รายงานวิจัยฉบับสมบูรณ์

โครงการเทคนิคที่มีประสิทธิภาพสำหรับการสังเคราะห์ระดับสถาปัตยกรรมสำหรับระบบเรียลไทม์ VLSI

โดย นางสาวจันทนา จันทราพรชัย และคณะ

30 มิถุนายน พ.ศ. 2548

รายงานวิจัยฉบับสมบูรณ์

โครงการเทคนิคที่มีประสิทธิภาพสำหรับการสังเคราะห์ระดับสถาปัตยกรรมสำหรับระบบเรียลไทม์ VLSI

โดย นางสาวจันทนา จันทราพรชัย และคณะ

30 มิถุนายน พ.ศ. 2548

รายงานวิจัยฉบับสมบูรณ์

โครงการ เทคนิคที่มีประสิทธิภาพสำหรับการสังเคราะห์ระดับสถาปัตยกรรมสำหรับระบบเรียลไทม์ VLSI

นางสาวจันทนา จันทราพรชัย มหาวิทยาลัยศิลปากร

ศ. ดร. วัลลภ สุระกำพลธร สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สนับสนุนโดยทบวงมหาวิทยาลัย และสำนักงานกองทุนสนับสนุนการวิจัย

(ความเห็นในรายงานนี้เป็นของผู้วิจัย ทบวงฯ และ สกว. ไม่จำเป็นต้องเห็นด้วยเสมอไป)

บทคัดย่อ

ในงานวิจัยนี้ได้นำเสนอ Framework สำหรับวิธีการสำรวจการออกแบบที่เหมาะสมซึ่งพิจารณาปัจจัยความไม่แน่นอน (Impreciseness) ในข้อกำหนดของการออกแบบ ในการสังเคราะห์วงจรระดับสถาปัตยกรรมนั้น มักจะมีความไม่แน่นอนเกี่ยวกับข้อมูลของข้อกำหนด ในงานวิจัยนี้ ผู้วิจัยได้พิจารณาประเภทของความไม่แน่นอนของประเภท ได้แก่ ความไม่แน่นอนในด้านข้อกำหนดของหน่วยคำนวณ และความไม่แน่นอนในข้อกำหนดด้านเงื่อนไขด้านเวลาและเงื่อนไขด้านจำนวนรีจิสเตอร์ ใน Framework นี้ จะอ้างอิงรูปแบบ iterative และอ้างอิงกับวิธีการจัดลำดับอันได้แก่ Register-Constrained Inclusion Scheduling ในบทความนี้ได้นำเสนอตัวอย่างถึงวิธีการทำงานของการจัดลำดับแบบนี้ และได้ทดลองการใช้ Framework การออกแบบนี้กับตัวอย่าง benchmark อันได้แก่ Discrete Cosine Transform และ Voltera Filter แนวทางการออกแบบที่เลือกสำหรับการออกแบบทั้งสองนี้ได้เหมาะสมกับระดับการยอมรับได้ (Acceptability Criteria) และประหยัดจำนวนรีจีสเตอร์ทั้งหมดโดยประมาณด้วย

คำสำคัญ การสำรวจการออกแบบโดยอาศัยความไม่แน่นอน การจัดลำดับ คุณลักษณะการออกแบบหลายปัจจัย ข้อมูลความไม่แน่นอน เงื่อนไขด้านเวลา และ Inclusion Scheduling

Keywords Imprecise Design Exploration, Scheduling/Allocation, Multiple Design Attribute, Imprecise Information, Register Constraint, Inclusion Scheduling

Abstract

We propose a design exploration framework which consider impreciseness in design speciÞcation. In high-level synthesis, imprecise information is often encountered. We consider two types of imprecesness: impreciseness underlying on functional unit speciÞcations and on contraints: latency and register. The framework is iterative and based on a core scheduling called, *Register-Constrained Inclusion Scheduling*. An example how the scheduling algorithm work is shown. We demonstrate the effectiveness of our framework for imprecise speciÞcation by exploring a design solution for a well-known benchmark, *Discrete Cosine Transform*, and *Voltera Filter*. The selected solution meets the acceptability criteria while minimizing the total number of registers.

สัญญาเลขที่ MRG4680115

โครงการเทคนิคที่มีประสิทธิภาพสำหรับการสังเคราะห์ระดับสถาปัตยกรรมสำหรับระบบเรียลไทม์ VLSI

แบบฟอร์มรายงานโครงการวิจัยประจำปี

ชื่อโครงการ เทคนิคที่มีประสิทธิภาพสำหรับการสังเคราะห์ระดับสถาปัตยกรรมสำหรับระบบเรียลไทม์ VLSI

ระยะเวลาโครงการ 1 กรกฎาคม 2546- 30 มิถุนายน 2548

ชื่อหัวหน้าโครงการผู้ได้รับทุน นางสาวจันทนา จันทราพรชัย

ชื่อนักวิจัยที่ปรึกษา ศ. ดร. วัลลภ สุระกำพลธร

รายงานช่วงวันที่ 1 มกราคม 2548 ถึง 30 มิถุนายน 2548

1. สำหรับหัวหน้าโครงการผู้ได้รับทุน

1.1 การดำเนินงาน

☑ * ได้ดำเนินการตามแผนที่วางไว้

☐ ได้ดำเนินการงำช้ากว่าแผนที่วางไว้

☐ ได้เปลี่ยนแผนงานที่วางไว้ดังนี้

1.2 รายละเอียดผลการดำเนินงานของโครงการ

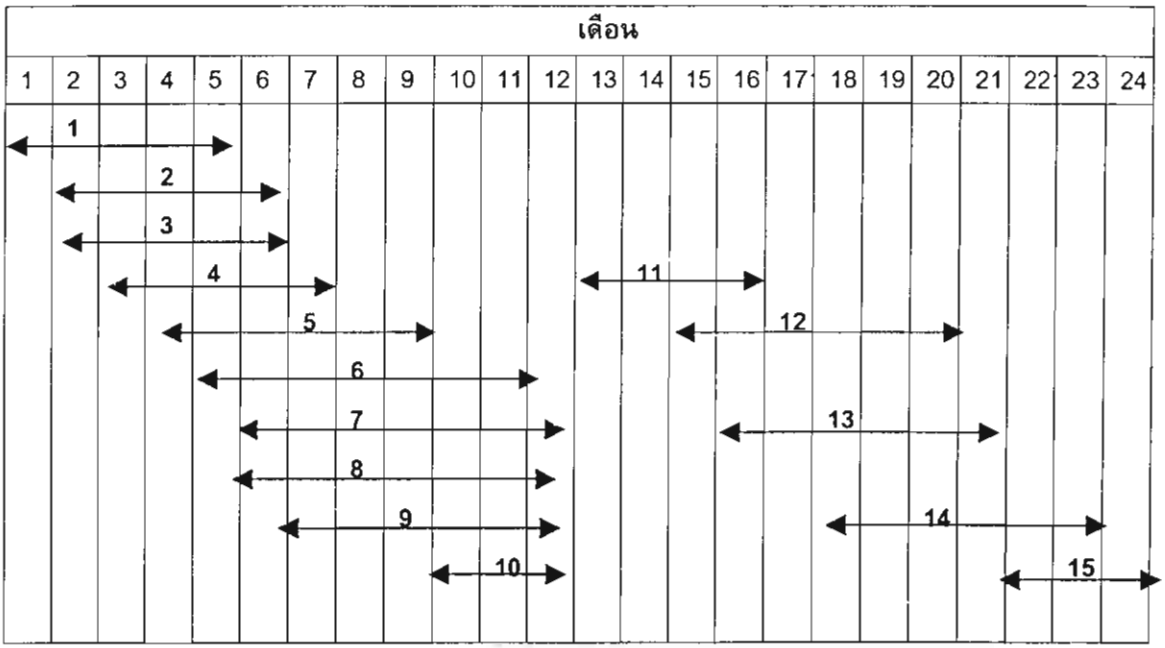
1. สรุปย่อ (Summary)

<u>วัตถุประสงค์</u>

1. เพื่อเป็นการสร้างนักวิจัยที่มีความรู้ความสามารถในทางเทคนิคการออกแบบวงจร VLSI
2. เพื่อเป็นการพัฒนาความรู้ที่เป็นพื้นฐานในการพัฒนาซอฟต์แวร์สำหรับการสังเคราะห์วงจร VLSI แบบอัตโนมัติ
3. เพื่อเป็นการศึกษาเทคนิคที่อยู่ในปัจจุบันในการออกแบบวงจร VLSI และวิเคราะห์ข้อดีเสีย
4. เพื่อเป็นการพัฒนาเทคนิคใหม่เพื่อช่วยในการออกแบบวงจร VLSI ที่พิจารณาคุณลักษณะของวงจรหลายอย่างพร้อมๆกัน และปัจจัยความไม่แน่นอน
5. เพื่อเป็นการนำเทคนิคไปใช้ในการสังเคราะห์วงจร VLSI แบบอัตโนมัติ ในการสังเคราะห์วงจรสำหรับอุปกรณ์ทางอิเล็กทรอนิกส์ต่างๆ

<u>การดำเนินงานวิจัย</u>

กิจกรรมที่ได้วางแผนไว้



1. ศึกษาเทคนิคอัตโนมัติในการสังเคราะห์วงจร ในด้าน scheduling, allocation, module selection ที่ได้ถูกพัฒนาขึ้นมาก่อน และวิเคราะห์ข้อดีเสียของเทคนิคเหล่านี้
2. ศึกษาปัจจัยความไม่แน่นอนในการสังเคราะห์วงจรชั้นต้น
3. ศึกษาปัจจัยคุณลักษณะของวงจรผลลัพธ์ ในแง่ของเวลา ขนาด การใช้พลังงาน รวมถึงความสัมพันธ์ระหว่างกันและกัน
4. ปรับปรุง model สำหรับการวิเคราะห์คุณลักษณะของวงจรผลลัพธ์ที่มีความไม่แน่นอนรวมอยู่ด้วย
5. พัฒนา หรือปรับปรุงเทคนิคในการสังเคราะห์วงจร VLSI ตามแบบ model ที่วางไว้ ที่พิจารณา scheduling, allocation พร้อมกันก่อน สำหรับ Functional และ Register allocation
6. พัฒนา ซอฟต์แวร์เพื่อจำลองการทำงานของเทคนิคดังกล่าว
7. เลือก Benchmark หรือตัวอย่าง Application ที่ใช้อยู่ทั่วไป เพื่อการทดลองเปรียบเทียบการทำงานของเทคนิคใหม่ที่พัฒนาขึ้นและเทคนิคเก่า
8. ทดลองประสิทธิผลของเทคนิคที่คิดค้นขึ้นจากโปรแกรมจำลองที่พัฒนาขึ้น
9. วิเคราะห์ผลการทดลองและศึกษาแนวทางการนำเทคนิคไปใช้ในการออกแบบจริง
10. เขียนรายงานสรุปผลการวิจัย เพื่อนำเสนอผลงานในวารสารทางวิชาการในสาขาที่เกี่ยวข้อง
11. ศึกษาวิธีการ Design Exploration ที่มีอยู่
12. นำเทคนิคข้อ 5 มาเป็นสร้างเป็น Design Framework ใหม่ที่มีปัจจัยความไม่แน่นอนมากเกี่ยวข้อง โดยอาศัยความรู้ข้อ 11
13. แก้ไข และพัฒนาแบบจำลองการทำงานเพื่อหา Design Solution ที่ได้จากวิธีใหม่นี้
14. วัดประสิทธิผลของเทคนิคที่มีอยู่ และ เทคนิคใหม่ เปรียบเทียบกัน
15. เขียนรายงานสรุปผลการวิจัยทั้งหมด เพื่อประกอบการนำเสนอในการประชุมทางวิชาการ หรือตีพิมพ์ในวารสารทางวิชาการ

กิจกรรมที่ได้ทำจริง

ได้ดำเนินการกิจกรรมที่ 15 เสร็จสิ้นแล้ว

สรุปการดำเนินงาน

1. ได้ทำการศึกษาค้นคว้างานวิจัยที่เกี่ยวข้องกับวิธีการทำ Register Allocation ที่ใกล้เคียงกัน และทบทวนวิธีการ Inclusion Scheduling ที่จะใช้สำหรับทำ Resource binding และ Allocation เมื่อพิจารณาปัจจัยความไม่แน่นอนมาเกี่ยวข้องด้วยการใช้ Fuzzy Set โดยละเอียด และพิจารณาข้อเด่นของการนำ Register constraint ระหว่างการทำ Scheduling/Allocation เมื่อมีปัจจัยความไม่แน่นอน
2. วิเคราะห์ปัญหาที่เกิดขึ้นและนิยามปัญหาเมื่อต้องการพิจารณาปัจจัย Register constraint ในสภาวะที่มีความไม่แน่นอน ด้วยฟัซซีเซท
3. ได้ทำการปรับปรุงเทคนิคเพื่อรองรับปัจจัยของความไม่แน่นอนในส่วนของริจีสเตอร์แล้ว
4. ได้ปรับปรุงและพัฒนาซอฟต์แวร์ตามข้อ 3 เรียบร้อยแล้ว
5. ได้ทำการทดลองกับ Benchmark เบื้องต้นแล้ว และได้ผลดีกว่าแบบดั้งเดิมที่ไม่ได้พิจารณาปัจจัย register ถึง 35%
6. ได้ปรับปรุงและพัฒนา Design Exploration Framework แบบ iterative ซึ่งนำปัจจัยความไม่แน่นอนด้าน Register Constraint มาเกี่ยวข้อง
7. ได้ทำการทดลองกับ Benchmark และเก็บข้อมูลเพิ่มเติมสำหรับ Design Exploration
8. ได้เขียนสรุปรายงาน **และส่ง manuscript ไปยัง IEEE Transactions on Computer-Aided Design (Impact Factor 0.782) เมื่อวันที่ 30 เมษายน 2548**

3. ผลงานวิจัยที่ตีพิมพ์ในวารสารวิชาการระดับนานาชาติ

- ได้ตีพิมพ์เรื่อง Efficient Scheduling for Design Exploration with Imprecise Latency and Register Constraints ใน Lecture Notes in Computer Science (LNCS 3207) ปี 2004 หน้า 259-**270 impact factor 0.413**

- ได้ตีพิมพ์เรื่อง Design Exploration Framdwork under Impreciseness based on Inclusion Scheduling ใน Lecture Notes in Computer Science (LNCS 3321) ปี 2004 หน้า 78-93 **impact factor 0.413**

4. กิจกรรมอื่นๆที่เกี่ยวข้อง

- ได้นำเสนอผลงานทางวิชาการเรื่อง Efficient Scheduling for Design Exploration with Imprecise Latency and Register Constraints ในการประชุมวิชาการ The 2004 International Conference on Embedded And Ubiquitous Computing (EUC04) ที่ University of Aizu ประเทศญี่ปุ่น ในวันที่ 25-27 สิงหาคม 2547
- ได้นำเสนอผลงานทางวิชาการเรื่อง Register-Constrained Inclusion Scheduling for Imprecise Specification ในงานประชุม IEEE Analog and Digital Techniques in Electrical Engineering ปี 2004 ณ โรงแรมโลตัสปางสวนแก้ว เชียงใหม่ วันที่ 20-24 พฤศจิกายน 2547 และลงตีพิมพ์ใน Proceeding ของงานประชุม หน้า 160-163
- ได้นำเสนอผลงานทางวิชาการเรื่อง Design Exploration Framdwork under Impreciseness based on Inclusion Scheduling ในงานประชุมวิชาการ 9th ASIAN Computing Science Conference ณ มหาวิทยาลัยเชียงใหม่ ในวันที่ 10-12 ธันวาคม 2548

5. ปัญหาและอุปสรรค

-

6. ความเห็นและข้อเสนอแนะ

-

7. งานที่จะทำในปีต่อไป

-

2. สำหรับนักวิจัยที่ปรึกษา

จะส่งแบบประเมินตามมาภายหลัง

ลงนาม_____

(หัวหน้าโครงการผู้ได้รับทุน)

บทคัดย่อ

ในงานวิจัยนี้ได้นำเสนอ Framework สำหรับวิธีการสำรวจการออกแบบที่เหมาะสมซึ่งพิจารณาปัจจัยความไม่แน่นอน (Impreciseness) ในข้อกำหนดของการออกแบบ ในการสังเคราะห์วงจรระดับสถาปัตยกรรมนั้น มักจะมีความไม่แน่นอนเกี่ยวกับข้อมูลของข้อกำหนด ในงานวิจัยนี้ ผู้วิจัยได้พิจารณาประเภทของความไม่แน่นอนของประเภท ได้แก่ ความไม่แน่นอนในด้านข้อกำหนดของหน่วยคำนวณ และความไม่แน่นอนในข้อกำหนดด้านเงื่อนไขด้านเวลาและเงื่อนไขด้านจำนวนรีจิสเตอร์ ใน Framework นี้ จะอ้างอิงรูปแบบ iterative และอ้างอิงกับวิธีการจัดลำดับอันได้แก่ Register-Constrained Inclusion Scheduling ในบทความนี้ได้นำเสนอตัวอย่างถึงวิธีการทำงานของการจัดลำดับแบบนี้ และได้ทดลองการใช้ Framework การออกแบบนี้กับตัวอย่าง benchmark อันได้แก่ Discrete Cosine Transform และ Voltera Filter แนวทางการออกแบบที่เลือกสำหรับการออกแบบทั้งสองนี้ได้เหมาะสมกับระดับการยอมรับได้ (Acceptability Criteria) และประหยัดจำนวนรีจีสเตอร์ทั้งหมดโดยประมาณด้วย

คำสำคัญ การสำรวจการออกแบบโดยอาศัยความไม่แน่นอน การจัดลำดับ คุณลักษณะการออกแบบหลายปัจจัย ข้อมูลความไม่แน่นอน เงื่อนไขด้านเวลา และ Inclusion Scheduling

Keywords Imprecise Design Exploration, Scheduling/Allocation, Multiple Design Attribute, Imprecise Information, Register Constraint, Inclusion Scheduling

สารบัญ

บทคัดย่อ 1
บทที่ 1 บทนำ 4
บทที่ 2 งานวิจัยที่เกี่ยวข้อง 6
บทที่ 3 Model และความรู้เกี่ยวกับทฤษฎีฟัซซี่เซต 8
3.1 การ model ระบบและคุณลักษณะของระบบ 8
3.2 ทฤษฎีฟัซซี่เซต 11
บทที่ 4 Iterative Design Framework 13
บทที่ 5 Register-Constrained Inclusion Scheduling 14
5.1 Latency-based Inclusion Scheduling 14
5.2 อัลกอริทึม Register-Constrained Inclusion Scheduling 16
5.2.1 Imprecise Timing Attribute 16
5.2.2 การคำนวณการใช้งานรีจิสเตอร์ 18
5.2.3 อัลกอริทึม 19
บทที่ 6 ตัวอย่างการทำงานของ Register-Constrained Inclusion Scheduling 22
บทที่ 7 ผลการทดลอง 26
7.1 Discrete Cosine Transform 26
บทที่ 8 สรุปผล 30
เอกสารอ้างอิง 31

สารบัญรูปและตาราง

รูปที่ 3.1 โค้ดและตัวอย่าง DFG 8

รูปที่ 3.2 การพล็อตของฟังก์ชัน [illegible] 9

รูปที่ 3.3 Projection ของรูป 3.2 10

รูปที่ 3.4 (a) Z-Shaped acceptability function (b) projection 10

รูปที่ 3.5 การบวกตัวเลขฟัซซี่ A+B 11

รูปที่ 4.1 การหา design solution แบบ iterative 13

รูปที่ 5.1 ค่า FST(u) และ FFT(u) สองกรณี 17

รูปที่ 5.2 ความสัมพันธ์ระหว่างโหนดที่ถูกจัดลำดับแล้วและ life time 19

รูปที่ 6.1 กราฟตัวอย่าง 22

ตารางที่ 6.1 แสดงคุณลักษณะของหน่วยคำนวณ 22

รูปที่ 6.2 System Specification สำหรับกราฟตัวอย่าง 23

รูปที่ 6.3 (a) ตารางจัดลำดับที่ได้จาก RCIS (b) ตารางจัดลำดับที่ได้ inclusion scheduling 23

รูปที่ 6.4 (a) FLT (A) (b) FLT(B) 23

รูปที่ 6.5 FLT (A) และ FLT(B) 24

รูปที่ 6.6 FLT สำหรับทุกโหนดในกราฟ (a) FST (b) MFFT 24

รูปที่ 6.7 Register count และค่าความเป็นไปได้ ณ เวลาต่าง ๆ 25

รูปที่ 6.8 กราฟใหม่เมื่อเติมโหนดเข้าไป 25

ตารางที่ 7.1 ลักษณะของ adder และ multiplier 26

ตารางที่ 7.2 ผลการทดลองของ DCT เปรียบเทียบ RCIS และ IS สำหรับจำนวนหน่วยคำนวณต่างๆ กัน 27

รูปที่ 7.1 system specification ของ DCT 27

ตารางที่ 7.3 ค่าความเป็นไปได้สำหรับการใช้งานจำนวนรีจิสเตอร์ต่าง ๆ กัน สำหรับ กรณี adder 7ตัว และ multiplier 5 ตัว ของ RCIS 27

ตารางที่ 7.4 ค่าความเป็นไปได้สำหรับการใช้งานจำนวนรีจิสเตอร์ต่าง ๆ กัน สำหรับ กรณี adder 7ตัว และ multiplier 5 ตัว ของ IS 27

รูปที่ 7.2 System Specification ของ Voltera filter 28

ตารางที่ 7.5 ผลการทดลองของ Voltera filter เปรียบเทียบ RCIS และ IS สำหรับจำนวนหน่วยคำนวณต่าง ๆ กัน 29

รูปที่ 7.3 ค่า acceptability degree ของแต่ละการออกแบบ 29

บทที่ 1 บทนำ

ในการสังเคราะห์วงจรระดับสถาปัตยกรรม มักมีความไม่แน่นอนในข้อมูลต่างๆ ได้แก่ ในแง่ของการ implement การเลือกองค์ประกอบ (component) ของวงจรสำหรับการออกแบบ (design) อาจจะยังไม่สามารถกำหนดได้เนื่องจากหลายสาเหตุ อาจจะเป็นเพราะมีหลายประเภทขององค์ประกอบให้เลือก เช่น มี module ที่ทำหน้าที่เป็น ตัวคูณ (multiplier) หลายประเภท หรืออาจจะเป็นเพราะ module บาง module ยังไม่ได้ถูกออกแบบในระดับ physical ทำให้คุณลักษณะบางอย่างของ module นั้นยังไม่แน่นอน หรือแม้ว่า module นั้นได้ถูกออกแบบเรียบร้อยแล้ว คุณลักษณะบางอย่างอาจจะมีการเปลี่ยนแปลงเนื่องจากขั้นตอนการ fabricate ก็ได้ อีกประเภทหนึ่งของความไม่แน่นอนได้แก่ ความไม่แน่นอนหรือความกำกวมในด้านระดับการยอมรับได้ (acceptability level) หรือระดับความพึงพอใจของการออกแบบที่ได้ในระดับสถาปัตยกรรม เช่นถ้ามีความพึงพอใจกับการออกแบบที่สามารถทำงานได้ภายในเวลา 50 cycle และถ้าการออกแบบที่ได้ทำงานได้ในเวลา 51 cycle จะยังสามารถยอมรับได้หรือไม่ โดยเฉพาะอย่างยิ่งเมื่อมีปัจจัยการออกแบบหลายปัจจัยมาเกี่ยวข้องซึ่งปัจจัยเหล่านั้นอาจจะขัดแย้งกันอยู่ในตัวเอง เช่นในแง่ประสิทธิภาพและการใช้พลังงาน ประสิทธิภาพความเร็วสูงอาจจะไม่ประหยัดพลังงาน หรือ ในแง่ของจำนวนรีจิสเตอร์ที่ใช้กับเวลาในการทำงานของการออกแบบที่ได้ ถ้าวงจรทำงานเร็วอาจจะใช้จำนวนรีจิสเตอร์มากขึ้น จะยอมรับได้หรือไม่ ถ้าวงจรนั้นทำงานช้าลง 1 หรือ 2 clock cycle แต่ประหยัดรีจิสเตอร์ได้ 1 ตัว หรือ ถ้าใช้เวลามากขึ้นกว่าเดิม 10 clock cycle จะยังยอมรับได้หรือไม่ มากน้อยเท่าไร ดังนั้นการพิจารณาปัจจัยความไม่แน่นอนอย่างเหมาะสมในการสังเคราะห์วงจรรวมระดับสถาปัตยกรรมจะมีผลต่อการออกแบบที่ได้ด้วย

ในงานวิจัยนี้ ได้มีการนำเสนอ framework สำหรับการสำรวจการออกแบบที่เหมาะสม ซึ่งพิจารณาปัจจัยความไม่แน่นอนทั้งในด้านข้อกำหนดของระบบผลลัพธ์และเงื่อนไข (requirement) ด้านเวลา (latency) และการใช้งานรีจิสเตอร์ อย่างไรก็ดีวิธีการดังกล่าวสามารถขยายต่อไปเพื่อพิจารณาความไม่แน่นอนในเงื่อนไขหลายๆ ด้านในการออกแบบด้วย ลักษณะข้อกำหนดของระบบถูก model อ้างอิงกับทฤษฎีฟัซซี่เซต ในงานวิจัยนี้จะพิจารณาจำนวนรีจิสเตอร์ที่ใช้เป็นอีกมิติหนึ่งของเงื่อนไขของระบบงาน ผู้วิจัยได้คิดค้นวิธีการจัดลำดับ (scheduling) เมื่อพิจารณาปัจจัยด้านเวลา[3,7] เพื่อใช้เป็นอัลกอริทึมหลักใน iterative design framework นี้ ตารางจัดลำดับที่ได้จะพยายามลดการใช้งานในแง่ของจำนวนริจิสเตอร์ ถ้าตารางจัดลำดับนั้นยอมรับได้ ก็จะได้ผลลัพธ์การออกแบบ ถ้าไม่เช่นนั้น จะมีการปรับจำนวนทรัพยากร หรือหน่วยคำนวณและทดลองจัดลำดับด้วยวิธีดังกล่าวใหม่ จนกว่าได้ผลลัพธ์การออกแบบที่ยอมรับได้ ข้อมูลเข้าของ framework นี้ได้แก่ data flow graph ซึ่งประกอบด้วยพารามิเตอร์ด้านเวลาที่มีความไม่แน่นอนอยู่ งานประยุกต์ของระบบประเภทดังกล่าวได้แก่ ระบบ digital signal processing หรือ communication switch และ งานประเภท real-time multimedia rendering เป็นต้น ข้อกำหนดที่ไม่แน่นอนทั้งในด้านพารามิเตอร์และเงื่อนไขนั้นจะมีผลอย่างมากต่อการจัด

สรรทรัพยาการ (resource allocation) และการจัดลำดับสำหรับการออกแบบงานประยุกต์เหล่านี้ ดังนั้น การพัฒนาเทคนิคการสังเคราะห์และเทคนิคการ optimize วงจรรวม ซึ่งพิจารณาปัจจัยความไม่แน่นอนเหล่านี้จึงมีความสำคัญอย่างยิ่ง

ในการสังเคราะห์วงจรและเครื่องมือต่างๆ ที่มีอยู่โดยทั่วไปจะไม่พิจารณาปัจจัยความไม่แน่นอนในข้อกำหนดดังกล่าว ส่วนมากจะสมมติว่าข้อมูลต่างๆ ที่ได้จะได้จากพิจารณากรณี worst case หรือกรณีทั่วไป (typical case) แล้ว เช่นพิจารณาเวลาในการทำงานของ module ที่ใช้ในแบบ worst case เงื่อนไขด้านเวลามักจะกำหนดมาในรูปแบบของค่าคงที่ (fixed value) ซึ่งแม้ว่าในความเป็นจริงอาจจะยืดหยุ่นได้บ้างเนื่องจากระดับความพึงพอใจของนักออกแบบแต่ละคนมักจะแตกต่างกัน การตั้งสมมติฐานแบบดังกล่าวอาจจะเป็นแหล่งกำเนิดทำให้การต้องมีการปรับการออกแบบหลายครั้งกว่าจะได้ผลลัพธ์ หรืออาจจะนำมาซึ่งผลลัพธ์ของการออกแบบที่ใช้ทรัพยากรต่างๆ มากเกินความจำเป็น การพิจารณาปัจจัยความไม่แน่อย่างเหมาะสมตั้งแต่ขั้นแรกๆ ของการออกแบบจะทำให้ได้การออกแบบเริ่มต้นที่ดี และทำให้การปรับการออกแบบทำได้เร็วกว่า

การใช้ตัวแปรสุ่มและความน่าจะเป็นเป็นอีกแนวทางหนึ่งซึ่งใช้ model ความไม่แน่นอน อย่างไรก็ดี การเก็บข้อมูลเกี่ยวกับความน่าจะเป็น บางครั้งก็ยากลำบากและใช้เวลาในการเก็บข้อมูล นอกจากนี้ความไม่แน่นอนบางอย่างเช่นระดับความพึงพอใจของการออกแบบที่ได้ก็ไม่สามารถ model ได้ด้วยความน่าจะเป็น

ในงานวิจัยนี้ ผู้วิจัยได้พิจารณาทั้งปัจจัยความไม่แน่นอนด้านเวลาและการใช้งานรีจิสเตอร์ ผู้วิจัยได้พัฒนา framework สำหรับการสำรวจการออกแบบที่พิจารณาความไม่แน่นอนในข้อกำหนดคุณลักษณะของระบบผลลัพธ์และเงื่อนไข framework ดังกล่าวเป็นแบบ iterative ซึ่งอาศัยวิธีการจัดลำดับที่ได้พัฒนาขึ้น ที่เรียกว่า RCIS (Register-Constrained Inclusion Scheduling) ซึ่งพิจารณาข้อมูลที่มีความไม่แน่นอน จากการทดลองพบว่าผู้วิจัยสามารถหาคำตอบสำหรับการออกแบบที่ลดจำนวนรีจิสเตอร์ได้

ในรายงานนี้ได้แบ่งออกเป็นบทต่างๆ ดังนี้

-บทที่ 2 นำเสนองานวิจัยที่เกี่ยวข้อง

-บทที่ 3 อธิบายถึง model ที่ใช้ในงานวิจัยทั้ง data flow graph และ model คุณลักษณะที่ไม่แน่นอน และนำเสนอความรู้พื้นฐานต่างๆเกี่ยวกับทฤษฎีฟัซซี่ที่จำเป็น

-บทที่ 4 อธิบายถึง iterative design framework ที่ได้พัฒนาขึ้น

-บทที่ 5 นำเสนอวิธีการจัดลำดับแบบ RCIS ซึ่งใช้ในบทที่ 3 โดยละเอียด และ นำเสนอนอกจากนี้ได้นำเสนอประเด็นต่างๆ ที่เกี่ยวข้องเมื่อพิจารณาการใช้งานรีจิสเตอร์ และการคำนวณจำนวนรีจิสเตอร์ที่ใช้ระหว่างการจัดลำดับเมื่อมีปัจจัยความไม่แน่นอน

-บทที่ 6 นำเสนอตัวอย่างการทำงานของอัลกอริทึมดังกล่าวในบทที่ 4

-บทที่ 7 นำเสนอผลการทดลองเมื่อใช้ framework ดังกล่าวกับตัวอย่าง benchmark

-บทที่ 8 สรุปงานวิจัยและนำเสนอแนวทางการวิจัยต่อไปในอนาคต

บทที่ 2 งานวิจัยที่เกี่ยวข้อง

ได้มีนักวิจัยหลายๆ ท่านได้ใช้ตรรกะแบบฟัซซี่เพื่อแก้ปัญญาการจัดลำดับในงานประยุกต์ต่างๆ กัน เช่น ฟิลด์ compiler optimization ได้มีการใช้ทฤษฎีฟัซซี่เซตในการพิจารณา real-time event และ ความไม่แน่นอนเกี่ยวกับตัวแปร (variable) [18] Lee et. al ใช้การอนุมานแบบฟัซซีมาช่วยหาตารางจัดลำดับที่เป็นไปได้สำหรับระบบ real-time โดยให้แต่ละ task ทำได้งานตาม deadline ภายใต้เงื่อนไขการใช้ทรัพยากรที่กำหนด [23] ในงานด้าน production management ได้มีการใช้กฎฟัซซี่เพื่อทำการจัดลำดับสำหรับ job shop และ floor shop [27,33] Kaviani และ Vranesic ใช้กฎฟัซซี่เพื่อตัดสินหาจำนวนโพรเซสเซอร์ที่เหมาะสมสำหรับเซตของ task และ deadline ที่กำหนดให้สำหรับระบบ real-time [22] Soma et.al พิจารณาการทำ schedule optimization โดยใช้การอนุมานแบบฟัซซี่ [32] งานวิจัยเหล่านี้ ไม่ได้พิจารณากรณีที่ว่าข้อกำหนดด้านเวลาของแต่ละ task อาจจะไม่ใช่ค่าแน่นอน หรือไม่ได้พิจารณาคุณลักษณะอื่นๆ นอกจากด้านเวลาของตารางการจัดลำดับ

งานวิจัยอื่นๆ เกี่ยวกับการสำรวจการออกแบบวงจร (design exploration) ได้แก่ [1,9,15,26] งานเหล่านี้แตกต่างกันในด้านเทคนิคที่ใช้ในการหาผลลัพธ์การออกแบบและการตัดสินใจเลือกผลลัพธ์การออกแบบ อย่างไรก็ดีงานเหล่านี้ไม่ได้พิจารณาปัจจัยความไม่แน่นอนในคุณลักษณะของระบบเช่น เงื่อนไขด้านเวลา และความไม่แน่นอนเกี่ยวกับเวลาที่ใช้ในการรันงานในหน่วยคำนวณ Karkowski และ Otten ได้เสนอ model ที่ใช้ในการจัดการความไม่แน่นอนด้านเวลาของหน่วยคำนวณโดยใช้ทฤษฎีฟัซซี่เซต วิธีการที่นำเสนอได้อ้างอิงวิธีการ possibilistic programming โดยอาศัย integer linear programming (ILP) เพื่อหาตารางจัดลำดับและการจัดสรรทรัพยากรหน่วยคำนวณโดยภายใต้เงื่อนไขด้านเวลาและพื้นที่ โดยเงื่อนไขก็ได้ใช้ฟัซซี่เซต model เช่นกัน อย่างไรก็ดีการใช้ ILP กับ fuzzy constraint และ fuzzy coefficient นั้น ใช้เวลาในการประมวลผลมาก นอกจากนี้งานวิจัยนี้ไม่ได้พิจารณาถึงระดับความพึงพอใจที่เกิดขึ้นสำหรับผลลัพธ์การออกแบบที่ได้ ได้มีงานวิจัยหลายงานเกี่ยวกับการประมาณการการใช้ทรัพยากร (resource estimation) [10,28,31] ซึ่งงานเหล่านี้ก็ไม่ได้พิจารณาคุณลักษณะของการแบบหลายปัจจัย หรือไม่ได้พิจารณาถึงความไม่แน่นอนในคุณลักษณะของระบบผลลัพธ์

งานวิจัยเกี่ยวกับการจัดลำดับและการจัดสรรการใช้รีจิสเตอร์มีอีกมากมายใน area ของการสังเคราะห์วงจรระดับสถาปัตยกรรม และ compiler optimization สำหรับสถาปัตยกรรมของ VLIW เช่น Varatkar et. al . ได้นำเสนออัลกอริทึมการจัดลำดับสำหรับระบบมัลติโพรเซสเซอร์ซึ่งพิจารณาการประหยัดพลังงานทั้งหมด [34] Shao et. al. ได้นำเสนอการจัดลำดับสำหรับระดับคำสั่ง (instruction level scheduling) สำหรับงานประยุกต์ที่มีลูปและพิจารณาประหยัด

พลังงานโดยการลด switching activity [30] Chen et. al. ได้เสนอการจัดลำดับสำหรับงานประยุกต์ที่มีลูปสำหรับการ optimize ระบบในด้านเวลาและจำนวน memory operation ภายใต้เงื่อนไขข้อจำกัดด้านจำนวนรีจิสเตอร์ [16] เทคนิคดังกล่าวใช้เทคนิคของ multidimensional retiming Eichenerger et. al. เสนอวิธีการจัดสรรทรัพยากรรีจิสเตอร์สำหรับสถาปัตยกรรมแบบ VLIW และ superscalar โดยใช้ stage scheduling [13,14] Akturan และ Jacome นำเสนออัลกอริทึมการจัดลำดับที่พิจารณาการใช้จำนวนรีจิสเตอร์อย่างประหยัดโดยอาศัยเทคนิค software pipelining [2] อัลกอริทึมนี้ใช้เทคนิค retiming และ force directed scheduling มาช่วย และพิจารณาข้อดีข้อเสียระหว่างขนาดของโค้ดผลลัพธ์ ประสิทธิภาพการทำงานของโค้ดผลลัพธ์ และ การใช้รีจิสเตอร์ของโค้ดนั้นๆ Wong et. al. ได้พัฒนาวิธีการแทรก objective function เข้าไประหว่างการจัดลำดับและการจัดสรรการใช้ทรัพยากร [35] อัลกอริทึมดังกล่าวชื่อย่อว่า FLOF จะพยายามลดจำนวนการใช้งานรีจิสเตอร์ภายใต้เงื่อนไขด้านเวลาและจำนวนทรัพยากรที่กำหนดให้ Dani et. al. ได้นำเสนอ heuristic ในการใช้ stage scheduling เพื่อลดจำนวนรีจิสเตอร์ ซึ่งงานนี้ก็มุ่งไปยังการจัดลำดับสำหรับระดับคำสั่งเช่นกัน [11] Zalamea et. al. ได้นำเสนอวิธีการทางด้านฮาร์ดแวร์และซอฟต์แวร์เพื่อลดการใช้งานรีจิสเตอร์โดยเน้นไปยังสถาปัตยกรรมแบบ VLIW [24,25,38] ในส่วนของซอฟต์แวร์ งานวิจัยนี้ได้ทำการปรับปรุง modulo scheduling ให้พิจารณาเงื่อนไขด้านจำนวนรีจิสเตอร์ และ register spilling อย่างไรก็ดี งานวิจัยนี้เน้นไปยังการจัดลำดับสำหรับงานประยุกต์ที่มีลูปในโค้ดและไม่ได้พิจารณาปัจจัยความไม่แน่นอนในคุณลักษณะของระบบหรือข้อกำหนดของระบบผลลัพธ์

ในงานวิจัย [3] ได้มีการพัฒนา inclusion scheduling ซึ่งพิจารณาปัจจัยความไม่แน่นอนคุณลักษณะของระบบขึ้น อัลกอริทึมนี้ได้ถูกปรับปรุงและนำมาใช้ในการสำรวจการออกแบบภายใต้เงื่อนไขความไม่แน่นอนต่างๆ ด้าน และใช้ในการประมาณขอบเขตในแง่ของจำนวนทรัพยากร [5,6,8] อย่างไรก็ดีงานวิจัยดังกล่าวก็ไม่ได้พิจารณาปัจจัยของการใช้ทรัพยากรรีจิสเตอร์ในการจัดลำดับ

บทที่ 3 Model และความรู้เกี่ยวกับทฤษฎีฟัซซี่เซต

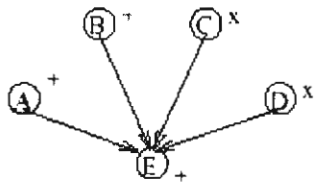
ในบทนี้จะได้อธิบายถึง model ที่สำคัญๆ ที่ใช้ในการ model ระบบและคุณลักษณะ รวมทั้งการ model ปัจจัยความไม่แน่นด้วยทฤษฎีฟัซซี่เซต และ operation ต่างๆ ของฟัซซี่เซตที่เกี่ยวข้อง

3.1 การ model ระบบและคุณลักษณะของระบบ

ในการ model ระบบและคุณลักษณะของระบบส่วนใหญ่นั้น ในระดับสถาปัตยกรรมได้อ้างอิงการใช้กราฟนั่นคือการใช้ data flow graph (DFG) $G = (V, \mathcal{E}, \beta)$ ซึ่งในที่นี้จะพิจารณาเป็นแบบ directed acyclic graph ใน data flow graph นั้น จะประกอบด้วยโหนด V , edge $\mathcal{E}$ และ weight function β บนตัวโหนด $v \in V$

operation ในโค้ดจะหมายถึงโหนดในกราฟ และความสัมพันธ์ระหว่าง operation จะหมายถึงการไหลของข้อมูลจาก operation หนึ่งไปยังอีก operation หนึ่งซึ่งจะหมายถึง directed edge ในกราฟ และ weight function จะ map จากเซตของโหนดในกราฟไปยังค่าคุณลักษณะของ operation ที่สัมพันธ์กับโหนดนั้น อันได้แก่ ประเภทของ operation ดังตัวอย่างในรูป 3.1

```
A:   R1 = R2+R3
B:   R4 = R5+R6
C:   R7 = R8 * R9
D:   R10= R11 *R12
E:   R13 = R1+R4+R7+R10
```



รูปที่ 3.1 โค้ดและตัวอย่าง DFG

ในรูปที่ 3.1 แสดงโค้ดตัวอย่างที่ประกอบด้วย operation และ DFG ที่สัมพันธ์กัน ในกราฟนี้ประกอบด้วย 5 โหนด โหนด A,B,E เป็น operation การบวกและ โหนด C,D เป็น operation การคูณ กล่าวคือ $V = \{A, B, C, D, E\}$, $\mathcal{E} = \{A \rightarrow E, B \rightarrow E, C \rightarrow E, D \rightarrow E\}$ และ $\beta(A) = \beta(B) = \beta(E) =$ add และ $\beta(C) = \beta(D) =$ multiply

operation ใน DFG จะสามารถถูกคำนวณในหน่วยคำนวณต่างๆ รูปแบบกันได้ ดังนั้นจะหมายถึงว่าผู้ออกแบบสามารถเลือกหน่วยคำนวณได้หลายรูปแบบสำหรับ operation หนึ่งๆ ซึ่งจะหมายถึงว่าในการออกแบบนั้นจะมีความไม่แน่นอนของการเลือกหน่วยคำนวณมาใช้ด้วยนั่นเอง

สำหรับเงื่อนไขสำหรับผลลัพธ์การออกแบบนั้น อาจจะได้แก่ข้อจำกัดด้านการใช้พลังงานและข้อจำกัดด้านต้นทุนการผลิต ข้อกำหนดทั้งหลายนี้จะถูก model ได้ด้วย tuple $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$ โดย $\mathcal{F}$ หมายถึง เซตของประเภทของหน่วยคำนวณที่มีอยู่ในระบบทั้งหมด เช่นอาจจะหมายถึงเซ็ตของ ตัวบวกและตัวคูณ $\mathcal{F}$ =(add, mul) และ $\mathcal{A}$ หมายถึง $\{A_f : \forall f \in \mathcal{F}\}$

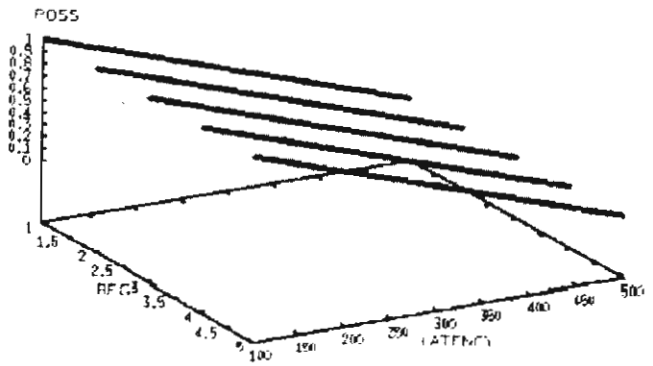
ซึ่งแต่ละ A_f คือเซตของ tuple $(a_1,\ldots,a_k)$ โดยที่ a_1 ... a_k หมายถึงคุณลักษณะของแต่ละหน่วยคำนวณ f ในที่นี้จะพิจารณาเฉพาะคุณลักษณะด้านเวลาเป็นหลัก ซึ่งจะสังเกตว่าใน model นี้สามารถขยายให้พิจารณาคุณลักษณะอื่นๆ เช่นด้านการใช้พลังงานหรือพื้นที่ของหน่วยคำนวณได้ ดังนั้น $A_f = \{x : \forall x\}$ โดยที่ x เป็นคุณลักษณะด้านเวลาของหน่วยคำนวณ f ส่วน M เป็น $\{\mu_f : \forall f \in F\}$ ซึ่ง μ_f เป็น การ map จาก A_f ไปยังเซตของจำนวนจริงในช่วง [0..1] หมายถึง possible degree ในการใช้แต่ละค่าในสมาชิกของเซต และ Q เป็นฟังก์ชันที่หมายถึง degree ที่การออกแบบนั้นจะถูกยอมรับได้สำหรับผลลัพธ์การออกแบบที่มีคุณลักษณะต่างๆ กันไป ถ้า $Q(a_1,\ldots,a_k) = 1$ หมายถึง การออกแบบนั้นเป็นพึงพอใจมามากที่สุด และ $Q(a_1,\ldots,a_k) = 0$ เมื่อ หมายถึงการออกแบบนั้นไม่ถูกยอมรับได้

การใช้ฟังก์ชัน Q ในการนิยามระดับการยอมรับได้ (acceptability level) ของระบบนั้นมีประโยชน์หลายอย่าง เช่น สามารถแสดงถึงขอบเขตเงื่อนไขของระบบผลลัพธ์ และสามารถแสดงถึง design goal ที่มีครอบคลุมถึงปัจจัยความต้องการหลายด้าน ตัวอย่างเช่น ผู้ออกแบบอาจจะสนใจระบบผลลัพธ์ที่มีทำงานได้ในเวลาไม่เกิน 500 หน่วยเวลาและจำนวนรีจิสเตอร์ที่ใช้ไม่เกิน 6 และถ้าใช้เวลาในการทำงานและจำนวนรีจิสเตอร์ยิ่งน้อยก็จะพึงพอใจมากขึ้น ระบบที่*ดีที่สุด (ideal system) อาจจะมีเวลาในการทำงานน้อยกว่า 100 หน่วยเวลา และใช้จำนวนรีจิส*เตอร์เท่ากับ 1 ความต้องการดังกล่าว model ได้ด้วยฟังก์ชัน $Q(a_1,a_2)$ ดังนี้

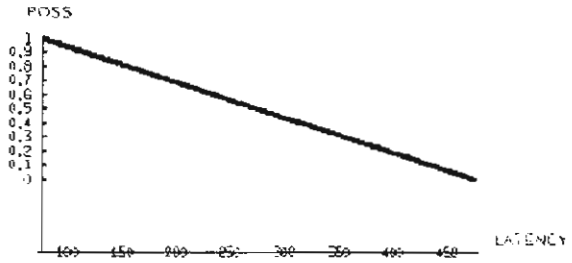
$$Q(a_1,a_2) = \begin{cases} 0 & \text{if } a_1 > 500 \text{ or } a_2 > 6 \\ 1 & \text{if } a_1 \le 100 \text{ and } a_2 \le 1 \\ F(a_1,a_2) & \text{otherwise.} \end{cases} \qquad (2.1)$$

โดยที่ F เป็น linear function และหมายถึง $F(a_1,a_2) = 1.249689(a_1 + 2a_2) - 0.001242$ ซึ่งจะให้ค่าระดับการยอมรับได้ระหว่าง [0..1]

รูปที่ 3.2 แสดงการพลอตของฟังก์ชัน Q ข้างต้น ซึ่งแสดงถึง weighted sum ของคุณลักษณะทั้งสองโดยให้ความสำคัญในการลดจำนวนรีจิสเตอร์มากกว่าการลดเวลาในการทำงานเป็น 2 เท่า (register count: latency = 1:2) อีกนัยหนึ่งคือสามารถยินยอมให้เพิ่มเวลาในการทำงาน 2 หน่วยเวลาถ้าสามารถลดจำนวนรีจิสเตอร์ได้ 1 ตัว รูปที่ 2.3 แสดง projection ของรูป 2.2 ในการ possibility และ latency (ในที่นี้บางครั้งเรียกว่า tradeoff)



รูปที่ 3.2 การพล็อตของฟังก์ชัน $F(a_1,a_2) = 1.249689(a_1 + 2a_2) - 0.001242$

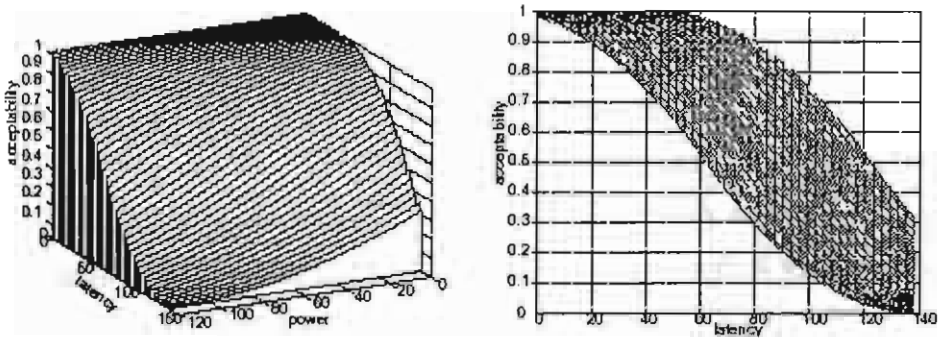


รูปที่ 3.3 Projection ของรูป 3.2

ในกรณีทั่วไปจะได้ดังสมการนี้

$$Q(a_1, a_2, \ldots a_n) = \begin{cases} 0 & \text{if } a_1 > p_{1\,max} \text{ or } a_n > p_{n\,max} \\ 1 & \text{if } a_1 \le p_{1\,min} \text{ and } a_n \le p_{n\,min} \\ F(w_1 a_1 + w_2 a_2 + \ldots w_n a_n) & \text{otherwise.} \end{cases} \quad (2.2)$$

โดยที่ผู้ออกแบบอาจจะสามารถแทนที่แกน register count ด้วยปัจจัยอื่นๆ เช่น การใช้พลังงาน (power) รูป 3.4 (a) แสดงตัวอย่างของข้อกำหนด Q เมื่อพิจารณา tradeoff ทั้งสองปัจจัยเป็น $w_1 = 2, w_2 = 1$ โดย F หมายถึงฟังก์ชัน curve รูปร่าง Z-shaped ในรูป 3.4 (b) แสดง projection ของรูป 3.4 (a) โดยแต่ละ Z-curve จะหมายถึงแต่ละ projection ของแต่ละฟังก์ชัน Q ในระนาบ acceptability และ possibility Curve ด้านในสุดจะหมายถึงเวลาในการทำงานที่น้อยที่สุดและจะมีค่าการใช้พลังงานสูงสุด ดังนั้นถ้าการออกแบบที่มีค่าระดับการยอมรับได้มากจะหมายถึงการออกแบบที่ optimize ตามเป้าหมายของการออกแบบมากนั่นเอง



(a) (b)

รูปที่ 3.4 (a) Z-Shaped acceptability function (b) projection

จาก model ข้างต้น ในงานวิจัยนี้ได้พิจารณาปัญหาการจัดลำดับและการจัดสรรทรัพยากรดังต่อไปนี้

กำหนดข้อกำหนด $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$ และกราฟ $G = (V, E, B)$ และ α ระดับการยอมรับได้ α หาตารางการจัดลำดับภายใต้จำนวนหน่วยคำนวณและจำนวนรีจิสเตอร์ที่กำหนด สำหรับแต่ละ f ใน $\mathcal{F}$ ซึ่งมีระดับการยอมรับได้มากกว่าหรือเท่ากับ α ภายใต้ Q

3.2 ทฤษฎีฟัซซี่เซต

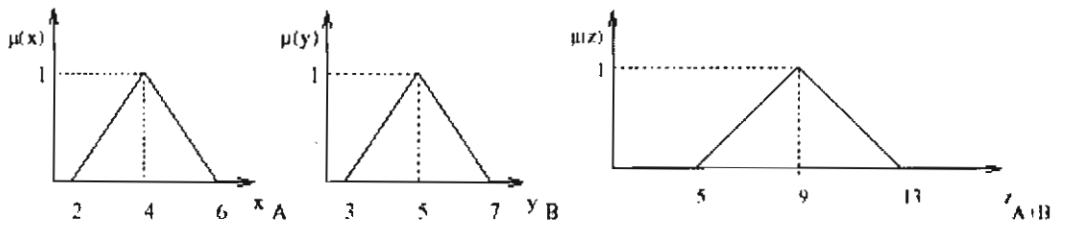
ฟัซซี่เซตกำเนิดโดย Zadeh เป็นเซตที่มีขอบเขตไม่แน่นอน (imprecise boundary) [36,37] เซตตามความหมายดั้งเดิมนั้น องค์ประกอบ (element) หนึ่งๆ จะสามารถเป็นหรือไม่เป็นสมาชิกของเซตได้เท่านั้น อีกนัยหนึ่งจะหมายความว่าจะมีค่าฟังก์ชันสมาชิก หรือ membership degree เป็น 0 (ไม่เป็นสมาชิก) หรือเป็น 1 (เป็นสมาชิก) เท่านั้น สำหรับฟัซซี่เซตนั้นจะสามารถให้ค่าฟังก์ชันสมาชิกสำหรับแต่ละ element ภายใต้ universe เป็นค่าที่อยู่ระหว่าง [0..1] ซึ่งจะบอกถึง degree การเป็นสมาชิกของ element นั้นๆ ดังนั้นค่า degree ของการเป็นสมาชิก (membership value) นั้น $\mu[x] : x \rightarrow [0, 1]$

ฟัซซี่เซตจะเรียกว่าเป็น normal set ถ้าภายในเซตนั้นมีสมาชิกอย่างน้อยหนึ่งตัวที่มีค่าเป็น membership degree เป็น 1 ส่วน convex ฟัซซี่เซตจะหมายถึงฟัซซี่เซตที่มี element x,y,z ใดๆ ซึ่ง $x < y < z$ กล่าวคือ $\mu_A[y] \geq \min[\mu_A[x], \mu_A[z]]$ ตัวเลขฟัซซี่ (fuzzy number) จะเป็น convex, normal ฟัซซี่เซต กำหนดให้ A, B เป็นตัวเลขฟัซซี่ และมีฟังก์ชันสมาชิก $\mu_A[x]$ และ $\mu_B[y]$ตามลำดับ กำหนดให้ * เป็นการกระทำแบบ binary ซึ่งอาจจะเป็นหนึ่งใน operation ดังนี้ $\{+, -, \times, \div, \min, \max\}$ การกระทำทางคณิตศาสตร์ของตัวเลขฟัซซี่สองตัวเลข A,B ได้เป็นเซตใหม่และมีฟังก์ชันสมาชิก $A * B$ คำนวณดังนี้ โดย extended principle [17]

$$\mu_{A*B}[z] = \bigvee_{z=x*y} (\mu_A[x] \wedge \mu_B[y])$$

โดยที่ $\vee$ และ $\wedge$คือ operation max และ min ตามลำดับ

รูปที่ 3.5(a) แสดงตัวเลขฟัซซี่ A ซึ่งเป็นเซต normal และรูปแบบ triangular พิจารณาในช่วง universe (2..6) จะเห็นค่าที่เป็นไปได้มากที่สุดของ A คือค่า 4 เนื่องจากมี ระดับความเชื่อมั่น 1 (presumption level) และรูปที่ 3.5(b) แสดงตัวเลขฟัซซี่อีกตัว B ซึ่งมี universe อยู่ในช่วง (3,7) และรูปที่ 2.5(c) แสดงผลของการกระทำ A+B ด้วยสมการข้างต้น



(a) A (b) B (c) A + B

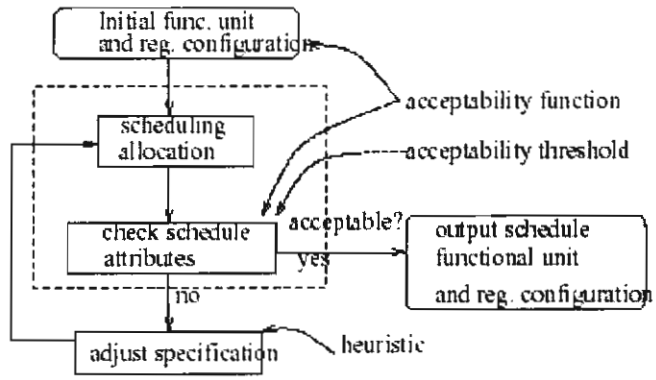
รูปที่ 3.5 การบวกตัวเลขฟัซซี่ A+B

ในการเปรียบเทียบตัวเลขฟัซซี่ของตัวมีหลายวิธีที่สามารถทำได้ วิธีการต่างๆ เหล่านี้ต่างกันที่การเลือกตัวแทนของเซตมาเปรียบเทียบ [21] วิธีการอันหนึ่งในการเลือกตัวแทนของเซตนั้นเช่น การใช้ removal ซึ่งสัมพัทธ์กับ k จะหมายถึงระยะห่างจากค่า k คำนวณได้จาก $R[A,k] = \frac{1}{2}[R_l[A,k] + R_r[A,k]]$ (สำหรับตัวเลขฟัซซี่ A) โดยที่ k เป็นระยะห่างบนแกน x และ R_l เป็นพื้นที่ด้านซ้ายของ curve membership function กับเส้นตรง x=k และ R_r เป็นพื้นที่ทางด้านขวาของ curve membership function กับเส้นตรง x=k อีกตัวอย่างได้แก่การใช้ mode ซึ่งจะเป็นการเลือกค่า x โดยที่ $\mu[x] = \max_i[\mu[x_i]]$ สำหรับทุก x_i และอีกตัวอย่างได้แก่การใช้ divergence ซึ่งหมายถึงความกว้างของเซตคำนวณโดย $x_{max} - x_{min}$ และการใช้ค่า defuzzied value เป็นตัวแทนของเซตก็สามารถทำได้เช่นเดียวกัน ซึ่งวิธีการ defuzzified ได้มีหลายวิธีในเอกสารต่างๆ [29]

จากฟัซซี่เซตดังกล่าว ในงานวิจัยนี้ได้แสดงความสัมพันธ์ระหว่างหน่วยคำนวณและค่าความเป็นไปได้ของการที่หน่วยคำนวณนั้นจะมีคุณลักษณะหนึ่งๆ ด้วยฟัซซี่เซตของคุณลักษณะ (fuzzy set of characteristics) A_f โดยกำหนดให้ $\mu_f[a] \in [0,1], \forall a \in A_f$ หมายถึง degree ความเป็นไปได้ในการมีคุณลักษณะ a ของหน่วยคำนวณ f ซึ่งในบทความนี้จะใช้ตัวอย่างของคุณลักษณะอันได้แก่เวลา (timing attribute) เช่น fuzzy timing attribute สำหรับหน่วยคำนวณ f ได้แก่ $\{\frac{10}{.2}, \frac{20}{.4}, \frac{35}{1}, \frac{70}{.7}\}$ หมายถึงว่า f สามารถใช้เวลา 10,20,35, หรือ 70 หน่วยเวลาในการประมวลผลด้วยความเป็นไปได้ $\mu_f[10] = .2, \mu_f[20] = .4, \mu_f[35] = 1, \mu_f[70] = .7$ เป็นต้น

บทที่ 4 Iterative Design Framework

รูปที่ 4.1 แสดงภาพรวมของขั้นตอนการออกแบบแบบ iterative สำหรับการหาการออกแบบที่ใช้เลือกการออกแบบที่เป็นคำตอบที่ต้องการ นักออกแบบอาจจะเลือกตัวแบบเริ่มต้นที่(initial design configuration) จาก heuristic เช่นจากการจัดลำดับแบบ ALAP (as late as possible) หรือ ASAP (as soon as possible) [8] ในงานวิจัยนี้อัลกอริทึม RCIS ถูกใช้ในขั้นตอน scheduling และ allocation ในรูปนี้ ซึ่งเป็นอัลกอริทึมที่ให้ค่าคุณลักษณะแบบ imprecise ซึ่งใช้ในการตัดสินต่อไปว่าการออกแบบที่ได้เป็นที่พึงพอใจหรือไม่



รูปที่ 4.1 การหา design solution แบบ iterative

RCIS เป็นอัลกอริทึมการจัดลำดับที่ได้ใช้ข้อมูลความไม่แน่นอนเกี่ยวกับหน่วยคำนวณมาช่วยในการจัดลำดับ โดยการป้อนอินพุตที่เป็นรูปแบบกราฟแบบ direct acyclic และป้อนจำนวนหน่วยคำนวณที่ต้องการทดลองจัดลำดับลงไป จากนั้นอัลกอริทึมจะให้ตารางการจัดลำดับเป็นคำตอบ ซึ่งตารางนี้จะแสดงถึงลำดับการทำงานของแต่ละ operation ในกราฟในหน่วยคำนวณที่กำหนด และคำนวณคุณลักษณะของตารางผลลัพธ์ที่ได้ด้วย จากนั้นคุณลักษณะดังกล่าวจะถูกนำไปตรวจสอบกับค่าระดับความพอใจที่ได้เซตไว้ (acceptability threshold) หากน้อยกว่าผู้ออกแบบจะได้ทำการปรับจำนวนและประเภทของหน่วยคำนวณที่มีอยู่และทดลองรันอัลกอริทึมอีกครั้งต่อไป อนึ่งในการปรับจำนวนและประเภทของหน่วยคำนวณนั้นขึ้นกับลักษณะของงานประยุกต์หรือกราฟที่ใช้ด้วย

บทที่ 5 Register-Constrained Inclusion Scheduling

ในบทนี้จะได้กล่าวถึงอัลกอริทึม Register-Constraint Inclusion Scheduling (RCIS) ซึ่งอาศัย core อัลกอริทึม inclusion scheduling ตาม Algorithm 5.1 อัลกอริทึมนี้จะทำการตรวจสอบคุณลักษณะของตารางจัดลำดับที่ได้โดยพิจารณาถึงปัจจัยการใช้งานรีจิสเตอร์ด้วย

5.1 Latency-based Inclusion Scheduling

ใน inclusion scheduling นั้น จะพิจารณาคุณลักษณะแบบฟัซซี่ซึ่งได้พิจารณาค่าคุณลักษณะด้านเวลาของแต่ละหน่วยคำนวณ สำหรับตารางจัดลำดับผลลัพธ์ที่ได้นั้นจะมีคุณลักษณะแบบฟัซซี่ด้วย อีกนัยหนึ่งในอัลกอริทึม inclusion scheduling จะใช้การคำนวณทางคณิตศาสตร์แบบฟัซซี่มาช่วยเนื่องจากเวลาของหน่วยคำนวณแต่ละตัวเป็นตัวเลขฟัซซี่ ในการบค่าเวลาสะสมแต่ละขั้นตอนจึงต้องใช้การคำนวณทางคณิตศาสตร์แบบฟัซซี่นั่นเอง ผลของตัวเลขฟัซซี่ที่ได้แสดงถึงค่าเวลาที่เป็นไปได้ทั้งของตารางจัดลำดับผลลัพธ์ จุดเด่นของอัลกอริทึมนี้คือการเลือกที่จะจัดสรร operation ไปยังหน่วยคำนวณหนึ่งๆ ซึ่งจะอาศัย heuristic แบบที่ทดลองจัดสรรลงไปก่อนและคำนวณคุณลักษณะของตารางผลลัพธ์ที่ได้ และเปรียบเทียบกับการทดลองจัดสรรไปยังหน่วยคำนวณอื่นๆที่เหลือ และเลือกตารางผลลัพธ์ที่มีคุณลักษณะที่"ดีที่สุด" จากหลักการง่ายๆ ดังกล่าวนี้ทำให้ได้คุณลักษณะของตารางจัดลำดับผลลัพธ์ที่ละเอียดและใช้เป็นข้อมูลในการตัดสินใจต่างๆ ได้ เช่นการเลือกใช้ module [4]

Algorithm 5.1 Inclusion Scheduling

Input: $G = [V, E, \beta]$, *Spec* $= [F, A, M, Q]$, *and* $N =$ *#FUs*

Output: *A schedule* S, *with imprecise latency*

```
1  Q = vertices in G with no incoming edges                          finding root nodes
2  while Q ≠ empty do
3      Q = prioritized(Q)
4      u = dequeue(Q); mark u scheduled
5      good_S = NULL;
6      foreach f ∈ {f_j : where f_j is able to perform β(u), 1 ≤ j ≤ N} do
7          temp_S = assign_heuristic(S, u, f)                        assign u at FU f
8          if Eval_Schedule(good_S, temp_S, G, Spec)
9              then good_S = temp_S fi od
10     S = good_S                                                    keep good schedule
11     foreach v : (u, v) ∈ E do
12         indegree(v) = indegree(v) − 1
13         if indegree(v) = 0 then enqueue(Q, v) fi od
14 od
15 return(S)
```

อัลกอรีทึม 5.1 นี้ การคำนวณทางฟัซซี่เกิดขึ้นในฟังก์ชัน Eval_Schedule บรรทัดที่ 8 ซึ่งจะทำการประเมินค่าคุณลักษณะของตารางหลังจากได้จัดสรร operation ให้หน่วยคำนวณแล้ว และจะทำการเปรียบเทียบว่าตารางที่ได้มีคุณลักษณะดีกว่าตารางอื่นๆที่ได้ทดลองมาหรือไม่ และจดจำตารางที่ดีที่สุดที่พบมาไว้

Algorithm 5.2 Eval_Schedule

Input: *schedules* S_1, S_2, $G = [V, E, \beta]$, *and Spec* $= [F, A, M, Q]$

Output: *1 if* S_1 *is better than* S_2, *0 otherwise.*

```
1  // construct a modified graph
2  G0 = [V0, E0, β] where V0 = V—{unscheduled nodes}, E0 = ∅
3  foreach schedule Si = S1 to S2 do
4    E0 = {[u, v] : u, v ∈ V0, if u, v in same f.u. in Si
5       and v is immediately after u}
6    Sort graph G according to the topological order
7    foreach level i of graph G in topological order do
8      // γ[u] returns the functional unit binding for u
9      u.attr = fuzzyadd_time[u.attr, attr[γ[u]]]
10     foreach u : v → u, ∀v ∈ Vi do
11       u.attr = fuzzymax_time[v.attr, u.attr]
12     od
13   od
14   Let W is a set of leaves in G0
15   // merge all values in quality[Si]
16   quality[Si] = fuzzymax_time[W]
17 od
18 // comparing the overall attributes of both schedules
19 return[compare[quality[S1], quality[S2]]]
```

อัลกอริทึม 5.2 เป็นอัลกอริทึม Eval_Schedule ซึ่งจะทำการเติม edge ลงไปในกราฟกลายเป็นกราฟแสดงการจัดลำดับ ถ้าโหนดสองโหนดถูกนำมาจัดลำดับต่อกันและโหนดทั้งสองไม่มีความสัมพันธ์กันมาก่อนในกราฟ ดังบรรทัดที่ 4 ดังนั้นตารางจัดลำดับที่ได้จะเป็นอยู่ในรูปของกราฟอันใหม่นี้ ซึ่งแสดงลำดับการทำงานของแต่ละโหนด สมมติให้ค่าเวลาของแต่ละหน่วยคำนวณเป็นแบบ discrete การคำนวณในบรรทัดที่ 9 จะอาศัยการกระทำแบบฟัซซี่ตามที่ได้กล่าวในบทที่ 3 ซึ่งหมายถึงการบวก และบรรทัดที่ 11 หมายถึงการคำนวณ max

เช่นถ้ากำหนดให้ DFG โดยที่ $V = \{A, B, C\}, E = \{A \rightarrow C\}$ และ $\beta(A) = \beta(B) = \beta(C) =$ add. และมีหน่วยคำนวณ $\mathcal{F} = \{FU_1, FU_2\}$ โดยที่ FU_1 มีค่าลักษณะทางด้านเวลาเป็น $\{[5, 0.05], [10, 0.7], [20, 0.8], [30, 1]\}$ และ FU_2 มีค่าลักษณะทางด้านเวลาเป็น $\{[5, 0.05], [15, 0.5], [22, 0.9], [35, 1]\}$ หลังจากทำการจัดลำดับแล้วได้ ว่า A ถูกจัดสรรให้ FU_1 และ B ถูกจัดสรรให้ FU_2 เพราะถ้าจัดสรร B ให้ FU_1 จะได้ตารางที่มีคุณลักษณะไม่ดีเท่า และจัดสรร C ให้ FU_1 ทำให้ได้ตารางผลลัพธ์ที่มี weight latency 47.9 แต่ถ้าจัดสรร C ให้ FU_2 จะได้ตารางผลลัพธ์ที่มี weight latency 52.5 ดังนั้นตารางนี้จึงเป็นตารางที่ดีกว่า สังเกตว่านี้เป็นเพียงตัวอย่างซึ่งพิจารณาแต่คุณลักษณะทางด้านเวลาเท่านั้น

5.2 อัลกอริทึม Register-Constrained Inclusion Scheduling

อัลกอริทึม Register-Constrained Inclusion Scheduling ดังแสดงดังอัลกอริทึม 5.3 จะแตกต่างจากอัลกอริทึมในรูป 5.1 ในบรรทัดที่ 8 ซึ่งเป็น routine ที่ใช้ประเมินว่าตารางจัดลำดับที่ได้ดีหรือไม่เมื่อพิจารณาปัจจัยรีจิสเตอร์มาเกี่ยวข้องแล้ว

Algorithm 5.3 Register-Constrained Inclusion Scheduling (RCIS)
Input: $G = (V, E, \beta)$, *Spec* $= (F, A, M, Q)$, *and* $N =$ #*FUs*
Output: *A schedule* S, *with imprecise latency*

```
1  Q = vertices in G with no incoming edges                          // finding root nodes
2  while Q ≠ empty do
3      Q = prioritized(Q)
4      u = dequeue(Q); mark u scheduled
5      good_S = NULL;
6      foreach f ∈ {f_j ; where f_j is able to perform β(u), 1 ≤ j ≤ N} do
7          temp_S = assign_heuristic(S, u, f)                        // assign u at FU f
8          if Eval_Schedule_with_Reg(good_S, temp_S, G, Spec)
9              then good_S = temp_S fi od
10     S = good_S                                                    // keep good schedule
11     foreach v : (u, v) ∈ E do
12         indegree(v) = indegree(v) − 1
13         if indegree(v) = 0 then enqueue(Q, v) fi od
14 od
15 return(S)
```

ซึ่งในการพิจารณาปัจจัยรีจิสเตอร์มาเกี่ยวข้องนั้นมีความซับซ้อนอยู่มาก เนื่องจากเวลาของแต่ละ operation หรือโหนดเป็นรูปแบบของฟัซซี่เซตซึ่งมีค่าฟังก์ชันสมาชิกอยู่ $\mu(x) = y$ จะหมายถึงว่าโหนดนั้นจะใช้เวลา x ด้วย possibility y ดังนั้นเวลาเริ่มต้นการทำงานของโหนดและเวลาสิ้นสุดการทำงานของโหนดในตารางจัดลำดับนั้นจะมีค่าเป็นรูปแบบฟัซซี่เซตไปด้วย ดังนั้นในการคำนวณหาเวลาเริ่มต้นและเวลาที่สิ้นสุดการทำงานของโหนดในรูปแบบฟัซซี่นี้ ก็จะอาศัยกราฟจัดลำดับมาช่วย ดังที่ได้อธิบายในหัวข้อที่แล้วถึงการสร้างกราฟจัดลำดับ ต่อไปนี้จะอธิบายถึงการคำนวณการใช้งานรีจิสเตอร์ในกราฟจัดลำดับ

5.2.1 Imprecise Timing Attribute

ในการคำนวณการใช้งานรีจิสเตอร์แบบดั้งเดิม จำเป็นต้องพิจารณาการ life time ของแต่ละโหนดในตารางจัดลำดับ แต่เนื่องจากในงานนี้ได้ใช้กราฟจัดลำดับแทน และโหนดแต่ละโหนดมีเวลาเริ่มต้นและเวลาสิ้นสุดเป็นรูปแบบฟัซซี่การคำนวณการใช้งานรีจิสเตอร์ต้องพิจารณาฟังก์ชันสมาชิกด้วย ต่อนี้จะของนิยาม FST(u) และ FFT(u) ซึ่งหมายถึงฟัซซี่เซตของเวลาเริ่มต้นและฟัซซีเซตของเวลาสิ้นสุดการทำงานของโหนด u ก่อน

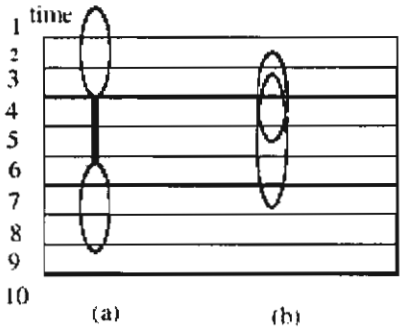
นิยาม 5.1 กำหนดกราฟ $G = (V, E, \beta)$ และตารางจัดลำดับ เวลาเริ่มต้นการทำงานของโหนด u ในกราฟ ในรูปแบบฟัซซีเซต FST(u) โดยค่าสมาชิก $\mu_{FST(u)}(x) = y$ จะหมายถึงว่าโหนด u อาจจะเริ่มต้น ณ เวลา x ด้วย possibility y

สำหรับโหนดซึ่งเริ่มที่เวลา 0 ในแต่ละหน่วยคำนวณ จะให้ $FST[u_i] = 0$ ซึ่งเป็นค่า crisp

นิยาม 5.2 กำหนดกราฟ $G = [V, E, \beta]$ และตารางจัดลำดับ เวลาเริ่มต้นการทำงานของโหนด u ในกราฟ ในรูปแบบฟัซซีเซต FFT(u) โดยค่าสมาชิก $\mu_{FFT[u]}[x] = y$ จะหมายถึงว่าโหนด u อาจจะสิ้นสุดการทำงาน ณ เวลา x ด้วย possibility y

ดังนั้น $FFT[v] = FST[v] + EXEC[v]$ โดยที่ EXEC(v) จะหมายถึงเวลาที่โหนด v ใช้ในการคำนวณแบบฟัซซี่ เมื่อพิจารณาเวลาเริ่มต้นที่เร็วที่สุดที่เป็นไปได้ (earliest start time) จะได้แก่ $FST[v] = \max_i[FFT[u_i]] + 1$ สำหรับ $\forall u_i \rightarrow v$

ความหมายของการใช้ตัวเลขฟัซซี่เมื่อแทนค่าเวลาเริ่มต้นและเวลาสิ้นสุดเป็นดังรูป 5.1 จะเห็นว่าเวลาเริ่มต้นและเวลาสิ้นสุดเป็นเซตที่มีขอบเขตไม่แน่นอน และอาจจะทับซ้อนกันได้ในรูป 5.1(b) ด้วย ดังนั้นเมื่อโหนดมีการใช้งานทรัพยากร ณ เวลาหนึ่งๆ จะต้องมีค่า possibility กำกับอยู่ด้วย



รูปที่ 5.1 ค่า FST(u) และ FFT(u) สองกรณี

ปกติแล้วเมื่อค่าเวลาในการทำงานเป็นค่า crisp ค่าเวลาเริ่มต้นและเวลาสิ้นสุดจะเป็นทำให้เกิดช่วงเวลา life time ระหว่าง [x...y] ซึ่งหมายถึงเวลาที่โหนดต้องการใช้งานรีจิสเตอร์ระหว่างการคำนวณนั้น ในกรณีนี้เมื่อพิจารณาเป็นค่าฟัซซี่ก็จะทำให้เกิด fuzzy life time ดังนั้นจึงกำหนดให้ค่า fuzzy life time ของโหนด u ประกอบด้วยฟัซซี่เซต 2 เซตได้แก่ FST(u) และ MFFT(u) ซึ่งหมายถึงค่าเวลาเริ่มต้นที่มากที่สุดที่เป็นไปได้สำหรับโหนดลูกทุกตัวของ u

นิยาม 5.3 สำหรับกราฟ $G = (V, E, \beta)$ และตารางจัดลำดับที่กำหนด ค่า fuzzy life time ของโหนด u FLT(u) จะประกอบด้วย [MFST(u), MFFT(u)] โดย $\mu_{MFFT[u]} = FFT[u] - \max FST[v_i]$ สำหรับ $u \rightarrow v_i \in E$ และ $-, \max$ เป็นตัวดำเนินการทางฟัซซี่สำหรับการบวกและการหาค่า max

สำหรับ FLT(u) กำหนดให้ min_st เป็นค่าเวลาที่น้อยที่สุดสำหรับ FST(u) เมื่อ $\mu_{FST[u]} > 0$ และกำหนดมให้ min_fin เป็นค่าเวลาสิ้นสุดที่น้อยที่สุดจาก MFFT(u) เมื่อ $\mu_{MFFT[u]} > 0$ และ max_fin เป็นค่าเวลาสิ้นสุดที่มากที่สุดจาก MFFT(u) เมื่อ $\mu_{MFFT[u]} > 0$ สมมติกำหนดให้สมาชิกใน FST(u) และ MFFT(u) ถูกเรียงลำดับได้จากค่าเวลาน้อยไปมาก สร้างฟัซซี่เซตอีก

เซตได้แก่ IFST(u) ซึ่งการ map ค่าเวลาที่อยู่ในช่วง [min_st..max_st] ไปยังเลขจำนวนจริง [0..1] ซึ่งหมายถึงค่าความเป็นไปได้ที่โหนด u ที่กำลังรันอยู่ ณ เวลา x ที่จะใช้รีจิสเตอร์ เช่นเดียวกับ IMFFT(u) สำหรับ MFFT(u) ตามนิยาม 5.4-5.5

นิยาม 5.4 สำหรับกราฟ $G = (V,E,\beta)$ และตารางจัดลำดับที่กำหนด [min_st..max_st] และ FLT(u)

$$\mu_{IFST_u}(c) = \begin{cases} 0 & \text{ถ้า } c < min\_st \text{ หรือ } c > max\_st \\ \max_{x, u \le x \le c, u = maxIFST_u \text{ and } u < c} [\mu_{IFST_u}(x)] & \text{กรณีอื่นๆ} \end{cases}$$

นิยาม 5.5 สำหรับกราฟ $G = (V,E,\beta)$ และตารางจัดลำดับที่กำหนด [min_fin..max_fin] และ FLT(u)

$$\mu_{IMFFT_u}(c) = \begin{cases} 0 & \text{ถ้า } c < min\_fiin \text{ หรือ } c > max\_fin \\ \max_{x, u \le x \le max\_fin} [\mu_{MFFT_u}(x)] & \\ u = maxIMFFT_u \text{ and } u < c & \text{กรณีอื่นๆ} \end{cases}$$

จากนิยามข้างต้น สมมติให้ ค่าเวลา a,b ∈FST(u) เมื่อ a < b ถ้าโหนด u เริ่มต้น ณ เวลา a จะได้ว่าโหนด u จะเริ่มต้นแล้ว ณ เวลา b ด้วย สำหรับ MFFT(u) เมื่อ a < b และ a,b ∈MFFT(u) ถ้าโหนด u ได้สิ้นสุดการทำงานไปแล้ว ณ เวลา a จะได้ว่าโหนด u จะก็จะสิ้นสุดการทำงาน ณ เวลา b ด้วยเช่นกัน ดังคุณลักษณะต่อไปนี้

คุณสมบัติที่ 5.1 ค่าความเป็นไปได้สำหรับ IFST(u) ถูกเรียงลำดับจากมากไปน้อย

คุณสมบัติที่ 5.2 ค่าความเป็นไปได้สำหรับ IMFFT(u) ถูกเรียงลำดับจากน้อยไปมาก

เมื่อได้ IFST(u) และ IMFFT(u) จากนั้นจะทำการรวมเซตทั้งสองเพื่อสร้าง fuzzy interval ด้วยนิยาม 5.6

นิยาม 5.6 สำหรับกราฟ $G = (V,E,\beta)$ และตารางจัดลำดับที่กำหนด และ IFST(u), IMFFT(u)

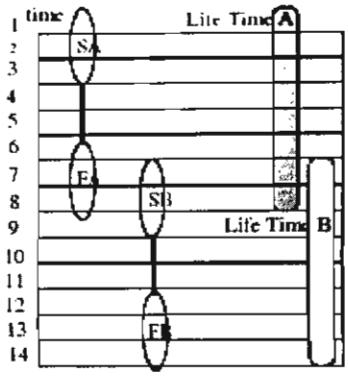
$$\mu_{FLT_u}(c) = \begin{cases} 0 & \text{ถ้า } c < \min(min\_st, min\_fin) \text{ หรือ } c > \max(max\_st, max\_fin) \\ \max(\mu_{IFST_u}(c), \mu_{IMFFT_u}(c)) & \text{ถ้า } min\_st \le c \le max\_st \text{ หรือ } min\_fin \le c \ge \le max\_fin \\ 1 & \text{กรณีอื่นๆ} \end{cases}$$

หลังจากได้ fuzzy life time ที่อยู่ในรูปแบบ interval สำหรับแต่ละโหนดแล้ว ก็จะคำนวณการใช้รีจีสเตอร์ได้ในแต่ละเวลา

5.2.2 การคำนวณการใช้งานรีจิสเตอร์

เมื่อได้ตารางจัดลำดับในรูปแบบของกราฟแล้ว จะต้องทำการคำนวณ FST(u) และ FFT(u) สำหรับ u ∈V รูปที่ 5.2 แสดงความหมายของ fuzzy life time ตามนิยาท 4.4-4.5 โดย

SA และ FA หมายถึง FST และ FFT ของโหนด A เช่นเดียวกันสำหรับ SB และ FB หมายถึง FST และ FFT ของโหนด B และ fuzzy life time ของโหนด A และ B เป็นดังรูปด้านขวา



รูปที่ 5.2 ความสัมพันธ์ระหว่างโหนดที่ถูกจัดลำดับแล้วและ life time

ในรูปนี้ life time ของโหนด A และ B อาจจะทับซ้อนกันได้ โดยทั่วไปแล้วเมื่อค่าเวลาจะเป็นค่าคงที่ซึ่งไม่ใช่เซต การที่ช่วงเวลาทับซ้อนกันของทั้งสองโหนดหมายถึงว่าณ เวลานั้นจะต้องการรีจีสเตอร์ 2 ตัว จากในตัวอย่างนี้คือ ณ เวลาที่ 7 และ 8

เมื่อเวลาในการทำงานเป็นรูปแบบฟัซซี่ กล่องแต่ละกล่องในรูปจะยังคงหมายถึงว่ายังต้องการรีจิสเตอร์จำนวนหนึ่งตัวอยู่ แต่ว่าจะต้องมีการคำนวณค่าความเป็นไปได้ในการใช้งานรีจีสเตอร์สำหรับแต่ละช่วงเวลาด้วย ซึ่งหมายถึงว่า ณ เวลานั้นโหนดอาจจะไม่ได้รันอยู่จริงก็ได้ เช่น โหนดนั้นอาจจะเริ่มหลังจากนั้น หรืออาจจะทำงานเสร็จสิ้นไปแล้ว นั่นคือหมายถึงว่าความเป็นไปได้ที่โหนดจะไม่ใช้รีจิสเตอร์ ณ เวลานั้น อีกนัยหนึ่งมีโอกาสเป็นไปได้ที่รีจิสเตอร์อาจจะมีการถูกใช้ร่วมกันกับโหนดอื่น พิจารณาการทับซ้อนในรูปที่ 5.2 ณ เวลาที่ 7 อาจจะมีการใช้ รีจิสเตอร์หนึ่งตัวหรือสองตัวก็ได้ ทั้งนี้ขึ้นอยู่กับว่าความสัมพันธ์ $A \rightarrow B$ เกิดขึ้นหรือไม่ ถ้า $A \rightarrow B$ เกิดขึ้นตามกราฟ DFG หมายถึงว่าจำนวนรีจิสเตอร์ที่ใช้จะเป็นได้แค่หนึ่งเท่านั้น ซึ่งกรณีนี้เซต IFLT(A) และ IFLT(B) จะมีผลการอินเตอร์ไม่เท่ากับเซตว่าง แต่ถ้า โหนดทั้งสองไม่สัมพันธ์กัน จำนวนรีจิสเตอร์ที่ต้องการอาจจะเป็นสองตัวก็ได้แม้ว่าเซต IFLT(A) และ IFLT(B) จะมีผลการอินเตอร์ไม่เท่ากับเซตว่าง ซึ่งทั้งสองกรณีนี้ก็ต้องถูกนำมาพิจารณาในระหว่างการคำนวณการใช้งานรีจิสเตอร์

5.2.3 อัลกอริทึม

อัลกอริทึม 5.4 อธิบายถึงการคำนวณค่าเวลาแบบ fuzzy และการคำนวณการใช้งานรีจิสเตอร์จากกราฟจัดลำดับ ซึ่งอัลกอริทึมนี้สามารถใช้ในบรรทัดที่ 8 ของอัลกอริทึม 5.3 ได้

ในอัลกอริทึมนี้บรรทัดที่ 6 จะเรียกอัลกอริทึม 5.5 เพื่อคำนวณการ life time ของโหนดแบบฟัซซี่และหาจำนวนการใช้งานรีจิสเตอร์สูงสุดของตารางจัดลำดับนี้ ซึ่งการใช้งานรีจิสเตอร์ของตารางนั้นจะถูกเก็บไว้ใน Reg[S_i] สำหรับตาราง S_i และ จากนั้นจะคำนวณเวลาที่ใช้ทั้งหมดของตารางนี้ ซึ่งหลังจากเรียกอัลกอริทึม 5.5 แล้วก็จะได้ค่าเวลาของแต่ละโหนดทั้งหมดใน

กราฟออกมา ดังนั้นเวลาที่ใช้ทั้งหมดของตารางนี้จะได้มาจากการใช้ตัวดำเนินการแบบฟัซซี่สำหรับ max โดยกระทำกับเวลาสิ้นสุดของทุกโหนดในกราฟจัดลำดับที่เป็น leaf ในบรรทัดที่ 9 จะทำการรวมค่าเวลาและการใช้งานรีจิสเตอร์เข้าด้วยกันโดยอาศัย heuristic function ค่าใหม่ที่ได้จะเรียกว่าเป็นคุณภาพของกราฟจัดลำดับดังกล่าว ซึ่งก็จะถูกนำมาเปรียบเทียบสำหรับแต่ละตารางจัดลำดับที่ได้ในแต่ละรอบเพื่อเก็บตารางที่ดีที่สุดที่หามาได้ต่อไป

Algorithm 5.4 Eval_Schedule_with_Reg

Input: *schedules* S_1, S_2, $G = (V, E, \beta)$, *and Spec* $= (\Gamma, A, M, Q)$

Output: *1 if* S_1 *is better than* S_2, *0 otherwise.*

```
1  G_0 = (V_0, E_0, β) where V_0 = V−{unscheduled nodes}, E_0 = ∅
2  foreach schedule S_i = S_1 to S_2 do
3     E_0 = {(u,v) : u,v ∈ V_0, if u,v in same f.u. in S_i
4        and v is immediately after u}
6     Calculate register usage for G_0 using Algorithm 4.3
7     Let W is a set of leaves in G_0
8     latency[S_i] = fuzzymax_time[W]
9     quality[S_i] = Combine[latency[S_i], Reg[S_i]]
10 od
12 // comparing the overall attributes of both schedules
13 return[compare[quality[S_1], quality[S_2]]]
```

Algorithm 5.5 Calculate_Register_Count

Input: *Scheduled Graph* G_0 *for schedule S and, original DFG* $G = (V, E, \beta)$ *Spec* $= (\Gamma, A, M, Q)$

Output: *Reg[S] contains register counts needed and its possibility*

```
1  Calculate FLT[u] ∀u ∈ G_0 by Definition 4.3
2  Calculate IFLT[u] ∀u ∈ G_0 by Definitions 4.4–4.5
3  Let max_cs be max. finished time , ∀u ∈ G_0
4  for cs = 1 to max_cs do
5     [RegAt[cs].reg, RegAt[cs].poss] = Count_Node[IFLT, cs, G_0] od
6  ∀n, FReg[n] = 0
7  for cs = 1 to max_cs do
8     FReg[RegAt[cs].reg].reg = RegAt[cs].reg
9     FReg[RegAt[cs].reg].poss =
10       max(FReg[RegAt[cs].reg].poss, RegAt[cs].poss) ] od
12 Reg[S] = FReg
```

ในอัลกอริทึม 5.5 RegAt จะเก็บค่าจำนวนรีจิสเตอร์ที่มากที่สุดที่ต้องการ ณ แต่ละเวลา cs รวมทั้งเก็บค่าความเป็นไปได้ ค่าทั้งสองจะถูกคำนวณด้วยอัลกอริทึม 5.6 Count_Node บรรทัด 7-10 จะคำนวณจำนวนรีจิสเตอร์ทั้งหมดที่ต้องการและค่าความเป็นไปได้

Algorithm 5.6. Count_Node

Input: [illegible] *cs*

Output: *# registers needed and its possibility at cs*

```
1  node_set ← {nodes occupy reg at cs}
2  set Gs in topological order
3  Let sorted_node be node_set sorted in by sorted Gs
4  poss ← 0, reg ← 0
5  ∀i ∈ sorted_node, i.ok ← FALSE, i.count ← FALSE
6  for every i ∈ sorted_node do
7     for j ← i + 1 to last node in sorted_node do
8        if i.ok ← TRUE and i.count ← FALSE
9          then
10              reg + +; poss ← max{poss, μreg++, i,cs}
11              i.count ← TRUE fi
12       if FindPath{i, j}
13         then j.ok ← FALSE // don't count descendant fi
14    od
15    Let j be the last node in sorted_node
16    if j.ok ← TRUE
17      then
18           reg + +; poss ← max{poss, μreg++, j,cs}
19           j.count ← TRUE fi
20 od
21 return {reg, poss}
```

ในอัลกอริทึม 5.6 จะเป็นเพียง heuristic ที่ใช้ในการพิจารณาหาโหนดที่เป็น ancestor ณ เวลาหนึ่งๆ โดยสมมติให้ ancestor นั้นทำงานเสร็จก่อนโหนดลูกก่อนที่โหนดลูกจะเริ่มทำงานได้ ตัว flag ok ใช้บอกว่าโหนดควรจะถูกนับไปด้วยหรือไม่ ณ เวลานั้น ถ้ามีโหนดที่เป็นลูกหรือหลานของโหนดที่รันอยู่ ณ เวลาเดียวกัน flag จะถูกเซตเพื่อบอกว่าจะไม่นับโหนดลูกหรือหลานนั้นๆ ไปด้วย เนื่องจากตารางจัดลำดับนี้ประกอบด้วยทุกโหนดในกราฟ โหนดลูกหรือหลานนั้นก็จะถูกนับรวมไปด้วยในที่สุด ค่า reg และ poss บอกถึงจำนวนที่นับได้ทั้งหมดและค่าความเป็นไปได้สูงสุด บรรทัดที่ 3 โหนที่อยู่รันอยู่ ณ เวลานี้ซึ่งจะถูกเรียงลำดับตามลำดับของ topological order 1 และอัลกอริทึมจะนำโหนดออกมาจากลิสต์ที่เรียงลำดับแล้วที่ละตัวมาพิจารณาว่าโหนดโหนดนั้นมีความสัมพันธ์กับโหนดอื่นในลิสต์หรือไม่โดยฟังก์ชัน Find_Path ในบรรทัดที่ 13

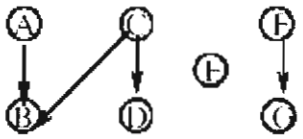
ถ้าพิจารณาเวลาในการทำงานทั้งหมดของอัลกอริทึมนี้จะได้ว่า เวลาที่ใช้ส่วนใหญ่จะอยู่ที่บรรทัด 6-21 ซึ่งคือ $O(|V|^3|V|-|E|)$ สำหรับ DAG และฟังก์ชัน Find_Path ใช้เวลา $O(|V| \cdot |E|)$

ในอัลกอริทึม 5.5 การคำนวณ FLT(u) ขึ้นกับว่า FST(u) และ MFFT(u) เป็นเซตมีจำนวนสมาชิกเท่าไร ถ้าให้ N_i เป็นจำนวนสมาชิกของ FST(u) และ MFFT(u) บรรทัดที่ 1-2 จะใช้เวลาในการคำนวณเท่ากับ $O(N_1|V||E|)$ และการคำนวณสำหรับ IFLT(u) จะอาศัยลูปซ้อนลูปเท่านั้น ดังนั้น อัลกอริทึม 5.5 รันในเวลาฟังก์ชันโพลิโนเมียล

บทที่ 6 ตัวอย่างการทำงานของ Register-Constrained Inclusion Scheduling

หลังจากได้รวมอัลกอริทึมทั้งหมดเข้าด้วยกันเป็นอัลกอริทึม RCIS แล้ว ต่อไปนี้จะนำเสนอตัวอย่างการคำนวณ FLT และตารางจัดลำดับ

พิจารณากราฟในรูป 6.1 สมมติว่ามีหน่วยคำนวณแบบ general purpose อยู่ 4 ตัว โดยให้ FU1, FU3 มีลักษณะเหมือนกัน และ FU2, FU4 มีลักษณะเหมือนกันเช่นกันดังตาราง 6.1 ในรูปคอลัมน์ (lat,poss) แสดงถึงเวลาและค่าความเป็นได้ของเวลานั้นถ้าโหนดมารันในหน่วยคำนวณดังกล่าว



รูปที่ 6.1 กราฟตัวอย่าง

FUs	(lat,poss)		(lat,poss)		(lat,poss)		(lat,poss)	
	lat	poss	lat	poss	lat	poss	lat	poss
FU1,FU3	5	0.05	10	1	15	0.9	23	0.1
FU2,FU4	7	0.5	12	0.7	17	1	29	0.05

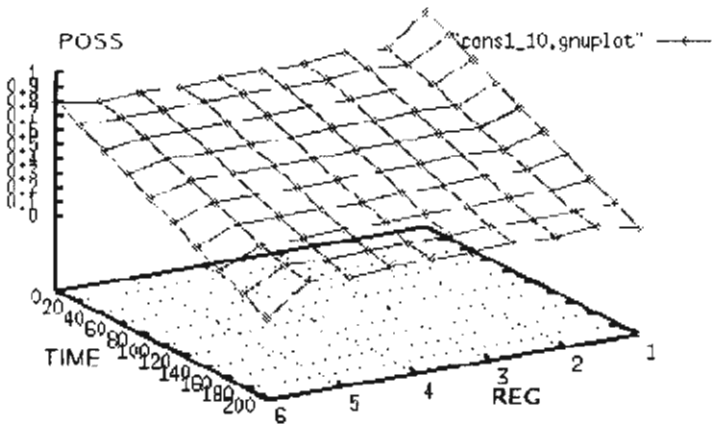
ตารางที่ 6.1 แสดงคุณลักษณะของหน่วยคำนวณ

กำหนดให้ระบบที่ต้องการมีคุณลักษณะดังรูปที่ 6.2 โดยแกนรีจิสเตอร์อยู่ระหว่าง [1..7] และค่าเวลาอยู่ในช่วง [1..200] ซึ่งในรูปนี้ได้ใช้ weighted sum เช่นเดียวกับสมการ (2.1) เมื่อกำหนดสัดส่วน (latency:register count)=1:10

ในจากบทที่แล้วรูป 6.3 (a) แสดงตารางจัดลำดับที่ได้และสังเกตว่า FU1 และ FU3 จะเป็นหน่วยคำนวณที่น่าใช้มากกว่า ในการคำนวณ FST(u) สมมติให้โหนดเริ่มต้นทำงานได้เร็วที่สุดเท่าที่จะทำได้ พิจารณาโหนด B ในตาราง รูป 6.4 (a) แสดง FLT(A) และ MFFT(A) รูป 6.5(b) แสดง FLT(B) สำหรับ FST(A) ค่าความเป็นไปได้ ณ เวลา x หมายถึงค่าความเป็นไปได้ที่โหนด A จะใช้รีจิสเตอร์หนึ่งตัว ณ เวลา x (จากรูปสี่เหลี่ยมในตาราง) เช่น กันสำหรับ MFFT(A) จะหมายถึงค่าความเป็นไปได้สำหรับโหนด A ที่จะสิ้นสุด ณ เวลา x สำหรับทั้งสองโหนด A และ B ที่ไม่ขึ้นแก่กัน รูป 6.5 เปรียบเทียบ FLT(A) และ FLT(B) จะพบว่า FST(B) จะทับซ้อนกับ MFFT(A) รูป 6.6 แสดง FLT สำหรับทุกโหนด

ค่า register count และค่าความเป็นไปได้ ณ เวลาต่างๆ แสดงในรูปที่ 6.7 จากรูปสรุปได้ว่าการใช้งานรีจีสเตอร์เป็นดังนี้ (1,0.1) และ (2,1) ซึ่งหมายถึงว่าบางครั้งอาจจะมีการใช้จำนวนรีจิสเตอร์เพียง 1 ตัวด้วยความเป็นไปได้ 0.1 และค่าเวลาสิ้นสุดที่มากที่สุดที่เป็นไปได้จากตารางจัดลำดับนี้ได้แก่ 92 ด้วยความเป็นไปได้ 0.1 พิจารณา weighted sum ของเวลาทั้งหมดที่ใช้และการใช้งานรีจิสเตอร์เป็น 79.53 ซึ่งค่าเฉลี่ยของเวลาอยู่ที่ 52 เปรียบเทียบเงื่อนไข

ด้านเวลาในรูป 6.2 แล้วพบว่าถ้าเวลาที่ใช้ทั้งหมดของตารางจัดลำดับเป็น 52 และใช้จำนวนรีจิสเตอร์เป็น 2 ตัวจะได้ acceptability degree เท่ากับ 0.76 ซึ่งจะเห็นว่าค่าความพึงพอใจดังกล่าวใกล้เคียงกับที่ได้เมื่อใช้ inclusion scheduling ซึ่งได้เวลาทั้งหมดเป็น 41 ก็จะได้ acceptability degree 0.76 เช่นกัน ซึ่งในกรณีนี้จะใช้จำนวนรีจิสเตอร์เท่ากับ 3 ตัว ตารางนี้เป็นดังรูป 6.3(b)



รูปที่ 6.2 System Specification สำหรับกราฟตัวอย่าง

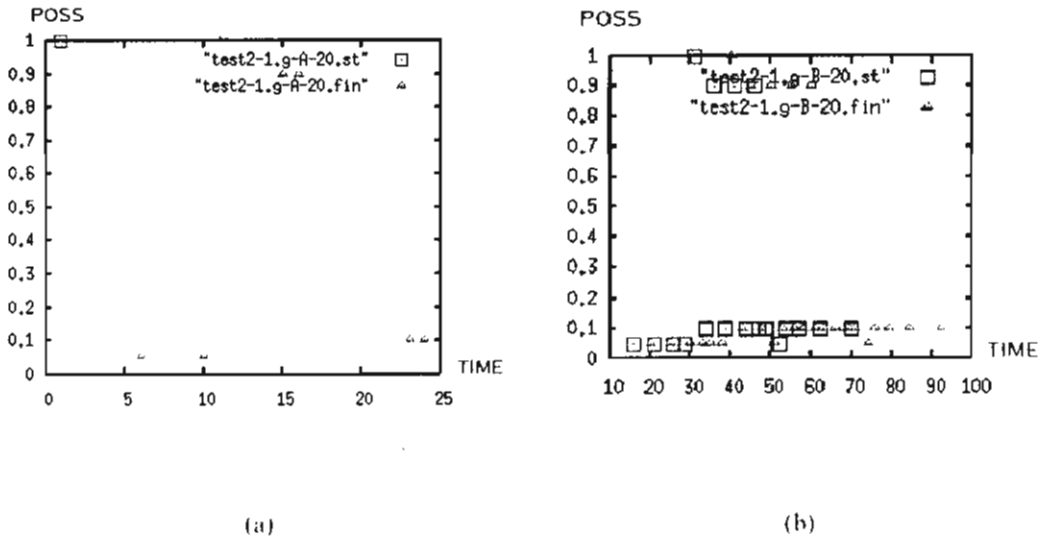
FU1	FU2	FU3	FU4
A	-	E	-
F	-	C	-
G	-	D	-
B	-	-	-

(a)

FU1	FU2	FU3	FU4
A	F	E	-
G	-	C	-
B	-	D	-

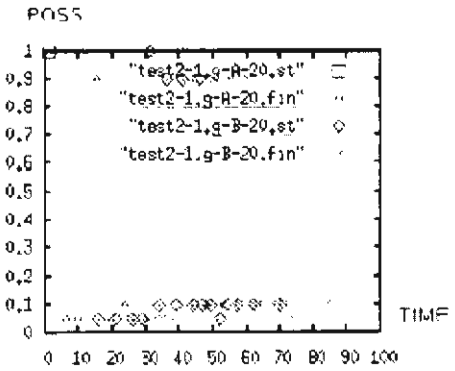
(b)

รูปที่ 6.3 (a) ตารางจัดลำดับที่ได้จาก RCIS (b) ตารางจัดลำดับที่ได้ inclusion scheduling

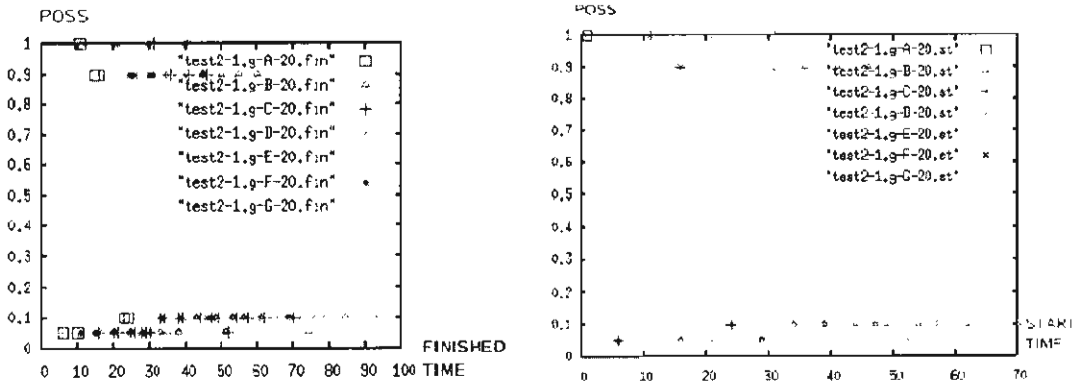


(a) (b)

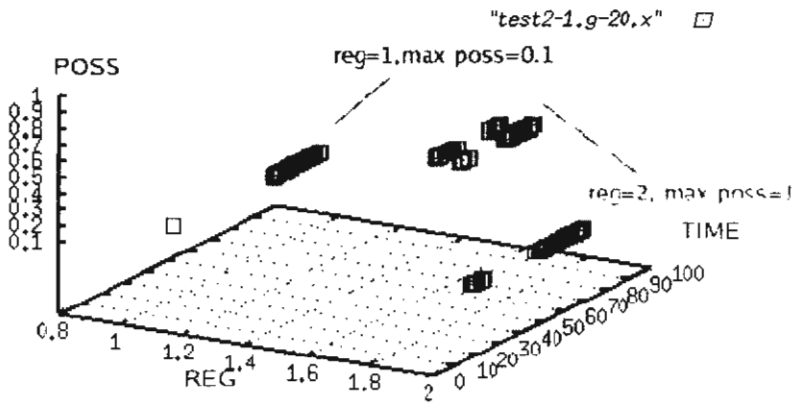
รูปที่ 6.4 (a) FLT (A) (b) FLT(B)



รูปที่ 6.5 FLT (A) และ FLT(B)

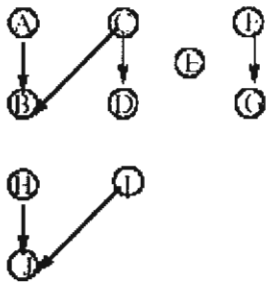


รูปที่ 6.6 FLT สำหรับทุกโหนดในกราฟ (a) FST (b) MFFT



รูปที่ 6.7 Register count และค่าความเป็นไปได้ ณ เวลาต่าง ๆ

หากเติมโหนดเข้าไปในกราฟอีก เป็นกราฟใหม่ดังรูป 6.8 และใช้ system specification เช่นเดิมและลักษณะทางหน่วยคำนวณเหมือนเดิม RCIS จะให้ตารางซึ่งใช้เวลาเฉลี่ยทั้งหมด 57 และจำนวนรีจิสเตอร์ที่ใช้ทั้งหมดเท่ากับ 2 ตัว ทำให้ได้ acceptability degree เท่ากับ 0.74 เทียบกับ inclusion scheduling ซึ่งให้ค่าเฉลี่ยเวลาเท่ากับ 37 และใช้จำนวนรีจิสเตอร์ทั้งหมดเท่ากับ 4 ตัวที่ให้ค่า acceptability degree เท่ากัน



รูปที่ 6.8 กราฟใหม่เมื่อเติมโหนดเข้าไป

บทที่ 7 ผลการทดลอง

ในที่นี้พิจารณาผลการทดลอง โดยการสำรวจการออกแบบสำหรับ benchmark Discrete Cosine Transform (DCT) [12] และ Volter filter [8]

7.1 Discrete Cosine Transform

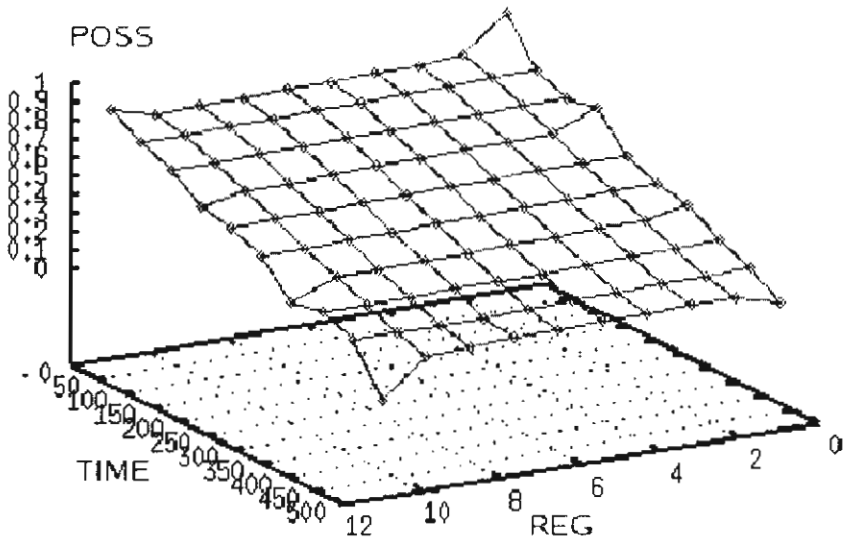
ตัวอย่าง DCT นี้ประกอบด้วย 48 โหนด ซึ่งสมมติให้แต่ละหน่วยคำนวณมีลักษณะดังในตาราง 7.1 และ system specification เป็นดังรูป 7.1 โดยให้แกนรีจิสเตอร์อยู่ในช่วง [1..12] และแกนเวลาอยู่ในใช่วง [1..500] ตาราง 7.2 แสดงผลการทดลองโดยเปรียบเทียบกรณีต่างๆ ของจำนวนหน่วยคำนวณที่ต่างๆ กัน คอลัมน์ RCIS และ IS แสดงการเปรียบเทียบการทำงานของตารางที่ได้จากอัลกอริทึม RCIS และ inclusion scheduling แถว Avg Latency แสดงถึง weighted sum ของ latency เพียงอย่างเดียว แถว Max Reg แสดงจำนวนรีจิสเตอร์สูงสุดที่ต้องการ แถว acceptability แสดง ค่า acceptability degree ของค่า (Avg Latency,Max Reg) แถว Max Latency แสดงค่าเวลาที่มากที่สุดที่เป็นไปได้ และแถว Avg Weight แสดงค่า weighted sum ของ RCIS และ IS โดยสำหรับ RCIS w_1=1 และ w_2 = 10 สำหรับ IS ค่านี้จะเหมือนกันกับค่าในช่อง Avg Latency เนื่องจาก IS จะพิจารณาการลดค่าเวลาเป็นหลัก ตาราง 7.3-7.4 แสดงค่าความเป็นไปได้ของการใช้รีจิสเตอร์แต่ละจำนวนสำหรับ RCIS และ IS ซึ่งจะพบว่า IS จะพยายามลดค่าเวลาที่ใช้ทั้งหมดของตารางจัดลำดับโดยไม่คำนึงถึงการใช้งานรีจิสเตอร์ ซึ่งจากตารางทั้งหมดสรุปได้ว่า RCIS จะให้ค่า acceptability degree ที่เท่าๆ กับ IS หรือดีกว่า โดยพิจารณาการใช้งานรีจิสเตอร์ ทำให้ใช้งานรีจิสเตอร์ไปประหยัดกว่ามากถึง 37 % ในกรณีการออกแบบโดยใช้ adder 7 ตัว และ multiplier 5 ตัว ซึ่งจากกรณีต่างๆ ที่ทดลองทั้งหมดพบว่ากรณี การใช้ adder 5 ตัว และ multiplier 4 ตัว เป็นกรณีที่ให้ค่า acceptability degree สูงสุด ในการทดลองทั้งหมดดังกล่าว การทดลองที่ใช้เวลานานที่สุดได้แก่ กรณีที่ใช้ adder 7 ตัว และ multiplier 5 ตัว โดยใช้เวลาทั้งหมดประมาณ 1 นาที 50 วินาที บนเครื่อง Pentium IV 1.8 GHz 1GB RAM

FU's	(lat,poss)		(lat,poss)		(lat,poss)		(lat,poss)	
	lat	poss	lat	poss	lat	poss	lat	poss
Adder	5	0.05	10	1	15	0.9	23	0.1
Multiplier	7	0.5	12	0.7	17	1	29	0.05

ตารางที่ 7.1 ลักษณะของ adder และ multiplier

	5 adds 4 muls		6 adds 4 muls		6 adds 5 muls		7 adds 4 [illegible]		[illegible] adds 5 muls	
	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS
Avg Latency	122	111	132	98	117	99	124	[illegible]	[illegible]	94
Max Reg	**6**	8	7	10	8	10	7	10	[illegible]	11
Acceptability	**0.719**	0.704	0.69	0.69	0.694	0.691	0.699	0.683	0.691	0.683
Max Latency	226	252	296	224	213	197	255	226	230	179
Avg Weight	188	111	198	98	206	99	210	104	209	94

ตารางที่ 7.2 ผลการทดลองของ DCT เปรียบเทียบ RCIS และ IS สำหรับจำนวนหน่วยคำนวณต่าง ๆ กัน



รูปที่ 7.1 system specification ของ DCT

#reg	2	3	4	5	6	7
poss	0.1	3	0.1	0.1	1	1

ตารางที่ 7.3 ค่าความเป็นไปได้สำหรับการใช้งานจำนวนรีจิสเตอร์ต่าง ๆ กัน สำหรับ กรณี adder 7ตัว และ multiplier 5 ตัว ของ RCIS

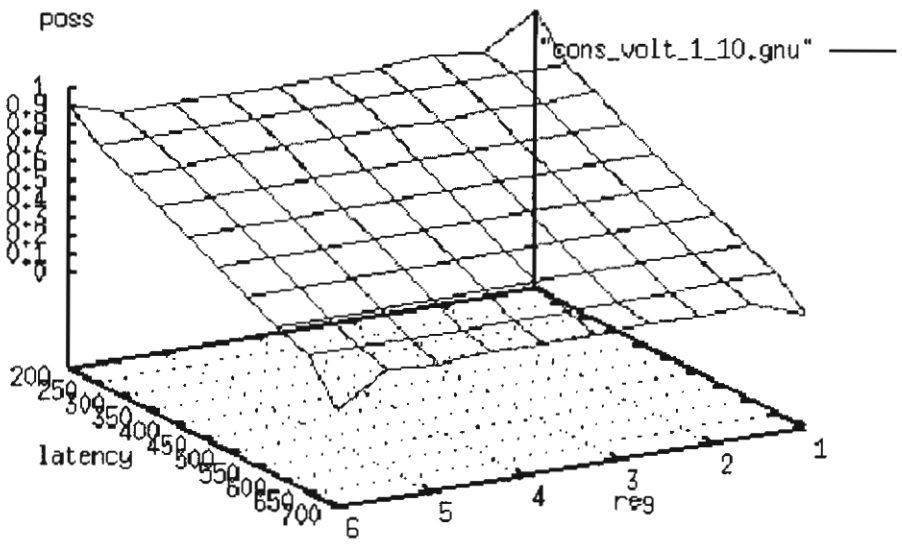
#reg	2	4	5	6	7	8	10	11
poss	0.05	0.05	1	0.05	1	0.1	1	1

ตารางที่ 7.4 ค่าความเป็นไปได้สำหรับการใช้งานจำนวนรีจิสเตอร์ต่าง ๆ กัน สำหรับ กรณี adder 7ตัว และ multiplier 5 ตัว ของ IS

7.2 Volter Filter

ใน benchmark นี้ประกอบด้วย 27 โหนดโดย 10 โหนดเป็นโหนดที่ต้องการหน่วย adder และ ที่เหลือต้องการ multiplier สมมติให้ระบบประกอบด้วย adder และ multiplier ที่มีลักษณะเช่นเดียวกับในการทดลองแรก กำหนด system specification เป็นดังรูป 7.2 โดยให้แกนรีจิสเตอร์อยู่ในช่วง [1..7] และแกนเวลาอยู่ในช่วง [200..700] เนื่องจากลักษณะของกราฟประกอบด้วยโหนดที่ต้องการ multiplier มากและโหนดเหล่านี้ส่วนใหญ่ไม่ขึ้นแก่กัน จึงทำให้การเพิ่มจำนวน multiplier จะมีทำให้ได้ตารางจัดลำดับที่ใช้เวลาลดลง สมมติให้พิจารณา acceptability threshold เท่ากับ 0.8 และ w_1=1 และ w_2 = 10 โดย RCIS พยายามจะสร้างตารางซึ่งลดค่า weighted sum ของค่าเวลาและการใช้งานรีจิสเตอร์ $w_1x+ w_2 y$ เมื่อ x,y เป็นค่าเวลาและจำนวนรีจิสเตอร์ของตารางที่ได้ตามลำดับ รูปที่ 7.4 แสดง acceptability degreeที่ได้สำหรับแต่ละการออกแบบสำหรับ RCIS เมื่อได้เพิ่มจำนวนหน่วยคำนวณจะพบว่าเวลาที่ใช้ของตารางจัดลำดับที่ได้จะลดลงด้วย จนถึงกรณีเมื่อเพิ่ม multiplier เป็น 4 ตัวหรือมากกว่า RCIS จะให้ตารางจัดลำดับที่ให้ค่า acceptability degree 0.84 ซึ่งมากกว่า acceptability threshold 0.8 ที่ต้องการ และเมื่อตรวจสอบตารางจัดลำดับก็จะพบว่าการเพิ่ม multiplier ต่อไปจะทำให้ค่า acceptability degree ลดลงด้วยเนื่องจากจะมีการใช้งานจำนวนรีจิสเตอร์ที่มากขึ้น ซึ่งต่างจาก IS ที่จะะพยายามเพิ่มจำนวน multiplier โดยไม่คำนึงถึงจำนวนรีจิสเตอร์ที่เพิ่มขึ้นตามว่าจะมีผลกระทบกับ acceptability degree ที่ลดลงไป

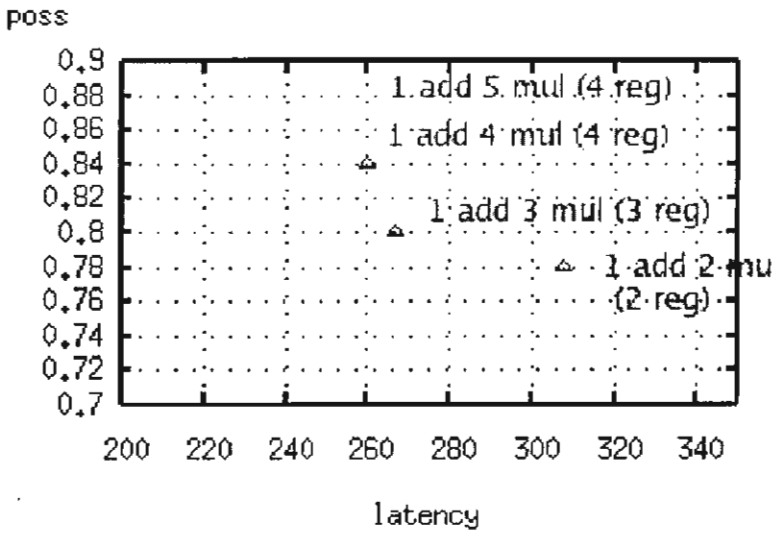
จากผลการทดลอง จะพบว่า RCIS จะให้การออกแบบผลลัพธ์ที่ดีกว่าโดยใช้จำนวนรีจิสเตอร์น้อยกว่า สำหรับเวลาในการทำงาน กรณีที่ช้าที่สุดในการทดลองชุดนี้ใช้เวลา 2.8 วินาที นั่นคือกรณี adder 1 ตัว และ multiplier 5 ตัว



รูปที่ 7.2 System Specification ของ Voltera filter

	1 add 2 muls		1 add 3 muls		1 add 4 muls		1 add 5 muls	
	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS
Avg Latency	308	300	267	270	260	260	260	246
Max Reg	2	2	3	3	**4**	4	4	5
Acceptability	0.78	0.80	0.84	0.84	**0.84**	0.84	0.84	0.84
Max Latency	561	561	474	477	445	445	445	416

ตารางที่ 7.5 ผลการทดลองของ Voltera filter เปรียบเทียบ RCIS และ IS สำหรับจำนวนหน่วยคำนวณต่าง ๆ กัน



รูปที่ 7.3 ค่า acceptability degree ของแต่ละการออกแบบ

บทที่ 8 สรุปผล

ในงานวิจัยนี้ได้นำเสนอ framework เพื่อสำรวจการออกแบบโดยพิจารณาถึงปัจจัยความไม่แน่นอน โดย framework นี้ได้อ้างอิงถึงการจัดลำดับ RCIS ซึ่งได้พิจารณาความไม่แน่นอนใน system specification และเงื่อนไข ซึ่งจะพยายามที่จะสร้างตาราจัดลำดับที่ใช้เวลาในการทำงานน้อยที่สุดและมีการใช้งานรีจิสเตอร์น้อยที่สุด framework ดังกล่าวได้สามารถนำไปใช้ในการสร้างการออกแบบภายใต้ปัจจัยความไม่แน่นอนต่างๆ และเลือกตัวออกแบบที่เหมาะสมยอมรับได้ภายใต้เงื่อนไขเวลาและการใช้งานรีจิสเตอร์ ในงานวิจัยนี้ได้ทดสอบผลการออกแบบที่เลือกโดยใช้ benchmark Discrete Cosine Transform และ Voltera filter ซึ่งพบว่าจะให้ผลการออกแบบที่เหมาะสมตามระดับความพึงพอใจที่ต้องการได้

เอกสารอ้างอิง

[1] I. Ahmad, M. K. Dhodhi, and C.Y.R. Chen. Integrated scheduling, allocation and module selection for design- space exploration in high-level synthesis. *IEEE Proc.-Comput. Digit. Tech.*, 142:65.71, January 1995.

[2] Cagdas Akturan and Margarida F. Jacome. RS-FDRA - a register sensitive software pipelining algorithm for embedded VLIW processors. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 12(21):1395.1415, December 2002.

[3] C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient scheduling for imprecise timing based on fuzzy theory. In *Proceedings of Midwest Symposium on Circuits and Systems*, pages 272.275, 1998.

[4] C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient algorithms for Finding highly acceptable designs based on module-utility selections. In *Proceedings of the Great Lake Symposium on VLSI*, pages 128.131, 1999.

[5] C. Chantrapornchai, E. H-M. Sha, and X. S. Hu. Efficient module selections for Finding highly acceptable designs based on inclusion scheduling. *J. of System Architecture*, 11(4):1047.1071, 2000.

[6] C. Chantrapornchai, E. H-M. Sha, and Xiaobo S. Hu. Efþcient acceptable design exploration based on module utility selection. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 19:19.29, Jan. 2000.

[7] C. Chantrapornchai,W. Surakumpolthorn, and E.H. Sha. Efþcient scheduling for design exploration with imprecise latency and register constraints. In *Lecture Notes in Computer Science: 2004 International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 259.270, 2004.

[8] C. Chantrapornchai and S. Tongsima. Resource estimation algorithm under impreciseness using inclusion scheduling. *Intl. J. on Foundation of Computer Science, Special Issue in Scheduling*, 12(5):581.598, 2001.

[9] S. Chaudhuri, S. A. Bylthe, and R. A Walker. An exact methodology for scheduling in 3D design space. In *Proceedings of the 1995 International Symposium on System Level Synthesis*, pages 78.83, 1995.

[10] S. Chaudhuri and R. Walker. Computing lower bounds on functional units before scheduling. In *Proceedings of the International Symposium on System Level Synthesis*, pages 36.41, 1994.

[11] A. Dani, V. Ramanan, and R. Govindarajan. Register-sensitive software pipelining. In *Proceedings. of the Merged 12th Internationa Parallel Processing and 9th International Symposium on Parallel and Distributed Systems*, pages 194.198, April 1998.

[12] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker. Datapath synthesis using a problem-space genetic algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(8):934.944, August 1995.

[13] A. Eichenberger and E. S. Davidson. Register allocation for predicated code. In *Proceeding of MICRO*, 1995. [14] Alexandre E. Eichenberger and Edward S. Davidson. Stage scheduling: A technique to reduce the register requirements of a modulo schedule. In *Proceedings of MICRO-28*, pages 338.349, 1995.

[15] H. Esbensen and E. S. Kuh. Design space exploration using the genetic algorithm. In *Proceedings of the 1996 International Symposium on Circuits and Systems*, pages 500.503, 1996.

[16] F.Chen, S. Tongsima, and E. H. Sha. Loop scheduling algorithm for timing and memory operation minimization with register constraint. In *Proceedings of SiP'98*, 1998.

[17] K. Gupta. *Introduction to fuzzy arithmetics.* Van Nostrand, 1985.

[18] O. Hammami. Fuzzy scheduling in compiler optimizations. In *Proceedings of the ISUMA-NAFIPS*, 1995.

[19] I. Karkowski. Architectural synthesis with possibilistic programming. In *HICSS-28*, January 95.

[20] I. Karkowski and R. H. J. M. Otten. Retiming synchronous circuitry with imprecise delays. In *Proceedings of the 32nd Design Automation Conference*, pages 322.326, San Francisco, CA, 1995.

[21] A. Kaufmann and M. M. Gupta. *Fuzzy mathematical models in engineering and management science*. North-Holland, 1988.

[22] A. S. Kaviani and Z. G. Vranesic. On scheduling in multiprocessor systems using fuzzy logic. In *Proceedings of the International Symposium on Multiple-valued Logic*, pages 141.147, 1994.

[23] J. Lee, A. Tiao, and J. Yen. A fuzzy rule-based approach to real-time scheduling. In *Proceedings of Intl. Conf. FUZZ-94*, volume 2, 1994.

[24] Josep Llosa, Eduard Ayguade, Antonio Gonzalez, Mateo Valero, and Jason Eckhardt. Lifetime-sensitive modulo scheduling in a production environment. *IEEE Transactions on Computers*, 50(3):234.249, 2001.

[25] Josep Llosa, Mateo Valero, and Eduard Ayguade. Heuristics for register-constrained software pipelining. In *International Symposium on Microarchitecture*, pages 250.261, 1996.

[26] C. A. Mandal, P. O. Chakrabarti, and S. Ghose. Design space exploration for data path synthesis. In *Proceedings of the10th International Conference on VLSI Design*, pages 166.170, 1996.

[27] K. Mertins et al. Set-up scheduling by fuzzy logic. In *Proceedings of the International Conference on ComputerIntegrated Manufacturing and Automation Technology*, pages 345.350, 1994.

[28] J. Rabaey and M. Potkonjak. Estimating implementation bounds for real time DSP application speciÞc circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(6), June 1994.

[29] T. J. Ross. *Fuzzy Logic with Engineering Applications*. McGrawHill, 1 edition, 1995.

ภาคผนวก

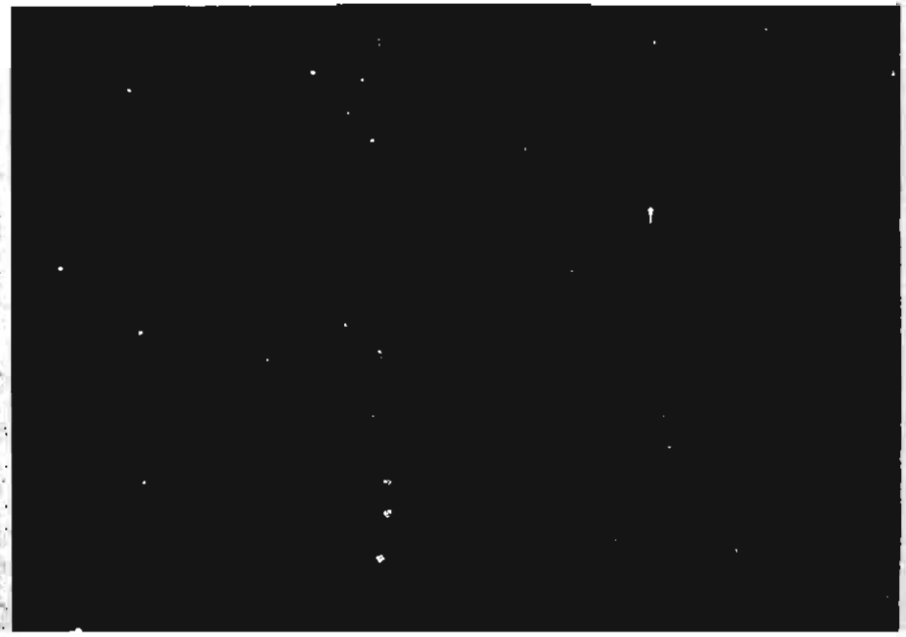
ผลงานตีพิมพ์ใน LNCS 3207 และ LNCS 3321

Laurence T. Yang
Minyi Guo
Guang R. Gao
Niraj K. Jha (Eds.)

LNCS 3207

Embedded and Ubiquitous Computing

International Conference, EUC 2004
Aizu-Wakamatsu City, Japan, August 2004
Proceedings



Component Composition Tools: Process Call Scheduling According to the Priority in Connector 107
Hwa-Young Jeong, Young-Jae Song

Experimental Performance Evaluation of Embedded Linux Using Alternative CPU Core Organizations 119
Sangsoo Park, Yonghee Lee, Heonshik Shin

A Mobile Way-Finding Application Controlling Internet-Distributed Signaling Devices via LAN Concento 131
Shuuhei Ishikawa, Michael Cohen, Kazuhiko Sawahata

Track 2: Real-Time Systems

Experimental Assessment of Scenario-Based Multithreading for Real-Time Object-Oriented Models: A Case Study with PBX Systems 143
Saehwa Kim, Michael Buettner, Mark Hermeling, Seongsoo Hong

Providing Protected Execution Environments for Embedded Operating Systems Using a μ-Kernel 153
Shuichi Oikawa, Hiroo Ishikawa, Masatoshi Iwasaki, Tatsuo Nakajima

A Virtual Reality Based System for Remote Maintenance of Rotating Machinery 164
Moez Bellamine, Norihiro Abe, Kazuaki Tanaka, Peng Chen, Hirokazu Taki

Comparison of Tie-Breaking Policies for Real-Time Scheduling on Multiprocessor 174
Minkyu Park, Sangchul Han, Heeheon Kim, Seongje Cho, Yookun Cho

Deductive Probabilistic Verification Methods for Embedded and Ubiquitous Computing 183
Satoshi Yamane, Takashi Kanatani

An Integrated Scheduling Algorithm for Precedence Constrained Hard and Soft Real-Time Tasks on Heterogeneous Multiprocessors 196
Nitin Auluck, Dharma P. Agrawal

Track 3: Power-Aware Computing

An Integrated Multichannel Selection and Dynamic Power Control Scheme for IEEE 802.11 Wireless Ad Hoc Networks 207
Jim Chung, Kuocheng Wang, Hung Chang Shi

Non-uniform Set-Associative Caches for Power-Aware Embedded Processors 217
Seiichiro Fujii, Toshinori Sato

Power-Aware Scheduling of Mixed Task Sets in Priority-Driven Systems 227
Dongkun Shin, Jihong Kim

Power Consumption of Wireless NIC and Its Impact on Joint Routing and Power Control in Ad Hoc Network 238
Min Li, Xiaobo Wu, Menglian Zhao, Hui Wang, Xiaolang Yan

Track 4: Hardware/Software Co-design and System-on-Chip

A Hardware/Software Co-design Method and Its Evaluation to ITS Image Processing and Driver-Support Systems 249
Yu Endo, Jun Sawamoto, Hisao Koizumi

Efficient Scheduling for Design Exploration with Imprecise Latency and Register Constraints 259
Chantana Chantrapornchai, Wanlop Surakumpolthorn, Edwin Sha

Hardware Mediators: A Portability Artifact for Component-Based Systems 271
Fauze Valério Polpeta, Antônio Augusto Fröhlich

On Implementation of MPEG-2 Like Real-Time Parallel Media Applications on MDSP SoC Cradle Architecture 281
Ganesh Yadav, R.K. Singh, Vipin Chaudhary

FERP Interface and Interconnect Cores for Stream Processing Applications 291
Jeff Young, Ron Sass

Folded Fat H-Tree: An Interconnection Topology for Dynamically Reconfigurable Processor Array 301
Yutaka Yamada, Hideharu Amano, Michihiro Koibuchi, Akiya Jouraku, Kenichiro Anjo, Katsunobu Nishimura

Track 5: Mobile Computing

The 3DMA Middleware for Mobile Applications 312
Tore Fjellheim, Stephen Milliner, Marlon Dumas, Kim Elms

Mining Physical Parallel Pattern from Mobile Users 324

allocated were mainly interface processing, warning processing and control processing, when written in C++, amounts to some 15k steps. Speed of execution of the processing for a single video frame at the model simulation stage was 600sec., but in real time the actual execution of processing for a single frame was completed within 100msec., satisfying the design objectives.

The driving safety support system configured in SpecC quickly detected the online IP modules, and we were able to use RPC to perform the functional evaluations.

From the above, we were able to evaluate that the proposed design method enables design objectives to be satisfied at the simulation level, and validation with a radio-controlled car was confirmed, satisfying design objectives at speeds up to 10km.

5 Conclusions

We proposed a hardware/software co-design method of the optimal allocation of functions to HW and SW. We performed simulations by connecting the object to be controlled with the control system being designed. Our proposed design method uses these simulations, following a staged approach to detailed design in allocating the functions called for by the design specifications to HW and SW. And we proposed an online method of validation using RPC to access IP components in the network environment. We confirmed the validity of this approach in an application to the design of image processing for ITS. We configured dynamic models for the control system, extended the validation system to apply to the ITS system for control processing, and evaluated its effectiveness. By using modeling without separating the functions into HW and SW, there was no need to configure the system in multiple languages. And the method of trade-offs was used to shorten development time.

In the near future, we will apply the design method to other application and evaluate the degree of effectiveness.

References

1. Hiroaki Takada "Present situation and future prospects in development technologies for embedded systems," IPSJ Journal Vol. 42, No. 4, pp. 930-938 (2001) (in Japanese)
2. Hisao Koizumi, Katsuhiko Seo, Fumio Suzuki, Yohsuke Ohtsuru, and Hiroto Yasuura, "A Proposal for a Co-design Method in Control System Using Combination of Models", IEICE Trans. on Information and Systems, vol. E78-D No. 3, pp. 237-247 (1955).
3. Thomas D.E.,Adams J.K,and Schmit H.,"A Method and Method for Hardware Software Codesign", IEEE Design and Test of Computers, Vol. 10, No. 3, pp. 6-15(1993).
4. Takashi Naitoh, Keiichi Yamada & Shin Yamamoto: "A robust method of numberplate recognition at the image taking point," IEICE Transactions A. Vol. J81-A, No. 4, pp. 536-545 (1998) (in Japanese)
5. Daniel D. Gajski et al., "The SpecC specification description language and how to use it," CQ Press K.K. (2000)
6. Developments in traffic systems (ITS) for dedicated express motorways: http://www.iijnet.or.jp/vertis/j-frame.html (2002)

Efficient Scheduling for Design Exploration with Imprecise Latency and Register Constraints

Chantana Chantrapornchai[1]*, Wanlop Surakumpolthorn[2], and Edwin Sha[3]**

[1] Faculty of Science, Silpakorn University, Nakorn Pathom, Thailand
[2] Faculty of Engineering, King Mongkut's Institute of Technology, Ladkrabang, Thailand
[3] Department of Computer Science, University of Texas, Richardson, Texas, USA

Abstract. In archiectural synthesis, scheduling and resource allocation are important steps. During the early stage of the design, imprecise information is unavoidable. Under the imprecise system characteristics and constraints, this paper proposes a polynomial-time scheduling algorithm which minimizes both functional units and registers while scheduling. The algorithm can be used in design exploration for exploring the trade-off between latency and register counts and selecting a solution with satisfactory performance and cost. The experiments show that we can achieve a schedule with the same acceptable degree while saving register upto 37% compared to the traditional algorithm.

1 Introduction

In architectural level synthesis, imprecise information is almost unavoidable. For instance, there may be various choices of modules implementing the functions or the component may have not been completely designed down to the geometry level. Another kind of impreciseness or vagueness arises from the way a design is considered to be acceptable at architecture level. If a design with latency of 50 cycles is acceptable, what about a design with 51 cycles? Especially when there are multiple conflicting design criteria, for example, is it worth to expand a latency by two cycles while saving one register and what about expanding 10 more cycles ? Effective treatment of such impreciseness in high level synthesis can undoubtedly play a key role in finding optimal design solutions.

In this paper, we present an approach to handle certain imprecise specification and use them during architectural synthesis. The system characteristics are modeled based on the fuzzy set theory. Register count is considered as another dimension of imprecise system requirement. We extend the work in [1] to create a schedule subject to register constraints under impreciseness. The proposed algorithm can be integrated into design exploration framework which considers

* This work was supported in part by the TRF under grant number MRG4680115,

the tradeoff between latency and register usage to find an acceptable solution. Such systems can be found in many digital signal processing applications, e.g., communication switches and real-time multimedia rendering systems.

Many researchers have applied the fuzzy logic approach to various kinds of scheduling problem [10,11]. These approaches, however, do not take into account the fact that an execution delay of each job can be imprecise and/or multiple attributes of a schedule. Research works related to register allocation exists in high-level synthesis and compiler optimization area for VLIW architecture. For example, Chen et. al. proposed a loop scheduling for timing and memory operation optimization under register constraint [8]. The technique is based on multi-dimensional retiming. Eichenberger et. al. presented an approach for register allocation for VLIW and superscalar code via stage scheduling [7]. Dani et. al. also presented a heuristic which uses stage scheduling to minimize register requirement. They also target at instruction level scheduling [5]. Nonetheless, these works focus on loop scheduling and do not consider handling the imprecise system characteristics or specification.

The inclusion scheduling which takes the imprecise system characteristic was proposed in [1]. The algorithm was expanded and used in design exploration under imprecise system requirement as well as the estimation of resource bounds [2,3,4]. However, it does not take register criteria in creating a schedule.

In this paper, we particularly consider both latency and register constraints. We propose an extended inclusion scheduling which considers the register usage while performing scheduling. The developed scheduling core, RCIS, *Register-Constrained Inclusion Scheduling*, takes imprecise information into account. Since the latency of the system specification is imprecise, the register usage of the schedule is imprecise. We study the imprecise register usage and propose a heuristic to estimate the register count in the imprecise schedule. Given a functional specification (in the form of a directed acyclic graph) and a number of available functional units, an inclusion schedule can be efficiently generated in polynomial time. Our proposed approach can efficiently be used in an iterative design cycle to find an initial design to reduce the number cycles of design improvements. Experimental results show that, we can achieve a better design when the number of registers is limited while keeping the same satisfactory requirement.

This paper is organized as follows: Section 2 describes our models. Section 3 presents the iterative design framework which may integrate our scheduling approach and introduce the inclusion scheduling framework. Section 4 presents necessary definitions, properties, and heuristics which integrate register consideration into inclusion scheduling framework. Section 5 displays some experimental results. Finally, Section 6 draws a conclusion from our work.

2 Overview and Models

Operations and their dependencies in an application are modeled by a vertex-weighted directed acyclic graph, called a [illegible]

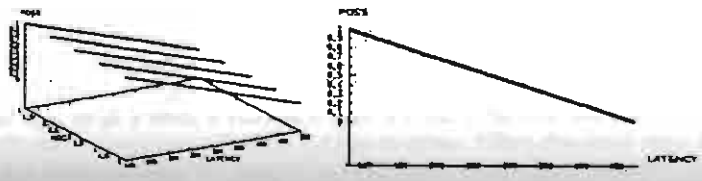
each vertex in the vertex set $\mathcal{V}$ corresponds to an operation and $\mathcal{E}$ is the set of edges representing data flow between two vertices. Function β defines the type of operation for node $v \in \mathcal{V}$.

Operations in a data flow graph can be mapped to different functional units which in turn can have varying characteristics. Such a system must also satisfy certain design constraints, for instance, power and cost limitations. These specifications are characterized by a tuple $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$, where $\mathcal{F}$ is the set of functional unit types available in the system, e.g., {add, mul}. $\mathcal{A}$ is $\{A_f : \forall f \in \mathcal{F}\}$. Each A_f is a set of tuples $(a_1, \ldots, a_k)$, where a_1 to a_k represent attributes of particular f. In this paper, we use only latency as an example attribute. (Note that our approach is readily applicable to include other constraints such as power and area). Hence, $A_f = \{x : \forall\, x\}$ where x refers to the latency attribute of f. $\mathcal{M}$ is $\{\mu_f : \forall f \in \mathcal{F}\}$ where μ_f is a mapping from A_f to a set of real number in [0,1], representing a possible degree of using the value. Finally, $\mathcal{Q}$ is a function that defines the degree of a system being acceptable for different system attributes. If $\mathcal{Q}(a_1, \ldots, a_k) = 0$ the corresponding design is totally unacceptable while $\mathcal{Q}(a_1, \ldots, a_k) = 1$, the corresponding design is definitely acceptable.

Using a function $\mathcal{Q}$ to define the acceptability of a system is a very powerful model. It can not only define certain constraints but also express certain design goals. For example, one is interested in designing a system with latency under 500 and register count being less than 6 respectively. Also, the smaller latency and register count, the better a system is. The best system would have both latency and register count being less than or equal to 100 and 1 respectively. An acceptability function, $\mathcal{Q}(a_1, a_2)$ for such a specification is formally defined as:

$$\mathcal{Q}(a_1, a_2) = \begin{cases} 0 & \text{if } a_1 > 500 \text{ or } a_2 > 6 \\ 1 & \text{if } a_1 \leq 100 \text{ and } a_2 \leq 1 \\ F(a_1, a_2) & \text{otherwise,} \end{cases} \quad (1)$$

where F is assumed to be linear functions, e.g., $F(a_1, a_2) = 1.249689(a_1 + 2a_2) - 0.001242$ which returns the acceptability between $(0, 1)$. Figures 1(a) and 1(b) illustrates Equation (1) graphically.



(a) (b)

Fig. 1. (a) Imprecise constraint Latency : Register = 1 2 (b) Its projection.

Based on the above model, the combined scheduling/binding we intend to solve can be formulated as follows:

Given a specification containing $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, Q)$, $G = (\mathcal{V}, \mathcal{E}, \beta)$, *and acceptability level* α, *find a schedule under functional unit and register constraints for each* f *in* $\mathcal{F}$ *whose the acceptability degree is greater than or equal to* α *subject* Q.

Fuzzy sets, proposed by Zadeh, represent a set with imprecise boundary [12]. A fuzzy set x is defined by assigning each element in a universe of discourse its membership degree $\mu(x)$ in the unit interval $[0, 1]$, conveying to what degree x is a member in the set. Let A and B be fuzzy numbers with membership functions $\mu_A(x)$ and $\mu_B(y)$, respectively. Let $*$ be a set of binary operations $\{+, -, \times, \div, \min, \max\}$. The arithmetic operations between two fuzzy numbers, defined on $A * B$ with membership function $\mu_{A*B}(z)$, can use the extension principle, by [9]: $\mu_{A*B}(z) = \bigvee_{z=x*y}(\mu_A(x) \wedge \mu_B(y))$ where $\vee$ and $\wedge$ denote max and min operations respectively.

Based on the basic fuzzy set concept, we model the relationship between functional units and possible characteristics such that each functional unit is associated with a fuzzy set of characteristics. Given a functional unit f and its possible characteristic set A_f let $\mu_f(a) \in [0, 1], \forall a \in A_f$, describe a possibility of having attribute a for a functional unit f.

3 Iterative Design Process

Figure 2 presents an overview of our iterative design process for finding a satisfactory solution. One may estimate the initial design configuration with any heuristic. The scheduling and allocation process produces the imprecise schedule attributes which are used to determine whether or not the design configuration is acceptable. The dashed block elements contain the scheduling core which attempts to minimize both latency and register usage. Our scheduling and allocation process incorporates varying information of each operation. It takes an application modeled by a directed acyclic graph as well as the number of functional units that can be used to compute this application. Then, the schedule of the application is derived. This schedule shows an execution order of operations in the application based on the available functional units. The total attributes of the application can be derived after the schedule is computed. The given acceptability function is then checked with the derived attributes of the schedule. In order to determine whether or not the resource configuration is satisfied the objective function, we use the *acceptability threshold*. If the schedule attributes lead to the acceptability level being greater than the threshold, the process stops. Otherwise, the resource configuration is updated and this process is repeated until the design solution cannot be improved or the design solution is found.

Specifically, inclusion scheduling is a scheduling method which takes into con-

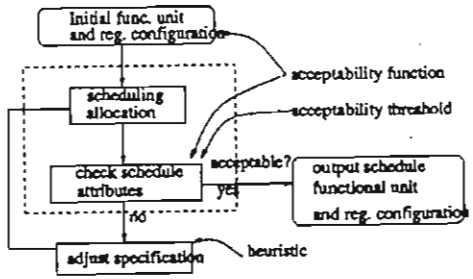


Fig. 2. Iterative design solution finding process

consists of fuzzy attributes. In a nutshell, inclusion scheduling simply replaces the computation of accumulated execution times in a traditional scheduling algorithm by the fuzzy arithmetic-based computation (See Section 2). Hence, fuzzy arithmetics is used to compute possible latency from the given functional specification. Then, using a fuzzy scheme, latency of different schedules are compared to select a functional unit for scheduling an operation. Though the concept is simple, the results are very informative. They can be used in many ways such as module selection [2].

In the inclusion scheduling, to compute a fuzzy latency, it creates a new partition graph. based on the original data flow graph and adding extra edges which connect consecutive nodes in the same function unit. Then a dummy sink node is created and connected to all leaves in the graph.

According to the scheme, we can see that the schedule table is not explicitly created. Thus, the notion of control step is implicit. This raises a few aspects. First, the nodes are assumed to start as early as possible. If the register constraint is considered, the ASAP approach may not give a good result. That is it may be good to start node later while latency is kept the same or even a little larger to maximize acceptability. Second, the graph is directly used to calculate fuzzy maximum latency. When considering only overall fuzzy latency,it is not important that nodes start ASAP or not, since in overall, the fuzzy length remains the same. However, when taking registers into account, the start time of a node becomes important since it can affect the register usage. Different start time can imply different register usage at each time step. As a result, a scheduling heuristic must be modified to consider register usage at each time step. Third, in order to consider this, since a node' s execution time is a fuzzy number, the start time of the node and its successors become a fuzzy number. When the start time of the node is a fuzzy number, the finished time of the node is a fuzzy number. We need to define the fuzzy life time of a node. Hence at each control step, a node may occupy the functional unit with some possibility. To minimize the number of registers in this way, we must also minimize the possibility of using certain

4 Register Count Consideration Under Impreciseness and RCIS

To consider register constraint, we should count the number of registers used at each time step. In specific, when we place a node on a schedule, we consider a life time of the node in the schedule. Traditionally, a life time of a node depends on the location of the node's successors in the schedule. That is the value produced by the node must be held until its successors have consumed it until the maximum time step that its successors can start. When an execution time of a node is a fuzzy number, the fuzzy life time of a node needs to be defined. In other words, at each control step, a node may occupy the functional unit with some possibility. In the following, we establish a notion of fuzzy start time, fuzzy finished time, and fuzzy life time. We then propose an algorithm to calculate fuzzy register usage for a schedule. Then both register usage and latency characteristics of the schedule are used to choose the best schedule.

Definition 1 ($FST(u)$ and $FFT(u)$). *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, a fuzzy start time and a fuzzy finished time of node of node $u \in V$,*

1. *$FST(u)$ is a fuzzy set whose membership degree is defined by $\mu_{FST(u)}(x) = y$, i.e, node u may start at time step x with possibility y.*
2. *$FFT(u)$ is a fuzzy set whose membership degree is defined by $\mu_{FFT(u)}(x) = y$, i.e, node u may finish at time step x with possibility y.*

For nodes that are executed at time step 0 in each functional unit, $FST(u) = 0$, which is a crisp value. Further, $FFT(v) = FST(v) \tilde{+} EXEC(v)$, where $EXEC(v)$ is the fuzzy latency of v. When considering earliest start time of a node, $FST(v) = \max_i(FFT(u_i)) + 1$, $\forall u_i \rightarrow v$.

The general idea of using fuzzy numbers is depicted in Figure 3(a) for both start time and finished time. Circles denote the fuzzy boundary which means that

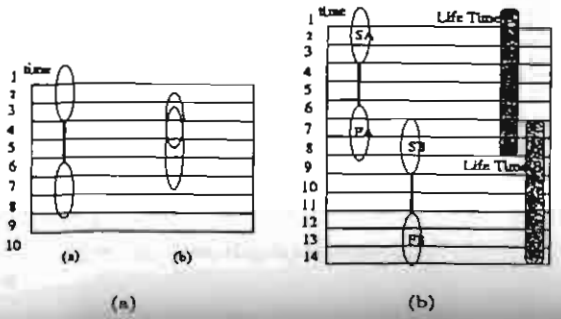


Fig. 3. (a) A view of FST,FFT (b) IFLT

the start time and finished time boundary of a node is unclear. Indeed, they may also be overlapped as shown in Figure 3(a). When a node occupies a resource at a certain time step, a possibility value is associated with the assignment. Computing a fuzzy life time for node u requires two fuzzy sets: $FST(u)$ and $MFFT(u)$, the maximum of start time of all its successors.

Definition 2. *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, fuzzy life time of node u, $FLT(u)$ is a pair of $[FST(u), MFFT(u)]$, where $\mu_{MFFT(u)} = FFT(u) \tilde{+} \max(FST(v_i))$, where $u \rightarrow v_i \in \mathcal{E}$ and $\tilde{+}$, $\tilde{\max}$ are fuzzy addition, and fuzzy maximum respectively.*

Given $FLT(u)$, let min_st be the minimum time step from $FST(u)$ whose $\mu_{FST(u)}$ is nonzero, and max_st be the maximum time step from $FST(u)$ whose $\mu_{FST(u)}$ is nonzero and similarly , for min_fin and max_fin for $\mu_{MFFT(u)}$. Without loss of generality, assume that $FST(u)$ and $MFFT(u)$ are sorted in the increasing order of the time step. We create a fuzzy set $IFST(u)$, mapping for a discrete time domain $[min\_st...max\_st]$ to real value in [0..1], showing the possibility that at time step x, node u will occupy a register for $FST(u)$ and likewise for $IMFFT(u)$ for $MFFT(u)$ as in Definition 3.

Definition 3 ($IFST(u)$ and $IMFFT(u)$). *Given $G = (\mathcal{V}, \mathcal{E}, \beta)$, a schedule, $[min\_st...max\_st]$, $[min\_fin...max\_fin]$ and $FLT(u)$*

1. *$\mu_{IFST(u)}(c) = 0$ if $c < min\_st \vee c > max\_st$ and otherwise $= \max_{\forall x, min\_st \le x < y}(\mu_{FST(u)}(x))$, where $y = max(FST(u)) \wedge y < c$*
2. *$\mu_{IMFFT(u)}(c) = 0$ if $c < min\_fin \vee c > max\_fin$ and otherwise $= \max_{\forall x, y < x \le max\_fin}(\mu_{MFFT(u)}(x))$ where $y = max(MFFT(u)) \wedge y < c$*

From the above calculation, we assume that for any two starting time value $a, b \in FST(u)$ where $a < b$, if node u starts at time a, it will be already started at time b. For $MFFT(u)$, when $a < b$, $a, b \in MFFT(u)$, if the value for node u will not be needed at time a, it will not be needed at time b and vice versa. Notice that from Definition 3, the possibility of $IFST(u)$ is in nondecreasing order and the possibility of $IMFFT(u)$ is in non-increasing order.

From $IFST(u)$ and $IMFFT(u)$, we merge the two sets to create a fuzzy interval for a node by defining Definition 4.

Definition 4. *Given $G = (\mathcal{V}, \mathcal{E}, \beta)$, a schedule, $IFST(u)$ and $IMFFT(u)$. $\mu_{IFLT(u)}(c) = \max(\mu_{IFST(u)}(c), \mu_{IMFFT(u)}(c))$ if $min\_st \le c \le max\_st \vee min\_fin \le c \le max\_fin$, or $\mu_{IFLT(u)}(c) = 0$ if $c < \min(min\_st, min\_fin) \vee c > \max(max\_st, max\_fin)$, and otherwise $\mu_{IFLT(u)}(c) = 1$.*

After we compute the fuzzy life time interval for each node, we can start compute register usage for each time step which is used to evaluate the quality of the fuzzy schedule.

Next, we explain an RCIS framework, which is based on existing inclusion scheduling core. The only difference is that RCIS uses a new heuristic which is based on previous knowledge presented in the above section. The heuristic

attempts to estimate a fuzzy property of a schedule containing both latency and register usage. It also keeps a "better" schedule at any iteration.

In RCIS, we simply replace the portion of code in inclusion scheduling which evaluates the fuzzy attributes of a schedule by *Eval_Schedule_Reg*. Then, the better schedule is chosen at each iteration. In *Eval_Schedule_Reg*, the fuzzy latency of the intermediate schedule and fuzzy register usage are computed. Algorithm 1 presents a framework which evaluates the quality of the schedule. Our heuristic considers the register usages by Algorithm 2.

Algorithm 1 (Eval_Schedule_Reg)
Input: *schedules S_1, S_2, $G = (\mathcal{V}, \mathcal{E}, \beta)$, and Spec $= (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$*
Output: *1 if S_1 is better than S_2, 0 otherwise.*

1 $G_0 = (\mathcal{V}_0, \mathcal{E}_0, \beta)$ where $\mathcal{V}_0 = \mathcal{V}$−{unscheduled nodes}, $\mathcal{E}_0 = \emptyset$
2 foreach schedule $S_i = S_1$ to S_2 do
3 $\mathcal{E}_0 = \{(u, v) : u, v \in \mathcal{V}_0$, if u, v in same f.u. in S_i
4 and v is immediately after $u\}$
6 Calculate register usage for G_0 using Algorithm 2
7 Let W is a set of leaves in G_0
8 *latency*$[S_i]$ = *fuzzymax_time*(W)
9 *quality*$[S_i]$ = *Combine*(*latency*$[S_i]$, *Reg*$[S_i]$)
10 od
12 // comparing the overall attributes of both schedules
13 return(*compare*(*quality*$[S_1]$, *quality*$[S_2]$))

Algorithm 2 (Calculate_Register_Count)
Input: *Scheduled Graph G_0 for schedule S and, original DFG $G = (\mathcal{V}, \mathcal{E}, \beta)$*
Spec $= (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$
Output: *Reg$[S]$ contains register counts needed and its possibility*

1 Calculate $FLT(u)$ $\forall u \in G_0$ by Definition 2
2 Calculate $IFLT(u)$ $\forall u \in G_0$ by Definition 3
3 Let *max_cs* be max. finish time , $\forall u \in G_0$
4 for cs = 1 to *max_cs* do
5 (*RegAt*[cs].*reg*, *RegAt*[cs].*poss*) = *Count_Node*($IFLT$, cs, G_0) od
6 $\forall n$, *FReg*[n] = 0
7 for cs = 1 to *max_cs* do
8 *FReg*[*RegAt*[cs].*reg*].*reg* = *RegAt*[cs].*reg*
9 *FReg*[*RegAt*[cs].*reg*].*poss* =
10 max(*FReg*[*RegAt*[cs].*reg*].*poss*, *RegAt*[cs].*poss*) od
11 *Reg*$[S]$ = *FReg*

In Algorithm 2, *RegAt* stores maximum number of registers needed at each cs and its associated possibility. *FReg* stores the accumulated register counts

Consider Figure 3(b). When an execution time becomes a fuzzy number, each box implies that one register is needed. However, the derived possibility associated with a time step indicates that that node may not actually exist during the time step. For example, node may start later or finish earlier. In other words, there is a possibility that a node may not use such a register. With this knowledge, the register may be shared with others with high possibility. Consider the overlap interval in Figure 3(b) at time step 7. One or two registers may be used with some possibility. This depends on whether the dependency between $A \rightarrow B$ exists. If edge $A \rightarrow B$ exists in the original data flow graph, the total register count would be one. Algorithm *Count_Node* is simply a heuristic which attempts to count only the ancestor at the current time step.

5 Experiments

Consider the simple DFG, containing nodes $\{A, B, C, D, E, F, G\}$ and edges $\{A \rightarrow B, C \rightarrow B, F \rightarrow G, C \rightarrow D\}$ Assume that 4 general functional units are available and their characteristics are according to Figure 6(a). In the Table, Columns "lat" and "pos" show the latency and its possibility of having the latency value if the nodes are executed in a functional unit. Assume the system specification where register axis contains a discrete value ranged in [1..7) and latency axis ranged in [1..200] and where latency : register count is 1:10.

FU1	FU2	FU3	FU4
A	-	E	-
F	-	C	-
G	-	D	-
B	-	-	-

(a)

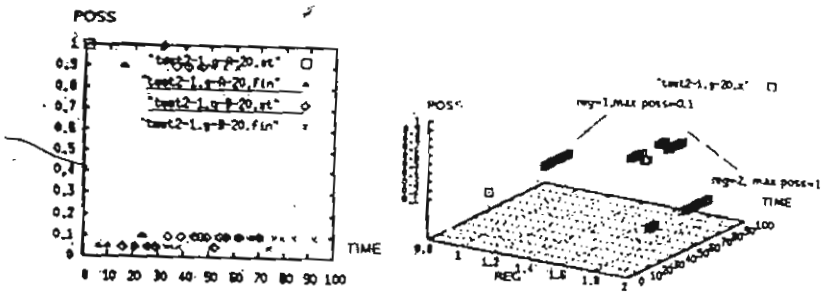
FU1	FU2	FU3	FU4
A	F	E	-
G	-	C	-
B	-	D	-

(b)

Fig. 4. Schedule obtained by (a) RCIS (b) the original IS.

Figure 4(a) shows the resulting schedule we obtain. We notice that FU1 and FU3 are preferable. To calculate $FST(u)$, we assume a heuristic where a node starts as early as possible. Figure 5(a) compares $FLT(A)$ and $FLT(B)$. We can see that $FST(B)$ overlaps with $MFFT(A)$.

We summarize the register count and its possibility value for each time step as shown in Figure 5(b). Then we conclude that the register usage as following: (1,0.1) and (2,1). It implies that at some control step, 1 register is needed with very low possibility, e.g. 0.05 and 0.1. The maximum possible finished time of the schedule is at 92 with possibility 0.1. With this schedule, the average weighted sum of latency and register is 79.53. Considering only the average latency, the value is 52. Compared to the constraint, with latency 52 and register count 2, the acceptability degree is 0.76. In fact, this gives the same acceptability level as

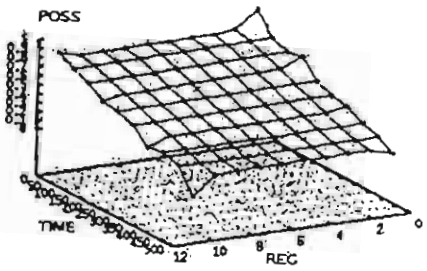


(a) (b)

Fig. 5. (a) $FLT(A)$ and $FLT(B)$ (b) Register counts and possibility each time step.

FU1		FU2		FU3		FU4	
lat	pos	lat	pos	lat	pos	lat	pos
5	0.05	7	0.5	5	0.05	7	0.5
10	1	12	0.7	10	1	12	0.7
15	0.9	17	1	15	0.9	17	1
23	0.1	29	0.05	23	0.1	29	0.05

(a)



(b)

Fig. 6. (a) FU characteristics (b) Constraint for DCT

Consider a well-known benchmark, discrete cosine transform [6], containing 48 nodes. Assume the same functional unit specification for both adders and multipliers and the constraint in Figure 6(b) where the register axis is [1..12] and the latency axis is [1..500]. We compare the results obtained from various cases of varying the number of functional units. The results are shown in Table 1. Columns "RCIS"and "IS" compare the performance of the schedule by Register-Constrained Inclusion Scheduling and the original inclusion scheduling (IS). Row "Avg Latency" shows the weighted sum of latency for each case. Row "Max Reg" displays the maximum number of registers. Row "Acceptability" shows the acceptability value obtained using the "Avg latency" and "Max Reg". Row "Max Latency" presents the maximum latency values and Row "Avg Weight" presents weighted sum value for RCIS and IS. For RCIS, recall that $\alpha = 1$ and

we only consider minimizing latency. Tables 2(b)–2(c) shows the summarized possibility values for using certain register counts for RCIS and IS respectively. It is obvious that IS attempts to minimize latency while not considering the register usage. From these tables, we can achieve about the same acceptability (and even better acceptability in some case) with fewer number of registers, which is upto 37% saving for the number of registers for the case of 7 adders and 5 multipliers. Among all these cases, we see that the configuration with 5 adders and 4 multipliers should be the best. Consider the running time. For all the cases, the maximum running time is approximately 1 minute 50 seconds to achieve the results for 7 adders and 5 multipliers under Pentium 4 2.8GHz, 1GB RAM.

Table 1. (a) Comparison of RCIS and IS when varying the number of functional units. (b)-(c) Possibility values of register counts for case case 7 adders and 5 multipliers for RCIS and IS.

	5 adds 4 muls		6 adds 4 muls		6 adds 5 muls		7 adds 4 muls		7 adds 5 muls	
	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS
Avg Latency	122	111	132	98	117	99	124	104	127	94
Max Reg	6	8	7	10	8	10	7	10	7	11
Acceptability	0.719	0.704	0.69	0.69	0.694	0.691	0.699	0.683	0.691	0.683
Max Latency	226	252	296	224	213	197	255	226	230	179
Avg Weight	188	111	198	98	206	99	210	104	209	94

(a)

#reg	2	3	4	5	6	7
poss	0.1	3	0.1	0.1	1	1

(b)

#reg	2	4	5	6	7	8	10	11
poss	0.05	0.05	1	0.05	1	0.1	1	1

(c)

6 Conclusion

We propose a polynomial-time scheduling algorithm which considers impreciseness in the system specification, constraint and attempts to create a schedule which minimizes both latency and register usages. The algorithm can be integrated into an iterative design process to find acceptable solutions. Our algorithm considers imprecise functional unit characteristic and system requirement. When the timing characteristic is imprecise, life time of a node in the schedule is imprecise. We investigate the imprecise life time of a node in the schedule and analyze the register usage. The algorithm can be integrated into a design exploration which explores an acceptable solution trading off latency cycles with register saving. The experiments show that the better and same quality schedule can be achieved using fewer number of registers compared to the traditional scheduling algorithm.

References

1. C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient scheduling for imprecise timing based on fuzzy theory. In *Proc.Midwest Symposium on Circuits and Systems*, pages 272–275, 1998.
2. C. Chantrapornchai, E. H-M. Sha, and X. S. Hu. Efficient module selections for finding highly acceptable designs based on inclusion scheduling. *J. of System Architecture*, 11(4):1047–1071, 2000.
3. C. Chantrapornchai, E. H-M. Sha, and Xiaobo S. Hu. Efficient acceptable design exploration based on module utility selection. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 19:19–29, Jan. 2000.
4. C. Chantrapornchai and S. Tongsima. Resource estimation algorithm under impreciseness using inclusion scheduling. *Intl. J. on Foundation of Computer Science, Special Issue in Scheduling*, 12(5):581–598, 2001.
5. A. Dani, V. Ramanan, and R. Govindarajan. Register-sensitive software pipelining. In *Proceedings. of the Merged 12th International Parallel Processing and 9th International Symposium on Parallel and Distributed Systems*, pages 194–198, April 1998.
6. M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker. Datapath synthesis using a problem-space genetic algorithm. *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, 14(8):934–944, August 1995.
7. A. Eichenberger and E. S. Davidson. Register allocation for predicated code. In *Proceeding of MICRO*, 1995.
8. F.Chen, S. Tongsima, and E. H. Sha. Loop scheduling algorithm for timing and memory operation minimization with register constraint. In *Proc. SiP'98*, 1998.
9. K. Gupta. *Introduction to fuzzy arithmetics*. Van Nostrand, 1985.
10. J. Lee, A. Tiao, and J. Yen. A fuzzy rule-based approach to real-time scheduling. In *Proc. Intl. Conf. FUZZ-94*, volume 2, 1994.
11. H. Soma, M. Hori, and T. Sogou. Schedule optimization using fuzzy inference. In *Proc. FUZZ-95*, pages 1171–1176, 1995.
12. L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Science*, 8:199–249, 1975.

Hardware Mediators: A Portability Artifact for Component-Based Systems

Fauze Valério Polpeta[1] and Antônio Augusto Fröhlich[1]

Federal University of Santa Catarina, PO Box 476
88049-900, Florianópolis - SC, Brazil
{fauze,guto}@lisha.ufsc.br,
http://www.lisha.ufsc.br

Abstract. In this article we elaborate on portability in component-based operating systems, focusing in the *hardware mediator* construct proposed by Fröhlich in the *Application-Oriented System Design* method. Differently from hardware abstraction layers and virtual machines, hardware mediators have the ability to establish an interface contract between the hardware and the operating system components and yet incur in very little overhead.
The use of hardware mediators in the EPOS system corroborates the portability claims associated to the techniques explained in this article, for it enabled EPOS to be easily ported across very distinct architectures, such as the H8 and the IA-32, without any modification in its software components.

1 Introduction

Portability has always been a matter for operating system developers, because the very own nature of an operating system has to do with abstracting hardware components in a way that is suitable for application programmers to develop "architecture-independent software". It is expected that an application developed on top of a chosen operating system will run unmodified in all architectures supported by that operating system. Therefore, operating systems constitute one of the main pillars of applicative software portability.

Traditional approaches to make the operating system itself portable are mainly concentrated in two flanks: *Virtual Machines* (VM) and *Hardware Abstraction Layers* (HAL). While considering the virtual machine approach to operating system portability, one cannot forget that the virtual machine itself is part of the operating system—according to Habermann, the operating system extends from the hardware to the application [10]. The virtual machine would thus constitute the architecture-dependent portion of the operating system, while granting portability for the components above. The main deficiencies of this approach is the overhead of translating VM operations into native code. Several "JAVA operating systems" rely on this approach to achieve portability.

A second major alternative for operating system portability is based on hard-

Lecture Notes in Computer Science

The LNCS series reports state-of-the-art results in computer science research, development, and education, at a high level and in both printed and electronic form. Enjoying tight cooperation with the R&D community, with numerous individuals, as well as with prestigious organizations and societies, LNCS has grown into the most comprehensive computer science research forum available.

The scope of LNCS, including its subseries LNAI and LNBI, spans the whole range of computer science and information technology including interdisciplinary topics in a variety of application fields. The type of material published traditionally includes

- proceedings (published in time for the respective conference)
- post-proceedings (consisting of thoroughly revised final full papers)
- research monographs (which may be based on outstanding PhD work, research projects, technical reports, etc.)

More recently, several color-cover sublines have been added featuring, beyond a collection of papers, various added-value components; these sublines include

- tutorials (textbook-like monographs or collections of lectures given at advanced courses)
- state-of-the-art surveys (offering complete and mediated coverage of a topic)
- hot topics (introducing emergent topics to the broader community)

In parallel to the printed book, each new volume is published electronically in LNCS Online.

Detailed information on LNCS can be found at
http://www.springeronline.com

Proposals for publication should be sent to

LNCS Editorial, Tiergartenstr. 17, 69121 Heidelberg, Germany

E-mail: lncs@springer.de

ISSN 0302-9743

Lecture Notes in Computer Science
LNCS LNAI LNBI

springeronline.com

Maher (Ed.)

LNCS 3321

LNCS
3321

Advances in Computer Science –
ASIAN 2004

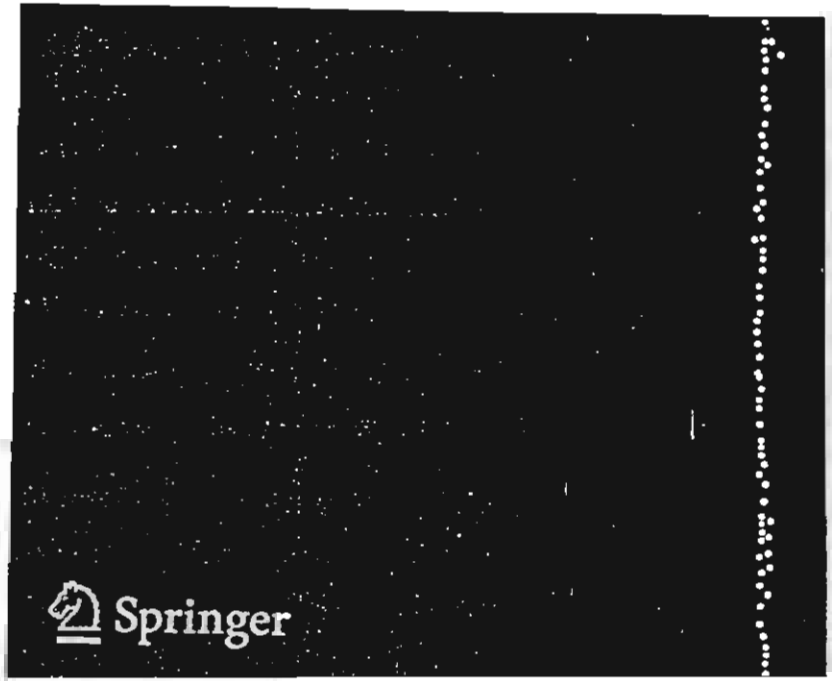
ASIAN
2004

Michael J. Maher (Ed.)

Advances in Computer Science – ASIAN 2004

Higher-Level Decision Making

9th Asian Computing Science Conference
Dedicated to Jean-Louis Lassez on the Occasion of His 5th Cycle Birthda
Chiang Mai, Thailand, December 2004, Proceedings



Referees

David Austin
Steven Bird
Colin Boyd
Gerd Brewka
Christophe Cerisara
Lai-Wan Chan
Jeff Choi
Peter Chubb
Leonid Churilov
Johanne Cohen
Véronique Cortier
Christophe Doche
Alan Dorin
Spencer Fung
Claude Godart
Michael Goldwasser
Guido Governatori
Philippe de Groote
James Harland
L.C.K. Hui
Ryutaro Ichise
Yan Jin
Norman Foo
Waleed Kadous
Yoshitaka Kameya
Yukiyoshi Kameyama
Vellaisamy Kunalmani
Shonali Krishnaswam
François Lamarche
Phu Le
Dong Hoon Lee
Ho-Fung Leung
Guoliang Li
Lin Li
Jimmy Liu
Jim Lipton
Eric Martin
Ludovic Mé
Bernd Meyer
Thomas Meyer
Dale Miller
George Mohay
Yi Mu
Lee Naish
Amedeo Napoli
Nicolas Navet
Barry O'Sullivan
Maurice Pagnucco
Kuldip Paliwal
Shawn Parr
Silvio Ranise
Jochen Renz
Seiichiro Sakurai
Abdul Sattar
Jun Shen
John Shepherd
Arcot Sowmya
Peter Stuckey
Changai Sun
Willy Susilo
Antony Tang
Peter Tischer
Takehiro Tokuda
Mark Wallace
Kin-Hong Wong
Mariko Yasugi
Akihiro Yamamoto
Haruo Yokota
Jane You
Janson Zhang

Table of Contents

Keynote Papers

Counting by Coin Tossings
Philippe Flajolet 1

On the Role Definitions in and Beyond Cryptography
Phillip Rogaway 13

Meme Media for the Knowledge Federation Over the Web and Pervasive Computing Environments
Yuzuru Tanaka, Jun Fujima, Makoto Ohigashi 33

Contributed Papers

Probabilistic Space Partitioning in Constraint Logic Programming
Nicos Angelopoulos 48

Chi-Square Matrix: An Approach for Building-Block Identification
Chatchawit Aporntewan, Prabhas Chongstitvatana 63

Design Exploration Framework Under Impreciseness Based on Register-Constrained Inclusion Scheduling
Chantana Chantrapornchai, Wanlop Surakumpolthorn, Edwin Sha 78

Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction
Daniel Cabeza, Manuel Hermenegildo, James Lipton 93

Assessment Aggregation in the Evidential Reasoning Approach to MADM Under Uncertainty: Orthogonal Versus Weighted Sum
Van-Nam Huynh, Yoshiteru Nakamori, Tu-Bao Ho 109

Learnability of Simply-Moded Logic Programs from Entailment
M.R.K. Krishna Rao 128

A Temporalised Belief Logic for Specifying the Dynamics of Trust for Multi-agent Systems
Chuchang Liu, Maris A. Ozols, Mehmet Orgun 142

Design Exploration Framework Under Impreciseness Based on Register-Constrained Inclusion Scheduling

Chantana Chantrapornchai[1,*], Wanlop Surakumpolthorn[2], and Edwin Sha[3,**]

[1] Faculty of Science, Silpakorn University, Nakorn Pathom, Thailand
[2] Faculty of Engineering, King Mongkut's Institute of Technology, Ladkrabang, Thailand
[3] Department of Computer Science, University of Texas, Richardson, Texas, USA

Abstract. In this paper, we propose a design exploration framework which consider impreciseness in design specification. In high-level synthesis, imprecise information is often encountered. Two kinds of impreciseness are considered here: imprecise characteristics of functional units and imprecise design constraints. The proposed design exploration framework is based on efficient scheduling algorithm which considers impreciseness, *Register-Constrained Inclusion Scheduling*. We demonstrate the effectiveness of our framework by exploring a design solution for a well-known benchmark, *Voltera filter*. The selected solution meets the acceptability criteria while minimizing the total number of registers.

Keywords: Imprecise Design Exploration, Scheduling/Allocation, Multiple design attributes, Imprecise information, Register constraint, Inclusion Scheduling.

1 Introduction

In architectural level synthesis, imprecise information is almost unavoidable. For instance, an implementation of a particular component in a design may not be known due to several reasons. There may be various choices of modules implementing the functions or the component may have not been completely designed down to the geometry level. Even if it has been designed, variation in fabrication process will likely induce varying area and time measurements. Another kind of impreciseness or vagueness arises from the way a design is considered to be acceptable at architecture level. If a design with latency of 50 cycles is acceptable, what about a design with 51 cycles versus a design with 75 cycles? This even becomes imprecise especially when there are multiple conflicting design criteria. For example, is it worth to expand a latency by two cycles while saving one register and what about expanding 10 more cycles ? Effective treatment of such impreciseness in high level synthesis can undoubtedly play a key role in finding optimal design solutions.

In this paper, we propose a design exploration framework which considers imprecise information underlying in system specification and requirements. Particularly, we are interested in the latency and register constraints. However, the approach can be extended to handle other multiple design criteria. The system characteristics are modeled based on the fuzzy set theory. Register count is considered as another dimension of imprecise system requirement. The work in [2, 6] is used as a scheduling core in the iterative design refinement process. The imprecise schedule which minimizes the register usage is generated. If the schedule meets acceptability criteria, the design solution is selected. Otherwise, the resources are adjusted an the process is repeated. Our input system is modeled using a data flow graph with imprecise timing parameters. Such systems can be found in many digital signal processing applications, e.g., communication switches and real-time multimedia rendering systems. Imprecise specification on both system parameters and constraints can have a significant impact on component resource allocation and scheduling for designing these systems. Therefore, it is important to develop synthesis and optimization techniques which incorporate such impreciseness.

Most traditional synthesis tools ignore these vagueness or impreciseness in the specification. In particular, they assume the worst case (or sometimes typical case) execution time of a functional unit. The constraints are usually assumed to be a fixed precise value although in reality some flexibility can be allowed in the constraint due to the individual interpretation of an "acceptable" design. Such assumptions can be misleading, and may result in a longer design process and/or overly expensive design solutions. By properly considering the impreciseness up front in the design process, a good initial design solution can be achieved with provable degree of acceptance. Such a design solution can be used effectively in the iterative process of design refinement, and thus, the number of redesign cycles can be reduced.

Random variables with probability distributions may be used to model such uncertainty. Nevertheless, collecting the probability data is sometimes difficult and time consuming. Furthermore, some imprecise information may not be correctly captured by the probabilistic model. For example, certain inconspicuousness in the design goal/constraint specification, such as the willingness of the user to accept certain designs or the confidence of the engineer towards certain designs, cannot be described by probabilistic distribution.

Many researchers have applied the fuzzy logic approach to various kinds of scheduling problem. In compiler optimization, fuzzy set theory has been used to represent unpredictable real-time events and imprecise knowledge about variables [16]. Lee et.al. applied the fuzzy inference technique to find a feasible real-time schedule where each task satisfies its deadline under resource constraints [20]. In production management area, fuzzy rules were applied to job shop and shop floor scheduling [24, 28]. Kaviani and Vranesic used fuzzy rules to determine the appropriate number of processors for a given set of tasks and deadlines for real-time systems [19]. Soma et.al. considered the schedule optimization based on fuzzy inference engine [27]. These approaches, however, do not take into account the fact that an execution delay of each job can be imprecise and/or multiple attributes of a schedule.

Many research results are available for design space exploration [1, 8, 13, 23]. All of these works differ in the techniques used to generate a design solution as well as the solution justification. These works, however, do not consider the impreciseness in the system attributes such as latency constraints and the execution time of a functional unit.

* This work was supported in part by the TRF under grant number MRG4680115, Thailand.
** This work was supported in part by TI University Program, NSF EIA 0103709, Texas ARP-009741-0028-2001 and NSF CCR-0309461, USA.

M.J. Maher (Ed.): ASIAN 2004, LNCS 3321, pp. 78–92, 2004.

Recently, Karkowski and Otten introduced a model to handle the imprecise propagation delay of events [17, 18]. In their approach, the fuzzy set theory was employed to model imprecise computation time. Their approach applies possibilistic programming based on the integer linear programming (ILP) formulation to simultaneously schedule and select a functional unit allocation under fuzzy area and time constraints. Nevertheless, the complexity of solving the ILP problem with fuzzy constraints and coefficients can be very high. Furthermore, they do not consider multiple degrees in acceptability of design solutions. Several papers were published on the resource estimation [9, 25, 26]. These approaches, however, neither consider multiple design attributes nor impreciseness in system characteristics.

Many research works related to register allocation exists in high-level synthesis and compiler optimization area for VLIW architecture. For example, Chen et. al. proposed a loop scheduling for timing and memory operation optimization under register constraint [14]. The technique is based on multi-dimensional retiming. Eichenberger et. al. presented an approach for register allocation for VLIW and superscalar code via stage scheduling [11, 12]. Dani et. al. also presented a heuristic which uses stage scheduling to minimize register requirement. They also target at instruction level scheduling [10]. Zalamea et. al. presented hardware and software approach to minimize the register's usage targeting VLIW architecture [21, 22, 30]. On the software side, they proposed an extended version of modulo scheduling which considers register constraint, and register spilling. However, these work focus on loop scheduling and do not consider handling the imprecise system characteristics or specification.

In [2], the inclusion scheduling which takes the imprecise system characteristic was proposed. The algorithm was expanded and used in design exploration under imprecise system requirement as well as the estimation of resource bounds [4, 5, 7]. However, it does not take register criteria in creating a schedule.

In this paper, we particularly consider both imprecise latency and register constraints. We develop a design exploration framework under imprecise specification and constraints. The framework is iterative and based on the developed scheduling core, RCIS, *Register-Constrained Inclusion Scheduling* that takes imprecise information into account. Experimental results show that we can achieve an accpetable design solution with minmized number of registers.

This paper is organized as follows: Section 2 describes our models. It also presents some backgrounds in fuzzy set. Section 3 presents the iterative design framework. Section 4 presents the scheduling core (RCIS) used in the design exploration framework. It also addresses some issues when the register count is calculated during scheduling. Section 5 displays some experimental results. Finally, Section 6 draws a conclusion from our work.

2 Overview and Models

Operations and their dependencies in an application are modeled by a vertex-weighted directed acyclic graph, called a *Data Flow Graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \beta)$, where each vertex in the vertex set $\mathcal{V}$ corresponds to an operation and $\mathcal{E}$ is the set of edges representing data flow between two vertices. Function β defines the type of operation for node $v \in \mathcal{V}$.

Operations in a data flow graph can be mapped to different functional units which in turn can have varying characteristics. Such a system must also satisfy certain design constraints, for instance, power and cost limitations. These specifications are characterized by a tuple $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$, where $\mathcal{F}$ is the set of functional unit types available in the system, e.g., {add, mul}. $\mathcal{A}$ is $\{A_f : \forall f \in \mathcal{F}\}$. Each A_f is a set of tuples $(a_1, \ldots, a_k)$, where a_1 to a_k represent attributes of particular f. In this paper, we use only latency as an example attribute. (Note that our approach is readily applicable to include other constraints such as power and area). Hence, $A_f = \{x : \forall\, x\}$ where x refers to the latency attribute of f. $\mathcal{M}$ is $\{\mu_f : \forall f \in \mathcal{F}\}$ where μ_f is a mapping from A_f to a set of real number in [0,1], representing a possible degree of using the value. Finally, $\mathcal{Q}$ is a function that defines the degree of a system being acceptable for different system attributes. If $\mathcal{Q}(a_1, \ldots, a_k) = 0$ the corresponding design is totally unacceptable while $\mathcal{Q}(a_1, \ldots, a_k) = 1$, the corresponding design is definitely acceptable.

Using a function $\mathcal{Q}$ to define the acceptability of a system is a very powerful model. It can not only define certain constraints but also express certain design goals. For example, one is interested in designing a system with latency under 500 and register count being less than 6 respectively. Also, the smaller latency and register count, the better a system is. The best system would have both latency and register count being less than or equal to 100 and 1 respectively. An acceptability function, $\mathcal{Q}(a_1, a_2)$ for such a specification is formally defined as:

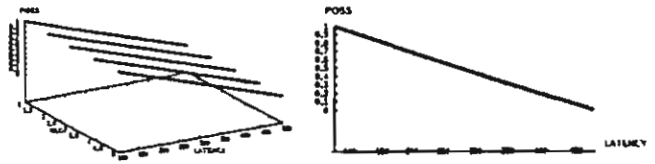
$$\mathcal{Q}(a_1, a_2) = \begin{cases} 0 & \text{if } a_1 > 500 \text{ or } a_2 > 6 \\ 1 & \text{if } a_1 \leq 100 \text{ and } a_2 \leq 1 \\ F(a_1, a_2) & \text{otherwise,} \end{cases} \quad (1)$$

where F is assumed to be linear functions, e.g., $F(a_1, a_2) = 1.249689(a_1 + 2a_2) - 0.001242$ which returns the acceptability between $(0, 1)$. Figures 1(a) and 1(b) illustrates Equation (1) graphically.

Based on the above model, the design solution we would like to find is formulated as following.

Given a specification containing $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$, $G = (\mathcal{V}, \mathcal{E}, \beta)$, and acceptability level α, find a design solution whose the acceptability degree is greater than or equal to α subject $\mathcal{Q}$.

Fuzzy sets, proposed by Zadeh, represent a set with imprecise boundary [29]. A fuzzy set x is defined by assigning each element in a universe of discourse its member-



(a) (b)

Fig. 1. (a) Imprecise constraint Latency : Register = 1 : 2 (b) Its projection

ship degree $\mu(x)$ in the unit interval $[0, 1]$, conveying to what degree x is a member in the set. Let A and B be fuzzy numbers with membership functions $\mu_A(x)$ and $\mu_B(y)$, respectively. Let $*$ be a set of binary operations $\{+, -, \times, \div, \min, \max\}$. The arithmetic operations between two fuzzy numbers, defined on $A * B$ with membership function $\mu_{A*B}(z)$, can use the extension principle, by [15]: $\mu_{A*B}(z) = \bigvee_{z=x*y}(\mu_A(x) \wedge \mu_B(y))$ where $\vee$ and $\wedge$ denote max and min operations respectively.

Based on the basic fuzzy set concept, we model the relationship between functional units and possible characteristics such that each functional unit is associated with a fuzzy set of characteristics. Given a functional unit f and its possible characteristic set A_f let $\mu_f(a) \in [0, 1], \forall a \in A_f$, describe a possibility of having attribute a for a functional unit f.

3 Iterative Design Framework

Figure 2 presents an overview of our iterative design process for finding a satisfactory solution. One may estimate the initial design configuration with any heuristic for example using ALAP, and/or ASAP scheduling [7]. The RCIS scheduling and allocation process produces the imprecise schedule attributes which are used to determine whether or not the design configuration is acceptable.

RCIS is a scheduling and allocation process which incorporates varying information of each operation. It takes an application modeled by a directed acyclic graph as well as the number of functional units that can be used to compute this application. Then, the schedule of the application is derived. This schedule shows an execution order of operations in the application based on the available functional units. The total attributes of the application can be derived after the schedule is computed. The given acceptability function is then checked with the derived attributes of the schedule.

In order to determine whether or not the resource configuration is satisfied the objective function, we use the *acceptability threshold*. If the schedule attributes lead to the acceptability level being greater than the threshold, the process stops. Otherwise, the resource configuration is adjusted using a heuristic and this process is repeated until the design solution cannot be improved or the design solution is found.

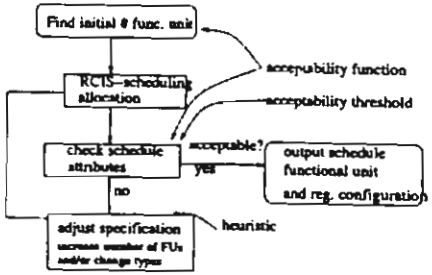


Fig. 2. Design solution finding process using RCIS

4 Register-Constraint Inclusion Scheduling

In this section, we present the register-constraint inclusion scheduling (RCIS) algorithm. The algorithm is based on the *inclusion scheduling* core presented in Algorithm 1. The algorithm evaluates the quality of the schedule by considering imprecise register criteria which will be discussed later subsections

Specifically, inclusion scheduling is a scheduling method which takes into consideration of fuzzy characteristics which in this case is fuzzy set of varying latency values associated with each functional unit. The output schedule, in turn, also consists of fuzzy attributes. In a nutshell, inclusion scheduling simply replaces the computation of accumulated execution times in a traditional scheduling algorithm by the fuzzy arithmetic-based computation. Hence, fuzzy arithmetics is used to compute possible latency from the given functional specification. Then, using a fuzzy scheme, latency of different schedules are compared to select a functional unit for scheduling an operation. Though the concept is simple, the results are very informative. They can be used in many ways such as module selection [3]. Algorithm 1 presents a list-based inclusion scheduling framework.

Algorithm 1 (Register-Constrained Inclusion scheduling)
Input: $G = (\mathcal{V}, \mathcal{E}, \beta)$, *Spec* $= (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$, *and* $N =$ *#FUs*
Output: A schedule S, with imprecise latency

```
1  Q = vertices in G with no incoming edges                    // finding root nodes
2  while Q ≠ empty do
3      Q = prioritized(Q)
4      u = dequeue(Q); mark u scheduled
5      good_S = NULL;
6      foreach f ∈ {f_j : where f_j is able to perform β(u), 1 ≤ j ≤ N} do
7        temp_S = assign_heuristic(S, u, f)                    // assign u at FU f
8        if Eval_Schedule_with_Reg(good_S, temp_S, G, Spec)
9          then good_S = temp_S fi od
10     S = good_S                                              // keep good schedule
11     foreach v : (u, v) ∈ E do
12       indegree(v) = indegree(v) − 1
13       if indegree(v) = 0 then enqueue(Q, v) fi od
14 od
15 return(S)
```

After node u is assigned to f, the imprecise attributes of the intermediate schedule, is computed. *Eval_Schedule_with_Reg* compares the current schedule with the "best" one found in previous iterations. The better one of the two is then chosen and the process is repeated for all nodes in the graph.

In Algorithm 1, fuzzy arithmetic simply takes place in routine *Eval_Schedule_with_Reg* (Line 8). In this routine, we also consider the register count used in the schedule. Since an execution time of a node is imprecise, the life time of a node is imprecise. Traditionally, a life time of a node depends on the location of the node's successors in the schedule. That is the value produced by the node must be

held until its successors have consumed it. For simplicity, let the successors consume the value right after they start.

Recall that a node's execution time is a fuzzy set, where the membership function is defined by $\mu(x) = y$. It implies that the node will take x time units with possibility y. Consequently, a start time and finished time of a node are fuzzy numbers. To be able to calculate fuzzy start time and finished time, we must assume that all nodes have been assigned to functional units already. We assume that resource binding and order of nodes executing in these resources are given based on the modified DFG (i.e, *scheduled DFG*). The modified DFG is just the original DFG where extra edges due to independent nodes executing in the same functional units are inserted (as constructed in Algorithm 3)

4.1 Imprecise Timing Attributes

In the following, we present basic terminologies used in the algorithm which calculates register usage under impreciseness.

Definition 1. *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, a fuzzy start time of node $u \in V$, $FST(u)$ is a fuzzy set whose membership degree is defined by $\mu_{FST(u)}(x) = y$, i.e, node u may start at time step x with possibility y.*

For nodes that are executed at time step 0 in each functional unit, $FST(u) = 0$, which is a crisp value.

Definition 2. *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, a fuzzy finished time of node $u \in V$, $FFT(u)$ is a fuzzy set whose membership degree is defined by $\mu_{FFT(u)}(x) = y$, i.e, node u may finish at time step x with possibility y.*

Hence, $FFT(v) = FST(v) + EXEC(v)$, where $EXEC(v)$ is the fuzzy latency of v. When considering earliest start time of a node, $FST(v) = \max_i(FFT(u_i)) + 1$, $\forall u_i \rightarrow v$.

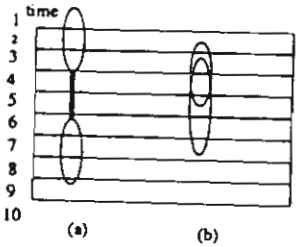


Fig. 3. A view of fuzzy start time and finished time

The general idea of using fuzzy numbers is depicted in Figure 3 for both start time and finished time. Circles denote the fuzzy boundary which means that the start time and finished time boundary of a node is unclear. Indeed, they may also be overlapped as shown in Figure 3(b). When a node occupies a resource at a certain time step, a possibility value is associated with the assignment.

Traditionally, when the timing attribute is a crisp value, the start time x and finished time y of a node form an integer interval $[x...y]$, which will be used to compute the register usage in the schedule. In our case, a fuzzy life time for node u contains two fuzzy sets: $FST(u)$ and $MFFT(u)$, the maximum of start time of all its successors.

Definition 3. *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, fuzzy life time of node u, $FLT(u)$ is a pair of $[FST(u), MFFT(u)]$, where $\mu_{MFFT(u)} = FFT(u) + \max(FST(v_i))$, where $u \rightarrow v_i \in \mathcal{E}$ and $+$, $\max$ are fuzzy addition and fuzzy maximum respectively.*

Given $FLT(u)$, let min_st be the minimum time step from $FST(u)$ whose $\mu_{FST(u)}$ is nonzero, and max_st be the maximum time step from $FST(u)$ whose $\mu_{FST(u)}$ is nonzero. Similarly, let min_fin be the minimum time step from $MFFT(u)$ whose $\mu_{MFFT(u)}$ is nonzero, and let max_fin be the maximum time step from $MFFT(u)$ whose $\mu_{MFFT(u)}$ is nonzero. Without loss of generality, assume that $FST(u)$ and $MFFT(u)$ are sorted in the increasing order of the time step. We create a fuzzy set $IFST(u)$, mapping for a discrete time domain $[min\_st...max\_st]$ to a real value in $[0..1]$, showing the possibility that at time step x, node u will occupy a register for $FST(u)$ and likewise for $IMFFT(u)$ for $MFFT(u)$ as in Definitions 4–5.

Definition 4. *$G = (\mathcal{V}, \mathcal{E}, \beta)$, a given schedule, $[min\_st...max\_st]$ and $FLT(u)$*

$$\mu_{IFST(u)}(c) = \begin{cases} 0 & \text{if } c < min\_st \text{ or } c > max\_st \\ \max_{\forall x, min\_st \le x < y, \; y = max(FST(u)) \text{ and } y < c}(\mu_{FST(u)}(x)) & \text{otherwise} \end{cases}$$

Definition 5. *$G = (\mathcal{V}, \mathcal{E}, \beta)$, a given schedule, $[min\_fin...max\_fin]$ and $FLT(u)$*

$$\mu_{IMFFT(u)}(c) = \begin{cases} 0 & \text{if } c < min\_fin \text{ or } c > max\_fin \\ \max_{\forall x, y < x \le max\_fin, \; y = max(MFFT(u)) \text{ and } y < c}(\mu_{MFFT(u)}(x)) & \text{otherwise} \end{cases}$$

From the above calculation, we assume that for any two starting time value $a, b \in FST(u)$ where $a < b$, if node u starts at time a, it will be already started at time b. For $MFFT(u)$, when $a < b$, $a, b \in MFFT(u)$, if the value for node u will not be needed at time a, it will not be needed at time b and vice versa. Thus, Definitions 4–5 give the following properties.

Property 1. The possibility of $IFST(u)$ is in nondecreasing order.

Property 2. The possibility of $IMFFT(u)$ is in non-increasing order.

From $IFST(u)$ and $IMFFT(u)$, we merge the two sets to create a fuzzy interval for a node by defining Definition 6.

Definition 6. *$G = (\mathcal{V}, \mathcal{E}, \beta)$, a given schedule, $IFST(u)$ and $IMFFT(u)$.*

$$\mu_{IFLT(u)}(c) = \begin{cases} 0 & \text{if } c < \min(min\_st, min\_fin) \text{ or } c > \max(max\_st, max\_fin) \\ \max(\mu_{IFST(u)}(c), \mu_{IMFFT(u)}(c)) & \text{if } min\_st \le c \le max\_st \text{ or } min\_fin \le c \le max\_fin \\ 1 & \text{otherwise} \end{cases}$$

After we compute the fuzzy life time interval for each node, we can start compute register usage for each time step.

4.2 Register Usage Calculation

Once a scheduled DFG is created, $FST(u)$ and $FFT(u)$ must be calculated for all $u \in V$. Figure 4 displays the meaning of fuzzy life time implied by Definitions 4–5. SA and FA denote the fuzzy start time and the fuzzy finished time of node A respectively. Similarly, SB and FB denote the start time and the fuzzy finished time of node B The fuzzy life times of A and B are shown in the filled boxes on the right side.

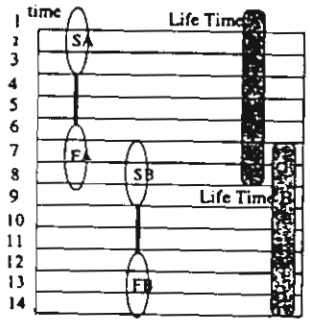


Fig. 4. Relationship between scheduled nodes and life time

In the figure, the life time of A and the life time of B may overlap. Traditionally, when the timing attribute is precise, the overlapped interval implies that two registers are needed during these time steps. In particular, during time steps 7 and 8, two registers are needed.

When an execution time becomes a fuzzy number, each box still implies that one register is needed. However, the derived possibility associated with a time step indicates that that node may not actually exist during the time step. For example, node may start later or finished earlier. In other words, there is a possibility that a node may not use such a register. With this knowledge, the register may be shared with others with high possibility. Consider the overlap interval in Figure 4 at time step 7. One or two registers may be used with some possibility. This depends on whether the dependency between $A \rightarrow B$ exists. If edge $A \rightarrow B$ exists in the original data flow graph, the total register count would be one. Notice that in this case, the intersection of $IFLT(A)$ and $IFLT(B)$ is not empty. On the contrary, if A and B are independent, the total register count would be two although the intersection may not be empty as well. This issue must be considered in calculating register usages.

4.3 Algorithms

Algorithm 2 presents a framework in evaluating fuzzy latency and register counts of a schedule. This algorithm is called after Line 7 in Algorithm 1 which already assigns the start time for each node.

In Algorithm 2, Line 6 invokes Algorithm 3 to calculate the life time of all nodes in schedule and find the maximum register usage. The register usage is then kept in $Reg[S_i]$ for a schedule S_i. Next, the latency of the whole schedule is then calculated. Note that after invoking Algorithm 3, necessary timing attributes for all nodes in G_0 can be obtained. The latency of the schedule is obtained by just fuzzy maximizing the finished time of all leaves in G_0. Line 9 merges latency and register usage attributes of the schedule using some heuristic function. The combined attribute is denoted as a *quality* of the schedule. This quality is then compared in Line 13 to select the best one.

Algorithm 2 (Eval_Schedule_with_Reg)
Input: schedules S_1, S_2, $G = (\mathcal{V}, \mathcal{E}, \beta)$, and $Spec = (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$
Output: 1 if S_1 is better than S_2, 0 otherwise.

```
1  G0 = (V0, E0, β) where V0 = V−{unscheduled nodes}, E0 = ∅
2  foreach schedule Si = S1 to S2 do
3     E0 = {(u, v) : u, v ∈ V0, if u, v in same f.u. in Si
4        and v is immediately after u}
6     Calculate register usage for G0 using Algorithm 3
7     Let W is a set of leaves in G0
8     latency[Si] = fuzzymax_time(W)
9     quality[Si] = Combine(latency[Si], Reg[Si])
10 od
12 // comparing the overall attributes of both schedules
13 return (compare(quality[S1], quality[S2]))
```

Algorithm 3 (Calculate_Register_Count)
Input: Scheduled Graph G_0 for schedule S and, original DFG $G = (\mathcal{V}, \mathcal{E}, \beta)$ $Spec = (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$
Output: $Reg[S]$ contains register counts needed and its possibility

```
1  Calculate FLT(u) ∀u ∈ G0 by Definition 3
2  Calculate IFLT(u) ∀u ∈ G0 by Definitions 4–5
3  Let max_cs be max. finished time , ∀u ∈ G0
4  for cs = 1 to max_cs do
5     (RegAt[cs].reg, RegAt[cs].poss) = Count_Node(IFLT, cs, G0) od
```

```
6  ∀n, FReg[n] = 0
7  for cs = 1 to max_cs do
8      FReg[RegAt[cs].reg].reg = RegAt[cs].reg
9      FReg[RegAt[cs].reg].poss =
10     max(FReg[RegAt[cs].reg].poss, RegAt[cs].poss) od
12 Reg[S] = FReg
```

In Algorithm 3, *RegAt* stores maximum number of registers needed at each *cs* and its associated possibility. The values are obtained by Algorithm *Count_Node*. Lines 7–10 summarize the overall number of registers needed and its possibility. Algorithm *Count_Node* is described in Algorithm 4.

Algorithm 4 (Count_Node)
Input: $IFLT$, G_0, *cs*
Output: *# registers needed and its possibility at cs*

```
1  node_set = {nodes occupy reg at cs}
2  set G0 in topological order
3  Let sorted_node be node_set sorted in by sorted G0
4  poss = 0, reg = 0
5  ∀i ∈ sorted_node, i.ok = FALSE, i.count = FALSE
6  for every i ∈ sorted_node do
7      for j = i + 1 to last node in sorted_node do
8          if i.ok = TRUE and i.count = FALSE
9            then
10                 reg + +; poss = max(poss, μ_IFLT(i)(cs)
11                 i.count = TRUE fi
13         if FindPath(i, j)
14           then j.ok = FALSE // don't count descendant fi
15     od
16     Let j be the last node in sorted_node
17     if j.ok = TRUE
18       then
19           reg + +; poss = max(poss, μ_IFLT(j)(cs)
20           j.count = TRUE fi
21 od
22 return (reg, poss)
```

In Algorithm 4, our heuristic only attempts to consider the ancestor at the current time step. In other words, we assume that the ancestor finishes first and then its descendants can start. Flag *ok* uses to indicate that the associated node should be counted at the current step or not. If it is a descendant of any of nodes in the current step, the flag will be disable. Since the schedule contains every node, the descendant will be started eventually. *reg* and *poss* store the current number of counted nodes and maximum possibility. At Line 3, the nodes currently in this time step indicated by $IFLT$ are sorted in the topological order according to G_0. Then we extract each node in the sorted list to check if any pair are dependent by using *FindPath* in Line 13. In the loop, it selectively marks descendant nodes in the current step.

Let us consider the complexity of Algorithm 4. The time complexity is dominated by Lines 6–21, which is $O(|V|^2(|V|+|E|))$. Since for DAG, *FindPath* takes $O(|V|+|E|)$.

In Algorithm 3, the calculation for $FLT(u)$ depends on $FST(u)$ and $MFFT(u)$. Let N_1 be the number of discrete points in $FST(u)$ and $MFFT(u)$. Lines 1–2 perform the calculation whose upper bound is of $O(N_1|V||E|)$. The computation for $IFLT(u)$ is simply a double loop for each node. In overall, Algorithm 3 runs in polynomial time.

5 Experimental Results

We present experimental results on the voltera filter benchmark [7], containing 27 nodes, where 10 nodes require adder units and the rest requires multiplier units. Assume that we have two types of functional units: adder and multiplier, whose latencies are as shown according to Table 1. In the figure, an adder may have different latency values with the given possibility. Columns "lat" and "pos" show the latency and its possibility of having the latency value for each adder and multiplier. Thus, if the nodes are executed in the functional unit, the node may have variable latency values as well.

Table 1. Adder and multiplier characteristics

FUs	(lat,poss)		(lat,poss)		(lat,poss)		(lat,poss)	
	lat	poss	lat	poss	lat	poss	lat	poss
adder	5	0.05	10	1	15	0.9	23	0.1
multiplier	7	0.5	12	0.7	17	1	29	0.05

Assume the constraint is depicted in Figure 5 where the register axis is [1..7] and the latency axis is [200..700]. We demonstrate by considering various design configuration of varying the number of functional units using RCIS and original inclusion scheduling as a scheduling core in the design exploration. Due to the characteristics of the filter, increasing the number of multipliers will help reduce the overall latency. Suppose that we set the acceptability threshold to be 0.8. The results are shown in Table 2. In the table, In particular, Columns "RCIS"and "IS" compare the performance of the schedule by RCIS and the original inclusion scheduling (IS) for each functional unit configuration. Row "Avg Latency" shows the weighted sum of latency for each case. Row "Max Reg" displays the maximum number of registers. Row "Acceptability" shows the acceptability value obtained using the "Avg latency" and "Max Reg". Row "Max Latency" presents the maximum latency values for each case. For RCIS, recall that $w_1 = 1$ and $w_2 = 10$. That is we consider register criteria ten times as much as the latency value. RCIS attempts to create a schedule which minimizes the total weighted sum of $w_1x + w_2y$ where x and y are the weighted latency and weighted register counts of the resulting schedule. Figure 6 we depict acceptability values for each design configuration based on RCIS. When we increase the number of functional units the latency decreases while the number of register counts needed increases. However, when the number of multipliers becomes 4 or more, RCIS can create a schedule which gives the maximum acceptability values 0.84 (which is greater than the threshold defined at 0.8).

By inspecting the resulting schedule, we conclude that 4 multipliers would be sufficient and adding more multipliers will be wasteful. Compared this the schedule generated by IS, we found that since IS does not consider the register criteria, IS attempts to utilize all available resources to minimize the overall latency values. Thus, the latency of schedule generated by IS keeps decreasing and the number of register counts keep increasing. This will finally decrease the acceptability value according the constraint.

From the results, we can see that to achieve the acceptability threshold 0.8, using RCIS will give a better design solution using fewer number of registers. Consider the running time. For all the cases, the maximum running time is approximately 2.8 seconds to achieve the results for 1 adder and 5 multipliers under Pentium 4 2.8GHz, 1GB RAM.

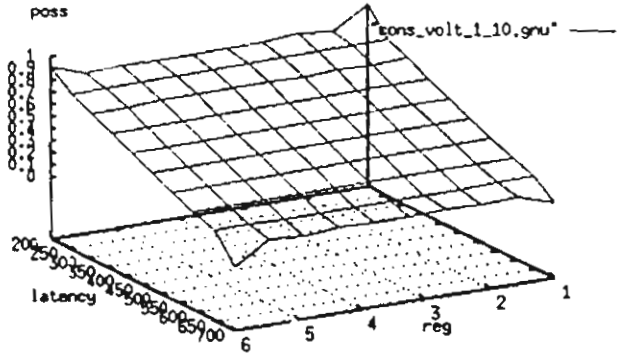


Fig. 5. Constraint for Voltera filer

Table 2. Exploring various number of functional units using RCIS and IS

	1 add 2 muls		1 add 3 muls		1 add 4 muls		1 add 5 muls	
	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS
Avg Latency	308	300	267	270	260	260	260	246
Max Reg	2	2	3	3	4	4	4	5
Acceptability	0.78	0.80	0.84	0.84	0.84	0.84	0.84	0.84
Max Latency	561	561	474	477	445	445	445	416

6 Conclusion

We propose a design exploration framework considering impreciseness. The framework is based on the scheduling core, RCIS which considers impreciseness in the system specification,and constraint and attempts to create a schedule which minimizes both

latency and register usages. The framework can be used to generate various design solutions under imprecise system constraints and characteristics and select an acceptable solution under latency and register criteria. The experiments demonstrate the usage of the framework on a well-known benchmark, where the selected design solution can be found with a given acceptability level.

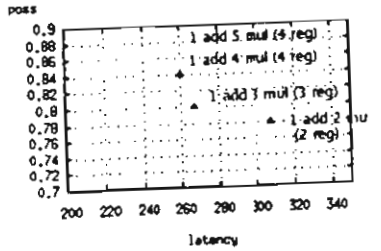


Fig. 6. Acceptability values for each configuration

References

1. I. Ahmad, M. K. Dhodhi, and C.Y.R. Chen. Integrated scheduling, allocation and module selection for design-space exploration in high-level synthesis. *IEEE Proc.-Comput. Digit. Tech.*, 142:65–71, January 1995.
2. C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient scheduling for imprecise timing based on fuzzy theory. In *Proc.Midwest Symposium on Circuits and Systems*, pages 272–275, 1998.
3. C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient algorithms for finding highly acceptable designs based on module-utility selections. In *Proceedings of the Great Lake Symposium on VLSI*, pages 128–131, 1999.
4. C. Chantrapornchai, E. H-M. Sha, and X. S. Hu. Efficient module selections for finding highly acceptable designs based on inclusion scheduling. *J. of System Architecture*, 11(4):1047–1071, 2000.
5. C. Chantrapornchai, E. H-M. Sha, and Xiaobo S. Hu. Efficient acceptable design exploration based on module utility selection. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 19:19–29, Jan. 2000.
6. C. Chantrapornchai, W. Surakumpolthorn, and E.H. Sha. Efficient scheduling for design exploration with imprecise latency and register constraints. In *Lecture Notes in Computer Science: 2004 International Conference on Embedded and Ubiquitous Computing*, 2004.
7. C. Chantrapornchai and S. Tongsima. Resource estimation algorithm under impreciseness using inclusion scheduling. *Intl. J. on Foundation of Computer Science, Special Issue in Scheduling*, 12(5):581–598, 2001.
8. S. Chaudhuri, S. A. Bylthe, and R. A Walker. An exact methodology for scheduling in 3D design space. In *Proceedings of the 1995 International Symposium on System Level Synthesis*, pages 78–83, 1995.
9. S. Chaudhuri and R. Walker. Computing lower bounds on functional units before scheduling. In *Proceedings of the International Symposium on System Level Synthesis*, pages 36–41, 1994.

10. A. Dani, V. Ramanan, and R. Govindarajan. Register-sensitive software pipelining. In *Proceedings. of the Merged 12th International Parallel Processing and 9th International Symposium on Parallel and Distributed Systems*, pages 194–198, April 1998.
11. A. Eichenberger and E. S. Davidson. Register allocation for predicated code. In *Proceeding of MICRO*, 1995.
12. Alexandre E. Eichenberger and Edward S. Davidson. Stage scheduling: A technique to reduce the register requirements of a modulo schedule. In *Proceedings of MICRO-28*, pages 338–349, 1995.
13. H. Esbensen and E. S. Kuh. Design space exploration using the genetic algorithm. In *Proceedings of the 1996 International Symposium on Circuits and Systems*, pages 500–503, 1996.
14. F.Chen, S. Tomgsima, and E. H. Sha. Loop scheduling algorithm for timing and memory operation minimization with register constraint. In *Proc. SiP'98*, 1998.
15. K. Gupta. *Introduction to fuzzy arithmetics*. Van Nostrand, 1985.
16. O. Hammami. Fuzzy scheduling in compiler optimizations. In *Proceedings of the ISUMA-NAFIPS*, 1995.
17. I. Karkowski. Architectural synthesis with possibilistic programming. In *HICSS-28*, January 95.
18. I. Karkowski and R. H. J. M. Otten. Retiming synchronous circuitry with imprecise delays. In *Proceedings of the 32nd Design Automation Conference*, pages 322–326, San Francisco, CA, 1995.
19. A. S. Kaviani and Z. G. Vranesic. On scheduling in multiprocess systems using fuzzy logic. In *Proceedings of the International Symposium on Multiple-valued Logic*, pages 141–147, 1994.
20. J. Lee, A. Tiao, and J. Yen. A fuzzy rule-based approach to real-time scheduling. In *Proc. Intl. Conf. FUZZ-94*, volume 2, 1994.
21. Josep Llosa, Eduard Ayguade, Antonio Gonzalez, Mateo Valero, and Jason Eckhardt. Lifetime-sensitive modulo scheduling in a production environment. *IEEE Transactions on Computers*, 50(3):234–249, 2001.
22. Josep Llosa, Mateo Valero, and Eduard Ayguade. Heuristics for register-constrained software pipelining. In *International Symposium on Microarchitecture*, pages 250–261, 1996.
23. C. A. Mandal, P. O. Chakrabarti, and S. Ghose. Design space exploration for data path synthesis. In *Proceedings of the10th International Conference on VLSI Design*, pages 166–170, 1996.
24. K. Mertins et al. Set-up scheduling by fuzzy logic. In *Proceedings of the International conference on computer integrated manufacturing and automation technology*, pages 345–350, 1994.
25. J. Rabaey and M. Potkonjak. Estimating implementation bounds for real time DSP application specific circuits. *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, 13(6), June 1994.
26. A. Sharma and R. Jain. Estimating architectural resources and performance for high-level synthesis applications. *IEEE Transactions on VLSI systems*, 1(2):175–190, June 1993.
27. H. Soma, M. Hori, and T. Sogou. Schedule optimization using fuzzy inference. In *Proc. FUZZ-95*, pages 1171–1176, 1995.
28. I.B. Turksen et al. Fuzzy expert system shell for scheduling. *SPIE*, pages 308–319, 1993.
29. L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Science*, 8:199–249, 1975.
30. J. Zalamea, J. Llosa, E. Ayguade, and M. Valero. Software and hardware techniques to optimize register file utilization in vliw architectures. In *Proceedings of the International Workshop on Advanced Compiler Technology for High Performance and Embedded Systems (IWACT)*, July 2001.

Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction

Daniel Cabeza[1], Manuel Hermenegildo[1,2], and James Lipton[1,3]

[1] Technical University of Madrid, Spain
{dcabeza, herme, jlipton}@fi.upm.es
[2] University of New Mexico, USA
herme@unm.edu
[3] Wesleyan University, USA
jlipton@wesleyan.edu

Abstract. A new formalism, called Hiord, for defining type-free higher-order logic programming languages with predicate abstraction is introduced. A model theory, based on partial combinatory algebras, is presented, with respect to which the formalism is shown sound. A programming language built on a subset of Hiord, and its implementation are discussed. A new proposal for defining modules in this framework is considered, along with several examples.

1 Introduction

This paper presents a new declarative formalism, called Hiord, for logic programming with untyped higher-order logic and predicate abstractions. This is followed by a discussion of various practical restrictions of this logic to make it amenable to speedy translation to WAM-compilable code and static analysis.

A number of proposals have been made over the past two decades to introduce higher-order features into logic programming in a declarative fashion by extending the underlying logic, among them λProlog and Hilog [1–4]. This has proven a very useful way to place on a solid logical ground certain natural steps that, in the original first-order context of pure logic programming, seem to compromise declarative transparency. For example, the simple transformation of code such as the following:

```
all(Prop,[]).
all(Prop,[H|Tl]) :- call(Prop,H), all(Prop,Tl).
```

to

```
all(Prop,[]).
all(Prop,[H|Tl]) :- Prop(H), all(Prop,Tl).
```

or a typed version thereof, turns a Prolog meta-program into a fully declarative program in higher-order logic. This simple example tells only a small part

M.J. Maher (Ed.): ASIAN 2004, LNCS 3321, pp. 93–108, 2004.

ภาคผนวก

Manuscript สำหรับ IEEE Transactions on Computer-Aided Design Submission

BOX: TCAD 2358: Acknowledgement of paper
of 319)

Sat, 30 Apr 2005 22:56:33 -0700
tcad@ece.orst.edu
tana@su.ac.th
tcad@ece.orst.edu
TCAD 2358: Acknowledgement of paper

ntana chantrapornchai:

ou for submitting the following paper to TCAD.

Design Exploration with Imprecise Latency and Register Constraints
C. Chantrapornchai, W. Surakumpolthorn, E. H-M. Sha

you sent for this paper printed fine.

takes about three months to make a decision about a paper.
be informed via e-mail when the decision has been made.

meantime, you may use your id and password to
e status of your paper periodically.

f of the editorial board, I would like to thank you for
bmission.

ly,
Nyaram
n-Chief
Transactions on CAD

essage has been scanned for viruses and
ous content by MailScanner, and is
d to be clean.
anner thanks transtec Computers for their support.

Design Exploration with Imprecise Latency and Register Constraints

C. Chantrapornchai[†]
Faculty of Science
Silpakorn University
Nakorn Pathom, Thailand

W. Surakumpolthorn
Faculty of Engineering
King Mongkut's Institute of Technology
Ladkrabang, Thailand

E. H-M. Sha[‡]
Dept. of Computer Science
University of Texas at Dallas
Richardson, U.S.A

Abstract

We propose a design exploration framework which consider impreciseness in design specification. In high-level synthesis, imprecise information is often encountered. We consider two types of impreciseness: impreciseness underlying on functional unit specifications and on contraints: latency and register. The framework is iterative and based on a core scheduling called, *Register-Constrained Inclusion Scheduling*. An example how the scheduling algorithm work is shown. We demonstrate the effectiveness of our framework for imprecise specification by exploring a design solution for a well-known benchmark, *Discrete Cosine Transform*, and *Voltera Filter*. The selected solution meets the acceptability criteria while minimizing the total number of registers.

Keywords: Imprecise Design Exploration, Scheduling/Allocation, Multiple design attributes, Imprecise information, Register constraint, Inclusion Scheduling

Corresponding author:

Chantana Chantrapornchai
Faculty of Science, Silpakorn University, Nakorn Pathom, Thailand 73000

ctana@su.ac.th

[†]This work was supported in part by the TRF under grant number MRG4680115.

[‡]This work was supported in part by TI University Program, NSF EIA 0103709, Texas ARP-009741-0028-2001 and NSF CCR-0309461, USA.

1 Introduction

In architectural level synthesis, imprecise information is almost unavoidable. For instance, an implementation of a particular component in a design may not be known due to several reasons. There may be various choices of modules implementing the functions or the component may have not been completely designed down to the geometry level. Even if it has been designed, variation in fabrication process will likely induce varying area and time measurements. Another kind of impreciseness or vagueness arises from the way a design is considered to be acceptable at architecture level. If a design with latency of 50 cycles is acceptable, what about a design with 51 cycles versus a design with 75 cycles? This even becomes imprecise especially when there are multiple conflicting design criteria. For example, is it worth to expand a latency by two cycles while saving one register and what about expanding 10 more cycles ? Effective treatment of such impreciseness in high level synthesis can undoubtedly play a key role in finding optimal design solutions.

In this paper, we propose a design exploration framework which considers imprecise information underlying in system specification and requirements. Particularly, we are interested in the latency and register constraints. However, the approach can be extended to handle other multiple design criteria. The system characteristics are modeled based on the fuzzy set theory. Register count is considered as another dimension of imprecise system requirement. The work in [3, 7] is used as a scheduling core in the iterative design refinement process. The imprecise schedule which minimizes the register usage is generated. If the schedule meets acceptability criteria, the design solution is selected. Otherwise, the resources are adjusted an the process is repeated. Our input system is modeled using a data flow graph with imprecise timing parameters. Such systems can be found in many digital signal processing applications, e.g., communication switches and real-time multimedia rendering systems. Imprecise specification on both system parameters and constraints can have a significant impact on component resource allocation and scheduling for designing these systems. Therefore, it is important to develop synthesis and optimization techniques which incorporate such impreciseness.

Most traditional synthesis tools ignore these vagueness or impreciseness in the specification. In particular, they assume the worst case (or sometimes typical case) execution time of a functional unit. The constraints are usually assumed to be a fixed precise value although in reality some flexibility can be allowed in the constraint due to the individual interpretation of an "acceptable" design. Such assumptions can be misleading, and may result in a longer design process and/or overly expensive design solutions. By properly considering the impreciseness up front in the design process, a good initial design solution can be achieved with provable degree of acceptance. Such a design solution can be used effectively in the iterative process of design refinement, and thus, the number of redesign cycles can be reduced.

Random variables with probability distributions may be used to model such uncertainty. Nevertheless, collecting the probability data is sometimes difficult and time consuming. Furthermore, some imprecise information may not be correctly captured by the probabilistic model. For example, certain inconspicuousness in the design goal/constraint specification, such as the willingness of the user to accept certain designs or the confidence of the engineer towards certain designs, cannot be described by probabilistic distribution.

Many researchers have applied the fuzzy logic approach to various kinds of scheduling problem. In compiler optimization, fuzzy set theory has been used to represent unpredictable real-time events and imprecise knowledge about variables [18]. Lee et.al. applied the fuzzy inference technique to find a feasible real-time schedule where each

task satisfies its deadline under resource constraints [23]. In production management area, fuzzy rules were applied to job shop and shop floor scheduling [27, 33]. Kaviani and Vranesic used fuzzy rules to determine the appropriate number of processors for a given set of tasks and deadlines for real-time systems [22]. Soma et.al. considered the schedule optimization based on fuzzy inference engine [32]. These approaches, however, do not take into account the fact that an execution delay of each job can be imprecise and/or multiple attributes of a schedule.

Many research results are available for design space exploration [1, 9, 15, 26]. All of these works differ in the techniques used to generate a design solution as well as the solution justification. These works, however, do not consider the impreciseness in the system attributes such as latency constraints and the execution time of a functional unit. Karkowski and Otten introduced a model to handle the imprecise propagation delay of events [19, 20]. In their approach, the fuzzy set theory was employed to model imprecise computation time. Their approach applies possibilistic programming based on the integer linear programming (ILP) formulation to simultaneously schedule and select a functional unit allocation under fuzzy area and time constraints. Nevertheless, the complexity of solving the ILP problem with fuzzy constraints and coefficients can be very high. Furthermore, they do not consider multiple degrees in acceptability of design solutions. Several papers were published on the resource estimation [10, 28, 31]. These approaches, however, neither consider multiple design attributes nor impreciseness in system characteristics.

Many research works related to scheduling and register allocation exist in high-level synthesis and compiler optimization area for VLIW architecture. Varatkar et. al. proposed a scheduling algorithm for multiprocessor systems which consider minimizing total system energy [34]. Shao et. al. presented instruction scheduling for loop applications which considers minimizing switching activity [30]. These work, however, do not consider register usage minimization. Chen et. al. proposed a loop scheduling for timing and memory operation optimization under register constraint [16]. The technique is based on multi-dimensional retiming. Eichenberger et. al. presented an approach for register allocation for VLIW and superscalar code via stage scheduling [13, 14]. Akturan and Jacome also proposed a scheduling algorithm which considers minimizing register usage for software pipelining [2]. The algorithm uses retiming and force directed scheduling and explores the trade-off between code size , performance, and register requirements. Wong et. al. developed a strategy to insert objective functions during scheduling and register allocation steps [35]. Their algorithm is called, scheduling FLOF, which attempts to minimize register usage subjected to the latency and resource constraints. Dani et. al. also presented a heuristic which uses stage scheduling to minimize register requirement. They also targeted at instruction level scheduling [11]. Zalamea et. al. presented hardware and software approach to minimize the register's usage targeting VLIW architecture [24, 25, 38]. On the software side, they proposed an extended version of modulo scheduling which considers register constraint, and register spilling. However, these work focus on loop scheduling and do not consider handling the imprecise system characteristics or specification.

In [3], the inclusion scheduling which takes the imprecise system characteristic was proposed. The algorithm was expanded and used in design exploration under imprecise system requirement as well as the estimation of resource bounds [5, 6, 8]. However, it does not take register criteria in creating a schedule.

In this paper, we particularly consider both imprecise latency and register constraints. We develop a design exploration framework under imprecise specification and constraints. The framework is iterative and based on the developed scheduling core, RCIS, *Register-Constrained Inclusion Scheduling* that takes imprecise information into account. Experimental results show that we can achieve an acceptable design solution with minimized number of registers.

This paper is organized as follows: Section 2 describes our models. It also presents some backgrounds in fuzzy set. Section 3 presents the iterative design framework. Section 4 presents the scheduling core (RCIS) used in the design exploration framework. It also addresses some issues when the register count is calculated during scheduling. An example how the scheduling algorithm works is shown in Section 5. Section 6 displays some experimental results. Finally, Section 7 draws a conclusion from our work.

2 Overview and Models

In this section, we first describe our model as well as problem description. Since in developing an inclusion schedule some fuzzy arithmetics is involved, we also review some basic concepts in fuzzy computation.

2.1 Model Descriptions

Operations and their dependencies in an application are modeled by a vertex-weighted directed acyclic graph, called a *Data Flow Graph*, $G = (\mathcal{V}, \mathcal{E}, \beta)$, where each vertex in the vertex set $\mathcal{V}$ corresponds to an operation and $\mathcal{E}$ is the set of edges representing data flow between two vertices. Function β defines the type of operation for node $v \in \mathcal{V}$. Figure 1

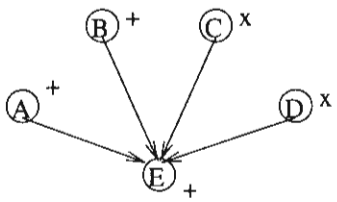


Figure 1: Test1: Data flow graph example

shows a five-node data flow graph, where $\mathcal{V} = \{A, B, C, D, E\}$, $\mathcal{E} = \{A \rightarrow E, B \rightarrow E, C \rightarrow E, D \rightarrow E\}$, ($u \rightarrow v$ defines a directed edge from u to v), $\beta(A) = \beta(B) = \beta(E) =$ add, and $\beta(C) = \beta(D) =$ multiply.

Operations in a data flow graph can be mapped to different functional units which in turn can have varying characteristics. Such a system must also satisfy certain design constraints, for instance, power and cost limitations. These specifications are characterized by a tuple $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$, where $\mathcal{F}$ is the set of functional unit types available in the system, e.g., {add, mul}. $\mathcal{A}$ is $\{A_f : \forall f \in \mathcal{F}\}$. Each A_f is a set of tuples $(a_1, \ldots, a_k)$, where a_1 to a_k represent attributes of particular f. In this paper, we use only latency as an example attribute. (Note that our approach is readily applicable to include other constraints such as power and area). Hence, $A_f = \{x : \forall x\}$ where x refers to the latency attribute of f. $\mathcal{M}$ is $\{\mu_f : \forall f \in \mathcal{F}\}$ where μ_f is a mapping from A_f to a set of real number in [0,1], representing a possible degree of using the value. Finally, $\mathcal{Q}$ is a function that defines the degree of a system being acceptable for different system attributes. If $\mathcal{Q}(a_1, \ldots, a_k) = 0$ the corresponding design is totally unacceptable while $\mathcal{Q}(a_1, \ldots, a_k) = 1$, the corresponding design is definitely acceptable.

Using a function $\mathcal{Q}$ to define the acceptability of a system is a very powerful model. It can not only define certain constraints but also express certain design goals. For example, one is interested in designing a system with latency under 500 and register count being less than 6 respectively. Also, the smaller latency and register count, the

better a system is. The best system would have both latency and register count being less than or equal to 100 and 1 respectively. An acceptability function, $Q(a_1, a_2)$ for such a specification is formally defined as:

$$Q(a_1, a_2) = \begin{cases} 0 & \text{if } a_1 > 500 \text{ or } a_2 > 6 \\ 1 & \text{if } a_1 \leq 100 \text{ and } a_2 \leq 1 \\ F(a_1, a_2) & \text{otherwise,} \end{cases} \quad (1)$$

where F is assumed to be linear functions, e.g., $F(a_1, a_2) = 1.249689(a_1 + 2a_2) - 0.001242$ which returns the acceptability between $(0, 1)$.

Figure 2 illustrates Equation (1) graphically. In this constraint, we express the weighted sum of the two criteria which gives the preference to minimizing register count twice as much as minimizing latency, that is latency: register count is 1:2. In other words, we are willing to spend two more latency cycles if one register can be saved.

Figure 3 shows the projection of Equation (1) on latency and possibility axis.

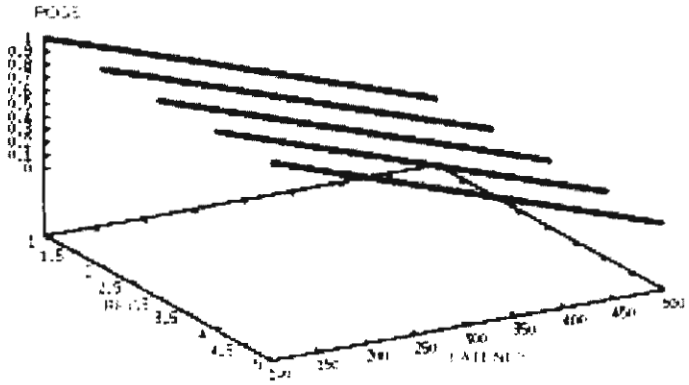


Figure 2: Imprecise constraint where Latency : Register is 1 : 2.

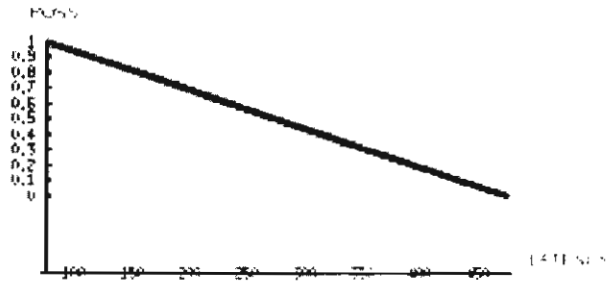


Figure 3: Projection of constraint in Figure 2.

In general, one may model any criteria by

$$\mathcal{Q}(a_1, a_2, ...a_n) = \begin{cases} 0 & \text{if } a_1 > p_{1_{max}} ... \text{ or } a_n > p_{n_{max}} \\ 1 & \text{if } a_1 \leq p_{1_{min}} ... \text{ and } a_n \leq p_{n_{min}} \\ F(w_1 a_1 + w_2 a_2 + ..w_n a_n) & \text{otherwise.} \end{cases} \quad (2)$$

where

For example, we can simply replace the register constraint by others such as power. Figure 4(b) depicts an example of the specification concerning the tradeoff graphically where $w_1 = 2, w_2 = 1$. Hence, F refers to a z-shaped curve function which produces a smooth transition between two given points. Figure 4(c) shows the projection of the 3-dimensional acceptability model to the latency and acceptability plane. In this figure, each z curve represents a projection of $\mathcal{Q}$ function to a latency-acceptability plane. An inner curve (tighter latency constraint) corresponds to larger power values. Based on the acceptability model, a design with high acceptability implies an optimized design towards certain goals.

Based on the above model, the combined scheduling/binding we intend to solve can be formulated as follows:

Given a specification containing $S = (\mathcal{F}, \mathcal{A}, \mathcal{M}, \mathcal{Q})$, $G = (\mathcal{V}, \mathcal{E}, \beta)$, *and acceptability level* α, *find a schedule under functional unit and register constraints for each* f *in* $\mathcal{F}$ *whose the acceptability degree is greater than or equal to* α *subject* Q.

2.2 Fuzzy Sets

In this section, we give a quick review of fuzzy set theory as it relates to our work. Readers familiar with the theory can skip this part.

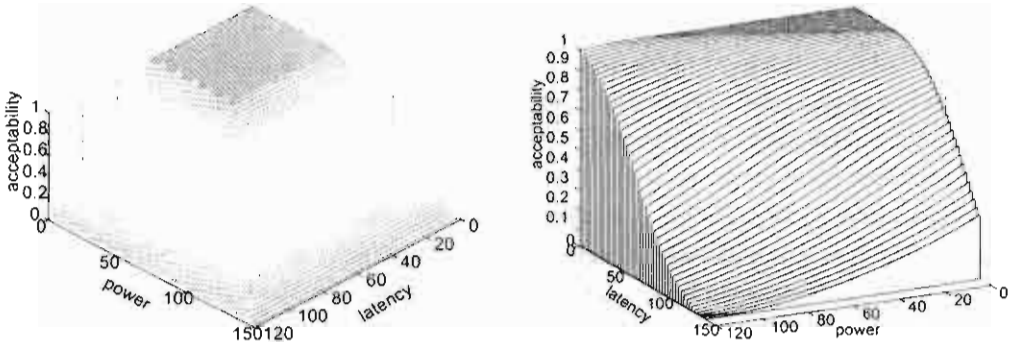
Fuzzy sets, proposed by Zadeh, represent a set with imprecise boundary [36, 37]. In classical (crisp) sets, an element can either be a member of a set or not at all; hence, its membership degree is either 1 or 0. A fuzzy set is defined by assigning each element in a universe of discourse its membership degree in the unit interval [0, 1], conveying to what degree x is a member in the set. This membership value can be defined as a membership function of an element in the set, $\mu(x) : x \rightarrow [0, 1]$.

A fuzzy set is said to be *normal* if there exists at least one member in the set whose membership value is unity. A *convex* fuzzy set is defined as: for any x, y, and z in the fuzzy set A, the relation $x < y < z$ implies that $\mu_A(y) \geq \min(\mu_A(x), \mu_A(z))$. A fuzzy number is a normal, convex fuzzy set defined on the real line R. Let A and B be fuzzy numbers with membership functions $\mu_A(x)$ and $\mu_B(y)$, respectively. Let $*$ be a set of binary operations $\{+, -, \times, \div, \min, \max\}$. The arithmetic operations between two fuzzy numbers, defined on $A * B$ with membership function $\mu_{A*B}(z)$, can use the extension principle, by [17]:

$$\mu_{A*B}(z) = \bigvee_{z=x*y} (\mu_A(x) \wedge \mu_B(y)) \quad (3)$$

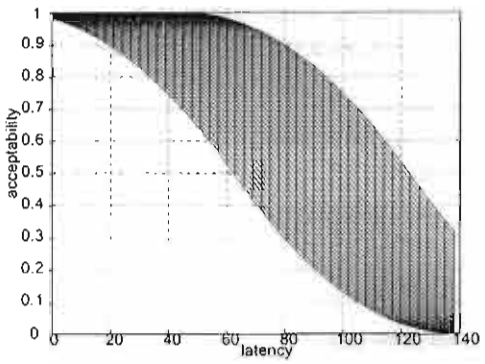
where $\vee$ and $\wedge$ denote max and min operations respectively.

Fuzzy arithmetic is used to compute an arithmetic operation between two fuzzy numbers. Figure 5(a) shows a fuzzy number A, which is assumed to be normal triangular-shaped lied on an real line. In this figure, let A be assigned with the confidence interval $(2, 6)$. The most possible value of A is 4 since its *confidence level* or *presumption level* is 1. Similarly, Figure 5(b) shows the fuzzy number with the confidence interval $(3, 7)$ representing B. Figure 5(c)



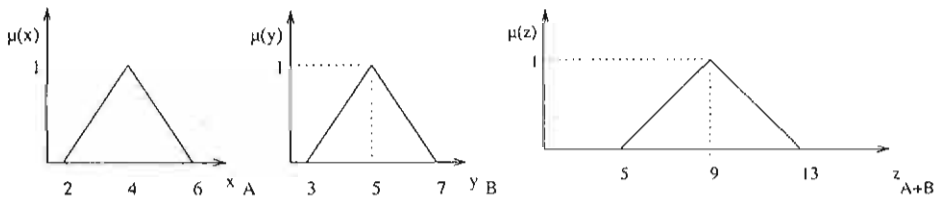
(a) Linear acceptability

(b) z curve acceptability with trade-off



(c) Latency acceptability curves corresponding to different power constraints derived from Figure 4(b)

Figure 4: Various kinds of acceptability functions



(a) A

(b) B

(c) A + B

Figure 5: Adding two fuzzy numbers, A and B

demonstrates a graphical result of adding two fuzzy numbers defined on the integer line from Figures 5(a)–5(b), using Equation (3).

In order to compare two fuzzy numbers, several methods can be used. All of these methods are based on selecting a representative for each fuzzy number and compare the representatives [21]. One way to obtain the representatives is using the *removal* with respect to k, which is a measure of distance from k, computed by $R(A,k) = \frac{1}{2}(R_l(A,k) + R_r(A,k))$, where A is a fuzzy number, k is a reference position on the x-axis, R_l is the area bounded by the left side of the curve and the line $x = k$ and similarly for the right side, R_r. Another can be *mode*, which uses the value x such that $\mu(x) = \max_i\{\mu(x_i)\}$ for all x_i in the fuzzy set. *Divergence* is another way to calculate the representatives. It represents the width of the set which is computed by $x_{max} - x_{min}$. In addition, the defuzzified value can be used to represent the fuzzy set. Several defuzzified methods can be found in [29].

Based on the fuzzy set concept, we model the relationship between functional units and possible characteristics such that each functional unit is associated with a fuzzy set of characteristics. Given a functional unit f and its possible characteristic set A_f let $\mu_f(a) \in [0,1], \forall a \in A_f$, describe a possibility of having attribute a for a functional unit f. Let us focus on the timing attribute. A fuzzy set of timing characteristic of a functional unit f may be $\{ \frac{10}{.2}, \frac{20}{.4}, \frac{35}{1}, \frac{70}{.7} \}$. That is f may use 10, 20, 35, and 70 time units to execute with different possibility, i.e., $\mu_f(10) = .2, \mu_f(20) = .4, \mu_f(35) = 1, \mu_f(70) = .7$.

3 Iterative Design Framework

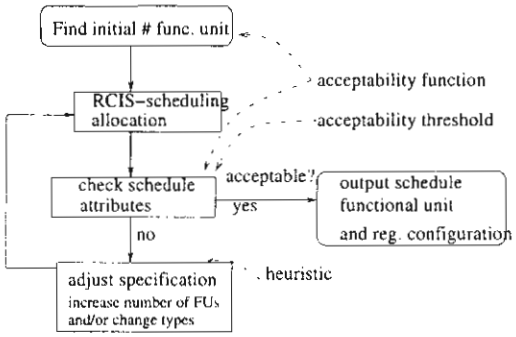


Figure 6: Design solution finding process using RCIS

Figure 6 presents an overview of our iterative design process for finding a satisfactory solution. One may estimate the initial design configuration with any heuristic for example using ALAP, and/or ASAP scheduling [8]. The RCIS scheduling and allocation process produces the imprecise schedule attributes which are used to determine whether or not the design configuration is acceptable.

RCIS is a scheduling and allocation process which incorporates varying information of each operation. It takes an application modeled by a directed acyclic graph as well as the number of functional units that can be used to compute this application. Then, the schedule of the application is derived. This schedule shows an execution order of operations

in the application based on the available functional units. The total attributes of the application can be derived after the schedule is computed. The given acceptability function is then checked with the derived attributes of the schedule.

In order to determine whether or not the resource configuration is satisfied the objective function, we use the *acceptability threshold*. If the schedule attributes lead to the acceptability level being greater than the threshold, the process stops. Otherwise, the resource configuration is adjusted using a heuristic and this process is repeated until the design solution cannot be improved or the design solution is found.

4 Register-Constraint Inclusion Scheduling

In this section, we present the register-constraint inclusion scheduling (RCIS) algorithm.The algorithm is based on the *inclusion scheduling* core presented in Algorithm 4.1. The algorithm evaluates the quality of the schedule by considering imprecise register criteria which will be discussed later subsections

Specifically, inclusion scheduling is a scheduling method which takes into consideration of fuzzy characteristics which in this case is fuzzy set of varying latency values associated with each functional unit. The output schedule, in turn, also consists of fuzzy attributes. In a nutshell, inclusion scheduling simply replaces the computation of accumulated execution times in a traditional scheduling algorithm by the fuzzy arithmetic-based computation. Hence, fuzzy arithmetics is used to compute possible latency from the given functional specification. Then, using a fuzzy scheme, latency of different schedules are compared to select a functional unit for scheduling an operation. Though the concept is simple, the results are very informative. They can be used in many ways such as module selection [4]. Algorithm 4.1 presents a list-based inclusion scheduling framework.

Algorithm 4.1 (Register-Constrained Inclusion scheduling)
Input: $G = (\mathcal{V}, \mathcal{E}, \beta)$, *Spec* $= (F, \mathcal{A}, \mathcal{M}, Q)$, *and* $N =$*#FUs*
Output: *A schedule* S, *with imprecise latency*

```
1  Q = vertices in G with no incoming edges                          // finding root nodes
2  while Q ≠ empty do
3        Q = prioritized(Q)
4        u = dequeue(Q); mark u scheduled
5        good_S = NULL;
6        foreach f ∈ {f_j : where f_j is able to perform β(u), 1 ≤ j ≤ N} do
7          temp_S = assign_heuristic(S, u, f)                        // assign u at FU f
8          if Eval_Schedule_with_Reg(good_S, temp_S, G, Spec)
9            then good_S = temp_S fi od
10       S = good_S                                                  // keep good schedule
11       foreach v : (u, v) ∈ E do
12         indegree(v) = indegree(v) − 1
13         if indegree(v) = 0 then enqueue(Q, v) fi od
14 od
15 return(S)
```

After node u is assigned to f, the imprecise attributes of the intermediate schedule, is computed. *Eval_Schedule_with_Reg* compares the current schedule with the "best" one found in previous iterations. The better one of the two is then chosen and the process is repeated for all nodes in the graph.

In Algorithm 4.1, fuzzy arithmetic simply takes place in routine *Eval_Schedule_with_Reg* (Line 8). In this routine, we also consider the register count used in the schedule. Since an execution time of a node is imprecise, the life time of a node is imprecise. Traditionally, a life time of a node depends on the location of the node's successors in the schedule. That is the value produced by the node must be held until its successors have consumed it. For simplicity, let the successors consume the value right after they start.

Recall that a node's execution time is a fuzzy set, where the membership function is defined by $\mu(x) = y$. It implies that the node will take x time units with possibility y. Consequently, a start time and finished time of a node are fuzzy numbers. To be able to calculate fuzzy start time and finished time, we must assume that all nodes have been assigned to functional units already. We assume that resource binding and order of nodes executing in these resources are given based on the modified DFG (i.e, *scheduled DFG*). The modified DFG is just the original DFG where extra edges due to independent nodes executing in the same functional units are inserted (as constructed in Algorithm 4.3).

4.1 Imprecise Timing Attributes

In the following, we present basic terminologies used in the algorithm which calculates register usage under impreciseness.

Definition 4.1 *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, a fuzzy start time of node $u \in V$, $FST(u)$ is a fuzzy set whose membership degree is defined by $\mu_{FST(u)}(x) = y$, i.e, node u may start at time step x with possibility y.*

For nodes that are executed at time step 0 in each functional unit, $FST(u) = 0$, which is a crisp value.

Definition 4.2 *For $G = (\mathcal{V}, \mathcal{E}, \beta)$, and a given schedule, a fuzzy finished time of node $u \in V$, $FFT(u)$ is a fuzzy set whose membership degree is defined by $\mu_{FFT(u)}(x) = y$, i.e, node u may finish at time step x with possibility y.*

Hence, $FFT(v) = FST(v) + EXEC(v)$, where $EXEC(v)$ is the fuzzy latency of v. When considering earliest start time of a node, $FST(v) = \max_i(FFT(u_i)) + 1, \forall u_i \rightarrow v$.

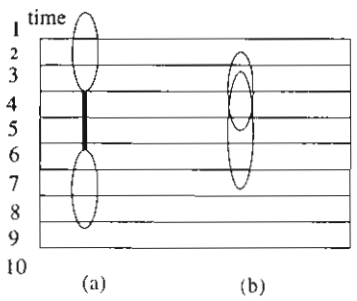


Figure 7: A view of fuzzy start time and finished time

The general idea of using fuzzy numbers is depicted in Figure 7 for both start time and finished time. Circles denote the fuzzy boundary which means that the start time and finished time boundary of a node is unclear. Indeed, they may also be overlapped as shown in Figure 7(b). When a node occupies a resource at a certain time step, a possibility value is associated with the assignment.

Traditionally, when the timing attribute is a crisp value, the start time x and finished time y of a node form an integer interval [x...y], which will be used to compute the register usage in the schedule. In our case, a fuzzy life time for node u contains two fuzzy sets: FST(u) and MFFT(u), the maximum of start time of all its successors.

Definition 4.3 *For* $G = (\mathcal{V}, \mathcal{E}, \beta)$, *and a given schedule, fuzzy life time of node* u, FLT(u) *is a pair of* [FST(u), MFFT(u)], *where* $\mu_{MFFT(u)} = FFT(u) + \max(FST(v_i))$, *where* $u \rightarrow v_i \in \mathcal{E}$ *and* $+$, max *are fuzzy addition and fuzzy maximum respectively.*

Given FLT(u), let min_st be the minimum time step from FST(u) whose $\mu_{FST(u)}$ is nonzero, and max_st be the maximum time step from FST(u) whose $\mu_{FST(u)}$ is nonzero. Similarly, let min_fin be the minimum time step from MFFT(u) whose $\mu_{MFFT(u)}$ is nonzero, and let max_fin be the maximum time step from MFFT(u) whose $\mu_{MFFT(u)}$ is nonzero. Without loss of generality, assume that FST(u) and MFFT(u) are sorted in the increasing order of the time step. We create a fuzzy set IFST(u), mapping for a discrete time domain [min_st...max_st] to a real value in [0..1], showing the possibility that at time step x, node u will occupy a register for FST(u) and likewise for IMFFT(u) for MFFT(u) as in Definitions 4.4–4.5.

Definition 4.4 $G = (\mathcal{V}, \mathcal{E}, \beta)$, *a given schedule,* [min_st...max_st] *and* FLT(u)

$$\mu_{IFST(u)}(c) = \begin{cases} 0 & \text{if } c < \text{min\_st or } c > \text{max\_st} \\ \max_{\forall x, \text{min\_st} \le x < y, y = \max(FST(u)) \text{ and } y < c} (\mu_{FST(u)}(x)) & \text{otherwise} \end{cases}$$

Definition 4.5 $G = (\mathcal{V}, \mathcal{E}, \beta)$, *a given schedule,* [min_fin...max_fin] *and* FLT(u)

$$\mu_{IMFFT(u)}(c) = \begin{cases} 0 & \text{if } c < \text{min\_fin or } c > \text{max\_fin} \\ \max_{\forall x, y < x \le \text{max\_fin}, y = \max(MFFT(u)) \text{ and } y < c} (\mu_{MFFT(u)}(x)) & \text{otherwise} \end{cases}$$

From the above calculation, we assume that for any two starting time value $a, b \in FST(u)$ where $a < b$, if node u starts at time a, it will be already started at time b. For MFFT(u), when $a < b$, $a, b \in MFFT(u)$, if the value for node u will not be needed at time a, it will not be needed at time b and vice versa. Thus, Definitions 4.4–4.5 give the following properties.

Property 4.1 *The possibility of* IFST(u) *is in nondecreasing order.*

Property 4.2 *The possibility of* IMFFT(u) *is in non-increasing order.*

From IFST(u) and IMFFT(u), we merge the two sets to create a fuzzy interval for a node by defining Definition 4.6.

Definition 4.6 $G = (\mathcal{V}, \mathcal{E}, \beta)$, *a given schedule,* IFST(u) *and* IMFFT(u).

$$\mu_{IFLT(u)}(c) = \begin{cases} 0 & \text{if } c < \min(min\_st, min\_fin) \text{ or} \\ & c > \max(max\_st, max\_fin) \\ \max(\mu_{IFST(u)}(c), \mu_{IMFFT(u)}(c)) & \text{if } min\_st \leq c \leq max\_st \\ & \text{or } min\_fin \leq c \leq max\_fin \\ 1 & \text{otherwise} \end{cases}$$

After we compute the fuzzy life time interval for each node, we can start compute register usage for each time step.

4.2 Register Usage Calculation

Once a scheduled DFG is created, FST(u) and FFT(u) must be calculated for all $u \in V$. Figure 8 displays the meaning of fuzzy life time implied by Definitions 4.4–4.5. SA and FA denote the fuzzy start time and the fuzzy finished time of node A respectively. Similarly, SB and FB denote the start time and the fuzzy finished time of node B The fuzzy life times of A and B are shown in the filled boxes on the right side.

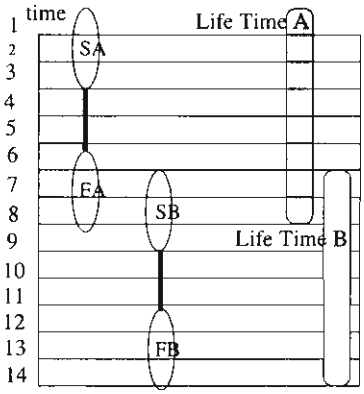


Figure 8: Relationship between scheduled nodes and life time

In the figure, the life time of A and the life time of B may overlap. Traditionally, when the timing attribute is precise, the overlapped interval implies that two registers are needed during these time steps. In particular, during time steps 7 and 8, two registers are needed.

When an execution time becomes a fuzzy number, each box still implies that one register is needed. However, the derived possibility associated with a time step indicates that that node may not actually exist during the time step. For example, node may start later or finished earlier. In other words, there is a possibility that a node may not use such a register. With this knowledge, the register may be shared with others with high possibility. Consider the overlap interval in Figure 8 at time step 7. One or two registers may be used with some possibility. This depends on whether the dependency between $A \rightarrow B$ exists. If edge $A \rightarrow B$ exists in the original data flow graph, the total register count would be one. Notice that in this case, the intersection of IFLT(A) and IFLT(B) is not empty. On the contrary, if A

and B are independent, the total register count would be two although the intersection may not be empty as well. This issue must be considered in calculating register usages.

4.3 Algorithms

Algorithm 4.2 presents a framework in evaluating fuzzy latency and register counts of a schedule. This algorithm is called after Line 7 in Algorithm 4.1 which already assigns the start time for each node.

In Algorithm 4.2, Line 6 invokes Algorithm 4.3 to calculate the life time of all nodes in schedule and find the maximum register usage. The register usage is then kept in $Reg[S_i]$ for a schedule S_i. Next, the latency of the whole schedule is then calculated. Note that after invoking Algorithm 4.3, necessary timing attributes for all nodes in G_0 can be obtained. The latency of the schedule is obtained by just fuzzy maximizing the finished time of all leaves in G_0. Line 9 merges latency and register usage attributes of the schedule using some heuristic function. The combined attribute is denoted as a *quality* of the schedule. This quality is then compared in Line 13 to select the best one.

Algorithm 4.2 (Eval_Schedule_with_Reg)

Input: *schedules* S_1, S_2, $G = (\mathcal{V}, \mathcal{E}, \beta)$, *and Spec* $= (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$

Output: *1 if* S_1 *is better than* S_2, *0 otherwise.*

1 $G_0 = (\mathcal{V}_0, \mathcal{E}_0, \beta)$ where $\mathcal{V}_0 = \mathcal{V} - \{\text{unscheduled nodes}\}, \mathcal{E}_0 = \emptyset$
2 **foreach** schedule $S_i = S_1$ **to** S_2 **do**
3 $\mathcal{E}_0 = \{(u, v) : u, v \in \mathcal{V}_0$, if u, v in same f.u. in S_i
4 and v is immediately after $u\}$
6 Calculate register usage for G_0 using Algorithm 4.3
7 Let W is a set of leaves in G_0
8 $latency[S_i] = fuzzymax\_time(W)$
9 $quality[S_i] = Combine(latency[S_i], Reg[S_i])$
10 **od**
12 // comparing the overall attributes of both schedules
13 **return**$(compare(quality[S_1], quality[S_2]))$

Algorithm 4.3 (Calculate_Register_Count)

Input: *Scheduled Graph* G_0 *for schedule S and, original DFG* $G = (\mathcal{V}, \mathcal{E}, \beta)$ *Spec* $= (F, \mathcal{A}, \mathcal{M}, \mathcal{Q})$

Output: $Reg[S]$ *contains register counts needed and its possibility*

1 Calculate $FLT(u)\ \forall u \in G_0$ by Definition 4.3
2 Calculate $IFLT(u)\ \forall u \in G_0$ by Definitions 4.4–4.5
3 Let *max_cs* be max. finished time , $\forall u \in G_0$
4 **for** $cs = 1$ **to** *max_cs* **do**
5 $(RegAt[cs].reg, RegAt[cs].poss) = Count\_Node(IFLT, cs, G_0)$ **od**
6 $\forall n, FReg[n] = 0$
7 **for** $cs = 1$ **to** *max_cs* **do**
8 $FReg[RegAt[cs].reg].reg = RegAt[cs].reg$
9 $FReg[RegAt[cs].reg].poss =$

```
10      max{FReg[RegAt[cs].reg].poss, RegAt[cs].poss} od
12  Reg[S] = FReg
```

In Algorithm 4.3, *RegAt* stores maximum number of registers needed at each cs and its associated possibility. The values are obtained by Algorithm *Count_Node*. Lines 7–10 summarize the overall number of registers needed and its possibility. Algorithm *Count_Node* is described in Algorithm 4.4.

Algorithm 4.4 (Count_Node)

Input: IFLT, G_0, cs

Output: *# registers needed and its possibility at* cs

```
1  node_set = {nodes occupy reg at cs}
2  set G0 in topological order
3  Let sorted_node be node_set sorted in by sorted G0
4  poss = 0, reg = 0
5  ∀i ∈ sorted_node, i.ok = FALSE, i.count = FALSE
6  for every i ∈ sorted_node do
7     for j = i + 1 to last node in sorted_node do
8        if i.ok = TRUE and i.count = FALSE
9          then
10              reg + +; poss = max(poss, μIFLT(i)(cs)
11              i.count = TRUE fi
13       if FindPath(i, j)
14         then j.ok = FALSE // don't count descendant fi
15    od
16    Let j be the last node in sorted_node
17    if j.ok = TRUE
18      then
19           reg + +; poss = max(poss, μIFLT(j)(cs)
20           j.count = TRUE fi
21 od
22 return (reg, poss)
```

In Algorithm 4.4, our heuristic only attempts to consider the ancestor at the current time step. In other words, we assume that the ancestor finishes first and then its descendants can start. Flag ok uses to indicate that the associated node should be counted at the current step or not. If it is a descendant of any of nodes in the current step, the flag will be disable. Since the schedule contains every node, the descendant will be started eventually. *reg* and *poss* store the current number of counted nodes and maximum possibility. At Line 3, the nodes currently in this time step indicated by IFLT are sorted in the topological order according to G_0. Then we extract each node in the sorted list to check if any pair are dependent by using *FindPath* in Line 13. In the loop, it selectively marks descendant nodes in the current step.

Let us consider the complexity of Algorithm 4.4. The time complexity is dominated by Lines 6–21, which is $O(|V|^2(|V|+|E|))$, since for DAG, *FindPath* takes $O(|V|+|E|)$.

In Algorithm 4.3, the calculation for FLT(u) depends on FST(u) and MFFT(u). Let N_1 be the number of discrete points in FST(u) and MFFT(u). Lines 1–2 perform the calculation whose upper bound is of $O(N_1|V||E|)$. The computation for IFLT(u) is simply a double loop for each node. In overall, Algorithm 4.3 runs in polynomial time.

5 Example

We integrate Algorithm 4.2 into Algorithm 4.1. The new algorithm is called Register-Constrained Inclusion Scheduling (RCIS). In this section, we present an example which shows the calculation for FLT and the resulting schedule. Then we discuss the results on other benchmarks.

Consider the simple DFG presented in Figure 9. Assume that there are four general functional units available, where FU1 and FU3 have the same characteristics as well as FU2 and FU4 as shown in Table 1. In the figure, Columns "(lat,poss)" show the latency and its possibility of having the latency value if the nodes are executed in a functional unit. In this case, FU1 and FU3 have the same characteristics while FU2 and FU4 have the same characteristics.

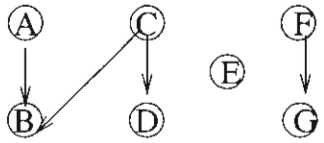


Figure 9: A simple DFG example.

FUs	(lat,poss)		(lat,poss)		(lat,poss)		(lat,poss)	
	lat	poss	lat	poss	lat	poss	lat	poss
FU1,FU3	5	0.05	10	1	15	0.9	23	0.1
FU2,FU4	7	0.5	12	0.7	17	1	29	0.05

Table 1: Functional unit characteristics

Given the system specification shown in Figure 10, where register axis contains a discrete value ranged in [1..7] and latency axis ranged in [1..200]. In the figure, we use the weighted sum as a criteria similar to Equation (1), where latency : register count is 1:10.

According to Section 4, Figure 11(a) shows the resulting schedule we obtain. We notice that FU1 and FU3 are preferable. To calculate FST(u), we assume a heuristic where a node starts as early as possible. From this schedule, consider node B. Figure 12(a) presents FLT(A) containing FST(A) and MFFT(A). Figure 12(b) presents FLT(B). For FST(A), possibility y at time x represents a possibility that node A occupies register at time x with respect to the schedule, denoted by rectangles. Notice that for A, there is only one possible start time whose possibility is one. Similarly for MFFT(A), possibility y at time x represents a possibility that the life time of node A ends at time x,

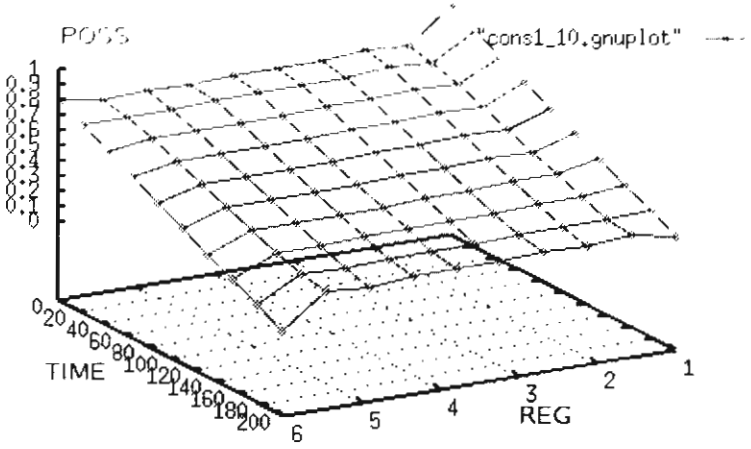


Figure 10: Constraint for Figure 9.

FU1	FU2	FU3	FU4
A	-	E	-
F	-	C	-
G	-	D	-
B	-	-	-

(a)

FU1	FU2	FU3	FU4
A	F	E	-
G	-	C	-
B	-	D	-

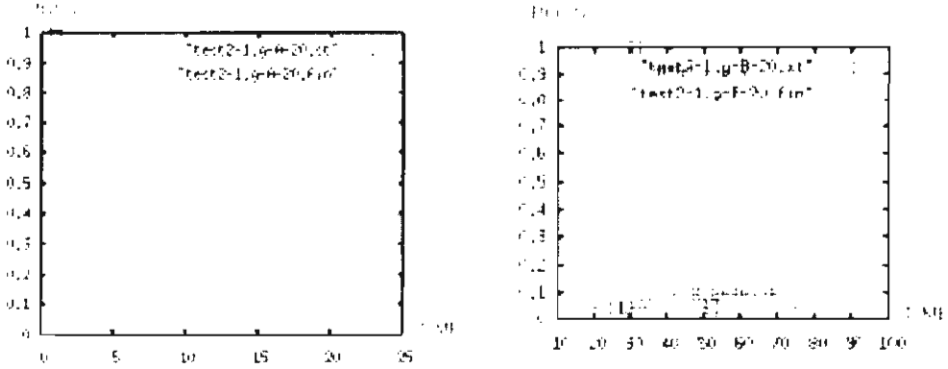
(b)

Figure 11: (a) Schedule obtained RCIS for Figure 9 (b) Schedule obtained using the original inclusion scheduling.

denoted by triangles in the Figure. For two dependent nodes A, B, Figure 13 compares FLT(A) and FLT(B). We can see that FST(B) overlaps with MFFT(A). Figures 14(b)– 14(a) shows the FLT(u) all the nodes.

We summarize the register count and its possibility value for each time step as shown in Figure 15. Then we conclude that the register usage as following: **(1,0.1)** and **(2,1)**. It implies that at some control step, 1 register is needed with very low possibility, e.g. 0.05 and 0.1. The maximum possible finished time of the schedule is at 92 with possibility 0.1. With this schedule, the average weighted sum of latency and register is 79.53. Considering only the average latency, the value is 52. Compared to the constraint, with latency 52 and register count 2, the acceptability degree is 0.76. In fact, this gives the same acceptability level as the original inclusion scheduling whose average latency is 41 and the maximum register count is 3. The schedule of this case is given in Figure 11(b).

We have tried to experiment on larger graphs. For instance, we expand the graph in Figure 9 by adding nodes, and following edges $\{H \rightarrow J, I \rightarrow J\}$ (See Figure 16). Using the same constraint as in Figure 10, and the same functional unit specification in Table 1 while allowing 4 functional units, RCIS yields the schedule with average latency 57 and



(a) (b)

Figure 12: (a) FLT(A) (b) FLT(B).

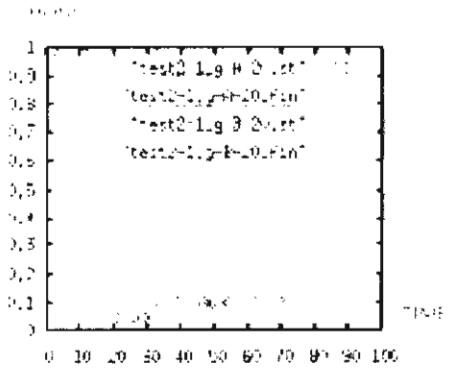
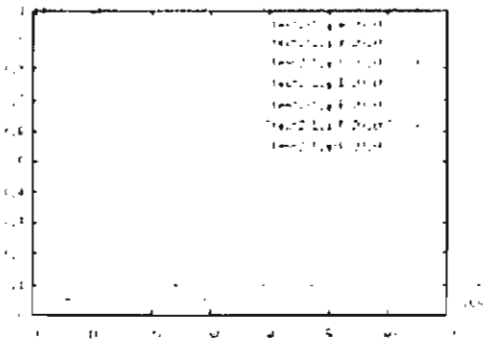
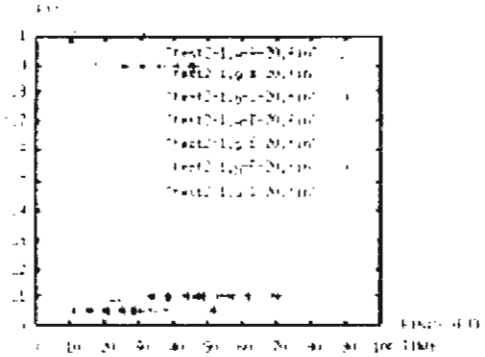


Figure 13: FLT(A) and FLT(B).



(a)



(b)

Figure 14: FLT(u), for all nodes in Figure 9 (a) FST(u) (b) MFFT(u).

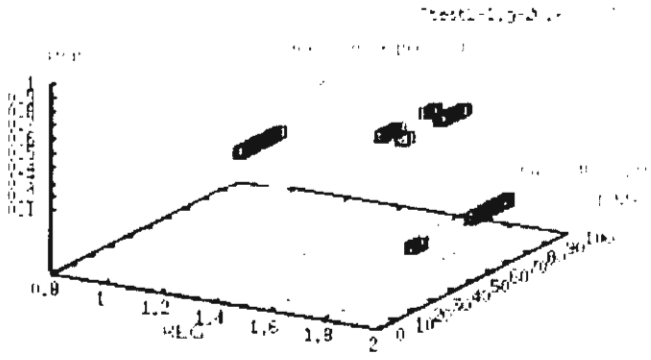


Figure 15: Register counts and possibility each time step.

maximum registers of 2. This yields acceptability value 0.74. Compared to the latency-based inclusion scheduling, it results in average latency 37 with the maximum register of 4 which also gives the same acceptability level.

6 Experimental Results

We consider the expereiments on exploring design solutions for Discrete Cosine Transform (DCT) [12] and Voltera filter benchmark [8].

6.1 Discrete Cosine Transform

Consider a well-known benchmark, Discrete Cosine Transform, containing 48 nodes. Assume the same functional unit specification for both adders and multipliers and the constraint in Figure 17 where the register axis is [1..12] and the latency axis is [1..500]. We assume the functional unit characteristics similar to the example in the previous section. In the experiment as shown in Table 2. We compare the results obtained from various cases of varying the number of functional units. The results are shown in Table 3. Columns "RCIS"and "IS" compare the performance of the schedule by Register-Constrained Inclusion Scheduling and the original inclusion scheduling (IS). Row "Avg Latency" shows the weighted sum of latency for each case. Row "Max Reg" displays the maximum number of registers. Row "Acceptability" shows the acceptability value obtained using the "Avg latency" and "Max Reg". Row "Max Latency" presents the maximum latency values and Row "Avg Weight" presents weighted sum value for RCIS and IS. For RCIS, recall that $w_1 = 1$ and $w_2 = 10$ and for IS, this is the same value as shown in Row "Avg Latency" since we only consider minimizing latency. Tables 4–5 shows the summarized possibility values for using certain register counts for RCIS and IS respectively. It is obvious that IS attempts to minimize latency while not considering the register usage. From these tables, we can achieve about the same acceptability (and even better acceptability in some case) with fewer number of registers, which is upto **37%** saving for the number of registers for the case of 7 adders and 5 multipliers. Among all these cases, we see that the configuration with 5 adders and 4 multipliers should

be the best. Consider the running time. For all the cases, the maximum running time is approximately 1 minute 50 seconds to achieve the results for 7 adders and 5 multipliers under Pentium 4 2.8GHz, 1GB RAM.

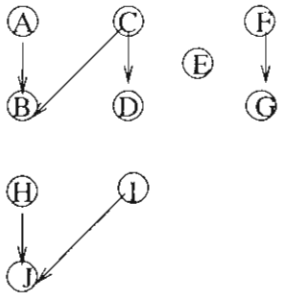


Figure 16: A simple DFG example 2.

FUs	(lat,poss)		(lat,poss)		(lat,poss)		(lat,poss)	
	lat	poss	lat	poss	lat	poss	lat	poss
Adder	5	0.05	10	1	15	0.9	23	0.1
Multiplier	7	0.5	12	0.7	17	1	29	0.05

Table 2: Adder and multiplier characteristics

	5 adds 4 muls		6 adds 4 muls		6 adds 5 muls		7 adds 4 muls		7 adds 5 muls	
	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS
Avg Latency	122	111	132	98	117	99	124	104	127	94
Max Reg	**6**	8	7	10	8	10	7	10	7	11
Acceptability	**0.719**	0.704	0.69	0.69	0.694	0.691	0.699	0.683	0.691	0.683
Max Latency	226	252	296	224	213	197	255	226	230	179
Avg Weight	188	111	198	98	206	99	210	104	209	94

Table 3: Comparison of RCIS and IS when varying the number of functional units.

6.2 Votera filter

We present experimental results on voltera filter benchmark, containing 27 nodes, where 10 nodes require adder units and the rest requires multiplier units. Assume that we have two types of functional units: adder and multiplier, whose latencies are as shown according to Table 1. In the figure, an adder may have different latency values with the given possibility. Columns "lat" and "pos" show the latency and its possibility of having the latency value for each adder

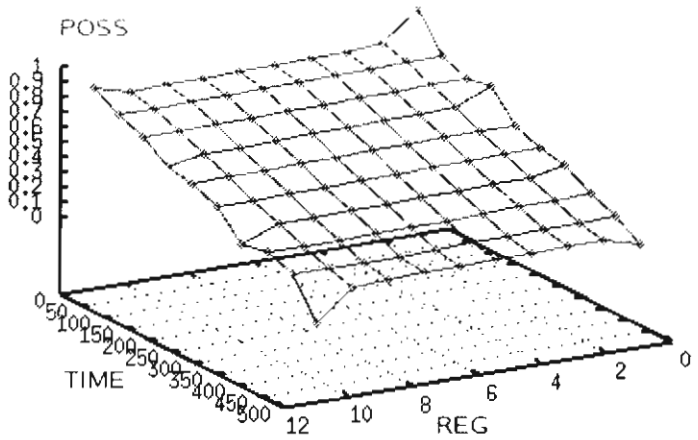


Figure 17: Constraint for DCT.

#reg	2	3	4	5	6	7
poss	0.1	3	0.1	0.1	1	1

Table 4: Possibility values of register counts for case 7 adders and 5 multipliers for RCIS.

#reg	2	4	5	6	7	8	10	11
poss	0.05	0.05	1	0.05	1	0.1	1	1

Table 5: Possibility values of register counts for case 7 adders and 5 multipliers for IS.

and multiplier. Thus, if the nodes are executed in the functional unit, the node may have variable latency values as well.

Assume the constraint is depicted in Figure 18 where the register axis is [1..7] and the latency axis is [200..700]. We demonstrate by considering various design configuration of varying the number of functional units using RCIS and original inclusion scheduling as a scheduling core in the design exploration. Due to the characteristic of the filter, increasing the number of multipliers will help reduce the overall latency. Suppose that we set the acceptability threshold to be 0.8. The results are shown in Table 6. Recall that $w_1 = 1$ and $w_2 = 10$. That is we consider register criteria ten times as much as the latency value. RCIS attempts to create a schedule which minimizes the total weighted sum of $w_1x + w_2y$ where x and y are the weighted latency and weighted register counts of the resulting schedule. Figure 19 depicts acceptability values for each design configuration based on RCIS. When we increase the number of functional units the latency decreases while the number of register counts needed increases. However, when the number of multipliers becomes 4 or more, RCIS can create a schedule which gives the maximum acceptability values 0.84 (which is greater than the threshold defined at 0.8). By inspecting the resulting schedule, we conclude that 4 multipliers would be sufficient and adding more multipliers will be wasteful. Compared this the schedule generated by IS, we found that since IS does not consider the register criteria, IS attempts to utilize all available resources to minimize the overall latency values. Thus, the latency of schedule generated by IS keeps decreasing and the number of register counts keep increasing. This will finally decrease the acceptability value according the constraint.

From the results, we can see that to achieve the acceptability threshold 0.8, using RCIS will give a better design solution using fewer number of registers. Consider the running time. For all the cases, the maximum running time is approximately 2.8 seconds to achieve the results for 1 adder and 5 multipliers on the same computer.

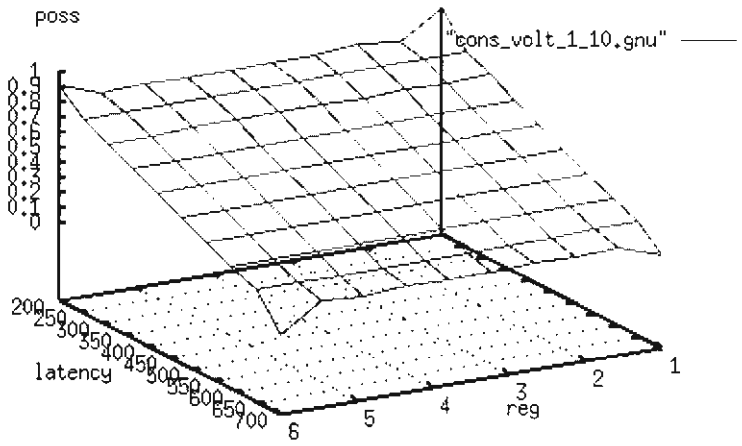


Figure 18: Constraint for Voltera filer.

	1 add 2 muls		1 add 3 muls		1 add 4 muls		1 add 5 muls	
	RCIS	IS	RCIS	IS	RCIS	IS	RCIS	IS
Avg Latency	308	300	267	270	260	260	260	246
Max Reg	2	2	3	3	**4**	4	4	5
Acceptability	0.78	0.80	0.84	0.84	**0.84**	0.84	0.84	0.84
Max Latency	561	561	474	477	445	445	445	416

Table 6: Exploring various number of functional units using RCIS and IS.

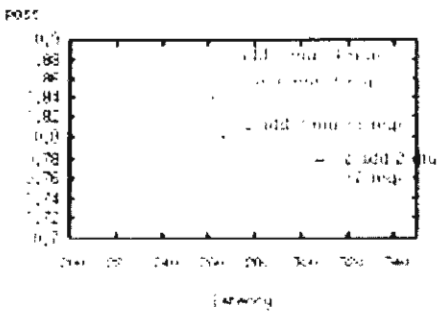


Figure 19: Acceptability values for each configuration.

7 Conclusion

We propose a design exploration framework considering impreciseness. The framework is based on the scheduling core, RCIS which considers impreciseness in the system specification and constraint and attempts to create a schedule which minimizes both latency and register usages. The framework can be used to generate various design solutions under imprecise system constraints and characteristics and select an acceptable solution under latency and register criteria. The experiments demonstrate the usage of the framework on a well-known benchmark, where the selected design solution can be found with a given acceptability level.

References

[1] I. Ahmad, M. K. Dhodhi, and C.Y.R. Chen. Integrated scheduling, allocation and module selection for design-space exploration in high-level synthesis. *IEEE Proc.-Comput. Digit. Tech.*, 142:65–71, January 1995.

[2] Cagdas Akturan and Margarida F. Jacome. RS-FDRA - a register sensitive software pipelining algorithm for embedded VLIW processors. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 12(21):1395–1415, December 2002.

[3] C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient scheduling for imprecise timing based on fuzzy theory. In *Proceedings of Midwest Symposium on Circuits and Systems*, pages 272–275, 1998.

[4] C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient algorithms for finding highly acceptable designs based on module-utility selections. In *Proceedings of the Great Lake Symposium on VLSI*, pages 128–131, 1999.

[5] C. Chantrapornchai, E. H-M. Sha, and X. S. Hu. Efficient module selections for finding highly acceptable designs based on inclusion scheduling. *J. of System Architecture*, 11(4):1047–1071, 2000.

[6] C. Chantrapornchai, E. H-M. Sha, and Xiaobo S. Hu. Efficient acceptable design exploration based on module utility selection. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 19:19–29, Jan. 2000.

[7] C. Chantrapornchai, W. Surakumpolthorn, and E.H. Sha. Efficient scheduling for design exploration with imprecise latency and register constraints. In *Lecture Notes in Computer Science: 2004 International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 259–270, 2004.

[8] C. Chantrapornchai and S. Tongsima. Resource estimation algorithm under impreciseness using inclusion scheduling. *Intl. J. on Foundation of Computer Science, Special Issue in Scheduling*, 12(5):581–598, 2001.

[9] S. Chaudhuri, S. A. Bylthe, and R. A Walker. An exact methodology for scheduling in 3D design space. In *Proceedings of the 1995 International Symposium on System Level Synthesis*, pages 78–83, 1995.

[10] S. Chaudhuri and R. Walker. Computing lower bounds on functional units before scheduling. In *Proceedings of the International Symposium on System Level Synthesis*, pages 36–41, 1994.

[11] A. Dani, V. Ramanan, and R. Govindarajan. Register-sensitive software pipelining. In *Proceedings of the Merged 12th International Parallel Processing and 9th International Symposium on Parallel and Distributed Systems*, pages 194–198, April 1998.

[12] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, and J. Bhasker. Datapath synthesis using a problem-space genetic algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(8):934–944, August 1995.

[13] A. Eichenberger and E. S. Davidson. Register allocation for predicated code. In *Proceeding of MICRO*, 1995.

[14] Alexandre E. Eichenberger and Edward S. Davidson. Stage scheduling: A technique to reduce the register requirements of a modulo schedule. In *Proceedings of MICRO-28*, pages 338–349, 1995.

[15] H. Esbensen and E. S. Kuh. Design space exploration using the genetic algorithm. In *Proceedings of the 1996 International Symposium on Circuits and Systems*, pages 500–503, 1996.

[16] F.Chen, S. Tongsima, and E. H. Sha. Loop scheduling algorithm for timing and memory operation minimization with register constraint. In *Proceedings of SiP'98*, 1998.

[17] K. Gupta. *Introduction to fuzzy arithmetics*. Van Nostrand, 1985.

[18] O. Hammami. Fuzzy scheduling in compiler optimizations. In *Proceedings of the ISUMA-NAFIPS*, 1995.

[19] I. Karkowski. Architectural synthesis with possibilistic programming. In *HICSS-28*, January 95.

[20] I. Karkowski and R. H. J. M. Otten. Retiming synchronous circuitry with imprecise delays. In *Proceedings of the 32nd Design Automation Conference*, pages 322–326, San Francisco, CA, 1995.

[21] A. Kaufmann and M. M. Gupta. *Fuzzy mathematical models in engineering and management science*. North-Holland, 1988.

[22] A. S. Kaviani and Z. G. Vranesic. On scheduling in multiprocess systems using fuzzy logic. In *Proceedings of the International Symposium on Multiple-valued Logic*, pages 141–147, 1994.

[23] J. Lee, A. Tiao, and J. Yen. A fuzzy rule-based approach to real-time scheduling. In *Proceedings of Intl. Conf. FUZZ-94*, volume 2, 1994.

[24] Josep Llosa, Eduard Ayguade, Antonio Gonzalez, Mateo Valero, and Jason Eckhardt. Lifetime-sensitive modulo scheduling in a production environment. *IEEE Transactions on Computers*, 50(3):234–249, 2001.

[25] Josep Llosa, Mateo Valero, and Eduard Ayguade. Heuristics for register-constrained software pipelining. In *International Symposium on Microarchitecture*, pages 250–261, 1996.

[26] C. A. Mandal, P. O. Chakrabarti, and S. Ghose. Design space exploration for data path synthesis. In *Proceedings of the10th International Conference on VLSI Design*, pages 166–170, 1996.

[27] K. Mertins et al. Set-up scheduling by fuzzy logic. In *Proceedings of the International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 345–350, 1994.

[28] J. Rabaey and M. Potkonjak. Estimating implementation bounds for real time DSP application specific circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(6), June 1994.

[29] T. J. Ross. *Fuzzy Logic with Engineering Applications*. McGrawHill, 1 edition, 1995.

[30] Z. Shao, Q.Zhuge, M. Liu, B. Xiao, and E. H-M. Sha. Switching activity minimization on instruction-level loop scheduling for VLIW DSP applications. In *Proceedings of ASAP*, 2004.

[31] A. Sharma and R. Jain. Estimating architectural resources and performance for high-level synthesis applications. *IEEE Transactions on VLSI systems*, 1(2):175–190, June 1993.

[32] H. Soma, M. Hori, and T. Sogou. Schedule optimization using fuzzy inference. In *Proceedings FUZZ-95*, pages 1171–1176, 1995.

[33] I.B. Turksen et al. Fuzzy expert system shell for scheduling. *SPIE*, pages 308–319, 1993.

[34] G. Varatkar and R. Marculescu. Communication-aware task scheduling and voltage selection for total systems energy minimization. In *IEEE/ACM Intl. Conf. on Computer Aided Design*, November 2003.

[35] J. L. Wong, S. Megerian, and M. Potkonjak. Forward-looking objective functions: Concepts & applications in high level synthesis. In *Proceedings of Design Automation Conference*, 2002.

[36] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Science*, 8:199–249, 1975.

[37] L. A. Zadeh. Fuzzy Logic. *Computer*, 1:83–93, 1988.

[38] J. Zalamea, J. Llosa, E. Ayguade, and M. Valero. Software and hardware techniques to optimize register file utilization in vliw architectures. In *Proceedings of the International Workshop on Advanced Compiler Technology for High Performance and Embedded Systems (IWACT)*, July 2001.