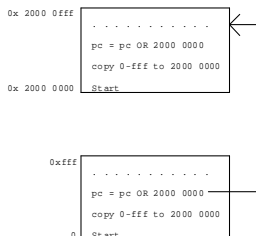


# STR7int firmware rev. 1.2d description

## Startup

At startup the first half of sector 0 bank 0 of the Flash memory (from 0 to 0xfff) is copied in RAM, then a jump is done on the current Program Counter value ORed with 0x2000 0000, that is the start address of the RAM.



The execution follows in RAM and the main loop is executed, that waits a valid message from the serial line and then executes the command contained.

The basic operations, memory read and write and flash programming, are handled by code that is inside the copied block (0 to fff), instead the routines that handle the serial lines are stored into the Flash memory starting from 1fff, and are called from there.

Working in RAM give the possibility to program every part of the Flash memory, otherwise is only possible to program sectors that are not in the same bank where we are fetching code.

## RS232 messages

### RS232 settings

The STR7int's serial port is set with the following parameters:

- Speed: **115200** baud
- Data length: **8** bits
- Parity: **odd**
- Stop bits: **1**

### RS232 operation

The incoming data from the RS232 line are stored, by an interrupt routine, in a circular buffer 256 bytes long. The main program loops continuously checking if data is present in the circular buffer: if yes, then one byte is read and the Sync bit value is check.

If Sync is 1 then the buffer that stores the current message is resetted otherwise the 7 other data bits of the byte read will be added to the message buffer.

At every byte read a check on the message length is done: if it is 1,5,8 or 10 then an operation (Read 32, Write 32, and so on) is performed and the message's buffer is emptied.

### Read 32

The Read 32 message reads a memory word in the STR7's workspace, and sends back its value on the serial line. To start the operation the followings 5 bytes must be sent on the serial line:

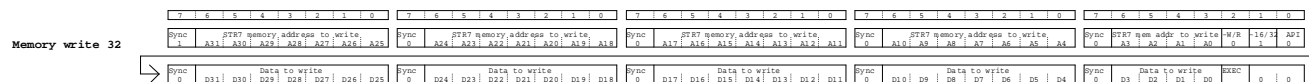


The answer will be sent on the serial line by 4 bytes, starting from the least significant:



### Write 32

The Write 32 message writes a memory word in the STR7 workspace and, if the EXEC bit is set, runs one routine after the write operation. The bytes needed are 10:



### Read 32 Address Pre Incremented (Read 32 API)

This operation uses, as address, the last used in a read or write operation, incremented before by 4 (one word). It needs only 1 byte:



The answer is the same of Read 32: 4 bytes, starting from the least significant.

This operation **need** to have the Sync bit reset to 0.

### Write 32 Address Pre Incremented (Write 32 API)

This operation uses, as address, the last used in a read or write operation, incremented before by 4 (one word). It needs 5 bytes:



# Routines

## Routines run

To run a routine, the EXEC bit of a Write 32 message, the last that sets routine parameters, must be set.

## Routines selection

The routines are selected by an Opcode, written in the least significant 4 bits of Command0 (the word at address 0x20001000), with the following coding:

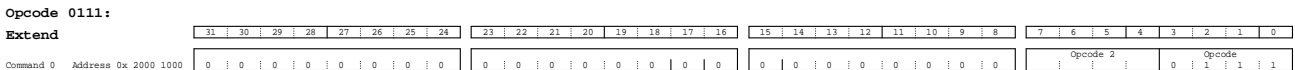
Opcode	Routine name
0x7	Extend
0xD	Flash programming routine
0xF	Jump to routine

## Routines parameters

The parameters of input/output routines are written/read by accessing some RAM words in the workspace of the STR7 microcontroller. Addresses and meanings are written under the routine's description paragraphs.

### Extend

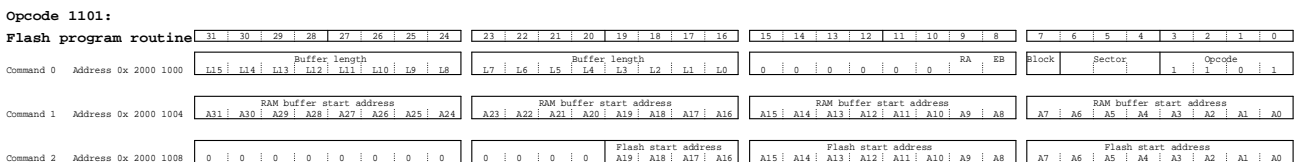
The Extend Opcode is used to extend the routine number over the possibility given by the 4 bit of the Opcode written in the least significant bits of Command 0. Extend needs an "Opcode 2" written in the bits from 4 to 7 of Command 0:



The meanings of the other bits of Command 0 are described in the operations selected by Opcode 2.

### Flash programming

Is possible to program every block of the Flash memory thanks to the fact that the writing routine is copied and executed in RAM. The parameters to set are the following:



The data to write inside the flash memory must be stored in a temporary RAM buffer, by some Write 32 or Write 32 API. The start address of the Ram buffer is then written into Command1, and the start address of the flash memory block to write into Command2. The start addresses of the flash memory blocks are in the range from 0 to 0xfffff:

Bank	Sector	Addresses	Size (bytes)
Bank 0 256 Kbytes Program Memory	Bank 0 Flash Sector 0 (B0F0)	0x00 0000 - 0x00 1FFF	8K
	Bank 0 Flash Sector 1 (B0F1)	0x00 2000 - 0x00 3FFF	8K
	Bank 0 Flash Sector 2 (B0F2)	0x00 4000 - 0x00 5FFF	8K
	Bank 0 Flash Sector 3 (B0F3)	0x00 6000 - 0x00 7FFF	8K
	Bank 0 Flash Sector 4 (B0F4)	0x00 8000 - 0x00 FFFF	32K
	Bank 0 Flash Sector 5 (B0F5)	0x01 0000 - 0x01 FFFF	64K
	Bank 0 Flash Sector 6 (B0F6)	0x02 0000 - 0x02 FFFF	64K
	Bank 0 Flash Sector7 (B0F7)	0x03 0000 - 0x03 FFFF	64K
Bank 1 16 Kbytes Data Memory	Bank 1 Flash Sector 0 (B1F0)	0x0C 0000 - 0x0C 1FFF	8K
	Bank 1 Flash Sector 1 (B1F1)	0x0C 2000 - 0x0C 3FFF	8K

The buffer length is written into Command0.

If a sector is already written then we need to erase it, before write it. To do this we we may set the EB (Erase Before) bit adding the sector number (from 0 to 7) and the Block's number to erase.

We may also use the STR7's built in erasing routine by the following sequence:

- Write 32 of 0x08000000 at address 0x40100000
- Write 32 of sector(s) to erase in 0x40100004
- Write 32 of 0x88000000 at address 0x40100000 (erase start)

The sectors to erase are selected setting the following bits of 0x40100004:

- bit 0: B0F0
- bit 1: B0F1
- bit 2: B0F2
- bit 3: B0F3
- bit 4: B0F4
- bit 5: B0F5
- bit 6: B0F6
- bit 7: B0F7
- bit 16: B1F0
- bit 17: B1F1

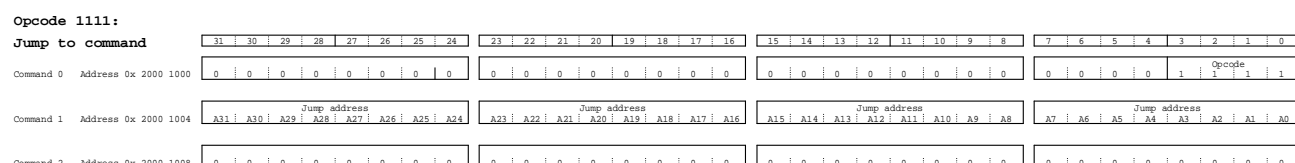
After the start of the erasing we need to wait its end, until bit 1, 2 and 4 of 0x40100000 are all at 0. If the erase ends without errors then the bits 24 (for bank 0) and 25 (for bank 1) of 0x40100004 will be reset to 0.

If we are erasing the sector 0 of the Bank 0, containing the IRQ routine for the RS232 line, during the erase we wouldn't able to send/receive messages on the RS232 line because the entire Block 0 will be unreadable. We need to wait the end of the erasing operation.

If we are erasing the sector containing the IRQ routine of the RS232 line (the B0F0), the STR7 board wouldn't able to communicate more with the external world. To permit the erasing and the programming of the B0F0 sector, we must use both the EB (Erase Before) and the RA (Reboot After) bits. Setting the EB and RA bits, the sector will be erased, then written with the contents of the buffer, and finally a jump to the address 0 will be done, starting the execution of the new code downloaded.

### Jump to

The Jump to routine permits the execution of a code block loaded into RAM by some Write 32 / Write 32 API. The start address, of the code block, must be set in Command1:

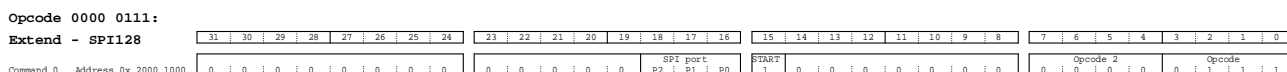


At the end of the code block, the control must be returned to the main program with the following instruction:

```
ldmfd sp!,{r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,pc}
```

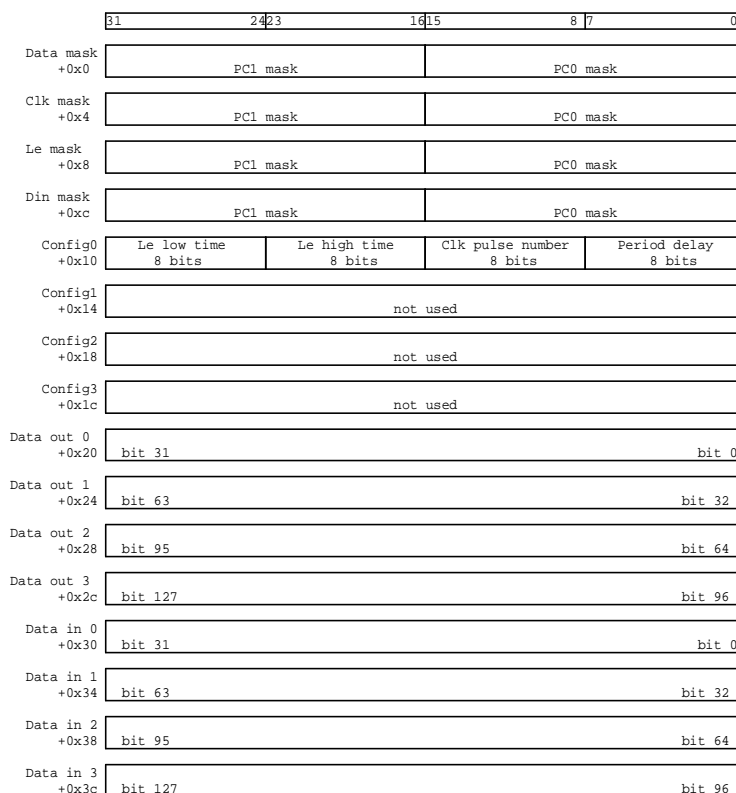
### Extend-SPI128

This routine is selected by an extended opcode:



It implement eight general purpose serial ports long up to 128 bits. The serial port to use is selected by the SPI port bits. For configuration and the control there are 16 words for each port, starting at:  
 SPI128-0: 0x200011dc  
 SPI128-n: SPI128-0+n\*0x40 (n=1..7)

Each block of configuration and control words is organized as follow:



The masks PC0 and PC1 are used to determine where is located the related signal (Clk, Data, Le and Din), in the SPI port considered. PC0 and PC1 are the general purpose I/O port of the STR7 device, and there must be only one bit set in the masks to indicate in which I/O port and which bit the signal is assigned.

**Data** is the **output** data of the port and **Din** is the **input** data.

Le low and Le high times are the clock pulse numbers where the signal Le goes low and high. Clock pulses are numbered starting from zero.

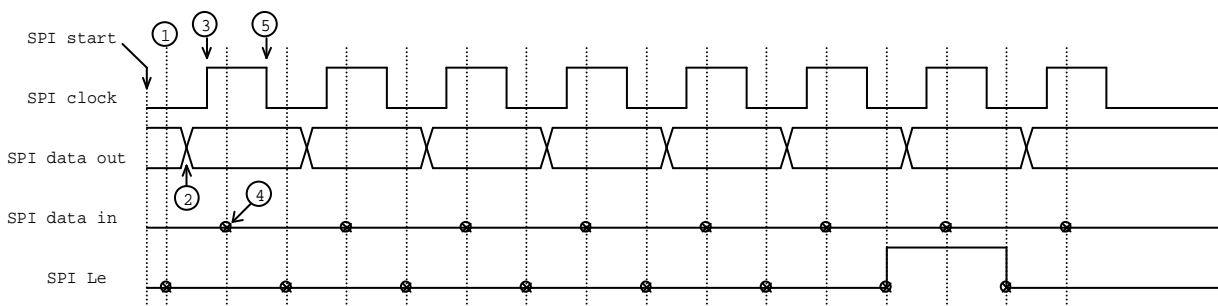
The clock pulse number is the number of clock pulses generated minus 1, for example for 8 clock pulses we must set it to 7, and Period delay is used to determine the speed of the port:

$$\begin{aligned} \text{If Period delay} = 0 \text{ then Clock period} &= 1.27\mu\text{s} \\ \text{If Period delay} > 0 \text{ then Clock period} &= 3.54\mu\text{s} + 148 \mu\text{s} * (\text{Period delay}-1) / 255 \end{aligned}$$

The output bits are shifted out when the clock is at 0, starting from the most significant. Even the Le's value is assigned when clk is zero. The value of Din is instead read after that Clk is gone to "1". The first value of Din sampled is the most significant.

Data and Din are sent and sampled at each clock pulse, so if the answer comes after a certain clock pulse number after the start, the remaining bits must be discarded by the PC software. In the same manner if the message requires some extra clock pulses before to send the useful data bits, the PC software must load the data out registers with the bits already shifted in the correct position.

SPI128 sequence of operations



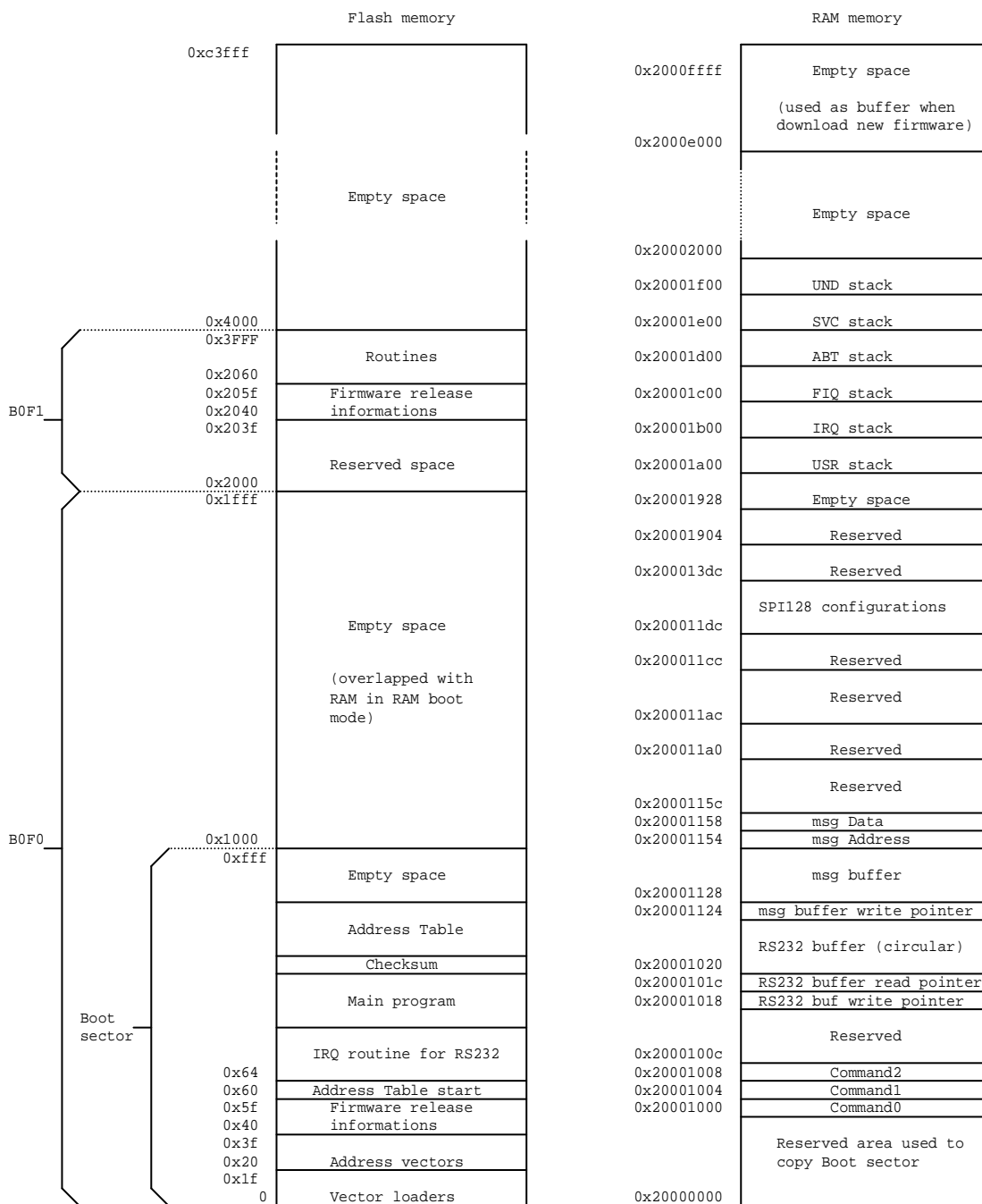
- 1) Le value is set
- 2) Data out is set
- 3) Clock goes high
- 4) Data in is sampled
- 5) Clock goes low

In this example the SPI port has been programmed with:

- Clk pulse number = 7
- Le low time = 7
- Le high time = 6

# Memory map

The memory of the STR7 is viewed by the firmware with the following meanings:



The unused RAM may be used as temporary storage, but is recommended to not overwrite the other variables.

## Vector loaders and Address vectors

The vector loaders are instructions like `ldr pc,address_vector` that runs the code that handle the exception occurred (reset, IRQ, and so on). The addresses of jumps (the vectors) are written in the area 0x20-0x3f

## Firmware release informations

At the fixed address 0x40 there is an ASCII string, 32 bytes long, containing a description of the firmware loaded in the flash memory. There are 2 fields containing informations about firmware: one for the Boot Area (from 0 to 0xffff) and another one for the Routine Area (from 0x2000 to 0xc3fff).

## Address Table

At the fixed address 0x60 there is the address (32 bits long) of a table of relevant firmware addresses:

+ 0x20	P1.8 invert routine address
+ 0x1c	Firmware checksum address
+ 0x18	P1.8 off routine address
+ 0x14	P1.8 on routine address
+ 0x10	Clock test address
+ 0xc	Flash program routine address
+ 0x8	Selector routine address
+ 0x4	RS232 loop address
Address Table Start + 0	Main program start address

These addresses are usually useful during the debug of the firmware, but some may be useful also for the user:

- P1.8 on/off/invert turns on/off/invert the P1.8 line and the related led of the board. They must be used with a **Jump to routine call**
- Firmware checksum may be read to know the rest of the sum of the firmware's words divided by 0x10000

## Main program and Routines

The executable code is divided into 2 parts:

- a fixed part that don't need to be updated frequently (Boot sector)
- a variable part that is related with the ST device driven by the STR7int and that need to be updated frequently to add support for new devices (Routines)

The fixed code is stored in a reserved area from 0 to 0xffff (B0F0) and include the RS232 interrupt routine, the RS232 loop that check if a message is coming, the read/write operations, the Flash program routine, the Jump to routine and a selector between these routines and other routines that must be handled by code starting from B0F1.

The variable code is stored in other sectors than B0F0, starting from B0F1, address 0x2060.

## RS232 buffer

The reception of a byte from the serial line triggers an interrupt routine that take the data and store it in an buffer, the RS232 buffer. This is a circular FIFO buffer with two pointers: one containing the address of the first location free to write, and the other containing the address of the first byte to read. If these pointers are equal, the buffer is empty.

## Message buffer

The main program waits, inside a loop, until the RS232 buffer is not empty. When this happens, read the first byte available and add it to the Message buffer, trying to assemble a valid message. When a complete message is received, the data, address and flag contained will be extracted, and will be executed the operation contained. At the end of the execution the message buffer will be cleared and the processor goes again in the loop that waits bytes from RS232