บทที่ 4

การออกแบบและวิเคราะห์วงจรรวมที่ใช้กับเทเลเท็กซต์ด้วยภาษาวีเอชดีแอล

Top-Down Design กับ Bottom-Up Design

ตามปกติผู้ออกแบบมักจะแบ่งงานใหญ่ ๆ ออกเป็นออกเป็นชิ้นงานเล็ก ๆ เป็นลำดับชั้นก่อน แล้วจึงทำการแก้ปัญหา โดยสามารถกระทำได้สองวิธีคือ

1. **Top-Down Design** เป็นกระบวนการออกแบบวงจรที่แก้ปัญหาโดยการแก้ปัญหาจากส่วนบนก่อน ซึ่งเป็นการอธิบายความสัมพันธ์ของสัญญาณต่าง ๆ อย่างหยาบ ๆ เทียบได้กับการอธิบายข้อกำหนดของวงจร (Specification) ทำให้ผู้ออกแบบทราบความต้องการของผู้ใช้ได้ชัดเจนและมองเห็นแนวทางในการออกแบบในขั้นล่างต่อไปได้ดียิ่งขึ้น จากนั้นจึงทำการแบ่งปัญหาหลักเป็นปัญหาย่อย ๆ แล้วทำการแก้ปัญหาในทำนองเดียวกัน การแก้ปัญหาเป็นลำดับขั้นและการแบ่งปัญหาจะนี้จะทำจนถึงขั้นที่สามารถทำให้เป็นจริงได้ นั่นคือขั้นที่สามารถสังเคราะห์วงจรจากภาษาวีเอชดีแอลหรือไอซีที่มีอยู่ได้

2. **Bottom-Up Design** เป็นกระบวนการออกแบบอีกวิธีหนึ่งที่มีมุมมองที่กลับกัน คือพยายามแก้ปัญหาจากปัญหาด้านล่างก่อน โดยพยายามใช้สิ่งที่มีอยู่แล้วมาประกอบกัน เพื่อใช้แก้ปัญหาที่มีขั้นสูง ๆ ขึ้นไป จนกระทั่งถึงปัญหาหลัก

ในทางปฏิบัติการออกแบบครั้งหนึ่งอาจใช้การออกแบบทั้งสองวิธีรวมกัน เพื่อให้เกิดประสิทธิภาพสูงสุด กล่าวคือมักใช้ Top-Down Design ในขั้นแรก ๆ แล้วพยายามแบ่งปัญหาใหญ่เป็นปัญหาย่อยที่ใกล้เคียงกับสิ่งที่มีอยู่แล้ว ยกตัวอย่างเช่นการออกแบบ CPU หลังจากได้ออกแบบสถาปัตยกรรมภายในซึ่งใช้ Top-Down Design แล้วหากพบว่ามีบางส่วนที่มีลักษณะคล้ายกัน

กับที่มีอยู่ใน CPU รุ่นก่อน ๆ เช่น หน่วยควบคุมแคช หรือ หน่วยประมวลผลเลขจำนวนเต็ม ก็อาจนำวงจรดังกล่าวมาประกอบกับวงจรอีกบางส่วนเพื่อใช้เป็นวงจรของ CPU รุ่นใหม่ได้อย่างรวดเร็ว ซึ่งจะเรียกวิธีการนี้ว่า Bottom-Up Reuse ส่วนวงจรอื่น ๆ อาจจะออกแบบโดยใช้ Top-Down ลงไป หรือทำการออกแบบในระดับขั้นนั้นเลยก็ได้

ภาษาวีเอชดีแอลได้ถูกออกแบบมาให้สนับสนุนการออกแบบทั้งสองวิธี กล่าวคือ Behavioral Description และ Data-Flow Description สนับสนุน Top-Down Design ส่วน Structural Description และการเรียกใช้ฟังก์ชันหรือแพคเกตก็คือ Bottom-Up Reuse นั่นเอง

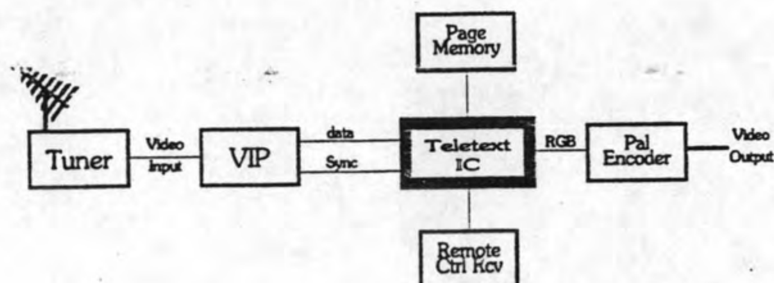
ภาษาวีเอชดีแอลเพิ่มประสิทธิภาพการออกแบบ

ดังได้กล่าวมาแล้วในบทที่ 2 ว่า การใช้ Generic Clauseในการเรียกใช้อุปกรณ์จะทำให้การวิเคราะห์วงจรใกล้กับความเป็นจริงได้มากขึ้น Generic Clause ยังสามารถเปลี่ยนแปลงคุณสมบัติของอุปกรณ์ได้ด้วยการส่งค่าเริ่มต้น ยกตัวอย่างเช่น การอธิบายการทำงานของ ROM เมื่อส่งชื่อไฟล์เริ่มต้นที่ไม่เหมือนกันก็จะได้ ROM ที่ไม่เหมือนกัน

การส่งผ่านพารามิเตอร์ของภาษาวีเอชดีแอลเป็นแบบไดนามิค ตัวคอมไพเลอร์จะไม่สนใจขนาดของพารามิเตอร์ โดยจะปล่อยให้เป็นหน้าที่ของซอฟต์แวร์วิเคราะห์และสังเคราะห์วงจรทำหน้าที่ตัดสินใจขณะทำงาน ยกตัวอย่างเช่นการเรียกใช้ ฟังก์ชัน "Compare(A,B)" ผู้ออกแบบไม่จำเป็นต้องทราบว่า A และ B มีขนาดกี่บิตเพราะไม่ว่า A และ B จะมีขนาด 2 บิต หรือ 40 บิต ก็สามารถเรียกใช้ฟังก์ชันเดียวกันนี้ได้ ซึ่งหากผู้ออกแบบใช้แผนภาพ Schematic จะไม่สามารถทำเช่นนี้ได้เพราะแผนภาพของตัวเปรียบเทียบขนาด 2 บิต กับ 40 บิตแตกต่างกันมาก

ภาษาวีเอชดีแอลมีลักษณะคล้ายกับภาษาแบบ Object-Oriented Programming ทำให้การแก้ไขทำได้ง่าย เช่น ในคำสั่ง FOR *i* IN Object'RANGE LOOP หรือ ฟังก์ชันที่ เรียกใช้โดยผ่านพารามิเตอร์ Object หากมีการแก้ไขขนาดหรือของตัวแปร Object ก็ไม่มีความจำเป็นใด ๆ ที่จะต้องแก้ไขโปรแกรม

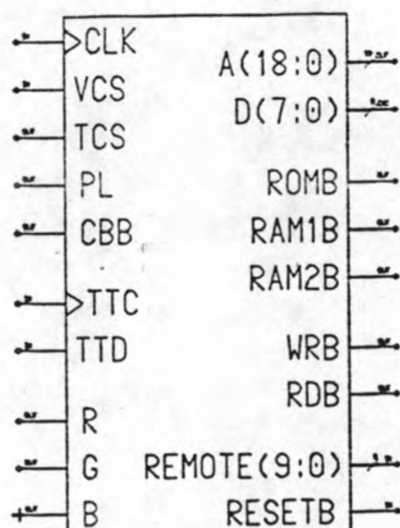
Top-Level Block of Teletext Decoder



รูปที่ 4-1 แสดงบล็อกไดอแกรมของเครื่องถอดรหัสเทเลเท็กซต์

จากรูปที่ 4-1 ที่ได้รับสัญญาณภาพที่มอดดูเลตมากับคลื่นวิทยุจากสายอากาศ เมื่อผ่าน Tuner ที่ปรับไว้ให้รับสัญญาณของช่อง 5 แล้วก็จะได้สัญญาณภาพพร้อมสัญญาณเทเลเท็กซต์ จากนั้นเมื่อสัญญาณผ่านไอซี Video Input Processor (VIP) แล้ว VIP จะกรองสัญญาณภาพทิ้งไปคงเหลือแต่สัญญาณเทเลเท็กซต์ ส่งให้ TELETEXT IC ประมวลผลและส่งออกไปเก็บในหน่วยความจำ Page Memory ในรูปของข้อมูลขนาน 8 บิตนอกจากนั้นวงจรรวมนี้ยังสามารถรับคำสั่งการแสดงผลจากตัวควบคุมรีโมท เพื่อนำข้อมูลใน Page Memory กลับมาประมวลผลและสร้างสัญญาณ RGB สำหรับแสดงภาพ ข้อมูลเทเลเท็กซต์บนจอภาพอีกด้วย

Top-Level Entity




```
ENTITY thesis IS
PORT(   clk, vcs   : IN std_ulogic;
        ttd,ttc    : IN std_ulogic;
        pl,cbb,tcs: OUT std_ulogic;
        R,G,B      : OUT std_ulogic;
        romb,ram1b,ram2b, rdb,wrb : OUT std_ulogic;
        D          : INOUT std_logic_vector(7 downto 0);
        A          : OUT std_ulogic_vector(18 downto 0);
        remote     : IN std_ulogic_vector(9 downto 0));
END thesis;
```

รูปที่ 4-2 แสดง Symbol และ Entity ของ Teletxt IC

จากรูปที่ 4-2 แสดง Symbol และ Entity ของ Teletxt IC ซึ่งมีสัญญาณดังนี้

สัญญาณ clk เป็นสัญญาณนาฬิกาของฟลิบฟลอบเกือบทุกตัวใน Teletext IC เป็นสัญญาณที่ได้มาจาก VIP

สัญญาณ vcs เป็นสัญญาณเข้า มีหน้าที่ในการทำให้ Teletext IC ทำงานเข้าจังหวะกับสัญญาณภาพในแนวตั้ง ได้มาจาก VIP

สัญญาณ pl และ cbb เป็นสัญญาณออก ที่ Teletext IC ส่งสัญญาณ pl กลับไปที่วงจร Phase-Locked Loop ของ VIP เพื่อทำให้ Teletext IC ทำงานเข้าจังหวะกับสัญญาณภาพในแนวนอน

สัญญาณ ttc และ ttd เป็นข้อมูลเข้าหลักของ Teletext IC ซึ่ง VIP ได้ถอดข้อมูลเทเลเท็กซต์ออกจากสัญญาณภาพรวมเกิดเป็นสัญญาณ ttd และได้มีการสร้างสัญญาณนาฬิกาที่มีความถี่เดียวกัน และเข้าจังหวะกับสัญญาณเทเลเท็กซต์ ก็คือสัญญาณ ttc ซึ่งข้อมูลที่ ttd จะมีค่าถูกต้องที่ขอบขาขึ้นของ ttc

สัญญาณ R,G,B และ tcs เป็นสัญญาณขาออกหลักของ Teletext IC ซึ่งจะส่งไปที่ Pal Encoder นำสัญญาณ RGB ซึ่งเป็นสัญญาณภาพสี มารวมกับสัญญาณ tcs ซึ่งเป็นสัญญาณรวมของสัญญาณเข้าจังหวะของสัญญาณภาพทั้งแนวนอนและแนวตั้ง มาทำให้เกิดสัญญาณภาพ

สัญญาณ adr และ data เป็นสัญญาณที่ติดต่อกับหน่วยความจำทั้งชนิด RAM ซึ่งเก็บข้อมูลเทเลเท็กซต์ และ ROM ซึ่งเก็บ font ของตัวอักษรภาษาไทย-อังกฤษ สัญญาณ adr เป็นสัญญาณออกมีหน้าที่บอกตำแหน่งข้อมูลในหน่วยความจำที่ Teletext IC ต้องการอ่านหรือเขียน ส่วน data เป็นเป็นทั้งสัญญาณเข้าและสัญญาณออก ซึ่งจะนำข้อมูลเทเลเท็กซต์ที่อ่านได้ไปเขียนใน RAM หรือจะนำข้อมูลที่เก็บไว้ใน RAM ในหน้าที่ผู้อ่านต้องการรับชมอ่านเข้ามาเพื่อแสดงภาพ รวมทั้งอ่าน font ของตัวอักษรจาก ROM ด้วย

สัญญาณ rdb, wrb, ram1b, ram2b และ romb เป็นสัญญาณออกเพื่อไปควบคุมการอ่านเขียนของ ROM และ RAM รวมทั้งทำการ Refresh RAM ด้วย เพราะว่า RAM ที่ใช้จะเป็นหน่วยความจำแบบ Pseudo Static RAM

สัญญาณ Remote เป็นสัญญาณเข้าจาก Remote Control Decoder เพื่อทำการเลือกหน้า

Top-Level Architecture

การเขียนภาษาในระดับนี้ใช้การอธิบายแบบ Behavioral ซึ่งมีไว้เพื่อใช้กำหนดขอบเขตของงาน และใช้ตรวจสอบกับการอธิบายในขั้นต่อ ๆ ไป และจะมีบางส่วนที่สามารถนำไปสังเคราะห์วงจรได้หากมีการแก้ไขปรับปรุงในระดับต่อไปเพียงเล็กน้อยเท่านั้น

เนื่องจากหน้าที่การทำงานของ Teletext IC มีหลายอย่าง จากการออกแบบและจากคู่มือ Enhanced Computer Controlled Teletext ได้แบ่งหน้าที่ออกเป็น 8 ชุด แต่ละชุดสามารถแบ่งออกจากกันได้ด้วยคำสั่ง PROCESS ดังรายชื่อต่อไปนี้

1. Main_Time_Base_Gen
2. Video_Synchronize
3. Shift_Data_In
4. Process_Data
5. Display_Time_Base_Gen
6. Decode_Command
7. RGB_Dot_Out
8. Memory_Remote_Interface

<u>1. Main_Time_Base_Gen</u> มีหน้าที่ในการสร้างสัญญาณนาฬิกาให้กับ Process อื่น ๆ เนื่องจากสัญญาณขาเข้าของ Teletext IC ที่เกี่ยวกับฐานเวลามีเพียงสัญญาณเดียวคือ สัญญาณ Clk ซึ่งมีความถี่ 6 MHz จึงต้องมีการสร้างฐานเวลาที่ความถี่ต่าง ๆ ดังนี้คือ 1 MHz, 15.625 kHz, และ 25 Hz ซึ่งสัญญาณที่ได้เกิดมาจากการหารความถี่ 6 MHz ด้วย 6, 64 และ 625 ตามลำดับ รูปที่ 4-3 แสดงโปรแกรมอธิบาย Main_Time_Base_Gen Process ในที่นี้มีการเรียกใช้ฟังก์ชัน increase ซึ่งมีหน้าที่ในการเพิ่มค่าของพารามิเตอร์ตัวที่หนึ่งไปหนึ่งค่าหากมันมีค่าน้อยกว่าพารามิเตอร์ตัวที่สอง แต่ถ้ามีค่าเท่ากันจะคืนค่าศูนย์

```
Main_Time_Base_Gen:
PROCESS (clk,rst_FieldSync)
        VARIABLE OneMeg_var   : INTEGER RANGE 0 to 5      := 0;
        CONSTANT OneMeg_max :INTEGER                      := 5;
        VARIABLE Line_var      : INTEGER RANGE 0 to 63     := 0;
```

```
        CONSTANT Line_max      : INTEGER                       := 63;
        VARIABLE Field_var     : INTEGER RANGE 0 to 312        := 0;
        CONSTANT Field_max     : INTEGER                       := 312;
        VARIABLE even          : BOOLEAN                       := FALSE;
BEGIN
        IF rst_FieldSync THEN
                -- after this double dash is comment of next line.
                -- async reset for synchronization with vertical sync.
                Field_var := 0;
        ELSIF rising_edge(clk) THEN
                -- divide by 6 to get 1 MHz
                increase(OneMeg_var,OneMeg_max);
                IF OneMeg_var=5 THEN
                        -- divide by 64 to get line rate = 15.625 kHz
                        increase (Line_var,Line_max);
                ELSIF OneMeg_var = 2 AND Line_var = 48 THEN
                        -- divide by 313 if even field else divide by 312 (odd field)
                        IF even THEN
                                -- divide by 313
                                increase(Field_var,Field_max);
                        ELSE    -- divide by 312
                                increase(Field_var,Field_max-1);
                        END IF;
                        IF Field_var = 310 THEN
                                -- toggle odd and even field
                                even := not even;
                        END IF;
                END IF;
        END IF;
        OneMeg <= OneMeg_var;
        Line   <= Line_var;
        Field  <= Field_var;
END PROCESS Main_Time_Base_Gen;
```

รูปที่ 4-3 แสดงโปรแกรมส่วน Main_Time_Base_Gen Process.

2. <u>Video_Synchronize</u> มีหน้าที่ในการเข้าจังหวะกับสัญญาณภาพทั้งแนวนอนและแนวตั้ง การเข้าจังหวะในแนวนอนทำได้โดยส่งสัญญาณ pl กลับไปควบคุมวงจร Phased-Lock Loop ในตัว VIP ส่วนในแนวตั้งทำได้โดยตรวจจับสัญญาณเข้าจังหวะแนวตั้งจากสัญญาณ vcs ซึ่งสัญญาณ vcs จะประกอบด้วยสัญญาณเข้าจังหวะทั้งในแนวนอนและแนวตั้งปนกันอยู่ ทั้งนี้อาศัยหลักที่ว่าขณะที่ไม่มีสัญญาณเข้าจังหวะแนวตั้ง สัญญาณ vcs จะมีระดับลอจิกเป็นศูนย์ยาวนานกว่าหนึ่งมาก ในทางกลับกันหากขณะที่มีสัญญาณเข้าจังหวะแนวตั้งแล้ว สัญญาณ vcs จะมีระดับลอจิกเป็นหนึ่งยาวนานกว่าศูนย์มากเช่นกัน

```
Video_Synchronize:
PROCESS (clk)
        VARIABLE   FieldSync_Count : INTEGER:=0;
        CONSTANT  FieldSync_Count_max : INTEGER := 63;
        CONSTANT  Threshold : INTEGER:=37;
BEGIN
        IF rising_edge(clk) AND OneMeg=5 THEN
                -- phase lock and color burst band sent back to VIP
                pl <=     (Line > 17) AND (Line <= 49);
                cbb <=  (Line > 17) AND (Line <= 56);
                IF (vcs='1') THEN
                        increase (FieldSync_Count,FieldSync_Count_max);
                ELSE
                        IF FieldSync_Count >0 THEN
                                decrease(FieldSync_Count);
                        END IF;
                END IF;
        END IF;
        -- generate rst_FieldSync if '1' >> '0'  (FieldSync_Count > Threshold)
        Rst_fieldSync <= (FieldSync_Count >Threshold);
END PROCESS Video_Synchronize;
```

รูปที่ 4-4 แสดงโปรแกรมในการอธิบาย Video_Synchronize Process

3. Shift_Data_In มีหน้าที่ในการรับสัญญาณเทเลเท็กซต์ ttd ที่เป็นสัญญาณอนุกรม แปลงให้เป็นสัญญาณขนานแปดบิต รหัสเริ่มต้นของข้อมูลเทเลเท็กซต์แต่ละชุดคือ รหัสเฟรมมิง (ดูรายละเอียดในบทที่ 3) ซึ่งเมื่อได้รับรหัสเฟรมมิง สัญญาณดังกล่าวจะไปรีเซตฟลิบฟลอบที่นับการเก็บข้อมูลทั้งหมด แล้วเริ่มนับตำแหน่งข้อมูล อันประกอบด้วยการนับตำแหน่งบิต (นับแปดบิต) และนับตำแหน่งไบต์ของข้อมูลที่ได้รับขณะนั้น

เนื่องจากสัญญาณเทเลเท็กซต์มีความถี่ 6.9375 Mbits/sec หรือ ใช้เวลาในการส่งตัวอักษรขนาดแปดบิตด้วยเวลา 1.15 μsec แต่ความถี่ของระบบเท่ากับ 1 MHz หากสุ่มอ่านข้อมูลทุกๆ 1 μsec จะทำให้ได้ข้อมูลซ้ำเดิมทุกการอ่าน ๆ 5-6 ครั้ง ดังนั้นเพื่อไม่ให้ค่าของตัวนับตำแหน่งของข้อมูลมีค่าเพิ่มขึ้นในกรณีที่อ่านข้อมูลซ้ำตัวเดิม จึงจำเป็นต้องมีการตรวจสอบว่าได้อ่านข้อมูลตัวนั้นไปหรือยังด้วย (ดูที่สัญญาณ Repeat)

```
Shift_Data_In:
        PROCESS (ttc,clk)
                VARIABLE parallel          : std_ulogic_vector(7 downto 0);
                VARIABLE Teletext_Async: std_ulogic_vector (7 downto 0);
                VARIABLE BitCount          : INTEGER := 0;
                VARIABLE Repeat            : BOOLEAN :=FALSE;
                CONSTANT BitCount_max :INTEGER :=7;
        BEGIN
                IF rising_edge(ttc) THEN
                        -- Shift in Teletext Data
                        parallel := ttd & parallel (7 downto 1);
                        -- Check if it is framming code or not .
                        FramingCode := (parallel = 0x27) AND (Line >60 AND Line <62);
                        -- If  yes, reset all positioned flip-flop
                        IF FramingCode THEN
                                BitCount :=0;
                                Column_w <= -3;
                                Repeat  := TRUE;
                        ELSE
                                increase (BitCount,BitCount_max);
                                IF BitCount = BitCount_max THEN
                                        -- Store 8 bit Data
```

```
                        Teletext_async := parallel;
                        -- Declare New Data
                        Repeat := FALSE;
                END IF;
            END IF;
        END IF;
        IF rising_edge(clk) AND OneMeg = 5 THEN
            IF not Repeat THEN
                -- increase Byte Count
                increase (Column_w,Column_max);
                -- Read Data
                TeletextData <= Teletext_Async;
            END IF;
            -- Declare Data has been read.
            Repeat := True;
        END IF;
    END PROCESS Slice_Teletext_Data;
```

รูปที่ 4-5 แสดงโปรแกรมในการอธิบาย Shift_Data_In Process

4. Process Data มีหน้าที่ในการตรวจสอบความถูกต้องของข้อมูลที่ได้รับ จัดเก็บข้อมูลที่เป็นตำแหน่งของข้อมูลบนจอภาพเช่นเลขที่หน้าของข้อมูล แถวของข้อมูลบนจอภาพ รวมทั้งตีความหมายว่าข้อมูลที่ได้รับเป็นข้อความภาษาไทยหรือภาษาอังกฤษด้วย

ในข้อมูลแถวที่ 0 (Header Row) จะประกอบด้วยข้อมูลที่ใช้บอกตำแหน่งของข้อมูลบนจอภาพสี่ไบต์คือเลขที่แมกกาซีนและแถวสองไบต์กับเลขที่หน้าอีกสองไบต์ ส่วนแถวอื่น ๆ มีเพียงสองไบต์คือเลขที่แมกกาซีนและแถว โดยจะถือว่าเลขที่หน้าเป็นเลขเดิมจนกว่าจะได้รับข้อมูลแถวที่ศูนย์ใหม่ ข้อมูลประเภทนี้มีความสำคัญมาก เพราะหากได้รับข้อมูลผิดไป จะทำให้ข้อมูลที่ตามมาจะถูกเขียนในตำแหน่งที่ไม่ถูกต้อง ดังนั้นจึงจำเป็นต้องใช้รหัสแฮมมิง และหากพบว่ารหัสแฮมมิงตรวจสอบว่าข้อมูลประเภทนี้ไม่ถูกต้อง จะไม่รับข้อมูลที่ตามมาเลย การตรวจสอบนี้ทำได้โดยการเก็บความถูกต้องของการถอดรหัสแฮมมิงของเลขที่หน้าไว้ใน PageAccept และของแมกกาซีนกับแถวของข้อมูลไว้ใน LineAccept โดยที่สัญญาณในการควบคุมการเขียนข้อมูลคือ WriteEnable จะมีค่าเป็นหนึ่งก็ต่อเมื่อ ทั้ง LineAccept และ PageAccept มีค่าเป็นหนึ่ง

ในการตรวจชนิดของภาษาของหน้าใด ๆ อาศัยข้อมูลของข้อมูลแถวที่ 1 ในตำแหน่งแรก หากข้อมูลหน้านั้นเป็นภาษาไทยข้อมูลในแถวที่หนึ่งตำแหน่งที่แรกจะต้องมีค่าเป็น 0Ah หาก

เป็นค่าอื่น จะถือว่าเป็นหน้าภาษาอังกฤษ ซึ่งหากพบว่าข้อมูลเป็นภาษาไทยจะเปลี่ยนบิตสุดท้าย (MSB) ของข้อมูลซึ่งแสดงชนิดของภาษาที่ตามมาให้เป็นหนึ่ง (ดูรายละเอียดในบทที่ 3) อย่างไรก็ตามข้อความในหน้าใดหน้าหนึ่ง อาจมีความจำเป็นที่ต้องใช้ทั้งภาษาไทยและภาษาอังกฤษ จึงจำเป็นต้องมีรหัสสลับภาษา (Twist Code) ซึ่งมีค่าเท่ากับ 1Bh ซึ่งมีความสามารถสลับสถานะลอจิกของบิตสุดท้ายเหมือนกัน แต่จะมีผลในหนึ่งบรรทัดข้อมูลเท่านั้น รูปที่ 4-6 แสดงโปรแกรมในการอธิบาย Process Data

```
Process_Data:
    PROCESS(clk,TeletextData)
        FUNCTION parity_check(x:std_ulogic_vector) RETURN BOOLEAN;
        PROCEDURE hamming_check(x: IN std_ulogic_vector(7 DOWNTO 0)
                                y: OUT std_ulogic_vector(3 DOWNTO 0)
                                ok: OUT BOOLEAN);
        TYPE language IS (ENGLISH,THAI);
        VARIABLE PageLanguage,LanguageBit : language:=ENGLISH;
        VARIABLE PageUnit,PageTen            := std_ulogic_vector (3 downto 0);
        VARIABLE Parity_ok,Hamming_ok        := BOOLEAN ;
        VARIABLE LineAccept,PageAccept       := BOOLEAN ;
        VARIABLE HammingData                 := std_ulogic_vector (3 downto 0);
        CONSTANT THAICODE  := std_ulogic_vector (7 downto 0):="00001010"; --0AH
        ONSTANT TWISTCODE := std_ulogic_vector (7 downto 0);="00011011";--1BH
    BEGIN
        -- Function Parity Call
        Parity_Ok := parity_check(TeletextData);
        -- Procedure Hamming Check Call, two results return, HammingData and Hamming_ok
        hamming_check(TeletextData,HammingData,Hamming_Ok);
        IF rising_edge(clk) THEN
            -- get magzine and row data from every frame of data
            IF Column_w=-2 THEN
                magazine <= HammingData(2 Downto 0);
                row_w <=  to_int(HammingData(3));
                LineAccept := Hamming_ok;
                -- clear the result of TWISTCODE every begin of line.
```

```
                LanguageBit:=PageLanguage;
        ELSIF Column_w=-1 THEN
                row_w <= to_integer(HammingData)*2+row_w;
                LineAccept := LineAccept AND Hamming_ok;
        END IF;
        -- get page of data only if row=0
        IF row_w = 0 THEN
                IF Column_w=0 THEN
                        PageUnit := HammingData;
                        PageAccept :=Hamming_ok AND LineAccept;
                ELSIF Column_w=1 THEN
                        PageTen := HammingData;
                        PageAccept := PageAccpet AND Hamming_ok;
                END IF;
        -- Check Language of Page when row = 1, col = 0
        ELSIF row_w = 1 AND Column_w=0 THEN
                IF TeletextData=THAICODE THEN
                        PageLanguage := THAI;
                        LanguageBit := THAI;
                ELSE
                        PageLanguage := ENGLISH;
                        LanguageBit := ENGLISH;
                END IF;
        END IF;
        -- Check if TeletextData=TWISTCODE or not, if yes, swap content of
        -- Language bit.
        IF TeletextData = TWISTCODE THEN
                IF LanguageBit = ENGLISH THEN
                        LanguageBit := THAI;
                ELSE
                        LanguageBit := ENGLISH;
                END IF;
        END IF;
END IF;
```

```
-- Prepare Data to write to RAM
IF LanguageBit = ENGLISH THEN
        Data_w <= '0'&TeletextData(6 downto 0);
ELSE
        Data_w <= '1'&TeletextData(6 downto 0);
END IF;

-- Write Control, Check all Conditions
WriteEnable <= Parity_ok AND LineAccept AND PageAccept
                AND Column_w >= 0 AND Column_w <40;

-- mapping two bcd codes (8bits) to one binary code (7 bits) to reduce unused addresses
-- in RAM
IF PageUnit(3)='0' THEN
        Page_w <= PageTen&PageUnit(2 downto 0) ;
ELSE
        Page_w <= "11"&PageTen(1 downto 0)&PageTen(3 downto 2)&PageUnit(0);
END IF;

END PROCESS Process_Data;
```

รูปที่ 4-6 แสดงส่วนโปรแกรม Process_Data

5. Display Time Base Gen มีหน้าที่ในการนับจำนวนแถวของเส้นของตัวอักษรซึ่งหนึ่งตัวอักษรจะใช้ เส้นของจอภาพ 10 เส้น และนับจำนวนแถวของข้อความ โดยหนึ่งจอภาพจะมีข้อความมากที่สุดได้ 25 แถว รวมทั้งสร้างความถี่ที่ตัวอักษรจะกระพริบเมื่อมีคำสั่งให้ตัวอักษรนั้นกระพริบด้วย

```
Display_Time_Base:
    PROCESS (clk,Field)
        VARIABLE Row_r_var          : INTEGER ;
        CONSTANT Row_r_max          : INTEGER :=31;
```

I16724975

```
        VARIABLE CharRow_var            : INTEGER ;
        CONSTANT CharRow_max            : INTEGER :=9;
        VARIABLE Blink_Count            : INTEGER ;
        CONSTANT Blink_Count_max        : INTEGER :=31;
BEGIN
        IF Field = 40 THEN
                -- Display Window Start from Field=40 to 290
                Row_r_var := 0;
                CharRow_var := 0;
        ELSIF rising_edge(clk)  AND OneMeg=5 AND Line = 48 THEN
                -- divide by 10 to get Row of Character Font
                increase (CharRow_var,CharRow_max);
                IF CharRow_var = CharRow_max THEN
                        -- then divide by 25 to get Row of Data to read
                        increase (Row_r_var,Row_rmax);
                        IF (Row_r_var = Row_r_max) THEN
                                -- then divide by 32 to get Blink rate ( about 1.28 sec )
                                increase (Blink_Count,Blink_Count_max);
                        END IF;
                END IF;
        Row_r <= Row_r_var;
        CharRow <= CharRow_var;
        BlinkPeriod <= (Blink_Count >16);
        END IF;
END PROCESS Display_Time_Base_Gen;
```

รูปที่ 4-7 แสดงโปรแกรม Display_Time_Base_Gen

6. <u>Decode Command</u> มีหน้าที่ในการตรวจจับและตีความหมายของข้อมูลที่อ่านมาจากหน่วยความจำภายนอก หากข้อมูลที่ได้รับเป็นข้อมูลคำสั่ง และเก็บคำสั่งเหล่านั้นไว้ใน Register ต่าง ๆ แต่ถ้าเป็นข้อมูลกราฟฟิคก็จะส่งถอดรหัสให้เป็นข้อมูลจุดที่จะแสดงบนจอภาพ หรือถ้าเป็นข้อมูลตัวอักษร ก็จะส่งข้อมูลนั้นไปรับ Font จากหน่วยความจำภายนอกเพื่อให้ได้ข้อมูลจุดที่จะแสดงบนจอภาพอีกครั้งหนึ่ง

```
Command_Decode:
PROCESS(clk,Line)
        VARIABLE  Command              : INTEGER RANGE 0 TO 127;
        -- Table of Command
        CONSTANT ALPHA_BLACK           : INTEGER := 0X00;
        CONSTANT ALPHA_RED             : INTEGER := 0X01;
        CONSTANT ALPHA_GREEN           : INTEGER := 0X02;
        CONSTANT ALPHA_YELLOW          : INTEGER := 0X03;
        CONSTANT ALPHA_BLUE            : INTEGER := 0X04;
        CONSTANT ALPHA_MAGENTA         : INTEGER := 0X05;
        CONSTANT ALPHA_CYAN            : INTEGER := 0X06;
        CONSTANT ALPHA_WHITE           : INTEGER := 0X07;
        CONSTANT GRAPHICS_BLACK        : INTEGER := 0X10;
        CONSTANT GRAPHICS_RED          : INTEGER := 0X11;
        CONSTANT GRAPHICS_GREEN        : INTEGER := 0X12;
        CONSTANT GRAPHICS_YELLOW: INTEGER := 0X13;
        CONSTANT GRAPHICS_BLUE         : INTEGER := 0X14;
        CONSTANT GRAPHICS_MAGENTA:INTEGER:= 0X15;
        CONSTANT GRAPHICS_CYAN         : INTEGER := 0X16;
        CONSTANT GRAPHICS_WHITE        : INTEGER := 0X17;
        CONSTANT FLASH                 : INTEGER := 0X08;
        CONSTANT STEADY                : INTEGER := 0X09;
        CONSTANT BLACK_BACKGROUND : INTEGER := 0X1C;
        CONSTANT NEW_BACKGROUND: INTEGER :=0X1D;
        CONSTANT HOLD_GRAPHICS         : INTEGER :=0X1E;
        CONSTANT RELEASE_GRAPHICS: INTEGER:=0X1F;
  BEGIN
-- Assume Teletext Code is valid at OneMeg = 1-2  Font of Character is valid at OneMeg=3-5.
--Teletext Code consists of :
```

```
--          1. Commands (0x00-0x1F)
--          2. ASCII Code of some Characters in English (0x20-0x7F), TLI Code in Thai (0xA0-0xFF)
--          OR Graphics Code (0x20-0x3F AND 0x60-0x7F), Selected by Command.
                IF Line = 63 THEN
                        -- New Line Reset Command
                        -- foreground color = white
                        ForeColor <= white;
                        -- background color = black
                        BackColor <= black;
                        -- Display = Alpha
                        AlphaGraphics <= alpha;
                        -- no blink
                        Blink <= FALSE;
                        -- release graphics
                      · HoldGraphics := FALSE;
                ELSIF rising_edge(clk) THEN
                        IF OneMeg = 1 THEN
                                -- Store Command, some need to decode later.
                                Command := to_integer(data_r(6 downto 0));
                                -- Decode Some Command
                                CASE Command IS
                                WHEN HOLD_GRAPHICS =>
                                        GraphicsHold := TRUE;
                                WHEN RELEASE_GRAPHICS =>
                                        GraphicsHold := FALSE;
                                WHEN    ALPHA_BLACK | ALPHA_RED | ALPHA_GREEN |
                                ALPHA_YELLOW | ALPHA_BLUE | ALPHA_CYAN
                        | ALPHA_MAGENTA | ALPHA_WHITE =>
                                        AlphaGraphics <= alpha;
                                WHEN    GRAPHICS_BLACK | GRAPHICS_RED |
                                        GRAPHICS_GREEN | GRAPHICS_YELLOW |
                                        GRAPHICS_BLUE | GRAPHICS_MAGENTA |
                                        GRAPHICS_CYAN | GRAPHICS_WHITE =>
                                        AlphaGraphics := graphics;
```

```
                WHEN OTHERS => NULL;
                END CASE;
        ELSIF OneMeg=2 AND not GraphicsHold THEN
                -- Generate Graphics Character
                CASE CharRow IS
                WHEN 0 | 1 | 2 =>
                        GraphicsDot := Data_r(1 downto 0);
                WHEN 3 | 4 | 5 |6
                        GraphicsDot := Data_r(3 downto 2);
                WHEN 7 | 8 | 9 =>
                        GraphicsDot := Data_r(6)&Data_r(4);
                WHEN OTHERS =>
                        ASSERT FALSE REPORT "CharRow is Out of Range";
                END CASE;
        ELSIF OneMeg = 3 THEN
                -- Decode Color And Blink Information.
                CASE Command IS
                WHEN ALPHA_BLACK | GRAPHICS_BLACK =>
                        ForeColor <= black;
                WHEN ALPHA_RED | GRAPHICS_RED =>
                        ForeColor <= red;
                WHEN ALPHA_GREEN | GRAPHICS_GREEN =>
                        ForeColor <=green;
                WHEN ALPHA_YELLOW | GRAPHICS_YELLOW =>
                        ForeColor <= yellow;
                WHEN ALPHA_BLUE | GRAPHICS_BLUE =>
                        ForeColor <= blue;
                WHEN ALPHA_MAGENTA | GRAPHICS_MAGENTA =>
                        ForeColor <= magenta;
                WHEN ALPHA_CYAN | GRAPHICS_CYAN =>
                        ForeColor <= cyan;
                WHEN ALPHA_WHITE | GRAPHICS_WHITE =>
                        ForeColor <= white;
                WHEN FLASH =>
```

```
                        Blink <= TRUE;
                WHEN STEADY =>
                        Blink <= FALSE;
                WHEN NEW_BACKGROUND =>
                        -- send foreground color to background color.
                        BackColor <= ForeColor;
                WHEN BLACK_BACKGROUND =>
                        -- clear background color to black.
                        BackColor <= black;
                WHEN OTHERS =>
                        NULL;
                END CASE;
        END IF;
    END IF;
END PROCESS Command_decode;
```

รูปที่ 4-8 แสดงส่วนของโปรแกรม Command_Decode

7. RGB Dot Out มีหน้าที่ในการส่งข้อมูลที่เป็นจุดที่จะแสดงบนจอภาพที่เป็นแบบขนานให้เป็นแบบอนุกรม รวมทั้งสร้างสัญญาณสี กระพริบตัวอักษร ตามที่มีคำสั่ง

```
RGB_Dot_Out: PROCESS (clk)
        VARIABLE Parallel          : std_ulogic_vector (11 downto 0);
        VARIABLE DotColor: color;
BEGIN
        -- two phase shift
        IF rising_edge(clk) or falling_edge(clk) THEN
                IF OneMeg=4 AND rising_edge(clk) THEN
                        -- Synchronous Load data
                        IF AlphaGraphics = graphics AND Data_r(5) = '1' THEN
                                -- Select Graphics to show when AlphaGraphics = graphics
                                -- And Control bit is high.
                                Parallel(11 downto 6) := (OTHERS => GraphicsDot(0));
```

```
                    Parallel(5 downto 0)  := (OTHERS => GraphicsDot(1));
               ELSE
                    --Select 8-bit Font of a Character and add pre and post blank.
                    Parallel := "00"&Data_r&"00";
               END IF;
          ELSE
               -- Shift Data when not load
               Parallel := Parallel sll 1;
          END IF;
     END IF;
     Dot := Parallel(11);
     R <= '0'; G <= '0'; B <= '0';
     -- show backgroud color when dot is '0' or blink
     IF dot = '0' OR (Blink AND BlinkPeriod)  THEN
          DotColor := BackColor;
     ELSE
          DotColor := ForeColor;
     END IF;
     -- window horizontal from column 0-39 and vertical from row 0-25
     IF (Line >= 0 AND Line < 40) AND (Row_r >= 0 AND Row_r < 25) THEN
          CASE DotColor IS
          -- convert color type to std_ulogic type in R,G,B
          WHEN black =>       R <= '0';      G <= '0';      B <= '0';
          WHEN red =>         R <= '1';      G <= '0';      B <= '0';
          WHEN green =>       R <= '0';      G <= '1';      B <= '0';
          WHEN yellow =>      R <= '1';      G <= '1';      B <= '0';
          WHEN blue =>        R <= '0';      G <= '0';      B <= '1';
          WHEN magenta =>     R <= '1';      G <= '0';      B <= '1';
          WHEN cyan =>        R <= '0';      G <= '1';      B <= '1';
          WHEN white =>       R <= '1';      G <= '1';      B <= '1';
          END CASE;
     END IF;
END PROCESS RGB_Dot_Out;
```

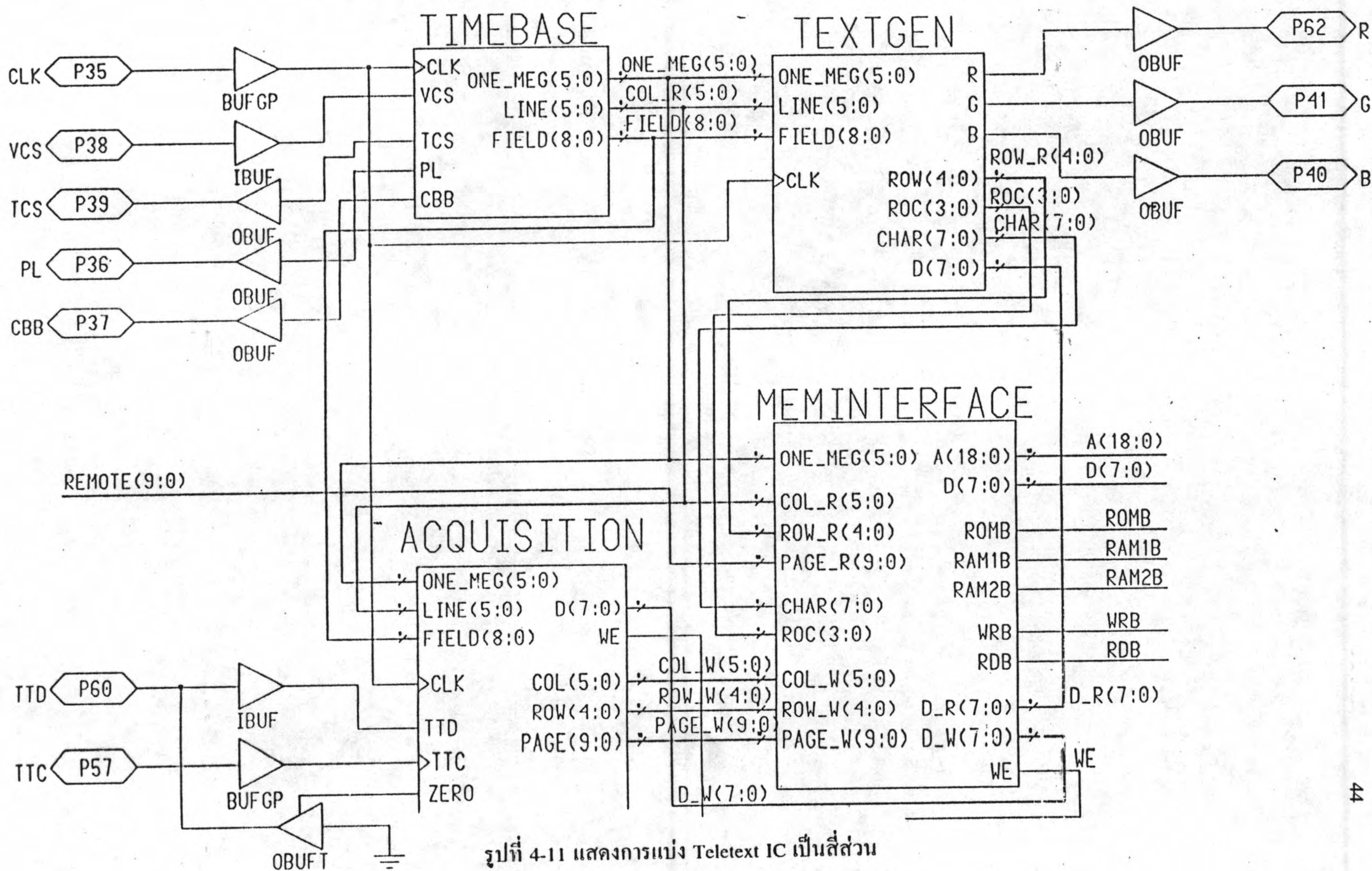
รูปที่ 4-9 แสดงส่วนของโปรแกรม RGB_Dot_Out

8. <u>Memory_Remote_Interface</u> มีหน้าที่ในการติดต่ออ่านเขียนข้อมูลกับหน่วยความจำ โดยกำหนดช่วงเวลาในการอ่านหรือเขียน RAM หรืออ่าน ROM และการแปลงตำแหน่งของข้อมูล เช่น row หรือ column ให้เป็นตำแหน่งของหน่วยความจำ (Address) อีกทั้งยังรับข้อมูลจาก Remote Control Decoder มาเลือกหน้าที่ต้องการแสดงผลด้วย

```
Memory_Remote_interface:
    PROCESS(row_r,row_w,line,column_w,page_r,page_w,char,roc,OneMeg,remote)
        VARIABLE    page        : std_ulogic_vector(9 downto 0) ;
        VARIABLE    column      ; std_ulogic_vector(5 downto 0);
        VARIABLE    row         : std_ulogic_vector(4 downto 0);
        VARIABLE    adr_char : std_ulogic_vector(19 downto 0);
    BEGIN
        -- Read Teletext Code from RAM when OneMeg=0-1
        IF (OneMeg=0 OR OneMeg=1) THEN
            column := To_StdUlogicVector(column_r,column'LENGTH);
            row := To_StdUlogicVector(row_r,row'LENGTH);
            page := page_r;
        -- Write Received data to RAM when OneMeg=3-5
        ELSE
            column := To_StdUlogicVector(column_w,column'LENGTH);
            row := To_StdUlogicVector(row_w,row'LENGTH);
            -- if row = 0 write to display page to update time
            IF (row_w /=0) THEN
                page := page_w;
            ELSE
                page := page_r;
            END IF;
        END IF;
        -- mapping column, row and page to address
        adr_char(2 downto 0) := column(2 downto 0);
        adr_char(19 downto 10) := page(9 downto 0);
        CASE column(5) IS
            WHEN '0' =>
```

```
adr_char(9 downto 3) := row(4 downto 0) & column(4 downto 3);
WHEN OTHERS =>
adr_char(9 downto 3) := "11" & not (row(4 downto 0));
END CASE;
-- Read Font from ROM when OneMeg = 2
IF (OneMeg/=2) THEN
ADR <= adr_char(18 downto 0);
ELSE
ADR <= "00000100" & char & RoC;
END IF;
-- hold external Data Bus if want to write Data
IF (OneMeg = 4 OR OneMeg = 5) THEN
Data <= (std_logic_vector) Data_w;
Else
Data <= (OTHERS <= 'Z');
END IF;
-- Read external data
IF (OneMeg'Event AND (OneMeg = 1 OR OneMeg = 3) THEN
Data_r <= (std_ulogic_vector) Data;
END IF;
Ram1b <= adr_char(19) OR ( OneMeg /=0 AND OneMeg /=4);
Ram2b <= not adr_char(19) OR (OneMeg/=0 AND OneMeg/=4);
Rdb <= OneMeg/=0 AND OneMeg/=2;
Wrb <= OneMeg/=4;
Romb <= OneMeg/=2;
IF (Remote'Event And Remote/="0000000000") THEN
mag_r := PageTen(2 downto 0);
PageTen := PageUnit;
PageUnit := One_Hot (Remote);
END IF;
END PROCESS ;
```

รูปที่ 4-10 แสดงส่วนโปรแกรม Memory_Remote_Interface



รูปที่ 4-11 แสดงการแบ่ง Teletext IC เป็นสี่ส่วน

<u>การออกแบบลงไปลำดับขั้นแรก (First Decomposition)</u>

หลังจากได้ออกแบบในระดับ Top-Level ซึ่งใช้การอธิบายแบบเชิงพฤติกรรมซึ่งได้แบ่งการอธิบายเป็น 8 Process ทำให้พิจารณาแบ่งกลุ่ม Process เป็นอุปกรณ์ได้ 4 กลุ่มคือ

1. Time_Base
2. Acquisition
3. Character_Gen
4. Memory_Remote_Interface

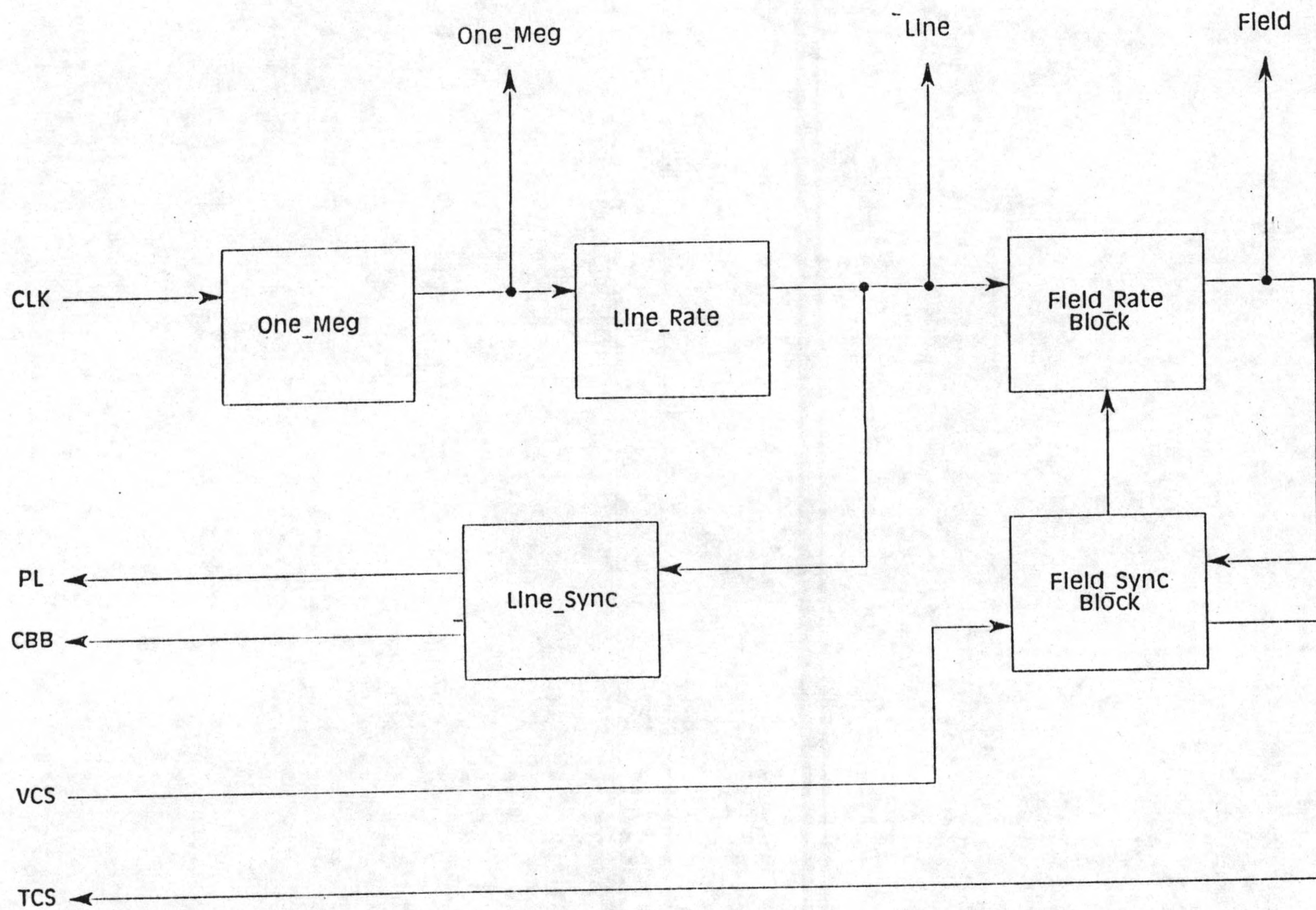
การอธิบายในขั้นนี้ใช้การอธิบายแบบ Data-Flow ทำให้เมื่ออธิบายและวิเคราะห์วงจรเรียบร้อยแล้วจะสามารถสังเคราะห์วงจรได้ในทันที

การอธิบายบางส่วนของโปรแกรมนำมาจากการอธิบายเชิงพฤติกรรม มาปรับปรุงเพียงเล็กน้อยก็สามารถสังเคราะห์วงจรได้ บางส่วนอธิบายในขั้นนี้คือส่วนที่ใช้คำสั่ง PROCESS บางส่วนอาจมีการแบ่งการออกแบบลงไปอีกลำดับชั้นหนึ่งคือส่วนที่ใช้ในคำสั่ง BLOCK แล้วจึงใช้คำสั่ง PROCESS ออกแบบในลำดับขั้นนั้นอีกทีหนึ่ง (Second Decomposition) บล็อกไดอะแกรมของ Teletext IC แสดงในรูปที่ 4-11 รายละเอียดการอธิบายวงจรทั้งสี่จะมีดังหัวข้อต่อๆ ไป

<u>Time Base</u>

วงจรส่วน Time_Base เกิดจากการรวม Process สองอันเข้าด้วยกันคือ Main_Time_Base_Gen และ Video_Synchonize จากนั้นจึงได้พิจารณาแยก Process ทั้งสองเป็นส่วนย่อย ๆ ดังบล็อกไดอะแกรมในรูปที่ 4-12

<u>1. One_Meg Process</u> ทำหน้าที่กำเนิดสัญญาณ OneMeg ซึ่งสัญญาณดังกล่าวจะเป็นฐานเวลาความถี่ 1 MHz และมี 6 phase ซึ่งสามารถทำได้สองวิธีคือการออกแบบได้ใช้การหมุนวนของระดับสัญญาณลอจิกหนึ่งไปใน Flip-Flop 6 ตัวที่ต่อเชื่อมกันอยู่เป็นลูกโซ่กับการใช้วงจรนับ 6 จากการพิจารณาได้เลือกใช้วงจรหมุนวนเนื่องจากใช้ Flip-Flop เพียงแค่หกตัว เมื่อเปรียบเทียบกับวงจรนับซึ่งใช้ Flip-Flop 3 ตัวและต้องมีการถอดรหัสค่า 0-6 (0-5 เป็นเอาท์พุท ส่วน 6 เป็นสัญญาณรีเซต) จะประหยัดกว่าและเร็วกว่าด้วย (ที่ความถี่ 6 MHz วงจรนับหกก็ยังทำงานได้อยู่) รูปที่ 4-13 แสดงโปรแกรม OneMeg Process คำสั่ง " OneMeg_i <= OneMeg_i RLL 1; " แสดงให้เห็นถึงการใช้วงจรหมุนวนซ้าย สังเกตว่าขนาดของวงจรที่สังเคราะห์ได้จะขึ้นกับขนาดของ OneMeg ซึ่งมีลักษณะคล้ายกับหลักการของ Object-Oriented Program

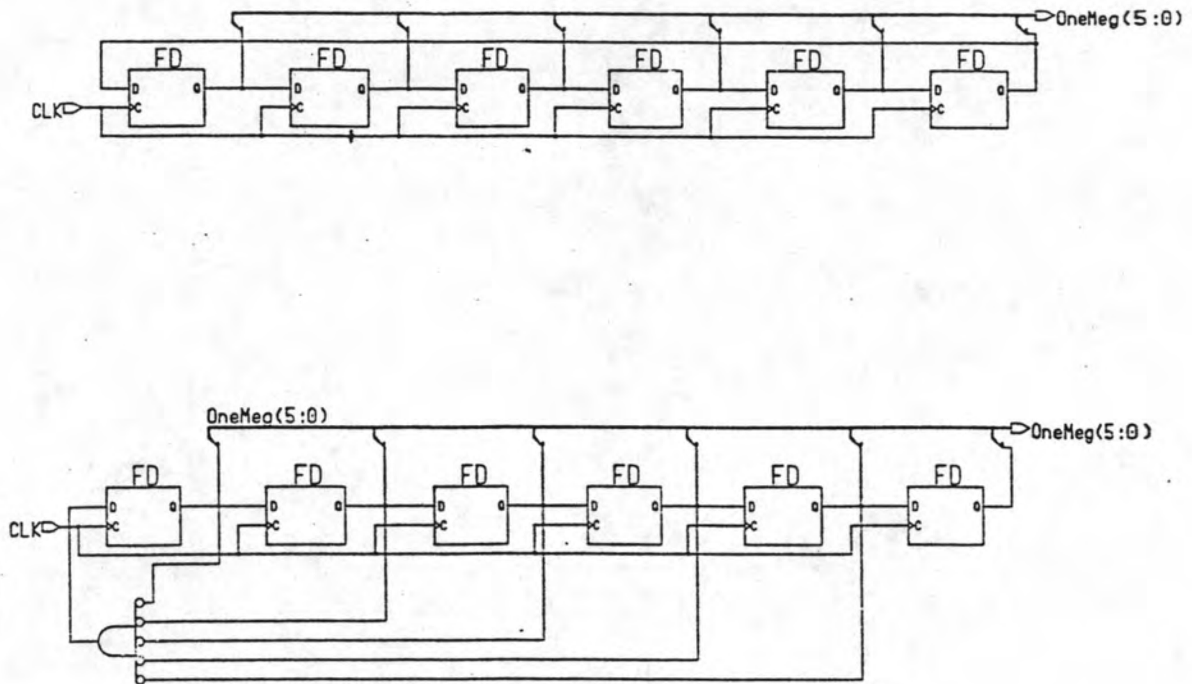


รูปที่ 4-12 แสดงบล็อกไดอะแกรมของ Time_Base

```
One_Meg: PROCESS(clk )
BEGIN
        IF (rising_edge(clk)) THEN
                OneMeg_i  <= OneMeg_i RLL 1 ;
        END IF;
END PROCESS One_Meg;
```

รูปที่ 4-13 แสดงส่วนโปรแกรมของ One_Meg Process

ปัญหา วงจรส่วนนี้ไม่น่าจะมีปัญหาอะไร เนื่องจากเป็นวงจรที่ง่ายและไม่ซับซ้อน เมื่อเกิดปัญหาขึ้นในทางฮาร์ดแวร์ผู้ออกแบบจึงไม่ได้คำนึงถึงวงจรส่วนนี้เลย รวมทั้งวงจรส่วน Time_Base นี้ได้เขียนโปรแกรมพร้อมกับสังเคราะห์เป็นฮาร์ดแวร์เป็นอันดับแรก และผ่านการทดสอบทั้งทางซอฟต์แวร์และฮาร์ดแวร์แล้ว แต่ปัญหาได้เกิดขึ้นเมื่อได้มีการประกอบวงจรส่วนอื่นๆ เข้าไปมากแล้ว หลังจากได้ศึกษาปัญหาอย่างจริงจัง ผู้ออกแบบจึงพบว่าปัญหาเกิดจากสัญญาณ OneMeg ที่เป็นฐานเวลาหลักทั้งในวงจรส่วน Time_Base เองและ ส่วนวงจรอื่น ๆ มีค่า Fan-Out น้อยกว่าที่มันไปขับเกตตัวอื่น ๆ (Fan-Out ของ Xilinx FPGA มีค่าประมาณ 4-5) ทำให้วงจรอาจทำงานผิดพลาดได้ในชั่วเวลาเพียง 2-3 นาที กล่าวคือโดยปกติ Flip-Flop ทั้งหกตัวจะมีระดับสัญญาณลอจิกที่เป็นหนึ่งเพียงแค่ตัวเดียวและจะทำการส่งค่าหนึ่งไปให้ Flip-Flop ตัวถัดไปเมื่อได้รับสัญญาณ clk แต่เมื่อสัญญาณดังกล่าวไปขับเกตตัวอื่น ๆ มากกว่าค่า Fan-Out จะทำให้เกิด Fault ชนิดที่เรียกว่า Struct-at One ได้ในบางชั่วขณะ และหากเกิด Fault ในช่วงเวลาที่มีสัญญาณ Clk พอดีจะเป็นเหตุให้ Flip-Flop ในลูกโซ่มีระดับสัญญาณลอจิกเป็นหนึ่งมากกว่าหนึ่งตัว เป็นสองตัว,สามตัวจนถึงครบทั้งหกตัว และเป็นเหตุให้ฐานเวลาในวงจรส่วนอื่นๆ ผิดพลาดทั้งหมด เหตุที่การวิเคราะห์ทางซอฟต์แวร์ตรวจจับไม่ได้ เนื่องจากในส่วนของการสังเคราะห์วงจรได้มีการใส่เกตที่เรียกว่าบัฟเฟอร์อย่างถูกต้องแล้ว แต่ในส่วนของซอฟต์แวร์ที่วางตัวเกตที่สังเคราะห์ลงใน FPGA ที่เรียกว่า Lay-Out Software (ซอฟต์แวร์ของ Xilinx มีชื่อเรียกว่า Place, Partition and Route Software) มีความมีความจำเป็นบางประการ ที่ต้องมองข้ามเกตบัฟเฟอร์ดังกล่าว เมื่อสอบถามกลับไปที่บริษัท Xilinx ก็ไม่ได้รับคำตอบที่ชัดเจนในการแก้ไข ผู้ออกแบบจึงได้ปรับปรุงวงจรหมุนวนซึ่งเมื่อเกิด Fault แล้วจะมีผลต่อวงจรโดยรวมอย่างถาวร ให้เป็นวงจรเลื่อนซ้าย ซึ่งเมื่อเกิด Fault ขึ้นแล้วจะมีผลต่อวงจรเพียงชั่วขณะเพราะมีส่วนของวงจรที่ตรวจสอบว่ามีสถานะลอจิกหนึ่งมากกว่าหนึ่งตำแหน่งหรือไม่ดังแสดงได้ดังรูปที่ 4-14 อย่างไรก็ดีการปรับปรุงวงจรดังกล่าวไม่ได้กำจัดปัญหาให้หมดไป แต่จากการทดสอบทางฮาร์ดแวร์ วงจรโดยรวมก็ยังทำงานได้



รูปที่ 4-14 แสดงวงจรที่เกิดขึ้นจาก One_Meg Process ก่อนและหลังการแก้ไข

2. Line_Rate Process หลังจากที่ได้หารความถี่ให้เหลือ 1 MHz แล้ว ได้มีการหารความถี่ต่อไปอีก 64 เพื่อให้ได้ความถี่เดียวกันกับความถี่สัญญาณแนวนอนของสัญญาณภาพ การหารความถี่ในที่นี้ได้ใช้วงจรนับ 64 ซึ่งก็คือ 6-Bit Counter นั่นเอง และค่าของการนับนี้จะใช้เป็นค่าของ Column ในการแสดงผลด้วย รูปที่ 4-15 แสดงโปรแกรม Line_Rate Process คำสั่ง "Line_i <= Line_i + "1"; " แสดงการใช้วงจรนับดังกล่าว

```
Line_Rate: PROCESS(clk)
    BEGIN
        IF (rising_edge(clk)) THEN
            IF (OneMeg_i(5) = '1') THEN
                Line_i <= Line_i + "1";
            END IF;
        END IF;
    END PROCESS Line_Rate;
```

รูปที่ 4-15 แสดงส่วนของโปรแกรม Line_Rate

<u>3. Line_Sync Process</u> มีหน้าที่ในการสร้างสัญญาณที่จะส่งกลับไปควบคุมวงจร Phase-Locked Loop ใน VIP เพื่อให้การทำงานของ Teletext IC เข้าจังหวะกับสัญญาณภาพในแกนนอน สัญญาณดังกล่าวคือสัญญาณ pl เมื่อ Teletext IC ยังทำงานไม่เข้าจังหวะกับ VIP เช่น pl มีเฟสนำหน้าสัญญาณภาพอยู่ วงจรก็จะสั่งให้ VIP สั่นความถี่ clk ให้ช้าลง เป็นผลทำให้ pl มีคาบมากขึ้นและทำให้ผลต่างของเฟสที่ pl นำหน้าอยู่ลดลง จนกระทั่ง pl มีเฟสเดียวกันกับสัญญาณภาพ ขณะเดียวกัน VIP ก็จะสร้างความถี่ clk ด้วยความถี่ 6 MHz พอดีด้วย หาก pl มีเฟสล้าหลังก็อธิบายได้ในทำนองกลับกัน

สัญญาณอีกสัญญาณหนึ่งก็คือ สัญญาณ cbb จะถูกส่งกลับไปบอกช่วงเวลาที่มีสัญญาณ Color Burst ที่มีอยู่ในสัญญาณภาพทุก ๆ เส้น (รวมทั้งเส้นที่มีสัญญาณเทเลเท็กซต์) ให้แก่ VIP ทั้งนี้เพื่อไม่ให้ VIP เข้าใจผิดคิดว่าสัญญาณนั้นเป็นสัญญาณเทเลเท็กซต์

สัญญาณ pl จะมีระดับลอจิกเป็นหนึ่งครึ่งคาบและเป็นศูนย์ครึ่งคาบ ส่วนที่เป็นหนึ่งเริ่มตั้งแต่ช่วงเวลา 33.5 ถึง 1.5 μsec สัญญาณ cbb จะมีสถานะเป็นหนึ่งจากเวลา 1.5 -7.5 μsec นับจากต้นเส้นสัญญาณภาพ

สัญญาณทั้งสองได้มาจากการถอดรหัสจากสัญญาณ Line ในช่วงเวลาที่สัญญาณมีการเปลี่ยนค่า และใช้คุณสมบัติของ Reset-Set Flip-Flop ดังนี้

```
eq_17 := compare(Line,To_StdUlogicVector(17,Line'Length);
eq_49 := compare(Line,To_StdUlogicVector(49,Line'Length);
eq_56 := compare(Line,To_StdUlogicVector(56,Line'Length);
pl_i <= eq_17 OR (pl_i AND not eq_49);
cbb_i <= eq_49 OR (cbb_i AND not eq_56);
```

ฟังก์ชัน To_StdUlogicVector เป็นฟังก์ชันที่เปลี่ยนชนิดของสัญญาณบางชนิดให้เป็นชนิด Std_Ulogic_Vector ซึ่งเป็นชนิดสัญญาณมาตรฐาน ในที่นี้เปลี่ยนไปจากสัญญาณชนิด Integer ส่วนฟังก์ชั่น compare เป็นฟังก์ชั่นที่ผู้ออกแบบเขียนขึ้นเพื่อใช้ในการเปรียบเทียบค่าสัญญาณสองสัญญาณว่าเท่ากันหรือไม่ ค่าตัวแปร eq_17,eq_49 และ eq_56 จะเป็นการตรวจสอบว่าสัญญาณ pl และ cbb จะมีการเปลี่ยนแปลงค่าหรือยัง แล้วนำค่าตัวแปรดังกล่าวมาทำการ set หรือ reset Flip-Flop (แบบ Synchronous)

```
Line_Sync: PROCESS(clk)
        VARIABLE eq_17,eq_49,eq_56        : std_ulogic;
        BEGIN
                IF (rising_edge(clk)) THEN
                        IF (OneMeg_i(5)='1') THEN
                                eq_17:= compare(Line_i,To_StdUlogicVector(17,Line_i'LENGTH);
                                eq_49:= compare(Line_i,To_StdUlogicVector(49,Line_i'LENGTH);
                                eq_56:= compare(Line_i,To_StdUlogicVector(56,Line_i'LENGTH);
                                pl_i <= eq_17 OR (pl_i AND not eq_49);
                                cbb_i <=  eq_49 OR (cbb_i AND not eq_56);
                        END IF;
                END IF;
        END PROCESS Line_Sync;
```

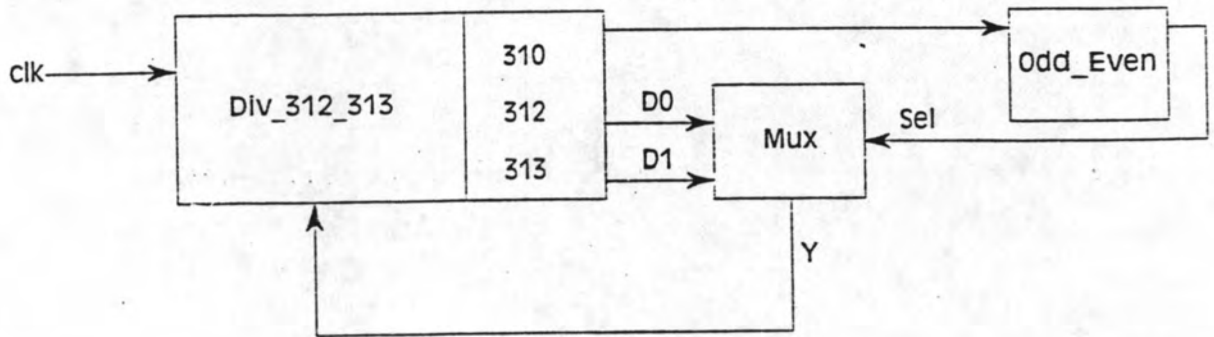
รูปที่ 4-16 แสดงส่วนโปรแกรมของ Line_Sync

4. Field_Rate Block หลังจากที่ได้มีการหารความถี่ให้เป็นความถี่ของสัญญาณภาพแนวนอนแล้วขั้นต่อมาคือการหารความถี่ไปอีก 625 เพื่อให้ได้ความถี่สัญญาณภาพในแนวแกนตั้ง ดังที่กล่าวมาแล้วในบทที่ 3 ว่า สัญญาณภาพหนึ่งเฟรมจะประกอบด้วยสองฟิลด์ คือฟิลด์คู่และฟิวด์คี่ ดังนั้นการหารความถี่ในแนวแกนตั้งต้องแบ่งการหารเป็นสองชุดคือ การหาร 313 ในฟิลด์คู่และการหาร 312 ในฟิวด์คี่ สลับกันไปเรื่อย ๆ ใน Field_Rate Block นี้จะประกอบด้วยโปรเซสสองอันดังแสดงในบล็อกไดอะแกรมรูปที่ 4-18

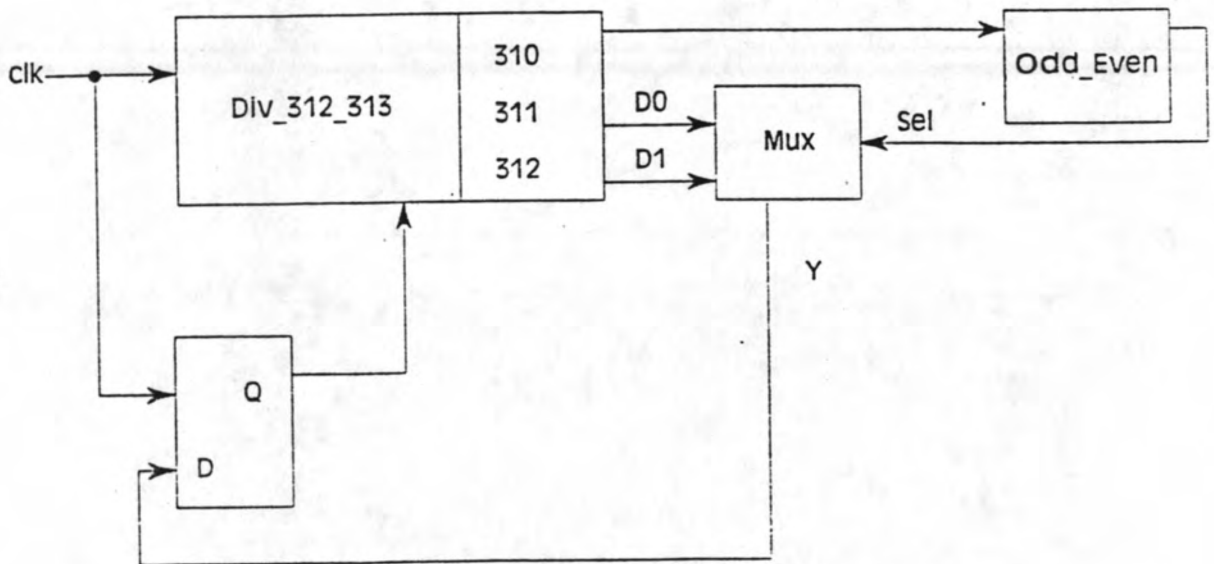
4.1 Div_312_313 Process มีหน้าที่ในการหารความถี่ 312 หรือ 313 โดยใช้ 9-bit Counter with Asynchronous Reset โดยค่าที่จะมารีเซตเกิดจากการถอดรหัสค่า 312 หรือ 313 โดยใช้ค่าเอาท์พุทของโปรเซสต่อไปเป็นตัวเลือก

4.2 Odd_Even Process ใช้สร้างสัญญาณ Even เพื่อบอกโปรเซสก่อนหน้านี้ว่าขณะนี้เป็นฟิลด์คู่หรือฟิลด์คี่ การออกแบบใช้คุณสมบัติของ T-FF

อย่างไรก็ตามทั้งสองโปรเซสจะมีสัญญาณ Asynchronous Reset อีกอันหนึ่งร่วมกันอยู่คือ Rst_FieldSync ซึ่งจะได้มาจากบล็อกในหัวข้อถัดไปเพื่อให้ Teletext IC ทำงานเข้ากับสัญญาณภาพในแนวตั้ง



รูปที่ 4-18 แสดงบล็อกไดอะแกรมของ Field_Rate Block ก่อนการแก้ไข



รูปที่ 4-19 แสดงบล็อกไดอะแกรมของ Field_Rate Block หลังการแก้ไข

ปัญหา พบขณะที่วิเคราะห์วงจรหลังจากสังเคราะห์วงจร Time_Base แล้ว เนื่องจากโปรเซส Div_312_313 ใช้การรีเซตแบบ Asynchronous และเนื่องจาก Counter มีความกว้างของบิตมากถึง 9 บิต ทำให้ต้องใช้เกตถึงสองชั้นในการถอดรหัสค่า 312 และ 313 (ความกว้างของเกตใน Xilinx FPGA เท่ากับห้า) เป็นเหตุให้เกิดความผิดพลาดจากการถอดรหัสได้ในขณะที่ Flip-Flop มีการเปลี่ยนแปลงค่า และทำให้เกิดสัญญาณรีเซตที่ไม่พึงประสงค์ได้ การแก้ไขอาจทำได้

โดยเปลี่ยนมาใช้วงจรนับที่ใช้ Synchronous Reset แทน แต่เนื่องจากวงจรที่ใช้ Synchronous Reset จะมีขนาดใหญ่กว่าแบบ Asynchronous มาก ผู้ออกแบบจึงได้แก้ไขโดยการเพิ่ม Flip-Flop ที่คอยเก็บค่าสัญญาณ Asynchronous Reset ไว้ก่อนแล้วจึงส่งค่า Reset ดังกล่าวออกไปใน clk ถัดไป จะกำจัดสัญญาณรีเซตที่ไม่พึงประสงค์โดยยังใช้วงจร Asynchronous อยู่ได้ รูปที่ 4-19 แสดงบล็อกไดอแกรมหลังการแก้ไข รูปที่ 4-20 แสดงโปรแกรมที่แก้ไขแล้ว

```
Field_Rate: BLOCK
        SIGNAL R,UP: std_ulogic;
        SIGNAL odd,even : std_ulogic;
        SIGNAL rst_count : std_ulogic;
        SIGNAL ce_even : std_ulogic;
BEGIN
        UP <= OneMeg_i(2) AND compare(line_i,To_StdUlogicVector(48,line_i'LENGTH));
        R <= rst_count OR rst_FieldSync;
        Div_312_313: PROCESS(clk,R)
        BEGIN
                IF (R = '1') THEN
                        -- reset event
                        field_i <= (OTHERS => '0');
                        rst_count <= '0';
                ELSIF (rising_edge(clk)) THEN
                        IF (UP='1')        THEN
                                -- clocked count up event
                                field_i <= field_i+"1";
                                -- 1-bit Synchronous Reset Signal to Reset 9-bit Asynchronously
                                IF (even = '1') THEN
                                        rst_count <=        compare(field_i,To_StdUlogicVector(311,
                                                            field_i'LENGTH));
                                ELSE
                                        rst_count <=        compare(field_i,To_StdUlogicVector(312,
                                                            field_i'LENGTH));
                                END IF;
                        END IF;
                END IF;
```

```
        END PROCESS Div_312_313;

FieldSync_want <= compare(field_i, To_StdUlogicVector(310,field_i'LENGTH));
ce_even <= UP AND FieldSync_want;

odd_even: PROCESS(clk, rst_FieldSync)
        BEGIN
                IF (rst_FieldSync = '1') THEN
                        even <= '0';
                ELSIF (rising_edge(clk)) THEN
                        IF (ce_even='1') THEN
                                even <= not even;
                        END IF;
                END IF;
        END PROCESS odd_even;
END BLOCK Field_Rate;
```

รูปที่ 4-20 แสดงส่วนของโปรแกรม Field_Rate หลังการแก้ไข

5. Field_Sync Block วงจรมีหน้าที่สร้างสัญญาณรีเซต Rst_FieldSync ในช่วงสัญญาณภาพเส้นที่ศูนย์ให้แก่ Field_Rate Block เพื่อให้สัญญาณ Field และ Even มีค่าเป็นศูนย์และทำให้ Teletext IC ทำงานเข้าจังหวะกันกับสัญญาณภาพในแนวตั้ง การสร้างสัญญาณรีเซตทำด้วยการตรวจสอบสัญญาณ vcs ดังกล่าวในช่วงต้น ในการที่จะตรวจสอบว่าระดับสัญญาณลอจิกหนึ่งนานกว่าลอจิกศูนย์หรือไม่ ทำได้โดยใช้วงจรดิจิตัลอินทิเกรเตอร์ซึ่งก็คือ Up/Down Counter ที่จะนับเฉพาะค่าที่เป็นบวก มาคอยตรวจสอบสถานะของ vcs ทุกๆ 1 μsec ซึ่งจะนับขึ้นเมื่อ vcs มีสถานะเป็นหนึ่งและจะนับลงเมื่อ vcs มีสถานะเป็นศูนย์ แต่จะไม่นับลงไปกว่าค่าศูนย์ เมื่อ Counter นับถึงค่า 37 (จะเกิดหลังจากเกิด Field Sync ประมาณ 3/4 คาบ) จะเกิดพัลส์ที่สัญญาณ FieldSync_Found ทั้งนี้วงจรอินทิเกรเตอร์จะช่วยกรองสัญญาณรบกวนใน vcs ได้อีกด้วย

คำสั่ง

```
IF vcs='1' THEN     FieldSync_Count <= FieldSync_Count + "1";
ELSE                FieldSync_Count <= FieldSync_Count - "1";
```

แสดงการใช้ Up/Down Counter ส่วนการป้องกันไม่ให้วงจรนับนับค่าที่ต่ำกว่าค่าศูนย์ทำได้ด้วยการตรวจสอบ sign bit ซึ่งเป็นบิตสุดท้าย (MSB) ของสัญญาณ FieldSync_Count โดยนำ sign bit ไปรีเซต FieldSync_Count ก็คือตัวมันเองให้มีค่ากลับเป็นศูนย์ใหม่

สัญญาณ FieldSync_Found จะถูกเปลี่ยนไปเป็นสัญญาณ Rst_FieldSync ก็ต่อเมื่อค่าของ State มีค่าเป็น Wait_State ซึ่งค่า State จะเปลี่ยนเป็น Wait_State ก็ต่อเมื่อค่าของ FieldSync_Want เป็นหนึ่ง (Field > 310) การที่มีการส่งสัญญาณกลับมาล็อกกันนี้เป็นการตัดสัญญาณรบกวนช่วงสั้นได้อย่างดีมาก มิฉะนั้นหากเกิดสัญญาณ FieldSync_Reset ที่ผิดพลาดคือในช่วง DontCare_State จะทำให้วงจร Acquisition เข้าใจเส้นสัญญาณผิดคิดว่าเส้นสัญญาณภาพโทรทัศน์เป็นเส้นสัญญาณเทเลเท็กซ์ทำให้ได้รับข้อมูลที่ผิดพลาดได้

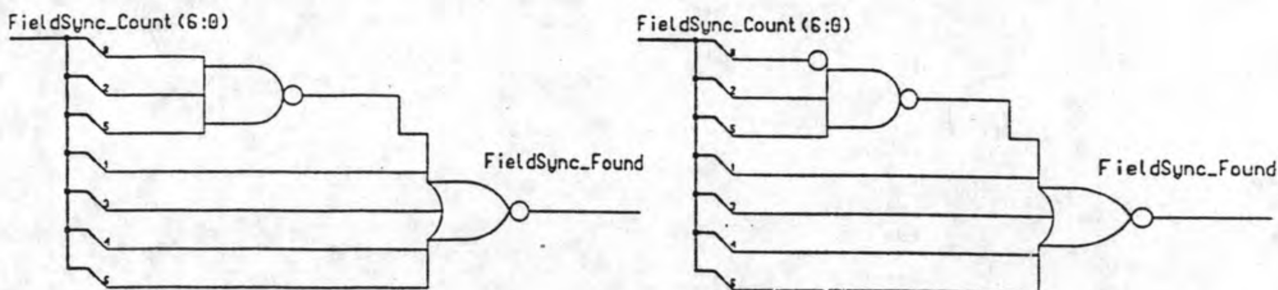
```
field_sync: BLOCK
        Type    State is (Wait_State,DontCare_State);
        SIGNAL present_state, next_state : state:= Wait_State;
        SIGNAL R,FieldSync_found: std_ulogic;
        SIGNAL FieldSync_Count : std_ulogic_vector(6 downto 0);
BEGIN
        -- signbit resets itself.
        R <= FieldSync_Count(6);
        Intregrate_Process: PROCESS(clk,R)
        BEGIN
                IF (R = '1') THEN
                        -- reset event
                        FieldSync_Count <= (OTHERS => '0');
                ELSIF (rising_edge(clk)) THEN
                        IF (OneMeg_i(5)='1') THEN
                                -- Up/Down Count Event
                                IF (vcs='1') THEN
                                        FieldSync_Count <=FieldSync_Count + "1";
                                ELSE
                                        FieldSync_Count <=FieldSync_Count - "1";
                                END IF;
                        END IF;
                END IF;
```

```
        END PROCESS Count_Process;
        FieldSync_found <= compare(FieldSync_Count,
                                To_StdUlogicVector(37,FieldSync_Count'LENGTH));
        State_Clock: PROCESS(clk)
        BEGIN
                IF (rising_edge(clk)) THEN
                        present_state <= next_state;
                END IF;
        END PROCESS State_Clock;
        State_Transition: Process(present_state,FieldSync_want,FieldSync_found)
        BEGIN
                IF (present_state = Wait_State) THEN
                        IF ( FieldSync_found = '0') THEN
                                next_state <= Wait_State;
                        ELSE
                                next_state <= DontCare_State;
                        END IF;
                ELSE -- present_state = DontCare_State
                        IF ( FieldSync_want = '0') THEN
                                next_state <= DontCare_State;
                        ELSE
                                next_state <= Wait_State;
                        END IF;
                END IF;
                        END PROCESS State_Transition;
Output:
rst_FieldSync <=  '1' WHEN (present_state = Wait_State AND FieldSync_found = '1' AND half_line = '1')
ELSE              '0';
END BLOCK field_sync;
```

รูปที่ 4-21 แสดงส่วนโปรแกรม Field_Sync

ปัญหา เกิดจากการสังเคราะห์วงจรในการถอดรหัสค่า 37 ดังรูปที่ 4-22a ในการวิเคราะห์วงจรจะพบว่าหาก FieldSync_Count มีค่าเปลี่ยนจาก -1 ไปเป็น 0 (1111111b ไปเป็น

0000000b) ซึ่งเป็นเหตุการณ์ที่เกิดขึ้นจริงทุก ๆ เส้นสัญญาณภาพ จะทำให้เกิดสัญญาณ FieldSync_Found ขึ้นโดยไม่ตั้งใจด้วย การแก้ไขทำได้ง่ายคือยอมแก้ไขแผนภาพ Schematic ให้เป็นดังรูปที่ 4-22b ซึ่งจะทำให้ค่าที่ FieldSync_Count จะกำเนิดสัญญาณ FieldSync_Found มีค่าลดลงจาก 37 เป็น 36 ซึ่งเป็นค่าที่ยังใช้ได้อยู่ และค่าที่จะทำให้เกิดความผิดพลาดคือจาก -2 ไปเป็น 0 ซึ่งในความเป็นจริงจะไม่เกิดเหตุการณ์นี้ขึ้น



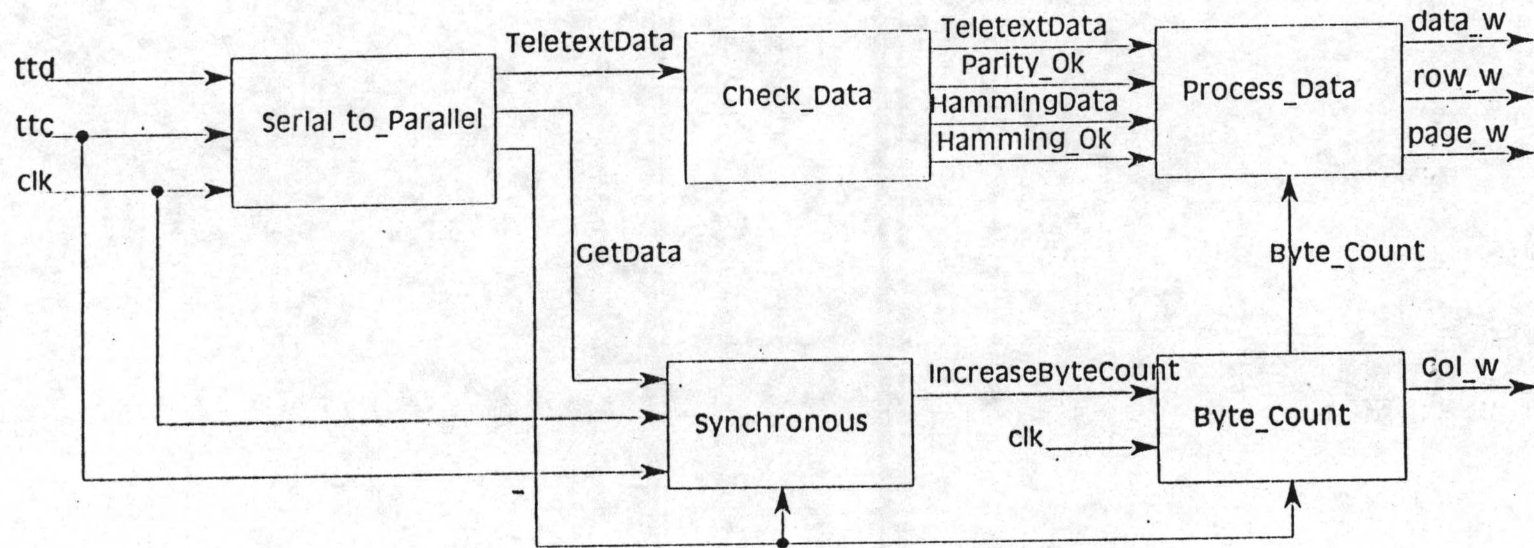
รูปที่ 4-22 แสดงผลที่ได้จากการสังเคราะห์วงจรถอดรหัสค่า 37 ก่อนและหลังการแก้ไข

Acquisition

วงจรส่วน Acquisition เกิดจาก 2 Process รวมเข้าไว้ด้วยกันคือ Slice_Data_in กับ Process_Data ซึ่งสามารถแยกเป็นการอธิบายแบบเชิงกระแสข้อมูล ได้ดังบล็อกไดอะแกรมในรูปที่ 4-23

1. Serial_to_Parallel Block วงจรส่วนนี้ทำหน้าที่รับและแปลงสัญญาณเทเลเท็กซต์ที่เป็นสัญญาณอนุกรมให้เป็นสัญญาณขนานแปดบิต ซึ่งการเข้าจังหวะกับข้อมูลเทเลเท็กซต์แต่ละเฟรมทำได้โดยการส่งรหัสเฟรมมิง (Framing Code) ซึ่งมีค่าเท่ากับ "11100100b" ดังที่กล่าวมาแล้วในบทที่ 3 เมื่อวงจรตรวจจับรหัสเฟรมมิงได้แล้ว ก็จะทำการเก็บข้อมูลที่รับเข้ามาทุก ๆ ช่วงเวลาแปดบิต

1.1 Decoder Process มีหน้าที่ในการสร้างสัญญาณ DEW (Data Entry Window) และสัญญาณ FCW (Framing Code Window) สัญญาณ DEW จะทำให้วงจรรับสัญญาณจากสัญญาณ ttc และ ttd เข้ามาในเส้นสัญญาณที่มีสัญญาณเทเลเท็กซต์เท่านั้น (เส้นที่ 6 ถึง 22) ส่วนสัญญาณ FCW จะทำให้วงจรสนใจที่จะรับรหัสเฟรมมิงในช่วงระยะเวลาที่ควรจะเกิด



รูปที่ 4-23 แสดงบล็อกไดอะแกรมของ Acquisition

คือเริ่มตั้งแต่ 12-14 μsec นับจากเริ่มต้นเส้นสัญญาณภาพเท่านั้น ทั้งนี้การกำเนิดสัญญาณทั้งสองอาศัยหลักการของ RS-Flip-Flop ดังที่ได้กล่าวมาแล้ว

1.2 Bit_Count Process มีหน้าที่ในหารความถี่ ttc ไปแปดครั้งเพื่อให้ได้สัญญาณ GetData ในการอ่านข้อมูลเข้ามาเก็บเมื่อรับสัญญาณมาครบหนึ่งไบต์ ทำได้โดยการใช้วงจรนับแปดซึ่งมีสัญญาณรีเซตคือ FramingCode

1.3 Shift_Data Process มีหน้าที่ในรับข้อมูลอนุกรมและเก็บข้อมูลแบบขนานโดยอาศัยวงจรเลื่อน ดังคำสั่ง parallel := ttd & parallel(7 downto 1);

1.4 Store_Data Process มีหน้าที่ในเก็บข้อมูลจาก parallel ไปเก็บไว้ใน Teletext_Async เมื่อมีสัญญาณ GetData

```
Serial_to_Parallel: BLOCK
        SIGNAL FramingCode,fcw,dew,GetData          : std_ulogic:='0';
        SIGNAL parallel,Teletext_Async      : std_ulogic_vector(7 downto 0):=(OTHERS => '0');
        SIGNAL BitCount  : std_ulogic_vector(2 downto 0):=(OTHERS => '0');
        BEGIN
        Decoder: Process
                VARIABLE line_0,line_60,field_6,field_22    : std_ulogic:='0';
        BEGIN
                IF (rising_edge(clk)) THEN
                        IF (one_meg(2) = '1') THEN
                                line_62   := compare(line,To_StdUlogicVector(62,line'LENGTH));
                                line_0    :=compare(line,To_StdUlogicVector(0,line'LENGTH));
                                field_6   :=compare(field,To_StdUlogicVector(6,field'LENGTH));
                                field_22  :=compare(field,To_StdUlogicVector(22,field'LENGTH));
                                fcw <= line_62 OR (not line_0 AND fcw);
                                dew <= field_6 OR (not field_22 AND dew);
                        END IF;
                END IF;
        END PROCESS Decoder;
        Bit_Count: PROCESS(ttc,FramingCode)
        BEGIN
                IF (FramingCode='1') THEN
                        -- reset if Framming Code is Found.
```

หอสมุดกลาง สถาบันวิทยบริการ จุฬาลงกรณ์มหาวิทยาลัย

```
                        BitCount <= (OTHERS => '0');
                ELSIF (rising_edge(ttc)) THEN
                        -- Count 8 Bit
                        BitCount <= BitCount+"1";
                END IF;
        END PROCESS Bit_Count;
        GetData <= compare(BitCount,To_StdUlogicVector(7, BitCount'Length));
        Shift_Data: PROCESS(ttc)
        BEGIN
                IF (rising_edge(ttc)) THEN
                        IF (dew = '1') THEN
                                parallel <= ttd&parallel(7 downto 1);
                        END IF;
                END IF;
        END PROCESS Shift_Data;
        FramingCode <= compare(parallel,std_ulogic_vector'("00100111")) AND fcw;
        Store_Data_1: PROCESS(ttc)
        BEGIN
                IF (rising_edge(ttc)) THEN
                        IF (GetData='1') THEN
                                Teletext_Async <= Parallel;
                        END IF;
                END IF;
        END PROCESS Store_Data_1;
        Store_Data_2: PROCESS(clk)
        BEGIN
                IF (rising_edge(clk)) THEN
                        IF (OneMeg(5)='1') THEN
                                TeletextData <= Teletext_Async;
                        END IF;
                END IF;
        END PROCESS Store_Data_2;
END BLOCK Serial_to_Parallel;
```

รูปที่ 4-24 แสดงส่วนของโปรแกรม Serial_to_Parallel

2. Synchronous Block เนื่องจากสัญญาณเทเลเท็กซ์มีความถี่ 6.9375 Mbits/sec หรือใช้เวลาในการส่งตัวอักษรขนาดแปดบิตด้วยเวลา 1.15 μsec แต่ความถี่ของระบบเท่ากับ 1 MHz หากสุ่มอ่านข้อมูลทุก ๆ 1 μsec จะทำให้ได้ข้อมูลซ้ำเดิมทุก ๆ 5-6 ครั้ง ดังนั้นเพื่อไม่ให้ค่าของตัวนับตำแหน่งของข้อมูลมีค่าเพิ่มขึ้นในกรณีที่อ่านข้อมูลตัวเดิม จึงจำเป็นต้องมีการตรวจสอบว่ามีสัญญาณ GetData อันใหม่หลังจากได้อ่านข้อมูลไปแล้วหรือไม่ เมื่อเกิดสัญญาณ OneMeg(5) ติดกันโดยไม่มีสัญญาณ GetData มาเปลี่ยนค่า NewData จะทำให้ค่าของ NewData กับ Data_Read มีค่าเดียวกัน ซึ่งจะทำให้สัญญาณ IncreseByteCounter มีค่าเป็นศูนย์ เพื่อให้ Byte_Count ซึ่งเป็นวงจรในหัวข้อถัดไปไม่ทำงาน ทำให้การเขียนข้อมูลที่ได้รับถัดไปจะเขียนในตำแหน่งเดิมที่เป็นข้อมูลซ้ำ

```
Synchronize: BLOCK
        SIGNAL NewData,Data_Read : std_ulogic;
BEGIN
                Clock1:PROCESS(ttc,FramingCode)
                BEGIN
                        IF (FramingCode='1') THEN
                                NewData <= '0';
                        ELSIF (rising_edge(ttc)) THEN
                                IF (GetData='1') THEN
                                        NewData <= not NewData;
                                END IF;
                        END IF;
                END PROCESS clock1;
                Clock2: PROCESS(clk,FramingCode)
                BEGIN
                        IF (FramingCode='1') THEN
                                Data_Read <= '0';
                                IncreaseByteCount <='0';
                        ELSIF (rising_edge(clk)) THEN
                                IF (one_meg(5)='1') THEN
                                        Data_Read <= NewData;
                                        IF (Data_Read = NewData) THEN
                                                IncreaseByteCount <='0';
```

```
                        ELSE
                                IncreaseByteCount <= '1';
                        END IF;
                End IF;
            END IF;
    END PROCESS Clock2;
END BLOCK Synchronize;
```

รูปที่ 2-25 แสดงส่วนโปรแกรม Synchronize

3. Byte_Count Block ในหนึ่งเส้นข้อมูลของสัญญาณเทเลเท็กซต์จะประกอบด้วยข้อมูลสองส่วนคือ ข้อมูลที่ใช้บอกตำแหน่งของข้อมูลบนจอภาพมีขนาด 2-4 ไบต์ กับข้อมูลที่เป็นข้อความบนจอภาพมีขนาด 40 ไบต์ (ดูรายละเอียดในบทที่ 3) จึงมีวงจรนับอยู่สองส่วนคือวงจรนับสี่ กับวงจรนับสี่สิบ ในส่วนของวงจรนับสี่ได้เลือกใช้การวงจรเลื่อนซ้ายเพราะเป็นวงจรนับค่าน้อยๆ จะประหยัดส่วนวงจรถอดรหัส ส่วนวงจรนับสี่สิบ ใช้ 6-bit Counter ตามปกติ

เมื่อได้รับสัญญาณ FramingCode ฟลิบฟลอบทุกตัวของวงจรเลื่อนซ้ายจะต้องถูกรีเซต ยกเว้นฟลิบฟลอบตัวแรกจะถูกเซตให้เป็นหนึ่ง แล้วส่งค่าหนึ่งต่อให้กับฟลิบฟลอบตัวถัดไปในช่วงสัญญาณนาฬิกาถัดไป และเมื่อฟลิบฟลอบตัวที่สองมีค่าเป็นหนึ่งจะส่งค่ารีเซต ให้วงจรนับสี่สิบเริ่มนับตำแหน่งของข้อมูลบนจอภาพด้วย

```
Byte_Count: BLOCK
BEGIN
    Slice_Byte_Process : PROCESS(clk,FramingCode)
    BEGIN
        IF (FramingCode = '1') THEN
            -- Set 1st flip-flop, others flip-flop reset
            SliceCount <= (0 => '1',OTHERS => '0');
        ELSIF (rising_edge(clk)) THEN
            IF (IncreaseByteCount = '1' AND one_meg(5) = '1') THEN
                SliceCount <= SliceCount(SliceCount'HIGH-1 downto 0)&'0';
            END IF;
        END IF;
    END PROCESS Slice_Byte_Process;
```

```
        Count_Byte_Process : PROCESS(clk,SliceCount(1))
        BEGIN
                IF (SliceCount(1) = '1') THEN
                        ByteCount <= (OTHERS => '0');
                ELSIF (rising_edge(clk)) THEN
                        IF (IncreaseByteCount = '1' AND one_meg(5) = '1') THEN
                                ByteCount <= ByteCount + "1";
                        END IF;
                END IF;
        END PROCESS Count_Byte_Process;
END BLOCK Byte_Count;
```

รูปที่ 4-26 แสดงส่วนโปรแกรม Byte_Count

4. Check_Data Process มีหน้าที่ในการตรวจสอบความถูกต้องของข้อมูล ซึ่งมีอยู่สองประเภทคือ ตรวจสอบพาริตี้ กับตรวจสอบรหัสแฮมมิ่ง (ดูรายละเอียดในบทที่ 3) ฟังก์ชัน "parity" เป็นฟังก์ชันที่เขียนขึ้นเพื่อตรวจสอบว่าข้อมูลเป็นพาริตี้คู่หรือไม่ ซึ่งสามารถนำมาเขียนโปรแกรมได้ดังรูปที่ 4-27

```
Check_data: PROCESS (TeletextData)
        VARIABLE A,B,C,D : std_ulogic;
        VARIABLE E : std_ulogic_vector(2 downto 0);
        BEGIN
                A := parity(TeletextData(0)&TeletextData(1)&TeletextData(5)&TeletextData(7));
                B := parity(TeletextData(1)&TeletextData(2)&TeletextData(3)&TeletextData(7));
                C := parity(TeletextData(1)&TeletextData(3)&TeletextData(4)&TeletextData(5));
                D := parity(TeletextData);
                E := A&B&C;
                IF ( D = '0') THEN
                        ParCheck <= '1';
                        IF ( E ="000" ) THEN
                                HamCheck <= '1';
                        ELSE
                                HamCheck <= '0';
```

```
            END IF;
        ELSE
            HamCheck <= '1';
            ParCheck <= '0';
        END IF;
        IF E = "110" THEN
            HamData(3) <=  not SyncData(7);
        ELSE
            HamData(3) <= SyncData(7);
        END IF;
        IF E = "101" THEN
            HamData(2) <=  not SyncData(5);
        ELSE
            HamData(2) <= SyncData(5);
        END IF;
        IF E = "011" THEN
            HamData(1) <=  not SyncData(3);
        ELSE
            HamData(1) <= SyncData(3);
        END IF;
        IF E = "111" THEN
            HamData(0) <=  not SyncData(1);
        ELSE
            HamData(0) <= SyncData(1);
        END IF;
        END PROCESS Check_Data;
```

รูปที่ 4-27 แสดงส่วนโปรแกรม Check_Data

5. Process_Data Process เป็นส่วนที่ผู้ออกแบบไม่ต้องออกแบบใหม่มากนัก เพราะสามารถนำโปรเซสนี้จากการการอธิบายแบบเชิงพฤติกรรมมาปรับปรุงเพียงเล็กน้อย ก็สามารถใช้งานได้ ส่วนโปรแกรม Process_Data แสดงได้ดังรูปที่ 4-28

```
Process_Data : PROCESS
        CONSTANT ENGLISH     : std_ulogic := '0';
        CONSTANT THAI        : std_ulogic := '1';
        CONSTANT THAIPAGE    : std_ulogic_vector(7 downto 0):= "10001010";
        CONSTANT TWISTCODE : std_ulogic_vector(7 downto 0):= "10011011";
        Variable RowZero,RowOne : std_ulogic := '0';
BEGIN

        IF (rising_edge(clk)) THEN
                RowZero := compare(row_i,To_StdUlogicVector(0,row_i'LENGTH));
                RowOne := compare(row_i,To_StdUlogicVector(1,row_i'LENGTH));
                IF (OneMeg(5) = '1') THEN
                        IF (SyncData=TWISTCODE) THEN
                                LangBit <= not LangBit;
                        END IF;
                        IF (SliceCount(0) = '1') THEN
                                LineAccept(0)      <= HamCheck;
                                Mag                <= HamData(2 downto 0);
                                Row_i(0)           <= HamData(3);
                                LangBit  <= PageLang;
                        ELSIF (SliceCount(1) = '1') THEN
                                LineAccept(1)      <= HamCheck AND LineAccept(0);
                                Row_i(4 downto 1) <= HamData;
                        ELSIF (SliceCount(2) = '1') THEN
                                IF (RowZero = '1') THEN
                                        PageLang           <= ENGLISH;
                                        LangBit            <= ENGLISH;
                                        PageAccept(0) <= HamCheck AND LineAccept(1);
                                        PageUnit <= HamData;
                                ELSIF (RowOne = '1' AND SyncData = THAIPAGE)THEN
                                        PageLang                   <= THAI;
                                        LangBit                    <= THAI;
                                END IF;
                        ELSIF (SliceCount(3) = '1') THEN
```

```
                        IF (RowZero = '1') THEN
                                PageAccept(1) <= Hamcheck AND PageAccept(0);
                                PageTen <= HamData;
                        END IF;
                END IF;
        END IF;
    END IF;
END PROCESS Process_Data;
```

รูปที่ 4-28 แสดงส่วนโปรแกรม Process_Data

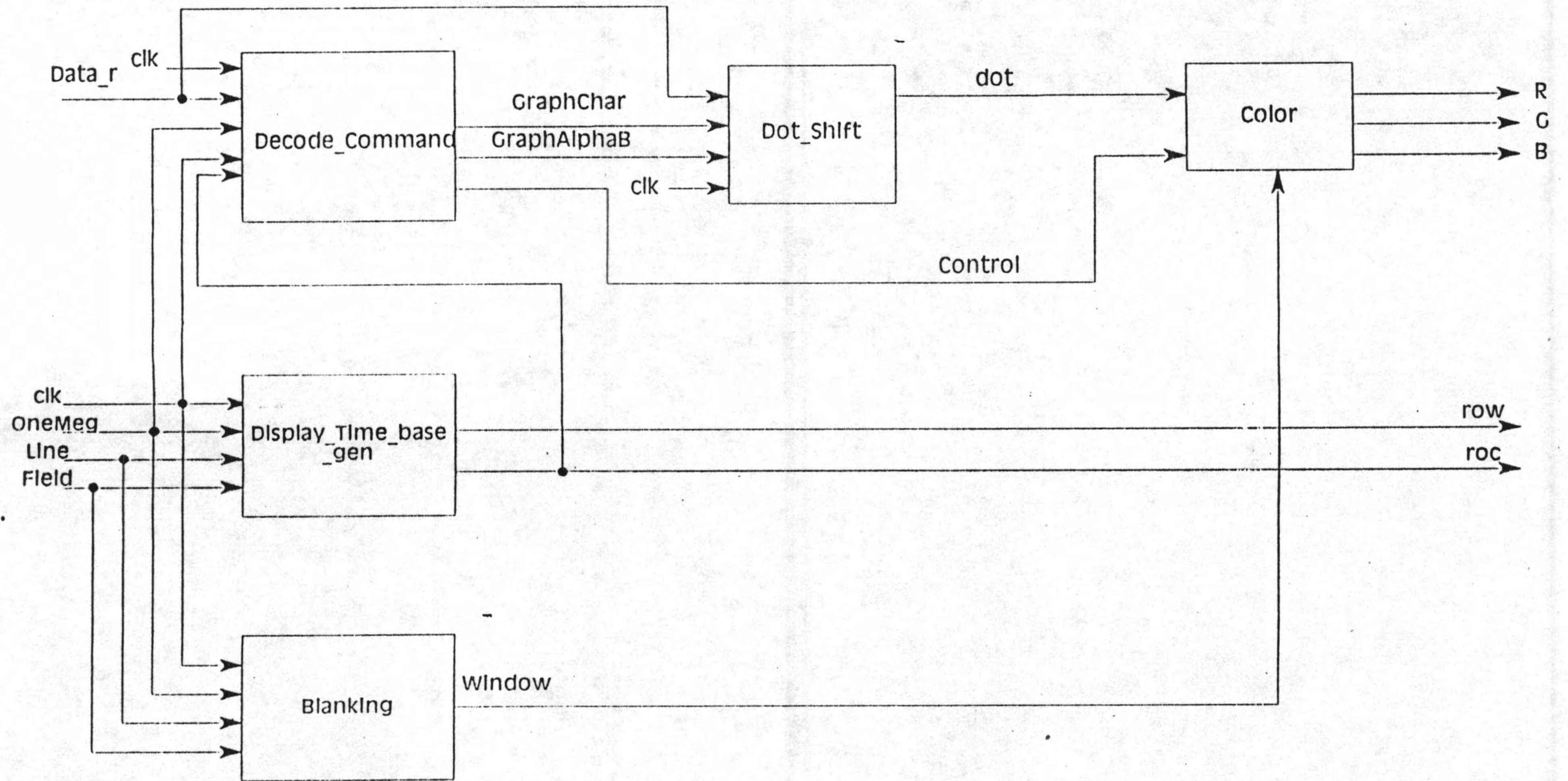
Text_Gen

การอธิบายวงจรส่วนเกิดจาก 3 Process คือ Display_Time_Base_Gen, Decode_Commad และ RGB_Dot_Out แล้วมาออกแบบใหม่ดังบล็อกไดอะแกรมในรูปที่ 4-29

1. Display_Time_Base_Gen Block มีหน้าที่สร้างสัญญาณเวลาให้กับ การแสดงผลดังกล่าวแล้วในการอธิบายเชิงพฤติกรรมซึ่งเมื่อนำมาปรับปรุงเพื่อให้สามารถสังเคราะห์วงจรได้ จะสามารถเขียนโปรแกรมได้ดังรูปที่ 4-30 วงจรทั้งหมดสามารถสร้างได้จาก Counter ที่มี Asynchronous Reset

```
Display_Time_Base_Gen : BLOCK
        SIGNAL rst_roc,FirstLine,CountUp1,CountUp2,BlinkClk : std_ulogic;
BEGIN
        FirstLine <= compare(Field, To_StdUlogicVector(40,Field'Length));
        rst <= compare(roc_i, To_StdUlogicVector(10,roc_i'Length)) OR FirstLine
        CountUp1 <= one_meg(2) AND compare(Column,To_StdUlogicVector(48,Column'Length));
        Countup2 <= CountUp1 AND compare(roc_i, To_StdUlogicVector(9,roc_i));
        BlinkClk <= compare(row_i,To_StdUlogicVector(24,row_i'Length));

        Div_10: PROCESS (clk,rst)
        BEGIN
                IF (rst_roc = '1') THEN
```



รูปที่ 4-29 แสดงบล็อกไดอะแกรมของ Text_Gen

```
                roc_i <= (OTHERS => '0');
        ELSIF (rising_edge(clk)) THEN
                IF (Countup1 = '1') THEN
                        roc_i <= roc_i + "1";
                END IF;
        END IF;
    END PROCESS Div_10;

    Div_25: PROCESS (clk,FirstLine)
    BEGIN
        IF (FirstLine = '1') THEN
                row_i <= (OTHERS => '0');
        ELSIF (rising_edge(clk)) THEN
                IF (ce2 = '1') THEN
                        row_i <= row_i + "1";
                END IF;
        END IF;
    END PROCESS Div_25;

    Div_32: PROCESS (BlinkClk)
    BEGIN
        IF (rising_edge(BlinkClk)) THEN
                BlinkCount <= BlinkCount + "1"
        END IF;
    END PROCESS Div_32;
END BLOCK Display_Time_Base_Gen;
```

รูปที่ 4-30 แสดงส่วนโปรแกรม Display_Time_Base_Gen

2. Blanking Block มีหน้าที่สร้างหน้าต่างในการแสดงผลของจอภาพ การสร้างหน้าต่างนี้จะต้องสร้างทั้งในแนวตั้งและในแนวนอน ซึ่งในแนวนอนเริ่มแสดงผลตั้งแต่คอลัมน์ที่ศูนย์ถึงสี่สิบ (ข้อมูลเทเลเท็กซต์มีสี่สิบตัวอักษรต่อหนึ่งแถว) ส่วนในแนวตั้งเริ่มตั้งแต่เส้นที่สี่สิบถึงสองร้อยเก้าสิบ (หนึ่งแถวตัวอักษรใช้ 10 เส้น, 25 แถวใช้ 250 เส้น)

```
Blanking: BLOCK
        Signal VerBlk, HorBlk : std_ulogic;
BEGIN
        Horizontal_Blank: PROCESS(clk)
                VARIABLE column0,column39 : std_ulogic;
        BEGIN
                IF (rising_edge(clk)) THEN
                        column0 := compare(column,To_StdUlogicVector(0,Column'Length));
                        column39 := compare(column,To_StdUlogicVector(0,Colum'Length));
                        HorBlk <= Column0 OR (HorBlk AND not Column39);
                END IF;
        END PROCESS Horizontal_Blank;
        Vertical_Blank: PROCESS(clk)
                VARIABLE FirstLine,LastLine : std_ulogic;
        BEGIN
                IF (rising_edge(clk)) THEN
                        FirstLine := compare(Field,To_StdUlogicVector(40,Field'Length));
                        LastLine := compare(Field,To_StdUlogicVector(290,Field'Length));
                        VerBlk <= FirstLine OR (VerBlk AND not LastLine);
                END IF;
        END PROCESS Vertical_Blank;
        Window <= HorBlk AND VerBlk;
END BLOCK Blanking;
```

รูปที่ 4-31 แสดงส่วนโปรแกรม Blanking

3. Decode_Command Process เป็นส่วนที่ปรับปรุงจากการอธิบายเชิงพฤติกรรม ซึ่งมีหน้าที่ในการตรวจจับและจัดเก็บข้อมูลที่เป็นคำสั่งต่าง ๆ เพื่อใช้ประกอบในการแสดงผลต่อไป

ในการตรวจจับข้อมูลที่เป็นคำสั่งทำได้โดยการตรวจสอบ ค่าในบิตที่ 5 และ 6 ของข้อมูลว่าเป็นศูนย์หรือไม่ หากใช่จึงมาตรวจสอบข้อมูลอีกห้าบิตที่เหลือว่าเป็นคำสั่งใด

ยกตัวอย่างเช่น หากข้อมูลในบิตที่ 3 เป็นศูนย์ แสดงว่าข้อมูลนั้นเป็นข้อมูลที่เกี่ยวกับสีและการเลือกการแสดงผลระหว่าง ตัวอักษรปกติกับตัวอักษรกราฟฟิก ข้อมูลสีนั้นจะอยู่ที่สามบิต

หลังคือบิตที่ 0-2 ซึ่งก็คือสัญญาณสี RGB นั่นเอง ส่วนข้อมูลในบิตที่ 4 จะเลือกการแสดงผลตัวอักษรปกติหากมันมีค่าเป็นศูนย์ เป็นต้น

```
Decode_Command: PROCESS (clk,NewLine)
        CONSTANT COMMAND         : std_ulogic_vector (1 downto 0):= "00";
        CONSTANT FLASH           : std_ulogic_vector (2 downto 0):= "000";
        CONSTANT BACKGROUND      : std_ulogic_vector (2 downto 0):= "110";
        CONSTANT GRAPHICS        : std_ulogic_vector (2 downto 0):= "111";
BEGIN
        IF (NewLine = '1') THEN
                ForeColor <= (OTHERS => '1');     -- White Foreground
                BackColor <= (OTHERS => '0');     -- Black Background
                GraphAlphaB     <= '0';
                Blink           <= '0';
                HoldB           <= '1';
        ELSIF (falling_edge(clk)) THEN
                IF ( control(6 downto 5) = COMMAND  ) THEN
                        IF (control(3) = '0') THEN
                                IF (OneMeg(5)='1') THEN
                                        ForeColor       <= control(2 downto 0);
                                        GraphAlphaB     <= control(4);
                                END IF;
                        ELSIF (control(4)&control(2 downto 1) = FLASH
                                        AND OneMeg(5)='1') THEN
                                Blink   <= not control(0);
                        ELSIF (control(4)&control(2 downto 1) = BACKGROUND
                                        AND OneMeg(5)='1') THEN
                                IF (control(0)='0') THEN    -- Black Background
                                        BackColor <= (OTHERS => '0');
                                ELSE                        -- New Background
                                        BackColor <= ForeColor;
                                END IF;
                        ELSIF (control(4)&control(2 downto 1) = GRAPHICS) THEN
```

```
                    IF (control(0)='0') THEN
                        HoldB   <= '0';
                    ELSIF (OneMeg(5)='1') THEN
                        HoldB <= '1';
                    END IF;
                END IF;
            END IF;
        END IF;
    END PROCESS Command_Decode;
```

รูปที่ 4-32 แสดงส่วนโปรแกรม Command_Decode

4. Dot_Shift Block จะทำหน้าที่ในการส่งข้อมูลที่เป็นจุดความสว่างบนจอภาพออกไป ตัวอักษรหนึ่งตัวจะใช้เวลาในการแสดงผล 1 ไมโครวินาที และข้อมูลจุดต่อหนึ่งตัวอักษรมีสิบสอง ตัว หรือต้องส่งจุดเหล่านี้ออกไปด้วยความถี่ 12 Mbit/sec แต่ความถี่สูงสุดของระบบเป็น 6 Mhz ดังนั้นจึงจำเป็นต้องแบ่งข้อมูลเป็นสองชุดคือข้อมูลในชุดบิตคู่ (บิตที่ 0,2,4,...) และบิตคี่ (บิตที่ 1,3,5,...) แล้วส่งค่าข้อมูลทั้งสองออกไปในทั้งสองขอบการเปลี่ยนแปลงของสัญญาณ clk

```
Dot_Shift: BLOCK
        SIGNAL OddChar,EvenChar         : std_ulogic_vector (5 downto 0);
        SIGNAL Odd,Even                 : std_ulogic_vector (5 downto 0);
BEGIN
        -- Group Odd and Even Bit and Select GraphChar or AlphaChar to Display
        OddChar  <=     (5 | 4 | 3 => GraphChar(1), OTHERS => GraphChar(0))
                WHEN (GraphAlphaB = '1' AND control(5) = '1') OR holdb='0' ELSE
                        ('0',AlphaChar(7),AlphaChar(5),AlphaChar(3),Alphachar(1),'0');

        EvenChar  <=    (5 | 4 | 3 => GraphChar(1),OTHERS => GraphChar(0))
                WHEN (GraphAlphaB = '1' AND control(5) = '1') OR HoldB='0' ELSE
                        ('0',AlphaChar(6),AlphaChar(4),AlphaChar(2),Alphachar(0),'0');

        Odd_Shift: PROCESS (clk)
        BEGIN
```

```
            IF (rising_edge(clk)) THEN
                    IF (OneMeg(4) = '1') THEN
                            -- Synchronous Load
                            odd <= OddChar;
                    ELSE
                            -- Shift event
                            odd <= odd(odd'HIGH-1 downto 0)&'0';
                    END IF;
            END IF;
    END PROCESS Odd_Shift;
    Even_Shift: PROCESS (clk)
    BEGIN
            IF (falling_edge(clk)) THEN
                    IF (OneMeg(5) = '1') THEN
                            -- Synchronous Load
                            even <= EvenChar;
                    ELSE
                            -- Shift Event
                            even <= even(even'HIGH-1 downto 0)&'0';
                    END IF;
            END IF;
    END PROCESS Even_Shift;

    -- Select  odd dot  or even dot to get 12 Mbit / sec
    dot <=   odd(5) WHEN clk = '0' ELSE
             even(5);
END BLOCK Dot_Shift;
```

รูปที่ 4-33 แสดงส่วนโปรแกรม Dot_Shift

5. Color_out Block มีหน้าที่จัดการเกี่ยวกับสีบนจอภาพ โดยการรวมข้อมูลจุดความสว่างจากบล็อกที่แล้วกับข้อมูลของสีพื้น (Background Color) และสีตัวอักษร (Foreground Color) มาสร้างสัญญาณ R,G,B ซึ่งเป็น Output หลักของ Teletext IC

```
Color_Out: BLOCK
        SIGNAL Color : std_ulogic_vector(2 downto 0);
BEGIN
        -- select blackgroud color when dot = 0 or blink character
        Color <=BackColor WHEN dot = '0' OR (blink='1' AND BlinkPeriod='1') ELSE
                ForeColor;
        -- check screen window too.
        R <= Color(0) AND window;
        G <= Color(1) AND window;
        B <= Color(2) AND window;
END BLOCK Color_Out;
```

รูปที่ 4-34 แสดงส่วนของโปรแกรม Color_Out

Memory_Remote_Interface

การอธิบายวงจรเพื่อการสังเคราะห์วงจรในส่วนของสัญญาณ Data ซึ่งเป็นทั้งสัญญาณเข้าและสัญญาณออก ทางบริษัท Xilinx แนะนำให้ใช้การเขียนแผนภาพแล้วนำมาต่อเชื่อมกับการใช้ภาษาวีเอชดีแอลในภายหลังจะสะดวกกว่า จึงไม่ได้อธิบายสัญญาณดังกล่าวในที่นี้

ส่วนสัญญาณอื่น ๆ ได้ปรับปรุงจากการอธิบายเชิงพฤติกรรมเพื่อให้สังเคราะห์วงจรได้ดังแสดงในรูปที่ 4-35

1. Select_read_write Process ทำหน้าที่จัดการกำหนดขา address ให้แก่ตำแหน่งข้อมูล row,column และ page ต่าง ๆ, รวมทั้งกำหนดจังหวะการอ่านเขียนดังได้กล่าวมาแล้ว

2. Remote_Interface Block ทำหน้าที่ในรับสัญญาณ remote control ที่ได้รับเข้ามา ซึ่งสัญญาณดังกล่าวเข้ารหัสแบบ one-hot กล่าวคือ มีขนาด 10 บิต แต่จะมีบิตที่เป็นหนึ่งได้เพียงบิตเดียว ตำแหน่งของข้อมูลที่เป็นหนึ่งจะเป็นค่าของข้อมูลทั้งหมดเช่น "00_0000_0001" ตำแหน่งที่มีค่าเป็นหนึ่งคือศูนย์ ค่าของมูลเท่ากับศูนย์ หรือ "00_0010_0000" จะเท่ากับห้า มาแปลงให้เป็นข้อมูลเลขฐานสองแล้วจัดเก็บเพื่อใช้ในการเลือกหน้าข้อมูลต่อไป

2.1 Clock_Gen Process มีหน้าที่ในการตรวจจับว่ามีสัญญาณจากรีโมทคอนโทรล หรือไม่โดยการตรวจสถานะสัญญาณ page_r ว่ามีบิตใดมีค่าเป็นหนึ่งหรือไม่ หากมีจะสร้างสัญญาณ rclk เพื่อกระตุ้นให้ Process ต่อไปทำงาน

2.2 Store_Process เมื่อเกิดสัญญาณ rclk ขึ้นข้อมูล one-hot จะถูกเปลี่ยนเป็นข้อมูลแบบเลขฐานสองและเก็บไว้ในรีจีสเตอร์ เพื่อที่จะ mapping เป็น address ที่จะไปอ่านข้อมูลในการแสดงผลต่อไป

```
select_read_write:
        PROCESS(row_r,row_w,col_r,col_w,page_disp,page_w,char,roc,one_meg)
                VARIABLE    page        : std_ulogic_vector(9 downto 0) ;
                VARIABLE    col         : std_ulogic_vector(5 downto 0) ;
                VARIABLE    row         : std_ulogic_vector(4 downto 0) ;
                VARIABLE    adr_char : std_ulogic_vector(19 downto 0);
        BEGIN
                IF (one_meg(5)='1' OR one_meg(0)='1') THEN
                        col := col_r;
                        row := row_r;
                        page := page_disp;
                ELSE
                        col := col_w;
                        row := row_w;
                        IF (row_w /= "00000") THEN
                                page := page_w;
                        ELSE
                                page := page_disp;
                        END IF;
                END IF;
                adr_char(2 downto 0) := col(2 downto 0);
                adr_char(19 downto 10) := page(9 downto 0);
                CASE col(5) IS
                        WHEN '0' =>
                        adr_char(9 downto 3) := row(4 downto 0) & col(4 downto 3);
```

```
                    WHEN OTHERS =>
                    adr_char(9 downto 3) := "11" & not (row(4 downto 0));
                    END CASE;
                    IF (one_meg(2)/='1') THEN
                            ADR <= adr_char(18 downto 0);
                    ELSE
                            ADR <= "00000100" & char & RoC;
                    END IF;
                END PROCESS ;

Remote_Interface Block:
BEGIN
clock_gen: PROCESS(page_r)
            VARIABLE sum : std_ulogic;
        BEGIN
            sum := '0';
            FOR i IN page_r'RANGE LOOP
                    sum := sum OR page_r(i);
            END LOOP;
            rclk <= sum;
        END PROCESS clock_gen;

        store_process: PROCESS(rclk)
        BEGIN
            IF (rising_edge(rclk)) THEN
                    remote_reg(0) <= encode(page_r);
                    remote_reg(1) <= remote_reg(0);
                    remote_reg(2) <= remote_reg(1);
            END IF;
        END PROCESS store_process;

        Map_Page:
        page_disp <=
        remote_reg(2)(2 downto 0)&remote_reg(1)&remote_reg(0)(2 downto 0)
```

```
WHEN remote_reg(0)(3) = '0' ELSE
remote_reg(2)(2 downto 0)&"11"&remote_reg(1)(1 downto 0)&remote_reg(1)(3 downto 2)&
remote_reg(0)(0);

END BLOCK Remote_Interface;
```

รูปที่ 4-35 แสดงส่วนโปรแกรมของ Memory_Remote_Interface

การวิเคราะห์วงจรด้วยซอฟต์แวร์

หลังจากที่ได้ออกแบบวงจรด้วยภาษาวีเอชดีแอลแล้วขั้นตอนต่อไปคือการวิเคราะห์วงจรด้วยซอฟต์แวร์

องค์ประกอบที่สำคัญของการวิเคราะห์วงจรด้วยซอฟต์แวร์ก็คือชุดทดสอบสัญญาณ (Test Vector) ซึ่งจะเป็นสัญญาณเข้าและ/หรือรับสัญญาณออก เพื่อกระตุ้นให้วงจรที่ออกแบบทำงาน

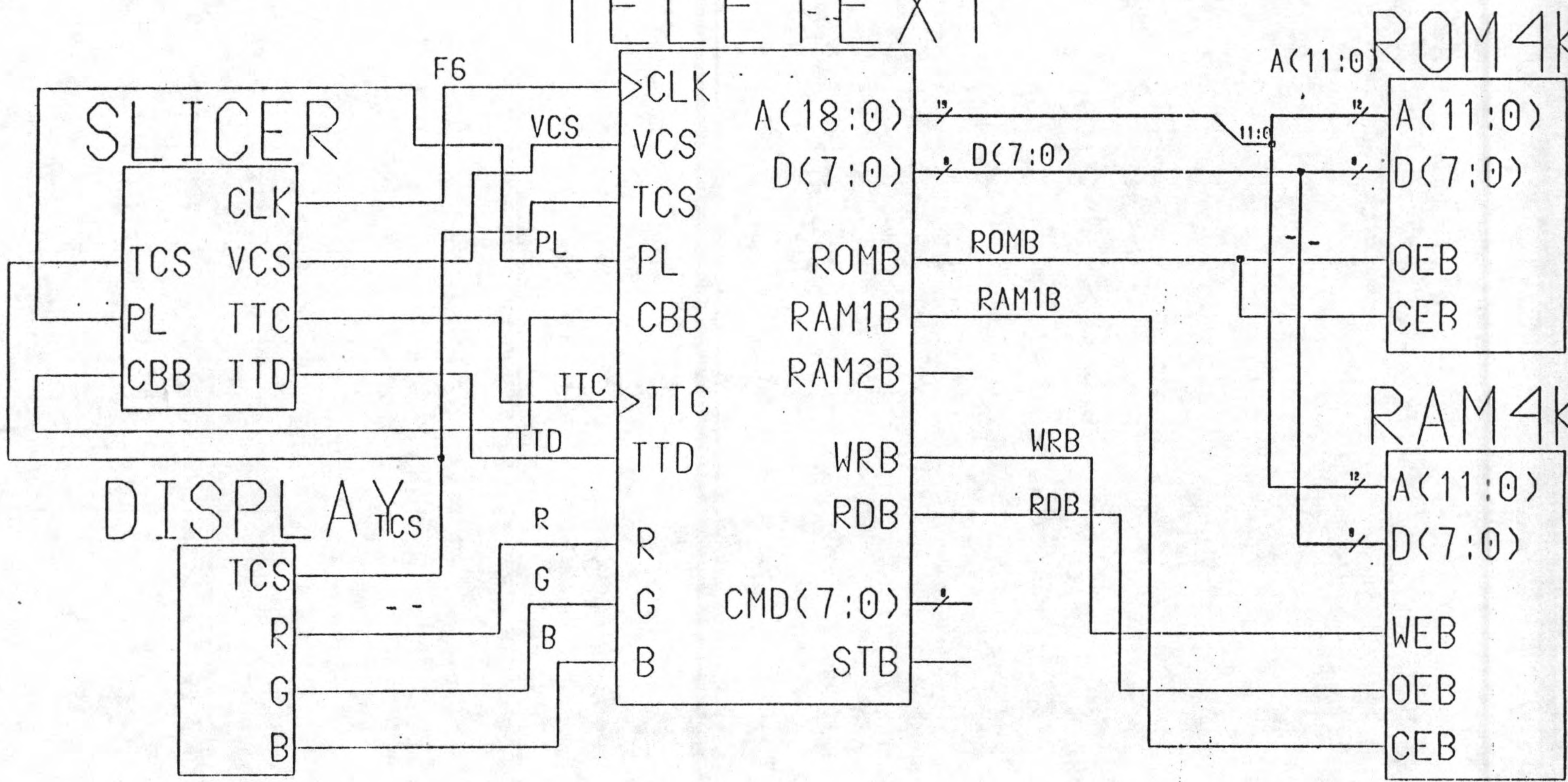
การใส่ค่าชุดทดสอบสัญญามีหลายวิธี เช่น การใส่ที่ละค่า, การใส่ค่าเป็นชุดคำสั่งของซอฟต์แวร์วิเคราะห์วงจร แต่วิธีดังกล่าวไม่เหมาะกับวงจรเทเลเท็กซต์เพราะเหตุว่าสัญญาณ ttd ต้องเปลี่ยนแปลงค่าตลอดเวลาและจะต้องมีการเก็บข้อมูลไว้ในหน่วยความจำก่อนที่จะนำมาแสดงผลด้วย

ในวิทยานิพนธ์ได้สร้างแบบจำลองภาษาวีเอชดีแอลของอุปกรณ์อีกชุดหนึ่งจำลองการทำงานของส่วนต่างๆที่ Teletext IC ดังแสดงรายละเอียดการเขียนแบบจำลองไว้ในภาคผนวก ก. ซึ่งจะไปรับหรือส่งข้อมูลเพื่อใช้เป็นชุดทดสอบสัญญาณดังรูปที่ 4-36

1. Slicer มีหน้าที่คล้ายกับ VIP เพียงแต่สัญญาณเข้าของ Slicer มิใช้สัญญาณภาพ หากแต่เป็นแฟ้มข้อมูลที่เก็บตัวอักษรที่เป็นข้อความหรือเป็นคำสั่ง ซึ่ง Slicer มีหน้าที่สร้างความถี่ 6 MHz, สร้างสัญญาณ vcs โดยเฟสของ vcs จะต้องเข้าจังหวะกับสัญญาณ pl ที่รับเข้ามาจาก Teletext IC ด้วย ส่วนสัญญาณ ttc จะสังเคราะห์ให้มีความถี่ประมาณ 6.9375 MHz และสัญญาณ ttd ได้มาจากการอ่านแฟ้มข้อมูลจากภายนอกที่เป็นตัวอักษร 8 บิต แล้วเปลี่ยนเป็นข้อมูลแบบอนุกรมส่งไปให้กับ Teletext IC ต่อไป

2. Monitor มีหน้าที่รับสัญญาณ RGB และ tcs มาเก็บไว้ในแฟ้มข้อมูลหลั้งจากนั้นจะเขียนโปรแกรมภาษา C เพื่ออ่านคปรแกรมนั้นเพื่อแสดงผลยนเครื่องคอมพิวเตอร์ PC ในการที่จะตรวจสอบผลการวิเคราะห์ข้อมูลต่อไป

3. หน่วยความจำ RAM และ ROM เป็นส่วนที่ใช้เก็บข้อมูลต่างๆ โดยการอ่านข้อมูลเริ่มต้นจากแฟ้มข้อมูลโดยที่หน่วยความจำ ROM จะเก็บข้อมูลที่เป็น font ของด้วยอักษร ส่วนหน่วยความจำ RAM จะถูกเขียนข้อมูบที่รับได้จาก Slicer ซึ่งจะถูกอ่านกลับในตำแหน่งที่ผูใช้เลือกชมเพื่อให้ Monitor แสดงผลในหน้าดังกล่าว



รูปที่ 4-36 แสดงบล็อกไดอะแกรมในการวิเคราะห์วงจร