รายการอ้างอิง

1. Bill Moyer. Low-Power Design for Embedded Processors. Proceedings of the IEEE Vol. 89, No. 11 (November 2001): 1576-1587.
2. Jaewon Oh and Massoud Pedram. Gated Clock Routing for Low-Power Microprocessor Design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems Vol. 20, No. 6 (June 2001): 715-722
3. Michael Gschwind, Valentina Salapura, and Dietmar Maurer. FPGA Prototyping of a RISC Processor Core for Embedded Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems Vol. 9, No. 2 (April 2001): 241-250
4. A. Tansathit and E. Leelarasmee. A Design of a TV Microcontroller Chip with Built-in Thai-English On Screen Display. Engineering Transactions, a Research Publication of Mahanakorn University of Technology (May 1999):129-134.
5. Hing-Yip Tong. A single chip micro-computer for A/V monitor and TV Receiver. IEEE Transactions on Consumer Electronics Vol. No. (November 1990): 825-831
6. Alf-Egil Bogen, Vegard Wollan. AVR Enhanced RISC Microcontrollers. ATMEL Development Center, Trondheim, Norway
7. Microchip Technology Inc. PIC16C6X 8-Bit CMOS Microcontrollers. http://www.microchip.com
8. Atmel Corporation .AT90S1200 8-Bit AVR Microcontroller Data Sheet. http://www.atmel.com/
9. Philips Semiconductors. SAA55XX TV microcontrollers with Closed Captioning and On-Screen Display. Preliminary specification, February 2000.
10. Zilog. Z9036x 32 KWORD TELEVISION CONTROLLER WITH OSD. CAMPBELL, Cannada, 1999
11. Motorola. 68HC05CCV Technical Data. United State of America, 1996.
12. Xilinx Incorporation. The Programmable Logic Data Book 2000. United State of America, 2000.
13. Alcatel Microelectronics. C05M Design Rule Manual Supplement for Analogue Option. April 1999

14. อัษฎางค์ แทนสถิตย์. การพัฒนาทีวีไมโครคอนโทรลเลอร์ที่สามารถถอดรหัสคำบรรยายภาพไทย-อังกฤษแบบซ่อนได้. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2541.

15. Dave Patterson. CS152 Computer Architecture and Engineering Notes to Lecturers Using Notes. http://bwrc.eecs.berkeley.edu/classes/cs152/, 2002

16. Victor P. Nelson, H. Troy Nagle, Bill D. Carroll, and J. David Irwin. Digital Logic Circuit Analysis & Design. 1st ed. New Jersey: Prentrice-Hall, 1995

17. Robert J. Baron and Lee Higbie. Computer Architecture. 1st ed.: Addison-Wesley Publishing Company, 1994.

ภาคผนวก

ภาคผนวก ก
รายละเอียดโปรแกรมส่วนควบคุมหลัก

MAIN.ASM

```
------------------------------------------------------------------------
      CPU   "DEMO2.TBL"
      HOF   "INT8"
      WDLN  3
      INCL  "D:\PROJECT\DEV_SW\ASM\IR.ASM"
      INCL  "D:\PROJECT\DEV_SW\ASM\ISR.ASM"
;=*********************************************************
;This section defines alias name of memory.
;There are also define address of peripherals.
PWM_VT:           EQU   0000H
PWM_VOL:    EQU   0001H
PWM_BRI:    EQU   0002H
PWM_CON:    EQU   0003H
PWM_COL:    EQU   0004H
PWM_STL:    EQU   0005H
PB_CT:            EQU   001AH
PB_OUT:           EQU   001BH
OSD_R:            EQU   0017H
PA_CT:            EQU   0014H
PA_IN:            EQU   0016H
PA_OUT:           EQU   0015H
TC0_CT:           EQU   0018H
TC0_DT:           EQU   0019H
TC1_CT:           EQU   001CH
TC1_DT:           EQU   001DH
TC1_IC:           EQU   001FH
CH0:        EQU   8060H
M_FIELD:    EQU   8070H
M_CHANNEL:  EQU   8071H
M_DELAY:    EQU   8072H
M_RESPONSE: EQU   8073H
M_IR:       EQU   8074H
M_TUNE:           EQU   8075H
M_IR_BIT:   EQU   8076H
M_MODE:           EQU   8077H
M_PIC:            EQU   8078H
M_BAR:            EQU   8079H
M_SHOW:           EQU   807AH
M_CAPTION:  EQU   807BH
M_CC:       EQU   807CH
M_R2TMP:    EQU   807EH
M_R3TMP:    EQU   807DH
M_R1TMP:    EQU   807FH
M_VDO:            EQU   80E0H
M_VOL:            EQU   80E1H
M_BRI:            EQU   80E2H
M_CON:            EQU   80E3H
M_COL:            EQU   80E4H
M_TRACK:    EQU   80E5H
M_CC2:            EQU   80E6H
```

```
M_PARITY:    EQU   80E7H
M_CURSOR:    EQU   80E8H
M_R14TMP:    EQU   80EFH
;-------------------------------------------------------------

;=*************************************************************
;This section is the origin of main program.
;RESET ENTRY
      ORG   0000H
      JMP   MAIN
      JMP   MAIN
;INTERRUPT VECTOR ENTRY OF PORT A4
      ORG   0002H
      NOP
      JMP   PA4_INT
;INTERRUPT VECTOR ENTRY OF PORT A5
      ORG   0004H
      NOP
      JMP   PA5_INT
;-------------------------------------------------------------

;f: DELAY *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
; CYCLE = N*71 + 12
DELAY:
      ST    R1,M(M_DELAY)
      MOV   R1,R2
      ST    R1,M(M_R2TMP)
      MOV   R1,R3
      ST    R1,M(M_R3TMP)
      LD    R1,M(M_DELAY)
      MOV   R2,R1
DELAY_LP:
      MOV   R1,#000DH
      MOV   R3,R1
      MOV   R1,#0001H
      NOP
      NOP
DEL_LP:
      SUB   R3,R1       ;13*4-1=51
      JMP   DEL_EX,ZE
      JMP   DEL_LP
DEL_EX:
      SUB   R2,R1
      JMP   DELAY_EX,ZE
      JMP   DELAY_LP    ;60*N-1
DELAY_EX:
      LD    R1,M(M_R2TMP)
      MOV   R2,R1
      LD    R1,M(M_R3TMP)
      MOV   R3,R1
      RET               ;60*N-1+13+(3)
;-------------------------------------------------------------

;f: TURN_ON *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
TURN_ON:
      ;Disable the global interrupt flag
      XOR   R1,R1
```

```
      MOV   FLAG,R1
      MOV   FLAG,R1
      ST    R1,M(M_FIELD)      ;Reset Field

      MOV   R1,#0023H   ;MUTE on, BAND = UHF
      ST    R1,M(PB_OUT)
      MOV   R1,#0F96H   ;20 mSEC
      CALL  DELAY

      ;Get channel number
      LD    R1,M(M_CHANNEL)
      MOV   R14,R1
      MOV   R1,#8000H
      AND   R1,R14
      ;OR   R1,R1        ;DUMMY
      JMP   OPEN_TV,ZE  ;If M_CHANNEL != 8XXX Then OpenTV
      ;Else OpneAV
      MOV   R1,#0043H
      ST    R1,M(PB_OUT)
      MOV   R1,#90DEH   ;150 mSEC
      CALL  DELAY
      JMP   TURN_FIELD
OPEN_TV:
      MOV   R1,#8060H
      OR    R14,R1              ;Set R14 as index
      LD    R2,M(R14)   ;Get the channel's data

      ;MOV  R1,#0023H   ;MUTE on, BAND = UHF
      ;ST   R1,M(PB_OUT)
      ;MOV  R1,#0D96H   ;20 mSEC
      ;CALL DELAY

      MOV   R1,R2        ;R1 <= the channel's data
      AND   R1,#0003H   ;mask BAND
      OR    R1,#0000H   ;MUTE on, BAND = data
      ST    R1,M(PB_OUT)
      LSR   R1,R2        ;Set tuning voltage
      LSR   R1,R1
      ST    R1,M(PWM_VT)
      MOV   R1,#90DEH   ;150 mSEC
      CALL  DELAY

      LD    R1,M(PB_OUT)
      AND   R1,#0003H   ;TURN ON,TV,REMOVE MUTE
      ST    R1,M(PB_OUT)
TURN_FIELD:
      MOV   R1,#0A000H  ;20 mSEC
      CALL  DELAY
      MOV   R1,#0000H
      ST    R1,M(M_FIELD)
      ;Enable the global interrupt flag
      MOV   R1,#8000H
      MOV   FLAG,R1
      MOV   R1,#0010H   ;prepare PA4 for field determine
      ST    R1,M(PA_CT)
TURN_ONL:
      LD    R1,M(M_FIELD)
```

```
        MOV    R2,R1
        MOV    R1,#0FF0H
        AND    R2,R1
        ;OR    R2,R2        ;DUMMY
        JMP    TURN_ONL,ZE
        MOV    R1,#1400H
        ST     R1,M(M_RESPONSE)
        XOR    R1,R1
        ST     R1,M(M_BAR)
        ST     R1,M(80F0H)
        ST     R1,M(M_CAPTION)   ;OFF CAPTION
        ;A FIELD DETECTING IS RUNNING
        RET
;------------------------------------------------------------

;s: INIT_DATA *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
INIT_DATA:
  ;Hardware initializations
        MOV    R1,#0000H    ;Disable the global interrupt flag
        MOV    FLAG,R1
        ;PORT_B setting
        MOV    R1,#0000H
        ST     R1,M(PB_CT)
        MOV    R1,#0083H    ;Turn TV-Power off
        ST     R1,M(PB_OUT)
        ;Set 4 lower bit of PORT_A as input port,
        ;enable PA4 and PA5's INT on their negative edge
        MOV    R1,#0090H
        ST     R1,M(PA_CT)
        ;Set TC0 as counter, count the falling edge of PA5
        MOV    R1,#0006H
        ST     R1,M(TC0_CT)
        ;Set TC1 as input capture function
        ;It captures on negative edge of PA6
        ;and it's counter is driven by CLK/8.
        MOV    R1,#0883H
        ST     R1,M(TC1_CT)
        ;Delay
        MOV    R1,#0FFFEH
        CALL   DELAY

  ;Load TV-setting data from EEPROM.
;[=============/ * /================================ >
        MOV    R1,#0004H    ;VOLUME
        ST     R1,M(M_VOL)
        MOV    R1,#00D0H    ;BRIGHT CONTRAST COLOR
        ST     R1,M(M_BRI)
        ST     R1,M(M_CON)
        ST     R1,M(M_COL)
        MOV    R1,#CH0            ;SET INDEX TO CH0
        MOV    R14,R1
        MOV    R1,#8FFBH    ;ITV DATA
        MOV    R2,R1
        MOV    R1,#0010H
        MOV    R3,R1
        MOV    R1,#0001H
INIT_DATAL:
```

```
      ST    R2,M(R14+)
      SUB   R3,R1
      ;OR   R3,R3       ;DUMMY
      JMP   INIT_DATAL,NZ
      MOV   R1,#0000H   ;SET M_CHANNEL<= CH0
      ST    R1,M(M_CHANNEL)
      ;SET VOL,BRIGHT,CONTRAST,COLOR
      LD    R1,M(M_VOL)
      ST    R1,M(PWM_VOL)
      LD    R1,M(M_BRI)
      ST    R1,M(PWM_BRI)
      LD    R1,M(M_CON)
      ST    R1,M(PWM_CON)
      LD    R1,M(M_COL)
      ST    R1,M(PWM_COL)
; > ==========/ * /==============================]
  ;Flag initializations
      XOR   R1,R1
      ST    R1,M(M_IR)
      ST    R1,M(M_IR_BIT)
      ST    R1,M(M_RESPONSE)
      ST    R1,M(M_MODE)
      ST    R1,M(M_SHOW)
      ST    R1,M(M_CAPTION)
      MOV   R1,#0000H
      ST    R1,M(M_VDO)
      MOV   R1,#0001H
      ST    R1,M(M_PIC)
      MOV   R1,#0080H
      ST    R1,M(M_TRACK)
      ;//////DUMMY
      MOV   R1,#0C000H
      ST    R1,M(0C000H)
      ;//////DUMMY
      JMP   INIT_DATA_BACK
;-------------------------------------------------------------
;=************************************************************
;MAIN PROGRAM
MAIN:

      JMP   INIT_DATA   ;Initialization
INIT_DATA_BACK:
      MOV   R1,#2000H
      CALL  DELAY
      CALL  TURN_ON
MAIN_LP:
      ;CHECK      IR PRESSING
      LD    R1,M(TC1_CT)
      MOV   R2,R1
      MOV   R1,#0400H
      AND   R1,R2
      OR    R1,R1       ;DUMMY
      JMP   IR_BACK,ZE
      JMP   IR_ACTION
IR_BACK:
      JMP   MAIN_LP
;-------------------------------------------------------------
```

ภาคผนวก ข

รายละเอียดโปรแกรมส่วนถอดรหัสคำสั่งผู้ใช้

IR.ASM

```
;-------------------------------------------------------------------------
;PROGRAM'S RULES
;DON'T USE R5,R6,R7,R8,R9,R10,R11,R12,R13
;USE R2,R3,R4,R14,R15
;=*********************************************************
; SUB PROGRAMs
      ORG   0100H
;s: IR_ACTION-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
;This sub-program determine IR command.
;Private resouce: R3,R4,TC1,M_IR,M_IR_BIT
;Output           : M_IR_CMD
;             M_RESPONSE, Display memory
;Input            : -
;Sub-->           : IR_DECODE
IR_ACTION:
  ;this section processes IR pluse.
      LD    R1,M(TC1_IC)
      MOV   R3,R1
      MOV   R1,#0883H
      ST    R1,M(TC1_CT)       ;CLEAR FLAG
      ST    R1,M(TC1_DT)       ;CLEAR TC1
      ;TEST PULS WIDTH
      MOV   R1,#0040H    ;MINIMUM PW.
      SUB   R1,R3
      JMP   IR_OUT,NC    ;R3 < 0040 ==> SPICE
      MOV   R1,#0600H    ;MAXIMUM PW OF HEAD
      SUB   R1,R3
      JMP   IR_OUT,CY    ;R3 > 0600 ==> INTERVAL SYMBOL
      MOV   R1,#0400H    ;LOWER HEAD PW
      SUB   R1,R3
      JMP   HEAD_FOUND,CY      ;0400 < R3 < 0600 ==> HEAD
      MOV   R1,#01D0H    ;LOWER LONG PULS
      SUB   R1,R3
      JMP   LONG_PULS,CY       ;01D0 < R3 < 0400 ==> LONG
SHORT_PULS:
      LD    R1,M(M_IR)   ;GET AND TEST STATE
      MOV   R1,R1
      ;OR    R1,R1        ;DUMMY
      JMP   IR_OUT,ZE    ;Don't process any things
      LD    R1,M(M_IR_BIT)    ;PROCESS
      LSR   R1,R1
      ST    R1,M(M_IR_BIT)     ;ADD 0 TO IR_BIT
      MOV   R1,#0001H
      ADD   R4,R1
      MOV   R1,#0010H
      XOR   R1,R4
      ;OR    R1,R1        ;DUMMY
      JMP   IR_COMPLETE,ZE     ;IF R4 == 16 THEN COMPLETE
      JMP   IR_BACK            ;Not complete
LONG_PULS:
```

```
        LD      R1,M(M_IR)   ;GET AND TEST STATE
        MOV     R1,R1
        ;OR     R1,R1        ;DUMMY
        JMP     IR_OUT,ZE    ;Don't process any things
        LD      R1,M(M_IR_BIT)     ;PROCESS
        LSR     R1,R1
        MOV     R3,R1
        MOV     R1,#8000H
        OR      R1,R3
        ST      R1,M(M_IR_BIT)     ;ADD 1 TO IR_BIT
        MOV     R1,#0001H
        ADD     R4,R1
        MOV     R1,#0010H
        XOR     R1,R4
        ;OR     R1,R1        ;DUMMY
        JMP     IR_COMPLETE,ZE     ;IF R4 == 16 THEN COMPLETE
        JMP     IR_BACK            ;Not complete
HEAD_FOUND:
        MOV     R1,#0001H
        ST      R1,M(M_IR)   ;STATE = 1
        XOR     R4,R4        ;CLEAR PULS COUNTER
        XOR     R1,R1
        ST      R1,M(M_IR_BIT)     ;CLEAR BIT
IR_OUT:
        JMP     IR_BACK

  ;IR's code, check ID
IR_COMPLETE:
        XOR     R1,R1
        ST      R1,M(M_IR)   ;CLEAR STATE
        MOV     R4,R1        ;CLEAR COMNTER
        LD      R1,M(M_IR_BIT)     ;CHECK 1111 1DDD DD11 1111
        MOV     R3,R1
        MOV     R1,#0F83FH
        AND     R3,R1        ;ID CODE 0101 0XXX XX00 1010
        MOV     R1,#500AH
        XOR     R1,R3
        ;OR     R1,R1        ;DUMMY
        JMP     IR_DECODE,ZE       ;If ID is OK then IR_DECODE
        ;Else IR_ERROR
IR_ERROR:
        XOR     R1,R1
        ST      R1,M(M_IR_BIT)
        MOV     R4,R1
        JMP     IR_BACK

IR_DECODE_BACK:
        MOV     R1,#0FFFEH
        MOV     R3,R1
        MOV     R1,#0001H
IR_LOOP:
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
```

```
      SUB   R3,R1
      OR    R3,R3
      OR    R3,R3
      JMP   IR_LOOP,NZ
      XOR   R1,R1
      ST    R1,M(M_IR_BIT)
      MOV   R4,R1
      JMP   IR_BACK

;-----------------------------------------------------

;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
IR_DECODE:
      LD    R1,M(M_IR_BIT)    ;Decode, get 5-bit code
      MOV   R3,R1
      MOV   R1,#07C0H
      AND   R1,R3
      LSR   R1,R1
      LSR   R1,R1
      LSR   R1,R1
      LSR   R1,R1
      LSR   R1,R1
      LSR   R1,R1
      MOV   R3,R1

      MOV   R1,#80F0H
      MOV   R14,R1


      ;test mode, OPERATE or TUNE
      LD    R1,M(M_MODE)
      MOV   R4,R1
      MOV   R1,#0F00H
      AND   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   IR_DEC0,ZE
      JMP   IR_TUNE      ;If M_MODE == 0F00 Then TUNE
IR_DEC0:
      ;Else OPERATE
            ;POWER code
      MOV   R1,#000AH
      XOR   R1,R3
      ;OR   R1,R1 ;DUMMY
      JMP   IR_DEC1,NZ
      JMP   IR_DO_POWER
IR_DEC1:

      ;Test mode, POWER is ON
      LD    R1,M(M_MODE)
      MOV   R4,R1
      MOV   R1,#0F000H
      AND   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   IR_DEC1A,ZE
      JMP   IR_DECODE_BACK    ;If M_MODE == F000 Then BACK
IR_DEC1A:
            ;MUTE code
```

```
        MOV     R1,#0000H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC2,NZ
        JMP     IR_DO_MUTE
IR_DEC2:
                ;DISPLAY code
        MOV     R1,#000FH
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC3,NZ
        JMP     IR_DO_DISPLAY
IR_DEC3:
                ;VOL+ code
        MOV     R1,#0006H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC4,NZ
        JMP     IR_DO_VOLUMEI
IR_DEC4:
                ;VOL- code
        MOV     R1,#0007H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC5,NZ
        JMP     IR_DO_VOLUMED
IR_DEC5:
                ;PIC code
        MOV     R1,#0001H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC6,NZ
        JMP     IR_DO_PIC
IR_DEC6:
                ;+ code
        MOV     R1,#0002H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC7,NZ
        JMP     IR_DO_PLUS
IR_DEC7:
                ;- code
        MOV     R1,#0003H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC8,NZ
        JMP     IR_DO_MINUS
IR_DEC8:
                ;- AV
        MOV     R1,#000DH
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_DEC9,NZ
        JMP     IR_DO_AV
IR_DEC9:
                ;SYS code
        MOV     R1,#0005H
```

```
        XOR    R1,R3
        ;OR    R1,R1 ;DUMMY
        JMP    IR_DEC13,NZ
        JMP    IR_DO_SYS
IR_DEC13:
  ;Test M_CHANNEL is TV:000X or AV:800X
        LD     R1,M(M_CHANNEL)
        MOV    R4,R1
        MOV    R1,#8000H
        AND    R1,R4
        ;OR    R1,R1 ;DUMMY
        JMP    IR_TV,ZE
        ;//// AV mode don't accept below key
        JMP    IR_DECODE_BACK
        ;////
IR_TV:
               ;CH+ code
        MOV    R1,#0008H
        XOR    R1,R3
        ;OR    R1,R1 ;DUMMY
        JMP    IR_DEC10,NZ
        JMP    IR_DO_CHUP
IR_DEC10:
               ;CH- code
        MOV    R1,#0009H
        XOR    R1,R3
        ;OR    R1,R1 ;DUMMY
        JMP    IR_DEC11,NZ
        JMP    IR_DO_CHDOWN
IR_DEC11:

        MOV    R1,#000FH
        SUB    R1,R3
        JMP    IR_N_OVER,NC
        MOV    R1,#0019H
        SUB    R1,R3
        JMP    IR_N_OVER,CY
        JMP    IR_DO_N
IR_N_OVER:
               ;CLK code
        MOV    R1,#000CH
        XOR    R1,R3
        ;OR    R1,R1 ;DUMMY
        JMP    IR_DEC12,NZ
        JMP    IR_DO_CLK
IR_DEC12:


        JMP    IR_DECODE_BACK
;-----------------------------------------------------


;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
;=====================================================
IR_DO_POWER:
        LD     R1,M(M_MODE)
        MOV    R2,R1
```

```
        MOV    R1,#0F000H
        AND    R1,R2
        ;OR    R1,R1 ;DUMMY
        JMP    POWER_ON,NZ
POWER_OFF:
        XOR    R1,R1
        MOV    FLAG,R1
        XOR    R1,R1
        MOV    FLAG,R1
        LD     R1,M(PB_OUT)
        MOV    R2,R1
        MOV    R1,#0080H
        OR     R1,R2
        ST     R1,M(PB_OUT)
        MOV    R1,#0F000H
        ST     R1,M(M_MODE)

        MOV    R1,#0FFF0H
        CALL   DELAY

        JMP    IR_DECODE_BACK

POWER_ON:
        MOV    R1,#0000H
        ST     R1,M(M_MODE)
        CALL   TURN_ON

        JMP    IR_DECODE_BACK
;=================================================
IR_DO_N:
        ;//Action of N key
        MOV    R1,#0019H
        XOR    R1,R3
        ;OR    R1,R1 ;DUMMY
        JMP    IR_N_X1,ZE
        MOV    R1,#0001H
        ADD    R3,R1
        MOV    R1,#000FH
        AND    R1,R3
        JMP    IR_N_TURN_ON
IR_N_X1:
        MOV    R1,#0000H
IR_N_TURN_ON:
        ST     R1,M(M_CHANNEL)
        CALL   TURN_ON
        ;// END Action of N key
        JMP    IR_DECODE_BACK
;=================================================
IR_DO_MUTE:
        MOV    R1,#0A9A0H  ;ASCII M
        ST     R1,M(R14+)
        MOV    R1,#0AAA0H  ;ASCII U
        ST     R1,M(R14+)
        MOV    R1,#0AA80H  ;ASCII T
        ST     R1,M(R14+)
        MOV    R1,#0A8A0H  ;ASCII E
        ST     R1,M(R14+)
```

```
        MOV    R1,#00000H  ;\0
        ST     R1,M(R14+)
        ST     R1,M(M_BAR)
        ;ACTION
        MOV    R1,#0000H
        ST     R1,M(PWM_VOL)
        ;END ACTION
        JMP    ADD_RESPONSE
;=================================================
IR_DO_VOLUMEI:
        MOV    R1,#M_VOL
        CALL   INC_256
        JMP    IR_DO_VOLUME
IR_DO_VOLUMED:
        MOV    R1,#M_VOL
        CALL   DEC_256
IR_DO_VOLUME:
        MOV    R1,#0AAC0H  ;ASCII V
        ST     R1,M(R14+)
        MOV    R1,#0A9E0H  ;ASCII O
        ST     R1,M(R14+)
        MOV    R1,#0A980H  ;ASCII L
        ST     R1,M(R14+)
        MOV    R1,#0AAA0H  ;ASCII U
        ST     R1,M(R14+)
        MOV    R1,#0A9A0H  ;ASCII M
        ST     R1,M(R14+)
        MOV    R1,#0A8A0H  ;ASCII E
        ST     R1,M(R14+)
        MOV    R1,#00000H  ;\0
        ST     R1,M(R14+)
        JMP    ADD_RESPONSE
;=================================================
IR_DO_PIC:
        LD     R1,M(M_PIC)
        MOV    R4,R1
        LD     R1,M(M_RESPONSE)
        MOV    R3,R1
        MOV    R1,#0000H
        XOR    R1,R3
        ;OR    R1,R1 ;DUMMY
        JMP    SKIP_CH_PIC,ZE    ;If No-response
        ;Else change PIC
        MOV    R1,#0001H
        ADD    R4,R1
        MOV    R1,#0003H
        AND    R4,R1
        ;OR    R4,R4 ;DUMMY
        JMP    SKIP_CH_PIC,NZ
        ;Else adjust 00 to 1
        MOV    R1,#0001H
        OR     R4,R1
SKIP_CH_PIC:
        MOV    R1,R4        ;Store PIC
        ST     R1,M(M_PIC)
        MOV    R1,#0F00H    ;NONE code to write out
        MOV    R3,R1
```

```
IR_DO_PIC2I:
      MOV   R1,#0001H
      XOR   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   IR_DO_BRIGHT,ZE
      MOV   R1,#0002H
      XOR   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   IR_DO_CONTRAST,ZE
IR_DO_COLOR:
      MOV   R1,#0A860H  ;ASCII C
      ST    R1,M(R14+)
      MOV   R1,#0A9E0H  ;ASCII O
      ST    R1,M(R14+)
      MOV   R1,#0A980H  ;ASCII L
      ST    R1,M(R14+)
      MOV   R1,#0A9E0H  ;ASCII O
      ST    R1,M(R14+)
      MOV   R1,#0AA40H  ;ASCII R
      ST    R1,M(R14+)
      MOV   R1,#0000H   ;\0
      ST    R1,M(R14+)
      LD    R1,M(M_COL)
      ST    R1,M(M_BAR)
      JMP   ADD_RESPONSE
IR_DO_CONTRAST:
      MOV   R1,#0A860H  ;ASCII C
      ST    R1,M(R14+)
      MOV   R1,#0A9E0H  ;ASCII O
      ST    R1,M(R14+)
      MOV   R1,#0A9C0H  ;ASCII N
      ST    R1,M(R14+)
      MOV   R1,#0AA80H  ;ASCII T
      ST    R1,M(R14+)
      MOV   R1,#0AA40H  ;ASCII R
      ST    R1,M(R14+)
      MOV   R1,#0A820H  ;ASCII A
      ST    R1,M(R14+)
      MOV   R1,#0AA60H  ;ASCII S
      ST    R1,M(R14+)
      MOV   R1,#0AA80H  ;ASCII T
      ST    R1,M(R14+)
      MOV   R1,#0000H   ;\0
      ST    R1,M(R14+)
      LD    R1,M(M_CON)
      ST    R1,M(M_BAR)
      JMP   ADD_RESPONSE
IR_DO_BRIGHT:
      MOV   R1,#0A840H  ;ASCII B
      ST    R1,M(R14+)
      MOV   R1,#0AA40H  ;ASCII R
      ST    R1,M(R14+)
      MOV   R1,#0A920H  ;ASCII I
      ST    R1,M(R14+)
      MOV   R1,#0A8E0H  ;ASCII G
      ST    R1,M(R14+)
      MOV   R1,#0A900H  ;ASCII H
```

```
        ST      R1,M(R14+)
        MOV     R1,#0AA80H  ;ASCII T
        ST      R1,M(R14+)
        MOV     R1,#0000H   ;\0
        ST      R1,M(R14+)
        LD      R1,M(M_BRI)
        ST      R1,M(M_BAR)
        JMP     ADD_RESPONSE
;============================
IR_DO_PLUS:
        MOV     R1,#M_VOL
        MOV     R2,R1
        LD      R1,M(M_PIC)
        ADD     R1,R2
        CALL    INC_256
        LD      R1,M(M_PIC)
        MOV     R4,R1
        JMP     IR_DO_PIC2I
;============================
IR_DO_MINUS:
        MOV     R1,#M_VOL
        MOV     R2,R1
        LD      R1,M(M_PIC)
        ADD     R1,R2
        CALL    DEC_256
        LD      R1,M(M_PIC)
        MOV     R4,R1
        JMP     IR_DO_PIC2I
;================================================
IR_DO_DISPLAY:
        LD      R1,M(M_SHOW)
        MOV     R2,R1
        MOV     R1,#0F00H
        XOR     R1,R2
        ST      R1,M(M_SHOW)
        JMP     IR_DECODE_BACK
;================================================
IR_DO_AV:
        LD      R1,M(M_CHANNEL)
        MOV     R2,R1
        MOV     R1,#08000H
        XOR     R1,R2
        ST      R1,M(M_CHANNEL)
        CALL    TURN_ON
        JMP     IR_DECODE_BACK
;================================================
IR_DO_CHUP:
        LD      R1,M(M_CHANNEL)
        MOV     R2,R1
        MOV     R1,#0001H
        ADD     R2,R1
        MOV     R1,#0009H
        SUB     R1,R2
        ;OR     R1,R1 ;DUMMY
        JMP     CHUP_SKIP,NC
        XOR     R2,R2
CHUP_SKIP:
```

```
        MOV     R1,R2
        ST      R1,M(M_CHANNEL)
        CALL    TURN_ON
        JMP     IR_DECODE_BACK
;=====================================================
IR_DO_CHDOWN:
        LD      R1,M(M_CHANNEL)
        MOV     R2,R1
        MOV     R1,#0001H
        SUB     R2,R1
        JMP     CHDOWN_SKIP,NC
        MOV     R1,#0009H
        MOV     R2,R1
CHDOWN_SKIP:
        MOV     R1,R2
        ST      R1,M(M_CHANNEL)
        CALL    TURN_ON
        JMP     IR_DECODE_BACK
;=====================================================
IR_DO_CLK:
        MOV     R1,#0F00H
        ST      R1,M(M_SHOW)
        ST      R1,M(M_MODE)

        LD      R1,M(M_CHANNEL)
        MOV     R2,R1
        MOV     R1,#8060H
        OR      R1,R2
        MOV     R2,R14              ;SAVE R14
        MOV     R14,R1
        LD      R1,M(R14)
        MOV     R14,R2              ;RESTORE R14
        ST      R1,M(M_TUNE)

        MOV     R2,R1
        MOV     R1,#0003H
        AND     R2,R1

        MOV     R1,#0002H
        XOR     R1,R2
        ;OR     R1,R1 ;DUMMY
        JMP     BAND1,ZE
        MOV     R1,#0001H
        XOR     R1,R2
        ;OR     R1,R1 ;DUMMY
        JMP     BAND2,ZE
BAND3:
        MOV     R1,#0AAA0H  ;ASCII U
        ST      R1,M(R14+)
        MOV     R1,#0A900H  ;ASCII H
        ST      R1,M(R14+)
        MOV     R1,#0A8C0H  ;ASCII F
        ST      R1,M(R14+)
        MOV     R1,#0000H   ;\0
        ST      R1,M(R14+)
        MOV     R1,#0013H
        ST      R1,M(PB_OUT)
```

```
        JMP     IR_DECODE_BACK
BAND2:
        MOV     R1,#0AAC0H   ;ASCII V
        ST      R1,M(R14+)
        MOV     R1,#0A900H   ;ASCII H
        ST      R1,M(R14+)
        MOV     R1,#0A8C0H   ;ASCII F
        ST      R1,M(R14+)
        MOV     R1,#0A5A0H   ;ASCII -
        ST      R1,M(R14+)
        MOV     R1,#0A900H   ;ASCII H
        ST      R1,M(R14+)
        MOV     R1,#0000H    ;\0
        ST      R1,M(R14+)
        MOV     R1,#0011H
        ST      R1,M(PB_OUT)
        JMP     IR_DECODE_BACK
BAND1:
        MOV     R1,#0AAC0H   ;ASCII V
        ST      R1,M(R14+)
        MOV     R1,#0A900H   ;ASCII H
        ST      R1,M(R14+)
        MOV     R1,#0A8C0H   ;ASCII F
        ST      R1,M(R14+)
        MOV     R1,#0A5A0H   ;ASCII -
        ST      R1,M(R14+)
        MOV     R1,#0A980H   ;ASCII L
        ST      R1,M(R14+)
        MOV     R1,#0000H    ;\0
        ST      R1,M(R14+)
        MOV     R1,#0012H
        ST      R1,M(PB_OUT)
        JMP     IR_DECODE_BACK
;=================================================
IR_DO_SYS:

        MOV     R4,R14              ;SAVE R14
        MOV     R1,#8000H
        MOV     R14,R1
        MOV     R1,#0060H
        MOV     R2,R1
        MOV     R1,#0001H
        MOV     R3,R1
        MOV     R1,#0004H
SYS1:
        ST      R1,M(R14+)
        SUB     R2,R3
        JMP     SYS1,NZ

        MOV     R1,#8080H
        MOV     R14,R1
        MOV     R1,#0060H
        MOV     R2,R1
        MOV     R1,#0004H
SYS2:
        ST      R1,M(R14+)
        SUB     R2,R3
```

```
        JMP     SYS2,NZ

        MOV     R14,R4              ;RESTORE R14

        MOV     R1,#0000H
        ST      R1,M(M_CURSOR)

        XOR     R1,R1
        ST      R1,M(M_BAR)
        MOV     R1,#0A860H  ;ASCII C
        ST      R1,M(R14+)
        MOV     R1,#0A820H  ;ASCII A
        ST      R1,M(R14+)
        MOV     R1,#0AA00H  ;ASCII P
        ST      R1,M(R14+)
        MOV     R1,#0A400H  ;ASCII ' '
        ST      R1,M(R14+)

        LD      R1,M(M_CAPTION)
        MOV     R2,R1
        MOV     R1,#0F000H
        AND     R1,R2
        JMP     TURN_CC0,ZE ;OFF -> CC0
        MOV     R1,#0F00H
        AND     R1,R2
        JMP     TURN_CC1,ZE ;CC0 -> CC1
TURN_CC_OFF:
        MOV     R1,#0A9E0H  ;ASCII O
        ST      R1,M(R14+)
        MOV     R1,#0A8C0H  ;ASCII F
        ST      R1,M(R14+)
        MOV     R1,#0A8C0H  ;ASCII F
        ST      R1,M(R14+)
        MOV     R1,#0000H   ;\0
        ST      R1,M(R14+)

        XOR     R1,R1
        ST      R1,M(M_CAPTION)
        JMP     ADD_RESPONSE

TURN_CC0:
        MOV     R1,#0A620H  ;ASCII 1
        ST      R1,M(R14+)
        MOV     R1,#0000H   ;\0
        ST      R1,M(R14+)

        MOV     R1,#0F000H
        ST      R1,M(M_CAPTION)
        JMP     ADD_RESPONSE

TURN_CC1:
        MOV     R1,#0A640H  ;ASCII 2
        ST      R1,M(R14+)
        MOV     R1,#0000H   ;\0
        ST      R1,M(R14+)

        MOV     R1,#0FF00H
```

```
        ST      R1,M(M_CAPTION)
        JMP     ADD_RESPONSE

        JMP     IR_DECODE_BACK
;=================================================
ADD_RESPONSE:
        MOV     R1,#1400H
        ST      R1,M(M_RESPONSE)
        JMP     IR_DECODE_BACK
;=================================================

;-------------------------------------------------

;=*************************************************************
IR_TUNE:
                ;RETUN code
        MOV     R1,#001FH
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_TUNE1,NZ
        JMP     TUNE_RET
IR_TUNE1:
                ;PP code
        MOV     R1,#001DH
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_TUNE2,NZ
        JMP     TUNE_PP
IR_TUNE2:
                ;-- code
        MOV     R1,#001AH
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_TUNE3,NZ
        JMP     TUNE_MEM
IR_TUNE3:
                ;+ code
        MOV     R1,#0002H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_TUNE4,NZ
        JMP     TUNE_UP
IR_TUNE4:
                ;- code
        MOV     R1,#0003H
        XOR     R1,R3
        ;OR     R1,R1 ;DUMMY
        JMP     IR_TUNE5,NZ
        JMP     TUNE_DOWN
IR_TUNE5:

        JMP     IR_DECODE_BACK
;=================================================
TUNE_RET:
        MOV     R1,#0000H
        ST      R1,M(M_SHOW)
        ST      R1,M(M_MODE)
```

```
        LD      R1,M(PB_OUT)
        MOV     R3,R1
        MOV     R1,#00EFH
        AND     R1,R3
        ST      R1,M(PB_OUT)

        LD      R1,M(M_CHANNEL)
        MOV     R3,R1
        MOV     R1,#8060H
        OR      R1,R3
        MOV     R14,R1
        LD      R1,M(R14)
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)


        JMP     IR_DECODE_BACK
;=================================================
TUNE_PP:
        LD      R1,M(M_MODE)
        MOV     R3,R1
        MOV     R1,#00F0H
        XOR     R1,R3
        ST      R1,M(M_MODE)
        JMP     IR_DECODE_BACK
;=================================================
TUNE_MEM:
        MOV     R1,#0000H
        ST      R1,M(M_MODE)
        ST      R1,M(M_SHOW)
        ST      R1,M(M_BAR)
        MOV     R1,#0AA60H   ;ASCII S
        ST      R1,M(R14+)
        MOV     R1,#0A820H   ;ASCII A
        ST      R1,M(R14+)
        MOV     R1,#0AAC0H   ;ASCII V
        ST      R1,M(R14+)
        MOV     R1,#0A8A0H   ;ASCII E
        ST      R1,M(R14+)
        MOV     R1,#0000H    ;\0
        ST      R1,M(R14+)

        LD      R1,M(PB_OUT)
        MOV     R3,R1
        MOV     R1,#00EFH
        AND     R1,R3
        ST      R1,M(PB_OUT)

        LD      R1,M(M_CHANNEL)
        MOV     R3,R1
        MOV     R1,#8060H
        OR      R1,R3
        MOV     R14,R1
        LD      R1,M(M_TUNE)
        ST      R1,M(R14)
```

```
      JMP   ADD_RESPONSE
;=================================================
TUNE_UP:
      LD    R1,M(M_TUNE)
      MOV   R3,R1
      LD    R1,M(M_MODE)
      MOV   R4,R1
      MOV   R1,#00F0H
      AND   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   TUP_X1,ZE
      MOV   R1,#0004H
      JMP   TUP_X2
TUP_X1:
      MOV   R1,#0200H
TUP_X2:
      ADD   R3,R1
      JMP   TUP_OK,NC
      ;VHFL(2)=>VHFH(1)=>UHF(3)
      MOV   R1,#0003H
      AND   R1,R3
      MOV   R4,R1
      ;TEST
      MOV   R1,#0003H
      XOR   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   TUP_X3,NZ
      ;(3) ==> (2)
      MOV   R1,#0FFFEH
      AND   R3,R1
      MOV   R1,R3
      ST    R1,M(M_TUNE)
      LSR   R1,R1
      LSR   R1,R1
      ST    R1,M(PWM_VT)
      JMP   BAND1
TUP_X3:
      MOV   R1,#0002H
      XOR   R1,R4
      ;OR   R1,R1 ;DUMMY
      JMP   TUP_X4,NZ
      ;(2) ==> (1)
      MOV   R1,#0003H
      XOR   R3,R1
      MOV   R1,R3
      ST    R1,M(M_TUNE)
      LSR   R1,R1
      LSR   R1,R1
      ST    R1,M(PWM_VT)
      JMP   BAND2
TUP_X4:
      ;(1) ==> (3)
      MOV   R1,#0002H
      OR    R3,R1
      MOV   R1,R3
      ST    R1,M(M_TUNE)
```

```
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)
        JMP     BAND3

TUP_OK:
        MOV     R1,R3
        ST      R1,M(M_TUNE)
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)
        JMP     IR_DECODE_BACK
;=====================================================
TUNE_DOWN:
        LD      R1,M(M_TUNE)
        MOV     R3,R1
        LD      R1,M(M_MODE)
        MOV     R4,R1
        MOV     R1,#00F0H
        AND     R1,R4
        ;OR     R1,R1 ;DUMMY
        JMP     TDN_X1,ZE
        MOV     R1,#0004H
        JMP     TDN_X2
TDN_X1:
        MOV     R1,#0200H
TDN_X2:
        SUB     R3,R1
        JMP     TDN_OK,NC
        ;VHFL(2)<=VHFH(1)<=UHF(3)
        MOV     R1,#0003H
        AND     R1,R3
        MOV     R4,R1
        ;TEST
        MOV     R1,#0003H
        XOR     R1,R4
        ;OR     R1,R1 ;DUMMY
        JMP     TDN_X3,NZ
        ;(3) ==> (1)
        MOV     R1,#0FFFDH
        AND     R3,R1
        MOV     R1,R3
        ST      R1,M(M_TUNE)
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)
        JMP     BAND2
TDN_X3:
        MOV     R1,#0002H
        XOR     R1,R4
        ;OR     R1,R1 ;DUMMY
        JMP     TDN_X4,NZ
        ;(2) ==> (3)
        MOV     R1,#0001H
        OR      R3,R1
        MOV     R1,R3
        ST      R1,M(M_TUNE)
```

```
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)
        JMP     BAND3
TDN_X4:
        ;(1) ==> (2)
        MOV     R1,#0003H
        XOR     R3,R1
        MOV     R1,R3
        ST      R1,M(M_TUNE)
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)
        JMP     BAND1
TDN_OK:
        MOV     R1,R3
        ST      R1,M(M_TUNE)
        LSR     R1,R1
        LSR     R1,R1
        ST      R1,M(PWM_VT)
        JMP     IR_DECODE_BACK
;=====================================================
;-------------------------------------------------------------


;=*********************************************************
;=====================================================
;///
INC_256:
        MOV     R15,R1
        LD      R1,M(R15)
        MOV     R2,R1
        MOV     R1,#0002H
        ADD     R2,R1
        MOV     R1,#0FF00H
        AND     R1,R2
        ;OR     R1,R1 ;DUMMY
        JMP     SKIP_INC,ZE
        MOV     R1,#00FFH
        MOV     R2,R1
SKIP_INC:
        MOV     R1,R2
        ST      R1,M(R15)
        ST      R1,M(M_BAR)
        MOV     R1,#0007H
        AND     R15,R1
        ST      R2,M(R15)
        RET
DEC_256:
        MOV     R15,R1
        LD      R1,M(R15)
        MOV     R2,R1
        MOV     R1,#0002H
        SUB     R2,R1
        MOV     R1,#0FFFCH
        AND     R1,R2
        ;OR     R1,R1 ;DUMMY
```

```
      JMP   SKIP_DEC,NZ
      MOV   R1,#0004H
      MOV   R2,R1
SKIP_DEC:
      MOV   R1,R2
      ST    R1,M(R15)
      ST    R1,M(M_BAR)
      MOV   R1,#0007H
      AND   R15,R1
      ST    R2,M(R15)
      RET
;=================================================
```

ภาคผนวก ค

รายละเอียดโปรแกรมส่วนชุดคำสั่งประจำสำหรับการขัดจังหวะ

ISR.ASM

```
;-------------------------------------------------------------------------
;PROGRAM'S RULES
;DON'T USE R2,R3,R4,R15
;SAVE R14

;=*********************************************************
;This section contains interrupt-service routines (ISRs).
      ORG   0400H
;f: PA4_INT -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
;This ISR determine a current field is Odd or Even.
;Private resouce: R12,TC0_DT
;Output           : M_FIELD
;Input            : M_FIELD
;M_FILED's values
;     0000 = reset,count Hsync pulses that occurs between
;            two Vsync pluses.
;     000F = counting Hsync is complete then determine this
;            field is Odd-field or Even-field.
;     00FF = This field is Odd-field.
;     FF00 = This field is Even-field.
PA4_INT:

      ST    R1,M(M_R1TMP)      ;save R1
      LD    R1,M(M_FIELD)      ;get M_FIELD
      MOV   R12,R1
      MOV   R1,#000FH
      XOR   R1,R12
      JMP   PA4_DETERMINE,ZE;If M_FIELD == 000F then Determine
      ST    R1,M(TC0_DT)       ;Clear TC0
      MOV   R1,#0000H
      XOR   R1,R12
      ;OR   R1,R1        ;DUMMY
      JMP   PA4_SWAP,NZ ;If M_FIELD != 0000 then SwapField
  ;0000:Reset state: It must disable PA5_INT (Hsync_INT)
      MOV   R1,#000FH   ;Next state = 000F
      ST    R1,M(M_FIELD)
      MOV   R1,#0010H   ;Clear PA4_INF, Disable PA5_INT
      ST    R1,M(PA_CT)
      LD    R1,M(M_R1TMP)      ;Restore R1
      RETI
  ;000F:Counting complete:
PA4_DETERMINE:
      LD    R1,M(TC0_DT)
      MOV   R12,R1
      ST    R1,M(TC0_DT)       ;Clear TC0
      MOV   R1,#0001H
      AND   R1,R12
      ;OR   R1,R1        ;DUMMY
      JMP   PA4_SKIP,ZE ;If number of Hsync is even number Then ODD-
FIELD
```

```
      MOV   R1,#0FF00H  ;Else EVEN-FIELD
      ST    R1,M(M_FIELD)
      JMP   PA4_EXIT
PA4_SKIP:
      MOV   R1,#00FFH   ;ODD-FIELD
      ST    R1,M(M_FIELD)
      JMP   PA4_EXIT
  ;00FF|FF00:Change field:
PA4_SWAP:
      SWP   R12,R12
      MOV   R1,R12
      ST    R1,M(M_FIELD)
      MOV   R1,#0050H
      ST    R1,M(M_TRACK)
  ;Field detecting is complete so PA5_INT is also enabled.
PA4_EXIT:
      MOV   R1,#0090H   ;Clear INFs and enable INTs.
      ST    R1,M(PA_CT)
      LD    R1,M(M_R1TMP)     ;Restore R1
      RETI
;f-------------------------------------------------


;f: PA5_INT *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
;There are three major functions of PA5_INT.
;  First,in the begining of Hsync pulse, it have to track
;  Sync-Tip-Level voltage and provide Data-Slicing-Level(DSL)
;  voltage in line 10 to 18 of even field.
;  Second,it extracts closed-caption data inserted in line 18 even
field.
;  Third, it perform OSD function.
;Private resouce: R5,R6,R7,R8,R9,R13,R14
;Input(flag)       : M_FIELD,TC0_DT
;             M_TUNE,M_CHANNEL,M_RESPONSE[*],M_DIS_CH
;             Display memory at 80F0-80FA
;             M_CAPTION, CC display memory
;[*] will be changed by this ISR.

PA5_INT:
      ST    R1,M(M_R1TMP)     ;SAVE R1
      MOV   R1,R14
      ST    R1,M(M_R14TMP)    ;SAVE R14
      LD    R1,M(TC0_DT)      ;LINE TO R5
      MOV   R5,R1

      MOV   R1,#000DH
      SUB   R1,R5
      JMP   PA5_XX0,CY  ;
      JMP   TRACK       ;JUMP TO TRACK IF R5 =< 15
PA5_XX0:
;     MOV   R1,#000FH
;     XOR   R1,R5
;     JMP   PA5_XX1,NZ
;     JMP   DATA_SLICE17
;PA5_XX1:
      MOV   R1,#0010H
      XOR   R1,R5
```

```
        JMP     PA5_XX2,NZ
        JMP     DATA_SLICE18
PA5_XX2:
        MOV     R1,#0011H
        XOR     R1,R5
        JMP     PA5_XX3,NZ
        JMP     DATA_SLICE19
PA5_XX3:

        MOV     R1,#0017H
        XOR     R1,R5
        JMP     PA5_XX5,NZ
        JMP     TRACK_END
PA5_XX5:

        MOV     R1,#0018H
        XOR     R1,R5
        JMP     PA5_XX4,NZ
        JMP     DECODE_CC
PA5_XX4:

        MOV     R1,#0040H
        SUB     R1,R5
        JMP     PA5_X1,CY
        JMP     PA5_OUT
PA5_X1:
        MOV     R1,#0060H
        SUB     R1,R5
        JMP     PA5_X2,CY
        JMP     CHANNEL_LINE
PA5_X2:
        MOV     R1,#0070H
        SUB     R1,R5
        JMP     PA5_X3,CY
        JMP     CONTROL_LINE
PA5_X3:
        MOV     R1,#0080H
        SUB     R1,R5
        JMP     PA5_X4,CY
        ;TEST MODE
        LD      R1,M(M_MODE)
        MOV     R6,R1
        MOV     R1,#0F00H
        AND     R1,R6
        ;OR     R1,R1 ;DUMMY
        JMP     PA5_X31,ZE
        JMP     BAR_TUNE
PA5_X31:
        JMP     BAR_LINE
PA5_X4:
        MOV     R1,#0090H
        SUB     R1,R5
        JMP     PA5_X5,CY
        JMP     FINE_LINE
PA5_X5:
        MOV     R1,#00C0H
        SUB     R1,R5
```

```
        JMP     PA5_X6,CY
        JMP     PA5_OUT
PA5_X6:
        MOV     R1,#00F0H
        SUB     R1,R5
        JMP     PA5_X7,CY
        JMP     OSD
PA5_X7:
        JMP     PA5_OUT

;=================================================
CONTROL_LINE:
        LD      R1,M(M_MODE)
        OR      R1,R1
        ;OR     R1,R1           ;DUMMY
        JMP     CONTROL_X2,NZ
        LD      R1,M(M_RESPONSE)
        OR      R1,R1
        ;OR     R1,R1           ;DUMMY
        JMP     CONTROL_X1,NZ
        JMP     PA5_OUT                 ;OUT OF RESPONSE
CONTROL_X1:
        MOV     R7,R1           ;DECREASING TIMER
        MOV     R1,#0001H
        SUB     R7,R1
        MOV     R1,R7
        ST      R1,M(M_RESPONSE)
CONTROL_X2:
        MOV     R1,#0002H
        CALL    DELAY

        MOV     R7,R5           ;SHIFT LEFT OF LINE
        ADD     R7,R5
        MOV     R1,#80F0H       ;SET R14 TO MEMORY DISPLAY
        MOV     R14,R1
        LD      R1,M(M_FIELD)       ;GET FIELD
        MOV     R13,R1
        MOV     R1,#0001H
        AND     R13,R1
        OR      R7,R13
        MOV     R1,#001FH
        AND     R7,R1           ;R7 CONTAIN 5 LOWER BIT
        MOV     R1,#000AH       ;START HSSR
        ST      R1,M(PB_CT)
RESPONSE_LP:
        LD      R1,M(R14+)
        MOV     R1,R1
        ;OR     R1,R1           ;DUMMY
        NOP
        JMP     RESPONSE_EXT,ZE
        OR      R1,R7
        MOV     R13,R1
        LD      R1,M(R13,000H)
        ST      R1,M(OSD_R)
        NOP
        NOP
        NOP
```

```
      NOP
      NOP
      NOP
      JMP   RESPONSE_LP
RESPONSE_EXT:
      MOV   R1,#0002H
      ST    R1,M(PB_CT)
      JMP   PA5_OUT
;=================================================
BAR_LINE:
      LD    R1,M(M_RESPONSE)
      OR    R1,R1
      ;OR   R1,R1        ;DUMMY
      JMP   BAR_X1,NZ
      JMP   PA5_OUT           ;OUT OF RESPONSE
BAR_X1:
      MOV   R7,R1        ;DECREASING TIMER
      MOV   R1,#0001H
      SUB   R7,R1
      MOV   R1,R7
      ST    R1,M(M_RESPONSE)

      MOV   R1,#0002H
      CALL  DELAY

      LD    R1,M(M_BAR)
      MOV   R6,R1
      MOV   R1,#00FFH
      AND   R6,R1
      MOV   R1,#0008H
      MOV   R7,R1
      MOV   R1,#000AH   ;START HSSR
      ST    R1,M(PB_CT)
BAR_LP:
      SUB   R6,R7
      JMP   BAR_EXT,CY
      NOP
      NOP
      NOP
      NOP
      NOP
      MOV   R1,#0FFFFH
      ST    R1,M(OSD_R)
      NOP
      NOP
      NOP
      NOP
      NOP
      JMP   BAR_LP
BAR_EXT:
      MOV   R1,#0002H
      ST    R1,M(PB_CT)
      JMP   PA5_OUT

;=================================================
CHANNEL_LINE:
      LD    R1,M(M_SHOW)
```

```
        OR      R1,R1
        ;OR     R1,R1           ;DUMMY
        JMP     DIS_CH,NZ
        LD      R1,M(M_RESPONSE)
        OR      R1,R1
        ;OR     R1,R1           ;DUMMY
        JMP     CHANNEL_X1,NZ
        JMP     PA5_OUT                 ;OUT OF RESPONSE
CHANNEL_X1:
        MOV     R7,R1           ;DECREASING TIMER
        MOV     R1,#0001H
        SUB     R7,R1
        MOV     R1,R7
        ST      R1,M(M_RESPONSE)
DIS_CH:
        MOV     R1,#0008H
        CALL    DELAY

        MOV     R7,R5           ;SHIFT LEFT OF LINE
        MOV     R1,#001FH
        AND     R7,R1           ;R7 CONTAIN 5 LOWER BIT

        LD      R1,M(M_CHANNEL)
        MOV     R6,R1
        MOV     R1,#8000H
        AND     R1,R6
        ;OR     R1,R1 ;DUMMY
        JMP     DIS_AV,NZ

        MOV     R1,#000AH       ;START HSSR
        ST      R1,M(PB_CT)

        LD      R1,M(M_CHANNEL)
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        OR      R7,R1
        MOV     R1,#0A600H
        OR      R7,R1
        MOV     R13,R7
        LD      R1,M(R13,00H)
        ST      R1,M(OSD_R)
        NOP
        NOP
        NOP
        NOP

        MOV     R1,#0002H
        ST      R1,M(PB_CT)
        JMP     PA5_OUT

DIS_AV:
        MOV     R1,#000AH       ;START HSSR
        ST      R1,M(PB_CT)
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOV     R1,#0A820H
        OR      R1,R7
        MOV     R13,R1
        LD      R1,M(R13,00H)
        ST      R1,M(OSD_R)
        NOP
        NOP
        NOP
        NOP

        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOV     R1,#0AAC0H
        OR      R1,R7
        MOV     R13,R1
        LD      R1,M(R13,00H)
        ST      R1,M(OSD_R)
        NOP
        NOP
        NOP
        NOP

        MOV     R1,#0002H
        ST      R1,M(PB_CT)
        JMP     PA5_OUT
;=================================================
BAR_TUNE:
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        MOV     R1,#0002H
        CALL    DELAY

        LD      R1,M(M_TUNE)
        LSR     R1,R1
        LSR     R1,R1

        LSR     R1,R1
        LSR     R1,R1
        LSR     R1,R1
```

```
        LSR     R1,R1
        LSR     R1,R1
        LSR     R1,R1
        LSR     R1,R1

        MOV     R6,R1
        MOV     R1,#0004H
        MOV     R7,R1

        MOV     R1,#000AH    ;START HSSR
        ST      R1,M(PB_CT)
TBAR_LP:
        SUB     R6,R7
        JMP     TBAR_EXT,CY
        NOP
        NOP
        NOP
        NOP
        NOP
        MOV     R1,#0FFFFH
        ST      R1,M(OSD_R)
        NOP
        NOP
        NOP
        NOP
        NOP
        JMP     TBAR_LP
TBAR_EXT:
        MOV     R1,#0002H
        ST      R1,M(PB_CT)
        JMP     PA5_OUT
;=================================================
FINE_LINE:
        LD      R1,M(M_MODE)
        MOV     R6,R1
        MOV     R1,#00F0H
        AND     R1,R6
        ;OR     R1,R1        ;DUMMY
        JMP     FINE_X1,NZ
        JMP     PA5_OUT              ;OUT OF RESPONSE
FINE_X1:

        MOV     R1,#0002H
        CALL    DELAY

        MOV     R1,#000EH    ;START HSSR
        ST      R1,M(PB_CT)

        MOV     R7,R5        ;SHIFT LEFT OF LINE
        ADD     R7,R5
        LD      R1,M(M_FIELD)       ;GET FIELD
        MOV     R13,R1
        MOV     R1,#0001H
        AND     R13,R1
        OR      R7,R13
        MOV     R1,#001FH
        AND     R7,R1        ;R7 CONTAIN 5 LOWER BIT
```

```
      MOV   R1,#0A8C0H
      OR    R1,R7
      MOV   R13,R1
      LD    R1,M(R13,00H)
      ST    R1,M(OSD_R)
      NOP
      NOP

      LD    R1,M(M_TUNE)
      LSR   R1,R1
      LSR   R6,R1
      MOV   R1,#007FH
      AND   R6,R1
      MOV   R1,#0004H
      MOV   R7,R1
FINE_LP:
      SUB   R6,R7
      JMP   FINE_EXT,CY
      NOP
      NOP
      NOP
      MOV   R1,#0FFFFH
      ST    R1,M(OSD_R)
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      JMP   FINE_LP
FINE_EXT:
      MOV   R1,#0006H
      ST    R1,M(PB_CT)
      JMP   PA5_OUT
;=====================================================
OSD:
      LD    R1,M(M_CAPTION)
      MOV   R6,R1
      MOV   R1,#0F000H
      AND   R1,R6
      ;OR   R1,R1          ;DUMMY
      JMP   OSD_TT,NZ
      JMP   PA5_OUT
OSD_TT:
      MOV   R1,#0001H
      CALL  DELAY
      LD    R1,M(M_VDO)
      MOV   R14,R1
      MOV   R1,#8000H
      OR    R14,R1
      LD    R1,M(M_CAPTION)
      MOV   R8,R1
      MOV   R1,#0080H
      AND   R1,R8
      OR    R14,R1
   ;=========================ABOVE,MID,BELOW=======
```

```
        MOV    R8,R5        ;SHIFT LEFT OF LINE
        ADD    R8,R5
        LD     R1,M(M_FIELD)     ;GET FIELD
        MOV    R7,R1
        MOV    R1,#0001H
        AND    R7,R1
        OR     R8,R7
        MOV    R1,#001FH
        AND    R8,R1        ;R8 CONTAINS 5-LOWER BIT
        ;SELECT ABOVE,MID,BELOW
        MOV    R1,#000DH    ;AMB
        SUB    R1,R8        ;AMB
        JMP    DO_ABOVE,NC  ;AMB A
        MOV    R1,#0019H    ;MB
        SUB    R1,R8        ;MB
        JMP    DO_MID,NC    ;MB M

DO_BELOW:

        MOV    R1,#0A000H
        OR     R8,R1        ;R8 CONTAINS 5-LOWER BIT AND 3-UPPER BIT
        MOV    R1,#0020H
        MOV    R6,R1        ;R6 = 32
        MOV    R1,#0001H
        MOV    R7,R1        ;R7 = 1
        MOV    R1,#10A0H
        MOV    R10,R1            ;R10 = 10A0
        MOV    R1,#0C00H
        MOV    R11,R1            ;R11 = 0C00
        MOV    R1,#10E0H
        MOV    R5,R1        ;R5 = 10E0
        NOP
        NOP
        MOV    R1,#000AH
        NOP


        ST     R1,M(PB_CT)
        MOV    R1,#0E01FH
        MOV    R9,R1        ;R9 = E01F
BELOW:
        LD     R1,M(R14)
        AND    R1,R10
        JMP    USE_MID,ZE
        LD     R1,M(R14)
        AND    R1,R11
        LSR    R13,R1
        LSR    R13,R13
        LD     R1,M(R14+)
        AND    R1,R5
        OR     R13,R1
        OR     R13,R8
BACK:
        LD     R1,M(R13,00H)
        ST     R1,M(OSD_R)
        SUB    R6,R7
```

```
      JMP   BELOW,NZ
      JMP   BELOW_EXIT
USE_MID:
      LD    R1,M(R14+)
      AND   R1,R9
      SWP   R1,R1
      OR    R1,R8
      MOV   R13,R1
      JMP   BACK

BELOW_EXIT:
      LD    R1,M(PB_CT)
      MOV   R6,R1
      MOV   R1,#0007H
      AND   R1,R6
      ST    R1,M(PB_CT)
      JMP   OSD_LINE

DO_ABOVE:
      NOP               ;ADJUST TO MID
      NOP               ;ADJUST TO MID
      NOP               ;ADJUST TO MID

      MOV   R1,#0A000H
      OR    R8,R1       ;R8 CONTAINS 5-LOWER BIT AND 3-UPPER BIT
      MOV   R1,#0020H
      MOV   R6,R1       ;R6 = 32
      MOV   R1,#0001H
      MOV   R7,R1       ;R7 = 1
      MOV   R1,#0E01FH
      MOV   R10,R1            ;R10 = E01F
      MOV   R1,#13E0H
      MOV   R11,R1            ;R11 = 13E0
      NOP
      NOP
      NOP
      NOP

      MOV   R1,#000AH
      ST    R1,M(PB_CT)
      NOP
      NOP
ABOVE:
      LD    R1,M(R14)
      AND   R1,R10
      SWP   R1,R1
      OR    R1,R8
      MOV   R13,R1
      LD    R9,M(R13,00H)
      LD    R1,M(R14+)
      AND   R1,R11
      OR    R1,R8
      MOV   R13,R1
      LD    R1,M(R13,00H)
      OR    R1,R9
      ST    R1,M(OSD_R)
      SUB   R6,R7
```

```
        JMP     ABOVE,NZ

        LD      R1,M(PB_CT)
        MOV     R6,R1
        MOV     R1,#0007H
        AND     R1,R6
        ST      R1,M(PB_CT)
        JMP     OSD_LINE

DO_MID:
    ;LINE
        MOV     R1,#0A000H
        OR      R8,R1           ;R8 CONTAINS 5-LOWER BIT AND 3-UPPER BIT
        MOV     R1,#0020H
        MOV     R6,R1           ;R6 = 32
        MOV     R1,#0001H
        MOV     R7,R1           ;R7 = 1
        MOV     R1,#0E01FH
        MOV     R10,R1                  ;R10 = E01F
        MOV     R1,#001EH
        MOV     R11,R1                  ;R11 = 001E
        NOP
        NOP
        NOP
        NOP
        MOV     R1,#000AH

        ST      R1,M(PB_CT)
        MOV     R1,#01E0H
        MOV     R5,R1           ;R5 = 01E0
MID:
        NOP
        NOP
        NOP
        NOP
        LD      R1,M(R14+)
        AND     R1,R10
        SWP     R1,R1
        OR      R1,R8
        MOV     R13,R1
        NOP
        NOP
        LD      R1,M(R13,00H)
        ST      R1,M(OSD_R)
        SUB     R6,R7
        JMP     MID,NZ
MID_EXIT:
        LD      R1,M(PB_CT)
        MOV     R6,R1
        MOV     R1,#0007H
        AND     R1,R6
        ST      R1,M(PB_CT)
        JMP     OSD_LINE
    ;=========================ABOVE,MID,BELOW=======

OSD_LINE:
        LD      R1,M(M_VDO)
```

```
      MOV   R14,R1
      MOV   R1,#001EH   ;LINE 30TH OR 31ST
      MOV   R7,R1
      AND   R7,R8
      XOR   R7,R1
      JMP   OSD_UP,ZE
      JMP   OSD_SKIP
OSD_UP:
      MOV   R1,#0020H
      ADD   R14,R1
      MOV   R1,#0060H
      XOR   R1,R14
      JMP   OSD_SKIP,NZ
      MOV   R1,#0000H
      AND   R14,R1
OSD_SKIP:
      MOV   R1,R14
      ST    R1,M(M_VDO)
      JMP   PA5_OUT
;=====================================================
DATA_SLICE17:
      NOP
      NOP
      NOP
      NOP
DATA_SLICE18:
      NOP
      NOP
      NOP
      NOP
DATA_SLICE19:
      MOV   R1,#0000H
      ST    R1,M(M_VDO)
      NOP
      NOP
      MOV   R1,#0028H
      MOV   R5,R1
      MOV   R1,#0001H
PRE_DELAY:
      SUB   R5,R1
      OR    R5,R5
      JMP   PRE_DELAY,NZ
      MOV   R1,#0400H
      MOV   R11,R1
      XOR   R6,R6
      XOR   R8,R8
      MOV   R1,#0001H
      MOV   R9,R1
      NOP
      ;The first sample occurs at 192nd instruction.
RUN_IN:
      LD    R1,M(PA_IN) ;1
      AND   R1,R11            ;2
      JMP   CLK_HIGH,NZ ;3->4
      ADD   R6,R9       ;4 COUNT
      MOV   R1,#0003H   ;5
      XOR   R1,R6       ;6
```

```
        JMP     HEAD_LOW,ZE ;7->8
        NOP                 ;8
        NOP                 ;9
        NOP                 ;10
        JMP     RUN_IN              ;11 12
CLK_HIGH:
        XOR     R6,R6       ;5
        ADD     R8,R9       ;6
        MOV     R1,#0008H   ;7
        XOR     R1,R8       ;9
        JMP     NO_DATA,ZE  ;10
        JMP     RUN_IN              ;11

HEAD_LOW:
        NOP                 ;9
        NOP                 ;10
        NOP                 ;11
        XOR     R8,R8       ;12
HEAD_EDGE:
        LD      R1,M(PA_IN) ;1
        AND     R1,R11              ;2
        JMP     PRE_GET,NZ  ;3->4
        MOV     R1,#0001H   ;4
        ADD     R6,R1       ;5
        MOV     R1,#0007H   ;6
        XOR     R1,R6       ;7
        JMP     NO_DATA,ZE  ;8->9
        NOP                 ;9
        NOP                 ;10
        JMP     HEAD_EDGE   ;11 12

PRE_GET:
        MOV     R1,#0010H   ;1
        MOV     R6,R1       ;2
        MOV     R1,#0001H   ;3
        MOV     R7,R1       ;4
        MOV     R1,#0005H   ;5
PRE_GET_LP:
        SUB     R1,R7
        OR      R1,R1
        JMP     PRE_GET_LP,NZ       ;+19 = 24
        XOR     R8,R8       ;CLEAR DATA
        XOR     R5,R5       ;CLEAR PARITY


GET_DATA:
        LD      R1,M(PA_IN)
        AND     R1,R11
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        LSR     R8,R8
        OR      R8,R1
        JMP     COUNT_PAR,NEG
        NOP
```

```
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
GET_DATAX:
      NOP
      SUB   R6,R7
      OR    R6,R6
      JMP   GET_DATA,NZ
      JMP   SAVE_DATA
COUNT_PAR:
      MOV   R1,#0008H
      SUB   R1,R6
      JMP   CHAR1_PAR,CY
      MOV   R1,#0100H
      ADD   R5,R1
      NOP
      JMP   GET_DATAX
CHAR1_PAR:
      MOV   R1,#0001H
      ADD   R5,R1
      JMP   GET_DATAX
SAVE_DATA:
      MOV   R1,#7F7FH
      AND   R1,R8
      ST    R1,M(M_CC)
      MOV   R1,#0101H
      AND   R1,R5
      MOV   R1,R5
      ST    R1,M(M_PARITY)
NO_DATA:
      JMP   PA5_OUT

;=================================================
DECODE_CC:
      LD    R1,M(M_PARITY)
      MOV   R5,R1
      MOV   R1,#0FFFFH
      AND   R1,R5
      JMP   CC_XX1,NZ
      JMP   NO_DATA_CC
CC_XX1:
      ;TEST CTRL WD
      LD    R1,M(M_CC)
      SWP   R5,R1
      MOV   R1,#0F000H
      AND   R5,R1
      MOV   R1,#1000H
      XOR   R1,R5
      JMP   WRITE_TXT,NZ

      ;CTRL TEST REPEAT HERE
      LD    R1,M(M_CC)
```

```
        MOV    R5,R1
        LD     R1,M(M_CC2)
        XOR    R1,R5
        JMP    DECODE_READY,ZE    ;REPEAT OUT

        ;GET CC CHANNEL TO R6
        LD     R1,M(M_CAPTION)
        MOV    R6,R1
        MOV    R1,#0800H
        AND    R6,R1          ;R6 HOLD CHANNEL
        LD     R1,M(M_CC)
        SWP    R5,R1          ;R5 HOLD CC_CTRL

        ;TSET STATE
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#000FH
        AND    R7,R1
        JMP    CC_RESET,ZE
        ;NOT RESET HERE

        ;TEST THE OTHER CC
        MOV    R1,#0800H
        AND    R1,R5
        XOR    R1,R6
        JMP    CC_OTHER,NZ
        ;MATCH CC HEAR

        ;SET F STATE
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#000FH
        OR     R1,R7
        ST     R1,M(M_CAPTION)


        ;TEST IS EDM
        MOV    R1,#142CH
        OR     R1,R6
        XOR    R1,R5
        JMP    CC_NX3,NZ
        ;EDM
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#0080H    ;GET DISPLAY PAGE
        AND    R7,R1
        MOV    R1,#8000H
        OR     R1,R7
CLEAR:
        MOV    R14,R1
        MOV    R1,#0060H
        MOV    R7,R1
        MOV    R1,#0001H
        MOV    R8,R1
        MOV    R1,#0004H
EDM:
        ST     R1,M(R14+)
```

```
        SUB    R7,R8
        JMP    EDM,NZ
        JMP    DECODE_READY
CC_NX3:
        ;TEST IS EOC
        MOV    R1,#142FH
        OR     R1,R6
        XOR    R1,R5
        JMP    CC_NX1,NZ
        ;FILP PAGE
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#00F0H
        XOR    R1,R7
        ST     R1,M(M_CAPTION)
        JMP    DECODE_READY
CC_NX1:
        ;TEST IS ENM
        MOV    R1,#142EH
        OR     R1,R6
        XOR    R1,R5
        JMP    CC_NX2,NZ
        ;ENM
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#0080H    ;GET DISPLAY PAGE
        AND    R7,R1
        XOR    R7,R1
        MOV    R1,#8000H
        OR     R1,R7
        JMP    CLEAR
CC_NX2:
        ;TEST IS LINE13
        MOV    R1,#1360H
        OR     R1,R6
        XOR    R1,R5
        JMP    CC_NX4,NZ
        ;MOV CURSOR
        MOV    R1,#0000H
        ST     R1,M(M_CURSOR)
        JMP    DECODE_READY
CC_NX4:
        ;TEST IS LINE14
        MOV    R1,#1440H
        OR     R1,R6
        XOR    R1,R5
        JMP    CC_NX5,NZ
        ;MOV CURSOR
        MOV    R1,#0020H
        ST     R1,M(M_CURSOR)
        JMP    DECODE_READY
CC_NX5:
        ;TEST IS LINE15
        MOV    R1,#1460H
        OR     R1,R6
        XOR    R1,R5
        JMP    CC_NX6,NZ
```

```
        ;MOV CURSOR
        MOV    R1,#0040H
        ST     R1,M(M_CURSOR)
        JMP    DECODE_READY
CC_NX6:
        JMP    DECODE_READY

CC_RESET:
        ;TEST IS RCL
        MOV    R1,#1420H
        OR     R1,R6
        XOR    R1,R5
        JMP    DECODE_READY,NZ
        ;SET F STATE
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#000FH
        OR     R1,R7
        ST     R1,M(M_CAPTION)
        XOR    R1,R1
        ST     R1,M(M_CURSOR)
        JMP    DECODE_READY

CC_OTHER:
        ;SET C STATE
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#0FFFCH
        AND    R1,R7
        ST     R1,M(M_CAPTION)
        JMP    DECODE_READY

DECODE_READY:
        LD     R1,M(M_CC)
        NOP
        ST     R1,M(M_CC2)
        XOR    R1,R1
        ST     R1,M(M_PARITY)
NO_DATA_CC:
        JMP    PA5_OUT
;============================
WRITE_TXT:
        ;TSET STATE
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#0003H
        AND    R7,R1
        JMP    DECODE_READY,ZE
        LD     R1,M(M_CC)
        SWP    R5,R1
        ;WRITE TEXT
        LD     R1,M(M_CURSOR)
        MOV    R13,R1
        LD     R1,M(M_CAPTION)
        MOV    R7,R1
        MOV    R1,#0080H
        AND    R7,R1
```

```
        XOR     R7,R1
        MOV     R1,#8000H
        OR      R13,R1
        OR      R13,R7
        ;CHAR1
        MOV     R1,#0FF00H
        AND     R1,R5
        SWP     R7,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        ADD     R1,R1
        LSR     R7,R7
        LSR     R7,R7
        LSR     R7,R7
        OR      R1,R7
        ST      R1,M(R13,00H)
        ;CHAR2
        MOV     R1,#00FFH
        AND     R1,R5
        SWP     R7,R1
        ADD     R7,R7
        ADD     R7,R7
        ADD     R7,R7
        ADD     R7,R7
        ADD     R7,R7
        LSR     R1,R1
        LSR     R1,R1
        LSR     R1,R1
        OR      R1,R7
        ST      R1,M(R13,01H)
        MOV     R1,#0002H
        ADD     R1,R13
        ST      R1,M(M_CURSOR)

        JMP     DECODE_READY
;=================================================
PA5_OUT:
        LD      R1,M(PA_CT)
        MOV     R5,R1
        MOV     R1,#0EFFH   ;CLEAR INTERRUPT FLAG
        AND     R1,R5
        ST      R1,M(PA_CT)
        LD      R1,M(M_R14TMP)     ;RESTORE R14
        MOV     R14,R1
        LD      R1,M(M_R1TMP)      ;RESTORE R1
        RETI
;f-----------------------------------------------
;-------------------------------------------------------------
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
TRACK:
        MOV     R1,#000DH
        XOR     R1,R5
        JMP     VDSL,ZE
        LD      R1,M(PA_IN)
        MOV     R5,R1
```

```
      MOV   R1,#0400H
      AND   R1,R5
      JMP   TRACK_ADD,ZE
      LD    R1,M(M_TRACK)
      ADD   R1,#0004H
      JMP   TRACK_WORK
TRACK_ADD:
      LD    R1,M(M_TRACK)
      SUB   R1,#0004H
TRACK_WORK:
      ST    R1,M(PWM_STL)
      ST    R1,M(M_TRACK)
      JMP   PA5_OUT
VDSL:
      LD    R1,M(M_TRACK)
      ADD   R1,#0040H
      JMP   TRACK_WORK
TRACK_END:
      MOV   R1,#0058H
      ;LD   R1,M(M_TRACK)
      JMP   TRACK_WORK
;-------------------------------------------------
```

สถาปัตยกรรมอย่างง่ายของไมโครคอนโทรลเลอร์โทรทัศน์ที่ถอดรหัสคำบรรยายภาพแบบซ่อนได้
A Simplified Architecture of a Closed Caption TV Microcontroller

คณิตพงศ์ เพ็งวัน และ เอกชัย ลีลารัศมี
ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
เขตพญาไท กรุงเทพฯ 10330
โทรศัพท์ 0-2218-6537 โทรสาร 0-2218-6488 E-mail: kanitp@digital.ee.eng.chula.ac.th

บทคัดย่อ

ในบทความนี้ได้นำเสนอสถาปัตยกรรมอย่างง่ายของไมโครคอนโทรลเลอร์โทรทัศน์ที่สามารถถอดรหัสคำบรรยายภาพแบบซ่อนได้ ซึ่งประกอบด้วย หน่วยประมวลผลกลางแบบ RISC ทำงานด้วยความเร็ว 12 MIPS ที่สัญญาณนาฬิกาความถี่ 12 MHz และอุปกรณ์บริวารทั่วไป เช่น ตัวสร้างสัญญาณมอดูเลตแบบความกว้างพัลส์, ตัวตั้งเวลา-ตัวนับ, พอร์ตอินพุต-เอาต์พุต และตัวเปรียบเทียบแรงดัน ฟังก์ชันการทำงานทั้งหมดของชิปถูกควบคุมด้วยหน่วยประมวลผลกลาง รวมถึงฟังก์ชันการแสดงผลบนหน้าจอ และการแยกข้อมูลคำบรรยายภาพซึ่งใช้เทคนิกไบแอสด้วยระดับแรงดันแบบปรับได้ (Adaptable Bias-Level Technique)

Abstract

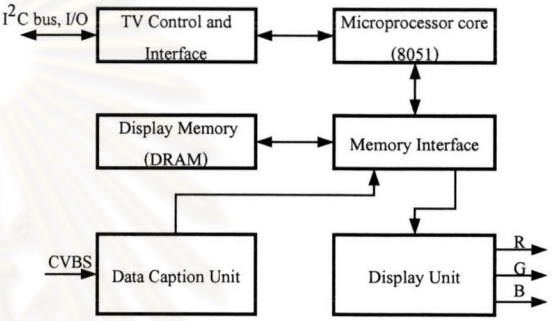
This paper presents a simplified architecture of a closed caption TV microcontroller. The architecture consists of a 16-bit RISC processor, running with speed of 12 MIPS at clock frequency of 12 MHz, and some general peripherals such as PWM generators, timer-counters, I/O ports, and a voltage comparator. The CPU performs all functions including OSD and caption data slicing. An adaptable bias-level technique is adopted in the data slicing function.

Keyword: TV Microcontroller, Closed-caption, VLSI design

1. คำนำ

การแสดงผลบนหน้าจอ (OSD: On Screen Display) เป็นฟังก์ชันสำคัญของเครื่องรับโทรทัศน์ในยุคปัจจุบัน เพื่อแสดงการตอบสนองต่อคำสั่งของผู้ใช้ และ แสดงคำบรรยายภาพ ไมโครคอนโทรลเลอร์โทรทัศน์ที่สามารถถอดรหัสคำบรรยายภาพได้ในท้องตลาดปัจจุบันดังเช่น ชิป SAAXX55 ของบริษัท Philips มีโครงสร้างตามรูปที่ 1 ประกอบด้วย 8051 เป็นหน่วยประมวลผลกลาง, หน่วยควบคุมและติดต่อโทรทัศน์, หน่วยความจำแสดงผล, หน่วยข้อมูลคำบรรยายภาพ และ หน่วยแสดงผล

หน่วยแสดงผลทำหน้าที่เป็น CRT controller เพื่อควบคุมการแสดงผลโดยเฉพาะ หน่วยประมวลผลกลางมีบทบาทในการแสดงผลด้วยการเขียนค่าลง SFR (Specific Function Register) เพื่อส่งพารามิเตอร์ให้กับหน่วยแสดงผลเท่านั้น การทำงานของหน่วยแสดงผลทำโดยการอ่านข้อมูลจากหน่วยความจำแสดงผล แล้วนำมาสร้างรูปแบบตัวอักษร โดยอาศัยรูปแบบตัวอักษรที่เก็บอยู่ในฟอนต์รอม(Font ROM) ซึ่งอยู่ภายในหน่วยแสดงผล เมื่อได้รูปแบบที่ต้องการจึงส่งออกมาเป็นสัญญาณ RGB



รูปที่ 1 โครงสร้างของชิป SAAXX55 ของบริษัท Pilips [3]

หน่วยข้อมูลคำบรรยายซึ่งภายในประกอบไปด้วยวงจรแปลงสัญญาณแอนาลอกเป็นดิจิตัล และตัวช่วยประมวลผลสัญญาณวีดีโอ ทำหน้าที่แยกข้อมูลคำบรรยายภาพที่แทรกมาในสัญญาณภาพ (ในชิปของบริษัทอื่น ใช้วงจรแคมป์เพื่อยกระดับแรงดันซิงค์ทิปของสัญญาณวีดีโอให้คงที่ก่อนนำไปแยกข้อมูล ซึ่งวงจรแคมป์เป็นวงจรแอนาลอกเช่นกัน)

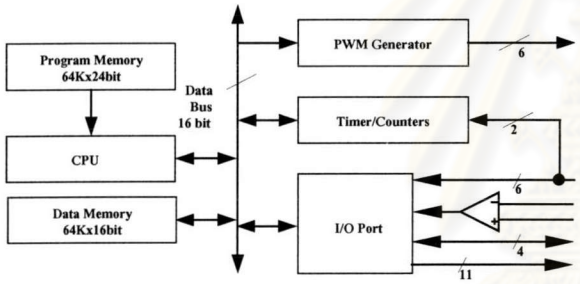
จากโครงสร้างดังกล่าวที่มีอุปกรณ์บริวารพิเศษ 2 ส่วนจะเห็นได้ว่าภาระงานของหน่วยประมวลผลกลาง 8051 เป็นเพียง รับคำสั่งจากผู้ใช้และควบคุมเครื่องรับโทรทัศน์ในส่วนที่เป็นการปรับภาพและเสียง และการปรับช่องสถานี ซึ่งเป็นภาระงานที่ไม่ซับซ้อน, ไม่ต้องการความเร็วในการประมวลผล, และมีปริมาณงานน้อย จึงเป็นการใช้งานหน่วยประมวลผลกลาง 8051 ซึ่งเป็นวงจรที่มีความซับซ้อนสูงอย่างไม่เต็มประสิทธิภาพ และการสร้างชิปดังกล่าวจำเป็นต้องเจือสารด้วยเทคโนโลยีสำหรับวงจรแอนาลอกด้วยเพื่อให้ฟังก์ชันการทำงานในส่วนวงจรแอนาลอกเป็นไปอย่างเที่ยงตรง

ดังนั้นในบทความนี้จึงนำเสนอไมโครคอนโทรลเลอร์โทรทัศน์ที่มีสถาปัตยกรรมอย่างง่ายมาควบคุมเครื่องรับโทรทัศน์และถอดรหัสคำบรรยายภาพแบบซ่อนได้ โดยฟังก์ชันการทำงานทั้งหมดถูกควบคุมด้วยหน่วยประมวลผลกลางที่สามารถประมวลผลด้วยความเร็วสูง ทำงานร่วมกับอุปกรณ์บริวารทั่วไปที่สามารถพบได้ในไมโครคอนโทรลเลอร์ใช้งาน

ทั่วไป ซึ่งจะทำให้วงจรทุกส่วนมีภาระงานอย่างสม่ำเสมอ โดยเฉพาะหน่วยประมวลผลกลางจะเกิดสภาวะว่างงาน (Idle state) ค่อนข้างน้อย เป็นการใช้งานอย่างมีประสิทธิภาพ เนื่องด้วยเทคโนโลยีวงจรรวมในปัจจุบันสามารถสร้างหน่วยประมวลผลกลางที่มีความเร็วสูงพอที่จะทำงานแทนอุปกรณ์พิเศษดังกล่าวข้างต้นได้ อีกทั้งสามารถปรับปรุงฟังก์ชันการทำงานของชิปได้ง่ายกว่าเพราะแก้ไขในส่วนที่เป็นซอฟท์แวร์เท่านั้น และวงจรไมโครคอนโทรลเลอร์ที่นำเสนอมีส่วนประกอบเป็นวงจรดิจิตัล จึงสามารถนำไปเจือสารด้วยเทคโนโลยีสำหรับวงจรดิจิตัล ที่มีต้นทุนต่ำกว่าเทคโนโลยีสำหรับวงจรแอนาลอก

2. สถาปัตยกรรม

ไมโครคอนโทรลมีหน่วยประมวลผลกลางแบบริสก์(RISC) ขนาด 16 บิต ทำงานที่สัญญาณนาฬิกาความถี่ 12 MHz มีสถาปัตยกรรมเป็นแบบฮาร์วาร์ด แบ่งการทำงานเป็นไปป์ไลน์ 2 ขั้นตอน คือขั้นตอนเฟทช์ และขั้นตอนดำเนินการ ความเร็วในการประมวลผล 12 MIPS ใช้เวลา 1 รอบสัญญาณนาฬิกาต่อการดำเนินการ 1 คำสั่ง (ยกเว้นคำสั่งในการกระโดดโปรแกรมใช้เวลาดำเนินการ 2 รอบสัญญาณนาฬิกา)



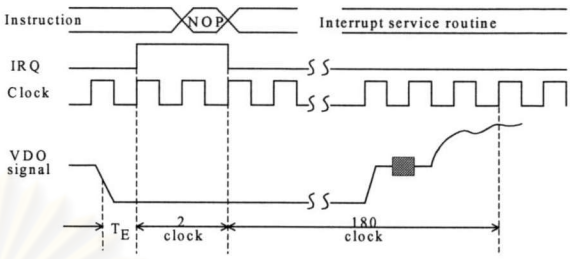
รูปที่ 2 สถาปัตยกรรมของไมโครคอนโทรลเลอร์

ส่วนอุปกรณ์บริวาร ประกอบด้วย ตัวสร้างสัญญาณมอดูเลตแบบความกว้างพัลส์, ตัวตั้งเวลา-ตัวนับ ขนาด 16 บิต 2 ชุด, พอร์ตอินพุต-เอาต์พุต, และ ตัวเปรียบเทียบแรงดัน

ตัวเปรียบเทียบแรงดันแม้ว่าเป็นวงจรที่มีอินพุตเป็นสัญญาณแอนาลอกก็ตาม แต่สามารถนำไปเจือสารด้วยเทคโนโลยีสำหรับวงจรดิจิตัลได้ เพียงแต่จะทำให้มีค่าแรงดัน Offset มากกว่าเทคโนโลยีสำหรับวงจรแอนาลอก[5] เมื่อนำไปใช้งานผลของ Offset สามารถถูกกำจัดได้โดย อัลกอริทึมในหัวข้อที่ 4 (ดูรูปที่ 8)

อุปกรณ์บริวารข้างต้นสามารถนำไปใช้ควบคุมเครื่องรับโทรทัศน์ได้ดังนี้ ตัวสร้างสัญญาณมอดูเลตแบบความกว้างพัลส์ใช้สร้างระดับแรงดันเพื่อควบคุมความดังเสียง, ความเข้ม, ความสว่าง, และ ระดับสีของภาพ ตัวตั้งเวลา-ตัวนับใช้เพื่อนับเส้นกวาดของสัญญาณภาพ และใช้ฟังก์ชันอินพุตแคปเจอร์ เพื่อหาระยะห่างของพัลส์ของสัญญาณจากรีโมทคอนโทรลแล้วนำข้อมูลดังกล่าวไปถอดรหัสคำสั่ง โดยหน่วยประมวลผลกลาง และพอร์ตอินพุต-เอาต์พุตเพื่อติดต่อควบคุมวงจรโทรทัศน์ส่วนอื่น ๆ เช่น วงจรจูน เป็นต้น

3. การแสดงผลบนหน้าจอ



รูปที่ 3 ช่วงเวลาเริ่มแสดงผลบนหน้าจอของแต่ละเส้นภาพ

เนื่องจากหน่วยประมวลผลกลางทำการควบคุมการแสดงผลจังหวะการทำงานจึงต้องสอดคล้องกับสัญญาณภาพ เพื่อให้ตำแหน่งของจุดในแนวเส้นตรงเดียวกันในแนวตั้งที่อยู่คนละเส้นกวาด หรือตำแหน่งของจุดเดียวกันที่อยู่คนละแฟรม อยู่ใกล้เคียงกันมากที่สุด จะทำให้ตัวอักษรที่แสดงผลบนหน้าจอมีคุณภาพดี

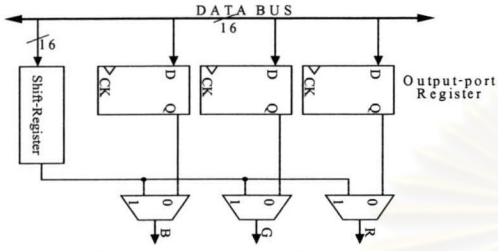
ดังนั้นโปรแกรมควบคุมการแสดงผลหน้าจอจึงถูกเก็บไว้ในชุดคำสั่งประจำสำหรับการขัดจังหวะ (ISR: Interrupt Service Routine) เมื่อเกิดขอบขาลงของสัญญาณ H-Sync หน่วยประมวลผลกลางจะถูกขัดจังหวะการทำงาน และจะดำเนินการคำสั่งในชุดคำสั่งประจำสำหรับการขัดจังหวะใน 2 รอบสัญญาณนาฬิกาถัดมา หน่วยประมวลผลกลางสามารถตอบสนองการขัดจังหวะการทำงานเท่ากันทุกครั้ง (Uniform Interrupt-Response Time) ดังนั้นตำแหน่งของจุดที่แสดงบนหน้าจอจึงผิดพลาดเท่ากับ T_E (ดูรูปที่ 3) ซึ่งน้อยกว่า 83 nsec

ตัวอักษรที่แสดงบนหน้าจอมีความละเอียดของตัวอักษรขึ้นอยู่ความถี่สัญญาณนาฬิกาจุด (Dot clock) ในเครื่องรับโทรทัศน์ทั่วไปสัญญาณนาฬิกาจุดเท่ากับ 12 MHz [2],[3] แสดงว่าทีวีไมโครคอนโทรลเลอร์ต้องส่งข้อมูลรูปแบบตัวอักษรออกไปให้วงจร PAL decoder ด้วยอัตราเร็ว 12 Mbit/sec ดังนั้นถ้าต้องการแสดงตัวอักษรที่มีความละเอียด 16 จุดใน 1 เส้นภาพ โดยใช้ไมโครคอนโทรลเลอร์ขนาด 16 บิต หน่วยประมวลผลกลางต้องทำงานอย่างน้อยที่สุด 6 คำสั่งคือ 3 คำสั่งต่อไปนี้

- อ่านข้อมูลจากหน่วยความจำแสดงผล (Display memory)
- อ่านข้อมูลรูปแบบอักษรจากฟอนต์รอมโดยใช้ข้อมูลจากหน่วยความจำแสดงผลเป็นตัวระบุตำแหน่ง
- เขียนข้อมูลรูปแบบอักษรให้รีจิสเตอร์เลื่อนข้อมูลออก

ซึ่งหน่วยประมวลผลกลางต้องสามารถประมวลผล ด้วยความเร็ว อย่างน้อยที่สุด 2.25 MIPS (= 3 instructions / (16 bit / 12 Mbit/sec)) และถ้าใช้ไมโครคอนโทรลเลอร์ 8 บิต ต้องมีความเร็ว 4.5 MIPS

เนื่องจากหน่วยประมวลผลกลางที่นำเสนอมีความเร็ว 12 MIPS ดังนั้นในการแสดงผล 1 ตัวอักษรหน่วยประมวลผลกลางจึงดำเนินการได้ 16 คำสั่ง ซึ่งเร็วเพียงพอที่คำนวณรูปแบบตัวอักษร โดยเฉพาะระบบตัวอักษรที่ซับซ้อนของภาษาไทย และสามารถใช้คำสั่งกระโดดในโปรแกรมแบบวนลูปเพื่อแสดงผลแบบต่อเนื่อง ซึ่งสามารถหลีกเลี่ยงการเขียนโปรแกรมแบบมาร์โครที่นิยมใช้งานที่ต้องการความเร็วแต่ใช้หน่วยความจำโปรแกรมจำนวนมาก



รูปที่ 4 โครงสร้างเอาต์พุตพอร์ตสำหรับแสดงผล

เมื่อหน่วยประมวลผลกลางคำนวณรูปแบบอักษรได้แล้ว ข้อมูลดังกล่าวจะถูกเขียนลงในรีจิสเตอร์เลื่อนข้อมูลขนาด 16 บิตซึ่งสัญญาณอนุกรมขาออกสามารถเลือกให้ออกทางขาของพอร์ตเอาต์พุตได้ 3 ขา (ตามรูปที่ 4) ทำหน้าที่เป็นสัญญาณ R, G, และ B ตามลำดับ รีจิสเตอร์ของพอร์ตเอาต์พุตทั้ง 3 ถูกเขียนให้มีค่าเป็น '0' ถ้าสัญญาณจากรีจิสเตอร์เลื่อนข้อมูลถูกเลือกให้ออกทั้ง 3 ขาพร้อมกันจะทำให้ได้จุดสีขาวบนหน้าจอ จากโครงสร้างดังกล่าวสามารถกำหนดสีของตัวอักษรได้ทั้งหมด 7 สี

4. การแยกข้อมูล

ข้อมูลคำบรรยายภาพแบบซ่อนได้สำหรับเครื่องรับโทรทัศน์ระบบ PAL จะถูกแทรกมากับสัญญาณภาพเส้นกวาดที่ 18 ในช่วงไร้สัญญาณภาพแนวตั้ง ในตอนเริ่มต้นของเส้นกวาดหลังจากสัญญาณ Color burst มีสัญญาณ Clock Run-in เป็นสัญญาณซานย์มี แอมปลิจูด 50 IRE (140 IRE = 1 volt) ความถี่ 500 kHz จำนวน 7 ลูกคลื่นหลังจากนั้นเป็นข้อมูลดิจิตัล ซึ่งช่วงคาบ 1 บิตเท่ากับ 2 μsec ลอจิก '1' มีระดับแรงดัน 50 IRE ส่วนลอจิก '0' มีระดับแรงดัน 0 IRE ข้อมูล 3 บิตแรกจะมีค่าเป็น 001 เสมอและมีข้อมูลจำนวน 2 ไบต์ตามมามีลักษณะตามรูปที่ 5

การแยกข้อมูลแบบเดิมใช้การแคมป์สัญญาณภาพแล้วนำไปเปรียบเทียบกับระดับแรงดันอ้างอิงค่าหนึ่ง แล้วอ่านผลลัพธ์ในแต่ละบิต หรือในกรณีชิป SAAXX55 แยกตัวแปลงสัญญาณแอนาลอกเป็นดิจิตัลแบบเฟลช (Flash ADC) ซึ่งมีความเร็วในการแปลงข้อมูลสูง ทำงานร่วมกับตัวช่วยประมวลผลสัญญาณวีดีโอ เพื่อใช้แยกข้อมูลคำบรรยายภาพ ซึ่งเป็นโครงสร้างที่ซับซ้อนเกินไปสำหรับข้อมูลคำบรรยายภาพที่มีอัตราของข้อมูลเพียง 500 kbit/sec

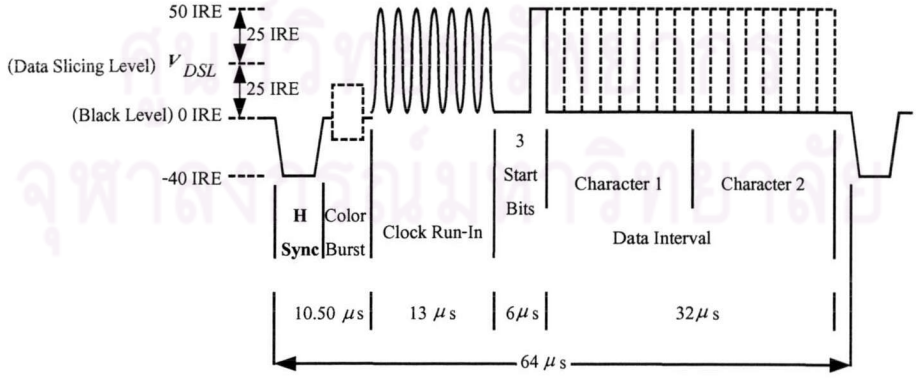
การแยกข้อมูลในบทความนี้ใช้วงจรตามรูปที่ 6 สัญญาณภาพจะถูกเชื่อมต่อผ่านตัวเก็บประจุ และถูกไบแอสด้วยแรงดันที่สร้างจากสัญญาณมอดูเลตความกว้างพัลส์ตามสมการ (1) ก่อนที่จะไปเปรียบเทียบกับแรงดันแยกข้อมูล V_{DSL} ซึ่งต่อกับแรงดันอ้างอิง 2 Volt เนื่องจากไม่ได้ใช้วงจรแคมป์ซึ่งเป็นผลให้แรงดันของสัญญาณภาพอาจมีค่าไม่แน่นอน เทคนิกการไบแอสด้วยระดับแรงดันแบบปรับได้จึงถูกนำมาใช้เพื่อหาระดับแรงดันไบแอสที่เหมาะสมและแยกข้อมูลได้อย่างถูกต้อง

$$V_{Bias} = \frac{R_2}{R_1 + R_2} \times \frac{PWM}{256} \times V_{ON} \qquad (1)$$

PWM: ค่าในรีจิสเตอร์ของตัวสร้างสัญญาณมอดูเลตความกว้างพัลส์

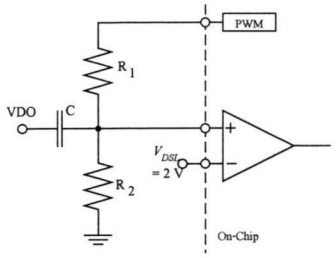
V_{ON}: แรงดัน High ของตัวสร้างสัญญาณมอดูเลตความกว้างพัลส์

เทคนิกการไบแอสด้วยระดับแรงดันแบบปรับ เริ่มต้นด้วยการคำนวณค่า PWM ให้ไบแอสเพื่อยกระดับแรงดันสีดำ (Black level ตามรูปที่ 5) ให้มีค่าตามสมการ (2)



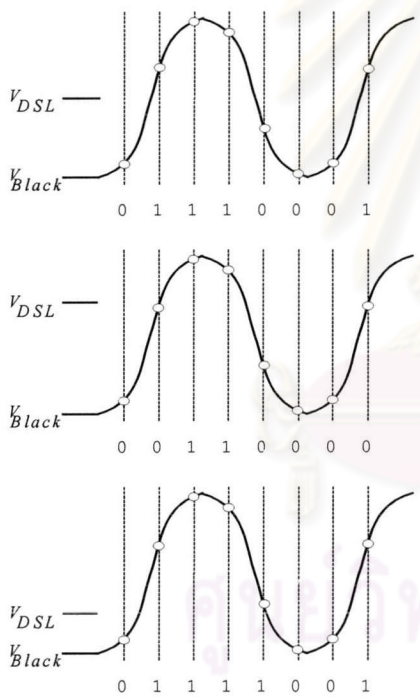
รูปที่ 5 สัญญาณภาพเส้นกวาดที่ 18 ซึ่งมีข้อมูลคำบรรยายภาพแทรกอยู่

$$V_{black} = V_{DSL} - 25 \text{ IRE} \quad (2)$$



รูปที่ 6 การต่อ R และ C เพื่อใช้แยกข้อมูลคำบรรยายภาพ

ผลลัพธ์จากตัวเปรียบเทียบแรงดันในช่วง Clock Run-in และช่วงข้อมูลถูกอ่านเข้าไปเก็บในหน่วยความจำทุก ๆ 4 รอบสัญญาณนาฬิกา (3 Mbit/sec) ผลลัพธ์ในช่วง Clock Run-in จะถูกใช้เพื่อตรวจจับการแทรกข้อมูลในเส้นกวาดที่ 18 และใช้เพื่อปรับแรงดันไบแอส



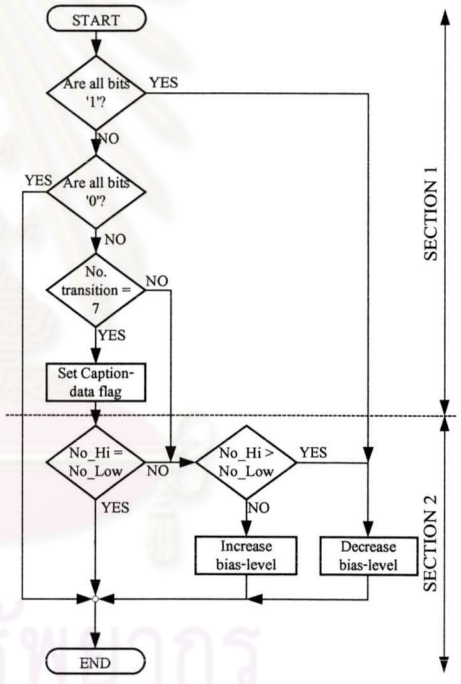
รูปที่ 7 สัญญาณ Clock Run-in และผลลัพธ์จากตัวเปรียบเทียบแรงดัน ในกรณีที่แรงดันไบแอส (บน) เหมาะสม (กลาง) ต่ำเกินไป (ล่าง) สูงเกินไป

พิจารณาในช่วง Clock Run-in ถ้าไบแอสด้วยแรงดันที่เหมาะสมทำให้ V_{DSL} อยู่บริเวณกึ่งกลางของสัญญาณซานย์ (ดูรูปที่ 7 บน) จะได้ผลลัพธ์จากตัวเปรียบเทียบแรงดันมีจำนวนบิต '1' และจำนวนบิต '0' ช่วงละ 3 บิต ถ้าแรงดันไบแอสต่ำเกินไปจำนวนบิต '1' จะน้อยกว่าจำนวนบิต '0' (ดูรูปที่ 7 กลาง) และถ้าแรงดันไบแอสสูงเกินไปจำนวนบิต '1' จะมากกว่าจำนวนบิต '0' (ดูรูปที่ 7 ล่าง)

รูปแบบของบิตที่ได้ในช่วง Clock Run-in จะถูกใช้เพื่อปรับแรงดันไบแอสที่ใช้ในแฟรมถัดไปตามอัลกอริทึมในรูปที่ 8 อัลกอริทึมในส่วนแรกเพื่อตรวจสอบว่ามีข้อมูลคำบรรยายภายแทรกมาในเส้นกวาดที่ 18 หรือไม่ ในส่วนที่ 2 เพื่อปรับระดับแรงดันไบแอส

เนื่องจากจังหวะการอ่านข้อมูลครั้งแรกของหน่วยประมวลผลกลาง อาจไม่สอดคล้องกับการเกิด Clock Run-in แต่ความคาดเคลื่อนมีค่าไม่เกิน T_E เช่นเดียวกับการแสดงผล (ดูรูปที่ 3) ดังนั้นขั้นแรงดันในการปรับแรงดันไบแอสแต่ละครั้งจึงกำหนดให้ เท่ากับค่ามากที่สุดของการเปลี่ยนแปลงสัญญาณ Clock Run-in ในช่วงเวลา 1 รอบสัญญาณนาฬิกา (T_{clk}= 83 nsec) ตามสมการ (3)

$$\text{Step size} = \max\{\, | A\sin(\omega t) - A\sin(\omega(t + T_{clk})) | \,\} \quad (3)$$



รูปที่ 8 อัลกอริทึมในการปรับค่าแรงดันไบแอส

การปรับแรงดันไบแอสด้วยขั้นแรงดันที่มาก ตามสมการ (3) อาจทำให้เกิดการแกว่งของแรงดันไบแอสในแต่และแฟรม แต่อย่างไรก็ตามอัลกอริทึมดังกล่าวยังจะทำให้ แรงดันของลอจิก '0' น้อยกว่า V_{DSL} และแรงดันของลอจิก '1' สูงกว่า V_{DSL} ซึ่งทำให้การแยกข้อมูลเป็นไปอย่างถูกต้อง นอกจากนี้การปรับแรงดันไบแอสยังชดเชยผลจากค่า Offset ของตัวเปรียบเทียบแรงดันได้ด้วย

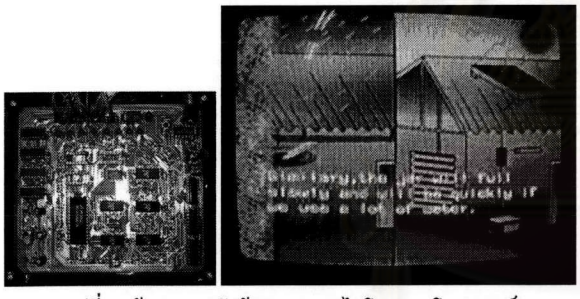
5. การสร้างและทดสอบตัวต้นแบบ

ตัวต้นแบบไมโครคอนโทรลเลอร์ ถูกสังเคราะห์ลงบน FPGA ของบริษัท Xilinx เบอร์ XCV300E (ใช้ทรัพยากรตามตารางที่ 1) ต่อร่วมกับไอซีภายนอกเช่น วงจรเปรียบเทียบแรงดันใช้ไอซีเบอร์ LM319N, และไอซีเบอร์ IDT 71V256 สำหรับใช้เป็นหน่วยความจำ การทดสอบตัวต้นแบบทำโดยการนำเครื่องรับโทรทัศน์สีขนาด 14 นิ้ว ของบริษัท Distar รุ่น DT-1416AV II ซึ่งมีไอซีเบอร์ MN152811DTC เป็นไมโคร-คอนโทรลเลอร์ มาถอดไมโคร-คอนโทรลเลอร์ออกและต่อวงจรตัวต้นแบบเข้าควบคุมเครื่องรับโทรทัศน์แทน

ตารางที่ 1 รายงานการใช้ทรัพยากรใน FPGA

Resource	Used	Max. Available	%Used
Slices	967	3,072	31
Flip Flops	831	6,144	13
LUTs	1250	6,144	20

ผลการทำงานตัวต้นแบบสามารถทำงานตามฟังก์ชันปกติของเครื่องรับโทรทัศน์ เช่นการปรับเสียง, การปรับความสว่าง, ความเข้ม, และ ระดับสีของภาพ ได้ และการปรับช่องสถานี ซึ่งวงจรจูนในเครื่องรับโทรทัศน์ดังกล่าวควบคุมด้วยสัญญาณมอดูเลตความกว้างพัลส์ และสามารถถอดรหัสคำบรรยายภาพแบบซ่อนได้ทั้งภาษาไทยและอังกฤษ



รูปที่ 9 (ซ้าย) วงจรตัวต้นแบบ ของไมโครคอนโทรลเลอร์
(ขวา) ผลการถอดรหัสคำบรรยายภาพ

6. บทสรุป

ในบทความนี้ได้นำเสนอสถาปัตยกรรมอย่างง่ายของไมโครคอนโทรลเลอร์โทรทัศน์ ซึ่งอาศัยหน่วยประมวลผลกลางที่มีความเร็วในความเร็วในการประมวลผล 12 MIPS ควบคุมการทำงานทั้งหมดรวมทั้งฟังก์ชันที่สำคัญ 2 อย่างคือ การควบคุมการแสดงผลบนหน้าจอ ที่สามารถปรับฟังก์ชันการแสดงผลให้เหมาะสมกับระบบภาษาไทยโดยทำการแก้ไขในส่วนของโปรแกรม และการควบคุมการแยกข้อมูลคำบรรยายภาพแบบซ่อนได้ด้วยเทคนิคการไบแอสด้วยระดับแรงดันแบบปรับได้เพื่อแยกข้อได้อย่างถูกต้องและลดผลจากค่า Offset ของตัวเปรียบเทียบแรงดัน จากสถาปัตยกรรมที่นำเสนอเป็นการใช้งานทุกส่วนของไมโครคอนโทรลเลอร์อย่างเต็มประสิทธิภาพโดยเฉพาะหน่วยประมวลผลกลาง

ตัวต้นแบบสามารถทำงานตามฟังก์ชันที่กำหนดได้ และพร้อมที่นำไปสร้างวางลายวงจรรวมเพื่อทำเป็นชิปเดี่ยวที่เจือสารด้วยเทคโนโลยีสำหรับวงจรดิจิตัลได้

7. กิตกรรมประกาศ

งานวิจัยนี้ได้รับทุนสนับสนุนจาก สำนักงานคณะกรรมการนโยบายพลังงานแห่งชาติ, ทุนอุดหนุนและส่งเสริมวิทยานิพนธ์ระดับปริญญาโท-เอก ทบวงมหาวิทยาลัย, และ กองทุกวิจัยรัชดาภิเษกสมโภช จุฬาลงกรณ์มหาวิทยาลัย

เอกสารอ้างอิง

[1] Hing-Yip Tong "A single chip micro-computer for A/V monitor and TV Receiver" *IEEE Transactions on Consumer Electronics,* November 1990, pp. 825-831.

[2] A. Tansathit and E. Leelarasmee "A Design of a TV Microcontroller Chip with Built-in Thai-English On Screen Display" *Engineering Transactions, a Research Publication of Mahanakorn University of Technology,* May 1999, pp.129-134.

[3] Philips Semiconductors "SAA55XX TV microcontrollers with Closed Captioning and On-Screen Display" *Preliminary specification*, February 2000.

[4] Alf-Egil Bogen and Vegard Wollan "AVR enhanced RISC Microcontroller. ATMEL Corporation" ATMEL Development Center, Trondheim, Norway

[5] Alcatel Microelectronics "C05M Design Rule Manual Supplement for Analogue Option" April 1999.

คณิตพงศ์ เพ็งวัน สำเร็จการศึกษาระดับปริญญาตรี สาขาวิศวกรรมไฟฟ้า จากมหาวิทยาลัยเชียงใหม่ เมื่อปี พ.ศ. 2540 จากนั้นได้เข้ารับราชการ ในตำแหน่งอาจารย์ ประจำภาควิชาวิศวกรรมไฟฟ้า มหาวิทยาลัยเชียงใหม่ ปัจจุบันกำลังศึกษาต่อในระดับปริญญาโท สาขาวิศวกรรมไฟฟ้า ที่จุฬาลงกรณ์มหาวิทยาลัย

เอกชัย ลีลารัศมี สำเร็จการศึกษาระดับปริญญาตรี สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย ในปี พ.ศ. 2517 ในปี พ.ศ. 2519 ถึง พ.ศ. 2525 ได้รับทุนอานันทมหิดล เพื่อไปศึกษาต่อ ในระดับปริญญาโทและเอก ณ University of California at Berkeley ประเทศสหรัฐอเมริกา ปัจจุบันดำรงตำแหน่งรองศาสตราจารย์ ประจำภาควิชาวิศวกรรมไฟฟ้า จุฬาลงกรณ์มหาวิทยาลัย

ประวัติผู้เขียนวิทยานิพนธ์

นายคณิตพงศ์ เพ็งวัน เกิดเมื่อวันที่ 27 สิงหาคม พ.ศ. 2519 ที่จังหวัดอุตรดิตถ์ สำเร็จการศึกษาระดับปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า จากคณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่ในปีการศึกษา 2539 และเข้ารับราชการในตำแหน่งอาจารย์ ที่ภาควิชาวิศวกรรมไฟฟ้า มหาวิทยาลัยเชียงใหม่ จากนั้นในปีการศึกษา 2543 ได้รับทุนการศึกษาจากสำนักนโยบายพลังงานแห่งชาติ(NEPO) เพื่อเข้าศึกษาต่อในระดับปริญญาวิศวศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า ณ ภาควิชาวิศวกรรมไฟฟ้า จุฬาลงกรณ์มหาวิทยาลัย