รายการอ้างอิง

1. Federal Communication Commission. REPORT and ORDER on GEN Docket NO. 91-1, April 12, 1991.

2. Hiroyasu Shindo, Tsutomu Nakazawa, Masaya Ohta, Kazuyuki Mitsuya, and Masaru Tonozuka. Microcontrollers for Closed Caption System. IEEE Transactions on Consumer Electronics, August 1992.

3. N. F. Hurley. A Single Chip Line 21 Captioning Decoder. IEEE Transactions on Consumer Electronics, August 1992.

4. U. Möller, W. Berthin, and R. Schwendt. A Single Chip Solution for Closed Caption Decoding. IEEE Transactions on Consumer Electronics, August 1992.

5. Zilog. Z86128 Line 21 Closed Caption Controller. Digital Television Controllers

6. สายัณห์ ธีรปัญญาวัฒน์ และ ดร. เอกชัย ลีลารัศมี. เครื่องแทรกคำบรรยายภาพแบบซ่อนได้ในระบบ PAL. ในหนังสือประกอบการประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 18, หน้า 542-547. พฤศจิกายน 2538

ภาคผนวก

ภาคผนวก ก

รายละเอียดของรหัสควบคุมเบ็ดเตล็ด

รหัสควบคุมเบ็ดเตล็ด

รหัสควบคุมเบ็ดเตล็ด (Miscellaneous Control Code) เป็นรหัสควบคุมชนิดหนึ่งของระบบคำบรรยายภาพแบบซ่อนได้ ใช้ควบคุมการแสดงผลของเครื่องถอดรหัส มีข้อมูลไบต์ที่ 1 และไบต์ที่ 2 ดังรูปที่ 1

ช่องสัญญาณข้อมูล CC1 หรือ T1		ช่องสัญญาณข้อมูล CC2 หรือ T2			
ข้อมูลไบต์ที่ 1	ข้อมูลไบต์ที่ 2	ข้อมูลไบต์ที่ 1	ข้อมูลไบต์ที่ 2	รหัสควบคุมเบ็ดเตล็ด	คำย่อ
14H	20H	1CH	20H	Resume Caption Loading	RCL
14H	21H	1CH	21H	Backspace	BS
14H	22H	1CH	22H	Reserved (formerly Alarm Off)	AOF
14H	23H	1CH	23H	Reserved (formerly Alarm On)	AON
14H	24H	1CH	24H	Delete to End of Row	DER
14H	25H	1CH	25H	Roll Up Caption 2 Rows	RU2
14H	26H	1CH	26H	Roll Up Caption 3 Rows	RU3
14H	27H	1CH	27H	Roll Up Caption 4 Rows	RU4
14H	28H	1CH	28H	Flash On	FON
14H	29H	1CH	29H	Resume Direct Captioning	RDC
14H	2AH	1CH	2AH	Text Restart	TR
14H	2BH	1CH	2BH	Resume Text Display	RTD
14H	2CH	1CH	2CH	Erase Displayed Memory	EDM
14H	2DH	1CH	2DH	Carriage Return	CR
14H	2EH	1CH	2EH	Erase Non Displayed Memory	ENM
14H	2FH	1CH	2FH	End of Caption (Flip Memories)	EOC
17H	21H	1FH	21H	Tab Offset 1 Column	TO1
17H	22H	1FH	22H	Tab Offset 2 Columns	TO2
17H	23H	1FH	23H	Tab Offset 3 Columns	TO3

รูปที่ 1 ตารางแสดงรหัสควบคุมเบ็ดเตล็ด

รหัสควบคุมเบ็ดเตล็ดแต่ละตัวมีผลต่อการแสดงผลของเครื่องถอดรหัสต่างกัน โดยแต่ละตัวมีรายละเอียดดังนี้

- *Resume Caption Loading* (RCL) เป็นรหัสควบคุมที่บอกเครื่องถอดรหัสให้รู้ว่า ข้อมูลที่ตามมาเป็นของช่องสัญญาณข้อมูลสำหรับโมดคำบรรยายภาพ (Caption Mode) ซึ่งจะถูกแสดงผลในแบบโผล่ทับ (Pop On Style) ถ้าโมดคำบรรยายภาพกำลังแสดงผลในแบบเลื่อนขึ้น (Roll Up Style) หรือแบบเขียนทับ (Paint On Style) คำบรรยายภาพที่แสดงอยู่จะยังไม่เปลี่ยนแปลง เนื่องจากว่าข้อมูลที่ตามมาจะถูกเก็บไว้ในหน่วยความจำไม่แสดงผล (Non Displayed Memory)

- *Backspace* (BS) เป็นรหัสควบคุมที่สั่งให้เครื่องถอดรหัสขยับเคอร์เซอร์ (Cursor) กลับไปทางซ้าย 1 สดมภ์ (Column) และลบตัวอักษร หรือรหัสกลางบรรทัด (Mid Row Code) ที่อยู่ ณ ตำแหน่ง

นั้น เนื่องจากว่าการแสดงผลของเครื่องถอดรหัสแบ่งเป็น 32 สดมภ์ต่อบรรทัด กรณีที่เคอร์เซอร์อยู่ที่สดมภ์ที่ 1 อยู่แล้ว หากได้รับรหัสควบคุมนี้เข้ามาก็จะไม่มีผลอันใด แต่ถ้าเคอร์เซอร์อยู่ที่สดมภ์ที่ 32 เคอร์เซอร์ก็จะย้ายมาอยู่ที่สดมภ์ที่ 31 และลบตัวอักษร หรือรหัสกลางบรรทัดที่อยู่ที่สดมภ์ที่ 31 โดยตัวอักษร หรือรหัสกลางบรรทัดที่อยู่ ณ สดมภ์ที่ 32 จะยังอยู่คงเดิม

- *Delete to End of Row* (DER) เป็นรหัสควบคุมที่สั่งให้เครื่องถอดรหัสลบตัวอักษร และรหัสกลางบรรทัด (Mid Row Code) ที่อยู่ตั้งแต่สดมภ์ (Column) ที่เคอร์เซอร์อยู่ไปทางขวาจนสุดบรรทัดออกจากหน่วยความจำ

- *Roll Up Caption 2-4 Rows* (RU2-RU4) เป็นรหัสควบคุมที่บอกเครื่องถอดรหัสให้รู้ว่า ข้อมูลที่ตามมาเป็นของช่องสัญญาณข้อมูลสำหรับโมดคำบรรยายภาพ (Caption Mode) ซึ่งจะถูกแสดงผลในแบบเลื่อนขึ้น (Roll Up Style) 2, 3 หรือ 4 บรรทัด ตามแต่รหัสควบคุมที่ส่งมา ถ้าโมดคำบรรยายภาพกำลังแสดงผลในแบบโผล่ทับ (Pop On Style) หรือแบบเขียนทับ (Paint On Style) คำบรรยายภาพที่แสดงอยู่จะถูกลบออกจากหน่วยความจำทันที

- *Flash On* (FON) เป็นรหัสควบคุมที่บอกเครื่องถอดรหัสว่า ตัวอักษรที่ตามมาจะถูกแสดงผลโดยมีลักษณะเป็นตัวกะพริบ ส่วนลักษณะอื่น ๆ อันได้แก่ สี, ตัวขีดเส้นใต้ และตัวเอียง จะคงเดิมไม่เปลี่ยนแปลง แต่ถ้าได้รับรหัสกลางบรรทัด (Mid Row Code) เข้ามาจะทำให้การแสดงตัวกะพริบหยุดลง รหัสควบคุมนี้จะปรากฏบนหน้าจอเหมือนกับว่าได้รับช่องว่าง 1 ตัวเข้ามา

- *Resume Direct Captioning* (RDC) เป็นรหัสควบคุมที่บอกเครื่องถอดรหัสให้รู้ว่า ข้อมูลที่ตามมาเป็นของช่องสัญญาณข้อมูลสำหรับโมดคำบรรยายภาพ (Caption Mode) ซึ่งจะถูกแสดงผลในแบบเขียนทับ (Paint On Style) ข้อมูลนี้จะถูกเขียนตรงไปที่หน่วยความจำแสดงผล (Displayed Memory) ทันทีที่ได้รับ

- *Text Restart* (TR) เป็นรหัสควบคุมที่บอกเครื่องถอดรหัสให้รู้ว่า ข้อมูลที่ตามมาเป็นของช่องสัญญาณข้อมูลสำหรับโมดข้อความ (Text Mode) เครื่องถอดรหัสต้องลบหน่วยความจำสำหรับโมดข้อความให้ว่างให้หมด และขยับเคอร์เซอร์ (Cursor) มาที่บรรทัดที่ 1 สดมภ์ (Column) ที่ 1

- *Resume Text Display* (RTD) เป็นรหัสควบคุมที่บอกเครื่องถอดรหัสให้รู้ว่า ข้อมูลที่ตามมาเป็นของช่องสัญญาณข้อมูลสำหรับโมดข้อความ (Text Mode) ใช้ในกรณีที่มีการส่งข้อมูลของช่องสัญญาณข้อมูลอื่นคั่นกลาง ข้อมูลที่ตามหลังรหัสควบคุมนี้จะปรากฏในตำแหน่งเดิม ที่เคอร์เซอร์ (Cursor) เคยอยู่ ก่อนส่งข้อมูลของช่องสัญญาณข้อมูลอื่นคั่น

- *Erase Displayed Memory* (EDM) เป็นรหัสควบคุมที่ใช้กับโมดคำบรรยายภาพ (Caption Mode) สั่งให้เครื่องถอดรหัสลบข้อมูลออกจากหน่วยความจำแสดงผล (Displayed Memory) ของโมดคำบรรยายภาพ รหัสควบคุมนี้ไม่ได้บอกเครื่องถอดรหัสว่า มีการเปลี่ยนช่องสัญญาณข้อมูล ข้อมูลที่ตามมายังอยู่ในช่องสัญญาณข้อมูลเดียวกันกับข้อมูลที่ส่งมาก่อนหน้ารหัสควบคุมนี้

- *Carriage Return* (CR) เป็นรหัสควบคุมที่สั่งให้เครื่องถอดรหัสขึ้นบรรทัดใหม่ ในกรณีที่ส่งมาในช่องสัญญาณข้อมูลสำหรับโมดคำบรรยายภาพ (Caption Mode) ถ้ากำลังแสดงคำบรรยายภาพในแบบเลื่อนขึ้น (Roll Up Style) หลังจากที่ได้รับรหัสควบคุมนี้แล้ว คำบรรยายภาพในบรรทัดบนสุดจะถูกลบออกจากหน่วยความจำ และจากหน้าจอ คำบรรยายภาพในบรรทัดอื่นจะเลื่อนขึ้นมาแทนที่ ส่วนเคอร์เซอร์ (Cursor) ยังอยู่บนบรรทัดเดิม แต่ย้ายไปที่สดมภ์ (Column) ที่ 1 ถ้าหากกำลังแสดงคำบรรยายภาพในแบบ (Style) อื่น หากได้รับรหัสควบคุมนี้เข้ามาก็จะไม่มีผลอันใด ในกรณีที่ส่งมาในช่องสัญญาณข้อมูลสำหรับโมดข้อความ (Text Mode) ถ้าเคอร์เซอร์ยังอยู่บนบรรทัดที่ 1 ถึง 14 หลังจากที่ได้รับรหัสควบคุมนี้แล้ว เคอร์เซอร์จะขยับลงมาอยู่บนบรรทัดถัดไป ณ สดมภ์ที่ 1 แต่ถ้าเคอร์เซอร์อยู่บนบรรทัดที่ 15 หลังจากได้รับรหัสควบคุมนี้แล้ว ข้อความในบรรทัดที่ 1 จะถูกลบออกจากหน่วยความจำ และจากหน้าจอ ข้อความในบรรทัดอื่นจะเลื่อนขึ้นมาแทนที่ ส่วนเคอร์เซอร์ยังอยู่บนบรรทัดที่ 15 เช่นเดิม แต่ย้ายไปที่สดมภ์ที่ 1

- *Erase Non Displayed Memory* (ENM) เป็นรหัสควบคุมที่ใช้กับโมดคำบรรยายภาพ (Caption Mode) สั่งให้เครื่องถอดรหัสลบข้อมูลออกจากหน่วยความจำไม่แสดงผล (Non Displayed Memory) ของโมดคำบรรยายภาพ รหัสควบคุมนี้ไม่ได้บอกเครื่องถอดรหัสว่า มีการเปลี่ยนช่องสัญญาณข้อมูล ข้อมูลที่ตามมายังอยู่ในช่องสัญญาณข้อมูลเดียวกันกับข้อมูลที่ส่งมาก่อนหน้ารหัสควบคุมนี้

- *End of Caption* (EOC) หรือ Flip Memories เป็นรหัสควบคุมที่ใช้กับโมดคำบรรยายภาพ (Caption Mode) สั่งให้เครื่องถอดรหัส สลับหน่วยความจำแสดงผล (Displayed Memory) กับหน่วยความจำไม่แสดงผล (Non Displayed Memory) ของโมดคำบรรยายภาพ ทำให้คำบรรยายภาพที่อยู่ในหน่วยความจำไม่แสดงผลปรากฏบนหน้าจอ รหัสควบคุมนี้ยังบอกเครื่องถอดรหัสให้รู้ว่า ข้อมูลที่ตามมาเป็นของช่องสัญญาณข้อมูลสำหรับโมดคำบรรยายภาพซึ่งจะถูกแสดงผลในแบบโผล่ทับ (Pop On Style)

- *Tab Offset 1-3 Columns* (TO1-TO3) เป็นรหัสควบคุมที่สั่งให้เครื่องถอดรหัสขยับเคอร์เซอร์ (Cursor) ของช่องสัญญาณข้อมูลนั้น ไปทางขวา 1, 2 หรือ 3 สดมภ์ (Column) โดยตัวอักษร หรือ

รหัสกลางบรรทัด (Mid Row Code) ที่อยู่ก่อนหน้าเคอร์เซอร์หลังจากขยับแล้วจะยังคงเดิม รหัสควบคุมนี้ใช้เพื่อกำหนดตำแหน่งเคอร์เซอร์ สำหรับจัดคำบรรยายภาพ หรือข้อความให้อยู่ในตำแหน่งที่เหมาะสม เนื่องจากการกำหนดตำแหน่งด้วยรหัสตำแหน่งเบื้องต้น (Preamble Address Code) ยังไม่ละเอียดพอ

ภาคผนวก ข

รายละเอียดของโปรแกรมถอดรหัส

```
mem_map                 byte    3f
capt_status             byte    0f0
capt_byte1              byte    0f1
capt_byte2              byte    0f2
osd_page                byte    0f4
osd_scroll              byte    0f5
io_switch               byte    0f8

info                    byte    0c0
style                   byte    0
language                byte    1
row                     byte    2
column                  byte    3
attribute               byte    4
height                  byte    5
disp_mem                byte    6
non_disp_mem            byte    7

roll_up                 byte    0
pop_on                  byte    1
paint_on                byte    2

solid_space             byte    20
legible_space           byte    1
;*******************************************************
                        org     0
start                   mov     r8,#80                  ;erase osd memory
                        mov     r9,#0
                        mov     ra,#40
                        mov     rb,#0
                        mov     r0,#0
start_erase_mem_loop    sto     r0,@r8:r9
                        add     r9,#1
                        adc     r8,#0
                        add     rb,#1
                        suc     ra,#0
                        jnz     start_erase_mem_loop

                        mov     r8,#0a0                 ;write black box
                        mov     r9,#82                  ;in text memory
                        mov     ra,#0f
                        mov     r0,#legible_space
                        mov     r1,#0
start_box_row_loop      mov     rb,#22
start_box_column_loop   sto     r0,@r8:r9
                        add     r8,#10
                        sto     r0,@r8:r9
                        sub     r8,#10
                        add     r9,#1
                        sto     r1,@r8:r9
                        add     r8,#10
                        sto     r1,@r8:r9
                        sub     r8,#10
                        add     r9,#1
                        sub     rb,#1
                        jnz     start_box_column_loop
                        add     r9,#3c
                        adc     r8,#0
                        sub     ra,#1
                        jnz     start_box_row_loop
```

```
                        mov     r0,#2                   ;write PAC
                        mov     r9,#80
                        mov     r8,#88
                        sto     r0,@r8:r9
                        mov     r8,#98
                        sto     r0,@r8:r9
                        mov     r8,#0a0
                        sto     r0,@r8:r9
                        mov     r8,#0b0
                        sto     r0,@r8:r9
                        mov     r0,#7
                        mov     r9,#81
                        mov     r8,#88
                        sto     r0,@r8:r9
                        mov     r8,#98
                        sto     r0,@r8:r9
                        mov     r8,#0a0
                        sto     r0,@r8:r9
                        mov     r8,#0b0
                        sto     r0,@r8:r9

                        mov     r8,#start_info_table_H  ;initial info table
                        mov     r9,#start_info_table_L
                        mov     ra,#info
                        mov     rc,#4
start_info_channel_loop mov     rb,#0
                        mov     rd,#8
start_info_data_loop    lod     r0,@r8:r9
                        sto     r0,@ra:rb
                        add     r9,#1
                        adc     r8,#0
                        add     rb,#1
                        sub     rd,#1
                        jnz     start_info_data_loop
                        add     ra,#1
                        sub     rc,#1
                        jnz     start_info_channel_loop

                        mov     r2,#0                   ;set up register
                        mov     r3,#0
                        mov     r6,#0
                        mov     r0,#0
                        mov     ra,#mem_map
                        mov     rb,#osd_page
                        sto     r0,@ra:rb
;*****************************************************
wait_capt_data          mov     ra,#mem_map             ;if data not come in
                        mov     rb,#capt_status
                        lod     r0,@ra:rb
                        xor     r0,#0c
                        jz      read_capt_data
                        mov     rb,#io_switch           ;then
                        lod     r0,@ra:rb               ;if switch change
                        xor     r0,r3
                        jz      wait_capt_data
                        xor      r3,r0                      ;then change disp
page
                        mov     ra,#info
                        add     ra,r3
                        mov     rb,#disp_mem
                        lod     r0,@ra:rb
```

```
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r0,#7
                        mov     ra,#mem_map
                        mov     rb,#osd_page
                        sto     r0,@ra:rb
                        jmp     wait_capt_data          ;wait data
read_capt_data          mov     rb,#capt_byte1          ;else
                        lod     r4,@ra:rb               ;read data
                        mov     rb,#capt_byte2
                        lod     r5,@ra:rb

                        mov      r0,r4                      ;if byte 1 par.
error
                        add     r0,#80
                        jc      byte1_check_00_0f
                        mov      r6,#0                     ;then clear last
pair
                        mov      r0,#solid_space           ;      print solid
spc.
                        mov     r8,#print_H
                        mov     r9,#print_L
                        mov     rc,#byte2_print_H       ;      print byte 2
                        mov     rd,#byte2_print_L
                        jmp     @r8:r9

byte1_check_00_0f       sub     r0,#10                  ;else if in 00-0f
                        jnc     byte1_check_10_1f
                        mov      r6,#0                     ;then clear last
pair
                        jmp     byte2_print             ;      print byte 2

byte1_check_10_1f       sub     r0,#10                  ;else if in 10-1f
                        jnc     byte1_print
                        mov     r0,r5                   ;then
                        add      r0,#80                     ;if byte 2 par.
error
                        jc      pair_check_redundant
                        mov      r6,#0                     ;then clear last
pair
                        jmp     wait_capt_data          ;      wait data
pair_check_redundant    mov     r0,r4                   ;else if same pair
                        xor     r0,r6
                        jnz     pair_execute
                        mov     r0,r5
                        xor     r0,r7
                        jnz     pair_execute
                        mov      r6,#0                     ;then clear last
pair
                        jmp     wait_capt_data          ;      wait data
pair_execute            mov     r6,r4                   ;else save last pair
                        mov     r7,r5
                        mov     r8,#command_H           ;      do command
                        mov     r9,#command_L
                        jmp     @r8:r9

byte1_print              mov      r6,#0                     ;else clear last
pair
                        mov      r0,r4                     ;      set byte 1
code
```

```
                        and     r0,#7f                  ;       respect to
                        mov      ra,#info                       ;       channel
language
                        add     ra,r2
                        mov     rb,#language
                        lod     r1,@ra:rb
                        add     r0,r1
                        mov     r8,#print_H             ;       print byte 1
                        mov     r9,#print_L
                        mov     rc,#byte2_print_H       ;       print byte 2
                        mov     rd,#byte2_print_L
                        jmp     @r8:r9

byte2_print               mov      r0,r5                      ;if byte 2 par.
error
                        add     r0,#80
                        jc      byte2_check_20_7f
                        mov      r0,#solid_space           ;then print solid
spc.
                        mov     r8,#print_H
                        mov     r9,#print_L
                        mov     rc,#wait_capt_data_H    ;       wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @r8:r9

byte2_check_20_7f       sub     r0,#20                  ;else if in 20-7f
                        jc      wait_capt_data
                        mov      r0,r5                      ;then set byte 2
code
                        and     r0,#7f                  ;       respect to
                        mov      ra,#info                       ;       channel
language
                        add     ra,r2
                        mov     rb,#language
                        lod     r1,@ra:rb
                        add     r0,r1
                        mov     r8,#print_H             ;       print byte 2
                        mov     r9,#print_L
                        mov     rc,#wait_capt_data_H    ;       wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @r8:r9
;*******************************************************
print                   mov     ra,#info                ;offset =
                        add     ra,r2                   ;row*128 + col*2 + 2
                        mov     rb,#row
                        lod     r8,@ra:rb
                        mov     rb,#column
                        lod     r9,@ra:rb
                        mov     r1,#0
                        shr     r8
                        src     r1
                        shl     r9
                        add     r9,r1
                        add     r9,#2

                        mov     rb,#style               ;if style = pop on
                        lod     r1,@ra:rb
                        xor     r1,#pop_on
                        jnz     print_disp_mem
                        mov      rb,#non_disp_mem           ;then address =
offset
```

```
                    lod     r1,@ra:rb           ;     + non disp mem
                    add     r8,r1
                    jmp     Mai_Han_Ar_Kad
print_disp_mem        mov     rb,#disp_mem          ;else address =
offset
                    lod     r1,@ra:rb           ;     + disp mem
                    add     r8,r1

Mai_Han_Ar_Kad      xor     r0,#0d1               ;if char = upper Sa
Ra
                    jnz     Sa_Ra_I
                    mov     r0,#4
                    jmp     print_upper
Sa_Ra_I             xor     r0,#5
                    jnz     Sa_Ra_E
                    mov     r0,#8
                    jmp     print_upper
Sa_Ra_E             xor     r0,#1
                    jnz     Sa_Ra_Ue
                    mov     r0,#0c
                    jmp     print_upper
Sa_Ra_Ue            xor     r0,#3
                    jnz     Sa_Ra_U
                    mov     r0,#10
                    jmp     print_upper
Sa_Ra_U             xor     r0,#1
                    jnz     Mai_Tai_Koo
                    mov     r0,#14
                    jmp     print_upper
Mai_Tai_Koo         xor     r0,#30
                    jnz     Sa_Ra_Ou
                    mov     r0,#18
print_upper         sub     r9,#1               ;then replace old
                    lod     r1,@r8:r9             ;     upper Sa Ra
code
                    and     r1,#0e3             ;     with new code
                    or      r0,r1
                    sto     r0,@r8:r9
                    jmp     @rc:rd              ;     return

Sa_Ra_Ou            xor     r0,#3f                ;if char = lower Sa
Ra
                    jnz     Sa_Ra_Oo
                    mov     r0,#1
                    jmp     print_lower
Sa_Ra_Oo            xor     r0,#1
                    jnz     lower_dot
                    mov     r0,#2
                    jmp     print_lower
lower_dot           xor     r0,#3
                    jnz     Mai_Eak
                    mov     r0,#3
print_lower         sub     r9,#1               ;then replace old
                    lod     r1,@r8:r9             ;     lower Sa Ra
code
                    and     r1,#0fc             ;     with new code
                    or      r0,r1
                    sto     r0,@r8:r9
                    jmp     @rc:rd              ;     return

Mai_Eak             xor     r0,#32              ;if char = Wa Na Yuk
```

```
                    jnz     Mai_To
                    mov     r0,#20
                    jmp     print_wanayuk
Mai_To              xor     r0,#1
                    jnz     Mai_Tri
                    mov     r0,#40
                    jmp     print_wanayuk
Mai_Tri             xor     r0,#3
                    jnz     Mai_Ja_Ta_Wa
                    mov     r0,#60
                    jmp     print_wanayuk
Mai_Ja_Ta_Wa        xor     r0,#1
                    jnz     Ka_Ran
                    mov     r0,#80
                    jmp     print_wanayuk
Ka_Ran              xor     r0,#7
                    jnz     print_middle
                    mov     r0,#0a0
print_wanayuk       sub     r9,#1               ;then replace old
                    lod     r1,@r8:r9           ;     Wa Na Yuk code
                    and     r1,#01f             ;     with new code
                    or      r0,r1
                    sto     r0,@r8:r9
                    jmp     @rc:rd              ;     return

print_middle        xor     r0,#0ec             ;else
                    sto     r0,@r8:r9           ;write char
                    add     r9,#1
                    mov     r0,#0
                    sto     r0,@r8:r9

                    sub     r9,#3               ;if left is empty
                    lod     r0,@r8:r9
                    xor     r0,#0
                    jnz     print_right_legible_spc
                    mov     r0,#legible_space   ;then write
                    sto     r0,@r8:r9           ;     legible space
                    add      r9,#1                  ;      at left of
char
                    mov     r0,#0
                    sto     r0,@r8:r9
                    sub     r9,#1

print_right_legible_spc add     r9,#4           ;if right is empty
                    lod     r0,@r8:r9
                    xor     r0,#0
                    jnz     print_step_cursor
                    mov     r0,#legible_space   ;then write
                    sto     r0,@r8:r9           ;     legible space
                    add      r9,#1                 ;      at right of
char
                    mov     r0,#0
                    sto     r0,@r8:r9

print_step_cursor   mov     rb,#column          ;col = col + 1
                    lod     r0,@ra:rb
                    add     r0,#1
                    mov     r1,r0               ;if col = 33
                    xor     r1,#21
                    jnz     print_return
                    mov     r0,#20              ;then col = 32
```

```
print_return            sto     r0,@ra:rb
                        jmp     @rc:rd                  ;return
;*****************************************************
command                 mov     r0,r5                   ;if byte 2 in 00-1f
                        sub     r0,#0a0
                        jnc     PAC_check
                        mov     rc,#wait_capt_data_H    ;then wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++
PAC_check               mov     r0,r5                   ;else if in 40-7f
                        and     r0,#40
                        jz      MID_check

                        mov     r0,r4                   ;then
                        and     r0,#8                   ;change data channel
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov     r0,r4                   ;look up row
                        and     r0,#7
                        shl     r0
                        mov     r1,r5
                        and     r1,#20
                        shl     r1
                        shl     r1
                        shl     r1
                        slc     r1
                        or      r0,r1
                        mov     ra,#PAC_row_table_H
                        mov     rb,#PAC_row_table_L
                        add     rb,r0
                        adc     ra,#0
                        lod     r8,@ra:rb
                        xor     r8,#0                   ;if row = 0
                        jnz     PAC_column
                        mov     rc,#wait_capt_data_H    ;then wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd

PAC_column              mov     r0,r5                   ;else if byte 2 in
                        and     r0,#10                  ;        50-5f,70-7f
                        jz      PAC_attribute
                        mov     r0,r5                   ;then look up column
                        and     r0,#0e
                        shr     r0
                        mov     ra,#PAC_column_table_H
                        mov     rb,#PAC_column_table_L
                        add     rb,r0
                        adc     ra,#0
                        lod     r9,@ra:rb
                        mov     r1,#7                   ;     attr = white
                        jmp     PAC_underline

PAC_attribute           mov     r0,r5                   ;else look up attr
                        and     r0,#0e
                        shr     r0
                        mov     ra,#PAC_attr_table_H
```

```
                    mov     rb,#PAC_attr_table_L
                    add     rb,r0
                    adc     ra,#0
                    lod     r1,@ra:rb
                    mov     r9,#1                   ;     column = 1

PAC_underline       mov     r0,r5                   ;underline = byte 2
                    and     r0,#1                   ;            bit 0
                    shl     r0
                    shl     r0
                    shl     r0
                    shl     r0
                    or      r1,r0

                    mov     r0,r2                   ;if channel = text
                    and     r0,#2
                    jz      PAC_caption
                    mov     ra,#info                ;then
                    add     ra,r2                   ;set cursor column
                    mov     rb,#column
                    sto     r9,@ra:rb
                    mov     rb,#attribute           ;set attribute
                    sto     r1,@ra:rb
                    mov     rb,#row                 ;address =
                    lod     r8,@ra:rb               ;row*128 + disp mem
                    mov     r9,#0
                    shr     r8
                    src     r9
                    mov     rb,#disp_mem
                    lod     r0,@ra:rb
                    add     r8,r0
                    mov     r0,#2                   ;write PAC
                    sto     r0,@r8:r9
                    add     r9,#1
                    sto     r1,@r8:r9
                    mov     rc,#wait_capt_data_H    ;wait data
                    mov     rd,#wait_capt_data_L
                    jmp     @rc:rd

PAC_caption         mov     ra,#info                ;else
                    add     ra,r2                   ;if style /= roll up
                    mov     rb,#style
                    lod     r0,@ra:rb
                    xor     r0,#roll_up
                    jz      PAC_roll_up
                    mov     rb,#row                 ;then
                    sto     r8,@ra:rb               ;set cursor row
                    mov     rb,#column              ;set cursor column
                    sto     r9,@ra:rb
                    mov     rb,#attribute           ;set attribute
                    sto     r1,@ra:rb
                    mov     r9,#0                   ;offset = row * 128
                    shr     r8
                    src     r9
                    mov     rb,#style               ;if style /= pop on
                    lod     r0,@ra:rb
                    xor     r0,#pop_on
                    jz      PAC_pop_on
                    mov      rb,#disp_mem                ;then address =
offset
                    lod     r0,@ra:rb               ;     + disp mem
```

```
                        add     r8,r0
                        jmp     PAC_pop_paint_finish
PAC_pop_on                mov     rb,#non_disp_mem          ;else address =
offset
                        lod     r0,@ra:rb                 ;     + non disp mem
                        add     r8,r0
PAC_pop_paint_finish    mov     r0,#2                     ;write PAC
                        sto     r0,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9
                        mov     rc,#wait_capt_data_H      ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd

PAC_roll_up             mov     rb,#attribute             ;else
                        sto     r1,@ra:rb                 ;set attribute
                        mov     rb,#column                ;set cursor column
                        sto     r9,@ra:rb
                        mov     rb,#row                   ;set cursor row
                        lod     rd,@ra:rb
                        sto     r8,@ra:rb

                        mov     r9,r8                     ;compare new row
                        sub     r8,rd                     ;with old row
                        jz      PAC_roll_up_finish
                        jnc     PAC_move_down

                        mov      rb,#height                  ;if new row < old
row
                        lod     rc,@ra:rb                 ;then
                        mov     r8,rd                     ;if top row src is
                        add     r8,#1                     ;   out of screen
                        sub     r8,rc
                        jz      PAC_move_up_skip
                        jc      PAC_move_up_skip
                        jmp     PAC_move_up_check_des
PAC_move_up_skip        mov     rc,rd                     ;then reduce counter
                        mov     r8,#1                      ;      top row src =
1
PAC_move_up_check_des   add     r9,#1                     ;if top row des is
                        sub     r9,rc                     ;   out of screen
                        jz      PAC_move_up_erase
                        jc      PAC_move_up_erase
                        jmp     PAC_move_up_move
PAC_move_up_erase       mov     rb,#disp_mem              ;then erase src row
                        lod      r0,@ra:rb                  ;      that des row
is
                        mov     ra,#1                     ;     out of screen
                        sub     ra,r9
                        sub     rc,ra
                        mov     r9,#0
                        shr     r8
                        src     r9
                        add     r8,r0
                        mov     r0,#0
PAC_move_up_erase_row   mov     rb,#46
PAC_move_up_erase_col   sto     r0,@r8:r9
                        add     r9,#1
                        sub     rb,#1
                        jnz     PAC_move_up_erase_col
                        add     r9,#3a
```

```
                        adc     r8,#0
                        sub     ra,#1
                        jnz     PAC_move_up_erase_row
PAC_move_up_move        mov     ra,#info                ;move src row
                        add     ra,r2                   ;to des row
                        mov     rb,#disp_mem
                        lod     r0,@ra:rb
                        mov     rb,#row
                        lod     r8,@ra:rb
                        add     r8,#1
                        sub     r8,rc
                        mov     r9,#0
                        shr     r8
                        src     r9
                        add     r8,r0
                        mov     ra,rd
                        add     ra,#1
                        sub     ra,rc
                        mov     rb,#0
                        shr     ra
                        src     rb
                        add     ra,r0
                        mov     r1,#0
PAC_move_up_move_row    mov     rd,#46
PAC_move_up_move_column lod     r0,@ra:rb
                        sto     r1,@ra:rb
                        add     rb,#1
                        sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     PAC_move_up_move_column
                        add     rb,#3a
                        adc     ra,#0
                        add     r9,#3a
                        adc     r8,#0
                        sub     rc,#1
                        jnz     PAC_move_up_move_row
                        jmp     PAC_roll_up_finish

PAC_move_down           mov     rb,#height              ;if new row > old
row
                        lod     rc,@ra:rb               ;then
                        mov     r8,rd                   ;if top row src is
                        add     r8,#1                   ;   out of screen
                        sub     r8,rc
                        jz      PAC_move_down_reduce
                        jc      PAC_move_down_reduce
                        jmp     PAC_move_down_move
PAC_move_down_reduce    mov     rc,rd                   ;then reduce counter
PAC_move_down_move      mov     ra,#info                ;move src row
                        add     ra,r2                   ;to des row
                        mov     rb,#disp_mem
                        lod     r0,@ra:rb
                        mov     rb,#row
                        lod     r8,@ra:rb
                        mov     r9,#0
                        shr     r8
                        src     r9
                        add     r8,r0
                        mov     ra,rd
                        mov     rb,#0
```

```
                        shr     ra
                        src     rb
                        add     ra,r0
                        mov     r1,#0
PAC_move_down_move_row  mov     rd,#46
PAC_move_down_move_col  lod     r0,@ra:rb
                        sto     r1,@ra:rb
                        add     rb,#1
                        sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     PAC_move_down_move_col
                        sub     rb,#0c6
                        suc     ra,#0
                        sub     r9,#0c6
                        suc     r8,#0
                        sub     rc,#1
                        jnz     PAC_move_down_move_row

PAC_roll_up_finish      mov     ra,#info                ;address =
                        add     ra,r2                   ;row*128 + disp mem
                        mov     rb,#row
                        lod     r8,@ra:rb
                        mov     rb,#disp_mem
                        lod     r0,@ra:rb
                        mov     r9,#0
                        shr     r8
                        src     r9
                        add     r8,r0

                        mov     r0,#2                   ;write PAC
                        sto     r0,@r8:r9
                        mov     rb,#attribute
                        lod     r0,@ra:rb
                        add     r9,#1
                        sto     r0,@r8:r9
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
MID_check               mov     r0,r4                   ;else
                        and     r0,#7                   ;if byte 1 = 11,19
                        xor     r0,#1
                        jnz     MISC_check
                        mov     r0,r5                   ;then
                        and     r0,#70                  ;if byte 2 in 20-2f
                        xor     r0,#20
                        jnz     spec_char_check

                        mov     r0,r4                   ;then
                        and     r0,#8                   ;change data channel
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov     r0,r5                   ;look up attr
                        and     r0,#0e
                        shr     r0
                        mov     ra,#MID_attr_table_H
```

```
                    mov     rb,#MID_attr_table_L
                    add     rb,r0
                    adc     ra,#0
                    lod     r0,@ra:rb
                    mov       ra,#info                    ;if new attr =
italic
                    add     ra,r2
                    mov     rb,#attribute
                    mov     r1,r0
                    xor     r0,#8
                    jnz     MID_underline
                    lod     r0,@ra:rb           ;then add italic
                    and     r0,#1f              ;     to old attr
                    or      r1,r0

MID_underline       mov     r0,r5               ;underline = byte 2
                    and     r0,#1               ;            bit 0
                    shl     r0
                    shl     r0
                    shl     r0
                    shl     r0
                    and     r1,#0ef
                    or      r1,r0
                    sto     r1,@ra:rb           ;set attribute

                    mov     rb,#row             ;offset =
                    lod     r8,@ra:rb           ;row*128 + col*2 + 2
                    mov     rb,#column
                    lod     r9,@ra:rb
                    mov     r0,#0
                    shr     r8
                    src     r0
                    shl     r9
                    add     r9,r0
                    add     r9,#2

                    mov     rb,#style           ;if style = pop on
                    lod     r0,@ra:rb
                    xor     r0,#pop_on
                    jnz     MID_disp_mem
                    mov       rb,#non_disp_mem          ;then address =
offset
                    lod     r0,@ra:rb           ;     + non disp mem
                    add     r8,r0
                    jmp     MID_print
MID_disp_mem          mov       rb,#disp_mem            ;else address =
offset
                    lod     r0,@ra:rb           ;     + disp mem
                    add     r8,r0

MID_print           mov     r0,#3               ;write MID
                    sto     r0,@r8:r9
                    add     r9,#1
                    sto     r1,@r8:r9

                    sub     r9,#3               ;if left is empty
                    lod     r0,@r8:r9
                    xor     r0,#0
                    jnz     MID_right_legible_spc
                    mov     r0,#legible_space   ;then write
                    sto     r0,@r8:r9           ;     legible space
```

```
                        add     r9,#1                   ;       at left of MID
                        mov     r0,#0
                        sto     r0,@r8:r9
                        sub     r9,#1

MID_right_legible_spc   add     r9,#4                   ;if right is empty
                        lod     r0,@r8:r9
                        xor     r0,#0
                        jnz     MID_step_cursor
                        mov     r0,#legible_space       ;then write
                        sto     r0,@r8:r9               ;       legible space
                        add      r9,#1                      ;       at right of
MID
                        mov     r0,#0
                        sto     r0,@r8:r9

MID_step_cursor         mov     rb,#column              ;col = col + 1
                        lod     r0,@ra:rb
                        add     r0,#1
                        mov     r1,r0                   ;if col = 33
                        xor     r1,#21
                        jnz     MID_finish
                        mov     r0,#20                  ;then col = 32
MID_finish              sto     r0,@ra:rb
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
spec_char_check         mov     r0,r5                   ;else
                        and     r0,#70                  ;if byte 2 in 30-3f
                        xor     r0,#30
                        jz      spec_char_print
                        mov     rc,#wait_capt_data_H
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd

spec_char_print         mov     r0,r4                   ;then
                        and     r0,#8                   ;change data channel
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov      r0,r5                      ;look up special
char
                        and     r0,#0f
                        mov     ra,#spec_char_table_H
                        mov     rb,#spec_char_table_L
                        add     rb,r0
                        adc     ra,#0
                        lod     r0,@ra:rb

                        mov     r8,#print_H             ;print spec char
                        mov     r9,#print_L
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @r8:r9
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
MISC_check              mov     r0,r4                   ;if byte 1 = 14,1c
                        and     r0,#7
```

```
                        xor     r0,#4
                        jnz     TOn_check
                        mov     r0,r5                   ;then
                        and     r0,#70                  ;if byte 2 in 20-2f
                        xor     r0,#20
                        jz      MISC_execute
                        mov     rc,#wait_capt_data_H
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd

MISC_execute            mov     r0,r5                   ;then
                        and     r0,#0f                  ;look up function
                        shl     r0
                        mov     ra,#MISC_func_table_H
                        mov     rb,#MISC_func_table_L
                        add     rb,r0
                        adc     ra,#0
                        lod     r8,@ra:rb
                        add     rb,#1
                        adc     ra,#0
                        lod     r9,@ra:rb
                        jmp     @r8:r9                  ;do function
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
TOn_check               mov     r0,r4                   ;else
                        and     r0,#7                   ;if byte 1 = 17,1f
                        xor     r0,#7
                        jnz     invalid_command
                        mov     r0,r5                   ;   byte 2 in 21-23
                        sub     r0,#0a1
                        jc      invalid_command
                        sub     r0,#3
                        jc      TOn
invalid_command         mov     rc,#wait_capt_data_H
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd

TOn                     mov     r0,r4                   ;then
                        and     r0,#8                   ;change data channel
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov     ra,#info                ;col = col + byte 2
                        add     ra,r2                   ;          - 20
                        mov     rb,#column
                        lod     r0,@ra:rb
                        add     r0,r5
                        sub     r0,#0a0
                        mov     r1,r0                   ;if col > 32
                        sub     r1,#21
                        jc      TOn_finish
                        mov     r0,#20                  ;then col = 32
TOn_finish              sto     r0,@ra:rb
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
RCL                     mov     r0,r4                   ;change data channel
                        and     r0,#8                   ;to caption
```

```
                    shr     r0
                    shr     r0
                    shr     r0
                    and     r2,#0fc
                    or      r2,r0

                    mov     ra,#info                ;set style to pop on
                    add     ra,r2
                    mov     rb,#style
                    mov     r0,#pop_on
                    sto     r0,@ra:rb

                    mov     rc,#wait_capt_data_H    ;wait data
                    mov     rd,#wait_capt_data_L
                    jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
BS                  mov     r0,r4                   ;change data channel
                    and     r0,#8
                    shr     r0
                    shr     r0
                    shr     r0
                    and     r2,#0fe
                    or      r2,r0

                    mov     ra,#info                ;if column = 1
                    add     ra,r2
                    mov     rb,#column
                    lod     r9,@ra:rb
                    mov     r0,r9
                    xor     r0,#1
                    jnz     BS_execute
                    mov     rc,#wait_capt_data_H    ;then wait data
                    mov     rd,#wait_capt_data_L
                    jmp     @rc:rd

BS_execute          mov     rb,#row                 ;else
                    lod     r8,@ra:rb               ;offset =
                    mov     r0,#0                   ;row*128 + col*2
                    shr     r8
                    src     r0
                    shl     r9
                    add     r9,r0

                    mov     rb,#style               ;if style = pop on
                    lod     r0,@ra:rb
                    xor     r0,#pop_on
                    jnz     BS_disp_mem
                    mov       rb,#non_disp_mem          ;then address =
offset
                    lod     r0,@ra:rb               ;      + non disp mem
                    add     r8,r0
                    jmp     BS_print_blank
BS_disp_mem           mov       rb,#disp_mem            ;else address =
offset
                    lod     r0,@ra:rb               ;      + disp mem
                    add     r8,r0

BS_print_blank      mov     r0,#0                   ;write blank
                    sto     r0,@r8:r9
                    add     r9,#1
                    sto     r0,@r8:r9
```

```
                        mov     rb,#column              ;column = column - 1
                        lod     r0,@ra:rb
                        sub     r0,#1
                        sto     r0,@ra:rb

                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DER                     mov     r0,r4                   ;change data channel
                        and     r0,#8
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov     ra,#info                ;offset =
                        add     ra,r2                   ;row*128 + col*2 + 2
                        mov     rb,#row
                        lod     r8,@ra:rb
                        mov     rb,#column
                        lod     r9,@ra:rb
                        mov     r0,#0
                        shr     r8
                        src     r0
                        shl     r9
                        add     r9,r0
                        add     r9,#2

                        mov     rb,#style               ;if style = pop on
                        lod     r0,@ra:rb
                        xor     r0,#pop_on
                        jnz     DER_disp_mem
                        mov       rb,#non_disp_mem            ;then address =
offset
                        lod     r0,@ra:rb               ;      + non disp mem
                        add     r8,r0
                        jmp     DER_erase_mem
DER_disp_mem              mov       rb,#disp_mem              ;else address =
offset
                        lod     r0,@ra:rb               ;      + disp mem
                        add     r8,r0

DER_erase_mem           mov     r0,#0                   ;write blank
DER_erase_mem_loop      sto     r0,@r8:r9               ;until end of row
                        add     r9,#1
                        sto     r0,@r8:r9
                        add     r9,#1
                        mov     r1,r9
                        and     r1,#7f
                        xor     r1,#46
                        jnz     DER_erase_mem_loop

                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
RUn                     mov     r0,r4                   ;change data channel
                        and     r0,#8                   ;to caption
```

```
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fc
                        or      r2,r0

                        mov     ra,#info                ;if style /= roll up
                        add     ra,r2
                        mov     rb,#style
                        lod     r0,@ra:rb
                        xor     r0,#roll_up
                        jz      RUn_roll_up
                        mov     rb,#disp_mem            ;then
                        lod     r8,@ra:rb               ;erase disp mem
                        mov     r9,#80
                        mov     r0,#0
                        mov     rc,#0f
RUn_erase_disp_row      mov     rd,#46
RUn_erase_disp_column   sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     RUn_erase_disp_column
                        add     r9,#3a
                        adc     r8,#0
                        sub     rc,#1
                        jnz     RUn_erase_disp_row
                        mov      r0,#roll_up              ;set style to roll
up
                        mov     rb,#style
                        sto     r0,@ra:rb
                        mov     r0,#0f                  ;set cursor row = 15
                        mov     rb,#row
                        sto     r0,@ra:rb
                        mov     r0,#7                   ;set attr = white
                        mov     rb,#attribute
                        sto     r0,@ra:rb
                        mov     rb,#disp_mem            ;address =
                        lod     r8,@ra:rb               ;15*128 + disp mem
                        add     r8,#7
                        mov     r9,#80
                        mov     r0,#2                   ;write PAC
                        sto     r0,@r8:r9
                        add     r9,#1
                        mov     r0,#7
                        sto     r0,@r8:r9
                        jmp     RUn_finish

RUn_roll_up             mov     r0,r5                   ;if new height <
                        sub     r0,#0a3                 ;   old height
                        mov     rb,#height
                        lod     rc,@ra:rb
                        mov     rd,rc
                        sub     rc,r0
                        jz      RUn_finish
                        jc      RUn_finish
                        mov     rb,#row                 ;then
                        lod     r8,@ra:rb               ;if top row is
                        add     r8,#1                   ;   out of screen
                        sub     r8,rd
                        jz      RUn_roll_up_erase_skip
                        jc      RUn_roll_up_erase_skip
```

```
                        jmp     RUn_roll_up_erase
RUn_roll_up_erase_skip  add     rc,r8                   ;then reduce counter
                        sub     rc,#1                   ;      top row = 1
                        mov     r8,#1
RUn_roll_up_erase       mov     r0,rc                   ;if counter > 0
                        sub     r0,#1
                        and     r0,#80
                        jnz     RUn_finish
                        mov     rb,#disp_mem            ;then erase row
                        lod     r0,@ra:rb               ;      that higher
                        mov      r9,#0                        ;      than new
height
                        shr     r8
                        src     r9
                        add     r8,r0
                        mov     r0,#0
RUn_roll_up_erase_row   mov     rd,#46
RUn_roll_up_erase_col   sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     RUn_roll_up_erase_col
                        add     r9,#3a
                        adc     r8,#0
                        sub     rc,#1
                        jnz     RUn_roll_up_erase_row

RUn_finish              mov     r0,#1                   ;column = 1
                        mov     rb,#column
                        sto     r0,@ra:rb
                        mov      r0,r5                    ;height = byte 2 -
23
                        mov     rb,#height
                        sub     r0,#0a3
                        sto     r0,@ra:rb
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
FON                     mov     r0,r4                   ;change data channel
                        and     r0,#8
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov     ra,#info                ;add flash to attr
                        add     ra,r2
                        mov     rb,#attribute
                        lod     r1,@ra:rb
                        or      r1,#20
                        sto     r1,@ra:rb               ;set attribute

                        mov     rb,#row                 ;offset =
                        lod     r8,@ra:rb               ;row*128 + col*2 + 2
                        mov     rb,#column
                        lod     r9,@ra:rb
                        mov     r0,#0
                        shr     r8
                        src     r0
                        shl     r9
```

```
                        add     r9,r0
                        add     r9,#2

                        mov     rb,#style               ;if style = pop on
                        lod     r0,@ra:rb
                        xor     r0,#pop_on
                        jnz     FON_disp_mem
                        mov       rb,#non_disp_mem            ;then address =
offset
                        lod     r0,@ra:rb               ;       + non disp mem
                        add     r8,r0
                        jmp     FON_print
FON_disp_mem              mov       rb,#disp_mem              ;else address =
offset
                        lod     r0,@ra:rb               ;       + disp mem
                        add     r8,r0

FON_print               mov     r0,#3                   ;write FON
                        sto     r0,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9

                        sub     r9,#3                   ;if left is empty
                        lod     r0,@r8:r9
                        xor     r0,#0
                        jnz     FON_right_legible_spc
                        mov     r0,#legible_space       ;then write
                        sto     r0,@r8:r9               ;       legible space
                        add     r9,#1                   ;       at left of FON
                        mov     r0,#0
                        sto     r0,@r8:r9
                        sub     r9,#1

FON_right_legible_spc   add     r9,#4                   ;if right is empty
                        lod     r0,@r8:r9
                        xor     r0,#0
                        jnz     FON_step_cursor
                        mov     r0,#legible_space       ;then write
                        sto     r0,@r8:r9               ;       legible space
                        add      r9,#1                     ;       at right of
FON
                        mov     r0,#0
                        sto     r0,@r8:r9

FON_step_cursor         mov     rb,#column              ;col = col + 1
                        lod     r0,@ra:rb
                        add     r0,#1
                        mov     r1,r0                   ;if col = 33
                        xor     r1,#21
                        jnz     FON_finish
                        mov     r0,#20                  ;then col = 32
FON_finish              sto     r0,@ra:rb
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
RDC                     mov     r0,r4                   ;change data channel
                        and     r0,#8                   ;to caption
                        shr     r0
                        shr     r0
                        shr     r0
```

```
                        and     r2,#0fc
                        or      r2,r0

                        mov     ra,#info                ;set style to paint
on
                        add     ra,r2
                        mov     rb,#style
                        mov     r0,#paint_on
                        sto     r0,@ra:rb

                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
TR                      mov     r0,r4                   ;change data channel
                        and     r0,#8                   ;to text
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fc
                        or      r2,r0
                        or      r2,#2

                        mov     ra,#info                ;write black box
                        add     ra,r2                   ;to text memory
                        mov     rb,#disp_mem
                        lod     r8,@ra:rb
                        mov     r9,#80
                        mov     r0,#legible_space
                        mov     r1,#0
                        mov     rc,#0f
TR_box_row_loop         sto     r1,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9
                        add     r9,#1
                        mov     rd,#22
TR_box_column_loop      sto     r0,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     TR_box_column_loop
                        add     r9,#3a
                        adc     r8,#0
                        sub     rc,#1
                        jnz     TR_box_row_loop

                        mov     r0,#1                   ;row = 1
                        mov     rb,#row
                        sto     r0,@ra:rb
                        mov     rb,#column              ;column = 1
                        sto     r0,@ra:rb
                        mov     r1,#7                   ;attribute = white
                        mov     rb,#attribute
                        sto     r1,@ra:rb

                        mov     rb,#disp_mem            ;write PAC
                        lod     r8,@ra:rb
                        mov     r9,#80
                        mov     r0,#2
                        sto     r0,@r8:r9
```

```
                        add     r9,#1
                        sto     r1,@r8:r9

                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
RTD                     mov     r0,r4                   ;change data channel
                        and     r0,#8                   ;to text
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fc
                        or      r2,r0
                        or      r2,#2

                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
EDM                     mov     ra,r4                   ;erase caption
                        and     ra,#8                   ;display memory
                        shr     ra
                        shr     ra
                        shr     ra
                        add     ra,#info
                        mov     rb,#disp_mem
                        lod     r8,@ra:rb
                        mov     r9,#80
                        mov     r0,#0
                        mov     rc,#0f
EDM_erase_row_loop      mov     rd,#46
EDM_erase_column_loop   sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     EDM_erase_column_loop
                        add     r9,#3a
                        adc     r8,#0
                        sub     rc,#1
                        jnz     EDM_erase_row_loop
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
CR                      mov     r0,r4                   ;change data channel
                        and     r0,#8
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fe
                        or      r2,r0

                        mov     r0,r2                   ;if data channel
                        and     r0,#2                   ;   = text
                        jz      CR_caption
                        mov     ra,#info                ;then
                        add     ra,r2                   ;if row = 15
                        mov     rb,#row
                        lod     r0,@ra:rb
                        xor     r0,#0f
                        jz      CR_move_up              ;then move up 1 row
```

```
                        xor     r0,#0f                  ;else row = row +1
                        add     r0,#1
                        sto     r0,@ra:rb
                        jmp     CR_finish

CR_caption              mov     ra,#info                ;else if style /=
                        add     ra,r2                   ;        roll up
                        mov     rb,#style
                        lod     r0,@ra:rb
                        xor     r0,#roll_up
                        jz      CR_move_up
                        mov     rc,#wait_capt_data_H    ;then wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd

CR_move_up              mov     r0,r2                   ;if data channel =
                        xor     r0,r3                   ;   disp channel
                        jnz     CR_move_up_check_des
                        mov     r8,#mem_map             ;then smooth scroll
                        mov     r9,#osd_scroll
                        sto     r0,@r8:r9

CR_move_up_check_des    mov     rb,#height              ;if top row des is
                        lod     rc,@ra:rb               ;   out of screen
                        sub     rc,#1
                        mov     rb,#row
                        lod     r8,@ra:rb
                        sub     r8,rc
                        jz      CR_move_up_skip
                        jc      CR_move_up_skip
                        jmp     CR_move_up_move
CR_move_up_skip         add     rc,r8                   ;then reduce counter
                        sub     rc,#1
                        mov     r8,#1                    ;      top row des =
1

CR_move_up_move         mov     rb,#disp_mem            ;move up 1 row
                        lod     r0,@ra:rb
                        mov     ra,r8
                        add     ra,#1
                        mov     r9,#0
                        shr     r8
                        src     r9
                        add     r8,r0
                        mov     rb,#0
                        shr     ra
                        src     rb
                        add     ra,r0
CR_move_up_row_loop     sub     rc,#1
                        jc      CR_base_row
                        mov     rd,#46
CR_move_up_column_loop  lod     r0,@ra:rb
                        add     rb,#1
                        sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     CR_move_up_column_loop
                        add     rb,#3a
                        adc     ra,#0
                        add     r9,#3a
                        adc     r8,#0
```

```
                        jmp     CR_move_up_row_loop

CR_base_row             mov     r0,r2                   ;if data channel
                        and     r0,#2                   ;   = text
                        jz      CR_base_row_erase
                        mov      r0,#legible_space          ;then write black
box
                        mov     r1,#0                   ;      in base row
                        sto     r1,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9
                        add     r9,#1
                        mov     rd,#22
CR_base_row_box_loop    sto     r0,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     CR_base_row_box_loop
                        jmp     CR_finish
CR_base_row_erase       mov     rd,#46                  ;else clear base row
                        mov     r0,#0
CR_base_row_erase_loop  sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     CR_base_row_erase_loop

CR_finish               mov     r0,#1                   ;set column = 1
                        mov     ra,#info
                        add     ra,r2
                        mov     rb,#column
                        sto     r0,@ra:rb
                        mov      r1,#7                      ;set attribute =
white
                        mov     rb,#attribute
                        sto     r1,@ra:rb
                        mov     rb,#disp_mem            ;write PAC
                        lod     r0,@ra:rb
                        mov     rb,#row
                        lod     r8,@ra:rb
                        mov     r9,#0
                        shr     r8
                        src     r9
                        add     r8,r0
                        mov     r0,#2
                        sto     r0,@r8:r9
                        add     r9,#1
                        sto     r1,@r8:r9
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
ENM                     mov     ra,r4                   ;erase caption
                        and     ra,#8                   ;non display memory
                        shr     ra
                        shr     ra
                        shr     ra
                        add     ra,#info
                        mov     rb,#non_disp_mem
                        lod     r8,@ra:rb
                        mov     r9,#80
```

```
                        mov     r0,#0
                        mov     rc,#0f
ENM_erase_row_loop      mov     rd,#46
ENM_erase_column_loop   sto     r0,@r8:r9
                        add     r9,#1
                        sub     rd,#1
                        jnz     ENM_erase_column_loop
                        add     r9,#3a
                        adc     r8,#0
                        sub     rc,#1
                        jnz     ENM_erase_row_loop
                        mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
EOC                     mov     r0,r4                   ;change data channel
                        and     r0,#8                   ;to caption
                        shr     r0
                        shr     r0
                        shr     r0
                        and     r2,#0fc
                        or      r2,r0

                        mov     ra,#info                ;set style to pop on
                        add     ra,r2
                        mov     rb,#style
                        mov     r0,#pop_on
                        sto     r0,@ra:rb

                        mov     rb,#disp_mem            ;swap disp mem
                        lod     r0,@ra:rb               ;with non disp mem
                        mov     rb,#non_disp_mem
                        lod     r1,@ra:rb
                        sto     r0,@ra:rb
                        mov     rb,#disp_mem
                        sto     r1,@ra:rb

                        mov     r0,r2                   ;if disp channel =
                        xor     r0,r3                   ;   data channel
                        jnz     EOC_finish
                        shr     r1                          ;then change disp
page
                        shr     r1
                        shr     r1
                        and     r1,#7
                        mov     ra,#mem_map
                        mov     rb,#osd_page
                        sto     r1,@ra:rb

EOC_finish              mov     rc,#wait_capt_data_H    ;wait data
                        mov     rd,#wait_capt_data_L
                        jmp     @rc:rd
;*******************************************************
start_info_table        db      pop_on                  ;CC 1
                        db      0                       ;English
                        db      1
                        db      1
                        db      7
                        db      4
                        db      80
                        db      88
```

```
                db      pop_on                  ;CC 2
                db      80                      ;Thai
                db      1
                db      1
                db      7
                db      4
                db      90
                db      98
                db      roll_up                 ;TXT 1
                db      0                       ;English
                db      1
                db      1
                db      7
                db      0f
                db      0a0
                db      0a8
                db      roll_up                 ;TXT 2
                db      80                      ;Thai
                db      1
                db      1
                db      7
                db      0f
                db      0b0
                db      0b8

PAC_row_table   db      0b                      ;10,18/4x,5x
                db      0                       ;10,18/6x,7x
                db      1                       ;11,19/4x,5x
                db      2                       ;11,19/6x,7x
                db      3                       ;12,1a/4x,5x
                db      4                       ;12,1a/6x,7x
                db      0c                      ;13,1b/4x,5x
                db      0d                      ;13,1b/6x,7x
                db      0e                      ;14,1c/4x,5x
                db      0f                      ;14,1c/6x,7x
                db      5                       ;15,1d/4x,5x
                db      6                       ;15,1d/6x,7x
                db      7                       ;16,1e/4x,5x
                db      8                       ;16,1e/6x,7x
                db      9                       ;17,1f/4x,5x
                db      0a                      ;17,1f/6x,7x

PAC_column_table db     1                       ;50,51,70,71
                db      5                       ;52,53,72,73
                db      9                       ;54,55,74,75
                db      0d                      ;56,57,76,77
                db      11                      ;58,59,78,79
                db      15                      ;5a,5b,7a,7b
                db      19                      ;5c,5d,7c,7d
                db      1d                      ;5e,5f,7e,7f

PAC_attr_table  db      7                       ;40,41,60,61
                db      2                       ;42,43,62,63
                db      1                       ;44,45,64,65
                db      3                       ;46,47,66,67
                db      4                       ;48,49,68,69
                db      6                       ;4a,4b,6a,6b
                db      5                       ;4c,4d,6c,6d
                db      0f                      ;4e,4f,6e,6f

MID_attr_table  db      7                       ;11,19/20,21
```

```
                    db      2                       ;11,19/22,23
                    db      1                       ;11,19/24,25
                    db      3                       ;11,19/26,27
                    db      4                       ;11,19/28,29
                    db      6                       ;11,19/2a,2b
                    db      5                       ;11,19/2c,2d
                    db      8                       ;11,19/2e,2f

spec_char_table     db      10                      ;11,19/30
                    db      11                      ;11,19/31
                    db      12                      ;11,19/32
                    db      13                      ;11,19/33
                    db      14                      ;11,19/34
                    db      15                      ;11,19/35
                    db      16                      ;11,19/36
                    db      17                      ;11,19/37
                    db      18                      ;11,19/38
                    db      08                      ;11,19/39
                    db      1a                      ;11,19/3a
                    db      1b                      ;11,19/3b
                    db      1c                      ;11,19/3c
                    db      1d                      ;11,19/3d
                    db      1e                      ;11,19/3e
                    db      1f                      ;11,19/3f

MISC_func_table     db      RCL_H                   ;14,1c/20
                    db      RCL_L
                    db      BS_H                    ;14,1c/21
                    db      BS_L
                    db      wait_capt_data_H        ;14,1c/22
                    db      wait_capt_data_L
                    db      wait_capt_data_H        ;14,1c/23
                    db      wait_capt_data_L
                    db      DER_H                   ;14,1c/24
                    db      DER_L
                    db      RUn_H                   ;14,1c/25
                    db      RUn_L
                    db      RUn_H                   ;14,1c/26
                    db      RUn_L
                    db      RUn_H                   ;14,1c/27
                    db      RUn_L
                    db      FON_H                   ;14,1c/28
                    db      FON_L
                    db      RDC_H                   ;14,1c/29
                    db      RDC_L
                    db      TR_H                    ;14,1c/2a
                    db      TR_L
                    db      RTD_H                   ;14,1c/2b
                    db      RTD_L
                    db      EDM_H                   ;14,1c/2c
                    db      EDM_L
                    db      CR_H                    ;14,1c/2d
                    db      CR_L
                    db      ENM_H                   ;14,1c/2e
                    db      ENM_L
                    db      EOC_H                   ;14,1c/2f
                    db      EOC_L
```

ภาคผนวก ค

รายละเอียดของโปรแกรมภาษา VHDL ที่ใช้ออกแบบตัวประมวลผล

```
ENTITY cpu IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    halt         :IN     vlbit;                    --Active high
    din          :IN     vlbit_1d(7 downto 0);
    dout         :OUT    vlbit_1d(7 downto 0);
    addr         :OUT    vlbit_1d(15 downto 0);
    rd           :OUT    vlbit;                    --Active low
    wr           :OUT    vlbit                     --Active low
  );
END cpu;

ARCHITECTURE struct OF cpu IS

  COMPONENT condecu
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
      halt       :IN     vlbit;
      inst       :IN     vlbit_1d(15 downto 0);
      cin        :IN     vlbit;
      zin        :IN     vlbit;
      rd         :OUT    vlbit;
      wr         :OUT    vlbit;
      irfeth     :OUT    vlbit;
      irfetl     :OUT    vlbit;
      iry        :OUT    vlbit_1d(1 downto 0);
      rfw        :OUT    vlbit_1d(4 downto 0);
      rfx        :OUT    vlbit_1d(4 downto 0);
      rfy        :OUT    vlbit_1d(4 downto 0);
      auoper     :OUT    vlbit_1d(4 downto 0);
      aden       :OUT    vlbit;
      adlat      :OUT    vlbit;
      dien       :OUT    vlbit;
      dolat      :OUT    vlbit
    );
  END COMPONENT;

  COMPONENT regfile
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
      wen        :IN     vlbit;
      wnum       :IN     vlbit_1d(3 downto 0);
      win        :IN     vlbit_1d(7 downto 0);
      xen        :IN     vlbit;
      xnum       :IN     vlbit_1d(3 downto 0);
      xout       :OUT    vlbit_1d(7 downto 0);
      yen        :IN     vlbit;
      ynum       :IN     vlbit_1d(3 downto 0);
      yout       :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

  COMPONENT alsu
    PORT (
      ren        :IN     vlbit;
      oper       :IN     vlbit_1d(3 downto 0);
      xin        :IN     vlbit_1d(7 downto 0);
      yin        :IN     vlbit_1d(7 downto 0);
```

```
      rout        :OUT    vlbit_1d(7 downto 0);
      cout        :OUT    vlbit;
      zout        :OUT    vlbit
    );
  END COMPONENT;

  COMPONENT instreg
    PORT (
      rst         :IN     vlbit;
      clk         :IN     vlbit;
      feth        :IN     vlbit;
      fetl        :IN     vlbit;
      yen         :IN     vlbit;
      ynum        :IN     vlbit;
      din         :IN     vlbit_1d(7 downto 0);
      inst        :OUT    vlbit_1d(15 downto 0);
      yout        :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

  COMPONENT addrreg
    PORT (
      rst         :IN     vlbit;
      clk         :IN     vlbit;
      en          :IN     vlbit;
      lat         :IN     vlbit;
      xin         :IN     vlbit_1d(7 downto 0);
      yin         :IN     vlbit_1d(7 downto 0);
      addr        :OUT    vlbit_1d(15 downto 0)
    );
  END COMPONENT;

  COMPONENT dinbuf
    PORT (
      en          :IN     vlbit;
      din         :IN     vlbit_1d(7 downto 0);
      rout        :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

  COMPONENT doutreg
    PORT (
      rst         :IN     vlbit;
      clk         :IN     vlbit;
      lat         :IN     vlbit;
      xin         :IN     vlbit_1d(7 downto 0);
      dout        :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

SIGNAL  inst_sig          :vlbit_1d(15 downto 0);
SIGNAL  cin_sig           :vlbit;
SIGNAL  zin_sig           :vlbit;
SIGNAL  irfeth_sig        :vlbit;
SIGNAL  irfetl_sig        :vlbit;
SIGNAL  iry_sig           :vlbit_1d(1 downto 0);
SIGNAL  rfw_sig           :vlbit_1d(4 downto 0);
SIGNAL  rfx_sig           :vlbit_1d(4 downto 0);
SIGNAL  rfy_sig           :vlbit_1d(4 downto 0);
SIGNAL  auoper_sig        :vlbit_1d(4 downto 0);
SIGNAL  aden_sig          :vlbit;
```

```
SIGNAL  adlat_sig       :vlbit;
SIGNAL  dien_sig        :vlbit;
SIGNAL  dolat_sig       :vlbit;
SIGNAL  xbus            :vlbit_1d(7 downto 0);
SIGNAL  ybus            :vlbit_1d(7 downto 0);
SIGNAL  rbus            :vlbit_1d(7 downto 0);

BEGIN

 u1:   condecu
 PORT MAP (
   rst          => rst,
   clk          => clk,
   halt         => halt,
   inst         => inst_sig,
   cin          => cin_sig,
   zin          => zin_sig,
   rd           => rd,
   wr           => wr,
   irfeth       => irfeth_sig,
   irfetl       => irfetl_sig,
   iry          => iry_sig,
   rfw          => rfw_sig,
   rfx          => rfx_sig,
   rfy          => rfy_sig,
   auoper       => auoper_sig,
   aden         => aden_sig,
   adlat        => adlat_sig,
   dien         => dien_sig,
   dolat        => dolat_sig
 );

 u2:   regfile
 PORT MAP (
   rst          => rst,
   clk          => clk,
   wen          => rfw_sig(4),
   wnum         => rfw_sig(3 downto 0),
   win          => rbus,
   xen          => rfx_sig(4),
   xnum         => rfx_sig(3 downto 0),
   xout         => xbus,
   yen          => rfy_sig(4),
   ynum         => rfy_sig(3 downto 0),
   yout         => ybus
 );

 u3:   alsu
 PORT MAP (
   ren          => auoper_sig(4),
   oper         => auoper_sig(3 downto 0),
   xin          => xbus,
   yin          => ybus,
   rout         => rbus,
   cout         => cin_sig,
   zout         => zin_sig
 );

 u4:   instreg
 PORT MAP (
   rst          => rst,
```

```
    clk          => clk,
    feth         => irfeth_sig,
    fetl         => irfetl_sig,
    yen          => iry_sig(1),
    ynum         => iry_sig(0),
    din          => din,
    inst         => inst_sig,
    yout         => ybus
  );

  u5:   addrreg
  PORT MAP (
    rst          => rst,
    clk          => clk,
    en           => aden_sig,
    lat          => adlat_sig,
    xin          => xbus,
    yin          => ybus,
    addr         => addr
  );

  u6:   dinbuf
  PORT MAP (
    en           => dien_sig,
    din          => din,
    rout         => rbus
  );

  u7:   doutreg
  PORT MAP (
    rst          => rst,
    clk          => clk,
    lat          => dolat_sig,
    xin          => xbus,
    dout         => dout
  );

END struct;
```

```
ENTITY condecu IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    halt         :IN     vlbit;                    --Active high
    inst         :IN     vlbit_1d(15 downto 0);
    cin          :IN     vlbit;                    --Active high
    zin          :IN     vlbit;                    --Active high
    rd           :OUT    vlbit;                    --Active low
    wr           :OUT    vlbit;                    --Active low
    irfeth       :OUT    vlbit;                    --Active high
    irfetl       :OUT    vlbit;                    --Active high
    iry          :OUT    vlbit_1d(1 downto 0);     --"1X"=Read to Y bus
    rfw          :OUT    vlbit_1d(4 downto 0);     --"1XXXX"=Write from R bus
    rfx          :OUT    vlbit_1d(4 downto 0);     --"1XXXX"=Read to X bus
    rfy          :OUT    vlbit_1d(4 downto 0);     --"1XXXX"=Read to Y bus
    auoper       :OUT    vlbit_1d(4 downto 0);     --"1XXXX"=ALS Op. to R bus
    aden         :OUT    vlbit;                    --Active high
    adlat        :OUT    vlbit;                    --Active high
    dien         :OUT    vlbit;                    --Active high
    dolat        :OUT    vlbit                     --Active high
  );
END condecu;

ARCHITECTURE behave OF condecu IS
TYPE state_type IS (RESET,HIGHZ,
                    FET1,FET2,FET3,FET4,FET5,FET6,
                    ALSOP,
                    LDST1,LDST2,LDST3,
                    JABS1,JABS2,
                    JREL1,JREL2);
SIGNAL  state            :state_type;
SIGNAL  rst_ctr          :vlbit_1d(5 downto 0);
SIGNAL  cflag            :vlbit;
SIGNAL  zflag            :vlbit;
SIGNAL  iflag            :vlbit;
BEGIN
--=======================================state PROCESS
  PROCESS       --State transition
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      state<=RESET;
    ELSE
      CASE state IS
       WHEN RESET =>   IF (rst_ctr(4 downto 0)/=B"11111") THEN
                         state<=RESET;
                       ELSE
                         state<=HIGHZ;
                       END IF;

       WHEN HIGHZ =>   IF (halt='1') THEN
                         state<=HIGHZ;
                       ELSE
                         state<=FET1;
                       END IF;

       WHEN FET1  =>   state<=FET2;
       WHEN FET2  =>   state<=FET3;
       WHEN FET3  =>   state<=FET4;
       WHEN FET4  =>   state<=FET5;
```

```
WHEN FET5  =>   state<=FET6;
WHEN FET6  =>   IF (inst(15)='0') THEN
                  state<=ALSOP;
                ELSE
                  IF (inst(14)='0') THEN
                    IF (inst(13)='0') THEN
                      state<=ALSOP;
                    ELSE
                      IF (inst(12)='0') THEN
                       state<=ALSOP;
                      ELSE
                       state<=JABS1;
                      END IF;
                    END IF;
                  ELSE
                    IF (inst(13)='0') THEN
                      state<=LDST1;
                    ELSE
                      IF (inst(12)='0') THEN
                       IF (inst(11)='0') THEN
                         IF (inst(10)=cflag) THEN
                           state<=JREL1;
                         ELSE
                           IF (halt='1') THEN
                             state<=HIGHZ;
                           ELSE
                             state<=FET1;
                           END IF;
                         END IF;
                       ELSE
                         IF (inst(10)=zflag) THEN
                           state<=JREL1;
                         ELSE
                           IF (halt='1') THEN
                             state<=HIGHZ;
                           ELSE
                             state<=FET1;
                           END IF;
                         END IF;
                       END IF;
                      ELSE
                       state<=JREL1;
                      END IF;
                    END IF;
                  END IF;
                END IF;

WHEN ALSOP =>   IF (halt='1') THEN
                  state<=HIGHZ;
                ELSE
                  state<=FET1;
                END IF;

WHEN LDST1 =>   state<=LDST2;
WHEN LDST2 =>   state<=LDST3;
WHEN LDST3 =>   IF (halt='1') THEN
                  state<=HIGHZ;
                ELSE
                  state<=FET1;
                END IF;
```

```
        WHEN JABS1 =>    state<=JABS2;
        WHEN JABS2 =>    IF (halt='1') THEN
                         state<=HIGHZ;
                       ELSE
                         state<=FET1;
                       END IF;

        WHEN JREL1 =>    state<=JREL2;
        WHEN JREL2 =>    IF (halt='1') THEN
                         state<=HIGHZ;
                       ELSE
                         state<=FET1;
                       END IF;
      END CASE;
    END IF;
  END PROCESS;
--=======================================rst_ctr PROCESS
  PROCESS       --Count from 0 to 31 when reset to clear all register
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      rst_ctr<="000000";
    ELSE
      rst_ctr<=ADDUM(rst_ctr(4 downto 0),"00001");
    END IF;
  END PROCESS;
--=======================================cflag PROCESS
  PROCESS       --Save carry flag when state=ALSOP
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      cflag<='0';
    ELSE
      IF (state=ALSOP) THEN
       IF (inst(14)='1') THEN
         cflag<=cin;
       ELSE
         IF (inst(15)='1') THEN
           IF (inst(13 downto 12)/=B"00") THEN
             cflag<=cin;
           END IF;
         END IF;
       END IF;
      END IF;
    END IF;
  END PROCESS;
--=======================================zflag PROCESS
  PROCESS       --Save zero flag when state=ALSOP
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      zflag<='0';
    ELSE
      IF (state=ALSOP) THEN
       IF (inst(15)='0') THEN
         IF (inst(14 downto 12)/=B"000") THEN
           zflag<=zin;
         END IF;
       ELSE
         IF (inst(13 downto 10)/=B"0000") THEN
           zflag<=zin;
```

```
        END IF;
      END IF;
     END IF;
   END IF;
  END PROCESS;
--=======================================iflag PROCESS
  PROCESS       --Save internal carry flag used for increasing IP
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      iflag<='0';
    ELSE
      CASE state IS
       WHEN FET2  =>   iflag<=cin;
       WHEN FET5  =>   iflag<=cin;
       WHEN JREL1 =>   iflag<=cin;
      END CASE;
    END IF;
  END PROCESS;
--=======================================Outout of ENTITY
  rd <= 'Z' WHEN state=RESET                          ELSE
       'Z' WHEN state=HIGHZ                          ELSE
       '0' WHEN state=FET2                           ELSE
       '0' WHEN state=FET3                           ELSE
       '0' WHEN state=FET5                           ELSE
       '0' WHEN state=FET6                           ELSE
       '0' WHEN ((state=LDST2) AND (inst(12)='0')) ELSE
       '0' WHEN ((state=LDST3) AND (inst(12)='0')) ELSE '1';

  wr <= '0' WHEN ((state=LDST2) AND (inst(12)='1')) ELSE
       '0' WHEN ((state=LDST3) AND (inst(12)='1')) ELSE '1';

  irfeth <= '1' WHEN state=FET3 ELSE '0';

  irfetl <= '1' WHEN state=FET6 ELSE '0';

  iry <= "10" WHEN ((state=ALSOP) AND (inst(15)='0')) ELSE
        "10" WHEN state=JREL1                          ELSE
        "11" WHEN state=JREL2                          ELSE "0X";

  rfw <= "1"&rst_ctr(3 downto 0) WHEN state=RESET
ELSE
        "11111"                 WHEN state=FET2                          ELSE
        "11110"                 WHEN state=FET3                          ELSE
        "11111"                 WHEN state=FET5                          ELSE
        "11110"                 WHEN state=FET6                          ELSE
        '1'&inst(11 downto 8)   WHEN ((state=ALSOP) AND (inst(15)='0')) ELSE
        '1'&inst(7 downto 4)    WHEN ((state=ALSOP) AND (inst(15)='1')) ELSE
        '1'&inst(11 downto 8)   WHEN ((state=LDST3) AND (inst(12)='0')) ELSE
        "11111"                 WHEN state=JABS1                         ELSE
        "11110"                 WHEN state=JABS2                         ELSE
        "11111"                 WHEN state=JREL1                         ELSE
        "11110"                 WHEN state=JREL2                         ELSE
        "0XXXX";

  rfx <= "1"&rst_ctr(3 downto 0) WHEN state=RESET
ELSE
        "11110"                 WHEN state=FET1                          ELSE
        "11110"                 WHEN state=FET4                          ELSE
        '1'&inst(11 downto 8)   WHEN ((state=ALSOP) AND (inst(15)='0')) ELSE
        '1'&inst(7 downto 4)    WHEN ((state=ALSOP) AND (inst(15)='1')) ELSE
```

```
        '1'&inst(7 downto 4)     WHEN state=LDST1                              ELSE
        '1'&inst(11 downto 8)    WHEN ((state=LDST2) AND (inst(12)='1')) ELSE
        "11111"                  WHEN state=JREL1                              ELSE
        "11110"                  WHEN state=JREL2                              ELSE
        "0XXXX";

  rfy <= "1"&rst_ctr(3 downto 0) WHEN state=RESET
ELSE
        "11111"                  WHEN state=FET1                               ELSE
        "11111"                  WHEN state=FET2                               ELSE
        "11110"                  WHEN state=FET3                               ELSE
        "11111"                  WHEN state=FET4                               ELSE
        "11111"                  WHEN state=FET5                               ELSE
        "11110"                  WHEN state=FET6                               ELSE
        '1'&inst(3 downto 0)     WHEN ((state=ALSOP) AND (inst(15)='1')) ELSE
        '1'&inst(3 downto 0)     WHEN state=LDST1                              ELSE
        '1'&inst(3 downto 0)     WHEN state=JABS1                              ELSE
        '1'&inst(7 downto 4)     WHEN state=JABS2                              ELSE
        "0XXXX";

  auoper <= "10011"                WHEN state=RESET
ELSE
          "11101"                  WHEN state=FET2
ELSE
          "1110"&iflag             WHEN state=FET3
ELSE
          "11101"                  WHEN state=FET5
ELSE
          "1110"&iflag             WHEN state=FET6
ELSE
          "100"&inst(13 downto 12) WHEN ((state=ALSOP) AND (inst(15 downto
14)=B"00"))     ELSE
          "110"&inst(13)&'0'       WHEN ((state=ALSOP) AND (inst(14 downto
12)=B"1X0"))    ELSE
          "110"&inst(13)&cflag     WHEN ((state=ALSOP) AND (inst(14 downto
12)=B"1X1"))    ELSE
          "100"&inst(11 downto 10) WHEN ((state=ALSOP) AND (inst(15 downto
12)=B"1X00"))   ELSE
          "110"&inst(11)&'0'       WHEN ((state=ALSOP) AND (inst(15 downto
10)=B"1XX1X0")) ELSE
          "110"&inst(11)&cflag     WHEN ((state=ALSOP) AND (inst(15 downto
10)=B"1XX1X1")) ELSE
          "101"&inst(11)&'0'       WHEN ((state=ALSOP) AND (inst(15 downto
10)=B"1X1XX0")) ELSE
          "101"&inst(11)&cflag     WHEN ((state=ALSOP) AND (inst(15 downto
10)=B"1X1XX1")) ELSE
          "10000"                  WHEN state=JABS1
ELSE
          "10000"                  WHEN state=JABS2
ELSE
          "11000"                  WHEN state=JREL1
ELSE
          "1100"&iflag             WHEN state=JREL2
ELSE
          "0XXXX";

  aden <= '0' WHEN state=RESET ELSE
         '0' WHEN state=HIGHZ ELSE '1';

  adlat <= '1' WHEN state=FET1  ELSE
           '1' WHEN state=FET4  ELSE
```

```
        '1' WHEN state=LDST1 ELSE '0';

  dien <= '1' WHEN ((state=LDST3) AND (inst(12)='0')) ELSE '0';

  dolat <= '1' WHEN ((state=LDST2) AND (inst(12)='1')) ELSE '0';
END behave;
```

```
ENTITY regfile IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    wen          :IN     vlbit;                    --Active high
    wnum         :IN     vlbit_1d(3 downto 0);
    win          :IN     vlbit_1d(7 downto 0);
    xen          :IN     vlbit;                    --Active high
    xnum         :IN     vlbit_1d(3 downto 0);
    xout         :OUT    vlbit_1d(7 downto 0);
    yen          :IN     vlbit;                    --Active high
    ynum         :IN     vlbit_1d(3 downto 0);
    yout         :OUT    vlbit_1d(7 downto 0)
  );
END regfile;

ARCHITECTURE behave OF regfile IS

  COMPONENT ram16x8b                               --XC4000 Library
    PORT (
      we         :IN     vlbit;
      a          :IN     vlbit_1d(3 downto 0);
      d          :IN     vlbit_1d(7 downto 0);
      o          :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

SIGNAL  we_sig          :vlbit;
SIGNAL  ax_sig          :vlbit_1d(3 downto 0);
SIGNAL  ay_sig          :vlbit_1d(3 downto 0);
SIGNAL  ox_sig          :vlbit_1d(7 downto 0);
SIGNAL  oy_sig          :vlbit_1d(7 downto 0);
SIGNAL  hclk1           :vlbit;
SIGNAL  hclk2           :vlbit;
SIGNAL  dclk            :vlbit;
SIGNAL  hdclk1          :vlbit;
SIGNAL  hdclk2          :vlbit;
SIGNAL  ddclk           :vlbit;
SIGNAL  ph1             :vlbit;
SIGNAL  ph2             :vlbit;
SIGNAL  ox_ff           :vlbit_1d(7 downto 0);
SIGNAL  oy_ff           :vlbit_1d(7 downto 0);

BEGIN
--=======================================Memory PORT MAP
  ux:   ram16x8b
  PORT MAP (
    we  => we_sig,
    a   => ax_sig,
    d   => win,
    o   => ox_sig
  );

  uy:   ram16x8b
  PORT MAP (
    we  => we_sig,
    a   => ay_sig,
    d   => win,
    o   => oy_sig
  );
--=======================================hclk1 PROCESS
```

```
  PROCESS       --Half frequency of clk
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      hclk1<='0';
    ELSE
      hclk1<=NOT hclk1;
    END IF;
  END PROCESS;
--=======================================hclk2 PROCESS
  PROCESS       --Half frequency of clk, delay from hclk1 1/2 period of clk
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(clk));
    IF (rst='0') THEN
      hclk2<='0';
    ELSE
      hclk2<=hclk1;
    END IF;
  END PROCESS;
--=======================================dclk Equation
  dclk <= hclk1 XOR hclk2;
--=======================================hdclk1 PROCESS
  PROCESS       --Half frequency of dclk
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(dclk));
    IF (rst='0') THEN
      hdclk1<='0';
    ELSE
      hdclk1<=NOT hdclk1;
    END IF;
  END PROCESS;
--=======================================hdclk2 PROCESS
  PROCESS       --Half frequency of dclk, delay from hdclk1 1/2 period of
dclk
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(dclk));
    IF (rst='0') THEN
      hdclk2<='0';
    ELSE
      hdclk2<=hdclk1;
    END IF;
  END PROCESS;
--=======================================ddclk, ph1, ph2 Equation
  ddclk <= hdclk1 XOR hdclk2;
  ph1   <= ddclk OR clk;
  ph2   <= ddclk AND clk;
--=======================================ox_ff PROCESS
  PROCESS       --Save reg x value after read
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(clk));
    IF (rst='0') THEN
      ox_ff<="00000000";
    ELSE
      ox_ff<=ox_sig;
    END IF;
  END PROCESS;
--=======================================oy_ff PROCESS
  PROCESS       --Save reg y value after read
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(clk));
    IF (rst='0') THEN
```

```
      oy_ff<="00000000";
    ELSE
      oy_ff<=oy_sig;
    END IF;
  END PROCESS;
--=======================================we_sig, ax_sig, ay_sig Equation
  we_sig <= wen AND NOT ph1;
  ax_sig <= xnum WHEN ph2='1' ELSE wnum;
  ay_sig <= ynum WHEN ph2='1' ELSE wnum;
--=======================================Output of ENTITY
  xout <= ox_ff WHEN xen='1' ELSE "ZZZZZZZZ";
  yout <= oy_ff WHEN yen='1' ELSE "ZZZZZZZZ";
END behave;
```

```
ENTITY alsu IS
  PORT (
    ren          :IN     vlbit;                    --Active high
    oper         :IN     vlbit_1d(3 downto 0);     --"0000"=  Y | "1000"=X+Y+0
    xin          :IN     vlbit_1d(7 downto 0);     --"0001"=X&Y | "1001"=X+Y+1
    yin          :IN     vlbit_1d(7 downto 0);     --"0010"=X|Y | "1010"=X-Y-0
    rout         :OUT    vlbit_1d(7 downto 0);     --"0011"=X^Y | "1011"=X-Y-1
    cout         :OUT    vlbit;                    --"0100"=X:0 | "1100"=  Y+0
    zout         :OUT    vlbit                     --"0101"=X:1 | "1101"=  Y+1
  );                                                 --"0110"=0:X | "1110"=  Y-0
END alsu;                                            --"0111"=1:X | "1111"=  Y-1

ARCHITECTURE struct OF alsu IS

  COMPONENT arithu
    PORT (
      en         :IN     vlbit;
      op         :IN     vlbit_1d(2 downto 0);
      xi         :IN     vlbit_1d(7 downto 0);
      yi         :IN     vlbit_1d(7 downto 0);
      ro         :OUT    vlbit_1d(7 downto 0);
      co         :OUT    vlbit
    );
  END COMPONENT;

  COMPONENT logicu
    PORT (
      en         :IN     vlbit;
      op         :IN     vlbit_1d(1 downto 0);
      xi         :IN     vlbit_1d(7 downto 0);
      yi         :IN     vlbit_1d(7 downto 0);
      ro         :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

  COMPONENT shiftu
    PORT (
      en         :IN     vlbit;
      op         :IN     vlbit_1d(1 downto 0);
      xi         :IN     vlbit_1d(7 downto 0);
      ro         :OUT    vlbit_1d(7 downto 0);
      co         :OUT    vlbit
    );
  END COMPONENT;

SIGNAL  aen_sig          :vlbit;
SIGNAL  len_sig          :vlbit;
SIGNAL  sen_sig          :vlbit;
SIGNAL  ro_sig           :vlbit_1d(7 downto 0);
SIGNAL  aco_sig          :vlbit;
SIGNAL  sco_sig          :vlbit;

BEGIN
  aen_sig <= '1' WHEN (ren='1' AND oper(3)='1')            ELSE '0';
  len_sig <= '1' WHEN (ren='1' AND oper(3 downto 2)=B"00") ELSE '0';
  sen_sig <= '1' WHEN (ren='1' AND oper(3 downto 2)=B"01") ELSE '0';

  au:   arithu
  PORT MAP (
    en  => aen_sig,
    op  => oper(2 downto 0),
```

```
    xi   => xin,
    yi   => yin,
    ro   => ro_sig,
    co   => aco_sig
  );

  lu:   logicu
  PORT MAP (
    en   => len_sig,
    op   => oper(1 downto 0),
    xi   => xin,
    yi   => yin,
    ro   => ro_sig
  );

  su:   shiftu
  PORT MAP (
    en   => sen_sig,
    op   => oper(1 downto 0),
    xi   => xin,
    ro   => ro_sig,
    co   => sco_sig
  );

  rout  <= ro_sig;
  cout  <= aco_sig WHEN aen_sig='1' ELSE
           sco_sig WHEN sen_sig='1' ELSE 'X';
  zout  <= '1' WHEN (ren='1' AND ro_sig=B"00000000")  ELSE
           '0' WHEN (ren='1' AND ro_sig/=B"00000000") ELSE 'X';
END struct;
```

```
ENTITY arithu IS
  PORT (
    en          :IN     vlbit;                    --Active high
    op          :IN     vlbit_1d(2 downto 0);     --"000"=X+Y+0 | "100"=Y+0
    xi          :IN     vlbit_1d(7 downto 0);     --"001"=X+Y+1 | "101"=Y+1
    yi          :IN     vlbit_1d(7 downto 0);     --"010"=X-Y-0 | "110"=Y-0
    ro          :OUT    vlbit_1d(7 downto 0);     --"011"=X-Y-1 | "111"=Y-1
    co          :OUT    vlbit                     --Active high
  );
END arithu;

ARCHITECTURE behave OF arithu IS

  COMPONENT addsub8b                              --XC4000 Library
    PORT (
      add       :IN     vlbit;
      a         :IN     vlbit_1d(7 downto 0);
      b         :IN     vlbit_1d(7 downto 0);
      ci        :IN     vlbit;
      s         :OUT    vlbit_1d(7 downto 0);
      co        :OUT    vlbit;
      ofl       :OUT    vlbit
    );
  END COMPONENT;

SIGNAL  add_sig         :vlbit;
SIGNAL  a_sig           :vlbit_1d(7 downto 0);
SIGNAL  b_sig           :vlbit_1d(7 downto 0);
SIGNAL  ci_sig          :vlbit;
SIGNAL  s_sig           :vlbit_1d(7 downto 0);
SIGNAL  co_sig          :vlbit;

BEGIN
  add_sig <= NOT op(1);
  a_sig   <= xi WHEN op(2)='0' ELSE yi;
  b_sig   <= yi WHEN op(2)='0' ELSE "00000000";
  ci_sig  <= op(1) XOR op(0);

  u1:   addsub8b
  PORT MAP (
    add => add_sig,
    a   => a_sig,
    b   => b_sig,
    ci  => ci_sig,
    s   => s_sig,
    co  => co_sig
  );

  ro    <= s_sig WHEN en='1' ELSE "ZZZZZZZZ";
  co    <= op(1) XOR co_sig;
END behave;
```

```
ENTITY logicu IS
  PORT (
    en          :IN     vlbit;                  --Active high
    op          :IN     vlbit_1d(1 downto 0);   --"00"=      Y
    xi          :IN     vlbit_1d(7 downto 0);   --"01"=X AND Y
    yi          :IN     vlbit_1d(7 downto 0);   --"10"=X OR  Y
    ro          :OUT    vlbit_1d(7 downto 0)    --"11"=X XOR Y
  );
END logicu;

ARCHITECTURE behave OF logicu IS
SIGNAL  ro_sig          :vlbit_1d(7 downto 0);
BEGIN
  ro_sig <=        yi WHEN op=B"00" ELSE
            xi AND yi WHEN op=B"01" ELSE
            xi OR  yi WHEN op=B"10" ELSE
            xi XOR yi;
  ro     <= ro_sig    WHEN en='1'   ELSE "ZZZZZZZZ";
END behave;
```

```
ENTITY shiftu IS
  PORT (
    en          :IN     vlbit;                  --Active high
    op          :IN     vlbit_1d(1 downto 0);   --"00"=X SHL '0'
    xi          :IN     vlbit_1d(7 downto 0);   --"01"=X SHL '1'
    ro          :OUT    vlbit_1d(7 downto 0);   --"10"=X SHR '0'
    co          :OUT    vlbit                   --"11"=X SHR '1'
  );
END shiftu;

ARCHITECTURE behave OF shiftu IS
SIGNAL  ro_sig          :vlbit_1d(7 downto 0);
BEGIN
  ro_sig <= xi(6 downto 0)&op(0) WHEN op(1)='0' ELSE op(0)&xi(7 downto 1);
  ro     <= ro_sig               WHEN en='1'    ELSE "ZZZZZZZZ";
  co     <= xi(7)                WHEN op(1)='0' ELSE xi(0);
END behave;
```

```
ENTITY instreg IS
  PORT (
    rst         :IN     vlbit;                  --Active low
    clk         :IN     vlbit;
    feth        :IN     vlbit;                  --Active high
    fetl        :IN     vlbit;                  --Active high
    yen         :IN     vlbit;                  --Active high
    ynum        :IN     vlbit;
    din         :IN     vlbit_1d(7 downto 0);
    inst        :OUT    vlbit_1d(15 downto 0);
    yout        :OUT    vlbit_1d(7 downto 0)
  );
END instreg;

ARCHITECTURE behave OF instreg IS
SIGNAL  reg             :vlbit_1d(15 downto 0);
SIGNAL  yout_sig        :vlbit_1d(7 downto 0);
BEGIN
--=======================================high byte reg PROCESS
  PROCESS       --Get high byte instruction from din when feth is active
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      reg(15 downto 8)<="00000000";
    ELSE
      IF (feth='1') THEN
        reg(15 downto 8)<=din;
      END IF;
    END IF;
  END PROCESS;
--=======================================low byte reg PROCESS
  PROCESS       --Get low byte instruction from din when fetl is active
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      reg(7 downto 0)<="00000000";
    ELSE
      IF (fetl='1') THEN
        reg(7 downto 0)<=din;
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  inst     <= reg;
  yout_sig <= reg(7 downto 0) WHEN ynum='0' ELSE
              reg(9)&reg(9)&reg(9)&reg(9)&reg(9)&reg(9)&reg(9)&reg(8);
  yout     <= yout_sig        WHEN yen='1'  ELSE "ZZZZZZZZ";
END behave;
```

```
ENTITY addrreg IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    en           :IN     vlbit;                    --Active high
    lat          :IN     vlbit;                    --Active high
    xin          :IN     vlbit_1d(7 downto 0);
    yin          :IN     vlbit_1d(7 downto 0);
    addr         :OUT    vlbit_1d(15 downto 0)
  );
END addrreg;

ARCHITECTURE behave OF addrreg IS
SIGNAL  reg              :vlbit_1d(15 downto 0);
BEGIN
--=======================================reg PROCESS
  PROCESS       --Update new address from xin:yin when lat is active
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      reg<="0000000000000000";
    ELSE
      IF (lat='1') THEN
        reg<=xin&yin;
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  addr <= reg WHEN en='1' ELSE "ZZZZZZZZZZZZZZZZ";
END behave;
```

```
ENTITY dinbuf IS
  PORT (
    en          :IN     vlbit;                    --Active high
    din         :IN     vlbit_1d(7 downto 0);
    rout        :OUT    vlbit_1d(7 downto 0)
  );
END dinbuf;

ARCHITECTURE behave OF dinbuf IS
BEGIN
  rout <= din WHEN en='1' ELSE "ZZZZZZZZ";
END behave;
```

```
ENTITY doutreg IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    lat          :IN     vlbit;                    --Active high
    xin          :IN     vlbit_1d(7 downto 0);
    dout         :OUT    vlbit_1d(7 downto 0)
  );
END doutreg;

ARCHITECTURE behave OF doutreg IS
SIGNAL  reg              :vlbit_1d(7 downto 0);
BEGIN
--======================================reg PROCESS
  PROCESS       --Update new data out from xin when lat is active
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      reg<="00000000";
    ELSE
      IF (lat='1') THEN
        reg<=xin;
      END IF;
    END IF;
  END PROCESS;
--======================================Output of ENTITY
  dout <= reg;
END behave;
```

```
ENTITY osdgen IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    vs           :IN     vlbit;                    --Negative polarity
    hs           :IN     vlbit;                    --Negative polarity
    line         :IN     vlbit_1d(5 downto 0);
    even         :IN     vlbit;                    --'1'=Even, '0'=Odd field
    ver_pos      :IN     vlbit_1d(3 downto 0);
    page         :IN     vlbit_1d(2 downto 0);
    back         :IN     vlbit;                    --Active high
    show         :IN     vlbit;                    --Active high
    din          :IN     vlbit_1d(7 downto 0);
    addr         :OUT    vlbit_1d(15 downto 0);
    rd           :OUT    vlbit;                    --Active low
    halt         :OUT    vlbit;                    --Active high
    rgb          :OUT    vlbit_1d(2 downto 0);
    sw           :OUT    vlbit                     --Active low
  );
END osdgen;

ARCHITECTURE struct OF osdgen IS

  COMPONENT osdtimer
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
      vs         :IN     vlbit;
      hs         :IN     vlbit;
      line       :IN     vlbit_1d(5 downto 0);
      even       :IN     vlbit;
      ver_pos    :IN     vlbit_1d(3 downto 0);
      halt       :OUT    vlbit;
      row        :OUT    vlbit_1d(3 downto 0);
      char_line  :OUT    vlbit_1d(4 downto 0);
      column     :OUT    vlbit_1d(5 downto 0);
      char_dot   :OUT    vlbit_1d(3 downto 0);
      en         :OUT    vlbit
    );
  END COMPONENT;

  COMPONENT osdaddr
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
      row        :IN     vlbit_1d(3 downto 0);
      char_line  :IN     vlbit_1d(4 downto 0);
      column     :IN     vlbit_1d(5 downto 0);
      char_dot   :IN     vlbit_1d(3 downto 0);
      en         :IN     vlbit;
      page       :IN     vlbit_1d(2 downto 0);
      code       :IN     vlbit_1d(15 downto 0);
      addr       :OUT    vlbit_1d(15 downto 0);
      rd         :OUT    vlbit
    );
  END COMPONENT;

  COMPONENT osdcode
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
```

```
    char_dot   :IN     vlbit_1d(3 downto 0);
    en         :IN     vlbit;
    din        :IN     vlbit_1d(7 downto 0);
    code       :OUT    vlbit_1d(15 downto 0)
  );
END COMPONENT;

COMPONENT osdfont
  PORT (
    rst        :IN     vlbit;
    clk        :IN     vlbit;
    char_dot   :IN     vlbit_1d(3 downto 0);
    en         :IN     vlbit;
    din        :IN     vlbit_1d(7 downto 0);
    font       :OUT    vlbit_1d(15 downto 0)
  );
END COMPONENT;

COMPONENT osdfore
  PORT (
    rst        :IN     vlbit;
    clk        :IN     vlbit;
    code       :IN     vlbit_1d(15 downto 0);
    char_dot   :IN     vlbit_1d(3 downto 0);
    en         :IN     vlbit;
    color      :OUT    vlbit_1d(2 downto 0);
    ital       :OUT    vlbit;
    under      :OUT    vlbit;
    flash      :OUT    vlbit
  );
END COMPONENT;

COMPONENT osdback
  PORT (
    rst        :IN     vlbit;
    clk        :IN     vlbit;
    back       :IN     vlbit;
    code       :IN     vlbit_1d(15 downto 0);
    char_dot   :IN     vlbit_1d(3 downto 0);
    en         :IN     vlbit;
    box        :OUT    vlbit
  );
END COMPONENT;

COMPONENT osdlogic
  PORT (
    rst        :IN     vlbit;
    clk        :IN     vlbit;
    even       :IN     vlbit;
    show       :IN     vlbit;
    char_line  :IN     vlbit_1d(4 downto 0);
    char_dot   :IN     vlbit_1d(3 downto 0);
    en         :IN     vlbit;
    box        :IN     vlbit;
    color      :IN     vlbit_1d(2 downto 0);
    ital       :IN     vlbit;
    under      :IN     vlbit;
    flash      :IN     vlbit;
    font       :IN     vlbit_1d(15 downto 0);
    rgb        :OUT    vlbit_1d(2 downto 0);
    sw         :OUT    vlbit
```

```
    );
  END COMPONENT;

SIGNAL  row_sig          :vlbit_1d(3 downto 0);
SIGNAL  char_line_sig    :vlbit_1d(4 downto 0);
SIGNAL  column_sig       :vlbit_1d(5 downto 0);
SIGNAL  char_dot_sig     :vlbit_1d(3 downto 0);
SIGNAL  en_sig           :vlbit;
SIGNAL  code_sig         :vlbit_1d(15 downto 0);
SIGNAL  font_sig         :vlbit_1d(15 downto 0);
SIGNAL  color_sig        :vlbit_1d(2 downto 0);
SIGNAL  ital_sig         :vlbit;
SIGNAL  under_sig        :vlbit;
SIGNAL  flash_sig        :vlbit;
SIGNAL  box_sig          :vlbit;

BEGIN

  u1:   osdtimer
    PORT MAP (
      rst       => rst,
      clk       => clk,
      vs        => vs,
      hs        => hs,
      line      => line,
      even      => even,
      ver_pos   => ver_pos,
      halt      => halt,
      row       => row_sig,
      char_line => char_line_sig,
      column    => column_sig,
      char_dot  => char_dot_sig,
      en        => en_sig
    );

  u2:   osdaddr
    PORT MAP (
      rst       => rst,
      clk       => clk,
      row       => row_sig,
      char_line => char_line_sig,
      column    => column_sig,
      char_dot  => char_dot_sig,
      en        => en_sig,
      page      => page,
      code      => code_sig,
      addr      => addr,
      rd        => rd
    );

  u3:   osdcode
    PORT MAP (
      rst       => rst,
      clk       => clk,
      char_dot  => char_dot_sig,
      en        => en_sig,
      din       => din,
      code      => code_sig
    );

  u4:   osdfont
```

```
    PORT MAP (
      rst       => rst,
      clk       => clk,
      char_dot  => char_dot_sig,
      en        => en_sig,
      din       => din,
      font      => font_sig
    );

  u5:   osdfore
    PORT MAP (
      rst       => rst,
      clk       => clk,
      code      => code_sig,
      char_dot  => char_dot_sig,
      en        => en_sig,
      color     => color_sig,
      ital      => ital_sig,
      under     => under_sig,
      flash     => flash_sig
    );

  u6:   osdback
    PORT MAP (
      rst       => rst,
      clk       => clk,
      back      => back,
      code      => code_sig,
      char_dot  => char_dot_sig,
      en        => en_sig,
      box       => box_sig
    );

  u7:   osdlogic
    PORT MAP (
      rst       => rst,
      clk       => clk,
      even      => even,
      show      => show,
      char_line => char_line_sig,
      char_dot  => char_dot_sig,
      en        => en_sig,
      box       => box_sig,
      color     => color_sig,
      ital      => ital_sig,
      under     => under_sig,
      flash     => flash_sig,
      font      => font_sig,
      rgb       => rgb,
      sw        => sw
    );

END struct;
```

```
ENTITY osdtimer IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    vs           :IN     vlbit;                    --Negative polarity
    hs           :IN     vlbit;                    --Negative polarity
    line         :IN     vlbit_1d(5 downto 0);
    even         :IN     vlbit;                    --'1'=Even, '0'=Odd field
    ver_pos      :IN     vlbit_1d(3 downto 0);
    halt         :OUT    vlbit;                    --Active high
    row          :OUT    vlbit_1d(3 downto 0);
    char_line    :OUT    vlbit_1d(4 downto 0);
    column       :OUT    vlbit_1d(5 downto 0);
    char_dot     :OUT    vlbit_1d(3 downto 0);
    en           :OUT    vlbit                     --Active high
  );
END osdtimer;

ARCHITECTURE behave OF osdtimer IS

  COMPONENT add8b                                  --XC4000 Library
    PORT (
      a          :IN     vlbit_1d(7 downto 0);
      b          :IN     vlbit_1d(7 downto 0);
      ci         :IN     vlbit;
      s          :OUT    vlbit_1d(7 downto 0);
      co         :OUT    vlbit;
      ofl        :OUT    vlbit
    );
  END COMPONENT;

SIGNAL  a_sig           :vlbit_1d(7 downto 0);
SIGNAL  b_sig           :vlbit_1d(7 downto 0);
SIGNAL  s_sig           :vlbit_1d(7 downto 0);
SIGNAL  top_line        :vlbit_1d(5 downto 0);
SIGNAL  stop_top        :vlbit;
SIGNAL  line_ctr        :vlbit_1d(8 downto 0);
SIGNAL  disp_line       :vlbit_1d(8 downto 0);
SIGNAL  left_ctr        :vlbit_1d(7 downto 0);
SIGNAL  stop_left       :vlbit;
SIGNAL  hor_ctr         :vlbit_1d(10 downto 0);
SIGNAL  stop_hor        :vlbit;
SIGNAL  ver_disp        :vlbit;
SIGNAL  en_sig          :vlbit;
SIGNAL  halt_sig        :vlbit;
BEGIN
--=======================================stop_top PROCESS
  a_sig <= "00011101";
  b_sig <= "0000"&ver_pos;

  u1:   add8b
  PORT MAP (
    a  => a_sig,
    b  => b_sig,
    ci => '0',
    s  => s_sig
  );

  top_line <= s_sig(5 downto 0);

  PROCESS       --vs='0' --> reset
```

```
  BEGIN                --vs='1' --> At falling edge of hs, active when
line=top_line
    WAIT UNTIL ((rst='0') OR (vs='0') OR PFALLING(hs));
    IF (rst='0') THEN
      stop_top<='0';
    ELSIF (vs='0') THEN
      stop_top<='0';
    ELSE
      IF (line=top_line) THEN
        stop_top<='1';
      END IF;
    END IF;
  END PROCESS;
--=======================================line_ctr PROCESS
  PROCESS       --stop_top='0' --> reset
  BEGIN           --stop_top='1' --> At falling edge of hs, count up until
MSB='1'
    WAIT UNTIL ((rst='0') OR (stop_top='0') OR PFALLING(hs));
    IF (rst='0') THEN
      line_ctr<="000000000";
    ELSIF (stop_top='0') THEN
      line_ctr<="000000000";
    ELSE
      IF (line_ctr(8)='0') THEN
        line_ctr<=ADDUM(line_ctr(7 downto 0),"00000001");
      END IF;
    END IF;
  END PROCESS;
  disp_line <= line_ctr(7 downto 0)&even;
--=======================================left_ctr PROCESS
  stop_left <= '1' WHEN left_ctr(6 downto 0)=B"1111111" ELSE '0';
  PROCESS       --hs='0' --> reset
  BEGIN         --hs='1' --> count up until stop_left active
    WAIT UNTIL ((rst='0') OR (hs='0') OR PRISING(clk));
    IF (rst='0') THEN
      left_ctr<="00000000";
    ELSIF (hs='0') THEN
      left_ctr<="00000000";
    ELSE
      IF (stop_left='0') THEN
        left_ctr<=ADDUM(left_ctr(6 downto 0),"0000001");
      END IF;
    END IF;
  END PROCESS;
--=======================================hor_ctr PROCESS
  stop_hor <= '1' WHEN hor_ctr(9 downto 0)=B"1000111111" ELSE '0';
  PROCESS       --hs='0' --> reset
  BEGIN         --hs='1' --> count up when en active until stop_hor active
    WAIT UNTIL ((rst='0') OR (hs='0') OR PRISING(clk));
    IF (rst='0') THEN
      hor_ctr<="00000000000";
    ELSIF (hs='0') THEN
      hor_ctr<="00000000000";
    ELSE
      IF ((en_sig='1') AND (stop_hor='0')) THEN
        hor_ctr<=ADDUM(hor_ctr(9 downto 0),"0000000001");
      END IF;
    END IF;
  END PROCESS;
--=======================================en_sig PROCESS
  ver_disp <= '1' WHEN ((stop_top='1') AND (line_ctr(8)='0')) ELSE '0';
```

```
  PROCESS        --ver_disp='0' --> reset
  BEGIN          --ver_disp='1' --> active when stop_left but not stop_hor
    WAIT UNTIL ((rst='0') OR (ver_disp='0') OR PRISING(clk));
    IF (rst='0') THEN
      en_sig<='0';
    ELSIF (ver_disp='0') THEN
      en_sig<='0';
    ELSE
      IF ((stop_left='1') AND (stop_hor='0')) THEN
        en_sig<='1';
      ELSE
        en_sig<='0';
      END IF;
    END IF;
  END PROCESS;
--=======================================halt_sig PROCESS
  PROCESS        --ver_disp='0' --> reset
  BEGIN             --ver_disp='1' --> active before stop_left 10 clk until
stop_hor
    WAIT UNTIL ((rst='0') OR (ver_disp='0') OR PRISING(clk));
    IF (rst='0') THEN
      halt_sig<='0';
    ELSIF (ver_disp='0') THEN
      halt_sig<='0';
    ELSE
      IF (left_ctr(6 downto 0)=B"1110101") THEN
        halt_sig<='1';
      ELSIF (stop_hor='1') THEN
        halt_sig<='0';
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  halt      <= halt_sig;
  row       <= disp_line(8 downto 5);
  char_line <= disp_line(4 downto 0);
  column    <= hor_ctr(9 downto 4);
  char_dot  <= hor_ctr(3 downto 0);
  en        <= en_sig;
END behave;
```

```
ENTITY osdaddr IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    row          :IN     vlbit_1d(3 downto 0);
    char_line    :IN     vlbit_1d(4 downto 0);
    column       :IN     vlbit_1d(5 downto 0);
    char_dot     :IN     vlbit_1d(3 downto 0);
    en           :IN     vlbit;                    --Active high
    page         :IN     vlbit_1d(2 downto 0);
    code         :IN     vlbit_1d(15 downto 0);
    addr         :OUT    vlbit_1d(15 downto 0);
    rd           :OUT    vlbit                     --Active low
  );
END osdaddr;

ARCHITECTURE behave OF osdaddr IS
SIGNAL  addr_sig         :vlbit_1d(15 downto 0);
SIGNAL  rd_sig           :vlbit;
SIGNAL  code_upper       :vlbit_1d(7 downto 0);
SIGNAL  code_lower       :vlbit_1d(7 downto 0);
CONSTANT upper_table     :vlbit_2d(0 to 63,13 downto 0):=(
        B"000000_00100000",B"000001_11010001",
        B"000010_11010100",B"000011_11010101",
        B"000100_11010110",B"000101_11010111",
        B"000110_11100111",B"000111_11101101",
        B"001000_11101000",B"001001_10000000",
        B"001010_10000100",B"001011_10001001",
        B"001100_10001101",B"001101_10010001",
        B"001110_11100111",B"001111_11111100",
        B"010000_11101001",B"010001_10000001",
        B"010010_10000101",B"010011_10001010",
        B"010100_10001110",B"010101_10010010",
        B"010110_11100111",B"010111_11111101",
        B"011000_11101010",B"011001_10000010",
        B"011010_10000110",B"011011_10001011",
        B"011100_10001111",B"011101_10010011",
        B"011110_11100111",B"011111_11111110",
        B"100000_11101011",B"100001_10000011",
        B"100010_10000111",B"100011_10001100",
        B"100100_10010000",B"100101_10010100",
        B"100110_11100111",B"100111_11111111",
        B"101000_11101100",B"101001_11010001",
        B"101010_10001000",B"101011_11010101",
        B"101100_11010110",B"101101_11010111",
        B"101110_11100111",B"101111_11101101",
        B"110000_00100000",B"110001_11010001",
        B"110010_11010100",B"110011_11010101",
        B"110100_11010110",B"110101_11010111",
        B"110110_11100111",B"110111_11101101",
        B"111000_00100000",B"111001_11010001",
        B"111010_11010100",B"111011_11010101",
        B"111100_11010110",B"111101_11010111",
        B"111110_11100111",B"111111_11101101");
CONSTANT lower_table     :vlbit_2d(0 to 3,9 downto 0):=(
        B"00_00100000",B"01_11011000",B"10_11011001",B"11_11011010");
BEGIN
--=======================================addr_sig PROCESS
  PLA_TABLE(code(7 downto 2),code_upper,upper_table);
  PLA_TABLE(code(1 downto 0),code_lower,lower_table);
  PROCESS       --en='0' --> reset
```

```
  BEGIN           --en='1' --> Generate address to access RAM(code) and ROM
(font)
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      addr_sig<="0000000000000000";
    ELSIF (en='0') THEN
      addr_sig<="0000000000000000";
    ELSE
      CASE char_dot(3 downto 2) IS
        WHEN B"00" => addr_sig<="10"&page&row&column&char_dot(1);
        WHEN B"01" => addr_sig<="01"&code(15 downto 8)&char_line&char_dot(1)
;
        WHEN B"10" => IF (code(15 downto 12)=B"0000") THEN
                        addr_sig<="0100100000"&char_line&char_dot(1);
                      ELSE
                        IF (char_line(4)='0') THEN
                          addr_sig<="01"&code_upper&char_line&char_dot(1);
                        ELSE
                          addr_sig<="01"&code_lower&char_line&char_dot(1);
                        END IF;
                      END IF;
        WHEN B"11" => addr_sig<="XXXXXXXXXXXXXXXX";
      END CASE;
    END IF;
  END PROCESS;
--=======================================rd_sig PROCESS
  PROCESS       --en='0' --> set
  BEGIN          --en='1' --> Generate LOW pulse from char_dot 0 to char_dot
11
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      rd_sig<='1';
    ELSIF (en='0') THEN
      rd_sig<='1';
    ELSE
      IF (char_dot(3 downto 2)=B"11") THEN
        rd_sig<='1';
      ELSE
        rd_sig<='0';
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  addr <= addr_sig WHEN en='1' ELSE "ZZZZZZZZZZZZZZZZ";
  rd   <= rd_sig WHEN en='1' ELSE 'Z';
END behave;
```

```
ENTITY osdcode IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    char_dot     :IN     vlbit_1d(3 downto 0);
    en           :IN     vlbit;                    --Active high
    din          :IN     vlbit_1d(7 downto 0);
    code         :OUT    vlbit_1d(15 downto 0)
  );
END osdcode;

ARCHITECTURE behave OF osdcode IS
SIGNAL  code_sig         :vlbit_1d(15 downto 0);
BEGIN
--======================================high byte code_sig PROCESS
  PROCESS        --en='0' --> reset
  BEGIN          --en='1' --> get high byte display code at char_dot 2
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      code_sig(15 downto 8)<="00000000";
    ELSIF (en='0') THEN
      code_sig(15 downto 8)<="00000000";
    ELSE
      IF (char_dot=B"0010") THEN
        code_sig(15 downto 8)<=din;
      END IF;
    END IF;
  END PROCESS;
--======================================low byte code_sig PROCESS
  PROCESS        --en='0' --> reset
  BEGIN          --en='1' --> get low byte display code at char_dot 4
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      code_sig(7 downto 0)<="00000000";
    ELSIF (en='0') THEN
      code_sig(7 downto 0)<="00000000";
    ELSE
      IF (char_dot=B"0100") THEN
        code_sig(7 downto 0)<=din;
      END IF;
    END IF;
  END PROCESS;
--======================================Output of ENTITY
  code <= code_sig;
END behave;
```

```
ENTITY osdfont IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    char_dot     :IN     vlbit_1d(3 downto 0);
    en           :IN     vlbit;                    --Active high
    din          :IN     vlbit_1d(7 downto 0);
    font         :OUT    vlbit_1d(15 downto 0)
  );
END osdfont;

ARCHITECTURE behave OF osdfont IS
SIGNAL  font_sig         :vlbit_1d(15 downto 0);
BEGIN
--=======================================high byte font_sig PROCESS
  PROCESS        --en='0' --> reset
  BEGIN           --en='1' --> get high byte font at char_dot 6 and char_dot
10
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      font_sig(15 downto 8)<="00000000";
    ELSIF (en='0') THEN
      font_sig(15 downto 8)<="00000000";
    ELSE
      IF (char_dot=B"0110") THEN
        font_sig(15 downto 8)<=din;
      ELSIF (char_dot=B"1010") THEN
        font_sig(15 downto 8)<=font_sig(15 downto 8) OR din;
      END IF;
    END IF;
  END PROCESS;
--=======================================low byte font_sig PROCESS
  PROCESS        --en='0' --> reset
  BEGIN          --en='1' --> get low byte font at char_dot 8 and char_dot 12
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      font_sig(7 downto 0)<="00000000";
    ELSIF (en='0') THEN
      font_sig(7 downto 0)<="00000000";
    ELSE
      IF (char_dot=B"1000") THEN
        font_sig(7 downto 0)<=din;
      ELSIF (char_dot=B"1100") THEN
        font_sig(7 downto 0)<=font_sig(7 downto 0) OR din;
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  font <= font_sig;
END behave;
```

```
ENTITY osdfore IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    code         :IN     vlbit_1d(15 downto 0);
    char_dot     :IN     vlbit_1d(3 downto 0);
    en           :IN     vlbit;                    --Active high
    color        :OUT    vlbit_1d(2 downto 0);     --RGB
    ital         :OUT    vlbit;                    --Active high
    under        :OUT    vlbit;                    --Active high
    flash        :OUT    vlbit                     --Active high
  );
END osdfore;

ARCHITECTURE behave OF osdfore IS
SIGNAL  fore_attr       :vlbit_1d(5 downto 0);
BEGIN
--=======================================fore_attr PROCESS
  PROCESS       --en='0' --> reset
  BEGIN           --en='1' --> get fore_attr at char_dot 0 when code=change
fore
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      fore_attr<="000000";
    ELSIF (en='0') THEN
      fore_attr<="000000";
    ELSE
      IF ((char_dot=B"0000") AND (code(15 downto 9)=B"0000XX1")) THEN
        fore_attr<=code(5 downto 0);
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  color <= fore_attr(2 downto 0);
  ital  <= fore_attr(3);
  under <= fore_attr(4) WHEN code(15 downto 12)/=B"0000" ELSE '0';
  flash <= fore_attr(5);
END behave;
```

```
ENTITY osdback IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    back         :IN     vlbit;                    --Active high
    code         :IN     vlbit_1d(15 downto 0);
    char_dot     :IN     vlbit_1d(3 downto 0);
    en           :IN     vlbit;                    --Active high
    box          :OUT    vlbit                     --Active high
  );
END osdback;

ARCHITECTURE behave OF osdback IS
SIGNAL  box_sig          :vlbit;
BEGIN
--======================================box_sig PROCESS
  PROCESS        --en='0' --> reset
  BEGIN          --en='1' --> get box at char_dot 0 when code!=no background
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      box_sig<='0';
    ELSIF (en='0') THEN
      box_sig<='0';
    ELSE
      IF (char_dot=B"0000") THEN
        IF (code(15 downto 8)=B"0000XXX0") THEN
          box_sig<='0';
        ELSE
          box_sig<=back;
        END IF;
      END IF;
    END IF;
  END PROCESS;
--======================================Output of ENTITY
  box <= box_sig;
END behave;
```

```
ENTITY osdlogic IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    even         :IN     vlbit;                    --'1'=Even, '0'=Odd field
    show         :IN     vlbit;                    --Active high
    char_line    :IN     vlbit_1d(4 downto 0);
    char_dot     :IN     vlbit_1d(3 downto 0);
    en           :IN     vlbit;                    --Active high
    box          :IN     vlbit;                    --Active high
    color        :IN     vlbit_1d(2 downto 0);     --RGB
    ital         :IN     vlbit;                    --Active high
    under        :IN     vlbit;                    --Active high
    flash        :IN     vlbit;                    --Active high
    font         :IN     vlbit_1d(15 downto 0);
    rgb          :OUT    vlbit_1d(2 downto 0);
    sw           :OUT    vlbit                     --Active low
  );
END osdlogic;

ARCHITECTURE behave OF osdlogic IS
SIGNAL  ital_ctr        :vlbit_1d(2 downto 0);
SIGNAL  ital_dir        :vlbit;
SIGNAL  ital_font       :vlbit_1d(15 downto 0);
SIGNAL  under_line      :vlbit;
SIGNAL  under_font      :vlbit_1d(15 downto 0);
SIGNAL  flash_timer     :vlbit_1d(5 downto 0);
SIGNAL  dot             :vlbit_1d(15 downto 0);
BEGIN
--=======================================ital_ctr PROCESS
  PROCESS       --en='0' --> reset
  BEGIN         --en='1' --> load counter at char_dot 12, count until max
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      ital_ctr<="100";
    ELSIF (en='0') THEN
      ital_ctr<="100";
    ELSE
      IF (char_dot=B"1100") THEN
        IF (ital='1') THEN
          ital_ctr<="0"&(char_line(4)&char_line(4)  XOR  char_line(3  downto
2));
        ELSE
          ital_ctr<="011";
        END IF;
      ELSE
        IF (ital_ctr(2)='0') THEN
          ital_ctr<=ADDUM(ital_ctr(1 downto 0),"01");
        END IF;
      END IF;
    END IF;
  END PROCESS;
--=======================================ital_font PROCESS
  ital_dir <= '0' WHEN ital='0' ELSE char_line(4);
  PROCESS       --en='0' --> reset
  BEGIN          --en='1' --> load font at char_dot 13, shift until ital_ctr
max
    WAIT UNTIL ((rst='0') OR (en='0') OR PFALLING(clk));
    IF (rst='0') THEN
      ital_font<="0000000000000000";
    ELSIF (en='0') THEN
```

```
      ital_font<="0000000000000000";
    ELSE
      IF (ital_ctr(2)='0') THEN
        IF (char_dot=B"1101") THEN
          IF (ital_dir='0') THEN
            ital_font<=font;
          ELSE
            ital_font<=font(14 downto 0)&"0";
          END IF;
        ELSE
          IF (ital_dir='0') THEN
            ital_font<="0"&ital_font(15 downto 1);
          ELSE
            ital_font<=ital_font(14 downto 0)&"0";
          END IF;
        END IF;
      END IF;
    END IF;
  END PROCESS;
--=======================================under_font SIGNAL
  under_line <= '0' WHEN under='0'          ELSE
                '1' WHEN char_line=B"11011" ELSE
                '1' WHEN char_line=B"11100" ELSE '0';
  under_font <= ital_font WHEN under_line='0' ELSE
                ((NOT(ital_font(13 downto 0))&"11") AND
                ("11"&NOT(ital_font(15 downto 2)))) OR ital_font;
--=======================================flash_timer PROCESS
  PROCESS       --Loop count every frame, bit 4's period is 32/25 second
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(even));
    IF (rst='0') THEN
      flash_timer<="000000";
    ELSE
      flash_timer<=ADDUM(flash_timer(4 downto 0),"00001");
    END IF;
  END PROCESS;
--=======================================dot PROCESS
  PROCESS       --en='0' --> reset
  BEGIN         --en='1' --> load font at char_dot 0, shift dot out
    WAIT UNTIL ((rst='0') OR (en='0') OR PRISING(clk));
    IF (rst='0') THEN
      dot<="0000000000000000";
    ELSIF (en='0') THEN
      dot<="0000000000000000";
    ELSE
      IF (char_dot=B"0000") THEN
        IF ((flash='0') OR (flash_timer(4)='1')) THEN
          dot<=under_font;
        ELSE
          dot<="0000000000000000";
        END IF;
      ELSE
        dot<=dot(14 downto 0)&"0";
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  rgb <= (dot(15)&dot(15)&dot(15)) AND color WHEN show='1' ELSE "000";
  sw  <= NOT(dot(15) OR box)                 WHEN show='1' ELSE '1';
END behave;
```

```
ENTITY osdctrl IS
  PORT (
    rst          :IN     vlbit;                  --Active low
    clk          :IN     vlbit;
    even         :IN     vlbit;                  --'1'=Even, '0'=Odd field
    cs           :IN     vlbit;                  --Active low
    wr           :IN     vlbit;                  --Active low
    a0           :IN     vlbit;                  --'0'=page, '1'=scroll
    din          :IN     vlbit_1d(7 downto 0);
    ver_pos      :OUT    vlbit_1d(3 downto 0);
    page         :OUT    vlbit_1d(2 downto 0)
  );
END osdctrl;

ARCHITECTURE behave OF osdctrl IS
SIGNAL  scroll           :vlbit;
SIGNAL  ver_pos_sig      :vlbit_1d(4 downto 0);
SIGNAL  page_sig         :vlbit_1d(2 downto 0);
BEGIN
--=======================================scroll PROCESS
  PROCESS       --Generate high pulse when cs and wr active at a0='1'
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(clk));
    IF (rst='0') THEN
      scroll<='0';
    ELSE
      IF ((cs='0') AND (wr='0') AND (a0='1')) THEN
        scroll<='1';
      ELSE
        scroll<='0';
      END IF;
    END IF;
  END PROCESS;
--=======================================ver_pos_sig PROCESS
  PROCESS       --scroll='1' --> set
  BEGIN         --scroll='0' --> at rising edge of even, count down until 0
    WAIT UNTIL ((rst='0') OR (scroll='1') OR PRISING(even));
    IF (rst='0') THEN
      ver_pos_sig<="01111";
    ELSIF (scroll='1') THEN
      ver_pos_sig<="01111";
    ELSE
      IF (ver_pos_sig(3 downto 0)/=B"0000") THEN
        ver_pos_sig<=SUBUM(ver_pos_sig(3 downto 0),"0001");
      END IF;
    END IF;
  END PROCESS;
--=======================================page_sig PROCESS
  PROCESS       --Update value when cs and wr active at a0='0'
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      page_sig<="000";
    ELSE
      IF ((cs='0') AND (wr='0') AND (a0='0')) THEN
        page_sig<=din(2 downto 0);
      END IF;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  ver_pos <= ver_pos_sig(3 downto 0);
```

```
  page    <= page_sig;
END behave;
```

```
ENTITY capdata IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    hs           :IN     vlbit;                    --Negative polarity
    line         :IN     vlbit_1d(5 downto 0);
    even         :IN     vlbit;                    --'1'=Even, '0'=Odd field
    slice        :IN     vlbit;
    cs           :IN     vlbit;                    --Active low
    rd           :IN     vlbit;                    --Active low
    a1           :IN     vlbit;                    --"00"=Frame
    a0           :IN     vlbit;                    --"01"=Byte 1, "10"=Byte 2
    dout         :OUT    vlbit_1d(7 downto 0)
  );
END capdata;

ARCHITECTURE struct OF capdata IS

  COMPONENT capclock
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
      hs         :IN     vlbit;
      line       :IN     vlbit_1d(5 downto 0);
      even       :IN     vlbit;
      slice      :IN     vlbit;
      cclk       :OUT    vlbit;
      finish     :OUT    vlbit
    );
  END COMPONENT;

  COMPONENT capshift
    PORT (
      rst        :IN     vlbit;
      cclk       :IN     vlbit;
      slice      :IN     vlbit;
      capreg     :OUT    vlbit_1d(18 downto 0)
    );
  END COMPONENT;

  COMPONENT capio
    PORT (
      rst        :IN     vlbit;
      clk        :IN     vlbit;
      finish     :IN     vlbit;
      cs         :IN     vlbit;
      rd         :IN     vlbit;
      a1         :IN     vlbit;
      a0         :IN     vlbit;
      capreg     :IN     vlbit_1d(18 downto 0);
      dout       :OUT    vlbit_1d(7 downto 0)
    );
  END COMPONENT;

SIGNAL  cclk_sig        :vlbit;
SIGNAL  finish_sig      :vlbit;
SIGNAL  capreg_sig      :vlbit_1d(18 downto 0);

BEGIN

  u1:   capclock
```

```
    PORT MAP (
      rst          => rst,
      clk          => clk,
      hs           => hs,
      line         => line,
      even         => even,
      slice        => slice,
      cclk         => cclk_sig,
      finish       => finish_sig
    );

    u2:   capshift
    PORT MAP (
      rst          => rst,
      cclk         => cclk_sig,
      slice        => slice,
      capreg       => capreg_sig
    );

    u3:   capio
    PORT MAP (
      rst          => rst,
      clk          => clk,
      finish       => finish_sig,
      cs           => cs,
      rd           => rd,
      a1           => a1,
      a0           => a0,
      capreg       => capreg_sig,
      dout         => dout
    );

END struct;
```

```
ENTITY capclock IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    hs           :IN     vlbit;                    --Negative polarity
    line         :IN     vlbit_1d(5 downto 0);
    even         :IN     vlbit;                    --'1'=Even, '0'=Odd field
    slice        :IN     vlbit;
    cclk         :OUT    vlbit;
    finish       :OUT    vlbit                     --Active high
  );
END capclock;

ARCHITECTURE behave OF capclock IS
SIGNAL  runin_ctr       :vlbit_1d(9 downto 0);
SIGNAL  stop_runin      :vlbit;
SIGNAL  cclk_pulse      :vlbit;
SIGNAL  cclk_ctr        :vlbit_1d(5 downto 0);
SIGNAL  cclk_sig        :vlbit;
SIGNAL  bit_ctr         :vlbit_1d(5 downto 0);
SIGNAL  stop_bit        :vlbit;
SIGNAL  finish_sig      :vlbit;
BEGIN
--=======================================runin_ctr PROCESS
  stop_runin <= '1' WHEN runin_ctr(8 downto 0)=B"100011101" ELSE '0';
  PROCESS       --hs='0' --> reset
  BEGIN         --hs='1' --> count up until stop_runin active
    WAIT UNTIL ((rst='0') OR (hs='0') OR PRISING(clk));
    IF (rst='0') THEN
      runin_ctr<="0000000000";
    ELSIF (hs='0') THEN
      runin_ctr<="0000000000";
    ELSE
      IF (stop_runin='0') THEN
        runin_ctr<=ADDUM(runin_ctr(8 downto 0),"000000001");
      END IF;
    END IF;
  END PROCESS;
--=======================================cclk_pulse PROCESS
  PROCESS       --Active when slice active in runin period
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(clk));
    IF (rst='0') THEN
      cclk_pulse<='0';
    ELSE
      IF ((slice='1') AND (stop_runin='0')) THEN
        cclk_pulse<='1';
      ELSE
        cclk_pulse<='0';
      END IF;
    END IF;
  END PROCESS;
--=======================================cclk_ctr PROCESS
  PROCESS       --cclk_pulse='1' --> reset
  BEGIN         --cclk_pulse='0' --> toggle MSB every 12 clk
    WAIT UNTIL ((rst='0') OR (cclk_pulse='1') OR PRISING(clk));
    IF (rst='0') THEN
      cclk_ctr<="000000";
    ELSIF (cclk_pulse='1') THEN
      cclk_ctr<="000000";
    ELSE
```

```
        IF (cclk_ctr(3 downto 0)=B"1011") THEN
          cclk_ctr<=(NOT cclk_ctr(5))&"00000";
        ELSE
          cclk_ctr<=cclk_ctr(5)&ADDUM(cclk_ctr(3 downto 0),"0001");
        END IF;
      END IF;
    END PROCESS;
  --=======================================cclk_sig PROCESS
    PROCESS       --Generate 19 cclk in line 18 odd field after runin
    BEGIN
      WAIT UNTIL ((rst='0') OR PFALLING(clk));
      IF (rst='0') THEN
        cclk_sig<='0';
      ELSE
        IF ((line=B"010001") AND (even='0') AND
            (stop_runin='1') AND (stop_bit='0')) THEN
          cclk_sig<=cclk_ctr(5);
        ELSE
          cclk_sig<='0';
        END IF;
      END IF;
    END PROCESS;
  --=======================================bit_ctr PROCESS
    stop_bit <= '1' WHEN bit_ctr(4 downto 0)=B"10011" ELSE '0';
    PROCESS        --hs='0' --> reset
    BEGIN          --hs='1' --> count up until 19
      WAIT UNTIL ((rst='0') OR (hs='0') OR PFALLING(cclk_sig));
      IF (rst='0') THEN
        bit_ctr<="000000";
      ELSIF (hs='0') THEN
        bit_ctr<="000000";
      ELSE
        IF (stop_bit='0') THEN
          bit_ctr<=ADDUM(bit_ctr(4 downto 0),"00001");
        END IF;
      END IF;
    END PROCESS;
  --=======================================finish_sig PROCESS
    PROCESS       --Generate HIGH pulse after all data is shift
    BEGIN
      WAIT UNTIL ((rst='0') OR PRISING(clk));
      IF (rst='0') THEN
        finish_sig<='0';
      ELSE
        IF (stop_bit='1') THEN
          finish_sig<=cclk_ctr(5);
        ELSE
          finish_sig<='0';
        END IF;
      END IF;
    END PROCESS;
  --=======================================Output of ENTITY
    cclk   <= cclk_sig;
    finish <= finish_sig;
  END behave;
```

```
ENTITY capshift IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    cclk         :IN     vlbit;
    slice        :IN     vlbit;
    capreg        :OUT     vlbit_1d(18 downto 0)    --Frame, Byte 1, 2 (MSB->
LSB)
  );
END capshift;

ARCHITECTURE  behave OF capshift IS
SIGNAL  reg              :vlbit_1d(18 downto 0);
BEGIN
--=======================================reg PROCESS
  PROCESS       --Shift data into form Byte 2, Byte 1, Frame (MSB->LSB)
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(cclk));
    IF (rst='0') THEN
      reg<="000000000000000000";
    ELSE
      reg<=slice&reg(18 downto 1);
    END IF;
  END PROCESS;
--=====================================Output of ENTITY
  capreg <= reg(2 downto 0)&reg(10 downto 3)&reg(18 downto 11);
END behave;
```

```
ENTITY capio IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    finish       :IN     vlbit;                    --Active high
    cs           :IN     vlbit;                    --Active low
    rd           :IN     vlbit;                    --Active low
    a1           :IN     vlbit;                    --"00"=Frame
    a0           :IN     vlbit;                    --"01"=Byte 1, "10"=Byte 2
    capreg        :IN      vlbit_1d(18 downto 0);   --Frame, Byte 1, 2 (MSB->
LSB)
    dout         :OUT    vlbit_1d(7 downto 0)
  );
END capio;

ARCHITECTURE behave OF capio IS
SIGNAL  cs_rd_clk       :vlbit;
SIGNAL  status          :vlbit;
SIGNAL  dout_sig        :vlbit_1d(7 downto 0);
SIGNAL  parity1         :vlbit;
SIGNAL  parity2         :vlbit;
BEGIN
--======================================cs_rd_clk PROCESS
  PROCESS       --Delay cs and rd pulse when both active
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(clk));
    IF (rst='0') THEN
      cs_rd_clk<='1';
    ELSE
      IF ((cs='0') AND (rd='0')) THEN
        cs_rd_clk<='0';
      ELSE
        cs_rd_clk<='1';
      END IF;
    END IF;
  END PROCESS;
--======================================status PROCESS
  PROCESS       --finish='1' --> set
  BEGIN         --finish='0' --> reset at rising edge of cs_rd_clk
    WAIT UNTIL ((rst='0') OR (finish='1') OR PRISING(cs_rd_clk));
    IF (rst='0') THEN
      status<='1';
    ELSIF (finish='1') THEN
      status<='1';
    ELSE
      status<='0';
    END IF;
  END PROCESS;
--======================================Output of ENTITY
  parity1  <= capreg(15) XOR capreg(14) XOR capreg(13) XOR capreg(12) XOR
              capreg(11) XOR capreg(10) XOR capreg(9)  XOR capreg(8);
  parity2  <= capreg(7)  XOR capreg(6)  XOR capreg(5)  XOR capreg(4) XOR
              capreg(3)  XOR capreg(2)  XOR capreg(1)  XOR capreg(0);
  dout_sig <= "0000"&status&capreg(18 downto 16) WHEN a1&a0=B"00" ELSE
              parity1&capreg(14 downto 8)        WHEN a1&a0=B"01" ELSE
              parity2&capreg(6 downto 0)         WHEN a1&a0=B"10" ELSE
              "XXXXXXXX";
  dout     <= "ZZZZZZZZ" WHEN (cs='1' OR rd='1') ELSE dout_sig;
END behave;
```

```
ENTITY ioport IS
  PORT (
    cs          :IN     vlbit;                  --Active low
    rd          :IN     vlbit;                  --Active low
    pin         :IN     vlbit_1d(1 downto 0);
    dout        :OUT    vlbit_1d(7 downto 0)
  );
END ioport;

ARCHITECTURE behave OF ioport IS
BEGIN
  dout <= "ZZZZZZZZ" WHEN (cs='1' OR rd='1') ELSE "000000"&pin;
END behave;
```

```
ENTITY syncpro IS
  PORT (
    rst          :IN     vlbit;                    --Active low
    clk          :IN     vlbit;
    cs           :IN     vlbit;                    --Negative polarity
    vs           :OUT    vlbit;                    --Negative polarity
    hs           :OUT    vlbit;                    --Negative polarity
    line         :OUT    vlbit_1d(5 downto 0);
    even         :OUT    vlbit                     --'1'=Even, '0'=Odd field
  );
END syncpro;

ARCHITECTURE behave OF syncpro IS
SIGNAL  cs_old           :vlbit;
SIGNAL  high_ctr         :vlbit_1d(9 downto 0);
SIGNAL  low_ctr          :vlbit_1d(8 downto 0);
SIGNAL  vs_sig           :vlbit;
SIGNAL  hs_sig           :vlbit;
SIGNAL  line_sig         :vlbit_1d(6 downto 0);
SIGNAL  equal_ctr        :vlbit;
SIGNAL  even_sig         :vlbit;
BEGIN
--=======================================cs_old PROCESS
  PROCESS       --Save value of cs at last clk
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      cs_old<='0';
    ELSE
      cs_old<=cs;
    END IF;
  END PROCESS;
--=======================================high_ctr PROCESS
  PROCESS       --cs_old='0' --> reset
  BEGIN         --cs_old='1' --> count up until MSB='1'
    WAIT UNTIL ((rst='0') OR (cs_old='0') OR PRISING(clk));
    IF (rst='0') THEN
      high_ctr<="0000000000";
    ELSIF (cs_old='0') THEN
      high_ctr<="0000000000";
    ELSE
      IF (high_ctr(9)='0') THEN
       high_ctr<=ADDUM(high_ctr(8 downto 0),"000000001");
      END IF;
    END IF;
  END PROCESS;
--=======================================low_ctr PROCESS
  PROCESS       --cs_old='1' --> reset
  BEGIN         --cs_old='0' --> count up until MSB='1'
    WAIT UNTIL ((rst='0') OR (cs_old='1') OR PRISING(clk));
    IF (rst='0') THEN
      low_ctr<="000000000";
    ELSIF (cs_old='1') THEN
      low_ctr<="000000000";
    ELSE
      IF (low_ctr(8)='0') THEN
       low_ctr<=ADDUM(low_ctr(7 downto 0),"00000001");
      END IF;
    END IF;
  END PROCESS;
--=======================================vs_sig PROCESS
```

```
  PROCESS       --At rising edge of cs, cs='0'>21.33us --> LOW
  BEGIN         --                     cs='0'<21.33us --> HIGH
    WAIT UNTIL ((rst='0') OR PRISING(cs));
    IF (rst='0') THEN
      vs_sig<='1';
    ELSE
      vs_sig<=NOT low_ctr(8);
    END IF;
  END PROCESS;
--=======================================hs_sig PROCESS
  PROCESS       --Generate 1 clk width LOW pulse at falling edge of cs
  BEGIN
    WAIT UNTIL ((rst='0') OR PRISING(clk));
    IF (rst='0') THEN
      hs_sig<='1';
    ELSE
      IF ((cs='0') AND (cs_old='1')) THEN
       hs_sig<='0';
      ELSE
       hs_sig<='1';
      END IF;
    END IF;
  END PROCESS;
--=======================================line_sig PROCESS
  PROCESS       --vs='0' --> reset
  BEGIN         --vs='1' --> At falling edge of hs, count up until max.
    WAIT UNTIL ((rst='0') OR (vs_sig='0') OR PFALLING(hs_sig));
    IF (rst='0') THEN
      line_sig<="0000000";
    ELSIF (vs_sig='0') THEN
      line_sig<="0000000";
    ELSE
      IF (line_sig(5 downto 0)/=B"111111") THEN
       line_sig<=ADDUM(line_sig(5 downto 0),"000001");
      END IF;
    END IF;
  END PROCESS;
--=======================================equal_ctr PROCESS
  PROCESS       --cs='1'>42.66us --> reset
  BEGIN         --cs='1'<42.66us --> At rising edge of cs_old, toggle
    WAIT UNTIL ((rst='0') OR (high_ctr(9)='1') OR PRISING(cs_old));
    IF (rst='0') THEN
      equal_ctr<='0';
    ELSIF (high_ctr(9)='1') THEN
      equal_ctr<='0';
    ELSE
      equal_ctr<=NOT equal_ctr;
    END IF;
  END PROCESS;
--=======================================even_sig PROCESS
  PROCESS       --At falling edge of vs, get field value
  BEGIN
    WAIT UNTIL ((rst='0') OR PFALLING(vs_sig));
    IF (rst='0') THEN
      even_sig<='0';
    ELSE
      even_sig<=equal_ctr;
    END IF;
  END PROCESS;
--=======================================Output of ENTITY
  vs   <= vs_sig;
```

```
  hs   <= hs_sig;
  line <= line_sig(5 downto 0);
  even <= even_sig;
END behave;
```

ประวัติผู้เขียน

นายกฤษณ์ อธิกุลวงศ์ เกิดวันที่ 25 ตุลาคม พ.ศ. 2517 ที่กรุงเทพมหานคร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2537 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า ที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2538 ในระหว่างการศึกษาระดับมหาบัณฑิตนี้ได้รับทุนผู้ช่วยวิจัยจากฝ่ายวิจัย จุฬาลงกรณ์มหาวิทยาลัย