# บทที่ 5

## การออกแบบซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทาง

ในบทที่ 4 ได้กล่าวถึงการออกแบบฮาร์ดแวร์ของอุปกรณ์สื่อสารปลายทาง ในบทนี้จะกล่าวถึงการออกแบบซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทางซึ่งเป็นส่วนที่ออกแบบและสร้างขึ้นใหม่หมด ส่วนสำคัญที่ต้องทราบในการออกแบบซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทาง ได้แก่ ความสามารถของ PEB2085 และการสั่งงานไปยัง PEB2085 จึงจำเป็นต้องศึกษาสมบัติทางซอฟต์แวร์ของ PEB2085 ซึ่งในบทนี้จะกล่าวเฉพาะส่วนที่สำคัญหรือเกี่ยวข้องกับการออกแบบเท่านั้น

สมบัติทางซอฟต์แวร์ของ PEB2085

PEB2085 ถูกออกแบบให้มีหน่วยความจำภายในสำหรับเก็บข้อมูลที่ต้องการส่งออกไปที่จุดอ้างอิง S ขนาด 64 ไบท์ หน่วยความจำภายในสำหรับเก็บข้อมูลที่รับได้จากจุดอ้างอิง S ขนาด 64 ไบท์ และหน่วยความจำสำหรับเก็บสถานะและสั่งงานภายใน เรียกว่ารีจิสเตอร์ (Register) โดยติดต่อกับตัวประมวลผลผ่านทางสัญญาณอินเตอร์รัพท์ (John B. Peatman, 1988) ซึ่งมีเพียง 1 อินเตอร์รัพท์เท่านั้น ดังนั้นตัวประมวลผลต้องอ่านสถานะหรือข้อมูลจากรีจิสเตอร์ของ PEB2085 หากได้รับสัญญาณอินเตอร์รัพท์ เพื่อแปลความหมายของสาเหตุของอินเตอร์รัพท์นั้น โครงสร้างทางซอฟต์แวร์ของ PEB2085 สามารถแบ่งได้เป็น 2 ส่วน คือ

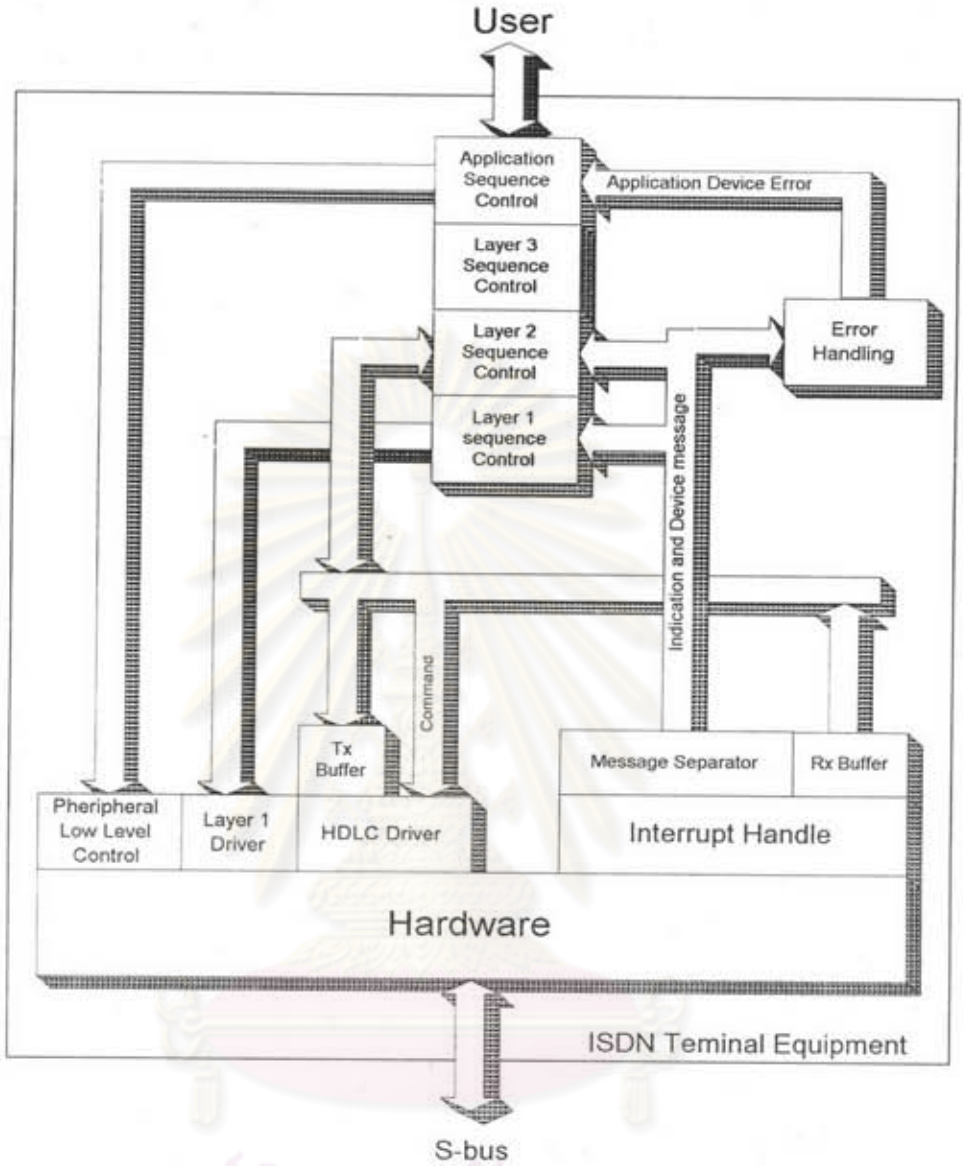
1. ส่วนที่เกี่ยวข้องกับชั้นที่ 1 ตามแบบจำลองของ OSI PEB2085 มีความสามารถในการทำงานคลอบคุมโปรโตคอลในชั้นที่ 1 ทั้งหมด ผู้ใช้เพียงแต่ส่งคำสั่งที่ต้องการ เช่น Class ที่ต้องการ Activate (ITU-T, 1988a) เป็นต้น ไปที่ PEB2085 หากมีการเปลี่ยนแปลงสถานะในชั้นที่ 1 PEB2085 จะรายงานให้ทราบผ่านทางรีจิสเตอร์และสัญญาณอินเตอร์รัพท์

2. ส่วนที่เกี่ยวข้องกับชั้นที่ 2 ตามแบบจำลองของ OSI ซึ่ง PEB2085 มีหน่วยความจำสำหรับรับและส่งข้อมูลเข้าออกสู่จุดอ้างอิง S การทำงานทั้งหมดผ่านทางสัญญาณอินเตอร์รัพท์ (John B. Peatman, 1988) เช่นกัน ผู้ใช้สามารถตรวจสอบสถานะของข้อมูลที่ได้รับและสถานะของหน่วยความจำสำหรับรับและส่งข้อมูลเข้าออกสู่จุดอ้างอิง S เช่น CRC หรือ FCS (ITU-T, 1988b) ของข้อมูลที่ได้รับถูกต้องหรือไม่ หน่วยความจำพร้อมจะรับข้อมูลใหม่หรือไม่ เป็นต้น ผ่านทาง

สัญญาณอินเตอร์รัพท์และรีจิสเตอร์ของ PEB2085 การทำงานในส่วนนี้แบ่งออกเป็น 2 แบบ ได้แก่ Auto mode และ Non-auto mode ใน Auto mode PEB2085 จะควบคุมการทำงานในสถานะ 7 และ 8 ตามมาตรฐาน Q.920-Q.921 โดยกำหนดให้ window size เท่ากับ 1 (ITU-T, 1988b) นอกจากนี้ PEB2085 ยังมี T200 Timer ภายใน การเติมค่าในฟิลด์ควบคุมโดยอัตโนมัติและสามารถควบคุมการโต้ตอบของ S-frame (ITU-T, 1988b) ได้ แต่สำหรับ Non-auto mode ผู้ใช้ต้องเขียนโปรแกรมสำหรับควบคุมสิ่งเหล่านี้เองทั้งหมด

<u>การออกแบบซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทาง</u>

หลังจากที่ได้ทราบสมบัติทางซอฟต์แวร์ของ PEB2085 แล้ว จึงทำการออกแบบซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทาง วิทยานิพนธ์นี้ได้สั่งงานให้ PEB2085 ทำงานใน Auto mode จากสมบัติของ PEB2085 พบว่าการทำงานของซอฟต์แวร์ต้องทำงานเป็นแบบอินเตอร์รัพท์ นอกจากนี้เพื่อความสะดวกในการพัฒนาอุปกรณ์สื่อสารปลายทางให้มีประสิทธิภาพมากขึ้นในภายหลัง จึงออกแบบให้ซอฟต์แวร์มีลักษณะเป็นฟังก์ชั่นดังที่ได้กล่าวมาแล้วในบทที่ 3 ซอฟต์แวร์ที่ออกแบบแสดงเป็นบล็อกไดอะแกรมได้ดังรูปที่ 5.1

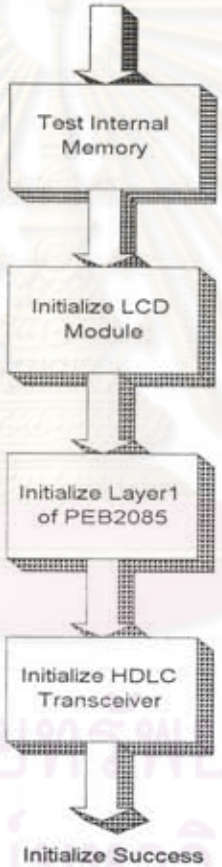



รูปที่ 5.1 แสดงบล็อกไดอะแกรมของซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทาง

รูปที่ 5.1 แสดงบล็อกไดอะแกรมของซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทาง ซึ่งแสดงถึงฟังก์ชั่นหลัก ความสัมพันธ์และทิศทางการไหลของข้อมูลระหว่างฟังก์ชั่นเหล่านั้น ซึ่งเป็นส่วนสำคัญในการออกแบบซอฟต์แวร์ เนื่องจากการแก้ไขฟังก์ชั่นหลัก ชนิด จำนวนและทิศทางการไหลของข้อมูลจะส่งผลอย่างมากต่อโครงสร้างของซอฟต์แวร์โดยรวม

หลังจากออกแบบบล็อกไดอะแกรมของซอฟต์แวร์แล้ว ต่อไปเป็นการแบ่งฟังก์ชั่นหลักที่มีความซับซ้อนออกเป็นฟังก์ชั่นย่อยหลายๆ ฟังก์ชั่น การออกแบบซอฟต์แวร์ของอุปกรณ์สื่อสารปลายทางในลักษณะนี้ ทำให้สะดวกต่อการแก้ไขหรือปรับปรุงประสิทธิภาพของอุปกรณ์สื่อสาร

ปลายทางในภายหลัง การแก้ไขโปรแกรมภายในฟังก์ชั่นสามารถทำได้โดยไม่ส่งผลกระทบต่อการทำงานในฟังก์ชั่นอื่นของซอฟต์แวร์หากไม่มีการเปลี่ยนแปลงข้อมูลที่เข้าออกจากฟังก์ชั่นนั้นๆ ซึ่งจะทำให้การแก้ไขหรือการพัฒนาซอฟต์แวร์ทำได้โดยง่าย หากซอฟต์แวร์ไม่ทำงานเมื่อมีการแก้ไขฟังก์ชั่น จะสามารถหาสาเหตุได้โดยง่ายโดยเกิดจากฟังก์ชั่นที่ทำการแก้ไขหรือฟังก์ชั่นข้างเคียงเท่านั้น ตัวอย่างเช่น ฟังก์ชั่น System Initialize ในบทที่ 3 ประกอบด้วยฟังก์ชั่นทดสอบหน่วยความจำภายใน ฟังก์ชั่น Initialize LCD ฟังก์ชั่น Initialize Layer1 ของ PEB2085 และฟังก์ชั่น Initialize HDLC Transceiver ของ PEB2085 ดังรูปที่ 5.2




รูปที่ 5.2 แสดงฟังก์ชั่นย่อยของฟังก์ชั่น Initialize System

การออกแบบเป็นฟังก์ชั่นในลักษณะนี้ทำให้ง่ายต่อการเขียนโปรแกรมให้ทำงานตามโปรโตคอลในชั้นที่ 1 ถึงชั้นที่ 3 ตามแบบจำลองของ OSI เนื่องจากแบบจำลองของแต่ละชั้นจะมีจุดเข้าที่เรียกว่า Service Access Point (ประสิทธิ์, 2535) ในแต่ละชั้นเพียงจุดเดียว และการทำงานหรือการเปลี่ยนแปลงรายละเอียดภายในของแต่ละชั้นจะไม่ส่งผลต่อชั้นที่อยู่ข้างเคียง ซึ่งจุด

Service Access Point นี้เปรียบเสมือนค่าพารามิเตอร์ที่ส่งเข้าสู่ฟังก์ชั่น และการเปลี่ยนแปลงรายละเอียดภายในฟังก์ชั่นต้องไม่ส่งผลต่อฟังก์ชั่นข้างเคียง

Program Description Language ของอุปกรณ์สื่อสารปลายทาง

หลังจากที่ได้ออกแบบซอฟต์แวร์แล้วจึงทำการแปลงฟังก์ชั่นที่ได้ออกแบบไว้ในรูปแบบต่างๆ เช่น บล็อกไดอะแกรม Functional Decomposition เป็นต้น ให้อยู่ในรูปของ Program Description Language หรือ PDL (Boehm, B.W., 1988) ซึ่งภาษา PDL นี้เป็นภาษาที่ไม่มีรูปแบบขึ้นอยู่กับผู้พัฒนา เป็นภาษาที่อธิบายการทำงานของโปรแกรม แบ่งได้เป็น 3 ระดับ คือ

1. PDL ระดับที่ 1 เป็นระดับที่อธิบายการทำงานของซอฟต์แวร์โดยรวม บอกถึงโครงสร้างหลักของโปรแกรม

2. PDL ระดับที่ 2 เป็นระดับที่บอกรายละเอียดมากขึ้นกว่าระดับที่ 1 บอกถึงโครงสร้างของโปรแกรมและโครงสร้างของฟังก์ชั่นที่สำคัญ

3. PDL ระดับที่ 3 เป็นระดับที่บอกรายละเอียดมากที่สุด บอกถึงรายละเอียดการเรียกใช้ฟังก์ชั่นต่างๆ ค่าที่ต้องเขียนลงในตัวแปรหรือหน่วยความจำต่างๆ ดังนั้น PDL ในระดับที่ 3 นี้จึงสามารถนำไปเข้ารหัสให้เป็นโปรแกรมเป็นภาษาใดก็ได้ เช่น ภาษาซีหรือภาษาแอสเซมบลี

ในวิทยานิพนธ์นี้ได้ใช้ PDL ระดับที่ 3 เข้ารหัสเป็นภาษาแอสเซมบลีของ 8031 แต่รายละเอียดที่จะแสดงต่อไปนี้เป็น PDL ในระดับที่ 1 และ 2 เพียงบางส่วน เนื่องจาก PDL ระดับที่ 2 และระดับที่ 3 มีรายละเอียดมาก นอกจากนี้ PDL ระดับที่ 2 และ 3 ในวิทยานิพนธ์นี้เป็น PDL สำหรับควบคุม PEB2085 เท่านั้น หากผู้พัฒนาผู้อื่นใช้ไอซีของบริษัทอื่นหรือเบอร์อื่น PDL ระดับที่ 2 บางส่วนและระดับที่ 3 ในวิทยานิพนธ์นี้ก็ไม่สามารถใช้งานได้ จึงใส่ไว้เฉพาะ PDL ระดับที่ 1 และระดับที่ 2 บางส่วนที่แสดงถึงโครงสร้างของซอฟต์แวร์และแนวความคิดในการออกแบบเท่านั้น

PDL ระดับที่ 1

```
Function: Main
{
        Initialize;
        Do
                Select Case
```

```
                    Case L1_ACQ:
                        Layer 1 Handle Routine;
                        Break;
                    Case L2_ACQ:
                        Layer 2 Handle Routine;
                        Break;
                    Case L3_ACQ:
                        Layer 3 Handle Routine;
                        Break;
                    Case ERROR_ACQ:
                        Error Handle Routine;
                        Break;
                    Case LP_ACQ:
                        Application Handle Routine;
                        Break;
                End Select;
        Loop;
}
Function: ISAC-S Interrupt Handle        /* Hardware Interrupt 0 */
{
        Select Case
                Case CISQ Interrupt:
                    Layer 1 Interrupt Handle Routine;
                    Setbit L1_ACQ;
                    Break;
                Case Else
                    HDLC Interrupt Handle Routine;
                    Setbit L2_ACQ;
                    Break;
```

```
        End Select;
}
Function: Application Interrupt Handle   /* Hardware Interrupt 1 */
{
        /* Upon Application */
        Setbit LP_ACQ;
}
```

จาก PDL ระดับที่ 1 จะเห็นว่าโครงสร้างของซอฟต์แวร์จะทำงานอยู่ที่ฟังก์ชั่น Main จนกว่าจะมีการเปลี่ยนแปลงของแฟลก (flag) สถานะจึงจะเข้าไปทำงานตามฟังก์ชั่นต่างๆ ซึ่งแฟลกเหล่านี้จะมีการเปลี่ยนแปลงเมื่อเกิดการอินเตอร์รัพท์จาก PEB2085 จาก Application Interrupt Handle โดยผู้ใช้ หรือจากฟังก์ชั่นอื่นๆ ที่มีผลจากอินเตอร์รัพท์ทั้งสอง เช่น ฟังก์ชั่น Layer 3 Handle Routine ทำงานเนื่องจากการเปลี่ยนแปลงค่าแฟลกจากฟังก์ชั่น Layer 2 Handle Routine โดยมีผลจากฟังก์ชั่น ISAC-S Interrupt Handle เป็นต้น ส่วน PDL ระดับที่ 2 แสดงถึงรายละเอียดของการทำงานของซอฟต์แวร์ที่มากขึ้น

PDL ระดับที่ 2

```
Function: Main
Parameter:
Var:
Return:
{
        Disable all intr. to avoid fault intr. at startup;
        Init_Proc();
        Clear all function request;
        REQ_L1_CLASS=Class8;
        L2_STATE=TEI unassign;
        L3_state=NULL;
        Do
```

```
        Select case
            Case L1_ACQ:            /* L1_ACQ indicate */
                Clearbit IE.2;
                Layer1();       /* Call Layer 1 handle routine */
                Setbit IE.2;
                Break;
            Case L2_ACQ or (L2_state=Link Establish and DL-data request):
                Clearbit IE.2;
                Layer2();       /* Call Layer 2 handle routine */
                Setbit IE.2;
                Break;
            Case L3_ACQ:            /* L3_ACQ indicate */
                Clearbit IE.2;
                Layer3();       /* Call Layer 3 handle routine */
                Setbit IE.2;
                Break;
            Case LP_ACQ:            /* LP_ACQ indicate */
                Clearbit IE.2;
                Application(); /* Call application handle routine */
                Setbit IE.2;
                Break;
            Case Error_ACQ:         /* Error_ACQ indicate */
                Clearbit IE.2;
                Error_Handle(); /* Call Error Handle routine */
                Setbit IE.2;
                Break;
        Endselect
    Loop
}
```

```
Function: Error_Handle
Parameter: ERROR_ACQ, error message;
Var:
Return:
{
        Clearbit ERROR_ACQ;
        Select Case
                Case Error message:
                        Error handle function upon error message;
                        Setbit L3_ACQ,L2_ACQ or LP_ACQ upon error message;
        Endselect
}
Function: Layer3
Parameter: L3_ACQ, message to layer 3;
Var:
Return: L2_ACQ, message to layer 2, LP_ACQ, message to  Application function;
{
        Clearbit L3_ACQ;
        if rframe_type=information then
                L3_Decode(information in buffer);   /* decode layer 3 frame */
        endif
        Select Case
                Case Layer 3 current state:
                        Select Case
                                Case Incoming Message:
                                        Handle Function upon incoming message;
        End Select
        Break;
}
```

```
Function: Layer2
Parameter: L2_ACQ message to layer 2;
Var:
Return: L1_ACQ, layer 1 activate class, L3_ACQ, message to layer 3;
{
        Clearbit L2_ACQ;
        if request to send U frame then
                if layer 1 not in F7 state then
                        Setbit L1_ACQ; /* Activate Layer 1 */
                        return;
                endif
                Sendframe_ICC(UI frame);
                return;
        endif
        Select Case
                Case Layer 2 current state:
                        Select Case
                                Case message to layer2:
                                        Handle function upon message to layer 2;
                                        Break;
                        End select
                        Break;
        End Select
}
Function: Layer1
Parameter: L1_ACQ, Layer 1 activate Class;
Var:
Return: L2_ACQ, Layer 1 state;
{
```

```
        Clear L1_ACQ;
        if deactivate request from LT-S then
                Send deactivet to ISAC-S;
                Wait for Layer1 State F3 or F7 or timeout;
        else
                if Layer1 not in state7 or current class<>Layer 1activate Class then
                        Act_L1(Layer 1 activate Class);
                endif
        endif
        Setbit L2_ACQ;
}
Function: Init_Proc
Parameter:
Var:
Return:
{
        Testram();
        Init_2085();
        Clear all message.
}
Function: Testram /* function to test external memory */
Parameter:
Var: pointer;
Return: System Halt or none;
{
        Write value 0ffh to memory;
        Read data from memory;
        if ~0ffh then
                Setbit ERROR_ACQ;
```

```
                return;
        endif
        Loop until all address are being tested;
}
Function: Init_2085     /* Initialize ISAC-S 2085 */
Parameter:
Var:
Return:
{
        Initialize Layer 1;
        Initialize Layer 2 HDLC Transceiver;
}
Interrupt Function: T20X_interrupt      /* use 8031 Timer1 interrupt */
Parameter: Use_timer1,T20x_counter,N20x_counter;
Var:
Return:
{
        Stop and reset Timer1;
        if Use_timer1=T203 then
                if T203 timeout then
                        if N203 expire then
                                Setbit ERROR_ACQ;
                                return;
                        endif
                        Send RR or RNR;
                endif
        elseif Use_timer1=T202 then
                if T202 timeout then
                        if N202 expire then
```

```
                    Setbit ERROR_ACQ;
                    return;
                endif
            endif
        elseif Use_timer1=T200 then
            if T200 timeout then
                if N200 expire then
                    Setbit ERROR_ACQ;
                    return;
                endif
            endif
        endif
        Start Timer1;
}
```

PDL ระดับที่ 2 ที่แสดงไว้ข้างต้นเป็นเพียงส่วนหนึ่งเท่านั้น เพื่อให้เห็นแนวความคิดในการออกแบบซอฟต์แวร์ เมื่อได้ PDL ระดับที่ 2 แล้ว จึงออกแบบ PDL ระดับที่ 3 เพื่อนำไปเข้ารหัสเป็นภาษาแอสเซมบลีต่อไป การแก้ไขโปรแกรมภาษาแอสเซมบลีจะมีผลต่อโครงสร้างของ PDL ระดับที่ 3 เท่านั้น เนื่องจากเป็นระดับในรายละเอียด แต่จะไม่มีผลต่อ PDL ระดับที่ 1 หรือ 2 แต่อย่างใด จาก PDL ระดับที่ 2 จะเห็นว่าเริ่มปรากฏฟังก์ชั่นที่ทำงานในชั้นที่ 1 2 และ 3 ตามแบบจำลองของ OSI ซึ่งจะปรากฏเป็นรายละเอียดการทำงานใน PDL ระดับที่ 3 อย่างไรก็ตาม PDL ระดับที่ 2 ก็ได้บอกถึงแนวคิดในขั้นตอนทำงานของโปรแกรมเพื่อให้โปรแกรมทำงานตามโปรโตคอลในระดับชั้นที่ 1 2 และ 3 ได้ สำหรับแนวคิดในการเปลี่ยนโปรโตคอลในชั้นที่ 1 2 และ 3 ให้เป็น PDL ระดับที่ 3 จะกล่าวถึงในเรื่องวิธีการสร้างซอฟต์แวร์ในบทที่ 6 ต่อไป

<u>เครื่องมือที่ใช้พัฒนาและทดสอบซอฟต์แวร์</u>

การออกแบบวิธีทดสอบซอฟต์แวร์นั้น มีผลโดยตรงต่อลักษณะการเขียนโปรแกรม จากการออกแบบในบทที่ 3 4 และ 5 หากสังเกตจะพบว่าการออกแบบนั้นจะทำจากระดับโครงสร้างที่มีขนาดใหญ่ไปสู่โครงสร้างที่มีขนาดเล็ก (Top-Down) คือตั้งแต่การทำงานหลักๆ ของอุปกรณ์สื่อ

สารปลายทางจนถึงการทำงานระดับฟังก์ชั่นย่อย ส่วนวิธีการเขียนโปรแกรมมีใช้กันอยู่หลายวิธี ในวิทยานิพนธ์นี้ได้เลือกใช้วิธีพัฒนาโปรแกรมโดยพัฒนาโครงสร้างที่มีขนาดเล็กขึ้นมาจนถึงโครงสร้างที่มีขนาดใหญ่ (Bottom-Up)

การเขียนโปรแกรมในลักษณะนี้ หลังจากเขียนโปรแกรมของฟังก์ชั่นย่อยหรือฟังก์ชั่นหนึ่งๆ เรียบร้อยแล้วให้ทำการทดสอบการทำงานของฟังก์ชั่นนั้นทันที ก่อนที่จะเขียนโปรแกรมของฟังก์ชั่นอื่นต่อไป ด้วยการเขียนโปรแกรมจำลองการทำงานของฟังก์ชั่นที่สูงกว่าให้เรียกใช้หรือติดต่อกับฟังก์ชั่นที่ต้องการทดสอบ เรียกฟังก์ชั่นนี้ว่า Stub (John B.Peatman, 1988) Stub จะรับส่งข้อมูลกับฟังก์ชั่นที่ทำการทดสอบ ข้อมูลที่ส่งกลับไปนั้นมิได้เกิดจากผลการทำงานตามหน้าที่ที่แท้จริงของฟังก์ชั่นที่ทำหน้าที่เป็น Stub นั้น แต่เป็นเป็นข้อมูลที่ stub สร้างขึ้นมาเอง ตัวอย่างเช่น ต้องการทดสอบการทำงานของฟังก์ชั่นการแสดงผลออกที่จอ LCD โดยฟังก์ชั่นนี้รับค่าตัวแปร คือ จำนวนตัวอักษรที่ต้องการแสดงผลและตำแหน่งเริ่มต้นของข้อความ ให้เขียนโปรแกรม Stub ขึ้นมา Stub จะเรียกใช้ฟังก์ชั่นแสดงผลนี้โดยส่งค่าตัวแปรค่าต่างๆ เข้าไปที่ฟังก์ชั่นแสดงผลทั้งค่าที่ถูกต้องซึ่งสามารถแสดงผลได้และค่าอื่นๆ ที่อาจทำให้เกิดข้อผิดพลาดขึ้น จากนั้นติดตามผลการทำงานและแก้ไขฟังก์ชั่นการแสดงผลนี้ให้ถูกต้องต่อไป

สำหรับซอฟต์แวร์ของ LT-S จากเดิมที่มีการรับส่งเฟรมที่เป็นไปตามโปรโตคอลในชั้นที่ 2 และ 3 ได้มีการปรับปรุงให้รับส่งหรือโต้ตอบเฟรมใดๆ ก็ได้ ไม่จำเป็นต้องเป็นไปตามโปรโตคอลในชั้นที่ 2 และ 3 เพื่อทดสอบการทำงานของอุปกรณ์สื่อสารปลายทางในกรณีที่ได้รับเฟรมที่ผิดพลาดจากชุมสาย

สำหรับเครื่องมือที่ใช้ในการพัฒนาซอฟต์แวร์นั้นได้ใช้โปรแกรม ASM51 ของบริษัท Intel Corporation. ซึ่งเป็นตัวแปรภาษาจากภาษาแอสเซมบลีไปเป็นภาษาเครื่องของ 8031 การติดตามและตรวจสอบการทำงานของซอฟต์แวร์นั้นใช้วิธีแทรกคำสั่งให้ LED สว่างหรือดับภายในตำแหน่งต่างๆ ของโปรแกรม ดังได้กล่าวไว้ในบที่ 4