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.

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.

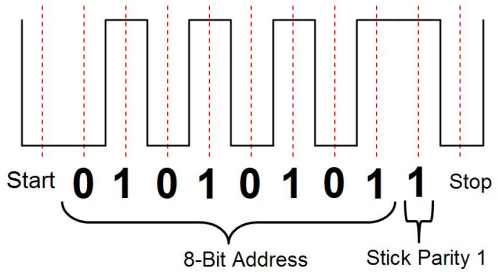

Figure 1. Sample Address Transfer

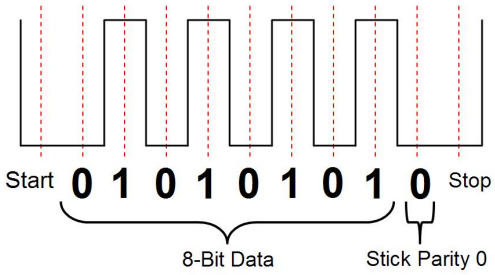

Figure 2. Sample Data Transfer

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.

5. Clear the interrupt request.
6. Return.

3.3 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

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

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