SIEMENS

Microcontrollers

ApNote AP083001

☐or ☑ additional file
AP083001.EXE available

Emulating a synchronous serial interface (SSC) interface via software routines

Abstract:

The solution presented in this paper and in the attached source files emulates the most important SSC functions by using SW routines implemented in C. 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

AP0830 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 usually provides only one on-chip synchronous serial channel (SSC). If a second SSC 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 SSC functions by using optimized SW routines with a performance up to 25 KBaud in Slave Mode with half duplex transmission and an overhead less than 60% 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-SSC and the emulated SW-SSC via 5 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 SSC.

2 General Operation and Hardware Environment

2.1 Supported Features

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

- 8 bit data frames with variable baud rate,
- master mode selected: reception and transmission of data in half duplex mode
- slave mode selected: either reception or transmission of data is possible at a falling clock edge
- Clock Polarity selection,
- Clock Phase selection,

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

- baud rates higher than 15 K in Master Mode or higher than 25K in Slave Mode @ SAB C513 with 12 MHz crystal,
- Slave Mode with data transmission or reception triggered by rising clock edge,
- full duplex communication.

2.2 Required Resources

To emulate the SSC interface by a set of software routines requires some resources listed in the following table:

Table 1
Resource Requirements

Resource	**Emulated SSC**
Number of required I/O pins	4
Number of interrupt pins	1
Interrupt Priority	yes
Timer	T0 or T1 (Auto Reload)
Program Memory (Emulation routines only)	218 Byte
Data Memory (Emulation routines only)	14 Byte

2.3 External Routing

An external wire connecting the SSC clock input with the External0 Interrupt pin is required to trigger the SW-SSC configured in Slave Mode. On test boards with C513 processor the on-chip HW-SSC may also be used as ‘external party’.

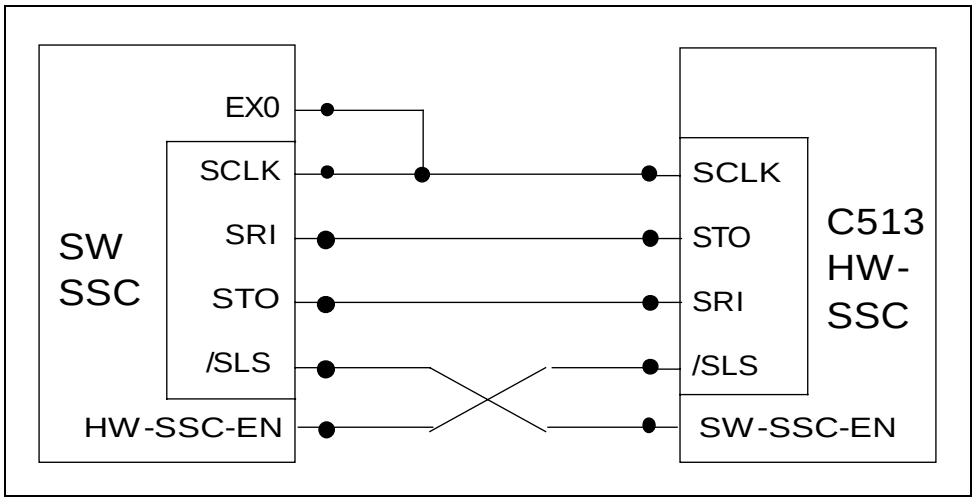

Figure 1
External Routing of Transmission and Control Lines

2.4 Principles of Emulation

The algorithms required for emulating the data transmission depend on the operation mode.

2.4.1 Master Mode

A timer is configured in 'Auto-Reload Mode' and is loaded with half a bit time of the required baud rate. The associated interrupt service routine toggles the SW-SCLK pin and transmits or receives one data bit. The timer is stopped after the 16th clock edge.

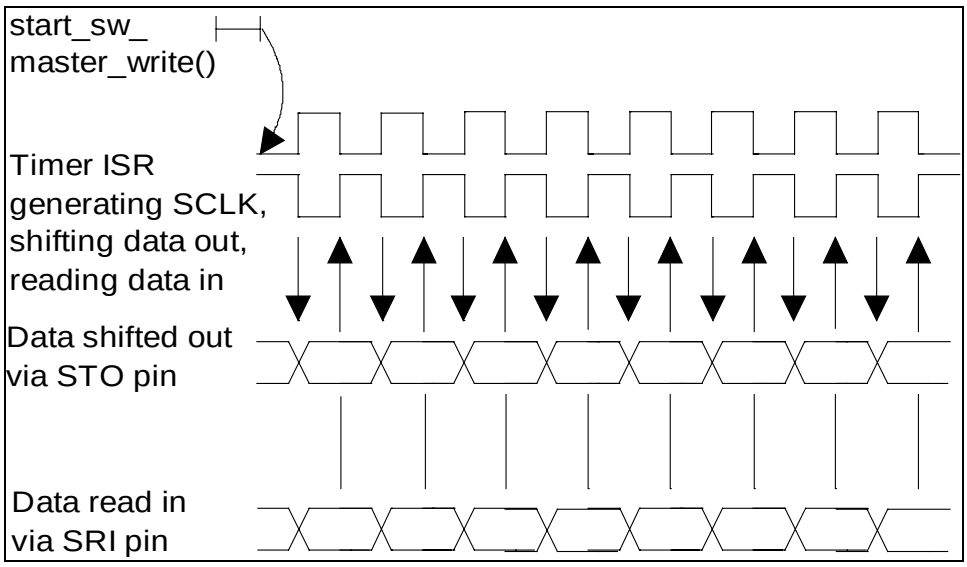

Figure 2
Schematic Diagram of an emulated SSC operating in Master Mode

2.4.2 Slave Mode

The Slave Mode is handled by an External Interrupt Pin triggered by falling edges of the external Master Clock. Due to the restrictions of the C51 controller architecture rising signal edges cannot be detected by External Interrupt pins which prevents a full duplex data transmission and an unlimited combination of Clock Phase and Clock Polarity selection. Therefore, falling edges of an external Master Clock may be used for sampling in externally provided data bits or for shifting out internally generated data bits only. At the file ssc_emul.c the communication direction has to be selected when the code is translated by the compiler.

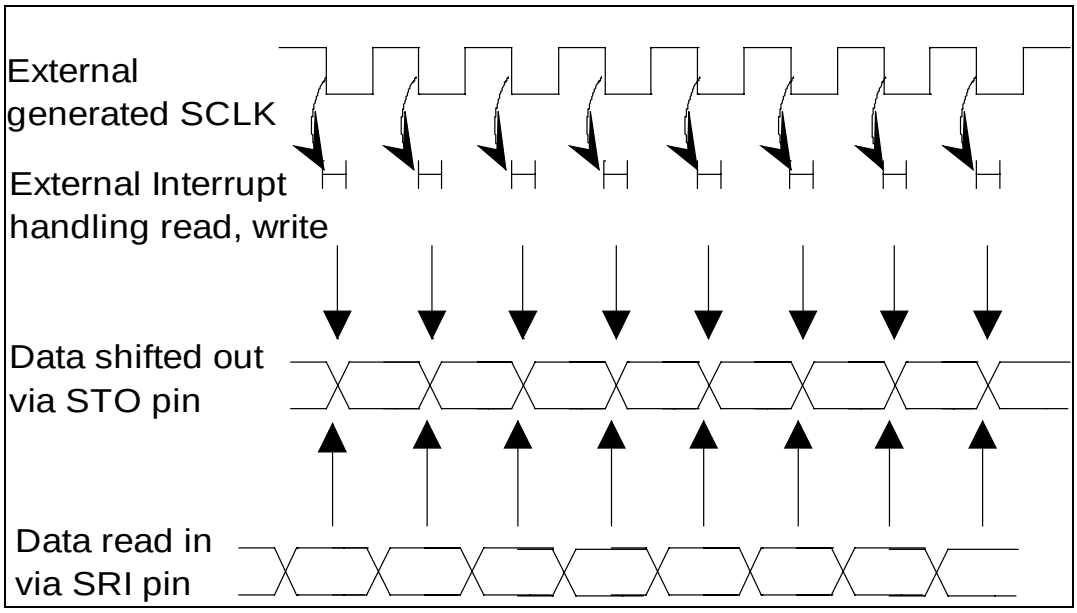

Figure 3
Schematic Diagram of an emulated SSC operating in Slave Mode

3 SSC Emulation Software Description

3.1 Software Structure

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

- ssc_emul.c contains all low level software drivers (subroutines and interrupt services) to emulate the SSC. This file may be directly added to the user's application software directory and may be included in his make file.
- ssc_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 SSC functions into his own application code segments.
- ssc_defi.h holds all definitions and declarations related to the emulation software (ssc_test.c, ssc_emul.c)

3.2 Main Program

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

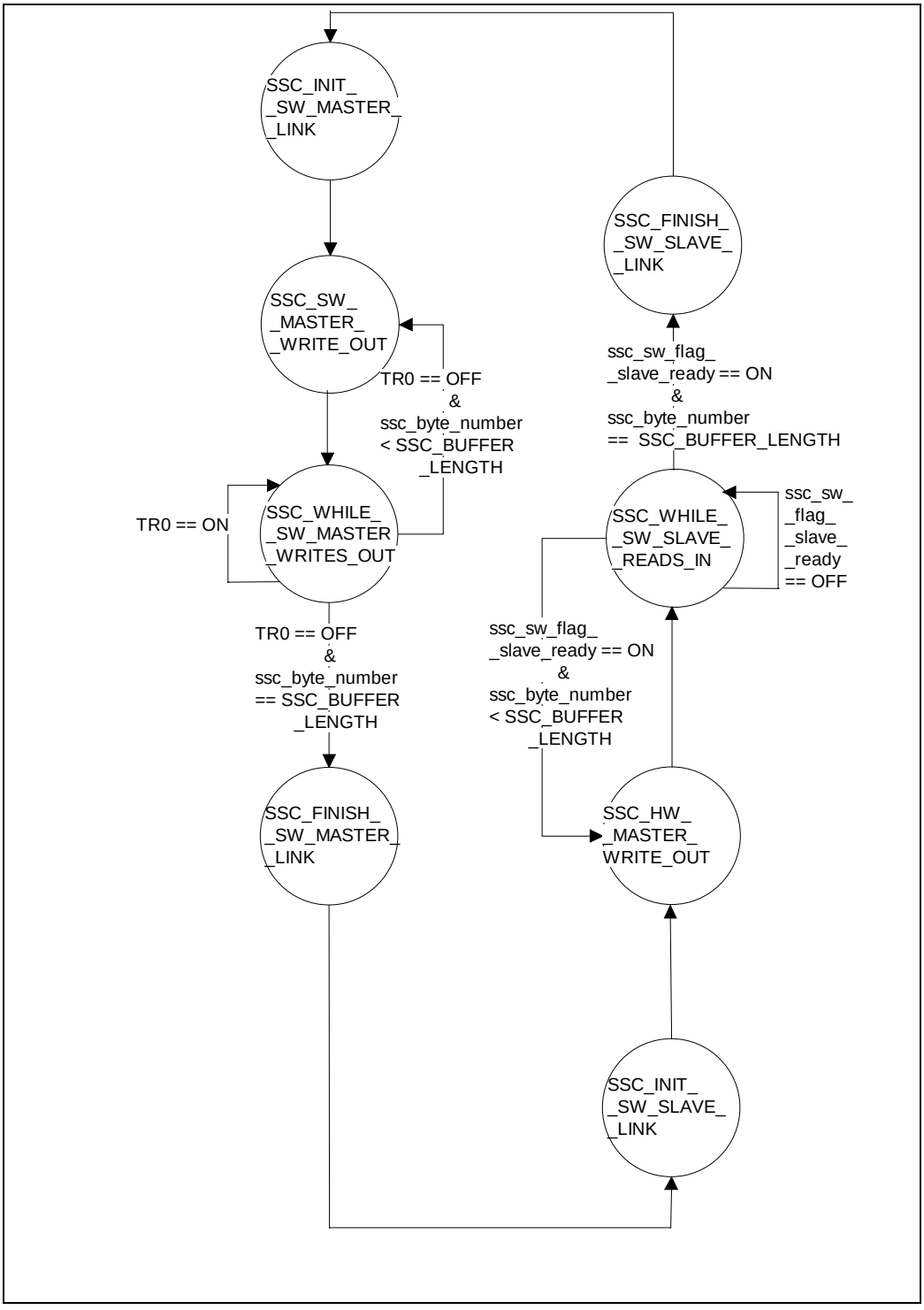

Figure 4
State Machine Diagram for test program ”ssc_test.c”

The first test case verifies the emulated SSC by a data transmission to an external source:

- In the first state ‘SSC_INIT_SW_MASTER_LINK’ the emulated SSC interface is initialized in Master Mode with the baud rate to be supported (15KBaud). As communication partner serves the on-chip HW-SSC which is set up in Slave Mode.

- The second state ‘SSC_SW_MASTER_WRITE_OUT’ starts the SW-SSC writing out a clock signal conform to selected Clock Phase and Clock Polarity.

- In the third state ‘SSC_WHILE_SW_MASTER_WRITES_OUT’ a flag is polled indicating the end of data transmission via the SW-SSC. User application code to be executed during the SW-SSC write out operation may be included here instead of wasting 8 bit times only for running a polling loop. After finishing the transmission of a whole message string containing a programmable number of bytes the state machine proceeds to the next state.

- The last state ‘SSC_FINISH_SW_MASTER_LINK’ disables all hardware and software modules required for data transmission. The state machine continues with the next test case.

In the second test case the communication is started with an altered transmission direction. The SW-SSC operates as Slave and receives a message string from the on-chip HW-SSC.

3.3 Emulation Subroutines

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

- 'ssc_init_sw_ssc' initializes all required auxiliary hardware modules like Timer and External Interrupt input by programming their control registers.

- 'ssc_start_sw_master_write()' prepares a data transmission via the SW-SSC by writing out the first data bit and the first clock state. A timer loaded with half a bit time of the desired baud rate is started in 'Auto-Reload Mode'. The timer load value for achieving 'half a bit time' is calculated by a formula presented in chapter 3.4.
  In addition, this subroutine has to be called if the software emulated SSC wants to read in a word in master mode, because the timer for the clock generation is started here.

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

- 'ssc_int0_interrupt_service()' is enabled by a SW-SSC initialization in 'Slave Mode' and is started by a 'Low' level at the SW-SSC-SCLK pin, which is externally wired to the interrupt0 pin. Because half duplex transmission is supported only the transmission direction is controlled by a Conditional Compiling Directive. Selecting a 'READ' directive produces code to sample the logic state of the SW-SSC-SRI pin as new input bit; otherwise code is generated which provides the SW-SSC-STO pin with a new output bit.

- 'ssc_timer0_interrupt_service()' is enabled in 'SW-SSC Master Mode' only. It generates a new clock edge at the SW-SSC-SCLK pin. The transmission direction is also controlled by a Conditional Compiling Directive. Selecting a 'READ' directive (SSC_SW_MASTER_DIRECTION_IS_READ) produces code to read in a new data bit from the SW-SSC-SRI pin according to Clock Phase initialization and SCLK edge count; otherwise code is generated which shifts out a new data bit to the SW-SSC-STO pin.
  A software emulated SSC which wants to make use of both directions on the fly may be modified easily at this point.

3.4 Baud Rate Calculation

Data transmission via an emulated SSC interface initialized in ‘Master Mode’ requires a baud rate generation by an internal timer. The transmission baud rate is determined by the clock oscillator base frequency and the selected prescaler factor.

The load and reload values for the timer emulating the SW-SSC baud rate generator are calculated as presented in the next formula:

$$\text{Half\_a\_Bit\_Time\_using\_timer0} = \frac{12*(256-\text{TL0})}{\text{fosc}}$$

The Timer Reload Register TH0 must be loaded with the same value.

Table 2
Calculated Timer0 Load and Reload Values for Master Mode

SSC Baud Rate fosc = 12 MHz	1 Bit Time	TL0, TH0 Load Value fosc = 12 MHz
15.0 K	66.66 us	0xDF
7.5 K	133.33 us	0xBE
3.7 K	266.66 us	0x7b

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 SSC is calculated by analyzing the compiler object module listing.

The next table presents load measurement results for a SSC emulation running with different baud rates and operation modes.

Table 3:
Load Measurement Values for a SSC emulation in Master Mode at SAB C513

Crystal Frequency	Baud Rate	Load
12 MHz	15.0 K	70.0%
12 MHz	7.5 K	35.0%
12 MHz	3.7 K	18.0%

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

Table 3:
Load Measurement Values for a SSC emulation in Slave Mode at SAB C513

Crystal Frequency	Baud Rate	Load
12 MHz	25 K	60%
12 MHz	15 K	36%
12 MHz	7.5 K	18%

3.6 Performance Limitations

The most severe limitation is seen in the timer interrupt service routine handling the Master mode at 15 KB baud rate. Every clock edge is generated by the timer ISR (twice per bit!) which also reloads the timer and reads in or writes out a data bit beginning with the MSB position.

Another fact which reduces the maximum baudrate of the application is the implementation in C. A solution in assembler would have much more performance. Of course, this solution would be not that easy understood like the solution in C code. So, it is advised to implement the CPU intensive routines in assembler if performance sensitive applications are used.

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 ssc_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 ‘ssc_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 selected for the SW-SSC are neighboured to the related HW-SSC pins and 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-SCLK (P1.2)	HW-SSC-SRI (P1.3)	HW-SSC-SLS (P1.5)
SW-SSC-SLK (P1.0)	X		
SW-SSC-STO (P1.1)		X	
SW-SSC-SLS (P1.6)			
Ext. Interrupt 0 (P3.2)	X		
Enable-HW-SSC-SLS (P1.7)			X

Note: SW-SSC-SRI and HW-SSC-STO are internally connected by sharing the same I/O pin (P1.4) and do not need any external wire.

Note: SW-SSC-SLS is directly controlled by the test state machine in file ssc_test.c and does not need any external wire.

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