SIEMENS

Microcontrollers

ApNote AP083201

☐or ☑ additional file
AP083201.EXE available

Emulating an asynchronous serial interface (USART) via the on-chip synchronous serial interface (SSC)

Abstract:

The solution presented in this paper and in the attached source files emulates the most important USART functions by using the on-chip synchronous serial communication channel (SSC). The code is focused on the SAB C513, but will fit to all C500 derivatives. Beyond the low level software drivers a test shell is delivered. This shell allows a quick test of the software drivers by an emulator or a starter kit demo board.

Author: W. Boelderl-Ermel, HL COM WN SE

AP0832 ApNote - Revision History		
Actual Revision : Rel.01	Previous Revision: none	
Page of actual Rel.	Page of prev. Rel.	(Subjects changes since last release)

1 Introduction

The C500 microcontroller family provides usually only one on-chip asynchronous serial communication channel (USART). If a second USART is required, an emulation of the missing interface may help to avoid an external hardware solution with additional electronic components.

The solution presented in this paper and in the attached source files emulates the most important USART functions by using the on-chip synchronous serial communication channel (SSC) with a performance up to 93 KBaud in half duplex mode and an overhead less than 18% at SAB C513 with 12 MHz. Due to the implementation in C this performance is not the limit of the chip. A pure implementation in assembler will result in a strong reduction of the CPU load and therefore increase the maximum speed of the interface. In addition, microcontrollers like the SAB C505 will speed up the interface by a factor of two because of an optimized architecture compared with the SAB C513.

Moreover, this solution lays stress on using as few on-chip hardware resources as possible. A more excessive consumption of those resources will result in a higher maximum speed of the emulated interface.

Due to the restricted performance of an 8 bit microcontroller a pin compatible solution is provided only; the internal register based programming interface is replaced by a set of subroutine calls.

The attached source files also contain a test shell, which demonstrates how to exchange information between an on-chip HW-USART and the emulated SSC-USART via three external wires in different operation modes. It is based on the SAB C513 (Siemens 8 bit microcontroller).

A table with load measurements is presented to give an indication for the fraction of CPU performance required by software for emulating the USART.

2 General Operation and Hardware Environment

2.1 Supported Features

The following enumeration summarizes all features of the on-chip USART to be emulated by software routines and the SSC:

- 8 bit data frames with variable baud rate,
- half duplex communication,
- baud rates between $f_{osc}/512$ and $f_{osc}/128$ @ SAB C513 with 12 MHz crystal

The following enumeration lists all functions of a SAB C500 on-chip USART, which could not be cloned due to technical limitations or performance restrictions:

- 8 bit shift register with fixed baud rate $f_{osc}/12$,
- 9 bit data frames,
- multiprocessor communication feature,
- full duplex communication.

2.2 Required Resources

To emulate the USART interface by a set of software routines and the on-chip SSC requires some resources, which are listed in the following table:

Table 1
Resource Requirements

Resource	Emulated USART
Number of required I/O pins	2
Number of interrupt pins	1
Interrupt Priority	yes
Timer	T0 (Timer Mode 3)
Additional On-Chip Modules	SSC
Program Memory (Emulation routines only)	250 Byte
Data Memory (Emulation routines only)	7 Byte

2.3 External Routing

An external wire connecting the SSC data input with the External0 Interrupt pin is required to activate the SSC via a Start Bit transmitted by the external communication partner. On test boards with C513 processor the on-chip USART may also be used as ‘external party’.

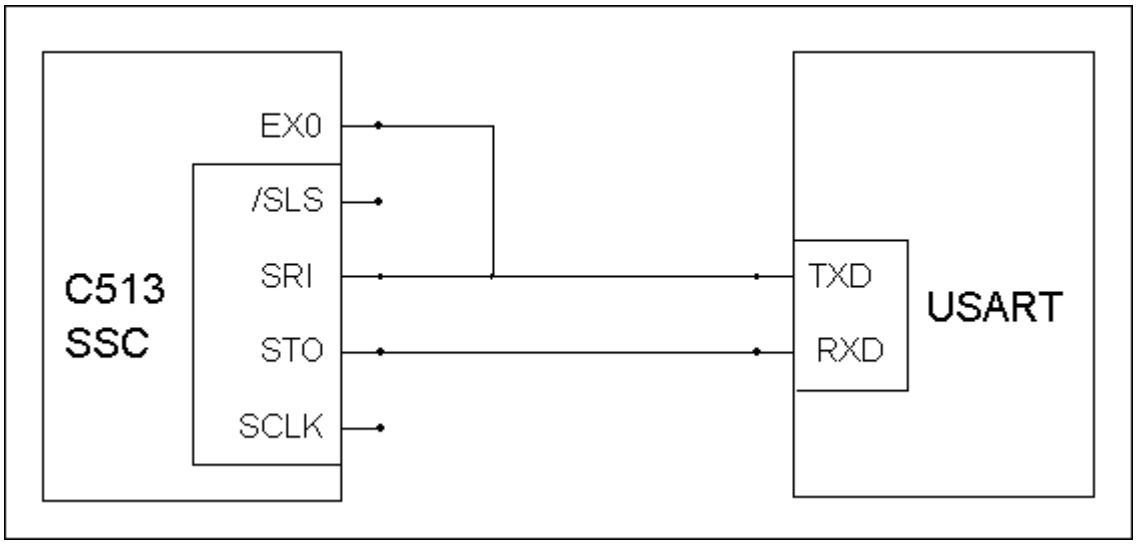

Figure 1
External Routing of Transmission Lines

2.4 Principles of Emulation

The algorithms required for emulating the data transmission depend on the transfer direction.

2.4.1 USART Write

A USART Write is initiated by pulling down the SSC-STO output pin (USART Start Bit) and starting a timer loaded with one bit time of the required baud rate.

The timer interrupt service routine starts the SSC in ‘Master Mode’ by writing the 8 data bits to be transmitted into the SSC-STB buffer and enables the SSC-STO output pin for ‘Alternate Function’.

The USART Stop Bit is automatically generated by the SSC-STO output going ‘High’ in idle mode. There is no explicit transmission of a stop bit. By restarting the write cycle a second time, there is (at higher baudrates) so much delay caused by the microcontroller compared to one bittime that an additional wait-until-stopbit-finished is not necessary. An application which is sensitive at this point has to be modified by the user.

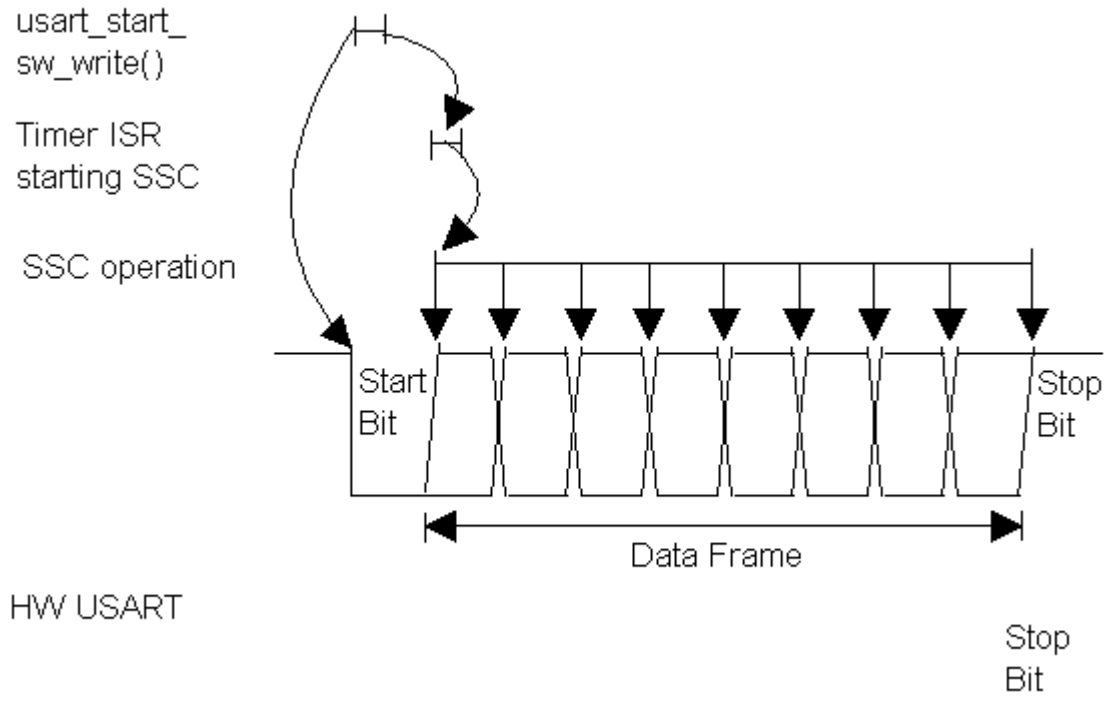

Figure 2
Schematic Diagram of Emulating an USART Write Operation using the on-chip SSC

2.4.2 USART READ

A USART READ is initiated by a USART Start Bit arriving at the SSC-SRI input pin, which is externally connected to an External Interrupt pin. The correlated interrupt service routine starts a timer loaded with one bit time of the required baud rate.

The timer interrupt service routine starts the SSC in ‘Master Mode’ by writing a dummy byte (0xFF) into the SSC-STB buffer, which simultaneously starts the SSC receive shift register sampling 8 data bits. Although the dummy byte 0xFF is really shifted out via the SSC-STO pin, the external communication partner’s receive HW will not be triggered, because the transmitted message byte does not contain any ‘0’ bit to be interpreted as Start Bit.

The final USART Stop Bit provided by the external communication partner is completely ignored.

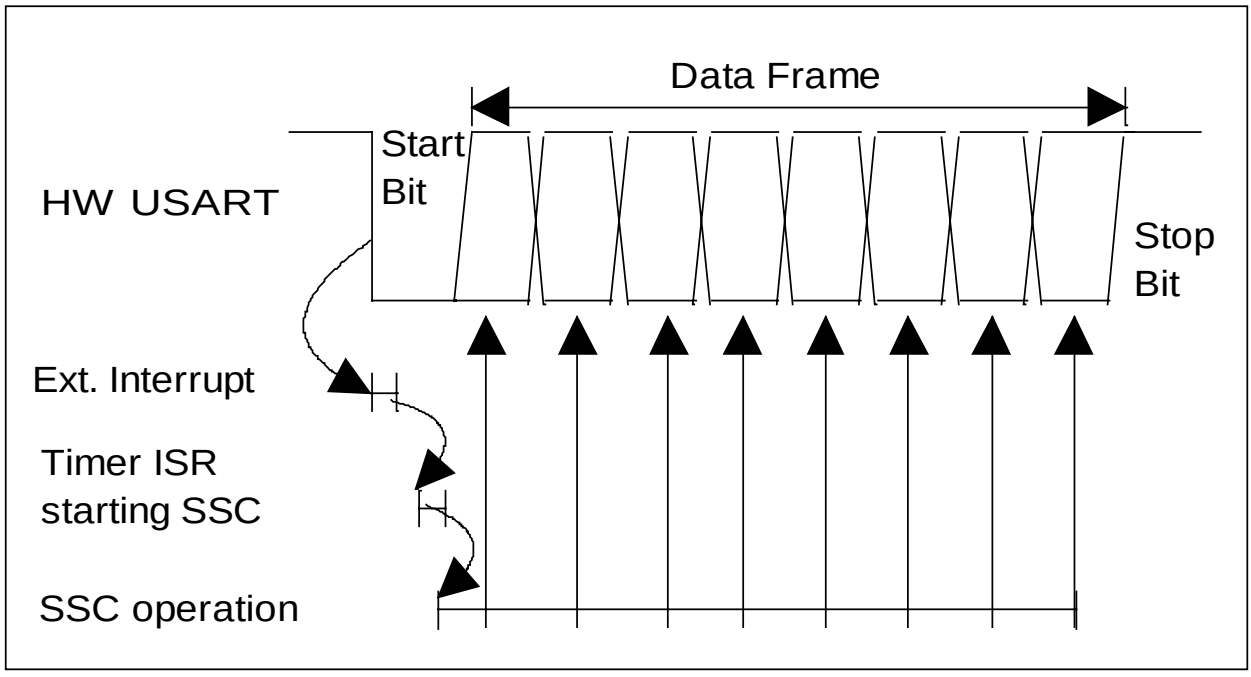

Figure 3
Schematic Diagram of Emulating an USART Read Operation using the on-chip SSC

3 USART Emulation Software Description

3.1 Software Structure

The emulation software is written in C and is split into 3 files:

- uss_emul.c contains all low level software drivers (subroutines and interrupt services) to emulate the USART with the on-chip SSC. This file may be directly added to the user's application software directory and may be included in his make file.
- uss_test.c demonstrates how to start, control and finish the emulation. The complete file (test shell) may be used to check the low level software drivers in a real application. Afterwards, the user may copy the required statements for calling the individual USART functions into his own application code segments.
- uss_defi.h holds all definitions and declarations related to the emulation software (uss_test.c, uss_emul.c).

3.2

Main Program

The main program (uss_test.c) is implemented as a state machine and handles several test cases.

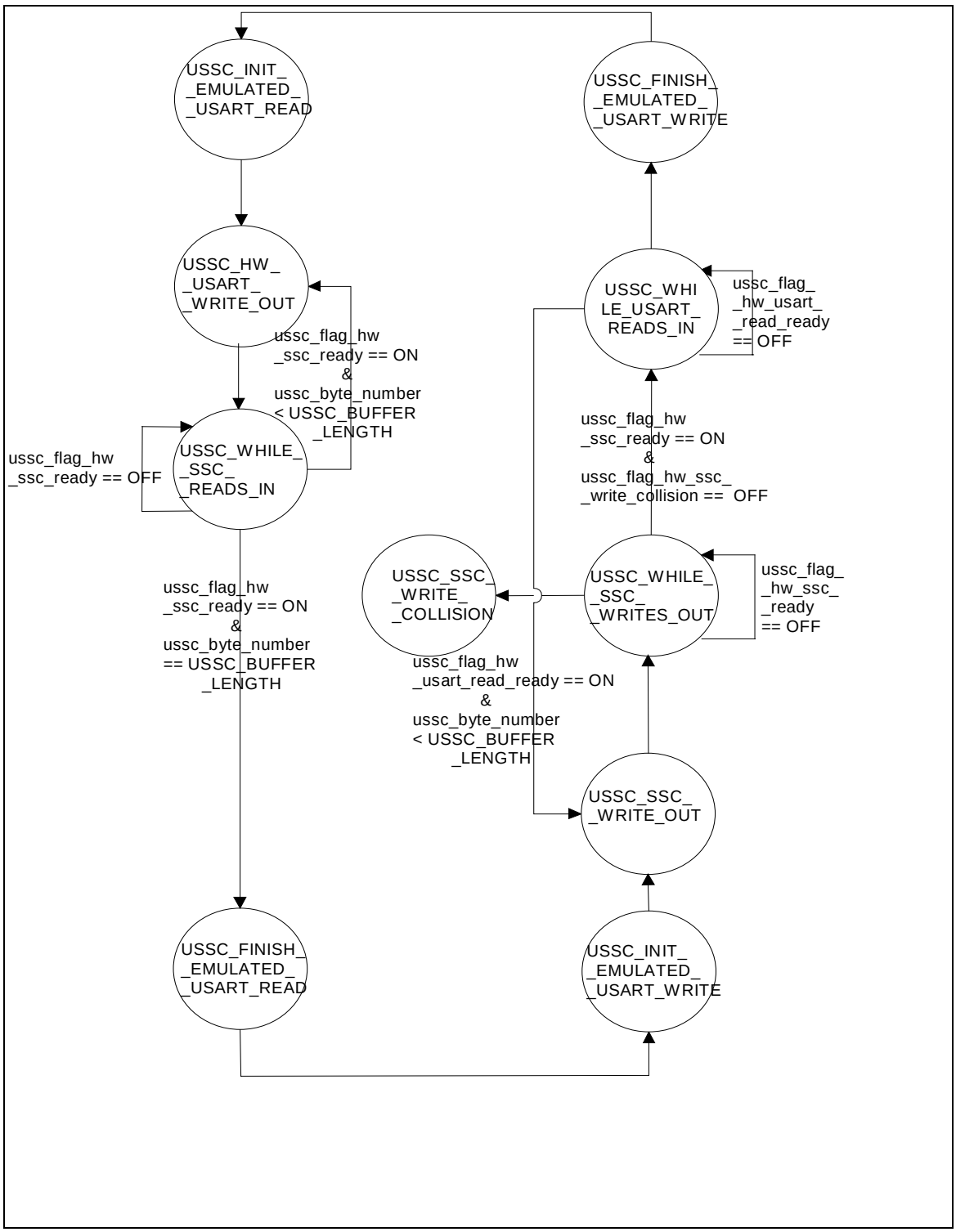

Figure 4
State Machine Diagram for test program ”uss_test.c”

The first test case verifies the emulated USART by a data reception from an external source:

- In the first state ‘USSC_INIT_EMULATED_USART_READ’ the emulated USART interface is initialized with the baud rate to be supported (f_{osc}/128). As communication partner serves the on-chip HW-USART which is set up in the same baud rate.
- The second state ‘USSC_HW_USART_WRITE_OUT’ starts the on-chip HW-USART and prepares a dummy byte used to activate the SSC after reception of the HW-USART start bit.
- In the third state ‘USSC_WHILE_SSC_READS_IN’ a flag is polled indicating the end of data reception via the SSC. User application code to be executed during the SSC read in operation may be included here instead of wasting 9 bit times only for running a polling loop. After finishing the transmission of a whole message containing a programmable number of bytes the state machine proceeds to the next state.
- The last state ‘USSC_FINISH_EMULATED_USART_READ’ starts with a rearrangement of bit order in the input message string, because a message sent by an USART interface is transmitted with LSB first while a SSC works with MSB first. The last subroutine disables all hardware modules required for data transmission.

In the second test case the communication is started with an altered transmission direction. The SSC operates as data source and provides the on-chip HW-USART with a message string.

3.3 Emulation Subroutines

The file uss_emul.c contains all subroutines and interrupt services required for controlling the USART emulation:

- 'ussc_init_ussc_hw ()' initializes all required auxiliary hardware modules like Timer, the External Interrupt input and the SSC by programming their control registers.

- 'ussc_rearrange_bit_order ()' copies a programmable number of bytes in a message string, rearranges the bit order of every byte and stores them back at the original location.

- 'ussc_write_out ()' prepares a data transmission via the SSC by pulling down the SSC-SLS enable input pin and the SSC-STO output pin to 'low' state, which is interpreted by the communication partner as a Start Bit transmission. Furthermore a timer is started for switching off the Start Bit after one bit time and activating the SSC 8 bit data transmission. The timer load value for achieving 'one bit time' is calculated by a formula presented in chapter 3.4.
  At maximum baud rate the Start bit can not be handled by a timer interrupt routine due to an Interrupt Service Delay being longer than one bit time. Therefore, the one bit time is generated by executing some 'NOP' statements before starting the SSC directly.

- 'ussc_disable_ussc_hw()' disables all required auxiliary hardware modules like Timer, the External Interrupt input and the SSC by setting their control registers respectively.

- 'ussc_int0_interrupt_service()' is started by a 'Low' level at the SSC-SRI pin, which is externally wired to the interrupt0 pin. The 'Low' level is interpreted as an USART Start Bit announcing the arrival of further data bits. Therefore a timer is started to activate a SSC 8 bit reception after an one bit time interval. Finally the interrupt0 service is disabled to avoid undesired reactions to incoming '0' data bits.
  At maximum baud rate the Start bit can not be handled by a timer interrupt routine due to an Interrupt Service Delay being longer than one bit time. Therefore, the one bit time is generated by executing some 'NOP' statements before starting the SSC directly.

- 'ussc_timer0_interrupt_service()' starts the HW-SSC by writing a byte into the STB output buffer. The next statement switches the SSC-STO output pin without any intermediate spike to its alternate function. Timer0 is disabled and reloaded with one bit time.

- 'ussc_hw_ssc_interrupt_service()' is called after completion of an 8 bit SSC data transfer or by a Data Collision Error. In case of a transfer completion event the received data byte is stored in the array 'ussc_ssc_receive_buffer' and the Transfer Completion Flag is reset. The External Interrupt is enabled again to be prepared for handling the Start Bit of the next data byte reception. In case of starting a transmission while a data reception is already running a global error flag is set and WCOL is reset.

3.4 Baud Rate Calculation

Data transmission via an USART interface requires an identical baud rate to be generated by both communication partners. This can be achieved by selecting a suitable clock oscillator base frequency and a corresponding prescaler factor.

The baud rate required for the HW-SSC is controlled by BRSx bits in register SSCCON:

$$\text{SSC\_Baud\_Rate} = \frac{\text{fosc}}{4 * 2^{**}\text{BRS}}$$

The initial load value for the timer ('One Bit Time') handling the Start Bit is calculated as presented in the next formula and stored in a constant (USSC_BIT_TIME_xxxx_BAUD) in file 'uss_defi.h':

$$\text{One\_Bit\_Time\_using\_timer0} = \frac{12 * (256 - \text{TL0} + \text{Interrupt\_Service\_Delay})}{\text{fosc}}$$

For emulated USART read operations the term 'Interrupt_Service_Delay' takes into account

- the Interrupt Response Time for External Interrupt0 after receiving a Start Bit,
- the Interrupt Response Time for Timer0 Overflow including the execution time for all statements in the corresponding interrupt service routine before starting the HW-SSC.

For emulated USART write operations the term 'Interrupt_Service_Delay' takes into account

- the Interrupt Response Time for the Timer0 Interrupt after transmitting a Start Bit and the execution time for all statements in the corresponding interrupt service routine before starting the SSC.

The exact value for 'Interrupt_Service_Delay' may be extracted by analyzing the assembler program listing.

Table 2
Calculated Interrupt Service Delays

SSC Baud Rate fosc = 12 MHz	Mode	External Interrupt Delay (fosc = 12 MHz)	Timer0 Int. Delay (fosc = 12 MHz)
93.750 K	Read	9.0 us	-
93.750 K	Write	-	-
46.875 K	Read	9.0 us	9.0 us
46.875 K	Write	-	9.0 us
23.437 K	Read	9.0 us	9.0 us
23.437 K	Write	-	9.0 us

[1)] To avoid a timer0 load statement within the EX0 interrupt service routine preparing the timer with the exact value for an emulated USART read operation the arithmetical mean of calculated load values for read and write operation is used as common timer0 load constant which is defined in a constant (USSC_BIT_TIME_xxxx_BAUD) in file 'uss_defi.h'.

In the ”test shell” (uss_test.c) a communication system is designed using one USART emulated by SSC and one on-chip USART. They both are connected to each other. These both peripherals are implemented on one SAB C513 device. So, after initializing the speed of the USART emulated by SSC an initialization of the speed of the on-chip hardware USART is necessary as well.

The reload values for the T2 timer configured as HW-USART baud rate generator are calculated as presented in the next formula:

The reload values for the T2 timer configured as HW-USART baud rate generator are calculated as presented in the next formula:

$$\text{HW\_ USART\_ Baud\_ Rate\_ using \_ Timer2} = \frac{\text{fosc}}{2*16*(65536 - \text{TH2,TL2})}$$

3.5 Load Measurement

Emulating a hardware module by a set of software subroutines decreases the processor performance available for user application software.

A load analysis shows the fraction of processor performance required for emulating routines.

The processor load generated by the emulation software is defined as:

$$\text{Load} = \frac{\text{Time spent in emulating and interrupt service routines}}{\text{Total amount of time for transmitting / receiving n bytes}} * 100\%$$

The execution time of the required interrupt service routines emulating the USART is calculated by analyzing the compiler object module listing.

The next table presents load measurement results for a USART emulation via the on-chip SSC running with different baud rates.

Table 3:
Load Measurement Values for an USART emulation via the on-chip SSC at SAB C513

Crystal Frequency	Baud Rate	Load
12 MHz	23.437 KB	6.5%
12 MHz	48.368 KB	13.0%
12 MHz	93.750 KB	18.0%

Attention: The load value increases with falling clock generator frequencies.

3.6 Performance Limitations

The most severe limitation is seen in READ mode at maximum baud rate. The incoming Start Bit triggers an external interrupt pin and the related interrupt service routine has to start the on-chip SSC directly because the interrupt service delay of EX0 is in the same order of magnitude as 1.0 bit time of 93.75 KBaud.

Moreover this application note is the attempt to use as few on-chip hardware resources as possible. This effort results in a higher demand of software performance.

In addition having a look at C500 derivatives it has to be taken care about the fact that e.g. SAB C505 has twice the performance than SAB C513. So, a solution using the SAB C505 will result in twice the maximum baudrate than SAB C513.

3.7 Make File

The file uss_make.bat contains all statements to start the Keil C51™ compiler, linker and locator. (Versions: C51 V5.10, L51 V3.52, OH V2.1) The paths to the source file and compiler / library directories must be modified by the user in respect to the individual file structure on his personal computer.

The Make-File is started by typing ‘uss_make.bat’ in a DOS window switched to the directory containing this batch file.

3.8 Support of KitCON-513 Evaluation Board

The KitCON-513 Evaluation Board is a starter kit (order at Siemens Semiconductors www) which helps for a general approach to the SAB C513. Generally speaking it is a printed circuit board which lets you load software down via the PC to the SAB C513. After that the SAB C513 executes that code and may be verified.

The starter kit is delivered with one SAB C513 romless device and one SAB C513 EEPROM device. It is advised firstly to make use of the SAB C513 romless device to load (program) the code down to the SAB C513 EEPROM device After that the two devices (romless and EEPROM) have to be changed and the programmed code is executed out of the on-chip EEPROM.

Using the ”test shell” usa_test.c the SW-USART of the SAB C513 EEPROM communicates with the on chip HW-USART of the SAB C513 EEPROM device.

The port pins occupied by HW-SSC and HW-USART can be easily connected by setting jumpers on the 152 pin KitCON application area connector: The next table presents all port pins to be externally wired:

Table 4:
Port pins to be externally wired on KitCON-513 Evaluation Board

	HW-SSC-SRI (P1.3)	HW-SSC-STO (P1.4)
HW-USART-RXD (P3.0)		X
HW-USART-TXD (P3.1)	X	
Ext. Interrupt 0 (P3.2)	X	

Note: The HW-SSC-SLK pin (P1.2) may be used for testing purpose (e.g. scanning the HW-SSC operation in respect to HW-USART Write timing).

After pressing the restart button the test program runs in an endless loop.