Application Note, V 1.0, Feb. 2004

AP16051

SAB C161K/V/O

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

Microcontrollers

Never stop thinking.

Revision History:	2004-02	**V 1.0**
Previous Version:	-	
Page	Subjects (major changes since last revision)	
All	Updated Layout to Infineon Corporate Design, updated release to 1.0, Content unchanged!	

Controller Area Network (CAN): License of Robert Bosch GmbH

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Edition 2004-02

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Table of Contents **Page**

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 the on-chip synchronous serial communication channel (SSC) with a performance up to 125 KBaud in half duplex mode and an overhead less than 12% 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 SSC-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 ASC to be emulated by software routines and the SSC:

- half duplex communication,
- baud rates up to 125 KBaud @ 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,
- full duplex communication,
- bit stream write capability.

2.2 Required Resources

To emulate the ASC 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 ASC
Number of required I/O pins	2
Number of interrupt pins	1
Interrupt Priority	high
Timer	T2 or T3 or T4 or ...
Additional On-Chip Modules	SSC
Program Memory (Emulation routines only)	650 Words
Data Memory (Emulation routines only)	20 Words

2.3 External Routing

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

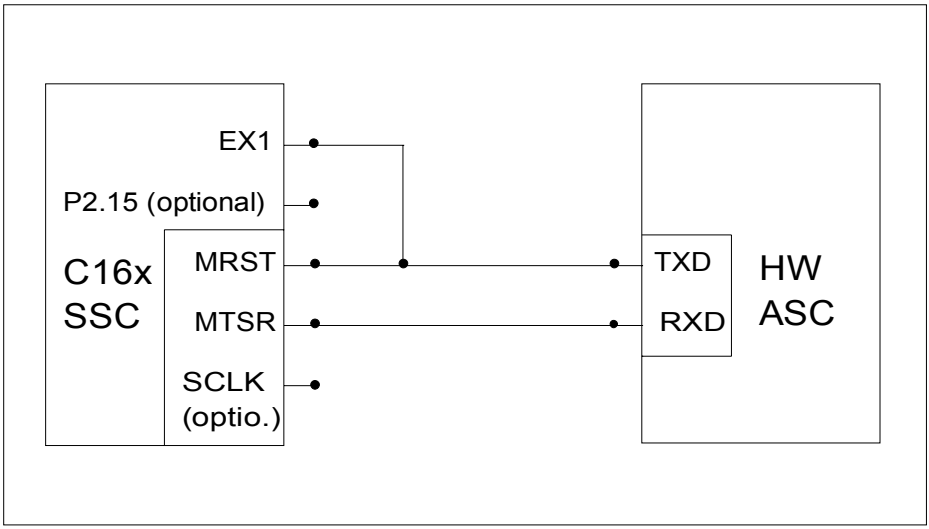

Figure 1 External Routing of Transmission Lines

2.4 Principles of Emulation

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

2.4.1 ASC Write

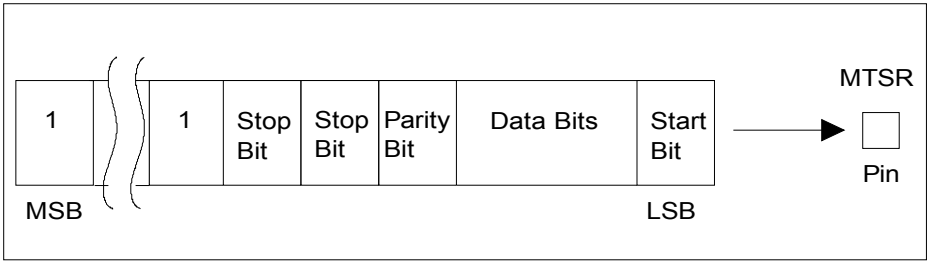

Figure 2 Schematic Diagram of Emulating an ASC Write Operation using the On-Chip SSC

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 copied to the SSC Transmit Buffer, which immediately starts the data transmission with the configured SSC baud rate and operation mode (Master Mode, LSB first, Data Width Selection = 1 Start Bit + n Data Bits + 1 Parity Bit + m Stop Bits).

The complete data transfer is automatically handled by the SSC-HW without any SW support; the CPU overhead for preparing Start, Stop and Parity Bits is negligible.

2.4.2 ASC READ

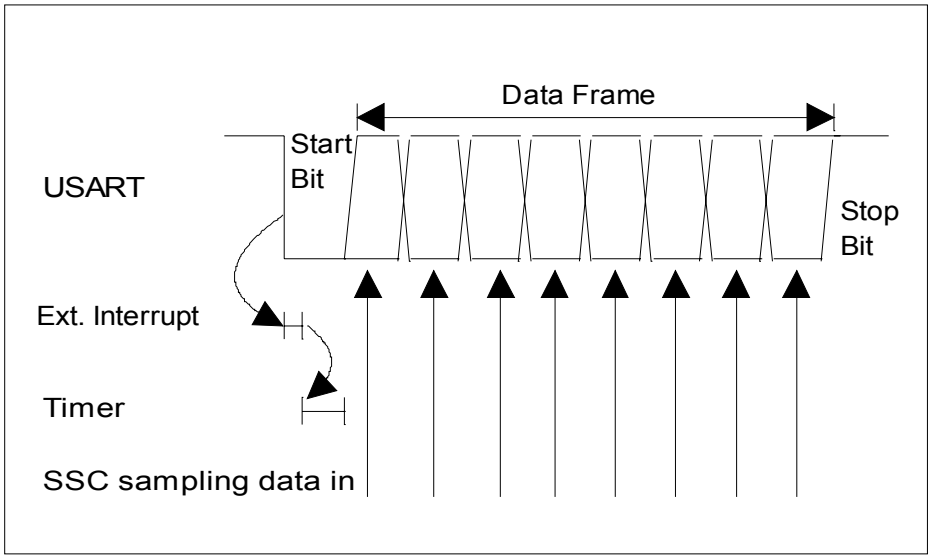

Figure 3 Schematic Diagram of Emulating an ASC Read Operation using the On-Chip SSC

An ASC READ is initiated by an ASC 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 0.5 bit time of the required baud rate.

The timer interrupt service routine starts the SSC in ‘Master Mode’ by writing a dummy byte (0xFFFF) into the SSCTB buffer, which simultaneously starts the SSC receive shift register sampling n data bits. Although the dummy byte 0xFFFF may be shifted out without triggering the external communication partner’s receive HW (because the transmitted message byte does not contain any ‘0’ bit to be interpreted as Start Bit), the MTSR output pin will be configured as input to avoid any noise transmission to the external communication partner.

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:

- uss_emul.c contains all low level software drivers (subroutines and interrupt services) to emulate the ASC 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 ASC 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).

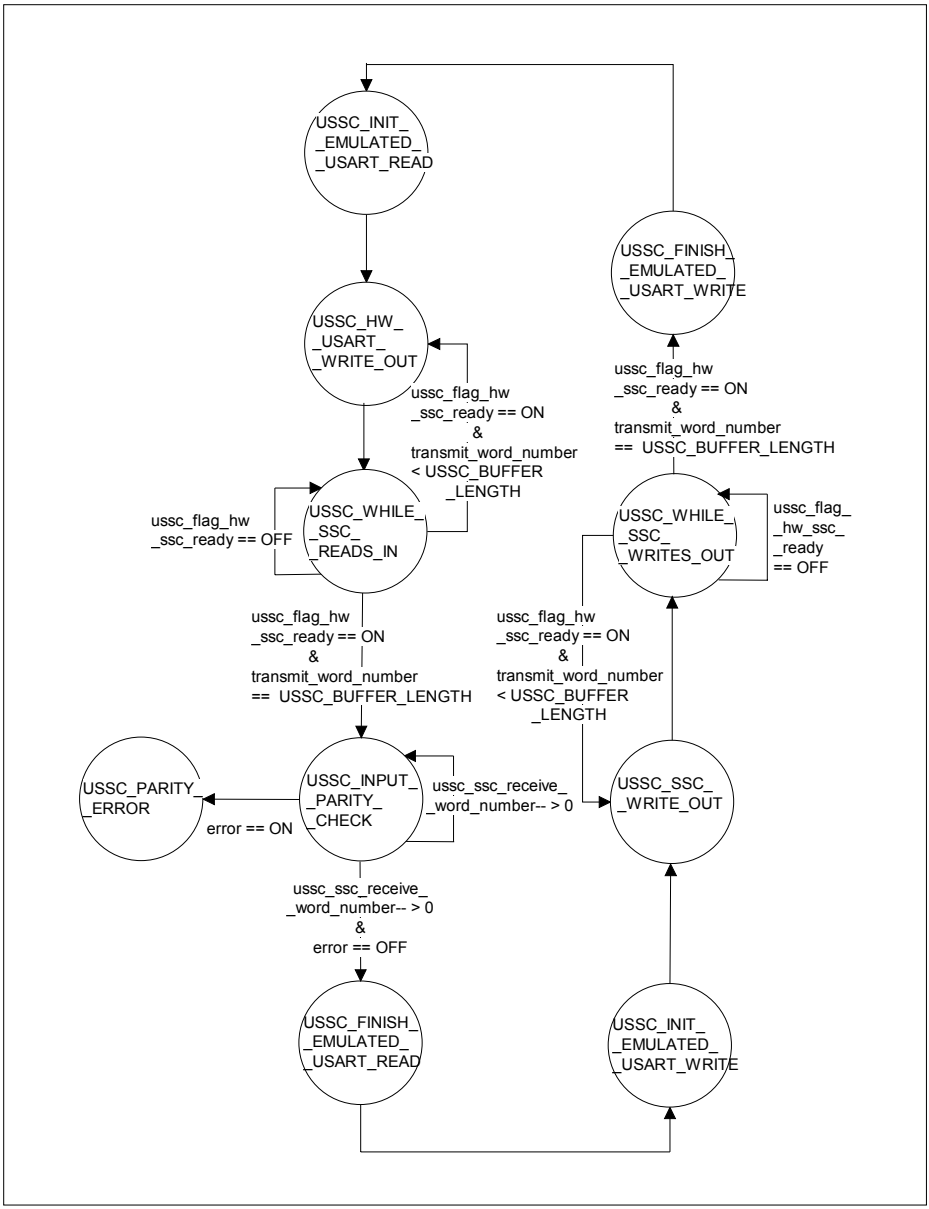

Figure 4 State Machine Diagram for test program ”uss_test.c”

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

- In the first state 'USSC_INIT_EMULATED_USART_READ' the emulated ASC interface is initialized with the baud rate to be supported (125 KBaud). As communication partner serves the on-chip HW-ASC which is set up in same baud rate.
- The second state 'USSC_HW_USART_WRITE_OUT' starts the on-chip HW-ASC.
- 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 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 'USSC_INPUT_PARITY_CHECK' analyzes the message string received by SSC. If a difference between received and calculated parity bit is detected the state machine goes to the error state 'USSC_PARITY_ERROR' and stops.
- The last state 'USSC_FINISH_EMULATED_USART_READ' 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-ASC with a message string.

3.3 Emulation Subroutines

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

- 'ussc_init_emulation_hw ()' initializes all required auxiliary hardware modules like Timer, External Interrupt, HW-SSC and the related port pins by programming their control registers.
  The variable *ussc_temp_hw_usart_control_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 *ussc_hw_usart_baudrate* sets one bit time of the bitrate which is desired.
  The SSC-MTSR output pin is always initialized as input (**even for WRITE operations!**) which prevents a noise transmission to the receiver of the external ASC in case of READ operations and suppresses (in case of a WRITE operation) the output of the random 0/1 SSC data line state until the first master clock edge is generated.

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

  Afterwards, the SSC is enabled and started by copying the temporary word buffer into the SSCTB buffer, but the MTSR output pin remains in input state.
- ‘ussc_parity_bit_calculation()’ calculates the parity bit in respect to odd or even parity mode and 8 or 9 bit data frame length.
- ‘ussc_analyse_input_word()’ compares the transmitted parity bit with the calculated parity bit of a received information.
- ‘ussc_disable_emulation_hw()’ disables all required auxiliary hardware modules like Timer, External Interrupt input and the SSC by setting their control registers respectively.
- ‘ussc_int1_interrupt_service()’ is started by a ‘Low’ level at the SSC-MRST 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 SSC read operation after a 0.5 bit time interval. Finally the interrupt1 service is disabled to avoid undesired reactions to incoming ‘0’ data bits.
- ‘ussc_timer2_interrupt_service()’ enables the HW-SSC and starts a READ operation by writing a dummy word into the SSCTB output buffer. One bit time later, the SSC will respond with a master clock edge for sampling data in. Timer2 is stopped.
- ‘ussc_hw_ssc_transmit_interrupt_service()’ is called immediately after writing out the first data bit to be transmitted by SSC. Therefore this interrupt service routine is used to configure the MTSR pin as output, which enables (after suppressing the leading random 0/1 SSC data line state) a data transmission to the external ASC receiver. The pulse width of the first SSC data bit to be sent out is reduced by about 1.8 µs.

  (To avoid this the following workaround is proposed: instead of sending the startbit as first bit, a leading ”1” should be sent. The period of this dummy ”1” is reduced by 1.8 usec and the startbit will have the correct bittime.)
- ‘ussc_hw_ssc_receive_interrupt_service()’ is called after completion of a complete SSC data transfer operation. (Remember: at SSC sending data and SSC receiving data is always done parallel). The received data word is stored in the array ‘ussc_ssc_receive_buffer’ 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.

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 SSC baud rate generator and the auxiliary timer (T2) are directly calculated from the load value of an ASC running at same baud rate (subroutine 'ussc_init_emulation_hw()). The required value for the ASC baud rate generator is stored as a constant in file 'uss_defi.h' and calculated by the formula:

$$\text{USSC\_HW\_USART\_xxxx\_BAUD} = \left(\frac{f_{OSC}}{2 * 16 * \text{Desired\_Baud\_Rate}} \right) - 1$$

For SSC read operations the load value of timer 2 (0.5 bit time delay) is corrected by a constant in file 'uss_defi.h' (USSC_INTERRUPT_DELAY), which takes into account

- the Interrupt Response Time for External Interrupt1 after receiving a Start Bit,
- the Interrupt Response Time for Timer2 underflow including the execution time for all statements in the corresponding interrupt service routine before starting a SSC operation.

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

Note: The 'USSC_INTERRUPT_DELAY' 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 the on-chip SSC running with different baud rates (data frames without parity bit).

Table 2 Load Measurement Values for an ASC emulation via the on-chip SSC (without parity bit) at SAB C161O

Crystal Frequency	Baud Rate	Write Load	Read Load
16 MHz	19.2 KB	1.6%	1.8%
16 MHz	38.4 KB	3.1%	3.6%
16 MHz	125.0 KB	10.0 %	12.0%

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

3.6 Performance Limitations

The most severe limitation is seen in READ mode at 125 KB baud rate. The incoming Start Bit triggers an external interrupt pin and the related interrupt service routine starts a timer loaded with ‘0.5 bit time - USSC_INTERRUPT_DELAY’. At 125 KB the USSC_INTERRUPT_DELAY value is equal 0.5 bit time and the timer is loaded with ”0”.

The next limitation is seen in WRITE mode at 125 KB baud rate. The ASC Start Bit sent out by the SSC is shortened by 1.8 µs due to the delayed configuration of the SSC-MTSR pin as output in the ‘ussc_hw_ssc_transmit_interrupt_service()’ subroutine. The receiving external ASC will not be able to sample in the first data bit at 0.5 bit period time; due to the shortened Start Bit sampling is done at 0.7 bit period time by the external ASC.

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 falling edge at **P3.13** (SCLK) samples in the logic state provided at **P3.8** (SSC-MRST); a rising edge writes out a new data bit via **P3.9** (SSC-MTSR).
- a 'LOW' state at **P2.11** indicates detection of a parity read error.

3.8 Make File

The file uss_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 'uss_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” uss_test.c the HW-SSC of the SAB C161 communicates with the on chip HW-ASC of the SAB C161 device.

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

The port pins of the HW-SSC 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 3 Port pins to be externally wired on KitCON-161 Evaluation Board

	HW-SSC-MRST (P3.8)	**HW-SSC-MTSR (P3.9)**
HW-ASC-TXD (P3.10)	X	
HW-ASC-RXD (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.

http://www.infineon.com

Published by Infineon Technologies AG