

การออกแบบและสร้างไมโครคอนโทรลเลอร์โดยใช้เอฟพีจีเอ

DESIGN AND IMPLEMENTATION OF MICROCONTROLLER USING
FPGA

วัชรกร หนูทอง
WATCHARAKON NOOTHONG

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาค้นคว้าหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2542
ISBN 974-622-587-1

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การออกแบบและสร้างไมโครคอนโทรลเลอร์โดยใช้เอฟพีจีเอ

DESIGN AND IMPLEMENTATION OF MICROCONTROLLER USING
FPGA



วัชรกร หนูทอง

WATCHARAKON NOOTHONG

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2542

ISBN 974-622-587-1

เลขหมู่.....
เลขทะเบียน.....34566
วัน, เดือน, ปี 16 พ.ย. 2542

**DESIGN AND IMPLEMENTATION OF MICROCONTROLLER USING
FPGA**

WATCHARAKON NOOTHONG

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUTS INSTITUTE OF TECHNOLOGY LADKRABANG**

1999

ISBN 974-622-587-1

COPYRIGHT 1999

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

หัวข้อวิทยานิพนธ์	การออกแบบและสร้างไมโครคอนโทรลเลอร์โดยใช้เอฟพีจีเอ
นักศึกษา	นายวัชรกร หนูทอง
รหัสประจำตัว	39061079
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2542
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.บรรจง ปิยะธำรง

บทคัดย่อ

วิทยานิพนธ์นี้เป็นการเสนอขั้นตอนการออกแบบและสร้างไมโครคอนโทรลเลอร์ต้นแบบโดยใช้เอฟพีจีเอ โดยกำหนดให้มีสถาปัตยกรรมภายในต่างๆ ดังต่อไปนี้ เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ที่มีสถาปัตยกรรมภายในแบบฮาร์ดแวร์ โดยมีขนาดคำสั่งกว้าง 14 บิต มีชุดคำสั่ง 36 คำสั่ง โดยทุกคำสั่งจะถูกเอนคีย์ด้วยตัวภายใน 1 ไชเคิล ยกเว้นกลุ่มคำสั่งบรานซ์ที่ต้องใช้เวลา 2 ไชเคิล มีหน่วยคำนวณและตรรกะขนาด 8 บิต มีโหมดการอ้างอิงตำแหน่งที่อยู่ของข้อมูล 4 โหมด คือ การกำหนดตำแหน่งที่อยู่แบบทันทีทันใด การกำหนดตำแหน่งที่อยู่แบบโดยตรง การกำหนดตำแหน่งที่อยู่แบบทางอ้อม การกำหนดตำแหน่งที่อยู่แบบสัมพัทธ์ มีหน่วยความจำโปรแกรมที่อยู่ภายนอกเป็นแบบอีพรอมขนาด 8 กิโลไบต์ มีเรจิสเตอร์ฟังก์ชันพิเศษ 7 ตัว มีสเตจเรจิสเตอร์ความลึก 8 ระดับ ใช้แรมเป็นเรจิสเตอร์ไฟล์ขนาด 32 ไบต์ มีไอโอพอร์ตแบบขนานสองทิศทางจำนวน 2 พอร์ต คือ พอร์ตเอขนาด 5 บิตและพอร์ตบีขนาด 8 บิต ในการออกแบบรูปแบบของระบบจะใช้วิธี “การออกแบบจากบนลงล่าง” โดยจะใช้ภาษาวีเอชดีแอล (VHDL) ในการเขียนรูปแบบ จำลองการทำงานและแก้ไขตรวจสอบ แล้วนำมาทำการสังเคราะห์ให้อยู่ในรูปของผังวงจรระดับเกต แล้วนำไปสร้างลงบนเอฟพีจีเอ เพื่อนำมาทดสอบการทำงานจริง

Thesis Title	Design and Implementation of Microcontroller Using FPGA
Student	Mr. Watcharakon Noothong
Student ID.	39061079
Degree	Master of Engineering in Electrical Engineering
Programme	Electrical Engineering
Year	1999
Thesis Advisor	Assis.Prof.Bunjong Piyatamrong

ABSTRACT

This thesis presents a microcontroller design by using FPGA (Field Programmable Gate Array). The architecture of microcontroller specification is as follows : 8-bits microcontroller, that has the Harvard architecture, a 14-bit wide instruction word, 36 instruction sets which are executed in a single-cycled except for program branches that require 2 cycles, 8-bit ALU (Arithmetic Logic Unit), 4 data addressing mode (Immediate, Direct, Indirect, Relative), 8K x 14 EPROM (Ultraviolet - Erasable Programmable Read Only Memory) external program memory and 7 special registers, 8-level deep stack registers, it use RAM as file registers that have 32x8 bytes and 2 input/output ports (port A 5 bits, port B 8 bits). The design will uses “Top-Down Design” which uses VHDL language for modeling, simulation, debugging and synthesis to schematic diagram. At the gate level, it can be placed and routed on the FPGA in order to implement the real working digital system.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ดี ด้วยคำแนะนำและคำปรึกษาและคำปรึกษาเกี่ยวกับการเขียนโปรแกรมและการทดสอบ ซึ่งผู้จัดทำขอขอบพระคุณแก่บุคคลดังต่อไปนี้

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ บรรจง ปิยะธำรง ที่ช่วยเหลือให้คำแนะนำและช่วยเหลือทุกๆ อย่างในการทำวิทยานิพนธ์

ขอขอบพระคุณ คุณมนูญ ชินการณ และ คุณวิริยะ เพียรพิริยะ จากบริษัทอาร์แอนด์ดี คอมพิวเตอร์จำกัด (R & D Computer System Inc.) ที่ให้คำแนะนำเกี่ยวกับรายละเอียดของสถาปัตยกรรมพีไอซีไมโครคอนโทรลเลอร์ที่ออกแบบ

ขอขอบพระคุณ คุณชำนาญ ปัญญาโส และพนักงานใน TMEC ทุกท่านที่อำนวยความสะดวกในการจัดพิมพ์รูปเล่มและข้อเสนอแนะต่างๆ

ขอขอบพระคุณ บิดา มารดา และ พี่น้องทุกๆ ท่าน ที่ให้การสนับสนุนและกำลังใจมาตลอด

สุดท้ายขอขอบคุณมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร (C&C) และบัณฑิตวิทยาลัย ที่ได้ให้ทุนสนับสนุนการทำวิทยานิพนธ์ครั้งนี้

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้จัดทำขอบแต่ผู้มีพระคุณทุกท่าน

วัชรกร หนูทอง

สารบัญ

	หน้า
บทคัดย่อภาษาไทย.....	I
บทคัดย่อภาษาอังกฤษ.....	II
กิตติกรรมประกาศ.....	III
สารบัญ.....	IV
สารบัญรูป.....	VIII
สารบัญตาราง.....	XI
บทที่ 1 บทนำ.....	1
1.1 ความเป็นมาและความสำคัญของปัญหา.....	1
1.2 ความมุ่งหมายและวัตถุประสงค์ของงานวิจัย.....	2
1.3 ขอบเขตของงานวิจัย.....	2
1.4 รายละเอียดของวิทยานิพนธ์.....	3
บทที่ 2 ภาษาวีเอชดีแอล.....	4
2.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล.....	4
2.2 ส่วนประกอบต่างๆ ของภาษาวีเอชดีแอล.....	6
2.2.1 หน่วยการออกแบบเอนทิตี.....	6
2.2.2 หน่วยการออกแบบสถาปัตยกรรม.....	8
2.2.3 หน่วยการออกแบบแพ็คเกจ.....	11
2.2.4 หน่วยการออกแบบโครงแบบ.....	12
2.3 ภาษาวีเอชดีแอลเพื่อการสังเคราะห์.....	13
2.3.1 ตัวอย่างรูปแบบการเขียนเกตพื้นฐาน.....	13
2.3.2 ตัวอย่างรูปแบบการเขียนฟลิปฟล็อปพื้นฐาน.....	14
2.4 การออกแบบจากบนลงล่าง.....	15
บทที่ 3 เอฟพีจีเอ.....	17
3.1 Field Programmable.....	17
3.1.1 ฟีแอลดี (PLD: Programmable Logic Device).....	18
3.1.2 พรอม (PROM: Programmable Read Only Memory).....	18

สารบัญ(ต่อ)

หน้า

3.1.3	พีเอแอล (PAL: Programmable Array Logic).....	19
3.1.4	พีแอลเอ (PLA: Programmable Logic Array).....	19
3.1.5	แอลซีเอ (LCA: Logic Cell Array).....	20
3.1.6	อีพีแอลดี (EPLD: Erasable Programmable Logic Device).....	21
3.2	Mask programmable.....	21
3.2.1	เกตอาร์เรย์ (Gate Array).....	21
3.2.2	เซลล์มาตรฐาน (Standard Cell).....	22
3.2.3	ฟูลคัสตัม (Full Custom)	22
3.3	เอฟพีจีเอ (FPGA: Field Programmable Gate Array).....	23
3.3.1	สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล XC4000	23
3.3.1.1	ซีแอลบี (CLB: Configuration Logic Block).....	24
3.3.1.2	ไอโอบี (IOB: Input Output Block)	24
3.3.1.3	อินเตอร์คอนเน็ค (Interconnect).....	25
3.3.2	คุณสมบัติโดยทั่วไปของเอฟพีจีเอตระกูล XC4000.....	26
3.4	การโปรแกรมเอฟพีจีเอตระกูล XC4000.....	28
3.4.1	โหมดการโปรแกรม.....	28
3.4.1.1	โหมดหลัก (Master modes).....	28
3.4.1.2	เพอริเฟอรัลโหมด (Peripheral modes)	29
3.4.1.3	โหมดรองแบบอนุกรม (Slave Series modes).....	30
3.5	การใช้ความสามารถของแรมในเอฟพีจีเอตระกูล XC4000.....	31
บทที่ 4	โครงสร้างสถาปัตยกรรมไมโครคอนโทรลเลอร์.....	34
4.1	คุณสมบัติของไมโครคอนโทรลเลอร์โดยทั่วไป	35
4.1.1	คุณสมบัติของหน่วยประมวลผลกลาง.....	35
4.1.2	คุณสมบัติทางเพอริเฟอรัล.....	35
4.1.3	คุณสมบัติพิเศษอื่นๆ	35
4.2	จังหวะสัญญาณนาฬิกาและไทม์ไลน์การทำงาน	36
4.3	สถาปัตยกรรมชุดคำสั่ง	36

สารบัญ(ต่อ)

	หน้า
4.4 การจัดการหน่วยความจำ.....	40
4.4.1 โหมดการอ้างอิงตำแหน่งที่อยู่ของข้อมูล.....	40
4.5 การอินเตอร์รัพต์ของไมโครคอนโทรลเลอร์.....	43
บทที่ 5 ขั้นตอนการออกแบบระบบ	45
5.1 กำหนดรายละเอียดของระบบ	45
5.1.1 โมดูลเอแอลเอ็ม.....	46
5.1.2 โมดูลไออาร์ไอดีเอ็ม.....	48
5.1.3 โมดูลพีซีเอสเอ็ม.....	52
5.1.4 โมดูลเอฟอาร์เอ็ม.....	54
5.1.5 โมดูลไอโอพีเอ็ม.....	57
5.2 การออกแบบระบบในรูปของภาษาวีเอชดีแอล.....	60
5.3 การสร้างต้นแบบจากภาษาวีเอชดีแอล.....	61
5.3.1 การออกแบบเอนทรี (Design Entry).....	61
5.3.2 การออกแบบอิมพลีเมนต์เดชัน (Design Implementation).....	62
5.3.3 การออกแบบในการตรวจสอบความถูกต้อง (Design Verification).....	63
บทที่ 6 ผลการออกแบบ	64
6.1 ผังวงจรของโมดูลเอแอลเอ็ม (ALM : Arithmetic and Logic Module).....	64
6.2 ผังวงจรของโมดูลไออาร์ไอดีเอ็ม (IRIDM : Instruction Register and Instruction Decoder Module).....	65
6.3 ผังวงจรของโมดูลพีซีเอสเอ็ม (PCSM : Program Counter and Stack Module).....	66
6.4 ผังวงจรของโมดูลเอฟอาร์เอ็มและโมดูลไอโอพีเอ็ม (FRM : File Register Module / IOPM : Input Output Port Module).....	67
บทที่ 7 การจำลองและการทดสอบการทำงาน	74

สารบัญ(ต่อ)

	หน้า
บทที่ 8 สรุปผลการวิจัยและบทวิจารณ์.....	84
8.1 วิธีการดำเนินงาน	84
8.2 การออกแบบและทดสอบ	84
8.3 ปัญหาและอุปสรรคในการดำเนินงาน.....	86
8.4 บทวิจารณ์	86
บรรณานุกรม.....	88
ภาคผนวก.....	89
ภาคผนวก ก. Soft Core ของไมโครคอนโทรลเลอร์ในรูปภาษาวีเอชดีแอล	90
ภาคผนวก ข. รายละเอียดชุดคำสั่งของไมโครคอนโทรลเลอร์.....	140
ผลงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์และได้รับการตีพิมพ์.....	156
ประวัติผู้เขียน.....	157

สารบัญรูป

รูปที่	หน้า
2.1 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี.....	6
2.2 แสดงรูปแบบของมัลติเพลกซ์ (a) หน่วยการออกแบบเอนทิตีในรูปของวีเอชดีแอล.....	7
2.3 รูปแบบมัลติเพลกซ์ที่ประกอบด้วยข้อมูลค่าเวลาหน่วยแพร่กระจาย.....	7
2.4 หน่วยการออกแบบเอนทิตีที่ไม่มีการกำหนดช่องทางที่ต่อกับภายนอก.....	8
2.5 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม.....	8
2.6 แสดงหน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ตามฟังก์ชันบูลีน $\text{output} = (\overline{\text{sel.in0}}) + (\text{sel.in1})$	9
2.7 แสดงโครงสร้างภายในสถาปัตยกรรมของมัลติเพลกซ์.....	9
2.8 หน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทโครงสร้าง.....	10
2.9 หน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทพฤติกรรม.....	10
2.10 แสดงโครงสร้างโดยทั่วไปของส่วนการประกาศแฟกต์เกิด.....	11
2.11 โครงสร้างของบอดีแฟกต์เกิด.....	12
2.12 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ.....	12
2.13 ขั้นตอนการออกแบบจากบนลงล่าง.....	15
3.1 ฟังก์ชันการแบ่งกลุ่มของวงจรรวม ASIC.....	17
3.2 แสดงวงจรพื้นฐานของอุปกรณ์พีแอลดีซึ่งอยู่ในรูปผลคูณร่วมบวก.....	18
3.3 แสดงลักษณะของพหุคูณเมื่อเปรียบเทียบเป็นวงจรในรูปผลคูณร่วมบวก.....	19
3.4 แสดงวงจรพื้นฐานภายในของพีแอลเอ.....	20
3.5 แสดงผังวงจรภายในของซีแอลพีของเอฟพีจีเอตระกูล XC4000.....	24
3.6 แสดงผังวงจรของไอโอบีภายในเอฟพีจีเอตระกูล XC4000.....	25
3.7 แสดงเส้นทางการเชื่อมต่อระหว่างไอโอบีกับซีแอลพีของเอฟพีจีเอตระกูล XC4000.....	25
3.8 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดหลักแบบขนาน.....	29
3.9 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดเพอร์ฟอร์มแบบอะซิงโครนัส.....	30
3.10 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดรอนแบบอนุกรม.....	30
4.1 แสดงโครงสร้างสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ตระกูลพีไอซี.....	34

สารบัญรูป(ต่อ)

รูปที่	หน้า
4.2 แสดงจังหวัดการทำงานของไมโครคอนโทรลเลอร์.....	36
4.3 รูปแบบของคำสั่ง (Instruction Format).....	37
4.4 โฟลวชาร์ตของโปรแกรมที่ใช้ในในการเข้าถึงแอดเดรสทางอ้อม.....	41
4.5 แสดงการทำงานของโปรแกรมน้อยที่ใช้การเข้าถึงแบบสัมผัส.....	42
4.6 แสดงการจัดสรรหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลในไมโครคอนโทรลเลอร์.....	43
4.7 แสดงรายละเอียดของเรจิสเตอร์ควบคุมการอินเตอร์รัพต์.....	44
5.1 บล็อกไดอะแกรมที่ถูกแบ่งออกเป็นโมดูลต่างๆ ในขั้นตอนวิเคราะห์ระบบ.....	46
5.2 โครงสร้างสถาปัตยกรรมภายในของโมดูลเอแอลเอ็ม.....	47
5.3 โครงสร้างสถาปัตยกรรมภายในของโมดูลไออาร์ไอดีเอ็ม.....	49
5.4 โครงสร้างสถาปัตยกรรมภายในของโมดูลพีซีเอสเอ็ม.....	53
5.5 แสดงไดอะแกรมการทำงานของไทเมอร์ศูนย์เรจิสเตอร์.....	54
5.6 แสดงสัญญาณลอจิกของการอินเตอร์รัพต์ภายในเรจิสเตอร์ควบคุมการอินเตอร์รัพต์.....	56
5.7 โครงสร้างสถาปัตยกรรมภายในของโมดูลเอฟอาร์เอ็มและโมดูลไอโอพีเอ็ม.....	58
5.8 แสดงโครงสร้างการเชื่อมต่อภายในของทุกโมดูลในไมโครคอนโทรลเลอร์.....	59
5.9 แสดงลำดับชั้นของไฟล์โมดูลที่ออกแบบทั้งหมด.....	60
5.10 ขั้นตอนการออกแบบเอนทรี.....	62
5.11 ขั้นตอนการออกแบบอิมพลิเมนต์เดชันและขั้นตอนการตรวจสอบความถูกต้อง.....	63
6.1 ผังวงจรของโมดูลเอแอลเอ็มที่ได้จากการสังเคราะห์.....	64
6.2 ผังวงจรโมดูลไออาร์ไอดีเอ็มที่ได้จากการสังเคราะห์.....	65
6.3 ผังวงจรโมดูลพีซีเอสเอ็มที่ได้จากการสังเคราะห์.....	66
6.4 ผังวงจรโมดูลเอฟอาร์เอ็มและโมดูลไอโอพีเอ็มที่ได้จากการสังเคราะห์.....	67
6.5 แสดงการเชื่อมต่อโมดูลต่างๆ ในระดับบนสุดของผังวงจรไมโครคอนโทรลเลอร์.....	68
6.6 แสดงผังวงจรในระดับบนสุดก่อนนำไปเฟลตเมนต์ลงบนเอฟพีจีเอ.....	69
6.7 แสดงภาพตำแหน่งของเซลล์ภายในเอฟพีจีเอที่ถูกใช้ในการเฟลตเมนต์ผังวงจรที่ออกแบบ.....	72
6.8 แสดงภาพเลย์เอาต์ของผังวงจรภายในอุปกรณ์เอฟพีจีเอที่ถูกเฟลตเมนต์และเร้าต์ติ้ง.....	73

สารบัญรูป(ต่อ)

รูปที่	หน้า
7.1 แสดงขั้นตอนการจำลองและทดสอบการทำงาน.....	74
7.2 แสดงขั้นตอนการสร้างรอมเจนที่ใช้ในการทดสอบการทำงานของคอร์.....	75
7.3 แสดงไทม์มิงไคอะแกรมของการซิมมูลേഷันในระดับฟังก์ชันในโปรแกรมทดสอบที่ 1.....	77
7.4 แสดงไทม์มิงไคอะแกรมของการซิมมูลേഷันในระดับเกิดในโปรแกรมทดสอบที่ 1.....	78
7.5 แสดงไทม์มิงไคอะแกรมของการซิมมูลേഷันในระดับฟังก์ชันในโปรแกรมทดสอบที่ 2.....	80
7.6 แสดงไทม์มิงไคอะแกรมของการซิมมูลേഷันในระดับเกิดในโปรแกรมทดสอบที่ 2.....	81
7.7 แสดงเลย์เอาต์ของบอร์ดทดสอบคอร์ของไมโครคอนโทรลเลอร์.....	82
7.8 แสดงระบบที่ใช้ทดสอบคอร์ของไมโครคอนโทรลเลอร์ในระดับฮาร์ดแวร์.....	83
7.9 แสดงไทม์มิงไคอะแกรมที่ได้จากสโคปในการทำงานของคอร์ในระดับฮาร์ดแวร์.....	83

สารบัญตาราง

ตารางที่	หน้า
3.1 แสดงรายละเอียดของอุปกรณ์ภายในเอฟพีจีเอตระกูล XC4000.....	27
3.2 แสดงรูปแบบของโหมดต่างๆ ในการโปรแกรมเอฟพีจีเอตระกูล XC4000.....	28
3.3 จำนวนของแรมภายในเอฟพีจีเอตระกูล XC4000	31
3.4 รายละเอียดของขาอุปกรณ์เอฟพีจีเอเบอร์ XC4010.....	32
4.1 แสดงรายละเอียดของตัวแปรที่เกี่ยวข้องในรูปแบบของคำสั่ง.....	37
4.2 รูปแบบของคำสั่งของไมโครคอนโทรลเลอร์.....	38
5.1 รายละเอียดสัญญาณควบคุมที่ได้จากการถอดรหัสคำสั่งต่างๆ.....	50
5.2 แสดงการถอดรหัสของรูปแบบคำสั่งต่างๆ เป็นสัญญาณควบคุมแบบมีเงื่อนไข.....	51
6.1 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรเอแอลเอ็ม โมดูล	65
6.2 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรไออาร์ไอดีเอ็ม โมดูล	66
6.3 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรพีซีเอสเอ็ม โมดูล	67
6.4 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรเอฟอาร์เอ็มและไอโอพีเอ็ม โมดูล.....	68
6.5 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรไมโครคอนโทรลเลอร์ที่ออกแบบ.....	69
6.6 แสดงขาไอโอของไมโครคอนโทรลเลอร์กับขาไอโอของเอฟพีจีเอ XC4010PC84-5.....	70
6.7 แสดงความหมายของไอโอพอร์ตต่างๆ ในไมโครคอนโทรลเลอร์	71
6.8 แสดงค่าการใช้ทรัพยากรภายในอุปกรณ์เอฟพีจีเอในการเพลตเมนต์และเรอ์ตตั้ง	72
8.1 เปรียบเทียบคุณสมบัติของ PIC16C84 กับไมโครคอนโทรลเลอร์ที่ออกแบบ.....	85

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันประเทศไทยมีการนำไมโครคอนโทรลเลอร์เข้ามาใช้งานอย่างกว้างขวาง ไม่ว่าจะเป็นอุปกรณ์เครื่องใช้ภายในบ้านตลอดจนในธุรกิจอุตสาหกรรม โดยทั่วไปผู้ใช้จะนิยมตั้งชื่อชิปหรือไมโครคอนโทรลเลอร์จากบริษัทผู้ผลิต ซึ่งส่วนใหญ่จะเป็นบริษัทในต่างประเทศ โดยที่ไม่ได้คำนึงถึงการสร้างและพัฒนาชิปที่นำมาใช้กับผลิตภัณฑ์ของตนเอง สาเหตุนี้เนื่องมาจากขาดเครื่องมือและความเข้าใจในการออกแบบนั่นเอง หรือเกิดจากความคิดที่ว่า การออกแบบระบบฮาร์ดแวร์ดิจิทัลเป็นเรื่องที่ยากและไม่สามารถที่จะออกแบบได้ ดังนั้นจึงเป็นที่มาของวิทยานิพนธ์เล่ม ซึ่งได้นำเสนอขั้นตอนการออกแบบไมโครคอนโทรลเลอร์โดยใช้เอฟพีจีเอ (FPGA: Field Programmable Gate Array) โดยเริ่มจากออกแบบพฤติกรรมการทำงานของไมโครคอนโทรลเลอร์โดยเลือกนำสถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูลพีไอซี (PIC: Peripheral Interface Controller) ของบริษัทไมโครชิปเทคโนโลยี (Microchip Technology Inc.) ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 8 บิตที่เป็นแบบชิปเดี่ยว (Single Chip) ซึ่งมีหน่วยซีพียู หน่วยความจำ ไอโอพอร์ต หน่วยควบคุมการทำงาน หน่วยของการบริการอินเตอร์รัพต์ เคนำเตอร์และไทมเมอร์อยู่ในชิปเดี่ยว และไมโครคอนโทรลเลอร์ตระกูลนี้ยังจัดอยู่ในกลุ่มของไมโครโพรเซสเซอร์แบบรีดิวซ์ (RISC: Reduced Instruction Set Computer) กล่าวคือมีชุดคำสั่งน้อยมาก ซึ่งมีเพียง 36 คำสั่ง และสามารถทำงานในลักษณะไปป์ไลน์ (Pipeline) คือการทำงานที่เหลื่อมกันในแต่ละสเตจการทำงาน ซึ่งจะทำให้ความเร็วในการทำงานสูงเมื่อเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นที่ความถี่สัญญาณพิกษาเท่ากัน มาทำการออกแบบเพราะเป็นไมโครคอนโทรลเลอร์ที่มีโครงสร้างสถาปัตยกรรมที่ไม่ซับซ้อนมากนัก รูปแบบคำสั่งที่น้อย ด้วยเริ่มทำการออกแบบจากการเขียนพฤติกรรมการทำงานด้วยภาษาวีเอชดีแอล (VHDL: VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) จากนั้นทำการจำลองการทำงาน โดยใช้โปรแกรมช่วยในการออกแบบอัตโนมัติ (EDA: Electronics Design Automation) และสังเคราะห์ให้อยู่ในรูปผังวงจร เพื่อนำไปโปรแกรมลงเอฟพีจีเอ เพื่อนำไปทดสอบการทำงานในรูปของฮาร์ดแวร์ต่อไป

1.2 ความมุ่งหมายและวัตถุประสงค์ของงานวิจัย

ในการทำวิทยานิพนธ์เรื่องการออกแบบและสร้างไมโครคอนโทรลเลอร์โดยใช้เอฟพีจีเอ ได้ทำการศึกษารูปแบบของภาษาวีเอชดีแอลที่นำมาใช้ออกแบบและสร้างไมโครคอนโทรลเลอร์ต้นแบบ โดยนำเอาวิธีออกแบบจากบนลงล่าง (Top-Down design) มาเป็นขั้นตอนการออกแบบซึ่งเป็นวิธีการออกแบบระบบดิจิทัลที่ใช้กันในปัจจุบัน โดยการทำวิทยานิพนธ์นี้ได้กำหนดจุดประสงค์ไว้ดังนี้

- เพื่อศึกษารูปแบบการเขียนภาษาวีเอชดีแอลในการออกแบบระบบฮาร์ดแวร์ดิจิทัล ซึ่งเริ่มตั้งแต่การออกแบบ แก้วไขตรวจสอบ จำลองการทำงาน จนถึงขั้นผลิตวงจรหรือสังเคราะห์วงจร (Synthesis)
- เพื่อศึกษาวิธีการออกแบบจากบนลงล่าง และโปรแกรมที่ช่วยในการออกแบบอัตโนมัติ ได้แก่ Viewlogic, V-system, XACT Viewlogic interface , XILINX Foundation M 1.5 และ Exemplar logic design
- เพื่อศึกษาสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ PIC16C84 ที่นำมาเป็นสถาปัตยกรรมหลักในการออกแบบไมโครคอนโทรลเลอร์ต้นแบบ และข้อจำกัดในการออกแบบต่างๆ
- เพื่อศึกษาคุณสมบัติและการใช้อุปกรณ์เอฟพีจีเอมาสร้างไมโครคอนโทรลเลอร์ต้นแบบ (Prototype) และข้อจำกัดของอุปกรณ์เอฟพีจีเอในการสร้างไมโครคอนโทรลเลอร์ต้นแบบ
- เพื่อสร้างไมโครคอนโทรลเลอร์ต้นแบบในรูปของฮาร์ดแวร์โดยใช้อุปกรณ์เอฟพีจีเอและการประยุกต์ใช้งานจริง

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะเริ่มจากการออกแบบซอฟต์แวร์ (Soft Core) ของไมโครคอนโทรลเลอร์ขนาด 8 บิตให้ทำงานคล้ายกับการทำงานของไมโครคอนโทรลเลอร์ตระกูลพีไอซีของบริษัทไมโครชิป เทคโนโลยีเบอร์ PIC16C84 ด้วยภาษาวีเอชดีแอล โดยอาศัยหลักการออกแบบจากบนลงล่างมาเป็นขั้นตอนการออกแบบ จากนั้นจะใช้โปรแกรมช่วยในการออกแบบอัตโนมัติ ในการซิมมูลേഷัน แก้วไขตรวจสอบ และทำการสังเคราะห์วงจร เพื่อนำไปโปรแกรมลงอุปกรณ์เอฟพีจีเอ เพื่อทดสอบการใช้งานจริง

1.4 รายละเอียดของวิทยานิพนธ์

ในวิทยานิพนธ์นี้ได้แบ่งเนื้อหาออกเป็นบทได้ทั้งหมด 8 บท โดยบทที่ 1 จะเป็นการกล่าวนำถึง แนวความคิดและวัตถุประสงค์ในการทำวิทยานิพนธ์ และได้กล่าวถึงเนื้อหาโดยย่อของแต่ละบท ซึ่งในบทอื่นๆ จะมีเนื้อหา ดังนี้

บทที่ 2 กล่าวถึงรูปแบบการเขียนภาษาวีเอชดีแอล และวิธีการออกแบบจากบนลงล่าง

บทที่ 3 กล่าวถึงรายละเอียดของอุปกรณ์เอฟพีจีเอที่นำมาใช้และการใช้งาน

บทที่ 4 กล่าวถึงสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ที่นำมาออกแบบ

บทที่ 5 กล่าวถึงขั้นตอนการออกแบบและสร้างไมโครคอนโทรลเลอร์ต้นแบบ

บทที่ 6 กล่าวถึงผลการออกแบบผังวงจรของไมโครคอนโทรลเลอร์ที่ได้จากการสังเคราะห์

บทที่ 7 กล่าวถึงการจำลองและการทดสอบการทำงานของไมโครคอนโทรลเลอร์ต้นแบบ

บทที่ 8 กล่าวสรุปผลการวิจัยและบทวิจารณ์

ภาคผนวก แสดงรูปโมเดลของภาษาวีเอชดีแอลและรายละเอียดรูปแบบคำสั่งที่ออกแบบ

บทที่ 2

ภาษาวีเอชดีแอล

วีเอชดีแอล ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (high level language) ที่ใช้ในการออกแบบฮาร์ดแวร์ในระบบดิจิทัล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (hierarchy) ได้ และสามารถที่จะเขียนได้หลายรูปแบบซึ่งจะกล่าวต่อไป จึงทำให้ภาษาวีเอชดีแอลเป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปถึงขั้นตอนจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาสามารถเปิดโอกาสให้วิศวกรได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชันการทำงานของวงจรอย่างสังเขป โดยที่ยังไม่ต้องไปคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้นวีเอชดีแอลยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิทัลที่มีความซับซ้อนได้ทั้งหมด จึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งาน

2.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล

วิวัฒนาการของภาษาวีเอชดีแอล นั้นเริ่มต้นประมาณปี ค.ศ. 1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกา หรือ ดีโอดี (DoD: Department of Defense) มองเห็นว่าอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร เป็นอุปกรณ์ที่ได้รับการพัฒนามาเมื่อประมาณ 20 ปีก่อน เพราะเทคโนโลยีในขณะนั้นทำให้การพัฒนาอุปกรณ์อิเล็กทรอนิกส์เป็นไปอย่างล่าช้า ซึ่งเป็นสภาพที่ไม่อาจยอมรับได้ในปัจจุบัน เพราะเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์ ได้รับการพัฒนาไปอย่างรวดเร็ว ดังที่จะเห็นได้ว่ามีวงจรดิจิทัลอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นมาจากชิ้น ถูกนำประกอบกันอยู่บนแผงวงจรไฟฟ้า ที่มีขนาดใหญ่ แต่ในปัจจุบันสามารถที่จะใช้เทคโนโลยีการออกแบบและผลิตวงจรรวมขนาดใหญ่มาก (VLSI: Very Large Scale Integration) รวมอุปกรณ์ต่างๆ เหล่านั้นให้อยู่บนชิ้นอุปกรณ์สารกึ่งตัวนำ ที่มีขนาดประมาณ 1-2 ตร.ซม. ได้ ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้น (ความเร็วในการทำงานของวงจร) ลดลงจนความน่าเชื่อถือในการทำงาน และความคงทนต่อสภาพแวดล้อมสูง ขณะเดียวกันนั้นในวงการทหารได้มีการนำระบบคอมพิวเตอร์และอิเล็กทรอนิกส์ มาใช้ในระบบอาวุธอย่างแพร่หลาย ดังนั้นอุปกรณ์ที่มีใช้อยู่จึงไม่เหมาะสมกับเทคโนโลยีด้านอาวุธของประเทศคู่แข่งกัน การที่จะเปลี่ยนอุปกรณ์ใหม่เป็นสิ่งที่ต้องใช้งบประมาณมาก และก็จะประสบกับปัญหาเช่นเดิมคือ อุปกรณ์ใหม่ได้รับการพัฒนามานานแล้วเช่นกัน เพราะในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ต้องใช้วิศวกร และเวลาสำหรับดำเนินการมาก ฉะนั้นทาง

ดีไอดีจึงตั้งโครงการขึ้นมาเพื่อศึกษา วิธีการที่จะช่วยพัฒนาวงจรรีเลย์ทรอนิกส์ โดยเฉพาะอย่างยิ่ง วงจรระบบดิจิทัล ให้สามารถนำไปผลิตได้เร็วขึ้น และโครงการดังกล่าวมีชื่อว่า “Very High Speed Integrated Circuits” หรือ วีเอชเอสไอซี (VHSIC) ในระยะแรกนั้นโครงการเป็นความลับทางด้านความมั่นคงของประเทศ และอยู่ในความดูแลควบคุมของ United States International Traffic and Arms Regulations หรือ ไอทีเออาร์ (ITAR) ในปี ค.ศ. 1983 ตามคำแนะนำของคณะทำงาน (“Woods Hole” workshop) ทาง ดีไอดีได้ออกความต้องการมาตรฐานของภาษาที่ใช้สำหรับบรรยาย พฤติกรรมของวงจรหรือฮาร์ดแวร์ของระบบสำหรับโครงการวีเอชเอสไอซี ซึ่งมีสาระสำคัญพอสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิทัล และมีคุณสมบัติที่สามารถจะเข้าใจได้ทั้งคน และเครื่องโดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้อาจจัดเป็นภาษาโปรแกรมระดับสูง เช่นเดียวกับภาษาปาสคาล หรือภาษา ซี ซึ่งในทางวิศวกรรมการออกแบบฮาร์ดแวร์เรียกว่า “Hardware Description Language” หรือ เอชดีแอล (HDL) เริ่มต้นโครงการดีไอดีได้มอบหมายให้บริษัทไอบีเอ็มและบริษัทเท็กซัสอินสตรูเมนต์ และบริษัทอินเตอร์เมทริกซ์เป็นผู้ศึกษาและพัฒนา การดำเนินการได้กระทำไปอย่างต่อเนื่อง และได้ผลเป็นที่น่าพอใจ จนกระทั่งปี ค.ศ. 1985 ทางไอทีเออาร์ได้ยกเลิกข้อจำกัดในการถ่ายทอด เทคโนโลยีทางทหาร ออกจากโครงการนี้ ดังนั้นภาษาวีเอชดีแอล จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป จนกระทั่งทางไออีอีอี (IEEE) จึงได้รับภาษานี้เข้ามาศึกษาและประมาณปี ค.ศ. 1987 ได้ยอมรับกำหนด มาตรฐานของภาษา โดยให้ชื่อว่า IEEE 1076-1987 และมีชื่อเรียกว่าวีเอชดีแอลมาตรฐานนี้ก็ได้รับการปรับปรุงจนปัจจุบัน ได้ชื่อว่า IEEE 1076-1993 หรือ วีเอชดีแอล 1993 การที่ทางดีไอดีในขณะ นั้น เป็นลูกค้ารายใหญ่ของอุตสาหกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ จึงมีผู้รับโครงการต่างๆ จากดีไอดี ไปดำเนินการด้านวิจัยและพัฒนา มาก เพื่อที่จะให้เป็นมาตรฐานเดียวกันหมด ทางดีไอดี จึงกำหนดว่า ในการส่งโครงการนั้นจะต้องเขียนอยู่ในรูปของภาษาวีเอชดีแอลเท่านั้น ซึ่งทำให้เกิด ข้อดีต่อดีไอดีเองที่เป็นมาตรฐานเดียวกัน สามารถนำไปจำลองกับเครื่องคอมพิวเตอร์ได้หลายๆ ระบบ

2.2 ส่วนประกอบต่างๆ ของภาษาวีเอชดีแอล

ในการเขียนรูปแบบบรรยายระบบดิจิทัลในมุมมองของการออกแบบลักษณะบนลงล่าง จะต้องทำความเข้าใจในเรื่องของโครงสร้างและส่วนประกอบต่างๆ ของรูปแบบภาษาวีเอชดีแอลเสียก่อน ซึ่งส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียนมี 4 หน่วยคือ

- หน่วยการออกแบบเอนทิตี (Entity Design Unit)
- หน่วยการออกแบบสถาปัตยกรรม (Architecture Design Unit)
- หน่วยการออกแบบแพ็คเกจ (Package Design Unit)
- หน่วยการออกแบบโครงแบบ (Configuration Design Unit)

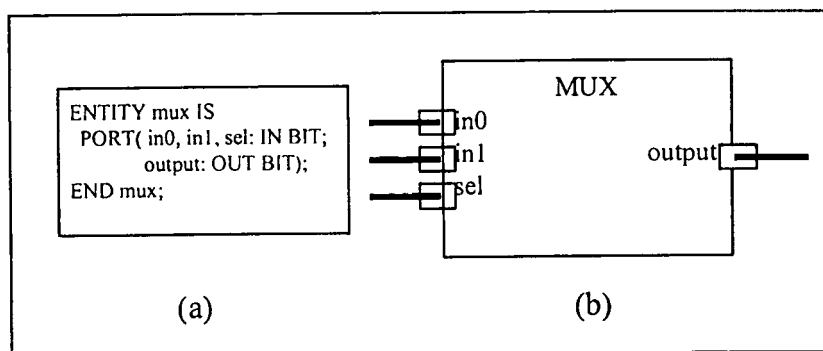
2.2.1 หน่วยการออกแบบเอนทิตี

หน่วยการออกแบบนี้เป็นส่วนที่ใช้สำหรับติดต่อระหว่างโลกภายนอกกับรูปแบบที่เขียนขึ้น ที่เรียกว่า หน่วยการออกแบบเอนทิตี ในส่วนนี้ใช้กำหนดจุดเชื่อมต่อ ของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆ ของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น รูปที่ 2.1 แสดงให้เห็นโครงสร้างอย่างง่าย ๆ ของ หน่วยการออกแบบเอนทิตี

```
ENTITY component_name IS
    Input and output ports
    Physical and other parameters
END [component_name];
```

รูปที่ 2.1 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี

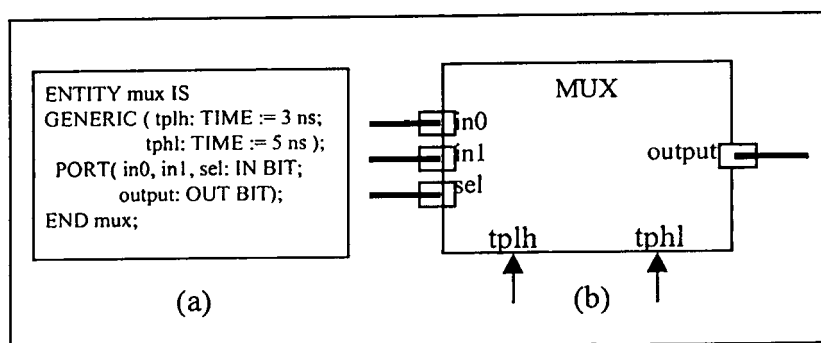
ส่วนนี้จะขึ้นต้นด้วยคำ ENTITY และ IS ระหว่างคำทั้งสองเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน (component_name) สำหรับการตั้งชื่อนั้นต้องเป็นไปตามกฎเกณฑ์ของภาษาหลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและออกของข้อมูล (input-output) รวมทั้งพารามิเตอร์อื่นๆ ส่วนนี้เรียกว่าส่วนหัว (entity header) และที่สำคัญคือ หน่วยการออกแบบเอนทิตีจะต้องปิดท้ายด้วยคำว่า END และเครื่องหมายอัฒภาคเสมอ (;)



รูปที่ 2.2 แสดงรูปแบบของมัลติเพลกซ์ (a) หน่วยการออกแบบเอนทิตีในรูปแบบของวีเอชดีแอล
(b) มุมมองของตัวเชื่อมประสาน (Interfacing)

ในรูปที่ 2.2 เป็นหน่วยการออกแบบเอนทิตี ที่บรรยายอุปกรณ์ที่มีชื่อว่ามัลติเพลกซ์ หรือ MUX ในส่วนหัวของเอนทิตี มีการกำหนดจุดต่อ 4 จุดภายใต้ชุดคำสั่ง PORT โดยที่ 3 จุดแรกเป็นจุดให้ข้อมูลไหลผ่านเข้า ได้แก่ in0, in1, sel ซึ่งกำหนดด้วยทิศทาง การติดต่อกับภายนอกเป็นการไหลเข้าของข้อมูล (IN) ที่แสดงด้วยรูปสี่เหลี่ยมโป่งในรูปที่ 2.2 ส่วนจุดเอาต์พุตเป็นจุดให้ข้อมูลไหลออก ซึ่งกำหนดด้วยทิศทาง การติดต่อกับภายนอกเป็นการไหลออก (OUT) ที่แสดงด้วยรูปสี่เหลี่ยมทึบในรูปที่ 2.2 ส่วนประเภทของข้อมูลที่จะไหล เข้าและออก นั้นเป็นประเภท BIT ที่สามารถมีค่าได้เพียงสองค่าคือ '0' และ '1' เท่านั้น

นอกจากนั้นผู้ออกแบบยังสามารถกำหนดค่าพารามิเตอร์ทางฟิสิกส์ที่เป็นข้อมูลเพิ่มเติมอื่นๆ ลงในส่วนหัวของเอนทิตีได้อีก เช่น ข้อมูลเกี่ยวกับความเร็วในการทำงานของอุปกรณ์ อันได้แก่ ค่าเวลาหน่วงแพร่กระจาย (Propagation delay time) พารามิเตอร์เหล่านี้ เรียกว่า เจเนริก (Generic) ที่กำหนดด้วยคำสั่ง GENERIC จากตัวอย่างในรูปที่ 2.3



รูปที่ 2.3 รูปแบบมัลติเพลกซ์ที่ประกอบด้วยข้อมูลค่าเวลาหน่วงแพร่กระจาย
(a) หน่วยการออกแบบเอนทิตีในรูปแบบของวีเอชดีแอล (b) มุมมองของตัวเชื่อมประสาน

ในบางกรณีสามารถใช้ภาษาวีเอชดีแอล สร้างรูปแบบที่ปราศจากช่องทางไหล เข้าและออกของข้อมูล ได้ ซึ่งส่วนใหญ่จะพบในการสร้างรูปแบบ สำหรับตรวจสอบการทำงานของอีกรูปแบบหนึ่ง คือ วีเอชดีแอลสำหรับการทดสอบเปรียบเทียบ (Test bench)

```
ENTITY test_bench IS
END test_bench;
```

รูปที่ 2.4 หน่วยการออกแบบเอนทิตีที่ไม่มีการกำหนดช่องทางที่ต่อกับภายนอก

2.2.2 หน่วยการออกแบบสถาปัตยกรรม

คือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบ ในมุมมองของการจำลองการทำงาน พฤติกรรมต่างๆ ที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออก ตรงช่องทางตลอดจนพารามิเตอร์ต่างๆ ที่กำหนดใน หน่วยการออกแบบเอนทิตี รูปที่ 2.5 แสดงให้เห็นถึงโครงสร้างอย่างง่ายของหน่วยการออกแบบสถาปัตยกรรม

```
ARCHITECTURE identifier OF component_name IS
  [declaration]
BEGIN
  specification of the functionality of the
  component in terms of its input lines and as
  influenced by physical and other parameters
END [identifier];
```

รูปที่ 2.5 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม

ส่วนของหน่วยการออกแบบสถาปัตยกรรม เริ่มต้นด้วยคำ ARCHITECTURE และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่ง que แสดงให้เห็นว่า ARCHITECTURE นั้นใช้บรรยาย หน่วยการออกแบบเอนทิตีใดๆ (OF <entity design unit> IS) ส่วนที่อยู่ระหว่าง ARCHITECTURE และ BEGIN เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (architecture declarative area) ที่เป็นเพียงส่วนเพื่อเลือก (option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าต่างๆ ที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ อาทิเช่นประเภท (type) ต่างๆ (ตัวอย่างเช่น bit, bit_vector), สัญญาณ (signal), ตัวคงที่ (constant), โปรแกรมย่อย (ได้แก่ function และ procedure) และอุปกรณ์

(component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า และไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง PORT) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า BEGIN กับ END ของหน่วยการออกแบบสถาปัตยกรรม และนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้ จะเป็นชุดคำสั่งแบบแข่งขันกัน (concurrent statement) เท่านั้น หน่วยการออกแบบสถาปัตยกรรม จะต้องปิดท้ายด้วยคำสั่ง END และชื่อของสถาปัตยกรรมนั้นๆ ที่เป็นส่วนเพื่อเลือกโดยทั่วไปการเขียนรูปแบบระบบดิจิทัลด้วยภาษาวีเอชดีแอล สามารถเขียนได้ในลักษณะต่างๆ ดังนี้

- ประเภทการไหลของข้อมูล (Dataflow description)
- ประเภทพฤติกรรม (Behavioral description)
- ประเภทโครงสร้าง (Structure description)
- ประเภทผสม (Mixed model description)

```

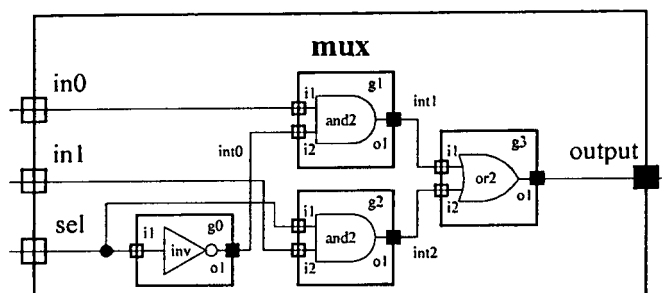
ARCHITECTURE data_flow OF mux IS
BEGIN
    output <= ((NOT sel) AND in0) OR (sel AND in1);
END data_flow;

```

รูปที่ 2.6 แสดงหน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ตามฟังก์ชันบูลีน

$$\text{output} = \overline{(\text{sel.in0})} + (\text{sel.in1})$$

รูปที่ 2.6 ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (in0 , in1) กับข้อมูลที่ไหลออก (output) ประกอบด้วยชุดคำสั่งแบบแข่งขันกันเพียงชุดเดียว ซึ่งเขียนเป็นประเภทการไหลของข้อมูลของมัลติเพลกซ์ หรือ ระดับการถ่ายโอนข้อมูลระหว่างเรจิสเตอร์ (RTL: Register Transfer Level)



รูปที่ 2.7 แสดงโครงสร้างภายในสถาปัตยกรรมของมัลติเพลกซ์

รูปที่ 2.7 เป็นหน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทโครงสร้าง โดยใช้ อินเวอร์เตอร์ (inv ที่ตำแหน่ง g0), แอนด์เกต 2 อินพุตจำนวน 2 ตัว (and2 ที่ตำแหน่ง g1 และ g2) และ ออร์เกต 2 อินพุต (or2 ที่ตำแหน่ง g3) มาสร้างตามฟังก์ชันบูลีนของรูปที่ 2.6

```

ARCHITECTURE struc OF mux IS
    COMPONENT inv
    PORT ( i1 : IN BIT ; o1 : OUT BIT );
    COMPONENT and2
    PORT ( i1, i2 : IN BIT ; o3 : OUT BIT );
    COMPONENT or2
    PORT ( i1, i2 : IN BIT ; o1 : OUT BIT );
END COMPONENT;
    SIGNAL int0, int1, int2 : BIT;
BEGIN
    g0 : inv  PORT MAP (i1 => sel, o1 => int0);
    g1 : and2 PORT MAP (i1 => in0, i2 => int0, o1 => int1);
    g3 : or2  PORT MAP (i1 => int1, i2 => int2, o1 => ouput);
END struc;

```

รูปที่ 2.8 หน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทโครงสร้าง

```

ARCHITECTURE behav OF mux IS
BEGIN
    PROCESS (in0, in1, sel)
    BEGIN
        IF (sel = '0') THEN output <= in0;
        ELSE output <= in1;
        END IF;
    END PROCESS;
END behav;

```

รูปที่ 2.9 หน่วยการออกแบบสถาปัตยกรรมของมัลติเพลกซ์ประเภทพฤติกรรม

ไม่ว่าเขียนบรรยายส่วนของสถาปัตยกรรมของมัลติเพล็กซ์ในลักษณะของ ประเภทพฤติกรรม ประเภทการไหลของข้อมูล ประเภทโครงสร้างหรือประเภทผสมที่นำเอาแต่ละประเภทมาเขียนไว้ในส่วนของสถาปัตยกรรม ก็ตามต่างก็มีพฤติกรรมเดียวกัน และจะให้ผลลัพธ์จากการจำลองการทำงานที่เหมือนกัน ซึ่งนี่ก็เป็นข้อดีของภาษาวีเอชดีแอล

2.2.3 หน่วยการออกแบบแพ็คเกจ

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิทัล สามารถเก็บไว้ในส่วนของแพ็คเกจได้ และข้อมูลเหล่านี้สามารถเรียกไปใช้ได้โดย หน่วยการออกแบบเอนทิตี หน่วยการออกแบบสถาปัตยกรรม หรือ จากหน่วยการออกแบบแพ็คเกจอื่นๆ นอกจากนั้นสิ่งที่นิยามทำกันมากคือรูปแบบมาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐาน (เช่น ไอซีตระกูล 74XX เป็นต้น) จะถูกเก็บไว้ในแพ็คเกจ ที่ทุกคนสามารถเข้าถึง โดยปกติแล้ว แพ็คเกจจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็คเกจ (Package declaration) และ ส่วนของบอดีแพ็คเกจ (Package body) เนื่องจาก แพ็คเกจถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้น การที่นำแพ็คเกจไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษาวีเอชดีแอล สามารถกระทำได้ด้วยชุดคำสั่ง USE

● PACKAGE DECLARATION

ส่วนที่มีความสำคัญที่สุดของแพ็คเกจ (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ ส่วนการประกาศแพ็คเกจ เพราะจะเป็นส่วนที่กำหนดชื่อ ของสิ่งที่ประกาศอยู่ภายในแพ็คเกจ สำหรับนำไปใช้ภายนอกตัวของแพ็คเกจเอง สิ่งใดๆ ถูกประกาศในส่วนของ ส่วนบอดีแพ็คเกจ แต่ไม่ถูกประกาศในส่วนการประกาศแพ็คเกจ จะไม่สามารถถูกนำค่า และพฤติกรรมไปใช้ส่วนนอกได้ ซึ่งสามารถเปรียบเทียบกับสิ่งที่ประกาศไว้ในส่วนของการประกาศเอนทิตีคือ จุดเชื่อมต่อ หรือ พอร์ต ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้ว แพ็คเกจ สามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี และยังสามารถถูกนำไปใช้จากรูปแบบภายนอกได้เช่น ใช้สำหรับประกาศ ชนิด (Type) หรือ สัญญาณ เช่นเดียวกับกับ ส่วนบอดีแพ็คเกจ ที่ไม่จำเป็นต้องมี ส่วนของการประกาศแพ็คเกจ แต่แพ็คเกจ นั้นจะไม่สามารถถูกนำไปใช้จากรูปแบบอื่นได้

```
PACKAGE package_name IS
    Package_declarative_part
END package_name;
```

รูปที่ 2.10 แสดงโครงสร้างโดยทั่วไปของส่วนการประกาศแพ็คเกจ

● PACKAGE BODY

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆ ในรูปของคำสั่งลำดับ ที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อย (Subprogram)ทั้งหลายที่ชื่อของโปรแกรมย่อยนั้นๆ ที่ถูกประกาศไปในส่วนของการประกาศแพ็คเกจ แล้วจะถูกเก็บไว้ในส่วนบอดีแพ็คเกจ ทั้งนี้รวมทั้ง การกำหนดค่าคงที่ต่างๆ อันได้แก่ตัวคงที่ที่ถูกประกาศชื่อก่อนในส่วนของการประกาศแพ็คเกจ แต่ถูกกำหนดค่าในส่วนของบอดีแพ็คเกจ) ฉะนั้นส่วนบอดีแพ็คเกจ จึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็คเกจ ไม่มีการประกาศชื่อ ที่เป็นโปรแกรมย่อย หรือ ค่าคงที่ การเขียนบอดีแพ็คเกจนั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 2.11

```
PACKAGE BODY package_name IS
    declarative part
END package_name;
```

รูปที่ 2.11 โครงสร้างของบอดีแพ็คเกจ

2.2.4 หน่วยการออกแบบโครงสร้าง

ดังที่ทราบกันแล้วว่ารูปแบบหนึ่งของระบบดิจิทัลไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนทิตีได้ เพียงหนึ่งเดียวเท่านั้น แต่ในขณะที่หน่วยการออกแบบเอนทิตี หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรม ที่เป็นหน่วยรองได้หลายหน่วย ดังนั้นจะต้องมี หน่วยการออกแบบโครงสร้างมาเพื่อกำหนดการใช้โครงสร้าง (Configuration) ประกอบเอนทิตีกับหน่วยการออกแบบสถาปัตยกรรม หน่วยไหนเข้าด้วยกัน

```
CONFIGURATION identifier OF entity_name IS
    Configuration_declarative_part
END ;
```

รูปที่ 2.12 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงสร้าง

2.3 ภาษาวีเอชดีแอลเพื่อการสังเคราะห์

ภาษาวีเอชดีแอล เป็นภาษาที่เขียนเพื่อจำลองการทำงานของวงจร ซึ่งในบางรูปแบบการเขียนไม่สามารถที่จะนำไปสังเคราะห์ได้ทั้งหมด ดังนั้นถ้าต้องการเขียนเพื่อนำไปสังเคราะห์ ควรหลีกเลี่ยงรูปแบบต่างๆ ที่ไม่สามารถนำไปสังเคราะห์ได้ ในที่นี้ขึ้นอยู่กับความสามารถของโปรแกรมที่ใช้สังเคราะห์แต่ละโปรแกรม ดังนั้นในหัวข้อนี้จะแสดงตัวอย่างของการเขียนโมเดลในรูปแบบต่างๆ ที่สามารถนำไปสังเคราะห์ ซึ่งยึดหลักการเขียนตาม ViewSynthesis User's Guide ของโปรแกรม Viewlogic ซึ่งเป็นโปรแกรมที่ใช้ในสังเคราะห์วงจรทั้งหมดในการออกแบบไมโครคอนโทรลเลอร์ โดยแบ่งออกเป็น 2 กลุ่ม คือ เกตและฟลิปฟล็อปประเภทต่างๆ

2.3.1 ตัวอย่างรูปแบบการเขียนเกตพื้นฐาน

```
SIGNAL a, b, c, d, input, output : vlbit_1d(3 DOWNTO 0);
```

```
SIGNAL sel : vlbit_1d(1 DOWNTO 0);
```

```
SIGNAL enb : vlbit;
```

```
...
```

```
output <= a AND b;
```

```
...
```

```
output <= a OR b;
```

```
...
```

```
output <= a XOR b;
```

```
...
```

```
output <= NOT input;
```

```
...
```

```
output <= input WHEN enb='1' ELSE "ZZ";
```

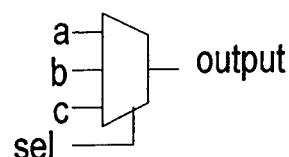
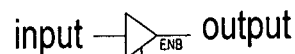
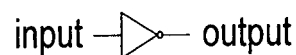
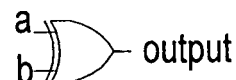
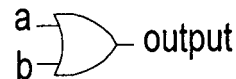
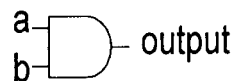
```
...
```

```
WITH sel SELECT
```

```
    output <= a WHEN "00" ELSE
```

```
        b WHEN "01" ELSE
```

```
        c;
```



2.3.2 ตัวอย่างรูปแบบการเขียนฟลิปฟล็อปพื้นฐาน

SIGNAL input, output : vlbit_1d(3 DOWNT0 0);

SIGNAL clock, enable, reset : vlbit;

...

PROCESS BEGIN

 WAIT UNTIL PRISING(clock);

 output <= input;

END PROCESS;

...

PROCESS BEGIN

 WAIT UNTIL PFALLING(clock);

 output <= input;

END PROCESS;

...

PROCESS BEGIN

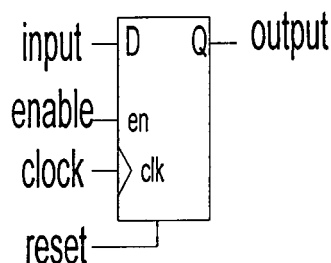
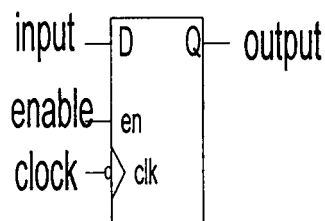
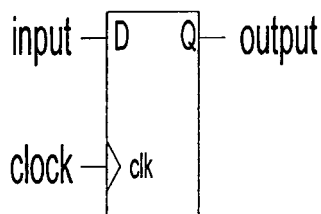
 WAIT UNTIL PRISING(clock) OR (reset='1');

 IF (reset = '1') THEN output <= "00";

 ELSIF (enble = '1') THEN ouput <= input;

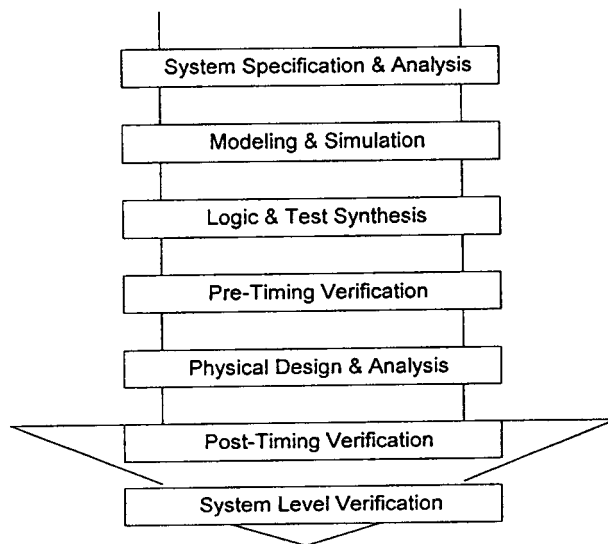
 END IF;

END PROCESS;



2.4 การออกแบบจากบนลงล่าง

ในการพัฒนางจรรวมดิจิทัลขนาดใหญ่ที่มีความซับซ้อน เช่น วงจรรวม (ASIC: Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของของบล็อกโคอะแกรมเสียก่อน ก่อนที่จะวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอลนั้นอนุญาตให้อธิบายการทำงานของแต่ละบล็อก และวิเคราะห์การทำงาน แก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ และเพิ่มเติมในรายละเอียดที่ละขั้นนี้คือ หลักการออกแบบจากบนลงล่าง (Top-Down Design) ถ้าทดลองเปรียบเทียบกับ การออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เพราะเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ จำลองการทำงาน ตรวจสอบความถูกต้อง ซึ่งใช้เวลามาก และถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากให้การออกแบบในลักษณะนี้ ดังนั้นการใช้ภาษาวีเอชดีแอลกับหลักการออกแบบจากบนลงล่างจึงเป็นทางเลือกให้กับวิศวกรออกแบบที่จะสามารถออกแบบและพัฒนางจรรวมที่ซับซ้อนได้มากขึ้น และช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ



รูปที่ 2.13 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.13 แสดงให้เห็นถึงขั้นตอนของการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย ก็เนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายๆ เทคโนโลยี เช่น ฟิแอลดี (PLD: Programmable Logic Device) อันได้แก่

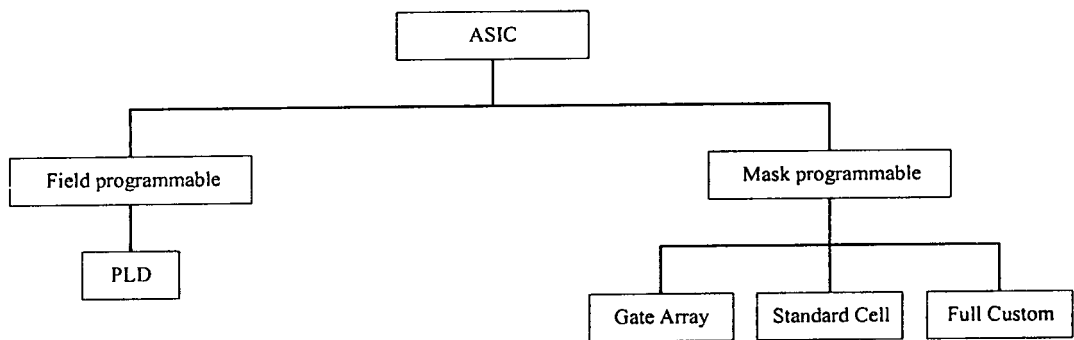
พีแอลเอ (PLA: Programmable Logic Array), เอฟพีจีเอ (FPGA: Field Programmable Gate Array), ซีพีแอลดี (CPLD: Cell Programmable Logic Device) เป็นต้น นอกนั้นยังมี เซมิคัสตัมไอซี (Semi-Custom IC) ได้แก่ เกตอะเรย์ (Gate array), เซลล์มาตรฐาน (Standard Cell) ขั้นตอนของการออกแบบจากบนลงล่างมีรายละเอียดดังนี้

1. ขั้นตอนการสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา
2. ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษาวีเอชดีแอล หรือ ภาษาเอชดีแอลอื่นๆ สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด
3. หลังจากที่ได้หลักการขั้นต้นพร้อมกับแนวความคิดที่ผ่านการตรวจสอบแล้ว หลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมาเป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจร หรือสังเคราะห์ ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้น ให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกต และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของเน็ตลิสต์ (Netlist) ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้
4. หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกตหรือเน็ตลิสต์แล้ว ข้อมูลที่ได้จากผู้ผลิตอุปกรณ์วงจรมานั้น นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงาน ในเรื่องของความถูกต้องของฟังก์ชันแล้ว ยังมีข้อมูลเกี่ยวกับเวลาด้วย ซึ่งเป็นความจริงที่ว่า อุปกรณ์ทางอิเล็กทรอนิกส์ทุกชิ้นจะมี เวลาหน่วงของการแพร่กระจาย (Propagation delay time) เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับ นาโนวินาที (10^{-9} นาติ) แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกตของฟังก์ชันต่างๆ จำนวน 10,000 เกต ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจจะทำให้การทำงานของวงจรรวมทั้งหมดผิดไป หรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้
5. ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้น หรืออยู่ในรูปของวงจรรวม (ASIC)
6. หลังจากที่ได้วงจรจริงมาแล้ว ยังต้องมีความจำเป็นที่ต้องตรวจสอบการทำงานที่คำนึงถึงเวลา ด้วย เพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิทัล เพราะในขั้นตอนนี้วงจรที่ออกแบบ จะประกอบด้วยอินพุตและเอาต์พุตแพด (Pad) ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก
7. หลังจากที่น่าวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิทัลแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆ อีกครั้ง เป็นการควบคุมคุณภาพของผลิตภัณฑ์

บทที่ 3

เอฟพีจีเอ

ความก้าวหน้าของอุตสาหกรรมอิเล็กทรอนิกส์ปัจจุบันทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆ มากมายซึ่งทำให้เกิดการลดค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโพรเซสเซอร์และหน่วยความจำปัจจุบัน ทุกๆ ครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างวงจรรวมและไอซีมาตรฐานมากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน ฟังก์ชันลอจิกที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิทัลได้พิจารณาถึงการผลิตให้ขนาดหลายๆ และการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวม จะแบ่งตามการสร้างออกเป็น 2 กลุ่ม คือ Field programmable และ Mask programmable ดังแสดงในรูปที่ 3.1



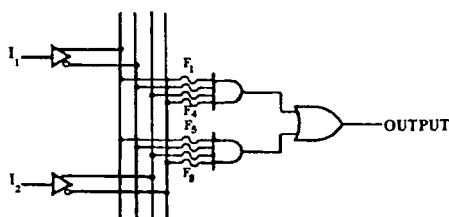
รูปที่ 3.1 แสดงการแบ่งกลุ่มของวงจรรวม ASIC

3.1 Field Programmable

อุปกรณ์วงจรรวม ASIC แบบ field programmable มีอยู่มากมายหลายชนิด แต่มีลักษณะการสร้างหรือกำหนดการทำงานของวงจรที่เหมือนกัน กล่าวคือ ผู้ใช้งานสามารถออกแบบและสร้างวงจรที่ต้องการใช้ลงในตัวอุปกรณ์ได้เองโดยไม่ต้องไปโรงงานเพื่อผลิต โดยเฉพาะอย่างยิ่งในปัจจุบันนี้มีเครื่องมือที่ใช้ช่วยในการออกแบบ และสร้างวงจรร่วมกับไมโครคอมพิวเตอร์ที่มีความสามารถสูงในการพัฒนาตั้งแต่ขั้นการออกแบบ การจำลองการทำงาน จนถึงจัดสร้างวงจรลงในอุปกรณ์ รวมทั้งอุปกรณ์ field programmable เหล่านี้สามารถหาซื้อได้ง่ายทำให้การสร้างวงจรรีเลย์ทรอนิกส์จนถึงระบบไมโครโพรเซสเซอร์หันมาใช้อุปกรณ์จำพวกนี้ เป็นอุปกรณ์ประกอบในวงจรแทนอุปกรณ์ย่อยๆ แยกชิ้น (Discrete component) มาก

3.1.1 พีแอลดี (PLD: Programmable Logic Device)

ภายในอุปกรณ์พีแอลดีถูกเตรียมเป็นวงจรพื้นฐานทางด้านลอจิกต่อกันอยู่เป็นกลุ่มมีทั้ง วงจรคอมบิเนชัน (Combination) และซีเควนเชียล (Sequential) ซึ่งมีส่วนประกอบเป็นวงจรภายใน เทคโนโลยีของวงจรที่ใช้สร้างพีแอลดีมีทั้ง ทีทีแอล (TTL) อีซีแอล (ECL) และ ซีเอ็มอส (CMOS) ตามความเหมาะสมของแต่ละระบบ อุปกรณ์พีแอลดีทุกชนิดมีหลักการพื้นฐานของวงจรภายในที่ เหมือนกันโดยมีวงจรหลักเป็นวงจรคอมบิเนชันที่ให้ผลเป็นผลคูณร่วมบวก (Sum of product) ประกอบไปด้วยชุดของแอนด์เกตที่ต่อร่วมกับออคเกตการโปรแกรมคือ การเลือกที่จะให้มีการต่อ อินพุตภายในของแอนด์เกตกับสัญญาณอินพุตใดบ้างซึ่งมีทั้งจากภายนอกและสัญญาณป้อนกลับ จากเอาต์พุตภายในเอง การติดต่ออินพุตของออคเกตกับเอาต์พุตของ แอนด์เกต ตัวต่างๆ วิธีการเลือก หรือการโปรแกรมทางกายภาพ อินพุตต่างๆ ของอุปกรณ์ทุกตัวจะถูกต่อผ่านฟิวส์เข้ากับแหล่ง สัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใดจะตัดฟิวส์ทำให้สามารถโปรแกรมได้ครั้งเดียว อุปกรณ์พี แอลดีบางชนิดใช้มอสทรานซิสเตอร์แทนฟิวส์ทำให้สามารถโปรแกรมโดยใช้กระแสไฟฟ้า และ สามารถลบและโปรแกรมใหม่เข้าไปได้อีก

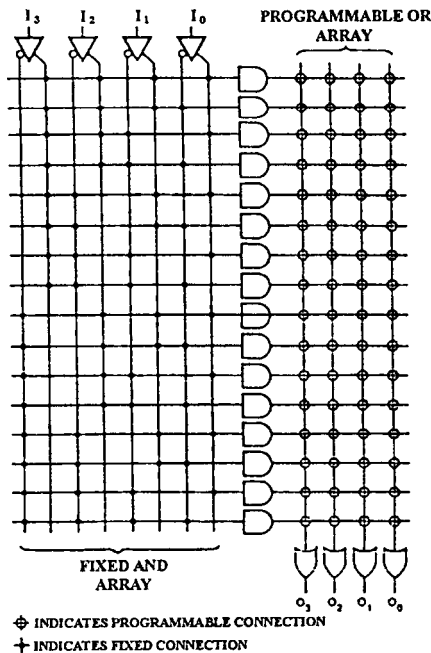


รูปที่ 3.2 แสดงวงจรพื้นฐานของอุปกรณ์พีแอลดีซึ่งอยู่ในรูปผลคูณร่วมบวก

3.1.2 พรอม (PROM: Programmable Read Only Memory)

พรอมคือหน่วยความจำรอม (ROM) ที่โปรแกรมได้ ซึ่งนับว่าเป็นอุปกรณ์พีแอลดี ชนิดหนึ่งซึ่งวงจรภายในของพรอมเสมือนกับประกอบไปด้วยแถวลำดับของแอนด์เกตและออคเกต (And - Or Array) ผลเอาต์พุตที่ขาเอาต์พุตสามารถแสดงในสมการของฟังก์ชันผลคูณร่วมบวก (Sum of product) ของสัญญาณอินพุตที่ขาแอดเดรส รูปที่ 3.3 แสดงถึงลักษณะการต่อเป็นแถวลำดับของ แอนด์เกตและออคเกตของพรอมขนาด 16x4 บิต วงจรทางด้านซ้ายบนสุดเป็น แอนด์เกตที่ให้ผลเป็น ผลคูณ (Product) ของกรณีอินพุตเป็น 0000 แอนด์เกตที่อยู่ถัดลงมาเป็นผลคูณของกรณีที่อินพุต เป็น 0001, 0010, ... จนถึงตัวล่างสุดคือผลคูณในกรณีที่อินพุตเป็น 1111 ที่เอาต์พุตแต่ละบิตของหน่วย ความจำ สามารถเลือกได้ว่าจะให้เป็น 1 ในกรณีที่อินพุตจากแอดเดรส เป็นอย่างไรบ้างเหมือนกัน เป็นจากนำเอาต์พุตจากผลคูณที่ต้องการให้เอาต์พุตแต่ละบิตเป็น 1 ไปออคกันจึงเปรียบเหมือนกับว่า ในพรอมมีจำนวนแอนด์เกตเท่ากับจำนวนตำแหน่งความจำและมีออคเกตจำนวนเท่ากับจำนวนบิต

ของสัญญาณข้อมูลออก (Data output) อินพุตของอเกตทุกตัวสามารถต่อเข้ากับแอนด์เกตตัวใดก็ได้ทุกตัว ซึ่งอาจเรียกได้ว่าเป็นพีแอลดีแบบ fixed AND/programmable OR



รูปที่ 3.3 แสดงลักษณะของพอรอมเมื่อเปรียบเทียบเป็นวงจรในรูปผลคูณร่วมบวก

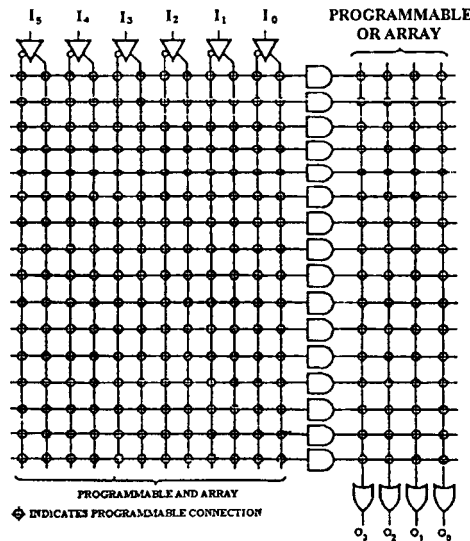
3.1.3 พีเอแอล (PAL: Programmable Array Logic)

ในช่วงกลางปี ค.ศ. 1970 บริษัทเอ็มเอ็มไอ (MMI: Monolithic Memory) ในประเทศสหรัฐอเมริกา ได้พัฒนาอุปกรณ์พีเอแอล เป็นพีแอลดีชนิดใหม่โดยใช้เทคโนโลยีแบบแอลเอสไอ (LSI: Large Scale Integration) สามารถโปรแกรมเลือกวงจรภายใน โดยใช้ฟิวส์ที่เชื่อมต่ออยู่ระหว่างสัญญาณอินพุตภายนอกและการป้อนกลับจากภายในกับแอนด์เกตที่ต่อเป็นฟังก์ชันผลคูณ (Product) อยู่ในตัววงจรรวม

3.1.4 พีแอลเอ (PLA: Programmable Logic Array)

อุปกรณ์ที่สามารถโปรแกรมได้แบบพีแอลเอเกิดขึ้นเมื่อปี ค.ศ. 1975 โดยบริษัทซิกเนติกส์ (Signetics) สหรัฐอเมริกา ซึ่งเป็นบริษัทผู้ผลิตวงจรรวมภายในใหญ่รายหนึ่ง ผลิตและนำเสนออุปกรณ์โดยใช้ชื่อว่า เอฟพีแอลเอ (FPLA: Field Programmable Logic Array) สามารถโปรแกรมการต่อลอจิกทั้งทางด้านแอนด์เกตและอเกตได้ และยังสามารถเลือกเอาต์พุตเป็น active high หรือ active low โดยต่อผ่านเอ็กคูลส์บอเกต ให้ทำหน้าที่เป็นนอนอินเวอร์เตอร์หรือเป็น อินเวอร์เตอร์แล้วแต่ภายในของพีแอลเอต่อมาปี ค.ศ. 1979 บริษัทซิกเนติกส์ ได้สร้างเอฟพีแอลเอใหม่ที่มี

เรจิสเตอร์ต่ออยู่ภายในวงจร เพิ่มขึ้นรวมทั้งสามารถเลือกสัญญาณอินพุตที่มาจากการป้อนกลับจากเรจิสเตอร์ได้ด้วย ทำให้สามารถใช้อุปกรณ์พีแอลเอใหม่นี้สร้างวงจร State machine ได้ อุปกรณ์ใหม่ที่มีเรจิสเตอร์ อยู่ด้วยนี้ถูกตั้งชื่อใหม่เป็น เอฟพีแอลเอส (FPLS: Field Programmable Logic Sequencer) มีทั้งที่เป็นทีทีแอลและซีมอส



รูปที่ 3.4 แสดงวงจรพื้นฐานภายในของพีแอลเอ

3.1.5 แอลซีเอ (LCA: Logic Cell Array)

อุปกรณ์ชนิดนี้ถูกสร้างขึ้นเมื่อประมาณปี ค.ศ. 1986 โดยบริษัทไซริง (XILINX Inc.) ซึ่งเป็นบริษัทที่ร่วมทำการค้นคว้ากับบริษัทเอ็มเอ็มไอ (MMI) สร้างเป็นอาเรย์ที่ประกอบด้วยเกตจำนวน 1,200-1,800 เกต มีลักษณะของสถาปัตยกรรมที่ใกล้เคียงกับเกตอาเรย์ (Gate array) โดยโปรเซสแบบซีมอส 1.6 ไมครอนชั้นโลหะคู่ (CMOS 1.6 microns double-layer metal) สามารถโปรแกรมและลบได้โดยใช้กระแสไฟฟ้า (Static RAM based) ภายในจัดเรียงเป็นเมทริกซ์ของลอจิกเซลล์ ล้อมรอบภายนอกด้วยอินพุต เอาต์พุตเซลล์ อุปกรณ์แอลซีเอตัวแรกของบริษัทไซริงคือ แอลซีเอเบอร์ 2064 ประกอบด้วยเซลล์เรียงเป็นเมทริกซ์มีจำนวน 64 เซลล์ แต่ละเซลล์เรียกว่า ซีแอลบี (CLB: Configurable Logic Block) แต่ในปัจจุบันได้พัฒนามาอยู่ในรูปของ เอฟพีจีเอ (FPGA: Field Programmable Gate Array) ซึ่งมีประสิทธิภาพ ความจุของเกตสูงมากขึ้น โดยสร้างออกมาเป็นอนุกรม (Series) ต่างๆ เช่น ตระกูล XC3000 และ ตระกูล XC4000 เป็นต้น

3.1.6 อีพีแอลดี (EPLD: Erasable Programmable Logic Device)

อีพีแอลดีเป็นอุปกรณ์สามารถโปรแกรมได้ที่สามารถลบและทำการโปรแกรมใหม่ได้ เพื่อใช้ทำงานจรต้นแบบ ตัวอย่างได้แก่ พีแอลดี ในอนุกรมอีพี (EP series) ของบริษัท อัลเทอร์รา (Altera Inc.) ประเทศสหรัฐอเมริกาซึ่งเป็นบริษัทที่ผลิตอีพีแอลดี เป็นรายแรกโดยเริ่มเมื่อปี ค.ศ. 1984 เป็น พีแอลดี ที่ใช้โปรเซสเหมือนกับซีมอสอีพรอม (CMOS EPROM) คือใช้มอสทรานซิสเตอร์เชื่อมต่อระหว่างสัญญาณอินพุตกับจุดที่ต้องการแทนการการใช้ฟิวส์ดั้งเดิม ทำให้สามารถโปรแกรมการต่อวงจรภายในอุปกรณ์ด้วยการจ่ายไฟฟ้าตามขนาดที่กำหนดเข้าไปยังตัวอุปกรณ์ และลบได้โดยใช้แสงอุลตราไวโอเลตฉายผ่านช่องหน้าต่างกระจกเข้าไปตกกระทบในตัวชิปของอุปกรณ์

3.2 Mask programmable

การใช้งานวงจรรวม ASIC ในเชิงพาณิชย์ จำเป็นต้องใช้วงจรรวม ASIC แบบ Mask programmable เนื่องจากต้นทุนต่อหนึ่งตัวต่ำกว่าวงจรรวม Field programmable ASIC ในกรณีที่ปริมาณการผลิตสูงนับพันนับหมื่นตัวขึ้นไป ตัวอย่างเช่น วงจรอีพีแอลดี ตัวหนึ่งอาจสูงถึงหนึ่งพันบาท ในขณะที่ถ้าผลิตวงจรรวมที่มีคุณสมบัติเหมือนกันทุกประการโดยใช้ Mask programmable แล้ว ราคาตัวหนึ่งจะลดลงเหลือเพียงไม่ถึงหนึ่งร้อยบาท การใช้วงจรรวมแบบ Mask programmable จึงมีบทบาทสำคัญในการผลิตสินค้าอิเล็กทรอนิกส์ในเชิงพาณิชย์ในปัจจุบัน

วงจรรวมประเภทนี้ หลังจากผู้ใช้ออกแบบวงจรและตรวจสอบการทำงานจนเป็นที่น่าพอใจแล้ว ต้องส่งให้ผู้ผลิตทำการเจียสกร ไม่สามารถโปรแกรมได้ด้วยตนเองเหมือนกับวงจรรวมแบบ Field programmable ช่วงเวลาการผลิตออกใช้งานจึงใช้เวลานานนับเดือนและมีค่าใช้จ่ายเบื้องต้นในการเจียสกรสูง วงจรรวมแบบ mask programmable ASIC ในปัจจุบันได้แก่ เกตอาเรย์, เซลล์มาตรฐาน และฟูลคัสตัม (full custom)

3.2.1 เกตอาเรย์ (Gate Array)

วงจรรวมนี้ประกอบด้วยแถวลำดับของวงจรเกต ซึ่งอาจจะเป็นวงจรเกตประเภทเดียวกันหรือต่างชนิดกันก็ได้ กับขั้วต่อสายไฟ (Pad) สำหรับต่อกับวงจรรายนอก ผู้ใช้มีหน้าที่ออกแบบการเชื่อมโยงทางไฟฟ้าระหว่างวงจรเกตแต่ละตัวและขั้วต่อสายไฟเพื่อให้ทำหน้าที่ตามต้องการ แล้วจึงส่งผลการออกแบบนั้นไปยังโรงงานผู้ผลิตวงจรรวมเกตอาเรย์ นั้นไปทำการเจียสกรต่อไป โดยทั่วไปแล้ว ปรากฏในการใช้วงจรรวมเกตอาเรย์สร้างวงจรที่ต้องการใช้งานทั่วไปจะมีเพียง 25-30% ของพื้นที่ซิลิคอนที่ใช้สำหรับวงจรเกต ส่วนพื้นที่ที่เหลือจะใช้กับการเชื่อมโยงวงจรเกตเข้าด้วยกัน

เองและเข้ากับขั้วต่อสายไฟ พื้นที่ซิลิคอนจะใช้ประโยชน์สูงถึง 75% สำหรับวงจรเชิงเลขที่มีลักษณะสม่ำเสมอ เช่น หน่วยความจำ เป็นต้น

3.2.2 เซลล์มาตรฐาน (Standard Cell)

ปัญหาการใช้พื้นที่ซิลิคอนอย่างมากมาจากการเชื่อมโยงวงจรของวงจรรวม เกตอาเรย์ทำให้เกิดขีดจำกัดในความซับซ้อนของวงจรที่ใช้ ประกอบกับผู้ใช้จำนวนมากต้องการรวบรวมวงจรรวมมาตรฐาน อาทิ วงจรตระกูล 7400 วงจรตระกูล 4000 จำนวนหลายตัวเข้าเป็นวงจรรวม ASIC เพียงตัวเดียว ทำให้เกิดวงจรรวม ASIC แบบเซลล์มาตรฐานขึ้น วงจรรวม เซลล์มาตรฐานนี้ ผู้ใช้เป็นผู้เลือกกลุ่มวงจรทำหน้าที่ต่างๆ เช่น เกต ฟลิปฟล็อป ตัวนับ ตัวเลื่อน หน่วยความจำ หรือ กระทั่งไมโครโปรเซสเซอร์จากแฟ้มข้อมูล (Library) ของผู้ผลิต ซึ่งอาจจะเป็นแฟ้มข้อมูลคอมพิวเตอร์ เหมาะสมสำหรับวงจรเชิงเลขที่มีความซับซ้อน การผลิตวงจรรวม เซลล์มาตรฐานจะมีต้นทุนสูงกว่าวงจรรวมเกตอาเรย์และใช้เวลาในการออกแบบและเจือสารยาวกว่าสองถึงสามเท่าตัว วงจรรวมเซลล์มาตรฐานจึงเหมาะสมกับการใช้งานเชิงพาณิชย์ที่มีปริมาณการผลิตนับหมื่นตัวขึ้นไป บทบาทวงจรรวมเซลล์มาตรฐานจะมีเพิ่มมากขึ้นในอนาคต เมื่อต้นทุนการเจือสารลดลง

3.2.3 ฟูลล์คัสตัม (Full Custom)

วงจรรวมฟูลล์คัสตัม นี้ผู้ใช้เป็นผู้ออกแบบเองทั้งหมด ตั้งแต่ระดับวงจรเชิงเลขจนกระทั่งถึงระดับกายภาพ แล้วจึงส่งข้อมูลการออกแบบไปให้ผู้ผลิตเจือสารในรูปแฟ้มข้อมูลมาตรฐาน เช่น GDS II, CIF การใช้ฟูลล์คัสตัม ASIC นี้เท่าที่แพร่หลายอยู่ในปัจจุบัน จะนำไปเพื่อการศึกษาและการวิจัยและเพื่อการผลิตจำนวนน้อย ส่วนใหญ่จะเป็นการเจือสารในลักษณะที่ใช้ค่าใช้จ่ายร่วมกัน กล่าวคือ รวบรวมการออกแบบหลายวงจรลงบนแผ่นซิลิคอนเดียวกันเพื่อประหยัดค่าเจือสาร

รูปแบบการเจือสารวงจรรวมฟูลล์คัสตัมที่ประกอบด้วยการออกแบบหลายวงจรรวมอยู่บนแผ่นซิลิคอนเดียวกัน มีกระทำอยู่ 3 รูปแบบดังนี้

1. Multi-project wafer (MPW) การเจือสารวงจรรวมฟูลล์คัสตัมแบบนี้ แผ่นเวเฟอร์ (Wafer) จะแบ่งเป็นส่วนๆ ที่เรียกว่าได (Die) และในแต่ละชุดจะเจือสารวงจรต่างกัน ผู้ผลิตที่ให้บริการแบบนี้ได้แก่ โมซิส (MOSIS) และ ออบิต (ORBIT) ในสหรัฐอเมริกา

2. Multi-project chip (MPC) เป็นการเจือสารการออกแบบหลายวงจรลงบนไดชุดเดียวกัน โดยที่ไดทุกชุดบนเวเฟอร์เดียวกันจะมีลักษณะเหมือนกัน การจัดวางวงจรที่ออกแบบลงบนได เน้นการใช้พื้นที่ซิลิคอนให้เป็นประโยชน์สูงสุดเป็นสำคัญ ตัวอย่างผู้ผลิตที่ให้บริการแบบนี้ได้แก่ อาวา (AWA) ในออสเตรเลีย และ อีเอสทู (ES II) ในสหราชอาณาจักร

3. Multi-project reticle (MPR) เป็นการเจือสารในลักษณะผสมผสานระหว่างฟูลล์คัสตัมกับเซลล์มาตรฐาน กล่าวคือ ใดจะแบ่งออกเป็นส่วนๆ อย่างสม่ำเสมอแต่ละส่วนบรรจุการออกแบบแต่

ละวงจร โดยไอซูดหนึ่งจะบรรจุวงจรที่ออกแบบได้ประมาณ 8 วงจร และไอซูดทุกชุดในแผ่นเวเฟอร์จะมีลักษณะเหมือนกัน ในปัจจุบันมีเพียงอวาที่ให้บริการการเจือสารลักษณะนี้ โดยจำกัดเฉพาะประเภทเซลล์มาตรฐานเท่านั้น

3.3 เอฟพีจีเอ (FPGA: Field Programmable Gate Array)

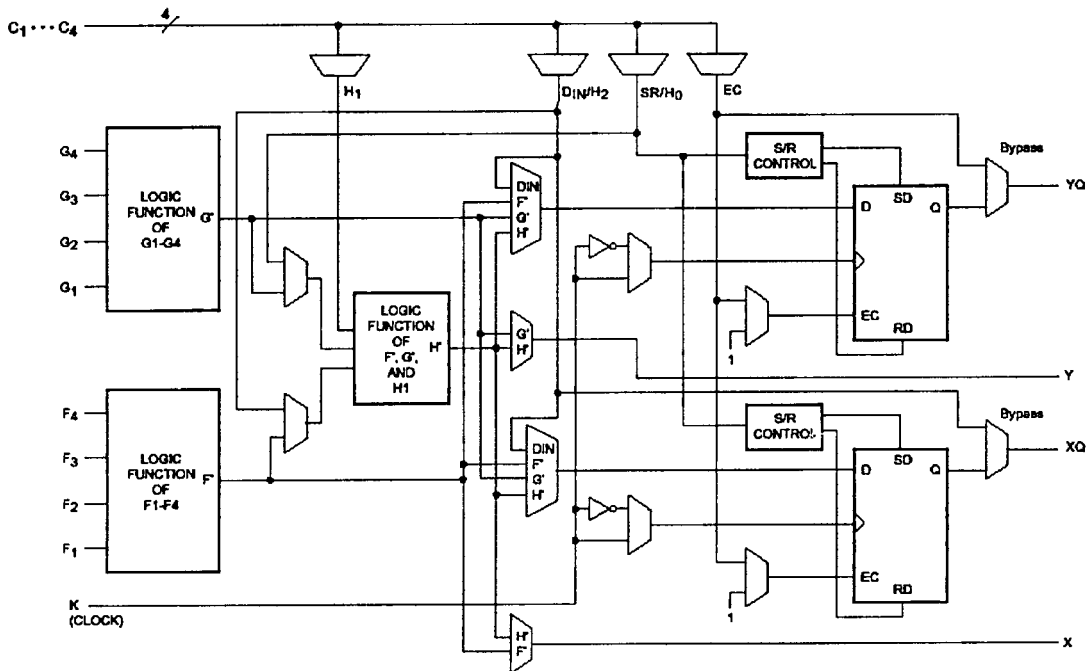
เป็นอุปกรณ์ที่ถูกพัฒนาต่อจากอุปกรณ์แอลซีเอของบริษัทยูนิเซกซ์ (XILINX Inc.) โดยมีประสิทธิภาพการทำงานและมีปริมาณความหนาแน่นของเกตสูง สามารถจะกำหนดฟังก์ชันการทำงานได้ความต้องการของผู้ใช้โดยผ่านการโปรแกรมเอฟพีจีเอได้รวบรวมข้อดีทั้งหมดของการทำคัสตัมวีแอลเอสไอ (Custom VLSI) มารวมไว้ทั้งหมดได้แก่ การออกแบบการผลิต, ลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรเป็นอย่างมาก นักออกแบบเพียงกำหนดฟังก์ชันการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้เอฟพีจีเอ สามารถออกแบบและทดสอบภายในเวลาเพียง 2-3 วัน เท่านั้น ตรงกันข้ามกับการออกแบบโดยใช้เกตอาร์เรย์ ซึ่งใช้เวลาหลายอาทิตย์ การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกัน จากประโยชน์ของเอฟพีจีเอ ดังกล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์ ลดค่าเอ็นอาร์อี (NRE: Nonrecurring Engineering Cost) ลงไปด้วย

3.3.1 สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล XC4000

สถาปัตยกรรมภายในคล้ายๆ กับเกตอาร์เรย์โดยทั่วไป ภายในมีลักษณะเป็นเมทริกซ์ของลอจิกบล็อก (Logic block) และล้อมรอบไปด้วยบล็อกการเชื่อมต่อของไอโอ (I/O Interface block) การเชื่อมต่อระหว่างซีแอลบี (CLB: Configuration Logic Block) และไอโอบี (IOB: Input Output Block) ทำได้โดยผ่านช่องที่วางพาดผ่านระหว่างแถว (Row) และคอลัมน์ (Column) มีการทำงานเหมือนกันไมโครโพรเซสเซอร์ ตัวแอลซีเอจะทำงานได้ต้องใช้ Program-driven logic device หน้าที่ของซีแอลบีและไอโอบีแต่ละตัว การเชื่อมต่อภายใน (Interconnection) ถูกกำหนดไว้ในโปรแกรมคอนฟิกูเรชัน (Configuration program) หรือเก็บไว้ในอีพรอม ภายในแอลซีเอโปรแกรมจะถูกโหลดเข้าสู่แอลซีเอเมื่อมีการจ่ายไฟ (Power-up) โดยทางคำสั่ง (Command) ซึ่งเป็นส่วนหนึ่งของการเริ่มต้นระบบ (System initialization) ประสิทธิภาพของแอลซีเอกำหนดโดย ความเร็วของลอจิกส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่างๆ ความเร็วของอัตราของระบบสัญญาณนาฬิกา (System clock rate) ถูกกำหนดด้วย ทอกเกิลฟลิปฟล็อป สำหรับการประยุกต์ใช้โดยทั่วไปจะอยู่ที่ประมาณ 1/3 ถึง 1/2 ค่าสูงสุดของทอกเกิลเกต (Maximum toggle gate)

3.3.1.1 ซีแอลบี (CLB: Configuration Logic Block)

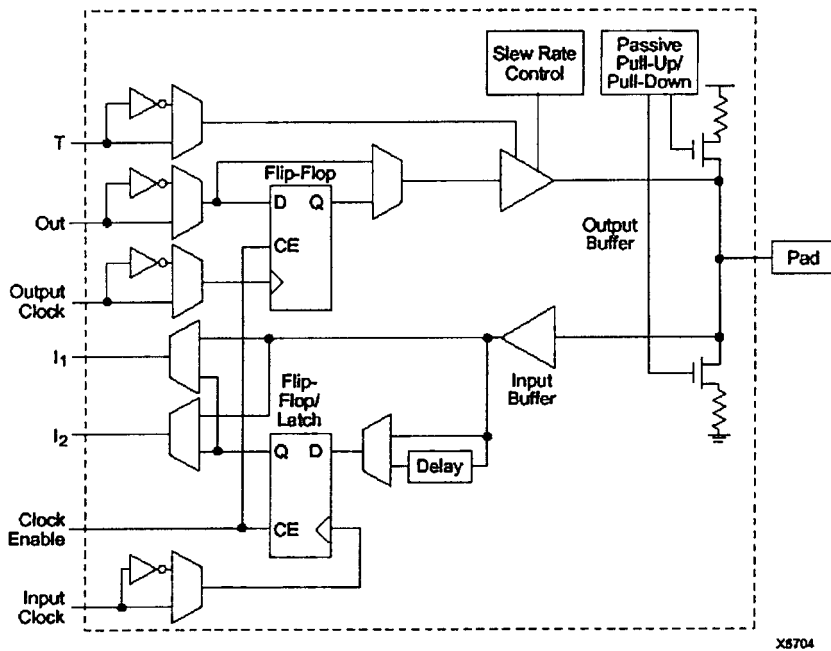
ภายในแอลซีเอคือเมทริกซ์ของซีแอลบีแต่ละตัวประกอบด้วยหน่วยของคอมบิเนชันลอจิกที่สามารถโปรแกรมได้ (Programmable combination logic) และส่วนของเรจิสเตอร์เก็บข้อมูล (Storage register) ส่วนของวงจรคอมบิเนชันลอจิกสามารถใช้สร้างวงจรทางด้านฟังก์ชันบูลีนของอินพุต ส่วนเรจิสเตอร์รับค่าจากส่วนคอมบิเนชันลอจิกหรือโดยตรงจากเอาต์พุตของซีแอลบี สามารถจับวงจรคอมบิเนชันลอจิกโดยตรงผ่านเส้นทางเดินย้อนกลับ (Feedback path)



รูปที่ 3.5 แสดงผังวงจรภายในของซีแอลบีของเอฟพีจีเอตระกูล XC4000

3.3.1.2 ไอโอบี (IOB: Input Output Block)

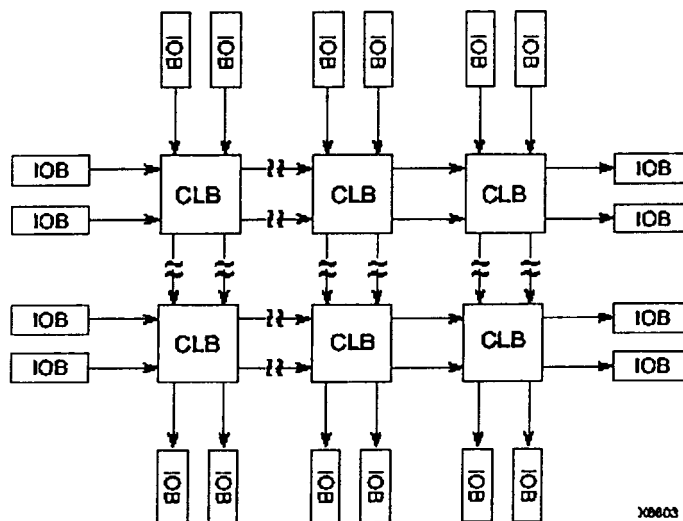
เป็นส่วนติดต่อกับวงจรภายนอกของแอลซีเอสร้างมาจากส่วนของอุปกรณ์อินพุต/เอาต์พุตที่สามารถโปรแกรมได้ (Programmable Input/Output device) แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้เป็นอินพุต/เอาต์พุตแบบ 3 สถานะหรือไอโอแบบสองทิศทางก็ได้ โดยอินพุตสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอลและซีมอสเทรคโฮล ของไอโอบี แต่ละตัวมีฟลิปฟล็อปสามารถใช้เป็นบัฟเฟอร์สำหรับอินพุตและเอาต์พุต



รูปที่ 3.6 แสดงผังวงจรของไอ โอบีภายในเอฟพีจีเอตระกูล XC4000

3.3.1.3 อินเทอร์เน็ตคอนเน็ค (Interconnect)

ความยืดหยุ่นของการใช้แอลซีเอมาทำเป็นอุปกรณ์ขึ้นอยู่กับโปรแกรม ทรัพยากรต่างๆ ที่อยู่ภายในเข้าด้วยกันการที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปเหมือนกับเกตอาเรียต่างๆ ไป การเชื่อมต่อภายในแอลซีเอประกอบด้วยเน็ตเวิร์ค 2 ทิศทางคือทาง แกวและคอลัมน์ซึ่งวางอยู่ระหว่าง CLB programmable switch จะทำการเชื่อมต่ออินพุตและเอาต์พุตของไอโอบีและซีแอลบีที่จุดต่อร่วมระหว่างแอกกับคอลัมน์สามารถสลับสัญญาณจาก เส้นทางไปยังส่วนต่างๆ



รูปที่ 3.7 แสดงเส้นทางการเชื่อมต่อระหว่างไอโอบีกับซีแอลบีของเอฟพีจีเอตระกูล XC4000

3.3.2 คุณสมบัติโดยทั่วไปของเอฟฟิเจ็ตระกูล XC4000

1. เป็นอุปกรณ์รุ่นที่สามของเอฟฟิเจ็เอ

- มีฟลิปฟล็อปเป็นจำนวนมาก
- ในการผลิตฟังก์ชันของการทำงานมีความยืดหยุ่นสูง
- มีจำนวนเกตภายในจำนวน 2,000 – 10,000 เกต
- เพิ่มความสามารถพิเศษของเรจิสเตอร์และอินพุต/เอาต์พุต
- มีค่าแฟนเอาต์ (fan-out) สูง
- มีบัสภายใน 3 สถานะ
- ทำงานกับสัญญาณที่ทีแอลและซีมอส
- มีออสซิลเลเตอร์แอมพลิฟลายเออร์ภายใน
- มีแรมภายในความเร็วสูง (< 25 Ns)
- ใช้กับงานที่ต้องการความเร็วสูง (ใช้งานได้ที่ความถี่ 70/100/125 MHz)
- มี Wide edge decoder
- เส้นทางการเชื่อมต่อ (Interconnect line) เป็นแบบลำดับชั้น
- มีการกระจายกำลังงานของสัญญาณต่ำ

2. มีสถาปัตยกรรมภายในที่ยืดหยุ่น

- มีลอจิกบล็อกและไอโอบล็อกที่สามารถโปรแกรมได้
- มีอินเตอร์คอนเน็คและ Wide decoder ที่โปรแกรมได้

3. ทำกระบวนการจับไมครอนชนิดซีมอสได้

- มีลอจิกและอินเตอร์คอนเน็คที่มีความเร็วสูง
- ใช้กำลังงานต่ำ

4. คุณลักษณะทาง System-Oriented

- รองรับมาตรฐาน IEEE 1149.1 ในการทำ boundary-scan logic
- สามารถโปรแกรมค่า output slew rate ได้
- สามารถโปรแกรมให้อินพุตมีลักษณะพูลอัพ (Pull-up) หรือ พูลดาวน์ (Pull-down) เรจิสเตอร์ได้
- ให้กระแสเอาต์พุตได้ตั้งแต่ 12-24 มิลลิแอมป์ (ขึ้นอยู่กับแต่ละรุ่น)

5. ทำการโหลดเอาต์พุตเพิ่มข้อมูลประเภทไบนารี

- ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
- มีโหมดในการโปรแกรมให้เลือก 6 โหมด

6. มีโปรแกรมช่วยพัฒนาได้แก่ XACT Development System ที่ทำงานบน

คอมพิวเตอร์รุ่นต่างๆ เช่น '486/Pentiums, NEC PC, Apollo, Sun-4, HP700

- สามารถติดต่อกับโปรแกรมอื่นๆ ได้ เช่น Viewlogic, Mentor Graphic และ OrCAD เป็นต้น
- มีโปรแกรมการวางและเชื่อมโยงอุปกรณ์ภายในแบบอัตโนมัติ (automatic place and routing) ที่ครบสมบูรณ์
- มี Interactive Design Editor ที่ใช้สำหรับการทำ optimization
- มี 288 มาโคร 34 ฮาร์ดมาโคร และ แรม/รอมคอมพายเลอร์

ตารางที่ 3.1 แสดงรายละเอียดของอุปกรณ์ภายในเอฟพีจีเอตระกูล XC4000

Device	Max Logic Gate (No RAM)	Max. RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total Logic Blocks	Number of Flip-Flops	Max. Decode Input per side	Max. User I/O
XC4003E	3,000	3,200	2,000-5,000	10X10	100	360	30	80
XC4005E/L	5,000	6,272	3,000-9,000	14X14	196	616	42	112
XC4006E	6,000	8,192	4,000-12,000	16X16	256	768	48	128
XC4008E	8,000	10,368	6,000-15,000	18X18	324	936	54	144
XC4010E/L	10,000	12,800	7,000-20,000	20X20	400	1,120	60	160
XC4013E/L	13,000	18,432	10,000-30,000	24X24	576	1,536	72	192
XC4020E	20,000	25,088	13,000-40,000	28X28	784	2,016	84	224
XC4025E	25,000	32,768	15,000-45,000	32X32	1,024	2,560	96	256
XC4028EX/XL	28,000	32,768	18,000-50,000	32X32	1,024	2,560	96	256
XC4036EX/XL	36,000	41,472	22,000-65,000	36X36	1,296	3,168	108	288
XC4044EX/XL	44,000	51,200	27,000-80,000	40X40	1,600	3,840	120	320
XC4052XL	52,000	61,952	33,000-100,000	44X44	1,936	4,576	132	352
XC4062XL	62,000	73,728	40,000-130,000	48X48	2,304	5,376	144	384
Larger Devices Available in the First Half of 1997								

* Max value of Typical Gate Range include 20-30% of CLBs used as RAM

3.4 การโปรแกรมเอฟพีจีเอตระกูล XC4000

คือกระบวนการในการโหลดข้อมูลในโปรแกรมไปยังแอลซีเอ เพื่อกำหนดหน้าที่ของการทำงานในแต่ละบล็อกภายใน และการเชื่อมต่อ ซึ่งเอฟพีจีเอตระกูล XC4000 จะต้องใช้ข้อมูลเกี่ยวกับการโปรแกรมประมาณ 350 บิตต่อซีแอลบี โดยแต่ละบิตจะบอกถึงสถานะของหน่วยความจำสแตติกที่ควบคุมบิตในการควบคุมตารางฟังก์ชัน (function table bit) และมัลติเพลกซ์อินพุตหรือการเชื่อมต่อกันระหว่างทรานซิสเตอร์

3.4.1 โหมดการโปรแกรม

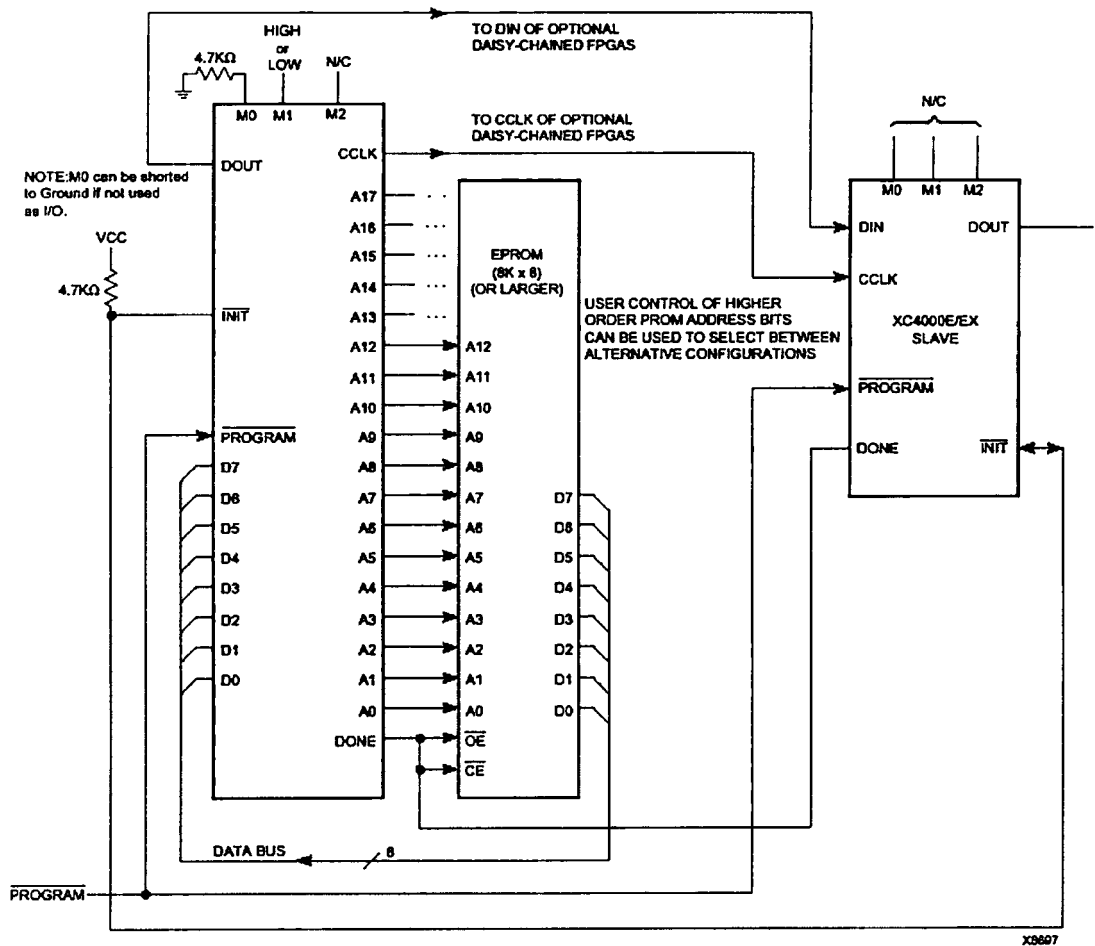
เอฟพีจีเอในตระกูล XC4000 มีโหมดการโปรแกรม 6 โหมด ซึ่งแต่ละโหมดจะถูกกำหนดโดยจากบิตโหมด ได้แก่ บิต M0, M1 และ M2 โดยมีโหมดการโปรแกรมดังตารางที่ 3.2

ตารางที่ 3.2 แสดงรูปแบบของโหมดต่างๆ ในการโปรแกรมเอฟพีจีเอตระกูล XC4000

Mode	M2	M1	M0	Clock	Data
Master Serial	0	0	0	Output	Bit-serial
Slave Serial	1	1	1	Input	Bit-serial
Master parallel up	1	0	0	Output	Byte-Wide,00000
Master parallel down	1	1	0	Output	Byte-Wide,3FFFF
Peripheral Synch.	0	1	1	Input	Byte-Wide
Peripheral Asynch.	1	0	1	Output	Byte-Wide

3.4.1.1 โหมดหลัก (Master modes)

ในการทำงานในโหมดนี้ตัวแอลซีเอจะถูกโหลดข้อมูลโครงแบบ (configuration data) จากหน่วยความจำภายนอกเข้ามาโดยอัตโนมัติ มีโหมดที่แตกต่างกัน 8 โหมดโดยใช้ ช่วงเวลาภายในจ่ายให้ซีคล็อก (CCLK: Configuration clock) เพื่อที่จะเป็นฐานเวลาในการนำข้อมูลที่เข้ามาทางโหมดหลักแบบอนุกรม (serial master mode) และรับข้อมูลโครงแบบเข้ามาทางขาสัญญาณดิจิตอล (DIN: Data in) จากแหล่งสัญญาณเชิงโครนัสเช่น Xilinx serial configuration PROM, parallel master low and master high mode โดยจะรับข้อมูลแบบขนานมาจากบิต D0-D7 โดยสัมพันธ์กับแอดเดรสที่กำหนดโดยแอลซีเอดังรูปที่ 3.4 จะเป็นการแสดงตัวอย่างการเชื่อมต่อในโหมดหลักแบบขนาน (Master Parallel mode) โดยเริ่มต้นที่แอดเดรส 0000 ไบต์ข้อมูล (data byte) จะถูกอ่านเข้ามาแบบขนานทุกๆ สัญญาณอาร์คล็อก (RCLK: Read Clock) และส่งเข้าไปภายในแบบอนุกรมโดยอาศัยสัญญาณนาฬิกาโครงแบบ (configuration clock)

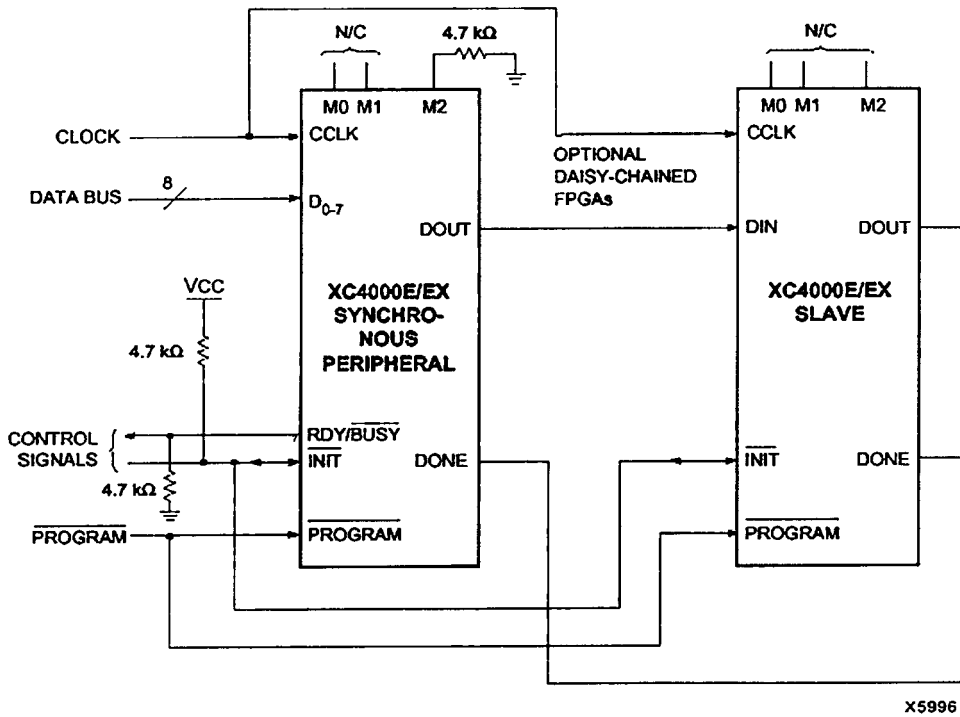


รูปที่ 3.8 แสดงผังวงจรการเชื่อมต่อเอฟฟี่อีเอนโมดหลักแบบขนาน

3.4.1.2 เพอริเฟอรัลโหมด (Peripheral modes)

โมดนี้จะรับข้อมูลเป็นไบต์จากบัส สถานะพร้อม (Ready) / กำลังทำงาน (Busy) จะมีอยู่ เพื่อให้สัญญาณต่างๆ สื่อสารกันได้มีประสิทธิภาพมากขึ้น

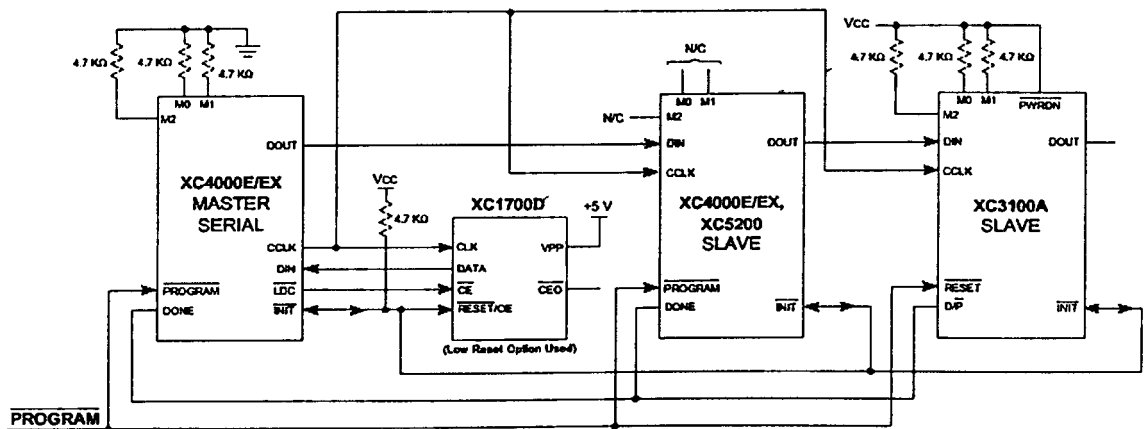
ในอะซิงโครนัสโมดนั้นออสซิลเลเตอร์ภายในจะสร้างสัญญาณ CCLK เพื่อรับข้อมูลที่เป็นไบต์ให้อยู่ในรูปแบบที่เหมาะสม และในซิงโครนัสโมดนั้นสัญญาณนาฬิกาภายนอกจะเป็นตัวควบคุมการรับข้อมูลที่เป็นไบต์



รูปที่ 3.9 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดเพอริเฟอรัลแบบอะซิงโครนัส
(Asynchronous Peripheral mode)

3.4.1.3 โหมดรอนแบบอนุกรม (Slave Series modes)

ในโหมดนี้แอลซีเอจจะรับข้อมูล โครแบบที่เป็นอนุกรมในขาขึ้นของสัญญาณ CCLK และหลังจากรับข้อมูลมาแล้วจะส่งข้อมูลเพิ่มเติมออกไปด้วยและแอลซีเอก็จะถูกทำการซิงโครไนซ์ในขาลงถัดไปของสัญญาณ CCLK



รูปที่ 3.10 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดรอนแบบอนุกรม

3.5 การใช้ความสามารถของแรมในเอฟพีจีเอตระกูล XC4000

แอลซีเอทำงานโดยใช้ตารางการค้นหา (Look-up table) ซึ่งจะทำให้การเก็บตารางที่ว่ามีในสแตติกแรม ซึ่งจะถูกเขียนในระหว่างการโปรแกรมโครงสร้างบนแอลซีเอและจะถูกอ่านในการโอเปอเรชั่น ดังนั้นแรมภายในจึงควรถูกรวมไว้ในการออกแบบของผู้ใช้ด้วย

หน้าที่ของแรมในเอฟพีจีเอตระกูล XC4000 มีหน้าที่คล้ายแรมโดยทั่วไป เช่น เอฟไอเอฟโอ (FIFO: First In First Out) แอลไอเอฟโอ (LIFO: Last In First Out) เรจิสเตอร์ไฟล์รวมทั้งแอปพลิเคชันบางอย่างอย่างเช่น เรจิสเตอร์เลื่อนข้อมูล (Shift register) แรมของเอฟพีจีเอตระกูล XC4000 มีความเร็วสูงเสมือนเอสแรม (SRAM) จึงไม่จำเป็นต้องคำนึงถึงเวลาหน่วงของการเชื่อมต่อ (Interconnection delay)

ตารางที่ 3.3 จำนวนของแรมภายในเอฟพีจีเอตระกูล XC4000

RAM Module	Equivalent Logic	XC4003	XC4005	XC4010
16 x 1	4 – input Function Generator (F or G)	200	392	800
32 x 1	Two – 4 – input Function Geranators and One 3 – input Function Geranator (F+G+H)	100	196	400

ตารางที่ 3.4 รายละเอียดของขาอุปกรณ์เฟิร์มแวร์ XC4010

Pad Name	Pin number	Pad Name	Pin Name
VCC	P2	O(M1)	P30
I/O(A8)	P3	GND	P31
I/O(A9)	P4	I(M0)	P32
I/O(A10)	P5	VCC	P33
I/O(A11)	P6	I(M2)	P34
I/O(A12)	P7	I/O,PGCK2	P35
I/O(A13)	P8	I/O(HDC)	P36
I/O(A14)	P9	I/O(LDC)	P36
I/O,SGCK1(A15)	P10	I/O	P37
VCC	P11	I/O	P38
GND	P12	I/O	P39
I/O,PGCK1(A16)	P13	I/O	P40
I/O(A17)	P14	I/O(INIT)	P41
I/O,TDI	P15	VCC	P42
I/O,TCK	P16	GND	P43
I/O,TMS	P17	I/O	P44
I/O	P18	I/O	P45
I/O	P19	I/O	P46
I/O	P20	I/O	P47
GND	P21	I/O	P48
VCC	P22	I/O	P49
I/O	P23	I/O	P50
I/O	P24	I/O,SGCK3	P51
I/O	P25	GND	P52
I/O	P26	DONE	P53
I/O	P27	VCC	P54
I/O	P28	<u>PROGRAM</u>	P55
I/O,SCGK2	P29	I/O (D7)	P56
O(M1)	P30	I/O, PGCK3	P57

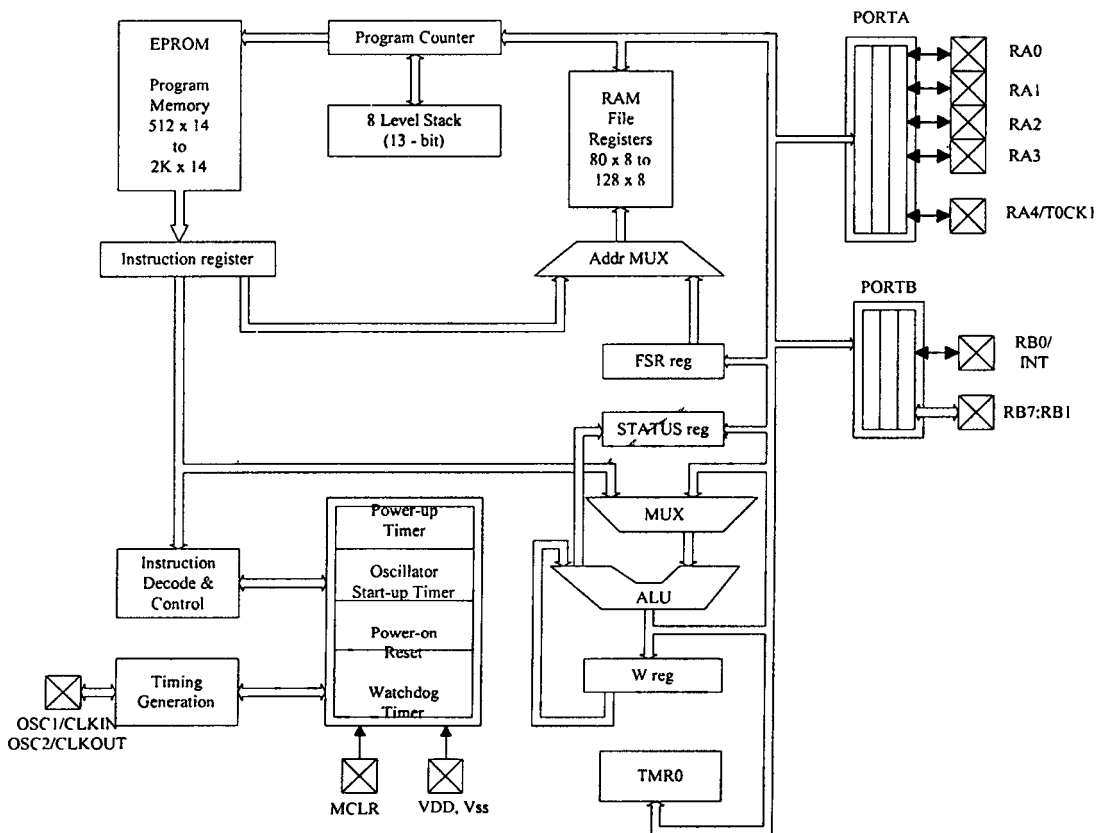
ตารางที่ 3.4 (ต่อ)

Pad Name	Pin number
I/O(D6)	P58
I/O(D5)	P59
I/O($\overline{\text{CS0}}$)	P60
I/O(D4)	P61
I/O	P62
VCC	P63
GND	P64
I/O(D3)	P65
I/O($\overline{\text{RS}}$)	P66
I/O(D2)	P67
I/O	P68
I/O(D1)	P69
I/O(RCLK,RDY/BUSY)	P70
I/O(D0,DIN)	P71
I/O,SGLK4(DOUT)	P72
CCLK	P73
VCC	P74
O, TDO	P75
GND	P76
I/O (A0,WS)	P77
I/O, PGCK4 (A1)	P78
I/O (CS1,A2)	P79
I/O(A3)	P80
I/O(A4)	P81
I/O(A5)	P82
I/O(A6)	P83
I/O(A7)	P84
GND	P1

บทที่ 4

โครงสร้างสถาปัตยกรรมไมโครคอนโทรลเลอร์

ในงานวิจัยนี้ได้เลือกเอาสถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูลพีไอซี มาเป็นสถาปัตยกรรมในการออกแบบไมโครคอนโทรลเลอร์ต้นแบบเพราะไมโครคอนโทรลเลอร์ตระกูลพีไอซีจัดอยู่ในไมโครโพรเซสเซอร์แบบรีสก์ทำให้มีชุดคำสั่งน้อยมากถ้าเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นๆ ซึ่งมีเพียง 36 คำสั่ง' และยังเป็นไมโครคอนโทรลเลอร์แบบชิปเดี่ยว (Single Chip) กล่าวคือภายในไมโครคอนโทรลเลอร์ได้รวบรวมเอาซีพียู หน่วยความจำ อินพุตเอาต์พุต หน่วยควบคุมการทำงาน และการบริการอินเตอร์รัพต์ ตัวจับเวลาและตัวนับไว้บนชิปเดียวกัน โดยมีสถาปัตยกรรมภายในแบบฮาร์เวิร์ด (Harvard) ทำให้ระบบบัสของหน่วยความจำข้อมูลและระบบบัสของหน่วยความจำโปรแกรมสามารถแยกจากกันทำให้สามารถทำการเฟตช์และเอ็กคิวต์คำสั่งแบบซ้อนทับกันหรือทำงานในลักษณะไปป์ไลน์ (Pipeline) นั้นเองทำให้มีประสิทธิภาพในการทำงานสูง ดังแสดงในรูปที่ 4.1



รูปที่ 4.1 แสดงโครงสร้างสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ตระกูลพีไอซี

4.1 คุณสมบัติของไมโครคอนโทรลเลอร์โดยทั่วไป

สามารถแบ่งออกเป็น 3 ส่วนคือ หน่วยประมวลผลกลาง (CPU: Central processing unit) ส่วนของเพอริเฟอรัล (Peripheral) และคุณสมบัติพิเศษอื่นๆ โดยมีรายละเอียดดังนี้

4.1.1 คุณสมบัติของหน่วยประมวลผลกลาง

- หน่วยประมวลผลกลางเป็นแบบริสก์ (RISC)
- มีชุดคำสั่ง 36 คำสั่ง ขนาด 14 บิต
- ทุกคำสั่งถูกเอ็กซิกิวต์ใน 1 ไซเคิล
(ยกเว้นคำสั่งบรานซ์จะถูกเอ็กซิกิวต์ 2 ไซเคิล)
- ประมวลข้อมูลขนาด 8 บิต
- มีหน่วยความจำโปรแกรมซึ่งเป็นอีพรอมขนาด 8 กิโลไบต์ขนาด 14 บิต
- มีเรจิสเตอร์ฟังก์ชันพิเศษ 7 ตัว
- มีสเตก 8 ระดับแต่ละระดับมีขนาด 13 บิต
- มีโหมดการอ้างอิงแอดเดรส 4 โหมด คือ

แบบทันทีทันใด (Immediate addressing) แบบโดยตรง (Direct addressing)

แบบโดยอ้อม (Indirect addressing) และ แบบสัมพันธ์ (Relative addressing)

4.1.2 คุณสมบัติทางเพอริเฟอรัล

- มีขาอินพุตและเอาต์พุต 13 ขา สามารถกำหนดเป็นขาอินพุตหรือเอาต์พุต
- มีไทมเมอร์/เคาน์เตอร์ขนาด 8 บิตที่สามารถโปรแกรมได้

4.1.3 คุณสมบัติพิเศษอื่นๆ

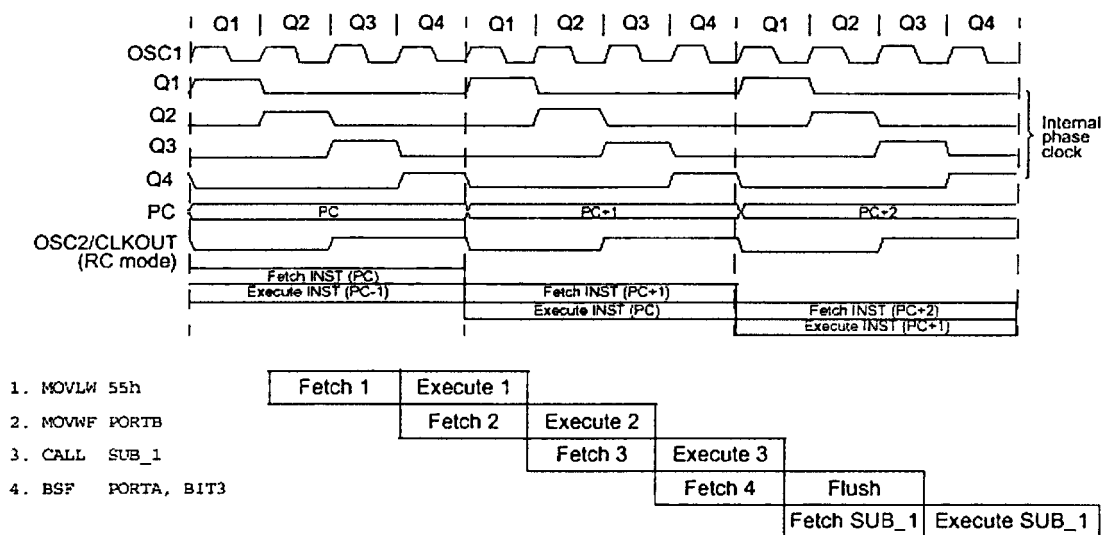
- มีเพาเวอร์ออนรีเซ็ตในตัว (POR: Power-On Reset)
- มีเพาเวอร์อัพไทมเมอร์ในตัว (PWRT: Power-up Timer)
- มีออสซิลเลเตอร์สตาร์ทอัพไทมเมอร์ (OST: Oscillator Start-up Timer)
- มีวอตช์ด็อกไทมเมอร์ (WDT: Watch Dog Timer) กับวงจรออสซิลเลเตอร์
ภายในเพื่อช่วยในการทำงานของไมโครคอนโทรลเลอร์ ให้มีความแน่นอนยิ่งขึ้น
- มีโหมดประหยัดพลังงานหรือสลิปโหมด (Sleep mode)

4.2 จังหวะสัญญาณนาฬิกาและไซเกิลการทำงาน

ในการปฏิบัติคำสั่งจะแบ่งขั้นตอนออกเป็น ไปป์ไลน์ 2 สเตท คือ เฟตช์และเอ็กซีคิวต์ โดยแต่ละสเตทมีการทำงานดังนี้

เฟตช์ (Fetch) เป็นสเตทของการอ่านคำสั่งจากหน่วยความจำโปรแกรมโดยการเพิ่มค่าของโปรแกรมเคาน์เตอร์เพื่อชี้ตำแหน่งและทำการถอดรหัส และส่งไปยังซีพียูเตรียมประมวลผล

เอ็กซีคิวต์ (Execute) เป็นสเตทของการประมวลผลตามคำสั่งที่ถูกเฟตช์เข้ามาและเขียนค่าผลลัพธ์กลับไปยังตำแหน่งปลายทางที่กำหนด



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

รูปที่ 4.2 แสดงจังหวะการทำงานของไมโครคอนโทรลเลอร์

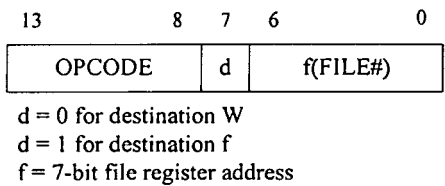
4.3 สถาปัตยกรรมชุดคำสั่ง

ในการออกแบบครั้งนี้มีชุดคำสั่ง 36 คำสั่ง (อ้างอิงจากไมโครคอนโทรลเลอร์ตระกูลพีไอซี เบอร์ PIC16C84) โดยแบ่งชุดคำสั่งออกเป็น 3 กลุ่มใหญ่ คือ กลุ่มคำสั่งจัดการข้อมูลระดับไบต์ (Byte-oriented), กลุ่มคำสั่งจัดการระดับบิต (Bit-oriented) และ กลุ่มคำสั่งควบคุมการทำงาน (Literal and control operations) โดยมีรายละเอียดดังนี้

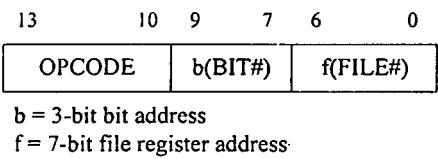
ตารางที่ 4.1 แสดงรายละเอียดของตัวแปรที่เกี่ยวข้องในรูปแบบของคำสั่ง

ตัวแปร	รายละเอียดของความหมาย
f	เรจิสเตอร์ไฟล์
W	เรจิสเตอร์ W (แอกคิวมูเลเตอร์)
b	ข้อมูลระดับบิตหรือบิตแอดเดรสภายในเรจิสเตอร์ภายในเรจิสเตอร์ไฟล์
k	ค่าคงที่ใด ๆ
x	ตำแหน่งที่ไม่สนใจ
d	ที่เก็บปลายทางถ้า d = 0 : ให้เก็บค่าที่เรจิสเตอร์ W ถ้า d = 1 : ให้เก็บค่าที่เรจิสเตอร์ไฟล์ f

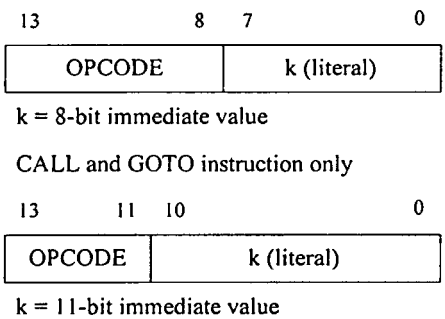
Byte-oriented file register operation



Bit-oriented file register operation



Literal and Control operation



รูปที่ 4.3 รูปแบบของคำสั่ง (Instruction Format)

ตารางที่ 4.2 รูปแบบของคำสั่งของไมโครคอนโทรลเลอร์

Mnemonic, Operand	ไซเคิล	รายละเอียด	14-Bit Opcode Msb Lsb	Status Affected
กลุ่มคำสั่งจัดการข้อมูลระดับไบต์				
ADDWF f,d	1	W + f	00 0111 dfff ffff	C,DC,Z
ANDWF f, d	1	W AND f	00 0101 dfff ffff	Z
CLRF f	1	เคลียร์ค่าของ f	00 0001 1fff ffff	Z
CLRW -	1	เคลียร์ค่าของ W	00 0001 0xxx xxxx	Z
COMF f, d	1	คอมพลีเมนต์ f	00 1001 dfff ffff	Z
DECF f, d	1	ลดค่า f	00 0011 dfff ffff	Z
DECFSZ f, d	1(2)	ลดค่า f, ข้ามหนึ่งแอดเดรส ถ้าเป็น 0	00 1011 dfff ffff	-
INCF f, d	1	เพิ่มค่า f	00 1010 dfff ffff	Z
INCFSZ f, d	1(2)	เพิ่มค่า f, ข้ามหนึ่งแอดเดรส ถ้าเป็น 0	00 1111 dfff ffff	-
IORWF f, d	1	W OR f	00 0100 dfff ffff	Z
MOVF f, d	1	โอนย้ายข้อมูลไปยัง f	00 1000 dfff ffff	Z
MOVWF f	1	โอนย้ายข้อมูลจาก W ไปยัง f	00 0000 1fff ffff	-
NOP -	1	ไม่มีการทำงาน	00 0000 0xx0 0000	-
RLF f, d	1	เลื่อนข้อมูลไปทางซ้าย 1 บิต	00 1101 dfff ffff	C
RRF f, d	1	เลื่อนข้อมูลไปทางขวา 1 บิต	00 1100 dfff ffff	C
SUBWF f, d	1	f - W	00 0010 dfff ffff	C,DC,Z
SWAPF f, d	1	สลับค่า 4 บิตบนกับ 4 บิตล่างของ f	00 1110 dfff ffff	-
TRIS f	1	กำหนดค่าของ TRIS	00 0000 0110 0fff	-
XORWF f, d	1	W XOR f	00 0110 dfff ffff	Z
กลุ่มคำสั่งจัดการข้อมูลระดับบิต				
BCF f, b	1	เคลียร์บิตของ f	01 00bb bfff ffff	-
BSF f, b	1	เซตบิตของ f	01 01bb bfff ffff	-
BTFSC f, b	1(2)	ตรวจสอบบิต ข้ามหนึ่งแอดเดรส ถ้าเป็น "0"	01 10bb bfff ffff	-

ตารางที่ 4.2 (ต่อ)

Mnemonic, Operand	ไชนเคิล	รายละเอียด	14-Bit Opcode		Status Affected
			Msb	Lsb	
BTFSS f, b	1(2)	ตรวจสอบบิต ข้ามหนึ่งแอดเดรส ถ้าเป็น “1”	01	11bb bfff ffff	-
กลุ่มคำสั่งจัดการกับข้อมูลค่าคงที่และควบคุมการทำงาน					
ADDLW k	1	ค่าคงที่ + W	11	111x kkkk kkkk	C,DC,Z
ANDLW k	1	ค่าคงที่ AND W	11	1001 kkkk kkkk	Z
CALL k	2	เรียกโปรแกรมย่อย	10	0kkk kkkk kkkk	-
CLRWDT -	1	เคลียร์ค่าของ WDT	00	000 0110 0100	TO,PD
GOTO -	2	กระโดดไปยังแอดเดรสที่กำหนด	10	1kkk kkkk kkkk	-
IORLW k	1	ค่าคงที่ OR W	11	1000 kkkk kkkk	Z
MOVLW k	1	นำค่าคงที่ไปเก็บไว้ใน W	11	00xx kkkk kkkk	-
RETFIE -	2	ออกจากโปรแกรมย่อยบริการการ อินเทอร์รัพต์	00	0000 0000 1001	-
RETLW k	2	กลับสู่โปรแกรมด้วยค่าคงที่ใน W	11	01xx kkkk kkkk	-
RETURN -	2	ออกจากโปรแกรมย่อย	00	0000 0000 1000	-
SLEEP -	1	เข้าสู่โหมดสลีป	00	0000 0110 0011	TO,PD
SUBLW -	1	ค่าคงที่ - W	11	110x kkkk kkkk	C,DC,Z
XORLW -	1	ค่าคงที่ XOR W	11	1010 kkkk kkkk	Z

4.4 การจัดการหน่วยความจำ

หน่วยความจำของระบบไมโครคอนโทรลเลอร์ที่ออกแบบจะแบ่งออกเป็น หน่วยความจำโปรแกรม (Program memory) และหน่วยความจำข้อมูล (Data memory) โดยหน่วยความจำโปรแกรมจะมีขนาด 8 กิโลเวิร์ด (จำนวน 1 เวิร์ดเท่ากับ 14 บิต) อยู่ในรูปแบบของอีพรอมภายนอก ส่วนหน่วยความจำข้อมูลจะแบ่งออกเป็น 2 ส่วน คือ ส่วนของเรจิสเตอร์ฟังก์ชันพิเศษ (SFR: Special Function Registers) และส่วนของเรจิสเตอร์ใช้งานทั่วไป (GPR: General Purpose Registers) ที่มีขนาด 32 ไบต์ โดยถูกจัดสรรไว้ที่ตำแหน่ง 0x000 - 0x02F โดยการจัดสรรหน่วยความจำข้อมูลทั้งเรจิสเตอร์ฟังก์ชันพิเศษและเรจิสเตอร์ใช้งานทั่วไปจะจัดแบ่งออกเป็นแบงก์ (Bank) คือ แบงก์ 0 และ แบงก์ 1 โดยการติดต่อหน่วยความจำแต่ละแบงก์จะถูกควบคุมจากบิตอาร์พีศูนย์ (RP0) และบิตอาร์พีหนึ่ง (RP1) ในเรจิสเตอร์สถานะโดยที่บิตอาร์พีศูนย์มีค่าเป็นศูนย์จะเป็นการเลือกหน่วยความจำข้อมูลแบงก์ศูนย์ แต่เมื่อบิตอาร์พีหนึ่งมีค่าเป็นหนึ่ง จะเป็นการเลือกหน่วยความจำแบงก์หนึ่ง

4.4.1 โหมดการอ้างอิงตำแหน่งที่อยู่ของข้อมูล

ในไมโครคอนโทรลเลอร์พีไอซีมีกระบวนการการเข้าถึงหน่วยความจำและเรจิสเตอร์อยู่ 4 รูปแบบ คือ

1. แบบทันทีทันใด (Immediate addressing) คือ เป็นการเข้าถึงค่าคงที่โดยตรง ด้วยคำสั่งที่เกี่ยวข้องกับค่าคงที่ (Literal) ร่วมกับเรจิสเตอร์ดับเบิลยูตัวอย่างเช่น

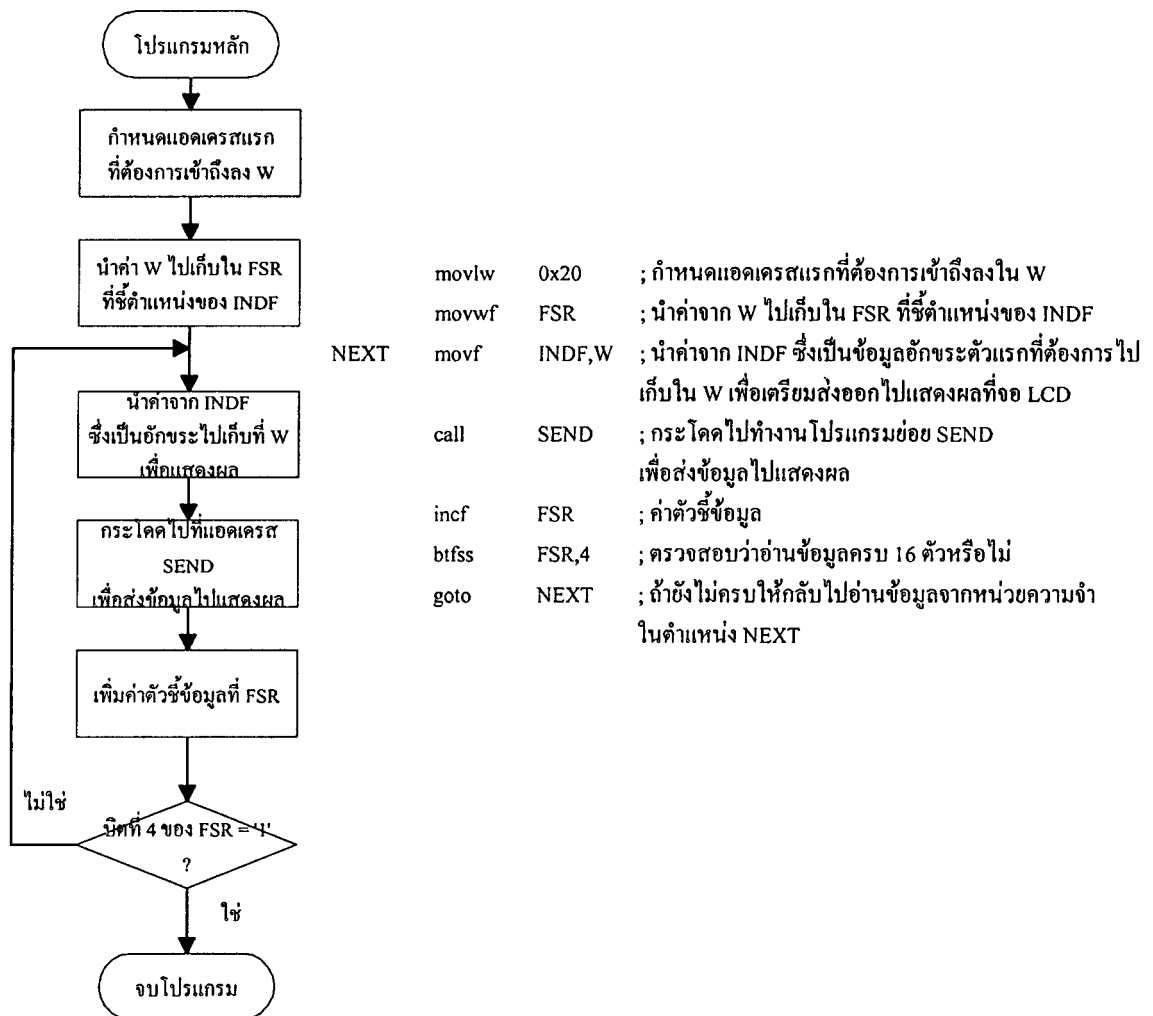
`MOVLW k` ; นำค่าคงที่ k ไปเก็บใน W แบบทันทีทันใด

2. แบบโดยตรง (Direct addressing) คือ เป็นการเข้าถึงข้อมูลหรือเรจิสเตอร์ด้วยการกำหนดแอดเดรสที่ต้องการ โดยตรง หรือระบุผ่านชื่อของเรจิสเตอร์ได้โดยตรง ตัวอย่างเช่น

`CLRF TEMP` ; เป็นคำสั่งที่ต้องการเคลียร์ค่าเรจิสเตอร์ TEMP โดยตรง

3. แบบทางอ้อม (Indirect addressing) โดยผ่านเรจิสเตอร์ FSR โดยวิธีการนี้ จะเริ่มจากนำค่าของเรจิสเตอร์ที่ต้องการเข้าถึงเก็บลงไปในเรจิสเตอร์ W จากนั้นนำค่าในเรจิสเตอร์ W ไปเก็บไว้ในเรจิสเตอร์ FSR เพื่อให้เรจิสเตอร์ FSR เป็นตัวชี้ข้อมูลที่ต้องการเข้าถึง การใช้เรจิสเตอร์ FSR จะมีลักษณะคล้ายกับการอ้างอิงแบบดัชนี (Index addressing) ของไมโครคอนโทรลเลอร์เบอร์อื่นๆ แต่เนื่องจากไมโครคอนโทรลเลอร์มีเรจิสเตอร์สำหรับใช้งานทั่วไปเพียง 32 ตัว ดังนั้นการใช้วิธีการ

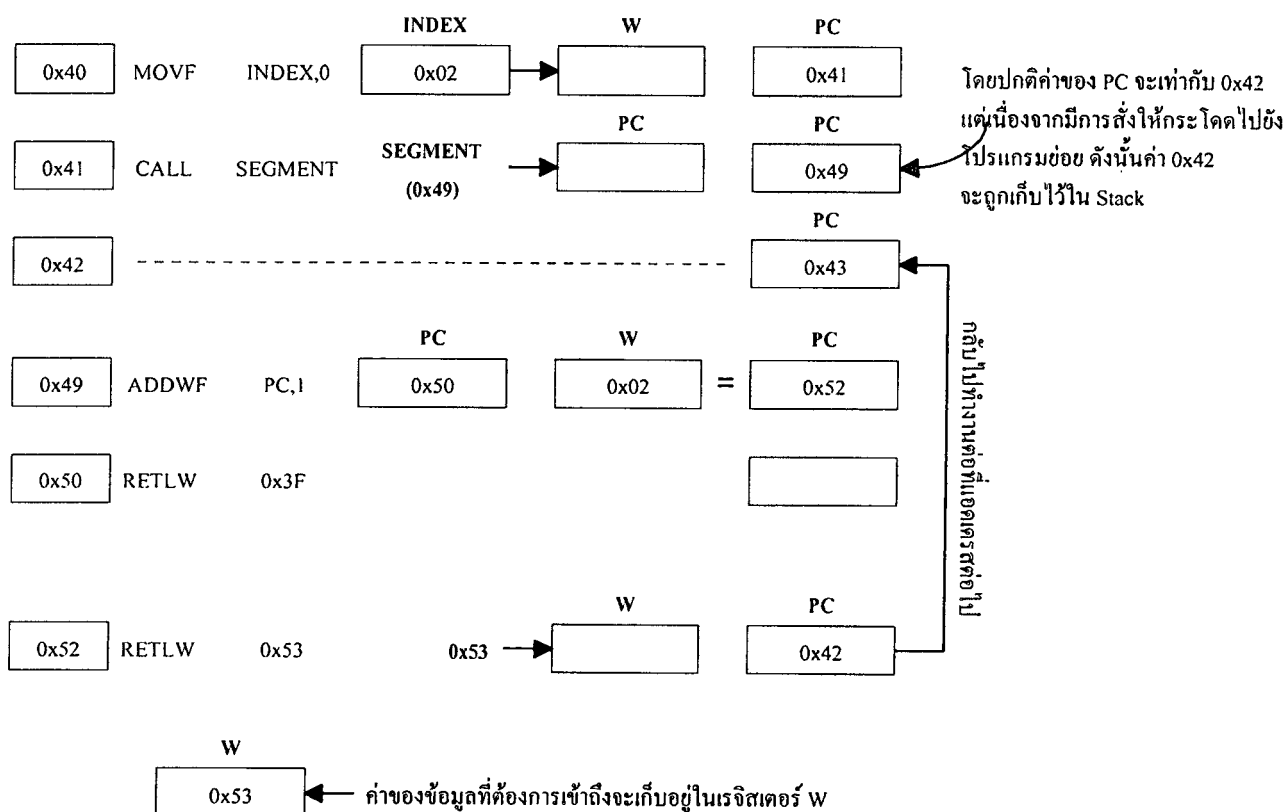
เข้าถึงแบบโดยอ้อมอาจใช้ไม่บ่อยนัก ดังโปรแกรมตัวอย่างดังนี้ที่เป็นโปรแกรมย่อยของการส่งข้อมูลตัวอักษรจากเรจิสเตอร์ไฟล์ 16 ตัว ตั้งแต่แอดเดรส 0x020-0x02f ไปแสดงที่ LCD โมดูล



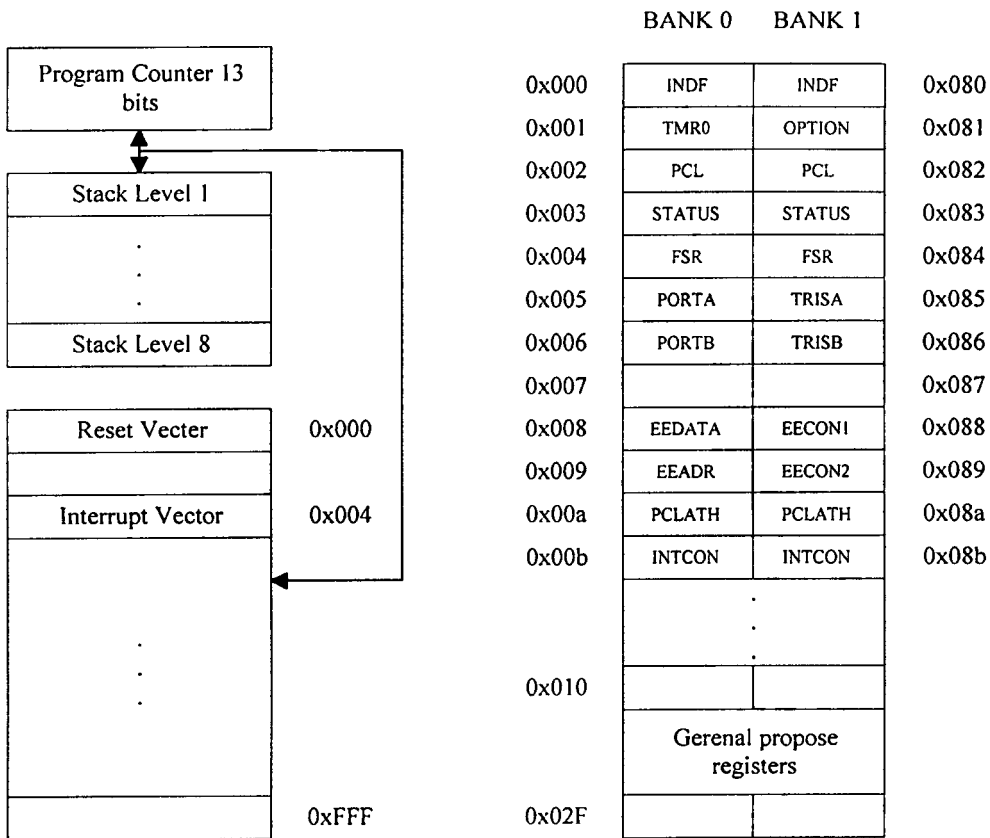
รูปที่ 4.4 โฟลวชาร์ตของโปรแกรมที่ใช้ในการเข้าถึงแอดเดรสทางอ้อม

4. แบบสัมพัทธ์ (Relative addressing) การเข้าถึงข้อมูลแบบนี้จะประกอบด้วยการคำนวณค่าสัมพัทธ์ของระยะห่างระหว่างแอดเดรสที่เริ่มต้นทำงานกับแอดเดรสที่ต้องการเข้าถึง โดยมีการใช้ค่าของโปรแกรมเคาน์เตอร์เข้ามาช่วย คำสั่งหนึ่งที่น่าสนใจในการเข้าถึงข้อมูลแบบสัมพัทธ์นี้คือ RETLW กระบวนการจะเริ่มด้วยการกำหนดค่าออฟเซตลงในเรจิสเตอร์ดัชนีเบิลยูแล้วนำไปรวมกับค่าของโปรแกรมเคาน์เตอร์จะได้ค่าของแอดเดรสของหน่วยความจำที่ต้องการเข้าถึง การเข้าถึงแบบนี้มีข้อจำกัดคือ ระยะห่างของแอดเดรสเริ่มต้นกับแอดเดรสที่ต้องการเข้าถึงต้องไม่เกิน 256 ตำแหน่ง ตัวอย่างของการเข้าถึงข้อมูลแบบสัมพัทธ์คือ การเปิดตารางข้อมูล (Look Up Table) ซึ่งมีโปรแกรมตัวอย่างดังนี้

PC	EQU 0x02	; กำหนดแอดเดรสของ PC
INDEX	EQU 0x0C	; กำหนดแอดเดรสของ INDEX
	MOVLW 0x02	; กำหนดค่าเพื่อทำการชี้ตำแหน่งเก็บใน INDEX ลงใน W
	MOVLW INDEX	; นำค่าของการชี้ตำแหน่งเก็บใน INDEX
	MOVF INDEX, 0	; นำค่าของ INDEX กลับมาเก็บใน W
	CALL SEGMENT	; กระโดดไปทำงานที่ SEGMENT
SEGMENT	ADDWF PC, 1	; บวกค่าของการชี้ตำแหน่งที่เก็บอยู่ใน W เข้ากับ PC
	RETLW 3F	; ค่าของข้อมูลที่ต้องการเข้าถึง
	RETLW 06	; ค่าของข้อมูลที่ต้องการเข้าถึง
	RETLW 53	; ค่าของข้อมูลที่ต้องการเข้าถึง
	RETLW 4F	; ค่าของข้อมูลที่ต้องการเข้าถึง
	RETLW 66	; ค่าของข้อมูลที่ต้องการเข้าถึง
	RETLW 60	; ค่าของข้อมูลที่ต้องการเข้าถึง
	RETLW 70	; ค่าของข้อมูลที่ต้องการเข้าถึง



รูปที่ 4.5 แสดงการทำงานของโปรแกรมย่อยที่ใช้การเข้าถึงแบบสัมพันธ์



รูปที่ 4.6 แสดงการจัดสรรหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลในไมโครคอนโทรลเลอร์

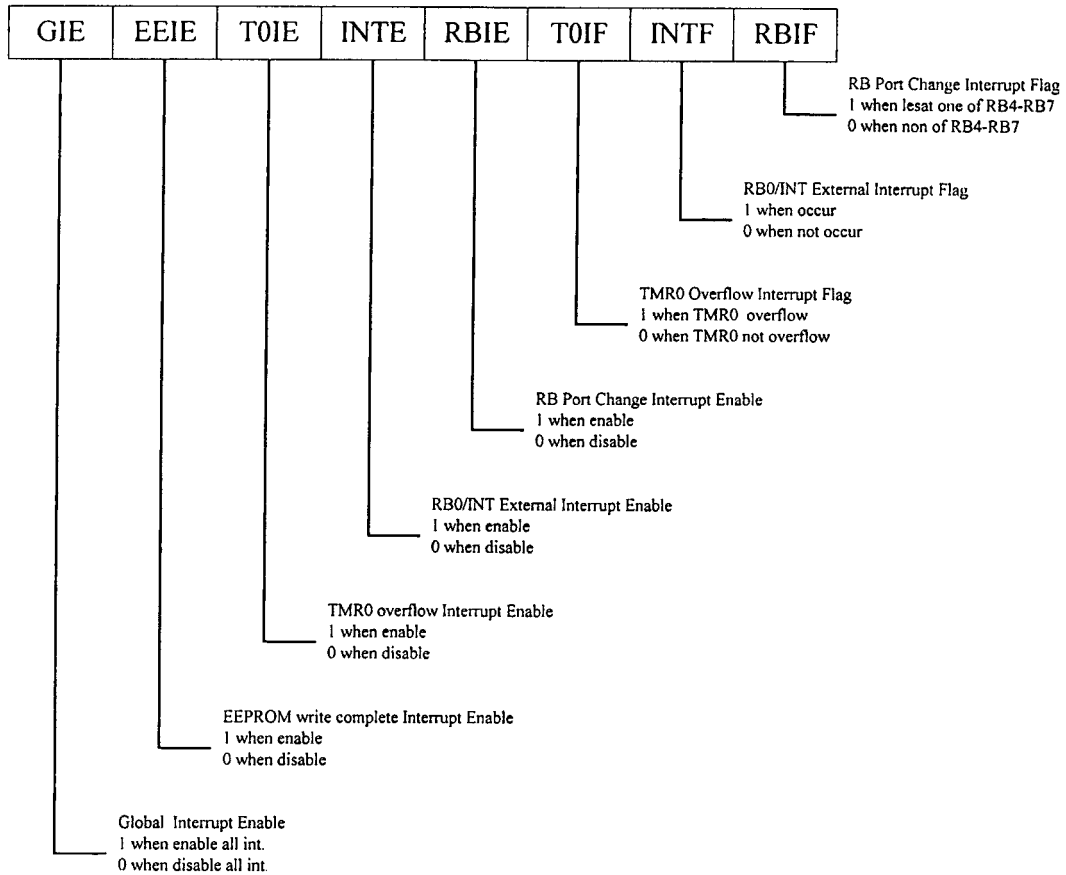
4.5 การอินเทอร์รัพต์ของไมโครคอนโทรลเลอร์

การอินเทอร์รัพต์ (Interrupt) คือ การขัดจังหวะการทำงานปกติของซีพียู เมื่อเกิดการอินเทอร์รัพต์ขึ้น ซีพียูจะทำงานจนจบไซเคิลนั้น แล้วจึงมาตอบสนองการอินเทอร์รัพต์โดยกระโดดมาทำงานยังตำแหน่งที่กำหนดไว้ จนกระทั่งเมื่อตอบสนองเรียบร้อยแล้ว ซีพียูจะกลับไปทำงานต่อจากเหตุการณ์ที่เกิดขึ้นก่อนหน้าการอินเทอร์รัพต์

การเกิดอินเทอร์รัพต์ในระบบไมโครคอนโทรลเลอร์จะมีทั้งจากภายในและภายนอกตัวซีพียู หรือจากกระบวนการประมวลผลของซอฟต์แวร์ก็ได้ ซึ่งอาจสรุปได้ 4 แหล่งคือ

- สัญญาณอินเทอร์รัพต์จากภายนอกป้อนเข้าที่ขา RB0/INT
- การอินเทอร์รัพต์อันเนื่องมาจากไทเมอร์ศูนย์เรจิสเตอร์ (TMR0) เกิดโอเวอร์โฟลว
- การอินเทอร์รัพต์อันเนื่องมาจากการเปลี่ยนแปลงของระดับสัญญาณที่ขา RB4-RB7
- การอินเทอร์รัพต์อันเนื่องมาจากการเขียนข้อมูลในหน่วยความจำข้อมูลอีพროมสมบูรณ์

โดยตัวไมโครคอนโทรลเลอร์สามารถที่จะเลือกตอบสนองการอินเทอร์รัพต์ทั้ง 4 รูปแบบ หรือเลือกตอบสนองบางแหล่งกำเนิดก็ได้ขึ้นอยู่กับข้อกำหนดที่เรจิสเตอร์ควบคุมการอินเทอร์รัพต์ (INTCON register) ดังแสดงในรูปที่ 4.7



รูปที่ 4.7 แสดงรายละเอียดของเรจิสเตอร์ควบคุมการอินเทอร์รัพต์

จากรูปที่ 4.7 จะสังเกตได้ว่าบิตจีไออี (GIE: Global Interrupt Enable) จะเป็นบิตที่มีนัยสำคัญสูงสุด ถ้าหากบิตจีไออีมีค่าเป็นศูนย์ จะเป็นการดิสเอเบิลการอินเทอร์รัพต์ทั้งหมด นั่นคือไมโครคอนโทรลเลอร์จะไม่ตอบสนองการอินเทอร์รัพต์จากทุกแหล่งกำเนิด ในขณะเดียวกันถ้าหากบิตจีไออีมีค่าเป็นหนึ่ง ก็จะเป็นการกำหนดในไมโครคอนโทรลเลอร์พร้อมที่จะตอบสนองการอินเทอร์รัพต์จากทุกแหล่งกำเนิด ขึ้นอยู่กับว่าบิตควบคุมการอินเทอร์รัพต์อื่นๆ ในเรจิสเตอร์ควบคุมการอินเทอร์รัพต์ จะมีการอินาเบิลไว้หรือไม่ แต่เมื่อเกิดการอินเทอร์รัพต์ขึ้น ไมโครคอนโทรลเลอร์จะกระโดดไปทำงานที่แอดเดรส 0x004 ของหน่วยความจำโปรแกรมเสมอซึ่งโปรแกรมย่อยที่แอดเดรส 0x004 จะเรียกว่า โปรแกรมย่อยของการบริการอินเทอร์รัพต์ (Interrupt Service Routine)

บทที่ 5

ขั้นตอนการออกแบบระบบ

ในการออกแบบไมโครคอนโทรลเลอร์ต้นแบบ ข้อมูลที่นำมาประกอบการวิเคราะห์ได้มาจากดาต้าบุ๊ก (Data Book) ของไมโครคอนโทรลเลอร์ตระกูลพีไอซี ซึ่งมีเฉพาะรายละเอียดของรูปแบบการทำงานและการใช้งาน และบล็อกไดอะแกรมของสถาปัตยกรรมเท่านั้น ไม่มีรายละเอียดของสถาปัตยกรรมภายในทั้งหมด ทำให้การออกแบบจะต้องออกแบบสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ขึ้นมาเอง ซึ่งทำให้จังหวะการทำงานไม่เหมือนกับพีไอซีจริง แต่ในการออกแบบจะคำนึงถึงรูปแบบฟังก์ชันการทำงานให้สามารถทำงานได้ถูกต้องใกล้เคียงกับพีไอซีให้มากที่สุดเท่าที่จะสามารถออกแบบได้ ส่วนในเรื่องของความเร็วและคุณสมบัติพิเศษต่างๆ จะไม่คำนึงถึงมากนัก สำหรับขั้นตอนการออกแบบจะแบ่งเป็น 3 ขั้นตอนใหญ่ๆ คือ ขั้นตอนของการกำหนดรายละเอียดของระบบที่ทำการออกแบบ ต่อมาเป็นขั้นตอนของการออกแบบระบบให้อยู่ในรูปภาษาวีเอชดีแอล ที่สามารถจำลองการทำงานในระดับฟังก์ชันการทำงานเพื่อตรวจสอบความถูกต้อง จากนั้นเป็นขั้นตอนที่ทำการสร้างต้นแบบระบบจากภาษาวีเอชดีแอล โดยต้นแบบที่ได้จะอยู่ในรูปของวงจรระดับเกตที่สามารถนำไปสร้างลงบนอุปกรณ์เอฟพีจีเอ เพื่อทดสอบการทำงานจริง เป็นขั้นตอนสุดท้ายของการออกแบบทั้งหมด

5.1 กำหนดรายละเอียดของระบบ

เป็นขั้นตอนของการวิเคราะห์ระบบและสร้างข้อกำหนดของระบบที่ทำการออกแบบ ในขั้นตอนนี้จะทำการแบ่งระบบ (System partitioning) ให้เป็นระบบย่อยๆ หรือโมดูลต่างๆ เพื่อช่วยต่อการออกแบบ แก้ไขและทดสอบความถูกต้อง โดยทำการแบ่งออกตามหน้าที่การทำงาน ดังต่อไปนี้

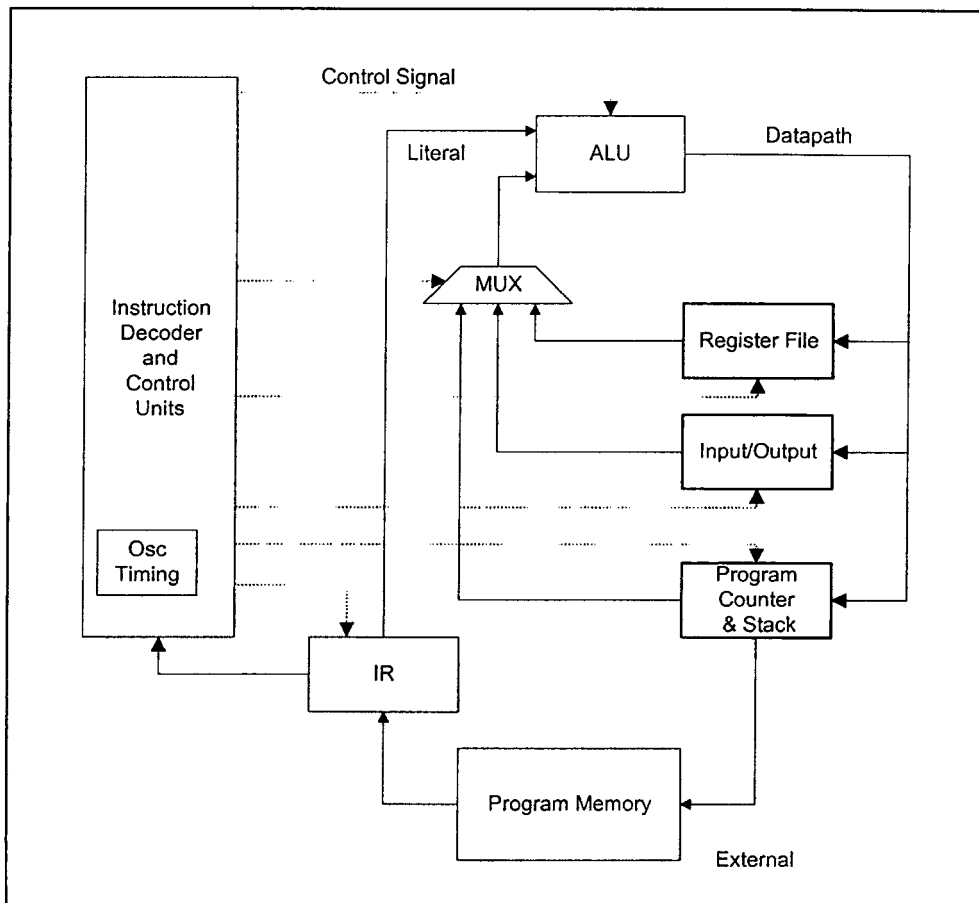
5.1.1 โมดูลเอแอลเอ็ม (ALM: Arithmetic & Logic Module)

5.1.2 โมดูลไออาร์ไอดีเอ็ม (IRIDM: Instruction Register & Instruction Decoder Module)

5.1.3 โมดูลพีซีเอสเอ็ม (PCSM: Program Counter & Stack Module)

5.1.4 โมดูลเอฟอาร์เอ็ม (FRM: File Registers Module)

5.1.5 โมดูลไอโอพีเอ็ม (IOPM: Input & Output Port Module)

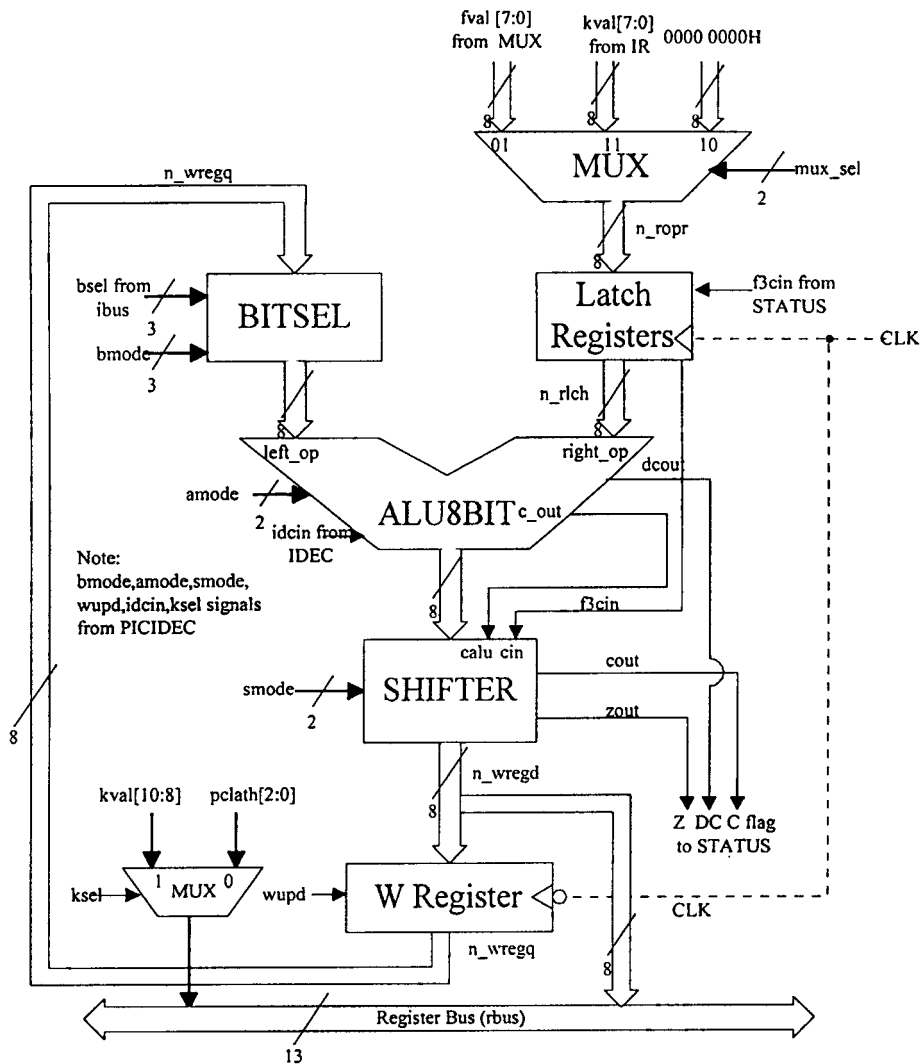


รูปที่ 5.1 บล็อกไดอะแกรมที่ถูกแบ่งออกเป็นโมดูลต่างๆ ในขั้นตอนการวิเคราะห์ระบบ

เนื่องจากหลักการออกแบบจากบนลงล่างทำให้สามารถออกแบบให้โมดูลต่างๆ มีสามารถแบ่งย่อยให้ลึกกลงไปได้อีกระดับ โดยมีรายละเอียดแต่ละ โมดูลต่างๆ ที่ออกแบบดังต่อไปนี้

5.1.1 โมดูลเอแอลเอ็ม

เป็นโมดูลที่ออกแบบให้ทำหน้าที่เป็นตัวประมวลผลทางคณิตศาสตร์และตรรกะ (ALU: Arithmetic Logic Unit) มีขนาด 8 บิต ที่สามารถทำการบวก, ลบ, เลื่อนบิต, สลับบิต และประมวลผลทางตรรกะ ในการทำงานของตัวประมวลผลทางคณิตศาสตร์และตรรกะ จะต้องทำงานร่วมกับเรจิสเตอร์ดับเบิลยู (W register) (ถ้าเปรียบเทียบกับไมโครคอนโทรลเลอร์เบอร์อื่นๆ เรจิสเตอร์ดับเบิลยูเทียบได้กับแอมคิวมูลเตอร์ (Accumulator) นั่นเอง) โดยจะประมวลผลระหว่างข้อมูลที่อยู่ในเรจิสเตอร์ดับเบิลยูกับเรจิสเตอร์ไฟล์ใดๆ โดยการทำงานจะมีผลต่อแฟล็กตัวทศหรือยืม (Carry flag) แฟล็กตัวทศหรือยืมระหว่างหลัก (Digit carry flag) แฟล็กศูนย์ (Zero flag) ในเรจิสเตอร์สถานะ ในการออกแบบจะให้เอแอลยูสามารถทำการประมวลผลให้เสร็จสิ้นเพียง 1 คาบสัญญาณนาฬิกาหรือออกแบบให้เอแอลยูทำงานในลักษณะ Arithmetic Pipeline โดยออกแบบดังรูปที่ 5.2



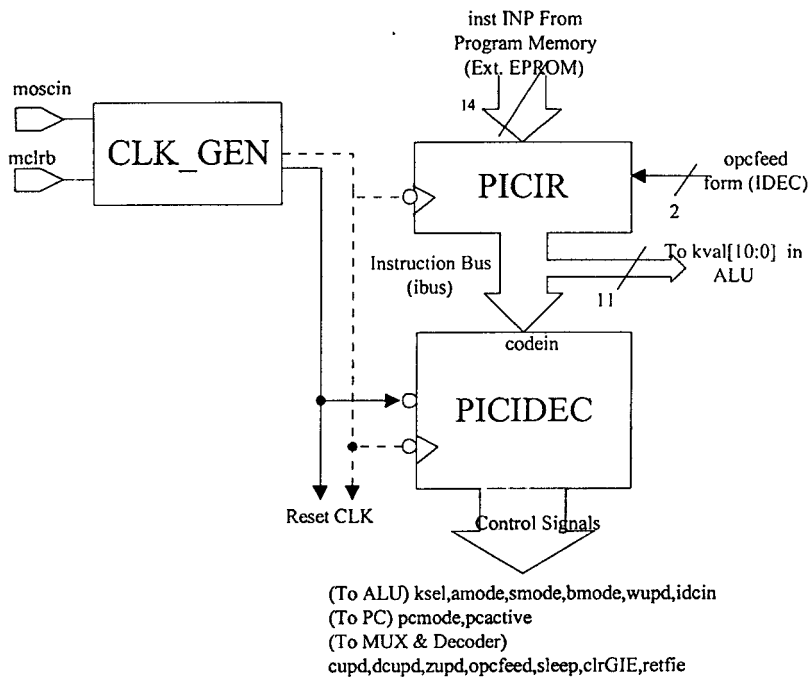
รูปที่ 5.2 โครงสร้างสถาปัตยกรรมภายในของโมดูลเอแอลเอ็ม

จากรูปที่ 5.2 โมดูลเอแอลเอ็มจะประกอบไปด้วยโมดูลย่อย ได้แก่ (1) Alu8bit ออกแบบให้เป็นโมดูลที่ทำหน้าที่ในการบวกและทางตรรกได้รูปแบบ 3 แบบ And, OR, Exclusive OR ระหว่างข้อมูลจากเรจิสเตอร์ไฟล์กับข้อมูลใน W-register (2) BITSEL ขนาด 8 บิตออกแบบให้ทำหน้าที่เกี่ยวกับการจัดการระดับบิตของข้อมูลโดยออกแบบให้มีฟังก์ชันทำงาน 8 แบบ คือ เคลียร์บิตที่ต้องการให้เป็นศูนย์, เซตทุกบิตให้เป็นศูนย์, วันคอมพลีเมนต์ (1's-Complement), วันคอมพลีเมนต์บิตข้อมูลทางขวา, เซตบิตเป็นหนึ่ง, เซตทุกบิตเป็นหนึ่ง, ส่งผ่านบิตข้อมูล, ส่งผ่านบิตข้อมูลทางขวามือ ซึ่งบิตการเลือกให้บิตใดถูกกระทำจะได้จากบิตที่ 9-7 จากบัสสัญญาณ (IBUS) ซึ่งเป็นบัสสัญญาณข้อมูลคำสั่ง (3) SHIFTER ขนาด 8 บิตออกแบบให้ทำหน้าที่ในการเลื่อนบิตข้อมูลโดยสามารถทำงานได้ 4 รูปแบบ คือ เลื่อนบิตไปทางขวา, เลื่อนบิตไปทางซ้าย, สลับค่านิบเบิล (Swap Nibble), ส่งผ่านบิต (4) W-register ขนาด 8 บิตออกแบบทำหน้าที่เก็บค่าผลลัพธ์ที่ได้จากการเอ็กซ์คิวต์ โดยทำงาน

ซึ่งโครไนซ์กับแลทซ์เรจิสเตอร์โดยอาศัยขอบขาขึ้นและขาลงของสัญญาณนาฬิกาในแต่ละลูก ทำให้การประมวลผลต่างๆ ภายในโมดูลเอแอลเอ็มสามารถประมวลผลให้เสร็จสิ้นได้ภายใน 1 คาบของสัญญาณนาฬิกา โดยโหมดฟังก์ชันการทำงานของโมดูลย่อยต่างๆ ภายในโมดูลเอแอลเอ็มจะถูกส่งมาจากโมดูลไออาร์ไอดีเอ็ม ที่ทำหน้าที่ถอดรหัสคำสั่งเพื่อใช้สร้างเป็นสัญญาณควบคุมประเภทต่างๆ ได้แก่ สัญญาณเอโหมด (Amode) ขนาด 2 บิตเป็นสัญญาณควบคุมการทำงานของเอแอลยูแปดบิตโมดูล สัญญาณบีโหมด (Bmode) ขนาด 3 บิตเป็นสัญญาณควบคุมการทำงานของบิตชีเร็กโมดูล สัญญาณเอสโหมด (Smode) ขนาด 2 บิตเป็นสัญญาณควบคุมการทำงานของชิปเตอร์โมดูล สัญญาณดับเบิลยูอัปเดต (Wupd) เป็นสัญญาณควบคุมการเขียนค่าผลลัพธ์ที่ได้จากการประมวลผลลงใน W-register สัญญาณเคซีเร็ก (Ksel) เป็นสัญญาณในการเลือกข้อมูลระหว่างสัญญาณข้อมูล Kval บิตที่ 10-8 ที่ส่งมาจากเรจิสเตอร์คำสั่ง กับข้อมูลของเรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์สูง (PCLATH register) ในกรณีที่ซีพียูกระทำคำสั่ง CALL หรือ GOTO จะต้องมีการโหลดค่าของโปรแกรมเคาน์เตอร์ใหม่เข้ามาจากเรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์สูง เพราะในส่วนของโปรแกรมเคาน์เตอร์ของระบบมีขนาด 13 บิตแบ่งออกเป็น 2 ส่วนคือ 5 บิตบนจะไม่สามารถอ่านและเขียนค่าได้โดยตรง ต้องอาศัยเขียนและอ่านค่าผ่านเรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์สูง (PCH) และส่วน 8 บิตล่างจะสามารถอ่านและเขียนค่าได้โดยตรงหรือเรียกว่า เรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์ต่ำ (PCL) และในการกระทำคำสั่งเกี่ยวกับคณิตศาสตร์และลอจิกเกิดขึ้นจะมีผลต่อบิตแฟลกแสดงสถานะได้แก่ แฟลกตัวทดหรือยืม (C: Carry flag) แฟลกตัวทดหรือยืมระหว่างหลัก (DC: Digit Carry/Borrow flag) แฟลกศูนย์ (Z: Zero flag) โดยบิตแฟลกทั้งสามบิตจะถูกส่งไปเก็บไว้ที่เรจิสเตอร์สถานะ (Status register) ใน โมดูลเอฟอาร์เอ็มต่อไป

5.1.2 โมดูลไออาร์ไอดีเอ็ม

เป็นโมดูลที่ออกแบบให้ทำหน้าที่หลักคือรับข้อมูลคำสั่งแล้วทำการถอดรหัสคำสั่งออกเป็นสัญญาณควบคุมเพื่อควบคุมโมดูลต่างๆ ภายในไมโครคอนโทรลเลอร์ และทำการสร้างสัญญาณรีเซ็ตและสัญญาณนาฬิกาภายในระบบ โดยการออกแบบให้รับข้อมูลคำสั่งขนาด 14 บิตมาจากหน่วยความจำโปรแกรมภายนอกที่ถูกเก็บอยู่ในรูปอีพรอม เมื่อซีพียูทำการเฟตช์ ข้อมูลคำสั่งจะถูกส่งเป็นเก็บไว้ที่เรจิสเตอร์คำสั่ง และส่งผ่านไปยังตัวถอดรหัสคำสั่ง เพื่อทำการถอดรหัสคำสั่งเป็นสัญญาณควบคุม โดยมีตัวสร้างสัญญาณนาฬิกา เป็นตัวควบคุมจังหวะการทำงาน โดยออกแบบให้มีโครงสร้างดังรูปที่ 5.3



รูปที่ 5.3 โครงสร้างสถาปัตยกรรมภายในของโมดูลไออาร์ไอซีเอ็ม

จากรูปที่ 5.3 โมดูลไออาร์ไอซีเอ็มจะประกอบไปด้วยโมดูลย่อยได้แก่ (1) PICIR ออกแบบให้ทำหน้าที่เป็นเรจิสเตอร์คำสั่งโดยทำหน้าที่เก็บข้อมูลคำสั่งไว้จนกว่าจะถูกถอดรหัส และสามารถควบคุมข้อมูลที่ออกจากเรจิสเตอร์คำสั่งนี้ได้จากสัญญาณควบคุม (Opcfeed) ในการทำงานที่มีเงื่อนไข เช่น เกิดอินเตอร์รัพต์โดยซีพียูจะต้องกระโดดไปทำงานที่ตำแหน่ง 0x004 ในหน่วยความจำโปรแกรมซึ่งเป็นที่เก็บค่าเวกเตอร์การอินเตอร์รัพต์ (Interrupt vector) หรือซีพียูเกิดสภาวะรีเซ็ต ซีพียูจะต้องกระโดดไปทำงานที่ตำแหน่ง 0x000 ในหน่วยความจำโปรแกรมซึ่งเป็นที่เก็บค่าเวกเตอร์ของการรีเซ็ต (Reset vector) เป็นต้น (2) PICIDEC จะทำหน้าที่ถอดรหัสดำเนินการที่มีขนาด 14 บิต ออกเป็นสัญญาณควบคุมเพื่อส่งไปควบคุมการทำงานของโมดูลต่างๆ ในการออกแบบ PICIDEC เราจะสร้างตารางความจริง (True table) ไว้ภายใน ดังแสดงในตารางที่ 5.1 และ 5.2 (3) CLK_GEN ทำหน้าที่สร้างสัญญาณนาฬิกาและสัญญาณรีเซ็ตของระบบ โดยโครงสร้างภายในประกอบไปด้วยอาร์ชีออสซิลเลเตอร์ขนาด 10 และ 12 บิต เพื่อทำหน้าที่เป็น ออสซิลเลเตอร์สตาร์ทอัพไทมเมอร์ (Oscillator start-up timer) ทำหน้าที่กำหนดสัญญาณเพื่อหน่วงเวลาขนาด 1,024 ไซเคิล หลังจากการสิ้นสุดการทำงานเพาเวอร์ออนรีเซ็ต ทั้งนี้เพื่อเป็นการตรวจสอบความถูกต้องของวงจรกำเนิดสัญญาณนาฬิกาภายนอกที่ป้อนเข้าสู่ระบบพร้อมที่จะทำงาน และ เพาเวอร์ออนรีเซ็ตไทมเมอร์ (Power-on-reset timer) การรีเซ็ตลักษณะนี้เกิดขึ้นเมื่อเกิดการตรวจจับแรงดันที่ขาไฟเลี้ยง ถ้าอยู่ในสถานะต่ำจะทำให้ฟลิปฟลอปเกิดการรีเซ็ต ถ้าอยู่ในสถานะสูงก็จะไม่เกิดเพาเวอร์ออนรีเซ็ตเกิดขึ้น

ตารางที่ 5.1 รายละเอียดสัญญาณควบคุมที่ได้จากการถอดรหัสคำสั่งต่างๆ

Opcode	Control Signals									
	ret/slp/ clrwdt	Oldinst	Dest. W f	Carry feed	Literal (k)	xqmode XXX	C,DC,Z	amode XX	bmode XXX	smode XX
ADDWF	000	0	W or F	0	0	NORM	111	ADD	PASS	PASS
ANDWF	000	0	W or F	-	0	NORM	001	AND	PASS	PASS
CLRF	000	0	F	-	-	NORM	001	AND	ALL0	PASS
CLRW	000	0	W	-	-	NORM	001	AND	ALL0	PASS
COMF	000	0	W or F	-	0	NORM	001	XOR	ALL1	PASS
DECf	000	0	W or F	0	0	NORM	001	ADD	ALL1	PASS
DECFSZ	000	0	W or F	0	0	SKZR	000	ADD	ALL1	PASS
INCF	000	0	W or F	1	0	NORM	001	ADD	ALL0	PASS
INCFSZ	000	0	W or F	1	0	SKZR	000	ADD	ALL0	PASS
IORWF	000	0	W or F	-	0	NORM	001	OR	PASS	PASS
MOVF	000	0	W or F	-	0	NORM	001	OR	ALL0	PASS
MOVWF	000	0	F	-	-	NORM	000	OR	X111	PASS
NOP	000	0	NO	-	-	NORM	000	-	-	-
RLF	000	0	W or F	-	0	NORM	100	OR	ALL0	S.L
RRF	000	0	W or F	-	0	NORM	100	OR	ALL0	S.R
SUBWF	000	0	W or F	1	0	NORM	111	ADD	INVT	PASS
SWAPF	000	0	W or F	-	0	NORM	000	OR	ALL0	SWAP
XORWF	000	0	W or F	-	0	NORM	001	XOR	PASS	PASS
BCF	000	0	F	-	0	NORM	000	AND	BIT0	PASS
BSF	000	0	F	-	0	NORM	000	OR	BIT1	PASS
BTfSC	000	0	NO	-	0	SKZR	000	AND	BIT1	PASS
BTfSS	000	0	NO	-	0	SKNZ	000	AND	BIT1	PASS
ADDLW	000	0	W	0	1	NORM	111	ADD	PASS	PASS
ANDLW	000	0	W	-	1	NORM	001	AND	PASS	PASS
CALL	000	0	NO	-	1	CALL	000	OR	ALL0	PASS
CLRWDt	001	0	NO	-	-	NORM	000	-	-	-

ตารางที่ 5.1 (ต่อ)

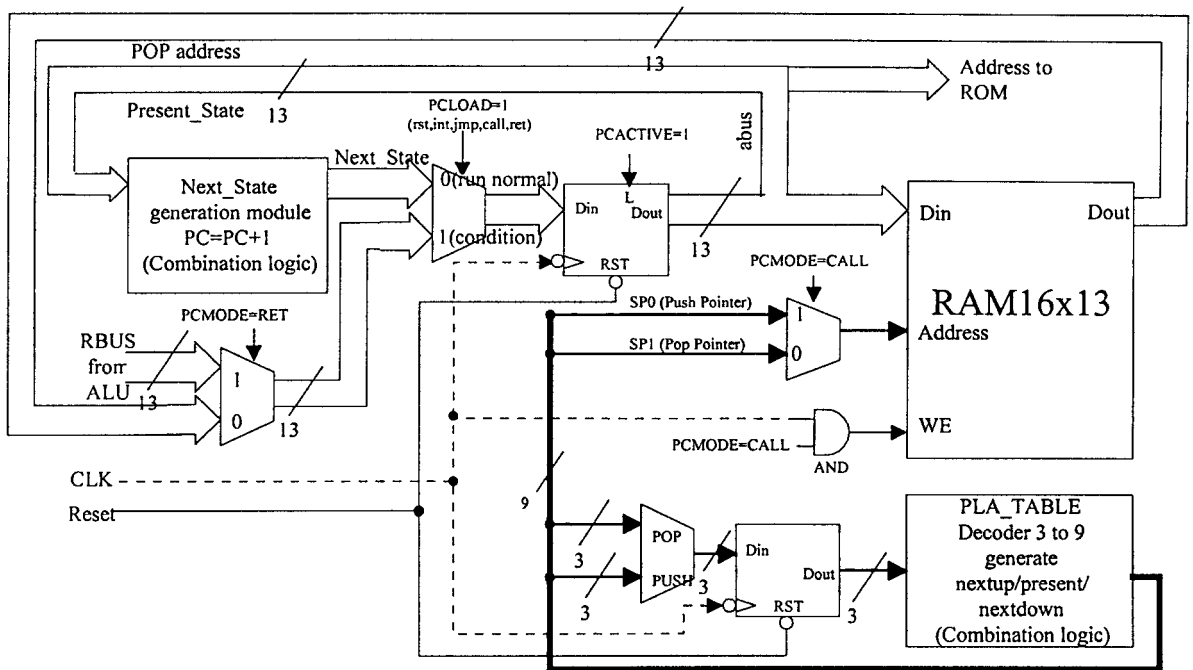
Opcode	Control Signals									
	ret/slp/ clrwdt	oldinst	Dest. W f	Carry feed	Literal (k)	xqmode XXX	C,DC,Z	amode XX	bmode XXX	smode XX
GOTO	000	0	NO	-	1	GOTO	000	OR	ALL0	PASS
IORLW	000	0	W	-	1	NORM	001	OR	PASS	PASS
MOVLW	000	0	W	-	1	NORM	000	OR	ALL0	PASS
RETFIE	100	0	NO	-	-	RTFI	000	-	-	-
RETLW	000	0	W	-	-	RETN	000	OR	ALL0	PASS
RETURN	000	0	NO	-	-	RETN	000	-	-	-
SLEEP	010	0	NO	-	-	NORM	000	-	-	-
SUBLW	000	0	W	1	1	NORM	111	ADD	INVT	PASS
XORLW	000	0	W	-	1	NORM	001	XOR	PASS	PASS
TRISA	000	1	F	-	-	NORM	000	OR	X111	PASS
TRISB	000	1	F	-	-	NORM	000	OR	X111	PASS
OPTION	000	1	F	-	-	NORM	000	OR	X111	PASS

ตารางที่ 5.2 แสดงการถอดรหัสของรูปแบบคำสั่งต่างๆ เป็นสัญญาณควบคุมแบบมีเงื่อนไข

Description	Intreg, execint	pclupd	xqmode	zflag	pactive	pcmode	opcfeed
Normal	X0	0	Norm	-	1	RUN	INST
Interrupt	01	0	Norm	-	0	RUN	C04
Interrupt	11	0	Norm	-	1	RUN	INST
Goto	XX	0	Goto	-	1	LOAD	NOP
Call	XX	0	Call,icall	-	1	CALL	NOP
Return	XX	0	Retn,rtfi	-	1	RETURN	NOP
Skip/zr(z=0)	XX	0	Skzr	0	1	RUN	INST
Skip/zr(z=1)	XX	0	Skzr	1	1	RUN	NOP
Skip/nz(z=0)	XX	0	Sknz	0	1	RUN	NOP
Skip/nz(z=1)	XX	0	Sknz	1	1	RUN	INST
Write PCL goto	XX	1	XXX	-	1	LOAD	LOAD

5.1.3 โมดูลพีซีเอสเอ็ม

เป็นโมดูลที่ออกแบบให้ทำหน้าที่ในส่วนของการสร้างตัวชี้ตำแหน่งแอดเดรสของหน่วยความจำโปรแกรมภายนอกของระบบไมโครคอนโทรลเลอร์ และทำหน้าที่เก็บค่าแอดเดรสเมื่อซีพียูเอ็กซีคิวต์คำสั่ง CALL หรือ GOTO โดยจะออกแบบให้มีโมดูลย่อย 2 โมดูล คือ (1) โปรแกรมเคาน์เตอร์ (PC: Program Counter) ขนาด 13 บิตทำหน้าที่สร้างตัวชี้ตำแหน่งของโปรแกรมคำสั่งในหน่วยความจำโปรแกรมภายนอกขนาด 8,096 ตำแหน่ง โดยโปรแกรมเคาน์เตอร์จะเซตทุกบิตให้เป็น 0x000 ถ้าระบบอยู่ในสถานะรีเซ็ต เพื่อที่จะให้ซีพียูกลับไปเริ่มต้นทำงานที่ตำแหน่ง 0x000 แต่ถ้าระบบไม่อยู่ในสถานะรีเซ็ต โปรแกรมเคาน์เตอร์จะเพิ่มค่าตัวเองโดยอัตโนมัติ หรือไม่เช่นนั้นผลลัพธ์ของการเอ็กซีคิวต์คำสั่งนั้นจะเป็นตัวกำหนดค่าให้กับโปรแกรมเคาน์เตอร์เอง (2) เรจิสเตอร์สแตก (Stack register) เป็นพื้นที่หน่วยความจำเพื่อใช้เก็บค่าของโปรแกรมเคาน์เตอร์ชั่วคราวมีความลึก 8 ระดับ โดยแต่ละระดับมีขนาด 13 บิต ถ้ามีการพุช (Push) 1 ครั้ง สแตกจะทำการเก็บค่าของโปรแกรมเคาน์เตอร์ไว้ 1 ค่า โดยสแตกจะข้อมูลในการพุช 8 ครั้งต่อเนื่องกัน หากมีการพุชเป็นครั้งที่ 9 ซีพียูจะนำค่าของโปรแกรมเคาน์เตอร์ในครั้งที่ 9 เก็บลงในสแตกที่เก็บค่าเคาน์เตอร์ในครั้งที่ 1 แทน แต่เนื่องจากพีไอซีไม่มีคำสั่งพุชและป๊อป (Pop) ให้ใช้ ดังนั้นการพุชจะเกิดในกรณีที่ คำสั่ง CALL ถูกเอ็กซีคิวต์ โดยจะนำค่าโปรแกรมเคาน์เตอร์ในขณะนั้น (เพิ่มค่าขึ้น 1) เก็บในสแตกระดับที่ 1 และสแตกลำดับที่ 1 จะดันค่าเข้าสแตกระดับที่ 2 โดยอัตโนมัติ ถ้ามีการ CALL มากกว่า 2 ครั้ง ส่วนการป๊อปเป็นการนำค่าระดับบนสุดของสแตก (TOS: Top of Stack) ที่ถูกเก็บเขียนลงในโปรแกรมเคาน์เตอร์ในกรณีที่ซีพียูเอ็กซีคิวต์คำสั่ง RETRUN, RETFIE หรือ RETLW จะเห็นได้ว่าเรจิสเตอร์สแตกจะมีการเก็บค่าโปรแกรมเคาน์เตอร์ในลักษณะข้อมูลเข้าก่อนออกทีหลังหรือเอฟไอแอลโอ (FILO: First In Last Out) ในการสร้างเรจิสเตอร์สแตกจะใช้แรมภายในของเอฟพีจีเอเพราะมีความเร็วสูงในการอ่านและเขียนโดยแรมที่ใช้มีขนาด 16X13 ไบต์โดย โดยออกแบบให้มีโครงสร้างดังรูปที่ 5.4



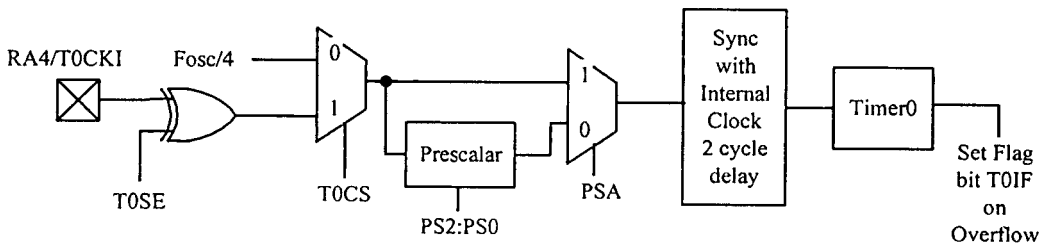
รูปที่ 5.4 โครงสร้างสถาปัตยกรรมภายในของไมโครพรีเซสเซอร์

จากรูปที่ 5.4 เมื่อชิพเริ่มทำงาน โมดูล Next_State generation ในไมโครพรีเซสเซอร์จะทำการสร้างค่าแอดเดรสเริ่มต้นซึ่งเริ่มจากค่า 0x000 ตามสัญญาณนาฬิกาขบขาลง เพื่อใช้ชี้ตำแหน่งแรกของหน่วยความจำโปรแกรม เพื่อให้ชิพทำการแพทช์ข้อมูลคำสั่งเข้าประมวลผล จากนั้นโมดูล Next_State generation จะทำการสร้างแอดเดรสค่าถัดไปในกรณีที่ชิพเอ็กซีคิวต์คำสั่งปกติ (สัญญาณ pload มีค่าเป็น 0) แต่ถ้าชิพเอ็กซีคิวต์คำสั่งบรานซ์, เกิดอินเตอร์รัพต์ หรือระบบเกิดการรีเซ็ต (สัญญาณ pload มีค่าเป็น 1) ค่าแอดเดรสที่ต้องการให้ชิพกระโดดไปทำงานจะมาจากสัญญาณอาร์บัส (RBUS) ก่อนที่ชิพจะทำการกระโดดไปยังแอดเดรสที่ต้องการ ระบบจะทำการพูค่าแอดเดรสปัจจุบันลงในเรจิสเตอร์สแตกก่อน แต่เมื่อกรณีที่ชิพเอ็กซีคิวต์คำสั่ง RETURN, RETFIE หรือ RETLW ระบบจะทำการป้อนค่าแอดเดรสจากเรจิสเตอร์สแตกได้เช่นกัน ในการออกแบบให้การพูและป้อนข้อมูลออกจากเรจิสเตอร์สแตกจะใช้เทคนิคการออกแบบโดยใช้การสร้างตัวชี้ตำแหน่งแบบคู่ (dual pointer) โดยที่ตัวชี้ตำแหน่งในเรจิสเตอร์สแตกจะมีขนาด 3 บิต เมื่อเริ่มต้นตัวชี้ตำแหน่งจะมีค่าเป็น 000 และทำการสร้างตัวชี้ตำแหน่งต่อไปโดยใช้การเปิดตารางค่าความจริงเพื่อใช้สร้างตัวชี้ตำแหน่งตัวต่อไปแต่วิธีนี้จะต้องใช้อุปกรณ์มัลติเพลกซ์ในการเลือกตัวชี้ตำแหน่งตัวใดในการทำงาน ซึ่งการออกแบบในลักษณะนี้ทำให้ลดไขเคลการทำงานลงเหลือเพียง 1 คาบสัญญาณนาฬิกาเท่านั้น

5.1.4 โมดูลเอฟอาร์เอ็ม

เป็นโมดูลออกแบบให้ทำหน้าที่เป็นหน่วยความจำข้อมูลของระบบโดยประกอบไปด้วย 2 ส่วนคือ เรจิสเตอร์ฟังก์ชันพิเศษและเรจิสเตอร์ใช้งานทั่วไปโดยหน่วยความจำข้อมูลทั้งสองส่วนจะถูกจัดเป็นแบงก์ ในการติดต่อหน่วยความจำในแต่ละแบงก์ จะต้องกำหนดค่าของบิตควบคุมอาร์พีศูนย์และบิตอาร์พีหนึ่งในเรจิสเตอร์สถานะ ในการออกแบบโมดูลเอฟอาร์เอ็มจะมีข้อจำกัดในเรื่องของเอฟฟิเจียที่ใช้มีจำนวนฟลิปฟลอปไม่มากพอที่สามารถทำการออกแบบหน่วยความจำข้อมูลทั้งหมดของพีไอซีไมโครคอนโทรลเลอร์ดังนั้น จะทำการออกแบบเพียงเรจิสเตอร์พิเศษที่จำเป็นต่อการทำงาน และส่วนของหน่วยความจำข้อมูลแบงก์ศูนย์เพียงแบงก์เดียว ส่วนแบงก์หนึ่งจะมีเพียงเรจิสเตอร์ฟังก์ชันพิเศษบางตัวเท่านั้น ดังมีรายละเอียดต่อไปนี้

(1) ไทเมอร์ศูนย์เรจิสเตอร์ (TMR0 register) เป็นเรจิสเตอร์ตัวนับ/ตัวจับเวลา (Real time clock / counter) ขนาด 8 บิต มีแอดเดรสอยู่ที่ตำแหน่ง 0x001 ในหน่วยความจำข้อมูลแบงก์ศูนย์ ไทเมอร์ศูนย์เรจิสเตอร์นี้สามารถถูกเขียนและอ่านโดยโปรแกรมและสามารถถูกเพิ่มค่าตัวเองได้ โดยการใช้สัญญาณนาฬิกาจากคำสั่งภายในหรือโดยใช้ขอบขาสัญญาณจากภายนอกที่ป้อนเข้าที่ขา RA4/T0CKI ดังรูปที่ 5.5



รูปที่ 5.5 แสดงไคอะแกรมการทำงานของไทเมอร์ศูนย์เรจิสเตอร์

(2) เรจิสเตอร์สถานะ (STATUS register) เป็นเรจิสเตอร์ที่ใช้เก็บสถานะทางคณิตศาสตร์ของเอแอลยู ได้แก่ แฟล็กตัวทดหรือยืม (C: Carry flag) แฟล็กตัวทดหรือยืมระหว่างหลัก (DC: Digit Carry/borrow) แฟล็กศูนย์ (Z: Zero flag) หรือระบบอยู่ในสภาวะรีเซ็ต และใช้เป็นตัวกำหนดการเลือกแบงก์ของหน่วยความจำข้อมูล การเข้าถึงเรจิสเตอร์สถานะเพื่ออ่านและเขียนข้อมูลสามารถกระทำได้ด้วยวิธีการเดียวกับการอ่านและเขียนเรจิสเตอร์ตัวอื่นๆ ทั่วไป มีแอดเดรสอยู่ที่ตำแหน่ง 0x003 ในหน่วยความจำข้อมูลแบงก์ศูนย์ โดยมีรายละเอียดค่าแต่ละบิตของเรจิสเตอร์สถานะดังนี้

7						0	
IRP	RP1	RP0	TO	PD	Z	DC	C
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x

IRP เป็นบิตเลือกแ่งก์ของเรจิสเตอร์ (ไม่ใช่ต้องกำหนดให้เป็น 0 เท่านั้น)

RP1-RP0 เป็นบิตเลือกแ่งก์ของเรจิสเตอร์ โดย 00 = แ่งก์ศูนย์ 01 = แ่งก์หนึ่ง (แ่งก์หนึ่งใช้ไม่ได้)

TO บิตขอบเขตเวลา (Time Out bit) อ่านได้อย่างเดียวเท่านั้น เป็น 1 เมื่อจ่ายไฟเลี้ยงให้แก่พีไอซี หรือมีการกระทำคำสั่ง CLRWDT หรือ SLEEP เป็น 0 เมื่อวอตช์ไทมเมอร์เกิดไทม์เอาต์

PD บิตเพาเวอร์ดาวน์ (Power Down bit) อ่านได้เช่นเดียวเท่านั้น เป็น 1 เมื่อการกระทำคำสั่ง CLRWDT เป็น 0 เมื่อการกระทำคำสั่ง SLEEP

Z บิตศูนย์ เป็น 1 เมื่อการกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้วเป็นศูนย์ เป็น 0 เมื่อการกระทำคำสั่งทางคณิตศาสตร์และลอจิกแล้วเป็นไม่เป็นศูนย์

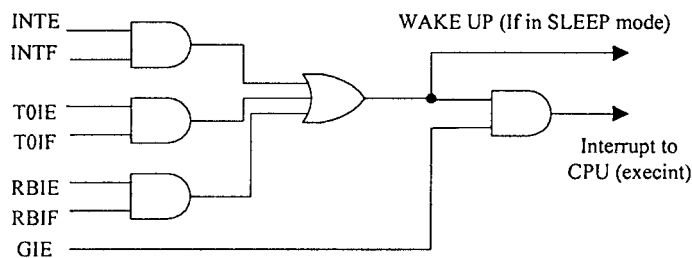
DC บิตทดหรือยืมระหว่างหลัก ในกรณีกระทำคำสั่ง ADDWF และ ADDLW บิต DC จะเกิดผลดังนี้ เป็น 1 เมื่อเกิดการทดจากบิตล่างที่ 4 ไปยังกลุ่มบิตบนที่ 5 เป็น 0 เมื่อไม่เกิดการทดจากบิตล่างที่ 4 ไปยังกลุ่มบิตบนที่ 5 หรือในกรณีกระทำคำสั่ง SUBWF และ SUBLW บิต DC จะเกิดผลดังนี้ เป็น 1 เมื่อไม่มีการยืมจากบิตล่างที่ 4 ไปกลุ่มบิตบนที่ 5 เป็น 0 เมื่อมีการยืมจากบิตล่างที่ 4 ไปกลุ่มบิตบนที่ 5

C บิตทดหรือยืม ในกรณีกระทำคำสั่ง ADDWF และ ADDLW บิต C จะเกิดผลดังนี้ เป็น 1 เมื่อเกิดการทดจากบิต MSB เป็น 0 เมื่อไม่เกิดการทด หรือในกรณีกระทำคำสั่ง SUBWF และ SUBLW บิต C จะเกิดผลดังนี้ เป็น 1 เมื่อไม่มีการยืมจากบิต MSB เป็น 0 เมื่อมีการยืมค่า

(3) เรจิสเตอร์เอฟเอสอาร์ (FSR: File Select Register) เป็นเรจิสเตอร์ที่ใช้เป็นตัวอ้างอิงแอดเดรสทางอ้อม มีขนาด 8 บิต มีแอดเดรสอยู่ที่ตำแหน่ง 0x004 ในหน่วยความจำข้อมูลแ่งก์ศูนย์ โดยที่บิต 0-4 จะเป็นการเลือกเรจิสเตอร์ไฟล์ตัวใดตัวหนึ่งจาก 32 ตัว ที่มีในการอ้างอิงแอดเดรสโดยอ้อม (โดยใช้เรจิสเตอร์ INDF ในกรณีที่เป็นคำสั่ง File Oriented) ส่วนบิตที่ 5-7 ของ เรจิสเตอร์จะอ่านได้อย่างเดียว โดยทั่วไปอ่านค่าได้เท่ากับ 1 แต่ถ้าไม่ใช้การอ้างอิงแอดเดรสทางอ้อม เรจิสเตอร์นี้จะใช้เป็นเรจิสเตอร์ทั่วไป ขนาด 5 บิตได้

(4) เรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์สูง (PCLATH register) เป็นเรจิสเตอร์ขนาด 5 บิตที่ใช้ช่วยในการเขียนหรืออ่านข้อมูล 5 บิตบนของโปรแกรมเคาน์เตอร์ไบต์สูง (PCH) เพราะไม่สามารถเขียนหรืออ่าน 5 บิตบนของโปรแกรมเคาน์เตอร์ได้โดยตรง เรจิสเตอร์นี้มีแอดเดรสอยู่ที่ตำแหน่ง 0x00A ในหน่วยความจำข้อมูลแบบกึ่งศูนย์ โดยข้อมูลจากเรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์สูงจะถูกส่งไปยังโปรแกรมเคาน์เตอร์ไบต์สูง ในกรณีเมื่อซีพียูเอ็กซีคิวต์คำสั่ง CALL หรือ GOTO ค่าของเรจิสเตอร์โปรแกรมเคาน์เตอร์ไบต์สูงบิตที่ 4 และ 3 จะถูกเขียนลงไปที่โปรแกรมเคาน์เตอร์ไบต์สูงบิตที่ 12 และ 11 ตามลำดับ

(5) เรจิสเตอร์ควบคุมการอินเทอร์รัพต์ (INTCON: Interrupt Control register) เป็นเรจิสเตอร์ที่เก็บค่าบิตของการอินเริตสัญญาณอินเทอร์รัพต์ต่างๆ มีแอดเดรสอยู่ที่ตำแหน่ง 0x00B ในหน่วยความจำข้อมูลแบบกึ่งศูนย์



รูปที่ 5.6 แสดงสัญญาณลอจิกของการอินเทอร์รัพต์ภายในเรจิสเตอร์ควบคุมการอินเทอร์รัพต์

(6) เรจิสเตอร์ออปชั่น (OPTION register) เป็นเรจิสเตอร์ขนาด 8 บิต ใช้สำหรับการกำหนดค่าปริสเกลเลอร์ (Prescaler) ที่จะใช้กับไทเมอร์ศูนย์เรจิสเตอร์หรือวอตช์ด็อกไทเมอร์ และบรรจิบิตควบคุมการอินเทอร์รัพต์จากสัญญาณภายนอก มีแอดเดรสอยู่ที่ตำแหน่ง 0x081 ในหน่วยความจำข้อมูล สำหรับรายละเอียดของเรจิสเตอร์นี้แต่ละบิตดังนี้

7							0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

โดยที่ บิต PSA เป็นตัวกำหนดทิศทางของปริสเกลเลอร์ถ้ามีค่าเป็น 1 หมายถึงทิศทางกำหนดไปทิวอตช์ด็อกไทเมอร์ แต่ถ้ามีค่า 0 หมายถึงทิศทางกำหนดไปที่ไทเมอร์ศูนย์เรจิสเตอร์ และบิต PS2: PS0 เป็นตัวกำหนดอัตราของปริสเกลเลอร์

(7) เรจิสเตอร์ใช้งานทั่วไป (GPR: General Purpose Register) เป็นเรจิสเตอร์ขนาด 32X8 ไบต์ใช้เป็นเรจิสเตอร์ใช้งานทั่วไป มีแอดเดรสอยู่ที่ตำแหน่ง 0x010 ถึง 0x02F ในหน่วยความจำข้อมูลแบ่งกึ่งศูนย์ จะทำการสร้างจากแรมภายในของเอฟพีจีเอทั้งหมด

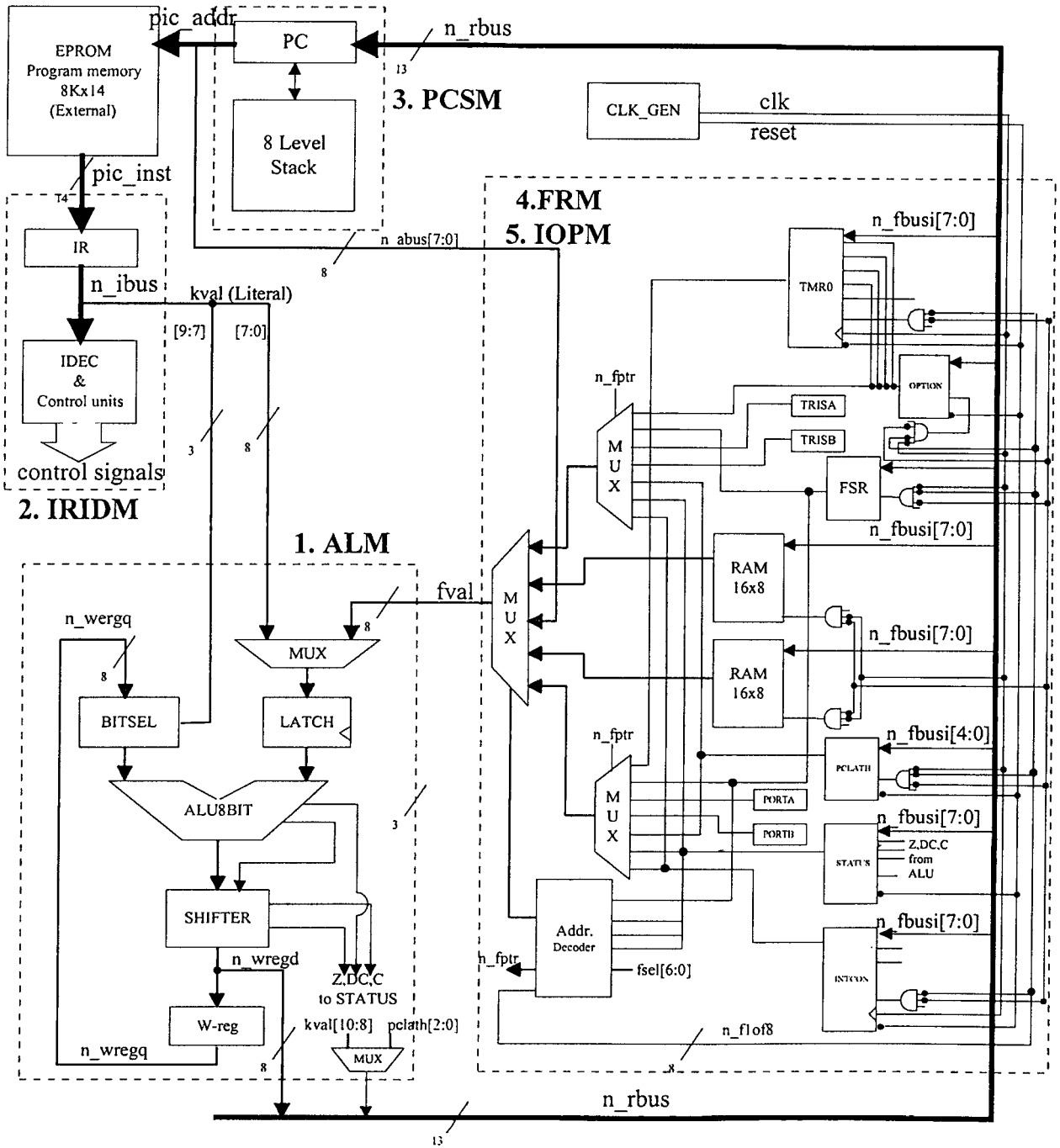
(8) วอตช์ด็อกไทมเมอร์ (WDT: Watch Dog Timer) เป็นวงจรกำเนิดสัญญาณอย่างต่อเนื่องที่โครงสร้างภายในเป็นอาร์ซี (RC: Ripple Counter) ขนาด 10 บิต ที่ทำหน้าที่ช่วยตัวจับการทำงาน ของซีพียูของไมโครคอนโทรลเลอร์ว่ายังทำงานเป็นปกติอยู่หรือไม่

(9) มัลติเพิลิกเซอร์และแอดเดรสดีโคเดอร์ทำหน้าที่ในการเลือกสัญญาณข้อมูลที่ได้มาจากไฟล์เรจิสเตอร์ชนิดต่างๆ ที่จะส่งไปยังโมดูลเอแอลเอ็ม โดยสัญญาณเลือกจะถูกดีโค็ดมาจากแอดเดรสดีโคเดอร์ โดยมีโครงสร้างโดยรวมของทุกโมดูลย่อยในรูปที่ 5.6

5.1.5 โมดูลไอโอพีเอ็ม

เป็นโมดูลที่ออกแบบให้ทำหน้าที่เป็นส่วนติดต่อกับอุปกรณ์ภายนอกในการส่งข้อมูลออกและข้อมูลเข้า หรือเป็นส่วนของไอโอพอร์ตนั่นเอง โดยมีสองพอร์ตซึ่งเป็นพอร์ตที่สามารถกำหนดทิศทางให้เป็นพอร์ตอินพุตหรือเอาต์พุตได้จากการกำหนดค่าในรีจิสเตอร์ควบคุมซึ่งประกอบด้วยโมดูลย่อย ๆ ดังนี้ (1) พอร์ตเอและเรจิสเตอร์ควบคุม (PORTA & TRISA register) พอร์ตเอเป็นไอโอพอร์ตนขนาด 5 บิต โดยถูกกำหนดจากเรจิสเตอร์ควบคุม ในการกำหนดพอร์ต เอเป็นอินพุตหรือเอาต์พุต ได้โดยโหลดค่า 1 ลงในบิตของ เรจิสเตอร์ควบคุมที่ต้องการให้เป็นขาอินพุต และโหลดค่า 0 ลงในบิตของเรจิสเตอร์ควบคุมที่ต้องการให้เป็นขาเอาต์พุต โดยพอร์ตเอมีแอดเดรสตำแหน่งที่ 0x005 ในหน่วยความจำข้อมูล และเรจิสเตอร์ควบคุมอยู่ที่แอดเดรสตำแหน่งที่ 0x085 ในหน่วยความจำข้อมูล เรจิสเตอร์ควบคุมนี้สามารถเขียนได้อย่างเดียวไม่สามารถอ่านได้ (2) พอร์ตบีและเรจิสเตอร์ควบคุม (PORTB & TRISB register) พอร์ตบีเป็นไอโอพอร์ตนขนาด 8 บิต โดยถูกกำหนดจากเรจิสเตอร์ควบคุมในการกำหนดพอร์ตบีเป็นอินพุตหรือเอาต์พุต ได้โดยโหลดค่า 1 ลงในบิตของ เรจิสเตอร์ควบคุมที่ต้องการให้เป็นขาอินพุต และโหลดค่า 0 ลงในบิตของเรจิสเตอร์ควบคุมที่ต้องการให้เป็นขาเอาต์พุต โดยพอร์ตบีมีแอดเดรสตำแหน่งที่ 0x006 ในหน่วยความจำข้อมูล และเรจิสเตอร์ควบคุมอยู่ที่แอดเดรสตำแหน่งที่ 0x086 ในหน่วยความจำข้อมูล เรจิสเตอร์ควบคุมนี้สามารถเขียนได้อย่างเดียวไม่สามารถอ่านได้ โดยโมดูลนี้จะรวมอยู่ในโครงสร้างของโมดูลเอฟอาร์เอ็มในรูปที่ 5.7

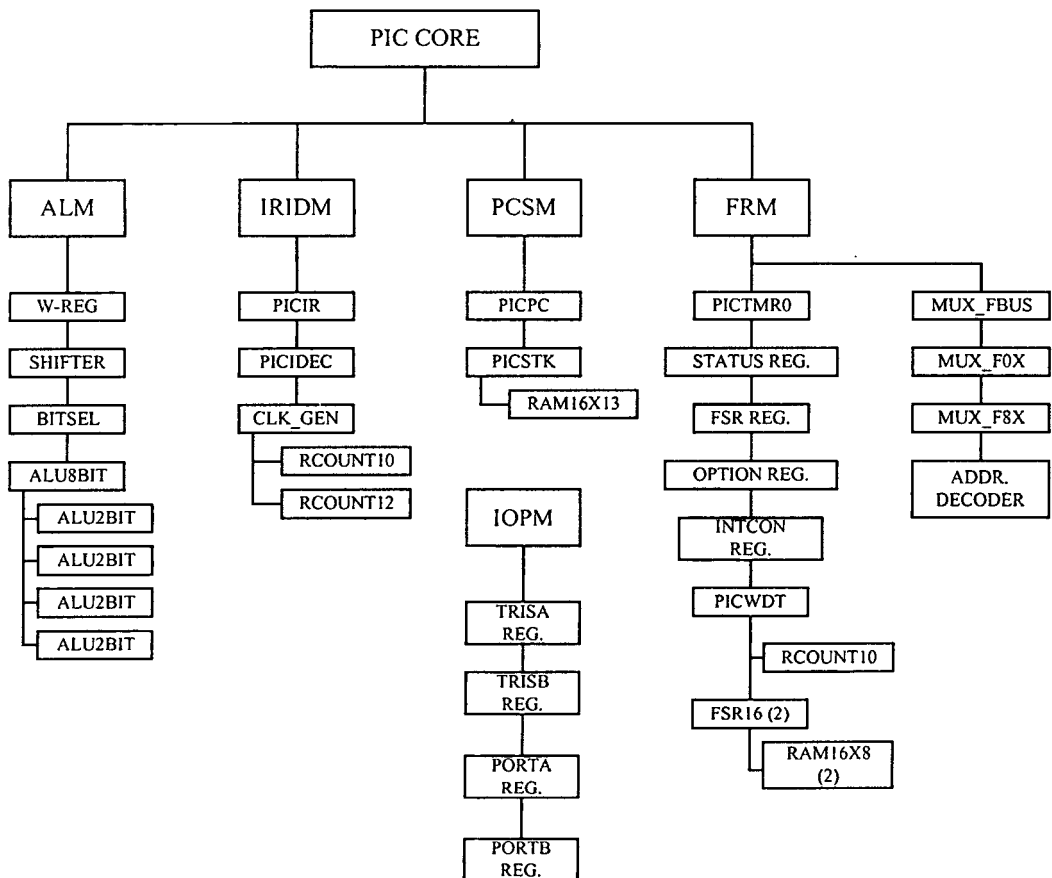
เมื่อนำทุกโมดูลที่ทำการออกแบบมาเชื่อมต่อกันจะได้โครงสร้างไมโครคอนโทรลเลอร์ที่
สมบูรณ์ดังแสดงในรูปที่ 5.8



รูปที่ 5.8 แสดงการเชื่อมต่อแต่ละ โมดูลภายในไมโครคอนโทรลเลอร์ที่ออกแบบ

5.2 การออกแบบระบบในรูปของภาษาวีเอชดีแอล

เมื่อทำการวิเคราะห์ถึงรายละเอียดของระบบที่ต้องการออกแบบจนได้รายละเอียดและข้อกำหนดต่างๆ ของการออกแบบ จากนั้นเขียนพฤติกรรมของแต่ละโมดูลที่ออกแบบในรูปของภาษาวีเอชดีแอลและจำลองฟังก์ชันการทำงาน (Function Simulation) เพื่อตรวจสอบความถูกต้องในแต่ละโมดูล โดยการเขียนจะต้องคำนึงรูปแบบการเขียนที่จะนำไปทำการสังเคราะห์ให้อยู่ในรูปของผังวงจรได้หรือไม่ นั่นคือขึ้นอยู่กับความสามารถและข้อกำหนดของรูปแบบการเขียน ของโปรแกรมที่เป็นเครื่องมือช่วยในการสังเคราะห์วงจร เนื่องจากระบบไมโครคอนโทรลเลอร์ที่ทำการออกแบบมีหลายโมดูลย่อยๆ ดังนั้นในการเขียนควรเขียนในลักษณะเป็นลำดับขั้น เพื่อเป็นการง่ายในการตรวจสอบแก้ไขได้ในรายละเอียดของแต่ละลำดับขั้น และช่วยให้โปรแกรมที่ใช้ในการสังเคราะห์วงจรทำการสังเคราะห์วงจรอย่างมีประสิทธิภาพ ในเรื่องของลดเวลาที่ใช้และขนาดของวงจรที่ถูกสังเคราะห์ และความเป็นระเบียบเรียบร้อยของวงจรที่ถูกสังเคราะห์ โดยเริ่มจาก



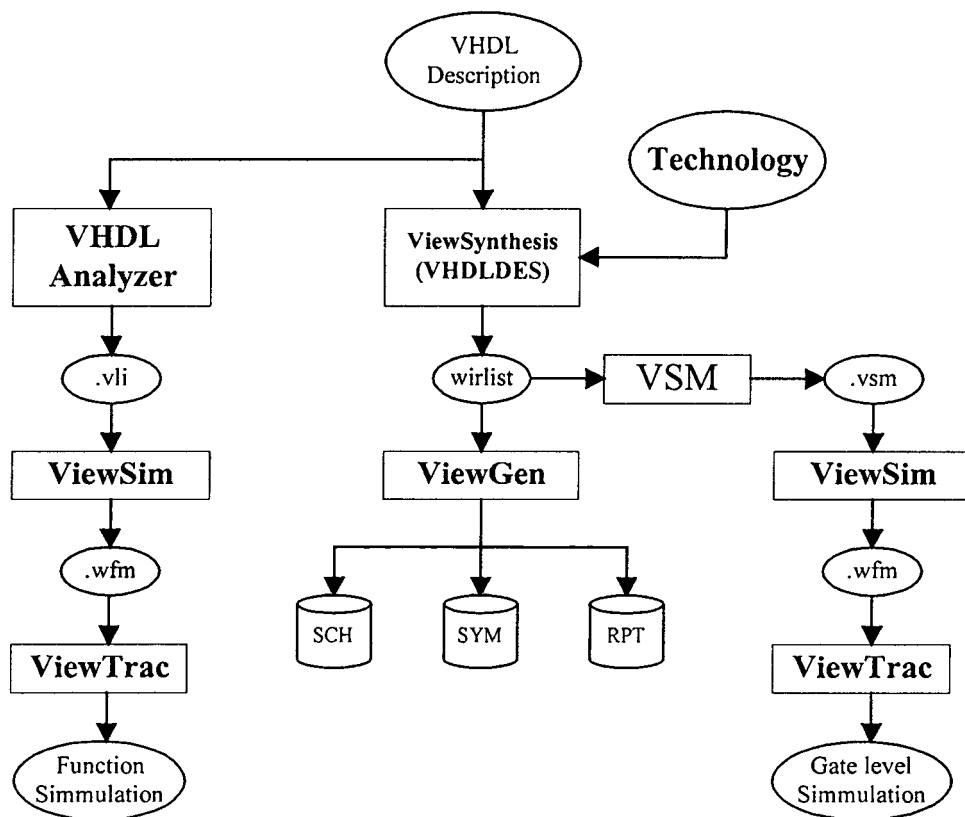
รูปที่ 5.9 แสดงลำดับชั้นของไฟล์โมดูลที่ออกแบบทั้งหมด

5.3 การสร้างต้นแบบจากภาษาวีเอชดีแอล

เมื่อได้โมเดลของระบบไมโครคอนโทรลเลอร์ในรูปภาษาวีเอชดีแอลในแต่ละโมดูลที่ทำการออกแบบแล้ว จะนำไปทดสอบโดยการจำลองการฟังก์ชันทำงานของแต่ละโมดูล เพื่อตรวจสอบความถูกต้องของฟังก์ชันและฐานเวลาในการทำงาน จากนั้นนำทุกโมดูลที่ออกแบบประกอบขึ้นเป็นระบบไมโครคอนโทรลเลอร์ที่สมบูรณ์ ทำการสังเคราะห์ให้อยู่ในรูปของลอจิกเกต (Logic gate) และทำการจำลองการทำงานในระดับเกต (Gate level simulation) ของผังวงจรที่ได้ เพื่อตรวจสอบความถูกต้องอีกครั้ง ก่อนนำไปสร้างลงบนอุปกรณ์เอพฟี่ไอ เพื่อนำไปทดสอบการทำงานในรูปของไมโครคอนโทรลเลอร์ต้นแบบต่อไป โดยมีสามขั้นตอนหลักคือ การออกแบบเอนทรี (Design Entry) การออกแบบอิมพลีเมนต์เตชัน (Design Implementation) การออกแบบในการตรวจสอบความถูกต้อง (Design Verification) ดังมีรายละเอียดดังนี้

5.3.1 การออกแบบเอนทรี (Design Entry)

การออกแบบเอนทรีจะเป็นขั้นตอนการนำโมเดลที่เขียนขึ้นจากภาษาวีเอชดีแอลมาทำการสังเคราะห์ให้อยู่ในรูปของผังวงจร ดังแสดงในรูปที่ 5.10 (เป็นขั้นตอนของโปรแกรม Viewlogic ซึ่งเป็นโปรแกรมช่วยในการสังเคราะห์วงจรที่ใช้) ซึ่งเริ่มต้นโดยนำโค้ดภาษาวีเอชดีแอลไปทำการคอมไพล์ (Compile) เพื่อตรวจสอบไวยากรณ์ของภาษาโดยใช้โปรแกรม VHDL Analyzer ถ้าไม่พบข้อผิดพลาดจะนำไปทำการจำลองการทำงานโดยใช้โปรแกรม ViewSim ในรูปของฟังก์ชันการทำงาน เมื่อตรวจสอบฟังก์ชันการทำงานถูกต้องก็จะนำไปทำการสังเคราะห์โดยใช้โปรแกรม Viewsynthesis ซึ่งในขั้นตอนนี้จะต้องเลือกใช้เทคโนโลยีเป้าหมายมาเป็นไลบรารีของผังวงจรที่จะถูกสังเคราะห์ขึ้น (โดยเลือกใช้เทคโนโลยี X4000 มาเป็นไลบรารี) ผลที่ได้จากการสังเคราะห์จะอยู่ในรูปของไฟล์เน็ตลิสต์ในระดับเกต (Gate level Netlist) ที่สามารถนำไปจำลองฐานเวลาการทำงานในระดับเกตได้โดยใช้โปรแกรม VSM โดยทำการแปลงไฟล์เน็ตลิสต์ไปเป็นไฟล์วีเอสเอ็ม (file.vsm) ที่โปรแกรม ViewSim สามารถนำไปจำลองการทำงาน โดยการจำลองการทำงานทั้งสองครั้งในรูปของฟังก์ชันการทำงานและระดับเกต เพื่อใช้ในการตรวจสอบความถูกต้องในการสังเคราะห์ผังวงจร เมื่อทำการตรวจสอบผลการจำลองทำงานถูกต้องเรียบร้อยแล้วจะเป็นขั้นตอนสุดท้ายของการออกแบบเอนทรีคือ การสร้างผังวงจรเพื่อนำไปใช้ในส่วนของการออกแบบอิมพลีเมนต์เตชัน โดยใช้โปรแกรม ViewGen ทำการสร้างผังวงจรหรือซิมโบลิก (Symbolic) ของผังวงจร และสร้างไฟล์ข้อมูลที่แสดงรายละเอียดอุปกรณ์ที่ใช้ในการสังเคราะห์

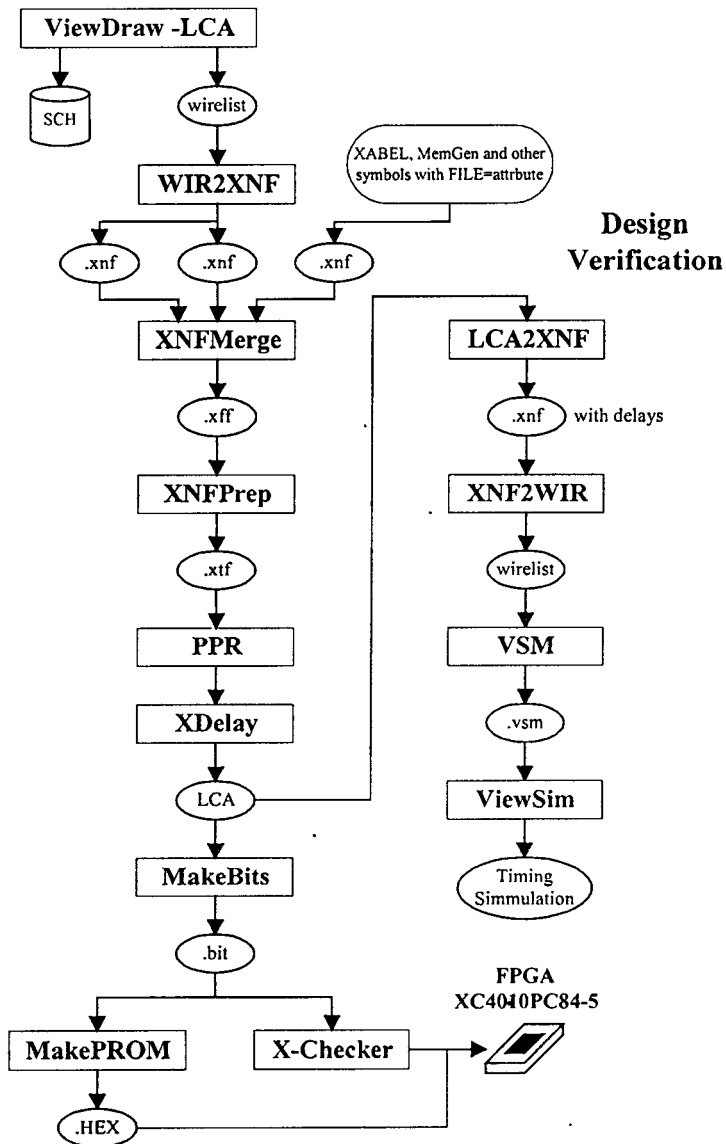


รูปที่ 5.10 ขั้นตอนการออกแบบเอนทรี (Design Entry)

5.3.2 การออกแบบอิมพลีเมนต์เตชั่น (Design Implementation)

เมื่อได้ผังวงจรทั้งหมดจากการออกแบบเอนทรีเรียบร้อยแล้ว จะนำวงจรทั้งหมดเข้าสู่ส่วนการออกแบบอิมพลีเมนต์เตชั่น ซึ่งเป็นการนำผังวงจรที่ได้มาสร้างลงบนอุปกรณ์เอฟพีจีเอดังรูปที่ 5.10 โดยจะใช้โปรแกรม XACT เวอร์ชัน 5.2 ของซึ่งมีขั้นตอนต่างๆ ดังนี้ เริ่มจากนำผังวงจรได้ทำการสังเคราะห์ได้มาทำการเติมบัฟเฟอร์ (Buffers) และแพด (Pad) ประเภทต่างๆ กับขาไอโอของวงจร และกำหนดขาไอโอกับหมายเลขขาไอโอของอุปกรณ์เอฟพีจีเอที่นำมาใช้ ตามข้อกำหนดของการออกแบบโดยใช้เอฟพีจีเอของไซริง (XILINX Inc.) โดยเลือกใช้เอฟพีจีเอตระกูล XC4010PC84-5 โดยใช้โปรแกรม ViewDraw เมื่อทำการแก้ไขเรียบร้อยแล้วจะอยู่ในรูปของไฟล์ Wirelist จากนั้นจะแปลงไฟล์ Wirelist ให้อยู่ในรูปของไฟล์เอ็กซ์เอ็นเอฟ (XNF: Xilinx Netlist Format) โดยใช้โปรแกรม WIR2XNF แล้วทำการรวมไฟล์เอ็กซ์เอ็นเอฟ ของแต่ละโมดูลย่อยๆ ที่ได้ ออกแบบไว้เป็นลำดับชั้นรวมเข้าเป็นไฟล์เดียวกันโดยใช้โปรแกรม XNFMerge แล้วทำการเพดเมนต์ (Placement) โดยใช้โปรแกรม PPR (Partition, Place and Route) จะได้ไฟล์แอลซีเอ (LCA: Logic Cell Array) ซึ่งสามารถนำไปโปรแกรมลงเอฟพีจีเอโดยผ่านสายเอ็กซ์เช็คเคอร์ (X-Checker cable) ทางคอมพิวเตอร์ได้โดยตรง หรือนำไปแปลงเป็นไฟล์ข้อมูลไบนารีที่สามารถ

นำไปโปรแกรมลงสู่อุปกรณ์แทนการใช้สายเอ็กซ์เชคเคอร์ได้ ซึ่งเป็นขั้นตอนสุดท้ายของการสร้างต้นแบบของระบบทั้งหมด



รูปที่ 5.11 ขั้นตอนการออกแบบอิมพลีเมนต์เตชั่นและขั้นตอนการตรวจสอบความถูกต้อง

5.3.3 การออกแบบในการตรวจสอบความถูกต้อง (Design Verification)

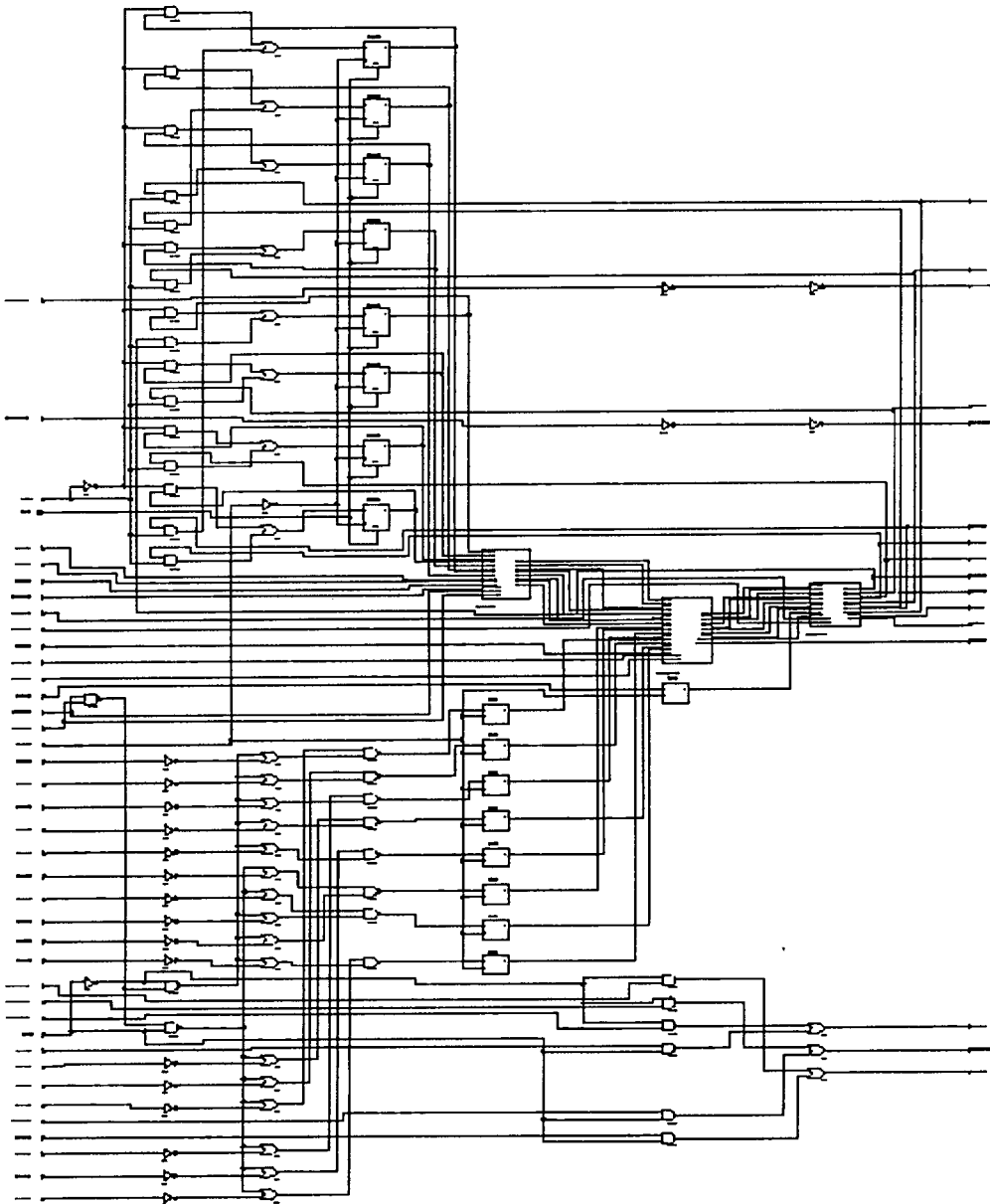
เป็นขั้นตอนตรวจสอบความถูกต้องในขั้นตอนการออกแบบอิมพลีเมนต์เตชั่น โดยจะทำการนำไฟล์ข้อมูลที่อยู่ในรูปแอลซีเอไปทำการจำลองฐานเวลาการทำงาน (Timing Simulation) ดังรูปที่ 5.11 โดยใช้โปรแกรม LCA2XNF แปลงไฟล์แอลซีเอให้กลับมาอยู่ในรูปของไฟล์เอ็กซ์เชคเคอร์เอฟ โดยจะมีข้อมูลของเวลาหน่วง (Delay times) ของวงจรรวมอยู่ด้วย เพื่อนำไปจำลองฐานเวลาโดยใช้โปรแกรม ViewSim อีกครั้งเพื่อยืนยันความถูกต้องของผังวงจรที่ได้

บทที่ 6

ผลการออกแบบ

จากโมเดลของภาษาวีเอชดีแอลที่เขียนอธิบายพฤติกรรมของไมโครคอนโทรลเลอร์ที่ออกแบบนำไปจำลองการทำงานหาข้อผิดพลาดและทำการแก้ไขจนถูกต้อง หลังจากนั้นนำมาสังเคราะห์ให้อยู่ในรูปของผังวงจร ได้ผังวงจรต่างๆ ดังต่อไปนี้

6.1 ผังวงจรของโมดูลเอแอลเอ็ม (ALM: Arithmetic and Logic Module)

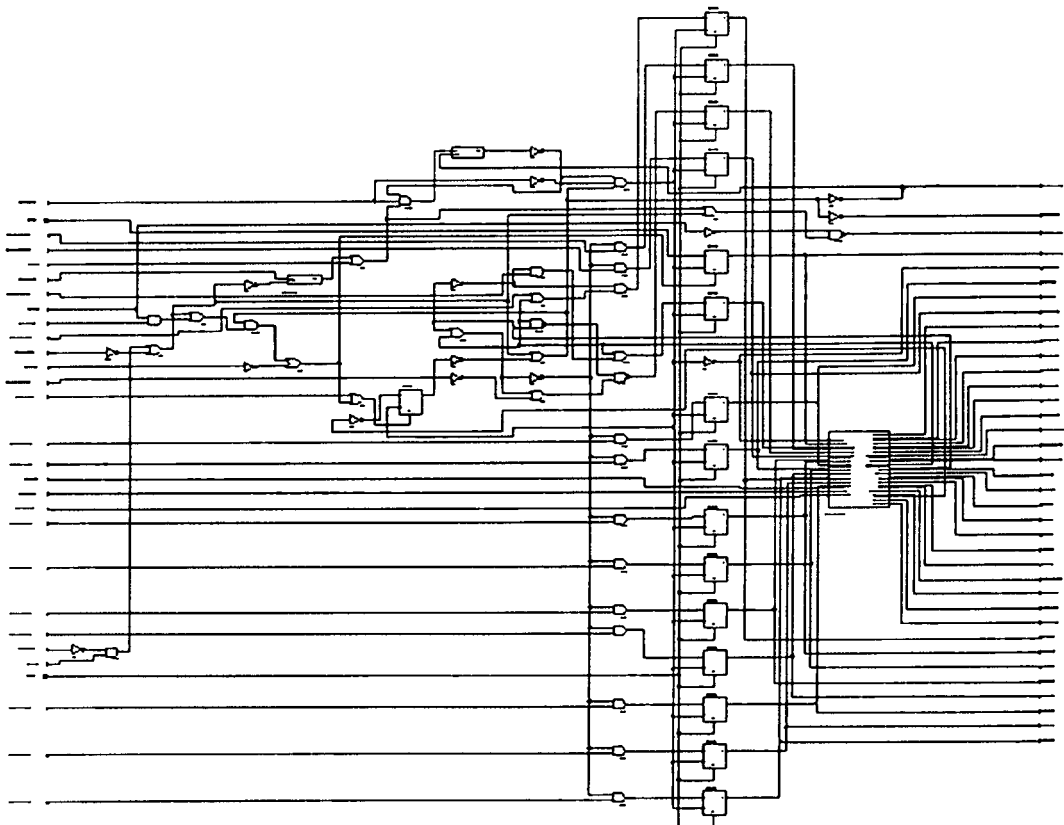


รูปที่ 6.1 ผังวงจรของ โมดูลเอแอลเอ็มที่ได้จากการสังเคราะห์

ตารางที่ 6.1 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรเอแอลเอ็มโมดูล

ชนิดเซลล์	จำนวน	พื้นที่/เซลล์	ชนิดเซลล์	จำนวน	พื้นที่/เซลล์
FD	9	0.00	FDR	8	0.00
AND2	111	0.25	INV	53	0.00
NAND2	44	0.25	NOR3	1	0.50
NOR4	1	0.75	OR2	80	0.25
OR3	2	0.50	OR4	25	0.75
XOR2	13	0.25			
เซลล์ทั้งหมด : 347 เซลล์			พื้นที่ทั้งหมด : 83.00		
Netlist Statistics					
Maximum level of gates		31	Total number of nets		387
Maximum pins per net		11	Average pins per net		2.86
Maximum load per net		16.00	Average load per net		1.84

6.2 ผังวงจรของโมดูลไออาร์ไอดีเอ็ม (IRIDM: Instruction Register and Instruction Decoder Module)

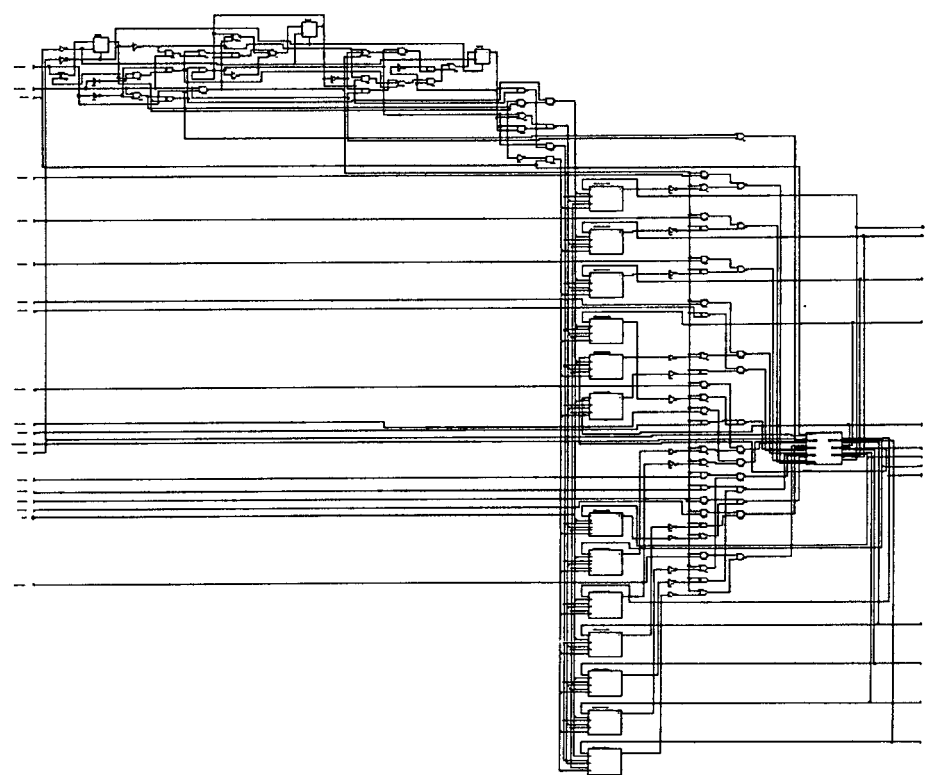


รูปที่ 6.2 ผังวงจร โมดูลไออาร์ไอดีเอ็มที่ได้จากการสังเคราะห์

ตารางที่ 6.2 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรไออาร์ไอดีเอ็มโมดูล

ชนิดเซลล์	จำนวน	พื้นที่/เซลล์	ชนิดเซลล์	จำนวน	พื้นที่/เซลล์
FDR	40	0.00	FDS	1	0.00
AND2	56	0.25	AND3	8	0.50
AND4	44	0.75	INV	66	0.00
NAND2	19	0.25	NAND3	7	0.50
NOR3	12	0.50	NOR4	2	0.75
OR2	55	0.25	OR3	13	0.50
OR4	11	0.75			
เซลล์ทั้งหมด : 291 เซลล์			พื้นที่ทั้งหมด : 63.00		
Netlist Statistics					
Maximum level of gates		10	Total number of nets		319
Maximum pins per net		19	Average pins per net		3.08
Maximum load per net		28.00	Average load per net		2.00

6.3 ผังวงจรของโมดูลพีซีเอสเอ็ม (PCSM: Program Counter and Stack Module)

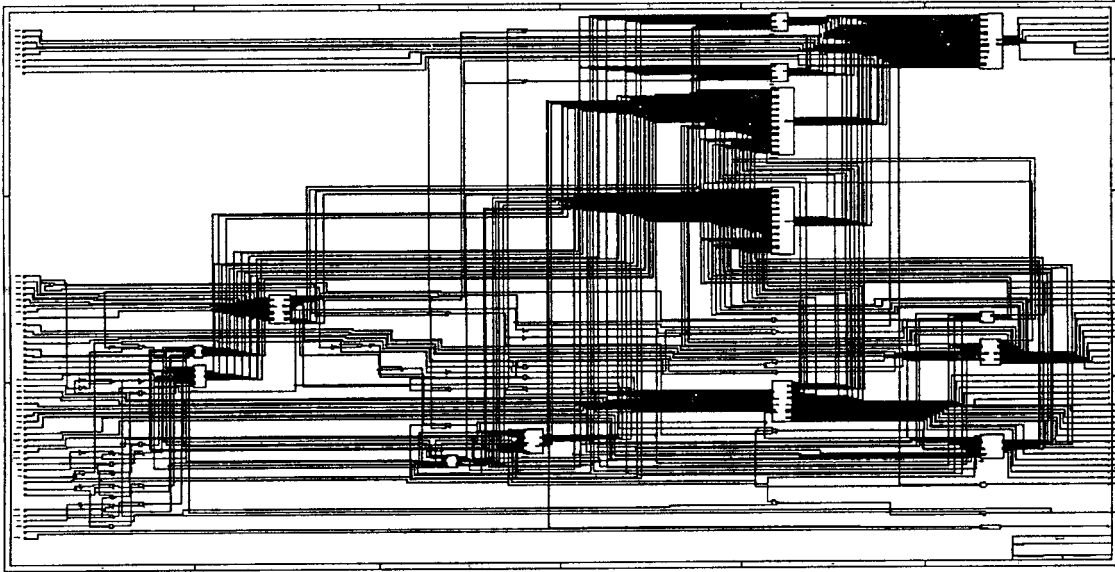


รูปที่ 6.3 ผังวงจร โมดูลพีซีเอสเอ็มที่ได้จากการสังเคราะห์

ตารางที่ 6.3 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรพีซีเอสเอ็ม โมดูล

ชนิดเซลล์	จำนวน	พื้นที่/เซลล์	ชนิดเซลล์	จำนวน	พื้นที่/เซลล์
RAM16X1	13	0.00	FRD	16	0.00
AND2	49	0.25	INV	34	0.00
NAND2	29	0.25	NAND3	1	0.50
NAND4	4	0.75	OR2	50	0.25
OR3	4	0.50	OR4	9	0.75
XOR2	1	0.25			
เซลล์ทั้งหมด : 210 เซลล์			พื้นที่ทั้งหมด : 44.50		
Netlist Statistics					
Maximum level of gates		15	Total number of nets		229
Maximum pins per net		17	Average pins per net		2.91
Maximum load per net		26.00	Average load per net		1.83

6.4 ผังวงจรของโมดูลเอฟอาร์เอ็มและโมดูลไอโอพีเอ็ม (FRM: File Register Module / IOPM: Input Output Port Module)

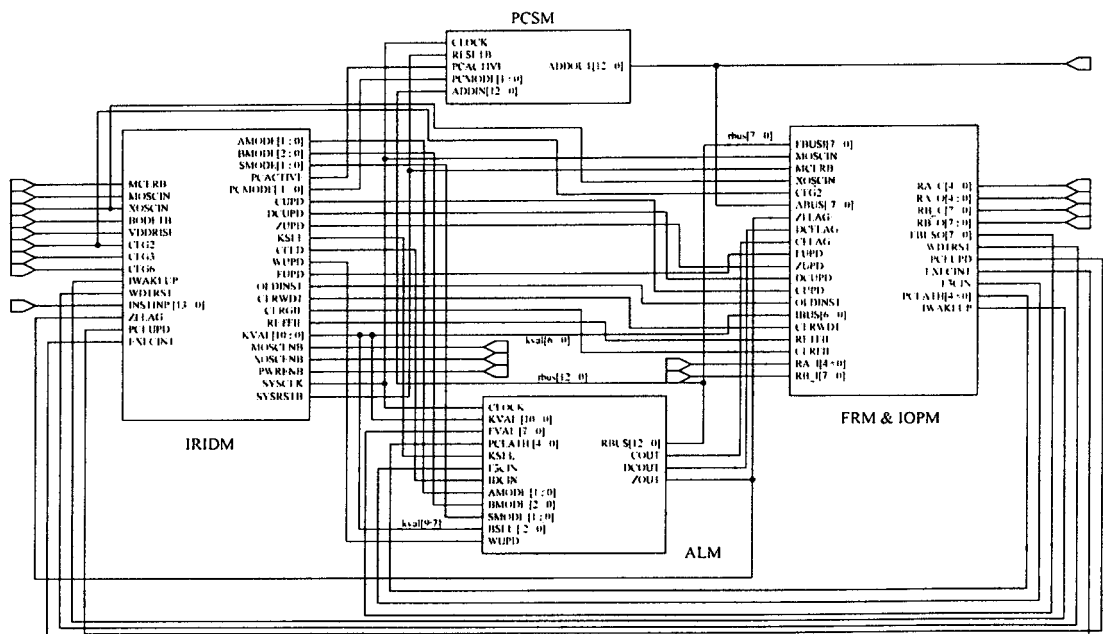


รูปที่ 6.4 ผังวงจร โมดูลเอฟอาร์เอ็มและ โมดูลไอ โอพีเอ็มที่ได้จากการสังเคราะห์

ตารางที่ 6.4 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรเอฟอาร์เอ็มและไอโอพีเอ็ม โมดูล

ชนิดเซลล์	จำนวน	พื้นที่/เซลล์	ชนิดเซลล์	จำนวน	พื้นที่/เซลล์
RAM16X1	16	0.00	FDR	46	0.00
FDS	13	0.00	LD	50	2.00
AND2	205	0.25	AND3	12	0.50
AND4	1	0.75	INV	119	0.00
NAND2	59	0.25	NAND3	14	0.50
NAND4	7	0.75	NOR3	21	0.50
NOR4	4	0.75	OR2	192	0.25
OR3	34	0.50	OR4	28	0.75
XOR2	16	0.25			
เซลล์ทั้งหมด : 837 เซลล์			พื้นที่ทั้งหมด : 288.50		
Netlist Statistics					
Maximum level of gates		20	Total number of nets		889
Maximum pins per net		27	Average pins per net		3.03
Maximum load per net		46.00	Average load per net		1.99

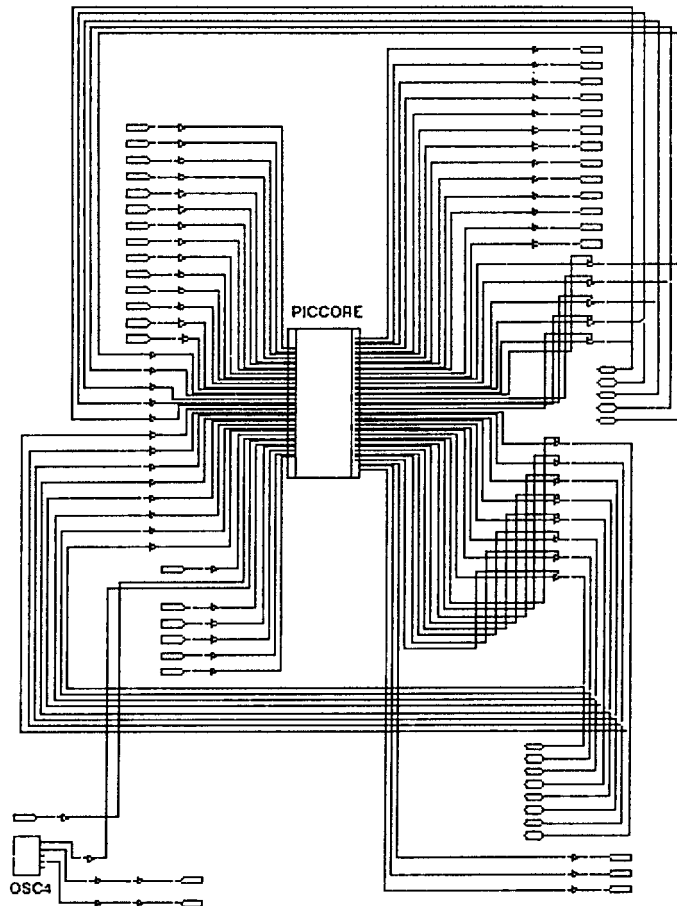
ซึ่งเมื่อได้ผังวงจรของแต่ละโมดูลที่สังเคราะห์เรียบร้อยแล้ว ก็จะนำมาเชื่อมต่อเข้าด้วยกัน ให้เป็นระบบไมโครคอนโทรลเลอร์ที่สมบูรณ์ดังแสดงในรูปที่ 6.10



รูปที่ 6.5 แสดงการเชื่อมต่อโมดูลต่างๆ ในระดับบนสุดของผังวงจรไมโครคอนโทรลเลอร์

ตารางที่ 6.5 แสดงรายละเอียดอุปกรณ์ภายในผังวงจรไมโครคอนโทรลเลอร์ที่ออกแบบ

ชนิดเซลล์	จำนวน	พื้นที่/เซลล์	ชนิดเซลล์	จำนวน	พื้นที่/เซลล์
FD	49	0.00	OR4	73	0.75
FDR	70	0.00	XOR2	30	0.25
FDS	14	0.00	NAND2	151	0.25
LD	50	2.00	NAND3	22	0.50
AND2	421	0.25	NAND4	11	0.75
AND3	20	0.50	NOR3	34	0.50
AND4	2	0.75	NOR4	7	0.75
OR2	377	0.25	INV	272	0.00
OR3	53	0.50	RAM16X1	29	0.00
เซลล์ทั้งหมด : 1,685 เซลล์			พื้นที่ทั้งหมด : 479		



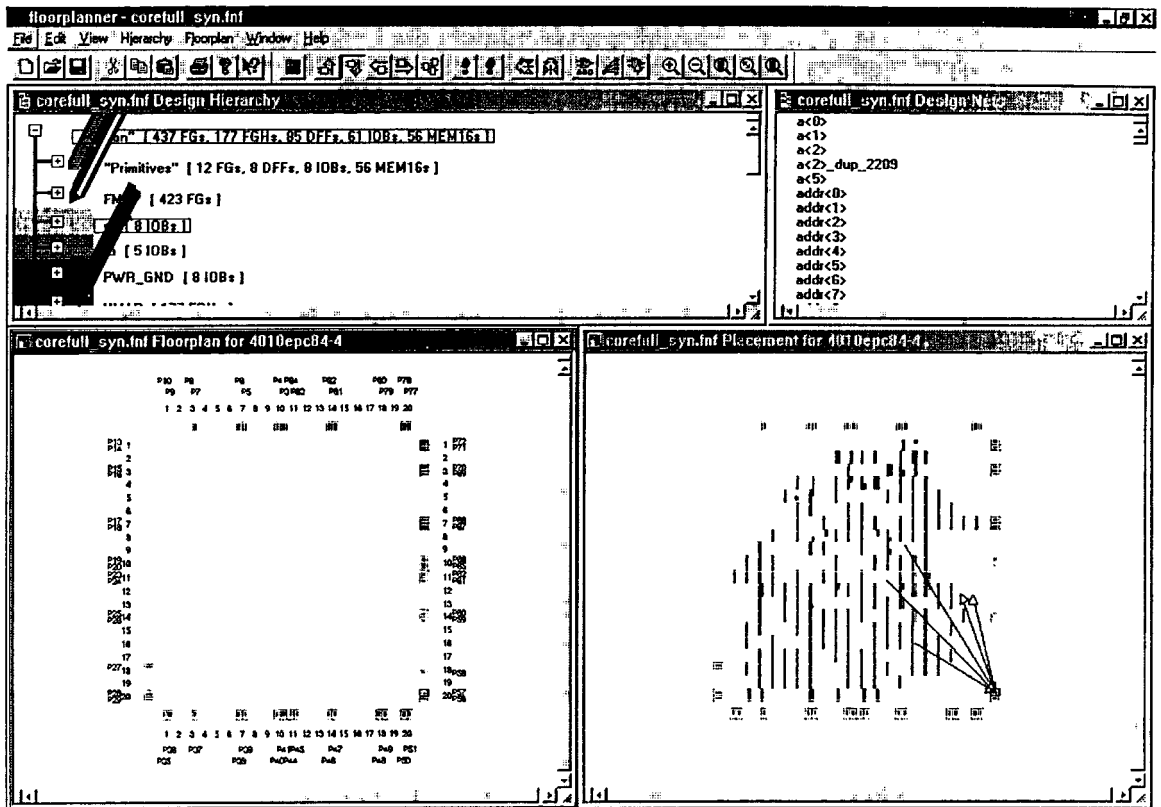
รูปที่ 6.6 แสดงผังวงจรในระดับบนสุดก่อนนำไปเฟลตเมนต์ลงบนเอฟพีจีเอ

ตารางที่ 6.6 แสดงขาไอโอของไมโครคอนโทรลเลอร์กับขาไอโอของเอฟพีจีเอ XC4010PC84-5

Input Pin	Pin in XC4010PC84-5	Output Pin	Pin in XC4010PC84-5
DATA13	P27	ADDR12	P48
DATA12	P28	ADDR11	P49
DATA11	P29	ADDR10	P50
DATA10	P35	ADDR9	P51
DATA9	P36	ADDR8	P56
DATA8	P37	ADDR7	P67
DATA7	P38	ADDR6	P68
DATA6	P39	ADDR5	P69
DATA5	P40	ADDR4	P70
DATA4	P41	ADDR3	P71
DATA3	P44	ADDR2	P72
DATA2	P45	ADDR1	P77
DATA1	P46	ADDR0	P78
DATA0	P47	XOSCENB	P15
MOSCIN	P80	MOSCENB	P13
BODTEB	P23	PWRENB	P17
VDDRISE	P20	In/Out Pin	Pin in XC4010PC84-5
CFG2	P24	RB7	P61
CFG3	P25	RB6	P62
CFG6	P26	RB5	P65
MCLRB	P19	RB4	P66
In/Out Pin	Pin in XC4010PC84-5	RB3	P57
RA4	P7	RB2	P58
RA3	P6	RB1	P59
RA2	P5	RB0/INT	P60
RA1	P4	490Hz *Output	P81
RA0	P3	500KHz *Output	P82

ตารางที่ 6.7 แสดงความหมายของไอโอพอร์ตต่างๆ ในไมโครคอนโทรลเลอร์

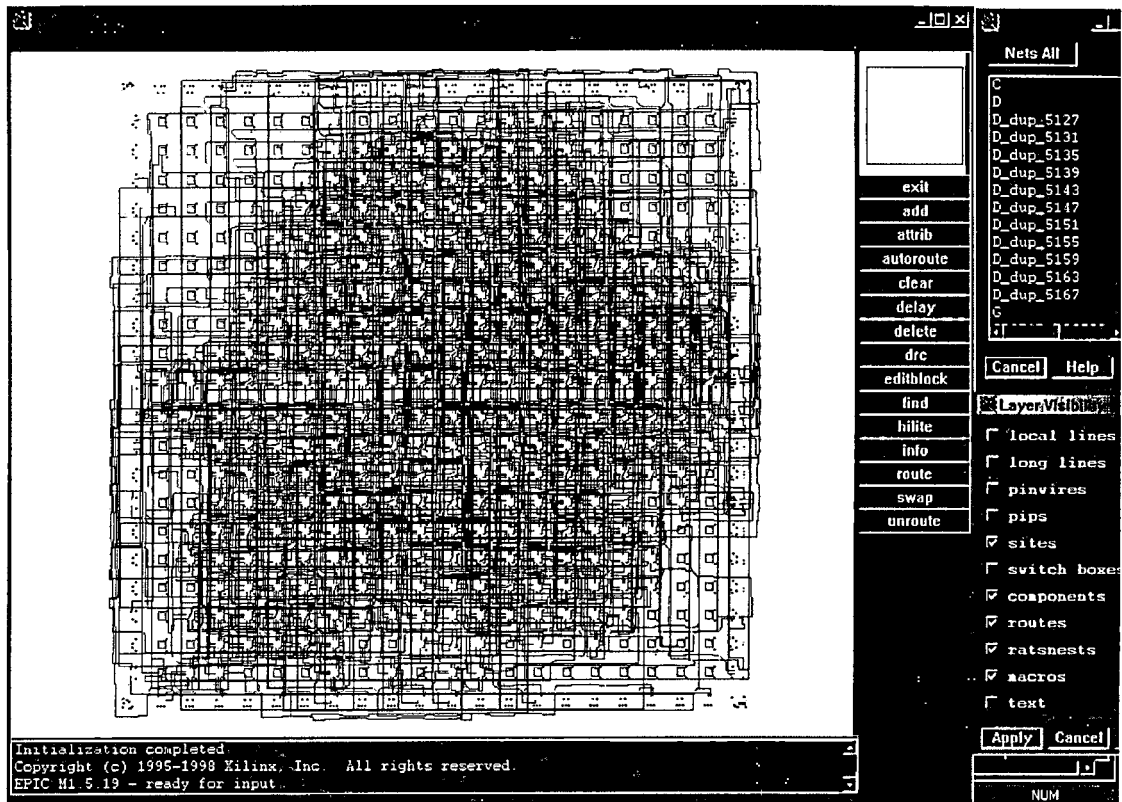
อินพุตพอร์ต		เอาต์พุตพอร์ต	
DATA	เป็นขารับข้อมูลเข้าจากหน่วยความจำโปรแกรมภายนอกมีขนาด 14 บิต	ADDR	เป็นขาส่งค่าแอดเดรสไปยังหน่วยความจำโปรแกรมภายนอกมีขนาด 13 บิต
MOSCIN	เป็นขารับสัญญาณนาฬิกาจากภายนอก	XOSCENB	เป็นขาแสดงสถานะของการทำงานของแหล่งกำเนิดสัญญาณนาฬิกาภายนอก
BODTEB	เป็นขารับสัญญาณจากวงจร Brown-Out Detector ภายนอก	MOSCENB	เป็นขาแสดงสถานะของการทำงานของแหล่งกำเนิดสัญญาณนาฬิกาของระบบ
VDDRISE	เป็นขารับสัญญาณจากวงจร VDD Comparator ภายนอก	PWRENB	เป็นขาแสดงสถานะของการทำงานของวงจร Brown-Out Detector ภายนอก
CFG2	เป็นขาอินพุตเบิ้ลวงจรวอตช์ด็อกไทเมอร์	RA	เป็นขาอินพุตเอาต์พุต 2 ทิศทางทุกขา ขนาด 5 บิต
CFG3	เป็นขาอินพุตเบิ้ลวงจรวอตช์ด็อกไทเมอร์	RB	เป็นขาอินพุตเอาต์พุต 2 ทิศทางทุกขา ขนาด 8 บิต
CFG6	เป็นขาอินพุตเบิ้ลวงจรวอตช์ด็อกไทเมอร์	490Hz	ขากำเนิดสัญญาณนาฬิกาความถี่ 490 เฮิร์ตซ์
MCLR	เป็นขาอินพุตเบิ้ลวงจรวอตช์ด็อกไทเมอร์	500KHz	ขากำเนิดสัญญาณนาฬิกาความถี่ 500 กิโลเฮิร์ตซ์



รูปที่ 6.7 แสดงภาพตำแหน่งของเซลล์ภายในเอฟพีจีเอที่ใช้ในการเฟลตเมนต์ผังวงจรที่ออกแบบ

ตารางที่ 6.8 แสดงค่าการใช้ทรัพยากรภายในอุปกรณ์เอฟพีจีเอในการเฟลตเมนต์และเรดติ้ง

FPGA XC4010PC84-5	Used Max.	Available	Used(%)
Packed CLBs	321	400	80.25%
Bond I/O PINs	52	61	85.25%
F and G Function Generators	643	800	80.38%
H Function Generators	125	400	31.25%
CLB Flip Flops	113	800	14.13%
IOB Input Flip Flops	0	160	0.00%
IOB Output Flip Flops	0	160	0.00%
Memory Write Control	15	400	3.75%



รูปที่ 6.8 แสดงภาพเลย์เอาต์ของผังวงจรภายในอุปกรณ์เอฟพีจีเอที่ถูกเพดเมนต์และเรดต์ตั้ง

บทที่ 7

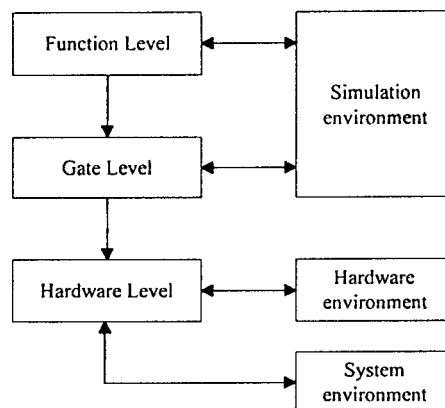
การจำลองและการทดสอบการทำงาน

ในการจำลองและทดสอบความถูกต้องในการทำงานของคอร์ของไมโครคอนโทรลเลอร์ที่ออกแบบจะแบ่งออกเป็น 3 ส่วนใหญ่ๆ คือ

1. การจำลองในระดับฟังก์ชันการทำงาน (Function simulation) เป็นการตรวจสอบความถูกต้องของคอร์ที่ออกแบบจากภาษาวีเอชดีแอลในเรื่องฟังก์ชันการทำงานว่าเป็นไปตามเงื่อนไขของการออกแบบที่วางแผนไว้หรือไม่ โดยทำซิมูเลชันโดยใช้โปรแกรม V-system เพื่อจำลองฟังก์ชันการทำงานทั้งหมดทุกๆ โมดูลภายในคอร์และทำการจำลองคอร์ที่ได้เชื่อมต่อโมดูลต่างๆ ทั้งหมดเข้าด้วยกัน โดยผลซิมูเลชันจะอยู่ในรูปของไทม์มิงไดอะแกรม (Timing diagram) ของสัญญาณภายใน

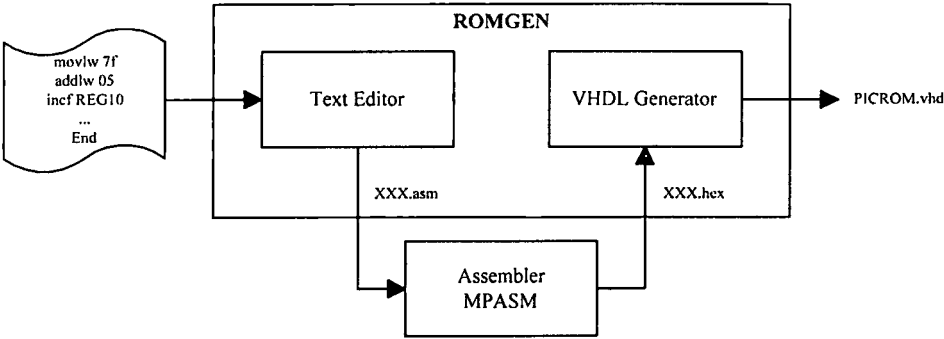
2. การจำลองฐานเวลาการทำงาน (Timing simulation) ในผังวงจรระดับเกต (Gate level) ที่ได้จากการสังเคราะห์จากโปรแกรม Viewsynthesis เป็นการตรวจสอบความถูกต้องของฐานเวลาการทำงานของคอร์ในระดับเกตซึ่งจะมีเรื่องของเวลาหน่วง (Delay time) เข้ามาเกี่ยวข้อง ซึ่งอาจจะทำให้การฟังก์ชันการทำงานผิดพลาดได้ ดังนั้นจะต้องทำซิมูเลชันโดยใช้โปรแกรม ViewSim เพื่อจำลองฐานเวลาการทำงานในผังวงจรที่ถูกสังเคราะห์ ซึ่งผลที่ได้จะนำไปเปรียบเทียบกับผลของการจำลองการทำงานในระดับฟังก์ชันเพื่อใช้วิเคราะห์หาข้อผิดพลาดที่เกิดขึ้น

3. การจำลองในระดับฮาร์ดแวร์ (Hardware simulation) เป็นการทดสอบคอร์ของไมโครโทรลเลอร์ต้นแบบครั้งสุดท้ายซึ่งอยู่ในรูปของอุปกรณ์เอฟพีจีเอกับบอร์ดทดสอบ (Demo board) ที่สร้างขึ้นในการทำงานจริง โดยทำการดาวน์โหลดข้อมูลลงในเอฟพีจีเอโดยผ่านทางสายเอ็กซ์เช็คเคอร์ (XChecker cable) เพื่อดูผลการทำงานในการควบคุมอุปกรณ์ต่างๆ ตามที่โปรแกรมไว้หรือไม่



รูปที่ 7.1 แสดงขั้นตอนการจำลองและทดสอบการทำงาน

โดยการทดสอบความถูกต้องในการทำงานของไมโครคอนโทรลเลอร์ต้นแบบทั้งหมดจะใช้วิธีการสร้างรอม (ROM) ที่ทำหน้าที่เป็นเก็บคำสั่งในการทำงานทั้งหมดเพื่อส่งให้กับคอร์ทำการประมวลผล ดังแสดงในรูปที่ 7.2



รูปที่ 7.2 แสดงขั้นตอนการสร้างรอมเจนที่ใช้ในการทดสอบการทำงานของคอร์

โดยเริ่มจากเขียนรูปแบบคำสั่งของไมโครคอนโทรลเลอร์ตระกูลพีไอซีเบอร์ PIC16C84 ที่อยู่ในรูปไฟล์แอสเซมบลีแล้วทำการแอสเซมเบลอร์โดยใช้โปรแกรมแอสเซมเบลอร์ MPASM ของบริษัทไมโครชิปเทคโนโลยี และจากนั้นทำการแปลงไฟล์ที่ได้จากการแอสเซมเบลอร์ในอยู่ในรูปโมเดลของรอมในภาษาวีเอชดีแอล เพื่อนำไปใช้ทดสอบกับคอร์ต่อไป โดยผลที่ได้จะอยู่ในรูปไทม์มิงไดอะแกรม (timing diagram) ในการทำงานตามโปรแกรมทดสอบตัวอย่างดังต่อไปนี้

โปรแกรมทดสอบที่ 1 (TEST_RELATIVE.ASM) ต้องการทดสอบการทำงานในโหมดการอ้างอิงแอดเดรสแบบสัมพันธ์และคำสั่งที่เกี่ยวข้อง

```
; *****
;
; Demo program display 7 segment
; Test Relative addressing mode
; *****

list P=16C84
```

```
STATUS      equ    0x03      ; STATUS register
PORTB       equ    0x06      ; Output Port B
PC           equ    0x02      ; Program Counter
COUNT      equ    0x12      ; Counter file
PCLATH      equ    0x0a      ; PCH register
RP0         equ    0x05
```

```

W      equ    0
F      equ    1
      org    0x00
      movlw  0x00      ; Initial Port B
      tris   PORTB     ; Port B as Output (Oldinst)

Start   movlw  0x09      ; Count down number
      movwf  COUNT

Loop    movf   COUNT,W   ; Counter contents to W
      call   Segment     ; Call subroutine
      movwf  PORTB       ; Send to Port B
      decfsz COUNT,F     ; Decrements counter Jump if counter = 0
      goto   Loop        ; Counter < 0
      movf   COUNT,W     ; Counter = 0
      call   Segment     ; Call segment subroutine for value 0
      movwf  PORTB
      goto   Start       ; Loop for counter again

Segment addwf  PC,F       ; Add Offset to Program Counter (PC = PC + W)
      retlw  0x3f        ; 0 (For drive 7 segment )
      retlw  0x06        ; 1
      retlw  0x53        ; 2
      retlw  0x4f        ; 3
      retlw  0x66        ; 4
      retlw  0x6d        ; 5
      retlw  0x7d        ; 6
      retlw  0x07        ; 7
      retlw  0x7f        ; 8
      retlw  0x6f        ; 9

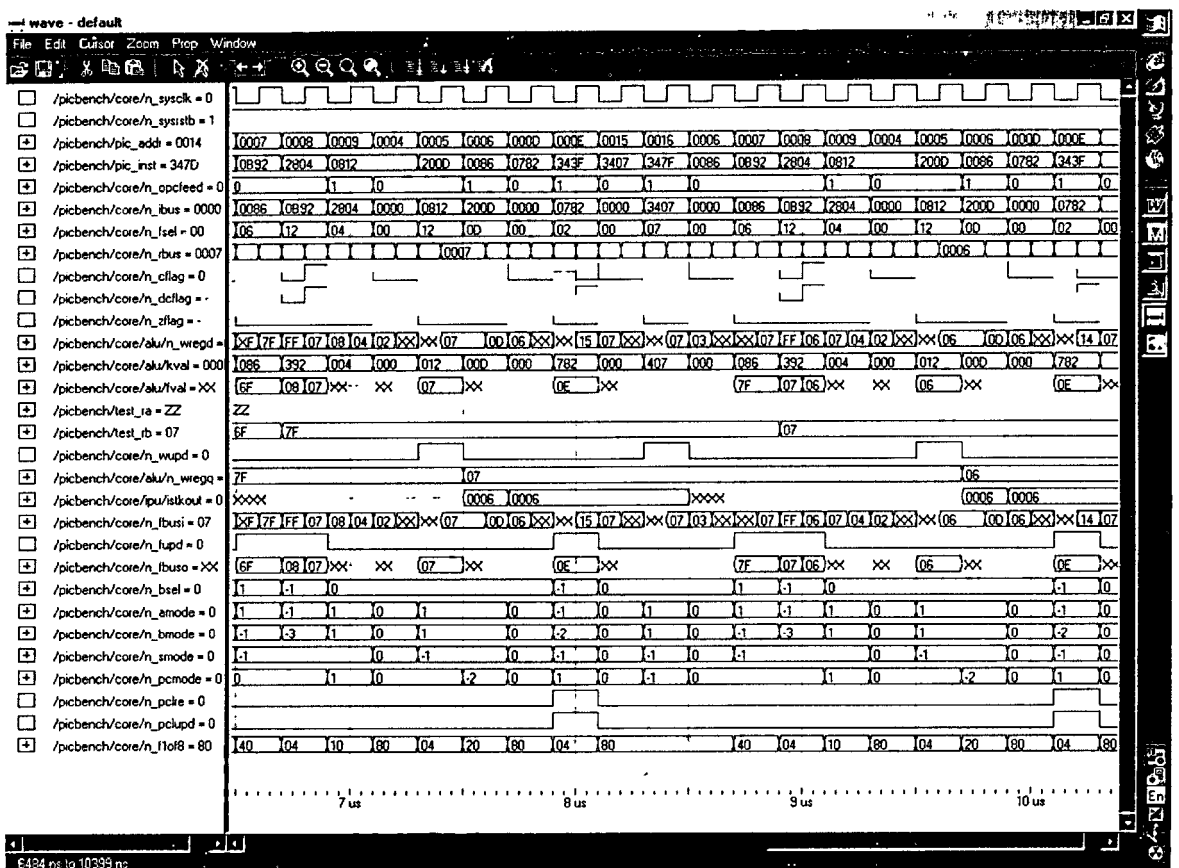
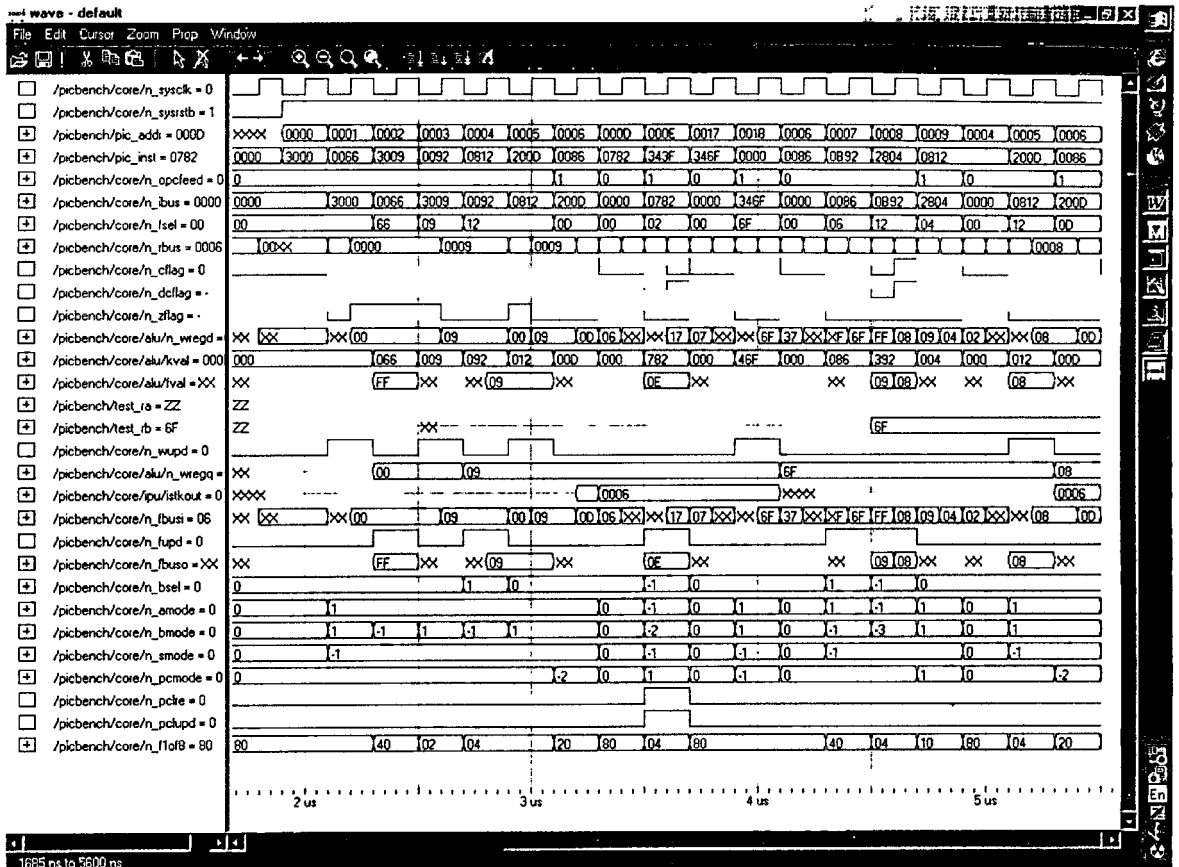
      END

```

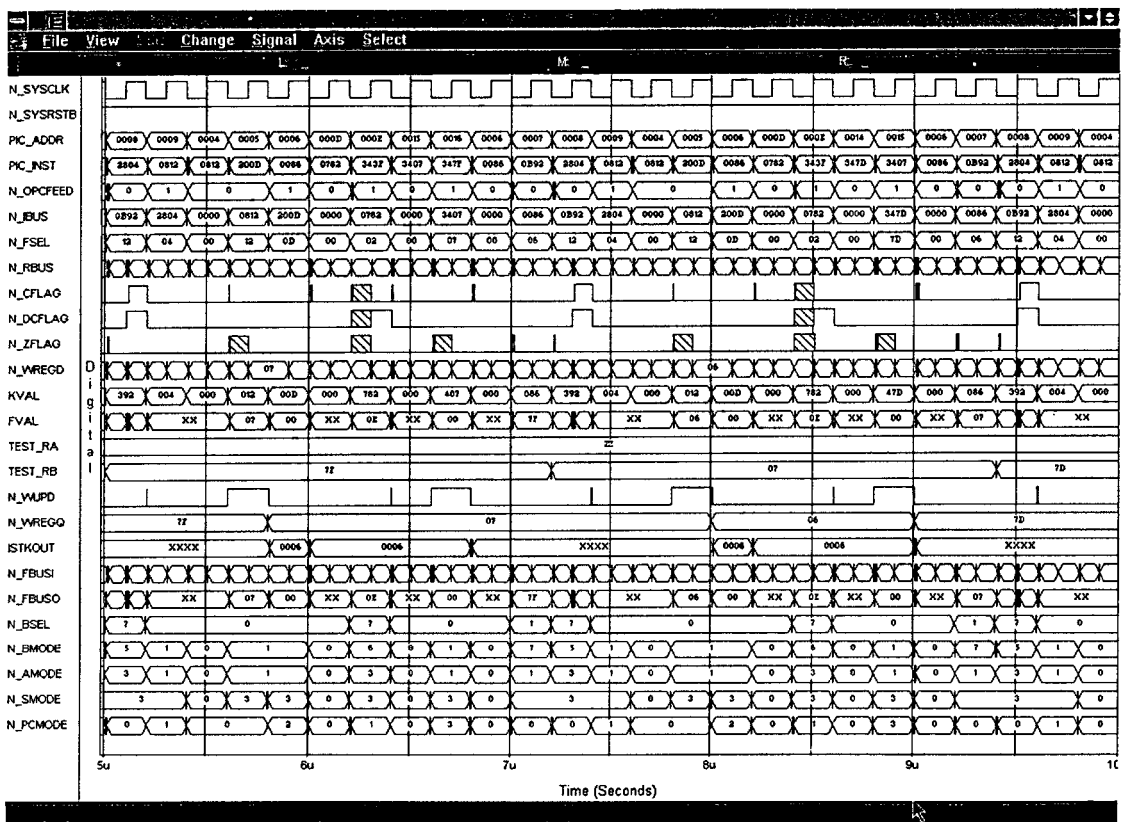
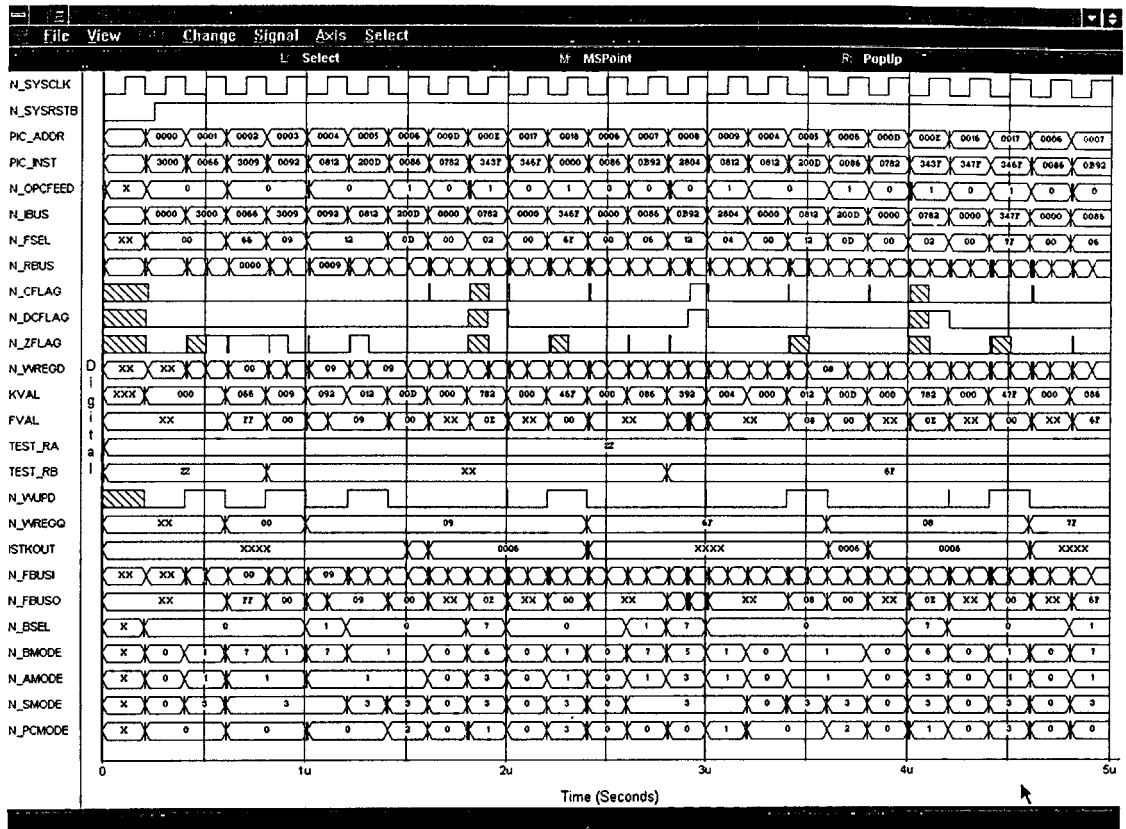
```

;*****

```



รูปที่ 7.3 แสดงไทม์มิงไดอะแกรมของการชุมนุมเล่นกันในระดับฟังก์ชันในโปรแกรมทดสอบที่ 1



รูปที่ 7.4 แสดงไทม์มิงไดอะแกรมของการซิมมูลชันในระดับเกตในโปรแกรมทดสอบที่ 1

โปรแกรมที่ 2 (SHIFT_LED.ASM) ต้องการทดสอบความถูกต้องของบิตทดในเรจิสเตอร์สถานะในการเลื่อนบิตในแบบต่าง ๆ ซึ่งโปรแกรมจะแสดงผลผ่านพอร์ตบีไปยังแอลอีดี

; *****

; Demo program sequencing display LED test Carry flag and Rotate Left Rotate Right

; *****

```

list      P=16C84

STATUS    equ    0x03      ; STATUS register
PORTB     equ    0x06      ; PORTB
SHIFT     equ    0x20      ; File register
W         equ    0
F         equ    1
RP0       equ    5         ; RP0 bit in STATUS
C         equ    0         ; Carry flag in STATUS

org       0x00

Init      movlw  0x00      ; Initial Port B
          tris   PORTB     ; Port B as Output (Oldinst)
          bcf   STATUS,C   ; Clear Carry flag
          clrf  PORTB      ; Clear output port

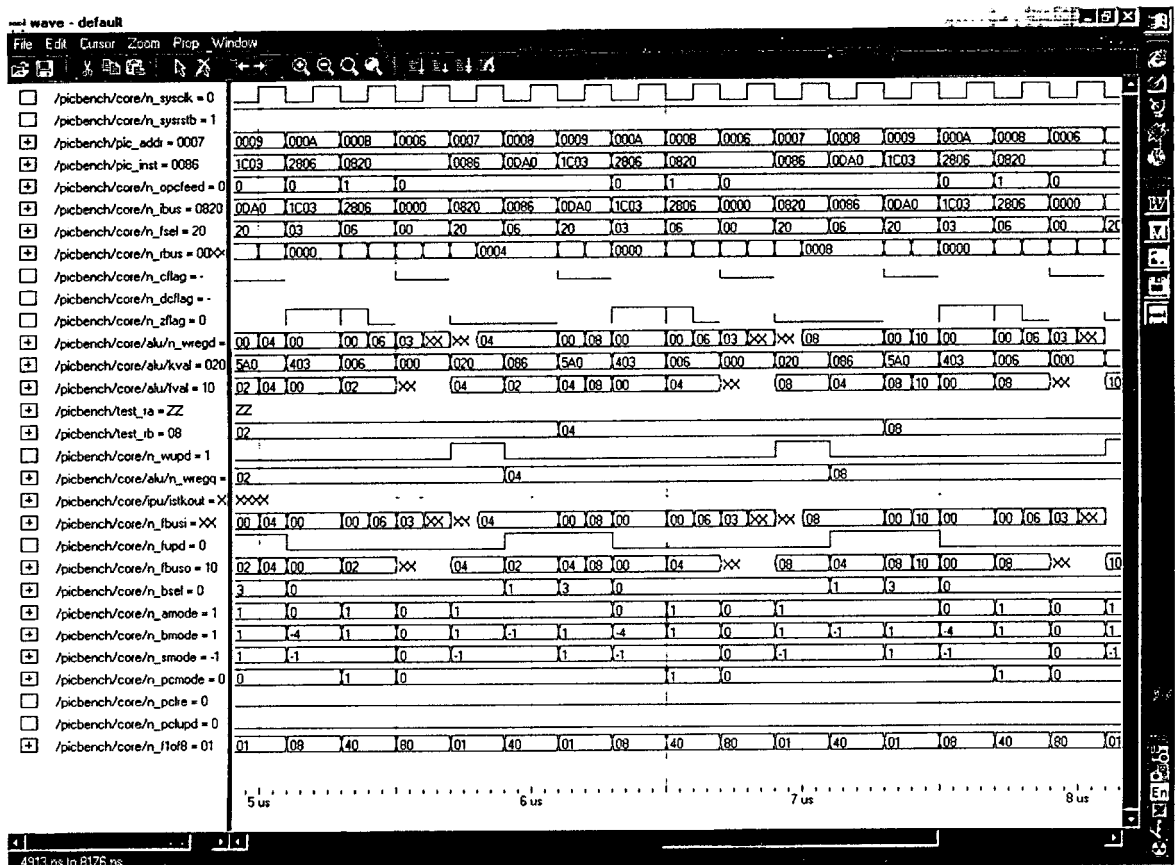
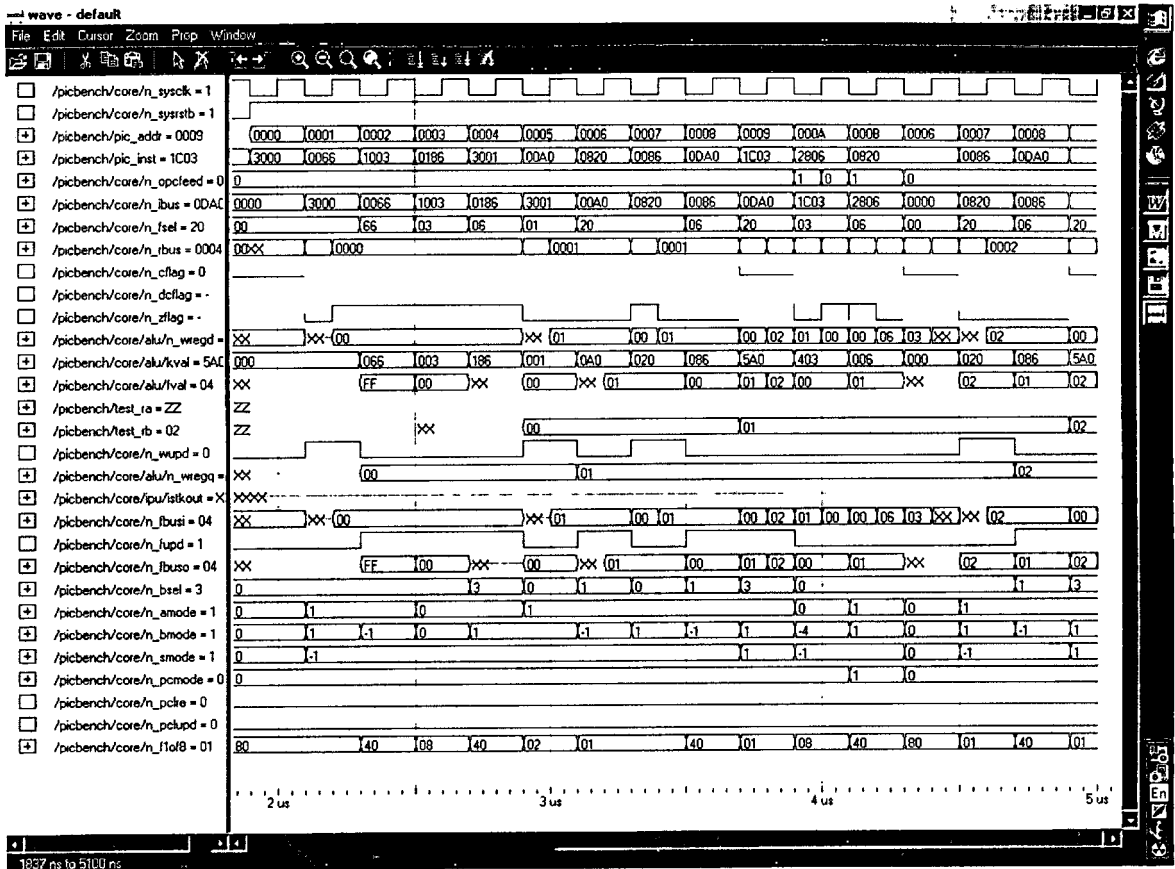
Start     movlw  b'00000001' ; Start with LSB
          movwf SHIFT      ; Store in SHIFT

Left      movf   SHIFT,W
          movwf  PORTB     ; Display
          rlf   SHIFT,F    ; Rotate left
          btfss STATUS,C   ; Carry set ?
          goto  Left       ; Not set, rotate left

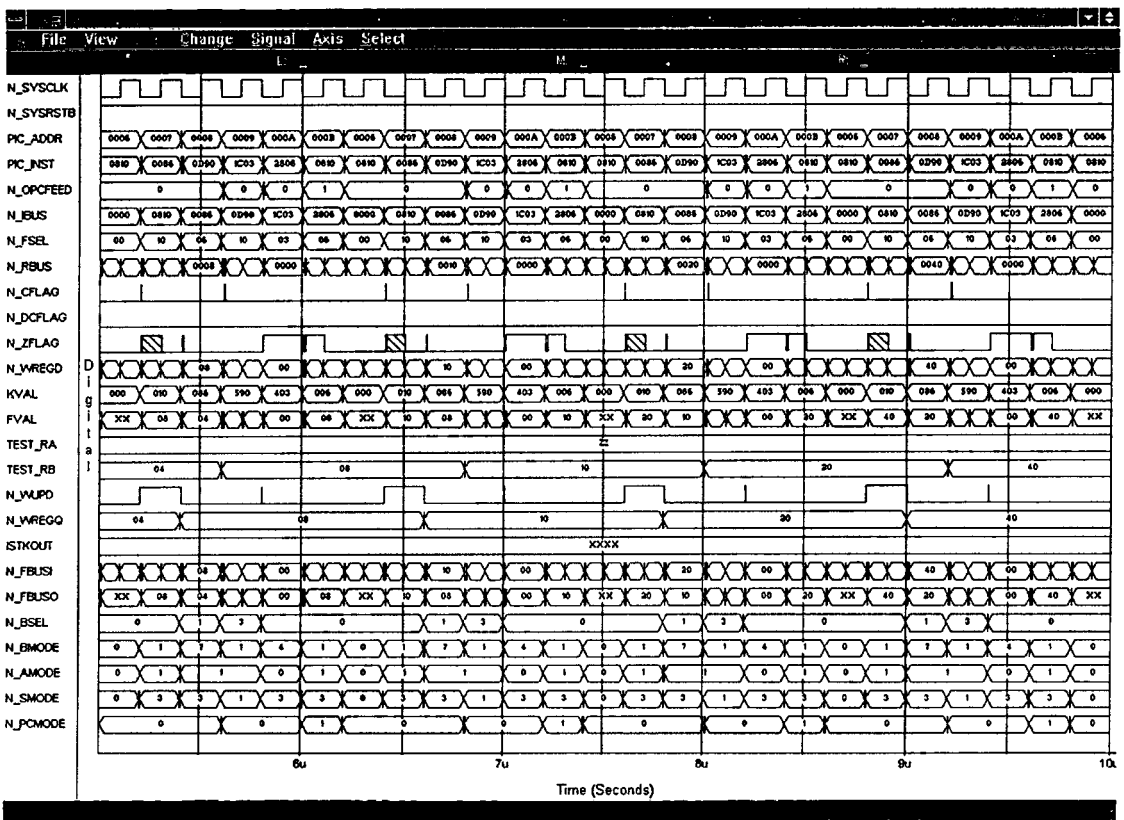
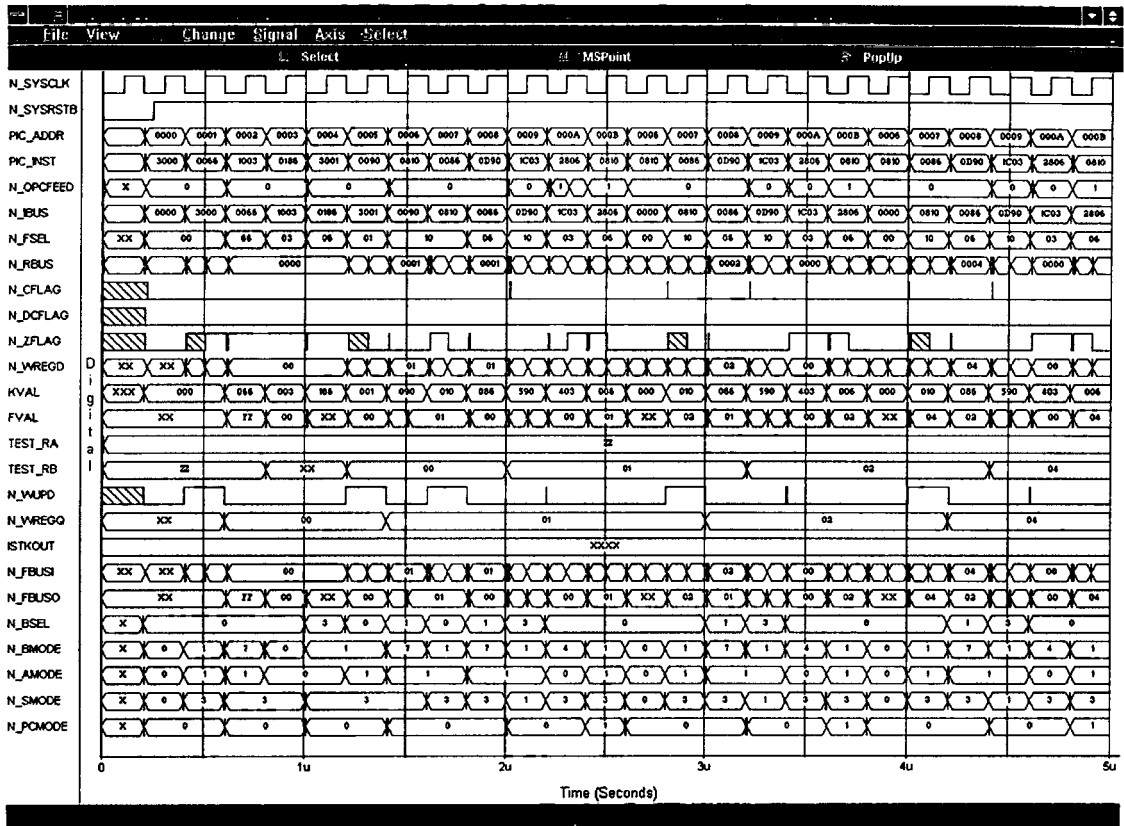
Right     movf   SHIFT,W   ; Carry set, rotate right
          movwf  PORTB
          rrf   SHIFT,F    ; Rotate right
          btfss STATUS,C   ; Carry Set ?
          goto  Right      ; Not set, rotate right
          goto  Left       ; Carry set, rotate left

END

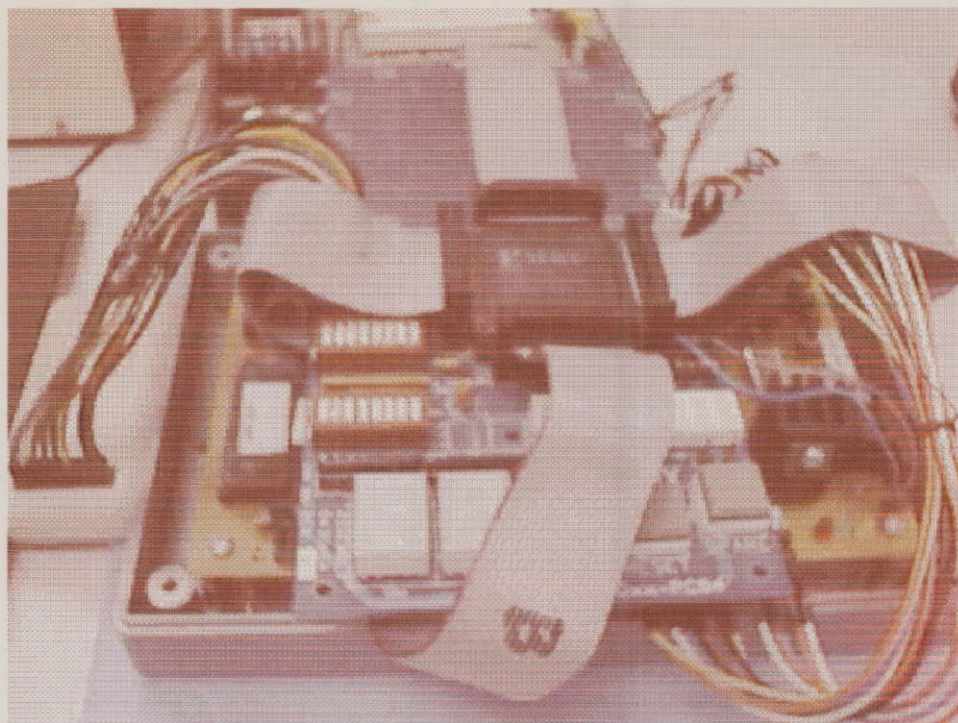
```



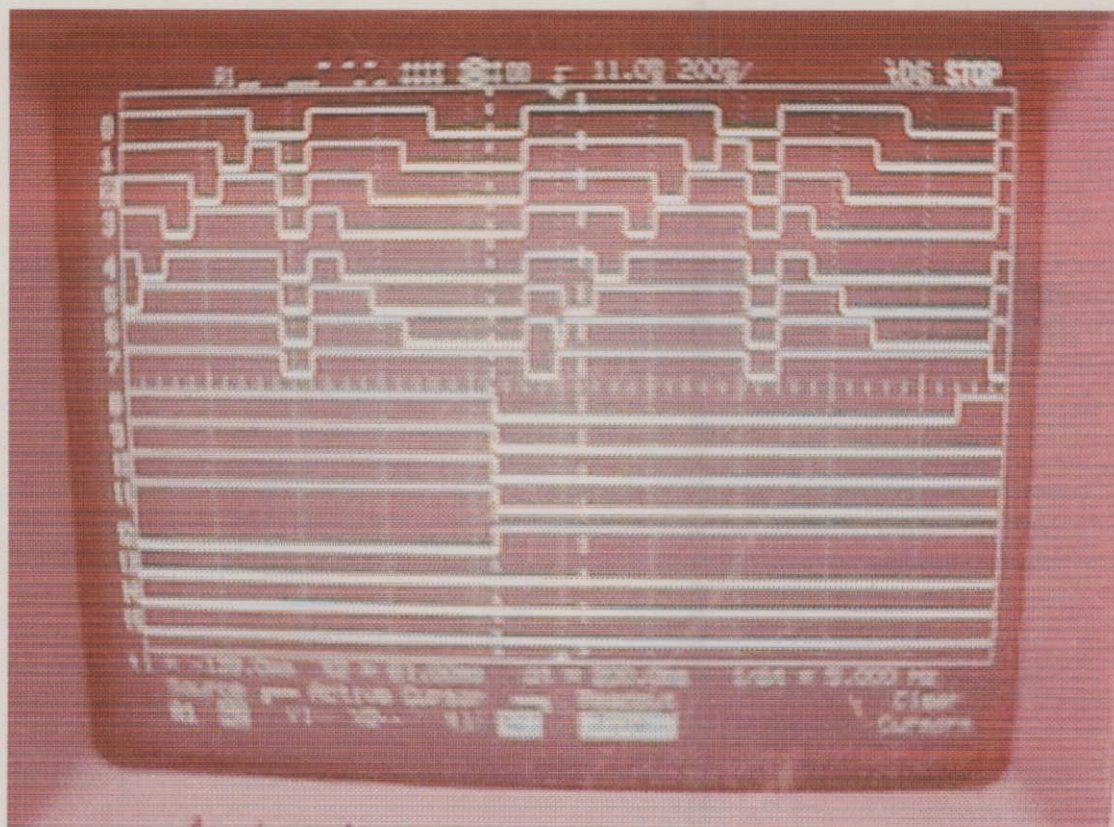
รูปที่ 7.5 แสดงไทม์มิงไคอะแกรมของการซิมมูลชันในระดับฟังก์ชันในโปรแกรมทดสอบที่ 2



รูปที่ 7.6 แสดงไทม์มิงไดอะแกรมของการซิงโครไนซ์ในระดับเกิดในโปรแกรมทดสอบที่ 2



รูปที่ 7.8 แสดงระบบที่ใช้ทดสอบคอร์ของไมโครคอนโทรลเลอร์ในระดับฮาร์ดแวร์



รูปที่ 7.9 แสดงไทม์มิงไดอะแกรมที่ได้จากสโคปในการทำงานของคอร์ในระดับฮาร์ดแวร์

บทที่ 8

สรุปผลการวิจัยและบทวิจารณ์

8.1 วิธีการดำเนินงาน

ในการดำเนินงานวิจัยนี้เป็นการศึกษาการออกแบบและสร้างไมโครคอนโทรลเลอร์ขนาด 8 บิต โดยใช้เอฟพีจีเอโดยได้นำสถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูลพีไอซีเบอร์ PIC16C84 ของบริษัทไมโครชิปเทคโนโลยี มาเป็นสถาปัตยกรรมหลักในการออกแบบสถาปัตยกรรมของไมโครคอนโทรลเลอร์ต้นแบบ โดยอาศัยหลักการออกแบบจากบนลงล่าง ซึ่งเริ่มจากการออกแบบในระดับบล็อกไดอะแกรมและโมดูลต่างๆ ของระบบและเจาะลึกลงไปรายละเอียดของแต่ละโมดูล และจากนั้นนำมาเชื่อมต่อกันจนเป็นระบบที่สมบูรณ์ โดยใช้ภาษาวีเอชดีแอลในการเขียนบรรยายพฤติกรรมของระบบ และใช้โปรแกรม V-system ในการจำลองการทำงานในระดับฟังก์ชัน และทำการสังเคราะห์โดยใช้โปรแกรม Viewsynthesis และจำลองการทำงานในระดับเกตโดยใช้โปรแกรม ViewSim เพื่อตรวจสอบความถูกต้องในการสังเคราะห์ จากนั้นนำไปเฟลตเมนต์และเร้าดติ้ง (Placement & Routing) ลงบนอุปกรณ์เอฟพีจีเอ โดยใช้โปรแกรม XACT 5.2 สุดท้ายจะเป็นขั้นตอนทดสอบไมโครคอนโทรลเลอร์ที่ออกแบบในการทำงานในระดับฮาร์ดแวร์โดยใช้อุปกรณ์เอฟพีจีเอของบริษัทไซริง เบอร์ XC4010PC84-5 ในการทดสอบกับอีอีพรมภายนอกที่ใช้เก็บโปรแกรมคำสั่งให้คอร์ทำงานในรูปแบบต่างๆ

8.2 การออกแบบและทดสอบ

จากการวิจัยพบว่าเราสามารถพัฒนาและสร้างไมโครคอนโทรลเลอร์ที่สามารถทำงานคล้ายกับ PIC16C84 (ยกเว้นในเรื่องของฟังก์ชันการทำงานพิเศษและในส่วน of ระบบ on-chip) โดยความเร็วที่สามารถรันได้ถึงความถี่ 2.948 MHz แบบแยกหน่วยความจำโปรแกรมไว้ภายนอก และผังวงจรที่สังเคราะห์ได้มีขนาดประมาณ 8,000 เกต ใช้พื้นที่ภายใน FPGA เบอร์ XC4010PC84-5 ประมาณ 80% โดยการประมวลผลในเพียง 1 ไมเนอร์คล็อกเท่านั้น ซึ่งแตกต่างจาก PIC16C84 ซึ่งจะต้องใช้ถึง 4 ไมเนอร์คล็อก (Q1-Q4) ทำให้คอร์ที่ออกแบบประมวลได้เร็วยิ่งขึ้น แต่เนื่องจากเราออกแบบโดยใช้เทคโนโลยีของเอฟพีจีเอ (FPGA) จึงทำให้ความเร็วในการประมวลผลไม่สูงมากนักจะนำไปเปรียบเทียบกับ PIC16C84 ที่ทางไมโครชิปเทคโนโลยีผลิตออกมาจำหน่าย เพราะใช้เทคโนโลยีคนละแบบ โดยทาง PIC16C84 ใช้เทคโนโลยีของซีมอส (CMOS) ซึ่งทำให้มีความเร็วสูงในการทำงานเมื่อเทียบกับคอร์ที่ออกแบบโดยใช้เทคโนโลยีเอฟพีจีเอ และจากการวิจัยพบในการ

ออกแบบในลักษณะ Top-down design ซึ่งอาศัยความสามารถของภาษา VHDL ทำให้เราสามารถออกแบบและแก้ไขวงจรและสามารถนำโปรแกรมลงบนอุปกรณ์ FPGA เพื่อทดสอบการทำงานในระดับฮาร์ดแวร์ได้อย่างสะดวกรวดเร็ว

ตารางที่ 8.1 เปรียบเทียบคุณสมบัติของ PIC16C84 กับไมโครคอนโทรลเลอร์ที่ออกแบบ

คุณสมบัติ	PIC16C84	PICCORE	หมายเหตุ
CMOS technology	Yes	No	ใช้ FPGA technology
High Performance RISC-like CPU	Yes	Yes	
36 single word instructions	Yes	Yes	
All single cycle instructions (400 ns) except for program branches which are 2 cycles	Yes	No	ประมาณ 335 ns
4 Minors cycle clocks	Yes	No	Only 1 Minor cycle
Operating Speed: DC - 10 MHz	Yes	No	DC - 2.948 MHz
14-bit wide instructions	Yes	Yes	
8-bit wide data path	Yes	Yes	
1024 x 14 on-chip EPROM program memory	Yes	No	External Chip
36 x 8 general purpose registers (SRAM)	Yes	No	32 x 8 (RAM16X1)
15 special function hardware registers	Yes	No	7 special function reg.
64 x 8 EEPROM data memory	Yes	No	Not implement
8 level deep hardware stack	Yes	Yes	
Direct, indirect and relative addressing modes	Yes	Yes	
Four interrupt sources:	Yes	No	Not test on FPGA
13 I/O pins with individual direction control	Yes	Yes	
8-bit real time clock/counter (RTCC) with 8-bit programmable prescaler	Yes	Yes	
Power-on reset	Yes	Yes	
Power up timer	Yes	Yes	
Oscillator start-up timer	Yes	Yes	
Watchdog timer (WDT) with its own on-chip RC oscillator for reliable operation	Yes	Yes	
Power saving SLEEP mode	Yes	No	

8.3 ปัญหาและอุปสรรคในการดำเนินงาน

เนื่องจากการออกแบบในลักษณะจากบนลงล่างจะเป็นการออกแบบลักษณะ High level Design ซึ่งจะต้องใช้ภาษาที่ใช้เขียนบรรยายพฤติกรรมฮาร์ดแวร์ (HDL) และโปรแกรมช่วยในการออกแบบอัตโนมัติ (EDA) ซึ่งต้องใช้เครื่องคอมพิวเตอร์ที่มีประสิทธิภาพสูงในการประมวลผลที่จะทำการสังเคราะห์ผังวงจรที่เขียนขึ้นจากภาษาวีเอชดีแอล และในการเพลดเมนต์ผังวงจรที่อยู่ในรูปของเน็ตลิสต์ลงในเอฟพีจีเอ ซึ่งทั้งสองขั้นตอนจะต้องใช้ความสามารถของคอมพิวเตอร์ที่มีหน่วยความจำมากและความเร็วสูงในการประมวลผล ซึ่งถ้าคอมพิวเตอร์ที่ใช้มีหน่วยความจำไม่พอจะทำให้เครื่องแฮงค์ (Hank) ได้ จากประสบการณ์ที่ทำวิจัยมา แนะนำว่าควรใช้เครื่องคอมพิวเตอร์ที่มีสเปกอย่างต่ำคือ Pentium II 350 MHz RAM 64 Mb Harddisk 4.3 GB Monitor 17 " ขึ้นไป

8.4 บทวิจารณ์

ในการออกแบบไมโครคอนโทรลเลอร์หรือระบบดิจิทัลอื่นๆ เพื่อการใช้งานอุปกรณ์ประเภทเอฟพีจีเอ นั้นจะมีขั้นตอนที่ไม่แตกต่างจากการออกแบบวงจรดิจิทัลประเภทเอสิก โดยเฉพาะอย่างยิ่ง การใช้ภาษาอธิบายฮาร์ดแวร์ในการออกแบบวงจร หรือที่เรียกว่า เอชดีแอล เช่น ภาษาวีเอชดีแอล (VHDL) หรือ เวิร์ล็อก (Verilog) เป็นต้น ซึ่งสามารถเขียนให้อยู่ในรูปของโค้ดภาษาอธิบายฮาร์ดแวร์ แล้วนำไปผ่านการสังเคราะห์วงจร (HDL Synthesis) โดยที่สามารถเลือกเทคโนโลยีเป้าหมายได้ตามความต้องการ เช่น เลือกเทคโนโลยีเป้าหมายเป็น เซลล์มาตรฐานเอสิก (Standard cell ASIC) หรือ เอฟพีจีเอ (FPGA) เป็นต้น แต่ละต้องเลือกใช้ไลบรารีสำหรับการสังเคราะห์อย่างเหมาะสมด้วย เนื่องจากปัญหาสำคัญส่วนหนึ่งของการออกแบบโดยตรงที่การกำหนดเทคโนโลยีเป้าหมาย ดังนั้นอาจจะมีจุดในโค้ดที่ได้ออกแบบต้องได้รับการเปลี่ยนแปลงแก้ไขก่อนเพื่อให้เหมาะสมและถูกต้องเมื่อต้องการเปลี่ยนจากเทคโนโลยีเป้าหมายหนึ่งไปยังเทคโนโลยีอื่นๆ แต่ก็ทำได้ไม่ยาก ซึ่งไม่เหมือนกับวิธีการออกแบบที่ใช้การวาดวงจร (Schematic Capture) เพราะวิธีหลังนี้ ดีไซน์จะถูกอธิบายให้อยู่ในรูปของสัญลักษณ์ทางกราฟิกที่ประกอบขึ้นจากองค์ประกอบพื้นฐาน (Primitive) และมีอยู่ในไลบรารีสำหรับเทคโนโลยีเป้าหมาย แล้วนำมาต่อเข้าด้วยกันเป็นบล็อกย่อยๆ หรือโมดูล และสามารถนำมาประกอบเข้าเป็นวงจรที่ซับซ้อนอีกได้หลายระดับขึ้นไปจนกลายเป็นผังวงจรที่เสร็จสมบูรณ์ ซึ่งการใช้การวาดวงจรนี้เป็นการอธิบายวงจรในเชิงโครงสร้าง (Structural Description) นั่นเอง ซึ่งมีจุดด้อยของวิธีนี้ก็คือ การเปลี่ยนแปลงแก้ไขการออกแบบทำได้ค่อนข้างยากลำบากและใช้เวลามากและอีกประการที่สำคัญก็คือ วงจรที่ออกแบบขึ้นอยู่กับไลบรารีเป้าหมายที่ได้เลือกใช้ ถ้าจะเปลี่ยนแปลงไลบรารีก็ย่อมทำได้ลำบากโดยเฉพาะอย่างยิ่งผังวงจรที่มีขนาดใหญ่ เพราะจะต้องมาแก้ไขผังวงจรใหม่ทั้งหมด ซึ่งในปัจจุบันแนวโน้มการออกแบบผังวงจรดิจิทัล

ขนาดใหญ่จะหันเหไปทางการใช้งานภาษาอธิบายฮาร์ดแวร์ ซึ่งมีอยู่สองภาษาที่ใช้เป็นมาตรฐานในทางอุตสาหกรรมด้วยกันคือ ภาษาวีเอสดีแอลและภาษาเวรียล็อก ซึ่งทั้งสองภาษานี้มีโปรแกรมหลายๆ ตัวและหลายแพลตฟอร์มที่สามารถทำการคอมไพล์ จำลองการทำงานและสังเคราะห์ผังวงจรที่ได้จากการเขียนอธิบายด้วยภาษาอธิบายฮาร์ดแวร์ ซึ่งทำให้การออกแบบระบบดิจิทัลในปัจจุบันเป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพยิ่งขึ้น

บรรณานุกรม

- [1] กฤษดา ใจเย็น และ ชัยวัฒน์ ลิ้มพรจิตรวิไล. PIC16F84 Theory & Practical Approach บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด พ.ศ. 2541
- [2] Sajjan., G. Shiva. Computer Design & Architecture 2nd ed. New York. HarperCollins Inc. 1991.
- [3] Juan, H.P. et. Al. "Top-Down Modeling of RISC Processors in VHDL." IEEE. 1993. pp. 454-459
- [4] Bhasker, J. VHDL Primer. Prentice-Hall. 1992.
- [5] Gibson, V.E. and Delmar Microprocessors: Fundamental Concepts and Application. University of California. 1993.
- [7] Z. Navabi. VHDL Analysis and Modeling of Digital Systems. McGraw-Hill Inc. 1993
- [8] Viewlogic Inc. VHDL Reference manual. Viewlogic Inc. 1993.
- [9] Viewlogic Inc. ViewDraw Reference manual. Viewlogic Inc. 1993.
- [10] Viewlogic Inc. ViewSim Reference manual. Viewlogic Inc. 1993.
- [11] Viewlogic Inc. ViewTrace Reference manual. Viewlogic Inc. 1993.
- [12] วัชรกร หนูทอง, บรรจง ปิยะธำรง. "การออกแบบไมโครคอนโทรลเลอร์โดยใช้ FPGA." การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 21 วันที่ 12-13 พฤศจิกายน 2541 ณ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ถ. สุขสวัสดิ์ 48 เขตทุ่งครุ กรุงเทพฯ หน้า 178-181
- [12] Ott, D. E. and Wilderotter, T. J. A Designer's Guide to VHDL Synthesis. Klower Academic. 1994.
- [13] Pak, K. Chan and Samiha Mourad. Digital Design using field programmable gate arrays. PTR Prentice Hall . 1994
- [14] Xilinx Inc. The Programmable Logic Data Book. Xilinx Inc. 1994.
- [15] Xilinx Inc. Xilinx XACT Viewlogic Interface user guide. Xilinx Inc. 1994.
- [16] K.C Chang., Digital design and Modeling with VHDL and Synthesis. IEEE Computer Society Pross. California. 1997.
- [17] University of Nottingham. "Getting Started with the PIC16C84." [Online]. Available : <http://www.ccc.nottingham.ac.uk/~cczsteve/pic84.html>. 1999.
- [18] Doulos. "For high quality VHDL and Verilog training for HDL-based FPGA and ASIC design leadership for true expertise in High Level Design." [Online]. Available : <http://www.doulos.co.uk> August. 1999.

ภาคผนวก

ภาคผนวก ก.

Soft Core ของไมโครคอนโทรลเลอร์ที่ออกแบบในรูปภาษาวีเอชดีแอลโดยจะแบ่งออกเป็นโมดูลที่ทำการออกแบบไว้ดังนี้

1. Arithmetic & Logic Module

```

-----
--      Title       : PIC's 8-bits ALU (with W-reg and bit operation)
--      Structure    : Arithmetic & Logic Module (ALM.vhd)
--                   +-- ALU8BIT
--                   +-- ALU2BIT (4 componets)
--                   +-- BITSEL
--                   +-- SHIFTER
--      LIBRARY      : wvlogic
--      Developers   : Mr.Watcharakon Noothong
--      Created date  : 17 Feb 1999
--      version      : 1.00
--      Modify date   :
-----

LIBRARY synth;
USE synth.stdsynth.all;

ENTITY ALM IS
  PORT( clock: IN      vlbit;
        kval:  IN      vlbit_1d(10 DOWNT0 0);
        fval:  IN      vlbit_1d(7 DOWNT0 0);
        pclath: IN     vlbit_1d(4 DOWNT0 0); -- from FSR0A
        ksel:  IN      vlbit; -- from instruction decoder with kval selected
        f3cin: IN      vlbit; -- from status
        idcin: IN      vlbit; -- from instruction decoder
        -- ALU's mode control --
        amode: IN      vlbit_1d(1 DOWNT0 0);
        bmode: IN      vlbit_1d(2 DOWNT0 0);
        smode: IN      vlbit_1d(1 DOWNT0 0);
        bsel:  IN      vlbit_1d(2 DOWNT0 0);
        -- W register control --
        wupd:  IN      vlbit;
        rbus:  OUT     vlbit_1d(12 DOWNT0 0);
        cout:  OUT     vlbit;
        dcout: OUT     vlbit;
        zout:  OUT     vlbit );
END ALM;

ARCHITECTURE mix OF ALM IS

COMPONENT alu8bit
  PORT( a8:  IN      vlbit_1d(7 DOWNT0 0);
        b8:  IN      vlbit_1d(7 DOWNT0 0);
        cin: IN      vlbit;

```

```

    amode: IN    vlbit_1d(1 DOWNT0 0);
    x8:    OUT   vlbit_1d(7 DOWNT0 0);
    cout:   OUT   vlbit;
    dcout:  OUT   vlbit );
END COMPONENT;

COMPONENT bitssel
PORT( din:    IN    vlbit_1d(7 DOWNT0 0);    -- data input (from w-reg)
      bsel:    IN    vlbit_1d(2 DOWNT0 0);    -- bit select (from inst latch)
      bmode:   IN    vlbit_1d(2 DOWNT0 0);    -- mode select (from inst dec)
      dout:    OUT   vlbit_1d(7 DOWNT0 0);    -- data output (to alu)
END COMPONENT;

COMPONENT shifter
PORT(din:    IN    vlbit_1d(7 DOWNT0 0);    -- data input (from alu)
      cin:    IN    vlbit;                  -- carry in (from status reg)
      calu:   IN    vlbit;                  -- carry in (from alu)
      smode:   IN    vlbit_1d(1 DOWNT0 0);    -- mode selection (from inst dec)
      dout:    OUT   vlbit_1d(7 DOWNT0 0);    -- data output (to result reg)
      cout:    OUT   vlbit;                  -- carry output (to status reg)
      zout:    OUT   vlbit );                -- zero out (to status reg)
END COMPONENT;

-- internal connections --
SIGNAL n_rgate:          vlbit;
SIGNAL n_wregd, n_wregq: vlbit_1d(7 DOWNT0 0);
SIGNAL n_ropr, n_rlch:   vlbit_1d(7 DOWNT0 0);
SIGNAL n_b2a:            vlbit_1d(7 DOWNT0 0); -- alu's left operator
SIGNAL n_a2s:            vlbit_1d(7 DOWNT0 0);
SIGNAL n_ca2s:           vlbit;
SIGNAL n_f3cin:          vlbit;

BEGIN
    -- right operator control --
    n_rgate <= '0' WHEN (bmode(1 DOWNT0 0)=B"11") ELSE '1';

    -- select right operator to ALU ( kval bus or fval bus or 0x0000h ) --
    n_ropr <= fval WHEN ((ksel='0') AND (n_rgate='1')) ELSE
              kval(7 DOWNT0 0) WHEN ((ksel='1') AND (n_rgate='1')) ELSE
              B"00000000";

    -- latch right operator during rising edge of clock --
    PROCESS BEGIN
        WAIT UNTIL prising(clock);
        n_rlch <= n_ropr;
        n_f3cin <= f3cin;
    END PROCESS;

    -- instant of bit selector --
    bitselector: bitssel PORT MAP(
        din    => n_wregq,
        bsel    => bsel,
        bmode   => bmode,
        dout    => n_b2a );

    -- instant of 8-bit alu --
    alu8b: alu8bit PORT MAP(
        a8      => n_b2a,

```

```

        b8      => n_rlch,
        cin     => idcin,
        amode   => amode,
        x8      => n_a2s,
        cout    => n_ca2s,
        dcout   => dcout    );

-- instant of shifter --
shifter8bit: shifter PORT MAP(
        din     => n_a2s,
        cin     => n_f3cin,
        calu    => n_ca2s,
        smode   => smode,
        dout    => n_wregd,
        cout    => cout,
        zout    => zout    );

-- latch result to W-reg during falling edge of clock --
PROCESS BEGIN
    WAIT UNTIL pfalling(clock);
    IF (wupd='1') THEN
        n_wregq <= n_wregd;
    END IF;
END PROCESS;

-- upper bit 12..11 of GOTO/CALL/(write PCL) are always PCLATH --
rbus(12 DOWNTO 11) <= pclath(4 DOWNTO 3);

-- select GOTO/CALL address upper 5 bits to PC --

rbus(10 DOWNTO 8)    <=    kval(10 DOWNTO 8) WHEN (ksel='1') ELSE
                        pclath(2 DOWNTO 0);
rbus(7 DOWNTO 0)     <=    n_wregd;

END mix;

ENTITY alu8bit IS
    PORT(
        a8:    IN    vlbit_1d(7 DOWNTO 0);
        b8:    IN    vlbit_1d(7 DOWNTO 0);
        cin:   IN    vlbit;
        amode: IN    vlbit_1d(1 DOWNTO 0);
        x8:    OUT   vlbit_1d(7 DOWNTO 0);
        cout:  OUT   vlbit;
        dcout: OUT   vlbit    );
END ALU8BIT;

ARCHITECTURE sturc OF alu8bit IS

    COMPONENT alu2bit
    PORT(a2:    IN    vlbit_1d(1 DOWNTO 0);
        b2:    IN    vlbit_1d(1 DOWNTO 0);
        cin:   IN    vlbit;
        amode: IN    vlbit_1d(1 DOWNTO 0);
        x2:    OUT   vlbit_1d(1 DOWNTO 0);
        cout:  OUT   vlbit    );
    END COMPONENT;

```

SIGNAL coa, cob, coc, cod : vlbit;

BEGIN

```
alu_u1 : alu2bit PORT MAP(  a2    =>    a8(1 DOWNTO 0),
                             b2    =>    b8(1 DOWNTO 0),
                             cin    =>    cin,
                             amode =>    amode,
                             x2    =>    x8(1 DOWNTO 0),
                             cout   =>    coa    );
```

```
alu_u2 : alu2bit PORT MAP(  a2    =>    a8(3 DOWNTO 2),
                             b2    =>    b8(3 DOWNTO 2),
                             cin    =>    coa,
                             amode =>    amode,
                             x2    =>    x8(3 DOWNTO 2),
                             cout   =>    cob    );
```

```
alu_u3 : alu2bit PORT MAP(  a2    =>    a8(5 DOWNTO 4),
                             b2    =>    b8(5 DOWNTO 4),
                             cin    =>    cob,
                             amode =>    amode,
                             x2    =>    x8(5 DOWNTO 4),
                             cout   =>    coc    );
```

```
alu_u4 : alu2bit PORT MAP(  a2    =>    a8(7 DOWNTO 6),
                             b2    =>    b8(7 DOWNTO 6),
                             cin    =>    coc,
                             amode =>    amode,
                             x2    =>    x8(7 DOWNTO 6),
                             cout   =>    cod    );
```

```
dcout <= cob;
cout  <= cod;
```

END sturc;

ENTITY ALU2BIT IS

```
PORT(
  a2:    IN    vlbit_1d(1 DOWNTO 0);
  b2:    IN    vlbit_1d(1 DOWNTO 0);
  cin:   IN    vlbit;
  amode: IN    vlbit_1d(1 DOWNTO 0);
  x2:    OUT   vlbit_1d(1 DOWNTO 0);
  cout:  OUT   vlbit );
```

END ALU2BIT;

ARCHITECTURE behav of ALU2BIT IS

```
CONSTANT AMODE_AND:    vlbit_1d(1 DOWNTO 0) := B"00";
CONSTANT AMODE_IOR:    vlbit_1d(1 DOWNTO 0) := B"01";
CONSTANT AMODE_XOR:    vlbit_1d(1 DOWNTO 0) := B"10";
CONSTANT AMODE_ADD:    vlbit_1d(1 DOWNTO 0) := B"11";
```

BEGIN

```

PROCESS(a2,b2,cin,amode)
  variable vsum: vlbit_1d(1 DOWNT0 0);
  variable cy: vlbit;
BEGIN
  -- generate 2-bit adder --
  cy := cin;
  FOR i IN 0 TO 1 LOOP
    vsum(i) := (a2(i) XOR b2(i)) XOR cy;
    cy := (a2(i) AND b2(i)) OR (cy AND (a2(i) OR b2(i)));
  END loop;

  IF (amode = AMODE_AND) THEN
    x2 <= a2 AND b2;
    cout <= 'X';
  ELSIF (amode = AMODE_IOR) THEN
    x2 <= a2 OR b2;
    cout <= 'X';
  ELSIF (amode = AMODE_XOR) THEN
    x2 <= a2 XOR b2;
    cout <= 'X';
  ELSE
    x2 <= vsum;
    cout <= cy;
  END IF;
END PROCESS;

```

END behav;

ENTITY bitsel IS

```

PORT(
  din:   IN      vlbit_1d(7 DOWNT0 0);    -- data input (from w-reg)
  bsel:  IN      vlbit_1d(2 DOWNT0 0);    -- bit select (from inst latch)
  bmode: IN      vlbit_1d(2 DOWNT0 0);    -- mode select (from inst dec)
  dout:  OUT     vlbit_1d(7 DOWNT0 0) );  -- data output (to alu)

```

END bitsel;

ARCHITECTURE behav OF bitsel IS

```

-- bmode 000 set selected bit to '0' (all others '1')
-- bmode 001 set all bits to '0'
-- bmode 010 complement
-- bmode 011 --- " --- (set alu's right opr to all '1's)
-- bmode 100 set selected bit to '1' (all others '0')
-- bmode 101 set all bits to '1'
-- bmode 110 pass all bits (pass through mode)
-- bmode 111 --- " --- (set alu's right opr to all '1's)

```

```

CONSTANT BMODE_BIT0:      vlbit_1d(2 DOWNT0 0) := B"000";
CONSTANT BMODE_ALL0:     vlbit_1d(2 DOWNT0 0) := B"001";
CONSTANT BMODE_INV1:     vlbit_1d(2 DOWNT0 0) := B"010";
CONSTANT BMODE_X011:     vlbit_1d(2 DOWNT0 0) := B"011"; -- set ropr to all '1's
CONSTANT BMODE_BIT1:     vlbit_1d(2 DOWNT0 0) := B"100";
CONSTANT BMODE_ALL1:     vlbit_1d(2 DOWNT0 0) := B"101";
CONSTANT BMODE_PASS:     vlbit_1d(2 DOWNT0 0) := B"110";
CONSTANT BMODE_X111:     vlbit_1d(2 DOWNT0 0) := B"111"; -- set ropr to all '1's

```

-- Internal signals connections --

SIGNAL odec: vlbit_1d(7 DOWNT0 0);
SIGNAL bitctl: vlbit_1d(4 DOWNT0 0);

BEGIN

bitctl <= bmode(1 DOWNT0 0) & bsel;

-- decode bit --

odec <= B"00000001" WHEN (bitctl=B"00000") ELSE
B"00000010" WHEN (bitctl=B"00001") ELSE
B"00000100" WHEN (bitctl=B"00010") ELSE
B"00001000" WHEN (bitctl=B"00011") ELSE
B"00010000" WHEN (bitctl=B"00100") ELSE
B"00100000" WHEN (bitctl=B"00101") ELSE
B"01000000" WHEN (bitctl=B"00110") ELSE
B"10000000" WHEN (bitctl=B"00111") ELSE
B"11111111";

-- select ALU's left operator --

dout <= NOT odec WHEN (bmode=BMODE_BIT0) ELSE
NOT odec WHEN (bmode=BMODE_ALL0) ELSE
NOT din WHEN (bmode=BMODE_INV1) ELSE
NOT din WHEN (bmode=BMODE_X011) ELSE
odec WHEN (bmode=BMODE_BIT1) ELSE
odec WHEN (bmode=BMODE_ALL1) ELSE
din WHEN (bmode=BMODE_PASS) ELSE
din;

END behav;

ENTITY SHIFTER IS

PORT(
din: IN vlbit_1d(7 DOWNT0 0); -- data input (from alu)
cin: IN vlbit; -- carry in (from status reg)
calu: IN vlbit; -- carry in (from alu)
smode: IN vlbit_1d(1 DOWNT0 0); -- mode selection (from inst dec)
dout: OUT vlbit_1d(7 DOWNT0 0); -- data output (to result reg)
cout: OUT vlbit; -- carry output (to status reg)
zout: OUT vlbit); -- zero out (to status reg)

END SHIFTER;

ARCHITECTURE behav of SHIFTER IS

-- smode 00 shift right (through carry)
-- smode 01 shift left (through carry)
-- smode 10 swap low with high nibble
-- smode 11 pass through mode

CONSTANT SMODE_SLR: vlbit_1d(1 DOWNT0 0) := "00";
CONSTANT SMODE_SLL: vlbit_1d(1 DOWNT0 0) := "01";
CONSTANT SMODE_SWAP: vlbit_1d(1 DOWNT0 0) := "10";
CONSTANT SMODE_PASS: vlbit_1d(1 DOWNT0 0) := "11";

BEGIN

```
shift_proc : PROCESS(smode,din,cin,calu)
```

```
BEGIN
```

```
IF (smode = SMODE_SLR) THEN
```

```
-- rotate right (through carry) --
```

```
    dout(7) <= cin;
```

```
    dout(6 DOWNT0 0) <= din(7 DOWNT0 1);
```

```
    cout <= din(0);
```

```
    zout <= 'X';
```

```
ELSIF (smode = SMODE_SLL) THEN
```

```
-- rotate left (through carry) --
```

```
    dout(7 DOWNT0 1) <= din(6 DOWNT0 0);
```

```
    dout(0) <= cin;
```

```
    cout <= din(7);
```

```
    zout <= 'X';
```

```
ELSIF (smode = SMODE_SWAP) THEN
```

```
-- swap nibble bits--
```

```
    dout(7 DOWNT0 4) <= din(3 DOWNT0 0);
```

```
    dout(3 DOWNT0 0) <= din(7 DOWNT0 4);
```

```
    cout <= calu;
```

```
    zout <= 'X';
```

```
ELSE
```

```
-- pass through --
```

```
    dout(7 DOWNT0 0) <= din(7 DOWNT0 0);
```

```
    cout <= calu;
```

```
-- test alu's result --
```

```
    IF (din=B"00000000") THEN
```

```
        zout <= '1';
```

```
    ELSE
```

```
        zout <= '0';
```

```
    END IF;
```

```
END IF;
```

```
END PROCESS;
```

```
END behav;
```

2. Instruction register & Instruction Decoder & Control units

```

-- Title      : PIC's Control Units
-- Structure   : Instruction reg, Ins. Decoder & Control (IRIDM.vhd)
--             +-- CLK_GEN
--             +-- RCOUNT10
--             +-- RCOUNT12
--             +-- PICIR
--             +-- PICIDEC
-- LIBRARY     : wvlogic
-- Developers  : Mr. Watcharakon Noothong
-- Created date : 17 Feb 1999
-- version    : 1.00
-- Modify date :

```

```

LIBRARY synth;
USE synth.stdsynth.all;

```

ENTITY IRIDM IS

```

PORT(
    SIGNAL mclrb:      IN    vbit;  -- act.l
    SIGNAL moscin:     IN    vbit;  -- clock from OSC external
    SIGNAL xoscin:     IN    vbit;  -- WDT clock ext.
    SIGNAL bodetb:     IN    vbit;  -- Brown OUT detect ext.
    SIGNAL vddrise:    IN    vbit;  -- Vddrise ext.
    SIGNAL cfg2,cfg3,cfg6: IN    vbit;  -- Configuration bits
    SIGNAL iwakeup:    IN    vbit;  -- act.h from INTCON
    SIGNAL wdrst:      IN    vbit;  -- act.h from WDT
    SIGNAL instinp:    IN    vbit_1d(13 DOWNT0 0); --inst.data ROM
    SIGNAL zflag:      IN    vbit;  -- from ALU
    SIGNAL pclupd:     IN    vbit;  -- from f02 PCL
    SIGNAL execint:    IN    vbit;  -- from INTCON
    -- Control signals --
    SIGNAL amode:      OUT    vbit_1d(1 DOWNT0 0);
    SIGNAL bmode:      OUT    vbit_1d(2 DOWNT0 0);
    SIGNAL smode:      OUT    vbit_1d(1 DOWNT0 0);
    SIGNAL pactive:    OUT    vbit;
    SIGNAL pcmode:     OUT    vbit_1d(1 DOWNT0 0);
    SIGNAL cupd:       OUT    vbit;
    SIGNAL dcupd:      OUT    vbit;
    SIGNAL zupd:       OUT    vbit;
    SIGNAL ksel:       OUT    vbit;  -- to alu
    SIGNAL cfed:       OUT    vbit;
    SIGNAL wupd:       OUT    vbit;
    SIGNAL fupd:       OUT    vbit;
    SIGNAL oldinst:    OUT    vbit;
    SIGNAL clrwtd:     OUT    vbit;
    SIGNAL clrgie:     OUT    vbit;
    SIGNAL retfie:     OUT    vbit;  -- from INTCON reg.
    -- kval signal to ALU --
    SIGNAL kval:       OUT    vbit_1d(10 DOWNT0 0); -- from ALU kval
    SIGNAL moscenb:    OUT    vbit;
    SIGNAL xoscenb:    OUT    vbit;
    SIGNAL pwrenb:     OUT    vbit;

```

```

        SIGNAL sysclk:          OUT  vlbit; -- System Clock
        SIGNAL sysrstb:        OUT  vlbit ); -- System resetb
END IRIDM;

```

ARCHITECTURE struc of IRIDM IS

COMPONENT clk_gen

```

PORT( mclrb:      IN    vlbit; -- act.l
      moscin:     IN    vlbit; -- clock from oscillator
      moscenb:    OUT   vlbit; -- act.h
      xoscin:     IN    vlbit; -- clock from oscillator
      xoscenb:    OUT   vlbit; -- act.h
      vddrise:    IN    vlbit; -- act.h from VDD comparator
      bodetb:     IN    vlbit; -- act.l from Brown-OUT detector
      SLEEP:      IN    vlbit; -- act.h
      iwakeup:    IN    vlbit; -- act.h
      wdtrst:     IN    vlbit; -- act.h
      pwrenb:     OUT   vlbit; -- act.h
      sysclk:     OUT   vlbit; -- System clock output
      sysrstb:    OUT   vlbit; -- System resetb output
      cfg2, cfg3, cfg6: IN  vlbit ); -- configuration bits
END COMPONENT;

```

COMPONENT picir

```

PORT( clock:      IN    vlbit;
      opcfeed:    IN    vlbit_1d(1 DOWNT0 0);
      instinp:    IN    vlbit_1d(13 DOWNT0 0);
      instout:    OUT   vlbit_1d(13 DOWNT0 0) );
END COMPONENT;

```

COMPONENT picidec

```

PORT( clock:      IN    vlbit;
      resetb:     IN    vlbit;
      codein:     IN    vlbit_1d(13 DOWNT0 0);
      zflag:      IN    vlbit;
      pclupd:     IN    vlbit;
      execint:    IN    vlbit;
      amode:      OUT   vlbit_1d(1 DOWNT0 0);
      bmode:      OUT   vlbit_1d(2 DOWNT0 0);
      smode:      OUT   vlbit_1d(1 DOWNT0 0);
      pactive:    OUT   vlbit;
      pcmode:     OUT   vlbit_1d(1 DOWNT0 0);
      opcfeed:    OUT   vlbit_1d(1 DOWNT0 0);
      cupd:       OUT   vlbit;
      dcupd:      OUT   vlbit;
      zupd:       OUT   vlbit;
      ksel:       OUT   vlbit;
      cfed:       OUT   vlbit;
      wupd:       OUT   vlbit;
      fupd:       OUT   vlbit;
      oldinst:    OUT   vlbit;
      clrwtd:     OUT   vlbit;
      sleep:      OUT   vlbit;
      clrgie:     OUT   vlbit;
      retfie:     OUT   vlbit );

```

END COMPONENT;

-- Internal signals connections --

```
SIGNAL n_codein:          vlbitt_1d(13 DOWNT0 0);
SIGNAL n_opcfeed:         vlbitt_1d(1 DOWNT0 0);
SIGNAL n_sleep,n_clk,n_rstb: vlbitt;
```

BEGIN

```
kval    <=    n_codein(10 DOWNT0 0); -- to ALM --
sysclk  <=    n_clk;
sysrstb <=    n_rstb;
```

```
clock_unit: clk_gen PORT MAP(
    mclrb      => mclrb,
    moscin     => moscin,
    moscenb    => moscenb,
    xoscin     => xoscin,
    xoscenb    => xoscenb,
    vddrise    => vddrise,
    bodetb     => bodetb,
    sleep      => n_sleep,
    iwakeup    => iwakeup,
    wdtrst     => wdtrst,
    pwrenb     => pwrenb,
    sysclk     => n_clk,
    sysrstb    => n_rstb,
    cfg2       => cfg2,
    cfg3       => cfg3,
    cfg6       => cfg6
);
```

```

ir: picir PORT MAP(
    clock      =>    n_clk,
    opcfed     =>    n_opcfed,
    instinp    =>    instinp,
    instout    =>    n_codein    );

```

```

idec: picidec PORT MAP(
    clock           =>    n_clk,
    resetb          =>    n_rstb,
    codein          =>    n_codein,
    zflag           =>    zflag,
    pclupd          =>    pclupd,
    execint         =>    execint,
    amode           =>    amode,
    bmode           =>    bmode,
    smode           =>    smode,
    pactive         =>    pactive,
    pcmode          =>    pcmode,
    opcfeed         =>    n_opcfeed,
    cupd            =>    cupd,
    dcupd           =>    dcupd,
    zupd            =>    zupd,
    ksel            =>    ksel,
    cfed            =>    cfed,
    wupd            =>    wupd,
    fupd            =>    fupd,
    oldinst         =>    oldinst,

```

```

                                clrwdt      =>    clrwdt,
                                sleep        =>    n_sleep,
                                clrgie       =>    clrgie,
                                retfie      =>    retfie                );
END struc;

```

```

ENTITY picir IS
    PORT ( clock:      IN    vlbit;
           opcfeed:    IN    vlbit_1d(1 DOWNT0 0);
           instinp:     IN    vlbit_1d(13 DOWNT0 0);
           instout:     OUT   vlbit_1d(13 DOWNT0 0)
         );
END picir;

```

ARCHITECTURE behav OF picir IS

```

CONSTANT OPC_NOP:      vlbit_1d(13 DOWNT0 0) := B"0000000000000000";
CONSTANT OPC_J00:      vlbit_1d(13 DOWNT0 0) := B"1010000000000000";
CONSTANT OPC_C04:      vlbit_1d(13 DOWNT0 0) := B"1000000000001000";

```

BEGIN

-- latch Instruction on falling edge of clock --

```

PROCESS BEGIN
    WAIT UNTIL pfalling(clock);
    IF      (opcfeed=B"00") THEN instout <= instinp;
    ELSIF   (opcfeed=B"01") THEN instout <= OPC_NOP;
    ELSIF   (opcfeed=B"10") THEN instout <= OPC_C04;
    ELSE    instout <= OPC_J00;
    END IF;
END PROCESS;

```

END behav;

```

ENTITY picidec IS
    PORT( clock:      IN    vlbit;
          resetb:     IN    vlbit;
          codein:     IN    vlbit_1d(13 DOWNT0 0);
          zflag:      IN    vlbit;
          pclupd:     IN    vlbit;
          execint:     IN    vlbit;
          amode:      OUT   vlbit_1d(1 DOWNT0 0);
          bmode:      OUT   vlbit_1d(2 DOWNT0 0);
          smode:      OUT   vlbit_1d(1 DOWNT0 0);
          pactive:    OUT   vlbit;
          pcmode:     OUT   vlbit_1d(1 DOWNT0 0);
          opcfeed:    OUT   vlbit_1d(1 DOWNT0 0);
          cupd:       OUT   vlbit;
          dcupd:      OUT   vlbit;
          zupd:       OUT   vlbit;
          ksel:       OUT   vlbit;
          cfed:       OUT   vlbit;
          wupd:       OUT   vlbit;
          fupd:       OUT   vlbit;
          oldinst:    OUT   vlbit;
          clrwdt:     OUT   vlbit;

```

```

        sleep:      OUT  vlbit;
        clrgie:     OUT  vlbit;
        retfie:     OUT  vlbit      );
END picidec;

```

ARCHITECTURE ture_table OF picidec IS

```

CONSTANT PCMODE_RUN:      vlbit_1d(1 DOWNT0 0) := B"00";
CONSTANT PCMODE_LOAD:     vlbit_1d(1 DOWNT0 0) := B"01";
CONSTANT PCMODE_CALL:     vlbit_1d(1 DOWNT0 0) := B"10";
CONSTANT PCMODE_RET:      vlbit_1d(1 DOWNT0 0) := B"11";
CONSTANT XQMODE_NORM:     vlbit_1d(2 DOWNT0 0) := B"000";
CONSTANT XQMODE_GOTO:     vlbit_1d(2 DOWNT0 0) := B"001";
CONSTANT XQMODE_CALL:     vlbit_1d(2 DOWNT0 0) := B"010";
CONSTANT XQMODE_RETN:     vlbit_1d(2 DOWNT0 0) := B"011";
CONSTANT XQMODE_SKZR:     vlbit_1d(2 DOWNT0 0) := B"100";
CONSTANT XQMODE_SKNZ:     vlbit_1d(2 DOWNT0 0) := B"101";
CONSTANT XQMODE_ICAL:     vlbit_1d(2 DOWNT0 0) := B"110";
CONSTANT XQMODE_RTFI:     vlbit_1d(2 DOWNT0 0) := B"111";

```

-- first level Instruction decoder (direct translates) --

```

CONSTANT inst1dec: vlbit_2d(0 to 36,34 DOWNT0 0) := (
--  ++++++----- Opcode Input (ref. PIC16C84)
--  0000000000000000++++----- RETFIE/SLEEP/CLRWDT
--  0000000000000000+------ oldinst
--  000000000000000000++++----- write dest 00:no 01:w 10:f, 11:w/f
--  00000000000000000000++++----- carry feed (14)
--  00000000000000000000++++----- literal (k) (13)
--  0000000000000000000000++++----- xqmode (12 DOWNT0 10)
--  000000000000000000000000++++----- C,DC,Z effected (9 DOWNT0 7)
--  000000000000000000000000++++----- amode (6 DOWNT0 5)
--  000000000000000000000000++++----- bmode (4 DOWNT0 2)
--  000000000000000000000000++++----- smode (1 DOWNT0 0)

```

```

B"000111XXXXXXXXX00001100000111111011", -- 00 OPC_ADDWF (add/pass)
B"000101XXXXXXXXX000011X00000010011011", -- 01 OPC_ANDWF (AND/pass)
B"0000011XXXXXXXXX000010XX0000010000111", -- 02 OPC_CLRF (AND/all0)
B"0000010XXXXXXXXX000001XX0000010000111", -- 03 OPC_CLRW (AND/all0)
B"001001XXXXXXXXX000011X00000011010111", -- 04 OPC_COMF (xor/all1)
B"000011XXXXXXXXX000011000000011110111", -- 05 OPC_DECF (add/-1)
B"001011XXXXXXXXX000011001000001110111", -- 06 OPC_DECFSZ (add/-1)
B"001010XXXXXXXXX000011100000011100111", -- 07 OPC_INCF (add/0/cf)
B"001111XXXXXXXXX000011101000001100111", -- 08 OPC_INCFSZ (add/0/cf)
B"000100XXXXXXXXX000011X00000010111011", -- 09 OPC_IORWF (or/pass)
B"001000XXXXXXXXX000011X00000010100111", -- 10 OPC_MOVF (or/all0)
B"0000001XXXXXXXXX000010XX0000000111111", -- 11 OPC_MOVWF (or/rop0)
B"0000000XX00000000000XX000000XXXXXXX", -- 12 OPC_NOP (no W-reg)
B"001101XXXXXXXXX000011X00001000100101", -- 13 OPC_RLF (or/all0)
B"001100XXXXXXXXX000011X00001000100100", -- 14 OPC_RRF (or/all0)
B"000010XXXXXXXXX000011100001111101011", -- 15 OPC_SUBWF (add/inv/cf)
B"001110XXXXXXXXX000011X00000000100110", -- 16 OPC_SWAPF (or/all0)
B"000110XXXXXXXXX000011X00000011011011", -- 17 OPC_XORWF (xor/pass)

```

```
-- Bit-oriented --
B"0100XXXXXXXXXX000010X00000000000011", -- 18 OPC_BCF (and/bit0)
B"0101XXXXXXXXXX000010X00000000110011", -- 19 OPC_BSF (or/bit1)
B"0110XXXXXXXXXX000000X01000000010011", -- 20 OPC_BTFS (and/bit1)
B"0111XXXXXXXXXX000000X01010000010011", -- 21 OPC_BTFS (and/bit1)
-- Literal and control --
B"11111XXXXXXXXX000001010001111111011", -- 22 OPC_ADDLW (add/pass)
B"111001XXXXXXXXX000001X10000010011011", -- 23 OPC_ANDLW (and/pass)
B"100XXXXXXXXXX000000X010100000100111", -- 24 OPC_CALL (or/all0)
B"00000001100100001000XX000000XXXXXXX", -- 25 OPC_CLRWD (special)
B"101XXXXXXXXXX000000X10010000100111", -- 26 OPC_GOTO (or/all0)
B"111000XXXXXXXXX000001X10000010111011", -- 27 OPC_IORLW (or/pass)
B"1100XXXXXXXXXX000001X10000000100111", -- 28 OPC_MOVLW (or/all0)
B"00000000001001100000XX111000XXXXXXX", -- 29 OPC_RETFIE (special)
B"1101XXXXXXXXXX000001X10110000100111", -- 30 OPC_RETLW (or/all0)
B"00000000001000000000XX011000XXXXXXX", -- 31 OPC_RETURN (no wres)
B"00000001100011010000XX000000XXXXXXX", -- 32 OPC_SLEEP (special)
B"11110XXXXXXXXXX000001110001111101011", -- 33 OPC_SUBLW (add/inv/cf)
B"111010XXXXXXXXX000001X10000011011011", -- 34 OPC_XORLW (xor/pass)
B"00000001100101000110XX000000011111", -- 35 OPC_TRIS A (or/rop0)
B"00000001100110000110XX000000011111", -- 36 OPC_TRIS B (or/rop0) *
);
```

```
-- second level Instruction decoder (On condition translates) --
```

```
CONSTANT inst2dec: vlbit_2d(0 to 10,11 DOWNT0 0) := (
-- ++----- n_intreg, execint (execint will go back to '0' with clrgie)
-- 00+----- pclupd
-- 000+++----- xqmode
-- 000000+----- zflag
-- 0000000+----- pactive -+- next execution after condition met
-- 00000000+----- pcmode -+
-- 0000000000+----- opcfed -+ ->00:INST,01:NOP,10:C04,11:J00
```

```
B"X00000X10000", -- 0 normal
B"010000X00010", -- 1 - Int. duplicate PC and replace fetched Inst with CALL 04
B"110000X10000", -- 2 - Int. response only if performing normal execution
B"XX0001X10101", -- 3 GOTO
B"XX0X10X11001", -- 4 CALL
B"XX0X11X11101", -- 5 RETURN
B"XX0100010000", -- 6 skip/zr (z=0)
B"XX0100110001", -- 7 skip/zr (z=1)
B"XX0101010001", -- 8 skip/nz (z=0)
B"XX0101110000", -- 9 skip/nz (z=1)
B"XX1XXXXX10101" --10 write to PCL GOTO.
);
```

```
-- Internal signals --
```

```
SIGNAL inst1out: vlbit_1d(20 DOWNT0 0);
SIGNAL inst_d: vlbit;
SIGNAL n_intreg: vlbit;
SIGNAL inst2inp: vlbit_1d(6 DOWNT0 0);
SIGNAL inst2out: vlbit_1d(4 DOWNT0 0);
```

```
BEGIN
```

```
inst_d <= codein(7);
```

-- perform first level Instruction decoding --

PLA_TABLE(codein,inst1out,inst1dec);

-- ALU's mode control --

amode <= inst1out(6 DOWNT0 5);

bmode <= inst1out(4 DOWNT0 2);

smode <= inst1out(1 DOWNT0 0);

-- status update control --

cupd <= inst1out(9);

dcupd <= inst1out(8);

zupd <= inst1out(7);

-- literal selector / carry feeder --

ksel <= inst1out(13);

cfed <= inst1out(14);

-- destination control --

wupd <= '1' WHEN ((inst1out(16 DOWNT0 15)=B"01") OR
((inst1out(16 DOWNT0 15)=B"11") AND (inst_d='0')))) ELSE '0';

fupd <= '1' WHEN ((inst1out(16 DOWNT0 15)=B"10") OR
((inst1out(16 DOWNT0 15)=B"11") AND (inst_d='1')))) ELSE '0';

-- special Instruction's control output --

oldinst <= inst1out(17);

clrwtd <= inst1out(18) OR NOT(resetb);

sleep <= inst1out(19);

retfie <= inst1out(20);

-- shift execint to 2nd state --

PROCESS BEGIN

WAIT UNTIL pfalling(clock) OR (resetb='0') OR (execint='0');

IF (resetb='0') THEN

n_intreg <= '0';

ELSIF (execint='0') THEN

n_intreg <= '0';

ELSE

n_intreg <= inst2out(1) AND NOT(inst2out(0));

END IF;

END PROCESS;

-- perform second level Instruction decoding --

inst2inp <= n_intreg & execint & pclupd & inst1out(12 DOWNT0 10) & zflag;

PLA_TABLE(inst2inp,inst2out,inst2dec);

-- next execution control --

clrgie <= n_intreg;

pactive <= inst2out(4);

pcmode <= inst2out(3 DOWNT0 2);

opcfeed <= inst2out(1 DOWNT0 0);

END ture_table;

ENTITY clk_gen IS

PORT(

```

    mclrb:      IN      vlbit; -- act.l
    moscin:     IN      vlbit; -- clock from oscillator
    moscenb:    OUT     vlbit; -- act.h
    xoscin:     IN      vlbit; -- clock from oscillator
    xoscenb:    OUT     vlbit; -- act.h
    vddrise:    IN      vlbit; -- act.h from VDD comparator
    bodetb:     IN      vlbit; -- act.l from Brown-OUT detector
    SLEEP:      IN      vlbit; -- act.h
    iwakeup:    IN      vlbit; -- act.h
    wdtrst:     IN      vlbit; -- act.h
    pwrenb:     OUT     vlbit; -- act.h
    sysclk:     OUT     vlbit; -- clock out
    sysrstb:    OUT     vlbit; -- act.l
    cfg2, cfg3, cfg6: IN  vlbit ); -- configuration bits

```

END clk_gen;

ARCHITECTURE mix OF clk_gen IS

component rcount10

```

    PORT( clk:  IN  vlbit;
          rstb: IN  vlbit;
          ovf:  OUT vlbit );

```

END component;

component rcount12

```

    PORT( clk:  IN  vlbit;
          rstb: IN  vlbit;
          ovf:  OUT vlbit );

```

END component;

SIGNAL cfg_wdte, cfg_bode, cfg_pwrte: vlbit;

SIGNAL n_pwrosc, n_ostclk, n_sclk, n_gclk, n_picwoke, n_picrun, n_mclr: vlbit;

SIGNAL n_por, n_porb, n_wdtrst, n_ffset, n_ffclr, n_chiprst, n_sysrstb: vlbit;

SIGNAL n_sleep, n_kickon, n_pwrtenb, n_pwrtdone, n_pwrtok, n_ostok: vlbit;

BEGIN

```

    cfg_wdte    <=    cfg2;
    cfg_pwrte   <=    not(cfg3);
    cfg_bode    <=    cfg6;
    moscenb     <=    n_picrun;
    n_sclk      <=    moscin;
    xoscenb     <=    cfg_wdte or n_pwrtenb;
    n_pwrosc    <=    xoscin;

```

-- power-on-reset signal coming from VDD-rising and Brown-out detection --

```

    n_por       <= '1' WHEN (vddrise='0') OR (bodetb='0' AND cfg_bode='1') ELSE '0';
    n_porb      <= NOT(n_por);
    n_mclr      <= NOT(mclrb);    -- need schmitt-trigger Inverter
    n_wdtrst    <= wdtrst AND cfg_wdte;

```

-- signals to control reset flip-flop --

```

    n_ffset <= n_mclr OR (n_wdtrst AND NOT(n_picrun)) OR n_por;

```

```

n_ffclr <= NOT(n_ffset) AND n_ostok AND n_pwrtok;

-- flip chip-reset signal --
PROCESS(n_ffset, n_ffclr) BEGIN
    IF (n_ffset='1') THEN
        n_chiprst <= '1';
    ELSIF (n_ffclr='1') THEN
        n_chiprst <= '0';
    END IF;
END PROCESS;

-- delay system reset signal for one system clock --
PROCESS BEGIN
    WAIT UNTIL pfalling(n_gclk) OR (n_ffset='1'); -- Async. n_ffset
    IF (n_ffset='1') THEN
        n_sysrstb <= '0';
    ELSE
        n_sysrstb <= '1';
    END IF;
END PROCESS;

-- kick it up by reset or wakeup --
n_kickon <= n_ffset OR iwakeup;

-- latch sleep state to stop clock and shutdown oscillator --
PROCESS BEGIN
    WAIT UNTIL pfalling(n_gclk) OR (n_kickon='1'); -- Async. n_kickon
    IF (n_kickon='1') THEN
        n_picwoke <= '1';
    ELSE
        n_picwoke <= NOT(SLEEP);
    END IF;
END PROCESS;

n_picrun <= n_picwoke AND n_porb;

-- power-on-reset timer is controlled by --
n_pwrtenb <= n_porb AND cfg_pwrte AND NOT(n_pwrtdone);

pwrtr_ctr: rcount12 PORT MAP(
    clk    => n_pwrosc,
    rstb   => n_porb,
    ovf    => n_pwrtdone
);

n_pwrtok <= n_pwrtdone OR NOT(cfg_pwrte);

-- oscillator start-up timer --
n_ostclk <= n_sclk AND n_pwrtok AND NOT(n_ostok);

ost_ctr: rcount10 PORT MAP(
    clk    => n_ostclk,
    rstb   => n_picrun,
    ovf    => n_ostok
);

```

```
-- gates clock on to run, off to SLEEP --
n_gclk <= n_sclk AND n_picrun AND n_ostok;

pwrenb <= n_picrun;
sysrstb <= n_sysrstb;
sysclk <= n_gclk;
```

```
END mix;
```

3. Program Counter & Stack Module

```
-----
-- Title      : PIC's 13-bits program counter and 8 levels hardware stack
-- Structure   : Program Counter & Stack Module (PCSM.vhd)
--             +-- PICPC
--             +-- PICSTK
--             +-- RAM16x1(13 components)
-- LIBRARY     : wvlogic
-- Developers  : Mr.Watcharakon Noothong
-- Created date : 17 Feb 1999
-- version     : 1.00
-- Modify date :
-----
```

```
LIBRARY synth;
USE synth.stdsynth.all;
```

```
ENTITY PCSM IS
```

```
    PORT( clock:      IN      vlbit;
          resetb:     IN      vlbit;
          pactive:    IN      vlbit;
          pcmode:     IN      vlbit_1d(1 DOWNT0 0);
          addin:      IN      vlbit_1d(12 DOWNT0 0);
          addout:     OUT     vlbit_1d(12 DOWNT0 0) );
```

```
END PCSM;
```

```
ARCHITECTURE struc of PCSM IS
```

```
CONSTANT PCMODE_RUN:      vlbit_1d(1 DOWNT0 0) := B"00";
CONSTANT PCMODE_LOAD:     vlbit_1d(1 DOWNT0 0) := B"01";
CONSTANT PCMODE_CALL:     vlbit_1d(1 DOWNT0 0) := B"10";
CONSTANT PCMODE_RET:      vlbit_1d(1 DOWNT0 0) := B"11";
```

```
COMPONENT picpc
```

```
PORT( pclk:      IN      vlbit;
      pcrset:    IN      vlbit;
      pactive:   IN      vlbit;
      pload:     IN      vlbit;
      pcin:      IN      vlbit_1d(12 DOWNT0 0);
      pcout:     OUT     vlbit_1d(12 DOWNT0 0) );
```

```
END COMPONENT;
```

```
COMPONENT picstk
```

```
PORT( stclk:     IN      vlbit;
      rstb:      IN      vlbit;
```

```

        stmde:      IN      vlbit_1d(1 DOWNT0 0);
        stin:       IN      vlbit_1d(12 DOWNT0 0);
        stout:      OUT     vlbit_1d(12 DOWNT0 0) );
END COMPONENT;

```

-- Internal connections --

```

SIGNAL      ilodpc, imuxsel:      vlbit;
SIGNAL      imuxout, ipcout, istkout: vlbit_1d(12 DOWNT0 0);

```

BEGIN

-- decode some control signal --

```

ilodpc      <=      '1'      WHEN (pcmode=PCMODE_LOAD) ELSE
                  '1'      WHEN (pcmode=PCMODE_CALL) ELSE
                  '1'      WHEN (pcmode=PCMODE_RET)  ELSE
                  '0';
imuxsel      <=      '1'      WHEN (pcmode=PCMODE_RET) ELSE
                  '0';
imuxout      <=      addin    WHEN (imuxsel='0') ELSE
                  istkout;

```

```

pc: picpc PORT MAP(      pcclk =>      clock,
                        pcreset =>      resetb,
                        pactive =>      pactive,
                        pload   =>      ilodpc,
                        pcin    =>      imuxout,
                        pcout   =>      ipcout );

```

```

stack: picstk PORT MAP(      stclk =>      clock,
                        rstb   =>      resetb,
                        stmde  =>      pmode,
                        stin   =>      ipcout,
                        stout  =>      istkout );

```

addout <= ipcout;

END struc;

ENTITY picpc IS

```

    PORT(      pcclk:      IN      vlbit;
              pcreset:     IN      vlbit;  -- active low
              pactive:     IN      vlbit;  -- active high
              pload:       IN      vlbit;  -- active high
              pcin:        IN      vlbit_1d(12 DOWNT0 0);
              pcout:       OUT     vlbit_1d(12 DOWNT0 0) );

```

END picpc;

ARCHITECTURE behav OF picpc IS

```

SIGNAL cps, cns:  vlbit_1d(12 DOWNT0 0);
SIGNAL n_pco:     vlbit_1d(12 DOWNT0 0);

```

BEGIN

cps <= n_pco; *-- feedback path*

```

-- generate Next state for counter --
PROCESS(cps)
    VARIABLE cbit: vlbit;
BEGIN
    cbit := '1';
    FOR i IN 0 TO 12 LOOP
        cns(i) <= cps(i) XOR cbit;
        cbit := cps(i) AND cbit;
    END LOOP;
END PROCESS;

-- advance program counter on falling edge of clock --
PROCESS BEGIN
    WAIT UNTIL pfalling(pcclock);
    IF (pcreset='0') THEN -- synchronous reset
        FOR i IN 0 TO 12 LOOP
            n_pco(i) <= '0';
        END LOOP;
    ELSIF (pcactive='1') THEN
        IF (pload='1') THEN -- rst/Int/jump/CALL/RETURN
            n_pco <= pcin;
        ELSE
            n_pco <= cns;
        END IF;
    END IF;
END PROCESS;

pcout <= n_pco;

END behav;

ENTITY picstk IS
    PORT(
        stclk: IN vlbit;
        rstb: IN vlbit;
        stmode: IN vlbit_1d(1 DOWNTO 0);
        stin: IN vlbit_1d(12 DOWNTO 0);
        stout: OUT vlbit_1d(12 DOWNTO 0)
    );
END picstk;

-- PIC's 8 levels hardware using dual-pointer technic and transparent latch
-- Good for cycle saving (required only 1 clock), but require output multiplexer.

ARCHITECTURE mix OF picstk IS

-- Use RAM16x1 in package XC4000 --
COMPONENT ram16x1
    PORT( d: IN vlbit;
          a0, a1, a2, a3: IN vlbit;
          o: OUT vlbit;
          we: IN vlbit );
END COMPONENT;

CONSTANT STMODE_PUSH: vlbit_1d(1 DOWNTO 0) := B"10";
CONSTANT STMODE_POP: vlbit_1d(1 DOWNTO 0) := B"11";
CONSTANT STMODE_HOLD: vlbit_1d(1 DOWNTO 0) := B"0X";

```

```

CONSTANT dsp3tab: vlbit_2d(0 TO 7, 11 DOWNT0 0) := (
    B"000001000111", -- input/next-up/present/next-down --
    B"001010001000",
    B"010011010001",
    B"011100011010",
    B"100101100011",
    B"101110101100",
    B"110111110101",
    B"111000111110" );

SIGNAL logic_0, stwe:      vlbit;
SIGNAL spst, sp0, sp1, sptr: vlbit_1d(2 DOWNT0 0);
SIGNAL spcmb:             vlbit_1d(8 DOWNT0 0);

BEGIN

    logic_0 <= '0';
    sptr <= sp0 WHEN (stmode=STMODE_PUSH) ELSE sp1;
    stwe <= '1' WHEN ((stmode=STMODE_PUSH) AND (stclk='1')) ELSE '0';

    ram1 : ram16x1 PORT MAP( d      => stin(0),
                             a0      => sptr(0),
                             a1      => sptr(1),
                             a2      => sptr(2),
                             a3      => logic_0,
                             o        => stout(0),
                             we      => stwe      );

    ram2 : ram16x1 PORT MAP( d      => stin(1),
                             a0      => sptr(0),
                             a1      => sptr(1),
                             a2      => sptr(2),
                             a3      => logic_0,
                             o        => stout(1),
                             we      => stwe      );

    ram3 : ram16x1 PORT MAP( d      => stin(2),
                             a0      => sptr(0),
                             a1      => sptr(1),
                             a2      => sptr(2),
                             a3      => logic_0,
                             o        => stout(2),
                             we      => stwe      );

    ram4 : ram16x1 PORT MAP( d      => stin(3),
                             a0      => sptr(0),
                             a1      => sptr(1),
                             a2      => sptr(2),
                             a3      => logic_0,
                             o        => stout(3),
                             we      => stwe      );

    ram5 : ram16x1 PORT MAP( d      => stin(4),
                             a0      => sptr(0),
                             a1      => sptr(1),

```

```

        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(4),
        we    => stwe    );

ram6 : ram16x1 PORT MAP( d    => stin(5),
        a0    => sptr(0),
        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(5),
        we    => stwe    );

ram7 : ram16x1 PORT MAP( d    => stin(6),
        a0    => sptr(0),
        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(6),
        we    => stwe    );

ram8 : ram16x1 PORT MAP( d    => stin(7),
        a0    => sptr(0),
        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(7),
        we    => stwe    );

ram9 : ram16x1 PORT MAP( d    => stin(8),
        a0    => sptr(0),
        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(8),
        we    => stwe    );

ram10 : ram16x1 PORT MAP( d    => stin(9),
        a0    => sptr(0),
        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(9),
        we    => stwe    );

ram11 : ram16x1 PORT MAP( d    => stin(10),
        a0    => sptr(0),
        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o     => stout(10),
        we    => stwe    );

ram12 : ram16x1 PORT MAP( d    => stin(11),
        a0    => sptr(0),

```

```

        a1    => sptr(1),
        a2    => sptr(2),
        a3    => logic_0,
        o      => stout(11),
        we     => stwe      );

ram13 : ram16x1 PORT MAP( d    => stin(12),
                          a0    => sptr(0),
                          a1    => sptr(1),
                          a2    => sptr(2),
                          a3    => logic_0,
                          o      => stout(12),
                          we     => stwe      );

-- process up/down counter on falling edge of clock --
PROCESS BEGIN
    WAIT UNTIL pfalling(stclk) OR (rstb='0');
    IF (rstb='0') THEN
        spst <= "000";
    ELSE
        IF (stmode=STMODE_POP) THEN
            spst <= spcmb(8 DOWNT0 6);
        ELSIF (stmode=STMODE_PUSH) THEN
            spst <= spcmb(2 DOWNT0 0);
        END IF;
    END IF;
END PROCESS;

-- generate counter next-state and output states --

PLA_TABLE(spst,spcmb,dsp3tab);

sp0 <= spcmb(2 DOWNT0 0); -- Push pointer
sp1 <= spcmb(5 DOWNT0 3); -- Pop pointer

END mix;
```

4. File Register Module & Input and Output Port Module

```

-- Title           : File Register Module & Input and Output Port Module
-- Structure       : FRM.vhd
--               +-- PICWDT (Watch-dog Oscillator)
--               +-- RCOUNT10
--               +-- DECODER (Decoder to F00 (of any bank))
--               +-- MUX_FBUS(Multiplexer to fbuso(rbus))
--               +-- MUX_F0X (Multiplexer to f0xout)
--               +-- MUX_F8X (Multiplexer to f8xout)
--               +-- PICTMR0 (fsr01 Timer0 register)
--               +-- FSR03 (status register)
--               +-- FSR04 (FSR Indirect data memory addr. pointer)
--               +-- PORTA (fsr05,TRISA(85h)) * IOPM
--               +-- PORTB (fsr06,TRISB(86h)) * IOPM
--               +-- FSR81 (option register)
--               +-- FSR0A (5-bits pclath register)
--               +-- INTCON (fsr0b) Interrupt control
--               +-- FSR16 (File Registers 10-1F, 20-2F)
--               +-- RAM16x1 ( 8 components)
--
-- LIBRARY        : wvlogic
-- Developers     : Mr.Watcharakon Noothong
-- Created date   : 17 Feb 1999
-- version       : 1.00
-- Modify date    :

```

```

LIBRARY synth;
USE synth.stdsynth.all;

```

```

ENTITY FRM IS

```

```

    PORT(
        SIGNAL fbusi:      IN      vlbit_1d(7 DOWNT0 0); -- rbus(7:0)
        SIGNAL moscin:     IN      vlbit;
        SIGNAL mclrb:      IN      vlbit;
        SIGNAL xoscin:     IN      vlbit;
        SIGNAL cfg2:       IN      vlbit;
        SIGNAL abus:       IN      vlbit_1d(7 DOWNT0 0); --from IPU relative
        SIGNAL zflag,dcflag,cflag: IN vlbit;
        SIGNAL fupd:       IN      vlbit; --from IDEC
        SIGNAL zupd:       IN      vlbit; --from IDEC
        SIGNAL dcupd:      IN      vlbit; --from IDEC
        SIGNAL cupd:       IN      vlbit; --from IDEC
        SIGNAL oldinst:    IN      vlbit; --from IDEC
        SIGNAL ibus:       IN      vlbit_1d(6 DOWNT0 0); --kval(6:0)
        SIGNAL clrwtdt:    IN      vlbit; --from IDEC
        SIGNAL retfie:     IN      vlbit; --from IDEC
        SIGNAL clrgie:     IN      vlbit; --from IDEC
        -- PORT A and B --
        SIGNAL ra_i:       IN      vlbit_1d(4 DOWNT0 0);
        SIGNAL rb_i:       IN      vlbit_1d(7 DOWNT0 0);
        SIGNAL ra_c:       OUT     vlbit_1d(4 DOWNT0 0);
        SIGNAL ra_o:       OUT     vlbit_1d(4 DOWNT0 0);
        SIGNAL rb_c:       OUT     vlbit_1d(7 DOWNT0 0);
    );

```

```

    SIGNAL rb_o:      OUT  vlbit_1d(7 DOWNT0 0);
    -- Output PORT --
    SIGNAL fbuso:      OUT  vlbit_1d(7 DOWNT0 0); --fval(7:0) to ALU
    SIGNAL wdtrst:     OUT  vlbit; --to CLK_GEN
    SIGNAL pclupd:     OUT  vlbit; --to IDEC
    SIGNAL execint:    OUT  vlbit; --to IDEC
    SIGNAL f3cin:      OUT  vlbit;
    SIGNAL pclath:     OUT  vlbit_1d(4 DOWNT0 0);
    SIGNAL iwakeup:    OUT  vlbit  );
END FRM;

```

ARCHITECTURE struc of FRM IS

```

COMPONENT picwdt
PORT( wdtclk:      IN    vlbit;
      wddenb:      IN    vlbit;
      clrwdt:      IN    vlbit;
      wdtrst:      OUT   vlbit  );
END COMPONENT;

COMPONENT decoder
PORT( fsel:        IN    vlbit_1d(6 DOWNT0 0);
      oldinst:     IN    vlbit;
      irp,rp0,rp1: IN    vlbit;
      indf:        IN    vlbit_1d(7 DOWNT0 0);
      f00en,f08en: OUT   vlbit;
      f1xen,f2xen: OUT   vlbit;
      f80en,f88en: OUT   vlbit;
      fl0f8:       OUT   vlbit_1d(7 DOWNT0 0);
      fptr:        OUT   vlbit_1d(8 DOWNT0 0) );
END COMPONENT;

COMPONENT mux_fbus
PORT( f8xout:      IN    vlbit_1d(7 DOWNT0 0);
      f2xout:      IN    vlbit_1d(7 DOWNT0 0);
      f1xout:      IN    vlbit_1d(7 DOWNT0 0);
      add_bus:     IN    vlbit_1d(7 DOWNT0 0);
      f0xout:      IN    vlbit_1d(7 DOWNT0 0);
      f80en,f88en,f2xen,f1xen,pclre: IN vlbit;
      fbuso:       OUT   vlbit_1d(7 DOWNT0 0) );
END COMPONENT;

COMPONENT mux_f0x
PORT( f0xsel:      IN    vlbit_1d(3 DOWNT0 0);
      f01out:      IN    vlbit_1d(7 DOWNT0 0);
      f03out:      IN    vlbit_1d(7 DOWNT0 0);
      f04out:      IN    vlbit_1d(7 DOWNT0 0);
      f05out:      IN    vlbit_1d(4 DOWNT0 0);
      f06out:      IN    vlbit_1d(7 DOWNT0 0);
      f0aout:      IN    vlbit_1d(4 DOWNT0 0);
      f0bout:      IN    vlbit_1d(7 DOWNT0 0);
      f0xout:      OUT   vlbit_1d(7 DOWNT0 0) );
END COMPONENT;

```

```

COMPONENT mux_f8x
PORT( f8xsel:    IN      vlbit_1d(3 DOWNTO 0);
      f81out:    IN      vlbit_1d(7 DOWNTO 0);
      f03out:    IN      vlbit_1d(7 DOWNTO 0);
      f04out:    IN      vlbit_1d(7 DOWNTO 0);
      f85out:    IN      vlbit_1d(4 DOWNTO 0);
      f86out:    IN      vlbit_1d(7 DOWNTO 0);
      f0aout:    IN      vlbit_1d(4 DOWNTO 0);
      f0bout:    IN      vlbit_1d(7 DOWNTO 0);
      f8xout:    OUT     vlbit_1d(7 DOWNTO 0) );
END COMPONENT;

```

```

COMPONENT fsr0a
PORT( d:      IN      vlbit_1d(4 DOWNTO 0);
      q:      OUT     vlbit_1d(4 DOWNTO 0);
      w:      IN      vlbit;      -- active high
      rstb:   IN      vlbit );    -- active low
END COMPONENT;

```

```

COMPONENT fsr81
PORT( d:      IN      vlbit_1d(7 DOWNTO 0);
      q:      OUT     vlbit_1d(7 DOWNTO 0);
      w:      IN      vlbit;      -- active high
      rstb:   IN      vlbit );    -- active low
END COMPONENT;

```

```

COMPONENT fsr04
PORT( d:      IN      vlbit_1d(7 DOWNTO 0);
      q:      OUT     vlbit_1d(7 DOWNTO 0);
      w:      IN      vlbit );    -- active high
END COMPONENT;

```

```

COMPONENT fsr03
PORT( d:      IN      vlbit_1d(7 DOWNTO 0);
      q:      OUT     vlbit_1d(7 DOWNTO 0);
      w:      IN      vlbit_1d(7 DOWNTO 0); -- active high
      rstb:   IN      vlbit );          -- active low
END COMPONENT;

```

```

COMPONENT fsr16
PORT( fin:    IN      vlbit_1d(7 DOWNTO 0);
      fout:   OUT     vlbit_1d(7 DOWNTO 0);
      fsel:   IN      vlbit_1d(3 DOWNTO 0);
      fwe:    IN      vlbit );
END COMPONENT;

```

```

COMPONENT porta
PORT( d:      IN      vlbit_1d(4 DOWNTO 0);
      q05:    OUT     vlbit_1d(4 DOWNTO 0);
      q85:    OUT     vlbit_1d(4 DOWNTO 0);
      pin_i:  IN      vlbit_1d(4 DOWNTO 0);
      pin_o:  OUT     vlbit_1d(4 DOWNTO 0);
      pin_c:  OUT     vlbit_1d(4 DOWNTO 0);
      w05:    IN      vlbit;      -- active high
      w85:    IN      vlbit;      -- active high

```

```

    r05:    IN    vlbit;    -- active high
    rstb:    IN    vlbit    );
END COMPONENT;

```

```

COMPONENT portb
PORT( d:      IN    vlbit_1d(7 DOWNT0 0);
      q06:    OUT   vlbit_1d(7 DOWNT0 0);
      q86:    OUT   vlbit_1d(7 DOWNT0 0);
      pin_i:   IN    vlbit_1d(7 DOWNT0 0);
      pin_o:   OUT   vlbit_1d(7 DOWNT0 0);
      pin_c:   OUT   vlbit_1d(7 DOWNT0 0);
      w06:    IN    vlbit;    -- active high
      w86:    IN    vlbit;    -- active high
      r06:    IN    vlbit;    -- active high
      rbx:    OUT   vlbit;    -- active high
      rstb:    IN    vlbit    );
END COMPONENT;

```

```

COMPONENT pictmr0
PORT(clk:      IN    vlbit;                -- clock
      rstb:    IN    vlbit;                -- active low reset
      d:      IN    vlbit_1d(7 DOWNT0 0);  -- fbusi
      q:      OUT   vlbit_1d(7 DOWNT0 0);  -- 3st to fbuso
      upd:    IN    vlbit;                -- f01 selected
      t0cki:  IN    vlbit;                -- RB0/INT
      t0cs:   IN    vlbit;                -- OPTION.5
      t0se:   IN    vlbit;                -- OPTION.4
      psa:    IN    vlbit;                -- OPTION.3
      psx:    IN    vlbit_1d(2 DOWNT0 0);  -- OPTION.2,1,0
      wdtcki: IN    vlbit;                -- watchdog oscilator in
      wdtcko: OUT   vlbit    );            -- watchdog clock out
END COMPONENT;

```

```

COMPONENT intcon
PORT( clk:      IN    vlbit; -- f.edge
      rstb:      IN    vlbit; -- act.l
      d:         IN    vlbit_1d(7 DOWNT0 0);
      q:         OUT   vlbit_1d(7 DOWNT0 0);
      upd:      IN    vlbit; -- act.h
      rbxint:   IN    vlbit; -- act.h
      tmr0upd:  IN    vlbit; -- act.h
      tmr0msb:  IN    vlbit; -- f.edge
      extint:   IN    vlbit; -- act.h
      intedge:  IN    vlbit;
      iwakeup:  OUT   vlbit;
      clrgie:   IN    vlbit; -- act.h
      RETFIE:   IN    vlbit; -- act.h
      execint:  OUT   vlbit    ); -- act.h
END COMPONENT;

```

-- Internal connections --

```

SIGNAL cfg_wdte, n_wdtrst, n_sleep:      vlbit;
SIGNAL n_pwrenb, n_iwakeup, n_wdtosc:     vlbit;
SIGNAL n_wdtclk, n_execint, n_f3cin:      vlbit;
SIGNAL n_t0cki, n_extint, n_pclre, n_pclupd: vlbit;

```

```

SIGNAL n_fptr:                                vlb1t_1d(8 DOWNT0 0);
SIGNAL n_irp, n_rp0, n_rp1:                   vlb1t;
SIGNAL n_indf:                                vlb1t_1d(7 DOWNT0 0);
SIGNAL n_f00en, n_f08en, n_f1xen, n_f2xen, n_f80en, n_f88en: vlb1t;
SIGNAL n_f1of8:                               vlb1t_1d(7 DOWNT0 0);
SIGNAL n_f01out, n_f03out, n_f04out:          vlb1t_1d(7 DOWNT0 0);
SIGNAL n_f05out, n_f85out:                    vlb1t_1d(4 DOWNT0 0);
SIGNAL n_f06out, n_f86out:                    vlb1t_1d(7 DOWNT0 0);
SIGNAL n_f0aout:                              vlb1t_1d(4 DOWNT0 0);
SIGNAL n_f0bout, n_f81out:                    vlb1t_1d(7 DOWNT0 0);
SIGNAL n_f0xout, n_f1xout, n_f2xout, n_f8xout: vlb1t_1d(7 DOWNT0 0);
SIGNAL n_f05re, n_f06re:                      vlb1t;
SIGNAL n_f03we, n_f04we, n_f05we, n_f06we, n_f0awe: vlb1t;
SIGNAL n_f81we, n_f85we, n_f86we:            vlb1t;
SIGNAL n_f01wg, n_f0bwg, n_f1xwe, n_f2xwe:   vlb1t;
SIGNAL n_f03u, n_f03x:                       vlb1t_1d(7 DOWNT0 0);
SIGNAL n_t0cs, n_t0se, n_psa:                 vlb1t;
SIGNAL n_psx:                                 vlb1t_1d(2 DOWNT0 0);
SIGNAL n_tmr0upd, n_tmr0msb, n_rbxint, n_intedge: vlb1t;

```

BEGIN

-- Map Indirect pointer when access to f00 (of any bank).

-- Decode select file select registers --

```

Indirect: decoder PORT MAP(  fsel    =>    ibus,
                             oldinst =>    oldinst,
                             irp     =>    n_irp,
                             rp0     =>    n_rp0,
                             rp1     =>    n_rp1,
                             indf    =>    n_indf,
                             f00en   =>    n_f00en,
                             f08en   =>    n_f08en,
                             f1xen   =>    n_f1xen,
                             f2xen   =>    n_f2xen,
                             f80en   =>    n_f80en,
                             f88en   =>    n_f88en,
                             f1of8   =>    n_f1of8,
                             fptr    =>    n_fptr      );

n_t0cki    <=    ra_i(4); -- need schmitt-trigger buffer
n_extint   <=    rb_i(0); -- need schmitt-trigger buffer

```

-- f0x00-0x07H Instantiation --

-- f0x00H - INDF is not physiCALLy present --

-- f0x01H - TIMER0 registers

```

n_f01wg    <=    n_f00en AND n_f1of8(1) AND fupd;
n_wdtosc   <=    xoscin;

```

```

f01h: pictmr0 PORT MAP(  clk    =>    moscin,
                         rstb    =>    mclrb,
                         d       =>    fbusi,
                         q       =>    n_f01out,

```

```

                                upd    =>    n_f0lwg,
                                t0cki  =>    n_t0cki,
                                t0cs   =>    n_t0cs,
                                t0se   =>    n_t0se,
                                psa    =>    n_psa,
                                psx    =>    n_psx,
                                wdtcki =>    n_wdtosc,
                                wdtcko =>    n_wdtclk    );

n_tmr0upd    <=    n_f0lwg;
n_tmr0msb    <=    n_f0lout(7);
cfg_wdte     <=    cfg2;

-- Watch-dog timer module --
wdt: picwdt PORT MAP(          wdtclk =>    n_wdtclk,
                                wdenb  =>    cfg_wdte,
                                clrwtd =>    clrwtd,
                                wdtrst =>    wdtrst    );

-- f0x02H - PCL register --
n_pclre      <=    (n_f00en OR n_f80en) AND n_f1of8(2);
pclupd       <=    (n_f00en OR n_f80en) AND n_f1of8(2) AND fupd;

-- f0x03H - STATUS register --

-- status (f03) update control --
n_f03we      <=    (n_f00en OR n_f80en) AND n_f1of8(3) AND fupd AND moscin;
n_f03u(7 DOWNTO 3) <=    n_f03we & n_f03we & n_f03we & n_f03we & n_f03we;
n_f03u(2)    <=    n_f03we OR (zupd AND moscin);
n_f03u(1)    <=    n_f03we OR (dcupd AND moscin);
n_f03u(0)    <=    n_f03we OR (cupd AND moscin);
f3cin        <=    n_f03u(0);

-- status (f03) Input select (flags update has higher priority than write) --
n_f03x(7 DOWNTO 3) <=    fbusi(7 DOWNTO 3);
n_f03x(2)    <=    zflag  WHEN (zupd='1') ELSE fbusi(2);
n_f03x(1)    <=    dcflag WHEN (dcupd='1') ELSE fbusi(1);
n_f03x(0)    <=    cflag  WHEN (cupd='1') ELSE fbusi(0);

f03h: f03 PORT MAP(          d    =>    n_f03x,
                                q    =>    n_f03out,
                                w    =>    n_f03u,
                                rstb =>    mclrb    );

-- Special bit of Status Register --
n_irp        <=    n_f03out(7);
n_rpl        <=    n_f03out(6);
n_rp0        <=    n_f03out(5);
n_f3cin      <=    n_f03out(0);

```

```

-- f0x04H -- FSR register --
n_f04we <= (n_f00en OR n_f80en) AND n_f1of8(4) AND fupd AND moscin;

f04h: fsr04 PORT MAP(      d    => fbusi,
                           q    => n_f04out,
                           w    => n_f04we      );

-- Indirect addressing mode --
n_indf <= n_f04out;

-- f0x05H PORTA register --
n_f05re <= n_f00en AND n_f1of8(5);
n_f05we <= n_f00en AND n_f1of8(5) AND fupd AND moscin;
n_f85we <= n_f80en AND n_f1of8(5) AND fupd AND moscin;

f05h: porta PORT MAP(      d    => fbusi(4 DOWNT0 0),
                           q05  => n_f05out(4 DOWNT0 0),
                           q85  => n_f85out(4 DOWNT0 0),
                           pin_i => ra_i,
                           pin_o => ra_o,
                           pin_c => ra_c,
                           w05  => n_f05we,
                           w85  => n_f85we,
                           r05  => n_f05re,
                           rstb  => mclrb      );

-- f0x06H PORTB register --
n_f06re <= n_f00en AND n_f1of8(6);
n_f06we <= n_f00en AND n_f1of8(6) AND fupd AND moscin;
n_f86we <= n_f80en AND n_f1of8(6) AND fupd AND moscin;

f06h: portb PORT MAP(      d    => fbusi,
                           q06  => n_f06out,
                           q86  => n_f86out,
                           pin_i => rb_i,
                           pin_o => rb_o,
                           pin_c => rb_c,
                           w06  => n_f06we,
                           w86  => n_f86we,
                           r06  => n_f06re,
                           rbx   => n_rbxint,
                           rstb  => mclrb      );

-- f0x0aH - PCLATH has to be 0's when reset --
n_f0awe <= (n_f08en OR n_f88en) AND n_f1of8(2) AND fupd AND moscin;

f0ah: fsr0a PORT MAP(      d    => fbusi(4 DOWNT0 0),
                           q    => n_f0aout(4 DOWNT0 0),
                           w    => n_f0awe,
                           rstb  => mclrb      );

pclath <= n_f0aOUT(4 DOWNT0 0); -- to ALM --

-- f0x0bH INTCON register --
n_f0bwg <= (n_f08en OR n_f88en) AND n_f1of8(3) AND fupd;

```

```
f0bh: intcon PORT MAP(
    clk      =>    moscin,
    rstb     =>    mclrb,
    d        =>    fbusi,
    q        =>    n_f0bout,
    upd      =>    n_f0bwg,
    rbxint   =>    n_rbxint,
    tmr0upd  =>    n_tmr0upd,
    tmr0msb  =>    n_tmr0msb,
    extint    =>    n_extint,
    intedge  =>    n_intedge,
    iwakep   =>    n_iwakep,
    clrgie   =>    clrgie,
    retfie   =>    retfie,
    execint  =>    execint
);
```

```

f80en      =>    n_f80en,
f88en      =>    n_f88en,
f2xen      =>    n_f2xen,
f1xen      =>    n_f1xen,
pclre      =>    n_pclre,
fbuso      =>    fbuso );

-- Multiplex signal file select to f0xout --
mux_f0xout : mux_f0x PORT MAP( f0xsel =>    n_fptr(3 DOWNT0 0),
                                f01out =>    n_f01out,
                                f03out =>    n_f03out,
                                f04out =>    n_f04out,
                                f05out =>    n_f05out,
                                f06out =>    n_f06out,
                                f0aout =>    n_f0aout,
                                f0bout =>    n_f0bout,
                                f0xout =>    n_f0xout );

-- Multiplex signal file select to f8xout --
mux_f8xout : mux_f8x PORT MAP( f8xsel =>    n_fptr(3 DOWNT0 0),
                                f81out =>    n_f81out,
                                f03out =>    n_f03out,
                                f04out =>    n_f04out,
                                f85out =>    n_f85out,
                                f86out =>    n_f86out,
                                f0aout =>    n_f0aout,
                                f0bout =>    n_f0bout,
                                f8xout =>    n_f8xout );

END struc;

ENTITY picwdt IS
  PORT(      wdtclk:      IN      vlbit; -- clock after prescaller
            wdenb:      IN      vlbit; -- 0:disable, 1:enable
            clrwdt:      IN      vlbit; -- act.h from CLRWDT Inst.
            wdtrst:      OUT     vlbit ); -- act.h watchdog timeout reset
END picwdt;

ARCHITECTURE struc OF picwdt IS

  COMPONENT rcount10
    PORT(      clk:      IN      vlbit;
            rstb:      IN      vlbit;
            ovf:      OUT     vlbit );
  END COMPONENT;

  SIGNAL n_wclr, n_wto: vlbit;

  BEGIN
    n_wclr <= not(CLRWDT);

    wdc: rcount10 PORT MAP( clk => wdtclk,
                           rstb => n_wclr,
                           ovf => n_wto );
    wdtrst <= n_wto AND wdenb;

  END struc;

```

ENTITY decoder IS

```

PORT(      fsel:      IN      v1bit_1d(6 DOWNT0 0);
          oldinst:    IN      v1bit;
          irp,rp0,rp1: IN      v1bit;
          indf:       IN      v1bit_1d(7 DOWNT0 0);
          f00en,f08en: OUT     v1bit;
          f1xen,f2xen: OUT     v1bit;
          f80en,f88en: OUT     v1bit;
          f1of8:      OUT     v1bit_1d(7 DOWNT0 0);
          fptr:       OUT     v1bit_1d(8 DOWNT0 0)
);

```

END decoder;

ARCHITECTURE behav OF decoder IS

SIGNAL n_fptr: v1bit_1d(8 DOWNT0 0);

BEGIN

-- Mapping File Pointer register (fptr) --

```

n_fptr <=      B"010000101" when ((oldinst='1') and (fsel(3 downto 0)=B"0101")) else
               B"010000110" when ((oldinst='1') and (fsel(3 downto 0)=B"0110")) else
               B"010000001" when ((oldinst='1') and (fsel(3 downto 0)=B"0010")) else
               irp & indf      when (fsel=B"0000000") else -- Indirect (indf=FSR)
               rp1 & rp0 & fsel; -- Direct addressing (fsel=File select register)

```

-- Decode SFR(Select File Register) or GFR(General File Register) --

```

f00en <= '1' when (n_fptr="X00000XXX") else '0'; -- SFR in Bank 0 with 00-07
f08en <= '1' when (n_fptr="X00001XXX") else '0'; -- SFR in Bank 0 with 08-0f
f80en <= '1' when (n_fptr="X10000XXX") else '0'; -- SFR in Bank 1 with 00-07
f88en <= '1' when (n_fptr="X10001XXX") else '0'; -- SFR in Bank 1 with 08-0f
f1xen <= '1' when (n_fptr="X0001XXXX") else '0'; -- GFR Register at 10-1f
f2xen <= '1' when (n_fptr="X0010XXXX") else '0'; -- GFR Register at 20-2f

```

PROCESS(n_fptr) BEGIN

```

  CASE v1d2int(n_fptr(2 DOWNT0 0)) IS
    WHEN 0  => f1of8 <= B"000000001";
    WHEN 1  => f1of8 <= B"00000010";
    WHEN 2  => f1of8 <= B"00000100";
    WHEN 3  => f1of8 <= B"00001000";
    WHEN 4  => f1of8 <= B"00010000";
    WHEN 5  => f1of8 <= B"00100000";
    WHEN 6  => f1of8 <= B"01000000";
    WHEN OTHERS => f1of8 <= B"10000000";
  END CASE;

```

END PROCESS;

fptr <= n_fptr;

END behav;

```

ENTITY mux_fbIS
  PORT(f8xout:   IN    vlbit_1d(7 DOWNT0 0);
        f2xout:   IN    vlbit_1d(7 DOWNT0 0);
        f1xout:   IN    vlbit_1d(7 DOWNT0 0);
        add_bus:  IN    vlbit_1d(7 DOWNT0 0);
        f0xout:   IN    vlbit_1d(7 DOWNT0 0);
        f80en,f88en,f2xen,f1xen,pclre: IN vlbit;
        fbuso:    OUT   vlbit_1d(7 DOWNT0 0)
  );
END mux_fbIS;

```

```

ARCHITECTURE behav OF mux_fbIS
BEGIN
  fbuso <=  f8xout  WHEN (f80en='1') or (f88en='1') ELSE
            f2xout  WHEN (f2xen='1') ELSE
            f1xout  WHEN (f1xen='1') ELSE
            add_bus WHEN (pclre='1') ELSE
            f0xout;
END behav;

```

```

ENTITY mux_f0IS
  PORT( f0xsel:  IN vlbit_1d(3 DOWNT0 0);
        f01out:  IN vlbit_1d(7 DOWNT0 0);
        f03out:  IN vlbit_1d(7 DOWNT0 0);
        f04out:  IN vlbit_1d(7 DOWNT0 0);
        f05out:  IN vlbit_1d(4 DOWNT0 0);
        f06out:  IN vlbit_1d(7 DOWNT0 0);
        f0aout:  IN vlbit_1d(4 DOWNT0 0);
        f0bout:  IN vlbit_1d(7 DOWNT0 0);
        f0xout:  OUT vlbit_1d(7 DOWNT0 0)
  );
END mux_f0IS;

```

```

ARCHITECTURE behav OF mux_f0IS
BEGIN
  f0xout <=  B"XXXXXXXX"  WHEN (f0xsel=B"0000") ELSE
            f01out        WHEN (f0xsel=B"0001") ELSE
            B"XXXXXXXX"  WHEN (f0xsel=B"0010") ELSE
            f03out        WHEN (f0xsel=B"0011") ELSE
            f04out        WHEN (f0xsel=B"0100") ELSE
            B"XXX" & f05out WHEN (f0xsel=B"0101") ELSE
            f06out        WHEN (f0xsel=B"0110") ELSE
            B"XXXXXXXX"  WHEN (f0xsel=B"0111") ELSE
            B"XXXXXXXX"  WHEN (f0xsel=B"1000") ELSE
            B"XXXXXXXX"  WHEN (f0xsel=B"1001") ELSE
            B"XXX" & f0aout WHEN (f0xsel=B"1010") ELSE
            f0bout        WHEN (f0xsel=B"1011") ELSE
            B"XXXXXXXX";
END behav;

```

ENTITY mux_f8x IS

```
PORT( f8xsel: IN vlbit_1d(3 DOWNT0 0);
      f81out: IN vlbit_1d(7 DOWNT0 0);
      f03out: IN vlbit_1d(7 DOWNT0 0);
      f04out: IN vlbit_1d(7 DOWNT0 0);
      f85out: IN vlbit_1d(4 DOWNT0 0);
      f86out: IN vlbit_1d(7 DOWNT0 0);
      f0aout: IN vlbit_1d(4 DOWNT0 0);
      f0bout: IN vlbit_1d(7 DOWNT0 0);
      f8xout: OUT vlbit_1d(7 DOWNT0 0)
    );
```

END mux_f8x;

ARCHITECTURE behav OF mux_f8x IS

BEGIN

```
f8xout <=  B"XXXXXXXX"      WHEN (f8xsel=B"0000") ELSE
           f81out          WHEN (f8xsel=B"0001") ELSE
           B"XXXXXXXX"     WHEN (f8xsel=B"0010") ELSE
           f03out          WHEN (f8xsel=B"0011") ELSE
           f04out          WHEN (f8xsel=B"0100") ELSE
           B"XXX" & f85out  WHEN (f8xsel=B"0101") ELSE
           f86out          WHEN (f8xsel=B"0110") ELSE
           B"XXXXXXXX"     WHEN (f8xsel=B"0111") ELSE
           B"XXXXXXXX"     WHEN (f8xsel=B"1000") ELSE
           B"XXXXXXXX"     WHEN (f8xsel=B"1001") ELSE
           B"XXX" & f0aout  WHEN (f8xsel=B"1010") ELSE
           f0bout          WHEN (f8xsel=B"1011") ELSE
           B"XXXXXXXX";
```

END behav;

ENTITY pictmr0 IS

```
PORT(clk:      IN      vlbit;                -- clock
      rstb:    IN      vlbit;                -- active low reset
      d:       IN      vlbit_1d(7 DOWNT00);  -- fbusi
      q:       OUT     vlbit_1d(7 DOWNT00);    -- 3st to fbuso
      upd:     IN      vlbit;                -- f01 selected
      t0cki:   IN      vlbit;                -- RB0/INT
      t0cs:    IN      vlbit;                -- OPTION.5
      t0se:    IN      vlbit;                -- OPTION.4
      psa:     IN      vlbit;                -- OPTION.3
      psx:     IN      vlbit_1d(2 DOWNT00);   -- OPTION.2,1,0
      wdcki:   IN      vlbit;                -- watchdog oscillator IN
      wdtko:   OUT     vlbit;                -- watchdog clock OUT
    );
```

END pictmr0;

ARCHITECTURE behav OF pictmr0 IS

```
SIGNAL n_psin, n_psout, n_t0cks:  vlbit;
SIGNAL pscaler, pscale_ns:        vlbit_1d(7 DOWNT0 0);
SIGNAL ctck, n_hold0, n_hold1:    vlbit;
SIGNAL n_ctck, n_ctck0, n_ctck1, n_t0cki: vlbit;
SIGNAL n_to, toclr:               vlbit;
SIGNAL tmr0_q, tmr0_ns, r, s:     vlbit_1d(7 DOWNT0 0);
```

BEGIN

-- generate counter hold signal --

PROCESS BEGIN

WAIT UNTIL pfalling(clk) OR (rstb='0'); -- Async. reset

IF (rstb='0') THEN

n_hold0 <= '0';

ELSE

n_hold0 <= upd;

END IF;

END PROCESS;

n_hold1 <= upd OR n_hold0;

-- Synchronize external clock --

PROCESS BEGIN

WAIT UNTIL pfalling(clk);

n_ctck1 <= n_ctck0;

n_ctck0 <= t0cki XOR t0se;

END PROCESS;

n_t0cki <= '0' WHEN (n_ctck0='1' AND n_ctck1='0') ELSE '1';

-- select timer0 clock --

n_t0cks <= clk WHEN (t0cs='0') ELSE n_t0cki;

-- select Input to pre-sCALLer --

n_psin <= n_t0cks WHEN (psa='0') ELSE wdtcki;

-- pre-scaler's nextstate --

pscale_feedback: PROCESS(pscaler)

VARIABLE cbit: vlbit;

BEGIN

cbit := '1';

FOR i IN 0 to 7 LOOP

pscale_ns(i) <= pscaler(i) XOR cbit;

cbit := cbit AND pscaler(i);

END LOOP;

END PROCESS;

-- pre-scaler's counter --

pscale_register: PROCESS BEGIN

WAIT UNTIL pfalling(n_psin);

pscaler <= pscale_ns;

END PROCESS;

-- select scale factor from pre-sCALLer --

WITH v1d2int(psx(2 DOWNT0 0)) SELECT

n_psout <= pscaler(0) WHEN 0,
pscaler(1) WHEN 1,
pscaler(2) WHEN 2,
pscaler(3) WHEN 3,
pscaler(4) WHEN 4,
pscaler(5) WHEN 5,
pscaler(6) WHEN 6,
pscaler(7) WHEN OTHERS;

```

-- select clock output for watchdog --
    wdtcko <=      wdtcki WHEN (psa='0') ELSE
                  n_psout;

-- select clock input for timer counter --
    ctck <= clk    WHEN (upd='1') ELSE
    n_psin  WHEN (psa='1') ELSE
    n_psout;

-- timer-0's next state --
tmr0_feedback: PROCESS(upd, n_hold1, d, tmr0_q)
    VARIABLE cbit: vlbit;
    BEGIN
        cbit := '1';
        for i IN 0 TO 7 LOOP
            IF (upd='1') THEN                                -- preload
                tmr0_ns(i) <= d(i);
            ELSIF (n_hold1='1') THEN                          -- hold
                tmr0_ns(i) <= tmr0_q(i);
            ELSE
                tmr0_ns(i) <= tmr0_q(i) XOR cbit; -- countup
                cbit := cbit AND tmr0_q(i);
            END IF;
        END LOOP;
    END PROCESS;

-- timer-0's counter --
tmr0_register: PROCESS BEGIN
    WAIT UNTIL pfalling(ctck) OR (rstb='0'); -- async. reset
    IF (rstb='0') THEN
        tmr0_q <= "00000000";
    ELSE
        tmr0_q <= tmr0_ns;
    END IF;
END PROCESS;
q <= tmr0_q;

END behav;

ENTITY fsr03 IS
PORT( d:    IN    vlbit_1d(7 DOWNT0 0);
      q:    OUT   vlbit_1d(7 DOWNT0 0);
      w:    IN    vlbit_1d(7 DOWNT0 0); -- active high
      rstb: IN    vlbit   );
END fsr03;

ARCHITECTURE behav OF fsr03 IS

SIGNAL iq: vlbit_1d(7 DOWNT0 0);

BEGIN

    PROCESS(w,rstb,d)

```

```

        BEGIN
            FOR i IN 7 DOWNTO 0 LOOP
                IF (rstb='0') THEN
                    iq(i) <= '0';
                ELSIF (w(i)='1') THEN
                    iq(i) <= d(i);
                END IF;
            END loop;
        END PROCESS;

    q <= iq;

END behav;

ENTITY fsr04 IS
PORT( d:      IN      vlbit_1d(7 DOWNTO 0);
      q:      OUT     vlbit_1d(7 DOWNTO 0);
      w:      IN      vlbit    );      -- act.h
END fsr04;

ARCHITECTURE behav OF fsr04 IS

SIGNAL iq: vlbit_1d(7 DOWNTO 0);

BEGIN
    PROCESS(w,d)
        BEGIN
            IF (w='1') THEN
                iq <= d;
            END IF;
        END PROCESS;

    q <= iq;

END behav;

ENTITY porta IS
PORT( d:      IN      vlbit_1d(4 DOWNTO 0);
      q05:    OUT     vlbit_1d(4 DOWNTO 0);
      q85:    OUT     vlbit_1d(4 DOWNTO 0);
      pin_i:  IN      vlbit_1d(4 DOWNTO 0);
      pin_o:  OUT     vlbit_1d(4 DOWNTO 0);
      pin_c:  OUT     vlbit_1d(4 DOWNTO 0);
      w05:    IN      vlbit;      -- active high
      w85:    IN      vlbit;      -- active high
      r05:    IN      vlbit;      -- active high
      rstb:   IN      vlbit
    );
END porta;

ARCHITECTURE behav OF porta IS

SIGNAL iqd, iqt, iqr: vlbit_1d(4 DOWNTO 0);

BEGIN

```

```

-- latch output data --
PROCESS BEGIN
    WAIT UNTIL pfalling(w05);
        iqd <= d;
END PROCESS;

-- latch 3-states control --
PROCESS BEGIN
    WAIT UNTIL pfalling(w85) OR (rstb='0'); -- async. reset
    IF (rstb='0') THEN
        iqt <= B"11111";
    ELSE
        iqt <= d;
    END IF;
END PROCESS;

-- latch input data --
PROCESS(pin_i,r05) BEGIN
    IF (r05='1') THEN
        iqr <= pin_i;
    END IF;
END PROCESS;

-- read input data --
q05(4 DOWNTO 0) <= iqr(4 DOWNTO 0);

-- send data out to PORT's pin --
pin_o <= iqd;
pin_c <= iqt;

-- read 3-states control --
q85(4 DOWNTO 0) <= iqt(4 DOWNTO 0);

END behav;

-- *** No pull-up resistor --
ENTITY portb IS
    PORT( d:    IN      vlbit_1d(7 DOWNTO 0);
          q06:  OUT     vlbit_1d(7 DOWNTO 0);
          q86:  OUT     vlbit_1d(7 DOWNTO 0);
          pin_i: IN      vlbit_1d(7 DOWNTO 0);
          pin_o: OUT     vlbit_1d(7 DOWNTO 0);
          pin_c: OUT     vlbit_1d(7 DOWNTO 0);
          w06:  IN      vlbit;  -- active high
          w86:  IN      vlbit;  -- active high
          r06:  IN      vlbit;  -- active high
          rbx:  OUT     vlbit;  -- active high
          rstb: IN      vlbit   );
END PORTB;

ARCHITECTURE behav OF portb IS

    SIGNAL iqd, iqt, iqr, pcd: vlbit_1d(7 DOWNTO 0);

BEGIN

```

```

-- latch output data --
PROCESS BEGIN
    WAIT UNTIL pfalling(w06);
        iqd <= d;
END PROCESS;

-- latch 3-states control --
PROCESS BEGIN
    WAIT UNTIL pfalling(w86) OR (rstb='0'); -- async reset
    IF (rstb='0') THEN
        iqt <= B"11111111";
    ELSE
        iqt <= d;
    END IF;
END PROCESS;

-- latch input data --
PROCESS(pin_i,r06) BEGIN
    IF (r06='1') THEN
        iqr <= pin_i;
    END IF;
END PROCESS;

-- read input data --
q06(7 DOWNT0 0) <= iqr(7 DOWNT0 0);

-- send data out to port's pin --
    pin_o <= iqd;
    pin_c <= iqt;

-- read 3-states control --
q86(7 DOWNT0 0) <= iqt(7 DOWNT0 0);

-- pin change detect --
pcd(7) <= (iqr(7) XOR pin_i(7)) AND iqt(7);
pcd(6) <= (iqr(6) XOR pin_i(6)) AND iqt(6);
pcd(5) <= (iqr(5) XOR pin_i(5)) AND iqt(5);
pcd(4) <= (iqr(4) XOR pin_i(4)) AND iqt(4);
pcd(3) <= (iqr(3) XOR pin_i(3)) AND iqt(3);
pcd(2) <= (iqr(2) XOR pin_i(2)) AND iqt(2);
pcd(1) <= (iqr(1) XOR pin_i(1)) AND iqt(1);
pcd(0) <= (iqr(0) XOR pin_i(0)) AND iqt(0);

rbx <= pcd(7) OR pcd(6) OR pcd(5) OR pcd(4) OR pcd(3) OR pcd(2) OR pcd(1) OR pcd(0);

END behav;

ENTITY fsr81 IS
    PORT( d:      IN    vlbit_1d(7 DOWNT0 0);
          q:      OUT   vlbit_1d(7 DOWNT0 0);
          w:      IN    vlbit;      -- active high
          rstb:   IN    vlbit       -- active low
        );
END fsr81;

```

ARCHITECTURE behav OF fsr81 IS

SIGNAL iq: vlbit_1d(7 DOWNT0 0);

BEGIN

 PROCESS(w,rstb,d) BEGIN

 IF (rstb='0') THEN

 iq <= B"11111111";

 ELSIF (w='1') THEN

 iq <= d;

 END IF;

 END PROCESS;

 q <= iq;

END behav;

ENTITY fsr0a IS

 PORT(d: IN vlbit_1d(4 DOWNT0 0);

 q: OUT vlbit_1d(4 DOWNT0 0);

 w: IN vlbit; -- *act.h*

 rstb: IN vlbit

);

END fsr0a;

ARCHITECTURE behav OF fsr0a IS

SIGNAL iq: vlbit_1d(4 DOWNT0 0);

BEGIN

 PROCESS(w,rstb,d)

 BEGIN

 IF (rstb='0') THEN

 iq <= B"00000";

 ELSIF (w='1') THEN

 iq <= d;

 END IF;

 END PROCESS;

 q <= iq;

END behav;

ENTITY intcon IS

 PORT(clk: IN vlbit; -- *f.edge*

 rstb: IN vlbit; -- *act.l*

 d: IN vlbit_1d(7 DOWNT0 0);

 q: OUT vlbit_1d(7 DOWNT0 0);

 upd: IN vlbit; -- *act.h*

 rbxint: IN vlbit; -- *act.h*

 tmr0upd: IN vlbit; -- *act.h*

 tmr0msb: IN vlbit; -- *f.edge*

 extint: IN vlbit; -- *act.h*

 intedge: IN vlbit; -- *0:falling, 1:rising*

 iwakeup: OUT vlbit; -- *act.h*

 clrgie: IN vlbit; -- *act.h*

```

        retfie:      IN      vlbit; -- act.h
        execint:    OUT     vlbit ); -- act.h
END INTCON;

```

ARCHITECTURE *behav* OF *intcon* IS

```

SIGNAL n_tfix, n_intx, n_rbix: vlbit;
SIGNAL iq: vlbit_1d(7 DOWNT0 0);
SIGNAL lq0, lq1, lq2, cq0, cq1, cq2: vlbit;
SIGNAL exitrg: vlbit;

```

BEGIN

```

ic0to6: PROCESS (clk,upd,d,rstb)
    BEGIN
        FOR i IN 0 TO 6 LOOP
            IF (rstb='0') THEN
                iq(i) <= '0';
            ELSIF (clk='1' AND upd='1') THEN
                iq(i) <= d(i);
            END IF;
        END LOOP;
    END PROCESS;

```

-- *modify GIE* --

```

ic7: PROCESS(clk,upd,d,rstb,RETFIE,clrgie)
    BEGIN
        IF (rstb='0') THEN
            iq(7) <= '0';
        ELSIF (clk='1') THEN
            IF (upd='1') THEN
                iq(7) <= d(7);
            ELSIF (RETFIE='1') THEN
                iq(7) <= '0';
            ELSIF (clrgie='1') THEN
                iq(7) <= '1';
            END IF;
        END IF;
    END PROCESS;

```

-- PROCESS RBIF (f0b.0)

-- latch to Int flag (edge trigger)

```

cc0: PROCESS BEGIN
    WAIT UNTIL prising(rbxint) OR (upd='1') OR (rstb='0');
    IF (rstb='0') THEN
        lq0 <= '0';
    ELSIF (upd='1') THEN
        lq0 <= '0';
    ELSE
        lq0 <= '1';
    END IF;
END PROCESS;

```

```

cq0 <= lq0 OR iq(0);
n_rbix <= cq0 AND iq(3);

```

```

-- process INTF (f0b.1) --
exitrg <= NOT(extint) XOR intedge;

-- latch to Int flag (edge trigger)
cc1: PROCESS BEGIN
    WAIT UNTIL prising(exitrg) OR (upd='1') OR (rstb='0');
    IF (rstb='0') THEN
        lq1 <= '0';
    ELSIF (upd='1') THEN
        lq1 <= '0';
    ELSE
        lq1 <= '1';
    END IF;
END PROCESS;

cql <= lq1 OR iq(1);
n_intx <= cql AND iq(4);

-- process T0IF (f0b.2) (tmr0msb is already synchronized with clk) --
cc2: PROCESS BEGIN
    WAIT UNTIL pfalling(tmr0msb) OR (upd='1') OR (rstb='0');
    IF (rstb='0') THEN
        lq2 <= '0';
    ELSIF (upd='1') THEN
        lq2 <= '0';
    ELSE
        lq2 <= NOT(tmr0upd);
    END IF;
END PROCESS;

cq2 <= lq2 OR iq(2);
n_t0ix <= iq(2) AND iq(5);

-- Int. event occurred drive sequencer to perform Int. routine --
execint <= (n_rbx OR n_intx OR n_t0ix) AND iq(7);
iwakeup <= n_rbx OR n_intx;
q(0) <= cq0;
q(1) <= cq1;
q(2) <= cq2;
q(7 DOWNT0 3) <= iq(7 DOWNT0 3);

END behav;

```

```

ENTITY fsr16 IS
  PORT( fin:   IN      vlbit_1d(7 DOWNT0 0);
        fout:  OUT     vlbit_1d(7 DOWNT0 0);
        fsel:   IN      vlbit_1d(3 DOWNT0 0);
        fwe:    IN      vlbit
        );
END FSR16;

```

```

ARCHITECTURE struc OF fsr16 IS

```

```

-- USE ram16x1 COMPONENT in package x4000 --

```

```

  COMPONENT ram16x1

```

```

    PORT(  d:      IN      vlbit;
           a0,a1,a2,a3: IN      vlbit;
           o:      OUT     vlbit;
           we:     IN      vlbit );

```

```

  END COMPONENT;

```

```

BEGIN

```

```

u0: ram16x1 PORT MAP(  d => fin(0),
                      a0 => fsel(0),
                      a1 => fsel(1),
                      a2 => fsel(2),
                      a3 => fsel(3),
                      o  => fout(0),
                      we => fwe   );

```

```

u1: ram16x1 PORT MAP(  d => fin(1),
                      a0 => fsel(0),
                      a1 => fsel(1),
                      a2 => fsel(2),
                      a3 => fsel(3),
                      o  => fout(1),
                      we => fwe   );

```

```

u2: ram16x1 PORT MAP(  d => fin(2),
                      a0 => fsel(0),
                      a1 => fsel(1),
                      a2 => fsel(2),
                      a3 => fsel(3),
                      o  => fout(2),
                      we => fwe   );

```

```

u3: ram16x1 PORT MAP(  d => fin(3),
                      a0 => fsel(0),
                      a1 => fsel(1),
                      a2 => fsel(2),
                      a3 => fsel(3),
                      o  => fout(3),
                      we => fwe   );

```

```

u4: ram16x1 PORT MAP(  d => fin(4),
                      a0 => fsel(0),

```


5. Top Level of PICCORE (PIC16C84 Microcontroller)

```

-----
-- Title       : PICCORE ( PIC16C84 of Microchip Inc.)
-- Structure    : PIC_CORE.vhd
--              +--ALM (Arithmetic and Logic Module)
--              +--PCSM (Program Counter and Stack Module)
--              +--IRIDM (Instruction Registers and Instruction Decoders Module)
--              +--FRM (IOPM) (Input and Output Port Module)
-- LIBRARY      : wvlogic
-- Developers    : Mr.Watcharakon Noothong
-- Created date  : 17 Feb 1999
-- version      : 1.00
-- Modify date   :
-----

```

```

LIBRARY synth;
USE synth.stdsynth.all;

```

```

ENTITY pic_core IS
  PORT(
    SIGNAL addr:      OUT  vlbit_1d(12 DOWNTO 0);
    SIGNAL data:      IN   vlbit_1d(13 DOWNTO 0);
    SIGNAL ra_i:      IN   vlbit_1d(4 DOWNTO 0);
    SIGNAL ra_o:      OUT  vlbit_1d(4 DOWNTO 0);
    SIGNAL ra_c:      OUT  vlbit_1d(4 DOWNTO 0);
    SIGNAL rb_i:      IN   vlbit_1d(7 DOWNTO 0);
    SIGNAL rb_o:      OUT  vlbit_1d(7 DOWNTO 0);
    SIGNAL rb_c:      OUT  vlbit_1d(7 DOWNTO 0);
    SIGNAL mclrb:     IN   vlbit;
    SIGNAL moscin:    IN   vlbit;
    SIGNAL moscenb:   OUT  vlbit;
    SIGNAL xoscin:    IN   vlbit;
    SIGNAL xoscenb:   OUT  vlbit;
    SIGNAL vddrise:   IN   vlbit;
    SIGNAL bodetb:    IN   vlbit;
    SIGNAL pwrenb:    OUT  vlbit;
    SIGNAL cfg2, cfg3, cfg6: IN vlbit );
END pic_core;

```

```

ARCHITECTURE struc OF pic_core IS

```

```

  COMPONENT alm
    PORT(
      clock:      IN   vlbit;
      kval:       IN   vlbit_1d(10 DOWNTO 0);
      fval:       IN   vlbit_1d(7 DOWNTO 0);
      pclath:     IN   vlbit_1d(4 DOWNTO 0);
      ksel:       IN   vlbit;
      f3cin:      IN   vlbit;    -- from status
      idcin:      IN   vlbit;    -- from Inst. dec.
      -- ALU's mode control --
      amode:      IN   vlbit_1d(1 DOWNTO 0);
      bmode:      IN   vlbit_1d(2 DOWNTO 0);
      smode:      IN   vlbit_1d(1 DOWNTO 0);
      bsel:       IN   vlbit_1d(2 DOWNTO 0);
      -- W Reg's control --

```

```

        wupd:      IN    vlbit;
        rbus:      OUT   vlbit_1d(12 DOWNT0 0);
        cout:      OUT   vlbit;
        dcout:     OUT   vlbit;
        zout:      OUT   vlbit
    );
END COMPONENT;

```

```

COMPONENT pcsbm
  PORT(
    clock:      IN    vlbit;
    resetb:     IN    vlbit;
    pactive:     IN    vlbit;
    pcmode:     IN    vlbit_1d(1 DOWNT0 0);
    addin:      IN    vlbit_1d(12 DOWNT0 0);
    addout:     OUT   vlbit_1d(12 DOWNT0 0) );
END COMPONENT;

```

```

COMPONENT iridm
  PORT(
    SIGNAL mclrb:      IN    vlbit; --act.l
    SIGNAL moscin:     IN    vlbit; -- clock from OSC external
    SIGNAL xoscin:     IN    vlbit; -- WDT clock ext.
    SIGNAL bodetb:     IN    vlbit; -- Brown OUT detect ext.
    SIGNAL vddrise:    IN    vlbit; -- Vddrise ext.
    SIGNAL cfg2,cfg3,cfg6: IN vlbit; -- Configuration bits
    SIGNAL iwakeup:    IN    vlbit; -- act.h from INTCON
    SIGNAL wdrst:      IN    vlbit; -- act.h from WDT
    SIGNAL instinp:     IN    vlbit_1d(13 DOWNT0 0); --Inst.data f Rom
    SIGNAL zflag:       IN    vlbit; -- from ALU
    SIGNAL pclupd:      IN    vlbit; -- from f02 PCL
    SIGNAL execint:     IN    vlbit; -- from INTCON
    -- Control signals --
    SIGNAL amode:       OUT   vlbit_1d(1 DOWNT0 0); -- to ALM
    SIGNAL bmode:       OUT   vlbit_1d(2 DOWNT0 0);
    SIGNAL smode:       OUT   vlbit_1d(1 DOWNT0 0);
    SIGNAL pactive:     OUT   vlbit; -- to PCSM
    SIGNAL pcmode:     OUT   vlbit_1d(1 DOWNT0 0);
    SIGNAL cupd:        OUT   vlbit;
    SIGNAL dcupd:       OUT   vlbit;
    SIGNAL zupd:        OUT   vlbit;
    SIGNAL ksel:        OUT   vlbit; -- to ALM
    SIGNAL cfed:        OUT   vlbit; -- to idcin signal of ALM
    SIGNAL wupd:        OUT   vlbit; -- to ALM
    SIGNAL fupd:        OUT   vlbit;
    SIGNAL oldinst:     OUT   vlbit;
    SIGNAL clrwtd:      OUT   vlbit;
    SIGNAL clrgie:      OUT   vlbit;
    SIGNAL retfie:      OUT   vlbit; -- from INTCON
    SIGNAL kval:        OUT   vlbit_1d(10 DOWNT0 0); -- from ALM kval
    SIGNAL moscenb:     OUT   vlbit; -- Show of enable mode
    SIGNAL xoscenb:     OUT   vlbit;
    SIGNAL pwrenb:      OUT   vlbit;
    SIGNAL sysclk:      OUT   vlbit; -- System Clock
    SIGNAL sysrstb:     OUT   vlbit -- System resetB );
END COMPONENT;

```

COMPONENT frm

```

PORT(
    SIGNAL fbusi:      IN    vlb1t_1d(7 DOWNT0 0); -- rbus(7:0)
    SIGNAL moscin:     IN    vlb1t;
    SIGNAL mclrb:      IN    vlb1t;
    SIGNAL xoscin:     IN    vlb1t;
    SIGNAL cfg2:       IN    vlb1t;
    SIGNAL abus:       IN    vlb1t_1d(7 DOWNT0 0); -- from PCSM
    SIGNAL zflag,dcflag,cflag: IN vlb1t;
    SIGNAL fupd:       IN    vlb1t; --from IDEC
    SIGNAL zupd:       IN    vlb1t; --from IDEC
    SIGNAL dcupd:      IN    vlb1t; --from IDEC
    SIGNAL cupd:       IN    vlb1t; --from IDEC
    SIGNAL oldinst:    IN    vlb1t; --from IDEC
    SIGNAL ibus:       IN    vlb1t_1d(6 DOWNT0 0); --kval(6:0)
    SIGNAL clrwtd:     IN    vlb1t; --from IDEC
    SIGNAL retfie:     IN    vlb1t; --from IDEC
    SIGNAL clrgie:     IN    vlb1t; --from IDEC
    -- PORT A and B --
    SIGNAL ra_i:       IN    vlb1t_1d(4 DOWNT0 0);
    SIGNAL rb_i:       IN    vlb1t_1d(7 DOWNT0 0);
    SIGNAL ra_c:       OUT   vlb1t_1d(4 DOWNT0 0);
    SIGNAL ra_o:       OUT   vlb1t_1d(4 DOWNT0 0);
    SIGNAL rb_c:       OUT   vlb1t_1d(7 DOWNT0 0);
    SIGNAL rb_o:       OUT   vlb1t_1d(7 DOWNT0 0);
    SIGNAL fbuso:      OUT   vlb1t_1d(7 DOWNT0 0); --fval(7:0) to ALU
    SIGNAL wdtrst:     OUT   vlb1t; -- to CLK_GEN
    SIGNAL pclupd:     OUT   vlb1t; -- to IDEC
    SIGNAL execint:    OUT   vlb1t; -- to IDEC
    SIGNAL f3cin:      OUT   vlb1t;
    SIGNAL pclath:     OUT   vlb1t_1d(4 DOWNT0 0);
    SIGNAL iwakeup:    OUT   vlb1t );

```

END COMPONENT;

-- Internal connections --

```

SIGNAL n_abus:          vlb1t_1d(12 DOWNT0 0);
SIGNAL n_inst           vlb1t_1d(13 DOWNT0 0);
SIGNAL n_sysclk,n_sysrstb: vlb1t;
SIGNAL n_mclrb,n_moscin,n_xoscin,n_bodetb,n_vddrise: vlb1t;
SIGNAL n_cfg2,n_cfg3,n_cfg6: vlb1t;
SIGNAL n_moscenb,n_xoscenb,n_pwrenb: vlb1t;
SIGNAL n_iwakeup,n_wdtrst,n_dcflag,n_cflag: vlb1t;
SIGNAL n_zflag,n_pclupd,n_execINT: vlb1t;
SIGNAL n_amode,n_smode,n_pcmode: vlb1t_1d(1 DOWNT0 0);
SIGNAL n_bmode,n_bsel: vlb1t_1d(2 DOWNT0 0);
SIGNAL n_pactive,n_cupd,n_dcupd,n_zupd,n_ksel: vlb1t;
SIGNAL n_cfed,n_wupd,n_fupd,n_oldinst,n_clrwtd: vlb1t;
SIGNAL n_clrgie,n_retjie,n_f3cin: vlb1t;
SIGNAL n_kval:          vlb1t_1d(10 DOWNT0 0);
SIGNAL n_fval:          vlb1t_1d(7 DOWNT0 0);
SIGNAL n_rbus:          vlb1t_1d(12 DOWNT0 0);
SIGNAL n_pclath:        vlb1t_1d(4 DOWNT0 0);

```

BEGIN

-- external ROM signals --

```
addr      <=    n_abus;
n_inst    <=    data;
```

-- external in PORT --

```
n_mclrb   <=    mclrb;
n_moscin  <=    moscin;
n_xoscin  <=    xoscin;
n_bodetb  <=    bodetb;
n_vddrise <=    vddrise;
n_cfg2    <=    cfg2;
n_cfg3    <=    cfg3;
n_cfg6    <=    cfg6;
```

-- external out PORT --

```
moscenb   <=    n_moscenb;
xoscenb   <=    n_xoscenb;
pwrenb    <=    n_pwrenb;
```

-- Instruction decoder and Control module --

PIC_IRIDM: IRIDM	PORT MAP(	mclrb	=>	n_mclrb, --from reset ext.
		moscin	=>	n_moscin, --from OSC ext.
		xoscin	=>	n_xoscin, --from XTAL xt.
		bodetb	=>	n_bodetb,
		vddrise	=>	n_vddrise,
		cfg2	=>	n_cfg2,
		cfg3	=>	n_cfg3,
		cfg6	=>	n_cfg6,
		iwakeup	=>	n_iwakeup, --from INCON
		wdtrst	=>	n_wdtrst, --from WDT
		instinp	=>	n_inst, --from ext. ROM
		zflag	=>	n_zflag,
		pclupd	=>	n_pclupd,
		execint	=>	n_execint,
		amode	=>	n_amode,
		bmode	=>	n_bmode,
		smode	=>	n_smode,
		pcactive	=>	n_pactive,
		pcmode	=>	n_pcmode,
		cupd	=>	n_cupd,
		dcupd	=>	n_dcupd,
		zupd	=>	n_zupd,
		ksel	=>	n_ksel,
		cfed	=>	n_cfed,
		wupd	=>	n_wupd,
		fupd	=>	n_fupd,
		oldinst	=>	n_oldinst,
		clrwdt	=>	n_clrwdt,
		clrgie	=>	n_clrgie,
		retfie	=>	n_retfie,
		kval	=>	n_kval,
		moscenb	=>	n_moscenb,
		xoscenb	=>	n_xoscenb,
		pwrenb	=>	n_pwrenb,

```

sysclk      =>    n_sysclk,  --system clk
sysrstb     =>    n_sysrstb ); --system resetb

```

-- Arithmetic and Logic module --

```
n_bsel <= n_kval(9 DOWNT0 7); -- to Bsel signal of ALM
```

```

PIC_ALU: alm PORT MAP( clock  =>    n_sysclk,
                        kval   =>    n_kval,
                        fval   =>    n_fval,
                        pclath =>    n_pclath,
                        ksel   =>    n_ksel,
                        f3cin  =>    n_f3cin,
                        idcin  =>    n_cfed,
                        amode  =>    n_amode,
                        bmode  =>    n_bmode,
                        smode  =>    n_smode,
                        bsel   =>    n_bsel,
                        wupd   =>    n_wupd,
                        rbus   =>    n_rbus,
                        cout   =>    n_cflag,
                        dcout  =>    n_dcflag,
                        zout   =>    n_zflag      );

```

-- 13-bits Program Counter & Stack Module --

```

PIC_PCSM: pcsm PORT MAP( clock  =>    n_sysclk,
                        resetb  =>    n_sysrstb,
                        pactive =>    n_pactive,
                        pcmode  =>    n_pcmode,
                        addin   =>    n_rbus,
                        addout  =>    n_abus );

```

-- File Registers Module --

```

PIC_FRM: frm PORT MAP( fbusi  =>    n_rbus(7 DOWNT0 0),
                        moscin =>    n_sysclk,
                        mclrb  =>    n_sysrstb,
                        xoscin =>    n_xoscin,
                        cfg2   =>    n_cfg2,
                        abus   =>    n_abus(7 DOWNT0 0),
                        zflag  =>    n_zflag,
                        dcflag =>    n_dcflag,
                        cflag  =>    n_cflag,
                        fupd   =>    n_fupd,
                        zupd   =>    n_zupd,
                        dcupd  =>    n_dcupd,
                        cupd   =>    n_cupd,
                        oldinst =>    n_oldinst,
                        ibus   =>    n_kval(6 DOWNT0 0),
                        clrwtd =>    n_clrwtd,
                        retfie =>    n_retfie,
                        clrgie =>    n_clrgie,
                        ra_i   =>    ra_i,
                        rb_i   =>    rb_i,

```

```

ra_c    =>    ra_c,
ra_o    =>    ra_o,
rb_c    =>    rb_c,
rb_o    =>    rb_o,
fbuso   =>    n_fval,
wdtrst  =>    n_wdtrst,
pclupd  =>    n_pclupd,
execint =>    n_execint,
f3cin   =>    n_f3cin,
pclath  =>    n_pclath,
iwakeup =>    n_iwakeup
END struc;
);

```

ภาคผนวก ข.

รายละเอียดชุดคำสั่งของไมโครคอนโทรลเลอร์โดยชุดคำสั่งจะแบ่งออกเป็น 6 กลุ่ม ตามลักษณะการทำงานดังนี้

1. กลุ่มคำสั่งการโอนย้ายหรือกำหนดค่าข้อมูล

มีทั้งสิ้น 3 คำสั่ง คือ MOVLW, MOVF, MOVWF ซึ่งมีรายละเอียดดังต่อไปนี้

1.1 คำสั่ง MOVLW (Move data to W register)

เป็นคำสั่งที่ใช้ในการกำหนดค่าคงที่ขนาด 8 บิตลงในเรจิสเตอร์ W ขอบเขตของค่าคงที่ที่ใช้ได้ระหว่าง 0-255 มีรูปแบบและรายละเอียดดังนี้

รูปแบบ : MOVLW k

โอเปอร์เรนด์ : $0 \leq k \leq 255$

การทำงาน : $k \rightarrow (W)$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : MOVLW 0x5A

หลังจากการกระทำคำสั่ง ค่าของ W = 0x5A

1.2 คำสั่ง MOVF (Move data from register to register)

เป็นคำสั่งที่ใช้ในการโอนย้ายข้อมูลของเรจิสเตอร์ที่กำหนดไปเก็บไว้ในเรจิสเตอร์ W หรือเรจิสเตอร์ที่กำหนด การเลือกตำแหน่งปลายทางกำหนดได้ด้วยข้อมูล “0” หรือ “1” ต่อหลังจากเรจิสเตอร์ไฟล์ต้นทาง หากกำหนดให้เป็นศูนย์จะหมายถึงให้โอนย้ายข้อมูลไปเก็บไว้ในเรจิสเตอร์ W ถ้ากำหนดให้เป็นหนึ่งข้อมูลจะโอนย้ายมาที่ตำแหน่งเดิม ซึ่งก็คือไม่เกิดการโอนย้ายข้อมูล แต่จะเกิดผลกับแฟลก โดยแฟลกศูนย์ (Z) จะถูกเซต ดังนั้นคำสั่งนี้จึงสามารถนำไปใช้เซตค่าแฟลกศูนย์ได้เลย โดยไม่ต้องใช้คำสั่งทางคณิตศาสตร์แต่อย่างใด สำหรับรูปแบบและรายละเอียดของคำสั่งมีดังนี้

รูปแบบ : MOVF f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127 ; d \in [0, 1]$

การทำงาน : $(f) \rightarrow (W)$ or $(f) \rightarrow (f), Z=1$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : MOVF FSR,0

หลังจากการกระทำคำสั่ง ค่าของ W = ค่าของเรจิสเตอร์ FSR

MOVWF FSR,I

หลังจากการกระทำคำสั่ง ค่าของ FSR จะเท่าเดิมแต่แฟล็ก Z จะถูกเซต

1.3 คำสั่ง MOVWF (Move data from W register to file register)

เป็นคำสั่งที่ให้ผลตรงข้ามกับ MOVF นั่นคือ เป็นคำสั่งโอนย้ายข้อมูลจากเรจิสเตอร์ W มาเก็บไว้ในเรจิสเตอร์ไฟล์ที่กำหนดมีรายละเอียดดังนี้

รูปแบบ : MOVWF f

โอเปอเรนด์ : $0 \leq f \leq 127$

การทำงาน : (W) $\rightarrow$ (f)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : MOVWF FSR

ก่อนกระทำคำสั่ง ค่าของ FSR = 0xFF ค่าของ W = 0x4F

หลังจากการกระทำคำสั่ง ค่าของ FSR = 0x4F ค่าของ W = 0x4F

2. กลุ่มคำสั่งเปลี่ยนแปลงค่าของเรจิสเตอร์

มีทั้งสิ้น 10 คำสั่ง คือ CLRF, CLRW, COMF, DECF, INCF, BCF, BSF, RLF, RRF และ SWAPF ซึ่งมีรายละเอียดดังต่อไปนี้

2.1 คำสั่ง CLRF (Clear file register)

เป็นคำสั่งเคลียร์ค่าของเรจิสเตอร์ไฟล์ใด ๆ ให้เป็นศูนย์ แล้วแฟล็กศูนย์ (Z) จะถูกเซต มีรายละเอียดดังนี้

รูปแบบ : CLRF f

โอเปอเรนด์ : $0 \leq f \leq 127$

การทำงาน : 00H $\rightarrow$ (f), 1 $\rightarrow$ Z

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : CLRF TMP0

ก่อนกระทำคำสั่ง ค่าของ TMP0 = 0x5F

หลังจากการกระทำคำสั่ง ค่าของ TMP0 = 0x00, Z = 1

2.2 CLRW (Clear W register)

เป็นคำสั่งเคลียร์ค่าของเรจิสเตอร์ W ให้เป็นศูนย์ โดยจะมีผลต่อแฟลกศูนย์ โดยมีรายละเอียดดังนี้

รูปแบบ : CLRW

โอเปอร์เรนด์ : -

การทำงาน : $00H \rightarrow (W), 1 \rightarrow Z$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : CLRW

ก่อนกระทำคำสั่ง ค่าของ W = 0x4F

หลังจากการกระทำคำสั่ง ค่าของ W = 0x00, Z = 1

2.3 คำสั่ง COMF (Complement register)

เป็นคำสั่งใช้กลับข้อมูล “0” ให้เป็น “1” และกลับข้อมูล “1” ให้เป็น “0” ของเรจิสเตอร์ไฟล์ใดๆ หรือเรียกว่าการทำคอมพลีเมนต์ หลังจากกระทำคำสั่งนี้แล้ว ค่าของเรจิสเตอร์ไฟล์ที่ถูกกำหนดจะมีค่าตรงกันข้าม เช่น จากเดิมมีค่า 10110011 หรือ 0xB3 จะกลายเป็น 01001100 หรือ 0x4C เป็นต้น รายละเอียดและรูปแบบมีดังนี้

รูปแบบ : COMF f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : $f \rightarrow (W)$ or $f \rightarrow (f)$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : COMF TMP0,0

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0x13

หลังกระทำคำสั่งค่าของ TMP0 = 0x13 แต่ W = 0xEC

COMF TMP0,1

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0x13

หลังกระทำคำสั่งค่าของ TMP0 = 0xEC แต่ W = No change

2.4 คำสั่ง DECF (Decrement register)

เป็นคำสั่งลดค่าของเรจิสเตอร์ไฟล์ใดๆ ลงหนึ่งค่า สามารถเก็บค่าได้ 2 แห่งคือ ที่เรจิสเตอร์ไฟล์ตัวเดิม หรือ เรจิสเตอร์ W มีรายละเอียดดังนี้

รูปแบบ : DECF f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : $(f) - 1 \rightarrow (W)$ when $d = 0$ or $(f) - 1 \rightarrow (f)$ when $d = 1$

$1 \rightarrow Z$ when result = 0

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : DECF TMP0,0

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0x01 , Z = 0

หลังการกระทำคำสั่งค่าของ TMP0 = 0x00 , W = 0x00 , Z = 1

DECF TMP0,1

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0x01 , Z = 0

หลังการกระทำคำสั่ง ค่าของ TMP0 = 0x00 , W = No change , Z = 1

2.5 คำสั่ง INCF (Increment register)

เป็นคำสั่งที่ใช้เพิ่มค่าของเรจิสเตอร์ไฟลต์ใด ๆ ขึ้นหนึ่งค่าการเก็บผลลัพธ์เหมือนกับคำสั่ง DECF ทุกประการ โดยมีรายละเอียดดังนี้

รูปแบบ : INCF f,d

โอเปอเรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : $(f) + 1 \rightarrow (W)$ when $d = 0$ or $(f) + 1 \rightarrow (f)$ when $d = 1$

$1 \rightarrow Z$ when result = 0

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : INCF TMP0,0

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xFF , Z = 0

หลังการกระทำคำสั่งค่าของ TMP0 = 0x00 , W = 0x00 , Z = 1

INCF TMP0,1

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xFF , Z = 0

หลังการกระทำคำสั่ง ค่าของ TMP0 = 0x00 , W = No change , Z = 1

2.6 คำสั่ง BCF (Clear bit in register)

เป็นคำสั่งเคลียร์บิตในเรจิสเตอร์ไฟลต์ที่กำหนดให้เป็นศูนย์ มีรูปแบบดังนี้

รูปแบบ : BCF f,b

โอเปอเรนด์ : $0 \leq f \leq 127, 0 \leq b \leq 7$

การทำงาน : $0 \rightarrow (f, \langle b \rangle)$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : BCF TMP0,7

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xC7 (1100 0111)
 หลังการกระทำคำสั่งค่าของ TMP0 = 0x4F (0100 0111)

2.7 คำสั่ง BSF (Set bit in register)

เป็นคำสั่งเซตบิตในรีจิสเตอร์ไฟล์ที่กำหนดให้เป็นศูนย์ มีรูปแบบดังนี้

รูปแบบ : BSF f,b

โอเปอเรนด์ : $0 \leq f \leq 127, 0 \leq b \leq 7$

การทำงาน : $1 \rightarrow (f,)$

จำนวนเวิร์ด : 1 จำนวนไบต์เกิด : 1

ตัวอย่าง : BSF TMP0,7

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0x0A (0000 1010)

หลังการกระทำคำสั่งค่าของ TMP0 = 0x8A (1000 1010)

2.8 คำสั่ง RLF (Rotate left register through carry)

เป็นคำสั่งหมุนข้อมูลในเรจิสเตอร์ไปทางซ้ายเข้าสู่แฟลกกท (Carry flag) มีรายละเอียดดังนี้

รูปแบบ : RLF f,d

โอเปอเรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน :



จำนวนเวิร์ด : 1 จำนวนไบต์เกิด : 1

ตัวอย่าง : RLF TMP0,0

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xC7 (1100 0111), C = 0

หลังการกระทำคำสั่งค่าของ TMP0 = 0xC7 (1100 0111), C = 1

ค่าของ W = 0x8E (1000 1110)

RLF TMP0,1

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xC7 (1100 0111), C = 0

หลังการกระทำคำสั่งค่าของ TMP0 = 0x8E (1000 1110), C = 1

ค่าของ W = No change

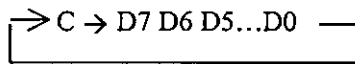
2.9 คำสั่ง RRF (Rotate right register through carry)

เป็นคำสั่งหมุนข้อมูลในเรจิสเตอร์ไปทางขวาเข้าสู่แฟลกทค (Carry flag) มีรายละเอียดดังนี้

รูปแบบ : RRF f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน :



จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : RRF TMP0,0

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xC7 (1100 0111), C = 0

หลังการกระทำคำสั่งค่าของ TMP0 = 0xC7 (1100 0111), C = 1

ค่าของ W = 0x63 (0110 0011)

RLF TMP0,1

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xC7 (1100 0111), C = 0

หลังการกระทำคำสั่งค่าของ TMP0 = 0x63 (0110 0011), C = 1

ค่าของ W = No change

2.10 คำสั่ง SWAPF (Swap register)

เป็นคำสั่งสลับข้อมูล 4 บิตบน หรือ 4 บิตล่างของเรจิสเตอร์ผลของการสลับข้อมูลจะเก็บไว้ในเรจิสเตอร์ W หรือเรจิสเตอร์ตัวเดิม โดยมีรายละเอียดดังนี้

รูปแบบ : swap f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : when d = 0 ($f<3:0> \rightarrow W<7:4>$ and $f<7:4> \rightarrow W<3:0>$)

when d = 1 ($f<3:0> \rightarrow f<7:4>$ and $f<7:4> \rightarrow f<3:0>$)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : SWAPF TMP0,0

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xA5

หลังการกระทำคำสั่งค่าของ TMP0 = 0xA5, W = 0x5A

SWAPF TMP0,1

ก่อนการกระทำคำสั่ง ค่าของ TMP0 = 0xA5

หลังการกระทำคำสั่งค่าของ TMP0 = 0x5A, W = No change

3. กลุ่มคำสั่งการกระโดดและควบคุมลำดับการทำงานของโปรแกรม

มีด้วยกัน 9 คำสั่ง คือ GOTO, CALL, RETURN, RETLW, RETFIE, BTFSC, BTFSS, DECFSZ และ INCFSZ ซึ่งมีรายละเอียดดังต่อไปนี้

3.1 คำสั่ง GOTO (Go to address)

เป็นคำสั่งที่กำหนดให้ซีพียูกระโดดไปทำงานยังแอดเดรสที่กำหนดไว้ในหน่วยความจำโปรแกรมโดยไม่มีเงื่อนไขค่าคงที่กำหนดให้กระโดดนี้ต้องไม่เกิน 2047 (2^{11}) โดยมีรายละเอียดดังนี้

รูปแบบ : GOTO k

โอเปอร์เรนด์ : $0 \leq k \leq 2047$

การทำงาน : $k \rightarrow (PC<10:0>) (PCLATH<4:3>) \rightarrow (PC<12:11>)$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 2

ตัวอย่าง : GOTO THERE

หลังกระทำคำสั่ง PC = ค่าแอดเดรส THERE

3.2 คำสั่ง CALL (Subroutine CALL)

เป็นคำสั่งที่กำหนดให้ซีพียูกระโดดไปทำงานที่โปรแกรมน้อย โดยไม่มีเงื่อนไขและจะกลับมายังโปรแกรมหลักก็ต่อเมื่อพบคำสั่ง RETURN หรือ RETLW เมื่อกระทำคำสั่งนี้ ซีพียูจะทำการเก็บค่า PC ไว้ในสแตคโดยอัตโนมัติ หลังจากทีซีพียูทำงานในโปรแกรมน้อยเสร็จ ก็จะกลับมาอ่านค่า PC จากสแตค แล้วกลับไปทำงานอย่างถูกต้อง

รูปแบบ : CALL k

โอเปอร์เรนด์ : $0 \leq k \leq 2047$

การทำงาน : $(PC) + 1 \rightarrow TOS (Top Of Stack)$

$k \rightarrow (PC<10:0>) \text{ and } (PCLATH<4:3>) \rightarrow PC<12:11>$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 2

ตัวอย่าง : HERE CALL THERE

ก่อนการกระทำคำสั่ง PC = address HERE

หลังกระทำคำสั่ง PC = address THERE ; TOS = address HERE

3.3 คำสั่ง RETURN (RETURN from subroutine)

เป็นคำสั่งที่แจ้งให้ซีพียูทราบว่าสิ้นสุดการทำงานของโปรแกรมย่อยแล้ว เมื่อซีพียูกระทำคำสั่งแล้ว จะไปอ่านค่า PC จากสแตคที่นำไปเก็บไว้ชั่วคราวก่อนหน้าที่กระทำคำสั่ง CALL จากนั้นกระโดดไปยังแอดเดรสที่กำหนดโดย PC ในการเขียนโปรแกรมย่อยทุกครั้งต้องมีคำสั่ง RETURN หรือ RETLW ต่อท้ายเสมอ

รูปแบบ : RETURN
 โอเปอร์เรนด์ : -
 การทำงาน : TOS $\rightarrow$ PC
 จำนวนเวิร์ด : 1 จำนวนไบต์ : 2
 ตัวอย่าง : RETURN
 หลังกระทำคำสั่ง PC = TOS

3.4 คำสั่ง RETLW (RETURN from subroutine and literal to W)

เป็นคำสั่งที่ใช้ในการออกจากโปรแกรมย่อยอีกแบบหนึ่งที่มีความแตกต่างจากคำสั่ง RETURN ตรงที่ หลังจากกระทำคำสั่งนี้แล้วซีพียูจะไปอ่านค่าจาก PC ในสแตค เพื่อกระโดดกลับไปทำงานต่อที่แอดเดรสซึ่งกำหนดไว้ใน PC อันเป็นแอดเดรสที่ต่อจากคำสั่งที่กำหนดให้ซีพียูกระโดดมาทำงานที่โปรแกรมย่อย พร้อมกันนั้นยังมีการโอนย้ายค่าคงที่ที่กำหนดไว้ต่อท้ายคำสั่ง RETLW ไปเก็บไว้ในเรจิสเตอร์ W ด้วย ดังนั้นคำสั่งนี้ใช้สำหรับการเปิดตารางเพื่อค้นหาหรือแปลงข้อมูล หรือที่เรียกว่า Look up table โดยมีรายละเอียดดังนี้

รูปแบบ : RETLW k
 โอเปอร์เรนด์ : $0 \leq k \leq 255$
 การทำงาน : $k \rightarrow W$; TOS $\rightarrow$ (PC)
 จำนวนเวิร์ด : 1 จำนวนไบต์ : 2
 ตัวอย่าง : CALL TABLE ; เรจิสเตอร์ W เก็บค่าออฟเซตของตาราง

TABLE ADDWF PC
 RETLW k1 ; เริ่มต้นตาราง
 RETLW k2

RETLW kn ; สิ้นสุดตาราง
 ก่อนการกระทำคำสั่ง ค่าของ W = 0x07
 หลังการกระทำคำสั่งค่าของ W = ค่าของ kn

3.5 คำสั่ง RETFIE (RETURN from interrupt service routine)

เป็นคำสั่งที่ใช้ในการออกแบบจากโปรแกรมย่อยของการบริการอินเทอร์รัพท์ หลังจากกระทำคำสั่งนี้ซีพียูจะกลับไปทำงานยังแอดเดรสที่ถัดจากแอดเดรสสุดท้ายที่ทำงานก่อนการเกิดอินเทอร์รัพท์โดยค่าแอดเดรสนี้จะเก็บไว้ในสแตคโดยฮาร์ดโนมิตี หลังจากที่เกิดการอินเทอร์รัพท์ขึ้น พร้อมกันนั้นยังทำการเซตบิต GIE (Global Interrupt Enable bit) ด้วยเพื่อทำการอินเเปิดการตอบสนองการอินเทอร์รัพท์ของซีพียูอีกครั้ง โดยมีรายละเอียดดังนี้

รูปแบบ : RETFIE
 โอเปอเรนด์ : -
 การทำงาน : TOS (PC) , 1 → GIE
 จำนวนเวิร์ด : 1 จำนวนไบต์ : 2
 ตัวอย่าง : RETFIE
 หลังการกระทำคำสั่ง PC = TOS , GIE = 1

3.6 คำสั่ง BTFSC (Bit test register, skip if clear)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขโดยทำการตรวจสอบบิตที่กำหนดในเรจิสเตอร์ว่ามีค่าเป็นศูนย์หรือไม่ ถ้าใช่จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วทำงานต่อที่แอดเดรสต่อไป

รูปแบบ : BTFSC f,b
 โอเปอเรนด์ : $0 \leq f \leq 127, 0 \leq b \leq 7$
 การทำงาน : Skip if ($f < b = 0$)
 จำนวนเวิร์ด : 1 จำนวนไบต์ : 1 or 2
 ตัวอย่าง : HERE BTFSC TMP0,1
 FALSE GOTO CODE
 TRUE
 ก่อนการกระทำคำสั่ง PC = address HERE
 หลังการกระทำคำสั่ง
 ถ้าบิตที่ 1 ของ TMP0 = 0, PC = address TRUE
 ถ้าบิตที่ 1 ของ TMP0 = 1, PC = address FALSE

3.7 คำสั่ง BTFSS (Bit test register, skip if set)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขโดยทำการตรวจสอบบิตที่กำหนดในเรจิสเตอร์ว่ามีค่าเป็นหนึ่งหรือไม่ ถ้าใช่จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วทำงานต่อที่แอดเดรสต่อไป

รูปแบบ : BTFSS f,b

โอเปอเรนด์ : $0 \leq f \leq 127, 0 \leq b \leq 7$

การทำงาน : Skip if (f = 1)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1 or 2

ตัวอย่าง : HERE BTFSS TMP0,1

FALSE GOTO CODE

TRUE

ก่อนการกระทำคำสั่ง PC = address HERE

หลังการกระทำคำสั่ง

ถ้าบิตที่ 1 ของ TMP0 = 1, PC = address TRUE

ถ้าบิตที่ 1 ของ TMP0 = 0, PC = address FALSE

3.8 คำสั่ง DECFSZ (Decrement register, skip if zero)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขโดยทำการลดค่าในเรจิสเตอร์ลงหนึ่งค่า แล้วตรวจสอบค่าเรจิสเตอร์ว่ามีค่าเป็นศูนย์หรือไม่ ถ้าใช่จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วทำงานต่อที่แอดเดรสต่อไป ค่าของเรจิสเตอร์ที่ทำการลดค่าลงแล้ว สามารถถูกเก็บไว้ในเรจิสเตอร์ตัวเดิมหรือเรจิสเตอร์ W

รูปแบบ : DECFSZ f,d

โอเปอเรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : when d = 0 ; (f) - 1 → (W) Skip if (f) = 0

when d = 1 ; (f) - 1 → (f) Skip if (f) = 0

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1 or 2

ตัวอย่าง : HERE DECFSZ TMP0,1

GOTO LOOP

CONT

ก่อนการกระทำคำสั่ง PC = address HERE

หลังการกระทำคำสั่ง TMP0 = TMP0 - 1

ถ้า TMP0 = 0, PC = address CONT

ถ้า TMP0 ≠ 0, PC = address HERE + 1

3.9 คำสั่ง INCFSZ (Increment register, skip if zero)

เป็นคำสั่งกระโดดอย่างมีเงื่อนไขโดยทำการเพิ่มค่าในเรจิสเตอร์ขึ้นหนึ่งค่า แล้วตรวจสอบค่าเรจิสเตอร์ว่ามีค่าเป็นศูนย์หรือไม่ ถ้าใช่จะกระโดดข้ามแอดเดรสถัดไป 1 แอดเดรส แล้วทำงานต่อที่แอดเดรสต่อไป ค่าของเรจิสเตอร์ที่ทำการเพิ่มค่าขึ้นแล้ว สามารถถูกเก็บไว้ในเรจิสเตอร์ตัวเดิมหรือเรจิสเตอร์ W

รูปแบบ : INCFSZ f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : when $d = 0$; $(f) + 1 \rightarrow (W)$ Skip if $(f) = 0$

when $d = 1$; $(f) + 1 \rightarrow (f)$ Skip if $(f) = 0$

จำนวนเวิร์ด : 1 จำนวนไบต์เกิด : 1 or 2

ตัวอย่าง : HERE INCFSZ TMP0,1

GOTO LOOP

CONT

ก่อนการกระทำคำสั่ง PC = address HERE

หลังการกระทำคำสั่ง $TMP0 = TMP0 + 1$

ถ้า $TMP0 = 0$, PC = address CONT

ถ้า $TMP0 \neq 0$, PC = address HERE + 1

4. กลุ่มคำสั่งควบคุมไมโครคอนโทรลเลอร์

มีด้วยกัน 4 คำสั่ง คือ CLRWDT, SLEEP, TRIS และ NOP ซึ่งมีรายละเอียดดังต่อไปนี้

4.1 คำสั่ง CLRWDT (Clear Watchdog Timer)

เป็นคำสั่งที่ใช้เคลียร์ค่าของวอตช์ด็อกไทมเมอร์และปริสเกลเลอร์ภายในตัวไมโครคอนโทรลเลอร์ พร้อมเซตบิต TO และ PD มีรายละเอียดดังนี้

รูปแบบ : CLRWDT

โอเปอร์เรนด์ : -

การทำงาน : $00H \rightarrow WDT$

$0 \rightarrow WDT \text{ Prescaler}$; $1 \rightarrow TO$, $1 \rightarrow PD$

จำนวนเวิร์ด : 1 จำนวนไบต์เกิด : 1

ตัวอย่าง : CLRWDT

ก่อนการกระทำคำสั่ง WDT counter = ?

หลังการกระทำคำสั่ง WDT counter = 0x00

WDT prescaler = 0 ; TO = 1, PD = 0

4.2 คำสั่ง SLEEP (Go to standby mode)

เป็นคำสั่งที่กำหนดให้ชิพหยุดทำงานในโหมดสแตนด์บาย เพื่อประหยัดพลังงาน หลังจากการกระทำคำสั่งนี้ ค่าการนับภายในวอตช์ด็อกไทมเมอร์จะถูกเคลียร์ให้เป็นศูนย์ พร้อมๆ กับค่าของปริสเกลเลอร์ภายในวอตช์ด็อกให้เป็นศูนย์ เพื่อแจ้งให้ทราบว่า ขณะนี้อยู่ในโหมดสแตนด์บายโหมด (คำสั่งนี้ไม่สามารถทำได้ในวิทยานิพนธ์นี้)

4.3 คำสั่ง TRIS (Load TRIS register)

เป็นคำสั่งที่ใช้ในการกำหนดค่าของเรจิสเตอร์ TRIS ซึ่งเป็นเรจิสเตอร์กำหนดทิศทางของการส่งข้อมูลเข้าหรือออกของพอร์ต โดยค่าที่กำหนดนี้จะเก็บไว้ในเรจิสเตอร์ W เพราะเราไม่สามารถจะเข้าเรจิสเตอร์นี้ได้โดยตรง โดยมีรายละเอียดดังนี้

รูปแบบ : TRIS f

โอเปอเรนด์ : $5 \leq f \leq 7$

การทำงาน : (W) $\rightarrow$ TRIS (f)

4.4 คำสั่ง NOP (No operation)

เป็นคำสั่งพิเศษ จะใช้ในการหน่วงเวลา เพราะเมื่อชิพหยุดการทำงานคำสั่งจะเกิดผลเพียงอย่างเดียวคือค่าของโปรแกรมเคาน์เตอร์เพิ่มขึ้น

5. กลุ่มคำสั่งประมวลผลทางลอจิก

มีด้วยกัน 6 คำสั่ง คือ ANDLW, ANDWF, IORLW, IORWF, XORLW และ XORWF

5.1 คำสั่ง ANDLW (AND literal with W)

เป็นคำสั่งให้ทำการแอนด์ค่าคงที่กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในเรจิสเตอร์ W

รูปแบบ : ANDLW k

โอเปอเรนด์ : $0 \leq k \leq 255$

การทำงาน : (W) AND (k) $\rightarrow$ (W)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : ANDLW 0x18
 ก่อนการกระทำคำสั่ง W = 0x33 (0011 0011)
 หลังการกระทำคำสั่ง W = 0x10 (0001 0000)

5.2 คำสั่ง ANDWF (AND W with f)

เป็นคำสั่งให้ทำการแอนด์ค่าในเรจิสเตอร์ใดๆ กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ที่เรจิสเตอร์ W หรือ เรจิสเตอร์นั้น

รูปแบบ : ANDWF f,d
 โอเปอเรนด์ : $0 \leq f \leq 127, d \in [0, 1]$
 การทำงาน : when d = 0 (W) AND (f) $\rightarrow$ (W)
 when d = 1 (W) AND (f) $\rightarrow$ (f)
 จำนวนเวิร์ด : 1 จำนวนไบต์ : 1
 ตัวอย่าง : ANDWF TMP0,0
 ก่อนการกระทำคำสั่ง W = 0x17, TMP0 = 0xC2
 หลังการกระทำคำสั่ง W = 0x02, TMP0 = 0xC2
 ANDWF TMP0,1
 ก่อนการกระทำคำสั่ง W = 0x17, TMP0 = 0xC2
 หลังการกระทำคำสั่ง W = 0x17, TMP0 = 0x02

5.3 คำสั่ง IORLW (OR literal with W)

เป็นคำสั่งให้ทำการออร์ค่าคงที่กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ที่เรจิสเตอร์ W

รูปแบบ : IORLW k
 โอเปอเรนด์ : $0 \leq k \leq 255$
 การทำงาน : (W) OR (k) $\rightarrow$ (W)
 จำนวนเวิร์ด : 1 จำนวนไบต์ : 1
 ตัวอย่าง : IORLW 0x18
 ก่อนการกระทำคำสั่ง W = 0x33 (0011 0011)
 หลังการกระทำคำสั่ง W = 0x3B (0011 1011)

5.4 คำสั่ง IORWF (OR W with f)

เป็นคำสั่งให้ทำการออร์ค่าในเรจิสเตอร์ใดๆ กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ที่เรจิสเตอร์ W หรือ เรจิสเตอร์นั้น

รูปแบบ : IORWF f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : when d=0 (W) OR (f) $\rightarrow$ (W)
when d=1 (W) OR (f) $\rightarrow$ (f)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : IORWF TMP0,0

ก่อนการกระทำคำสั่ง W = 0x91 , TMP0 = 0x13

หลังการกระทำคำสั่ง W = 0x93 , TMP0 = 0x13

IORWF TMP0,1

ก่อนการกระทำคำสั่ง W = 0x91 , TMP0 = 0x13

หลังการกระทำคำสั่ง W = 0x91 , TMP0 = 0x93

5.5 คำสั่ง XORLW (XOR literal with W)

เป็นคำสั่งให้ทำการเอ็กซลูซีฟออร์ค่าคงที่กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในเรจิสเตอร์ W

รูปแบบ : XORLW k

โอเปอร์เรนด์ : $0 \leq k \leq 255$

การทำงาน : (W) XOR (k) $\rightarrow$ (W)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : XORLW 0xAF

ก่อนการกระทำคำสั่ง W = 0xB5 (1011 0101)

หลังการกระทำคำสั่ง W = 0x1A (0001 1010)

5.6 คำสั่ง XORWF (XOR W with f)

เป็นคำสั่งให้ทำการเอ็กซลูซีฟออร์ค่าในเรจิสเตอร์ใดๆ กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในเรจิสเตอร์ W หรือ เรจิสเตอร์นั้น

รูปแบบ : XORWF f,d

โอเปอร์เรนด์ : $0 \leq f \leq 127, d \in [0, 1]$

การทำงาน : when d=0 (W) XOR (f) $\rightarrow$ (W)
when d=1 (W) XOR (f) $\rightarrow$ (f)

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : XORWF TMP0,0

ก่อนการกระทำคำสั่ง $W = 0xB5$, $TMP0 = 0xAF$
 หลังการกระทำคำสั่ง $W = 0x1A$, $TMP0 = 0xAF$
 XORWF TMP0,1
 ก่อนการกระทำคำสั่ง $W = 0xB5$, $TMP0 = 0xAF$
 หลังการกระทำคำสั่ง $W = 0xB5$, $TMP0 = 0x1A$

6. กลุ่มคำสั่งประมวลผลทางคณิตศาสตร์

มีด้วยกัน 4 คำสั่ง คือ ADDLW, ADDWF, SUBLW และ SUBWF ซึ่งมีรายละเอียดดังต่อไปนี้

6.1 คำสั่ง ADDLW (ADD literal with W)

เป็นคำสั่งให้ทำการบวกค่าคงที่กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในที่เรจิสเตอร์ W

รูปแบบ : ADDLW k

โอเปอเรนด์ : $0 \leq k \leq 255$

การทำงาน : $(W) \text{ ADD } (k) \rightarrow (W)$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : ADDLW 0x1D

ก่อนการกระทำคำสั่ง $W = 0x00$

หลังการกระทำคำสั่ง $W = 0x1D$

6.2 คำสั่ง ADDWF (ADD W with f)

เป็นคำสั่งให้ทำการบวกค่าในเรจิสเตอร์ใดๆ กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในที่เรจิสเตอร์ W หรือ เรจิสเตอร์นั้น

รูปแบบ : ADDWF f,d

โอเปอเรนด์ : $0 \leq f \leq 127$, $d \in [0, 1]$

การทำงาน : when $d=0$ $(W) \text{ ADD } (f) \rightarrow (W)$

when $d=1$ $(W) \text{ ADD } (f) \rightarrow (f)$

จำนวนเวิร์ด : 1 จำนวนไบต์ : 1

ตัวอย่าง : ADDWF TMP0,0

ก่อนการกระทำคำสั่ง $W = 0x17$, $TMP0 = 0xC2$

หลังกระทำคำสั่ง	$W = 0xD9$, $TMP0 = 0xC2$
ANDWF TMP0,1	
ก่อนการกระทำคำสั่ง	$W = 0x17$, $TMP0 = 0xC2$
หลังกระทำคำสั่ง	$W = 0x17$, $TMP0 = 0xD9$

6.3 คำสั่ง SUBLW (SUB literal with W)

เป็นคำสั่งให้ทำการลบค่าคงที่กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในที่เรจิสเตอร์ W

รูปแบบ :	SUBLW k
โอเปอร์เรนด์ :	$0 \leq k \leq 255$
การทำงาน :	$(W) \text{ SUB } (k) \rightarrow (W)$
จำนวนเวิร์ด :	1 จำนวนไบต์ : 1
ตัวอย่าง :	SUBLW 0x1D
ก่อนการกระทำคำสั่ง	$W = 0x00$
หลังกระทำคำสั่ง	$W = 0x1D$

6.4 คำสั่ง SUBWF (SUB W with f)

เป็นคำสั่งให้ทำการลบค่าในเรจิสเตอร์ใดๆ กับค่าในเรจิสเตอร์ W ผลลัพธ์จะถูกเก็บไว้ในที่เรจิสเตอร์ W หรือ เรจิสเตอร์นั้น

รูปแบบ :	SUBWF f,d
โอเปอร์เรนด์ :	$0 \leq f \leq 127$, $d \in [0, 1]$
การทำงาน :	when $d=0$ $(W) \text{ SUB } (f) \rightarrow (W)$ when $d=1$ $(W) \text{ SUB } (f) \rightarrow (f)$
จำนวนเวิร์ด :	1 จำนวนไบต์ : 1
ตัวอย่าง :	SUBWF TMP0,0
ก่อนการกระทำคำสั่ง	$W = 0x02$, $TMP0 = 0x03$
หลังกระทำคำสั่ง	$W = 0x01$, $TMP0 = 0x03$, $C = 1$
	SUBWF TMP0,1
ก่อนการกระทำคำสั่ง	$W = 0x02$, $TMP0 = 0x02$
หลังกระทำคำสั่ง	$W = 0x02$, $TMP0 = 0x00$, $C = 1$, $Z = 1$

ผลงานที่เกี่ยวข้องกับวิทยานิพนธ์และได้รับการตีพิมพ์

1. วัชรกร หนูทอง, บรรจง ปิยะธำรง. “การออกแบบวงจรเข้ารหัสและถอดรหัสแบบดีอีเอสด้วยภาษาวีเอชดีแอล.” วารสารคอมพิวเตอร์ ของสมาคมคอมพิวเตอร์แห่งประเทศไทย ปีที่ 22 ฉบับที่ 121 กันยายน-ตุลาคม 2539 หน้า 33-44
2. วัชรกร หนูทอง, บรรจง ปิยะธำรง. “การออกแบบวงจรเข้ารหัสและถอดรหัสแบบดีอีเอสด้วยภาษาวีเอชดีแอล.” วารสารวิศวกรรมสาร ของสมาคมวิศวกรรมสถานแห่งประเทศไทย ปีที่ 49 เล่มที่ 12 ธันวาคม 2539 หน้า 49-54
3. วัชรกร หนูทอง, บรรจง ปิยะธำรง. “การออกแบบวงจรเข้ารหัสและถอดรหัสแบบดีอีเอสด้วยภาษาวีเอชดีแอล.” วิศวกรรมสาร ฉบับวิจัยและพัฒนา ของสมาคมวิศวกรรมสถานแห่งประเทศไทย ปีที่ 7 ฉบับที่ 2 พค. 2539 หน้า 57-64
4. วัชรกร หนูทอง, บรรจง ปิยะธำรง. “การออกแบบไมโครคอนโทรลเลอร์โดยใช้ FPGA.” การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 21 วันที่ 12-13 พฤศจิกายน 2541 ณ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ถ. สุขสวัสดิ์ 48 เขตทุ่งครุ กรุงเทพมหานคร หน้า 178-181
5. วัชรกร หนูทอง, บรรจง ปิยะธำรง. “การออกแบบไมโครคอนโทรลเลอร์โดยใช้ FPGA.” วิศวกรรมอุตสาหกรรม ฉบับที่ 59 ปีที่ 5 มิถุนายน-กรกฎาคม 2542 หน้า 67-71
6. วัชรกร หนูทอง, บรรจง ปิยะธำรง. “การออกแบบและพัฒนาชิปเดี่ยว 8 บิตโดยการออกแบบจากบนลงล่าง.” วารสารวิศวกรรมสาร ของสมาคมวิศวกรรมสถานแห่งประเทศไทย ปีที่ 52 เล่มที่ 10 ตุลาคม 2542 หน้า 53-59

ประวัติผู้เขียน

นายวัชรกร หนูทอง เกิดเมื่อวันที่ 4 มีนาคม พ.ศ. 2517 ที่จังหวัดเพชรบูรณ์ สำเร็จการศึกษา วิทยาศาสตร์บัณฑิต สาขาฟิสิกส์ (วท.บ ฟิสิกส์) เกียรตินิยมอันดับ 2 จากมหาวิทยาลัยนเรศวร จังหวัดพิษณุโลก ปีการศึกษา 2537 และในปีการศึกษา 2540 ได้รับทุนการศึกษาจากมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร (C&C) และเคยทำงานเป็นผู้ช่วยวิจัยในโครงการร่วมมือพัฒนา ไมโครอิเล็กทรอนิกส์ขนาดเล็กของบริษัทอาร์เอนด์ดีซิสเต็มจำกัดร่วมกับสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และเคยได้รับรางวัลบทความดีเด่นในการประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 21 ณ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี พ.ศ 2541 และปัจจุบันทำงานในตำแหน่งผู้ช่วยนักวิจัยของหน่วยปฏิบัติการวิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ (TMEC) ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ กระทรวงวิทยาศาสตร์ เทคโนโลยีและสิ่งแวดล้อม