Minimizing Power Consumption for SPI EEPROMs

APPLICATION NOTE

Many industrial and automotive applications use external SPI EEPROMs to back-up key data. Power consumption must be minimized during back-up to SPI EEPROMs. This becomes critical when decoupling capacitors are used to store the energy necessary to complete the data back-up process.

This application note explains how to use SPI EEPROM Page Mode Operation for low power consumption, and gives advice on optimum data mapping and peripheral configurations. A C-language implementation of the SPI EEPROM Page Mode operation is contained in the appendix.

Configurations outlined in this application note keep the ST10 in idle mode for most of the write to the external EEPROM. A 50% savings in power consumption can be achieved for both ST10F168 and ST10C167 by using SPI EEPROM Page Mode, mapping data into internal RAM and stopping unused peripherals during data back-up. Power consumption savings are defined by the ratio: Idle current / Operating current. To assess the saving for your application, check the power consumption defined in the product Data Sheet.

Features such as, built-in error detection on SPI, and EEPROM write protection mechanism, can be used to further secure the back-up of key data into external SPI EEPROM.

1 Introduction

The ST10 microcontroller has 2 power reduction modes:

- Power down mode: the CPU and all the peripherals are stopped. Exit from power-down is only possible through an external interrupt signal. This mode gives the best power consumption savings.
- Idle mode: the CPU is stopped but all the peripherals remain active. Exit from idle mode is through the external interrupt signal or by internal interrupt.

For most applications, idle mode is preferred because the device can resume operation on a pre-defined peripheral event (end of transfer on SPI bus and/or end of write into external EEPROM). Program flows should take advantage of idle mode by relying on peripheral's hardware features rather than software instructions.

2 Minimizing power consumption

2.1 SPI EEPROM Page Mode operation for low power consumption

Many SPI EEPROMs operate read and write sequences in Page Mode (see Non-Volatile Data Books on http://www.st.com). This feature can be used to reduce ST10 power consumption during data transfer. The program flow described below and shown in Figure 1, transfers data to an SPI EEPROM but keeps the ST10 in idle mode for most of the transfer.

Program flow

1 Data is transferred from the RAM to the SPI transmit register by PECC transfer.

2 The ST10 is put in idle mode during page transfer to the external SPI EEPROM.

3 When the page transfer is complete, the CPU is put in idle mode for the duration of the write to SPI EEPROM (using an internal timer).

4 At the end of the write sequence, the timer wakes-up the ST10 and the CPU checks whether another page is to be written to the external SPI EEPROM.

A C-language implementation and assembly implementation of the power control instructions, are given in sections 2.3 and 2.4. A C-routine is given in Section "Appendix - C function example" on page 5.

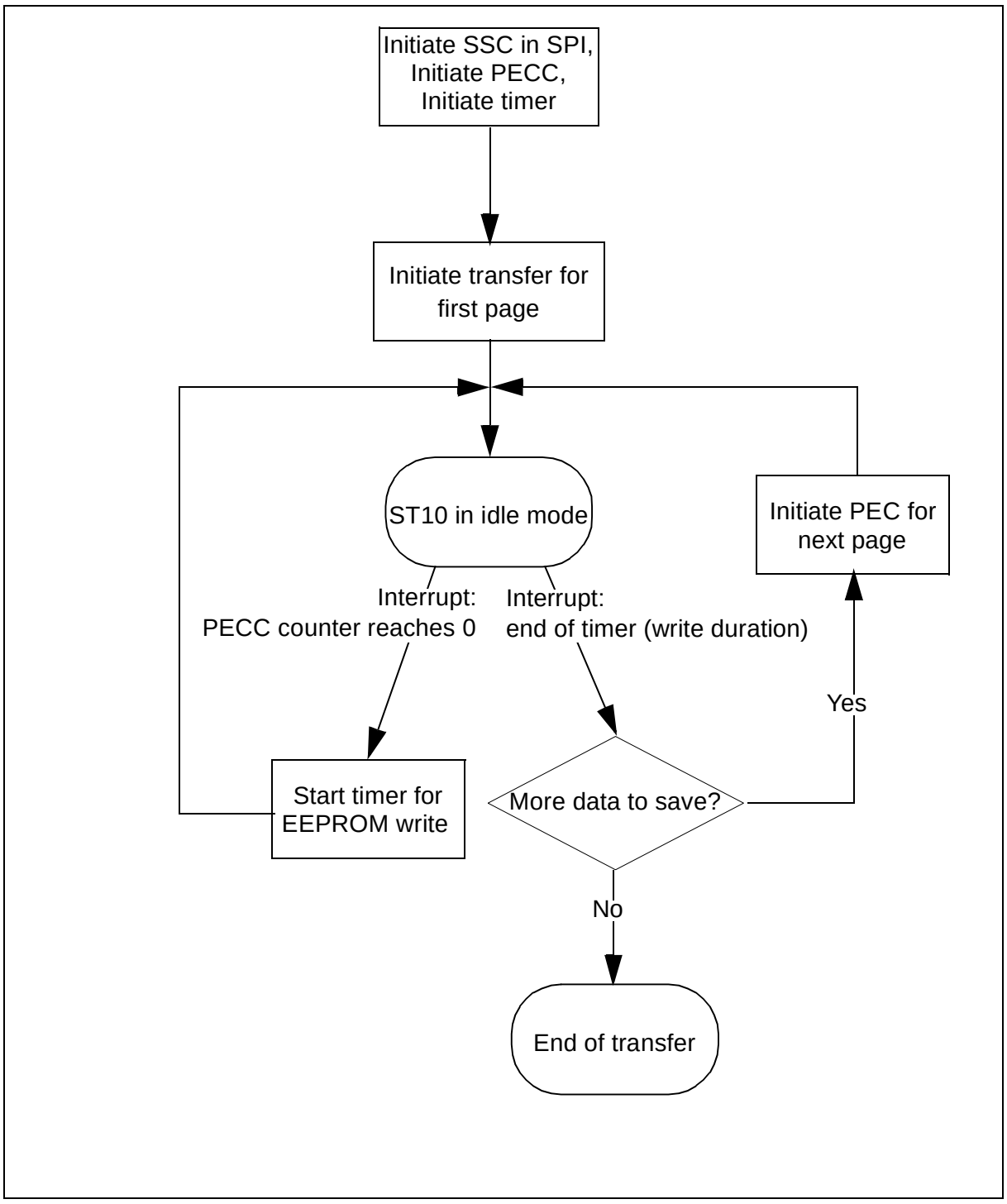

Figure 1 Program flow

2.2 Other ways to reduce power consumption

Mapping data into internal RAM only

Where possible, map data into ST10 internal RAM instead of external memory. The internal RAM can either be the internal dual-port RAM, or the internal XRAM. The current required to address external memory is greater than that for internal memory.

Stop unused peripherals during data back-up

During the data back-up process, stop all unused peripherals in the following ways:

- Unused timers: clear bit in the relevant TxCON registers,
- Unused CAPCOMs: stop the CAPCOM timers/counters by clearing bits TxR in T01CON or T78CON registers,
- Unused PWMs: stop the timers by clearing bit PTRx in PWMCONx registers.
- Asynchronous serial interface: the baud ASC baud rate generator can be stopped by clearing bit S0R in S0CON register.

Port configuration

Where possible, configure ports for minimum power consumption.

2.3 C language implementation

Power control instructions in C are usually handled by intrinsic functions. For the Tasking tool chain, the intrinsic function for idle is “_idle”. The Tasking tool chain can directly handle SFRs provided that the targeted ST10 register definition is included in the C source file.

As the ST10 resumes the program just after the “idle” instruction, the interrupt service routines for both the timer and SPI_transmit can be reduced to “RETI”. Then the whole program flow can be implemented in one C function. An example of such a C function is shown in Section . This example gives the ST10 configuration for the data back-up process. The program flow can be modified perform a data integrity check, or enable ST10 error detection on SPI (phase error, receive error)

2.4 Assembly implementation

The instruction to put the ST10 in idle is “IDLE”. The suggestions made for the interrupt service routines are also applicable here.

3 Conclusion

A 50% savings in current consumption can be achieved for both ST10F168 and ST10C167 by using SPI EEPROM Page Mode, mapping data into internal RAM and stopping unused peripherals during data back-up. Power consumption savings are defined by the ratio: Idle current / Operating current.

Features such as, built-in error detection on SPI, and EEPROM write protection mechanism, can be used to further secure the back-up of key data into external SPI EEPROM.

Appendix - C function example

The C routines below illustrates an implementation of the program flow detailed in Section 2.1.

```
// include to have declaration of SRFregisters and SFRbits
#include <reg167.h>

// function prototypes
void init_spi_page_write();
void spi_save(int addr);
void spi_send_write_cmd(int addr);

// functions

// main SPI transfer function
void spi_save(int addr)                    // assume write into EEPROM is enabled
{
int i;
init_spi_page_write();                     // init SSC in SPI for page transfer

for (i=0; i=3; i++)                        // example for fixed length transfer : 4 pages
{
SSCIE = 0;                                 // mask SPI interrupt during instruction + address
                                           phase
spi_send_write_cmd(int addr);              // send the write command : instruction + address
SSCTIC= SSCTIC | 0x00C0;                   // set Interrupt request to start  page transfer +
                                           enable interrupt
_idle;                                     // put the ST10 in idle until page transfer is
                                           completed
TxR=1;                                     // set bit TxR to start timer
_idle;                                     // put the ST10 in idle for the write duration
PECC0 = 0x01yy;                            // prepare PECC0 for next page transfer}
}
}                                          // EEPROM is assumed to be reprotected after this
```

```
                                        // function called by main SPI transfer
// other functions
void spi_send_write_cmd (int addr)      // send command and address for write
{
if (addr > 255)                         // EEPROM is assumed in write enable state
                        {
                        SSCTB = 0x000A;     // depend on EEPROM size
                        addr = addr -256;
                        }
                        else
                        {
                        SSCTB = 0x0002;
                        }
SSCTB = (char) addr;                    // take only low byte
while (SSCTIC & 0x0040 )                // wait end of transmit of 2nd byte
                        {
                        }
}

// function called by main SPI transfer function
void init_spi_page_write()              // init SPI to transfer pages
{
P3 = P3 | 0x2200;
DP3 = PP3 | 0x2200;
SSCBR = SPI_reload;     // depend on bit rate
PECC0 = 0x01YY;         // byte transfers, yy is the EEPROM page size in bytes
SSCTIC = 0x0078;
DSTP0 = 0xF0B0;         // SSCTB
SRCP0 = 0xE000;         // buffer is in XRAM starting at E000h
SSCCON = 0xC037;        // MSB first, ignore errors (check your configuration)
Tx = timer_x_reload;    // depend on EEPROM write timings
TxCON = 0x0;            // Timer in timer mode, count-up (check prescaler factor)
}
```