit UART's single programmable address:

- `NB_UARTAddressSet()`

  Sets the address for the UART hardware.

- `NB_UARTAddressGet()`

  Returns the current address of the UART hardware.

This address is used by the UART to decide when to acknowledge or discard data. You must properly set the device address prior to using the 9-bit UART. Unlike some hardware implementations of the 9-bit UART, the character used as the address is allowed to be received as a data. For details on receiving a character, see the “Receive” section for more information.

3 9-Bit UART Function

The 9-bit UART includes two sets of transmit functions; one set of functions is used to transmit data and the other set is to transmit an address. The received data is handled by the receive ISR and a software buffer. For 9-bit UART operation, you must only use the DriverLib function calls that have the designated NB_ prefix.

Note: See the API Functional Description in the accompanying source code package for this application note for more information on these functions.

3.1 Transmit

The 9-bit UART API includes the following functions for sending addresses and data:

- `NB_UARTBusy()`

Checks if the UART is busy sending or receiving another character.

- `NB_UARTAddrPut()`

  Puts an address character in the transmit register if there is space available. If there is no space available, it blocks the write of the address character until space is available.

- `NB_UARTAddrPutNonBlocking()`

  Puts an address character in the transmit register if there is space available. If there is no space available, it returns an error code.

- `NB_UARTDataPut()`

  Puts a data character in the transmit register if there is space available. If there is no space available, it blocks the write of the data character until space is available.

- `NB_UARTDataPutNonBlocking()`

  Puts a data character in the transmit register if there is space available. If there is no space available, it returns an error code.

Note: The 9-bit UART does not use a transmit FIFO. A single hardware register is used to transmit the data out of the UART.

The parity setting is used to differentiate between the send data and send address functions. The send data function sets the parity to Stick Zero to make the ninth bit of the UART transfer a 0 and the send address function sets the parity to Stick One to make the ninth bit of the UART transfer a 1. After setting the parity, the 9-bit UART functions use standard DriverLib function calls to send the character through the UART hardware.

The logic diagrams below show how the data and address transfers are performed. Figure 1 shows an address transmission (set parity bit to a 1) and Figure 2 shows a data transmission (parity bit is a 0). These diagrams have the UART configured with one stop bit.

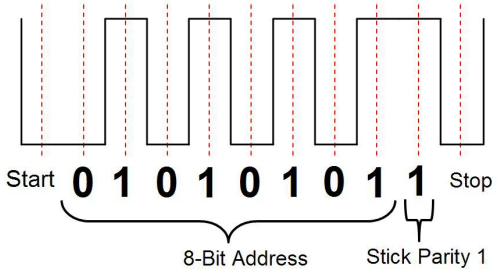

Figure 1. Sample Address Transfer

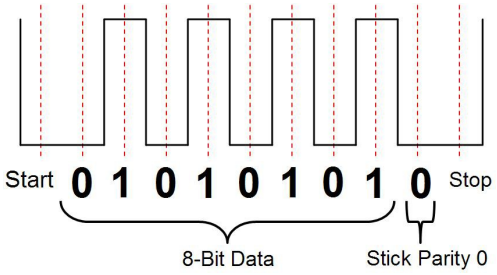

Figure 2. Sample Data Transfer

3.2 Receive

The following 9-bit UART API functions are used for receiving characters:

- `NB_UARTDataAvail()`

  Checks if there is any data in the software receive FIFO.

- `NB_UARTDataGet()`

  Returns a character if there is one available in the software receive buffer. If there is no character available, it blocks the read of the address character until space is available.

- `NB_UARTDataGetNonBlocking()`

  Returns a character if there is one available in the software receive buffer. If there is no character available, it returns an error code.

To properly receive UART characters in 9-bit UART mode, an interrupt must be generated every time a character is received so that the software can determine if the character received was data or an address. The interrupt handler checks the current state of the parity polarity bit and the parity error bit. The parity polarity is set using the `EPS` bit in the **UART Line Control** register. If there was no parity error, then the parity bit received is the same as the current UART parity setting. If there was a parity error, then the bit received is the opposite of the current UART parity setting. The truth table in Table 1 summarizes the logic used for the 9-bit UART.

Note: The truth table shows the logic used to determine if data or an address was received. The EPS bit sets the Stick Parity setting. If EPS = 0, then the parity bit is 1; if EPS = 1, then the parity bit is 0.

Table 1. 9-bit UART Truth Table

EPS Bit	Parity Error	Address or $\overline{\texttt{Data}}$
0	0	1
0	1	0
1	0	0
1	1	1

The 9-bit UART code provided performs the following actions when a character is received:

1. Enter the UART interrupt once a character is received.
2. Read the character from the UART receive register.
3. Check to see if the character is an address.
4. If the character is an address, check if it matches the programmed UART address. If there is an address match, then set the address match flag to indicate that data should be added to the receive FIFO. If the

character is data, check if the address match flag is set. If the flag is set then add the data to the receive FIFO. Otherwise, discard the data.

1. Clear the interrupt request.
2. Return.

4.2 Receive Buffer Configuration

A software circular FIFO buffer is implemented for received characters. The default size of this buffer is 16 characters. The size is configured by a definition in the nb_uart.h file in the section labeled, “Receive buffer size configuration.” The #define is shown below. This value is user-selectable, and must be greater than or equal to 1 to ensure correct operation.

```
#define RX_BUFFER_SIZE 16
```

4 9-Bit UART Limitations

The software-based 9-bit UART add-on works on top of the DriverLib uart.c and uart.h files. It uses a specific configuration of the UART to function as a 9-bit UART. This includes disabling the Rx and Txs FIFOs, enabling the receive interrupt, enabling stick parity, and using 8-bit data.

5 Application Information

The 9-bit UART software triggers a UART receive interrupt every time a character is received. The maximum interrupt process time is 93 cycles (69 cycles to process the interrupt plus 24 cycles to enter and exit the interrupt). If valid data is received, the data will be available in the receive buffer prior to calling the `NB_UserIntHandler()` function or once the interrupt exits. This cycle time for the interrupt assumes that the `NB_UserIntHandler()` function is not being called.

To put a character into the UART Tx register and initiate a transfer takes a maximum of 69 cycles. This assumes that the transmit logic is not already transmitting.

Note: You cannot use the transmit or hardware receive FIFOs with the 9-bit UART software solution.

These cycle times were calculated using the Keil RealView® Microcontroller Development Kit (MDK) at the default code optimization level.

6 API Functional Description

The detailed API functional description is available in the source code package for this application note.

7 Conclusion

The software-based 9-bit UART add-on can be used on any Stellaris® microcontroller to extend the functionality of the standard hardware UARTs available to an application. When the ninth bit is required to distinguish between an address or data character, the software-based 9-bit UART add-on provides a practical software solution.

8 References

The following are available for download at www.ti.com/stellaris:

- Stellaris microcontroller data sheet, Publication Number DS-LM3Snnn or DS-LM3Snnnn (where nnn or nnnn is the part number for that specific Stellaris® family device)
- *Stellaris® Peripheral Driver Library User's Guide*, Document order number SW-DRL-UG
- Stellaris Peripheral Driver Library, Order number SW-DRL
- Source code for application note AN01280 - Using a 9-bit Software UART with Stellaris® Microcontrollers

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265