SIEMENS

# Microcontrollers

# ApNote AP164901

☐or ☑ additional file
AP164901.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 C161V/K/O, but will fit to all C16x 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.
All files are available for Keil and Tasking C Cross Compiler.

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

| **AP1649 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 C16x microcontroller family provides only one on-chip fast synchronous serial communication 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 SW routines with a performance up to 50 KBaud in full duplex mode (or 100 KBaud in half duplex mode) and an overhead less than 75% at SAB C161O with 16 MHz. All files are available for Keil and Tasking C Cross Compiler. Due to the implementation in C this performance is not the limit of the chip. A pure implementation in assembler will result in a reduction of the CPU load and therefore increase the maximum speed of the interface.

Speaking about performance, it is strongly advised to have a close look at the assembler code generated by the different compilers. Moreover, at C16x architecture the speed of executing code strongly depends on the area where code and data are fetched from (external memory 16 bit data access, external memory 8 bit data access, Internal RAM, on-chip Flash, ...).

In addition, only a pin compatible solution is provided. 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 SSC via three external wires in different operation modes. It is based on the SAB C161O (Siemens 16 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:

- full duplex communication,
- baud rates up to 50 KBaud Mode @ SAB C161O with 16 MHz crystal,
- data width selection between 2 and 16 bit,
- clock phase and polarity control,
- master and slave operation capability,
- bit stream receive capability.

The following enumeration lists all functions of a SAB C16x on-chip SSC, which could not be cloned due to technical limitations or performance restrictions:

- phase error, baud rate error, transmit error, receive error check,
- automatic reset upon baud rate error,
- LSB first.

## 2.2 Required Resources

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

**Table 1**
**Resource Requirements**

| **Resource** | **Emulated SSC** |
|---|---|
| Number of required I/O pins | 3 |
| Number of interrupt pins | 1 |
| Interrupt Priority | high |
| Timer | T2 or T3 or T4 or ... |
| Additional On-Chip Modules | - |
| Program Memory (Emulation routines only) | 600 Words |
| Data Memory (Emulation routines only) | 20 Words |

### 2.3 External Routing

An external wire connecting the SW-SCLK line with the External1 Interrupt pin is required to activate the SW-SSC via a clock edge provided by the external communication partner. On test boards with C16X processor the on-chip SSC may also be used as ‘external party’.

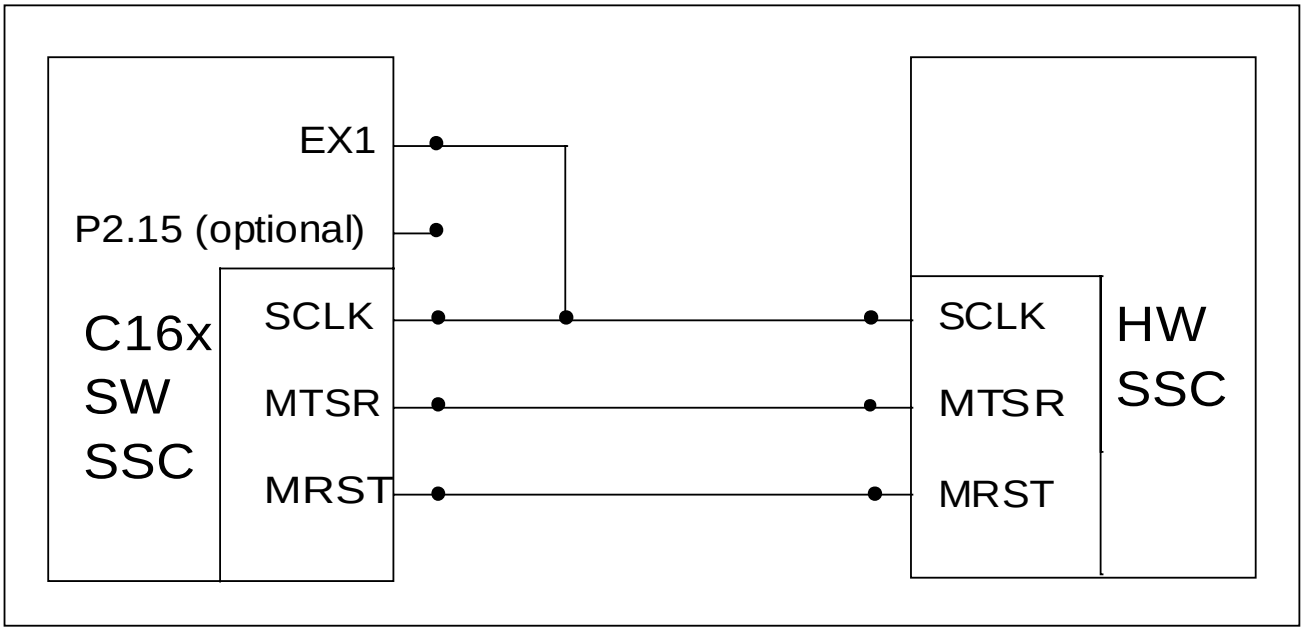


**Figure 1**
**External Routing of Transmission Lines**

## 2.4 Principles of Emulation

The algorithm required for emulating the data transmission depends on the operation mode.

### 2.4.1 SSC Master Mode

Figure 2 presents all auxiliary modules required for emulating the SSC interface in Master Mode.

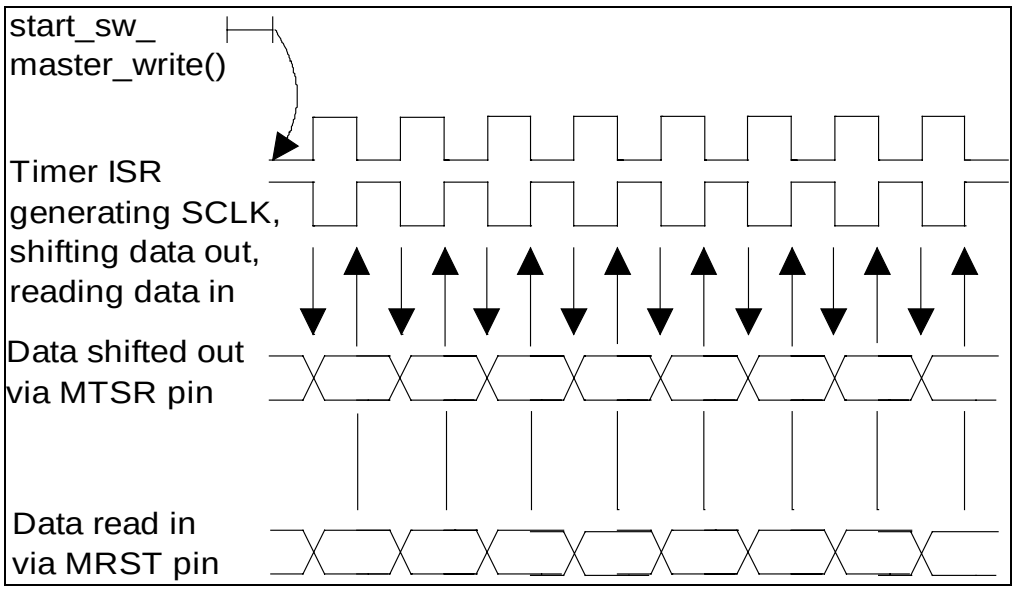


**Figure 2**
**Schematic Diagram of an emulated SSC operating in Master Mode**

A timer is loaded with half a bit time of the required baud rate. The associated interrupt service routine toggles the SW-SCLK pin and transmits one data bit to the SW-MTSR pin or receives one data bit from the SW-MRST pin. The timer is stopped after 2*n clock edges (n = number of bits to be handled by SW-SSC).

### 2.4.2 SSC Slave Mode

The Slave Mode is handled by an External Interrupt Pin triggered by rising and falling edges of the external Master Clock (Figure 3).

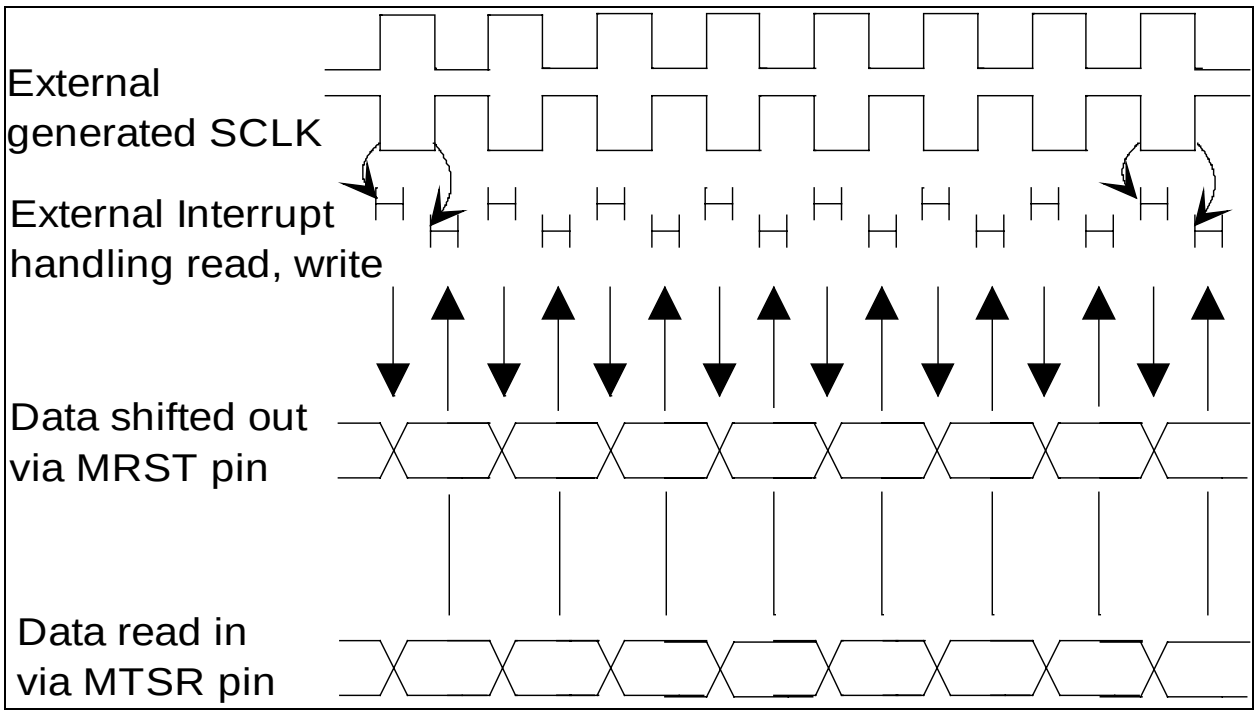


**Figure 3**
**Schematic Diagram of an emulated SSC operating in Slave Mode**

The first external SCLK edge generates a data shift out operation to the SW-MRST pin executed by the External Interrupt ISR; the next external clock edge starts a read operation from the SW-MTSR pin via the External Interrupt ISR. The External Interrupt is disabled after 2*n external clock edges (n = number of bits to be handled by SW-SSC).

# 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).

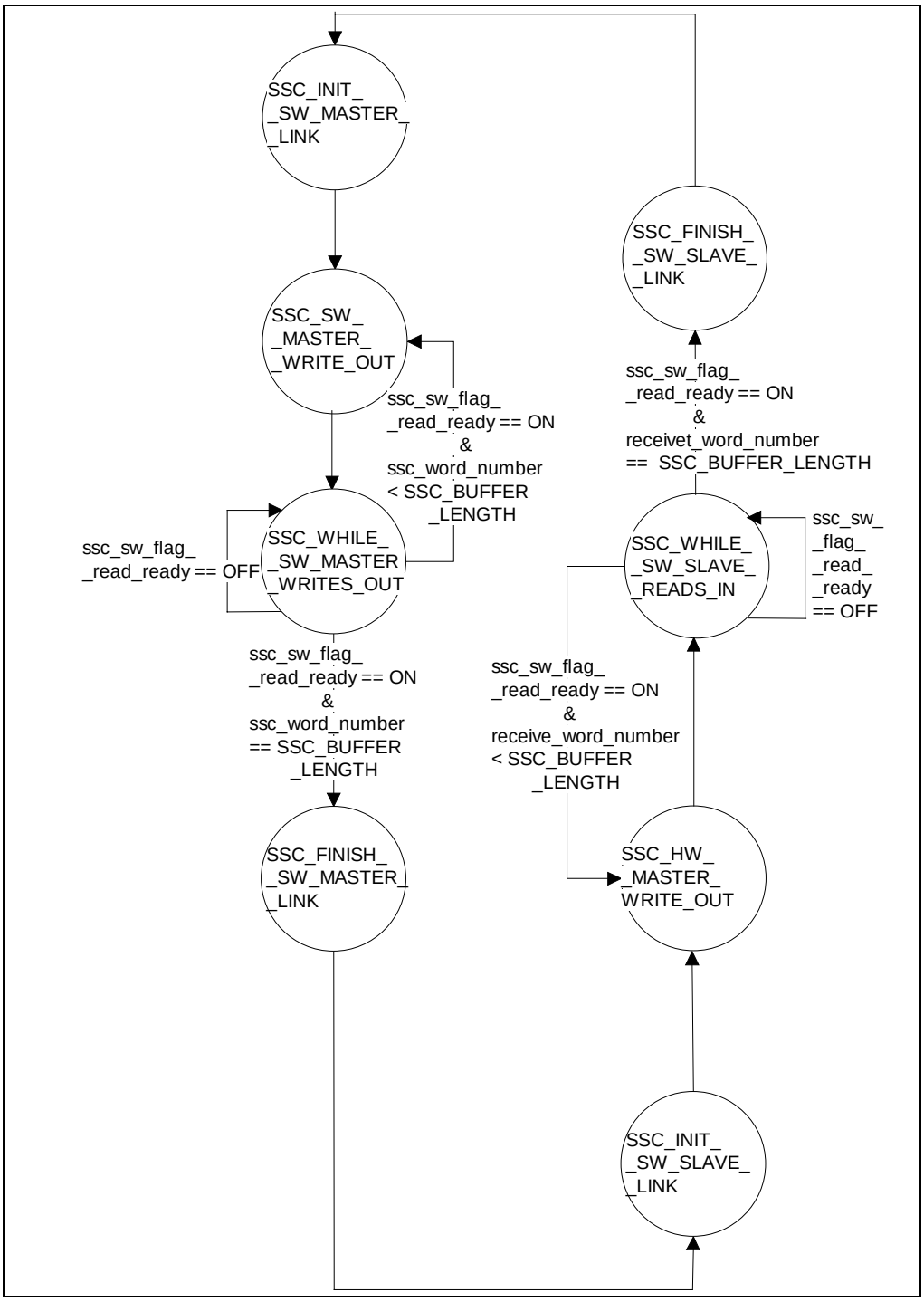


**Figure 4**
**State Machine Diagram for test program ”ssc_test.c”**

The first test case verifies the emulated SSC in Master Mode by a data exchange with an external communication partner:

- 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 (50 KBaud). As communication partner serves the on-chip HW-SSC which is set up in Slave Mode and in same baud rate.

- The second state ‘SSC_SW_MASTER_WRITE_OUT’ loads the on-chip HW-SSC with an information to be transferred and starts the SW-SSC after loading with a word to be transmitted.

- In the third state ‘SSC_WHILE_SW_MASTER_WRITES_OUT’ a flag is polled indicating the end of data reception via the SW-SSC. User application code to be executed during the SW-SSC read in operation may be included here instead of wasting 8 or 9 bit times only for running a polling loop. After finishing the transmission of a whole message containing a programmable number of words the state machine proceeds to the next state.

- The last state ‘SSC_FINISH_SW_MASTER_LINK’ disables all hardware modules required for data transmission.

In the second test case the communication is started with an altered operation mode. The SW-SSC operates as Slave and provides the on-chip HW-SSC with a message string clocked out by the external Master SCLK.

### 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, External Interrupt and the related port pins by programming their control registers. The variable *ssc_sw_ssccon* has to be handled in the same way like the C16x hardware special function register (SFR) SSCCON. This register contains all control bits which configure the SSC. In addition the variable *ssc_temp_sw_baudrate* sets half the time of the bitrate which is desired.

- 'ssc_start_sw_master_write()' prepares a data transmission via the SW-SSC in Master Mode by
  - forcing the SW-MTSR pin with the first bit to be transmitted,
  - driving the SW-SCLK pin with its start value according the selected clock phase and polarity configuration,
  - starting a timer loaded with 0.5 bit time.

- 'ssc_prepare_sw_slave_write()' prepares a data transmission via the SW-SSC in Slave Mode by
  - forcing the SW-MRST pin with the first bit to be transmitted if CLOCK_PHASE = 1 is selected,
  - driving the SW-SCLK pin with its start value if CLOCK_PHASE = 1 is selected.

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

- 'ussc_int1_interrupt_service()' is started in SW-SSC slave mode by an edge transition at the SW-SCLK pin, which is externally wired to the interrupt1 pin. According an even or odd clock edge number and the selected clock phase and polarity modes data bits are written out via SW-MRST pin or sampled in via SW-MTRS pin.

- 'ussc_timer2_interrupt_service()' is started in SW-SSC master mode. According an even or odd clock edge number and the selected clock phase and polarity modes data bits are written out via SW-MTRS pin or sampled in via SW-MRST pin.

### 3.4 Baud Rate Calculation

The load value for the SW-SSC baud rate generator (implemented with auxiliary timer T2) is directly calculated from the load value of a HW-SSC running at same baud rate (subroutine ‘ssc_init_sw_ssc() ). The required value for the HW-SSC baud rate generator is stored as a constant in file ‘ssc_defi.h’ and calculated by the formula:

$$\text{SSC\_HW\_SSC\_BAUDRATE\_xxxx} = \left(\frac{\text{fosc}}{2 * \textit{Desired\_Baud\_Rate}}\right) - 1$$

### 3.5 Load Measurement

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

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 and emulating routines is calculated by analyzing the compiler object module listing. The 'Total amount of time for transmitting / receiving n bytes' can be easily calculated by multiplying the number of bits to be transmitted with the bit period time related to selected baud rate.

For double checking purpose test statements are included in emulation subroutines indicating the begin and the end of an interrupt service or emulation routine by switching port pin P2.15 to 'Low' and to 'High' state. This port pin may be scanned by an oscilloscope. However, the pulse width measured at this test pin does not exactly represent the CPU load caused by a subroutine execution. Even if the statement 'ssc_test_pin = OFF' is found at the very beginning of a C coded subroutine or the statement 'ssc_test_pin = ON' is seen as last statement in C source code, several stack operations to be executed are found in the compiler's object module listing before or after the test pin is affected (PUSH register x, PULL register x).

The next table presents load calculation results for a SSC emulation via SW routines running with different baud rates.

**Table 3:**
**Load Measurement Values for a SSC emulation via SW Routines at SAB C161O**

| Crystal Frequency | Baud Rate | Load in Master Mode | Load in Slave Mode |
|---|---|---|---|
| 16 MHz | 1.0 KB | 1.5% | 1.6% |
| 16 MHz | 10.0 KB | 15.0% | 15.6% |
| 16 MHz | 50.0 KB | 75.0 % | 78.0% |

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

### 3.6 Performance Limitations

The most severe limitation is seen in the timer interrupt service routine handling the Master mode at 50 KB baud rate. Every clock edge is generated by the timer ISR which also reloads the timer and reads in or writes out a data bit beginning with the MSB position.

A half duplex operation mode (deleting all statements in the Timer and External Interrupt ISR executed for READ or WRITE direction) would double the maximum baud rate or would halve the CPU overhead at unchanged baud rate.

Another fact which reduces the maximum baudrate of the application is the implementation in C. A solution in assembler would have a positive impact in the 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.

Speaking about performance, it is strongly advised to have a close look at the assembler code generated by the different compilers. Moreover, at C16x architecture the speed of executing code strongly depends on the area where code and data are fetched from (external memory 16 bit data access, external memory 8 bit data access, Internal RAM, on-chip Flash, ...).

### 3.7 Debugging Support Pins

To support program debugging some signals are provided to trigger an oscilloscope:

- a falling edge at **P2.15** indicates the start of an emulation subroutine or an interrupt service routine; a rising edge indicates the end.
- a clock edge at **P3.5** (SCLK) samples in the logic state provided at **P3.10** (SW-SSC-MRST in Master Mode) or writes out a new data bit via **P3.11** (SW-SSC-MTSR in Master Mode).
- a clock edge at **P3.5** (SCLK) samples in the logic state provided at **P3.11** (SW-SSC-MTSR in Slave Mode) or writes out a new data bit via **P3.10** (SW-SSC-MRST in Slave Mode).

### 3.8 Make File

The file ssc_make.bat contains all statements to start the Keil or Tasking C cross compiler, linker and locator (Versions Keil: C166 V3.05a, L166 V3.05, A166 V3.05. Versions Tasking: C166 V6.0r2, L166 V6.0r2, A166 V6.0r2). 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.

### 3.9 Support of KitCON161 Evaluation Board

The KitCON-161 Evaluation Board is a starter kit (order at Siemens Semiconductors www) which helps for a general approach to the SAB C161. Generally speaking it is a printed circuit board which lets you load software down via the PC to the SAB C161. After that the SAB C161 executes that code out of the on-board flash memory.

Using the ”test shell” ssc_test.c the SW-SSC of the SAB C161 communicates with the on chip HW-SSC of the SAB C161 device.

The executable program (Keil: sw_ssc.h86, Tasking: sw_ssc.hex) can be directly downloaded to the KitCON161 evaluation board configured in address mode 1 (jumper 4 = open).

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-161 Evaluation Board**

| | HW-SSC-SCLK (P3.13) | HW-SSC-MRST (P3.8) | HW-SSC-MTSR (P3.9) |
|---|---|---|---|
| SW-SSC-SCLK (P3.5) | X | | |
| SW-SSC-MRST (P3.10) | | X | |
| SW-SSC-MTSR (P3.11) | | | X |
| Ext. Interrupt 1 (P2.9) | X | | |

After setting jumper 9 to position (2+3) and pressing the restart button the test program runs in an endless loop.