TECHNICAL NOTE

TN261

The Slave Port Driver

The Rabbit family of microprocessors has hardware for a slave port, allowing a master controller to read and write certain internal registers on the Rabbit. The library, `Slaveport.lib`, implements a complete master/slave protocol for the Rabbit slave port. Sample libraries, `Master_serial.lib` and `Sp_stream.lib` provide serial port and stream-based communication handlers using the slave port protocol. These slave port libraries are located in the folder `/Lib/.../SLAVEPORT/` where you installed Dynamic C.

Slave Port Driver Protocol

Given the variety of embedded system implementations, the protocol for the slave port driver was designed to make the software for the master controller as simple as possible. Each interaction between the master and the slave is initiated by the master. The master has complete control over when data transfers occur and can expect single, immediate responses from the slave.

Overview

1. Master writes to the command register after setting the address register and, optionally, the data register. These registers are internal to the slave.
2. Slave reads the registers that were written by the master.
3. Slave writes to command response register after optionally setting the data register. This also causes the SLAVEATTN line on the Rabbit slave to be pulled low.
4. Master reads response and data registers.
5. Master writes to the slave port status register to clear interrupt line from the slave.

Registers on the Slave

From the point of view of the master, the slave is an I/O device with four register addresses.

Table 1. The slave registers that are accessible by the master

Register Name	Internal Address of Register	Address of Register From Master's Perspective	Register Use
SPD0R	0x20	0	Command and response register
SPD1R	0x21	1	Address register
SPD2R	0x22	2	Optional data register
SPSR	0x23	3	Slave port status register. In this protocol the only bit used is for checking the command response register. Bit 3 is set if the slave has written to SPD0R. It is cleared when the master writes to SPSR, which also deasserts the SLAVEATTN line.

Accessing the same address (0, 1 or 2) uses two different registers, depending on whether the access was a read or a write. In other words, when writing to address 0, the master accesses a different location than when the it reads address 0.

Table 2. What happens when the master accesses a slave register

Register Address	Read	Write
0	Gets command response from slave	Sends command to slave, triggers slave response
1	Not used	Sets channel address to send command to
2	Gets returned data from slave	Sets data byte to send to slave
3	Gets slave port status (see below)	Clears slave response bit (see below)

The status port is a bit field showing which slave port registers have been updated. For the purposes of this protocol, only bit 3 needs to be examined. After sending a command, the master can check bit 3, which is set when the slave writes to the response register. At this point the response and returned data are valid and should be read before sending a new command. Performing a dummy write to the status register will clear this bit, so that it can be set by the next response.

Pin assignments for the Rabbit acting as a slave are as follows:

Table 3. Pin assignments for the Rabbit acting as a slave

Rabbit 2000/3000 Pin	Rabbit 4000 Pins	Function
PE7	PB6	/SCS chip select (active low to read/write slave port)
PB2		/SWR slave write (assert for write cycle)
PB3		/SRD slave read (assert for read cycle)
PB4		SA0 low address bit for slave port registers
PB5		SA1 high address bit for slave registers
PB7		/SLVATTN asserted by slave when it responds to a command. cleared by master write to status register
PA0-PA7		slave port data bus

For more details and read/write signal timing see the microprocessor user's manual for your Rabbit chip.

Polling and Interrupts

Both the slave and the master can use interrupt or polling for the slave. The parameter passed to `SPinit()` determines which one is used. In interrupt mode, the developer can indicate whether the handler functions for the channels are interruptible or non-interruptible.

Communication Channels

The Rabbit slave has 256 configurable channels available for communication. The developer must provide a handler function for each channel that is used. Some basic handlers are available in the library `Slave_Port.lib`. These handlers will be discussed later in this chapter.

When the slave port driver is initialized, a callback table of handler functions is set up. Handler functions are added to the callback table by `SPsetHandler()`.

Functions

`Slave_port.lib` provides the following functions:

```
SPinit()
SPsetHandler()
MyHandler()
SPtick()
SPclose()
```

SPinit

```
int SPinit ( int mode );
```

DESCRIPTION

This function initializes the slave port driver. It sets up the callback tables for the different channels. The slave port driver can be run in either polling mode where `SPtick()` must be called periodically, or in interrupt mode where an ISR is triggered every time the master sends a command. There are two version of interrupt mode. In the first, interrupts are reenabled while the handler function is executing. In the other, the handler function will execute at the same interrupt priority as the driver ISR.

PARAMETERS

mode

`0`: For polling
`1`: For interrupt driven (interruptible handler functions)
`2`: For interrupt driven (non-interruptible handler functions)

RETURN VALUE

`1`: Success
`0`: Failure

LIBRARY

SLAVE_PORT.LIB

SPsetHandler

```
int SPsetHandler ( char address, int (*handler)(), void
   *handler_params );
```

DESCRIPTION

This function sets up a handler function to process incoming commands from the master for a particular slave port address.

PARAMETERS

address	The 8-bit slave port address of the channel that corresponds to the handler function.
handler	Pointer to the handler function. This function must have a particular form, which is described by the function description for `MyHandler()` shown below. Setting this parameter to `NULL` unloads the current handler.
handler_params	Pointer that will be saved and passed to the handler function each time it is called. This allows the handler function to be parameterized for multiple cases.

RETURN VALUE

`1`: Success, the handler was set.
`0`: Failure.

LIBRARY

SLAVE_PORT.LIB

MyHandler

```
int MyHandler ( char command, char data_in, void *params );
```

DESCRIPTION

This function is a developer-supplied function and can have any valid Dynamic C name. Its purpose is to handle incoming commands from a master to one of the 256 channels on the slave port. A handler function must be supplied for every channel that is being used on the slave port.

PARAMETERS

command	This is the received command byte.
data_in	The optional data byte
params	The optional parameters pointer.

RETURN VALUE

This function must return an integer. The low byte must contains the response code and the high byte contains the returned data, if there is any.

LIBRARY

This is a developer-supplied function.

SPtick

```
void SPtick ( void );
```

DESCRIPTION

This function must be called periodically when the slave port is used in polling mode.

LIBRARY

SLAVE_PORT.LIB

SPclose

```
void SPclose( void );
```

DESCRIPTION

This function disables the slave port driver and unloads the ISR if one was used.

LIBRARY

SLAVE_PORT.LIB

Handler Examples

The rest of this technical note describes some useful handlers.

Status Handler

`SPstatusHandler()`, available in `Slave_port.lib`, is an example of a simple handler to report the status of the slave. To set up the function as a handler on slave port address 12, do the following:

```
SPsetHandler (12, SPstatusHandler, &status_char);
```

Sending any command to this handler will cause it to respond with a `1` in the response register and the current value of `status_char` in the data return register.

Serial Port Handler

`Slave_port.lib` contains handlers for all serial ports A, B, C and D on the slave. `Master_serial.lib` contains code for a master using the slave's serial port handler. This library illustrates the general case of implementing the master side of the master/slave protocol.

Commands to the Slave

The following table lists the command numbers (and their descriptions) that the master can send to the slave.

Table 4. Commands from master to slave

Command	Command Description
1	Transmit byte. Byte value is in data register. Slave responds with 1 if the byte was processed or 0 if it was not.
2	Receive byte. Slave responds with 2 if has put a new received byte into the data return register or 0 if there were no bytes to receive.
3	Combined transmit/receive. The response will also be a logical OR of the two command responses.
4	Set baud factor, byte 1 (LSB). The actual baud rate is the baud factor multiplied by 300.
5	Set baud factor, byte 2 (MSB). The actual baud rate is the baud factor multiplied by 300.
6	Set port configuration bits
7	Open port
8	Close port
9	Get errors. Slave responds with 1 if the port is open and can return an error bitfield. The error bits are the same as for the function serAgetErrors() and are put in the data return register by the slave.
10, 11	Returns count of free bytes in the serial port write buffer. The two commands return the LSB and the MSB of the count respectively. The LSB(10) should be read first to latch the count.
12, 13	Returns count of free bytes in the serial port read buffer. The two commands return the LSB and the MSB of the count respectively. The LSB(12) should be read first to latch the count.
14, 15	Returns count of bytes currently in the serial port write buffer. The two commands return the LSB and the MSB of the count respectively. The LSB(14) should be read first to latch the count.
16, 17	Returns count of bytes currently in the serial port write buffer. The two commands return the LSB and the MSB of the count respectively. The LSB(16) should be read first to latch the count.

Slave Side of Protocol

To set up the serial port handler to connect serial port A to channel 5 , do the following:

```
SPsetHandler (5, SPserAhandler, NULL);
```

Master Side of Protocol

The following functions are in `Master_serial.lib`. They are for a master using a serial port handler on a slave.

`cof_MSgetc()`	`MSopen()`
`cof_MSputc()`	`MSputc()`
`cof_MSread()`	`MSrdFree()`
`cof_MSwrite()`	`MSsendCommand()`
`MSclose()`	`MSread()`
`MSgetc()`	`MSwrFree()`
`MSgetError()`	`MSwrite()`

cof_MSgetc

```
int cof_MSgetc( char address );
```

DESCRIPTION

Yields to other tasks until a byte is received from the serial port on the slave.

PARAMETERS

address Slave channel address of the serial handler.

RETURN VALUE

Value of the received character on success.
`-1`: Failure.

LIBRARY

MASTER_SERIAL.LIB

cof_MSputc

```
void cof_MSputc( char address, char ch );
```

DESCRIPTION

Sends a character to the serial port. Yields until character is sent.

PARAMETERS

address	Slave channel address of serial handler.
ch	Character to send.

RETURN VALUE

0: Success, character was sent.
-1: Failure, character was not sent.

LIBRARY

MASTER_SERIAL.LIB

cof_MSread

```
int cof_MSread( char address, char *buffer, int length, unsigned long
   timeout );
```

DESCRIPTION

Reads bytes from the serial port on the slave into the provided buffer. Waits until at least one character has been read. Returns after buffer is full, or `timeout` has expired between reading bytes. Yields to other tasks while waiting for data.

PARAMETERS

address	Slave channel address of serial handler.
buffer	Buffer to store received bytes.
length	Size of buffer.
timeout	Time to wait between bytes before giving up on receiving anymore.

RETURN VALUE

`>0`: Bytes read.
`-1`: Failure.

LIBRARY

`MASTER_SERIAL.LIB`

cof_MSwrite

```
int cof_MSwrite( char address, char *data, int length );
```

DESCRIPTION

Transmits an array of bytes from the serial port on the slave. Yields to other tasks while waiting for write buffer to clear.

PARAMETERS

address	Slave channel address of serial handler.
data	Array to be transmitted.
length	Size of array.

RETURN VALUE

Number of bytes actually written or `-1` if error.

LIBRARY

MASTER_SERIAL.LIB

MSclose

```
int MSclose( char address );
```

DESCRIPTION

Closes a serial port on the slave.

PARAMETERS

address	Slave channel address of serial handler.

RETURN VALUE

`0`: Success.
`-1`: Failure.

LIBRARY

MASTER_SERIAL.LIB

MSgetc

```
int MSgetc( char address );
```

DESCRIPTION

Receives a character from the serial port.

PARAMETERS

address Slave channel address of serial handler.

RETURN VALUE

Value of received character.
-1: No character available.

LIBRARY

MASTER_SERIAL.LIB

MSgetError

```
int MSgetError( char address );
```

DESCRIPTION

Gets bitfield with any current error from the specified serial port on the slave. Error codes are:

SER_PARITY_ERROR
SER_OVERRUN_ERROR

PARAMETERS

address Slave channel address of serial handler.

RETURN VALUE

Number of bytes free: Success.
-1: Failure.

LIBRARY

MASTER_SERIAL.LIB

MSinit

```
int MSinit( int io_bank );
```

DESCRIPTION

Sets up the connection to the slave.

PARAMETERS

io_bank	The I/O bank and chip select pin number for the slave device. This is a number from 0 to 7 inclusive.

RETURN VALUE

1: Success.

LIBRARY

MASTER_SERIAL.LIB

MSopen

```
int MSopen( char address, unsigned long baud );
```

DESCRIPTION

Opens a serial port on the slave, given that there is a serial handler at the specified address on the slave.

PARAMETERS

address	Slave channel address of serial handler.
baud	Baud rate for the serial port on the slave.

RETURN VALUE

1: Baud rate used matches the argument.
0: Different baud rate is being used.
-1: Slave port comm error occurred.

LIBRARY

MASTER_SERIAL.LIB

MSputc

```
int MSputc( char address, char ch );
```

DESCRIPTION

Transmits a single character through the serial port.

PARAMETERS

address	Slave channel address of serial handler.
ch	Character to send.

RETURN VALUE

1: Character sent.
0: Transmit buffer is full or locked.

LIBRARY

MASTER_SERIAL.LIB

MSrdFree

```
int MSrdFree( char address );
```

DESCRIPTION

Gets the number of bytes available in the specified serial port read buffer on the slave.

PARAMETERS

address	Slave channel address of serial handler.

RETURN VALUE

Number of bytes free: Success.
-1: Failure.

LIBRARY

MASTER_SERIAL.LIB

MSsendCommand

```
int MSsendCommand( char address, char command, char data, char
   *data_returned, unsigned long timeout );
```

DESCRIPTION

Sends a single command to the slave and gets a response. This function also serves as a general example of how to implement the master side of the slave protocol.

PARAMETERS

address	Slave channel address to send command to.
command	Command to send to the slave. For a list of valid commands see Table 4.
data	Data byte to be sent to the slave.
data_returned	Address of variable to place data returned by the slave.
timeout	Time to wait before giving up on slave response.

RETURN VALUE

≥0: Response code.
-1: Timeout occured before response.
-2: Nothing at that address (response = 0xff).

LIBRARY

MASTER_SERIAL.LIB

MSread

```
int MSread( char address, char *buffer, int size, unsigned long
   timeout );
```

DESCRIPTION

Receives bytes from the serial port on the slave.

PARAMETERS

address	Slave channel address of serial handler.
buffer	Array to put received data into.
size	Size of array (max bytes to be read).
timeout	Time to wait between characters before giving up on receiving any more.

RETURN VALUE

The number of bytes read into the buffer (behaves like `serXread()`).

LIBRARY

MASTER_SERIAL.LIB

MSwrFree

```
int MSwrFree( char address );
```

DESCRIPTION

Gets the number of bytes available in the specified serial port write buffer on the slave.

PARAMETERS

address	Slave channel address of serial handler.

RETURN VALUE

Number of bytes free: Success.
`-1`: Failure.

LIBRARY

MASTER_SERIAL.LIB

MSwrite

```
int MSwrite( char address, char *data, int length );
```

DESCRIPTION

Sends an array of bytes out the serial port on the slave (behaves like `serXwrite()`).

PARAMETERS

address	Slave channel address of serial handler.
data	Array of bytes to send.
length	Size of array.

RETURN VALUE

Number of bytes actually sent.

LIBRARY

MASTER_SERIAL.LIB

Sample Program for Master

This sample program, /Samples/SlavePort/master_demo.c, treats the slave like a serial port.

```
#use "master_serial.lib"
#define SP_CHANNEL 0x42

char* const test_str = "Hello There";

main(){
   char buffer[100];
   int read_length;

   MSinit(0);

   // comment this line out if talking to a stream handler
   printf("open returned:0x%x\n", MSopen(SP_CHANNEL, 9600));

   while(1)
   {
      costate
      {
         wfd{cof_MSwrite(SP_CHANNEL, test_str, strlen(test_str));}
         wfd{cof_MSwrite(SP_CHANNEL, test_str, strlen(test_str));}
      }
      costate
      {
         wfd{ read_length = cof_MSread(SP_CHANNEL, buffer, 99, 10);
}
         if(read_length > 0)
         {
               buffer[read_length] = 0; //null terminator
               printf("Read:%s\n", buffer);
         }
         else if(read_length < 0)
         {
            printf("Got read error: %d\n", read_length);
         }
         printf("wrfree = %d\n", MSwrFree(SP_CHANNEL));
      }
   }
}
```

Byte Stream Handler

The library, SP_STREAM.LIB, implements a byte stream over the slave port. If the master is a Rabbit, the functions in MASTER_SERIAL.LIB can be used to access the stream as though it came from a serial port on the slave.

Slave Side of Stream Channel

To set up the function SPShandler() as the byte stream handler, do the following:

```
SPsetHandler (10, SPShandler, stream_ptr);
```

This function sets up the stream to use channel 10 on the slave.

A sample program described in the section titled "Byte Stream Sample Program" shows how to set up and initialize the circular buffers. An internal data structure, SPStream, keeps track of the buffers and a pointer to it is passed to SPsetHandler() and some of the auxiliary functions that supports the byte stream handler. This is also shown in the sample program.

Functions

These are the auxiliary functions that support the stream handler function, SPShandler().

```
cbuf_init()                SPSwrite()
cof_SPSread()              SPSwrFree()
cof_SPSwrite()             SPSrdFree()
SPSinit()                  SPSwrUsed()
SPSread()
```

---

cbuf_init

```
void cbuf_init( char *circularBuffer, int dataSize );
```

DESCRIPTION

This function initializes a circular buffer.

PARAMETERS

circularBuffer	The circular buffer to initialize.
dataSize	Size available to data. The size must be 9 bytes more than the number of bytes needed for data. This is for internal book-keeping.

LIBRARY

RS232.LIB

cof_SPSread

```
int cof_SPSread( SPStream *stream, void *data, int length, unsigned
   long tmout );
```

DESCRIPTION

Reads `length` bytes from the slave port input buffer or until `tmout` milliseconds transpires between bytes after the first byte is read. It will yield to other tasks while waiting for data. This function is non-reentrant.

PARAMETERS

stream	Pointer to the stream state structure.
data	Structure to read from slave port buffer.
length	Number of bytes to read.
tmout	Maximum wait in milliseconds for any byte from previous one.

RETURN VALUE

The number of bytes read from the buffer.

LIBRARY

SP_STREAM.LIB

cof_SPSwrite

```
int cof_SPSwrite( SPStream *stream, void *data, int length );
```

DESCRIPTION

Transmits `length` bytes to slave port output buffer.This function is non-reentrant.

PARAMETERS

stream	Pointer to the stream state structure.
data	Structure to write to slave port buffer.
length	Number of bytes to write.

RETURN VALUE

The number of bytes successfully written to slave port.

LIBRARY

SP_STREAM.LIB

SPSinit

```
void SPSinit( void );
```

DESCRIPTION

Initializes the circular buffers used by the stream handler.

LIBRARY

SP_STREAM.LIB

SPSread

```
int SPSread( SPStream *stream, void *data, int length, unsigned long
   tmout );
```

DESCRIPTION

Reads `length` bytes from the slave port input buffer or until `tmout` milliseconds transpires between bytes. If no data is available when this function is called, it will return immediately. This function will call `SPtick()` if the slave port is in polling mode.

This function is non-reentrant.

PARAMETERS

stream	Pointer to the stream state structure.
data	Buffer to read received data into.
length	Maximum number of bytes to read.
tmout	Time to wait between received bytes before returning.

RETURN VALUE

Number of bytes read into the data buffer

LIBRARY

SP_STREAM.LIB

SPSwrite

```
int SPSwrite( SPSream *stream, void *data, int length );
```

DESCRIPTION

This function transmits length bytes to slave port output buffer. If the slave port is in polling mode, this function will call `SPtick()` while waiting for the output buffer to empty. This function is non-reentrant.

PARAMETERS

stream	Pointer to the stream state structure.
data	Bytes to write to stream.
length	Size of write buffer.

RETURN VALUE

Number of bytes written into the data buffer.

LIBRARY

SP_STREAM.LIB

SPSwrFree

```
int SPSwrFree( void );
```

DESCRIPTION

Returns number of free bytes in the stream write buffer.

RETURN VALUE

Space available in the stream write buffer.

LIBRARY

SP_STREAM.LIB

SPSrdFree

```
int SPSrdFree( void );
```

DESCRIPTION

Returns the number of free bytes in the stream read buffer.

RETURN VALUE

Space available in the stream read buffer.

LIBRARY

SP_STREAM.LIB

SPSwrUsed

```
int SPSwrUsed( void );
```

DESCRIPTION

Returns the number of bytes currently in the stream write buffer.

RETURN VALUE

Number of bytes currently in the stream write buffer.

LIBRARY

SP_STREAM.LIB

SPSrdUsed

```
int SPSrdUsed( void );
```

DESCRIPTION

Returns the number of bytes currently in the stream read buffer.

RETURN VALUE

Number of bytes currently in the stream read buffer.

LIBRARY

SP_STREAM.LIB

Byte Stream Sample Program

This program, /Samples/SlavePort/Slave_Demo.c, runs on a slave and implements a byte stream over the slave port.

```
#class auto

#use "slave_port.lib"
#use "sp_stream.lib"

#define STREAM_BUFFER_SIZE 31

main()
{
   char buffer[10];
   int bytes_read;

   SPStream stream;

   // Circular buffers need 9 bytes for bookkeeping.
   char stream_inbuf[STREAM_BUFFER_SIZE + 9];
   char stream_outbuf[STREAM_BUFFER_SIZE + 9];

   SPStream *stream_ptr;

   // setup buffers
   cbuf_init(stream_inbuf, STREAM_BUFFER_SIZE);
   stream.inbuf = stream_inbuf;
   cbuf_init(stream_outbuf, STREAM_BUFFER_SIZE);
   stream.outbuf = stream_outbuf;

   stream_ptr = &stream;

   SPinit(1);

   SPsetHandler(0x42, SPShandler, stream_ptr);

   while(1)
   {
      bytes_read = SPSread(stream_ptr, buffer, 10, 10);
      if(bytes_read)
      {
         SPSwrite(stream_ptr, buffer, bytes_read);
      }
   }
}
```