SIEMENS

Microcontrollers

ApNote AP083101

☐or ☑ additional file
AP083101.EXE available

Emulating an asynchronous serial interface (USART) via software routines

Abstract:

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

1 Introduction 3
2 General Operation and Hardware Environment 4
2.1 Supported Features 4
2.2 Required Resources 5
2.3 External Routing 6
2.4 Principles of Emulation 7
2.4.1 USART Write 7
2.4.2 USART READ 8
3 USART Emulation Software Description 9
3.1 Software Structure 9
3.2 Main Program 10
3.3 Emulation Subroutines 12
3.4 Baud Rate Calculation 13
3.5 Load Measurement 15
3.6 Performance Limitations 16
3.7 Make File 17
3.8 Support of KitCON-513 Evaluation Board 18

AP0831 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 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 optimized SW routines with a performance up to 19.2 KBaud in half duplex mode and an overhead less than 63% at SAB C513 with 11.059 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 SW-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:

- 8 bit data frames with variable baud rate (1 start bit, 1 stop bit),
- half duplex communication,
- baud rates between 1.2 and 19.2 Kbaud @ SAB C513 with 11.059 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 USART with fixed baud rate $f_{osc}/32$, $f_{osc}/64$
- 9 bit data frames with variable baud rate,
- multiprocessor communication feature,
- full duplex communication.

2.2 Required Resources

To emulate the USART interface by a set of software routines 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 or T1 (Auto Reload)
Program Memory (Emulation routines only)	210 Byte
Data Memory (Emulation routines only)	8 Byte

2.3 External Routing

An external wire connecting the SW-USART data input with the External0 Interrupt pin is required to activate the emulation routines 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’.

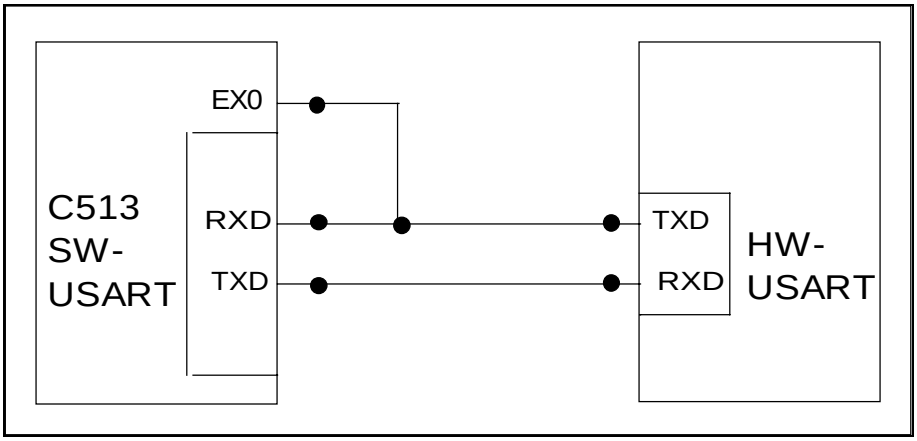

Figure 1
External Routing of Transmission Lines

2.4 Principles of Emulation

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

2.4.1 USART Write

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

The timer interrupt service routine writes out the LSB bit of the output byte buffer to be transmitted and executes a shift operation preparing a new LSB bit. The write operation is finished by the output of an USART Stop Bit.

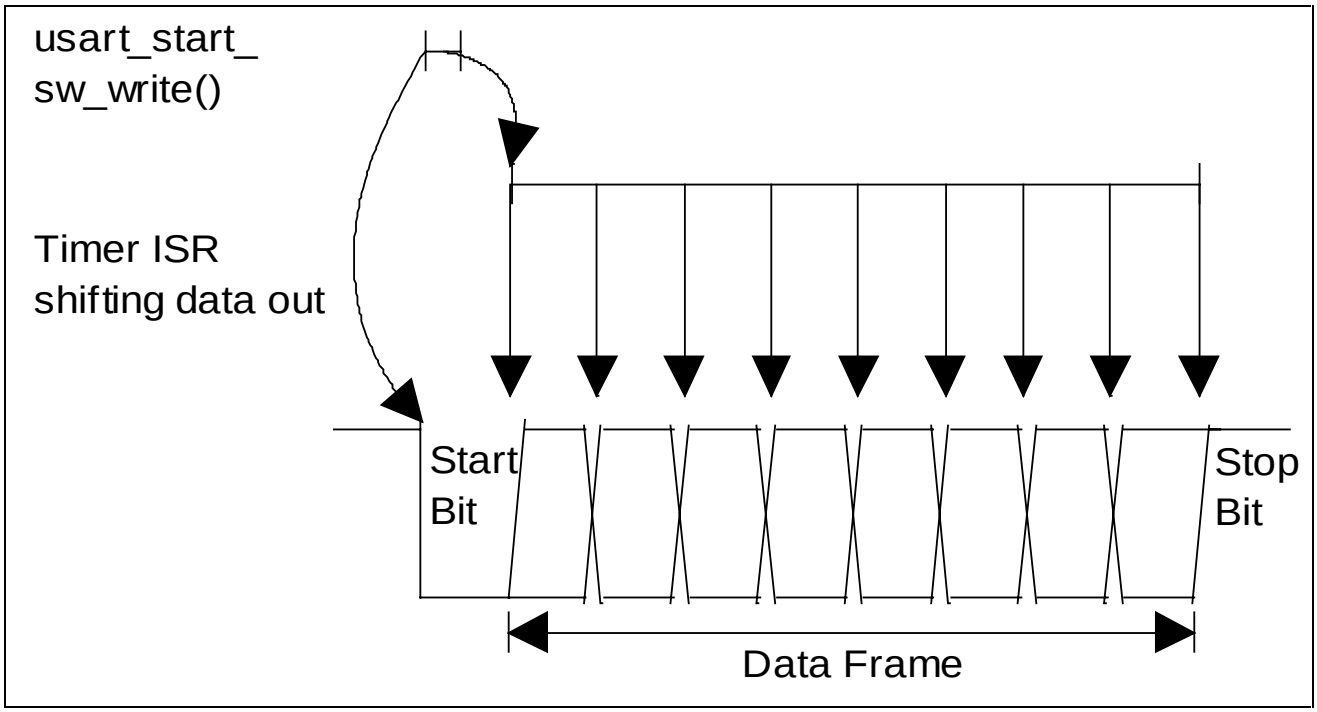

Figure 2
Schematic Diagram of Emulating an USART Write Operation using a Timer ISR

2.4.2 USART READ

A USART Start Bit arriving at the SW-USART-RXD input pin, which is externally connected to an External Interrupt pin initiates an USART READ. The correlated interrupt service routine starts a timer configured in 'Auto Reload Mode' and loaded with a start value of 1.5 bit time and a reload value of 1.0 bit time of the required baud rate.

The timer interrupt service routine samples and stores the logic state of the SW-USART-RXD pin into the LSB bit of the input byte buffer followed by a shift operation. The final USART Stop Bit provided by the external communication partner is completely ignored.

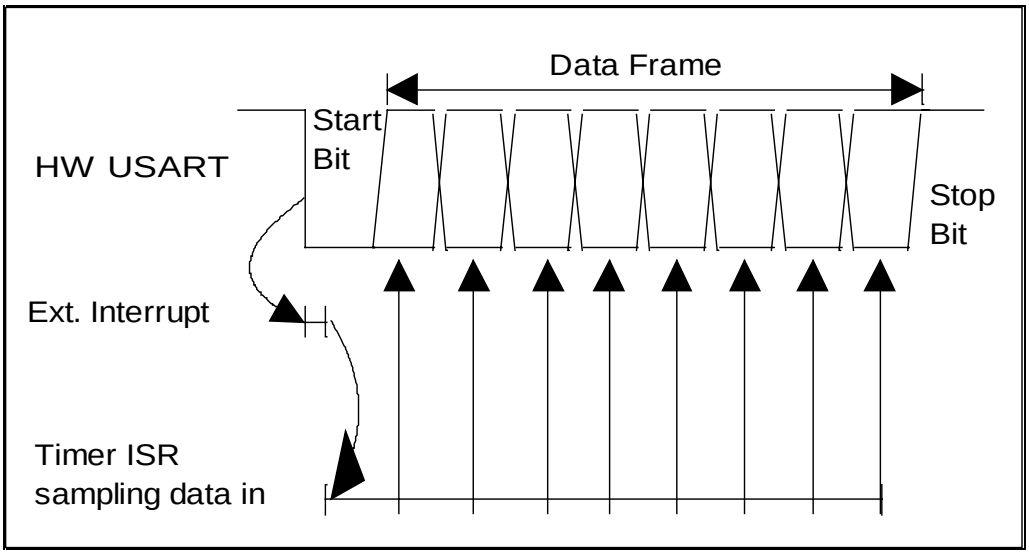

Figure 3
Schematic Diagram of Emulating an USART Read Operation using a Timer ISR

3 USART 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 USART functionality. 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 USART 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.

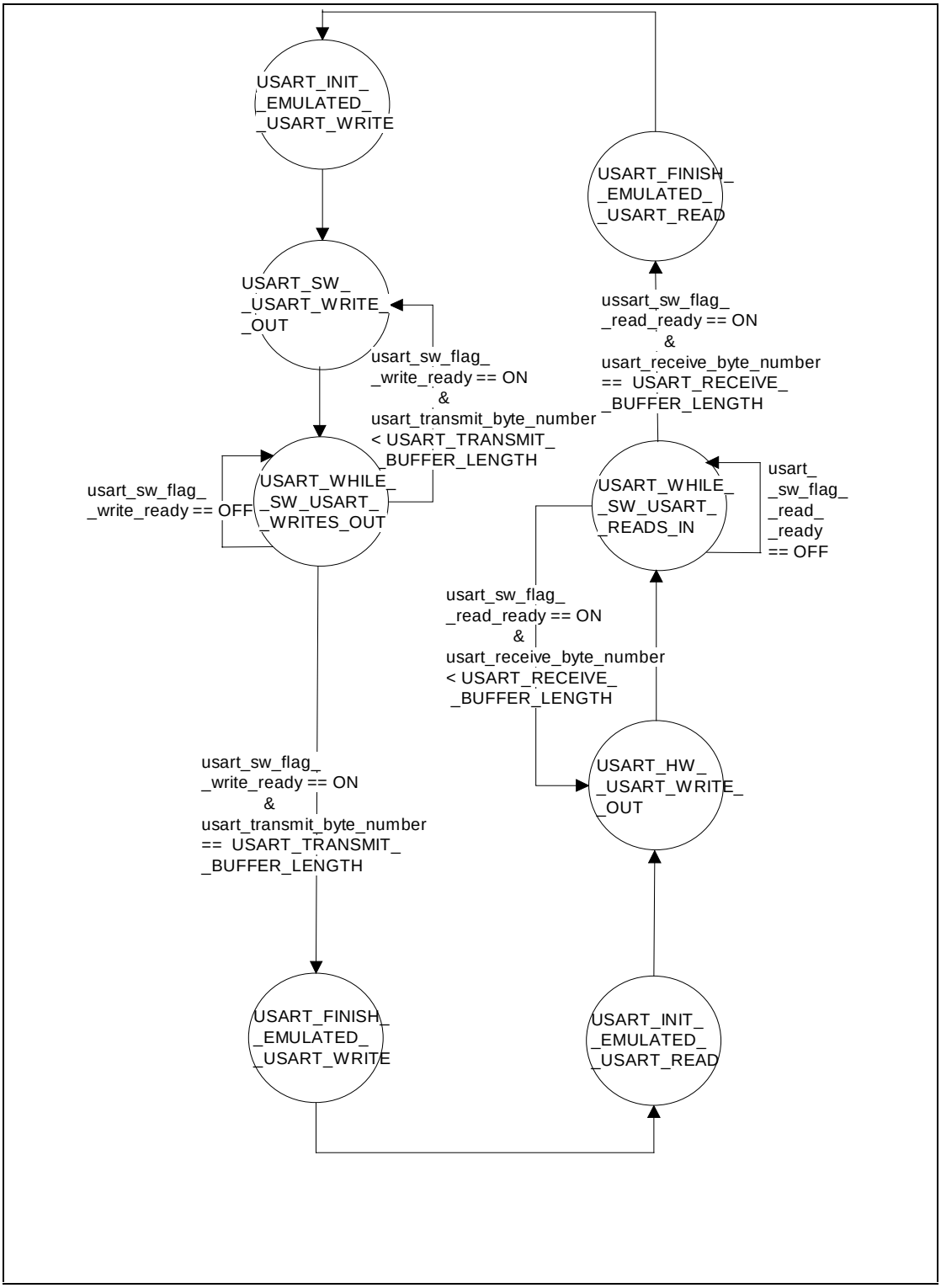

Figure 4
State Machine Diagram for test program ”usa_test.c”

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

- The first state ‘USART_INIT_EMULATED_USART_WRITE’ initializes the emulated USART interface with the baud rate to be supported (19.2 K). As communication partner serves the on-chip HW-USART which is set up in the same baud rate.
- The second state ‘USART_SW_USART_WRITE_OUT’ starts the SW-USART with the first byte of a message string to be transferred.
- In the third state ‘USART_WHILE_SW_USART_WRITES_OUT’ a flag is polled indicating the end of data transmission via the SW-USART. User application code to be executed during the SW-USART write 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 test case.
- The last state ‘USART_FINISH_EMULATED_USART_WRITE’ disables all hardware modules required for data transmission.

In the second test case the communication is started with an altered transmission direction. The SW-USART receives a message string provided by the on-chip HW-USART.

The auxiliary subroutines ‘usart_init_hw_usart’, ‘usart_disable_hw_usart’ and ‘usart_hw_interrupt_service’ help to control the HW-USART in the test environment.

3.3 Emulation Subroutines

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

- 'usart_init_sw_usart()' initializes all required auxiliary hardware modules like Timer and the External Interrupt input by programming their control registers. The Timer is configured in 'Auto-Reload Mode'; for SW-USART read operations its 'start interval register' (TL0) is loaded with 1.5 bit time while the reload register (TH0) is generally set to 1.0 bit time (see Figure 3).

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

- 'usart_start_sw_write()' prepares a data transmission by pulling down the SW-USART-TXD 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 SW-USART data bit transmission. The initial timer load value for achieving 'one bit time' is calculated by a formula presented in chapter 3.4.

- 'usart_int0_interrupt_service()' is started by a 'Low' level at the SW-USART-RXD 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 the data reception after a 1.5 bit time interval. Finally the interrupt0 service is disabled to avoid undesired reactions to incoming '0' data bits.

- 'usart_timer0_interrupt_service()' is split into different parts for write and read operations. In 'Write Mode' the SW-USART-TXD pin is provided with the LSB bit of the data byte to be transmitted; a 'Shift Right' operation is executed afterwards preparing the next data bit output. After 8 data bit outputs the USART Stop Bit is finally transmitted. In 'Read Mode' the SW-USART-RXD pin is scanned and its logic state is copied into the input byte buffer in ascending order. Finally the timer is stopped and the External Interrupt is enabled again after clearing the External Interrupt Flag indicating a 'High' - 'Low' transition in the received data bit stream.

3.4 Baud Rate Calculation

The reload values for the timer emulating the SW-USART baud rate generator are calculated as follows:

$$SW\_USART\_Baud\_Rate\_using\_Timer0 = \frac{fosc}{12*(256 - TH0)}$$

The TH0 reload register is generally loaded with 'One Bit Time' of the desired SW-USART baud rate.

The initial TL0 start value for SW-USART Read Mode ignoring the received Start Bit is calculated by subroutine 'usart_init_sw_usart()' as presented in the next formula:

$$TL0 = 256 - ((256 - TH0)*1,5) + USART\_EX0\_INT\_DELAY\_CORRECTION + USART\_TIMER\_INT\_DELAY\_CORRECTION$$

The terms 'XXXX_INT_DELAY_CORRECTION' take 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 sampling in 1st data bit in the middle of the first data bit period.

The initial TL0 start value for SW-USART Write Mode is calculated by subroutine 'usart_init_sw_usart()' as presented in the next formula:

$$TL0 = TH0 + USART\_TIMER\_INT\_DELAY\_CORRECTION$$

The term 'TIMER_INT_DELAY_CORRECTION' takes into account

- the Interrupt Response Time for Timer0 Overflow including the execution time for all statements in the corresponding interrupt service routine before shifting out 1st data bit at the beginning of the first data bit period.

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

Table 2
Calculated Interrupt Delay Correction Values

External Interrupt (fosc = 12 MHz)	Timer0 Interrupt (fosc = 12 MHz)
9 us	20 us

Attention: The Interrupt Service Delay value depends on the clock generator frequency.

In the ”test shell” (usa_test.c) a communication system is designed using one software emulated USART 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 software USART 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:

$$\mathrm{HW\_USART\_Baud\_Rate\_using\_Timer2} = \frac{\mathrm{fosc}}{2*16*(65536-\mathrm{TH2,TL2})}$$

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. For this purpose test statements are included in main program indicating the begin and the end of a byte transfer by switching a port pin to ‘Low’ and to ‘High’ state. An oscilloscope scans this port pin. The execution time of the required interrupt service routines emulating the USART is calculated by analyzing the compiler object module list.

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 next table presents load measurement results for an USART emulation via SW routines running with different baud rates.

Table 3:
Load Measurement Values for an USART emulation via SW Routines at SAB C513

Crystal Frequency	Baud Rate	Direction	Load
11.059 MHz	19.2 K	**Read**	63.0%
11.059 MHz	19.2 K	**Write**	54.0%
11.059 MHz	9.6 K	**Read**	31.5%
11.059 MHz	9.6 K	**Write**	27.0%
11.059 MHz	4.8 K	**Read**	15.7%
11.059 MHz	4.8 K	**Write**	13.5%

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 an emulated SW-USART read operation. The incoming bit stream is received with ‘LSB first’, but must be stored in ascending order.

Handling Read and Write operations with different timers (e.g. T0 for read direction and T1 for write direction) may increase the maximum baud rate or decrease the CPU overhead at a constant baud rate and would support full duplex capabilities at lower baud rates (max. 4.8 K).

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 usa_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.

Typing ‘usa_make.bat’ in a DOS window switched to the directory containing this batch file starts the Make-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-USART are neighbored to the related HW-USART 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-USART-TXD (P3.1)
Ext. Interrupt 0 (P3.2)	X

Note: sharing the same I/O pin (P3.1) and do not need any external wire internally connects SW-USART-RXD and HW-USART-TXD.

Note: sharing the same I/O pin (P3.0) and do not need any external wire internally connects SW-USART-TXD and HW-USART-RXD

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