SIEMENS

Microcontrollers

ApNote AP165001

☐or ☑ additional file
AP165001.EXE available

Emulating an asynchronous serial interface (ASC0) via software routines

Abstract:

The solution presented in this paper and in the attached source files emulates the most important ASC 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

AP1650 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 asynchronous serial communication channel (ASC). If a second ASC 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 ASC functions by using performance and resource balanced SW routines up to 38,4 KBaud in full duplex mode with an overhead less than 40% or up to 100 KBaud in half duplex mode with an overhead less than 55% 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-ASC and the emulated SW-ASC 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 ASC.

2 General Operation and Hardware Environment

2.1 Supported Features

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

- full and half duplex communication,
- baud rates up to 38.4 KBaud in full duplex and 100 KBaud in half duplex mode @ SAB C161O with 16 MHz crystal
- 7 bit asynchronous data frames with variable baud rate, even/odd parity and one or two stop bits,
- 8 bit asynchronous data frames with variable baud rate and one ore two stop bits,
- 8 bit asynchronous data frames with variable baud rate, even/odd parity and one ore two stop bits,
- 9 bit asynchronous data frames with variable baud rate and one ore two stop bits,.
- bit stream receive capability.

The following enumeration lists all functions of a SAB C16x on-chip ASC0, which could not be cloned due to technical limitations or performance restrictions:

- 8 bit synchronous operation,
- 8 bit data frames plus wake up bit,
- framing check,
- loop back mode,
- bit stream write capability.

2.2 Required Resources

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

Table 1
Resource Requirements

Resource	Emulated ASC
Number of required I/O pins	2
Number of interrupt pins	1
Interrupt Priority	very high
Timer	T2 and T4
Program Memory (Emulation routines only)	820 Words
Data Memory (Emulation routines only)	20 Words

2.3 External Routing

An external wire connecting the SW-ASC data input with the External1 Interrupt pin is required to activate the SW-ASC via a Start Bit transmitted by the external communication partner. On test boards with C16X processor the on-chip HW-ASC may also be used as ‘external party’.

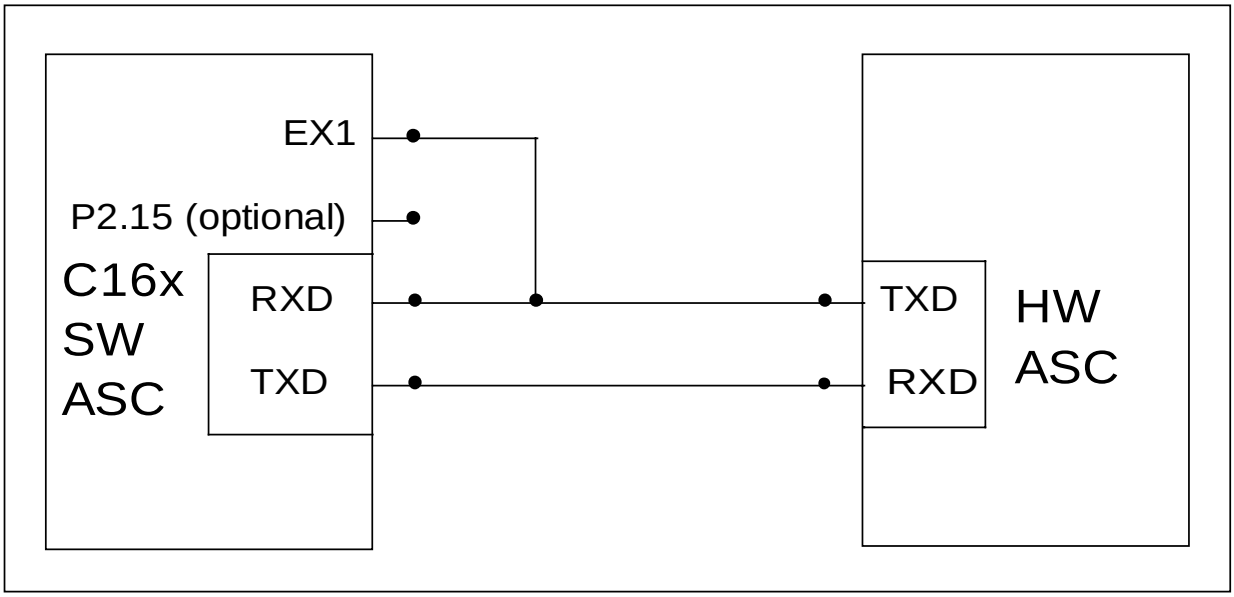

Figure 1
External Routing of Transmission Lines

2.4 Principles of Emulation

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

2.4.1 ASC Write

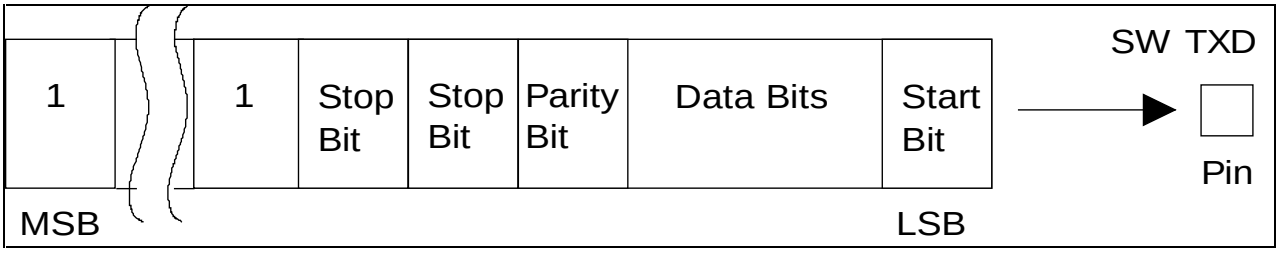

Figure 2
Schematic Diagram of Emulating an ASC Write Operation using a Timer ISR

An ASC Write is prepared by loading the information to be transmitted into a temporary buffer. After inserting parity, stop and start bits the whole buffer is shifted out within an interrupt service routine for a timer loaded with 1.0 bit time of the selected baud rate.

2.4.2 ASC READ

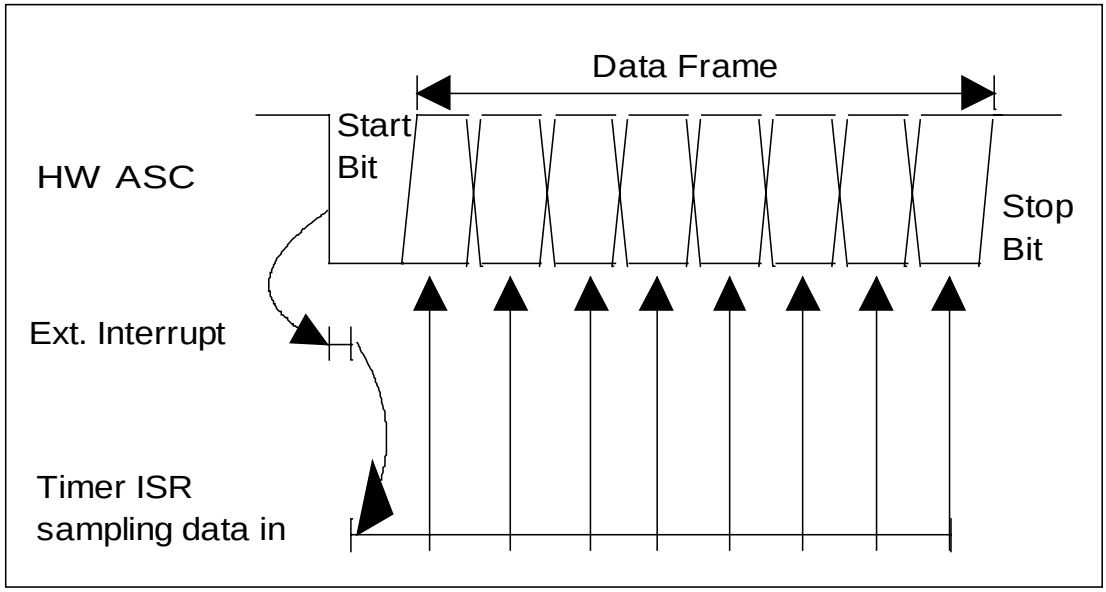

Figure 3
Schematic Diagram of Emulating an ASC Read Operation using a Timer ISR

An ASC READ is initiated by an ASC Start Bit arriving at the SW-RXD input pin, which is externally connected to an External Interrupt pin. The correlated interrupt service routine starts a timer loaded with 1.5 bit time of the required baud rate. At the beginning of the related interrupt service routine the timer is reloaded with 1.0 bit time and the logic state of the SW-RXD pin is sampled in.

The final ASC Stop Bit provided by the external transmitter is completely ignored to gain some time for preparing reception of the next byte of a continuous input data stream.

3 ASC Emulation Software Description

3.1 Software Structure

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

- usa_emul.c contains all low level software drivers (subroutines and interrupt services) to emulate the ASC by SW routines. This file may be directly added to the user's application software directory and may be included in his make file.
- usa_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 ASC functions into his own application code segments.
- usa_defi.h holds all definitions and declarations related to the emulation software (usa_test.c, usa_emul.c).

3.2 Main Program

The main program (usa_test.c) is implemented as a state machine and handles several test cases (Figure 4).

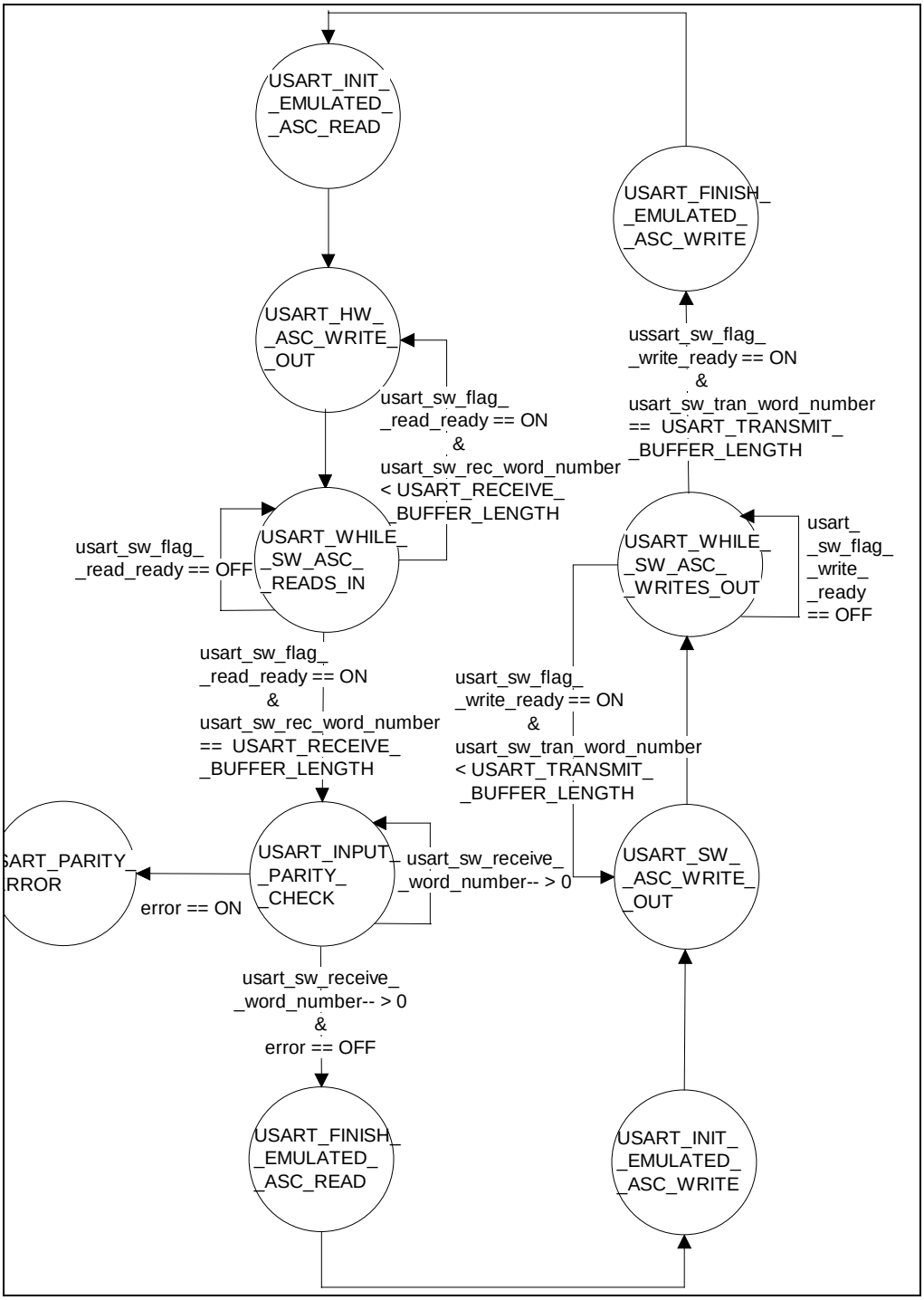

Figure 4
State Machine Diagram for test program ”usa_test.c”

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

- In the first state ‘USART_INIT_EMULATED_ASC_READ’ the emulated ASC interface is initialized with the baud rate to be supported (100 KBaud half duplex). As communication partner serves the on-chip HW-ASC which is set up in same baud rate.
- The second state ‘USART_HW_ASC_WRITE_OUT’ starts the on-chip HW-ASC.
- In the third state ‘USART_WHILE_SW_ASC_READS_IN’ a flag is polled indicating the end of data reception via the SW-ASC. User application code to be executed during the SW-ASC 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 state ‘USART_INPUT_PARITY_CHECK’ analyzes the message string received by SW-ASC. If a difference between received and calculated parity bit is detected the state machine goes to the error state ‘USART_PARITY_ERROR’ and stops.
- The last state ‘USART_FINISH_EMULATED_ASC_READ’ disables all hardware modules required for data transmission.

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

3.3 Emulation Subroutines

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

- 'usart_init_sw_asc ()' initializes all required auxiliary hardware modules like timers, External Interrupt and the related port pins by programming their control registers. The variable *usart_temp_sw_control_register_word* has to be handled in the same way like the C16x hardware special function register (SFR) S0CON. This register contains all control bits which configure the ASC. In addition the variable *usart_temp_sw_baudrate* sets one bit time of the bitrate which is desired.

- 'usart_start_sw_write()' prepares a data transmission via the SW-ASC by
  - copying the information to be transmitted into a temporary word,
  - inserting a "0" as Start Bit at LSB position.
  - filling up this word buffer with leading "1's" (partly used as Stop Bits),
  - calculating and inserting a parity bit (if required),

  Afterwards, the WRITE timer is enabled and started.

- 'usart_parity_bit_calculation()' calculates the parity bit in respect to odd or even parity mode and 8 or 9 bit data frame length.

- 'usart_analyse_input_word()' compares the transmitted parity bit with the calculated parity bit of a received information.

- 'usart_disable_sw_asc()' disables all required auxiliary hardware modules like timers and interrupts by setting their control registers respectively.

- 'usart_int1_interrupt_service()' is started by a 'Low' level at the SW-ASC-RXD pin, which is externally wired to the interrupt1 pin. The 'Low' level is interpreted as an ASC Start Bit announcing the arrival of further data bits. Therefore a timer is started to activate a SW-ASC read operation after an 1.5 bit time interval. Finally the interrupt1 service is disabled to avoid undesired reactions to incoming '0' data bits.

- 'usart_read_timer_interrupt_service()' starts with a self compensating timer reload. The Timer Interrupt Flag is set when the timer 'underflows'. The related interrupt service routine is entered while the timer continues counting down. The current timer value (0xFFFx) represents the time already spent in ISR. Adding 1 bit time to current timer value compensates this interrupt service delay time and enables next sw-rxd-pin sampling in 1 bit time distance exactly.
  A READ operation begins in the middle of the first received data bit. All incoming bits are stored in 'usart_sw_asc_receive_buffer' arranged in correct bit order (LSB first) and the Transfer Completion Flag is set. The External Interrupt is enabled again to be prepared for handling the Start Bit of the next data byte reception. The final stop bit is completely ignored, which decreases the CPU overhead and saves time to get prepared for the Start Bit of the next byte of an input message stream.

- ‘usart_write_timer_interrupt_service()’ handles a WRITE operation which is started with a self compensating timer reload before writing out the LSB of the ‘usart_output_word’. Every new ISR entry continues with bits of ‘usart_output_word’in ascending order.

3.4 Baud Rate Calculation

Data transmission via an ASC 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 load values for the HW-ASC baud rate generator is stored as a constant in file ‘usa_defi.h’ and calculated by the formula:

$$\text{HW\_ USART\_ xxxx\_ BAUD} = \left(\frac{\text{fosc}}{2 * 16 * \textit{Desired\_ Baud\_ Rate}}\right) - 1$$

The timer load and reload values (required for emulating a SW-ASC baud rate generator) are calculated automatically from this HW-ASC baud rate generator load value by the subroutine ‘usart_init_sw_asc ()’.

For read operations the initial load value of 1.5 bit time is corrected by a constant in file ‘usa_defi.h’ (USART_INTERRUPT_DELAY_CORRECTION), which takes into account

- the Interrupt Response Time for External Interrupt1 after receiving a Start Bit,
- the Interrupt Response Time for ‘READ Timer underflow’ including the execution time for all statements in the corresponding interrupt service routine before sampling in the first data bit.

The exact value for ‘USART_INTERRUPT_DELAY_CORRECTION’ may be extracted by analyzing the assembler program listing or by checking bit pulse signals with an oscilloscope.

Attention: The **‘**USART_INTERRUPT_DELAY_CORRECTION’ value depends on the clock generator frequency.

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 (including Start and Stop Bits) 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 macro 'reset_test_pin_latch()' is found at the very beginning of a C coded subroutine or the macro 'set_test_pin_latch()' 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 an ASC emulation via SW routines running with different baud rates (data frames without parity bit).

Table 3:
Load Measurement Values for an ASC emulation via SW routines (without parity bit) at SAB C161O

Crystal Frequency	Baud Rate	Half Duplex Write Load	Half Duplex Read Load	Full Duplex Load
16 MHz	19.2 KB	10.4%	9.2%	19.6%
16 MHz	38.4 KB	20.7%	18.4%	39.1%
16 MHz	100.0 KB	54.0 %	48.0%	-

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

3.6 Performance Limitations

The most severe limitation is seen in half duplex READ mode at 100 KB baud rate. The Read Timer interrupt service routine requires an execution time near 1 bit time. At the next higher baud rate (125 KBaud) the self calibrating timer reload mechanism fails after some ISR entries and loads the timer with a ‘negative’ value (0xFFFx).

Attention: The Write Timer always requires the highest interrupt priority within the system to generate a bit stream with a very accurate ASC timing.

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.
- data bits to be sampled in by the SW-ASC are provided at **P3.8** (SW-ASC-RXD); data bits to be written out by the SW-ASC are provided at via **P3.9** (SW-ASC-TXD).
- a ‘LOW’ state at **P2.11** indicates detection of a parity read error.

3.8 Make File

- The file usa_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 ‘usa_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” usa_test.c the SW-ASC of the SAB C161 communicates with the on chip HW-ASC of the SAB C161 device.

The executable program (Keil: sw_asc.h86, Tasking: sw_asc.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-ASC are neighboured to the related HW-ASC 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-ASC-RXD (P3.11)	HW-ASC-TXD (P3.10)
SW-ASC-TXD (P3.9)	X	
SW-ASC-RXD (P3.8)		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.