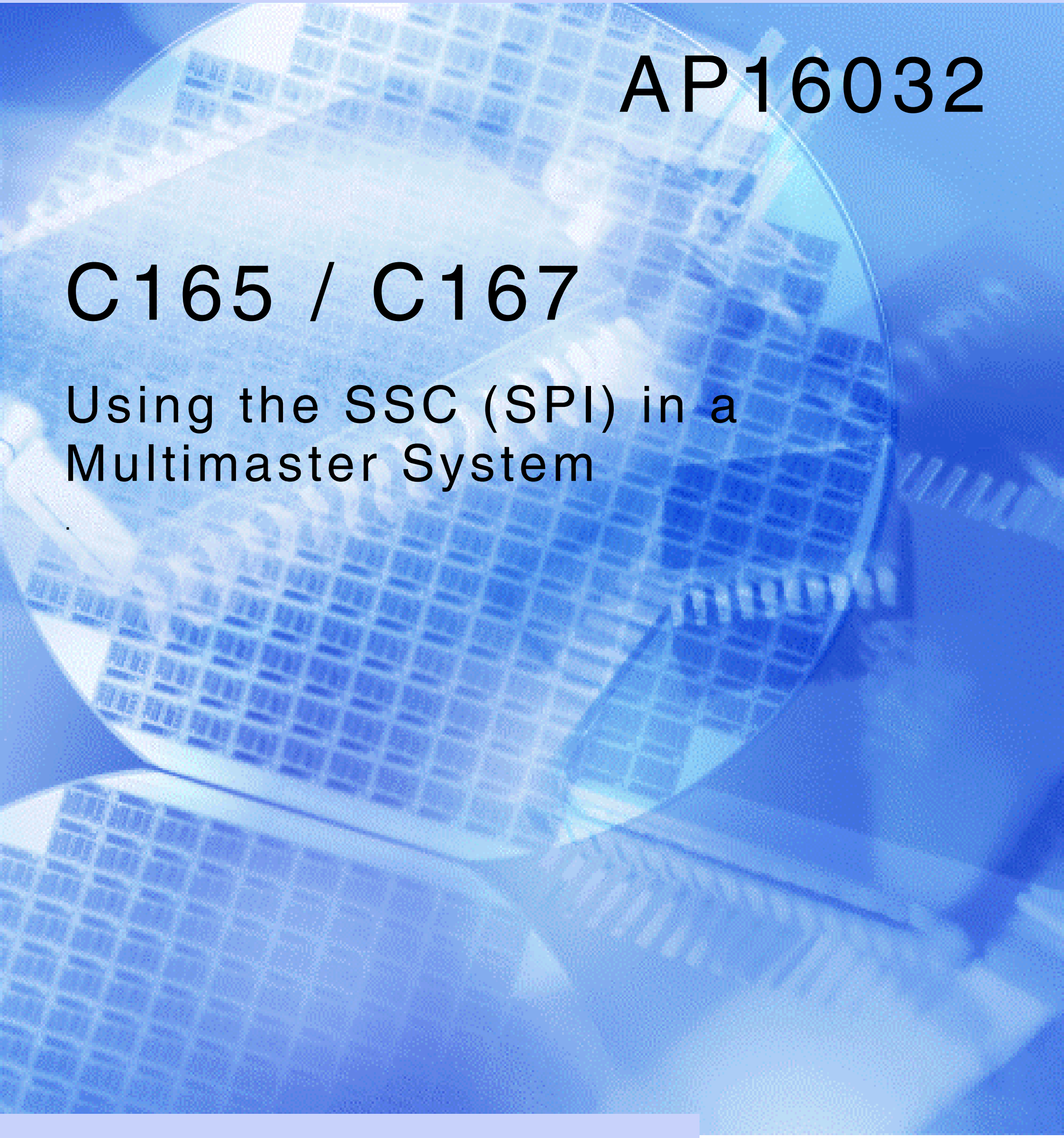
Application Note, V 1.0, Feb. 2004

AP16032



C165 / C167

Using the SSC (SPI) in a Multimaster System

.

Microcontrollers

Infineon technologies

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!	

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

Table of Contents

Table of Contents		Page
1	Introduction	2
2	General operation and hardware environment for the SSC demo	2
3	SSC demo software	2
3.1	Main program	2
3.2	System initialization	2
3.3	SSC initialization	2
3.4	Key service routine	2
3.5	Routines for SSC data transmission	2
3.5.1	Send data	2
3.5.2	Receive data	2
3.6	Running LED light	2
3.7	SSC scan routine	2
3.8	Creating the microcontroller executable files	2
3.9	Known problems	2
4	Source codes and compiling tools	2
4.1	SSC demo software: C code	2
4.1.1	Main program - SSC.C	2
4.1.2	System initialization - INIT_SYS.C	2
4.1.3	SSC initialization - INIT_SSC.C	2
4.1.4	Key interrupt service routine - KEY_INT.C	2
4.1.5	Running LED light - RUNLIGHT.C	2
4.1.6	SSC scan routine - SSC_SCAN.C	2
4.1.7	SSC receive interrupt service routine - RX_INT.C	2
4.1.8	SSC transmit routine - TX_SSC.C	2
4.1.9	SSC transmit interrupt service routine - TX_INT.C	2
4.1.10	C header file - SSC.H	2
4.2	System register initialization, ASM code - INIT_167.SRC	2
4.3	Linker and locater control file - SSC.ILO	2
4.4	Locater output file - SSC.MAP	2
4.5	Miscellaneous files for compiling	2
5	LED display board - schematic	2

1 Introduction

In the SAB C165 and C167, an internal High-Speed Synchronous Serial Interface is implemented providing serial communication between C167 / C165 or other microcontrollers with a transfer rate up to 5 MBaud at 20 MHz CPU clock. Due to the very flexible configuration options this interface can be used in a wide range of applications from simple external shift registers to expand the number of parallel ports or primitive pulse width modulation (PWM) to high-end protocol driven microcontroller networks. For a complete list of options for configuring the SSC refer to the C165 or C167 User's Manual, edition 8/94, section 11. Shown in the demo software is a multimaster full-duplex system in which at a given time one microcontroller is configured as master while all others are in slave mode.

This demo software has been created to show an example how to use the High-Speed Synchronous Interface in a non-trivial application and to support solving of user specific demands concerning the SSC. Due to pin limitations at Port2 (only P2.0-P2.7 available) of the C165 it is recommended to use this software with C167 based boards only.

2 General operation and hardware environment for the SSC demo

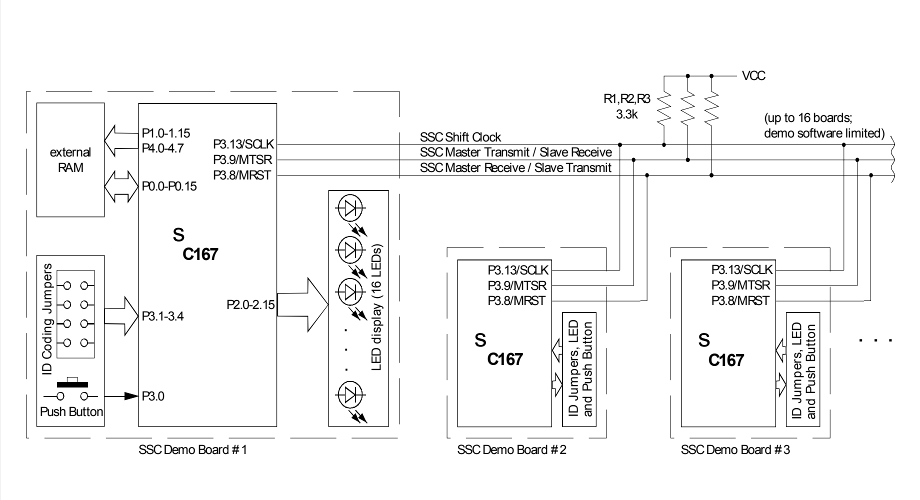


Figure 1 Hardware for the SSC demonstration software

As shown in fig. 1, up to 16 C167 based boards are connected in full-duplex operation via the SSC lines SCLK (SSC Shift Clock), MTSR (Master Transmit Slave Receive) and MRST (Master Receive Slave Transmit). Every board is identified by a combination of 4 jumpers on the LED display board providing a 4-bit board ID. These ID is used in creating and decoding messages to specify source and destination board. After starting the program, the current board ID value is displayed on the center part of the LED display.

Following the SSC and the board are configured to slave indicated by a running LED light. Pushing the button on the display board now forces the controller to create a message to be sent via the SSC requiring master status for the board when the current master is providing SSC Shift Clock. If there is no master in the system (e.g. after start-up of all boards) the controller waits for a certain time and automatically becomes master when no clock is applied. The LED display now shows in the center part the (binary) number to be added to the board ID resulting in the board ID of the remote controlled board. Remote controlled boards remaining in slave mode are accessed in ascending order by additional single clicks providing a wrap-around. Double-clicking the masters button will now result in toggling the direction of the running LEDs on the remote controlled board.

Additionally, there is a 16 word data transfer performed between specified memory areas of master and slave which can only be detected using monitor software to view the memory or recording the SSC data flow using a digitizing scope in single-shot mode. A brief graphical overview of the activities is shown in the chart in fig.2

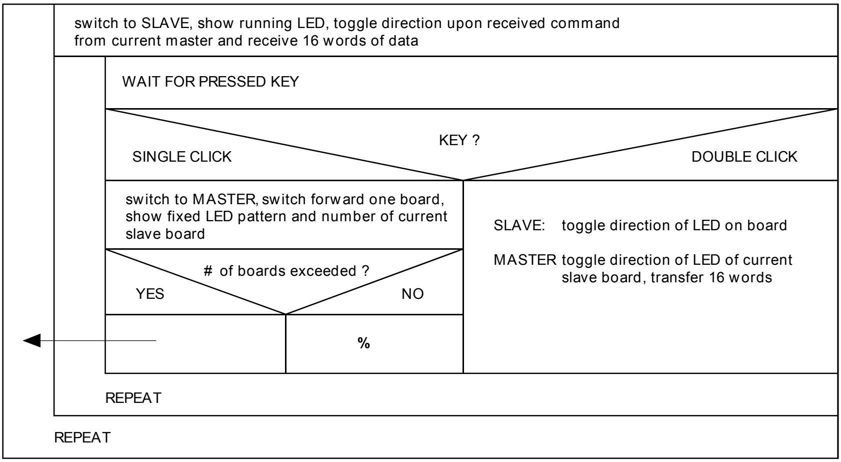


Figure 2 Flow of software from the user's view

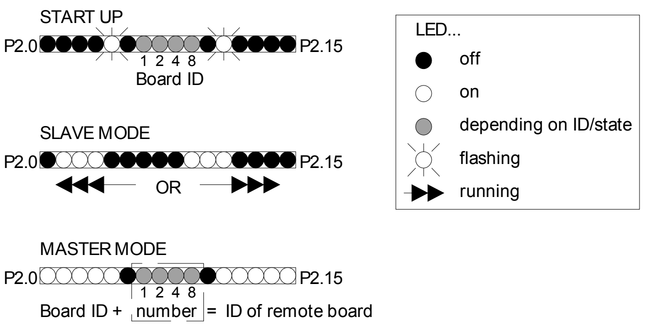


Figure 3 Functions of the LED display in different operating modes

3 SSC demo software

The software is divided into several modules each performing very specific actions described in the following sections.

3.1 Main program

Because the software is event-controlled (interrupts), the main program consist only of a system initialization routine call and a loop containing the IDLE instruction. Each time IDLE is executed the CPU is powered down while all peripherals like timers remain running. This state is held until an interrupt occurs (e.g. from SSC scan, running LED). After executing the system initialization all used state variables and on-chip devices like timers, PEC and SSC are configured. See also chapter 3.2 and 4.1.1.

3.2 System initialization

The system initialization routine INIT_SYS is executed only once during start-up and performs initialization of global variables and loading of the configuration registers of the used on-chip peripherals. These devices are configured as following:

- Port2 to output for driving the LED display
- Timer2 to 50 ms, used in master mode to provide all slave boards periodically with SSC clock
- Timer5 to 50 ms, used in slave mode for the running LED pattern (one shift per timer overflow)
- Timer0 as counter for P3.0 to detect pushed key
- PEC0 to transfer one word from SSC receive buffer to receive data buffer in memory
- PEC2 to transfer 16 words from SSC receive buffer to auto-incremented receive data buffer location in memory
- SSC with fixed baudrate as defined in the C header file SSC.H
- SSC to slave move (see also chapter 3.3)

For the code listing of INIT_SYS.C see 4.1.2.

3.3 SSC initialization

Prior to initialization it is required to disable the SSC by clearing the configuration register SSCCON. Then the pins used by the SSC (MRST, MTSR and SCLK) have to be configured whether to output or to input depending on the desired mode of the SSC. In master mode, SCLK providing shift clock for all SSC and the master transmit pin MTSR are outputs while the master receive pin has to be set to input (Port3 direction register DP3). In slave mode, all three port pins are configured to input (SCLK receives shift clock from the current master, MTSR is the slave receive line and MRST is only switched to output when the slave transmits data to the master to avoid collisions on this line). In order to use these alternate functions of Port3, the bits of the output latch are to be set to ‘1’ because it is ANDed with the alternate function.

In the demo software, the SSC is configured to a data width of 16 bit (SSCBM), transmit and receive MSB first (SSCHB), shift transmit data on the leading edge (SSCPH), idle clock line low (SSCPO) and ignoring all errors (SSCTEN,SSCREN,SSCPEN,SSCBEN), resulting in a initialization value of 0x805F in slave mode or 0xC05F in master mode (fig.3). Configuring and enabling the SSC is done by simply writing this value into SSCCON and takes only one instruction.

For the code listing of the SSC initialization file, INIT_SSC.C, see 4.1.3.

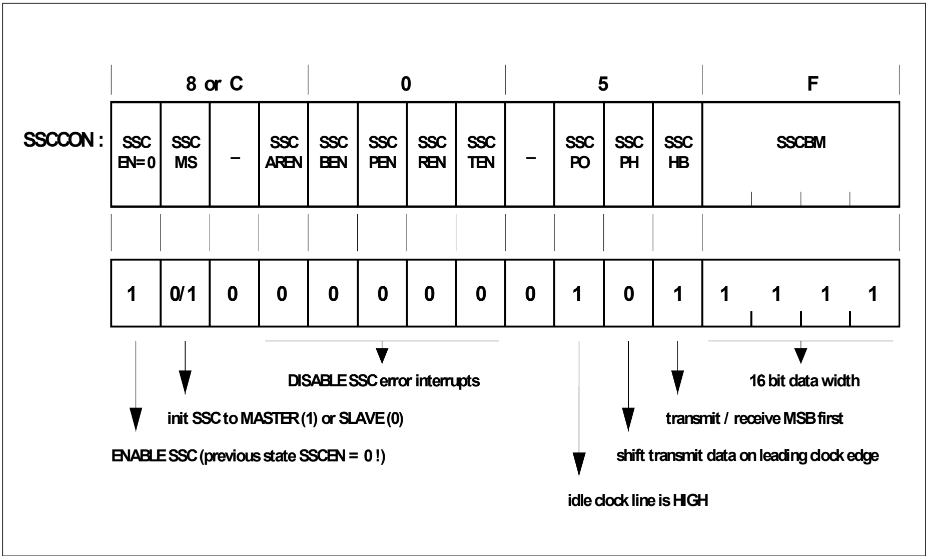


Figure 4 Configuring the SSC in the demo software

3.4 Key service routine

The key service routine is a interrupt service routine called by an Timer0 overflow interrupt caused by pressing the key or a spike on this line. Spikes and bursts induced by the key contact material of less than about 2 ms are filtered out by the routine SCAN and do not cause an erroneous action of the software. As shown in the flowchart (fig. 4) below the first assumption about the pressed key is to be a single click which would cause the transmission of master request and become master or, if already master, switching the remote slave boards as explained in chapter 1. But, if the key is pressed again within a time frame of 300 ms after releasing, this is recognized as an double click and forces the software in master mode to send a command to the remote controlled slave board to toggle the direction of the running LEDs and in slave mode to toggle the direction of the on-board LED display.

For the code listing of the key interrupt service routine, KEY_INT.C, see 4.1.4.

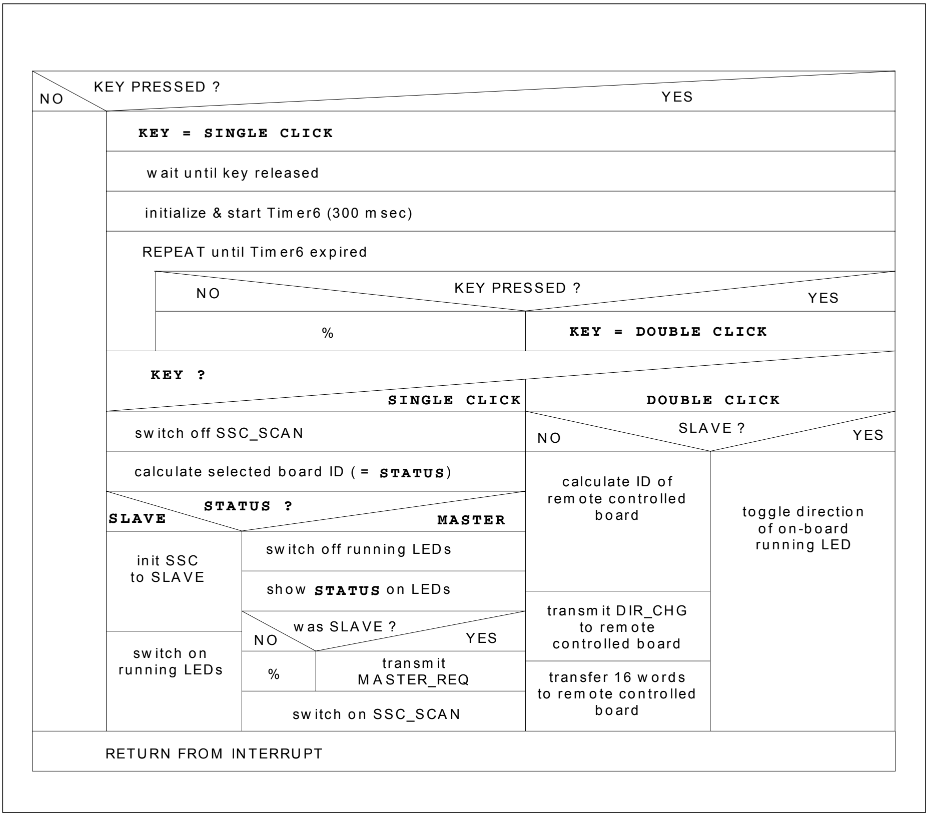


Figure 5 Key Interrupt Service Routine

Table 1 **Agenda of protocoll commands used in SSC communication**

COMMAND NAME	HEX VALUE	DESCRIPTION
IDLE	000 (0h)	dummy command for SSC_SCAN, sent by master
DIR_CHG	101 (5h)	command 'toggle direction of running LEDs' sent by master to current remote controlled slave
MASTER_REQ	111 (7h)	request become master, sent by slave
OK	010 (2h)	acknowledge for MASTER_REQ, sent by master

3.5 Routines for SSC data transmission

Since the serial SSC data is collected in a shift register transmitting and receiving is synchronized and performed at the same time. The pins MRST and MTSR are assigned to the input and the output of the shift register according to the operating mode (master or slave) so there is no need for external hardware to switch the pins in master or slave mode.

3.5.1 Send data

Transmitting data is performed by simply writing the data value into the SSC Transmit Buffer SSCTB. If the shift register is empty, that means, the last transmission is already finished, the contents of the SSCTB is copied immediately to the shift register. In master mode, transmission starts instantly by supplying SSC shift clock on SCLK and shifting out data on MTSR while in slave mode the data remains unchanged in the shift register until the remote master applies SSC shift clock. Data is then shifted out on MRST.

In all modes, after copying the data from SSCTB to the shift register, an SSC transmit interrupt indicates a request for new data to be transmitted. This is especially used for transferring 16 words after the command DIR_CHG when the SSC transmit interrupt causes PEC1 to transfer data from a memory array to SSCTB without any interrupt service routine.

For the C code of the routines TX_SSC.C and TX_INT.C see 4.1.8 and 4.1.9.

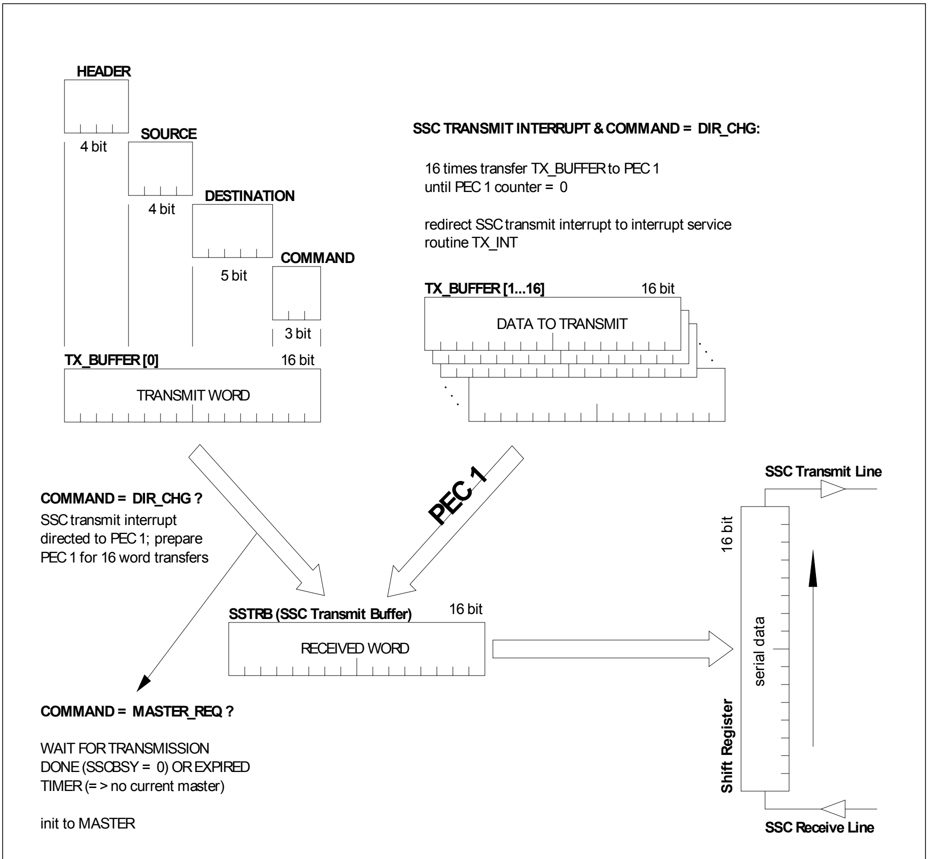


Figure 6 Transmitting Data via SSC

3.5.2 Receive data

As mentioned, receiving of data via the SSC is always synchronized with transmitting data. If the selected number of bits is received data is automatically transferred from the shift register into the SSC receive buffer SSCRB. The software is then notified by an SSC receive interrupt to copy the value of SSCRB into any software buffer before the next SSC data word is received. In the demo software, initially the SSC receive interrupt is directed to PEC0 which transfers the received word into RX_BUFFER[0] and then calls the interrupt service routine RX_INT because PEC0 counter has been decremented to 0. If the command DIR_CHG is decoded, the SSC receive interrupt is then directed to PEC2 which will transfer 16 consecutive words from the SSCRB to the RX_BUFFER. After that, RX_INT is called again (PEC2 counter = 0) and the SSC receive interrupt is redirected to PEC0 to prepare receiving of the next word.

Depending on the received command, several actions will take place as shown in the diagram in Fig. 6.

For the code listing of RX_INT refer to chapter 4.1.7.

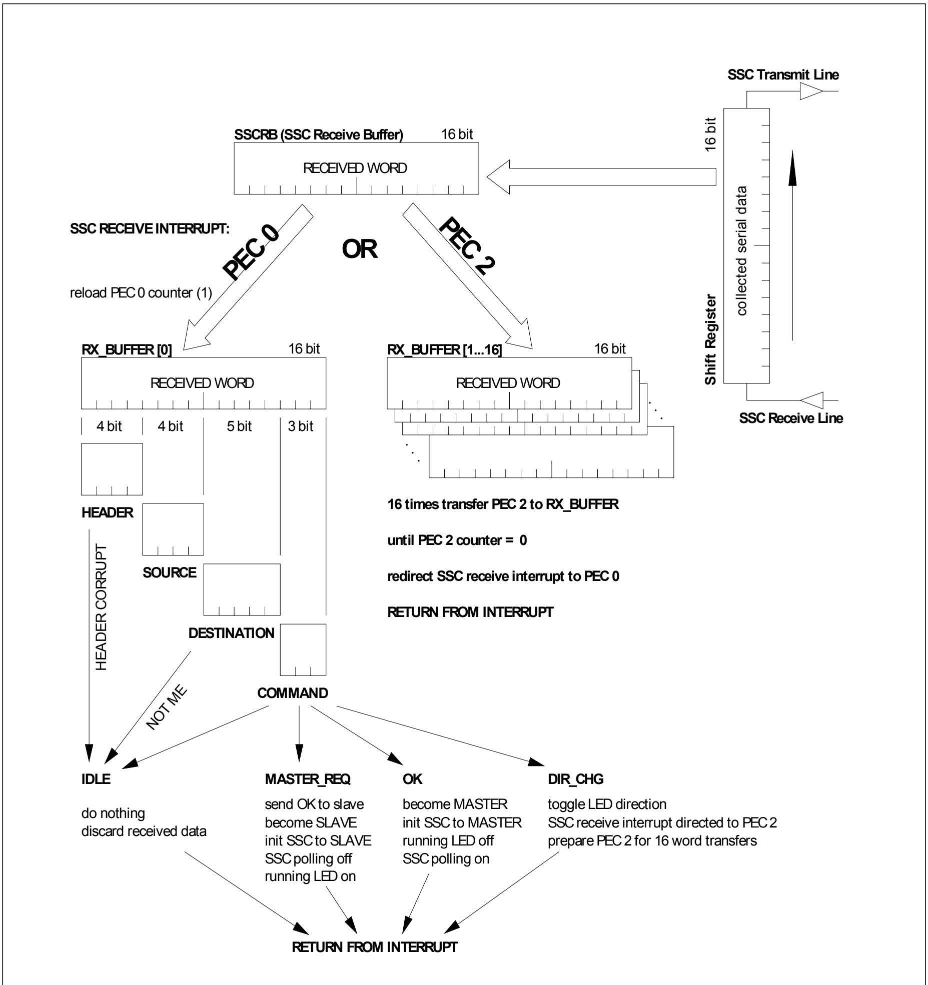


Figure 7 Receiving Data via SSC

3.6 Running LED light

The running LED light is controlled by the interrupt service routine RUNLIGHT which is called each time after an Timer5 overflow has occurred, that means, after the delay time for a LED left or right step is passed. The current LED status is obtained from a global variable, rotated to the right or to the left depending on the direction variable and written back to the global variable and to Port2 where the LEDs are connected to. Because Timer5 is not capable of self reload, the Timer5 control and timer registers have to be reloaded to feature the appropriate delay time and restart of the timer. For a graphical description see fig. 7, the C code of the routine RUNLIGHT.C can be found in chapter 4.1.5.

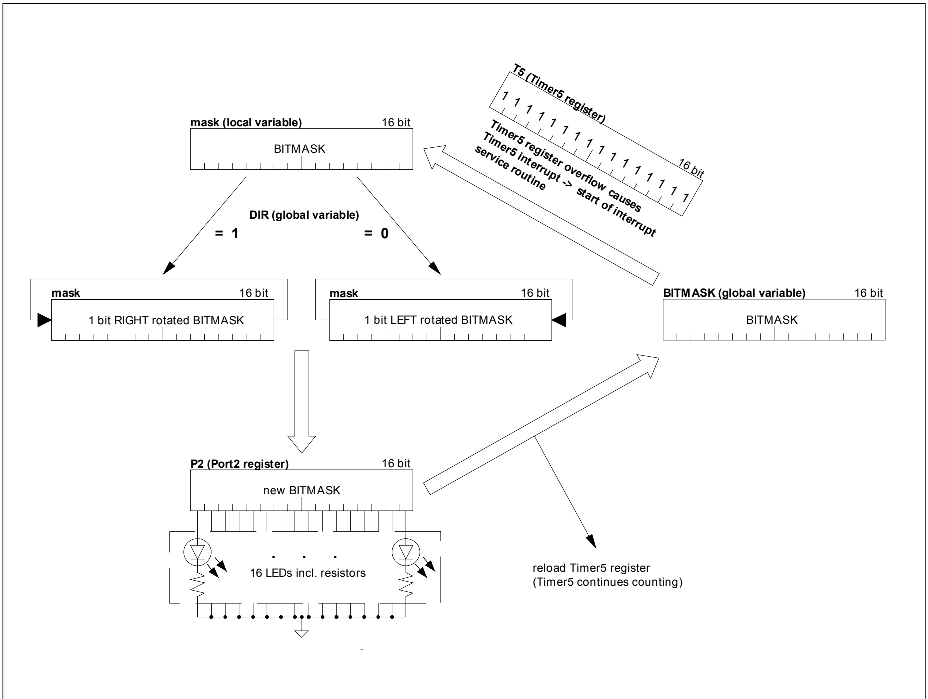


Figure 8 Timer 5 interrupt service routine for running LED light

3.7 SSC scan routine

In master mode, the SSC scan routine is called by an expired Timer2 causing an interrupt and performs periodical transmitting of dummy words. This must be done because the transmitting master supplies all slaves with SSC shift clock in order to make transmitting of MASTER_REQ for the slave possible. After the master SSC starts transmitting, the Timer2 is reloaded and started to obtain constant scanning.

For the code listing of this routine, SSC_SCAN.C, see 4.1.6.

3.8 Creating the microcontroller executable files

For running the SSC demo, the C code files and the assembly startup file have to be compiled, linked, located and converted into a format the monitor software is capable to upload into the microcontroller. When using the BSO/Tasking C compiler and the Hitop debugger light (Hitex), a make file (SSC.MAK; starting the make process by executing MAKE.BAT) has been prepared to control compiling. The make file is executed from the bottom of the file upwards depending on the file dates (newer source files will be converted, unaltered source files with existing converted files remain untouched). For control of linking of the modules and creating the interrupt vector table the file SSC.ILO is used containing information of module related interrupt numbers, classes memory locations and reserved areas not to be used by the user. A graphical overview of the use of the compiling tools by the MAKE utility for obtaining microcontroller executable files is shown in fig. 10. Note that the paths and DOS environment variables for the compiling tools have to be set by calling SETPATH.BAT (see listing in 4.5).

Before loading the program code from SSC.HTX into the microcontroller memory, the controller has to be booted by executing BOOT.BAT, which causes BTDL.EXE to load the files BOOT.BSL (boot software, sets the system registers to required values depending on the hardware, e.g. memory waitstates, system stack size and prepares the controller to load the monitor) and EVA167.HEX (monitor software) into the microcontroller's memory to make communication between the PC based debugging software and the microcontroller possible. The monitor software has been slightly changed (not possible using the 'light' version) to ensure matching hardware values in boot file and monitor. Then the debugging software has to be started by executing HIT1.BAT (edit this file for different COM port or transfer speed configurations).

After loading (SSC.HTX and the symbol file SSC.SYM) and starting the SSC demo software by the Hitop debugger, it is recommended to leave the debugger because the extensive use of timers in the SSC software can interfere with the communication between PC and the debugger which results in the error message "Could not send command".

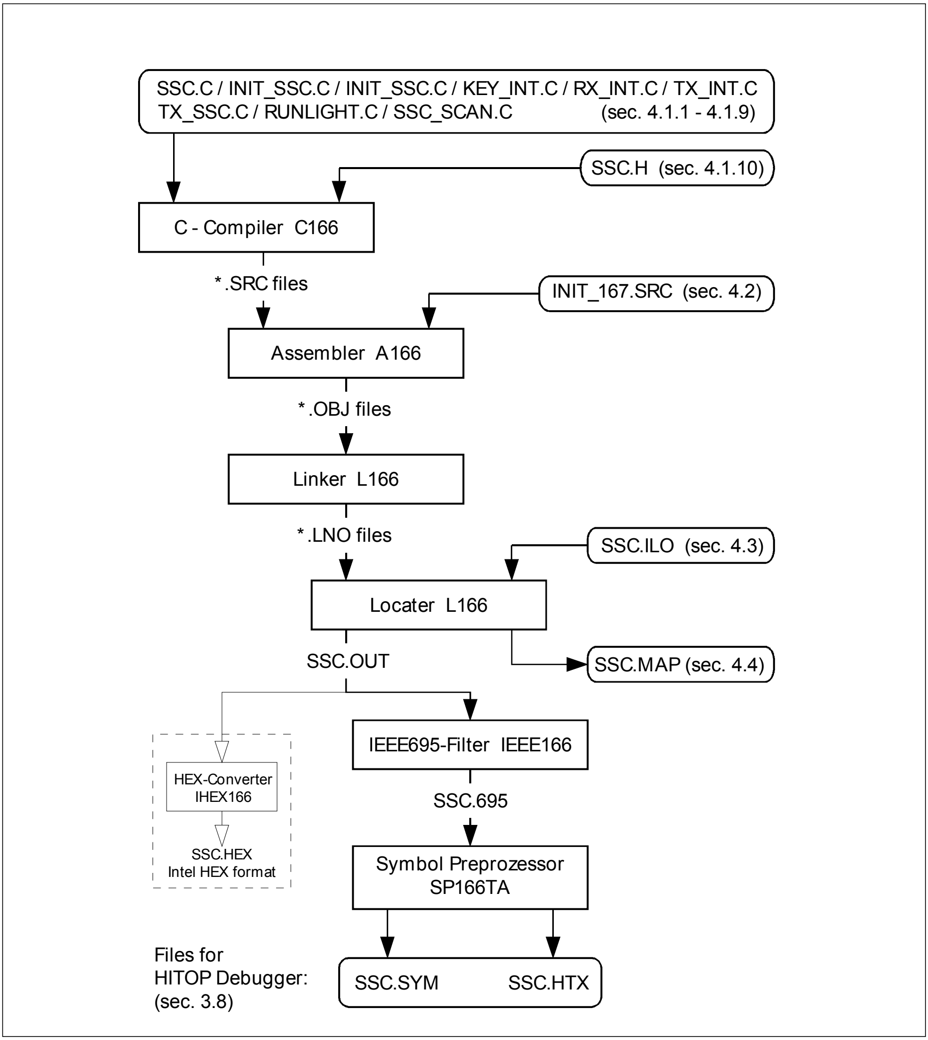


Figure 9 Making C16x controller executable files

3.9 Known problems

The SSC normally operates at speeds up to 5 MBaud. Long signal lines and improper PCB design can cause the transmission to fail at higher baud rates because the output driver of the SSC lines meet only the standard requirements for TTL compatibility. During software development for the SSC demo the available C167 boards operated correctly only up to approximately 115 kBaud when connected together to a 3 or more board system. Only a system consisting of two C167 boards worked correctly up to the maximum transfer rate.

4 Source codes and compiling tools

4.1 SSC demo software: C code

4.1.1 Main program - SSC.C

```
/*********************************************************************
 *  program   : ssc.c                                               *
 *  name      : Andreas Hettmann    Siemens, Cupertino/CA           *
 *  date      : 5'96                                                *
 *  function : SSC demo                                             *
 *********************************************************************/

/* #pragma mod167 */

#include <reg167.h>                        /* register definitions  */
#include "ssc.h"                           /* definitions           */
extern void INIT_SYS (void);               /* use external routines */

#pragma global          /* make next definitions useable for external C
                        modules */
unsigned int STATUS;    /* word STATUS = actual status of board (slave,
                         ... */
unsigned int BOARD_ID;/* word BOARD_ID = board id as jumpered on
                         P3.1-4 */
unsigned int BITMASK;   /* word BITMASK = running LED actual output
                        state */
bit DIR;                /* bit DIR = current direction of running LEDs */
bit KEY_IDLE;           /* bit KEY_IDLE = input level, key not pressed */
unsigned int RX_BUFFER [1 + TRANSFER_CNT], TX_BUFFER [1 + TRANSFER_CNT];
                        /* word array storing rec and transmit data */

interrupt (mainintno) void Int_MAIN (void); /* main task interrupt# */
                                            /* prototype definition */
#pragma public

interrupt (mainintno)   void Int_MAIN()      /* main task              */
{
     INIT_SYS ();       /* initialize all devices needed for the demo
                        program */
```

```
    while ( 1 )         /* infinite loop, program is event controlled
                        (interrupts) */
    {
        #pragma asm
        IDLE            /* enter CPU idle mode, return to normal
                        operation by interrupt */
        #pragma endasm
    }
}
```

4.1.2 System initialization - INIT_SYS.C

```
/**********************************************************************
 *  program  : init_sys.c                                             *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA              *
 *  date     : 5'96                                                   *
 *  function : initializes system                                     *
 **********************************************************************/

#include <reg167.h>                         /* register definitions */
#include "ssc.h"                            /* int # definitions */

extern bit DIR;                    /* global variables of bit type,
                                   defined in SSC.C */
extern bit KEY_IDLE;
extern unsigned int STATUS;        /* global variables of word type,
                                    def. in SSC.C */
extern unsigned int BOARD_ID;
extern unsigned int BITMASK;
extern unsigned int RX_BUFFER [1 + TRANSFER_CNT]; /* (word array) */
extern unsigned int TX_BUFFER [1 + TRANSFER_CNT];
extern void INIT_SSC (unsigned int);              /* use external
                                                  routine INIT_SSC */

void INIT_SYS (void);              /* prototype definition of routine */

#pragma global                     /* make key_stat useable for external
                                   C modules */

void INIT_SYS (void)
{
    DIR     = 0;                      /* shift LED light left (init) */
    BITMASK = LED_START_MASK;         /* init LED pattern */
    KEY_IDLE = _getbit ( P3, 0 );     /* get key input level (not
                                      pressed) */
```

```
_bfld ( DP3, 0x001E, 0x0000 );   /* switch P3.1 - P3.4 to input
                                 mode */
DP2        = ONES;               /* switch P2.0 - 2.15 to output
                                 mode (LED's) */
P2         = 0x001E;             /* output H on P2.1 - P2.4 */
BOARD_ID = ( P3 >> 1 ) & 0x000F; /* ...,get the jumper location,
                                 shift rightbound and store as ID */
P2         = BOARD_ID << 6;      /* show ID on LED panel */

T6IR   = 0;                      /* clear Timer6 INT request flag */
T6IE   = 0;                      /* and disable Timer 6 interrupt */
T6     = 0x0000;                 /* load timer register */
T6CON  = 0x0047;                 /* Timer6 counts up, fc=fcpu/512,
                                 timer starts */

while ( ~T6IR )                  /* wait for Timer6 overflow */
{
    if ( T6 & 0x1000 )           /* get flashing LED's @ P2.5 and
                                 P2.10 */
       P2 |= 0x0420;             /* while waiting for Timer6
                                 overflow /

    else
       P2 &= 0x03C0;
}
T6CON  = 0x0000;                 /* stop Timer6 */
T6IR   = 0;                      /* clear Timer6 interupt request
                                 flag */
_bfld ( T2IC, 0xFF, T2_INT );    /* set Timer2 INT priority & group
                                 level */
T2     = MASTER_IDLE / 0.0128;   /* init Timer2 for SSC_SCAN */
T2CON  = 0x00C5;                 /* count down, fc=fcpu/256, start
                                 timer rem.: Timer2 INT disabled,
                                 cf. SSC.H  */
_bfld ( T5IC, 0xFF, T5_INT );    /* set Timer5 INT priority & group
                                 level */
T5     = LED_STEP;               /* init Timer5 for RUNLIGHT */
T5CON  = 0x00C7;                 /* count down, fc=fcpu/512, start
                                 timer rem.: Timer5 INT enabled, cf.
                                 SSC.H   */
_bfld ( T0IC, 0xFF, T0_INT );    /* set Timer0 INT priority & group
                                 level */
T0     = 0xFFFF;                 /* preload timer register */
T0REL  = 0xFFFF;                 /* load reload register */
T01CON = 0x0049 + (char) KEY_IDLE; /* Timer0 as counter, input is
                                 P3.0 (HW) count up at L/H edge if
                                 key idle level is L or at H/L edge
                                 if ~ is H */
```

```
    SRCP0  = (int) & SSCRB;            /* PEC0 source is SSC receive
                                       buffer reg */
    DSTP0  = (int) & RX_BUFFER [0];    /* PEC0 dest is 1st word of rec
                                       array */
    PECC0  = 0x0000 + 1;               /* init PEC0, incr dest, 1 word
                                       transfer */
    SSCRIC = 0x78;                     /* set SSC receive interrupt to
                                       PEC0 */

    SRCP2  = (int) & SSCRB;            /* PEC2 source is SSC receive
                                       buffer reg */
    DSTP2  = (int) & RX_BUFFER [1];    /* PEC2 dest is 1st word of receive
                                       array */
    PECC2  = 0x200 + TRANSFER_CNT;     /* incr dest, transfer defined # of
                                       words */

    _bfld ( SSCTIC, 0xFF, SSC_T_INT ); /* set SSC transmit interrupt
                                       priority & group level */

    STATUS = SLAVE;                    /* initial status of board is
                                       slave */

    SSCCON = 0;                        /* reset SSC */
    SSCBR  = ( F_CPU * 1000000 / ( 2 * BAUD_RATE )) - 1; /* set baud
                                       rate reg */
    INIT_SSC ( SLAVE );                /* calls SSC initialization
                                       routine */
}
```

4.1.3 SSC initialization - INIT_SSC.C

```
/**********************************************************************
 *  program  : init_ssc.c                                             *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA              *
 *  date     : 5'96                                                   *
 *  function : initializes SSC                                        *
 **********************************************************************/

#include <reg167.h>               /* register definitions   */
#include "ssc.h"                  /* definitions            */

void INIT_SSC (unsigned int);     /* prototype definition for
                                  routine */

#pragma global                    /* make init_ssc useable for
                                  external C modules */
```

```
void INIT_SSC ( mode )
unsigned int mode;                  /* local variable, word type */
{
     SSCCON = ZEROS;                /* stop and reset SSC */

     _bfld ( P3, 0x2300, 0x2300 );  /* set P3.8 (MRST), P3.9 (MTSR) and
                                    P3.13 (SCLK) */
     /* _bfld ( ODP3, 0xFFFF, 0x2300 ); *//* open drain outp only for
                                    development */
     switch ( mode )                /* branch depending on value of
                                    'mode' */
     {
          case SLAVE:  _bfld ( DP3, 0x2300, 0x0000 ); /* switch
                                    MRST,MTSR and SCLK to input mode */

                       SSCCON = SSC_EN | SSC_SLAVE | SSCCON_INIT;
                                    /* init SSC as slave (cf. SSC.H) */
                       break;       /* exit branch */

          case MASTER: _bfld ( DP3, 0x2300, 0x2200 ); /* switch MRST to
                                    input; MTSR and SCLK to output
                                    mode */

                       SSCCON = SSC_EN | SSC_MASTER | SSCCON_INIT;
                                    /* init SSC as master (cf. SSC.H) */
                       break;       /* exit branch */
     }
}
```

4.1.4 Key interrupt service routine - KEY_INT.C

```
/**********************************************************************
 *  program  : key_int.c                                              *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA              *
 *  date     : 5'96                                                   *
 *  function : interupt service routine for push button               *
 **********************************************************************/

#include <reg167.h>                 /* register definitions */
#include "ssc.h"                    /* int # definitions */
extern unsigned int STATUS;         /* global variables of word type,
                                    def in SSC.C */
extern unsigned int BOARD_ID;
extern bit DIR;                     /* global variables of bit type,
                                    def in SSC.C  */
```

```
extern bit KEY_IDLE;
extern void INIT_SSC (unsigned int); /* use external routines */
extern void TX_SSC (unsigned int, unsigned int, unsigned int);

interrupt (keyintno) void KEY_INT (void); /* prototype def of
                                     routines */
bit SCAN (void);

#pragma global                       /* make key_int useable for
                                     external C modules */

interrupt (keyintno) void KEY_INT ()
{
    unsigned int key = 0, stat, board; /* local variables, word type
                                     key = type of button click
                                     stat = status of board
                                     board = # of remote board */
    T0IE  = 0; /* overhead */        /* inhibit INT of Timer0 */
    T0IR  = 0; /* overhead */        /* and clear request flag */

    if ( SCAN () )                   /* branch if key pressed (filter
                                     out spikes) */
    {
        key = SINGLE_CLICK;          /* first assumption: single
                                     click */
        while ( SCAN () )            /* wait for key release */
        {}

        T6IR  = 0;                   /* clear request flag of Timer6 */
        T6IE  = 0;                   /* and inhibit INT of Timer6 */
        T6    = T_1 / 0.0256;        /* [T6] = time to wait for second
                                     click */
        T6CON = 0x00C7;              /* count down, fc=fcpu/512, timer
                                     starts */

        while ( ~T6IR )              /* wait for a second click */
        {
           if ( SCAN () )            /* if second click detected */
             key = DOUBLE_CLICK;     /* it was a double click ! */
        }
        T6CON = 0x0000;              /* stop Timer6 */
    }
    T0IR = 0; /* overhead */         /* clear request flag of Timer0 */
    T0IE = 1; /* overhead */         /* enable Timer0 interrupt */

    stat = STATUS;                   /* get STATUS into local var for
                                     faster access */
```

```
switch ( key )                    /* branch as the type of key
                                  stroke */
{
case SINGLE_CLICK:
     T2IE = 0;                    /* SSC_SCAN off */
     stat ++;                     /* increase status */
     if ( stat > NO_OF_BOARDS - 1 ) /* wrap around */
        stat = 0;
     STATUS = stat;               /* put actual status to global
                                  var */

     if ( stat == SLAVE )         /* now slave ? */
     {
        INIT_SSC ( SLAVE );       /* init SSC to slave */
        T5IE = 1;                 /* and enable running LEDs */
     }
     else
     {
        T5IE = 0;                 /* stop running LEDs */
        P2 = BACKGND_PATTERN | ( stat << STATUS_SHIFT );
                                  /* and show actual status on
                                  LEDs */
        if ( stat == MASTER )     /* was slave ? */
        {
          _putbit ( 1, DP3, 8 ); /* switch MRST to output */
          TX_SSC ( MASTER_REQ, BOARD_ID, ALL ); /* send master
                                  req */
        }
        T2IE = 1;                 /* switch SSC_SCAN on */
     }
     break;

case DOUBLE_CLICK:
     if ( stat == SLAVE )         /* status eq slave ? */
        DIR = ~DIR;               /* toggle direction of running
                                  LEDs */
     else
     {
        board = stat + BOARD_ID; /* calc # of remote board */
        if ( board > NO_OF_BOARDS - 1 )
          board -= NO_OF_BOARDS;
```

```
            TX_SSC ( DIR_CHG, BOARD_ID, board ); /* and send command */
                                     to remote board to toggle running
                                     LED direction */
         }
         break;
     }
}

bit SCAN (void)
{
     int cnt = 0, status = 0;          /* local variables, word type */
                                       cnt = # of key scans
                                       status = key input value */
     T4IR  = 0;                        /* clear request flag of Timer4 */
     T4IE  = 0;                        /* and inhibit INT of Timer4 */
     T4    = T_2 / 0.0256;             /* [T4] = key scanning time */
     T4CON = 0x00C6;                   /* count down, fc=fcpu/512, start
                                       timer */

     while ( ~T4IR )                   /* while Timer4 not expired */
     {
          status += _getbit ( P3, 0 );/* accumulate key values */
          cnt ++;                      /* and count loop cycles */
     }
     T4CON = 0x0000;                   /* stop Timer4 */

     if ( (cnt - status) < status )    /* key pressed for more than 50% of
                                       time ? */
          return ( P30_ACTIVE ^ KEY_IDLE );
     else
          return ( P30_PASSIVE ^ KEY_IDLE );
}
```

4.1.5 Running LED light - RUNLIGHT.C

```
/**********************************************************************
 *  program  : runlight.c                                             *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA              *
 *  date     : 5'96                                                   *
 *  function : LED runlight interrupt service routine                 *
 **********************************************************************/

#include <reg167.h>                    /* register definitions */
#include "ssc.h"                       /* int # definitions */
```

```
extern bit DIR;                    /* global variables of bit and word
                                   type, def in SSC.C */
extern unsigned int BITMASK;

interrupt (t5intno) void RUNLIGHT (void); /* prototype of routine */

#pragma global                     /* make runlight useable for
                                   external C modules */

interrupt (t5intno) void RUNLIGHT (void)
{
    unsigned int mask;             /* local variable, word type
                                   mask = current LED output */
    mask = BITMASK;                /* get bitmask into local var for
                                   faster access */

    if ( DIR )                     /* shift LEDs depending on current
                                   direction flag DIR */
        mask = _ror ( mask, 1 );
    else
        mask = _rol ( mask, 1 );
    T5 = LED_STEP / 0.0256;        /* reload Timer5 for running
                                   LEDs */
    P2 = mask;                     /* output computed bitmask to
                                   LEDs */
    BITMASK = mask;                /* and store new value back in
                                   global variable */
}
```

4.1.6 SSC scan routine - SSC_SCAN.C

```
/*********************************************************************
 *  program  : ssc_scan.c                                            *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA             *
 *  date     : 5'96                                                  *
 *  function : Master SSC scanning for Slave commands                *
 *********************************************************************/

#include <reg167.h>                /* register definitions */
#include "ssc.h"                   /* int # definitions */

extern unsigned int BOARD_ID;
extern void TX_SSC (unsigned int, unsigned int, unsigned int);
                                   /* use external routine */
```

```
interrupt (t2intno) void SSC_SCAN (void); /* prototype definition */

#pragma global                       /* make ssc_scan useable for
                                     external C modules */

interrupt (t2intno) void SSC_SCAN (void)
{
    TX_SSC ( IDLE, BOARD_ID, ALL );  /* send command IDLE to all remote
                                     boards rem.: this procedure is
                                     needed to supply all slaves with
                                     SSC clock periodically to ensure a
                                     slave request can be received by
                                     the master */
    T2     = MASTER_IDLE / 0.0128;   /* reload timer */
    T2CON = 0x00C5;                  /* count down, fc=fcpu/256, start
                                     timer */
}
```

4.1.7 SSC receive interrupt service routine - RX_INT.C

```
/**********************************************************************
 *  program  : rx_int.c                                               *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA              *
 *  date     : 5'96                                                   *
 *  function : interupt service routine SSC receive                   *
 **********************************************************************/

#include <reg167.h>                /* register definitions */
#include "ssc.h"                   /* definitions */

extern bit DIR;                    /* global variables of bit and word
                                   type, def in SSC.C */
extern unsigned int STATUS;
extern unsigned int BOARD_ID;
extern unsigned int RX_BUFFER [1 + TRANSFER_CNT];
extern void TX_SSC (unsigned int, unsigned int, unsigned int);
extern void INIT_SSC (unsigned int); /* use external routines */

interrupt (rxintno) void RX_INT (void); /* prototype of routine */

#pragma global                     /* make rx_int useable for external
                                   C modules */
```

```
interrupt (rxintno) void RX_INT ()
{
    unsigned int rec_word, rx_src, rx_dest, rx_msg; /* loca vars,word
                                   type */

    PECC0 = 0x0000 + 1;            /* prepare PEC0 for receiving of
                                   next word via SSC */

    if (( SSCRIC & 0x3F ) == 0x3A ) /* last transfer SSCRB -> mem via
                                   PEC2 ? */
    {
        _bfld ( SSCRIC, 0x3F, 0x38 ); /* then switch back to PEC0 */
        DSTP2  = (int) & RX_BUFFER [1]; /* PEC2 destination is 1st word
                                   of receive array */
        PECC2  = 0x200 + TRANSFER_CNT; /* increment dest, transfer
                                   defined # of words */
    }
    else
    {                              /* (last transfer via PEC0) */
        rec_word  = RX_BUFFER [0]; /* get rec value into local var */
        rx_src    = (rec_word & 0x0F00) >> 8;
        rx_dest   = (rec_word & 0x00F8) >> 3; /* get received
                                   address */
        rx_msg    = rec_word & 0x0007; /* get received command */

        if ( ((rec_word & 0xF000) != HEADER) ||  ((rx_dest != ALL) &&
         (rx_dest != BOARD_ID)) )  /* header incorrect or received
                                   data
           return;                 /* doesn't belong to this board ->
                                   return */

        switch ( rx_msg )          /* branch as the command */
        {
        case DIR_CHG:
           SSCRIC = 0x7A;          /* SSC rec INT now served by PEC2 */

           DIR    = ~DIR;          /* toggle running LED direction */

           break;

        case MASTER_REQ:
           if ( STATUS != SLAVE )  /* is current master for */
           {                       /* any remote board ?    */
             T2IE = 0;
             TX_SSC ( OK, BOARD_ID, rx_src );
             STATUS = SLAVE;       /* status becomes 'slave' */
```

```
              INIT_SSC ( SLAVE );   /*  init SSC as slave */
              T5IE = 1;             /* switch on running LEDs */
            }
            break;

         case OK:
            if ( rx_dest == BOARD_ID )
            {
              INIT_SSC ( MASTER );
              T2IE = 1;
            }
            break;

         case IDLE:
            break;                  /* rec word ist from scanning
                                    master */
         }
     }
}
```

4.1.8 SSC transmit routine - TX_SSC.C

```
/***********************************************************************
 *  program  : tx_ssc.c                                                *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA               *
 *  date     : 5'96                                                    *
 *  function : SSC TX routines                                         *
 ***********************************************************************/

#include <reg167.h>                 /* register definitions */
#include "ssc.h"                    /* int # definitions */

extern unsigned int TX_BUFFER [1 + TRANSFER_CNT]; /* global word
                                    array */
void TX_SSC (unsigned int, unsigned int, unsigned int); /* prototype
                                    def */

#pragma global                      /* make tx_ssc useable for external
                                    C modules */

void TX_SSC ( cmd, src, dest )
```

```
unsigned int cmd, src, dest;        /* local variables, word type
                                    cmd = command to be transmitted
                                    dest = destination board address*/
{
    if ( cmd == DIR_CHG )           /* command for remote board to
                                    toggle direction of running LEDs? */
    {
        SRCP1  = (int) & TX_BUFFER [1]; /* PEC1 source is 1st word of
                                    transmit buffer array */
        DSTP1  = (int) & SSCTB;     /* PEC1 destination is SSC transmit
                                    buffer reg */
        PECC1  = 0x400 + TRANSFER_CNT; /* increment source address,
                                    transfer defined # of words */
        SSCTIC = 0x79;              /* SSC transmit interrupt now served
                                    by PEC1 */
    }

    SSCTB = TX_BUFFER [0] = HEADER | ( src << 8 ) | ( dest << 3 ) |
     cmd;                           /* build transmit word with header,
                                    address and command and write it
                                    into SSC transmit register. In
                                    master mode the transmission starts
                                    instantly, in slave mode the
                                    transmission starts when the remote
                                    master is transmitting */
}
```

4.1.9 SSC transmit interrupt service routine - TX_INT.C

```
/*********************************************************************
 *  program  : tx_int.c                                              *
 *  name     : Andreas Hettmann    Siemens, Cupertino/CA             *
 *  date     : 5'96                                                  *
 *  function : interupt service routine SSC transmit                 *
 *********************************************************************/

#include <reg167.h>                 /* register definitions */
#include "ssc.h"                    /* definitions */

extern void INIT_SSC (unsigned int);
interrupt (txintno) void TX_INT (void); /* prototype definition */

#pragma global                      /* make tx_int useable for external
                                    C modules */
```

```
interrupt (txintno) void TX_INT ()
{
    T3IR  = 0;                      /* clear Timer3 interrupt request
                                    flag */
    T3    = WAIT_FOR_MASTER / 0.0256; /* [T3] = time to wait for remote
                                    master */
    T3CON = 0x00C6;                 /* count down, fc=fcpu/512, start
                                    timer */

    while ( SSCBSY && ~T3IR )       /* wait for one of the following
                                    events: */
    { }                             /* - remote master got transmitted
                                    word by supplying SSC clock for the
                                    board or
                                    /* - all other boards are in slave
                                    mode too, no remote master with SSC
                                    clock available, timer expires.
                                    Sending of command not neccesary. */

    if ( T3IR )
    {
        INIT_SSC ( MASTER );
        T2IE = 1;
    }

    T3CON = 0x0000;                 /* stop Timer6 */

    _bfld ( SSCTIC, 0xFF, SSC_T_INT ); /* next SSC receive interrupt
                                    calls interrupt service routine
                                    tx_int, no transfer via PEC1 */

    _putbit ( 0, DP3, 8 );          /* switch MRST to input mode rem.:
                                    in master mode already assigned, in
                                    slave mode assures that all inactive
                                    slave SSCs have high impedance TX
                                    output pins */
}
```

4.1.10 C header file - SSC.H

```
#define NO_OF_BOARDS       3          /* # of boards being supported
                                      (max. 16) */

#define mainintno          0x50       /* software int for main program
                                      ssc.c */
#define keyintno           0x20       /* hardware interrupt # of Timer0 */
#define t2intno            0x22       /* hardware interrupt # of Timer2 */
#define t5intno            0x25       /* hardware interrupt # of Timer5 */
#define rxintno            0x2E       /* hardware interrupt # of SSC
                                      receive */
#define txintno            0x2D       /* hardware interrupt # of SSC
                                      transmit */

#define ENABLE_INT         0x40       /* sets xxxIE */   /* ILVL   GLVL */
#define T2_INT             4 * 0x09 + 0x00
#define T5_INT             ENABLE_INT + 4 * 0x06 + 0x00
#define T0_INT             ENABLE_INT + 4 * 0x07 + 0x00
#define SSC_T_INT          ENABLE_INT + 4 * 0x08 + 0x00

#define SLAVE              0
#define MASTER             1          /* const 'MASTER' defines ONLY the
                                      status after being slave ! */

#define KEY_NOT_PRESSED    0          /* consts for routine key_int */
#define SINGLE_CLICK       1
#define DOUBLE_CLICK       2

#define P30_PASSIVE        0          /* const for routine scan (key) */
#define P30_ACTIVE         1

#define T_1                300        /* msec, max. time betw. 2 clicks
                                      (dbl click) */
#define T_2                3          /* msec, key scanning time to detect
                                      spikes */
#define LED_STEP           50         /* msec, time step for running
                                      LEDs */
#define MASTER_IDLE        50         /* msec, scan every ~ for slave's
                                      requests */
#define WAIT_FOR_MASTER    500        /* msec, slave waits for remote
                                      master */
#define BACKGND_PATTERN    0xF81F     /* background LED pattern in master
                                      mode */
```

```
#define STATUS_SHIFT      6          /* displayed board # fits into
                                     pattern */

#define HEADER            0x5000     /* shifted header for SSC
                                     transmission */
#define IDLE              0x00       /* dummy command for ssc_scan */
#define DIR_CHG           0x05       /* command 'toggle dir of running
                                     LEDs' */
#define MASTER_REQ        0x07       /* request from slave to become
                                     master */
#define OK                0x02
#define ALL               0x10       /* address for broadcasting */
#define TRANSFER_CNT      16         /* words transferred data after
                                     DIR_CHG */

#define SSC_EN            0x8000     /* SSC enable bit SSCCON.15 */
#define SSC_SLAVE         0x0000     /* init values for register SSCCON
                                     (slave) */
#define SSC_MASTER        0x4000     /* ... (master) */
#define SSCCON_INIT       0x005F     /* SSC: MSB first, 16 bits data ea.
                                     TX/RX */
#define F_CPU             20         /* MHz, system clock frequency */
#define BAUD_RATE         115200     /* bd, SSC baud rate */

#define LED_START_MASK    0x0707     /* running LED starts with this
                                     pattern */
```

4.2 System register initialization, ASM code - INIT_167.SRC

```
$DEBUG
$SYMBOLS
$XREF
$SEGMENTED
$EXTEND
$NOMOD166
$STDNAMES(reg167b.def)              ;all C167 SFR's & Bit names
;----------- end of primary controls --------------- program header: --
;*********************************************************************
;*  program  : INIT_167.SRC                                          *
;*  name     : Harald Lehmann, Siemens, Cupertino/CA                 *
;*  date     : 12'95                                                 *
;*  function : System Initialization large memory model (SEGMENTED)  *
;*********************************************************************
;----------------------------------------------------- definitions: --
```

```
NAME init_167_segmented                  ;Modulname

JMP_MAIN    EQU 50H                      ;SW-interrupt for MAIN program

;--------------------------------- system configuration for EVA167: --
;--------------------------------------------- externals, publics: --
;------------------------------------------- stack, PEC, register: --
SSKDEF      001b                         ;128 Words
Init_RB     REGBANK R0                   ;not need just as an example

;----------------------------------------------- CGROUPs, DGROUPs: --
ROM_C_Group CGROUP  Init_sec

;----------------------------------------------------------- code: --
Init_sec    SECTION     CODE        'INIT_ROM'
Init_proc   PROC  TASK  Init_167_Tsk        INTNO   Init_167_Int

    ASSUME DPP3:SYSTEM

    MOV DPP3,#3d                         ;system datapage RESET Value
    NOP                                  ;necessary for the next instruction

    ;system configuration SYSCON, BUSCON0, BUSCON1 and BUSCON2:


    MOV SYSCON,    #0010000100000100b ;(2104H)
                   ;^^^^||||^^^^^|||+XPER-SHARE: 0 = no share of X-Periph
                   ;^^^^||||^^^^^||+VISIBLE: 0= no visible mode for XPeriph
                   ;^^^^||||^^^^^|+XRAMEN: 1 = XRAM selected
                   ;^^^^||||^++++ -- don't care
                   ;^^^^||||+ WRCFG: write configuration
                   ;^^^^|||+ CLKEN: 1 = ENables system clock output on P3.15
                   ;^^^^||+ BYTDIS: 0 = BHE ENable
                   ;^^^^|+ ROMEN: 0 = int. ROM DISable
                   ;^^^^+ SGTDIS: 0 = segmented memory model ON
                   ;^^^+ ROMS1: 0 = NO ROM mapping to segment 1
                   ;+++ STKSZ: 001 = 128 Words system stack size

    MOV BUSCON0,   #0000010010111111b    ;(04BFH)
                   ;^^^^||||^^^^++++ MCTC 1111 = 0 WS
                   ;^^^^||||^^^+ RWDC0 1 = NO delay
                   ;^^^^||||^^+ MTTC0 0 = 1 WS, 1 = 0 WS
                   ;^^^^||||++ BTYP bus mode: 16 Bit, DEMUX
                   ;^^^^|||+ -- don't care
                   ;^^^^||+ ALECTL0 0 = NO ALE lengthening
                   ;^^^^|+ BUSACT0 1 = enables the BUSCONx function
                   ;^^^^+ -- don't care
```

```
                    ;^^^+ RDYEN0 0 = DISables the READY# function
                    ;^^+ -- don't care
                    ;++ = 00 always 0 for BUSCON0

     MOV BUSCON1,   #0000010010111111b    ;(04BFH)
                    ;^^^^||||^^^^++++ MCTC 1111 = 0 WS
                    ;^^^^||||^^^+ RWDC0 1 = NO delay
                    ;^^^^||||^^+ MTTC0 0 = 1 WS, 1 = 0 WS
                    ;^^^^||||++ BTYP bus mode: 16 Bit, DEMUX
                    ;^^^^|||+ -- don't care
                    ;^^^^||+ ALECTL0 0 = NO ALE lengthening
                    ;^^^^|+ BUSACT0 1 = enables the BUSCONx function
                    ;^^^^+ -- don't care
                    ;^^^+ RDYEN0 0 = DISables the READY# function
                    ;^^+ -- don't care
                    ;++ = 00 = address chip select

     MOV ADDRSEL1,  #0000000000000100b    ;(0004H)
                    ;^^^^||||^^^^++++ 64K range size
                    ;++++++++++++start address at 00 0000 (1st 64K-block)

     ;MOV BUSCON2,  #0000010001111101b    ;(047DH) EXAMPLE NOT NEEDED!
                    ;^^^^||||^^^^++++ MCTC 1110 = 1 WS
                    ;^^^^||||^^^+ RWDC1 0= with delay
                    ;^^^^||||^^+ MTTC1 1= 0 WS
                    ;^^^^||||++ BTYP bus mode: 8Bit MUX
                    ;^^^^|||+ -- don't care
                    ;^^^^||+ ALECTL1 0= NO ALE lengthening
                    ;^^^^|+ BUSACT1 1 = enables the BUSCONx function
                    ;^^^^+ -- don't care
                    ;^^^+ RDYEN1 0= DISables the READY# function
                    ;^^+ -- don't care
                    ;++ = 00 = address chip select

     ;MOV ADDRSEL2, #0001000000010000b    ;(1010H) EXAMPLE NOT
                                                   ;NEEDED!
                    ;^^^^||||^^^^++++ 4K range size
                    ;++++++++start address at 10 1000 (2nd 4K-block 1M-border)

MOV CP,#Init_RB ; overwrites default value
     ;NOP is not necessary for the next instruction

     ; SP    = FC00 = reset value (Stack Pointer)
     ; STKUN = FC00 = reset value (STacK UNderflow)
     ; MOV STKOV,#0FC00H+512
```

```
    ;STacK OVerflow at base +(128 Word = 256Bytes)

    DISWDT                          ;disable watchdog timer

    EINIT                           ;end of initialization

    MOV DPP0,#2d                    ;page for data

    TRAP #JMP_MAIN                  ;jump to main program

STAY_IDLE:
    IDLE                            ;IDLE-MODE should never reached,
                                    ;smthg wrong!
    JMP STAY_IDLE                   ;NEVERENDING stay in IDLE mode
                                    ;forever

    RETV                            ;to avoid the warning 'missing
                                    ;return'

Init_proc     ENDP
Init_sec      ENDS
;---------------------------------------------- end --------------------
END
```

4.3 Linker and locater control file - SSC.ILO

```
;***** SSC.ILO ***** (for the EVA167-monitor of HITEX)

TASK          INTNO = 0             ;task-name, Nr. 0 = RESET
INIT_167.lno                        ;filename [.LNO]

TASK          INTNO = 50h           ;Task-Name, Int.Name + Nr.= SW-
                                    ;INTERRUPT
SSC.lno                             ;filename [.LNO]
INIT_SYS.lno
INIT_SSC.lno
TX_SSC.lno

TASK          INTNO = 20h           ;Task-Name, Int.Name + Nr.= TIMER0-
                                    ;INTERRUPT
KEY_INT.lno                         ;filename [.LNO]

TASK          INTNO = 22h           ;Task-Name, Int.Name + Nr.= TIMER2-
                                    ;INTERRUPT
SSC_SCAN.lno                        ;filename [.LNO]
```

```
TASK            INTNO = 25h             ;Task-Name, Int.Name + Nr.= TIMER5-
                                        ;INTERRUPT
RUNLIGHT.lno                            ;filename [.LNO]

TASK            INTNO = 2Dh             ;Task-Name, Int.Name + Nr.= SSC-TX-
                                        ;INTERRUPT
TX_INT.lno                              ;filename [.LNO]

TASK            INTNO = 2Eh             ;Task-Name, Int.Name + Nr.= SSC-RX-
                                        ;INTERRUPT
RX_INT.lno                              ;filename [.LNO]

IRAMSIZE(2048)                          ;Internal RAM size is 2 KBytes for
                                        ;C167

CLASSES ('INIT_ROM' (0A100H-0A200H)) ;code of INIT_167 source file,
                                        start up
CLASSES ('CPROGRAM' (0A300H-0AFFFH)) ;code area for all C modules
CLASSES ('CNEAR'    (0B000H-0B1FFH)) ;byte & word variables and arrays
CLASSES ('CINITROM' (0B200H-0B3FFH)) ;
;CLASSES ('CUSTACK' (0B400H-0B5FFH)) ;not needed here (C user stack)
CLASSES ('CBITS'    (0FD00H-0FD01H)) ;bit variables

VECTAB (1000H)                          ;user interrupt vector table
                                        ;location

;Memory reservation for EVA167/165 with HITEX Monitor Telemon 167

RESERVE         ( MEMORY     (00000H-00221H,
                             01200H-0A00AH,
                             0FA00H-0FA3FH,
                             0FCC0H-0FCDFH))
```

4.4 Locater output file - SSC.MAP

80166 linker/locator v5.0 r0 SN070076-042

Date: May 17 1996 Time: 11:43:34 Page: 2 ssc

Table 2 Memory Map

Name	No	Start	End	Length	Type	Algn	Comb	Mem	T	Group	Class	Module
Reserved		000000h	000221h	000222h								
?INTVECT		001000h	001003h	000004h				ROM				init_167
?INTVECT		001080h	001083h	000004h				ROM				key_int
?INTVECT		001088h	00108bh	000004h				ROM				ssc_scan
?INTVECT		001094h	001097h	000004h				ROM				runlight
?INTVECT		0010b4h	0010b7h	000004h				ROM				tx_int
?INTVECT		0010b8h	0010bbh	000004h				ROM				rx_int
?INTVECT		001140h	001143h	000004h				ROM				ssc
Reserved		001200h	00a00ah	008e0bh								
Init_sec	0	00A100h	00A12Dh	00002Eh	CO DE	WO RD	PRI V	ROM	P	ROM_C _Group	INIT_RO M	init_167
SSC_3_PR	6	00A300h	00A325h	000026h	CO DE	WO RD	PUB L	ROM			CPROG RAM	ssc
INIT_SYS_ 1_PR	8	00A320h	00A3FFh	0000E0h	CO DE	WO RD	PUB L	ROM			CPROG RAM	init_sys
INIT_SSC_ 1_PR	9	00A400h	00A425h	000026h	CO DE	WO RD	PUB L	ROM			CPROG RAM	init_ssc
TX_SSC_1 _PR	10	00A426h	00A45Fh	00003Ah	CO DE	WO RD	PUB L	ROM			CPROG RAM	tx_ssc
KEY_INT_1 _PR	11	00A460h	00A587h	000128h	CO DE	WO RD	PUB L	ROM			CPROG RAM	key_int
SSC_SCAN _1_PR	13	00A588h	00A5C7h	000040h	CO DE	WO RD	PUB L	ROM			CPROG RAM	ssc_scan
RUNLIGHT _1_PR	15	00A5C8h	00A605h	00003Eh	CO DE	WO RD	PUB L	ROM			CPROG RAM	runlight
TX_INT_1_ PR	16	00A606h	00A653h	00004Eh	CO DE	WO RD	PUB L	ROM			CPROG RAM	tx_int
RX_INT_1_ PR	18	00A654h	00A72Dh	0000DAh	CO DE	WO RD	PUB L	ROM			CPROG RAM	rx_int
RX_INT_2_ NB	19	00B000h		000000h	DA TA	WO RD	PUB L	RAM	P	C166_D GROUP	CNEAR	rx_int

TX_INT_2_NB	17	00B000h		000000h	DATA	WORD	PUBL	RAM	P	C166_DGROUP	CNEAR	tx_int
SSC_SCAN_2_NB	14	00B000h		000000h	DATA	WORD	PUBL	RAM	P	C166_DGROUP	CNEAR	ssc_scan
KEY_INT_2_NB	12	00B000h		000000h	DATA	WORD	PUBL	RAM	P	C166_DGROUP	CNEAR	key_int
SSC_1_NB	4	00B000h	00B049h	00004Ah	DATA	WORD	PUBL	RAM	P	C166_DGROUP	CNEAR	ssc
C166_BSS	7	00B200h	00B209h	00000Ah	DATA	WORD	GLOB	ROM			CINITROM	ssc
Extended SFR Area		00F000h	00F1FFh	000200h				RAM				
ESFR_AREA	3	00F000h	00F1D7h	0001D8h	DATA	WORD	AT	RAM	P	DATAGRP		init_167
Reg. bank 0		00F600h	00F601h	000002h		WORD		RAM				
Reg. bank 1		00F602h	00F621h	000020h		WORD		RAM				
Reg. bank 2		00F622h	00F641h	000020h		WORD		RAM				
Reg. bank 3		00F642h	00F661h	000020h		WORD		RAM				
Reg. bank 4		00F662h	00F681h	000020h		WORD		RAM				
Reg. bank 5		00F682h	00F6A1h	000020h		WORD		RAM				
Reg. bank 6		00F6A2h	00F6C1h	000020h		WORD		RAM				
Reserved		00FA00h	00FA3Fh	000040h								
System Stack		00FB00h	00FBFFh	000100h				RAM				
Reserved		00FCC0h	00fCDFh	000020h								
SFR_AREA1	1	00FCE0h	00FCFFh	000020h	DATA	WORD	AT	RAM	P	DATAGRP		init_167
SSC_2_BI	5	00FD00h.00	00FD00h.01	000002h	BIT	BIT	PUBL	RAM			CBITS	ssc
SFR Area		00FE00h	00FFFFh	000200h				RAM				
SFR_AREA	2	00FE00h	00FFD7h	0001D8h	DATA	WORD	AT	RAM	P	DATAGRP		init_167

Table 3 Interrupt table

Vector	Intno	Start	Intnoname	Taskname
0000000h	0000h	000A100h	Init_167_Int	Init_167_Tsk
0000080h	0020h	000A460h	KEY_INT_INUM	KEY_INT_TASK
0000088h	0022h	000A588h	SSC_SCAN_INUM	SSC_SCAN_TASK
0000094h	0025h	000A5C8h	RUNLIGHT_INUM	RUNLIGHT_TASK
00000B4h	002Dh	000A606h	TX_INT_INUM	TX_INT_TASK
00000B8h	002Eh	000A654h	RX_INT_INUM	RX_INT_TASK
0000140h	0050h	000A300h	SSC_INUM	SSC_TASK

Error report :

- W 141: module INIT_167.lno(init_167): overlapping memory ranges 'SFR_AREA' and 'SFR Area'
- W 141: module INIT_167.lno(init_167): overlapping memory ranges 'ESFR_AREA' and 'Extended SFR Area'
- total errors: 0, warnings: 2

4.5 Miscellaneous files for compiling

FILE MAKE.BAT:

```
mk166 -f ssc.mak >error.txt
```

FILE SSC.MAK:

```
# Makefile "ssc.mak" made by A.Hettmann, Siemens, Cupertino, CA, 03/96
# This is for the HITEX simulator LIGHT and the BSO/T.
# You need these kind of enviroment:
# > PATH
c:\BAT;C:\DOS;C:\;D:\NC;C:\...\TASKING\c166\BIN;C:\...\hitex\sim
# > set C166INC=c:\...\tasking\c166\include
# > set C166LIB=c:\...\tasking\c166\LIB\NP
# > set TMPDIR=C:\TEMP

# **** CREATE HITEX-FILES ***
# This is done with Symbol preprocessor T C166 V2.31 "SP166TA.EXE"
ssc.sym: ssc.695
\166\hitex\tools\pp166ta.231\sp166ta ssc.695 -v -fo
```

```
# IEEE695-FILTER
ssc.695: ssc.out
ieee166 ssc.out ssc.695
# HEX-CONVERTER  *** CREATE HEX-FILES ***
ihex166 ssc.out ssc.hex

# LOCATER
# *** CREATE AN ABSOLUTE OBJEKT-FILE ***
ssc.out: init_167.lno ssc.lno init_ssc.lno key_int.lno rx_int.lno
tx_int.lno runlight.lno ssc_scan.lno init_sys.lno tx_ssc.lno ssc.ilo
L166 @ssc.ilo NOVECINIT to ssc.out

# LINKER
init_167.lno: init_167.obj
L166 init_167.obj sg_sfrta.obj to init_167.lno
ssc.lno: ssc.obj
L166 ssc.obj to ssc.lno
init_ssc.lno: init_ssc.obj
L166 init_ssc.obj to init_ssc.lno
key_int.lno: key_int.obj
L166 key_int.obj to key_int.lno
rx_int.lno: rx_int.obj
L166 rx_int.obj to rx_int.lno
tx_int.lno: tx_int.obj
L166 tx_int.obj to tx_int.lno
runlight.lno: runlight.obj
L166 runlight.obj to runlight.lno
ssc_scan.lno: ssc_scan.obj
L166 ssc_scan.obj to ssc_scan.lno
init_sys.lno: init_sys.obj

L166 init_sys.obj to init_sys.lno
tx_ssc.lno: tx_ssc.obj
L166 tx_ssc.obj to tx_ssc.lno

# ASSEMBLER FOR MAINPROGRAM
ssc.obj: ssc.src
A166 ssc.src DB EP CASE

# ASSEMBLER FOR SSC INIT
init_ssc.obj: init_ssc.src
A166 init_ssc.src DB EP CASE

# ASSEMBLER FOR KEY INTERRUPT SERVICE ROUTINE
key_int.obj: key_int.src
A166 key_int.src DB EP CASE
```

```
# ASSEMBLER FOR RECEIVE INTERRUPT SERVICE ROUTINE
rx_int.obj: rx_int.src
A166 rx_int.src DB EP CASE

# ASSEMBLER FOR TRANSMIT INTERRUPT SERVICE ROUTINE
tx_int.obj: tx_int.src
A166 tx_int.src DB EP CASE

# ASSEMBLER FOR LED RUNLIGHT INTERRUPT SERVICE ROUTINE
runlight.obj: runlight.src
A166 runlight.src DB EP CASE

# ASSEMBLER FOR SSC SCAN INTERRUPT SERVICE ROUTINE
ssc_scan.obj: ssc_scan.src
A166 ssc_scan.src DB EP CASE

# ASSEMBLER FOR SYSTEM INIT
init_sys.obj: init_sys.src
A166 init_sys.src DB EP CASE

# ASSEMBLER FOR TRANSMIT ROUTINE
tx_ssc.obj: tx_ssc.src
A166 tx_ssc.src DB EP CASE
# C-COMPILER FOR MAINPROGRAM
ssc.src: ssc.c ssc.h
C166 -gf -t -Ml -x -s ssc.c

# C-COMPILER FOR SSC INIT
init_ssc.src: init_ssc.c ssc.h
C166 -gf -t -Ml -x -s init_ssc.c

# C-COMPILER FOR KEY INTERRUPT SERVICE ROUTINE
key_int.src: key_int.c ssc.h
C166 -gf -t -Ml -x -s key_int.c

# C-COMPILER FOR RECEIVE INTERRUPT SERVICE ROUTINE
rx_int.src: rx_int.c ssc.h
C166 -gf -t -Ml -x -s rx_int.c

# C-COMPILER FOR TRANSMIT INTERRUPT SERVICE ROUTINE
tx_int.src: tx_int.c ssc.h
C166 -gf -t -Ml -x -s tx_int.c

# C-COMPILER FOR LED RUNLIGHT INTERRUPT SERVICE ROUTINE
runlight.src: runlight.c ssc.h
C166 -gf -t -Ml -x -s runlight.c
```

```
# C-COMPILER FOR SSC SCAN INTERRUPT SERVICE ROUTINE
ssc_scan.src: ssc_scan.c ssc.h
C166 -gf -t -Ml -x -s ssc_scan.c

# C-COMPILER FOR SYSTEM INIT
init_sys.src: init_sys.c ssc.h
C166 -gf -t -Ml -x -s init_sys.c

# C-COMPILER FOR TRANSMIT ROUTINE
tx_ssc.src: tx_ssc.c ssc.h
C166 -gf -t -Ml -x -s tx_ssc.c


# ASSEMBLER FOR INIT
init_167.obj: init_167.src
A166 init_167.src DB EP CASE
```

FILE SETPATH.BAT:

```
PROMPT
rem *****************************************************************
rem *          "setpath.bat" prepare the enviroment for              *
rem *           the PECR167 demo on the EVAL167 kit with             *
rem *                Tasking 166 Evaluation Package                  *
rem *                      *** SAB-C167 ***                          *
rem *                Harald Lehmann, SIEMENS, cupertino 11'95        *
rem *****************************************************************
path=%PATH%;c:\166\TASKING\BIN;c:\EVAL167\HITEX;c:\EVAL167\HITEX\EXAMPLE
set CCDEMO=c:\EVAL167\TASKING
set C166INC=c:\EVAL167\TASKING\include
set LINK166=LIBPATH(c:\166\TASKING\lib\EXT) c166t.lib
set LOCATE166=CASE
prompt $p$g
```

FILE BOOT.BAT:

```
\eval167\hitex\btld eva167.hex -r
```

FILE BTDL.CFG:

```
// configuration file for BTLD bootstrap loader for Siemens 16x  V1.00
// Copyright (c) 1994                    Hitex Systementwicklung GmbH
//
// supported keywords and values:
//   COM         communication port, selection out of (1, 2)
//   BAUD        baudrate, selection out of 9600/19200/38400
//   TYPE        type of processor, selection out of
//               (0 = 8xC166, 1 = C165/C167)
//   SYSCON      hexadecimal value of register SYSCON to be loaded
//   BUSCON0     hexadecimal value of register BUSCON0 to be loaded
//   BUSCON1     hexadecimal value of register BUSCON1 to be loaded
//   ADDRSEL1    hexadecimal value of register ADDRSEL1 to be loaded
//   BUSCON2     hexadecimal value of register BUSCON2 to be loaded
//   ADDRSEL2    hexadecimal value of register ADDRSEL2 to be loaded
//   BUSCON3     hexadecimal value of register BUSCON3 to be loaded
//   ADDRSEL3    hexadecimal value of register ADDRSEL3 to be loaded
//   BUSCON4     hexadecimal value of register BUSCON4 to be loaded
//   ADDRSEL4    hexadecimal value of register ADDRSEL4 to be loaded
//
// the default value of not named busconfiguration registers is 0000
//
COM          1
BAUD         38400
TYPE         1          // C167
SYSCON       2100
ADDRSEL1     4
BUSCON0      04BF
BUSCON1      04BF
//
//  MOV SYSCON, #0010000100000000b      ;(2100H)
//              ;^^^^||||^^^^^|||+XPER-SHARE: 0 = no share of X-Periph
//              ;^^^^||||^^^^^||+VISIBLE: 0= no visible mode for XPeriph
//              ;^^^^||||^^^^^|+XRAMEN: 0 = XRAM deselected
//              ;^^^^||||^++++ -- don't care
//              ;^^^^||||+ WRCFG: write configuration
//              ;^^^^|||+ CLKEN:1 = ENables system clock output on P3.15
//              ;^^^^||+ BYTDIS: 0 = BHE ENable
//              ;^^^^|+ ROMEN: 0 = int. ROM DISable
//              ;^^^^+ SGTDIS: 0 = segmented memory model ON
//              ;^^^+ ROMS1: 0 = NO ROM mapping to segment 1
//              ;+++ STKSZ: 001 = 128 Words system stack size
//
```

```
//   MOV BUSCON0, #0000010010111111b      ;(04BFH)
//                ;^^^^||||^^^^++++ MCTC 1111 = 0 WS
//                ;^^^^||||^^^+ RWDC0 1 = NO delay
//                ;^^^^||||^^+ MTTC0 0 = 1 WS, 1 = 0 WS
//                ;^^^^||||++ BTYP bus mode: 16 Bit, DEMUX
//                ;^^^^|||+ -- don't care
//                ;^^^^||+ ALECTL0 0 = NO ALE lengthening
//                ;^^^^|+ BUSACT0 1 = enables the BUSCONx function
//                ;^^^^+ -- don't care
//                ;^^^+ RDYEN0 0 = DISables the READY# function
//                ;^^+ -- don't care
//                ;++ = 00 always 0 for BUSCON0
//
//   MOV BUSCON1, #0000010010111111b      ;(04BFH)
//                ;^^^^||||^^^^++++ MCTC 1111 = 0 WS
//                ;^^^^||||^^^+ RWDC0 1 = NO delay
//                ;^^^^||||^^+ MTTC0 0 = 1 WS, 1 = 0 WS
//                ;^^^^||||++ BTYP bus mode: 16 Bit, DEMUX
//                ;^^^^|||+ -- don't care
//                ;^^^^||+ ALECTL0 0 = NO ALE lengthening
//                ;^^^^|+ BUSACT0 1 = enables the BUSCONx function
//                ;^^^^+ -- don't care
//                ;^^^+ RDYEN0 0 = DISables the READY# function
//                ;^^+ -- don't care
//                ;++ = 00 = address chip select
//
//   MOV ADDRSEL1, #0000000000000100b      ;(0004H)
//                 ;^^^^||||^^^^++++ 64K range size
//                 ;++++++++++++start address at 00 0000 (1st 64K-block)
```

FILE HIT1.BAT:

```
\166\hitex\mon\hit_167.exe -p1 -b384 -y -rhit_167.rst

REM used command line parameters for HIT_167.EXE:
REM            -p1                    COM1 for communication
REM            -b384                  38400 baud
REM            -y                     assume YES for reloading of recent
                                      used files
REM            -rhit_167.rst          use restore file hit_167.rst
```

5 LED display board - schematic

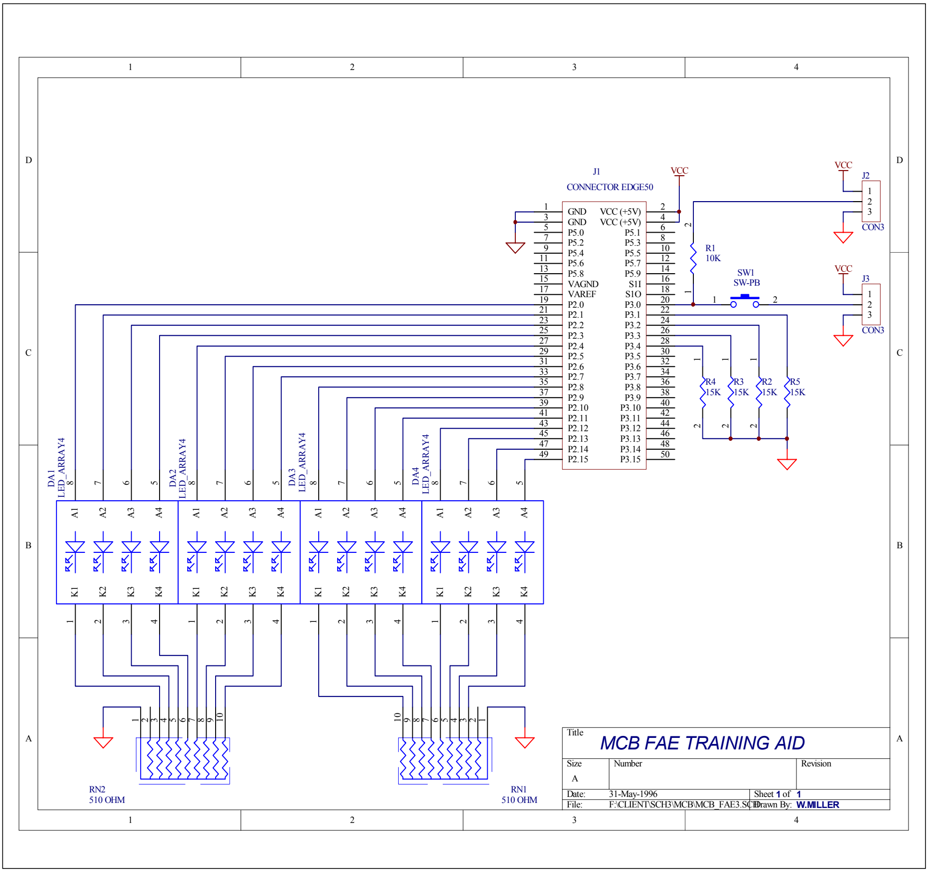


Figure 10 LED display board - schematic

Rem.: Jumpers for Board ID are to be set between P2.1 / P3.1 (Bit 0, LSB), P2.2 / P3.2 (Bit 1), P2.3 / P3.3 (Bit 2) and P2.4/P3.4 (Bit 3, MSB). When P2.1 through P2.4 are set to log. '1', reading the input levels of P3.1 through P3.4 results in log. '1' for positions with jumper and log. '0' without jumper. Additionally, there have jumpers to be set on the CON3 connectors so that pushing the button results in a H/L or L/H transition at P3.0 .

http://www.infineon.com

Published by Infineon Technologies AG