

## Emulation Parallel Mode Interface

### Using HMS99C58S FLASH MCU

#### INTRODUCTION

The Emulation Parallel (EP) Mode interface is for programming HMS99C58S FLASH MCU by its internal boot loader code not by general hardware programming logic implemented in the MCU device. Both the existing normal Parallel Mode interface and this Emulation Parallel Mode interface have the same pin I/F with host-connected programmer device that is comprised of fifteen addresses, eight data and eight controls, one reset, one oscillation clock, and two power lines. The merits of this Emulation Parallel mode in comparison to normal Parallel Mode are sought in the point of faster programming speed, that is, as the retrial count of FLASH programming or erasure increases, the work will complete comparatively faster. This is due to a few inner retrial action of the Emulation Parallel mode algorithm.

This application note is intended for design engineers who want to implement the Emulation Parallel Mode programming with boot ROM in MCU.

#### THE HARDWARE CONNECTION

The typical hardware connection is illustrated in Figure 1. The HMS99C58S has general parallel interface with host-connected programmer device. The fifteen address lines are connected to P1, P2.0~P2.5 and P3.4. The eight data lines are connected to P0, and this port is used as both input and output mode. The four mode command lines are connected to P2.7, P2.6, P3.7 and P3.6. The mode has total three commands that are Read, Program, Erase. The Start Pulse line is connected to P3.5, and this line informs the target MCU to start action corresponding to mode command by falling the port from normally high to low level. The EA, PROG and PSEN pins are always fixed to high or low level like Figure 1, that is necessary for inner bootloader code activation. The Power, Reset and Oscillation clock lines should be applied for proper MCU operation, and 4MHz oscillator is recommended.

#### THE SOFTWARE STRUCTURE

An example interface algorithm is listed in Appendix A. A flow diagram is given in Figure 2. The function of example source of HMS99C58S is to receive one of mode commands from host-connected programmer device and perform adequate action to the mode command.

**Note:** At Read and Program mode, the second address and following address need not to be transferred to target FLASH MCU because the addresses is auto-increased.

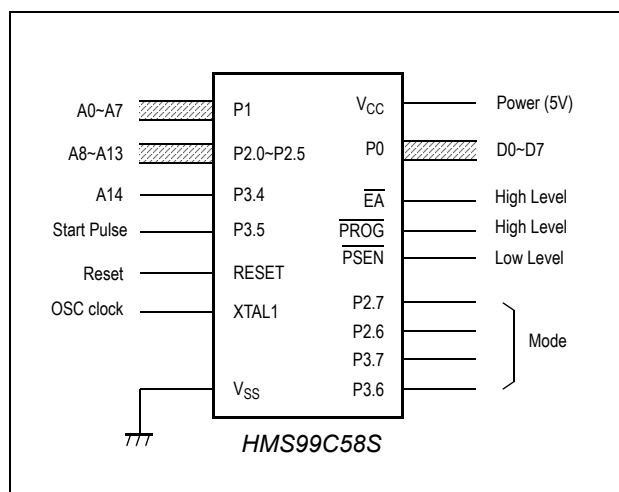


Figure 1. EP mode Interface

The user should take the following steps when implementing Emulation Parallel mode interface communication.

#### A) Bootloader code

1. Initialize the checking retrial counter to maximum value that is equivalent to 500us-limited time duration for checking Emulation Parallel Mode entry condition. The input Start Pulse line should be changed high to low level after reset operation, that is supposed to be happened within 500us from MCU initialization. The Start Pulse line should be maintained to high level for about 50us in order to be detected as the high state with time margin.
2. If the Start Pulse is detected to be fallen from high to low level, check whether the value of address lines are 5500h and that of data lines are aah. If passed successfully, this means the successful entry to Emulation Parallel Mode.
3. Once EP mode entry is succeeded, the Start Pulse line should be checked persistently for triggering transfer of mode command. When the Start Pulse is detected to be fallen from high to low level, read the mode command lines and take action adequate to the mode such as reading, programming, and erasure.

#### B) Host S/W for Programmer device

1. Make the Start Pulse line maintained to high level

for about 50us after MCU power-ON.

2. Set address lines to 5500H and data lines to AAH and make the Start Pulse line fallen high to low level to get Emulation Parallel Mode entry.
3. Set mode command lines to wanted mode value corresponding to reading, programming, and erasure, and address lines to wanted address values, and data lines to wanted data value if needed.
4. Make the Start Pulse line high to low level to trigger the target MCU to start appropriate inner command process operation of bootloader code.
5. If needed, get and check the return value from the P0 lines.

Code size: 1,155 bytes of program code

*Author: Vince Jeong  
MCU application team  
e-mail: byoungsu.jeong@hynix.com*

### COMMAND FORMAT

The mode commands are comprised of three command such as Read, Program, and Erase command. The respective mode command value is as follows. (order of P2.7 P2.6 P3.7 P3.6)

- Read Mode : 03H
- Program Mode : 0BH
- Erase Mode : 0EH

If the mode command is successfully transferred to target MCU, it will take different period of time to receive the operation result data of respective command.

### SUMMARY

The Emulation Parallel Mode interface is a simple, speedy and effective method of interfacing with host-connected programmer device in the HMS99C58S MCU application. The generic code in Appendix A makes it easy to incorporate in HMS99C58S series FLASH MCU application. Any of the HMS99C58S Series MCU with bootloader code can be applied to this example.

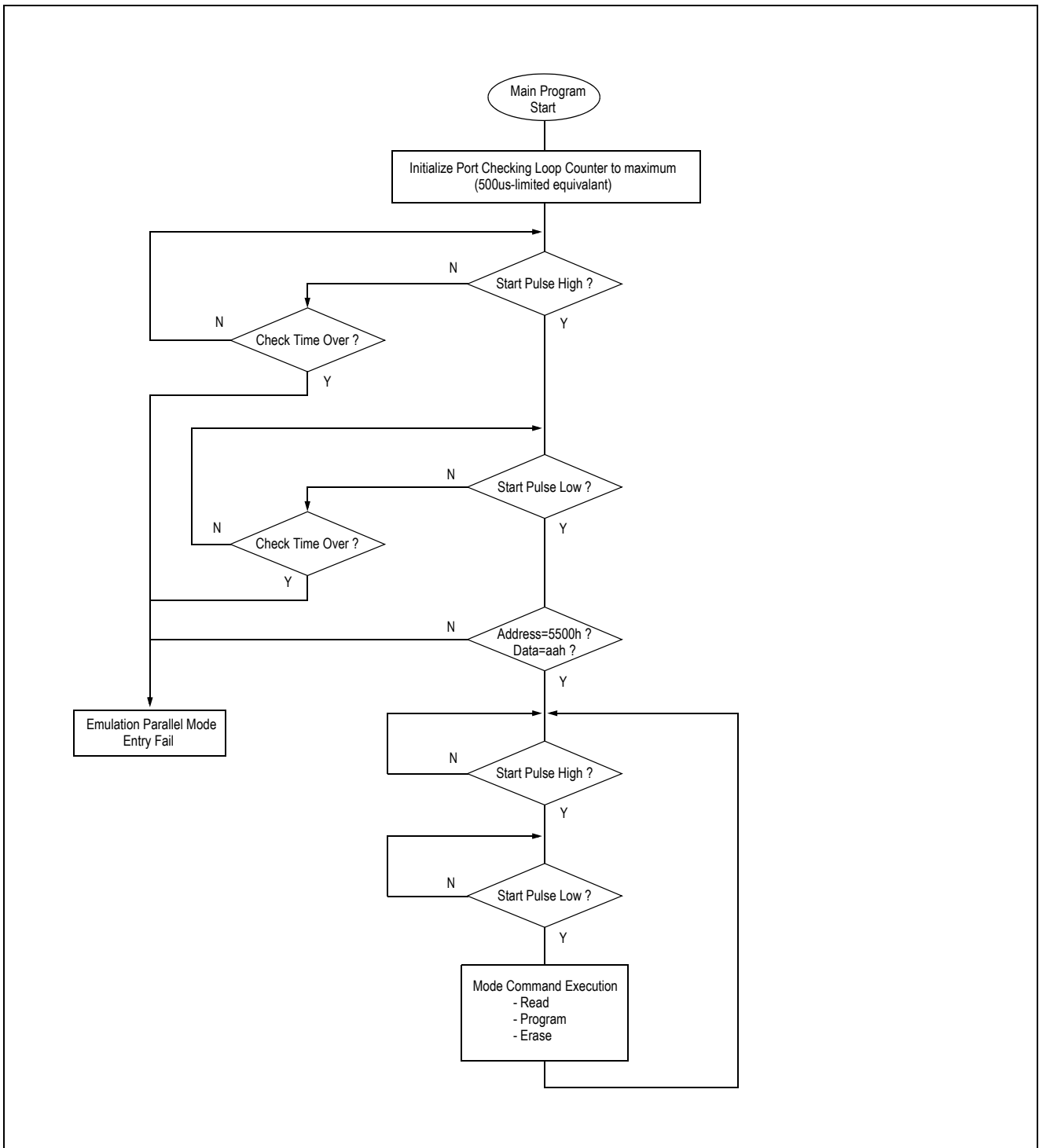


Figure 2. Flow Chart of Emulation Parallel Mode interface algorithm

## APPENDIX A

```

;-----
; PURPOSE:          This file contains the boot loader software
;
;-----
; NOTES:           This boot loader is based on the HMS99C58S
;                 It supports 32 Kbytes USER memory
;
$TITLE            (** Boot Loader **)

BOOTLOADER_MODULE SEGMENT CODE
BOOTLOADER_DATA   SEGMENT DATA

;_____ I N C L U D E S _____

#include (config.inc)
#include (register.inc)
#include (hex_cmd.inc)

;_____ D E F I N I T I O N _____

#define            FIRST_ADDRESS_MASK  01h

#define            READ_MODE            030h
#define            PROGRAM_MODE        0b0h
#define            ERASE_MODE          0e0h

#define            OK_MESSAGE           055h
#define            FAIL_MESSAGE         0aah
#define            NULL_BLOCK_MESSAGE  0aah

; *** DATA RAM INTERNAL ***

RSEG             BOOTLOADER_DATA
flash_flag:     DS             1
mode_value:     DS             1
write_buffer:   DS             1
stack:          DS             40h

PUBLIC          write_buffer, flash_flag

EXTRN          DATA (hex_address)
EXTRN          DATA (hex_buffer)
EXTRN          CODE (flash_programming_setup, flash_erase_setup)
EXTRN          CODE (flash_programming_start)
EXTRN          CODE (block0_erase, block1_erase, block2_erase, block3_erase)
EXTRN          CODE (block4_erase, block5_erase, block6_erase)
EXTRN          CODE (block0_prewrite, block1_prewrite, block2_prewrite, block3_prewrite)
EXTRN          CODE (block4_prewrite, block5_prewrite, block6_prewrite)

;_____ D E C L A R A T I O N _____

RSEG             BOOTLOADER_MODULE

;*****
; NAME:          boot_loader
;-----
; PURPOSE:       boot loader program
;*****
; NOTE:          located at address BOOT_ENTRY
;               Boot entry process:

```

```

;*****
boot_loader:
    mov     P3, #0fdh;for mode(P3.7, P3.6), Address high(P3.4), falling strobe(P3.5)
    mov     P2, #0ffh;for mode(P2.7, P2.6), ADDRESS HIGH(P2.5~P2.0)
    mov     P1, #0ffh;for ADDRESS LOW(P1.7~P1.0)
    mov     P0, #0ffh;data, input mode

    mov     R0, #41                ;500u/12u=41

initial_wait_rise:
    djnz   R0, polling_prog_port_rise;6u
    ajmp   EP_mode_entry_fail

polling_prog_port_rise:
    jnb    P3.5, initial_wait_rise;6u, rise check

initial_wait_fall:
    djnz   R0, polling_prog_port_fall;6u
    ajmp   EP_mode_entry_fail

polling_prog_port_fall:
    jnb    P3.5, initial_wait_fall;6u, falling check

;address low detect
    mov     A, P1                ;ADDR LOW, P1
    cjne   A, #00h, EP_mode_entry_fail

;address high detect
    mov     A, P2                ;ADDR HIGH, P3.4, P2.5~P2.0
    anl    A, #3fh
    mov     r5, A
    mov     A, P3
    rl     A
    rl     A
    anl    A, #040h
    orl    A, r5
    cjne   A, #055h, EP_mode_entry_fail

;data detect
    mov     A, P0
    cjne   A, #0aah, EP_mode_entry_fail

;EP entry success
    mov     R0,#IDATA_SIZE-1      ; initialise the IDATA space

idata_loop:
    mov     @R0,#00h
    djnz   R0,idata_loop
    mov     SP,#stack-1          ; initialize stack pointer

    mov     flash_flag, #0        ;init, important
    ajmp   start_emulation_parallel_mode;go to parallel pgm

;=====
EP_mode_entry_fail:
    sjmp   EP_mode_entry_fail

;*****
;start_parallel_write:
;*****
start_emulation_parallel_mode:
    mov     A, flash_flag
    orl    A, #FIRST_ADDRESS_MASK
    mov     flash_flag, A
    mov     mode_value, #0

    sjmp   polling_prog_port_rising

mission_complete:

```

```

        setb        P3.2                ;action start pulse

polling_prog_port_rising:
        mov        flash_flag, #00h    ;remove new_start_flag, important
        jnb       P3.5, polling_prog_port_rising;rising check
polling_prog_port_falling:
        jnb       P3.5, polling_prog_port_falling;falling check

        clr        P3.2                ;action start pulse

;=====
;mode catch
        mov        A, flash_flag
        anl        A, #FIRST_ADDRESS_MASK
        jz         mode_catch_omit
        mov        A, P3                ;for P3.7, P3.6
        rr         A
        rr         A
        anl        A, #30h
        mov        r5, A
        mov        A, P2                ;for P2.7, P2.6
        anl        A, #0c0h
        orl        A, r5                ;mode value : P2.7 P2.6 P3.7 P3.6 - - - -
        anl        A, #0f0h            ;mode value : P2.7 P2.6 P3.7 P3.6 - - - -
        mov        mode_value, A
mode_catch_omit:
        mov        A, mode_value

;=====
;read_mode_check:
        cjne       A, #READ_MODE, program_mode_check;if READ CODE DATA
        mov        A, flash_flag
        anl        A, #FIRST_ADDRESS_MASK
        jz         read_address_catch_omit
;address catch
        mov        hex_address+1, P1    ;ADDR LOW, P1
        mov        A, P2                ;ADDR HIGH, P3.4, P2.5~P2.0
        anl        A, #3fh
        mov        r5, A
        mov        A, P3
        rl         A
        rl         A
        anl        A, #040h
        orl        A, r5
        mov        hex_address, A      ;ADDR HIGH, P3.4, P2.5~P2.0

read_address_init:
        mov        DPH,hex_address      ; MSB of start address
        mov        DPL,hex_address+1    ; LSB of start address
read_address_catch_omit:
        ajmp       hex_read_parallel

;=====
program_mode_check:
        cjne       A, #PROGRAM_MODE, erase_mode_check;if READ CODE DATA
        mov        A, flash_flag
        anl        A, #FIRST_ADDRESS_MASK
        jz         pgm_address_catch_omit
;address catch
        mov        hex_address+1, P1    ;ADDR LOW, P1
        mov        A, P2                ;ADDR HIGH, P3.4, P2.5~P2.0

```

```

        anl        A, #3fh
        mov        r5, A
        mov        A, P3
        rl         A
        rl         A
        anl        A, #040h
        orl        A, r5
        mov        hex_address, A          ;ADDR HIGH, P3.4, P2.5~P2.0

program_address_init:
        mov        DPH,hex_address        ; DPTR = programming address
        mov        DPL,hex_address+1
        acall     flash_programming_setup;programming
pgm_address_catch_omit:
        mov        P0, #0ffh             ;important, write & verify info
        mov        hex_buffer, P0        ;DATA 8 bit
        ajmp      hex_program_data_parallel

;=====
erase_mode_check:
        cjne      A,#ERASE_MODE, mode_check_end;if erase check
        mov        P0, #0ffh             ;important, block info
        mov        hex_buffer, P0        ;DATA 8 bit
        ajmp      hex_erase_parallel

;=====
mode_check_end:
        ajmp      mission_complete

;*****
hex_erase_with_prewrite:
        acall     flash_programming_setup

hex_block_prewrite_0_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.0, hex_block_prewrite_1_parallel
        acall     block0_prewrite
hex_block_prewrite_1_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.1, hex_block_prewrite_2_parallel
        acall     block1_prewrite
hex_block_prewrite_2_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.2, hex_block_prewrite_3_parallel
        acall     block2_prewrite
hex_block_prewrite_3_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.3, hex_block_prewrite_4_parallel
        acall     block3_prewrite
hex_block_prewrite_4_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.4, hex_block_prewrite_5_parallel
        acall     block4_prewrite
hex_block_prewrite_5_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.5, hex_block_prewrite_6_parallel
        acall     block5_prewrite
hex_block_prewrite_6_parallel:
        mov        A,hex_buffer          ;block index for erase
        jnb       ACC.6, erase_after_prewrite_check
        acall     block6_prewrite

```

```

erase_after_prewrite_check:
    mov     A,hex_buffer           ;block index for erase
    anl    A, #07fh
    cjne   A, #00h, erase_after_prewrite
    sjmp   hex_block_prewrite_null

erase_after_prewrite:
    acall  flash_erase_setup
    ajmp   hex_erase_start        ;go to erase

hex_block_prewrite_null:
    mov    A, #NULL_BLOCK_MESSAGE
    ret

;*****
hex_erase_start:
hex_block_erase_0_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.0, hex_block_erase_1_parallel
    acall block0_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail
hex_block_erase_1_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.1, hex_block_erase_2_parallel
    acall block1_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail
hex_block_erase_2_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.2, hex_block_erase_3_parallel
    acall block2_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail
hex_block_erase_3_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.3, hex_block_erase_4_parallel
    acall block3_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail
hex_block_erase_4_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.4, hex_block_erase_5_parallel
    acall block4_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail
hex_block_erase_5_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.5, hex_block_erase_6_parallel
    acall block5_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail
hex_block_erase_6_parallel:
    mov    A,hex_buffer           ;block index for erase
    jnb   ACC.6, hex_erase_block_ok
    acall block6_erase
    cjne  A, #OK_MESSAGE, hex_erase_block_fail

hex_erase_block_ok:
    mov    A, #OK_MESSAGE
    sjmp   hex_erase_without_prewrite_end

hex_erase_block_fail:
    mov    A, #FAIL_MESSAGE
    sjmp   hex_erase_without_prewrite_end

hex_erase_without_prewrite_end:
    ret

```

```

;*****
hex_erase_parallel:
    mov     A,hex_buffer           ;block index for erase
;hex_erase_prewrite:
    acall   hex_erase_with_prewrite
    sjmp    hex_erase_parallel_end

hex_erase_parallel_end:
    mov     P0, A                 ;OK_MESSAGE or FAIL_MESSAGE or NULL_BLOCK_MESSAGE return
    ajmp    mission_complete

;*****
hex_read_parallel:
;parallel read routine
    mov     A,#00h
    movc    A,@A+DPTR
    mov     P0, A
    inc     DPTR                 ;for auto increase
    ajmp    mission_complete

;*****
hex_program_data_parallel:
    mov     A,hex_buffer
    acall   flash_programming_start
    cjne    A, #OK_MESSAGE, hex_program_fail
;hex_program_OK:
    inc     DPTR
    mov     A, hex_buffer
    CPL     A
    mov     P0, A                 ;#OK_MESSAGE
    ajmp    mission_complete
hex_program_fail:
    mov     P0, hex_buffer        ;#FAIL_MESSAGE
    ajmp    mission_complete

END

```