การออกแบบซีพียูโดยใช้ FPGA

CPU DESIGN BY FPGA

โดย

นาย รัฐกาล ฤทธิศักดิ์

นาย ชูชาติ สอดศรี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

การออกแบบซีพียูโดยใช้ FPGA

CPU DESIGN BY FPGA

โดย

นาย รัฐกาล ฤทธิศักดิ์ 43015272

นาย ชูชาติ สอดศรี 43015257

อาจารย์ที่ปรึกษา

รศ.ดร.มนัส สังวรศิลป์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

การออกแบบซีพียูโดยใช้ FPGA

CPU DESIGN BY FPGA

นายรัฐกาล ฤทธิศักดิ์ 43015272

นายชูชาติ สอดศรี 43015257

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....

(รศ.ดร. มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบซีพียูโดยใช้ FPGA

ผู้จัดทำ

1.นายรัฐกาล ฤทธิศักดิ์

2.นายชูชาติ สอดศรี

.....อาจารย์ที่ปรึกษา

(รศ.ดร. มนัส สังวรศิลป์)

การออกแบบซีพียูโดยใช้ FPGA

รัฐกาล ฤทธิศักดิ์
ชูชาติ สอดศรี
รศ.ดร. มนัส สังวรศิลป์
ปีการศึกษา 2545

บทคัดย่อ

ในปริญญานิพนธ์ได้นำเสนอการใช้เอฟพีจีเอ(FPGA : Field Programmable Gate Array) เพื่อออกแบบ ซีพียู ขนาด 8 บิท ซึ่งปัจจุบันเราได้นำสินค้าอุปกรณ์เหล่านี้เข้ามาในประเทศเป็นจำนวนมาก ทางคณะผู้จัดทำจึงมีแนวความคิดว่าเราน่าจะทำการออกแบบตัวซีพียู ขึ้นมาใช้เองได้ โดยมีชุดคำสั่งเป็นของเราเอง ประกอบกับปัจจุบันได้มีอุปกรณ์ ที่สามารถรองรับการออกแบบวงจรทางดิจิตอลโดยที่ไม่จำเป็นต้องนำอุปกรณ์แบบเดี่ยวมาต่อกันจึงทำให้เพิ่มความสะดวกในการออกแบบยิ่งขึ้น ซึ่งซีพียูที่จัดทำขึ้นแม้จะมีความสามารถไม่ทัดเทียมกับซีพียูขนาดเดียวกันที่มีขายตามท้องตลาดแต่ก็พอจะเป็นแนวทางในการที่จะพัฒนาให้ซีพียูรุ่นต่อไปมีความสมบูรณ์มากยิ่งขึ้น โดยการออกแบบส่วนต่างๆ ได้แสดงรายละเอียดไว้ในปริญญานิพนธ์ดังนี้คือ ส่วนประกอบต่างๆ ของซีพียู และรายละเอียดของแต่ละชุดคำสั่ง รวมทั้งผลการซิมมูเลท และ โปรแกรมที่ใช้ในการทดลอง

CPU DESIGN BY FPGA

Mr.Rattakan Rittisak

Mr.Chuchat Sodsri

Assoc.Prof.Dr.Manas Sangworasilp

Educational Year 2002

Abstract

This project presents the 8 bit CPU (Central Processing Unit), which is synthesized by using FPGA (Field Programmable Gate Array). This device has been used in many works one can design his/her personal CPU with its own instruction see without-importing costly CPU from abroad. The design process is very simple. Although the performance of the designed CPU using FPGA is not as good as the full custom version the improvement can be come out for the next design. This thesis shows the details of the CPU components, instruction set, simulation results and program used in the simulations.

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1 บทนำ

1.1 ความเป็นมา

ในปัจจุบันคอมพิวเตอร์เข้ามามีส่วนในการดำรงชีวิตของมนุษย์มากขึ้น ซึ่งการทำงานของคอมพิวเตอร์ประกอบไปด้วย ส่วนของ ซอฟต์แวร์และอาร์ดแวร์ การทำงานต่างๆ ถูกควบคุม ด้วยหน่วยประมวลผลกลาง (central processing unit) ซึ่งเป็นหัวใจหลักในการทำงานของระบบ รวมทั้งพวกวงจรรวมที่ทำ หน้าที่ในการควบคุมอุปกรณ์ต่างๆ ให้ทำงานตามโปรแกรมที่ถูกโปรแกรมลงไปก็ต้องมีส่วนของซีพียูเป็นหน่วยควบคุมการทำงาน ซึ่งตัวซีพียูก็คือการรวมเอาวงจรดิจิตอลที่ออกแบบให้ทำงานตามคำสั่งต่างๆ รวมเข้าด้วยกันเป็นตัวซีพียู ดังนั้นทางคณะผู้จัดทำจึงมีความคิดว่าเราหน้าจะออกแบบตัวซีพียูขึ้นใช้เอง รวมทั้งในปัจจุบันนี้การต่อวงจรทางดิจิตอลมีความง่ายขึ้นมาก ซึ้งโครงงานนี้คณะผู้จัดทำได้ทำการออกแบบซีพียู ขนาด 8 บิท โดยใช้ FPGA Flex10k20 เป็นชิปหลักในการออกแบบ

1.2 วัตถุประสงค์และประโยชน์ที่คาดว่าจะได้รับ

1.2.1 สามารถทำการออกแบบให้ซีพียูสามารถทำงานตามชุดคำสั่งต่าง ๆ ที่ทางคณะผู้จัด
ทำได้กำหนดขึ้น

1.2.2 สามารถนำซีพียูที่จัดทำขึ้นไปใช้งานจริง

1.2.3 เป็นแนวทางในการที่จะออกแบบ และ พัฒนาให้มีประสิทธิภาพที่ดีขึ้น

1.3 แผนการดำเนินงาน

1.3.1 ทำการศึกษาสถาปัตยกรรมภายในของซีพียูที่ใช้งานในปัจจุบัน

1.3.2 ศึกษาลักษณะการทำงานของชุดคำสั่งต่าง ๆ ของซีพียูเบอร์อื่นขนาดเดียวกัน

1.3.3 สร้างชุดคำสั่งขึ้นมาเองพร้อมฟังก์ชั่นการทำงาน

1.3.4 ร่างโครงแบบของสถาปัตยกรรมภายในซีพียู

1.3.5 เขียนบล็อกไดอะแกรมแยกเป็นโมดูลต่างๆ เพื่อง่ายต่อการออกแบบและเขียนโปรแกรม

1.3.6 ทำการเขียนแต่ละโมดูลแยกจากกัน พร้อมทั้งซิมมูเลท (simulate) ความถูกต้องของแต่ละโมดูล

1.3.7 รวมแต่ละโมดูลเข้าด้วยกันพร้อมกับทำการทดสอบความถูกต้อง

1.3.8 ทดลองโปรแกรมลง FPGA และทดลองกับบอร์ดทดลองที่จัดทำขึ้

บทที่ 2 VHDL

เป็นภาษาระดับสูงที่ใช้ในการออกแบบระบบ และ วงจรลอจิก VHDL เป็นภาษาที่สามารถออกแบบวงจรได้ในระดับต่างๆ จะประกอบไปด้วยโครงสร้างต่างๆที่ถูกออกแบบขึ้นมาในระดับที่สูงขึ้นตัวภาษานั้นสามารถที่จะออกแบบระบบโดยไม่คำนึงกระบวนการหรือวิธีการของวงจร เพราะวงจรจะถูกสร้างในรูปแบบของฟังก์ชั่น เราจะพิจารณาเพียงจุดมุ่งหมายของการออกแบบวงจรเท่านั้น แต่ถึงอย่างไรก็ตามการที่จะทำให้เราทราบถึงการทำงานและขอบเขตของจุดมุ่งหมายอย่างแท้จริงแล้ว เราจำเป็นจะต้องเข้าใจและคุ้นเคยกับโครงสร้างของวงจรนั้น และที่สำคัญอีกอย่างก็คือ ตัวภาษาที่ใช้ในการออกแบบวงจรเหล่านั้นด้วย ดังนั้นในบทความนี้จะกล่าวถึงหลักการและโครงสร้างต่างๆในการเขียนภาษา VHDL รวมไปถึงเทคนิคที่สำคัญในการออกแบบ

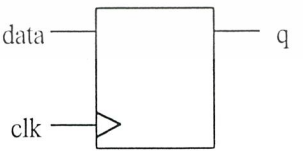
เทคโนโลยีของการออกแบบวงจรลอจิกได้มีการพัฒนาไปอยู่เสมอ จนได้มีการพัฒนาภาษาโปรแกรมที่ใช้ในการออกแบบวงจรลอจิกขึ้นมา VHDL เป็นหนึ่งในนั้นที่ได้มีการพัฒนาขึ้นมา VHDL ย่อมาจาก VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit) มันก็คือภาษาที่ใช้ในการออกแบบวงจรลอจิกเพื่อผลิตเป็นชิพไอซีขึ้นมา เป็นภาษามาตรฐานที่ IEEE ได้ให้การยอมรับ จัดอยู่ในกลุ่มภาษาระดับสูง รูปแบบคล้ายกับภาษาปาสคาล ข้อดีคือเป็นภาษาที่สามารถปรับเปลี่ยนโครงสร้างให้เข้ากับระบบฮาร์ดแวร์ต่างๆได้เป็นอย่างดีและอ่านเข้าใจได้ง่าย

2.1 องค์ประกอบที่สำคัญของ VHDL

VHDL มีองค์ประกอบที่สำคัญอยู่ 2 ส่วนคือ Entity และ Architecture

2.1.1 Entity เป็นเสมือน Block ที่เราสร้างขึ้นมาเพื่อจะบอกถึงจุดเชื่อมต่อภายนอก (I/O) ว่ามีอะไรบ้าง โดยไม่ต้องมีการอธิบายโครงสร้างภายใน

2.1.2 Architecture ใช้ในการอธิบายโครงสร้างภายในของ Entity



รูปที่ 2.1 แสดง D Filp-Flop

```
Entity dff is
port (data, clk : in std_logic;
q : out std_logic);
```

```
end dff;
architecture behav of dff is
begin
process (clk) begin
        if (clk'event and clk = '1') then
        q <= data;
        end if;
end process;
end behav;
```

โปรแกรมนี้เป็นตัวอย่างของการเขียนโปรแกรมเพื่ออธิบายคุณสมบัติของ D F/F (รูปที่ 1) Entity มีการประกาศอินพุทอยู่ 2 ตัวคือ data กับ clk เอาท์พุทคือ q ส่วน Architecture มีการบอกการทำงานของ D F/F ว่าถ้าสัญญาณ clk มีการเปลี่ยนแปลงจาก 0 เป็น 1 ให้มีการส่งค่า data ไป q ซึ่งตัวโครงสร้างไม่อาจจะแสดงหมดในรายงานฉบับนี้ได้ หากผู้สนใจท่านใดมีความสนใจเป็นพิเศษก็สามารถศึกษาเพิ่มเติมจากหนังสืออ้างอิงที่ให้มาได้

2.2 รูปแบบของภาษา VHDL

รูปแบบการเขียน Architecture ของ VHDL VHDL มีรูปแบบการเขียนอยู่ 3 รูปแบบ

- Structural Model การอธิบายวงจรโดยใช้โมดูลต่างๆมาเชื่อมต่อกันให้เห็นโครงสร้างภายใน

-Behavioral Model การอธิบายการทำงานของวงจรในระดับลอจิกเกท

-Sequential Model การอธิบายขั้นตอนการทำงานโดยใช้ Sequential Statement ทำงานทีละขั้นตอน

ต่อไปนี้จะเป็นตัวอย่างการเขียนโปรแกรมเพื่ออธิบายวงจร RS F/F โดยใช้ Architecture ที่แตกต่างกัน

Structural Model

```
u1: nand2 port map (set, q, qb);
u2: nand2 port map (reset, qb, q);
```

Behavioral Model

```
qb <= not (q and set) after 2 ns;
q  <= not (qb and reset) after 2 ns;
```

Sequential Model

```
process (set, reset)
  begin
  if set = '0' and reset = '1' then
  q <= '0' after 2 ns; qb <= '1' after 4 ns;
  if . . .
  . . .
  end process;
```

สำหรับรูปแบบการเขียนที่เราจะเลือกใช้นั้น ขึ้นอยู่กับความเหมาะสมของฮาร์ดแวร์ เราสามารถที่จะใช้รูป แบบต่างๆมาร่วมกันได้อยู่ใน Architecture ตัวเดียวกันก็ขึ้นอยู่กับความเหมาะสมเช่นกัน จะเห็นว่าในแต่ละแบบก็มีจุดเด่นของตัวมันเอง แต่ทั้ง 3 แบบก็ให้ผลลัพธ์เหมือนกัน

Sequential มีการอธิบาย Architecture เป็นลำดับขั้นตอนที่ชัดเจน เข้าใจง่าย โดยปกติแล้วโปรแกรมที่อยู่ภายใน Architecture นั้นจะมีการ process พร้อมๆกันทุกบรรทัด (เหมือนวงจรจริงที่มีการเชื่อมต่อสัญญาณกันทั้งวงจร เมื่อมีสัญญาณ input ตัวใดเปลี่ยน output ก็จะมีการเปลี่ยนตามทันที จะมี delay ก็เพียงเล็กน้อย) แต่แบบ Sequential สามารถจะมองเป็นขั้นตอนได้โดยใส่ Process Statement คร่อมเข้าไปในโปรแกรม ส่วน Structural ก็มีการดึง Component NAND มาจะมองเห็นการเชื่อมต่อที่ชัดเจน ส่วน Behavioral นั้นมีการอธิบายคุณสมบัติระดับลอจิกเกท

2.3 รูปแบบของบางคำสั่งที่ใช้อธิบายพฤติกรรมของอุปกรณ์(Behavioral Modeling)

ในหัวข้อที่แล้วได้กล่าวถึง Structural Model ไปบ้างแล้วในหัวข้อนี้เป็น Behavioral Model ซึ่งจะกล่าวถึง Statement และฟังก์ชั่นต่างๆที่นิยมใช้กับ Behavioral Model

2.3.1 การใช้ Selected signal assignment และ Conditional signal assignment statement

Statement ที่นิยมใช้อีก 2 แบบสำหรับ Behavioral Model ก็คือ Conditional และ Selected signal assignment โปรแกรมต่อไปนี้จะแสดงตัวอย่างของการใช้ 2 Statement ดังกล่าว

```
with sel select
  q <= i0 after 10 ns when 0,
    i1 after 10 ns when 1,
    i2 after 10 ns when 2,
    i3 after 10 ns when 3,
```

```
sel <= 0 when a = '0' and b = '0' else
  1 when a = '1' and b = '0' else
  2 when a = '0' and b = '1' else
  3 when a = '1' and b = '1' else
  4;
```

'x' after 10 ns when others;

โปรแกรมทางซ้ายคือ Selected signal assignment โดย q จะเลือกส่งค่า i0, i1, i2, i3 ออกไป ขึ้นอยู่กับค่าของ sel (sel เป็น integer) ส่วนโปรแกรมทางขวาก็คือ Condition signal assignment โดยตัวแปร sel จะมีค่าเป็น 0, 1, 2, 3 หรือ 4 ขึ้นอยู่กับเงื่อนไขว่าเงื่อนไขใดถูกต้อง

2.3.2 การใช้ delay

การใช้ delay ใน VHDL นั้นเป็นการทำงานเลียนแบบการทำงานจริงๆของวงจรที่จะมี delay ในการทำงานอยู่เช่นกัน

a <= b; (ส่งค่า b ไป a ทันทีโดยไม่มี delay time)

a <= b after 10 ns; (ส่งค่า b ไป a หลังจากที่ผ่านไปแล้ว 10 ns)

VHDL จะมี delay อยู่ 2 แบบ

1 Inertial delay ก่อนจะส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณทางขวาจะต้องคงค่านั้นได้นานเท่ากับค่า delay ดังนั้น delay แบบนี้สัญญาณของตัวส่งกับตัวรับไม่จำเป็นจะต้องเหมือนกัน ดังตัวอย่างการใช้งาน

a <= b after 20 ns; (a จะได้รับ ค่าจาก b เมื่อค่าของ b ไม่เกิดการเปลี่ยนแปลงเป็นเวลา 20 ns)

2 Transport delay เป็นการส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณของทั้งตัวส่งและตัวรับจะเหมือนกัน แตกต่างกันก็ตรงที่สัญญาณตัวรับจะช้ากว่าเป็นเวลาเท่ากับที่ delay ไว้ ดังตัวอย่างการใช้งาน

a <= transport b after 20 ns; (สัญญาณของ b จะเหมือนกับ a แต่ b จะมีสัญญาณที่ช้ากว่า a อยู่ 20 ns)

2.3.3 การใช้งาน Generics

Generics เป็นเครื่องมือที่ใช้ในการส่งข้อมูลต่างๆให้ Entity เช่น ค่า delay time, ค่าความจุ, ค่าความต้านทาน, ความกว้างของ data-path หรือว่า ความกว้างของ signal เป็นต้น ใช้งานได้โดยเราจะเพิ่มเติมส่วน Generic นี้แทรกเข้าไปใน Entity Block ดังตัวอย่างโปรแกรมดังนี้

```
entity buff is
  generic(delay : time; load : integer);
  port( a : in bit; b : out bit);
end buff;
```

```
architecture buff1 of buff is
begin
  b <= a after (delay * load);
end buff1;
```

จากตัวอย่างโปรแกรมเป็น buffer ที่มีการส่งค่า delay และ load ให้กับตัว buffer โดยจะนำค่าเหล่านั้นไปใช้ใน architecture ดังตัวอย่าง ภายใน Architecture จะมีการส่งค่าจาก a ไปยัง b โดยมี

delay หน่วงไว้โดยค่า delay นั้นเท่ากับ (delay * load) ซึ่งหลังจากที่เราได้เขียน entity buffer ตัวนี้ขึ้นมาแล้วก็สามารถจะนำไปใช้กับ entity ตัวอื่นได้เสมือนการเรียกใช้งาน function โดยเราจะใช้ entity buffer ตัวนี้ใน entity ตัวใดเราก็จะแทรก Component ไว้ใน Architecture ตัวนั้นโดยแทรกไว้ก่อน begin ดังตัวอย่าง

```
Architecture test1 of test is
        component buff
                generic(delay : time; load : integer);
                port( a : in bit; b : out bit);
        end component;
begin
```

เมื่อเราแทรก entity ตัวนั้นลงไปแล้วเราก็สามารถเรียกใช้ได้เสมือนเป็นฟังก์ชั่นตัวหนึ่ง และมีการส่งค่าให้ได้ด้วยดังนี้

u1: buff generic map (13 ns, 2) port map (ina, outb); (เป็นการเรียกใช้ในรูปแบบ Structure)

2.3.4 การใช้งาน Block Statements

เป็น Statement ที่เข้ามาช่วยให้ Architecture ของเรานั้นดูเป็นระบบ และสัดส่วนมากขึ้น อีกทั้งยังทำให้เราสามารถประกาศตัวแปรแบบ Local ขึ้นมาใช้ได้ ทำให้ตัวแปรมีการแยกแยะใช้เป็นสัดส่วน ตัวแปรที่ประกาศไว้ใน Block ก็สามารถใช้ครอบคลุมได้ภายใน Block เท่านั้น (เหมือนในภาษาปาสคาล)

```
[label] : block [ ( condition )]
        [ประกาศตัวแปร]
begin
        -- statements
end block [label];
```

ในแต่ละ Architecture สามารถที่จะมีได้หลาย block ขึ้นอยู่กับผู้เขียนว่าจะแบ่งเป็นสัดส่วนเช่นใด และในแต่ละ block ก็สามารถมี block ซ้อนอยู่ข้างในได้ด้วย โดย condition ที่อยู่หลัง block นั้นคือคุณสมบัติของ Guarded Blocks ที่เพิ่มเติมเข้ามาเพื่อนำเงื่อนไขดังกล่าวไปใช้ใน statement ดูได้จากตัวอย่าง

```
architecture test1 of test is
begin
```

```
g1 : block (clk = '1')
begin
  q <= guarded d after 5 ns;
end block g1;
end latch_guard;
```

การทำงานของโปรแกรมก็คือ จะเห็นว่าจะมี keyword GUARDED เพิ่มเข้ามาซึ่ง GUARDED ตัวนี้จะให้ค่าเป็นจริงเมื่อ clk = '1' นอกจากนี้จะเป็นเท็จ ดังนั้น statement d ที่ส่งค่าให้ q จะทำงานก็ต่อเมื่อ Guarded เป็นจริง

2.4 รูปแบบของการทำงานแบบลำดับ (Sequential Processing)

สำหรับการออกแบบวงจรที่มีความขนาดใหญ่และซับซ้อนนั้น รูปแบบของ Structure และ Behavioral Model นั้นยังไม่เพียงพอ รูปแบบของ Sequential เป็นรูปแบบที่มีความสำคัญมากในการใช้ เพราะมีรูปแบบอย่างเดียวกันกับภาษาระดับสูงอย่าง C หรือ ปาสคาลที่มีการทำงานแบบ Sequential Statement

2.4.1 Process Statement

การที่จะทำให้ statement ใน VHDL นั้นสามารถทำงานแบบ Sequential ได้นั้นเราจะต้องใช้ statement ที่เรียกว่า Process เข้าไปช่วย ดังตัวอย่างการใช้งาน

```
architecture test1 of test is
begin
 process (a, b)
  variable temp : std_logic;
 begin
 -- sequential statement;
 end process;
  -- statement;
end test1;
```

จากตัวอย่างการใช้งานจะเห็นว่า Process ที่แทรกเข้าไปใน Architecture นั้นจะทำให้ในส่วน statement ที่ถูกคั่นด้วย begin และ end process นั้นจะมีการทำงานเป็น step ที่ละบรรทัดทันที ซึ่งก็คือรูปแบบของ Sequential Processing ส่วน statement ที่อยู่ใน architecture เดียวกันแต่อยู่นอก block process ก็ยังคงมีการทำงานแบบปกติก็คือ process พร้อมกันทุกบรรทัด นอกจากนี้จากโปรแกรมจะว่าข้างหลัง process ยังมี (a, b) ใส่ไว้ ในวงเล็บนั้นก็คือสัญญาณที่ใช้ใน architecture ตัว architecture จะเข้าไปทำใน block process นี้ก็ต่อเมื่อสัญญาณ a หรือ b มีการเปลี่ยนแปลงเท่า

นั้น หรือที่เรียกว่า Event ทำให้ในการ simulate นั้นโปรแกรมไม่ต้องเข้าไปทำงานใน process ตลอดเวลาโดยไม่จำเป็น

2.4.2 คำสั่งแบบลำดับ(Sequential Statements)

นอกจาก process statement แล้วก็ยังมี statement ที่จำเป็นอยู่อีกมาก VHDL มี Sequential Statements ให้ใช้งานอยู่ 5 ตัวดังนี้ if, case, loop, assert, wait ให้เราได้เลือกใช้งานในงานต่างๆ มีรายละเอียดและรูปแบบดังต่อไปนี้

2.4.2.1 รูปแบบ IF statement

```
IF condition THEN
        Sequence Statements
[ELSEIF condition THEN Sequence Statements ]
[ELSE Sequence Statements ]
END IF;
```

if statement ใช้ในการตรวจสอบเงื่อนไขถ้า condition เป็นจริงก็จะทำใน sequential statement ของ then ถ้าไม่ก็จะไปทำใน sequential statement ของ else แทน ยกตัวอย่างการใช้งานง่ายๆ ถ้าเรามีเงื่อนไข ถ้า a = '1' ก็จะส่งค่า c ให้ b นอกจากนี้ก็จะส่งค่า d ให้ b เขียนเป็นโปรแกรมได้ดังนี้

```
if ( a = '1')  then b <= c;  else b <= d;  end if;
```

2.4.2.2 รูปแบบ Case Statements

```
Case expression IS
   WHEN choices [| choices ] => sequence statements
   WHEN choices [| choices ] => sequence statements
   . . .
END CASE;
```

case statement ใช้ในการตรวจตรวจสอบว่า expression (ตัวแปรที่จะนำมาเปรียบเทียบ) นั้นมีค่าเท่ากับตัวเลือกใด โปรแกรมก็จะเข้าไปทำ statement ในตัวเลือกนั้น แตกต่างกับ if statement ตรงที่ case นั้นจะใช้กับกรณีที่เงื่อนไขนั้นเป็นการเปรียบเทียบกับตัวแปรตัวเดียวกัน และมีหลายเงื่อนไข

2.4.2.3 รูปแบบของ LOOP Statements

VHDL มี loop ให้ใช้งานอยู่ 2 แบบ (ที่คุ้นเคยกันดีในภาษา C หรือ ปาสคาล) คือ while และ for การใช้ loop นั้นมีประโยชน์มากสำหรับข้อมูลที่เรามีหลายชุดเราไม่จำเป็นต้องเขียนโปรแกรมซ้ำๆกัน มีรูปแบบของการใช้งานดังนี้

WHILE condition LOOP
Sequential Statements;
END LOOP

FOR identifier IN start_value TO finish_value LOOP
Sequential Statements;
END LOOP;

ใน loop statement ยังมี statement อีก 2 ตัวที่ช่วยให้การทำงานของ loop นั้นมีความยืดหยุ่นขึ้นคือ next และ exit statement เมื่อเราเพิ่ม NEXT; เข้าไปใน loop เมื่อโปรแกรมได้ทำงานมาถึง statement ที่มี NEXT อยู่ โปรแกรมก็จะกระโดดกลับไปยังส่วนหัวของ loop ทันทีเพื่อทำงานในstep ถัดมาต่อ สำหรับ exit statement เมื่อเราเพิ่ม EXIT; เข้าไปเมื่อโปรแกรมทำงานมาเจอ EXIT แล้วก็จะทำให้หลุดออกจาก LOOP ทันทีไปทำงานนอก loop ต่อไป

2.4.2.4 รูปแบบของ ASSERT Statement

assert statement ใช้ในการแสดงข้อความออกจอภาพในขณะที่มีการ simulate เพื่อที่จะให้เราได้ใช้สำหรับตรวจสอบสภาวะต่างๆในการทำงานของโปรแกรมว่าถูกต้องหรือไม่อย่างไร โดยสิ่งที่แสดงออกไปนั้นจะมีอยู่ 2 ส่วนก็คือ report และ severity report ก็คือข้อความที่เราต้องการแสดงออกมา ส่วน severity ก็คือคำนำหน้าเพื่อใช้บอกสถานะของข้อความเหล่านั้น เพื่อบอกถึงความสำคัญของข้อความ มีคำนำหน้าให้ใช้ได้อยู่ 4 คำ คือ NOTE, WARNING, ERROR, FAILURE มีรูปแบบการใช้งานดังนี้

ASSERT condition
[REPORT " ข้อความที่ต้องการแสดงออก window "]
[SEVERITY คำนำหน้า]

ตัวอย่างการใช้งานเช่นถ้าเราต้องการแสดงข้อความแจ้ง note ว่า " signal a = '0' "เมื่อสัญญาณ a = '0' ทำได้ดังนี้

assert (a = '0')
report " signal a= '0' "
severity note;

2.4.3 คำสั่งที่ใช้หน่วงเวลา(Wait Statements)

Wait statement คือ การหน่วงเวลาใน sequential statement ที่จะทำให้โปรแกรมนั้นหยุดทำงานอยู่ใน statement นั้นเป็นเวลาตามที่กำหนดไว้ หรือตามเงื่อนไขที่ตั้งไว้ มีรูปแบบการหน่วงให้ใช้อยู่ 4 แบบคือ wait on signal, wait until condition, wait for time_expression, และ multiple wait condition Statement เหล่านี้เป็น statement ที่มีประโยชน์มากอีก statement หนึ่งมีการทำงานของแต่ละแบบดังนี้

2.4.3.1 Wait on signal

เมื่อโปรแกรมทำงานมาถึง statement นี้จะเป็นการหน่วงไว้กับที่ไปจนกว่าจะมีการเปลี่ยนแปลงของ signal ที่กำหนดให้ เช่น Wait on a, b; คือจะรออยู่กับที่จนกว่าค่าของ a หรือ b จะมีการเปลี่ยนแปลง

2.4.3.2 Wait until condition

เมื่อโปรแกรมทำงานมาถึง statement นี้จะเป็นการหน่วงไว้กับที่ไปจนกว่าเงื่อนไขนั้นๆจะเป็นจริงก็จะหลุดจากการหน่วง เช่น Wait until ((a = '1') and (b = '1')); คือจะหน่วงไปเรื่อยๆจนกว่า a และ b จะมีค่า '1' ก็จะหลุดจากการหน่วง

2.4.3.3 Wait for time_expression

เป็นการหน่วงตามเวลาที่เรากำหนดไว้ เช่น Wait for 10 ns; คือเป็นการหน่วงไว้เป็นเวลา 10 ns;

2.4.3.4 Multiple Wait condition

เป็นการใช้การหน่วงทั้ง 3 แบบมาร่วมกันเช่นดังตัวอย่าง

Wait on a, b until c ='1' for 2 usec;

คือจะหน่วงไว้จนกว่า a หรือ b จะมีการเปลี่ยนแปลงค่า และ c จะต้องเป็น 1 ถ้าไม่มีเงื่อนไขแบบนี้เกิดขึ้นก็จะหน่วงไว้นานสุดที่ 2 usec โดยการใช้ Multiple Wait นี้มีเงื่อนไขการใช้อยู่ด้วยคือสามารถจะนำมา wait แบบต่างๆมาใช้ร่วมกันได้ 4 แบบ โดยเวลาใช้ก็ต้องมีเรียงลำดับด้วยดังนี้ 1) on until for 2)on until 3)on for 4)until for

2.5 ชนิดและชั้นของข้อมูล(Types and class of Object)

2.5.1 ชั้นของข้อมูล (Class of Object)

VHDL ก็มีคุณสมบัติต่างๆเช่นเดียวกับภาษาระดับสูง มีการแบ่งชนิดของตัวแปรให้ใช้ในภาษาอยู่ด้วย 3 ชนิดด้วยกันคือ

- Signal ใช้เป็นสัญญาณเพื่อแสดงการเชื่อมต่อกันระหว่าง Component ต่างๆ
- Variable ใช้เป็น Temp เก็บข้อมูลชั่วคราวภายใน entity โดยใช้ในการ process
- Constant ชื่อที่มีการกำหนดค่าคงที่ให้ ไว้ใช้ใน program

2.5.1.1 Signal มีรูปแบบการใช้งานดังนี้

SIGNAL signal_name : signal_type [:= initial_value];

การใช้งาน Signal นอกจากจะเป็นการประกาศชนิดของสัญญาณแล้ว ยังสามารถจะกำหนดค่าเริ่มต้นให้กับสัญญาณนั้นได้ด้วยดังตัวอย่างเป็นการประกาศสัญญาณ vcc ให้มีชนิดเป็น std_logic และมีค่าเริ่มต้นเป็น 1

SIGNAL vcc : std_logic := '1';

(* std_logic คือ ชนิดของตัวแปรอีกชนิดหนึ่งมีด้วยกันทั้งหมด 9 ค่าคือ U, X, 0, 1, Z, W, L, H, -)

2.5.1.2 Variables มีรูปแบบการใช้งานดังนี้

VARIABLE variable_name [,variable_name] : variable_type [:= value];

การใช้งาน Variables ก็มีลักษณะคล้ายกับ signal คือสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรได้ด้วย ดังตัวอย่าง กำหนดให้ delay เป็น time มีค่าเริ่มต้นที่ 2 ns

VARIABLE delay : time := 2 ns;

2.5.1.3 Constants มีรูปแบบการใช้งานดังนี้

CONSTANT constant_name [,constant_name] : type_name [:= value];

การใช้งาน Constant คือจะต้องมีการกำหนดค่าเริ่มต้นให้กับตัวแปรเพื่อเป็นค่าคงที่นำไปใช้ในโปรแกรมได้ ดังตัวอย่างเป็นการกำหนดตัวแปร pi มีชนิดเป็น real มีค่าคงที่เท่ากับ 3.1414

CONSTANT pi : real := 3.1414;

2.5.2 ชนิดของข้อมูล (Data types)

ชนิดของตัวแปรที่เราใช้ประกาศใน Signal, Variable, หรือ Constant แบ่งออกมาได้เป็น 4 ประเภทใหญ่ๆ คือ Scalar, Composite, Access และ File Types

Scalar Types นั้นเป็นตัวแปรแบบพื้นฐานที่ใช้กันบ่อยเช่น integer, real เป็นต้น Composite เป็นชนิดที่ใช้สร้าง array และ record ส่วน Access types เป็นตัวแปรรูปแบบของ Pointer เหมือนกับภาษาโปรแกรมทั่วไป File เป็นตัวแปรที่เกี่ยวกับไฟล์

2.5.2.1 Scalar Types

Scalar Types แบ่งย่อยออกเป็น 4 ชนิดคือ Integer, Real, Enumerated, และ Physical types การใช้งานตัวแปรเหล่านี้จะต้องใส่ค่าให้ถูกต้องต้องใส่ค่าให้ถูกชนิดด้วยเช่น Integer ห้ามเอาค่า Real ใส่ให้เป็นต้น

Integer อ้างถึงข้อมูลตัวเลขจำนวนเต็มในช่วง -2,147,483,647 ถึง +2,147,483,647

Real อ้างถึงข้อมูลตัวเลขจำนวนจริงที่เป็นทศนิยมในช่วง -1.0E+38 ถึง +1.0E+38

Enumerated เป็นชนิดที่มีประสิทธิภาพมากอีกชนิดหนึ่ง เพราะเป็นชนิดที่มีการกำหนดขอบเขต หรือว่า set ของกลุ่มข้อมูลที่จะอ้างถึงขึ้นมาได้เองโดย Programmer ยกตัวอย่างถ้าเราต้องการสร้าง type ข้อมูลที่อ้างถึงข้อมูลได้ 3 ตัวคือ red, green และ yellow เพื่อใช้กับโปรแกรมไฟจราจรเพื่อทำให้มองโปรแกรมได้ง่าย

```
TYPE t_state IS ( red, green, yellow );
```

Physical เป็นการกำหนดตัวแปรที่ประกอบด้วย 2 ส่วน ส่วนแรกกำหนดขอบเขตของตัวแปร ส่วนที่สองกำหนดหน่วยขึ้นมาใช้งาน ดังตัวอย่างจะเป็นการกำหนดชนิดข้อมูลที่ชื่อว่า current มีขอบเขตที่ 0 ถึง 1000000000 และมีหน่วยให้ใช้งานอยู่ 3 หน่วยคือ na, ua และ ma ตัวแปรแบบนี้เหมาะที่จะใช้กับข้อมูลที่มีความละเอียดสูง มีตัวเลขอยู่หลายหลัก เมื่อมาใช้หน่วยช่วยจะทำให้อ้างถึงข้อมูลเหล่านั้นได้ง่าย และ สั้นกว่า

```
Type current is range 0 to 1000000000
  units
        na;
        ua = 1000 na;
        ma = 1000 ua;
  end units;
```

2.5.2.2 Composite Types

เป็นชนิดที่มีการเก็บข้อมูลที่มีลักษณะเป็นกลุ่มจึงแยกย่อยได้เป็น 2 ชนิดคือ array และ record สำหรับข้อมูลชนิด array แล้วช่วยได้มากในการทำ ROM หรือ RAM

2.5.2.2.1 Array รูปแบบของ array มิติเดียว

TYPE name IS Array (start TO stop) OF ชนิดของตัวแปร ตัวอย่างการประกาศ Array มิติเดียว data_bus เป็น type ที่มีขนาด 4 bit มีชนิด signal เป็น bit

```
TYPE data_bus IS Array ( 0 TO 3 ) OF bit;
```

ตัวอย่างการประกาศตัวแปรให้ x มีชนิดเป็น data_bus

```
VARIABLE x : data_bus;
```

ตัวอย่างการเข้าถึง Array แบบทีละตำแหน่ง `x(1) := '1';`

ตัวอย่างการเข้าถึง Array แบบเป็นกลุ่ม x := (‘1’, ‘0’, ‘1’, ‘0’);

Multidimensional Arrays เป็นการใช้ array หลายมิติ Array 2 มิตินิยมใช้ในการออกแบบ rom และ ram

ตัวอย่างการประกาศ Array หลายมิติ

```
TYPE mem_data IS Array (0 TO 32, 0 TO 32 ) OF std_logic;
```

ตัวอย่างการเข้าถึง Array หลายมิติ X(1,1) := ‘1’;

2.5.2.2.2 Record

Record คือการจะมองกลุ่มของข้อมูลชนิดต่างๆ เข้าด้วยกันเป็น Object เดียว ดังนั้น Record จะประกอบไปด้วยข้อมูลชนิดต่างๆอยู่ เราสามารถที่จะใช้ Record ซ้อน Record ได้เหมือนกับ Programming Language ทั่วๆไป

ตัวอย่างการประกาศ

```
type   test_record  is
        record
            d : integer;
            o : real;
        end record;
        variable   test : test_record;
```

เราสามารถอ้างถึงข้อมูลได้โดยใส่จุดไว้ข้างหลังตัวแปร และตามด้วยชื่อฟิลด์ใน record เช่น

```
test.d := 5;
test.o := 10.5;
```

2.5.2.3 Access Types

Access มีรูปแบบเหมือนกับ pointer ในภาษาปาสคาลสามารถนำ Pointer นั้นไป Link เชื่อมต่อกันเป็น Link List ได้ด้วย มี Function ที่ใช้กับ Access Types นี้อยู่ 2 ตัวคือ 1 NEW ใช้ในการ Allocate ตำแหน่งใน Memory 2 DEALLOCATE ใช้ในการยกเลิกการใช้งานของ Pointer

ตัวอย่างการใช้

```
TYPE  fifo  IS ARRAY (0 TO 3) OF std_logic;
TYPE fifo_access IS ACCESS fifo;
VARIABLE  fifo_ptr : fifo_access := NULL;
fifo_ptr := new fifo;
fifo_ptr.ALL := (‘0’,’0’,’1’,’1’);
```

2.5.2.4 File Types

เป็นข้อมูลที่ประกาศขึ้นเพื่อใช้ในการติดต่อกับไฟล์

รูปแบบการประกาศ

TYPE file_ name IS FILE OF file_ type;

file มี Function ให้ใช้งานอยู่ 3 Function

READ (file,data) (file คือชื่อ fileที่จะอ่านข้อมูล ; data คือตัวแปรที่จะรับค่าในไฟล์กลับ)

WRITE (file,data) (file คือชื่อ fileที่จะอ่านข้อมูล ; data คือตัวแปรที่เก็บค่าที่จะเขียนลงไปในไฟล์)

ENDFILE (file) (file คือชื่อ fileที่ปิด)

2.5.2.5 Subtypes

เป็นการนำชนิดของข้อมูลที่มีอยู่ดึงมาใช้เพียงบางส่วน มีข้อดีคือ ไม่ต้องกำหนด type ขึ้นมาใหม่, ใช้ขอบเขตของข้อมูลเท่าที่จำเป็น, เข้าใจขอบเขตการใช้งานได้ง่าย เช่นชนิดจำนวนเต็มบวก natural จาก type integer

SUBTYPE natural IS integer RANGE 0 to +2,147,483,647;

2.6 เทคนิคการใช้งาน VHDL

2.6.1 การใช้งาน signal ชนิด std_logic

Std_logic เป็นสัญญาณอีกชนิดหนึ่งที่นิยมใช้ซึ่งมีอยู่ด้วยกันทั้งหมด 9 ค่าคือ (U, X, 0, 1, Z, W, L, H, -) ดังนั้นในการใช้งานเราต้องคำนึงอยู่เสมอว่าไม่ได้มีเพียงค่า 1 กับ 0 ดังนั้นในโปรแกรมของเรานั้นจะต้องมีการใช้ other อยู่เสมอ (other เป็น keyword ตัวหนึ่งที่จะให้ค่าที่เหลือที่เราไม่ได้อ้างถึง) ดังตัวอย่าง กำหนดให้ s เป็น array ขนาด 2 bit ชนิด std_logic

```
case  s  is
    when "00" => muxout <= c;
    when "01" => muxout <= d;
    when "10" => muxout <= e;
    when others => muxout <= f;
end case;
```

จากโปรแกรมถ้าค่า s ไม่ได้เท่ากับ "00" หรือ "01" หรือ "10" แล้ว โปรแกรมจะส่งค่า f ให้ muxout ทันที

นอกจากการนำ others มาช่วยแล้ว เรายังสามารถใช้ don't care เข้ามาช่วยได้อีกด้วยในกรณีที่เราสนใจข้อมูลเฉพาะบางบิท เช่น ถ้าเรามีข้อมูลอยู่ 6 บิท แต่ต้องการตรวจสอบเฉพาะ 2 บิท หลังว่าเป็น 1 ทั้งคู่หรือไม่ ก็จะต้องนำข้อมูลนั้นไปเปรียบเทียบกับ "----11" เครื่องหมายลบแทน don't care การใช้งานอย่างนี้จะมีประโยชน์มากในการเปรียบเทียบเพราะเราไม่จำเป็นต้องแยกออกมาเปรียบเทียบกันทีละบิท

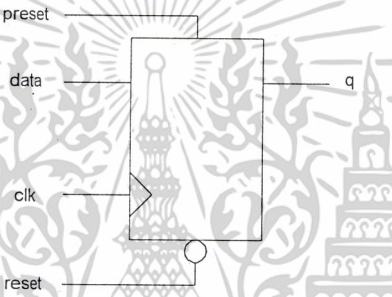
2.6.2 การใช้งาน VHDL กับ Flip-Flops

Flip-Flop จะทำงานได้นั้นจะต้องใช้ clk เข้ามาช่วยซึ่งการทำงานของ flip-flop นั้นอาศัยการทำงานของขอบขาขึ้น หรือ ขอบขาลงของ clk สามารถประยุกต์ใช้ได้ดังนี้

(clk'event and clk = '1') แทนขอบขาขึ้นของ clk

(clk'event and clk = '0') แทนขอบขาลงของ clk

ยกตัวอย่างการออกแบบ D Flip-Flop ที่มี preset และ reset



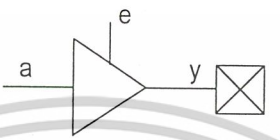
รูปที่ 2.2 แสดง D Flip-Flop มี preset และ reset

```
entity dff_async is     ( entity block )
port (data, clk, reset, preset : in std_logic;
      q : out std_logic);
end dff_async;

architecture behav of dff_async is
begin
process (clk, reset, preset ) begin
      if (reset = '0') begin       ( reset ค่า q เมื่อ reset เป็น '0' )
       q <= '0';
      elseif (preset = '1') then          ( set ค่า q เมื่อ preset เป็น '1' )
       q <= '1';
elseif ( clk'event and clk ='1') then   ( ส่ง data เมื่อมี clk เข้ามา )
       q <= data;
```

```
        end if;
    end process;
    end behav;
```

2.7 การออกแบบ buffer แบบต่างๆ

2.7.1 Tri-State



รูปที่ 2.3. แสดง Tri-State

การออกแบบ buffer ที่มี 3 สถานะโดยมี enable เป็นตัวควบคุมสามารถเขียนเป็น entity ได้ดังนี้

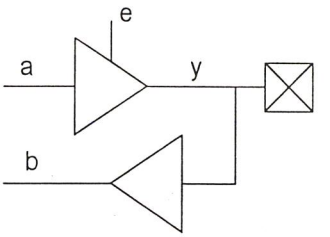
```
entity tristate is
port (e, a : in std_logic;
      y : out std_logic);
end tristate;
```

Architecture นั้นสามารถเขียนได้หลายแบบดังตัวอย่างทั้ง 2 แบบต่อไปนี้

```
architecture tri of tristate is
begin
  process (e, a)
    begin
      if e = '1' then y <= a; else y<= 'z'; end if;
    end process;
end tri;
```

```
architecture tri of tristate is
begin
  y <= a when ( e = '1' ) else 'z'
end tri;
```

2.7.2 Bi-Directional Buffer



รูปที่2.4 แสดง Bi-Directional Buffer

เป็น buffer 2 ทิศทางสามารถเขียนเป็นโปรแกรมส่วน Architecture ได้ดังนี้

```
architecture bi of bidir is
begin
  process (e, a)
   begin
        case e is
         when '1' => y <= a;
         when '0' => y <= 'z';
         when other => y <= 'x';
        end case;
  end process;
  b <= y;
end bi;
```

2.8 บทประยุกต์

2.8.1 ข้อดีของ VHDL

1. รูปแบบของภาษาที่เข้าใจได้ง่าย

2.มีโครงสร้างของภาษาที่สามารถปรับเปลี่ยนให้เข้ากับ hardware ได้ง่าย

3.มี statement ให้ใช้งานอยู่หลายตัว

4.สามารถออกแบบในรูปแบบ top-down design ได้

5. มี library ต่างๆให้เลือกใช้งานได้ทำให้ออกแบบได้ง่ายขึ้น เป็นประโยชน์ต่อ programmer

2.8.2 VHDL ช่วยในการออกแบบได้อย่างไรบ้าง

1.การออกแบบวงจรสามารถทำได้โดยง่าย

2. เมื่อมีการแก้ไข หรือ เปลี่ยนแปลงวงจรสามารถทำได้โดยง่าย เพียงโปรแกรมวงจรใหม่ลงไป ไม่ต้องมีการเปลี่ยนแปลงตัว hardware

3. การทดสอบไม่จำเป็นต้องทดลองกับวงจรจริง สามารถใช้การ simulate ทดลองดูผลลัพธ์ของวงจรที่เราออกแบบได้ว่าถูกต้องหรือไม่

2.9 บทสรุป

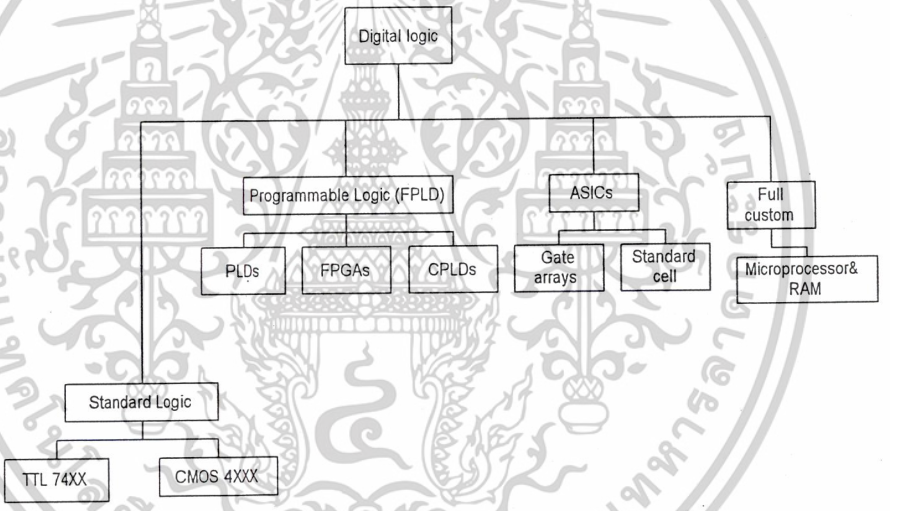
การศึกษา VHDL สามารถจะนำมาประยุกต์ใช้งานได้กับวงจรต่างๆ ได้ตั้งแต่การออกแบบ F/F, การออกแบบ Latch, การออกแบบวงจร Multiplex, การออกแบบ Counter และ การออกแบบ buffer ซึ่งเป็นการออกแบบวงจรในขั้นพื้นฐาน และยังสามารถนำมาดัดแปลงให้มีการทำงานแบบ Mealy machine หรือ Moore machine ได้ ซึ่งถ้าเป็นการออกแบบวงจรจริงๆ จะต้องมีขั้นตอนที่ยุ่งยากซับซ้อน แต่ถ้าใช้ภาษา VHDL ก็สามารถทำได้โดยง่าย

ในระดับที่สูงขึ้นเราก็จะนำพื้นฐานต่างๆเหล่านี้มาประยุกต์ พัฒนาเป็น RAM หรือ ตัวประมวลผลต่างๆ ได้โดยเดิมที่เราใช้เทคโนโลยีของ CPLD ที่มีการโปรแกรมวงจร block หนึ่งๆต่อชิพ 1 ตัว เราก็จะพัฒนามาใช้เทคโนโลยี FPGA ที่ใน 1 ชิพนั้นสามารถจะโปรแกรมวงจรหลายๆส่วนได้

บทที่ 3

เทคโนโลยีโปรแกรมเมเบิ้ลลอจิก(Programmable logic techology)

บล็อกไดอะแกรมของอุปกรณ์ ต่างๆ ในงานทางด้านดิจิตอลแสดงดังรูปที่ 3.1 วงจรรวมแบบ SSI (Small Scall Integrated Circuit) , MSI (Mediume Scall Integrated) ซึ่งทางผู้ผลิตทำการกำหนดฟังก์ชั่นการทำงานมาแล้ว ในการใช้งานต้องนำมาต่อรวมกันเป็นวงจรตามฟังก์ชั่นที่ต้องการ แต่ก็ได้มีการพัฒนาวงจรรวมที่ผู้ออกแบบสามารกำหนดฟังก์ชั่นการทำงานลงไปในตัวชิปได้เอง เช่น วงจรรวมประเภท แอสซิกส์ (ASIC: Application Specific logic devices) , ซีพีแอลดี (CPLD : Complex Programmable gate array) , เอฟพีจีเอ (FPGA : Field Programmable gate array) ซึ่งวงจรแบบแอสซิกส์ ต้องอาศัยเทคโนโลยีขั้นสูงอยู่ในการกำหนดฟังก์ชั่นการทำงานให้กันตัวอุปรกรณ์ แต่ CPLD และ FPGA ผู้ออกแบบสามารถทำการโปรแกรมฟังก์ชั่นการทำงานได้ด้วยตัวเอง

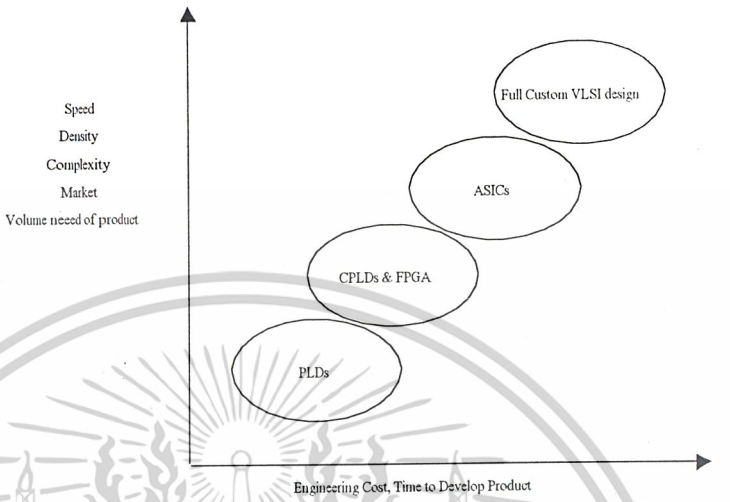


รูปที่ 3.1 แสดงบล็อกไดอะแกรมของอุปกรณ์ทางด้านดิจิตอล

ความแตกต่างของการออกแบบในแต่ละระดับแสดงในรูปที่ 3.2 จะเห็นว่า เทคโนโลยีแบบฟูลคัสทีมวีแอลเอสไอ(Full custom VLSI design) ต้องใช้เวลาเป็นแรมปีในการออกแบบและทำการทดสอบการทำงาน ซึ่งเหมาะกับอุปกรณ์ที่มีความต้องการทางตลาดสูง ซึ่งจะสามารถลดต้นทุนต่อหน่วยลงได้ ตัวอย่างของอุปกรณ์ เช่น หน่วยความจำ (RAM) ไมโครโปรเซสเซอร์

เทคโนโลยีแอสซิกส์ สามารถ แยกเป็นสองระดับคือ เกทอาร์เรย์ (Gate array) และ เซลล์มาตรฐาน (Standard cell) ซึ่ง เกทอาร์เรย์เกิดจากหลายลอจิกเซลล์ มาต่อรวมกัน โดยผู้ผลิตจะทำการเชื่อมต่อลอจิกเซลล์ต่างๆ เข้าด้วยกันตามฟังก์ชั่นที่ผู้ใช้ได้กำหนดมา ส่วนแบบเซลล์มาตฐานผู้ผลิตจะไม่กำหนดโครงสร้าง ภายในชิปตายตัว แต่จะทำหน้าที่จะสร้างชิปให้ผู้ใช้เป็นคนเลือกที่

ใช้ เช่น ALU,RAM ซึ่งเป็นมาตรฐานของผู้ผลิตโดยเฉพาะ อย่างไรก็ตามดทคโนโลยีนี้ ก็ยัง ต้องใช้เวลาและต้นทุนที่สูง รวมทั้งมีความซับซ้อนในการสร้างแต่ต้นทุนต่อหน่วยต่ำเมื่อเทียบกับ FPGA และ CPLD



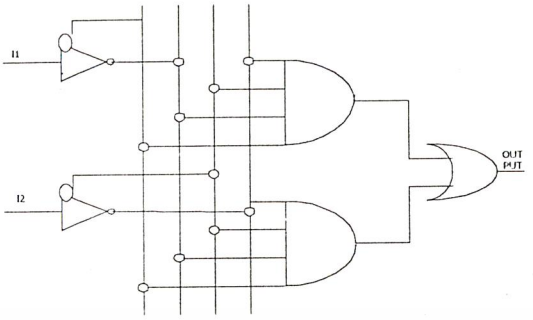
รูปที่ 3.2 แสดงความแตกต่างของเทคโนโลยีต่างๆในการออกแบบ

เทคโนโลยีโปรแกรมเมเบิ้ลลอจิก (Programmable logic) เช่น พีเอแอล(PAL : Programmable array logic) , พีแอลเอ (PLA : Programmable array logic) , พีแอลดี (PLD : Programmable logic device) ซึ่งมีการใช้งานมานานมากแล้ว และสามารถออกแบบในรูปของทีทีแอลหรือซีมอส นอกจากนี้ยังมี FPGA และ CPLD ซึ่งรายละเอียดจะแสดงเป็นข้อดังนี้

3.1 พีแอลดี (PLD : Programmable Logic Device)

ภายในอุปกรณ์พีแอลดีถูกเตรียมเป็นวงจรพื้นฐานทางด้านลอจิก ต่อกันเป็นกลุ่มมีทั้งวงจรคอมบิเนชั่น (Combination) และ ซีเควนเชียล (Sequention) ซึ่งมีส่วนประกอบเป็นวงจรภายในเทคโนโลยีของวงจรที่ใช้สร้าง มีทั้ง ทีทีแอล (TTL) ซีมอส (CMOS) ตามความเหมาะสมของแต่ละระบบอุปกรณ์พีแอลดี ทุกชนิดมีหลักการพื้นฐานของวงจรภายในที่เหมือนกันโดยมีวงจรหลักเป็นวงจรคอมบิเนชั่นที่ให้ผลเป็นผลคูณรวมบวก (sum of product) ประกอบไปด้วยชุดของแอนด์เกตที่ต่อร่วมกับออเกตการโปรแกรมคือ การเลือกว่าจะให้มีการต่ออินพุต ภานในของแอนด์เกตกับสัญญาณอินพุตใดบ้างซึ่งมีทั้งจากภายนอกและสัญญาณป้อนกลับจากเอาต์พุต ภายในเอง การติดต่ออินพุตของออเกตกับเอาต์ ของแอนด์ ตัวต่างๆ วิธีการเลือกหรือการโปรแกรมทางกายภาพ อินพุต ต่างๆ ของอุปกรณ์ ทุกตัวจะถูกต่อผ่านฟิวส์ เข้ากับแหล่ง สัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใดจะตัดฟิวส์ทำให้สามารถโปรแกรมได้ครั้งเดียว อุปกรณ์พีแอลดีบางชนิด ใช้มอสทรานซิสเตอร์ แทน

ฟิวส์ ทำให้สามารถใช้โปรแกรมโดยใช้กระแสไฟฟ้า และสามารถลบข้อมูลและโปรแกรมข้อมูลใหม่



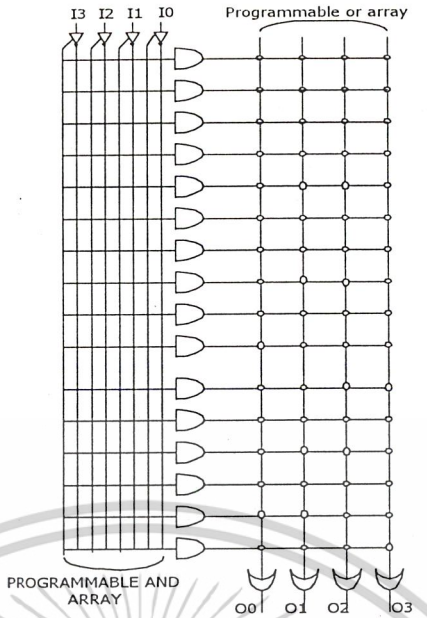
รูปที่ 3.3 แสดงวงจรพื้นฐานส่วนหนึ่งของพีแอลดี ในรูปผลคูณร่วมบวก

3.2 พีเอแอล (PAL : Programmable array logic)

ในช่วงกลางปี ค.ศ. 1970 บริษัทเอ็มเอ็มไอในประเทศสหรัฐอเมริกา ได้พัฒนาอุปกรณ์พีเอแอล เป็นพีเอแอลชนิดใหม่ โดยใช้เทคโนโลยีแบบแอลเอสไอ สามารถโปรแกรมเลือกวงจรภายใน โดยใช้ฟิวส์ที่เชื่อมต่ออยู่ระหว่างสัญญาณอินพุตภายนอกและการป้อนกลับจากภายในกับแอนด์เกตที่ต่อเป็นฟังก์ชั่นผลคูณ (Product) อยู่ในตัววงจรรวม

3.3 พีแอลเอ(PLA: Programmable Logic Array)

อุปกรณ์ที่สามารถโปรแกรมได้แบบพีแอลเอเกิดขึ้นเมื่อปี ค.ศ.1975 โดยบริษัทซิกเนทริส์ (Signetics) สหรัฐหมริกา ซึ่งเป็นบริษัทผู้ผลิตวงจรรวมรายใหญ่ รายหนึ่ง ผลิตและนำเสนออุปกรณ์โยใช้ชื่อว่า FPLA(Field Program logic array) สามารถโปรแกรมการต่อลอจิกทั้งทางด้านแอนด์เกตและออเกตได้ และยังเลือกเอาต์พุตเป็น active high หรือ active low โดยต่อผ่านเอ็กคูลซีบออเกต ให้ทำหน้าที่เป็นนอนอินเวอร์เตอร์หรือเป็น อินเวอร์เตอร์แล้ว แต่ภายในของพีแอลเอ ต่อมาปี ค.ศ. 1979 บริษัทซิกเนทริกส์ ได้สร้างเอฟพีแอลเอใหม่ที่มีรีจิสเตอร์ต่ออยู่ภายในวงจร เพิ่มขึ้นรวมทั้งสามารถเลือกสัญญาณอินพุทที่มาจากการป้อนกลับจากเรจิสเตอร์ได้ด้วย ทำให้สามารถใช้อุปกรณ์พีแอลเอใหม่นี้สร้างวงจร State machine ได้ อุปกรณ์ ใหม่ที่มีเรจิสเตอร์ อยู่ด้วยนี้ถูกเรียกว่า เอฟพีแอลเอส (FPLS : Field Programmable Logic Sequencer) มีทั้งที่เป็นทีทีแอลและซีมอส



รูปที่ 3.4 แสดงวงจรพื้นฐานภายในของพีแอลเอ

3.4 เอฟพีจีเอ จะเน้นที่เบอร์ Flex10K20 ของบริษัท Altera ซึ่งจะใช้ในโครงงานนี้

3.4.1 คุณสมบัติของเอฟพีจีเอ FLEX 10K

3.4.1.1 มีหน่วยความจำภายในที่มีประสิทธิภาพและสามารถสร้างฟังก์ชั่นทางตรรกะ (Logic) ที่พิเศษได้ เนื่องจากมีลักษณะของอาร์เรย์

3.4.1.2 High Densityคือ

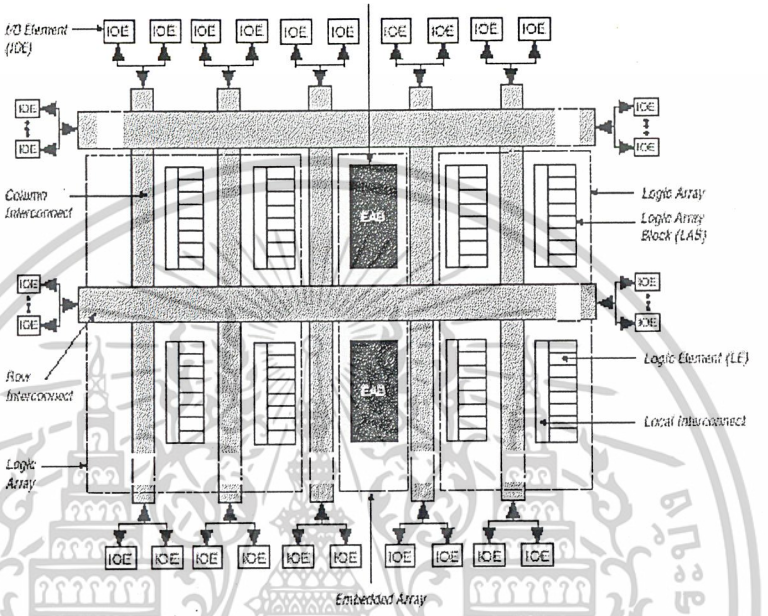
-มีจำนวนเกตภายในจำนวน 10000 เกต สามารถเพิ่มจำนวนRAM จาก 2048bits /EAB เป็น 40960 bits /EAB โดยที่ไม่ทำให้ค่าของ logic ภายในลดลง

3.4.1.3 คุณลักษณะ System-Level คือ

- multivolt I/O interface support
- FLEX 10Kใช้กับแรงดันไฟ 5.0 Volt
- มีการใช้พลังงานน้อย คือ โดยเวลาทำงานกินกระแสน้อยกว่า 5mA
- FLEK 10K เป็นอุปกรณ์ที่รองรับกับการต่อกับอุปกรณ์ภายนอกโดยใช้ PCI บัส เป็นตัวเชื่อมต่อ
- FLEK 10K ได้รองรับมาตรฐาน IEEE 1149.1-1990

3.4.2 ลักษณะโครงสร้างของ FLEX 10K

ภายในตัว FLEX 10K จะประกอบไปด้วยส่วนต่างๆดังนี้คือ ส่วน Embedded Array เป็นส่วนที่ใช้ในหน่วยความจำ , มีส่วนของ logic ที่เป็นพวกฟังชั้นพิเศษ และมีพวก logic ที่เป็นอุปกรณ์พื้นฐานของ logic ทั่วๆไป



รูปที่ 3.5 แสดงบล็อกไดอะแกรม ของ FLEX 10K

ในรูปที่ 3.5 เป็นบล็อกไดอะแกรมของ FLEX 10K ในกลุ่มของ Logic Element(LE) ในลักษณะ Logic Array Block (LAB) โดยLAB จะถูกจัดในลักษณะของแถวในแนวนอน และแนวตั้ง โดยในแต่ละแถวจะประกอบไปด้วยEAB เคียงๆอยู่ ซึ่งLABSและEABS จะมีการต่อภายในในลักษณะของ FastTrack Interconnect ส่วนIOE(INPUT/OUTPUT Element) จะอยู่ที่ปลายของแต่ละแถวและแต่ละคอร์ลัม โดยมีการต่อเชื่อมในลักษณะของ FastTrack Interconnect

FLEX 10K เป็นอุปกรณ์ที่มีการกำหนดให้อินพุท 6 อินพุท มีลักษณะเป็น Flip Flop ก็เลยทำให้เรามั่นใจได้ว่า FLEX 10K นี่จะทำให้สัญญาณที่รับเข้ามาทำงานลักษณะ high speed ,low skew (less than 15 ns) โดยสัญญาณที่รับมาจะถูกแยกไปตาม แชลแนลต่างๆ ซึ่งในแต่ละแชนแนลจะถูกแบ่งออกเป็นลักษณะของสัญญาณที่มีค่าของ delay time และค่าความผิดเพี้ยนต่ำแล้วก็จะถูกเชื่อมต่อไปใช้งานอย่างรวดเร็ว ส่วนอินพุทอีก4ส่วนจะถูกนำไปใช้กับสัญญาณ Four Gobal Signal โดยมีโลจิกภายในเป็นตัวควบคุมการทำงานของ Four Gobal Signal ซึ่งจะคอยกำหนดสัญญาณClock ให้เป็น Clock ในอุดมคติ หรือคอยทำไม่ให้เกิดการ Asynchronous ในการ Clear สัญญาณหรือ อาจจะใช้ในการ Clear Registers ในFLEK ก็ได้

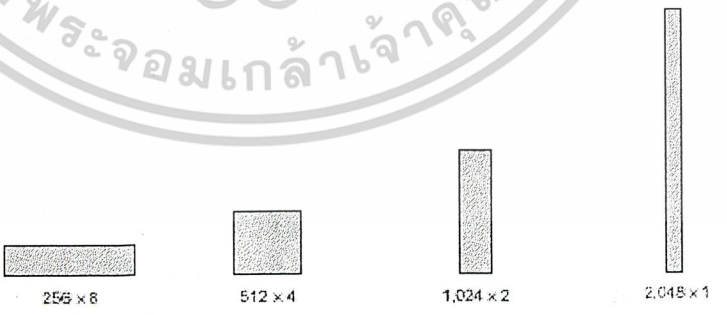
3.4.3 Embedded Array Block (EAB)

ในส่วนของ Embedded Array จะประกอบไปด้วย ส่วนของ EABs (Embedded Array Block)ที่ต่ออนุกรมกันอยู่ ซึ่งเป็นอุปกรณ์ที่ใช้เป็นฟังก์ชั่นของหน่วยความจำ โดย EAB แต่ละตัวจะมีขนาด 2048 บิท ทำให้เราสามารถเลือกใช้ Embedded Array ให้ทำหน้าที่เป็น ROM, RAM, Dual-Port RAM หรือ FIFO (first in first out) ส่วนของ EAB ที่ทำหน้าที่เป็น logic จะประกอบไปด้วยเกทประมาณ 100 ถึง 600 เกท เช่นพวก Multipliers, Microcontrollers , State MachinesและDSP นั้นเอง ในการใช้งาน EABsอาจจะใช้เป็นตัวเดียวๆหรือมีการต่อ EABs หลายๆ ตัวเข้าด้วยกันเพื่อทำให้เป็นฟังก์ชั่นที่มีขนาดใหญ่ขึ้นนั้นเอง

EAB จะเป็นส่วนของบล็อกไดอะแกรมที่แสดงในส่วนของ RAM ซึ่งจะเป็นRegister ของอินพุทและเอาท์พุทพอร์ท โดย EAB จะถูกนำไปใช้งานเกี่ยวกับพวก Multipliers ,Vector Scalars และ Error Correction Circuit เพราะขนาดของ EABนั้นมีขนาดที่ใหญ่และมีการใช้งานที่ง่าย นั้นเองเลยเหมาะที่จะไปใช้งานดังประเภทที่กล่าวมา นอกจากนี้เรายังสามารถนำ EAB ไปประยุกต์ใช้กับ Digital Filtersและ Microcontrollers

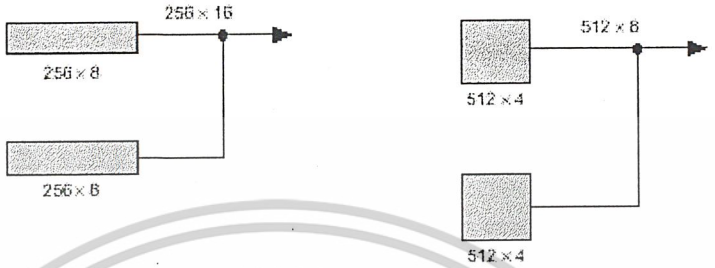
EAB จะถูกนำไปใช่ในลักษณะของ Synchronous RAM มากกว่า Asynchronous RAM เนื่องจากวงจรที่ใช้ EAB ในลักษณะ Asynchronous RAM จะถูกนำไปเป็น RAM write enable (WE) ซึ่งเวลานำไปใช้งานต้องมีข้อมูลและaddress สัญญาณที่จะทำการเขียนและต้องมีเวลาที่บอกรายละเอียดของสัญญาณ WEด้วย ซึ่งจะตรงกันข้ามกับการนำเอา EAB ไปใช้เป็น Synchronous RAM ที่มีการผลิตสัญญาณ WE และ เป็นตัวกำหนดเวลาทั้งหมดของclock

เราสามารถที่จะเลือกใช้ RAM ตามโครงสร้างของ EAB ได้คือจะมีขนาด 256*8,512*4,1024*2 หรือ 2048*1 โดยดูได้จากรูปที่ 3.6



รูปที่ 3.6 แสดงลักษณะการแบ่งขนาดของแรมภายใน FLEX

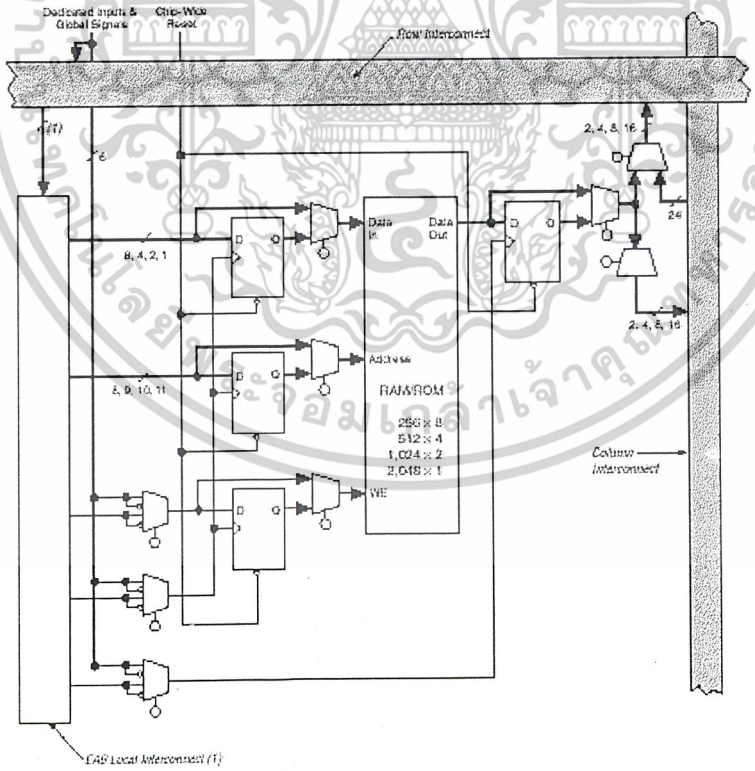
แต่เราก็สามารถสร้าง RAM ให้มี ขนาดใหญ่ขึ้นได้โดย การรวมของ EABs หลายตัวเข้าด้วยกัน เช่น เรานำ RAM ขนาด 256*8 จำนวนสองตัวมารวมกันจะทำให้เราได้ RAM ขนาด 256*16 หรือ อาจจะใช้ RAM ขนาด 512*4 จำนวนสองตัวมาต่อร่วมกันจะได้ RAM ขนาด 512*8 ซึ่งแสดงได้ดังรูปที่3.7



รูปที่3.7 แสดงตัวอย่างการต่อร่วมกันของ EABS

ในกรณีที่ต้องการใช้อุปกรณ์ภายในของ EABs ทั้งหมดมาต่อกันในลักษณะให้เป็น RAM ตัวเดียว ตัวEABจะสามรถสร้าง RAM ให้มีขนาด 2048 บิท โดยที่การทำงานของ RAM นั้นจะไม่มีผลทางด้านเวลาเลย

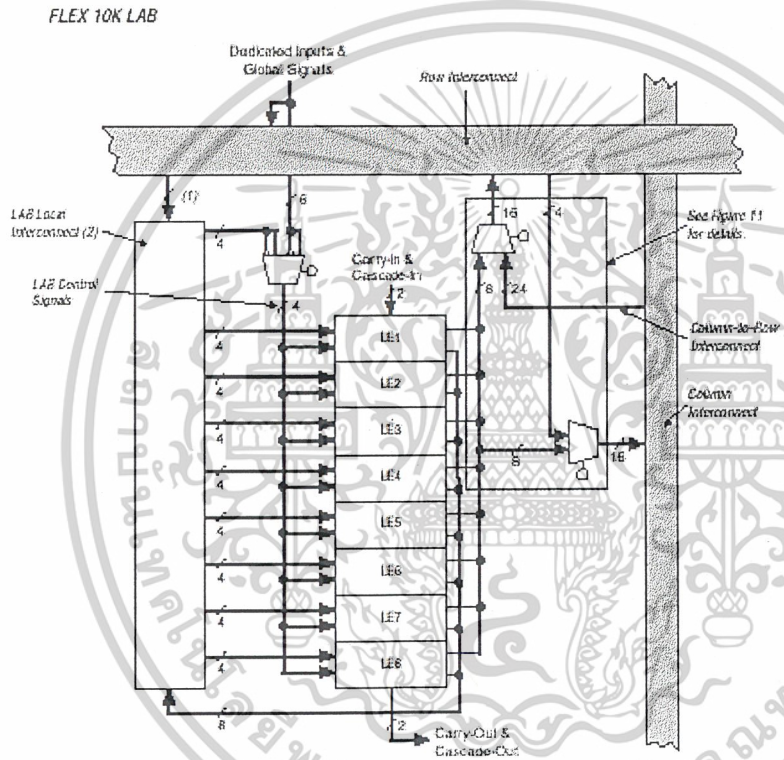
ส่วนต่างๆของ Embedded Array Block (EAB) แสดงได้ดังรูปที่ 3.8



รูปที่3.8 แสดง Embedded Array Block ของ Flex10k

3.4.4 Logic Array Block

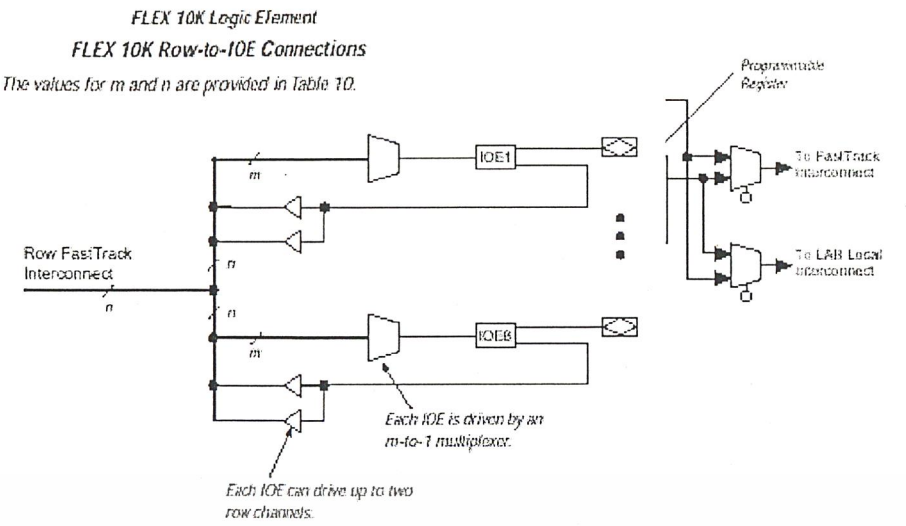
ในส่วนของ Logic Array จะมีการต่อกันเป็นในลักษณะ Logic Array Block(LAB) โดยในแต่ล่ะ Logic Array Block จะประกอบไปด้วย Logic 8 ตัว ที่มีการต่อร่วมภายในไว้แล้ว โดยใน Logic Element ใช้อินพุทอยู่ 4อินพุท ที่ต่ออยู่ในลักษณะ LUT (Look-Up table) จะถูกนำไปใช้ในการโปรแกรม Flip Flop ,ใช้เป็นเส้นทางของการเชื่อมต่อของสัญญาณ และการต่อคาสเคดของฟังก์ชั่น และถ้าเราใช้ Logic Element ทั้ง 8 ตัวที่ต่อกันจะจะถูกนำไปใช้ใน พวกวงจรนับ(Counter) ขนาด 8 บิท,ใช้เป็น Address ของวงจรถอดรหัส หรืออาจจะสร้าง LABsให้เป็นลักษณะของ Logic Block ที่มีขนาดใหญ่ได้ โดยส่วนต่างๆของ Logic Array Block จะแสดงได้รูปที่ 3.9



รูปที่ 3.9 แสดงลักษณะของลอจิกอาร์เรย์ของ Flex10K

3.4.5 Logic Element (LE)

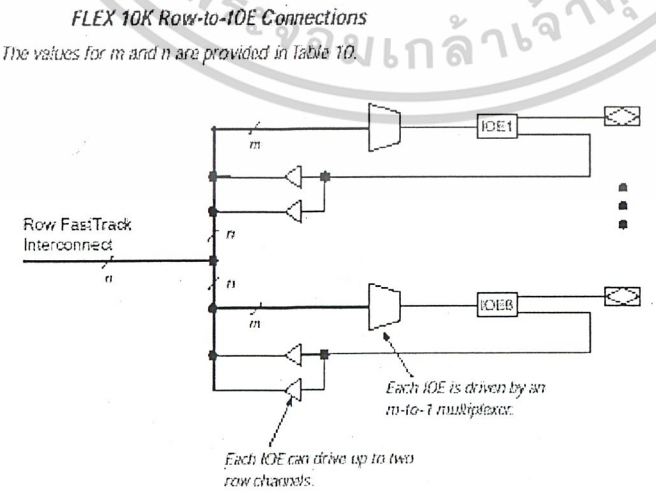
Logic Element เป็นส่วนที่เล็กที่สุดในบรรดา Logic ของ FLEX 10K ก็เพื่อที่จะทำให้ Logic นี้มีประสิทธิภาพในเวลาทำงาน Logic Element ใช้อินพุทอยู่ 4อินพุท ที่ต่ออยู่ในลักษณะ LUT (Look-Up table) ซึ่งเป็นฟังก์ชั่นที่ทำมาให้มีการคำนวณได้เร็วขึ้น โดยเป็นฟังก์ชั่น 4ตัวแปรเท่านั้น โดยแสดงส่วนต่างๆของ Logic Element ได้ดังรูปที่ 3.10



รูปที่ 3.10 แสดงลอจิกอิลิเมนต์(Logic element) ของ Flex10k

3.4.6 I/O Element (IOE)

ในแต่ละขา I/O จะมาจากส่วนของ I/O Element (IOE) โดยที่ IOE จะอยู่ที่ปลายของแต่ละแถวและแต่ละคอร์ลัมของ FastTrack Interconnect โดยในแต่ล่ะส่วนของ IOE จะประกอบไปด้วย วงจรกันชนสองทิศทางของ I/O และ Flip Flop ซึ่งสามารถที่จะนำไปใช้เป็นส่วนของเอาท์พุทหรืออินพุท Register โดยที่การทำงานของมันนั้นClock เป็นตัวกำหนดนั้นเอง โดยทางอินพุทมีการกำหนดให้ Clock ต้องน้อยกว่า 1.6 ns และค่า Hold Time ต้องเท่ากับศูนย์ ส่วนทางด้านเอาท์พุทจะกำหนดให้ Clock น้อยกว่า 5.3 ns โดยปกติ IOEs จะต้องมีคุณสมบัติดังนี้คือ จะต้องรองรับกับJTAG BST ,ต้องควบคุมค่า Slew-Rate ,Tri-State Buffers และOpen-drain Output ลักษณะของ Row –to- IOE Connection จะมีการต่อในลักษณะดังรูปข้างล่างนี้

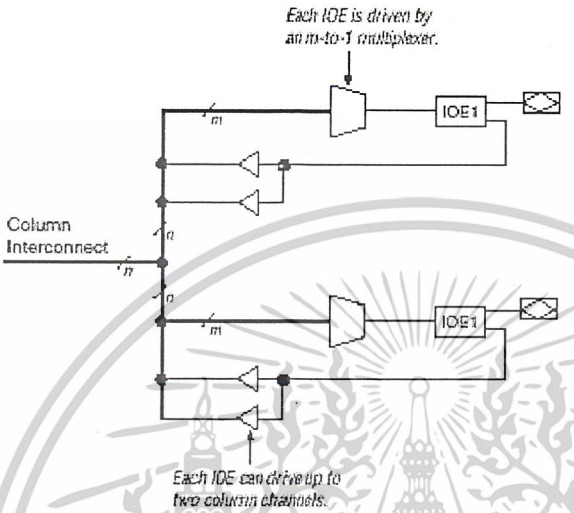


รูปที่ 3.11 แสดง Row –to- IOE Connection ของ FLEX 10K

ลักษณะของ Column -to- IOE Connection จะมีการต่อในลักษณะดังรูปข้างล่างนี้

FLEX 10K Column-to-IOE Connections

The values for m and n are provided in Table 11.



รูปที่ 3.12 แสดง Column –to- IOE Connection ของ FLEX 10K

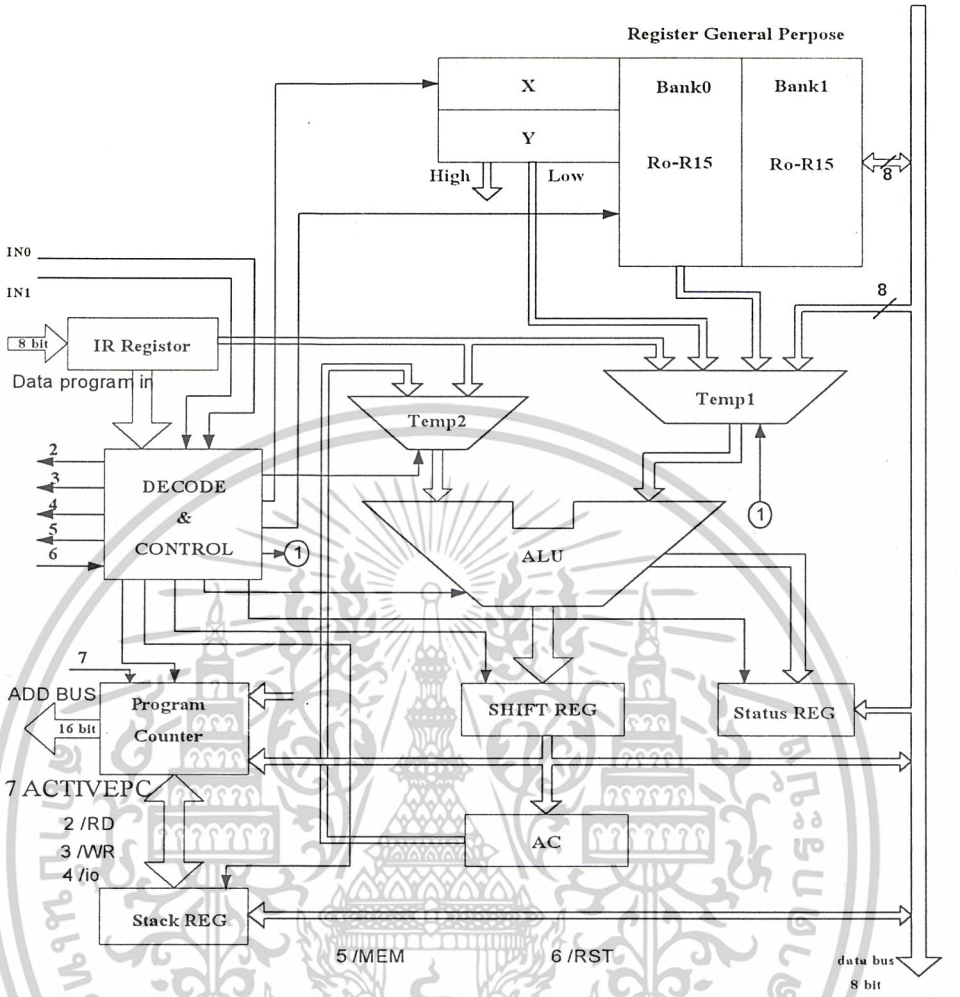
PIN NUMBER	FUNCTION	PIN NUMBER	FUNCTION
1	TCK	73	I/O
2	CONF_DONE	74	Nconfig
3	NCEO	75	VCCTNT
4	TDO	76	MSEL1
5	VCCIO	77	MSELO
6	VDDINT	78	I/O
7	I/0,CLKUSR	79	I/O
8	I/O	80	I/O
9	I/O	81	I/O
10	I/O	82	I/O
11	I/0,RDBUS	83	I/O
12	I/O	84	GNDIO
13	I/O	85	GNDIO
14	I/O,INTDONE	86	I/O
15	GNDIO	87	I/O
16	GNDINT	88	I/O
17	I/O	89	I/O
18	I/O	90	I/O
19	I/O	91	I/O
20	I/O	92	I/O
21	I/O	93	VCCINT
22	I/O	94	I/O
23	I/O	95	I/O
24	VCCIO	96	I/O
25	VCCINT	97	I/O
26	I/O	98	I/O
27	I/O	99	I/O
28	I/O	100	I/O
29	I/O	101	I/O

ตารางที่ 3.1 แสดงตำแหน่งขาของ FPGA

PIN NUMBER	FUNCTION	PIN NUMBER	FUNCTION
31	I/O	103	GNDINT
32	I/O	104	GNDIO
33	I/O	105	TDI
34	TMS	106	NCE
35	NSTATUS	107	DCLK
36	I/O	108	DATA0
37	I/O	109	I/O,DATA
38	I/O	110	I/O,DATA
39	I/O	111	I/O,DATA
40	GNDIO	112	I/O,DATA
41	I/O	113	I/O,DATA
42	I/O	114	I/O,DATA
43	I/O	115	VCCIO
44	I/O	116	I/O,DATA
45,71,61,134	VCCIO	117,144	I/O,I/O(NCS)
46	I/O	118,102	I/O
47	I/O	119,143	I/O,IO(CS)
48,30,60	I/O	120,142	I/O,I/O(NWS)
49,62,63,64,65,66	I/O	121,141	I/O,I/O (NRS)
50,139	GNDIO	122	I/O,DEVLR
51,67,68,69,70,72	I/O	123	VCCINT
52	VCCINT	124	DEV.INPUT
53	VCCINT	125	GOBAL CLK
54	DED.INPUT	126	Dev.Input
55	GL0BAL CLK	127	GNDINT
56	DED.INPUT	128	I/O,DEVOE
57	GNDINT	129	GNDIO
58	GNDINT	130,135,136,137	I/O
59	I/O,138,140	131,132,133,140	I/O

ตารางที่ 3.1 แสดงตำแหน่งขาของ FPGA

บทที่ 4 สถาปัตยกรรมและชุดคำสั่ง



รูปที่ 4.1 แสดงสถาปัตยกรรมของ KC-I

ซึ่งได้ทำการแยกส่วนของบัสข้อมูล ออกจากบัสโปรแกรม (data program) รวมทั้งได้รวมเอาสแต็ครีจีสเตอร์ใช้ภายในตัว KC-I ไม่สามารถอ้างอิงจากหน่วยความจำภายนอกได้ ชุดคำสั่งทั้งหมด 67 ชุดคำสั่งซึ่งประกอบด้วย ชุดคำสั่งของ กลุ่มต่าง ๆ คือ กลุ่มของคณิตศาสตร์และลอจิก, กลุ่มของการหมุนเลื่อนข้อมูล,กลุ่มควบคุมซีพียู,กลุ่มโอนย้ายข้อมูล,กลุ่มของการกระโดดไปทำงานที่ตำแหน่งต่าง ๆ และสามารถทำการขออินเทอร์รัพท์ภายนอกได้ 2 แหล่ง เมื่อเทียบกับซีพียู ที่เป็นลักษณะของฟูลคัสทัม (Full custum) KC-I ยังมีชุดคำสั่งน้อยอยู่ทั้งนี้ขึ้นกับแบบของการเขียนโปรแกรมและความจุของเกตในตัว FPGA

4.1 คุณสมบัติของ CPU KC-I

4.1.1.ประมวลผลข้อมูลขนาด 8 บิท

4.1.2. Register ภายใน ขนาด 8 บิท แบ่งเป็น 2 กลุ่ม คือ

Bank 0 16 ตัว

Bank 1 16 ตัว

4.1.3.อ้างตำแหน่งหน่วยความจำข้อมูล + โปรแกรมได้ 64 Kbyte

4.1.4.stack register 16 ตำแหน่งขนาด 16 บิท

4.1.5.Max clock 20 MHz จากการคำนวณตอนออกแบบยังไม่ทดลอง

4.1.6.I/0 บิท 0 ถึง 7 ของ Address bus สามารถดีโค้ดเป็น port ตำแหน่งต่าง ๆ 128 ตำแหน่ง

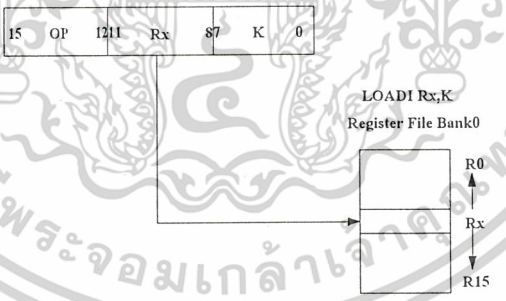
4.1.7.รับ interrupt 2 แหล่ง IN0, IN1

4.1.8ประกอบด้วยชุดคำสั่งจำนวน 67 คำสั่ง

4.2 โหมดการอ้างแอดเดรส (data Addressing mode)

4.2.1 แบบทันทีทันใด (immediate Addressing)

เป็นการเข้าถึงบองค่าคงที่โดยตรง ไม่ว่าจะเป็นคำสั่งทางคณิตศาสตร์ การนำข้อมูลเก็บรีจีสเตอร์



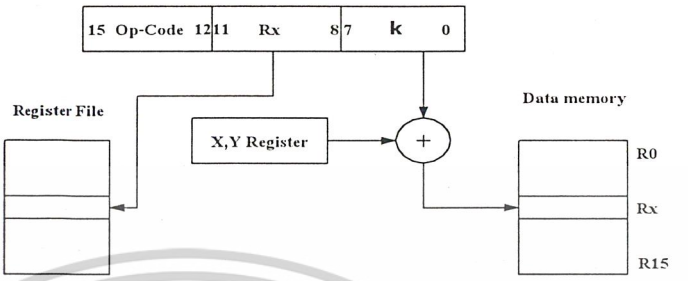
รูปที่ 4.2 แสดงการทำงานในโหมดการอ้างอิงแบบทันทีทันใด

ตัวอย่างคำสั่ง เช่น

SUBI AC, K	เป็นคำสั่งที่นำค่าใน AC-K โดยตรง
LOADI RX, K	นำค่าคงที่ K เก็บใน RX

4.2.2 การอ้างแอดเดรสแบบอ้อมด้วยการเพิ่มจากค่าเริ่มต้น

เป็นการใช้รีจีสเตอร์ x และ y ในการชี้ตำแหน่ง Address ในหน่วยความจำ



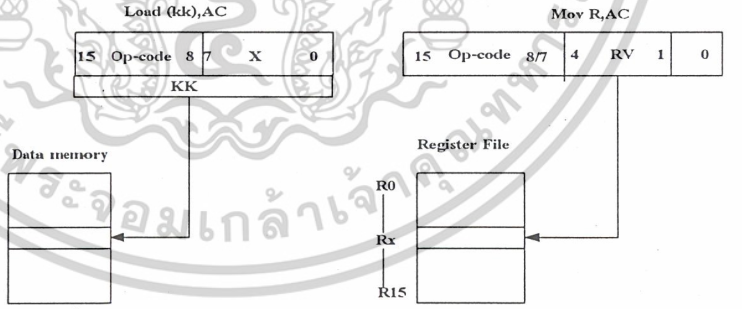
รูปที่ 4.3 แสดงการทำงานในโหมดการอ้างแอสเดรสแบบอ้อมด้วยการเพิ่มจากค่าเริ่มต้น

ตัวอย่างคำสั่ง เช่น

LOAD RX, (X+K) เป็นการนำค่าในตำแหน่ง (X+K) ไปเก็บใน RX

4.2.3 การอ้างแอดเดรสแบบโดยตรง (direct Addressing)

เป็นการใช้วิธีการเข้าถึงค่าในรีจีสเตอร์หรือตำแหน่งหน่วยความจำโดยอ้างถึงรีจีสเตอร์ หรือ ตำแหน่งหน่วยความจำโดยตรง



รูปที่ 4.4 แสดงรูปแบบการอ้างตำแหน่งแบบโดยตรง

ตัวอย่างคำสั่ง เช่น

LOAD (KK) , AC

MOV R, AC

4.2.4 การอ้าง Address แบบตีความหมายเอง (Implied Addressing)

เป็นการอ้าง Address แบบที่ CPU จะตีความหมายโดยอัตโนมัติว่าเป็นการใช้ Register ตัวหนึ่ง ใน CPUเป็นตัวที่ถูกกระทำ

ตัวอย่างคำสั่ง

ADD AC, RX รู้ว่าให้นำค่าใน RX บวกกับค่าใน AC แล้วเก็บในรีจีสเตอร์ AC

ROR รู้ว่าเป็นการกระทำกับ Register AC

4.2.5 การอ้างแอดเดรสแบบเข้าสู่บิท (Bit Addressing)

ใน KC-I มีคำสั่งที่ใช้ในการเข้าสู่บิทโดยตรง ในจำนวน 8 บิท ของข้อมูลเช่น

CLRB RX, K เป็นการสั่งให้ clear bit ที่ k ใน register rx

4.3 แฟล็กใน KC-I อยู่ในรีจีสเตอร์ สเตตัส(Status)

เป็นรีจีสเตอร์ที่ใช้แสดงสถานะ (status register) ซึ่งผลที่ได้จากการทำงานทางลอจิกหรือทางคณิตศาสตร์ จะถูกเก็บใน AC รีจีสเตอร์ ซึ่งค่าใน AC จะส่งผลต่อแฟลกบางตัวใน (status register) ซึ่งเราสามารถแสดงรายละเอียดดังนี้

X	X	X	X	CY	Z	HC	N

รูปที่ 4.5 แสดงตำแหน่งของแฟล็กในรีจีเตอร์สเตตัส

-bit (0)(N- FLAG) แฟล็กการลบ เป็นตัวบอกว่าคำสั่งที่กระทำทางคณิตศาสตร์เป็นบวกหรือลบ โดยถ้าเป็นลบ บิทนี้จะเป็น '1'

- bit (1) (HC-FLAG) แฟล็กตัวช่วยทด เป็นตัวที่เป็นตัวทดระหว่าง 4 บิทล่าง และ 4 บิทบน โดยถ้ามีการทด bit นี้จะเป็น '1'

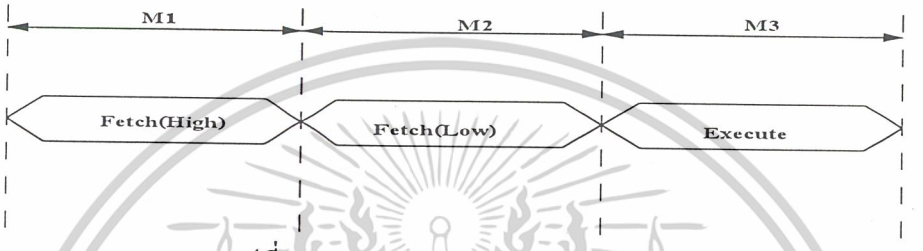
- bit (2)(Z-FLAG) แฟล็กศูนย์ (zero flag) เป็นตัวแสดงว่าค่าใน AC เป็น 0 หรือไม่ ถ้าค่าใน AC เท่ากับ 0 แฟลกนี้จะเป็น '1'

- bit (3) (CY-FLAG) แฟล็กตัวทด (carry flag) เป็นตัวแสดงค่าที่เกิดจากการกระทำทางคณิตศาสตร์ จากข้อมูลในรีจีสเตอร์ AC คือ ถ้ามีการทดหรือมีการยืมในบิทที่ 9 บิทที่เกินจะแสดงใน carry flag

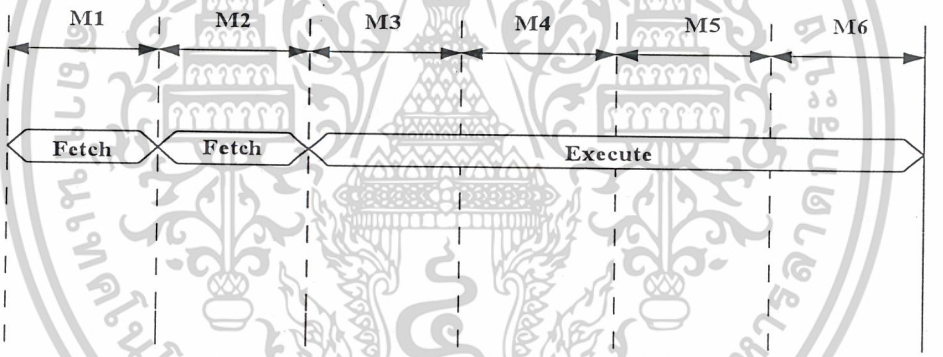
4.4 สัญญาณนาฬิกาแสดงจังหวะการทำงาน

หนึ่ง machine cycle ของ KC-I ประกอบด้วย 16 ลูกคลื่น ซึ่งชุดคำสั่งส่วนมากจะใช้จำนวน 3 แม็ทชีนไซเคิล ยกเว้นคำสั่งที่เกี่ยวกับการกระโดดไปทำงาน ณ ตำแหน่งต่าง ๆ คำสั่ง RETURN คือคำสั่งที่ใช้จำนวนหน่วยความจำ จำนวน 4 byte จะใช้จำนวน 6 แม็ทชีนไซเคิล

ซึ่งจะขอแสดงจังหวะการทำงานของคำสั่งที่ใช้จำนวน 3 ไซเคิล และจำนวน 6 ไซเคิล อย่างละหนึ่งตัวอย่าง ดังนี้



รูปที่ 4.6 แสดง Timming Execute 3 cycle



รูปที่ 4.7 แสดง Timming Excute 6 cycle

ทุกคำสั่งต้องประกอบด้วย 2 แม็ทชีนไซเคิลแรกเสมอ ซึ่งจะใช้ในการเฟลท program จาก memory program ครั้งละ 8 บิท เป็นจำนวน 2 ครั้ง คือ ใน M1 และ M2 ตามลำดับ และจะทำการ decode และ execute ใน m3 ถ้าเป็นคำสั่งที่ใช้จำนวน 3 cycle ก็จะทำงานเสร็จใน M3 แต่ถ้าเป็นคำสั่งที่ใช้จำนวน 6 cycle ก็จะทำงานเพิ่มไปใน M4 ,M5 และ M6 อีกดังรูปที่ (1) และ (2) ตามลำดับ

4.5 การอินเตอร์รัพท์ใน KC-I

การอินเตอร์รัพท์ใน KC-I มีอยู่ด้วยกัน 2 แหล่ง คือ IN0, IN1 ถึง การจะใช้ขาอินเตอร์รัพทุ ดังกล่าวมีความเกี่ยวข้องกับชุดคำสั่งของ KC-I คือ

DIS0 เป็นการกำหนดให้ IN0 ใช้งานไม่ได้

EN0 การกำหนดให้ IN0 สามารถใช้งานได้โดยเมื่อสถานะที่ขา IN0 เป็น '1'

KC-I จะกระโดดไปทำงานยังตำแหน่ง อินเตอร์รัพท์ที่กำหนดไว้ (แล้วแต่จะกำหนด)

EN1 เป็นการกำหนดให้ IN1 สามารถใช้งานได้โดยเมื่อสถานะที่ขา IN1 เป็น'1'

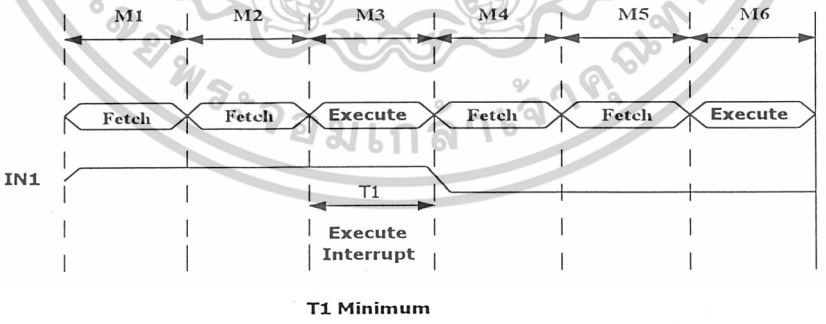
KC-I จะกระโดดไปทำงานยังตำแหน่งอินเตอร์รัพท์ (แล้วแต่จะกำหนด)

DIS1 เป็นการกำหนดให้ IN1 ใช้งานไม่ได้ไม่สามารถอินเตอร์รัพท์ได้

**(IN1 มีลำดับความสำคัญมากกว่า IN0)

โดยเมื่อสถานะที่ขา IN0 หรือ IN1 เป็น '1' KC-I จะทำการอินเตอร์รัพท์เมื่อสิ้นสุดกำลังสุดท้ายที่ทำอยู่ โดย KC-I จะเก็บค่าของ program counter ขณะนั้นไปเก็บใน stack ก่อนแล้วจึงกระโดดไปทำยังตำแหน่งอินเตอร์รัพท์ และในส่วนของโปรแกรมอินเตอร์รัพท์ต้องมีคำสั่ง RETI อยู่ในส่วนโปรแกรมด้วยเพื่อกลับสู่โปรแกรมหลัก

สถานะ '1' ที่ ขา IN0, IN1 ต้องนานพอที่จะทำให้เกิด Interrupt คือประมาณ 3 แม็ทชีนไซเคิล (16 ลูกคลื่น) 2.4 ms ที่ 20 MHz (ความถี่สูงสุดใช้งาน) แสดงไทมิ่งไดอะแกรมดังรูป



รูปที่ 4.8 แสดงไทม์มิ่งการทำ INTERRUPT

4.6 ชุดคำสั่ง (INSTRUCTION GROUP)

KC-I ประกอบด้วยชุดคำสั่งทั้งหมด 67 คำสั่ง ซึ่งแยกเป็นกลุ่มตามลักษณะการทำงานดังนี้

1. คณิตศาสตร์ และลอจิก (Arithmetic and Logic)
2. การจัดการเกี่ยวกับบิท (Bit-orient and bit-Test)
3. การหมุนและเลื่อนข้อมูล (Rotate and shift)
4. ควบคุมซีพียู (CPU control)
5. การโอนย้ายข้อมูล (Data Transfer)
6. การกระโดด (Jumpgroup)
7. การเรียกโปรแกรมย่อยและกลับสู่โปรแกรมหลัก (call and return)

รายละเอียดของแต่ละกลุ่มมีดังนี้

4.6.1 คณิตศาสตร์และลอจิก (Arithmetic and Logic) 13 คำสั่ง

รูปแบบคำสั่ง	แสดงการทำงาน	อ๊อฟโค้ด	ฐาน 16	ผลต่อแฟล็ก	จำนวน M	หมายเหตุ	
ADD AC,RV	AC ← AC+RV	00001101000BBBBd	0DHH	CY,Z,HC, N	3	BBBB	RV
ADC AC,RV	AC ← AC+RV+CY	00000001000BBBBd	01HH	CY,Z,HC, N	3	0000	R0
SUB AC,RV	AC ← AC-RV	00000010000BBBBd	02HH	CY,Z,HC, N	3	0001	R1
SUBC AC,RV	AC ← AC-RV-CY	00000011000BBBBd	03HH	CY,Z,HC, N	3	0010	R2
ADDI AC,K	AC ← AC+K	00000100bbbbbbbb	04HH	CY,Z,HC, N	3	0011	R3
SUBI AC,K	AC ← AC-K	00000101bbbbbbbb	05HH	CY,Z,HC, N	3	0100	R4
AND AC,RV	AC ← AC*RX	00000110000BBBBd	06HH	Z	3	0101	R5
ANDI AC,RV	AC ← AC*K	00000111bbbbbbbb	07HH	Z	3	0110	R6
OR AC,RV	AC ← AC+RV	00001000bbbbbbbb	08HH	Z	3	0111	R7
ORI AC,K	AC ← AC+K	00001001bbbbbbbb	09HH	Z	3	1000	R8
						1001	R9
						1010	R10
						1011	R11
						1100	R12
						1101	R13
						1110	R14
						1111	R15

ตารางที่ 4.1 แสดงชุดคำสั่งทางคณิตศาสตร์และตรรกะ

EXOR AC,RV	AC ← A ⊕ RV	00001010000BBBBd	0AHH	Z	3	Register Bank (0) d=don't care HH ขึ้นกับค่าที่เติม b=0 or 1
INC RV	RV ← RV+1	00001011000BBBBd	0BHH	-	3	
DEC RV	RV ← RV-1	00001100000BBBBd	0CHH	-	3	

ตารางที่ 4.1 แสดงชุดคำสั่งทางคณิตศาสตร์และตรรกะ

4.6.2 การจัดการเกี่ยวกับบิท (Bit-orient and bit-test) 8 คำสั่ง

รูปแบบคำสั่ง	การทำงาน	ฐานสอง อ๊อฟโค้ด	ฐาน16	ผลต่อแฟล็ก	จำนวน M	หมายเหตุ
SETB RV,k	RV ← bit(k) = 1 others = 0	01100000xxxBBBBd	60HH	-	3	k= ตำแหน่ง bit
CLRB RV,K	RV ← bit(k) = 0 others = 1	01000000xxxBBBBd	40HH	-	3	
CLR RV	RV ← zero	00110000000BBBBd	30HH	-	3	
SET RV	RV ← High All bit	00010000000BBBBd	10HH	-	3	
OCOM RV	RV ← 1's of RV	01010000000BBBBd	50HH	-	3	
TCOM RV	RV ← 2's of RV	01110000000BBBBd	70HH	-	3	
CLRST	Status Reg ← zero	00001111ddddddddd	0FHH	-	3	
BITT RX,K	Test bit(k) of RV	00100000xxxBBBBd	20HH	-	3	

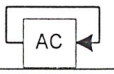
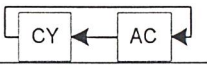
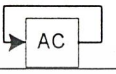
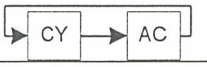
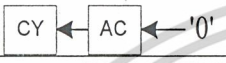
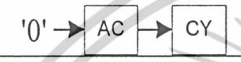
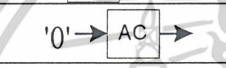
หมาย	xxx	000	001	010	011	100	101	110	111
เหตุ	Bit(k)	0	1	2	3	4	5	6	7

BBBB=0000 → 1111

RV= R0 → R15 (bank0)

ตารางที่ 4.2 แสดงชุดคำสั่งการจัดการเกี่ยวกับบิท

4.6.3 การหมุนและเลื่อนข้อมูล (Rotate and shift) 8 คำสั่ง

รูปแบบคำสั่ง	การทำงาน	ฐาน 2		ฐาน 16	ผลต่อแฟล็ก	จำนวน M
ROL		00001110	00000000	0E00	Z	3
ROLC		00001110	00100000	0E20	Z	3
ROR		00001110	01000000	0E40	Z	3
RORC		00001110	01100000	0E60	Z	3
SILC		00001110	10000000	0E80	Z	3
SIRC		00001110	10100000	0EA0	Z	3
SIL		00001110	11000000	0EC0	Z	3
SIR		00001110	11100000	0EE0	Z	3

ตารางที่ 4. 3 แสดงชุดคำสั่งการหมุนและเลื่อนข้อมูล

4.6.4 ควบคุมซีพียู (cpu control) 5 คำสั่ง

รูปแบบคำสั่ง	การทำงาน	ฐาน2	ฐาน16	ผลต่อแฟล็ก	จำนวนM
NOTOP	CPU No Operate	0000000000000000	0000	No	3
DINO	ยกเลิกการใช้ขา IN0	0000000000100000	0020	No	3
EINO	สามารถใช้ขา IN0ได้	0000000001100000	0060	No	3
DIN1	ไม่สามารถใช้ขา IN1ได้	0000000001000000	0040	No	3
EIN1	สามารถใช้ขาIN1ได้	0000000010000000	0080	No	3

ตารางที่ 4.4 แสดงชุดคำสั่งเกี่ยวกับการควบคุมซีพียู

4.6.5 การโอนย้ายข้อมูล (data transfer) 17 คำสั่ง

รูปแบบคำสั่ง	แสดงการทำงาน	อ๊อฟโค๊ด	ฐาน16	ผลต่อแฟล็ก	จำนวน M	หมายเหตุ
LOAD RV,(x+k)	RV ← (X+K)	1001AAAAbbbbbbbb	9HHH	NO	3	-
LOADI RV,K	RV ← K	1101AAAAbbbbbbbb	DHHH	NO		-
LOAD RV,(y+k)	RV ← (y+k)	1011AAAAbbbbbbbb	BHHH	NO		-
LOAD (y+k),RV	(y+k) ← RV	1010AAAAbbbbbbbb	AHHH	NO		bank(0)
LOAD (x+k),RV	(x+k) ← RV	1111AAAAbbbbbbbb	FHHH	NO		bank(0)
IN RV,P	RV ← PORT(p)	1100AAAAbbbbbbbb	CHHH	NO		8bit low is port
OUT P,RV	PORT(p) ← RV	1110AAAAbbbbbbbb	EHHH	NO		bank(0)
LOAD X,kk	x ← kk	0000000011100000	00E0	NO	6	-
LOAD Y,kk	y ← kk	0000000011110000	00F0	NO	6	-
MOV AC,RV	AC ← R	1000000ddddBBBBd	8HHH	Z	3	bank(0)
MOV RV,R1V	RV ← R1V	1000001AAAABBBBd	8HHH			-d=1 R1v to Rv -d=0 Rv to R1v
MOV RV,AC	RV ← AC	1000010ddddBBBBd	8HHH	NO	3	bank(0)
MOV RV,(K)	RV ← (k)	1000011ddddBBBBd	8HHH	NO	6	bank(0)
LOAD (KK),AC	(kk) ← AC	1000100ddddBBBBd	8HHH	NO	6	-
PUSH RV	stack ← RV,INC stack pointer	1000101ddddBBBBd	8HHH	NO	3	bank(0)
POP RV	RV ← stack,DEC stack pointer	1000110ddddBBBBd	8HHH	NO	3	bank(0)
LOAD AC,(kk)	AC ← (KK)	1000111ddddBBBBd	8HHH	NO	6	-

AAAA = "0000" → "1111"

Rv = Ro → R15 (bank 0)

BBBB = "0000" → "1111"

R1v = Ro → R15 (bank1)

Rv แทนตำแหน่ง รีจีสเตอร์ bank0, R1v แทนตำแหน่ง รีจีสเตอร์ bank1

ตารางที่ 4.5 แสดงชุดคำสั่งการโอนย้ายข้อมูล

4.6.6 การกระโดดไปยังตำแหน่งที่ต้องการ (Branch group) 5 คำสั่ง

รูปแบบคำสั่ง	แสดงการทำงาน	อ๊อปโค้ด	ฐาน16	ผลต่อแฟล็ก	จำนวน M	หมายเหตุ
BRNZ kk	z=1 pc ← kk,No contineous	0000000000000001	0001	NO	6	-
BRNNZ kk	z=0 pc ← kk,No Next	0000000000000010	0002	NO	6	-
BRNC kk	c=1 pc ← kk,No Next	0000000000000011	0003	NO	6	-
BRNNC kk	c=0 pc ← kk,No Next	0000000000000100	0004	NO	6	-
BRN kk	pc ← kk	0000000000000101	0005	NO	6	-

หมายเหตุ KK มาจาก 2 byte ต่อจาก opcode ยังไม่ได้แสดงในตาราง

ตารางที่ 4.6 แสดงชุดคำสั่งเกี่ยวกับการกระโดด

4.6.7 การเรียกโปรแกรมย่อยและกลับสู่โปรแกรมหลัก (Call and Return) 11 คำสั่ง

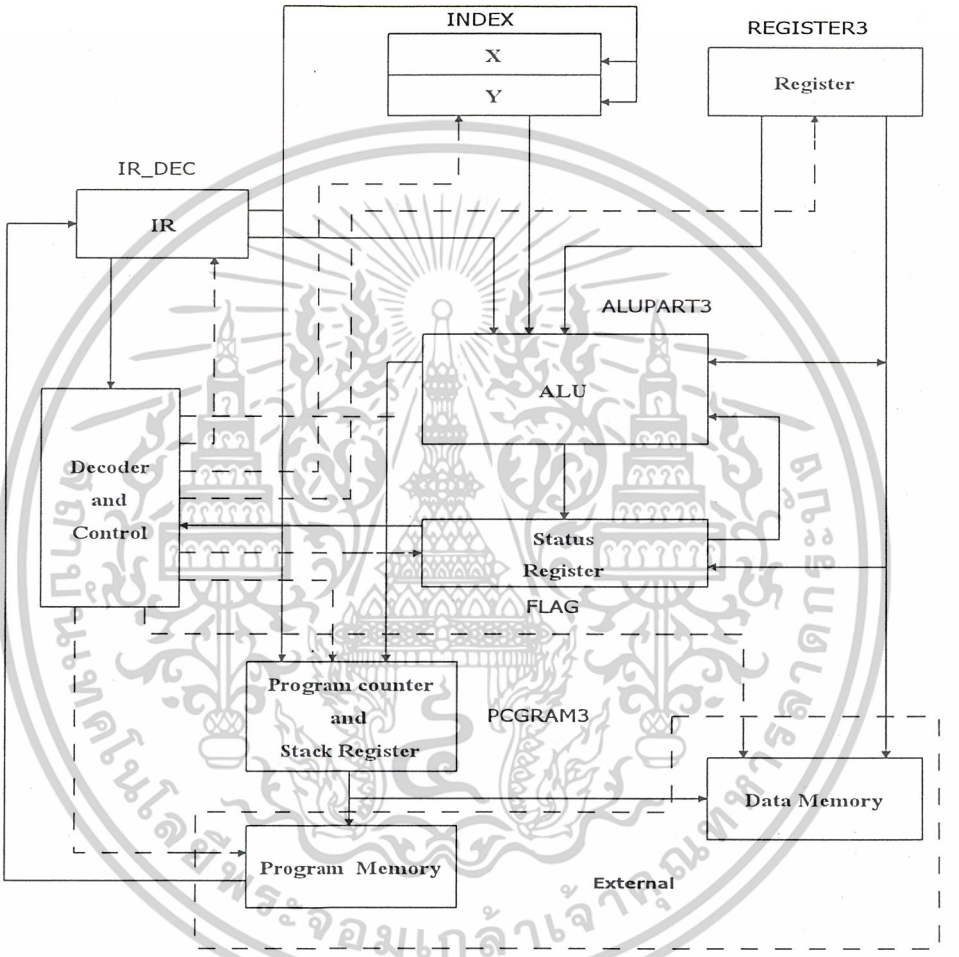
รูปแบบคำสั่ง	แสดงการทำงาน	อ๊อฟโค้ด	ฐาน16	ผลต่อแฟล็ก	จำนวน M	หมายเหตุ
ICALL KK	stack←PC,PC←KK	0000000010010000	0090	NO	6	-
CALLZ KK	Z=1,stack←PC,PC←KK	0000000010100000	00A0	NO	6	-
RET	PC←stack	0000000010110000	00B0	NO	6	-
RETZ	PC←stack if Z=1	0000000011000000	00C0	NO	6	-
RETIN	PC←stack	0000000011010000	00D0	NO	6	Interrupt only
CALLNZ KK	stack←PC,PC←KK ifZ=0	0000000000000110	0006	NO	6	-
CALLC KK	stack←PC,PC←KK ifC=1	0000000000000111	0007	NO	6	-
CALLNZ KK	stack←PC,PC←KK ifC=0	0000000000001000	0008	NO	6	-
RET NZ	PC←stack if Z=0	0000000000001001	0009	NO	6	-
RET C	PC←stack if C=1	0000000000001010	000A	NO	6	-
RET NC	PC←stack if C=0	0000000000001011	000B	NO	6	-

KK ต่อจาก opcode อีก 2 byte ยังไม่ได้เติมในตาราง

ตารางที่ 4.7 แสดงชุดคำสั่งการเรียกโปรแกรมย่อยและกลับสู่โปรแกรมหลัก

บทที่ 5 การออกแบบ

บล็อกไดอะแกรมที่ใช้ในการวิเคราะห์ในการออกแบบได้มาจากการแยกส่วนต่างภายในสถาปัตยกรรมของ KC-I และสามารถนำไปออกแบบแยกเป็น โมดูลต่างๆได้ดังบล็อกไดอะแกรมรูปที่ 5.1



รูปที่ 5.1 แสดงบล็อกไดอะแกรมที่ใช้ในการวิเคราะห์เพื่อออกแบบต่ละโมดูล

ซึ่งในการออกแบบได้แยกออกแบบเป็นแต่ละโมดูล ดังนี้

-โมดูล IR_DEC เป็นส่วนของ Intruction register

-โมดูล INDEX เป็นส่วนของ x, y register

-โมดูล Register3 เป็นส่วนของ Register Bank 0, และ Bank 1

-FLAG เป็นส่วนของ Status register

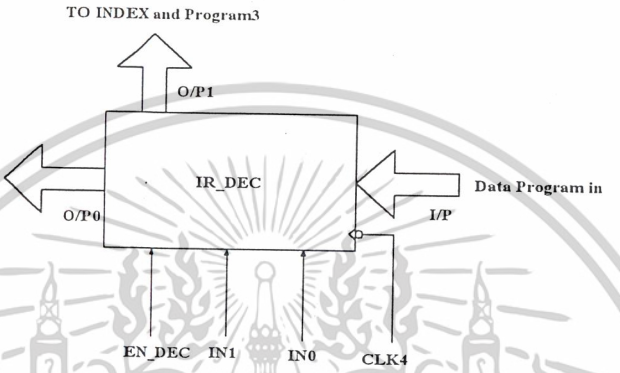
-ALUPART3 เป็นส่วนคำนวณทาง Logic,คณิตศาสตร์,และการกระทำเกี่ยวกับข้อมูลต่าง ๆ

-PCGRAM3 เป็นส่วนของ Program counter และ stack register

-CONTROL PART

ซึ่งรายละเอียดของแต่ละส่วนจะได้อธิบายดังนี้

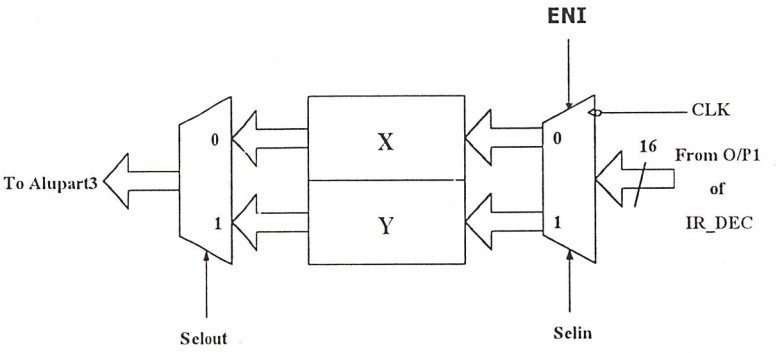
5.1 โมดูลไออาร์เด็ค(IR_DEC) เป็นส่วนของ Intruction register ซึ่งแสดงรูปโมดูลที่มีสัญญาณเข้าออกใช้จริงในการออกแบบ



รูปที่ 5.2 แสดงโมดูล IR_DEC

ทำหน้าที่ในการเก็บค่า data program ที่อ่านมาจาก memory program แล้วส่งค่านั้นต่อไปยังส่วนดีโค้ด โดยสัญญาณ EN-DEC เป็นตัวกำหนดว่าจะให้ข้อมูลไปทำการดีโค้ดหรือไม่ ถ้า EN-DEC = '1' ค่าที่ O/P1=I/P แต่ค่าที่ออกทาง O/P0 ก็จะคงที่แต่ถ้า ENDEC= '0' สัญญาณที่ O/P0 จะเท่ากับที่อินพุททุก ๆ ครั้งที่สัญญาณ CLK4 เปลี่ยนแปลงจาก High เป็น Low ส่วนสัญญาณ IN1 และ IND เป็นสัญญาณที่รับมาจากภายนอกเพื่อทำการอินเทอร์รัพท์ ซีพียู โดยที่ EN-DEC '0' และ IN1 หรือ IN0 '1' (หลังจากมีการใช้คำสั่ง EIN0 หรือ EIN1 ตามลำดับ โดย IN1 มีความสำคัญมากกว่า IN0) ก็จะทำให้KC-I กระโดดไปทำงานยังตำแหน่งอินเทอร์รัพท์ที่กำหนดอยู่ใน KC-I ซึ่งเราสามารถทำการเปลี่ยนแปลงได้ก่อนโปรแกรมลงชิพ

5.2 โมดูลอินเด็ค(INDEX) เป็นส่วนของ register x และ y ใช้ในการอ้างแอดเดรสแบบทางอ้อมเพิ่มค่า จากค่าเริ่มต้นใน x, y register

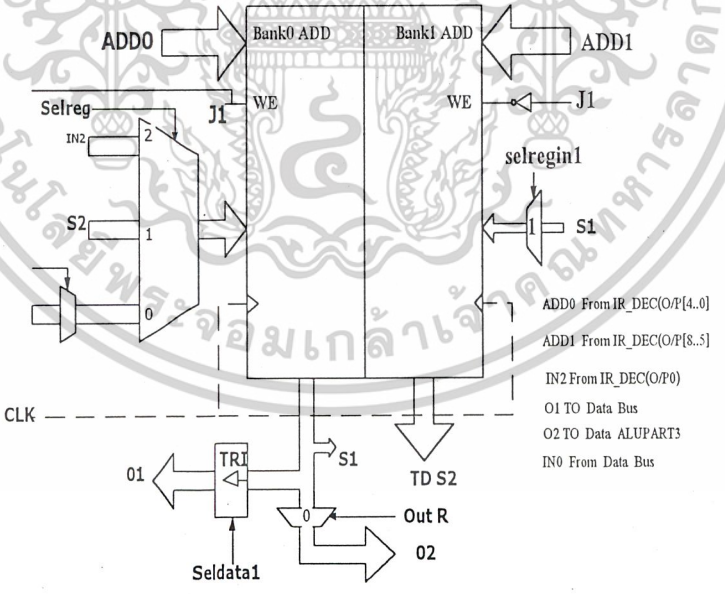


รูปที่ 5.3 แสดงโมดูล INDEX

โดยค่าที่มาจาก IR-DEX จะถูกเก็บใน x, y register เมื่อ Eni = '0' และ c/k เป็นจาก High ไป Low โดยถ้า selin = '0' ค่านั้นก็จะถูกเก็บใน y-register

ในการนำค่าออกจาก x, y-register โดยการกำหนดค่าให้ selout โดยปกติ selout จะเท่ากับ '0' เสมอ ดังนั้นในกรณีต้องการ เอา x ออกก็ไม่ต้องเปลี่ยนแปลงค่า แต่ถ้าต้องการเอาค่าใน y ออก ต้องกำหนดค่า '1' ให้กับ selout

5.3 โมดูลรีจีสเตอร์3(REGISTER3)



รูปที่ 5.4 แสดงโมดูล REGISTER3

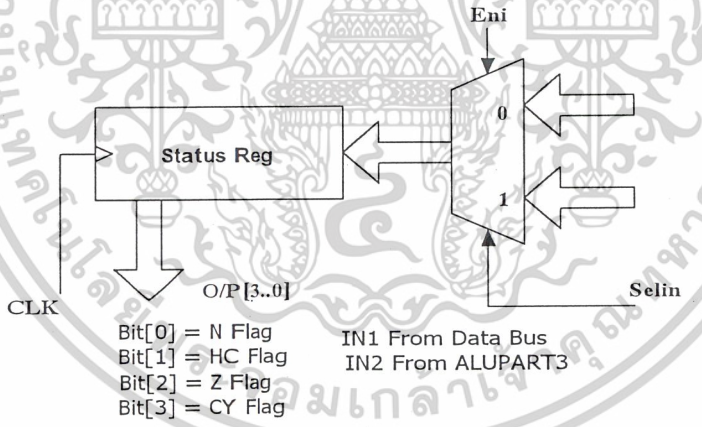
กำหนดให้ทำหน้าที่เป็น General perpose Register โดยแบ่งเป็น 2 แบงค์ คือ แบงค์ 0 กับ แบงค์ 1 โดยการทำงานส่วนมากจะทำงานร่วมกับ bank 0 มากกว่า สัญญาณ ADD1 รับค่ามาจาก 0/P(0) ของโมดูล IR-DEC เพื่อทำการกำหนดตำแหน่งให้ Register bank 1 สัญญาณ ADD0 รับค่ามาจาก 0/P (0) ของโมดูล IR-DEC เพื่อทำการกำหนด Address ของ bank 0 [4..1]

สัญญาณ selreg เป็นตัวเลือกว่าจะนำค่าไหนเก็บใน register bank 0 โดยถ้า selreg = '0' และ selbus 1 = '1' , selalu = '1' และมี CLK ก็จะสามารถนำค่าจาก IN0 เก็บใน register bank0 ตำแหน่งที่ชี้อยู่ได้ หรือถ้ากำหนด selalu = '0' outr = '0' และมี clk ก็จะเป็นการนำค่าจาก register bank (0) ณ ตำแหน่งปัจจุบันออกทาง o2 ได้

สัญญาณ seldata 1 ไว้สำหรับกำหนดค่าจาก register สู่ data bus โดยถ้ามีค่าเป็น '1' ก็จะเป็นการกำหนดค่าสู่ data bus ถ้ามีค่าเป็น '0' ก็จะทำให้ 01= 'Z'

ในกรณีที่จะนำค่าจาก bank (0) เก็บใน bank (1) ต้องกำหนดให้ selregin 1 = '1' , selalu = '0' พร้อมกับมีสัญญาณ CLK และ Address ชี้อยู่ในขณะนั้น

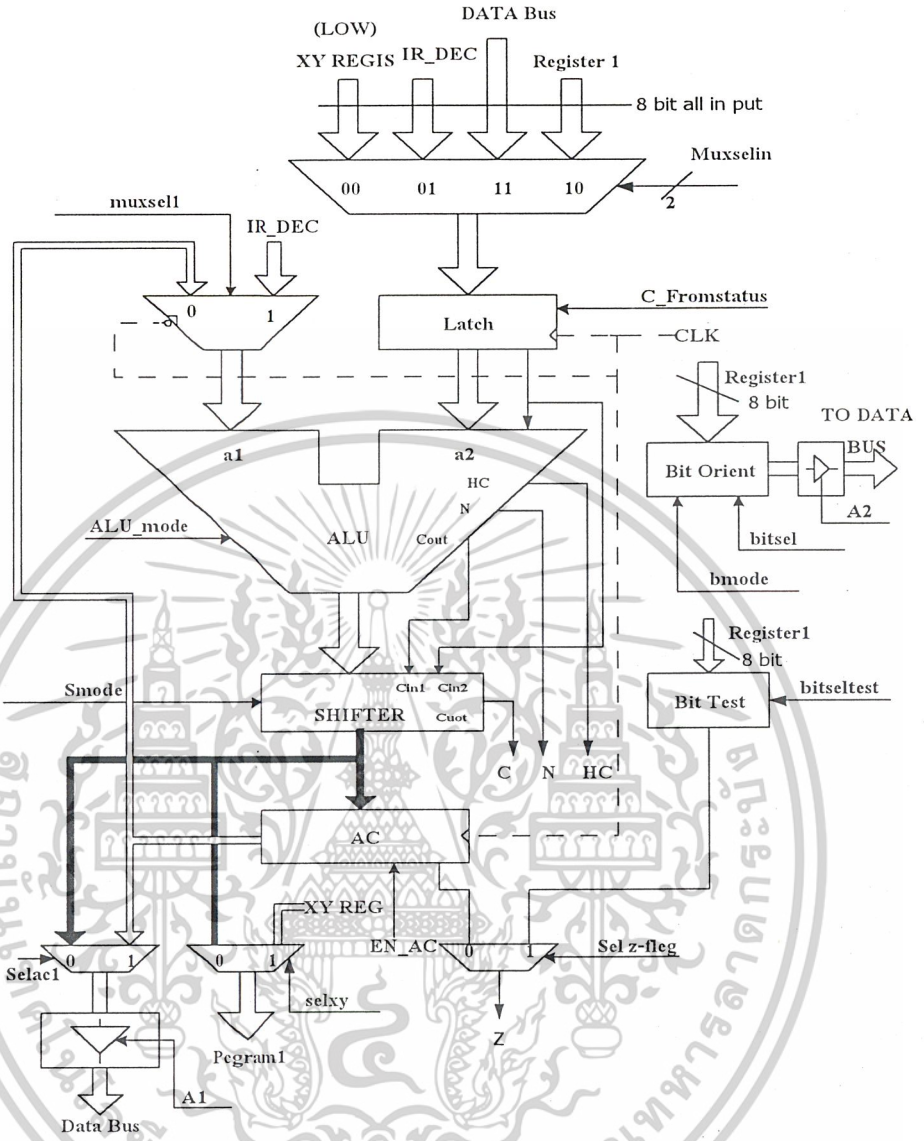
5.4 โมดูลแฟลก(FLAG)



รูปที่ 5.5 แสดงโมดูล FLAG

ใช้เก็บสถานะที่ผลมาจากการทำคำสั่งของ cpu โดยสัญญาณ selin เป็นตัวเลือกว่าจะนำค่าจาก IN0 หรือ IN1 เก็บใน status reg ถ้าเป็น '1' เป็นการนำค่ามาจากการทำงานของA/U หรือ ถ้าเป็น '0' ก็เป็นการนำค่าจาก data bus เข้าไปเก็บ แต่จะสามารถเก็บค่าต่าง ๆ ได้ก็เมื่อ Eni = '1' และ clk เป็นขอบขาขึ้น

5.5 โมดูลพาร์ต2(ALUPART2)



รูปที่ 5.6 แสดงโมดูล ALUPART3

เป็นส่วนที่ใช้ทำงานด้านการคำนวณทางคณิตศาสตร์ และการทำงานทางลอจิก การจัดการเกี่ยวกับข้อมูลภายในรีจีสเตอร์ โดยข้อมูลที่เข้าสู่ Alupart1 มาจาก โมดูล Register 1, IN-DEX, IR-DEC และจาก data bus โดยมีสัญญาณ muxselin เป็นตัวกำหนดว่าจะให้ข้อมูลเข้ามา สัญญาณ muxsel1 เป็นตัวเลือกว่าจะให้ข้อมูลใดผ่านเข้าสู่ ALU ด้าน a1 ALUMODE เป็นตัวกำหนดว่าจะให้ ALU ทำงานทางคณิตศาสตร์แบบใด selac1 เป็นตัวกำหนดว่าจะให้ข้อมูลใดออกสู่ bus data โดย

ขึ้นอยู่กับ สัญญาณ A 1 ด้วยว่าเป็น '1' หรือ '0' ถ้า A1= '1' ข้อมูลก็สามารถออกสู่ DATA bus ได้ แต่ถ้า A1 = '0' เอาท์พุทบัฟเฟอร์ก็จะเป็น high impedance

Smode เป็นตัวกำหนดว่ารจะให้มีการเลื่อนข้อมูล หรือหมุนข้อมูลแบบใด

EN-AC เป็นตัวกำหนดว่าจะให้ข้อมูลผ่านเข้าไปเก็บใน AC register หรือไม่ ถ้าเป็น '0' ข้อมูลจึงจะผ่านได้

Selxy เป็นตัวกำหนดให้ข้อมูลเข้าสู่ xy-bus ซึ่งต่อไปยัง pcgram1 โมดูล

Selzflag เป็นตัวเลือกว่าจะให้ข้อมูลที่เข้าสู่ z-flg เป็นผลจากการทำงาน จาก ALU หรือจากการ Test bit

Bitseltest เป็นตัวกำหนดว่า บิทใดของข้อมูลที่มาจาก register จะถูกทดสอบ c-fromstatus เป็นค่าของตัวทดที่จะเข้ามาใช้ในการทำงานของALU

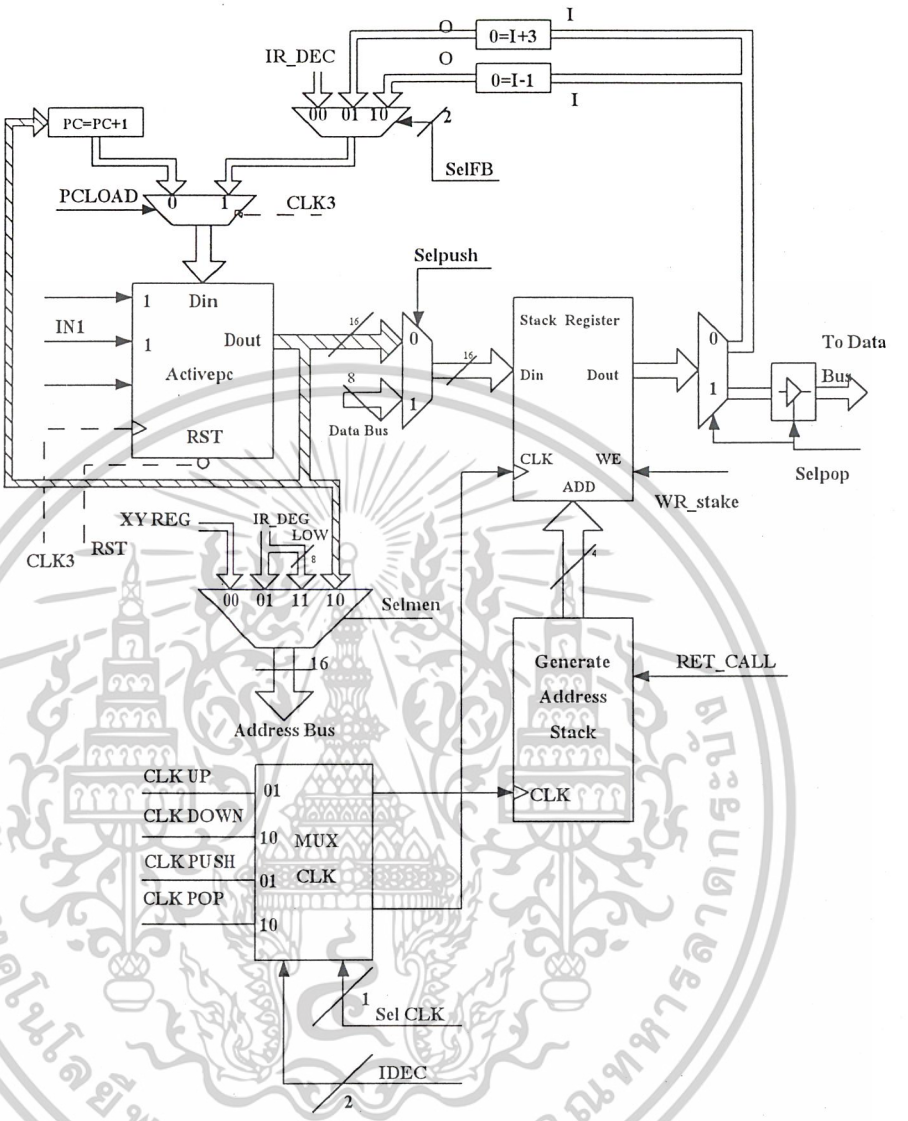
Bmode เป็นตัวเลือกว่าจะให้การจัดการกับข้อมูลที่เข้ามาอย่างไร โดยข้อมูลที่เข้ามา จะมาจาก register 1 โมดูล ซึ่งจะทำงานร่วมกับสัญญาณ bitsel ซึ่งจะคอยเป็นตัวกำหนดตำแหน่งบิทที่จะใช้ในการทำงาน ส่วน A2 เป็นสัญญาณที่จะกำหนดว่าจะให้ข้อมูลผ่านไปยัง data bus หรือไม่ โดยถ้า A2 = '1' ข้อมูลก็จะสามารถผ่านไปได้

จากที่กล่าวมาทั้งหมด ได้อธิบายหน้าที่ของสัญญาณต่าง ๆ ที่ใช้ควบคุมส่วนของ ALUPART1 ดังแสดงในรูปที่ 5.6

5.6 โมดูลพีซีแกรม3(PCGRAM3)

เป็นโมดูลที่ใช้ในการกำเนิดตำแหน่งของหน่วยความจำภายนอก ซึ่งสามารถกำหนดได้ 64 K byte ซึ่งประกอบด้วยส่วนสำคัญคือ stack register ที่ใช้ในการเก็บตำแหน่งหน่วยความจำและข้อมูลจากรีจีสเตอร์ใช้งานทั่วไป general Add stack ซึ่งคอยกำหนด แอดเดรส ให้กับ stack register กรณีที่มีการ push หรือ pop ข้อมูลออก Activepc คอยทำการกำหนดตำแหน่งต่อไปของหน่วยความจำ muxclk ที่คอยกำหนดสัญญาณนาฬิกาสำหรับการนำข้อมูลเข้าและออกจาก stack รีจีสเตอร์ และการกำเนิดตำแหน่งของ stack register ในกรณีที่มีการนำข้อมูลเข้า stack register ตำแหน่งของรีจีสเตอร์จะเพิ่มขึ้นหนึ่งตำแหน่งหลังจากนำข้อมูลเข้าเก็บแล้ว และในกรณีนำข้อมูลออกจาก stack register ตำแหน่งรีจีสเตอร์จะลดลงหนึ่งตำแหน่งก่อนนำข้อมูลออกโดยตำแหน่งของ stack registers จะวนไปเรื่อย ระหว่าง 00-FF โดย RET-CALL เป็นตัวกำหนดว่าจะเพิ่มหรือลดค่าตำแหน่งของ stack register ในกรณีที่จะรีเซ็ตการทำงานต้องให้สัญญาณ Rst = '0' จะทำให้ Address bus เริ่มต้นที่ ("0000"H) และกรณีมีการอินเทอร์รัพท์ขึ้น ก็จะทำให้ Address bus ไปยังตำแหน่งอินเทอร์รัพท์ โดยการกำหนดค่าจาก Activepc ซึ่งรับการควบคุมมาจากส่วนควบคุมว่าจะให้ทำอินเทอร์รัพท์โหมดใดผ่านมาทาง

IN0 และIN1 ส่วน ค่าต่าง ๆ ของแต่ละสัญญาณจะแสดงในส่วนของโมดูลการควบคุมซึ่งมีค่าต่าง ๆ กัน ในแต่ละคำสั่ง



รูปที่ 5.7 แสดงโมดูล PCGRAM3

5.7 โมดูลการควบคุม (control part)

ในการออกแบบได้แยกโมดูลควบคุมเป็นส่วน ๆ คือ แบ่งให้แต่ละโมดูลมีส่วนควบคุมของตัวมันเอง ดังนี้

5.7.1 ส่วนควบคุม โมดูล INDEX

โดยส่วนนี้จะมีชุดคำสั่งที่เกี่ยวข้องดังตารางที่ 5.1 รวมทั้งสัญญาณต่าง ๆ ที่ส่งไปควบ คุมโมดูลให้ทำงานตามคำสั่งแต่ละคำสั่ง จากการถอดรหัสของแต่ละคำสั่ง

ชุดคำสั่ง	Selin	Eni	Selout
LOAD RV,(Y+K)	0	1	1
LOAD (Y+K),RV	0	1	1
LOAD X,KK	0	0	0
LOAD Y,KK	1	0	0
OTHERS	0	1	0

ตารางที่ 5.1 แสดงค่าของสัญญาณต่าง ๆ ในการควบคุม โมดูล INDEX

5.7.2 ส่วนควบคุม โมดูล REGISTER3

ส่วนนี้จะมีชุดคำสั่งที่เกี่ยวข้องดังตารางที่ 5.2 รวมทั้งสัญญาณ ต่าง ๆ ที่ส่งไปควบคุม โมดูลให้ทำงานตามคำสั่งแต่ละคำสั่งดังนี้

ชุดคำสั่ง	สัญญาณ				
	Out R	Selbus 1	selregin1 (1)	selregin1 (0)	seldata1
ADDAC,AV	0	0	0	0	0
ADC AC,RV	0	0	0	0	0
SUB AC,RV	0	0	0	0	0
SUBC AC,RV	0	0	0	0	0
ADDI AC,K	1	0	0	0	0
SUBI AC,K	1	0	0	0	0
AND AC,RV	0	0	0	0	0
ANDI AC,K	1	0	0	0	0
OR AC,R	0	0	0	0	0
ORI AC,K	1	0	0	0	0
EXOP AC,RV	0	0	0	0	0
INC RV	0	1	0	0	0
DEC RV	0	1	0	0	0
SETB RV,K	1	1	0	0	0

ตารางที่ 5.2 แสดงสัญญาณต่าง ๆที่ใช้ควบคุมโมดูล REGISTER3

ชุดคำสั่ง	สัญญาณ				
	OUTR	Selbus1	Selregin1(1)	Selregin1(0)	Seldata1
CIRB RV,K	1	1	0	0	0
CLR RV	1	1	0	0	0
SET RV	1	1	0	0	0
DCOM RV	0	1	0	0	0
TCOM RV	0	1	0	0	0
BITT RV,K	0	0	0	0	0
LOAD RV,(X+K)	1	1	0	0	0
LOAD RV,K	1	1	1	0	0
LOAD RV,(Y+K)	1	1	0	0	0
LOAD (Y+K),RV	1	0	0	0	1
LOAD (X+K),RV	1	0	0	0	1
ชุดคำสั่ง	สัญญาณ				
	Out R	Selbus 1	selregin1 (1)	selregin1 (0)	seldata1
IN RV,P	1	1	0	0	0
OUT P,RV	1	0	0	0	1
MOV AC,RV	0	0	0	0	0
MOV RV,RV	1	0	0	1	0
MOV RV,AC	1	1	0	0	0
LOAD RV,(KK)	1	1	0	0	0
PUSH RV	1	0	0	0	1
POP RV	1	1	0	0	0
Others	1	0	0	0	0

ตารางที่ 5.2 แสดงสัญญาณต่าง ๆที่ใช้ควบคุมโมดูล REGISTER3(ต่อจากหน้าที่ 49)

5.7.3 ส่วนควบคุมโมดูล ALUPART3

ส่วนนี้จะมีชุดคำสั่งที่เกี่ยข้องดังตารางที่ 5.3 รวมทั้งสัญญาณที่ส่งไปควบคุมโมดูลให้ทำงานตามคำสั่งแต่ละคำสั่ง และรวมส่วน คอนโทรลสเตตัส(Status Control) ไว้ด้วย

	สัญญาณ																					
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M
ชุดคำสั่ง																						
Add AC,RV	0	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
ADC AC,RV	0	0	1	1	0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
SUB AC,RV	0	0	1	1	0	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0
SUBC AC,RV	0	0	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
ADDI AC,K	0	0	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
SUBI AC,K	0	0	1	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0
AND AC,RV	0	0	1	1	0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0
ANDI AC,K	0	0	1	1	0	0	1	0	1	1	0	1	0	1	0	0		0	0	0	0	0
OR AC,RV	0	0	1	1	0	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0
ORI AC,K	0	0	1	1	0	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0
EXOR AC,RV	0	0	1	1	0	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0
INC RV	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
DEC RV	0	1	1	1	0	1	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0
SETB RV,K	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	1	1	0	0	0	0	1
CLRB RV,K	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1
CLR RV	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	1
SET RV	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1
OCOM RV	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	0	0	1

ตารางที่ 5.3 แสดงสัญญาณการควบคุมโมดูล ALUPART3

	สัญญาณ																					
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M
ชุดคำสั่ง																						
BITT RV,K	0	0	1	1	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	1
ROL	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
ROLC	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
ROR	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
RORC0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0
SILC	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
SIRC	0	0	1	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0
SIL	0	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0
SIR	0	0	1	0	0	1	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0
LOADRV,(x+k)	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
LOADRV,(y+k)	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
LOAD(y+k),RV	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
LOAD(x+k),RV	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1
MOV AC,RV	0	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0
MOV RV,AC	0	1	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1
LOAD (KK),AC	0	1	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1
LOAD AC,(KK)	0	0	1	0	0	1	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0

ตารางที่ 5.3 แสดงสัญญาณการควบคุมโมดูล ALUPART3(ต่อจากหน้าที่ 51)

**** หมายเหตุ

1= A2 2=A1 3=SELIN 4=ENI 5=MUXSEL1 6=MUXSELIN(1)

7=MUXSELIN(0) 8=ALUMODE(3) 9=ALUMODE(2) A=ALUMODE(1)

B=ALUMODE(0) C=SMODE(3) D=SMODE(2) E=SMODE(1)

F=SMODE(0) G=BMODE(2) H=BMODE(1) I=BMODE(0)

J=SELAC1 K=SELXY L=ZSEL M=ENAC

ตารางที่ 5.3 แสดงสัญญาณการควบคุมโมดูล ALUPART3(ต่อจากหน้าที่ 52 และ 51)

5.7.4 ส่วนควบคุม program3 โมดูล

สัญญาณที่ได้จากการถอดรหัส ของโค้ดคำสั่งต่างๆ ที่เกี่ยวข้องแสดงดัง

ตารางที่ 5.4

ชุดคำสั่ง	สัญญาณ												
	1	2	3	4	5	6	7	8	9	A	B	C	D
LOAD RV,(X+K)	0	0	0	1	0	0	0	0	0	0	0	0	0
LOAD RV,(Y+K)	0	0	0	1	0	0	0	0	0	0	0	0	0
LOAD (Y+K),RV	0	0	0	1	0	0	0	0	0	0	0	0	0
LOAD (X+K),RV	0	0	0	1	0	0	0	0	0	0	1	1	0
IN RV	0	0	0	1	0	0	0	0	0	0	1	1	0
OUT RV	0	0	0	1	0	0	0	0	0	0	1	1	0
LOAD RV,(KK)	0	0	0	1	0	0	0	0	0	0	0	1	.0
LOAD (KK),AC	0	0	0	1	0	0	0	0	0	0	0	1	0
PUSH RV	0	0	1	0	0	0	0	1	0	0	1	0	1
POP RV	0	1	0	0	0	0	0	0	1	1	1	0	0
LOAD AC,(KK)	0	0	0	1	0	0	0	0	0	0	0	1	0
BRNZ	1	0	0	0	0	0	0	0	0	0	1	0	0
BRNNZ	1	0	0	0	0	0	0	0	0	0	1	0	0

ตารางที่ 5.4 แสดงสัญญาณควบคุมโมดูล PCGRAM3

	สัญญาณ												
ชุดคำสั่ง	1	2	3	4	5	6	7	8	9	A	B	C	D
BRNC	1	0	0	0	0	0	0	0	0	0	0	1	0
BRNNC	1	0	0	0	0	0	0	0	0	0	0	1	0
BRN	1	0	0	0	0	0	0	0	0	0	0	1	0
ICALL KK	1	0	1	0	0	0	0	0	0	0	1	0	1
CALLZ	1	0	1	0	0	0	0	0	0	0	1	0	1
CALLNZ	1	0	1	0	0	0	0	0	0	0	1	0	1
CALLC	1	0	1	0	0	0	0.	0	0	0	1	0	1
CALLNC	1	0	1	0	0	0	0	0	0	0	1	0	1
RET	1	1	0	0	0	1	0	0	1	0	1	0	0
RETZ	1	1	0	0	0	1	0	0	1	0	1	0	0
RETC	1	1	0	0	0	1	0	0	1	0	1	0	0
RETNC	1	1	0	0	0	1	0	0	1	0	1	0	0
RETI	1	1	0	0	1	0	0	0	1	0	1	0	0
LOAD X,KK	0	0	0	0	0	0	0	0	0	0	1	0	0
LOAD Y,KK			0	0	0	0	0	0	0	0	1	0	0

หมายเหตุ

1 = ATIVEJUMP 2= SELCLKCOUNT(1) 3 = SELCLKCOUNT(0)

4 = SEL_MEM 5 = SEL_FB(1) 6 = SEL_FB(0) 7= PCLODE

8 = SELPUSH 9=RET_CALL A = SELPOP B= SELMEM(1)

C= SELMEM(0) D=WERAM

ตารางที่ 5.4 แสดงสัญญาณควบคุมโมดูล PCGRAM3 (ต่อจากหน้า 53)

5.7.5 ส่วนควบคุมโมดูล CPUControl

ส่วนโมดูล CPU control เป็นส่วนที่ทำการกำเนิดสัญญาณ ไปควบคุมการทำงานของอุปกรณ์ภายนอก KC-I เช่น การอ่านข้อมูล การเขียนข้อมูล จากหน่วยความจำ การติดต่อ port ซึ่งเป็นสัญญาณ ที่ส่งได้จากการถอดรหัส คำสั่งแสดงดังตาราง (5.5) และมี สัญญาณบางสัญญาณที่ส่งไปควบคุมส่วนต่างๆภายใน KC-I ตัวโมดูล CPU Control เป็นส่วน ย่อยในโมดูล Controlpart ดังแสดงในบล็อกไดอะแกรม

ชุดคำสั่ง	สัญญาณ					
	IN1	IN0	WREQ	IORQ	WR	RD
LOAD RV,(X+K)	0	0	1	0	0	1
LOAD RV,(Y+K)	0	0	1	0	0	1
LOAD (Y+K),RV	0	0	1	0	1	0
LOAD (X+K),RV	0	0	1	0	1	0
IN RV,P	0	0	0	1	0	1
OUT P,RV	0	0	0	1	1	0
LOAD RV,(KK)	0	0	1	0	0	1
LOAD (KK),RV	0	0	1	0	1	0
LOAD AC,(KK)	0	0	1	0	0	1
DIN0	0	1	0	0	0	0
EIN0	0	1	0	0	0	0
DIN1	1	0	0	0	0	0
EIN1	1	0	0	0	0	0

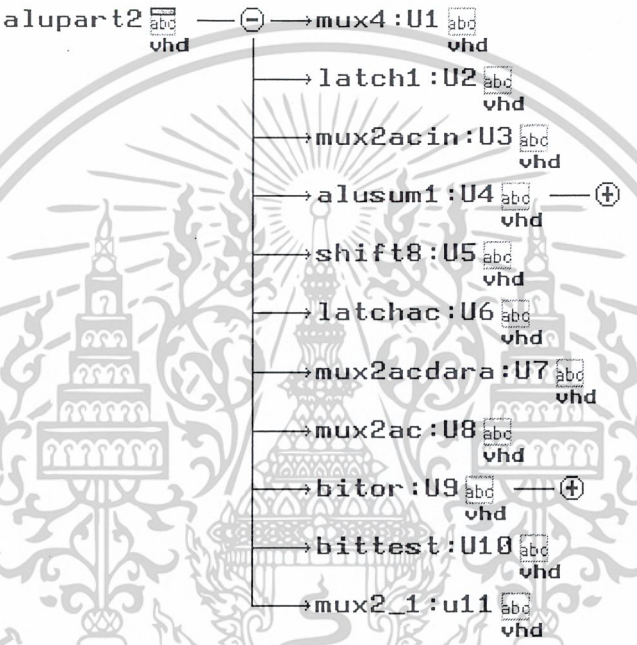
ตารางที่ 5.5 แสดงสัญญาณการควบคุมโมดูล CPUCONTROL

บทที่ 6 ผลการออกแบบ

จากการออกแบบโดยแยกออกเป็นโมดูลต่างๆ ซึ่งเขียนด้วยวีเอชดีแอล อธิบายพฤติกรรมการทำงานของแต่ละโมดูล สามารถที่จะแยกอธิบายรายละเอียด และโครงสร้างของแต่ละโมดูลที่เป็นโมดูลหลักที่สำคัญๆ ดังนี้

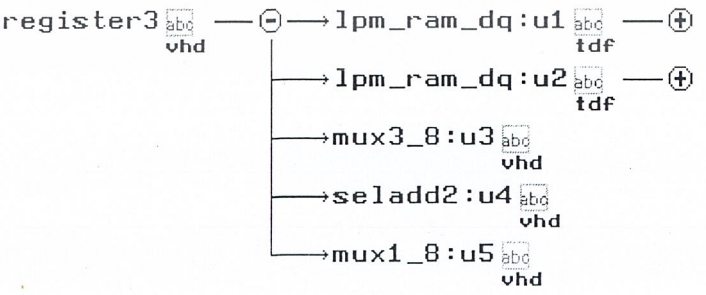
6.1 โมดูลเอแอลยูพาร์ต2 (ALUPART2)

สามารถแสดงโครงสร้างภายในโดยอยู่ในรูปของผังสำดับการเชื่อมต่อของแต่ละส่วนดังรูปที่ 6.1



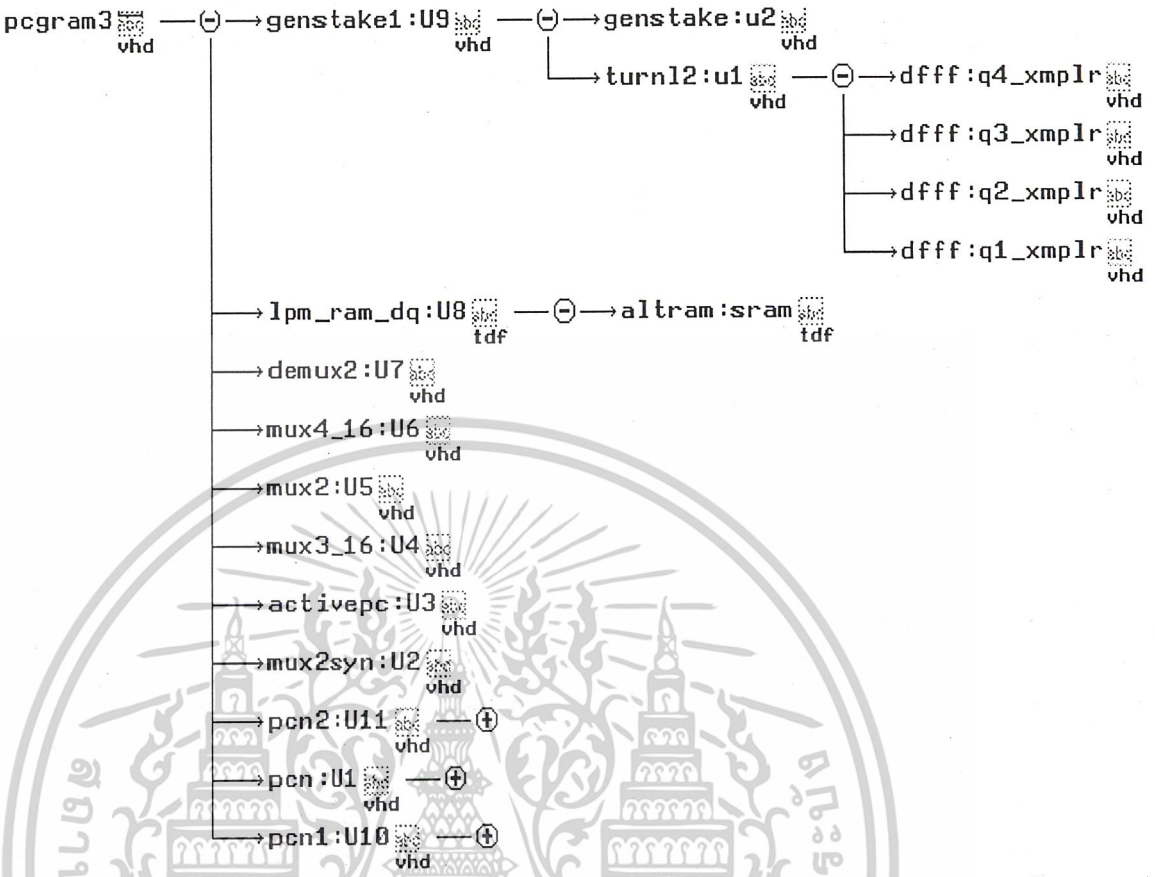
รูปที่ 6.1 แสดงผังการเชื่อมต่อส่วนประกอบต่าง ๆ ของโมดูลเอแอลยูพาร์ต 2

6.2 โมดูลรีจีเตอร์3 (Register3)



รูปที่ 6.2 แสดงผังการเชื่อมต่อส่วนประกอบต่าง ๆ ของโมดูลรีจีสเตอร์3

6.3 โมดูลพีซีแกรม3(Pcgram3)



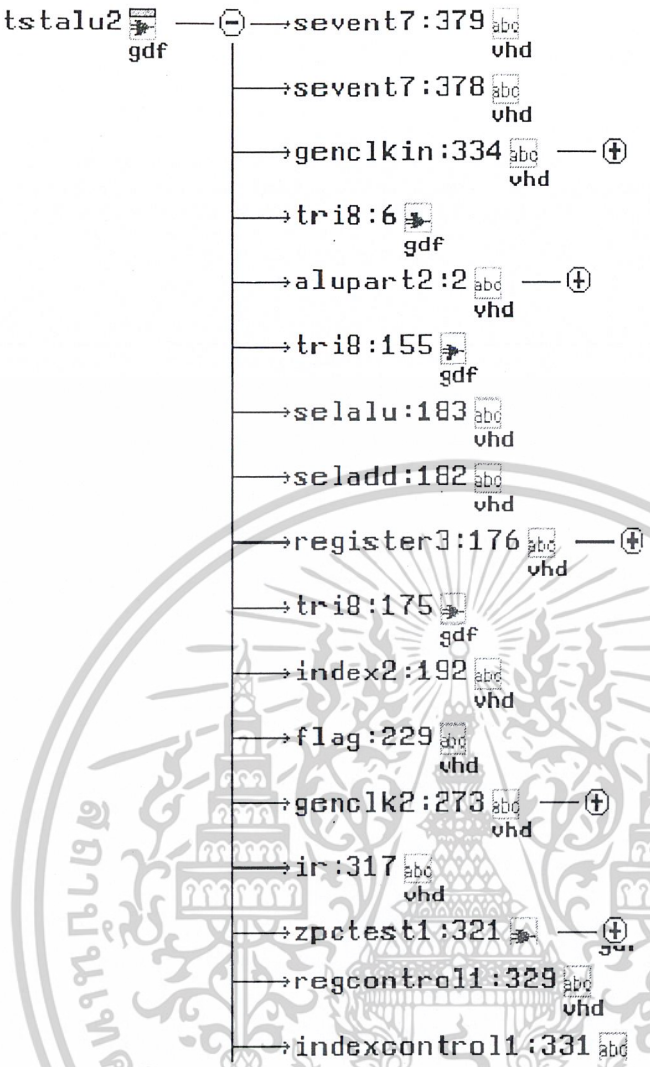
รูปที่ 6.3 แสดงผังการเชื่อมต่อส่วนประกอบต่าง ๆ ของโมดูลพีซีแกรม3

6.4 รายละเอียดการใช้ทรัพยากรภายในเอฟพีจีเอของแต่ละโมดูล รวมโมดูย่อยต่างๆ

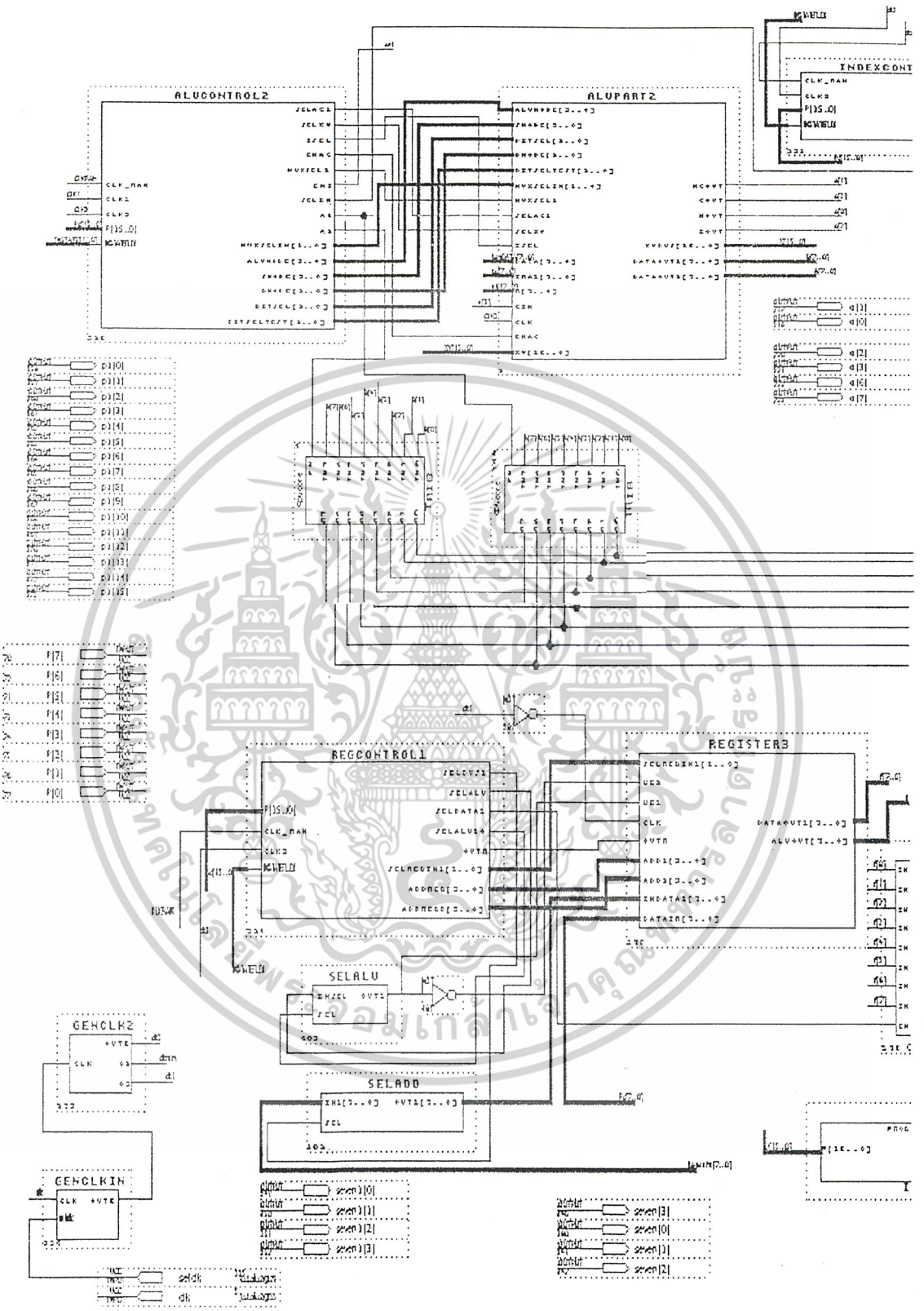
โมดูล	หน่วยความจำ(bit)	จำนวนเซลล์ที่ใช้(LC)	จำนวยเซลล์ที่ใช้(%)
ALUPART2	0	327	28
INDEX	0	48	4
REGISTER3	256	32	2
PCGRAM3	256	238	20
INDEXCONTR	0	13	1
ALUPARTCON	0	133	11
REGCONTR	0	67	5
PCCONTROL	1024	99	8

ตารางที่ 6.1 แสดงการใช้ทรัพยากรภายใน FPGA ของโมดูลต่างๆ

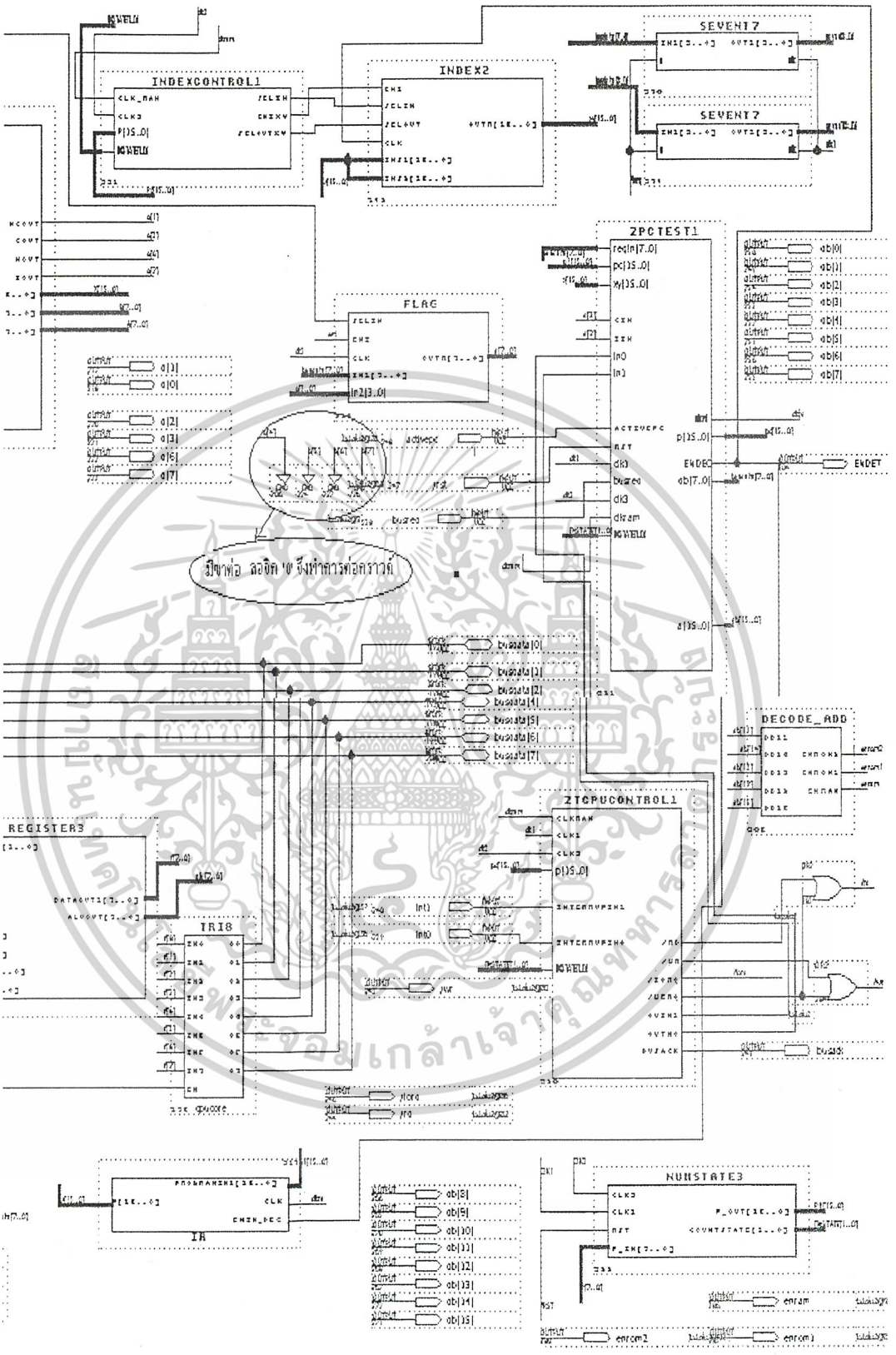
6.5 ผังวงจรรวมทั้งหมดที่ได้จากการเชื่อมต่อส่วนต่างๆ เข้าด้วยกัน



รูปที่ 6.4 แสดงผังการเชื่อมต่อของวงจรทั้งหมด



รูปที่ 6.5 แสดงภาพการเชื่อมต่อของโมดูลต่างๆ



รูปที่ 6.6 แสดงภาพการเชื่อมต่อของโมดูลต่างๆ ซีกทางขวาต่อจากภาพ 6. 5

Address bus	Pin of FPGA	Data Bus	Pin of FPGA
ADD0	141	DATA0	72
ADD1	138	DATA1	69
ADD2	136	DATA2	67
ADD3	133	DATA3	64
ADD4	131	DATA4	62
ADD5	128	DAYA5	59
ADD6	122	DATA6	51
ADD7	120	DATA7	48
ADD8	118	Control pin	Pin of FPGA
ADD9	116	/IORQ	86
ADD10	113	/RD	82
ADD11	111	/WR	80
ADD12	109	/WERQ	43
ADD13	101	/ACTIVE	28
ADD14	99	/RST	56
ADD15	97	IN0,IN1	36,37

ตารางที่ 6.2 แสดงรายละเอียดตำแหน่งขาของซีพียูที่สัมพันธ์กับตำแหน่งขาของเอฟพีจีเอ

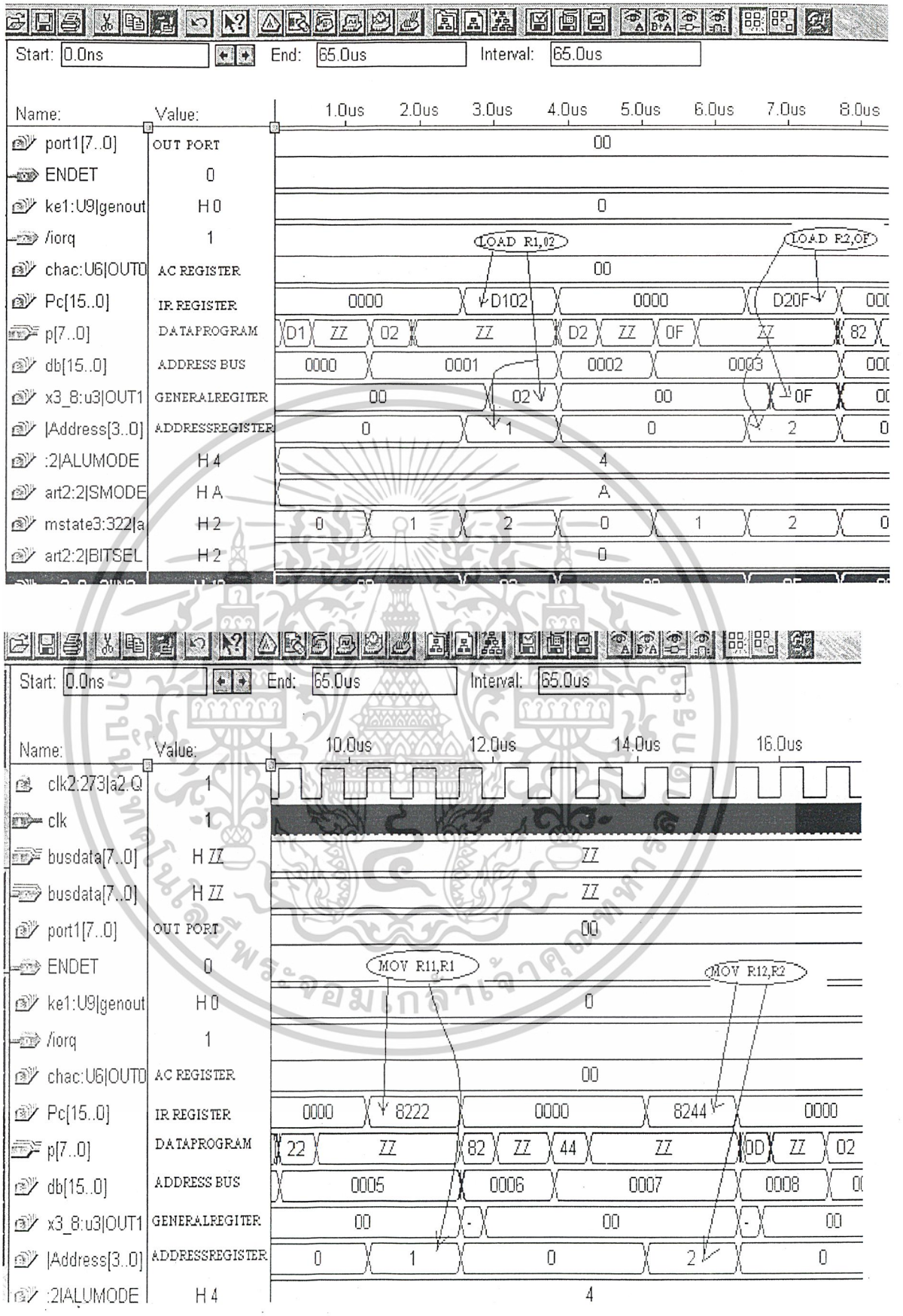
บทที่ 7 การทดลอง

การทดลองแบบเป็นสองระดับ คือ การซิมมูเลท และการทดลองโดยบอรด์ทอลองที่จดทำขึ้นโดยเฉพาะ ในการทดลอง ทางผู้จัดทำได้ทำการเขียน โปรแกรมเพื่อใช้ในการทดลองโดยเป็นการแสดงผลการทำงานทาง 7'segment และ แอลอีดี ในการซิมมูเลทได้ใช้ ซอฟ์ตแวร์ MAX+II

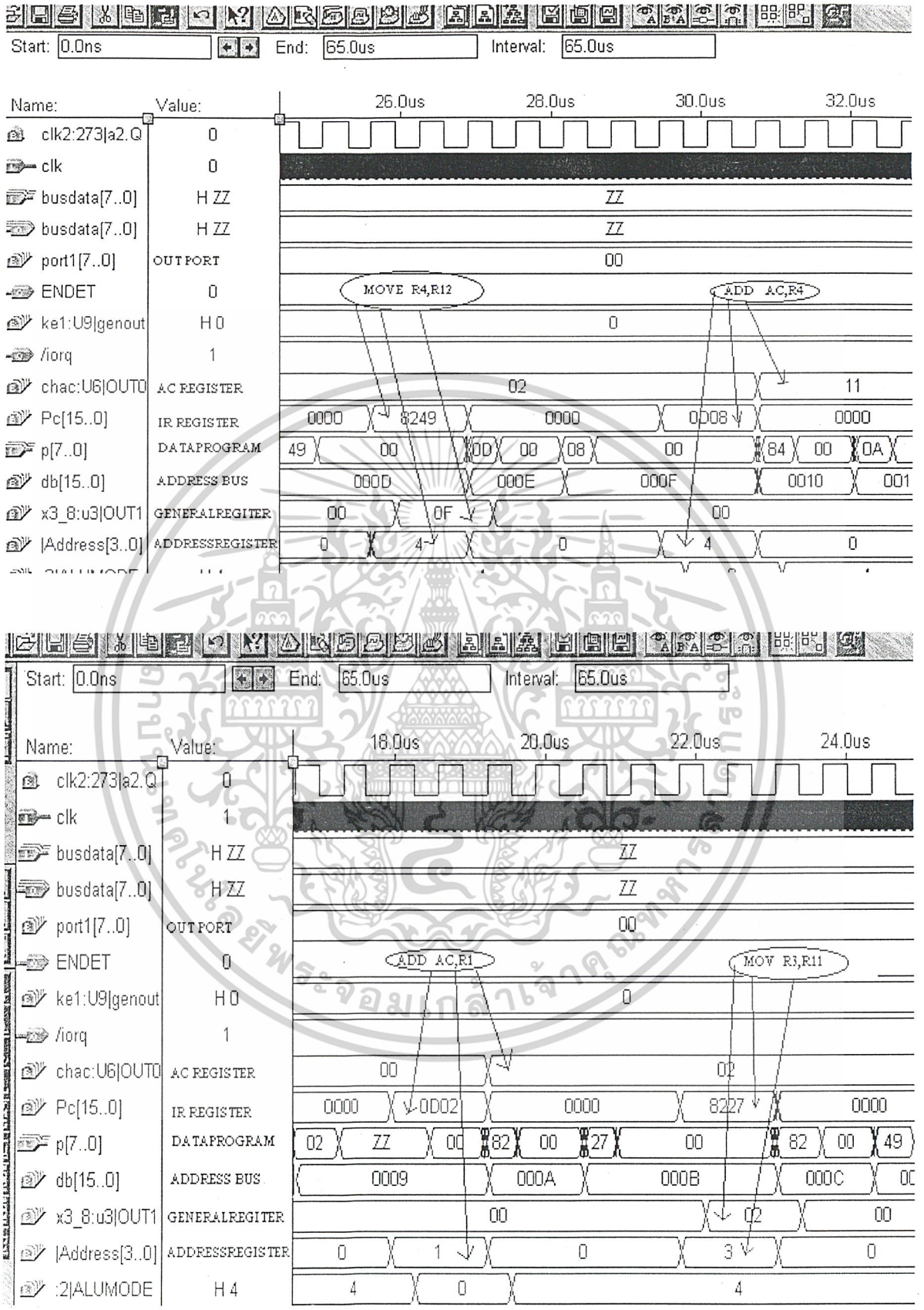
7.1 การซิมมูเลท(simulate)

7.1.1 โปรแกรมการทดสอบความถูกต้อง ของการ ทำงานทางคณิตศาสตร์ , ความถูกต้องของข้อมูลในรีจีสเตอร์ซึ่งผลของการทำงานจะให้ผลลัพท์แสดงผลยัง 7'segment เป็น 11H

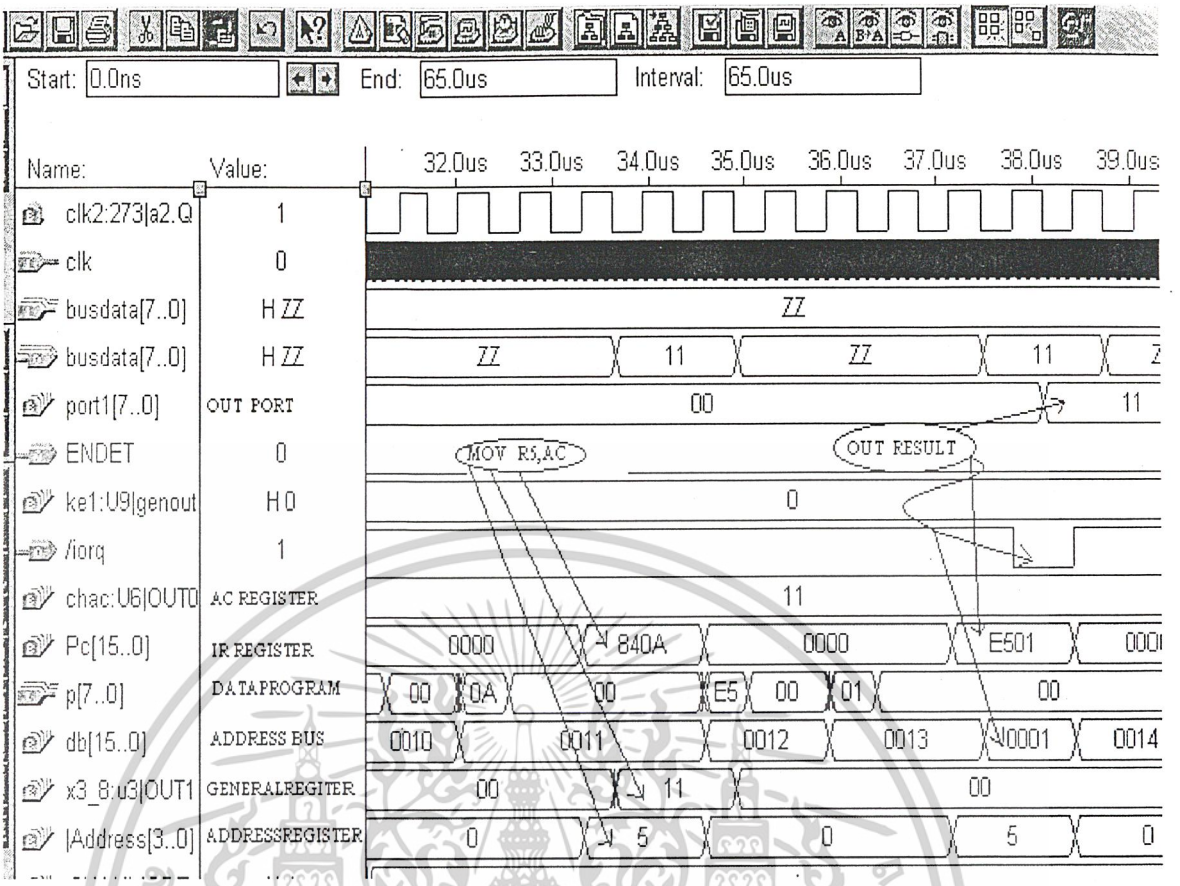
```
;TEST PROGRAM
;MNEMONIC                    ;OP-CODE
;START  0000H
        LOADI   R1,02H      D102H
        LOADI   R2,0FH      D20FH
        MOV     R11,R1      8222H
        MOV     R12,R2      8244H
        ADD     AC,R1       0D02H
        MOV     R3,R11      8227H
        MOV     R4,R12      8249H
        ADD     AC,R4       0D08H
        MOV     R4,AC       840AH
        OUT     (01H),R5    E501H
        END
```



รูปที่ 7.1(ก) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.1



รูปที่ 7.1(ข) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.1

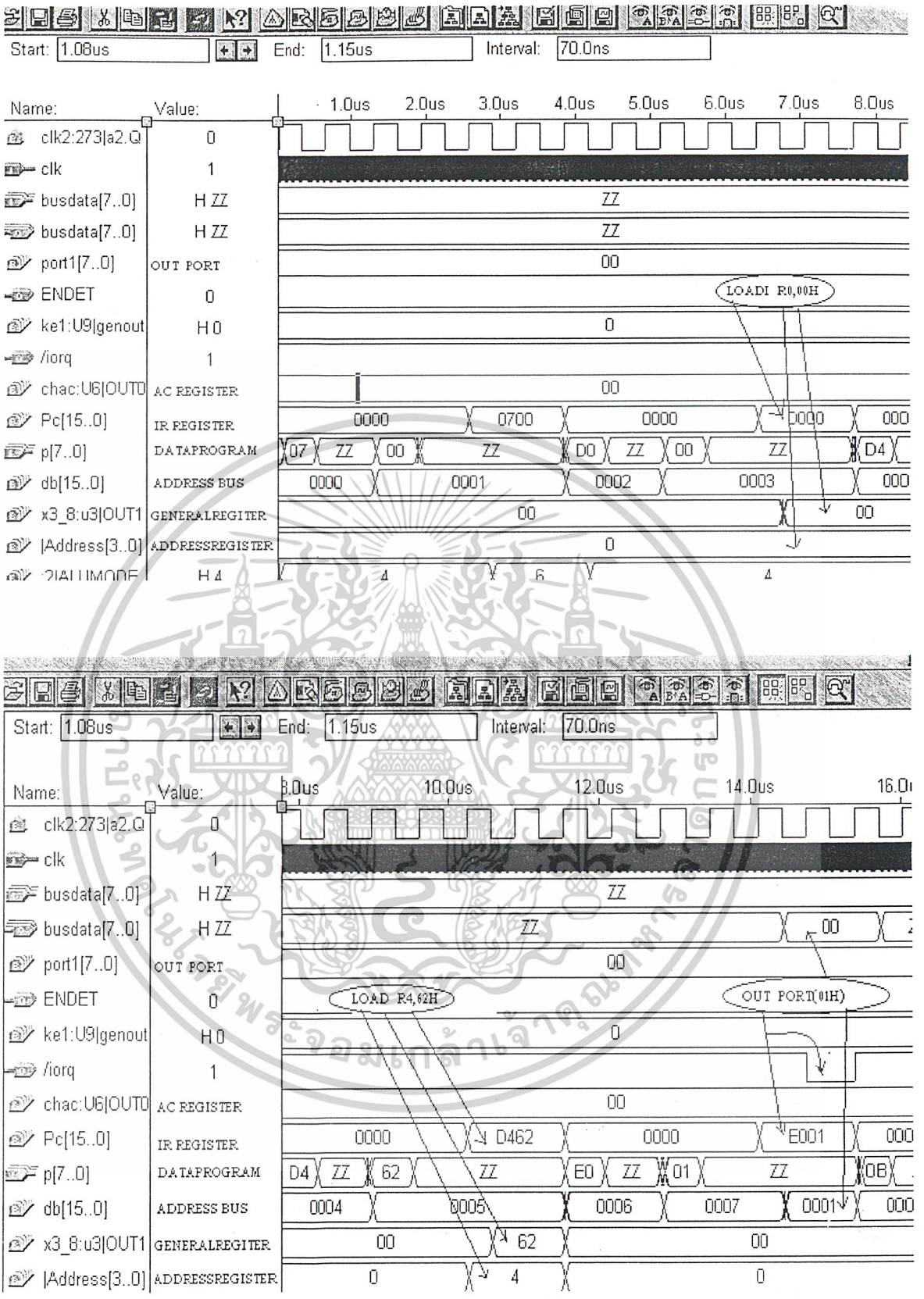


รูปที่ 7.1(ค) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.1

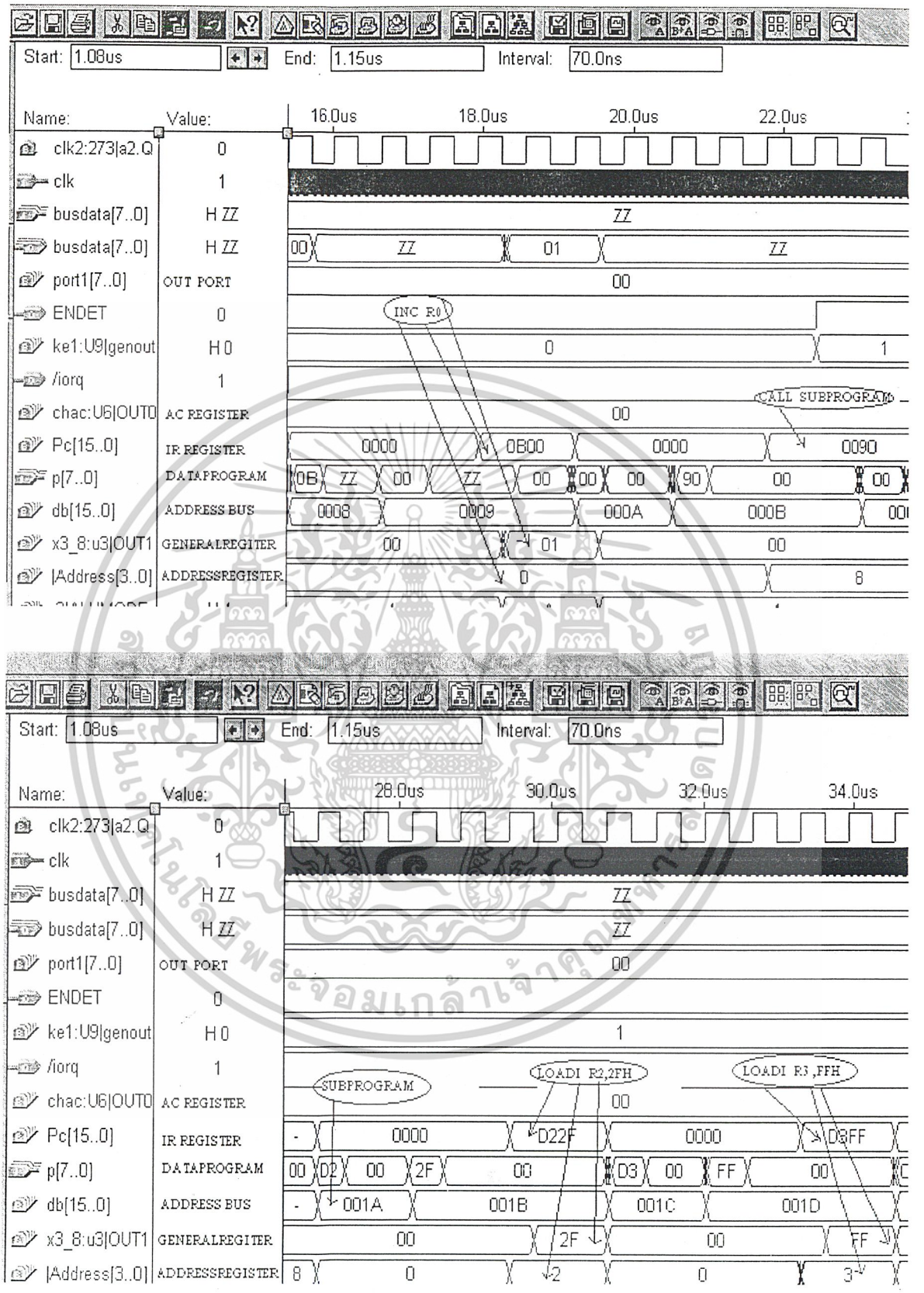
7.1.2 โปรแกรมการทดสอบความถูกต้องของแฟลก และคำสั่งกระโดด

เป็นโปรแกรมนับจำนวน 00H-62H วนไปเรื่อยจนกว่าจะทำการรีเซ็ต

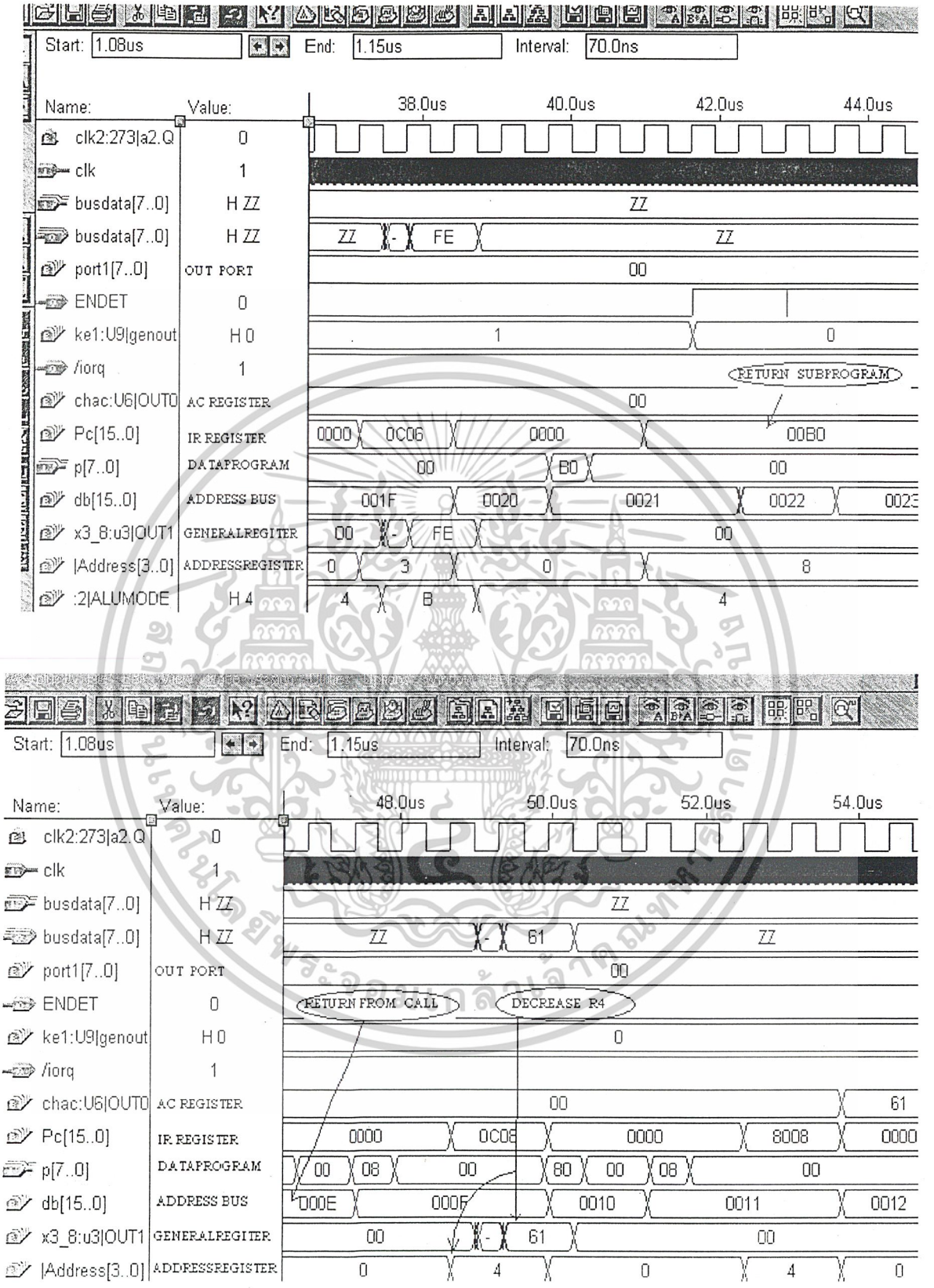
```
;COUNTER PROGRAM
;MNEMONIC                                   ;OP-CODE
;START  0000H
   LOOP0:ANDI     AC,00H                     0700H
         LOADI    R0,00H                     D000H
         LOADI    R4,62H                     D452H
   LOOP1:OUT      (01H),R0                   E001H
         INC      R0                         0B00H
         ICALL    LOOP2 ;CALL DELAY TIME     0090H  001AH
         DEC      R4                         0C08H
         MOV      AC,R4                      8008H
         BRANZ,LOOP0                         0001H  0000H
         BRN,LOOP1                           0005H  0006H
   LOOP2:LOADI    R2,2FH ;SUBPROGRAM         D22FH
   LOOP4:LOADI    R3,FFH                     D3FFH
   LOOP3:DEC      R3                         0C06H
         MOV      AC,R3                      8006H
         BRNNZ,LOOP3                         0002H  001EH
         DEC      R2                         0C04H
         MOV      AC,R2                      8004H
         BRNNZ ,LOOP4                        0002H  001CH
         RET                                 00B0H
;END
```



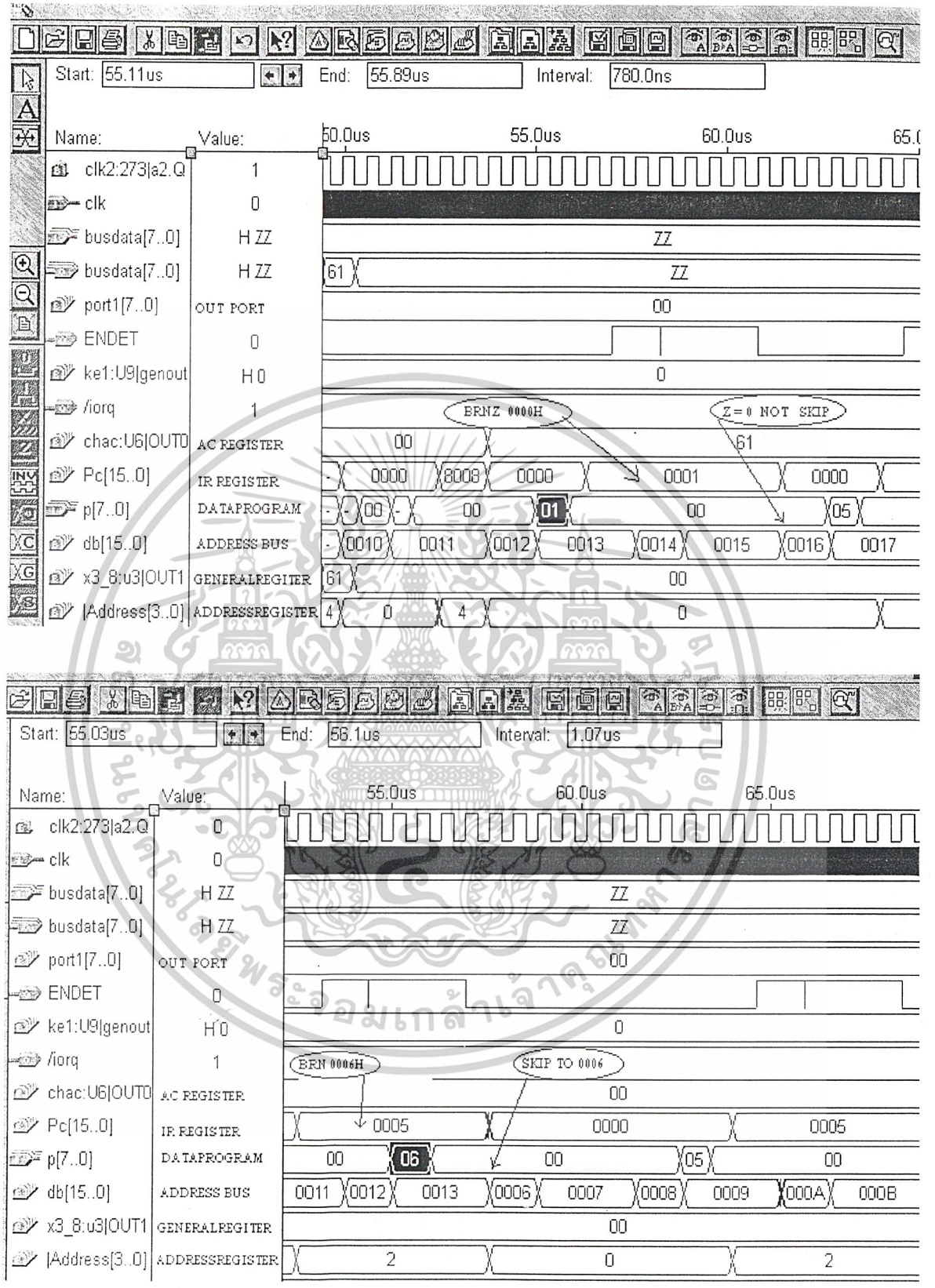
รูปที่ 7.2(ก) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.2



รูปที่ 7.2 (ข) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.2



รูปที่ 7.2 (ค)แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.2

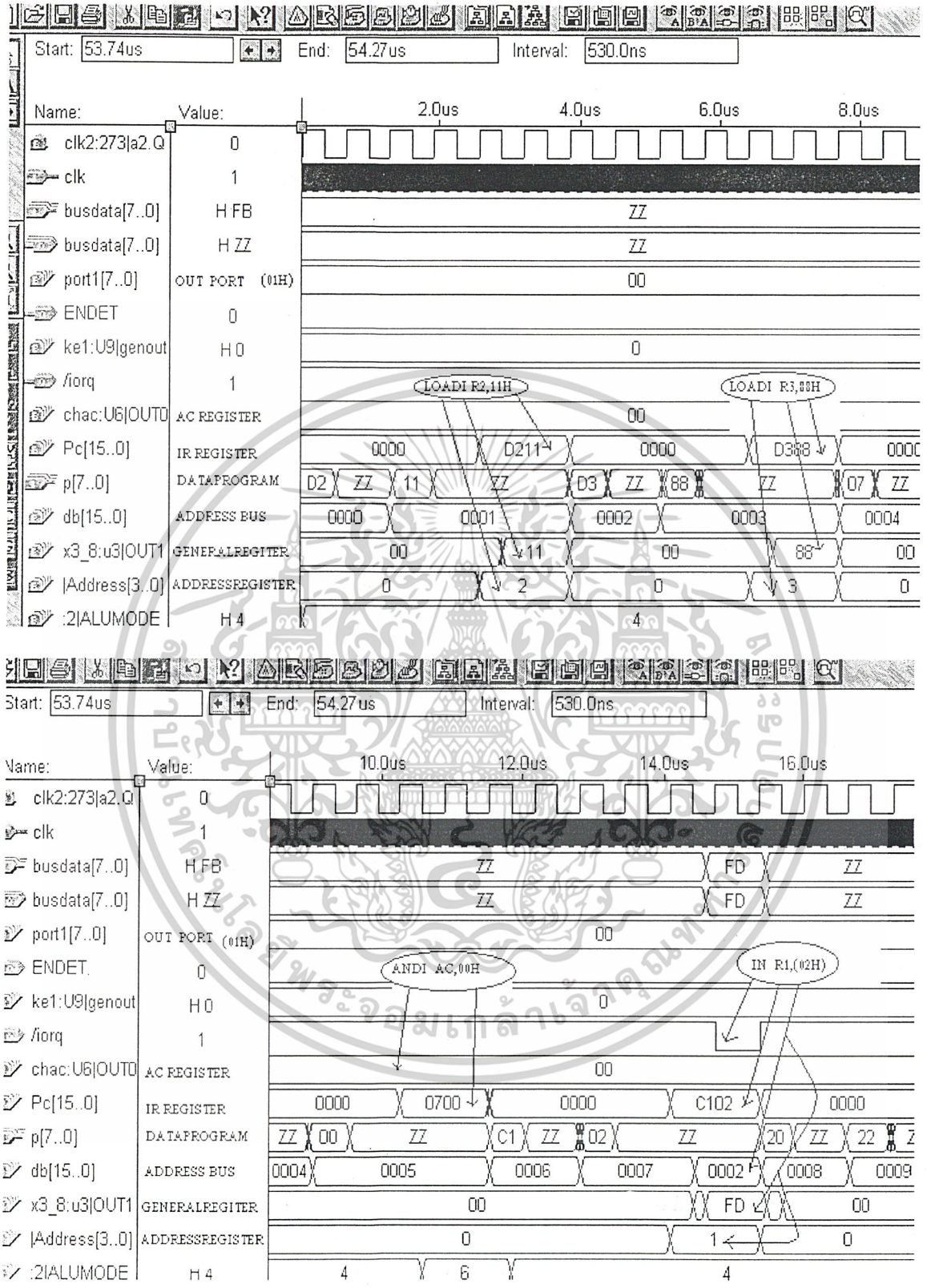


รูปที่ 7.2(ต) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.2

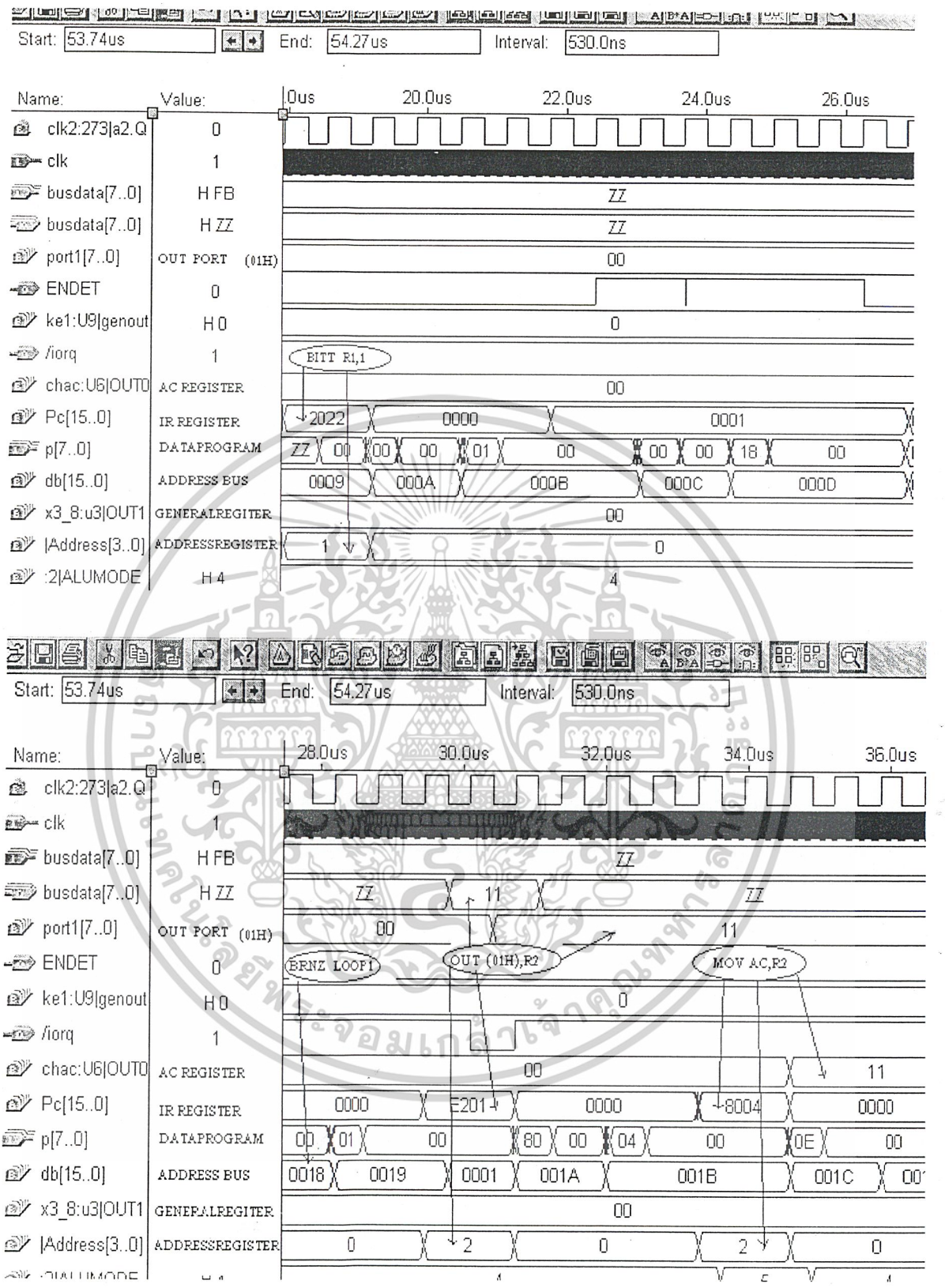
7.1.3 โปรแกรมควบคุมสเต็ปปิ้งมอเตอร์

สามารถควบคุมให้มอเตอร์หมุนซ้ายขวาโดยรับสัญญาณควบคุมจากสวิตซ์กดติดปล่อยดับ

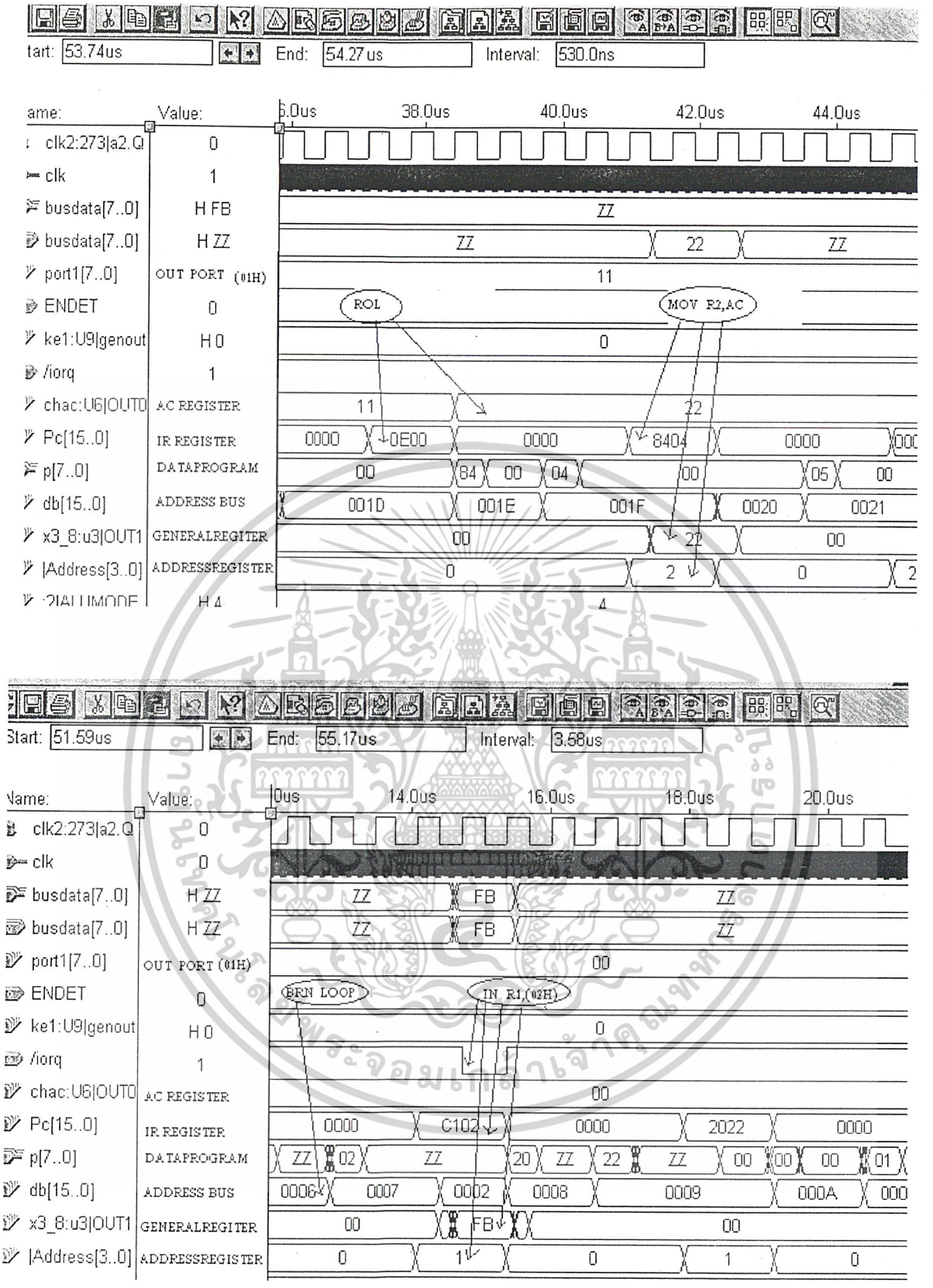
```
;STEPMOTOR CONTROL PROGRAM
;BEGINSTART (0000H)                    ;OP-CODE
        LOADI       R2,11H;           D211H
        LOADI       R3,88H;           D388H
        ANDI        AC,00H;           0700H
LOOP:   IN          R1,(02H);         C102H
        BITT        R1,1;             2022H
        BRNZ        LOOP1;            0001H  FOLLOW  BY  ADDRESS OF LOOP1
        BITT        R1,2;             2042H
        BRNZ  LOOP2;                  0001H  FOLLOW  BY ADDRESS OF LOOP2
        BRN                           0005H  FOLLOW  BY  ADDRESS  OF LOOP
LOOP1:OUT           (01H),R2;         E201H
        MOV         AC,R2;            8004H
        ROL  ;                        0E00H
        MOV         R2,AC;            8404H
        BRN  LOOP;                    0005H  FOLLOW  BY  ADDRESS  OF LOOP
LOOP2:  OUT         (01H), R3;        E301H
        MOV         AC,R3;            8006H
        ROR                           0E40H
        MOV         R3,AC             8406H
        BRN   LOOP ;                  0005 H  FOLLOW  BY  ADDRESS  OF LOOP
END ;
```



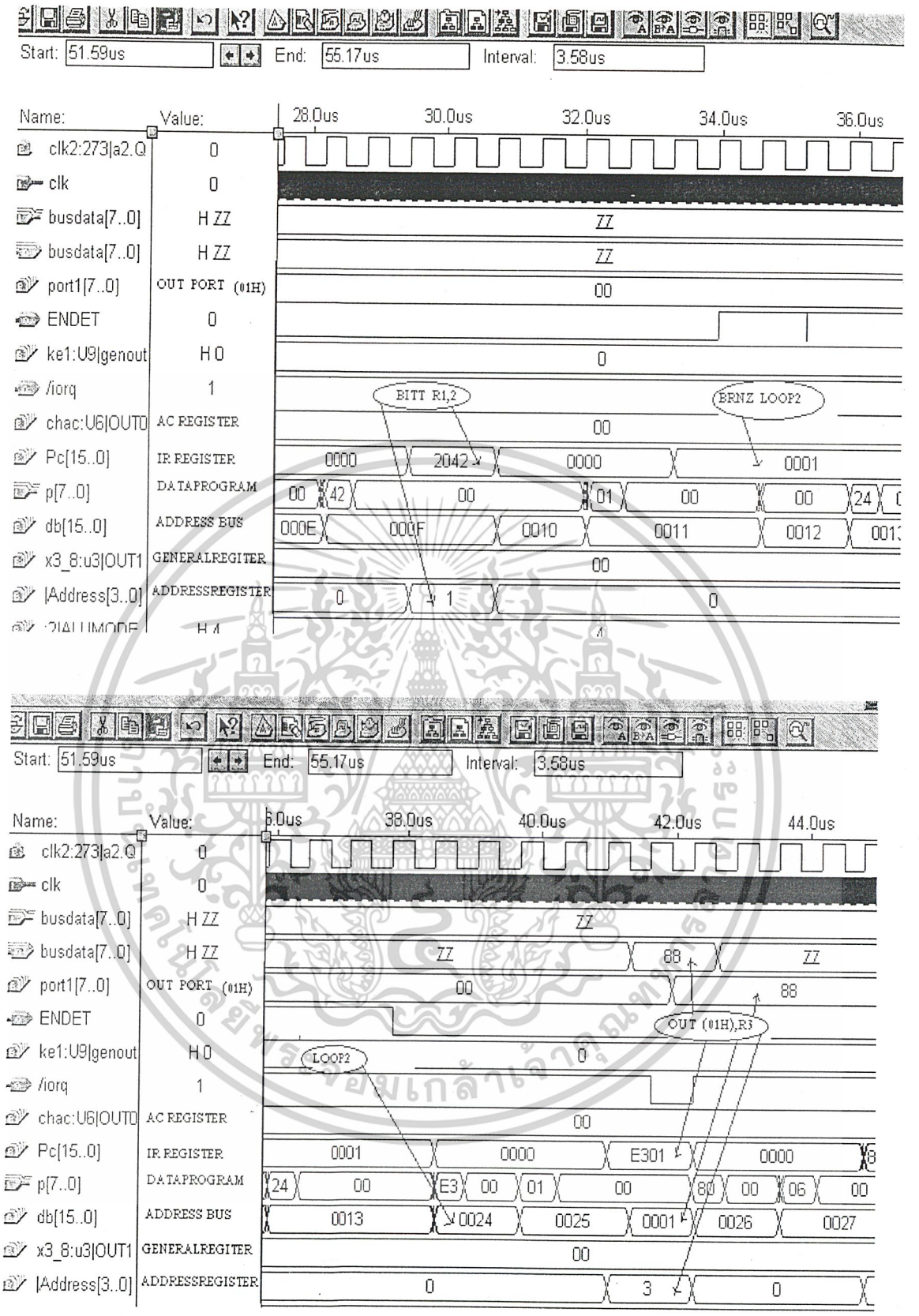
รูปที่ 7.3(ก) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.3



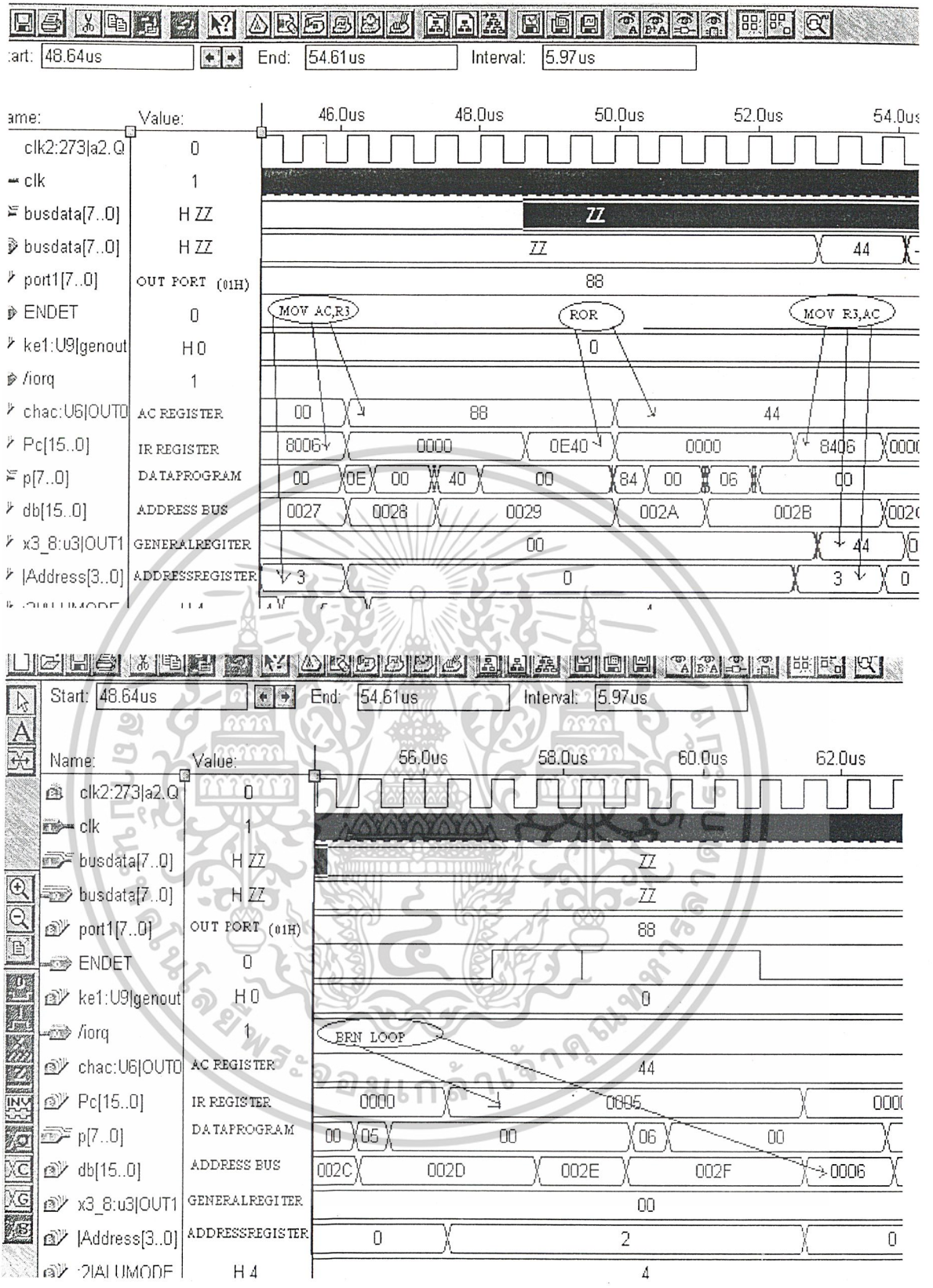
รูปที่ 7.3(ข) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.3



รูปที่ 7.3(ค) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.3



รูปที่ 7.3(ง) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.3



รูปที่ 7.3(จ) แสดงไทม์มิ่งไดอะแกรมการทำงานของโปรแกรมที่ 7.1.3

7.1.4 โปรแกรมควบคุมเด็อทเมตริกซ์

ในโปรแกรมนี้ทางคณะผู้จัดทำจะไม่แสดงผลการซิมมูเลทเนื่องจากผลการทดลองของโปรแกรมก่อนหน้านี้ ทำให้เชื่อได้ว่าชุดคำสั่งมีความถูกต้อง

```
;MOVING  ARROW
;BEGINSTART (0000H)
        ANDI        AC,00H;SET Z-FLAG
        LOADI       R0,FFH
        LOADI       R1,FBH
        LOADI       R2,F1H
        LOADI       R3,E0H
        LOADI       R7,01H
        LOADI       R8,02H
        LOADI       R9,04H
        LOADI       R10,08H
        LOADI       R11,10H
        LOADI       R12,20H
        LOADI       R13,40H
LOOPC:      LOADI       R4,24H;DEFINE DELAY TIME
LOOPA:LOADI     R14,64H;DEFINE DELAY TIME
LOOPB:OUT       (03H),R7
        OUT         (01H),R1
        DEC         R14
        MOV         AC,R14
        BRNNZ,      LOOPB
        DEC         R4
        MOV         AC,R4
        BRNNZ,      LOOPA
        LOADI       R4,24H
LOOPE:LOADI     R14,64H
LOOPF:OUT       (03H),R7
        OUT         (01H),R2
```

```
        OUT     (03H),R8
        OUT     (01H),R1
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOPF
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOPE

        LOADI   R4,24H
LOOPH:LOADI     R14,64H
LOOPI :OUT      (03H),R7
        OUT     (01H),R3
        OUT     (03H),R8
        OUT     (01H),R2
        OUT     (03H),R9
        OUT     (01H),R1
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOPI
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOPH

        LOADI   R4,24H
LOOPK:LOADI     R14,64H
LOOPL:OUT       (03H),R7
        OUT     (01H),R2
        OUT     (03H),R8
        OUT     (01H),R3
```

```
        OUT     (03H),R9
        OUT     (01H),R2
        OUT     (03H),R10
        OUT     (01H),R1
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOPL
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOPK

        LOADI   R4,24H
LOOPN:LOADI     R14,64H
LOOPO:OUT       (03H),R7
        OUT     (01H),R2
        OUT     (03H),R8
        OUT     (01H),R2
        OUT     (03H),R9
        OUT     (01H),R3
        OUT     (03H),R10
        OUT     (01H),R2
        OUT     (03H),R11
        OUT     (01H),R1
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOPO
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOPN
```

```
        LOADI       R4,24H
LOOPQ:LOADI         R14,64H
LOOPR:OUT           (03H),R7
        OUT         (01H),R2
        OUT         (03H),R8
        OUT         (01H),R2
        OUT         (03H),R9
        OUT         (01H),R2
        OUT         (03H),R10
        OUT         (01H),R3
        OUT         (03H),R11
        OUT         (01H),R2
        OUT         (03H),R12
        OUT         (01H),R1
        OUT         (01H),R0
        DEC         R14
        MOV         AC,R14
        BRNNZ,      LOOPR
        DEC         R4
        MOV         AC,R4
        BRNNZ,      LOOPQ

        LOADI       R4,24H
LOOPT:LOADI         R14,64H
LOOPU:      OUT             (03H),R8
        OUT         (01H),R2
        OUT         (03H),R9
        OUT         (01H),R2
        OUT         (03H),R10
        OUT         (01H),R2
        OUT         (03H),R11
        OUT         (01H),R3
```

```
        OUT     (03H),R12
        OUT     (01H),R2
        OUT     (03H),R13
        OUT     (01H),R1
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOPU
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOPT

        LOADI   R4,24H
LOOPW:LOADI     R14,64H
LOOPX:OUT       (03H),R9
        OUT     (01H),R2
        OUT     (03H),R10
        OUT     (01H),R2
        OUT     (03H),R11
        OUT     (01H),R3
        OUT     (03H),R12
        OUT     (01H),R3
        OUT     (03H),R13
        OUT     (01H),R2
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOPX
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOPW
```

```
        LOADI       R4,24H
LOOP2:LOADI         R14,64H
LOOP3:      OUT             (03H),R10
        OUT         (01H),R2
        OUT         (03H),R11
        OUT         (01H),R2
        OUT         (03H),R12
        OUT         (01H),R2
        OUT         (03H),R13
        OUT         (01H),R3
        OUT         (01H),R0
        DEC         R14
        MOV         AC,R14
        BRNNZ,      LOOP3
        DEC         R4
        MOV         AC,R4
        BRNNZ,      LOOP2

        LOADI       R4,24H
LOOP4:LOADI         R14,64H
LOOP5:      OUT             (03H),R11
        OUT         (01H),R2
        OUT         (03H),R12
        OUT         (01H),R2
        OUT         (03H),R13
        OUT         (01H),R2
        OUT         (01H),R0
        DEC         R14
        MOV         AC,R14
        BRNNZ,      LOOP5
        DEC         R4
```

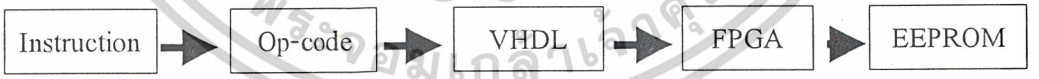
```
        MOV     AC,R4
        BRNNZ,  LOOP4

        LOADI   R4,24H
LOOP6:LOADI     R14,64H
LOOP7:      OUT         (03H),R12
        OUT     (01H),R2
        OUT     (03H),R13
        OUT     (01H),R2
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOP7
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOP6

        LOADI   R4,24H
LOOP8:LOADI     R14,64H
LOOP9:      OUT         (03H),R13
        OUT     (01H),R2
        OUT     (01H),R0
        DEC     R14
        MOV     AC,R14
        BRNNZ,  LOOP8
        DEC     R4
        MOV     AC,R4
        BRNNZ,  LOOP9

        LOADI   R4,24H
LOOPV:LOADI     R14,64H
LOOPZ:      OUT         (03H),R13
```

```
        OUT         (01H),R2
        OUT         (01H),R0
        DEC         R14
        MOV         AC,R14
        BRNNZ,      LOOPZ
        DEC         R4
        MOV         AC,R4
        BRNNZ,      LOOPV
        BRN,LOOPC ;RETURN TO START PROGRAM
END
```

7.2 การทดลองโดยใช้บอร์ดทดลอง

การทดลองขั้นนี้ได้ทำการต่อเชื่อม ซีพียูกับบอร์ดทดลองโดยมีส่วนแสดงผล 7'segment แสดงตำแหน่งของข้อมูล แสดงข้อมูลโปรแกรม และแสดงผลการทำงาน ส่วนของหน่วยความจำ EEPROM จำนวน 4 Kbyte และ RAM 32 Kbyte รวมทั้งแอลอีดี แสดงสถานะของแฟลกต่างๆ โดย การเชื่อมต่อระหว่างขาซีพียูกับบอร์ดทดลองใช้สายแพในการต่อเชื่อม

ในการทำการนำข้อมูลโปรแกรมเก็บใน EEPROM ได้ใช้ FPGA ช่วยในการนำข้อมูลใน EEPROM โดยได้ทำการแปลงข้อมูลอ๊อฟโค้ดของคำสั่งไปอยู่ในรูปของภาษา VHDL จากนั้นทำการโปรแกรมลง FPGA จากนั้นจึง ทำการต่อ EEPROM กับ FPGA แล้วทำการโปรแกรมข้อมูล จาก FPGA ลง EEPROM บล็อกไดอะแกรมด้านล่าง



รูปที่ 7.3 แสดงบล็อกไดอะแกรมการนำข้อมูลเก็บใน EEPROM

บทที่ 8 สรุปผลและวิจารณ์

8.1 การดำเนินงาน

ในการจัดทำปริญญานิพนธ์นี้เป็นการ ออกแบบซีพียูโดยใช้เอฟพีจีเอ ซึ่งได้นำโครงสร้างซีพียูขนาด 8 บิท(Z80) มาใช้ในการศึกษา โดยเริ่มจากการ ศึกษาลักษณะการทำงานของคำสั่งแต่ละคำสั่งของ Z80 รวมทั้งโครงสร้างภายใน จากนั้นจึงนำมาเขียนบล็อกไดอะแกรมเพื่อใช้ในการออกแบบซึ่งทำการแยกเป็นโมดูลย่อยๆ โดยแต่ละโมดูลทำการสร้างโดยใช้ภาษาวีเอชดีแอล ในการเขียน อธิบายการทำงานของแต่ละโมดูล ในการออกแบบได้ใช้ซอฟต์แวร์ MAX+II ในการ สังเคราะห์วงจรและซิมมูเลท จากนั้นจึงนำแต่ละโมดูล มาต่อร่วมกันเป็นระบบที่สมบูรณ์ แล้วจึงทำการซิมมูเลทอีกครั้งหนึ่ง เอฟพีจีเอที่ใช้คือ Flex10K20 ของบริษัท ALTERA ตลอดจนนำซีพียูที่ได้จากการออกแบบไปทดลองใช้งาน

8.2 การออกแบบและทดสอบ

ในการออกแบบได้เขียนบล็อกไดอะแกรมเพื่อให้ง่ายต่อการกำหนดโมดูลที่จะเขียนขึ้นในลักษณะของการออกแบบจากบนลงล่าง(TOP-DOWN DESIGN) โดยใช้ภาษาอธิบายพฤติกรรมการทำงานของแต่ละโมดูล คล้ายกับการออกแบบวงจรดิจิตอลทั่วไป แต่มีความง่ายและสะดวกกว่า โดยไม่ต้องคำนึงถึงฟังก์ชั่นการทำงาน ตัวชิฟเอฟพีจีเอที่ใช้ คือ Flex10k20 ซึ่งมีความจุเกท 20000 เกท และมีหน่วยความจำภายใน(RAM) ซึ่งตอนนี้ใช้ทรัพยากรภายในเอฟพีจีเอ ทั้งหมด 90 เปอร์เซ็นต์ และความถี่ที่ทำงานอยู่ไม่เร็วมากนักเนื่องจากในการออกแบบ โดยเอฟพีจีเอ เราไม่สามารถแก้ไขค่าเวลาหน่วงที่จุดต่าง ๆ ได้ จึงทำให้ ซีพียูทำงานช้า กว่าซีพียูขนาดเดียวกันที่ทำการออกแบบในลักษณะฟูลคัสตัม(Full custom) ในการทดลองกระทำสองขั้นคือ ขั้นการซิมมูเลท และทดลองกับบอร์ดทดลองที่จัดทำขึ้น เนื่องจากการซิมมูเลทเป็นการกำหนด ข้อมูลลงไปให้คอมพิวเตอร์ ทำการซิมมูเลทซึ่งที่ความถี่สูง ผลการซิมมูเลทยังมีความถูกต้อง แต่เมื่อนำมาทดลองกับบอร์ดทดลองซึ่งซีพียูต้องทำการอ่านข้อมูล จากหน่วยความจำ ภายนอกมาประมวลผล ทำให้ที่ความถี่สูง ข้อมูลที่อ่านได้มีความผิดพลาดซึ่งนี้เป็นข้อผิดพลาดที่ต้องทำการแก้ไขต่อไป

8.3 ปัญหาและอุปสรรคในการดำเนินงาน

ในการดำเนินงานออกแบบ ซีพียู ซึ่งเป็นการใช้ภาษาอธิบายลักษณะการทำงาน ของแต่ละโมดูลที่กำหนดขึ้นซึ่ง ส่วนใหญ่จะใช้คอมพิวเตอร์ ช่วยในการออกแบบตั้งแต่ การเขียน ภาษาวีเอชดีแอล จนถึงการทดสอบและนำข้อมูลที่ได้ลงเก็บในเอฟพีจีเอ ซึ่งในการนำข้อมูลลงเอฟพีจีเอจะต่อจากพอร์ตขนาน ของคอมพิวเตอร์ ซึ่งจากการทดลองพบว่า ช่วงที่ทำการนำข้อมูลลงเก็บใน

เอฟพีจีเอ แหล่งจ่ายจะถูกดึงกระแสมาก เพราะดูจากไฟที่แอลอีดี จะอ่อนลงขณะทำงานและสายที่ใช้ในการนำข้อมูลต้องยาวไม่มาก(ไม่ควรเกิน 1 เมตร) เพราะไม่สามารถทำการโหลดข้อมูลลงเอพีจีเอได้

และในกาสังเคราะห์วงจร ถ้าวงจรที่ออกแบบมีความซับซ้อนเครื่องคอมพิวเตอร์ จะต้องใช้หน่วยความจำมาก ดังนั้นขณะทำการสังเคราะห์วงจรอยู่ไม่ควร เปิดโปรแกรมอื่น ใช้งานพร้อมกันเพราะจะทำให้เครื่องหยุดทำงานไปเฉยๆ รวมทั้งความเร็วของซีพียูก็มีผลต่อความเร็วในการคอมไพล์ด้วย

8.4 บทวิจารณ์

ในการออกแบบ ซีพียู ก็คือการออกแบบวงจรดิจิตอลทั่วไปแต่ในการออกแบบซีพียูเป็นวงจรดิจิตอลขนาดใหญ่ ถ้าอาศัยการต่อวงจรด้วยอุปกรณ์ แบบเดี่ยวๆ คงไม่สามารถกระทำได้ แต่ด้วยเทคโนโลยีที่ก้าวหน้า ทำให้การออกแบบวงจรดิจิตอลที่มีขนาดใหญ่ ทำได้ง่ายขึ้น ยิ่งเป็นการใช้ภาษาอธิบายลักษณะการทำงานของอาร์ดแวร์ เช่น ภาษาวีเอชดีแอล ยิ่งทำให้การออกแบบทำได้ง่ายขึ้น โดยอาศัยลักษณะการออกแบบ จากบนลงล่าง คือเราสามารุกำหนดขอบเขต ของบล็อกการทำงานจากภายนอกก่อนว่ามันจะทำหน้าที่อะไร แล้วจึงค่อยลงไปในรายละเอียดของแต่ละบล็อกนั้นๆ จนถึงระดับล่างคือ ส่วนต่างๆ ที่จะใช้ในบล็อกหรือโมดูลนั้น ในการออกแบบครั้งนี้ทางผู้จัดทำได้ทำการแยกเป็นโมดูลย่อยๆ แล้วจึงนำแต่ละโมดูลมาต่อรวมกัน เป็นวงจรที่มีขนาดใหญ่และยุ่งยากมากขึ้น ซึ่งการออกแบบโดยการแบ่งส่วนย่อยๆจะ ทำง่ายขึ้น การเขียนโปรแกรมสามารถทำได้ง่ายขึ้น

ตัวซีพียูที่ได้จากการออกแบบประกอบด้วย 67 ชุดคำสั่ง ซึ่งการทำงานของทุกคำสั่งได้ทำการทดสอบหลายขั้นตอนตั้งแต่การทดสอบ ที่ละคำสั่ง ตลอดจนนำมาเขียนโปรแกรมทดลองใช้งาน จากการทดสอบถือได้ว่าทุกคำสั่งมีความถูกต้องในการทำงานเป็นที่หน้าพอใจ

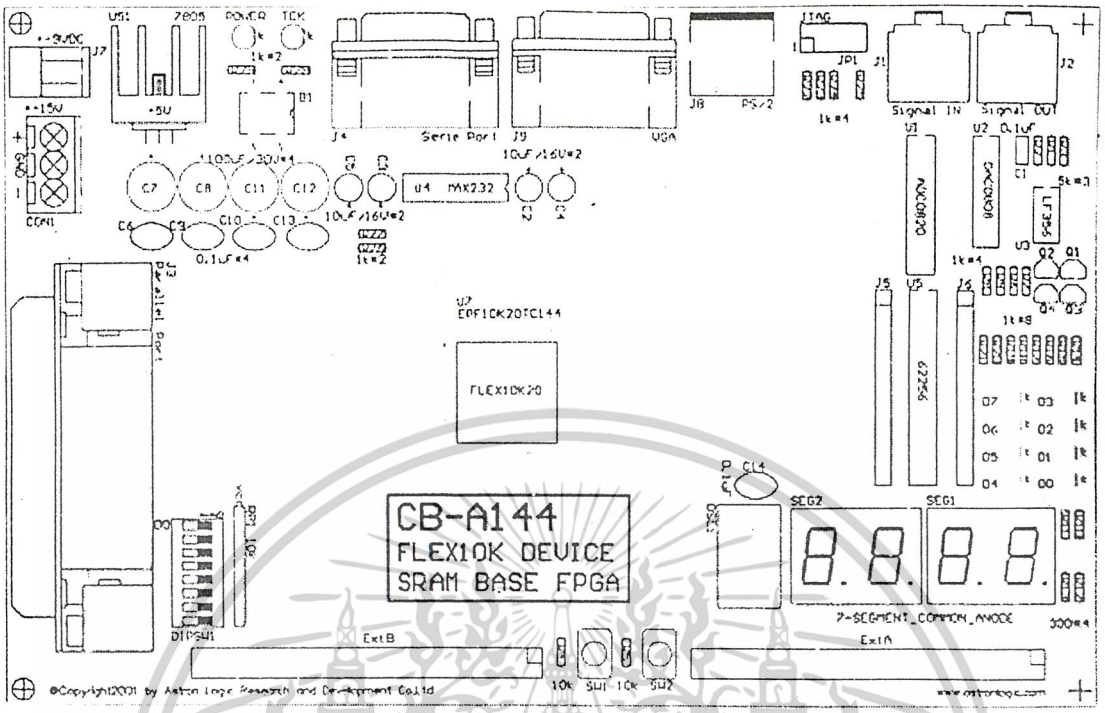
ภาคผนวก ก.

คู่มือใช้งาน MASTER FLEX –A01

บอร์ด MASTER FLEX – A01 ชุดนี้เป็นบอร์ดทดลอง FPGA ที่ออกแบบมาสำหรับใช้งานกับชิฟเอฟพีจีเอในตระกูล FLEX10K เบอร์ EPF10K20TC144 เป็นชิฟเอฟพีจีเอที่มีความจุของเกตประมาณ 20,000 เกต ทำให้สามารถออกแบบวงจรหรือระบบดิจิตอลที่มีขนาดใหญ่ หรือสลับซับซ้อนได้ดี อีกทั้งภายในบอร์ด MASTER FLEX-A01 ยังประกอบด้วยชุดอินเตอร์เฟสที่สำคัญอีกหลายตัวทำให้การออกแบบและใช้งานชิฟเอฟพีจีเอกับอุปกรณ์ภายนอกสามารถทำได้สะดวกมากยิ่งขึ้น

ภายในบอร์ด MASTER FLEX-A01 จะประกอบด้วย

- วงจรรวมตระกูล FLEX10K ในอนุกรม EPF10K20TC144
- JTAG CONNECTOR
- พอร์ตขยายช่องสัญญาณขนาด 26x2 2 ชุด
- คอนเน็กเตอร์สำหรับส่งข้อมูลอนุกรม แบบ DB9 ตัวผู้ 1 ชุด พร้อมไอซีตัวแปลงระดับสัญญาณดิจิตอลเป็นระดับของ RS-232
- คอนเน็กเตอร์สำหรับเชื่อมต่อกับจอ VGA
- คอนเน็กเตอร์แบบ PS/2
- คอนเน็กเตอร์แบบ CENTRONICS PORT สำหรับการส่งข้อมูลแบบขนาน
- ANALOG TO DIGITAL CONVERTER ขนาด 8 บิต
- DIGITAL TO ANALOG CONVERTER ขนาด 8 บิต
- หน่วยความจำแบบ SRAM 64 Kbyte
- สวิตซ์แบบกดติด - ปล่อยดับ 2 ตัว
- ดิฟสวิตซ์ 8 บิต 1 ชุด
- ไดโอดเปล่งแสง 8 ดวง
- ชุดแสดงผล 7- SEGMENT 4 หลักชนิด COMMOM ANODE แบบ MULTIPLEX
- โมดูลออสซิเลเตอร์ความถี่ 25.175 MHz
- คอนเน็กเตอร์แหล่งจ่ายไฟให้กับบอร์ดสำหรับไฟกระแสตรง +- 15 โวลท์



รูปที่ ก.1 ลักษณะการจัดวางอุปกรณ์ต่างๆ บนบอร์ด MASTER FLEX – A01

ตารางที่ ก.1 ความสัมพันธ์ระหว่างพอร์ตขยายช่องสัญญาณ Ext A และ Ext B กับตำแหน่งขาของ EPF10K20TC144

Ext A PIN NUMBER	PIN EPF10K20	Ext B PIN NUMBER	PIN EPF10K20
1	144(I/O,NCS)	1	72(I/O)
2	143(I/O,OS)	2	70(I/O)
3	142(I/O,NWS)	3	69(I/O)
4	141(I/O,NRS)	4	68(I/O)
5	140(I/O)	5	67(I/O)
6	138(I/O)	6	65(I/O)
7	137(I/O)	7	64(I/O)
8	136(I/O)	8	63(I/O)
9	135(I/O)	9	62(I/O)
10	133(I/O)	10	60(I/O)
11	132(I/O)	11	59(I/O)
12	131(I/O)	12	56(DED.INPUT)
13	130(I/O)	13	55(GLOBALCLK)
14	128(I/O,DEV_OE)	14	54(DED.INPUT)
15	126(Ded.Input)	15	51(I/O)
16	125(Global CLK)	16	49(I/O)
17	124(Ded.input)	17	48(I/O)
18	122(I/O.DEVCLR)	18	47(I/O)
19	121(I/O)	19	46(I/O)
20	120(I/O)	20	44(I/O)
21	119(I/O)	21	43(I/O)
22	118(I/O)	22	42(I/O)
23	117(I/O)	23	41(I/O)
24	116(I/O,DATA7)	24	39(I/O)
25	114(I/O,DATA6)	25	38(I/O)
26	113(I/O,DATA5)	26	37(I/O)
27	112(I/O,DATA4)	27	36(I/O)

Ext A PIN NUMBER	PIN EPF10K20	Ext A PIN NUMBER	PIN EPF10K20
28	111(I/O,DATA3)	28	33(I/O)
29	110I/O,DATA2)	29	32(I/O)
30	109(I/O,DATA1)	30	31(I/O)
31	102(I/O)	31	30(I/O)
32	101(I/O)	32	29(I/O)
33	100(I/O)	33	28(I/O)
34	99(I/O)	34	27(I/O)
35	98(I/O)	35	26(I/O)
36	97(I/O)	36	23(I/O)
37	96(I/O)	37	22(I/O)
38	95(I/O)	38	21(I/O)
39	92(I/O)	39	20(I/O)
40	91(I/O)	40	19(I/O)
41	90(I/O)	41	18(I/O)
42	89(I/O)	42	17(I/O)
43	88(I/O)	43	14(I/O,INTDONE)
44	87(I/O)	44	13(I/O)
45	86(I/O)	45	12(I/O)
46	83(I/O)	46	11(I/O,RDBUSY)
47	82(I/O)	47	10(I/O)
48	81(I/O)	48	9(I/O)
49	80(I/O)	49	8(I/O)
50	79(I/O)	50	7(I/O,CLKUSR)
51	78(I/O)	51	GND
52	73(I/O)	52	VCC

ตารางที่ ก.2 ความสัมพันธ์ระหว่างวงจรสื่อสารข้อมูลแบบอนุกรมกับตำแหน่งขาของ EPF10K20

Seerial Port Pin Nam	EPF10K20	Seerial Port Pin Nam	EPF10K20
RX	144(I/O.NCS)	TX	143(I/O,CS)

ตารางที่ ก.3 ความสัมพันธ์ระหว่างคอนเน็กเตอร์สำหรับเชื่อมต่อกับจอ VGA กับตำแหน่งขาของ EPF10K20TC144

VGA Port Pin Name	EPF10K20 Pin	VGA Port Pin Name	EPF10K20 Pin
R	235(I/O)	V-SYNC	140(I/O)
G	136(I/O)	H-SYNC	138((I/O)
B	137(I/O)	-	-

ตารางที่ ก.4 ความสัมพันธ์ระหว่างคอนเน็กเตอร์แบบ PS/2 กับ ตำแหน่งขาของ EPF10K20TC144

PS/2 PORT PIN	EPF10K20 PIN	PS/2 PORT PIN	EPF10K20 PIN
DATA	142(I/O,NWS)	CLK_PS2	141(I/O,NRS)

ตารางที่ ก.5 ความสัมพันธ์ระหว่างคอนเน็กเตอร์แบบ Centronics Port กับ ตำแหน่งขาของ EPF10K20TC144

Centronics Port	EPF10K20 Pin	Centronics Port	EPF10K20 Pin
DB0	7(I/O,CLKUSR)	/LE/CR	18(I/O)
DB1	8(I/O)	/INITIALIZE	19(I/O)
DB2	9(I/O)	/SLIN	20(I/O)
DB3	10(I/O)	/ERROR	21(I/O)
DB4	11(I/O,RDBUSY)	SLCT	22(I/O)
DB5	12(I/O)	PE	23(I/O)
DB6	13(I/O)	/ACK	27(I/O)
DB7	14(I/O),INTDONE)	BUSY	27(I/O)
/STROBE	17(I/O)	-	-

ตารางที่ ก.6 ความสัมพันธ์ระหว่างวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลกับตำแหน่งขาของ EPF10K20tc144

CIRCUIT PIN	EPF10K20 PIN	CIRCUIT PIN	EPF10K20 PIN
Clock200k	102(I/O)	ADC4	97(I/O)
ADC0	101(I/O)	ADC5	96(I/O)
ADC1	100(I/O)	ADC6	95(I/O)
ADC2	99(I/O)	ADC7	92(I/O)
ADC3	98(I/O)	-	-

ตารางที่ ก.7 ความสัมพันธ์ระหว่างวงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อกกับตำแหน่งขาของ EPF10K20TC144

Circuit Pin Name	EPF10K20 Pin	Circuit Pin Name	EPF10K20 Pin
DAC0	91(I/O)	DAC4	87(I/O)
DAC1	90(I/O)	DAC5	86(I/O)
DAC2	89(I/O)	DAC6	83(I/O)
DAC3	88(I/O)	DAC7	82(I/O)

ตารางที่ ก.8 ความสัมพันธ์ระหว่างวงจรหน่วยความจำกับตำแหน่งขาของ EPF10K20TC144

Circuit Pin Name	EPF10K20 Pin	Circuit Pin Name	EPF10K20 Pin
/OE	41(I/O)	AD11	64(I/O)
/WE	42(I/O)	AD10	65(I/O)
DM7	43(I/O)	AD9	67(I/O)
DM6	44(I/O)	AD8	68(I/O)
DM4	47(I/O)	AD6	70(I/O)
DM3	48(I/O)	AD5	72(I/O)
DM2	49(I/O)	AD4	73(I/O)
DM1	51(I/O)	AD3	78(I/O)
DM0	59(I/O)	AD2	79(I/O)
AD14	60(I/O)	AD1	80(I/O)
AD13	62(I/O)	AD0	81(I/O)
AD12	63(I/O)	-	-

ตารางที่ ก.9 ความสัมพันธ์ระหว่างสวิตช์กดติด- ปล่อยดับ กับตำแหน่งขาของ EPF10K20TC144

Circuit Pin Name	EPF10K20 Pin	Circuit Pin Name	EPF10K20
SW1	56(DED.INPUT)	SW2	54(DED.INPUT)

ตารางที่ ก.10 ความสัมพันธ์ระหว่างดิฟสวิตช์ 8 บิต กับตำแหน่งขาของ EPF10K20TC144

Circuit Pin Name	EPF10K20 Pin	Circuit Pin Name	EPF10K20 Pin
DSW0	28(I/O)	DSW4	32(I/O)
DSW1	29(I/O)	DSW5	33(I/O)
DSW2	30(I/O)	DSW6	36(I/O)
DSW3	31(I/O)	DSW7	37(I/O)

ตารางที่ ก.11 ความสัมพันธ์ระหว่างไดโอดเปล่งแสงกับตำแหน่งขาของ EPF10K20TC144

Circuit Pin Name	EPF10K20 Pin	Circuit Pin Name	EPF10K20 Pin
LED0	120(I/O)	LED4	130(I/O)
LED1	121(I/O)	LED5	131(I/O)
LED2	122(I/O,DEVCLR)	LED6	132(I/O)
LED3	128(I/0,DEVOE)	LED7	133(I/O)

ตารางที่ ก.12 ความสัมพันธ์ระหว่างวงจรแสดงผล 7 – Segment 4 หลักกับตำแหน่งขาของ EPF10K20TC144

Circuit Pin Name	EPF10K20 Pin	Circuit Pin Name	EPF10K20 Pin
SEG.A	109(I/O,DATA1)	SEG.G	116(I/O,DATA7)
SEG.B	110(I/O,DATA2)	SEG.DOT	117(I/O)
SEG.C	111(I/O,DATA3)	CM1	118(I/O)
SEG.D	112(I/O,DATA4)	CM2	119(I/O)
SEG.E	113(I/O,DATA5)	CM3	39(I/O)
SEG.F	114(I/O,DATA60	CM4	38(I/O)
OSC	55(GLOBAL CLK)		

กิตติกรรมประกาศ

ขอบพระคุณบิดามารดาเป็นที่ตั้ง ขอบพระคุณท่านอาจารย์ทุกท่านที่ได้ถ่ายทอดแนะแนวทางการศึกษา ขอบคุณพี่ ขอบใจน้อง ขอบใจเพื่อน ขอบใจตัวเอง

ตลอดทั้งทุกท่านที่ยังไม่ได้เอ่ยนามในที่นี้ที่มีส่วนช่วยให้ปริญญานิพนธ์นี้สำเร็จด้วยดี ทางคณะผู้จัดทำต้องขอขอบพระคุณไว้ ณ. โอกาสนี้

หนังสืออ้างอิง

1. วัชรากร หนูทอง,บรรจง ปียธำรง,การออกแบบไมโครคอนโทรเลอร์โดยใช้ FPGA , ฉบับที่ 59 ,2542 หน้าที่ 67-71.

2.Douglas L.perry, VHDL, McGRAW-Hill,463 p.,1998.

3.James O.Hamblen,Rapid Prototyping of digital systems, United States of America, 2000,pp.120-132.

4.U Heinkel ,M Padeffke,VHDL,John Willey & Son,.LTD,420p.,1998.

5.Zanalabedin Navabi,VHDL Analysis and Modeling of digital systems, McGraw-Hill,632p.,1998.