SIEMENS

Microcontrollers

ApNote AP1643

☑ additional file
AP164301.EXE available

C163 on-chip Flash Programming Tool

This Flash Programming Tool allows to program the on-chip Flash memory of the C163 (design step BA or later). Therefore a “boot support” (bootstrap loader program) must be available either in the Flash memory itself or in an external memory.
The Flash Programming Tool takes care that this bootstrap loader code will never be erased accidentally.

Author: Frank Riemenschneider / Siemens Microelectronics San Jose

AP1643 ApNote - Revision History		
Actual Revision : Rel.01	Previous Revison: none (Original Version)	
Page of actual Rel.	Page of prev. Rel.	

1 Introduction

To program the on-chip Flash memory of the C163 a bootstrap loader (BSL) is needed to load the flash programming routines and the program code via serial interface. On C16x microcontrollers the bootstrap loader code is usually stored in a special boot-ROM, which is not available on C163 Flash. Here a so called “boot support” can be used instead. The code of this bootstrap loader program may resides either in the external ROM or in the internal Flash memory in sector 2 or sector 0.

In case that this bootstrap loader code is located in the internal Flash memory some considerations have to be taken into account when reprogramming the Flash memory. Especially it is important to know, that there will be no way to reboot the C163 Flash without external memory, if the BSL-code has been overwritten. To avoid that the BSL-code is overwritten accidentally this Flash Programming Tool has been developed. This tool protects the bootstrap loader against unintentional erase and makes sure that even during the programming process a bootstrap loader code is always existent.

In addition, it is possible to change the location of the BSL-code. For example from external to internal memory or from sector 2 of the Flash memory to sector 0.

2 Activating the Bootstrap Loader (Boot Support)

The first step before programming the C163 is to activate the Bootstrap Loader. Therefore an appropriate bootstrap loader program (see Appendix A) must be located either in the external or internal memory.

2.1 BSL-code in external ROM

The BSL-code must be mapped either to location 00’0000 H or to location 02’0000 H.

After reset the microcontroller will execute the bootstrap loader program in the external memory if pin EA is sampled **low** during reset time.

Furthermore, it is possible to change the reset value of the code segment pointer CSP from 00H (default) to 02H by pulling down pin P0L.4 during reset. After reset code will be executed either from start address 00’0000 H or 02’0000 H (design step BA or later).

2.2 BSL-code in Flash memory

The BSL-code must reside either in sector 0 or in sector 2 of the Flash memory.

After reset the microcontroller will execute the bootstrap loader program in the internal memory if pin EA is sampled **high** during reset time.

Furthermore, it is possible to change the reset value of the code segment pointer CSP from 00H (default) to 02H by pulling down pin P0L.4 during reset. After reset code will be executed either from start address 00’0000 H (sector 0) or 02’0000 H (sector 2) (design step BA or later).

3 Running the Flash Programming Tool

If the bootstrap loader is activated the microcontroller executes the bootstrap loader code (see Appendix A) after reset. Then, the microcontroller expects exactly 32 bytes via the serial port. When starting the Flash Programming Tool these 32 bytes (a download routine, see Appendix B) are sent to the microcontroller. The 32 bytes will be stored at location 00'FA40 H in the internal RAM and executed after the last byte has been received by the microcontroller. This download routine will receive and store the flash programming routines (see Appendix C & D) that are sent by the Flash Programming Tool.

4 The Commands of the Bootstrap Loader Tool

4.1 Program Flash

The Flash memory can only be programmed with code and data available in Intel-Hex file format.

After selecting the Intel-Hex file you have to specify the location of the bootstrap loader code before and after the programming process. This information is very important to avoid unintentional erase of the bootstrap loader code.

It is also important to know where Flash sector 0 will be located during normal program execution. For example, if sector 0 is mapped to segment 1 it will be programmed with code or data that is located (by the linker) at 01’0000 H to 01’7FFF H.

Once you have selected the right settings the Intel-Hex file will be sort and a new file (same name but extension (*.NEW), same directory as the Hex file) will be created, where all bursts (group of 64 bytes) are listed. The user doesn’t have to care about this file and he can delete it after programming.

The next window shows all the sectors that will be erased and programmed when starting the program process. Depending on the Intel-Hex file and the settings concerning the locations of the bootstrap loader code the corresponding sector-check-boxes are selected or not and enabled (white colored) or disabled (gray colored).

4.2 Show Flash Status

This command shows the contents of the 4 Flash Status Registers.

4.3 Reset the microcontroller

On this command the microcontroller will execute a software reset. A software reset will not evaluate pin P0L.4. That means that the microcontroller will exit the bootstrap loader mode after a software reset **if** pin P0L.4 = low activates the bootstrap loader mode.
To continue with programming after execution of this command the bootstrap loader of the microcontroller needs to be reactivated and the Flash Programming Tool must be Re-Started.

4.4 Re-Start the Flash Tool

This command will restart the Flash Programming Tool. Note that the bootstrap loader of the microcontroller must also be reactivated.

4.5 Exit

To exit the Flash Programming Tool. The microcontroller will be left in an endless waiting loop.

Appendix

A The Bootstrap Loader Code

After reset the microcontroller will execute the following bootstrap loader program:

```
$ SEGMENTED
$INCLUDE (REG163.INC)
ASSUME DPP3: SYSTEM
;*****************************************************************************
Ack_byte        EQU     0D5h            ; Acknowledge Byte of Bootstrap Loader

BOOTSTRAP  SECTION CODE   AT 000000H
ASSUME DPP3: SYSTEM
BOOT        PROC    NEAR
;-----------------------------------------------------------------------------;*
;* Bootstrap Loader: Receives 32 bytes via serial port 0 and stores them *
;* into the internal RAM. Afterwards it performs a jump to the beginning of *
;* the loaded RAM area (0FA40h) and executes the loaded data as a program. *
;------------------------------------------------------------------------------
BSL_INIT:                           ; don't load syscon or buscon !
        DISWDT                      ; disable watchdog timer
        MOV     SP, #0FA40h         ; set stackpointer
        MOV     STKUN, #0FA40h      ; set stack underflow pointer
        MOV     STKOV, #0FA0Ch      ; set stack overflow pointer
        MOV     CP, #0FA00h         ; set registerbank
WaitStartBit:
        JB  P3.11,WaitStartBit      ; wait for start bit at RXD0
        BSET    T6R                 ; start timer T6
WaitStopBit:
        JNB     P3.11, WaitStopBit  ; wait for stop bit at RXD0
        BCLR    T6R                 ; stop timer T6
        MOV     MDL,T6
        SUB     MDL,#36             ; rounding & adjustment
        MOV     R1,#72              ; baudrate = (T6 / 72) - 1
        DIVU    R1
InitSerialPort:
        MOV     S0BG,MDL            ; load baudrate generator
        BSET    P3.10               ; initialize TXD0 output
```

```
        BSET    DP3.10
        MOV     S0CON,#8001h            ; initialize serial port 0:
                                        ; 8-bit data, no parity, one stop bit
                                        ; receiver not started: single wire option
SendAcknowledge:
        MOV     S0TBUF,#Ack_byte        ; send acknowledge byte for baudrate check
WaitTransmOver:
        JNB     S0TIR, WaitTransmOver   ; wait till end of transmission
        BCLR    S0TIR                   ; clear transmit interrupt request flag
        BSET    S0REN                   ; receiver enabled
ReceiveData:
        MOV     R0, #0FA40h
ReceiveLoop:
        JNB     S0RIR, ReceiveLoop      ; wait for receive interrupt request
        MOVB    [R0], S0RBUF            ; store received byte
        BCLR    S0RIR                   ; clear receive interrupt request flag
        CMPI1   R0, #0FA5Fh             ; all bytes received ?
        JMPR    CC_NE, ReceiveLoop      ; if not, continue loop
        JMPS    0, 0FA40h               ; yes: jump to RAM routine
        NOP                             ; dummy
;*******************************************************************************
BOOT        ENDP
BOOTSTRAP   ENDS
            END
```

B The Download Routine (32 bytes)

These 32 bytes are sent by the Flash Programming Tool when the microcontroller is executing the BSL-code. It is a download routine that receives and stores in the internal RAM the Flash Programming Routines.

```
$INCLUDE (REG163.INC)
BOOTSTRAP  SECTION CODE   AT 0FA40H

BYTE_COUNTER              EQU     R1
ADDRESS                   EQU     R2
NUMBER_BYTES              EQU     19H
TRANS_START               EQU     22H
STARTADDRESS              EQU     0F600H
;*******************************************************************************
BOOT        PROC    NEAR
START:
        MOV     S0TBUF, #TRANS_START
        MOV     BYTE_COUNTER, #NUMBER_BYTES
        MOV     ADDRESS, #STARTADDRESS
LOOP:
        BCLR S0RIR
WAIT:   JNB     S0RIR, WAIT
        MOVB    [ADDRESS], S0RBUF
        ADD     ADDRESS, #1
        CMPD1   BYTE_COUNTER, #1
        JMPR    CC_UGE, LOOP
        JMPA    CC_UC, STARTADDRESS
;*******************************************************************************
BOOT        ENDP
BOOTSTRAP   ENDS
            END
```

C The Flash Programming Routine

This program is running on the microcontroller while the Flash Programming Tool is running on PC.

```
$ SEGMENTED
ASSUME DPP3: SYSTEM
$ INCLUDE (REG163.INC)

BUFFER_START            EQU     0FA20h
;*******************************************************************************
DATA_BUFFER  SECTION  DATA AT BUFFER_START
    BUFFER:     DSW     32
DATA_BUFFER ENDS
;*******************************************************************************
STACK  SECTION  DATA AT 0FA60h                ;RESERVE INTERNAL RAM FOR STACK
    X:          DSB 1A0h
STACK   ENDS
;*******************************************************************************
REG_BANK  SECTION  DATA AT 0FA00h             ;RESERVE INTERNAL RAM FOR GPR's
    Y:          DSB 20h
REG_BANK    ENDS
;*******************************************************************************
SFR  SECTION  DATA AT 0FE00h                  ;RESERVE INTERNAL RAM FOR GPR's
    ZZ:         DSB 200h
SFR     ENDS
;*******************************************************************************
READ_BYTE                   EQU     RL0
READ_WORD                   EQU     R0

ERROR_STATUS                EQU     RL1

TEMP_WORD                   EQU     R2
TEMP_BYTE                   EQU     RL2

STATUS                      EQU     R3
STATUS_L                    EQU     RL3
STATUS_H                    EQU     RH3
```

```
DATA_ADDR                   EQU     R4
DATA_ADDR_LOW               EQU     RL4
DATA_ADDR_HIGH              EQU     RH4

COMM_ADDR_AAAA              EQU     R5
COMM_ADDR_5554              EQU     R6
COMM_ADDR_A0F2              EQU     R7
COMM_DATA                   EQU     R8
BUFFER_PTR                  EQU     R9
COUNTER                     EQU     R10
SEGMENT_ADDR                EQU     R11
CHECKSUM                    EQU     R12
_256_                       EQU     R13
;*******************************************************************************
START_COMMUNICATION         EQU     11H
END_COMMUNICATION           EQU     12H

ERASE_COMMAND               EQU     20H
ERASE_SECTOR_0_OPERAND      EQU     20H
ERASE_SECTOR_1_OPERAND      EQU     21H
ERASE_SECTOR_2_OPERAND      EQU     22H
ERASE_SECTOR_3_OPERAND      EQU     23H

PROGRAM_FLASH_COMMAND       EQU     33H

READ_STATUS_COMMAND         EQU     40H
STATUS_SECTOR_0_OPERAND     EQU     40H
STATUS_SECTOR_1_OPERAND     EQU     41H
STATUS_SECTOR_2_OPERAND     EQU     42H
STATUS_SECTOR_3_OPERAND     EQU     43H

RESET_TO_READ_COMMAND       EQU     55H
CLEAR_STATUS_COMMAND        EQU     66H
SOFTWARE_RESET_COMMAND      EQU     00H

NO_ERROR                    EQU     88H
COMMAND_ERROR               EQU     91H
TIME_OUT_ERROR              EQU     92H
FLAG_ERROR                  EQU     93H
```

```
OPERAND_ERROR              EQU     94H
;*******************************************************************************
MAIN_PRG  SECTION CODE   AT 0FC00h
MAIN        PROC    NEAR
;*******************************************************************************
        MOV     CP, #0FA00h
        DISWDT
        MOV     SYSCON, #1110$1110$0000$0000b     ;No circular stack
        MOV     DPP3, #3
        MOV     SP, #0FC00h
        MOV     STKUN, #0FC00h
        MOV     STKOV, #0FA60h
        MOV     COMM_ADDR_AAAA, #0AAAAh
        MOV     DPP2, #0Ah                        ;POINTER TO FLASH (HERE SEG2)
        MOV     COMM_ADDR_5554, #5554h
        MOV     DPP1, #09h                        ;POINTER TO FLASH (HERE SEG2)
        MOV     COMM_ADDR_A0F2, #0A0F2h
        BCLR    S0RIR
        MOV     READ_WORD, #00H
        MOV     _256_, #256

        MOV     S0TBUF, #START_COMMUNICATION      ;the micro is ready to receive
                                                  ;commands
  CONTINUE:                                       ;main loop
        CALLR   READ_COMMAND                      ;read & execute command
        CALLA   cc_UC, CHECK_SUM                  ;check the Checksum byte
        MOVB    S0TBUF, ERROR_STATUS              ;SEND ERROR-STATUS
        JMPR    cc_UC, CONTINUE
;*******************************************************************************
READ_COMMAND:
        MOV     CHECKSUM, #00H                    ;clear Checksum byte
        CALLA   cc_UC, READ                       ;read command
                                                  ;identify and execute command
        CMPB    READ_BYTE, #PROGRAM_FLASH_COMMAND
        JMPA    CC_EQ, PROGRAM_FLASH

        CMPB    READ_BYTE, #READ_STATUS_COMMAND
        JMPA    CC_EQ, READ_STATUS
```

```
        CMPB    READ_BYTE, #RESET_TO_READ_COMMAND
        JMPR    CC_EQ, RESET_TO_READ

        CMPB    READ_BYTE, #CLEAR_STATUS_COMMAND
        JMPR    CC_EQ, CLEAR_STATUS

        CMPB    READ_BYTE, #ERASE_COMMAND
        JMPR    CC_EQ, ERASE_SECTOR

        CMPB    READ_BYTE, #SOFTWARE_RESET_COMMAND
        JMPR    CC_EQ, SOFTWARE_RESET

        MOV     ERROR_STATUS, #COMMAND_ERROR
        RET
;*******************************************************************************
SOFTWARE_RESET:
        BCLR    S0TIR
        MOV     S0TBUF, #END_COMMUNICATION        ;signal end of communication
    XX: JNB     S0TIR, XX
        SRST
        NOP
;*******************************************************************************
RESET_TO_READ:
        MOV     COMM_DATA, #0F0h                  ;COMMAND DATA
        MOV     [COMM_ADDR_AAAA], COMM_DATA
        JMPA    cc_UC, READ_ERROR_FLAGS
;*******************************************************************************
CLEAR_STATUS:
        MOV     COMM_DATA, #0F5h                  ;COMMAND DATA
        MOV     [COMM_ADDR_AAAA], COMM_DATA
        JMPA    cc_UC, READ_ERROR_FLAGS
;*******************************************************************************
ERASE_SECTOR:
        CALLA   cc_UC, READ

        CMPB    READ_BYTE, #ERASE_SECTOR_0_OPERAND
        JMPR    CC_EQ, ERASE_SECTOR_0

        CMPB    READ_BYTE, #ERASE_SECTOR_1_OPERAND
```

```
        JMPR    CC_EQ, ERASE_SECTOR_1

        CMPB    READ_BYTE, #ERASE_SECTOR_2_OPERAND
        JMPR    CC_EQ, ERASE_SECTOR_2

        CMPB    READ_BYTE, #ERASE_SECTOR_3_OPERAND
        JMPR    CC_EQ, ERASE_SECTOR_3

        MOV     ERROR_STATUS, #OPERAND_ERROR
        RET

ERASE_SECTOR_0:
        MOV     DPP0, #00h                      ;POINTER TO SECTOR 0, SEGMENT 0
        JMPR    cc_UC, ERASE
ERASE_SECTOR_1:
        MOV     DPP0, #06h                      ;POINTER TO SECTOR 1, SEGMENT 1
        JMPR    cc_UC, ERASE
ERASE_SECTOR_2:
        MOV     DPP0, #08h                      ;POINTER TO SECTOR 2, SEGMENT 2
        JMPR    cc_UC, ERASE
ERASE_SECTOR_3:
        MOV     DPP0, #0Ah                      ;POINTER TO SECTOR 3, SEGMENT 2
;------------------------------------------------------------------------------
ERASE:
        MOV     DATA_ADDR, #00h  ;FIRST LOCATION WITHIN THE TARGET SECTOR (DPP0)
                                                ;CYCLE 1
        MOV     COMM_DATA, #0AAh                ;COMMAND DATA
        MOV     [COMM_ADDR_AAAA], COMM_DATA
                                                ;CYLCE 2
        MOV     COMM_DATA, #55h
        MOV     [COMM_ADDR_5554], COMM_DATA
                                                ;CYLCE 3
        MOV     COMM_DATA, #80h
        MOV     [COMM_ADDR_AAAA], COMM_DATA
                                                ;CYLCE 4
        MOV     COMM_DATA, #0AAh
        MOV     [COMM_ADDR_5554], COMM_DATA
                                                ;CYLCE 5
        MOV     COMM_DATA, #55h
```

```
        MOV     [COMM_ADDR_AAAA], COMM_DATA
                                                ;CYLCE 6
        MOV     COMM_DATA, #30h
        MOV     [DATA_ADDR], COMM_DATA

        JMPA    cc_UC, READ_ERROR_FLAGS
;*******************************************************************************
PROGRAM_FLASH:
        MOV     BUFFER_PTR, #BUFFER_START
        CALLR   READ                            ;read segment address
        MOV     SEGMENT_ADDR, READ_WORD

        CALLR   READ                            ;read data address high byte
        MOVB    DATA_ADDR_HIGH, READ_BYTE
        CALLR   READ                            ;read data address low byte
        MOVB    DATA_ADDR_LOW, READ_BYTE

   READ_LOOP:
        CALLR   READ                          ;LOAD BUFFER with 64 data bytes
        MOVB    [BUFFER_PTR], READ_BYTE         ;store received data byte
        CMPI1   BUFFER_PTR, #BUFFER_START + 63
        JMPR    cc_ULT, READ_LOOP
;-------------------------------------------------------------------------------
ENTER_BURST_MODE:      ;ADDRESS IN DATA_ADDR(MULTIPLE OF 40h) AND SEGMENT_ADDR
        MOV     BUFFER_PTR, #BUFFER_START
        MOV     TEMP_WORD, [BUFFER_PTR]
                                                ;CYCLE 1
        MOV     COMM_DATA, #050h                ;COMMAND DATA
        MOV     [COMM_ADDR_AAAA], COMM_DATA
                                                ;CYCLE 2
        EXTS    SEGMENT_ADDR, #1
        MOV     [DATA_ADDR], TEMP_WORD
;-------------------------------------------------------------------------------
LOAD_BURST_DATA:
        MOV     COUNTER, #29                    ;30 x
    wrt_loop:
        ADD     BUFFER_PTR, #2
        MOV     TEMP_WORD, [BUFFER_PTR]
        MOV     [COMM_ADDR_A0F2], TEMP_WORD
```

```
        CMPD1   COUNTER, #00h
        JMPR    cc_UGT, wrt_loop
;-------------------------------------------------------------------------
STORE_BURST_BUFFER:
        ADD     BUFFER_PTR, #2
        MOV     TEMP_WORD, [BUFFER_PTR]
                                                ;CYCLE 1
        MOV     COMM_DATA, #0AAh                ;COMMAND DATA
        MOV     [COMM_ADDR_AAAA], COMM_DATA
                                                ;CYLCE 2
        MOV     COMM_DATA, #55h
        MOV     [COMM_ADDR_5554], COMM_DATA
                                                ;CYCLE 3
        MOV     COMM_DATA, #0A0h                ;COMMAND DATA
        MOV     [COMM_ADDR_AAAA], COMM_DATA
                                                ;CYCLE 4
        EXTS    SEGMENT_ADDR, #1
        MOV     [DATA_ADDR], TEMP_WORD
                                                ;CHECK BUSY AND FSR
        JMPR    cc_UC, READ_ERROR_FLAGS
;*************************************************************************
READ_STATUS:
        CALLR   READ                            ;READ SECTOR NR.-OPERAND

        MOV     DATA_ADDR, #00h  ;FIRST LOCATION WITHIN THE TARGET SECTOR (DPP0)
        MOV     COMM_DATA, #0FAh                ;COMMAND DATA

        CMPB    READ_BYTE, #STATUS_SECTOR_0_OPERAND
        JMPR    CC_EQ, READ_STATUS_SECTOR_0

        CMPB    READ_BYTE, #STATUS_SECTOR_1_OPERAND
        JMPR    CC_EQ, READ_STATUS_SECTOR_1

        CMPB    READ_BYTE, #STATUS_SECTOR_2_OPERAND
        JMPR    CC_EQ, READ_STATUS_SECTOR_2

        CMPB    READ_BYTE, #STATUS_SECTOR_3_OPERAND
        JMPR    CC_EQ, READ_STATUS_SECTOR_3
```

```
        MOV     ERROR_STATUS, #OPERAND_ERROR
        RET
;-------------------------------------------------------------------------
READ_STATUS_SECTOR_0:
  SEGMENT_0:    MOV    DPP0, #00h             ;POINTER TO SECTOR 0, SEGMENT 0
                JMPR   cc_UC, READ_N_SEND_STATUS
READ_STATUS_SECTOR_1:
        MOV     DPP0, #06h                    ;POINTER TO SECTOR 1, SEGMENT 1
                JMPR   cc_UC, READ_N_SEND_STATUS
READ_STATUS_SECTOR_2:
        MOV     DPP0, #08h                    ;POINTER TO SECTOR 2, SEGMENT 2
                JMPR   cc_UC, READ_N_SEND_STATUS
READ_STATUS_SECTOR_3:
        MOV     DPP0, #0Ah                    ;POINTER TO SECTOR 3, SEGMENT 2
;-------------------------------------------------------------------------
READ_N_SEND_STATUS:
        MOV     [COMM_ADDR_AAAA], COMM_DATA   ;WRITE COMMAND
        MOV     STATUS, [DATA_ADDR]           ;READ STATUS
        BCLR    S0TIR
        MOVB    S0TBUF, STATUS_H
wait3:  JNB     S0TIR, wait3
        MOVB    S0TBUF, STATUS_L
        MOVB    ERROR_STATUS, #NO_ERROR
        RET
;*************************************************************************
READ:
WAIT1:  JNB     S0RIR, WAIT1
        MOVB    READ_BYTE, S0RBUF
        BCLR    S0RIR
        ADD     CHECKSUM, READ_WORD
        RET
;*************************************************************************
CHECK_SUM:                                    ;calculates Checksum byte
        MOV     MDL, CHECKSUM
        DIVU    _256_
        MOV     TEMP_WORD, MDH
        NEG     TEMP_WORD
        BCLR    S0TIR
        MOVB    S0TBUF, TEMP_BYTE
```

```
wait2:  JNB     S0TIR, wait2
        RET
;*******************************************************************************
READ_ERROR_FLAGS:                ;CHECK ERROR FLAGS - SAME FOR ALL SECTORS !
        MOV     COUNTER, #00h
        MOV     DPP0, #08h
        MOV     DATA_ADDR, #00h  ;FIRST LOCATION WITHIN THE TARGET SECTOR (DPP0)
        MOV     COMM_DATA, #0fah                ;COMMAND DATA

WAIT:   SUB     COUNTER, #01h                   ;POLL BUSY FLAG
        JMPA    cc_Z, TIMEOUT_ERROR
        MOV     [COMM_ADDR_AAAA], COMM_DATA     ;WRITE COMMAND
        MOV     STATUS, [DATA_ADDR]             ;READ STATUS
        JB      STATUS.0, WAIT
        CMPB    STATUS_L, #00h
        JMPR    cc_eq, RETURN_OK1

 FLAGERROR:
        MOV     ERROR_STATUS, #FLAG_ERROR
        RET
 TIMEOUT_ERROR:
        MOV     ERROR_STATUS, #TIME_OUT_ERROR
        RET
 RETURN_OK1:
        MOV     ERROR_STATUS, #NO_ERROR
        RET
;*******************************************************************************
MAIN        ENDP
MAIN_PRG    ENDS
            END
```

D Memory Map of Flash Programming Routines

This is the memory map of the microcontroller when executing the main program (Flash Programming Routines - Appendix C).

START	STOP	LENGTH	TYPE	ALIGN	TGR	GRP	COMB	CLASS	SECTION NAME
00FA00H	00FA1FH	000020H	DATA	AT..	---	---	PRIV	---	REG_BANK
00FA20H	00FA5FH	000040H	DATA	AT..	---	---	PRIV	---	DATA_BUFFER
00FA60H	00FBFFH	0001A0H	DATA	AT..	---	---	PRIV	---	STACK
00FC00H	00FDFDH	0001FEH	CODE	AT..	---	---	PRIV	---	MAIN_PRG
00FE00H	00FFFFH	000200H	DATA	AT..	---	---	PRIV	---	SFR’s