Drivers

Microchip Libraries for Applications (MLA)

Table of Contents

1 MLA Drivers 6

1.1 Legal Information 7

1.2 UART Driver 8

1.2.1 Using Driver 8

1.2.1.1 Abstraction Model 8

1.2.1.2 Initialization 9

1.2.1.3 Data Transfer 9

1.2.2 Configuring the Driver 10

1.2.2.1 DRV_UART1_CONFIG_8E1 Macro 11

1.2.2.2 DRV_UART1_CONFIG_8E2 Macro 11

1.2.2.3 DRV_UART1_CONFIG_8N1 Macro 12

1.2.2.4 DRV_UART1_CONFIG_8N2 Macro 12

1.2.2.5 DRV_UART1_CONFIG_8O1 Macro 12

1.2.2.6 DRV_UART1_CONFIG_8O2 Macro 12

1.2.2.7 DRV_UART1_CONFIG_9N1 Macro 13

1.2.2.8 DRV_UART1_CONFIG_9N2 Macro 13

1.2.2.9 DRV_UART1_CONFIG_BAUD_RATE Macro 13

1.2.2.10 DRV_UART1_CONFIG_RX_BYTEQ_LENGTH Macro 13

1.2.2.11 DRV_UART1_CONFIG_TX_BYTEQ_LENGTH Macro 14

1.2.2.12 DRV_UART2_CONFIG_8E1 Macro 14

1.2.2.13 DRV_UART2_CONFIG_8E2 Macro 14

1.2.2.14 DRV_UART2_CONFIG_8N1 Macro 15

1.2.2.15 DRV_UART2_CONFIG_8N2 Macro 15

1.2.2.16 DRV_UART2_CONFIG_8O1 Macro 15

1.2.2.17 DRV_UART2_CONFIG_8O2 Macro 15

1.2.2.18 DRV_UART2_CONFIG_9N1 Macro 16

1.2.2.19 DRV_UART2_CONFIG_9N2 Macro 16

1.2.2.20 DRV_UART2_CONFIG_BAUD_RATE Macro 16

1.2.2.21 DRV_UART2_CONFIG_RX_BYTEQ_LENGTH Macro 16

1.2.2.22 DRV_UART2_CONFIG_TX_BYTEQ_LENGTH Macro 17

1.2.2.23 DRV_UART3_CONFIG_8E1 Macro 17

1.2.2.24 DRV_UART3_CONFIG_8E2 Macro 17

1.2.2.25 DRV_UART3_CONFIG_8N1 Macro 18

1.2.2.26 DRV_UART3_CONFIG_8N2 Macro 18

1.2.2.27 DRV_UART3_CONFIG_8O1 Macro 18

1.2.2.28 DRV_UART3_CONFIG_8O2 Macro 18

1.2.2.29 DRV_UART3_CONFIG_9N1 Macro 19

1.2.2.30 DRV_UART3_CONFIG_9N2 Macro 19

1.2.2.31 DRV_UART3_CONFIG_BAUD_RATE Macro 19
1.2.2.32 DRV_UART3_CONFIG_RX_BYTEQ_LENGTH Macro 19
1.2.2.33 DRV_UART3_CONFIG_TX_BYTEQ_LENGTH Macro 20
1.2.2.34 DRV_UART4_CONFIG_8E1 Macro 20
1.2.2.35 DRV_UART4_CONFIG_8E2 Macro 20
1.2.2.36 DRV_UART4_CONFIG_8N1 Macro 21
1.2.2.37 DRV_UART4_CONFIG_8N2 Macro 21
1.2.2.38 DRV_UART4_CONFIG_8O1 Macro 21
1.2.2.39 DRV_UART4_CONFIG_8O2 Macro 21
1.2.2.40 DRV_UART4_CONFIG_9N1 Macro 22
1.2.2.41 DRV_UART4_CONFIG_9N2 Macro 22
1.2.2.42 DRV_UART4_CONFIG_BAUD_RATE Macro 22
1.2.2.43 DRV_UART4_CONFIG_RX_BYTEQ_LENGTH Macro 22
1.2.2.44 DRV_UART4_CONFIG_TX_BYTEQ_LENGTH Macro 23
1.2.3 Driver Interface 23
1.2.3.1 Data Transfer Functions 23
1.2.3.1.1 DRV_UART1_Peek Function 24
1.2.3.1.2 DRV_UART1_Read Function 24
1.2.3.1.3 DRV_UART1_ReadByte Function 25
1.2.3.1.4 DRV_UART1_Write Function 26
1.2.3.1.5 DRV_UART1_WriteByte Function 27
1.2.3.1.6 DRV_UART2_Peek Function 28
1.2.3.1.7 DRV_UART2_Read Function 28
1.2.3.1.8 DRV_UART2_ReadByte Function 29
1.2.3.1.9 DRV_UART2_Write Function 30
1.2.3.1.10 DRV_UART2_WriteByte Function 31
1.2.3.1.11 DRV_UART3_Peek Function 31
1.2.3.1.12 DRV_UART3_Read Function 32
1.2.3.1.13 DRV_UART3_ReadByte Function 33
1.2.3.1.14 DRV_UART3_Write Function 33
1.2.3.1.15 DRV_UART3_WriteByte Function 34
1.2.3.1.16 DRV_UART4_Peek Function 35
1.2.3.1.17 DRV_UART4_Read Function 36
1.2.3.1.18 DRV_UART4_ReadByte Function 37
1.2.3.1.19 DRV_UART4_Write Function 37
1.2.3.1.20 DRV_UART4_WriteByte Function 38
1.2.3.2 Data Types and Constants 39
1.2.3.2.1 DRV_UART1_STATUS Enumeration 39
1.2.3.2.2 DRV_UART1_TRANSFER_STATUS Enumeration 40
1.2.3.2.3 DRV_UART2_STATUS Enumeration 40
1.2.3.2.4 DRV_UART2_TRANSFER_STATUS Enumeration 41
1.2.3.2.5 DRV_UART3_STATUS Enumeration 42

1.2.3.2.6 DRV_UART3_TRANSFER_STATUS Enumeration 42
1.2.3.2.7 DRV_UART4_STATUS Enumeration 43
1.2.3.2.8 DRV_UART4_TRANSFER_STATUS Enumeration 44
1.2.3.3 Initialization and Setup Functions 44
1.2.3.3.1 DRV_UART1_InitializerDefault Function 45
1.2.3.3.2 DRV_UART1_TasksError Function 45
1.2.3.3.3 DRV_UART1_TasksRX Function 46
1.2.3.3.4 DRV_UART1_TasksTX Function 46
1.2.3.3.5 DRV_UART2_InitializerDefault Function 47
1.2.3.3.6 DRV_UART2_TasksError Function 48
1.2.3.3.7 DRV_UART2_TasksRX Function 48
1.2.3.3.8 DRV_UART2_TasksTX Function 49
1.2.3.3.9 DRV_UART3_InitializerDefault Function 49
1.2.3.3.10 DRV_UART3_TasksError Function 50
1.2.3.3.11 DRV_UART3_TasksRX Function 50
1.2.3.3.12 DRV_UART3_TasksTX Function 51
1.2.3.3.13 DRV_UART4_InitializerDefault Function 52
1.2.3.3.14 DRV_UART4_TasksError Function 52
1.2.3.3.15 DRV_UART4_TasksRX Function 53
1.2.3.3.16 DRV_UART4_TasksTX Function 53
1.2.3.4 Status Functions 54
1.2.3.4.1 DRV_UART1_RXBufferIsEmpty Function 55
1.2.3.4.2 DRV_UART1_RXBufferSizeGet Function 55
1.2.3.4.3 DRV_UART1_Status Function 56
1.2.3.4.4 DRV_UART1_TransferStatus Function 56
1.2.3.4.5 DRV_UART1_TXBufferIsFull Function 57
1.2.3.4.6 DRV_UART1_TXBufferSizeGet Function 57
1.2.3.4.7 DRV_UART2_RXBufferIsEmpty Function 58
1.2.3.4.8 DRV_UART2_RXBufferSizeGet Function 58
1.2.3.4.9 DRV_UART2_Status Function 59
1.2.3.4.10 DRV_UART2_TransferStatus Function 60
1.2.3.4.11 DRV_UART2_TXBufferIsFull Function 60
1.2.3.4.12 DRV_UART2_TXBufferSizeGet Function 60
1.2.3.4.13 DRV_UART3_RXBufferIsEmpty Function 61
1.2.3.4.14 DRV_UART3_RXBufferSizeGet Function 62
1.2.3.4.15 DRV_UART3_Status Function 62
1.2.3.4.16 DRV_UART3_TransferStatus Function 63
1.2.3.4.17 DRV_UART3_TXBufferIsFull Function 63
1.2.3.4.18 DRV_UART3_TXBufferSizeGet Function 64
1.2.3.4.19 DRV_UART4_RXBufferIsEmpty Function 64
1.2.3.4.20 DRV_UART4_RXBufferSizeGet Function 65
1.2.3.4.21 DRV_UART4_Status Function 65

1.2.3.4.22 DRV_UART4_TransferStatus Function 66
1.2.3.4.23 DRV_UART4_TXBufferIsFull Function 66
1.2.3.4.24 DRV_UART4_TXBufferSizeGet Function 67

1.3 SPI Driver 68
1.3.1 Using Driver 68
1.3.1.1 Abstraction Model 69
1.3.2 Configuring the Driver 70
1.3.2.1 DRV_SPI_CONFIG_CHANNEL_1_ENABLE Macro 70
1.3.2.2 DRV_SPI_CONFIG_CHANNEL_2_ENABLE Macro 70
1.3.2.3 DRV_SPI_CONFIG_CHANNEL_3_ENABLE Macro 71
1.3.2.4 DRV_SPI_CONFIG_CHANNEL_4_ENABLE Macro 71
1.3.2.5 DRV_SPI_CONFIG_ENHANCED_BUFFER_DISABLE Macro 71
1.3.3 Driver Interface 71
1.3.3.1 Initialization and Setup Functions 71
1.3.3.1.1 DRV_SPI_Deinitialize Function 72
1.3.3.1.2 DRV_SPI_Initialize Function 72
1.3.3.1.3 DRV_SPI_Lock Function 73
1.3.3.1.4 DRV_SPI_Unlock Function 74
1.3.3.2 Data Transfer Functions 74
1.3.3.2.1 DRV_SPI_Get Function 75
1.3.3.2.2 DRV_SPI_GetBuffer Function 75
1.3.3.2.3 DRV_SPI_Put Function 76
1.3.3.2.4 DRV_SPI_PutBuffer Function 77
1.3.3.3 Data Types and Constants 78
1.3.3.3.1 DRV_SPI_INIT_DATA Structure 78
1.3.3.3.2 SPI_BUS_MODES Enumeration 78

Index 80

Drivers

1 MLA Drivers

This section covers the drivers interfaces used across various libraries in MLA.

Modules

Name	Description
UART Driver	This library provides an interface to manage the UART module on the Microchip family of microcontrollers in different modes of operation.
SPI Driver	This library provides an interface to manage the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers in different modes of operation.

Description

The various drivers described in this section are used in either the libraries or applications provided with the MLA. These drivers can also be used by the application developers to accelerate development time.

1.1 Legal Information

This software distribution is controlled by the Legal Information at www.microchip.com/mla_license

1.2 UART Driver

This library provides an interface to manage the UART module on the Microchip family of microcontrollers in different modes of operation.

Description

Overview of UART

The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem of many embedded systems.

The UART driver can support the following modes of operation (refer to the specific device data sheet to determine which modes are supported on the device in use).

RS-232

RS-232 is an asynchronous full duplex serial communication protocol. It uses separate lines for transmitting and receiving data, point-to-point, between a Data Terminal Equipment (DTE) item and a Data Communication Equipment (DCE) item at a maximum speed of 20 kbps with a maximum cable length of 50 feet.

1.2.1 Using Driver

Module

UART Driver

Description

This topic describes the basic architecture of the UART Driver Library and provides information and examples on how to use it.

Interface Header File: drv_uart1.h, drv_uart2.h, drv_uart3.h, drv_uart4.h

The interface to the UART library is defined in the drv_uart1.h, drv_uart2.h, drv_uart3.h, drv_uart4.h header file

The table below lists the interface section and its brief description.

Library Interface Section	Description
Data Types and Constants	Provides macros for configuring the system. It is required that the system configures the driver to build correctly by choosing appropriate configuration options as listed in this section.
Configuration	Provides driver configuration macros
Initialization Functions	Provides system module interfaces, Device initialization
Data Transfer Functions	Provides data transfer functions available in the driver
Status Functions	Provides status functions

1.2.1.1 Abstraction Model

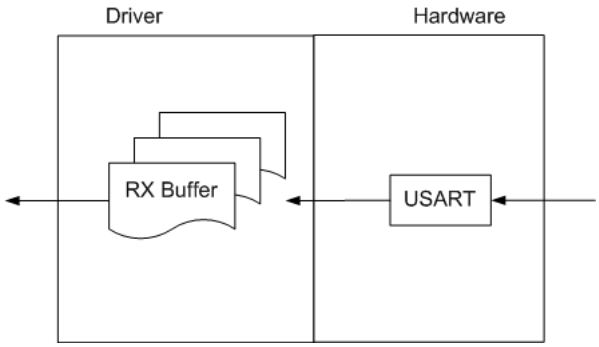
Different types of UARTs are available on Microchip microcontrollers. Some have a FIFO and some do not. The FIFO depth varies across part families. The UART driver abstracts out these differences and provides a unified model for data transfer

across different types of UARTS available.

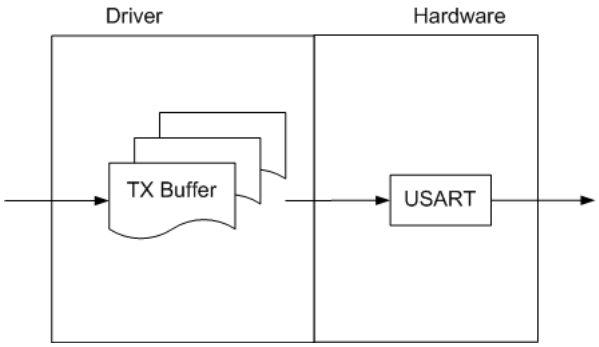
Both transmitter and receiver provide a buffer in the driver which transmits and receives data to/from the hardware. The UART driver provides a set of interfaces to perform the read and the write.

The diagrams below illustrates the model used by the UART driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



1.2.1.2 Initialization

The system performs the initialization of the device driver with settings that affect only the instance of the device that is being initialized. During system initialization each instance of the UART will be initialized with the configuration settings.

1. Baud rate
2. Stop bits
3. Size of the RX buffer
4. Size of TX buffer

1.2.1.3 Data Transfer

Transmitter Functionality

Application using the UART transmitter functionality, needs to perform the following:

1. The system should have completed necessary initialization and the DRV_UART_Initialize
2. Write a byte using DRV_UART_WriteByte or write a buffer using DRV_UART_Write

Example :

```
// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

DRV_UART1_WriteByte(myBuffer[numBytes++]);

// Do something else...
```

Receiver Functionality

Application using the UART receiver functionality, needs to perform the following:

1. The system should have completed necessary initialization
2. Read a byte using DRV_UART_ReadByte or a read a buffer using DRV_UART_Read

Example :

```
byte = DRV_UART_ReadByte();
```

1.2.2 Configuring the Driver

Macros

Name	Description
DRV_UART1_CONFIG_8E1	Macro defines the line control mode to 8-E-1 configuration
DRV_UART1_CONFIG_8E2	Macro defines the line control mode to 8-E-2 configuration
DRV_UART1_CONFIG_8N1	Macro defines the line control mode to 8-N-1 configuration
DRV_UART1_CONFIG_8N2	Macro defines the line control mode to 8-N-2 configuration
DRV_UART1_CONFIG_8O1	Macro defines the line control mode to 8-O-1 configuration
DRV_UART1_CONFIG_8O2	Macro defines the line control mode to 8-O-2 configuration
DRV_UART1_CONFIG_9N1	Macro defines the line control mode to 9-N-1 configuration
DRV_UART1_CONFIG_9N2	Macro defines the line control mode to 9-N-2 configuration
DRV_UART1_CONFIG_BAUD_RATE	Macro controls operation of the driver for Baud rate configuration
DRV_UART1_CONFIG_RX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the RX buffer
DRV_UART1_CONFIG_TX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the TX buffer
DRV_UART2_CONFIG_8E1	Macro defines the line control mode to 8-E-1 configuration
DRV_UART2_CONFIG_8E2	Macro defines the line control mode to 8-E-2 configuration
DRV_UART2_CONFIG_8N1	Macro defines the line control mode to 8-N-1 configuration
DRV_UART2_CONFIG_8N2	Macro defines the line control mode to 8-N-2 configuration
DRV_UART2_CONFIG_8O1	Macro defines the line control mode to 8-O-1 configuration
DRV_UART2_CONFIG_8O2	Macro defines the line control mode to 8-O-2 configuration
DRV_UART2_CONFIG_9N1	Macro defines the line control mode to 9-N-1 configuration
DRV_UART2_CONFIG_9N2	Macro defines the line control mode to 9-N-2 configuration
DRV_UART2_CONFIG_BAUD_RATE	Macro controls operation of the driver for Baud rate configuration
DRV_UART2_CONFIG_RX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the RX buffer
DRV_UART2_CONFIG_TX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the TX buffer
DRV_UART3_CONFIG_8E1	Macro defines the line control mode to 8-E-1 configuration
DRV_UART3_CONFIG_8E2	Macro defines the line control mode to 8-E-2 configuration
DRV_UART3_CONFIG_8N1	Macro defines the line control mode to 8-N-1 configuration
DRV_UART3_CONFIG_8N2	Macro defines the line control mode to 8-N-2 configuration
DRV_UART3_CONFIG_8O1	Macro defines the line control mode to 8-O-1 configuration
DRV_UART3_CONFIG_8O2	Macro defines the line control mode to 8-O-2 configuration
DRV_UART3_CONFIG_9N1	Macro defines the line control mode to 9-N-1 configuration
DRV_UART3_CONFIG_9N2	Macro defines the line control mode to 9-N-2 configuration

DRV_UART3_CONFIG_BAUD_RATE	Macro controls operation of the driver for Baud rate configuration
DRV_UART3_CONFIG_RX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the RX buffer
DRV_UART3_CONFIG_TX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the TX buffer
DRV_UART4_CONFIG_8E1	Macro defines the line control mode to 8-E-1 configuration
DRV_UART4_CONFIG_8E2	Macro defines the line control mode to 8-E-2 configuration
DRV_UART4_CONFIG_8N1	Macro defines the line control mode to 8-N-1 configuration
DRV_UART4_CONFIG_8N2	Macro defines the line control mode to 8-N-2 configuration
DRV_UART4_CONFIG_8O1	Macro defines the line control mode to 8-O-1 configuration
DRV_UART4_CONFIG_8O2	Macro defines the line control mode to 8-O-2 configuration
DRV_UART4_CONFIG_9N1	Macro defines the line control mode to 9-N-1 configuration
DRV_UART4_CONFIG_9N2	Macro defines the line control mode to 9-N-2 configuration
DRV_UART4_CONFIG_BAUD_RATE	Macro controls operation of the driver for Baud rate configuration
DRV_UART4_CONFIG_RX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the RX buffer
DRV_UART4_CONFIG_TX_BYTEQ_LENGTH	Macro controls operation of the driver for defining the size of the TX buffer

Module

UART Driver

Description

1.2.2.1 DRV_UART1_CONFIG_8E1 Macro

Macro defines the line control mode to 8-E-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_8E1
```

Description

Sets the UART for 8-E-1 configuration

This macro defines the line control mode as 8 data bits, even parity and 1 stop bit.

1.2.2.2 DRV_UART1_CONFIG_8E2 Macro

Macro defines the line control mode to 8-E-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_8E2
```

Description

Sets the UART for 8-E-2 configuration

This macro defines the line control mode as 8 data bits, even parity and 2 stop bit.

1.2.2.3 DRV_UART1_CONFIG_8N1 Macro

Macro defines the line control mode to 8-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_8N1
```

Description

Sets the UART for 8-N-1 configuration

This macro defines the line control mode as 8 data bits, none parity and 1 stop bit.

1.2.2.4 DRV_UART1_CONFIG_8N2 Macro

Macro defines the line control mode to 8-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_8N2
```

Description

Sets the UART for 8-N-2 configuration

This macro defines the line control mode as 8 data bits, none parity and 2 stop bit.

1.2.2.5 DRV_UART1_CONFIG_8O1 Macro

Macro defines the line control mode to 8-O-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_8O1
```

Description

Sets the UART for 8-O-1 configuration

This macro defines the line control mode as 8 data bits, odd parity and 1 stop bit.

1.2.2.6 DRV_UART1_CONFIG_8O2 Macro

Macro defines the line control mode to 8-O-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_8O2
```

Description

Sets the UART for 8-O-2 configuration

This macro defines the line control mode as 8 data bits, odd parity and 2 stop bit.

1.2.2.7 DRV_UART1_CONFIG_9N1 Macro

Macro defines the line control mode to 9-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_9N1
```

Description

Sets the UART for 9-N-1 configuration

This macro defines the line control mode as 9 data bits, none parity and 1 stop bit.

1.2.2.8 DRV_UART1_CONFIG_9N2 Macro

Macro defines the line control mode to 9-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_9N2
```

Description

Sets the UART for 9-N-2 configuration

This macro defines the line control mode as 9 data bits, none parity and 2 stop bit.

1.2.2.9 DRV_UART1_CONFIG_BAUD_RATE Macro

Macro controls operation of the driver for Baud rate configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_BAUD_RATE
```

Description

UART Baud Rate configuration

This macro controls the operation of the driver for Baud rate.

1.2.2.10 DRV_UART1_CONFIG_RX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the RX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_RX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer RX size configuration

This macro controls the operation of the driver for defining the size of the RX buffer

1.2.2.11 DRV_UART1_CONFIG_TX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the TX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART1_CONFIG_TX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer TX size configuration

This macro controls the operation of the driver for defining the size of the TX buffer

1.2.2.12 DRV_UART2_CONFIG_8E1 Macro

Macro defines the line control mode to 8-E-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_8E1
```

Description

Sets the UART for 8-E-1 configuration

This macro defines the line control mode as 8 data bits, even parity and 1 stop bit.

1.2.2.13 DRV_UART2_CONFIG_8E2 Macro

Macro defines the line control mode to 8-E-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_8E2
```

Description

Sets the UART for 8-E-2 configuration

This macro defines the line control mode as 8 data bits, even parity and 2 stop bit.

1.2.2.14 DRV_UART2_CONFIG_8N1 Macro

Macro defines the line control mode to 8-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_8N1
```

Description

Sets the UART for 8-N-1 configuration

This macro defines the line control mode as 8 data bits, none parity and 1 stop bit.

1.2.2.15 DRV_UART2_CONFIG_8N2 Macro

Macro defines the line control mode to 8-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_8N2
```

Description

Sets the UART for 8-N-2 configuration

This macro defines the line control mode as 8 data bits, none parity and 2 stop bit.

1.2.2.16 DRV_UART2_CONFIG_8O1 Macro

Macro defines the line control mode to 8-O-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_8O1
```

Description

Sets the UART for 8-O-1 configuration

This macro defines the line control mode as 8 data bits, odd parity and 1 stop bit.

1.2.2.17 DRV_UART2_CONFIG_8O2 Macro

Macro defines the line control mode to 8-O-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_8O2
```

Description

Sets the UART for 8-O-2 configuration

This macro defines the line control mode as 8 data bits, odd parity and 2 stop bit.

1.2.2.18 DRV_UART2_CONFIG_9N1 Macro

Macro defines the line control mode to 9-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_9N1
```

Description

Sets the UART for 9-N-1 configuration

This macro defines the line control mode as 9 data bits, none parity and 1 stop bit.

1.2.2.19 DRV_UART2_CONFIG_9N2 Macro

Macro defines the line control mode to 9-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_9N2
```

Description

Sets the UART for 9-N-2 configuration

This macro defines the line control mode as 9 data bits, none parity and 2 stop bit.

1.2.2.20 DRV_UART2_CONFIG_BAUD_RATE Macro

Macro controls operation of the driver for Baud rate configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_BAUD_RATE
```

Description

UART Baud Rate configuration

This macro controls the operation of the driver for Baud rate.

1.2.2.21 DRV_UART2_CONFIG_RX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the RX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_RX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer RX size configuration

This macro controls the operation of the driver for defining the size of the RX buffer

1.2.2.22 DRV_UART2_CONFIG_TX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the TX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART2_CONFIG_TX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer TX size configuration

This macro controls the operation of the driver for defining the size of the TX buffer

1.2.2.23 DRV_UART3_CONFIG_8E1 Macro

Macro defines the line control mode to 8-E-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_8E1
```

Description

Sets the UART for 8-E-1 configuration

This macro defines the line control mode as 8 data bits, even parity and 1 stop bit.

1.2.2.24 DRV_UART3_CONFIG_8E2 Macro

Macro defines the line control mode to 8-E-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_8E2
```

Description

Sets the UART for 8-E-2 configuration

This macro defines the line control mode as 8 data bits, even parity and 2 stop bit.

1.2.2.25 DRV_UART3_CONFIG_8N1 Macro

Macro defines the line control mode to 8-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_8N1
```

Description

Sets the UART for 8-N-1 configuration

This macro defines the line control mode as 8 data bits, none parity and 1 stop bit.

1.2.2.26 DRV_UART3_CONFIG_8N2 Macro

Macro defines the line control mode to 8-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_8N2
```

Description

Sets the UART for 8-N-2 configuration

This macro defines the line control mode as 8 data bits, none parity and 2 stop bit.

1.2.2.27 DRV_UART3_CONFIG_8O1 Macro

Macro defines the line control mode to 8-O-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_8O1
```

Description

Sets the UART for 8-O-1 configuration

This macro defines the line control mode as 8 data bits, odd parity and 1 stop bit.

1.2.2.28 DRV_UART3_CONFIG_8O2 Macro

Macro defines the line control mode to 8-O-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_8O2
```

Description

Sets the UART for 8-O-2 configuration

This macro defines the line control mode as 8 data bits, odd parity and 2 stop bit.

1.2.2.29 DRV_UART3_CONFIG_9N1 Macro

Macro defines the line control mode to 9-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_9N1
```

Description

Sets the UART for 9-N-1 configuration

This macro defines the line control mode as 9 data bits, none parity and 1 stop bit.

1.2.2.30 DRV_UART3_CONFIG_9N2 Macro

Macro defines the line control mode to 9-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_9N2
```

Description

Sets the UART for 9-N-2 configuration

This macro defines the line control mode as 9 data bits, none parity and 2 stop bit.

1.2.2.31 DRV_UART3_CONFIG_BAUD_RATE Macro

Macro controls operation of the driver for Baud rate configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_BAUD_RATE
```

Description

UART Baud Rate configuration

This macro controls the operation of the driver for Baud rate.

1.2.2.32 DRV_UART3_CONFIG_RX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the RX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_RX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer RX size configuration

This macro controls the operation of the driver for defining the size of the RX buffer

1.2.2.33 DRV_UART3_CONFIG_TX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the TX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART3_CONFIG_TX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer TX size configuration

This macro controls the operation of the driver for defining the size of the TX buffer

1.2.2.34 DRV_UART4_CONFIG_8E1 Macro

Macro defines the line control mode to 8-E-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_8E1
```

Description

Sets the UART for 8-E-1 configuration

This macro defines the line control mode as 8 data bits, even parity and 1 stop bit.

1.2.2.35 DRV_UART4_CONFIG_8E2 Macro

Macro defines the line control mode to 8-E-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_8E2
```

Description

Sets the UART for 8-E-2 configuration

This macro defines the line control mode as 8 data bits, even parity and 2 stop bit.

1.2.2.36 DRV_UART4_CONFIG_8N1 Macro

Macro defines the line control mode to 8-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_8N1
```

Description

Sets the UART for 8-N-1 configuration

This macro defines the line control mode as 8 data bits, none parity and 1 stop bit.

1.2.2.37 DRV_UART4_CONFIG_8N2 Macro

Macro defines the line control mode to 8-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_8N2
```

Description

Sets the UART for 8-N-2 configuration

This macro defines the line control mode as 8 data bits, none parity and 2 stop bit.

1.2.2.38 DRV_UART4_CONFIG_8O1 Macro

Macro defines the line control mode to 8-O-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_8O1
```

Description

Sets the UART for 8-O-1 configuration

This macro defines the line control mode as 8 data bits, odd parity and 1 stop bit.

1.2.2.39 DRV_UART4_CONFIG_8O2 Macro

Macro defines the line control mode to 8-O-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_8O2
```

Description

Sets the UART for 8-O-2 configuration

This macro defines the line control mode as 8 data bits, odd parity and 2 stop bit.

1.2.2.40 DRV_UART4_CONFIG_9N1 Macro

Macro defines the line control mode to 9-N-1 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_9N1
```

Description

Sets the UART for 9-N-1 configuration

This macro defines the line control mode as 9 data bits, none parity and 1 stop bit.

1.2.2.41 DRV_UART4_CONFIG_9N2 Macro

Macro defines the line control mode to 9-N-2 configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_9N2
```

Description

Sets the UART for 9-N-2 configuration

This macro defines the line control mode as 9 data bits, none parity and 2 stop bit.

1.2.2.42 DRV_UART4_CONFIG_BAUD_RATE Macro

Macro controls operation of the driver for Baud rate configuration

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_BAUD_RATE
```

Description

UART Baud Rate configuration

This macro controls the operation of the driver for Baud rate.

1.2.2.43 DRV_UART4_CONFIG_RX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the RX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_RX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer RX size configuration

This macro controls the operation of the driver for defining the size of the RX buffer

1.2.2.44 DRV_UART4_CONFIG_TX_BYTEQ_LENGTH Macro

Macro controls operation of the driver for defining the size of the TX buffer

File

drv_uart_config_template.h

Syntax

```
#define DRV_UART4_CONFIG_TX_BYTEQ_LENGTH 4
```

Description

UART Byte mode internal buffer TX size configuration

This macro controls the operation of the driver for defining the size of the TX buffer

1.2.3 Driver Interface

Module

UART Driver

Description

1.2.3.1 Data Transfer Functions

Functions

	Name	Description
	DRV_UART1_Peek	Returns the character in the read sequence at the offset provided, without extracting it
	DRV_UART1_Read	Returns the number of bytes read by the UART1 peripheral
	DRV_UART1_ReadByte	Reads a byte of data from the UART1
	DRV_UART1_Write	Returns the number of bytes written into the internal buffer
	DRV_UART1_WriteByte	Writes a byte of data to the UART1
	DRV_UART2_Peek	Returns the character in the read sequence at the offset provided, without extracting it
	DRV_UART2_Read	Returns the number of bytes read by the UART2 peripheral
	DRV_UART2_ReadByte	Reads a byte of data from the UART2
	DRV_UART2_Write	Returns the number of bytes written into the internal buffer
	DRV_UART2_WriteByte	Writes a byte of data to the UART2

	DRV_UART3_Peek	Returns the character in the read sequence at the offset provided, without extracting it
	DRV_UART3_Read	Returns the number of bytes read by the UART3 peripheral
	DRV_UART3_ReadByte	Reads a byte of data from the UART3
	DRV_UART3_Write	Returns the number of bytes written into the internal buffer
	DRV_UART3_WriteByte	Writes a byte of data to the UART3
	DRV_UART4_Peek	Returns the character in the read sequence at the offset provided, without extracting it
	DRV_UART4_Read	Returns the number of bytes read by the UART4 peripheral
	DRV_UART4_ReadByte	Reads a byte of data from the UART4
	DRV_UART4_Write	Returns the number of bytes written into the internal buffer
	DRV_UART4_WriteByte	Writes a byte of data to the UART4

Description

1.2.3.1.1 DRV_UART1_Peek Function

Returns the character in the read sequence at the offset provided, without extracting it

File

drv_uart1.h

Syntax

```
uint8_t DRV_UART1_Peek(uint16_t offset);
```

Description

This routine returns the character in the read sequence at the offset provided, without extracting it

Example

```
const uint8_t readBuffer[5];
unsigned int data, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART1_InitializerDefault();

while(numBytes < readbufferLen)
{
    DRV_UART1_TasksRX ( );
    //Check for data at a particular place in the buffer
    data = DRV_UART1_Peek(3);
    if(data == 5)
    {
        //discard all other data if byte that is wanted is received.
        //continue other operation
        numBytes += DRV_UART1_Read ( readBuffer + numBytes , readbufferLen ) ;
    }
    else
    {
        break;
    }
}
```

Function

uint8_t DRV_UART1_Peek(uint16_t offset)

1.2.3.1.2 DRV_UART1_Read Function

Returns the number of bytes read by the UART1 peripheral

File

drv_uart1.h

Syntax

```
unsigned int DRV_UART1_Read(uint8_t * buffer, const unsigned int numbytes);
```

Returns

Number of bytes actually copied into the caller's buffer or -1 if there is an error.

Description

This routine returns the number of bytes read by the Peripheral and fills the application read buffer with the read data.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART1_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART1_TransferStatus ( ) ;
    if (status & DRV_UART1_TRANSFER_STATUS_RX_FULL)
    {
        numBytes += DRV_UART1_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Parameters

Parameters	Description
uint8_t * buffer	Buffer into which the data read from the UART1
const unsigned int numbytes	Total number of bytes that need to be read from the UART1 (must be equal to or less than the size of the buffer)

Function

unsigned int DRV_UART1_Read(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.3 DRV_UART1_ReadByte Function

Reads a byte of data from the UART1

File

drv_uart1.h

Syntax

```
uint8_t DRV_UART1_ReadByte();
```

Returns

A data byte received by the driver.

Description

This routine reads a byte of data from the UART1.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if the receiver is not empty before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

numBytes = 0;
do
{
    if( DRV_UART1_TRANSFER_STATUS_RX_DATA_PRESENT & DRV_UART1_TransferStatus() )
    {
        myBuffer[numBytes++] = DRV_UART1_ReadByte();
    }

    // Do something else...

} while( numBytes < MY_BUFFER_SIZE);
```

Function

uint8_t DRV_UART1_ReadByte(void)

1.2.3.1.4 DRV_UART1_Write Function

Returns the number of bytes written into the internal buffer

File

drv_uart1.h

Syntax

```
unsigned int DRV_UART1_Write(const uint8_t * buffer, const unsigned int numbytes);
```

Description

This API transfers the data from application buffer to internal buffer and returns the number of bytes added in that queue

Remarks

None

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART1_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART1_TransferStatus ( ) ;
```

```
    if (status & DRV_UART1_TRANSFER_STATUS_TX_EMPTY)
    {
        numBytes += DRV_UART1_Write ( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < writebufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Function

unsigned int DRV_UART1_Write(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.5 DRV_UART1_WriteByte Function

Writes a byte of data to the UART1

File

drv_uart1.h

Syntax

```
void DRV_UART1_WriteByte(const uint8_t byte);
```

Returns

None.

Description

This routine writes a byte of data to the UART1.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if transmitter is not full before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    if( !(DRV_UART1_TRANSFER_STATUS_TX_FULL & DRV_UART1_TransferStatus()) )
    {
        DRV_UART1_WriteByte(handle, myBuffer[numBytes++]);
    }

    // Do something else...
}
```

Parameters

Parameters	Description
const uint8_t byte	Data byte to write to the UART1

Function

void DRV_UART1_WriteByte(const uint8_t byte)

1.2.3.1.6 DRV_UART2_Peek Function

Returns the character in the read sequence at the offset provided, without extracting it

File

drv_uart2.h

Syntax

```
uint8_t DRV_UART2_Peek(uint16_t offset);
```

Description

This routine returns the character in the read sequence at the offset provided, without extracting it

Example

```
const uint8_t readBuffer[5];
unsigned int data, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART2_InitializerDefault();

while(numBytes < readbufferLen)
{
    DRV_UART2_TasksRX ( );
    //Check for data at a particular place in the buffer
    data = DRV_UART2_Peek(3);
    if(data == 5)
    {
        //discard all other data if byte that is wanted is received.
        //continue other operation
        numBytes += DRV_UART2_Read ( readBuffer + numBytes , readbufferLen ) ;
    }
    else
    {
        break;
    }
}
```

Function

uint8_t DRV_UART2_Peek(uint16_t offset)

1.2.3.1.7 DRV_UART2_Read Function

Returns the number of bytes read by the UART2 peripheral

File

drv_uart2.h

Syntax

```
unsigned int DRV_UART2_Read(uint8_t * buffer, const unsigned int numbytes);
```

Returns

Number of bytes actually copied into the caller's buffer or -1 if there is an error.

Description

This routine returns the number of bytes read by the Peripheral and fills the application read buffer with the read data.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART2_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART2_TransferStatus ( ) ;
    if (status & DRV_UART2_TRANSFER_STATUS_RX_FULL)
    {
        numBytes += DRV_UART2_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Parameters

Parameters	Description
uint8_t * buffer	Buffer into which the data read from the UART2
const unsigned int numbytes	Total number of bytes that need to be read from the UART2 (must be equal to or less than the size of the buffer)

Function

unsigned int DRV_UART2_Read(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.8 DRV_UART2_ReadByte Function

Reads a byte of data from the UART2

File

drv_uart2.h

Syntax

```
uint8_t DRV_UART2_ReadByte();
```

Returns

A data byte received by the driver.

Description

This routine reads a byte of data from the UART2.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if the receiver is not empty before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
```

```
unsigned int    numBytes;

numBytes = 0;
do
{
    if( DRV_UART2_TRANSFER_STATUS_RX_DATA_PRESENT & DRV_UART2_TransferStatus() )
    {
        myBuffer[numBytes++] = DRV_UART2_ReadByte();
    }

    // Do something else...

} while( numBytes < MY_BUFFER_SIZE);
```

Function

uint8_t DRV_UART2_ReadByte(void)

1.2.3.1.9 DRV_UART2_Write Function

Returns the number of bytes written into the internal buffer

File

drv_uart2.h

Syntax

```
unsigned int DRV_UART2_Write(const uint8_t * buffer, const unsigned int numbytes);
```

Description

This API transfers the data from application buffer to internal buffer and returns the number of bytes added in that queue

Remarks

None

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART2_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART2_TransferStatus ( ) ;
    if (status & DRV_UART2_TRANSFER_STATUS_TX_EMPTY)
    {
        numBytes += DRV_UART2_Write ( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < writebufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Function

unsigned int DRV_UART2_Write(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.10 DRV_UART2_WriteByte Function

Writes a byte of data to the UART2

File

drv_uart2.h

Syntax

```
void DRV_UART2_WriteByte(const uint8_t byte);
```

Returns

None.

Description

This routine writes a byte of data to the UART2.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if transmitter is not full before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    if( !(DRV_UART2_TRANSFER_STATUS_TX_FULL & DRV_UART2_TransferStatus()) )
    {
        DRV_UART2_WriteByte(handle, myBuffer[numBytes++]);
    }

    // Do something else...
}
```

Parameters

Parameters	Description
const uint8_t byte	Data byte to write to the UART2

Function

void DRV_UART2_WriteByte(const uint8_t byte)

1.2.3.1.11 DRV_UART3_Peek Function

Returns the character in the read sequence at the offset provided, without extracting it

File

drv_uart3.h

Syntax

```
uint8_t DRV_UART3_Peek(uint16_t offset);
```

Description

This routine returns the character in the read sequence at the offset provided, without extracting it

Example

```
const uint8_t readBuffer[5];
unsigned int data, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART3_InitializerDefault();

while(numBytes < readbufferLen)
{
    DRV_UART3_TasksRX ( );
    //Check for data at a particular place in the buffer
    data = DRV_UART3_Peek(3);
    if(data == 5)
    {
        //discard all other data if byte that is wanted is received.
        //continue other operation
        numBytes += DRV_UART3_Read ( readBuffer + numBytes , readbufferLen ) ;
    }
    else
    {
        break;
    }
}
```

Function

uint8_t DRV_UART3_Peek(uint16_t offset)

1.2.3.1.12 DRV_UART3_Read Function

Returns the number of bytes read by the UART3 peripheral

File

drv_uart3.h

Syntax

```
unsigned int DRV_UART3_Read(uint8_t * buffer, const unsigned int numbytes);
```

Returns

Number of bytes actually copied into the caller's buffer or -1 if there is an error.

Description

This routine returns the number of bytes read by the Peripheral and fills the application read buffer with the read data.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function

Example

```
char                     myBuffer[MY_BUFFER_SIZE];
unsigned int             numBytes;
DRV_UART3_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART3_TransferStatus ( ) ;
    if (status & DRV_UART3_TRANSFER_STATUS_RX_FULL)
    {
        numBytes += DRV_UART3_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
```

```
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Parameters

Parameters	Description
uint8_t * buffer	Buffer into which the data read from the UART3
const unsigned int numbytes	Total number of bytes that need to be read from the UART3 (must be equal to or less than the size of the buffer)

Function

unsigned int DRV_UART3_Read(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.13 DRV_UART3_ReadByte Function

Reads a byte of data from the UART3

File

drv_uart3.h

Syntax

```
uint8_t DRV_UART3_ReadByte();
```

Returns

A data byte received by the driver.

Description

This routine reads a byte of data from the UART3.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if the receiver is not empty before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

numBytes = 0;
do
{
    if( DRV_UART3_TRANSFER_STATUS_RX_DATA_PRESENT & DRV_UART3_TransferStatus() )
    {
        myBuffer[numBytes++] = DRV_UART3_ReadByte();
    }

    // Do something else...

} while( numBytes < MY_BUFFER_SIZE);
```

Function

uint8_t DRV_UART3_ReadByte(void)

1.2.3.1.14 DRV_UART3_Write Function

Returns the number of bytes written into the internal buffer

File

drv_uart3.h

Syntax

```
unsigned int DRV_UART3_Write(const uint8_t * buffer, const unsigned int numbytes);
```

Description

This API transfers the data from application buffer to internal buffer and returns the number of bytes added in that queue

Remarks

None

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function

Example

```
char                     myBuffer[MY_BUFFER_SIZE];
unsigned int             numBytes;
DRV_UART3_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART3_TransferStatus ( ) ;
    if (status & DRV_UART3_TRANSFER_STATUS_TX_EMPTY)
    {
        numBytes += DRV_UART3_Write ( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < writebufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Function

unsigned int DRV_UART3_Write(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.15 DRV_UART3_WriteByte Function

Writes a byte of data to the UART3

File

drv_uart3.h

Syntax

```
void DRV_UART3_WriteByte(const uint8_t byte);
```

Returns

None.

Description

This routine writes a byte of data to the UART3.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if transmitter is not full before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    if( !(DRV_UART3_TRANSFER_STATUS_TX_FULL & DRV_UART3_TransferStatus()) )
    {
        DRV_UART3_WriteByte(handle, myBuffer[numBytes++]);
    }

    // Do something else...
}
```

Parameters

Parameters	Description
const uint8_t byte	Data byte to write to the UART3

Function

void DRV_UART3_WriteByte(const uint8_t byte)

1.2.3.1.16 DRV_UART4_Peek Function

Returns the character in the read sequence at the offset provided, without extracting it

File

drv_uart4.h

Syntax

```
uint8_t DRV_UART4_Peek(uint16_t offset);
```

Description

This routine returns the character in the read sequence at the offset provided, without extracting it

Example

```
const uint8_t readBuffer[5];
unsigned int data, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART4_InitializerDefault();

while(numBytes < readbufferLen)
{
    DRV_UART4_TasksRX ( );
    //Check for data at a particular place in the buffer
    data = DRV_UART4_Peek(3);
    if(data == 5)
    {
        //discard all other data if byte that is wanted is received.
        //continue other operation
        numBytes += DRV_UART4_Read ( readBuffer + numBytes , readbufferLen ) ;
    }
    else
    {
```

```
            break;
        }
    }
```

Function

uint8_t DRV_UART4_Peek(uint16_t offset)

1.2.3.1.17 DRV_UART4_Read Function

Returns the number of bytes read by the UART4 peripheral

File

drv_uart4.h

Syntax

```
unsigned int DRV_UART4_Read(uint8_t * buffer, const unsigned int numbytes);
```

Returns

Number of bytes actually copied into the caller's buffer or -1 if there is an error.

Description

This routine returns the number of bytes read by the Peripheral and fills the application read buffer with the read data.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART4_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART4_TransferStatus ( ) ;
    if (status & DRV_UART4_TRANSFER_STATUS_RX_FULL)
    {
        numBytes += DRV_UART4_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Parameters

Parameters	Description
uint8_t * buffer	Buffer into which the data read from the UART4
const unsigned int numbytes	Total number of bytes that need to be read from the UART4 (must be equal to or less than the size of the buffer)

Function

unsigned int DRV_UART4_Read(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.18 DRV_UART4_ReadByte Function

Reads a byte of data from the UART4

File

drv_uart4.h

Syntax

```
uint8_t DRV_UART4_ReadByte();
```

Returns

A data byte received by the driver.

Description

This routine reads a byte of data from the UART4.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if the receiver is not empty before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

numBytes = 0;
do
{
    if( DRV_UART4_TRANSFER_STATUS_RX_DATA_PRESENT & DRV_UART4_TransferStatus() )
    {
        myBuffer[numBytes++] = DRV_UART4_ReadByte();
    }

    // Do something else...

} while( numBytes < MY_BUFFER_SIZE);
```

Function

uint8_t DRV_UART4_ReadByte(void)

1.2.3.1.19 DRV_UART4_Write Function

Returns the number of bytes written into the internal buffer

File

drv_uart4.h

Syntax

```
unsigned int DRV_UART4_Write(const uint8_t * buffer, const unsigned int numbytes);
```

Description

This API transfers the data from application buffer to internal buffer and returns the number of bytes added in that queue

Remarks

None

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART4_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART4_TransferStatus ( ) ;
    if (status & DRV_UART4_TRANSFER_STATUS_TX_EMPTY)
    {
        numBytes += DRV_UART4_Write ( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < writebufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Function

unsigned int DRV_UART4_Write(uint8_t *buffer, const unsigned int numbytes)

1.2.3.1.20 DRV_UART4_WriteByte Function

Writes a byte of data to the UART4

File

drv_uart4.h

Syntax

```
void DRV_UART4_WriteByte(const uint8_t byte);
```

Returns

None.

Description

This routine writes a byte of data to the UART4.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function. The transfer status should be checked to see if transmitter is not full before calling this function.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    if( !(DRV_UART4_TRANSFER_STATUS_TX_FULL & DRV_UART4_TransferStatus()) )
```

```
    {
        DRV_UART4_WriteByte(handle, myBuffer[numBytes++]);
    }

    // Do something else...
}
```

Parameters

Parameters	Description
const uint8_t byte	Data byte to write to the UART4

Function

void DRV_UART4_WriteByte(const uint8_t byte)

1.2.3.2 Data Types and Constants

Enumerations

Name	Description
DRV_UART1_STATUS	Specifies the status of the hardware receive or transmit
DRV_UART1_TRANSFER_STATUS	Specifies the status of the receive or transmit
DRV_UART2_STATUS	Specifies the status of the hardware receive or transmit
DRV_UART2_TRANSFER_STATUS	Specifies the status of the receive or transmit
DRV_UART3_STATUS	Specifies the status of the hardware receive or transmit
DRV_UART3_TRANSFER_STATUS	Specifies the status of the receive or transmit
DRV_UART4_STATUS	Specifies the status of the hardware receive or transmit
DRV_UART4_TRANSFER_STATUS	Specifies the status of the receive or transmit

Description

1.2.3.2.1 DRV_UART1_STATUS Enumeration

Specifies the status of the hardware receive or transmit

File

drv_uart1.h

Syntax

```
typedef enum {
  DRV_UART1_RX_DATA_AVAILABLE,
  DRV_UART1_RX_OVERRUN_ERROR,
  DRV_UART1_FRAMING_ERROR,
  DRV_UART1_PARITY_ERROR,
  DRV_UART1_RECEIVER_IDLE,
  DRV_UART1_TX_COMPLETE,
  DRV_UART1_TX_FULL
} DRV_UART1_STATUS;
```

Members

Members	Description
DRV_UART1_RX_DATA_AVAILABLE	Indicates that Receive buffer has data, at least one more character can be read
DRV_UART1_RX_OVERRUN_ERROR	Indicates that Receive buffer has overflowed
DRV_UART1_FRAMING_ERROR	Indicates that Framing error has been detected for the current character

DRV_UART1_PARITY_ERROR	Indicates that Parity error has been detected for the current character
DRV_UART1_RECEIVER_IDLE	Indicates that Receiver is Idle
DRV_UART1_TX_COMPLETE	Indicates that the last transmission has completed
DRV_UART1_TX_FULL	Indicates that Transmit buffer is full

Description

UART1 Driver Hardware Flags

This type specifies the status of the hardware receive or transmit.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.2 DRV_UART1_TRANSFER_STATUS Enumeration

Specifies the status of the receive or transmit

File

drv_uart1.h

Syntax

```
typedef enum {
  DRV_UART1_TRANSFER_STATUS_RX_FULL,
  DRV_UART1_TRANSFER_STATUS_RX_DATA_PRESENT,
  DRV_UART1_TRANSFER_STATUS_RX_EMPTY,
  DRV_UART1_TRANSFER_STATUS_TX_FULL,
  DRV_UART1_TRANSFER_STATUS_TX_EMPTY
} DRV_UART1_TRANSFER_STATUS;
```

Members

Members	Description
DRV_UART1_TRANSFER_STATUS_RX_FULL	Indicates that the core driver buffer is full
DRV_UART1_TRANSFER_STATUS_RX_DATA_PRESENT	Indicates that at least one byte of Data has been received
DRV_UART1_TRANSFER_STATUS_RX_EMPTY	Indicates that the core driver receiver buffer is empty
DRV_UART1_TRANSFER_STATUS_TX_FULL	Indicates that the core driver transmitter buffer is full
DRV_UART1_TRANSFER_STATUS_TX_EMPTY	Indicates that the core driver transmitter buffer is empty

Description

UART1 Driver Transfer Flags

This type specifies the status of the receive or transmit operation.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.3 DRV_UART2_STATUS Enumeration

Specifies the status of the hardware receive or transmit

File

drv_uart2.h

Syntax

```
typedef enum {
  DRV_UART2_RX_DATA_AVAILABLE,
```

```
  DRV_UART2_RX_OVERRUN_ERROR,
  DRV_UART2_FRAMING_ERROR,
  DRV_UART2_PARITY_ERROR,
  DRV_UART2_RECEIVER_IDLE,
  DRV_UART2_TX_COMPLETE,
  DRV_UART2_TX_FULL
} DRV_UART2_STATUS;
```

Members

Members	Description
DRV_UART2_RX_DATA_AVAILABLE	Indicates that Receive buffer has data, at least one more character can be read
DRV_UART2_RX_OVERRUN_ERROR	Indicates that Receive buffer has overflowed
DRV_UART2_FRAMING_ERROR	Indicates that Framing error has been detected for the current character
DRV_UART2_PARITY_ERROR	Indicates that Parity error has been detected for the current character
DRV_UART2_RECEIVER_IDLE	Indicates that Receiver is Idle
DRV_UART2_TX_COMPLETE	Indicates that the last transmission has completed
DRV_UART2_TX_FULL	Indicates that Transmit buffer is full

Description

UART2 Driver Hardware Flags

This type specifies the status of the hardware receive or transmit.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.4 DRV_UART2_TRANSFER_STATUS Enumeration

Specifies the status of the receive or transmit

File

drv_uart2.h

Syntax

```
typedef enum {
  DRV_UART2_TRANSFER_STATUS_RX_FULL,
  DRV_UART2_TRANSFER_STATUS_RX_DATA_PRESENT,
  DRV_UART2_TRANSFER_STATUS_RX_EMPTY,
  DRV_UART2_TRANSFER_STATUS_TX_FULL,
  DRV_UART2_TRANSFER_STATUS_TX_EMPTY
} DRV_UART2_TRANSFER_STATUS;
```

Members

Members	Description
DRV_UART2_TRANSFER_STATUS_RX_FULL	Indicates that the core driver buffer is full
DRV_UART2_TRANSFER_STATUS_RX_DATA_PRESENT	Indicates that at least one byte of Data has been received
DRV_UART2_TRANSFER_STATUS_RX_EMPTY	Indicates that the core driver receiver buffer is empty
DRV_UART2_TRANSFER_STATUS_TX_FULL	Indicates that the core driver transmitter buffer is full
DRV_UART2_TRANSFER_STATUS_TX_EMPTY	Indicates that the core driver transmitter buffer is empty

Description

UART2 Driver Transfer Flags

This type specifies the status of the receive or transmit operation.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.5 DRV_UART3_STATUS Enumeration

Specifies the status of the hardware receive or transmit

File

drv_uart3.h

Syntax

```
typedef enum {
  DRV_UART3_RX_DATA_AVAILABLE,
  DRV_UART3_RX_OVERRUN_ERROR,
  DRV_UART3_FRAMING_ERROR,
  DRV_UART3_PARITY_ERROR,
  DRV_UART3_RECEIVER_IDLE,
  DRV_UART3_TX_COMPLETE,
  DRV_UART3_TX_FULL
} DRV_UART3_STATUS;
```

Members

Members	Description
DRV_UART3_RX_DATA_AVAILABLE	Indicates that Receive buffer has data, at least one more character can be read
DRV_UART3_RX_OVERRUN_ERROR	Indicates that Receive buffer has overflowed
DRV_UART3_FRAMING_ERROR	Indicates that Framing error has been detected for the current character
DRV_UART3_PARITY_ERROR	Indicates that Parity error has been detected for the current character
DRV_UART3_RECEIVER_IDLE	Indicates that Receiver is Idle
DRV_UART3_TX_COMPLETE	Indicates that the last transmission has completed
DRV_UART3_TX_FULL	Indicates that Transmit buffer is full

Description

UART3 Driver Hardware Flags

This type specifies the status of the hardware receive or transmit.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.6 DRV_UART3_TRANSFER_STATUS Enumeration

Specifies the status of the receive or transmit

File

drv_uart3.h

Syntax

```
typedef enum {
  DRV_UART3_TRANSFER_STATUS_RX_FULL,
  DRV_UART3_TRANSFER_STATUS_RX_DATA_PRESENT,
  DRV_UART3_TRANSFER_STATUS_RX_EMPTY,
  DRV_UART3_TRANSFER_STATUS_TX_FULL,
  DRV_UART3_TRANSFER_STATUS_TX_EMPTY
```

```
} DRV_UART3_TRANSFER_STATUS;
```

Members

Members	Description
DRV_UART3_TRANSFER_STATUS_RX_FULL	Indicates that the core driver buffer is full
DRV_UART3_TRANSFER_STATUS_RX_DATA_PRESENT	Indicates that at least one byte of Data has been received
DRV_UART3_TRANSFER_STATUS_RX_EMPTY	Indicates that the core driver receiver buffer is empty
DRV_UART3_TRANSFER_STATUS_TX_FULL	Indicates that the core driver transmitter buffer is full
DRV_UART3_TRANSFER_STATUS_TX_EMPTY	Indicates that the core driver transmitter buffer is empty

Description

UART3 Driver Transfer Flags

This type specifies the status of the receive or transmit operation.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.7 DRV_UART4_STATUS Enumeration

Specifies the status of the hardware receive or transmit

File

drv_uart4.h

Syntax

```
typedef enum {
  DRV_UART4_RX_DATA_AVAILABLE,
  DRV_UART4_RX_OVERRUN_ERROR,
  DRV_UART4_FRAMING_ERROR,
  DRV_UART4_PARITY_ERROR,
  DRV_UART4_RECEIVER_IDLE,
  DRV_UART4_TX_COMPLETE,
  DRV_UART4_TX_FULL
} DRV_UART4_STATUS;
```

Members

Members	Description
DRV_UART4_RX_DATA_AVAILABLE	Indicates that Receive buffer has data, at least one more character can be read
DRV_UART4_RX_OVERRUN_ERROR	Indicates that Receive buffer has overflowed
DRV_UART4_FRAMING_ERROR	Indicates that Framing error has been detected for the current character
DRV_UART4_PARITY_ERROR	Indicates that Parity error has been detected for the current character
DRV_UART4_RECEIVER_IDLE	Indicates that Receiver is Idle
DRV_UART4_TX_COMPLETE	Indicates that the last transmission has completed
DRV_UART4_TX_FULL	Indicates that Transmit buffer is full

Description

UART4 Driver Hardware Flags

This type specifies the status of the hardware receive or transmit.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.2.8 DRV_UART4_TRANSFER_STATUS Enumeration

Specifies the status of the receive or transmit

File

drv_uart4.h

Syntax

```
typedef enum {
  DRV_UART4_TRANSFER_STATUS_RX_FULL,
  DRV_UART4_TRANSFER_STATUS_RX_DATA_PRESENT,
  DRV_UART4_TRANSFER_STATUS_RX_EMPTY,
  DRV_UART4_TRANSFER_STATUS_TX_FULL,
  DRV_UART4_TRANSFER_STATUS_TX_EMPTY
} DRV_UART4_TRANSFER_STATUS;
```

Members

Members	Description
DRV_UART4_TRANSFER_STATUS_RX_FULL	Indicates that the core driver buffer is full
DRV_UART4_TRANSFER_STATUS_RX_DATA_PRESENT	Indicates that at least one byte of Data has been received
DRV_UART4_TRANSFER_STATUS_RX_EMPTY	Indicates that the core driver receiver buffer is empty
DRV_UART4_TRANSFER_STATUS_TX_FULL	Indicates that the core driver transmitter buffer is full
DRV_UART4_TRANSFER_STATUS_TX_EMPTY	Indicates that the core driver transmitter buffer is empty

Description

UART4 Driver Transfer Flags

This type specifies the status of the receive or transmit operation.

Remarks

More than one of these values may be OR'd together to create a complete status value. To test a value of this type, the bit of interest must be AND'ed with value and checked to see if the result is non-zero.

1.2.3.3 Initialization and Setup Functions

Functions

	Name	Description
	DRV_UART1_InitializerDefault	Initializes the UART instance : 1
	DRV_UART1_TasksError	Maintains the driver's error-handling state machine in a polled manner.
	DRV_UART1_TasksRX	Maintains the driver's receiver state machine in a polled manner.
	DRV_UART1_TasksTX	Maintains the driver's transmitter state machine in a polled manner
	DRV_UART2_InitializerDefault	Initializes the UART instance : 2
	DRV_UART2_TasksError	Maintains the driver's error-handling state machine in a polled manner.
	DRV_UART2_TasksRX	Maintains the driver's receiver state machine in a polled manner.
	DRV_UART2_TasksTX	Maintains the driver's transmitter state machine in a polled manner
	DRV_UART3_InitializerDefault	Initializes the UART instance : 3
	DRV_UART3_TasksError	Maintains the driver's error-handling state machine in a polled manner.
	DRV_UART3_TasksRX	Maintains the driver's receiver state machine in a polled manner.
	DRV_UART3_TasksTX	Maintains the driver's transmitter state machine in a polled manner
	DRV_UART4_InitializerDefault	Initializes the UART instance : 4
	DRV_UART4_TasksError	Maintains the driver's error-handling state machine in a polled manner.
	DRV_UART4_TasksRX	Maintains the driver's receiver state machine in a polled manner.
	DRV_UART4_TasksTX	Maintains the driver's transmitter state machine in a polled manner

Description

1.2.3.3.1 DRV_UART1_InitializerDefault Function

Initializes the UART instance : 1

File

drv_uart1.h

Syntax

```
void DRV_UART1_InitializerDefault();
```

Returns

None.

Description

This routine initializes the UART driver instance for : 1 index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other UART routine is called.

Preconditions

None.

Example

```
const uint8_t writeBuffer[35] = "1234567890ABCDEFGHIJKLMNOPn" ;
unsigned int numBytes = 0;
int  writebufferLen = strlen((char *)writeBuffer);
DRV_UART1_InitializerDefault();
while(numBytes < writebufferLen)
{
    int bytesToWrite = DRV_UART1_TXBufferSizeGet();
    numBytes = DRV_UART1_Write ( writeBuffer+numBytes, bytesToWrite)  ;
    DRV_UART1_TasksTX ( );
    if (!DRV_UART1_TXBufferisFull())
    {
        //continue other operation
    }
}
```

Function

void DRV_UART1_InitializerDefault(void)

1.2.3.3.2 DRV_UART1_TasksError Function

Maintains the driver's error-handling state machine in a polled manner.

File

drv_uart1.h

Syntax

```
void DRV_UART1_TasksError();
```

Returns

None.

Description

This routine is used to maintain the driver's internal error-handling state machine.This routine is called when the state of the errors needs to be maintained in a polled manner.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
while (true)
{
    DRV_UART1_TasksError ();

    // Do other tasks
}
```

Function

void DRV_UART1_TasksError (void);

1.2.3.3.3 DRV_UART1_TasksRX Function

Maintains the driver's receiver state machine in a polled manner.

File

drv_uart1.h

Syntax

```
void DRV_UART1_TasksRX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal receiver state machine. This routine is called when the state of the receiver needs to be maintained in a polled manner.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
const uint8_t readBuffer[35];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART1__InitializerDefault();

while(numBytes < readbufferLen)
{
    while(!DRV_UART1_RXBufferIsEmpty());
    numBytes += DRV_UART1_Read ( readBuffer + numBytes , readbufferLen ) ;
    DRV_UART1_TasksRX ( );
    status = DRV_UART1_TransferStatus ( ) ;
    if (status & DRV_UART1_TRANSFER_STATUS_RX_FULL)
    {
        //continue other operation
    }
}
```

Function

void DRV_UART1_TasksRX (void);

1.2.3.3.4 DRV_UART1_TasksTX Function

Maintains the driver's transmitter state machine in a polled manner

File

drv_uart1.h

Syntax

```
void DRV_UART1_TasksTX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal transmitter state machine.This routine is called when the state of the transmitter needs to be maintained in a polled manner.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function in a polled loop.

Example

Refer to DRV_UART1_InitializerDefault() for an example

Function

void DRV_UART1_TasksTX (void);

1.2.3.3.5 DRV_UART2_InitializerDefault Function

Initializes the UART instance : 2

File

drv_uart2.h

Syntax

```
void DRV_UART2_InitializerDefault();
```

Returns

None.

Description

This routine initializes the UART driver instance for : 2 index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other UART routine is called.

Preconditions

None.

Example

```
const uint8_t writeBuffer[35] = "1234567890ABCDEFGHIJKLMNOPn" ;
unsigned int numBytes = 0;
int  writebufferLen = strlen((char *)writeBuffer);
DRV_UART2_InitializerDefault();
while(numBytes < writebufferLen)
{
    int bytesToWrite = DRV_UART2_TXBufferSizeGet();
    numBytes = DRV_UART2_Write ( writeBuffer+numBytes, bytesToWrite)  ;
    DRV_UART2_TasksTX ( );
    if (!DRV_UART2_TXBufferisFull())
    {
        //continue other operation
    }
}
```

Function

void DRV_UART2_InitializerDefault(void)

1.2.3.3.6 DRV_UART2_TasksError Function

Maintains the driver's error-handling state machine in a polled manner.

File

drv_uart2.h

Syntax

```
void DRV_UART2_TasksError();
```

Returns

None.

Description

This routine is used to maintain the driver's internal error-handling state machine.This routine is called when the state of the errors needs to be maintained in a polled manner.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
while (true)
{
    DRV_UART2_TasksError ();

    // Do other tasks
}
```

Function

void DRV_UART2_TasksError (void);

1.2.3.3.7 DRV_UART2_TasksRX Function

Maintains the driver's receiver state machine in a polled manner.

File

drv_uart2.h

Syntax

```
void DRV_UART2_TasksRX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal receiver state machine. This routine is called when the state of the receiver needs to be maintained in a polled manner.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
const uint8_t readBuffer[35];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART2__InitializerDefault();

while(numBytes < readbufferLen)
{
    while(!DRV_UART2_RXBufferIsEmpty());
```

```
        numBytes += DRV_UART2_Read ( readBuffer + numBytes , readbufferLen ) ;
        DRV_UART2_TasksRX ( );
        status = DRV_UART2_TransferStatus ( ) ;
        if (status & DRV_UART2_TRANSFER_STATUS_RX_FULL)
        {
            //continue other operation
        }
    }
```

Function

void DRV_UART2_TasksRX (void);

1.2.3.3.8 DRV_UART2_TasksTX Function

Maintains the driver's transmitter state machine in a polled manner

File

drv_uart2.h

Syntax

```
void DRV_UART2_TasksTX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal transmitter state machine.This routine is called when the state of the transmitter needs to be maintained in a polled manner.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function in a polled loop.

Example

Refer to DRV_UART2_InitializerDefault() for an example

Function

void DRV_UART2_TasksTX (void);

1.2.3.3.9 DRV_UART3_InitializerDefault Function

Initializes the UART instance : 3

File

drv_uart3.h

Syntax

```
void DRV_UART3_InitializerDefault();
```

Returns

None.

Description

This routine initializes the UART driver instance for : 3 index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other UART routine is called.

Preconditions

None.

Example

```
const uint8_t writeBuffer[35] = "1234567890ABCDEFGHIJKLMNOPn" ;
unsigned int numBytes = 0;
int  writebufferLen = strlen((char *)writeBuffer);
DRV_UART3_InitializerDefault();
while(numBytes < writebufferLen)
{
    int bytesToWrite = DRV_UART3_TXBufferSizeGet();
    numBytes = DRV_UART3_Write ( writeBuffer+numBytes, bytesToWrite)  ;
    DRV_UART3_TasksTX ( );
    if (!DRV_UART3_TXBufferisFull())
    {
        //continue other operation
    }
}
```

Function

void DRV_UART3_InitializerDefault(void)

1.2.3.3.10 DRV_UART3_TasksError Function

Maintains the driver's error-handling state machine in a polled manner.

File

drv_uart3.h

Syntax

```
void DRV_UART3_TasksError();
```

Returns

None.

Description

This routine is used to maintain the driver's internal error-handling state machine.This routine is called when the state of the errors needs to be maintained in a polled manner.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
while (true)
{
    DRV_UART3_TasksError ();

    // Do other tasks
}
```

Function

void DRV_UART3_TasksError (void);

1.2.3.3.11 DRV_UART3_TasksRX Function

Maintains the driver's receiver state machine in a polled manner.

File

drv_uart3.h

Syntax

```
void DRV_UART3_TasksRX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal receiver state machine. This routine is called when the state of the receiver needs to be maintained in a polled manner.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
const uint8_t readBuffer[35];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART3__InitializerDefault();

while(numBytes < readbufferLen)
{
    while(!DRV_UART3_RXBufferIsEmpty());
    numBytes += DRV_UART3_Read ( readBuffer + numBytes , readbufferLen ) ;
    DRV_UART3_TasksRX ( );
    status = DRV_UART3_TransferStatus ( ) ;
    if (status & DRV_UART3_TRANSFER_STATUS_RX_FULL)
    {
        //continue other operation
    }
}
```

Function

void DRV_UART3_TasksRX (void);

1.2.3.3.12 DRV_UART3_TasksTX Function

Maintains the driver's transmitter state machine in a polled manner

File

drv_uart3.h

Syntax

```
void DRV_UART3_TasksTX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal transmitter state machine.This routine is called when the state of the transmitter needs to be maintained in a polled manner.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function in a polled loop.

Example

Refer to DRV_UART3_InitializerDefault() for an example

Function

void DRV_UART3_TasksTX (void);

1.2.3.3.13 DRV_UART4_InitializerDefault Function

Initializes the UART instance : 4

File

drv_uart4.h

Syntax

```
void DRV_UART4_InitializerDefault();
```

Returns

None.

Description

This routine initializes the UART driver instance for : 4 index, making it ready for clients to open and use it.

Remarks

This routine must be called before any other UART routine is called.

Preconditions

None.

Example

```
const uint8_t writeBuffer[35] = "1234567890ABCDEFGHIJKLMNOPn" ;
unsigned int numBytes = 0;
int  writebufferLen = strlen((char *)writeBuffer);
DRV_UART4_InitializerDefault();
while(numBytes < writebufferLen)
{
    int bytesToWrite = DRV_UART4_TXBufferSizeGet();
    numBytes = DRV_UART4_Write ( writeBuffer+numBytes, bytesToWrite)  ;
    DRV_UART4_TasksTX ( );
    if (!DRV_UART4_TXBufferisFull())
    {
        //continue other operation
    }
}
```

Function

void DRV_UART4_InitializerDefault(void)

1.2.3.3.14 DRV_UART4_TasksError Function

Maintains the driver's error-handling state machine in a polled manner.

File

drv_uart4.h

Syntax

```
void DRV_UART4_TasksError();
```

Returns

None.

Description

This routine is used to maintain the driver's internal error-handling state machine.This routine is called when the state of the errors needs to be maintained in a polled manner.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
while (true)
{
    DRV_UART4_TasksError ();

    // Do other tasks
}
```

Function

void DRV_UART4_TasksError (void);

1.2.3.3.15 DRV_UART4_TasksRX Function

Maintains the driver's receiver state machine in a polled manner.

File

drv_uart4.h

Syntax

```
void DRV_UART4_TasksRX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal receiver state machine. This routine is called when the state of the receiver needs to be maintained in a polled manner.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function in a polled loop.

Example

```
const uint8_t readBuffer[35];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART4__InitializerDefault();

while(numBytes < readbufferLen)
{
    while(!DRV_UART4_RXBufferIsEmpty());
    numBytes += DRV_UART4_Read ( readBuffer + numBytes , readbufferLen ) ;
    DRV_UART4_TasksRX ( );
    status = DRV_UART4_TransferStatus ( ) ;
    if (status & DRV_UART4_TRANSFER_STATUS_RX_FULL)
    {
        //continue other operation
    }
}
```

Function

void DRV_UART4_TasksRX (void);

1.2.3.3.16 DRV_UART4_TasksTX Function

Maintains the driver's transmitter state machine in a polled manner

File

drv_uart4.h

Syntax

```
void DRV_UART4_TasksTX();
```

Returns

None.

Description

This routine is used to maintain the driver's internal transmitter state machine.This routine is called when the state of the transmitter needs to be maintained in a polled manner.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function in a polled loop.

Example

Refer to DRV_UART4_InitializerDefault() for an example

Function

void DRV_UART4_TasksTX (void);

1.2.3.4 Status Functions

Functions

	Name	Description
	DRV_UART1_RXBufferIsEmpty	Returns the status of the receive buffer
	DRV_UART1_RXBufferSizeGet	Returns the size of the receive buffer
	DRV_UART1_Status	Returns the transmitter and receiver status
	DRV_UART1_TransferStatus	Returns the transmitter and receiver transfer status
	DRV_UART1_TXBufferIsFull	Returns the status of the transmit buffer
	DRV_UART1_TXBufferSizeGet	Returns the size of the transmit buffer
	DRV_UART2_RXBufferIsEmpty	Returns the status of the receive buffer
	DRV_UART2_RXBufferSizeGet	Returns the size of the receive buffer
	DRV_UART2_Status	Returns the transmitter and receiver status
	DRV_UART2_TransferStatus	Returns the transmitter and receiver transfer status
	DRV_UART2_TXBufferIsFull	Returns the status of the transmit buffer
	DRV_UART2_TXBufferSizeGet	Returns the size of the transmit buffer
	DRV_UART3_RXBufferIsEmpty	Returns the status of the receive buffer
	DRV_UART3_RXBufferSizeGet	Returns the size of the receive buffer
	DRV_UART3_Status	Returns the transmitter and receiver status
	DRV_UART3_TransferStatus	Returns the transmitter and receiver transfer status
	DRV_UART3_TXBufferIsFull	Returns the status of the transmit buffer
	DRV_UART3_TXBufferSizeGet	Returns the size of the transmit buffer
	DRV_UART4_RXBufferIsEmpty	Returns the status of the receive buffer
	DRV_UART4_RXBufferSizeGet	Returns the size of the receive buffer
	DRV_UART4_Status	Returns the transmitter and receiver status
	DRV_UART4_TransferStatus	Returns the transmitter and receiver transfer status
	DRV_UART4_TXBufferIsFull	Returns the status of the transmit buffer
	DRV_UART4_TXBufferSizeGet	Returns the size of the transmit buffer

Description

1.2.3.4.1 DRV_UART1_RXBufferIsEmpty Function

Returns the status of the receive buffer

File

drv_uart1.h

Syntax

```
bool DRV_UART1_RXBufferIsEmpty();
```

Returns

True if the receive buffer is empty False if the receive buffer is not empty

Description

This routine returns if the receive buffer is empty or not.

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART1_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART1_TransferStatus ( ) ;
    if (!DRV_UART1_RXBufferIsEmpty())
    {
        numBytes += DRV_UART1_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Function

bool DRV_UART1_RXBufferIsEmpty (void)

1.2.3.4.2 DRV_UART1_RXBufferSizeGet Function

Returns the size of the receive buffer

File

drv_uart1.h

Syntax

```
unsigned int DRV_UART1_RXBufferSizeGet();
```

Returns

Size of receive buffer.

Description

This routine returns the size of the receive buffer.

Example

```
const uint8_t readBuffer[5];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART1__InitializerDefault();

while(size < readbufferLen)
{
    DRV_UART1_TasksRX ( );
    size = DRV_UART1_RXBufferSizeGet();
}
numBytes = DRV_UART1_Read ( readBuffer , readbufferLen ) ;
```

Function

unsigned int DRV_UART1_RXBufferSizeGet (void)

1.2.3.4.3 DRV_UART1_Status Function

Returns the transmitter and receiver status

File

drv_uart1.h

Syntax

```
DRV_UART1_STATUS DRV_UART1_Status();
```

Returns

A DRV_UART1_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART1_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function

Example

```
    while(!(DRV_UART1_Status() & DRV_UART1_TX_COMPLETE ))
    {
       // Wait for the tranmission to complete
    }
```

Function

DRV_UART1_STATUS DRV_UART1_Status (void)

1.2.3.4.4 DRV_UART1_TransferStatus Function

Returns the transmitter and receiver transfer status

File

drv_uart1.h

Syntax

```
DRV_UART1_TRANSFER_STATUS DRV_UART1_TransferStatus();
```

Returns

A DRV_UART1_TRANSFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART1_TRANSFER_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART1_InitializerDefault function should have been called before calling this function

Example

Refer to DRV_UART1_Read and DRV_UART1_Write for example

Function

DRV_UART1_TRANSFER_STATUS DRV_UART1_TransferStatus (void)

1.2.3.4.5 DRV_UART1_TXBufferIsFull Function

Returns the status of the transmit buffer

File

drv_uart1.h

Syntax

```
bool DRV_UART1_TXBufferIsFull();
```

Returns

True if the transmit buffer is full False if the transmit buffer is not full

Description

This routine returns if the transmit buffer is full or not.

Example

Refer to DRV_UART1_InitializerDefault() for example.

Function

bool DRV_UART1_TXBufferIsFull (void)

1.2.3.4.6 DRV_UART1_TXBufferSizeGet Function

Returns the size of the transmit buffer

File

drv_uart1.h

Syntax

```
unsigned int DRV_UART1_TXBufferSizeGet();
```

Returns

Size of transmit buffer.

Description

This routine returns the size of the transmit buffer.

Example

Refer to DRV_UART1_InitializerDefault(); for example.

Function

unsigned int DRV_UART1_TXBufferSizeGet (void)

1.2.3.4.7 DRV_UART2_RXBufferIsEmpty Function

Returns the status of the receive buffer

File

drv_uart2.h

Syntax

```
bool DRV_UART2_RXBufferIsEmpty();
```

Returns

True if the receive buffer is empty False if the receive buffer is not empty

Description

This routine returns if the receive buffer is empty or not.

Example

```
char                     myBuffer[MY_BUFFER_SIZE];
unsigned int             numBytes;
DRV_UART2_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART2_TransferStatus ( ) ;
    if (!DRV_UART2_RXBufferIsEmpty())
    {
        numBytes += DRV_UART2_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
}
```

Function

bool DRV_UART2_RXBufferIsEmpty (void)

1.2.3.4.8 DRV_UART2_RXBufferSizeGet Function

Returns the size of the receive buffer

File

drv_uart2.h

Syntax

```
unsigned int DRV_UART2_RXBufferSizeGet();
```

Returns

Size of receive buffer.

Description

This routine returns the size of the receive buffer.

Example

```
const uint8_t readBuffer[5];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART2__InitializerDefault();

while(size < readbufferLen)
{
    DRV_UART2_TasksRX ( );
    size = DRV_UART2_RXBufferSizeGet();
}
numBytes = DRV_UART2_Read ( readBuffer , readbufferLen ) ;
```

Function

unsigned int DRV_UART2_RXBufferSizeGet (void)

1.2.3.4.9 DRV_UART2_Status Function

Returns the transmitter and receiver status

File

drv_uart2.h

Syntax

```
DRV_UART2_STATUS DRV_UART2_Status();
```

Returns

A DRV_UART2_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART2_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function

Example

```
    while(!(DRV_UART2_Status() & DRV_UART2_TX_COMPLETE ))
    {
       // Wait for the tranmission to complete
    }
```

Function

DRV_UART2_STATUS DRV_UART2_Status (void)

1.2.3.4.10 DRV_UART2_TransferStatus Function

Returns the transmitter and receiver transfer status

File

drv_uart2.h

Syntax

```
DRV_UART2_TRANSFER_STATUS DRV_UART2_TransferStatus();
```

Returns

A DRV_UART2_TRANSFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART2_TRANSFER_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART2_InitializerDefault function should have been called before calling this function

Example

Refer to DRV_UART2_Read and DRV_UART2_Write for example

Function

DRV_UART2_TRANSFER_STATUS DRV_UART2_TransferStatus (void)

1.2.3.4.11 DRV_UART2_TXBufferIsFull Function

Returns the status of the transmit buffer

File

drv_uart2.h

Syntax

```
bool DRV_UART2_TXBufferIsFull();
```

Returns

True if the transmit buffer is full False if the transmit buffer is not full

Description

This routine returns if the transmit buffer is full or not.

Example

Refer to DRV_UART2_InitializerDefault() for example.

Function

bool DRV_UART2_TXBufferIsFull (void)

1.2.3.4.12 DRV_UART2_TXBufferSizeGet Function

Returns the size of the transmit buffer

File

drv_uart2.h

Syntax

```
unsigned int DRV_UART2_TXBufferSizeGet();
```

Returns

Size of transmit buffer.

Description

This routine returns the size of the transmit buffer.

Example

Refer to DRV_UART2_InitializerDefault(); for example.

Function

unsigned int DRV_UART2_TXBufferSizeGet (void)

1.2.3.4.13 DRV_UART3_RXBufferIsEmpty Function

Returns the status of the receive buffer

File

drv_uart3.h

Syntax

```
bool DRV_UART3_RXBufferIsEmpty();
```

Returns

True if the receive buffer is empty False if the receive buffer is not empty

Description

This routine returns if the receive buffer is empty or not.

Example

```
char                    myBuffer[MY_BUFFER_SIZE];
unsigned int            numBytes;
DRV_UART3_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART3_TransferStatus ( ) ;
    if (!DRV_UART3_RXBufferIsEmpty())
    {
        numBytes += DRV_UART3_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
        {
            break;
        }
    }
    else
    {
        continue;
    }

    // Do something else...
```

```
}
```

Function

bool DRV_UART3_RXBufferIsEmpty (void)

1.2.3.4.14 DRV_UART3_RXBufferSizeGet Function

Returns the size of the receive buffer

File

drv_uart3.h

Syntax

```
unsigned int DRV_UART3_RXBufferSizeGet();
```

Returns

Size of receive buffer.

Description

This routine returns the size of the receive buffer.

Example

```
const uint8_t readBuffer[5];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART3__InitializerDefault();

while(size < readbufferLen)
{
    DRV_UART3_TasksRX ( );
    size = DRV_UART3_RXBufferSizeGet();
}
numBytes = DRV_UART3_Read ( readBuffer , readbufferLen ) ;
```

Function

unsigned int DRV_UART3_RXBufferSizeGet (void)

1.2.3.4.15 DRV_UART3_Status Function

Returns the transmitter and receiver status

File

drv_uart3.h

Syntax

```
DRV_UART3_STATUS DRV_UART3_Status();
```

Returns

A DRV_UART3_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART3_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function

Example

```
while(!(DRV_UART3_Status() & DRV_UART3_TX_COMPLETE ))
{
   // Wait for the tranmission to complete
}
```

Function

DRV_UART3_STATUS DRV_UART3_Status (void)

1.2.3.4.16 DRV_UART3_TransferStatus Function

Returns the transmitter and receiver transfer status

File

drv_uart3.h

Syntax

```
DRV_UART3_TRANSFER_STATUS DRV_UART3_TransferStatus();
```

Returns

A DRV_UART3_TRANSFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART3_TRANSFER_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART3_InitializerDefault function should have been called before calling this function

Example

Refer to DRV_UART3_Read and DRV_UART3_Write for example

Function

DRV_UART3_TRANSFER_STATUS DRV_UART3_TransferStatus (void)

1.2.3.4.17 DRV_UART3_TXBufferIsFull Function

Returns the status of the transmit buffer

File

drv_uart3.h

Syntax

```
bool DRV_UART3_TXBufferIsFull();
```

Returns

True if the transmit buffer is full False if the transmit buffer is not full

Description

This routine returns if the transmit buffer is full or not.

Example

Refer to DRV_UART3_InitializerDefault() for example.

Function

bool DRV_UART3_TXBufferIsFull (void)

1.2.3.4.18 DRV_UART3_TXBufferSizeGet Function

Returns the size of the transmit buffer

File

drv_uart3.h

Syntax

```
unsigned int DRV_UART3_TXBufferSizeGet();
```

Returns

Size of transmit buffer.

Description

This routine returns the size of the transmit buffer.

Example

Refer to DRV_UART3_InitializerDefault(); for example.

Function

unsigned int DRV_UART3_TXBufferSizeGet (void)

1.2.3.4.19 DRV_UART4_RXBufferIsEmpty Function

Returns the status of the receive buffer

File

drv_uart4.h

Syntax

```
bool DRV_UART4_RXBufferIsEmpty();
```

Returns

True if the receive buffer is empty False if the receive buffer is not empty

Description

This routine returns if the receive buffer is empty or not.

Example

```
char                     myBuffer[MY_BUFFER_SIZE];
unsigned int             numBytes;
DRV_UART4_TRANSFER_STATUS status ;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.

numBytes = 0;
while( numBytes < MY_BUFFER_SIZE);
{
    status = DRV_UART4_TransferStatus ( ) ;
    if (!DRV_UART4_RXBufferIsEmpty())
    {
        numBytes += DRV_UART4_Read( myBuffer + numBytes, MY_BUFFER_SIZE - numBytes )  ;
        if(numBytes < readbufferLen)
        {
            continue;
        }
        else
```

```
            {
                break;
            }
        }
        else
        {
            continue;
        }

        // Do something else...
    }
```

Function

bool DRV_UART4_RXBufferIsEmpty (void)

1.2.3.4.20 DRV_UART4_RXBufferSizeGet Function

Returns the size of the receive buffer

File

drv_uart4.h

Syntax

```
unsigned int DRV_UART4_RXBufferSizeGet();
```

Returns

Size of receive buffer.

Description

This routine returns the size of the receive buffer.

Example

```
const uint8_t readBuffer[5];
unsigned int size, numBytes = 0;
unsigned int readbufferLen = sizeof(readBuffer);
DRV_UART4__InitializerDefault();

while(size < readbufferLen)
{
    DRV_UART4_TasksRX ( );
    size = DRV_UART4_RXBufferSizeGet();
}
numBytes = DRV_UART4_Read ( readBuffer , readbufferLen ) ;
```

Function

unsigned int DRV_UART4_RXBufferSizeGet (void)

1.2.3.4.21 DRV_UART4_Status Function

Returns the transmitter and receiver status

File

drv_uart4.h

Syntax

```
DRV_UART4_STATUS DRV_UART4_Status();
```

Returns

A DRV_UART4_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART4_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function

Example

```
while(!(DRV_UART4_Status() & DRV_UART4_TX_COMPLETE ))
{
   // Wait for the tranmission to complete
}
```

Function

DRV_UART4_STATUS DRV_UART4_Status (void)

1.2.3.4.22 DRV_UART4_TransferStatus Function

Returns the transmitter and receiver transfer status

File

drv_uart4.h

Syntax

```
DRV_UART4_TRANSFER_STATUS DRV_UART4_TransferStatus();
```

Returns

A DRV_UART4_TRANSFER_STATUS value describing the current status of the transfer.

Description

This returns the transmitter and receiver transfer status.

Remarks

The returned status may contain a value with more than one of the bits specified in the DRV_UART4_TRANSFER_STATUS enumeration set. The caller should perform an "AND" with the bit of interest and verify if the result is non-zero (as shown in the example) to verify the desired status bit.

Preconditions

DRV_UART4_InitializerDefault function should have been called before calling this function

Example

Refer to DRV_UART4_Read and DRV_UART4_Write for example

Function

DRV_UART4_TRANSFER_STATUS DRV_UART4_TransferStatus (void)

1.2.3.4.23 DRV_UART4_TXBufferIsFull Function

Returns the status of the transmit buffer

File

drv_uart4.h

Syntax

```
bool DRV_UART4_TXBufferIsFull();
```

Returns

True if the transmit buffer is full False if the transmit buffer is not full

Description

This routine returns if the transmit buffer is full or not.

Example

Refer to DRV_UART4_InitializerDefault() for example.

Function

bool DRV_UART4_TXBufferIsFull (void)

1.2.3.4.24 DRV_UART4_TXBufferSizeGet Function

Returns the size of the transmit buffer

File

drv_uart4.h

Syntax

```
unsigned int DRV_UART4_TXBufferSizeGet();
```

Returns

Size of transmit buffer.

Description

This routine returns the size of the transmit buffer.

Example

Refer to DRV_UART4_InitializerDefault(); for example.

Function

unsigned int DRV_UART4_TXBufferSizeGet (void)

1.3 SPI Driver

This library provides an interface to manage the Serial Peripheral Interface (SPI) module on the Microchip family of microcontrollers in different modes of operation.

Description

Overview

The SPI module is a full duplex synchronous serial interface useful for communicating with other peripherals or microcontrollers in master/slave relationship and it can transfer data over short distances at high speeds. The peripheral devices may be serial EEPROMs, shift registers, display drivers, analog-to-digital converters, etc. The SPI module is compatible with Motorola's SPI and SIOP interfaces.

During data transfer devices can work either in master or in Slave mode. The source of synchronization is the system clock, which is generated by the master. The SPI module allows to connect one or more slave devices to a single master device via the same bus.

The SPI serial interface consists of four pins, which are further sub-divided into data and control lines:

Data lines:

• MOSI – Master Data Output, Slave Data Input

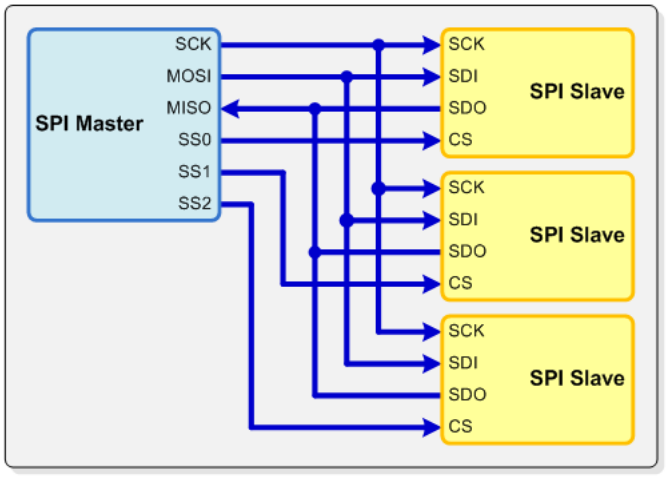
• MISO – Master Data Input, Slave Data Output

Control lines:

• SCLK – Serial Clock

• /SS – Slave Select (no addressing)

SPI Master-Slave Relationship



1.3.1 Using Driver

Module

SPI Driver

Description

This topic describes the basic architecture of the SPI Driver Library and provides information and examples on how to use it.

Interface Header File: drv_spi.h

The interface to the SPI Driver library is defined in the "drv_spi.h" header file. Any C language source (.c) file that uses the SPI Driver library should include this header.

The library interface routines are divided into various subsections, each of the sub section addresses one of the blocks or the overall operation of the SPI module.

Library Interface Section	Description
Initialization	Provides module initialization, deinitialization and setup functions
Data Transfer Functions	Provides data transfer functions available in the configuration.
Configuration	Provides driver configuration macros
Data Types and Constants	Provides data types and constants

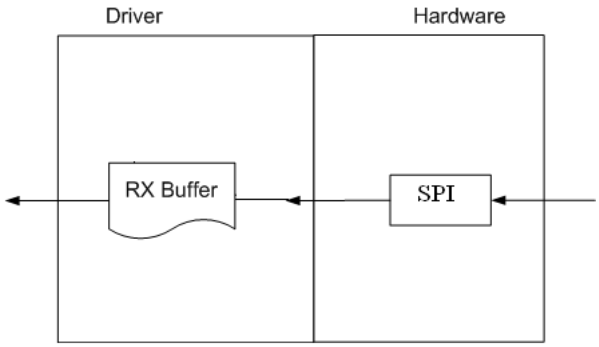
1.3.1.1 Abstraction Model

Different types of SPIs are available on Microchip microcontrollers. Some have an internal buffer mechanism and some do not. The buffer depth varies across part families. The SPI driver abstracts out these differences and provides a unified model for data transfer across different types of SPIs available.

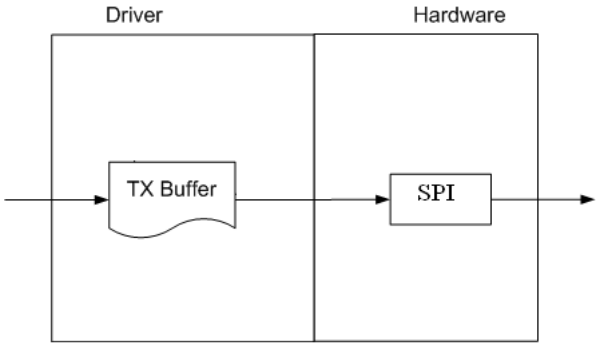
Both transmitter and receiver provides a buffer in the driver which transmits and receives data to/from the hardware. The SPI driver provides a set of interfaces to perform the read and the write.

The diagrams below illustrates the model used by the SPI driver for transmitter and receiver.

Receiver Abstraction Model



Transmitter Abstraction Model



1.3.2 Configuring the Driver

Macros

Name	Description
DRV_SPI_CONFIG_CHANNEL_1_ENABLE	Enable SPI channel 1
DRV_SPI_CONFIG_CHANNEL_2_ENABLE	Enable SPI channel 2
DRV_SPI_CONFIG_CHANNEL_3_ENABLE	Enable SPI channel 3
DRV_SPI_CONFIG_CHANNEL_4_ENABLE	Enable SPI channel 4
DRV_SPI_CONFIG_ENHANCED_BUFFER_DISABLE	Disable the SPI Enhanced Buffer Mode if the specific driver implementation supports it.

Module

SPI Driver

Description

1.3.2.1 DRV_SPI_CONFIG_CHANNEL_1_ENABLE Macro

File

drv_spi_config_template.h

Syntax

```
#define DRV_SPI_CONFIG_CHANNEL_1_ENABLE
```

Description

Enable SPI channel 1

1.3.2.2 DRV_SPI_CONFIG_CHANNEL_2_ENABLE Macro

File

drv_spi_config_template.h

Syntax

```
#define DRV_SPI_CONFIG_CHANNEL_2_ENABLE
```

Description

Enable SPI channel 2

1.3.2.3 DRV_SPI_CONFIG_CHANNEL_3_ENABLE Macro

File

drv_spi_config_template.h

Syntax

```
#define DRV_SPI_CONFIG_CHANNEL_3_ENABLE
```

Description

Enable SPI channel 3

1.3.2.4 DRV_SPI_CONFIG_CHANNEL_4_ENABLE Macro

File

drv_spi_config_template.h

Syntax

```
#define DRV_SPI_CONFIG_CHANNEL_4_ENABLE
```

Description

Enable SPI channel 4

1.3.2.5 DRV_SPI_CONFIG_ENHANCED_BUFFER_DISABLE Macro

File

drv_spi_config_template.h

Syntax

```
#define DRV_SPI_CONFIG_ENHANCED_BUFFER_DISABLE
```

Description

Disable the SPI Enhanced Buffer Mode if the specific driver implementation supports it.

1.3.3 Driver Interface

Module

SPI Driver

Description

1.3.3.1 Initialization and Setup Functions

Functions

	Name	Description
	DRV_SPI_Deinitialize	Deinitializes the SPI instance specified by the channel parameter
	DRV_SPI_Initialize	Initializes the SPI instance specified by the channel of the initialization structure.

	DRV_SPI_Lock	Locks the SPI instance specified using the channel parameter
	DRV_SPI_Unlock	Unlocks the SPI instance specified by channel parameter

Description

1.3.3.1.1 DRV_SPI_Deinitialize Function

Deinitializes the SPI instance specified by the channel parameter

File

drv_spi.h

Syntax

```
void DRV_SPI_Deinitialize(uint8_t channel);
```

Returns

None.

Description

This routine deinitializes the spi driver instance specified by the channel parameter.

Remarks

This routine must be called before any other SPI routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Example

```
uint8_t            myChannel = 2;

DRV_SPI_Deinitialize(myChannel);
```

Parameters

Parameters	Description
uint8_t channel	SPI instance which needs to be deinitialized.

Function

void DRV_SPI_Deinitialize (uint8_t channel)

1.3.3.1.2 DRV_SPI_Initialize Function

Initializes the SPI instance specified by the channel of the initialization structure.

File

drv_spi.h

Syntax

```
void DRV_SPI_Initialize(DRV_SPI_INIT_DATA * pData);
```

Returns

None.

Description

This routine initializes the spi driver instance specified by the channel of the initialization structure making it ready for clients to lock and use it.

Remarks

This routine must be called before any other SPI routine is called. This routine should only be called once during system initialization.

Preconditions

None.

Example

```
uint16_t           myBuffer[MY_BUFFER_SIZE];
unsigned int       total;
uint8_t            myChannel = 2;
DRV_SPI_INIT_DATA  spiInitData = {2, 3, 7, 0, SPI_BUS_MODE_3, 0};

DRV_SPI_Initialize(&spiInitData);
DRV_SPI_Lock(myChannel);

total = 0;
do
{
    total  += DRV_SPI_PutBuffer( myChannel, &myBuffer[total], MY_BUFFER_SIZE - total );

    // Do something else...

} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
DRV_SPI_INIT_DATA * pData	SPI initialization structure.

Function

void DRV_SPI_Initialize (DRV_SPI_INIT_DATA *pData)

1.3.3.1.3 DRV_SPI_Lock Function

Locks the SPI instance specified using the channel parameter

File

drv_spi.h

Syntax

```
int DRV_SPI_Lock(uint8_t channel);
```

Returns

Returns the status of the driver usage.

Description

This routine locks the SPI driver instance specified using the channel parameter

Remarks

None.

Preconditions

None.

Example

Refer to DRV_SPI_Initialize() for an example

Parameters

Parameters	Description
uint8_t channel	SPI instance through which the communication needs to happen

Function

bool DRV_SPI_Lock(uint8_t channel)

1.3.3.1.4 DRV_SPI_Unlock Function

Unlocks the SPI instance specified by channel parameter

File

drv_spi.h

Syntax

```
void DRV_SPI_Unlock(uint8_t channel);
```

Returns

None.

Description

This routine unlocks the SPI driver instance specified by channel parameter making it ready for other clients to lock and use it.

Remarks

None.

Preconditions

None.

Example

```
uint8_t myChannel = 2;

DRV_SPI_Unlock(myChannel);
```

Parameters

Parameters	Description
uint8_t channel	SPI instance through which the communication needs to happen

Function

void DRV_SPI_Unlock(uint8_t channel)

1.3.3.2 Data Transfer Functions

Functions

	Name	Description
	DRV_SPI_Get	Reads a byte/word of data from SPI from the specified channel
	DRV_SPI_GetBuffer	Reads a buffered data from SPI
	DRV_SPI_Put	Writes a byte/word of data to the SPI to the specified channel
	DRV_SPI_PutBuffer	Writes a data buffer to SPI

Description

1.3.3.2.1 DRV_SPI_Get Function

Reads a byte/word of data from SPI from the specified channel

File

drv_spi.h

Syntax

```
uint8_t DRV_SPI_Get(uint8_t channel);
```

Returns

A data byte received by the driver.

Description

This routine reads a byte/word of data from SPI from the specified channel

Remarks

This is blocking routine.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

Example

```
char            myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;
uint8_t         myChannel = 2;

numBytes = 0;
do
{
    myBuffer[numBytes++] = DRV_SPI_Get(myChannel);
    // Do something else...

} while( numBytes < MY_BUFFER_SIZE);
```

Parameters

Parameters	Description
uint8_t channel	SPI instance through which the communication needs to happen

Function

uint16_t DRV_SPI_Get(uint8_t channel)

1.3.3.2.2 DRV_SPI_GetBuffer Function

Reads a buffered data from SPI

File

drv_spi.h

Syntax

```
void DRV_SPI_GetBuffer(uint8_t channel, uint8_t * data, uint16_t count);
```

Returns

Number of bytes actually read from the read buffer.

Description

This routine reads a buffered data from the SPI.

Remarks

This is blocking routine.

Preconditions

The DRV_SPI_Initialize routine must have been called.

Example

```
uint16_t        myBuffer[MY_BUFFER_SIZE];
unsigned int    total;
uint8_t         myChannel = 2;

total = 0;
do
{
    total  += DRV_SPI_GetBuffer( myChannel, &myBuffer[total], MY_BUFFER_SIZE - total);

    // Do something else...

} while( total < MY_BUFFER_SIZE );
```

Parameters

Parameters	Description
uint8_t channel	SPI instance through which the communication needs to happen
buffer	Buffer into which the data read from the SPI instance will be placed.
numbytes	Total number of bytes that need to be read from the module instance (must be equal to or less than the size of the buffer)

Function

unsigned int DRV_SPI_GetBuffer (uint8_t channel, uint16_t *buffer, unsigned int numbytes)

1.3.3.2.3 DRV_SPI_Put Function

Writes a byte/word of data to the SPI to the specified channel

File

drv_spi.h

Syntax

```
void DRV_SPI_Put(uint8_t channel, uint8_t data);
```

Returns

None.

Description

This routine writes a byte/word of data to the SPI to the specified channel

Remarks

This is blocking routine.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

Example

```
uint16_t        myBuffer[MY_BUFFER_SIZE];
unsigned int    numBytes;
uint8_t         myChannel = 2;

// Pre-initialize myBuffer with MY_BUFFER_SIZE bytes of valid data.
```

```
numBytes = 0;
while( numBytes < MY_BUFFER_SIZE )
{
    // DRV_SPI_Put API returns data in any case, upto the user to use it
    DRV_SPI_Put( myChannel, myBuffer[numBytes++] );

    // Do something else...
}
```

Parameters

Parameters	Description
uint8_t channel	SPI instance through which the communication needs to happen
buffer	Data byte/word to write to the SPI

Function

void DRV_SPI_Put(uint8_t channel, const uint16_t buffer)

1.3.3.2.4 DRV_SPI_PutBuffer Function

Writes a data buffer to SPI

File

drv_spi.h

Syntax

```
void DRV_SPI_PutBuffer(uint8_t channel, uint8_t * data, uint16_t count);
```

Returns

Number of bytes actually written to the SPI

Description

This routine writes a buffered data to SPI.

Remarks

This is a blocking routine.

Preconditions

The DRV_SPI_Initialize routine must have been called for the specified SPI driver instance.

Example

Refer to DRV_SPI_Initialize() for an example

Parameters

Parameters	Description
uint8_t channel	SPI instance through which the communication needs to happen
buffer	Buffer containing the data write to the SPI instance
numbytes	Total number of bytes that to write to the SPI instance (must be equal to or less than the size of the buffer)

Function

void DRV_SPI_PutBuffer(uint8_t channel, const uint16_t *buffer, const unsigned int numbytes)

1.3.3.3 Data Types and Constants

Enumerations

Name	Description
SPI_BUS_MODES	Specifies the SPI modes which can be used in the initialization structure to initialize the SPI for operation.

Structures

Name	Description
DRV_SPI_INIT_DATA	Specifies the members which can be adjusted to allow the SPI to be initialized for each instance of SPI.

Description

1.3.3.3.1 DRV_SPI_INIT_DATA Structure

Specifies the members which can be adjusted to allow the SPI to be initialized for each instance of SPI.

File

drv_spi.h

Syntax

```
typedef struct {
  int channel;
  int baudRate;
  int dummy;
  int primaryPrescale;
  int secondaryPrescale;
  uint8_t divider;
  char cke;
  SPI_BUS_MODES spibus_mode;
  char mode;
} DRV_SPI_INIT_DATA;
```

Members

Members	Description
int channel;	Channel for the SPI communication
int baudRate;	Baud rate for the SPI communication
int primaryPrescale;	Primary and Secondary prescalers control the SPI frequency
char cke;	Clock Edge Selection Bits
SPI_BUS_MODES spibus_mode;	One of SPI Bus mode as specified SPI_BUS_MODES
char mode;	Select between 8 and 16 bit communication

Description

SPI Initialization structure

1.3.3.3.2 SPI_BUS_MODES Enumeration

Specifies the SPI modes which can be used in the initialization structure to initialize the SPI for operation.

File

drv_spi.h

Syntax

```
typedef enum {
  SPI_BUS_MODE_0 = 0x0050,
  SPI_BUS_MODE_1,
  SPI_BUS_MODE_2,
  SPI_BUS_MODE_3
} SPI_BUS_MODES;
```

Members

Members	Description
SPI_BUS_MODE_0 = 0x0050	smp = 0, ckp = 0
SPI_BUS_MODE_1	smp = 1, ckp = 0
SPI_BUS_MODE_2	smp = 0, ckp = 1
SPI_BUS_MODE_3	smp = 1, ckp = 1

Description

SPI Modes Enumeration

Index

A

Abstraction Model 8, 69

C

Configuring the Driver 10, 70

D

Data Transfer 9

Data Transfer Functions 23, 74

Data Types and Constants 39, 78

Driver Interface 23, 71

DRV_SPI_CONFIG_CHANNEL_1_ENABLE 70

DRV_SPI_CONFIG_CHANNEL_1_ENABLE macro 70

DRV_SPI_CONFIG_CHANNEL_2_ENABLE 70

DRV_SPI_CONFIG_CHANNEL_2_ENABLE macro 70

DRV_SPI_CONFIG_CHANNEL_3_ENABLE 71

DRV_SPI_CONFIG_CHANNEL_3_ENABLE macro 71

DRV_SPI_CONFIG_CHANNEL_4_ENABLE 71

DRV_SPI_CONFIG_CHANNEL_4_ENABLE macro 71

DRV_SPI_CONFIG_ENHANCED_BUFFER_DISABLE 71

DRV_SPI_CONFIG_ENHANCED_BUFFER_DISABLE macro 71

DRV_SPI_Deinitialize 72

DRV_SPI_Deinitialize function 72

DRV_SPI_Get 75

DRV_SPI_Get function 75

DRV_SPI_GetBuffer 75

DRV_SPI_GetBuffer function 75

DRV_SPI_INIT_DATA 78

DRV_SPI_INIT_DATA structure 78

DRV_SPI_Initialize 72

DRV_SPI_Initialize function 72

DRV_SPI_Lock 73

DRV_SPI_Lock function 73

DRV_SPI_Put 76

DRV_SPI_Put function 76

DRV_SPI_PutBuffer 77

DRV_SPI_PutBuffer function 77

DRV_SPI_Unlock 74

DRV_SPI_Unlock function 74

DRV_UART1_CONFIG_8E1 11

DRV_UART1_CONFIG_8E1 macro 11

DRV_UART1_CONFIG_8E2 11

DRV_UART1_CONFIG_8E2 macro 11

DRV_UART1_CONFIG_8N1 12

DRV_UART1_CONFIG_8N1 macro 12

DRV_UART1_CONFIG_8N2 12

DRV_UART1_CONFIG_8N2 macro 12

DRV_UART1_CONFIG_8O1 12

DRV_UART1_CONFIG_8O1 macro 12

DRV_UART1_CONFIG_8O2 12

DRV_UART1_CONFIG_8O2 macro 12

DRV_UART1_CONFIG_9N1 13

DRV_UART1_CONFIG_9N1 macro 13

DRV_UART1_CONFIG_9N2 13

DRV_UART1_CONFIG_9N2 macro 13

DRV_UART1_CONFIG_BAUD_RATE 13

DRV_UART1_CONFIG_BAUD_RATE macro 13

DRV_UART1_CONFIG_RX_BYTEQ_LENGTH 13

DRV_UART1_CONFIG_RX_BYTEQ_LENGTH macro 13

DRV_UART1_CONFIG_TX_BYTEQ_LENGTH 14

DRV_UART1_CONFIG_TX_BYTEQ_LENGTH macro 14

DRV_UART1_InitializerDefault 45

DRV_UART1_InitializerDefault function 45

DRV_UART1_Peek 24

DRV_UART1_Peek function 24

DRV_UART1_Read 24

DRV_UART1_Read function 24

DRV_UART1_ReadByte 25

DRV_UART1_ReadByte function 25

DRV_UART1_RXBufferIsEmpty 55

DRV_UART1_RXBufferIsEmpty function 55

DRV_UART1_RXBufferSizeGet 55

DRV_UART1_RXBufferSizeGet function 55

DRV_UART1_Status 56

DRV_UART1_STATUS 39

DRV_UART1_STATUS enumeration 39

DRV_UART1_Status function 56

DRV_UART1_TasksError 45

DRV_UART1_TasksError function 45

DRV_UART1_TasksRX 46

DRV_UART1_TasksRX function 46
DRV_UART1_TasksTX 46
DRV_UART1_TasksTX function 46
DRV_UART1_TRANSFER_STATUS 40
DRV_UART1_TRANSFER_STATUS enumeration 40
DRV_UART1_TransferStatus 56
DRV_UART1_TransferStatus function 56
DRV_UART1_TXBufferIsFull 57
DRV_UART1_TXBufferIsFull function 57
DRV_UART1_TXBufferSizeGet 57
DRV_UART1_TXBufferSizeGet function 57
DRV_UART1_Write 26
DRV_UART1_Write function 26
DRV_UART1_WriteByte 27
DRV_UART1_WriteByte function 27
DRV_UART2_CONFIG_8E1 14
DRV_UART2_CONFIG_8E1 macro 14
DRV_UART2_CONFIG_8E2 14
DRV_UART2_CONFIG_8E2 macro 14
DRV_UART2_CONFIG_8N1 15
DRV_UART2_CONFIG_8N1 macro 15
DRV_UART2_CONFIG_8N2 15
DRV_UART2_CONFIG_8N2 macro 15
DRV_UART2_CONFIG_8O1 15
DRV_UART2_CONFIG_8O1 macro 15
DRV_UART2_CONFIG_8O2 15
DRV_UART2_CONFIG_8O2 macro 15
DRV_UART2_CONFIG_9N1 16
DRV_UART2_CONFIG_9N1 macro 16
DRV_UART2_CONFIG_9N2 16
DRV_UART2_CONFIG_9N2 macro 16
DRV_UART2_CONFIG_BAUD_RATE 16
DRV_UART2_CONFIG_BAUD_RATE macro 16
DRV_UART2_CONFIG_RX_BYTEQ_LENGTH 16
DRV_UART2_CONFIG_RX_BYTEQ_LENGTH macro 16
DRV_UART2_CONFIG_TX_BYTEQ_LENGTH 17
DRV_UART2_CONFIG_TX_BYTEQ_LENGTH macro 17
DRV_UART2_InitializerDefault 47
DRV_UART2_InitializerDefault function 47
DRV_UART2_Peek 28
DRV_UART2_Peek function 28
DRV_UART2_Read 28
DRV_UART2_Read function 28
DRV_UART2_ReadByte 29
DRV_UART2_ReadByte function 29
DRV_UART2_RXBufferIsEmpty 58
DRV_UART2_RXBufferIsEmpty function 58
DRV_UART2_RXBufferSizeGet 58
DRV_UART2_RXBufferSizeGet function 58
DRV_UART2_Status 59
DRV_UART2_STATUS 40
DRV_UART2_STATUS enumeration 40
DRV_UART2_Status function 59
DRV_UART2_TasksError 48
DRV_UART2_TasksError function 48
DRV_UART2_TasksRX 48
DRV_UART2_TasksRX function 48
DRV_UART2_TasksTX 49
DRV_UART2_TasksTX function 49
DRV_UART2_TRANSFER_STATUS 41
DRV_UART2_TRANSFER_STATUS enumeration 41
DRV_UART2_TransferStatus 60
DRV_UART2_TransferStatus function 60
DRV_UART2_TXBufferIsFull 60
DRV_UART2_TXBufferIsFull function 60
DRV_UART2_TXBufferSizeGet 60
DRV_UART2_TXBufferSizeGet function 60
DRV_UART2_Write 30
DRV_UART2_Write function 30
DRV_UART2_WriteByte 31
DRV_UART2_WriteByte function 31
DRV_UART3_CONFIG_8E1 17
DRV_UART3_CONFIG_8E1 macro 17
DRV_UART3_CONFIG_8E2 17
DRV_UART3_CONFIG_8E2 macro 17
DRV_UART3_CONFIG_8N1 18
DRV_UART3_CONFIG_8N1 macro 18
DRV_UART3_CONFIG_8N2 18
DRV_UART3_CONFIG_8N2 macro 18
DRV_UART3_CONFIG_8O1 18
DRV_UART3_CONFIG_8O1 macro 18
DRV_UART3_CONFIG_8O2 18
DRV_UART3_CONFIG_8O2 macro 18
DRV_UART3_CONFIG_9N1 19

DRV_UART3_CONFIG_9N1 macro 19
DRV_UART3_CONFIG_9N2 19
DRV_UART3_CONFIG_9N2 macro 19
DRV_UART3_CONFIG_BAUD_RATE 19
DRV_UART3_CONFIG_BAUD_RATE macro 19
DRV_UART3_CONFIG_RX_BYTEQ_LENGTH 19
DRV_UART3_CONFIG_RX_BYTEQ_LENGTH macro 19
DRV_UART3_CONFIG_TX_BYTEQ_LENGTH 20
DRV_UART3_CONFIG_TX_BYTEQ_LENGTH macro 20
DRV_UART3_InitializerDefault 49
DRV_UART3_InitializerDefault function 49
DRV_UART3_Peek 31
DRV_UART3_Peek function 31
DRV_UART3_Read 32
DRV_UART3_Read function 32
DRV_UART3_ReadByte 33
DRV_UART3_ReadByte function 33
DRV_UART3_RXBufferIsEmpty 61
DRV_UART3_RXBufferIsEmpty function 61
DRV_UART3_RXBufferSizeGet 62
DRV_UART3_RXBufferSizeGet function 62
DRV_UART3_Status 62
DRV_UART3_STATUS 42
DRV_UART3_STATUS enumeration 42
DRV_UART3_Status function 62
DRV_UART3_TasksError 50
DRV_UART3_TasksError function 50
DRV_UART3_TasksRX 50
DRV_UART3_TasksRX function 50
DRV_UART3_TasksTX 51
DRV_UART3_TasksTX function 51
DRV_UART3_TRANSFER_STATUS 42
DRV_UART3_TRANSFER_STATUS enumeration 42
DRV_UART3_TransferStatus 63
DRV_UART3_TransferStatus function 63
DRV_UART3_TXBufferIsFull 63
DRV_UART3_TXBufferIsFull function 63
DRV_UART3_TXBufferSizeGet 64
DRV_UART3_TXBufferSizeGet function 64
DRV_UART3_Write 33
DRV_UART3_Write function 33
DRV_UART3_WriteByte 34
DRV_UART3_WriteByte function 34
DRV_UART4_CONFIG_8E1 20
DRV_UART4_CONFIG_8E1 macro 20
DRV_UART4_CONFIG_8E2 20
DRV_UART4_CONFIG_8E2 macro 20
DRV_UART4_CONFIG_8N1 21
DRV_UART4_CONFIG_8N1 macro 21
DRV_UART4_CONFIG_8N2 21
DRV_UART4_CONFIG_8N2 macro 21
DRV_UART4_CONFIG_8O1 21
DRV_UART4_CONFIG_8O1 macro 21
DRV_UART4_CONFIG_8O2 21
DRV_UART4_CONFIG_8O2 macro 21
DRV_UART4_CONFIG_9N1 22
DRV_UART4_CONFIG_9N1 macro 22
DRV_UART4_CONFIG_9N2 22
DRV_UART4_CONFIG_9N2 macro 22
DRV_UART4_CONFIG_BAUD_RATE 22
DRV_UART4_CONFIG_BAUD_RATE macro 22
DRV_UART4_CONFIG_RX_BYTEQ_LENGTH 22
DRV_UART4_CONFIG_RX_BYTEQ_LENGTH macro 22
DRV_UART4_CONFIG_TX_BYTEQ_LENGTH 23
DRV_UART4_CONFIG_TX_BYTEQ_LENGTH macro 23
DRV_UART4_InitializerDefault 52
DRV_UART4_InitializerDefault function 52
DRV_UART4_Peek 35
DRV_UART4_Peek function 35
DRV_UART4_Read 36
DRV_UART4_Read function 36
DRV_UART4_ReadByte 37
DRV_UART4_ReadByte function 37
DRV_UART4_RXBufferIsEmpty 64
DRV_UART4_RXBufferIsEmpty function 64
DRV_UART4_RXBufferSizeGet 65
DRV_UART4_RXBufferSizeGet function 65
DRV_UART4_Status 65
DRV_UART4_STATUS 43
DRV_UART4_STATUS enumeration 43
DRV_UART4_Status function 65
DRV_UART4_TasksError 52
DRV_UART4_TasksError function 52
DRV_UART4_TasksRX 53

DRV_UART4_TasksRX function 53
DRV_UART4_TasksTX 53
DRV_UART4_TasksTX function 53
DRV_UART4_TRANSFER_STATUS 44
DRV_UART4_TRANSFER_STATUS enumeration 44
DRV_UART4_TransferStatus 66
DRV_UART4_TransferStatus function 66
DRV_UART4_TXBufferIsFull 66
DRV_UART4_TXBufferIsFull function 66
DRV_UART4_TXBufferSizeGet 67
DRV_UART4_TXBufferSizeGet function 67
DRV_UART4_Write 37
DRV_UART4_Write function 37
DRV_UART4_WriteByte 38
DRV_UART4_WriteByte function 38

I

Initialization 9
Initialization and Setup Functions 44, 71

L

Legal Information 7

M

MLA Drivers 6

S

SPI Driver 68
SPI_BUS_MODES 78
SPI_BUS_MODES enumeration 78
Status Functions 54

U

UART Driver 8
Using Driver 8, 68