ระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุ
FALL DETECTING AND ALERT SYSTEM FOR ELDER

โดย

นายสันติ กิทำ

นายสุทธิเกียรติ อุ่นทอง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

ระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุ

Fall detecting and alert system for elder

โดย

นายสันติ กิทำ

นายสุทธิเกียรติ อุ่นทอง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

ปริญญานิพนธ์ปีการศึกษา 2560

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุ

Fall detecting and alert system for elder

ผู้จัดทำ

1. นายสันติ กิทำ 57011335
2. นายสุทธิเกียรติ อุ่นทอง 57011394

..... อาจารย์ที่ปรึกษา

ผศ.สุรพล บุญจันทร์

กิตติกรรมประกาศ

โครงงานจากการศึกษาฉบับนี้สำเร็จได้ด้วยความอนุเคราะห์ของบุคคลหลายท่าน โดยเฉพาะอย่างยิ่ง ผศ.สุรพล บุญจันทร์ อาจารย์ที่ปรึกษาที่ได้ให้คำปรึกษาแนวทางต่างๆในการริเริ่มทำโครงงาน ชี้แนะข้อบกพร่องต่างๆ และขอขอบคุณคณาจารย์ภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่สนับสนุนงบประมาณในการทำโครงงานนี้ขึ้น และกราบขอบพระคุณพ่อแม่ที่ได้สนับสนุนให้ผู้จัดทำได้เดินทางมาถึงจุดนี้

นายสันติ กิทำ
นายสุทธิเกียรติ อุ่นทอง
ผู้จัดทำ

ระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุ
FALL DETECTING AND ALERT SYSTEM FOR ELDER

โดย นายสันติ กิทำ 57011335
นายสุทธิเกียรติ อุ่นทอง 57011394

อาจารย์ที่ปรึกษา ผศ.สุรพล บุญจันทร์

บทคัดย่อ

จุดประสงค์ของโครงงานนี้ต้องการออกแบบระบบช่วยเหลือผู้สูงอายุในกรณีที่หกล้ม โดยในปัจจุบันการหกล้มในผู้สูงอายุมักเกิดขึ้นในกรณีที่ผู้สูงอายุอาศัยอยู่บ้านคนเดียวและไม่สามารถที่จะช่วยเหลือตัวเองได้ โดยการหกล้มนั้นเป็นสาเหตุหลักของการบาดเจ็บที่พบนอกเหนือจากโรคภัยไข้เจ็บในผู้สูงอายุที่มีอายุมากกว่า 65 ปี จากสถิติปัจจุบันอัตราการล้มของผู้สูงอายุนั้นเพิ่มสูงขึ้นถึง 65% และยังเป็นสาเหตุหลักของการเสียชีวิตอีกด้วย ในโครงงานนี้ผู้จัดทำจะแสดงการใช้อุปกรณ์ติดตั้งในผู้สูงอายุ เพื่อตรวจจับการล้มในกรณีที่ผู้สูงอายุอาศัยอยู่บ้านคนเดียว เมื่ออุปกรณ์ตรวจพบการล้ม จะทำการแจ้งเตือนไปยังลูกหลานผ่านแอพพลิเคชั่นในโทรศัพท์ทันที เพื่อให้สามารถขอความช่วยเหลือไปยังโรงพยาบาลได้ทันท่วงที

ABSTRACT

THE OBJECTIVE OF THIS PROJECT WERE TO DESIGN FALL DETECTING AND WARNING DEVICE FOR ELDERLY PEOPLE. IN THIS PRESENT, THE MOST OF FALLS OCCUR WHEN THE PERSON IS ALONE AND UNABLE TO GET UP. FALLS ARE THE LEADING CAUSE OF INJURY-RELATED IN PERSONS OVER THE AGE OF 65 YEARS. THE RATE FOR FALLS INCREASES TO 65% AND MAJOR CAUSE OF DEATH AND INJURY IS FALL.

IN THIS PROJECT THE RESEARCHER WILL SHOW METHOD OF THE FALL DETECTING AND WARNING CAN BE HELP SENIORS WHO STAY AT THEIR HOMES AND LIVE LONELY. THE DEVICE CAN DETECT FALL AND THEN SEND ALERT TO THEIR DESCENDANTS VIA SMARTPHONE APPLICATION AND SEND REQUEST HELP TO HOSPITAL IMMEDIATELY.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

จุดประสงค์ของโครงงานนี้ต้องการออกแบบระบบช่วยเหลือผู้สูงอายุในกรณีที่หกล้ม โดยในปัจจุบันการหกล้มในผู้สูงอายุมักเกิดในกรณีที่ผู้สูงอายุอาศัยอยู่บ้านคนเดียว และไม่สามารถช่วยเหลือตัวเองได้ โดยการหกล้มนั้นเป็นสาเหตุหลักของการบาดเจ็บที่พบนอกเหนือจากโรคภัย ในผู้สูงอายุที่มีอายุมากกว่า 65 ปี จากสถิติปัจจุบันอัตราการล้มของผู้สูงอายุนั้นเพิ่มสูงขึ้นถึง 65% และยังเป็นสาเหตุหลักของการเสียชีวิตอีกด้วย

ในโครงงานนี้ผู้จัดทำจะแสดงการใช้อุปกรณ์ติดตั้งในผู้สูงอายุ เพื่อตรวจจับการล้มในกรณีที่ผู้สูงอายุอาศัยอยู่บ้านคนเดียว เมื่อเครื่องตรวจพบการล้ม จะทำการส่งการแจ้งเตือนไปยังญาติหรือผู้ดูแล ผ่านแอพพลิเคชั่นในโทรศัพท์ทันที เพื่อให้สามารถขอความช่วยเหลือไปยังโรงพยาบาลได้อย่างทันท่วงที

1.2 วัตถุประสงค์

1. เพื่อศึกษาและประยุกต์การใช้งานอุปกรณ์ Arduino และเซ็นเซอร์ชนิดต่างๆ
2. เพื่อสร้างระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุ
3. เพื่อช่วยเหลือผู้สูงอายุที่อยู่บ้านคนเดียว เมื่อเกิดอุบัติเหตุ

1.3 ขอบเขตของปริญญานิพนธ์

ในโครงงานนี้ผู้จัดทำจะแสดงการใช้งานอุปกรณ์ติดตั้งในผู้สูงอายุเพื่อตรวจจับการล้มในกรณีที่ผู้สูงอายุอาศัยอยู่บ้านคนเดียว เมื่อเครื่องตรวจพบการล้ม จะทำการแจ้งเตือนไปยังลูกหลานหรือผู้ดูแลผ่านแอพพลิเคชั่นในโทรศัพท์ทันที เพื่อให้สามารถขอความช่วยเหลือไปยังโรงพยาบาล

บทที่ 2

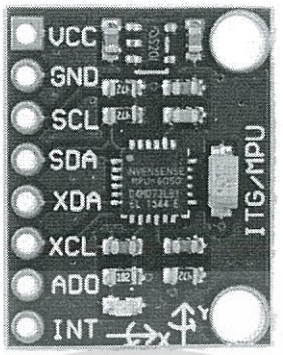
ทฤษฎีและหลักการที่เกี่ยวข้อง

ในโครงงานนี้ผู้จัดทำจะแสดงการใช้อุปกรณ์ติดตั้งในผู้สูงอายุ เพื่อตรวจจับการล้มสำหรับผู้สูงอายุที่อาศัยอยู่บ้านคนเดียว เมื่อเครื่องตรวจพบการล้มกระทันหัน ระบบจะทำการแจ้งเตือน ไปยังญาติหรือผู้ดูแลผ่านแอพพลิเคชั่นในโทรศัพท์ เพื่อให้แจ้งไปยังโรงพยาบาลเพื่อขอความช่วยเหลือได้อย่างทันท่วงที

2.1 เซ็นเซอร์ตรวจจับการเคลื่อนไหว (GY521 mpu6050)

GY521 เป็นโมดูลที่ตรวจจับการเคลื่อนไหวของวัตถุ โดยจะประกอบด้วยเซ็นเซอร์ 2 ตัว คือ Accelerometers และ Gyroscope เพื่อตรวจวัดจากความเร่งเชิงเส้น (Linear Acceleration) และความเร็วเชิงมุม (Angular Velocity) สำหรับความเร่งเชิงเส้น จะใช้ Accelerometers ในการตรวจจับการเปลี่ยนแปลงความเร่ง โดย output จะนำไปเทียบกับความเร่งเนื่องจากแรงโน้มถ่วงของโลก โดยตัว Accelerometers จะเป็นตัวบ่งบอกค่าความเอียง ณ ขณะนั้น เนื่องจากในเซ็นเซอร์มีตัววัด 3 แกน ดังรูปที่ 2.1 และหากมีการเอียงนั้นแกนต่าง ๆ จะมีความเร่งไม่เท่ากัน ขึ้นอยู่กับองศาการเอียง ซึ่งจะทำให้ทราบค่าว่าเซ็นเซอร์นั้นเอียงกี่องศา โดยเทียบจากค่า g และต่อมาคือ ความเร็วเชิงมุม (Angular Velocity) จะใช้ Gyroscope ในการตรวจจับความเร็วเชิงมุม ซึ่ง output ที่ได้จะพบว่า ค่าจะเกิดก็ต่อเมื่อมีการเอียงและมีการเคลื่อนไหว ซึ่งหากอยู่นิ่ง Gyroscope จะวัดค่าไม่ได้เพราะไม่มีความเร็ว ดังรูปที่ 2.2

โดยการทำงานของโมดูล จะนำการทำงานของเซ็นเซอร์ 2 ตัวมารวมในเวลาเดียวกัน เพื่อใช้ในการตรวจสอบทิศการเคลื่อนที่ และ สามารถใช้ในการตรวจสอบความเร็วในการเปลี่ยนแปลงทิศทางของแกน XYZ ได้ ยกตัวอย่าง ถ้าวัตถุเกิดการเคลื่อนที่หรือเอียง Output ของ Accelerometer จะบอกค่าของการเอียง ว่าสถานะปัจจุบันค่าของ XYZ อยู่ที่เท่าไร แต่ Gyroscope จะวัดค่าได้ตอนที่กำลังเอียงหรือตอนกำลังเคลื่อนไหว

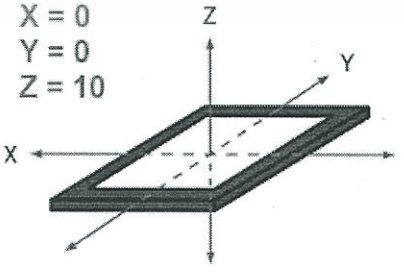


รูปที่ 2.1 เซ็นเซอร์ตรวจจับการเคลื่อนไหว (GY521)

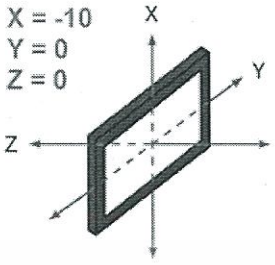
โมดูล GY521 MPU6050 มีข้อมูลรายละเอียดของโมดูลดังนี้ ใช้ MPU6050 ซึ่งมี 16bit A/D converter ภายในตัว เชื่อมต่อกับไมโครคอนโทรลเลอร์แบบ I2C ทำงานภายใต้ไฟเลี้ยง 3.3 โวลต์ มีช่วงของค่า Gyroscope อยู่ที่ ± 250 ± 500 ±1000 และ± 2000 °/s มีช่วงของค่า Acceleration อยู่ที่ : ± 2 ± 4 ± 8 ± 16 g มี Pin VCC, GND, SCL, SDA, XDA, XCL, ADO, INT

การทำงานของ Accelerometer

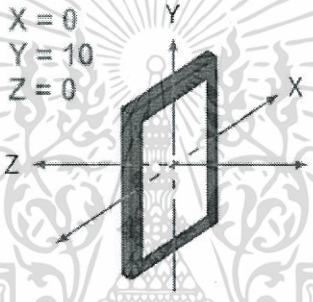
Accelerometer เป็นเครื่องมือวัดความเร่ง สำหรับแกน X,Y และ Z Accelerometer จะวัดความเร่งในแต่ละแกน คือ ขณะที่ไม่เคลื่อนที่ ค่าความเร่งในแต่ละแกนจะมีค่าเป็น 0 ในที่นี้ แรงโน้มถ่วงของโลกทำให้ค่าจาก Accelerometer ไม่เป็น 0 ทั้งหมด ขณะไม่เคลื่อนที่ ถ้ากำหนดให้แกน Z ตั้งฉากกับพื้นโลก แกน X และ Y จะมีค่าเป็น 0 แต่ว่าแกน Z จะไม่มีค่าเป็น 0 เพราะมีแรงโน้มถ่วงของโลกกระทำอยู่ ดังนั้นค่าที่ได้จากแกน Z จะเป็น 9.8 m/s^2 ซึ่งเป็นค่าในทางทฤษฎี ในทางปฏิบัติค่าที่ได้จะเปลี่ยนแปลงไปมาระหว่างค่า 9.8 m/s^2 โดยประมาณค่าเป็น 10 m/s^2



รูปที่ 2.2 ค่าแกน XYZ



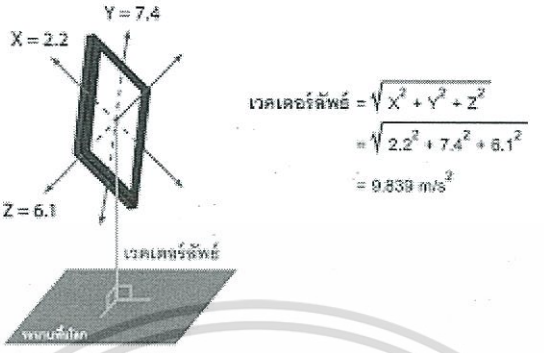
รูปที่ 2.3 ค่าแกน XYZ



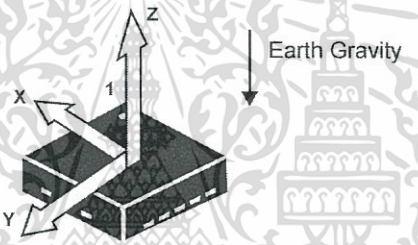
รูปที่ 2.4 ค่าแกน XYZ

หากวางอุปกรณ์ตั้งฉากกับพื้นโลก ที่ค่า X = 0, Y = 10 และ Z = 0 สมมติว่ามีการเคลื่อนที่ลง (ทิศเดียวกับแรงโน้มถ่วง) ค่าความเร่งที่ได้ก็จะมีมากกว่า 10 (แรงโน้มถ่วง+ความเร่งจากเครื่อง) แต่ถ้าเคลื่อนที่ขึ้นข้างบน ทิศทางกับแรงโน้มถ่วงโลก ค่าที่ได้ก็จะน้อยกว่า 10 (แรงโน้มถ่วง-ความเร่งจากเครื่อง)

ในกรณีที่ แต่แกน XYZ ไม่ตั้งฉากกับพื้นโลก แรงโน้มถ่วงของโลกที่กระทำกับแต่ละแกนของ Accelerometer ผลรวมเวกเตอร์ลัพธ์ที่ตั้งฉากกับพื้นโลกก็มีค่าประมาณ 9.8 และเมื่อมีเคลื่อนที่ความเร่งจะเปลี่ยนแปลงไปตามทิศทางการเคลื่อนที่



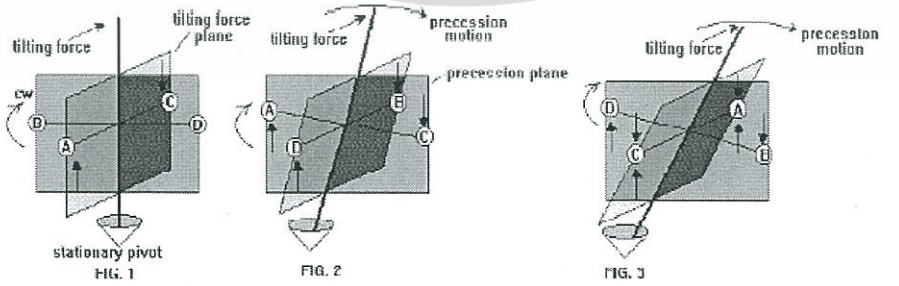
รูปที่ 2.5 ค่าแกน XYZ ที่ไม่ได้ตั้งฉากกับพื้นโลกโดยตรง



รูปที่ 2.6 แกนของ Accelerometers

การทำงานของ Gyroscope

กำหนดขอบด้วย A,B,C,D ทั้งแถบเพื่อดูว่า Gyroscope จุดล่างสุดเป็นแกนคงที่แต่สามารถหมุนได้รอบทิศทาง



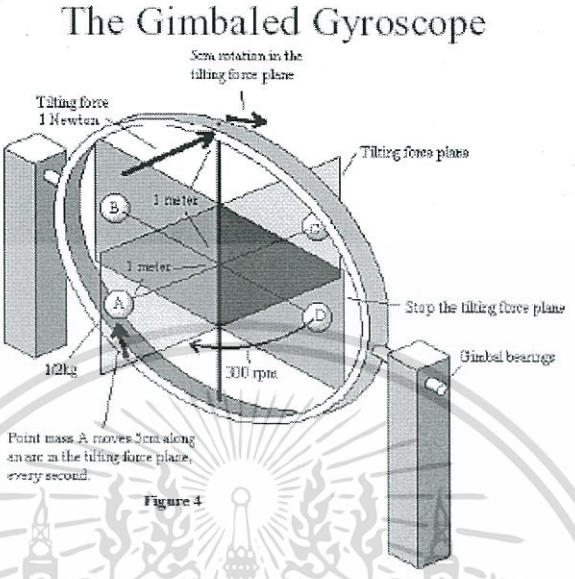
รูปที่ 2.7 การทำงานของ Gyroscope

เมื่อแรงกระทำ (tilting force) ที่ส่วนบนของแกน Gyroscope จุด A จะเคลื่อนที่ขึ้นตามแนวตั้ง จุดCเลื่อนลงตามแนวนอนพร้อมกัน A และ B หมุนไป 90^0 เช่นเดียวกับที่เกิดกับ C และ D โดยที่ A นั้นยังคงเลื่อนขึ้นในตำแหน่ง90^0 และ C เลื่อนลง ผลของการเลื่อนของ A และ C ทำให้แกนของ Gyroscope หมุนตามการกระทำของ precession plane เรียกการเกิดขึ้นของลักษณะนี้ว่า precession แกนของ Gyroscope จะหมุนไปทางมุมขวาเนื่องจากการหมุน ถ้า Gyroscope ถูกทำให้หมุนทวนเข็มนาฬิกามันก็จะไปทางมุมซ้าย หมายความว่าแรงกระทำตอนต้นเป็นการดึง

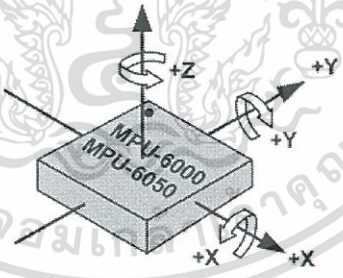
เมื่อ Gyroscope หมุนไปอีก 90^0 จุด C จะมาแทนจุด A ในจุดที่แรงกระทำไปแล้วครั้งแรก การเคลื่อนที่ลงของจุด C จะถูกต้านโดย tilting force ทำให้แกนของ Gyroscope ไม่เปลี่ยนแปลง ยิ่งมีแรง tilting กระทำมากขึ้นแกนของ Gyroscope ก็จะมีแรงดีดกลับมามากเมื่อขอบของ precession plane อยู่ที่ 180^0

จากข้างต้นทำให้ทราบว่าการหมุนของแกน Gyroscope เนื่องจากแรงจุด A และ C เคลื่อนที่ขึ้นลงฉะนั้นเมื่อหมุน Gyroscope ในทิศทางตรงกันข้ามกับข้างต้นจะเกิดแรงเคลื่อนที่ขึ้นลงมากขึ้น

บางครั้งการเกิดขึ้นของ precession เป็นสิ่งที่ไม่ต้องการจึงมีการสร้าง Gyroscope แบบมีแกนที่เรียกว่า Gimbaled Gyroscope ซึ่งเป็น Gyroscope พื้นฐานถูกติดตั้งไว้ในระนาบที่ตั้งฉากกับแนวแรง เมื่อหมุนขอบไปทางระนาบ Gimbal พลังงานจะถ่ายเทไปยังขอบโดย tilting force เพื่อหยุดการกระทำนั้น ขอบก็จะหมุนกลับในทิศทางระนาบของแรง tilting แต่ละครั้งที่ Gyroscope ถูกกระตุ้นแกนจะหมุนไปตามส่วนโค้งในระนาบของแรง tilting โดยที่ไม่มีการเปลี่ยนแปลงความเร็วรอบการหมุนของขอบรอบ ๆ แกน



รูปที่ 2.8 การทำงานของ Gyroscope



รูปที่ 2.9 แกนของ Gyroscope

2.2 การเชื่อมต่อแบบ i2c

I2C ย่อมาจาก Inter-IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซี โดยบัส I2C ได้รับการพัฒนาโดย ฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงาน และ ควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้กำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I2C ทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้ระหัสข้อมูลและการกำหนดสภาวะโลจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรม หรือ SDA (Serial Data line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรม หรือ SCL (Serial Clock line) ในการอธิบายต่อไปนี้จะเรียกสายสัญญาณทั้งสองว่า SDA และ SCL

คุณสมบัติโดยทั่วไปของบัสI2C

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-derectional line) ต้องมีการต่อตัวต้านทานพูลอัพกับแรงดัน +5V ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไมมีการติดต่อใช้งาน ทั้งยังช่วยป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาต์พุตของอุปกรณ์ที่ต่ออยู่บนบัส I2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (Open-drain) หรือ คอลเล็กเตอร์เปิด (Open-collector)

อัตราการถ่ายทอดข้อมูลบนบัส I2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปรกติ และ สูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง อุปกรณ์ที่ต่อร่วมบนบัส I2C จะต้องมีความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400pf การเข้าถึงอุปกรณ์บนบัส I2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่าคือ 7 บิต (7-bit addressing) หรือ 10 บิต (10-bit addressing)

หลักการของบัส I2C

บัส I2C ประกอบด้วยสายสัญญาณ 2 เส้นคือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส เพื่อให้ผู้ใช้งานทราบว่าอุปกรณ์ใดติดต่อกันอยู่และอุปกรณ์ใดทำหน้าที่เป็นตัวรับหรือตัวส่ง

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง (transmitter) และอุปกรณ์ที่เป็นผู้รับข้อมูล เรียกว่า ตัวรับ (receiver) อุปกรณ์บนบัส I2C สามารถเป็นได้ทั้งตัวรับและส่ง บางอุปกรณ์ทำหน้าที่

เป็นตัวรับอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I2C ที่ทำหน้าที่เป็นตัวส่งอย่างเดียวอุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I2C เรียกว่า มาสเตอร์ (master) อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I2C เรียกว่า สเลฟ (slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I2C คือ

(1) การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น

(2) ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดความเปลี่ยนแปลงเด็ดขาด มิฉะนั้นสัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

2.2.1 ข้อมูลพื้นฐานที่สำคัญของบัส I2C

บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อ สถานะลอจิกบนสาย SDA และ SCL มีลอจิกสูงทั้งคู่ หมายความว่าการถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้ เริ่มต้นการถ่ายทอดข้อมูล (start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะนี้ว่า สภาวะเริ่มต้น (START) ข้อมูลดำรงอยู่บนบัส(data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL มีลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับข้อมูลในจังหวะนั้นว่า เป็นบิต"0" หรือ "1" ข้อมูลอาจเกิดความเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุด หรือ สภาวะเริ่มต้นก็ได้ ทำให้ข้อมูลที่ทำการถ่ายทอดเกิดความผิดพลาดเกิดขึ้น การรับรู้ข้อมูล (acknowledge) เกิดขึ้นหลังจากการถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่า บิตรับรู้ (acknowledge bit) มีสถานะเป็นลอจิกสูง หลังการส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกา อุปกรณ์สเลฟที่ถูกอ้างอิงในการติดต่อ หรือ กำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้ที่มีสถานะลอจิกต่ำเพื่อตอบสนองให้ทราบว่าได้รับข้อมูลเรียบร้อยแล้ว หยุดการถ่ายทอดข้อมูล (stop dara transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูง ในขณะที่สาย SCL มีสถานะลอจิกสูงเรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะหยุด (STOP)

2.2.2 การทำงานบนบัส I2C

เริ่มต้นด้วยการเข้าถึงอุปกรณ์เสียก่อน โดยการเข้าถึงอุปกรณ์บนบัส I2C นั้นจะใช้การเข้าถึงแบบ 7 หรือ 10 บิต ในกรณีที่มีอุปกรณ์ต่ออยู่บนบัสไม่มาก ใช้การเข้าถึงแบบ 7 บิตก็เพียงพอ แต่ในบางอุปกรณ์ต้องใช้การเข้าถึงแบบ 10 บิต หลังจากที่ติดต่ออุปกรณ์แต่ละตัวเรียบร้อยแล้ว ก็จะเริ่มต้นการถ่ายทอดข้อมูลกันต่อไป

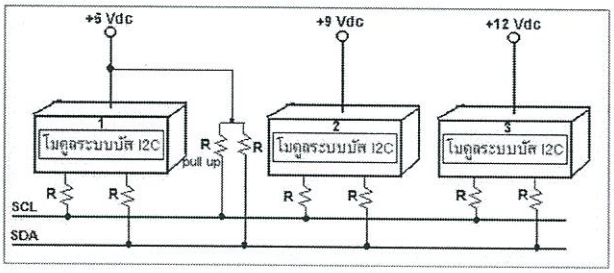
การเข้าถึงแบบ 7 บิต (7-bit addressing)

ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้นคือ ข้อมูลที่ใช้อ้างอิงอุปกรณ์ที่ต้องการติดต่อ โดยมีรูปแบบแสดงในรูป ใน 7 บิตบนรวมทั้งบิต LSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็น บิตกำหนดแอดเดรสคงที่ (fix address bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิต ไม่สามารถเปลี่ยนแปลงแก้ไขได้ ถัดมาอีก 3 บิตเป็นบิตกำหนดแอดเดรสที่สามารถโปรแกรมได้(programmable address bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0-A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I2C ส่วนในบิต LSB ที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวนั้น ๆ หากบิต LSB เป็น "0" หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น "1" จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ

ข้อมูลในไบท์ต่อมาคือ ข้อมูลควบคุม (control byte) ในอุปกรณ์แต่ละตัวจะมีการกำหนดข้อมูลควบคุมที่แตกต่างกันไป ยกตัวอย่างเช่น ไอซีเมมโมรีของทีวีตระกูล 24Cxx จะต้องส่งข้อมูลแอดเดรสของหน่วยความจำก่อนที่จะทำการส่งข้อมูลไปข้อมูลในไบท์ต่อมาคือ ข้อมูลที่ทำการถ่ายทอดจริง หลังจากการถ่ายทอดข้อมูลในแต่ละไบท์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง

การเข้าถึงแบบ 10 บิต (10-bit addressing)

จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อย โดยในไบท์แรกหลักจากสภาวะเริ่มต้น ต้องทำการกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของอุปกรณ์ที่ต้องการติดต่อ ในบิต LSB ของข้อมูลไบท์แรกยังคงเป็นการกำหนดว่า ต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบท์ต่อมาเป็นข้อมูลแอดเดรสในไบท์ที่ 2 ของอุปกรณ์ที่ต้องการติดต่อด้วย ข้อมูลไบท์ถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนี้ก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อ เช่นเดียวกันกับการเข้าถึงแบบ 7 บิตหลังจากถ่ายทอดข้อมูลครบทุกไบท์ ต้องมีสภาวะรับรู้เกิดขึ้น เพื่อให้ขบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้



รูปที่ 2.10 การเชื่อมต่อโมดูลในการสื่อสารแบบ I2C

2.3 Microcontroller (Arduino Nano)

Arduino เป็นไมโครคอนโทรลเลอร์บอร์ดแบบสำเร็จรูป ซึ่งถูกสร้างมาจาก Controller ตระกูล AVR ของ ATMEL มีการทำงาเป็น Open Source ที่สามารถนำไปพัฒนาต่อเป็นอุปกรณ์ต่างๆ เพราะสามารถโปรแกรมควบคุมเข้าตัวบอร์ดได้ง่ายขึ้น และยังมีการพัฒนาซอฟต์แวร์ที่ใช้ในการควบคุมตัวบอร์ดมีลักษณะเป็นภาษา C++ และยังสามารถนำโมดูลต่างๆมาต่อเพื่อประยุกต์การใช้งานและเพิ่มความสามารถให้กับตัวบอร์ดได้อีกด้วย โดยเราจะนำ Arduino มาเป็นตัวประมวลผลกลางระหว่างกับเซ็นเซอร์ตรวจจับการเคลื่อนไหว และ โมดูล WIFI ซึ่งจะประมวลข้อมูลดิบที่รับจากเซ็นเซอร์ตรวจจับการเคลื่อนไหว และส่งต่อผ่านโมดูล WIFI เพื่อส่งไปยัง Web server อีกที

ในโครงงานนี้จะเลือกใช้ไมโครคอนโทรลเลอร์ Arduino Nano เนื่องจากต้องการนำอุปกรณ์ไปติดตัว จึงต้องเลือกอุปกรณ์ที่มีขนาดเล็ก โดยตัว Arduino Nano มีขนาดเพียง 1.8 x 4.8 ซ.ม. ซึ่งมีขนาดเล็กมาก แต่ฟังก์ชันการใช้งานนั้นมีความสามารถเหมือนกับตัว Arduino UNO ที่มีขนาดใหญ่กว่า

2.4 ไมโครคอนโทรลเลอร์ (NodeMCU)

เป็นไมโครคอนโทรลเลอร์ที่ถูกบรรจุโมดูล WIFI ซึ่งพัฒนาต่อจาก ESP8266 ซึ่งนักพัฒนานั้นเจาะจงสำหรับการพัฒนา Internet of things (IoT) เนื่องจากจากเดิมจำเป็นต้องมีการเชื่อมต่อไมโครคอนโทรลเลอร์กับโมดูล WIFI ซึ่งจะมีขนาดใหญ่ และมีราคาแพง ใช้พลังงานเพิ่มขึ้น แต่ทาง NodeMCU นั้นมีการใช้พลังงานที่พอๆกับไมโครคอนโทรลเลอร์ตัวอื่นอย่างเช่น Arduino พร้อมทั้งมี I/O port เชื่อมต่อมาให้พัฒนาต่อกับโมดูลต่างๆเช่นเดียวกับ Arduino สามารถพัฒนาโปรแกรมได้เช่นเดียวกับ Arduino โดยผ่านทาง Arduino IDE ซึ่งใช้ภาษา C/C++ ได้เช่นกัน และสิ่งที่แตกต่างจาก Arduino คือใช้ CPU 32 bit จากเดิม Arduino ใช้เพียง 8 bit และมีพื้นที่บรรจุโปรแกรมมากขึ้น ซึ่งเป็นผลทำให้รันโปรแกรมที่มีความซับซ้อนและทำงานได้รวดเร็วมากกว่า Arduino

2.5 โครงสร้างโปรแกรมของ Arduino

ในการเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ Arduino จะต้องเขียนโปรแกรมโดยใช้ภาษาของ Arduino (Arduino Programming Language) ซึ่งตัวภาษาของ Arduino ก็เป็นการนำเอาโอเพ่นซอร์สโปรเจ็กต์ชื่อ Wiring มาพัฒนาต่อ ภาษา ของ Arduino แบ่งออกเป็น 2 ส่วนหลักคือ

- โครงสร้างภาษา (Structure) ตัวแปรและค่าคงที่
- ฟังก์ชัน (Function)

ภาษาของ Arduino จะอ้างอิงตามภาษา C/C++ จึงอาจกล่าวได้ว่าการเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ Arduino (ซึ่งก็รวมถึงบอร์ด Arduino) ก็คือการเขียนโปรแกรมภาษา C โดยเรียกใช้ฟังก์ชันและไลบรารีที่ทาง Arduino ได้ เตรียมไว้ให้แล้ว ซึ่งสะดวกและทำให้ผู้ที่ไม่มีความรู้ด้านไมโครคอนโทรลเลอร์อย่างลึกซึ้งสามารถเขียนโปรแกรม สั่งงานได้ ในส่วนต่อไปจะอธิบายถึงโครงสร้างโปรแกรมของ Arduino แบ่งได้เป็นสองส่วนคือ void setup() และ void loop()

โดยฟังก์ชัน setup() เมื่อโปรแกรมทำงานจะทำคำสั่งของฟังก์ชันนี้เพียงครั้งเดียว ใช้ในการกำหนดค่า เริ่มต้นของการทำงานส่วนฟังก์ชัน loop() เป็นส่วนทำงานโปรแกรมจะทำคำสั่งในฟังก์ชันนั่นต่อเนื่องกันตลอดเวลา โดยปกติใช้กำหนดโหมดการทำงานของขาต่างๆ กำหนดการสื่อสารแบบอนุกรม ฯลฯ ส่วนของฟังก์ชัน loop() เป็นโค้ดโปรแกรมที่ทำงาน เช่น อ่านค่าอินพุต ประมวลผล สั่งงานเอาต์พุต ฯลฯ โดยส่วน กำหนดค่าเริ่มต้น เช่นตัวแปรจะต้องเขียนที่ส่วนหัวของโปรแกรมก่อนถึงตัวฟังก์ชัน นอกจากนั้นยังต้องคำนึงถึง ตัวพิมพ์ เล็ก-ใหญ่ ของตัวแปรและชื่อฟังก์ชันนั้นให้ถูกต้อง

โดยส่วนของฟังก์ชัน setup() ฟังก์ชันนี้จะเขียนที่ส่วนต้นของโปรแกรม ทำงานเมื่อโปรแกรมเริ่มต้นเพียงครั้งเดียว ใช้เพื่อกำหนดค่าของตัวแปรโหมดการทำงานของขาต่างๆ เริ่มต้นเรียกใช้ไลบรารี่ ดังตัวอย่าง

```
 int buttonPin = 13;
void setup()
 {
Serial.begin(9600);
pinMode(buttonPin, INPUT);
}
 void loop()
```

```
{
if (digitalRead(buttonPin) == HIGH)
Serial.println('H');
else
Serial.println('L');
delay(1000);
}
```

ในขณะที่โปรแกรมภาษา C มาตรฐานที่เขียนบน AVR GCC (เป็นโปรแกรมภาษา C ที่ใช้ C คอมไพเลอร์ แบบ GCC สำหรับไมโครคอนโทรลเลอร์ AVR) จะเขียนได้ ดังนี้

```
int main(void)
{
init();
setup();
for (;;)
 loop();
return ;
}
```

ต่อมาส่วนของฟังก์ชั่น loop() หลังจากที่เขียนฟังก์ชั่น setup() ที่กำหนดค่าเริ่มต้นของโปรแกรมแล้ว ส่วนถัดมาคือฟังก์ชั่น loop() ซึ่งมีการทำงานตรงตามชื่อ คือจะทำงานตามฟังก์ชันวนต่อเนื่องตลอดเวลา ภายในฟังก์ชันจะมีโปรแกรมของผู้ใช้เพื่อรับค่าจากพอร์ต ประมวลผลแล้วสั่งเอาต์พุตออกขาต่างๆเพื่อควบคุมการทำงานของบอร์ด ดังตัวอย่างด้านล่างนี้

```
        int buttonPin = 13;
// setup initializes eerial and the button pin
        void setup()
                {
                Serial.begin(9600);
pinMode(buttonPin, INPUT);
```

```
        }
// loop checks the button pin each time,
// and will send serial if it is pressed
    void loop()
        {
        if (digitalRead(buttonPin) == HIGH)
        Serial.println('H');
        else
        Serial.println('L');
    delay(1000);
        }
```

โปรแกรมทำงานวนในฟังก์ชั่น loop() ตลอดเวลาหลังจากทำงานในฟังก์ชั่น setup() จึงสรุปได้ว่าฟังก์ชั่น setup() คือส่วนต้นของโปรแกรมที่ใช้ในการประกาศหรือตั้งค่าการทำงานในตอนเริ่มต้นทำงานในขณะที่ฟังก์ชั่น loop() เป็นเสมือนส่วนของโปรแกรมหลักที่ต้องวนทำงานอย่างต่อเนื่องตลอดเวลา อย่างไรก็ตามในบางโปรแกรมอาจมีเฉพาะส่วนของฟังก์ชั่น setup() และไม่มีฟังก์ชั่น loop() ก็ได้ แสดงว่าโปรแกรมนั้นๆต้องการตั้งค่าการทำงานหรือกำหนดให้มีการทำงานเพียงครั้งหรือรอบเดียว แล้วจบการทำงานทันที

2.6 โมดูล GSM A6

โมดูล A6 เป็นโมดูลการเชื่อมต่อแบบ GSM ซึ่งรองรับการใช้งานการส่ง SMS การโทรและการเชื่อมต่อการส่งข้อมูลแบบ GPRS ได้ โดยตัวโมดูลรองรับคลื่นความถี่ 850,900,1800 และ 1900 MHz โดยการทำงานจะสั่งการผ่าน AT command ซึ่งเป็นคำสั่งที่ใช้กันแพร่หลายในโมดูลการสื่อสารและเชื่อมต่อเข้ากับไมโครคอนโทรลเลอร์ผ่านทาง Serial (Tx,Rx)

2.7 ลักษณะการล้มในผู้สูงอายุ

รูปที่ 2.11 การหกล้มในผู้สูงอายุ

การหกล้มเป็นสาเหตุที่ทำให้ผู้สูงอายุต้องนอนโรงพยาบาลนานขึ้นเป็น 2 เท่าของการเข้าโรงพยาบาลเพื่อรักษาตัวด้วยเหตุผลอื่นๆของผู้สูงอายุ โดยที่ผู้สูงอายุเพศหญิงมีโอกาสหกล้มได้มากกว่าเพศชาย นอกจากนี้ยังมีการรายงานกรณีการหกล้มของผู้สูงอายุเพียงแค่ 1 ใน 3 ของเหตุการณ์ที่เกิดขึ้นจริงเท่านั้น ดังนั้น ข้อมูลการหกล้มของผู้สูงอายุที่มีการบันทึกไว้จึงต่ำกว่าความเป็นจริงมาก โดย 2 ใน 3 ของผู้สูงอายุที่เคยหกล้มไปแล้ว มีโอกาสที่จะเกิดการหกล้มครั้งใหม่ได้อีกภายในเวลา 6 เดือน การหกล้มเป็นสาเหตุสำคัญที่นำไปสู่การเสียชีวิตของ ผู้สูงอายุอเมริกันที่อายุ 65 ปีขึ้นไป โดยจากข้อมูลการเสียชีวิตของผู้สูงอายุอเมริกันในแต่ละปี พบว่าอย่างน้อยประมาณปีละ 9,500 ราย

สาเหตุและปัจจัยเสี่ยงที่สำคัญ

ปัญหากล้ามเนื้ออ่อนแรง โดยเฉพาะกล้ามเนื้อขา ดังนั้น การออกกำลังกล้ามเนื้อขาในผู้สูงอายุจึงมีความจำเป็น ทั้งนี้ เพื่อป้องกันการหกล้ม ปัญหาการทรงตัวไม่ดี พอเคลื่อนไหวร่างกายจึงเกิดการเสียหลักและหกล้มได้ง่าย ปัญหาการเวียนหน้าจากความดันเลือดตกเวลาเปลี่ยนท่า เช่น เปลี่ยนจากท่านอนหรือนั่งเป็นท่ายืน ร่างกายผู้สูงอายุปรับความดันเลือดได้ไม่ดี ทำให้ความดันเลือดต่ำชั่วขณะ เกิดอาการวิงเวียนหน้ามืด เซ และหกล้มได้ ความกระฉับกระเฉงของร่างกายลดลง เรื่องนี้เป็นปฏิกิริยาอัตโนมัติของร่างกาย ซึ่งโดยธรรมชาติจะลดลงตามวัย ในคนหนุ่มสาวแค่รู้สึกตัวว่ากำลังเซ ปฏิกิริยาอัตโนมัติของร่างกายจะทำงานทันที เกิดการปรับท่าทางอย่างคล่องแคล่วจนสามารถป้องกันตัวเองจากการหกล้มได้ ส่วนในผู้สูงอายุปฏิกิริยาอัตโนมัติที่ว่านี้ทำงานช้ากว่าจึงทำให้หกล้มไปแล้ว ปัญหาที่เท้าและรองเท้า การเจ็บที่เท้าทำให้ลงน้ำหนักไม่ได้เต็มที่และมั่นคงพอ เป็นเหตุให้หกล้มได้ง่ายเหมือนกัน หรือบางรายเกิดอาการชาบริเวณเท้าจะด้วยสาเหตุใดก็ตาม ทำให้เท้ารับความรู้สึกได้ไม่ดีเท่าที่ควร ซึ่งจะเป็นผลให้การสั่งการเกี่ยวกับการทรงตัวของสมองเสียไป ก็เกิดโอกาสที่จะหกล้มได้ง่าย

การใส่รองเท้าที่ไม่เหมาะสมก็เป็นเหตุได้ เช่น ส้นรองเท้าสูงไปทำให้ท่าการทรงตัวผิดไปจากธรรมชาติ ทำให้หกล้มได้ง่าย ใส่รองเท้าคับหรือหลวมไปจนเท้ารับสัมผัสได้ไม่ดีพอก็ทำให้หกล้มได้เช่นกัน ปัญหาเรื่องสายตาก็มีส่วนเนื่องจากการทรงตัวนั้นต้องอาศัยการมองเห็นด้วยร่างกายถึงจะทรงตัวได้ดี ผู้สูงอายุส่วนใหญ่มีปัญหาเรื่องนี้อยู่แล้วจึงหกล้มได้ง่ายกว่าคนหนุ่มสาว อีกอย่างคนที่สายตาไม่ดีมีโอกาสที่จะเดินสะดุดและหกล้มได้ง่ายอยู่แล้ว ปัญหาเรื่องการใช้ยา ผู้สูงอายุส่วนใหญ่มีโรคประจำตัวต้องกินยาประจำ ยาบางตัวอาจจะทำให้ความดันเลือดต่ำ เกิดอาการหน้ามืด เวียนหน้า ง่วงหรือมึนงง เป็นเหตุให้เกิดการหกล้มได้ ดังนั้น เวลาพาผู้สูงอายุไปหาหมอต้องถามหมอเสมอว่ายาที่ให้มานั้นมีผลต่อเรื่องนี้หรือไม่ ถ้ามีจะได้หาทางป้องกันเกี่ยวข้องกับปัญหาการหกล้มก่อนที่จะเสียชีวิต การหกล้มที่เป็นเหตุนำไปสู่การเสียชีวิตเกินครึ่งเกิดขึ้นในคนที่อายุ 75 ปีขึ้นไป ในผู้สูงอายุที่มีอายุระหว่าง 65-69 ปี เมื่อเกิดการหกล้มจะพบปัญหากระดูกสะโพกหักได้ถึง 1 ใน 200 ราย และจะพบปัญหานี้มากขึ้นเรื่อยๆ ตามอายุที่เพิ่มขึ้น ในผู้สูงอายุ 85 ปีขึ้นไป จะพบปัญหากระดูกสะโพกหักได้สูงถึง 1 ใน 10 ราย 1 ใน 4 ของผู้สูงอายุที่เกิดปัญหากระดูกสะโพกหักจากการหกล้มจะเสียชีวิตภายใน 6 เดือนหลังจากการหกล้ม แม้จะไม่เสียชีวิต แต่ปัญหาใหญ่ที่ผู้สูงอายุเหล่านี้ต้องเผชิญก็คือ การไม่สามารถใช้ชีวิตตามปรกติได้อีกต่อไป และกลายเป็นภาระต่อผู้ดูแลเป็นอย่างมาก โดยลักษณะการล้มในผู้สูงอายุแม้จะไม่รุนแรงมาก แต่จะส่งผลอันตรายมากกว่าในวัยอื่นๆอย่างมาก สำหรับการออกแบบการทำงานของเซ็นเซอร์ จะต้องทำการวิเคราะห์เพิ่มเติมในขั้นตอนต่อไป

บทที่ 3

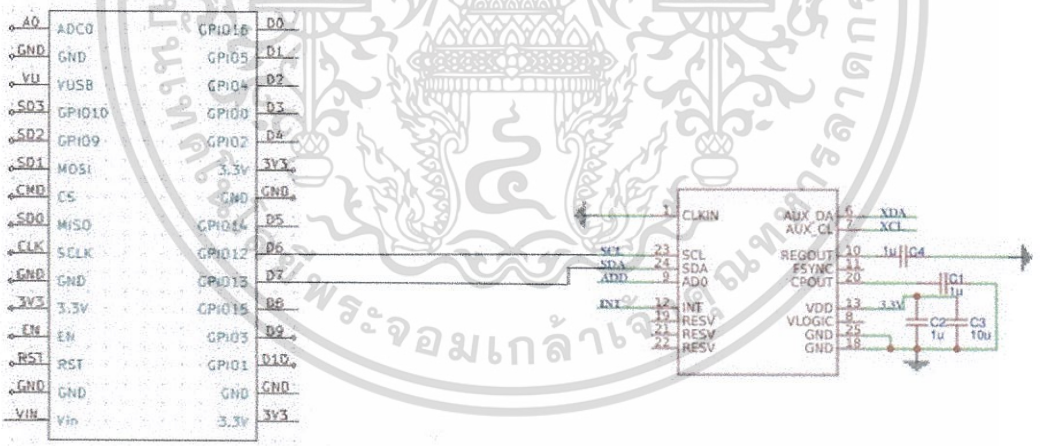
การออกแบบและการจัดทำปริญญานิพนธ์

3.1 การออกแบบ

สำหรับการออกแบบจะเป็นส่วนของการออกแบบวงจรทำงานตรวจจับการล้ม โดยมีส่วนประกอบที่ของไมโครคอนโทรลเลอร์กับโมดูลต่างๆดังนี้

3.1.1 การออกแบบวงจรของโมดูลวัดความเร่ง MPU6050 ต่อเข้ากับวงจรไมโครคอนโทรลเลอร์

ในการเชื่อมต่อเพื่อการเขียนโปรแกรมสั่งการและอ่านค่าความเร่งจากโมดูล MPU6050 จะใช้รูปแบบการติดต่อสื่อสารกับไมโครคอนโทรลเลอร์สำหรับการรับและส่งข้อมูลแบบ I2C โดยการเชื่อมต่อขาของโมดูลขา SCL ต่อเข้ากับกับไมโครคอนโทรลเลอร์ขา D6 สำหรับการส่งสัญญาณนาฬิกา และโมดูลขา SDA ต่อเข้ากับขา D7 ของโมโครคอนโทรลเลอร์สำหรับการรับและส่งข้อมูล ซึ่งวงจรการเชื่อมต่อแสดงได้ดังรูปที่ 3.1



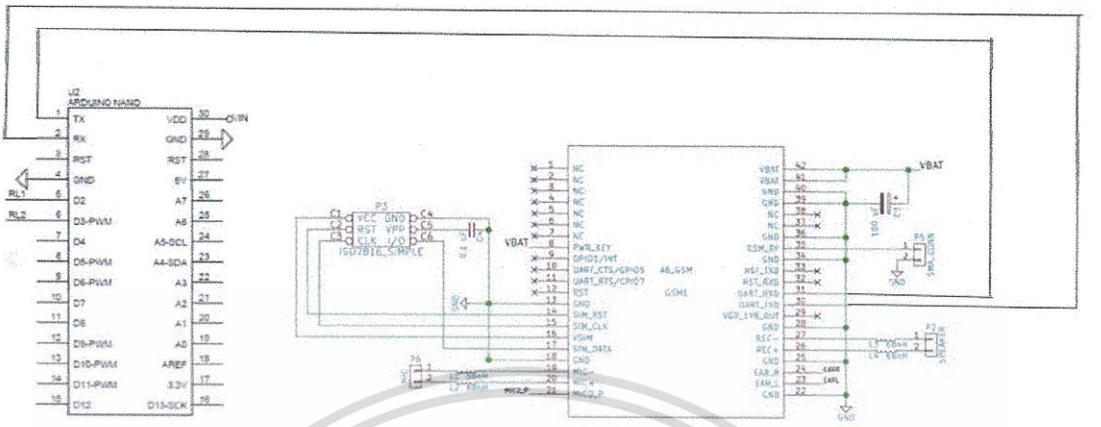
รูปที่ 3.1 วงจรการเชื่อมต่อโมดูลวัดความเร่งกับไมโครคอนโทรลเลอร์

โดยที่ไมโครคอนโทรลเลอร์ที่ทำการรับค่าความเร่งจากโมดูลคือ NodeMCU ซึ่งมีโมดูล WIFI ESP8266 ฝังมาในตัวซึ่งการทำงานส่วนของโปรแกรมของโมดูลวัดความเร่งกับไมโครคอนโทรลเลอร์แสดงโฟลว์ชาร์ตการทำงานได้ดังรูปที่ 3.2

รูปที่ 3.2 โฟลว์ชาร์ตการทำงานของโปรแกรมของโมดูลวัดความเร่งกับไมโครคอนโทรลเลอร์

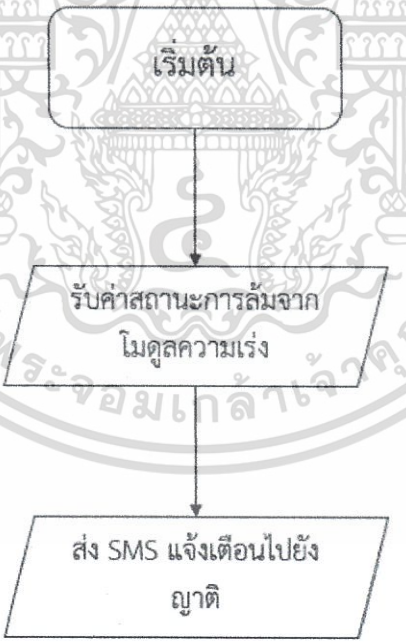
3.1.2 การออกแบบวงจรของโมดูลการเชื่อมต่อ GSM (A6) ต่อเข้ากับวงจรไมโครคอนโทรลเลอร์

ในการเชื่อมต่อเพื่อการเขียนโปรแกรมสั่งการและอ่านค่าจากโมดูล GSM จะใช้รูปแบบการติดต่อสื่อสารกับไมโครคอนโทรลเลอร์สำหรับการรับและส่งข้อมูลแบบอนุกรม (Serial Transmission) เป็นการรับส่งข้อมูลแบบทีละบิตแทนการรับส่งข้อมูลแบบพร้อมกันทุกบิตในเวลาเดียวกัน โดยการเชื่อมต่อขาของโมดูลขา TX ต่อเข้ากับกับไมโครคอนโทรลเลอร์ขา RX เพื่อทำการส่งข้อมูลให้ไมโครคอนโทรลเลอร์ และโมดูลขา RX ต่อเข้ากับขา TX สำหรับการรับค่า จากโมโครคอนโทรลเลอร์ ซึ่งวงจรการเชื่อมต่อแสดงได้ดังรูปที่ 3.3



รูปที่ 3.3 วงจรการเชื่อมต่อโมดูล GSM กับไมโครคอนโทรลเลอร์

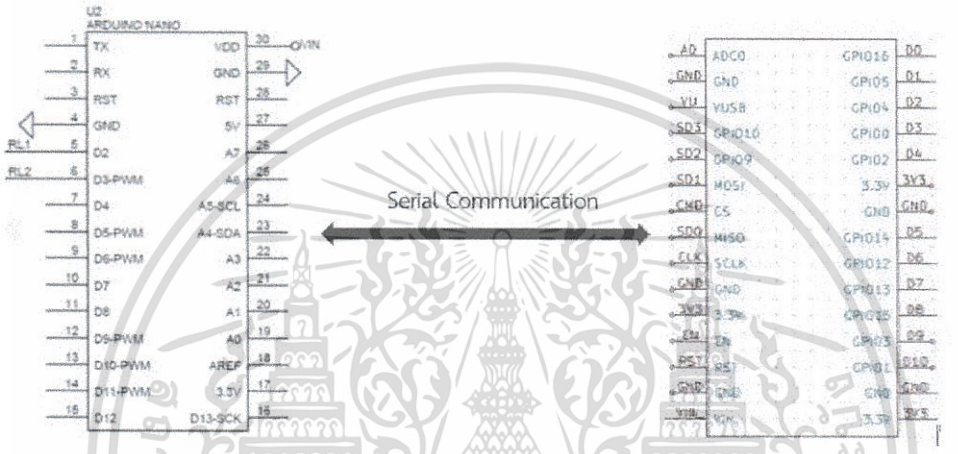
ซึ่งการทำงานส่วนของโปรแกรมของโมดูล WIFI กับไมโครคอนโทรลเลอร์แสดงโฟลว์ชาร์ตการทำงาน ได้ดังรูปที่ 3.4



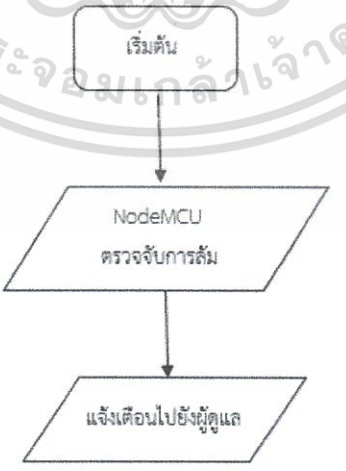
รูปที่ 3.4 โฟลว์ชาร์ตการทำงานของโปรแกรมของโมดูล GSM กับไมโครคอนโทรลเลอร์

3.1.3 การออกแบบวงจรของไมโครคอนโทรลเลอร์สองตัว

วงจรการเชื่อมต่อไมโครคอนโทรลเลอร์รุ่น Arduino Nano และไมโครคอนโทรลเลอร์ NodeMCU โดยที่ NodeMCU ทำหน้าที่เชื่อมต่อกับโมดูลความเร่งเพื่อตรวจจับการล้มแล้วส่งข้อมูลผ่าน WIFI แจ้งเตือนไปยังเซิร์ฟเวอร์ และเมื่อมีการล้มยังส่งข้อมูลผ่าน Serial ไปยัง Arduino เพื่อสั่งการให้โมดูล GSM ส่งข้อมูลไปยังเบอร์โทรศัพท์ญาติอีกทาง



รูปที่ 3.5 วงจรเชื่อมต่อระหว่างโมดูลสองตัว

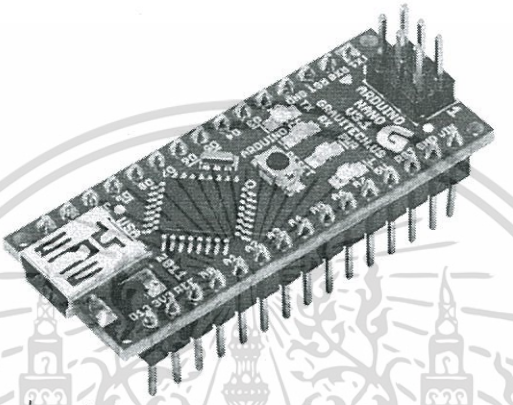


รูปที่ 3.6 โฟลว์ชาร์ตการทำงานของไมโครคอนโทรลเลอร์สองตัว

3.2 อุปกรณ์ที่ใช้ในการทดลอง

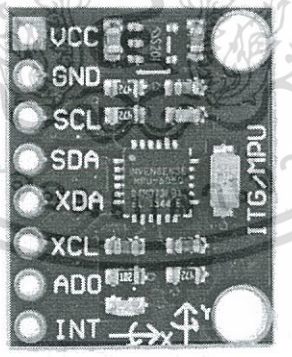
สำหรับอุปกรณ์ต่างๆที่ใช้ในการเก็บค่าผลการทดลองในการจัดทำปริญญานิพนธ์

3.2.1 ไมโครคอนโทรลเลอร์รุ่น Arduino Nano



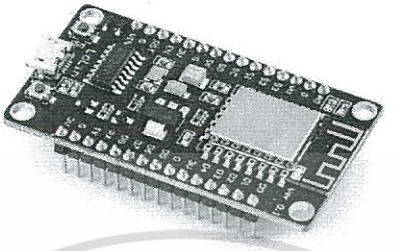
รูปที่ 3.7 ไมโครคอนโทรลเลอร์รุ่น Arduino Nano

3.2.2 โมดูลวัดความเร่ง MPU6050



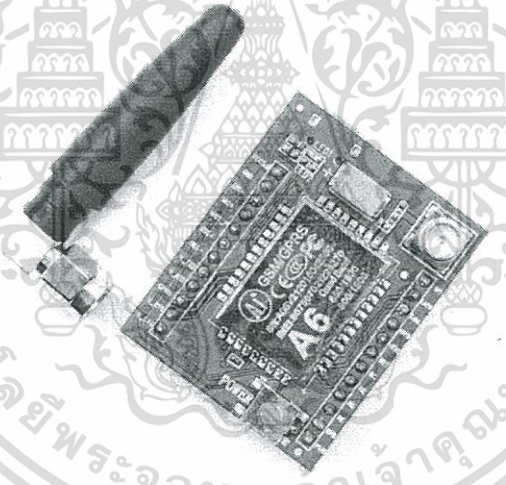
รูปที่ 3.8 โมดูลวัดความเร่ง MPU6050

3.2.3 ไมโครคอนโทรลเลอร์รุ่น NodeMCU



รูปที่ 3.9 ไมโครคอนโทรลเลอร์รุ่น NodeMCU

3.2.4 โมดูล GSM รุ่น A6



รูปที่ 3.10 โมดูล GSM รุ่น A6

3.3 การจัดเก็บผลการทดลอง

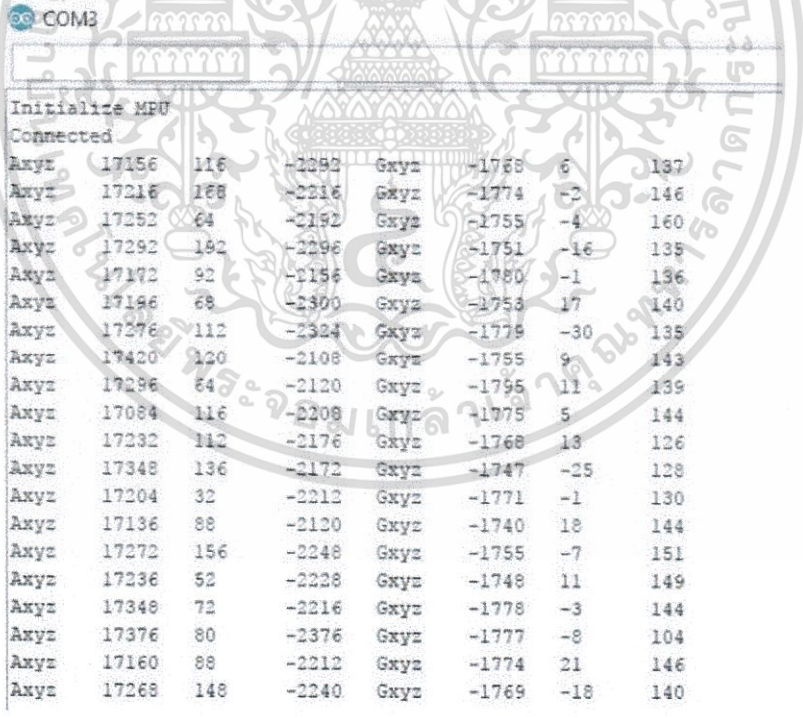
สำหรับวิธีการในการเก็บผลการทดลองในการจัดทำปริญญานิพนธ์ โดยจะเป็นการเก็บค่าจากโมดูลวัดความเร่ง เพื่อนำมาวิเคราะห์ความเร่งในขณะที่ล้มในทิศทางต่างๆ ประกอบด้วยทิศทางล้มทางซ้าย ล้มทางขวา ล้มข้างหน้า และล้มข้างหลัง โดยใช้ Gyroscope ช่วยในการหาทิศทางการล้ม และใช้ Accelerometer ในการดูแกนของความเร่งเนื่องจากความโน้มถ่วงของโลกว่ามีการเปลี่ยนแกนหรือไม่ หากมีการเปลี่ยนค่าแสดงว่ามีการล้ม โดยหากตรวจความเร่ง ไมโครคอนโทรลเลอร์จะทำการส่งค่าสถานะไปบอกเซิร์ฟเวอร์ Firebase แล้วแจ้งเตือนไปยังแอพพลิเคชั่น และการทำงานอีกส่วนจะเป็นการแจ้งเตือนผ่าน SMS โดยเมื่อไมโครคอนโทรลเลอร์ NodeMCU ตรวจจับความเร่ง จะทำการส่งข้อมูลผ่าน Serial port ไปยังไมโครคอนโทรลเลอร์ที่เชื่อมต่ออยู่กับโมดูล GSM แล้วสั่งการให้ส่ง SMS ไปยังเบอร์ของญาติ ซึ่งเป็นการแจ้งเตือน 2 ทาง เพื่อลดความเสี่ยงในกรณีที่สัญญาณ WIFI ในบ้านผู้สูงอายุเกิดมีปัญหา ซึ่งเครือข่ายโทรศัพท์นั้นมีสัญญาณครอบคลุมมากกว่า จึงใช้เป็น 2 ตัวเลือกในการแจ้งเตือน

บทที่ 4

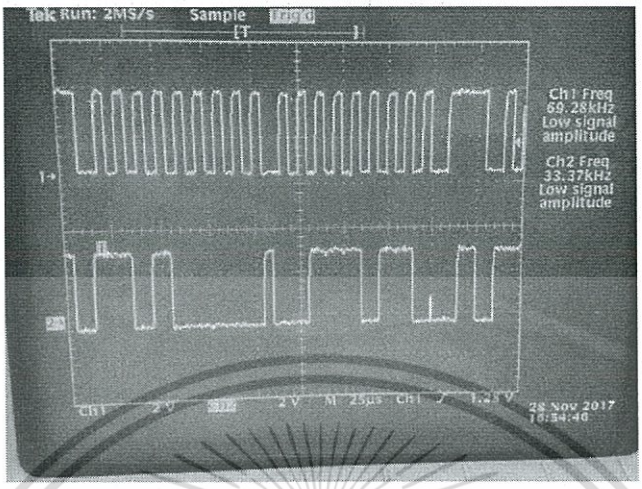
ผลการทดลอง

4.1 การอ่านค่าความเร่งเชิงเส้นจากโมดูลวัดความเร่ง MPU6050

โมดูล MPU6050 สามารถวัดได้ทั้งค่าความเร่งเชิงเส้น และความเร่งเชิงมุม แต่สำหรับการจัดทำปริญญานิพนธ์ของกลุ่มผู้จัดทำ ได้ทำการเลือกเพียงค่าความเร่งเชิงเส้นมาวิเคราะห์ โดยการอ่านค่าเริ่มจากการเขียนโปรแกรมลงบนโมดูล MPU6050 ผ่านไมโครคอนโทรลเลอร์ โดยมีการเชื่อมต่อกันแบบ I2C เมื่ออ่านค่าและแสดงผลใน Serial Monitor ของ Arduino โดยในขั้นแรกจะเป็นการอ่านค่าดิบที่รับได้จากโมดูลหลังจากที่มีการเคลื่อนที่แล้วเกิดความเร่งโมดูลจะทำการแปลงข้อมูลแอนะล็อกเป็นดิจิตอลโดยเก็บค่าเป็นบิต มีช่วงอยู่ระหว่างประมาณ -30,000 ถึง 30,000 ซึ่งมีหน่วยเป็น LSB แล้วส่งผ่าน I2C โดยแสดงค่าดิบผ่าน Serial Monitor ดังรูปที่ 4.1



รูปที่ 4.1 หน้าจอ Serial Monitor แสดงผลความเร่งที่เป็นค่าดิบจากโมดูล MPU6050



รูปที่ 4.2 สัญญาณการส่งข้อมูลแบบ I2C จากขา SCL และ SDA ของโมดูล MPU6050

จากรูปที่ 4.2 แสดงสัญญาณบิตข้อมูลที่วัดได้จากขา SCL และ SDA ที่ส่งจาก Arduino Nano โดยสัญญาณที่วัดได้จากขา SCL เป็นสัญญาณนาฬิกาที่ความถี่ 33.3 KHz และขา SDA ทำการส่งข้อมูล Serial Data ซึ่งเป็นค่าบิตของความเร่งที่อ่านได้ และจะเห็นได้ว่า SDA มีการส่งข้อมูลเมื่อ SCL มีสภาวะเป็นบิต 1

โดยในการทดลองนี้ ทางกลุ่มผู้จัดทำได้ทำการศึกษา Register ของโมดูล ค่า raw value ขนาด 16 bit โดยที่ module รับค่า analog จาก accelerometer และ gyroscope แล้วแปลงเป็น 16 bit ADC เก็บค่าลงใน register โดยอ้างอิงจาก Datasheet โดยค่าของ accelerometer ในแต่ละแกนจะเก็บค่าขนาด 16 bit ลงใน register ที่ 59-64 ด้วย Sample rate 1 KHz และของ gyroscope ในต่ละแกนจะเก็บค่าขนาด 16 bit จะเก็บลงใน register ที่ 67-72 ด้วย Sample rate 8 KHz ดังรูปที่ 4.3 ถึง 4.4

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

รูปที่ 4.3 register ในการเก็บข้อมูลของ accelerometer

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

รูปที่ 4.4 register ในการเก็บข้อมูลของ gyroscope

ต่อมาในการตั้งค่า Full scale range ของโมดูล สำหรับ accelerometer จะมีช่วงการรับค่าทั้งหมดอยู่ที่ ±2g , ±4g , ±8g และ ±16g เก็บใน Register ที่ 28 gyroscope จะมีช่วงการรับค่าทั้งหมดอยู่ที่ ±250 dps , ±500 dps , ±1000 dps และ ±2000 dps โดยที่หน่วย dps คือ degree per second หรืออัตราการเปลี่ยนแปลงมุม(องศา)ต่อวินาที โดยที่การตั้งค่า Full scale range จะสัมพันธ์กับการแปลงค่าดิบไปเป็นค่า g ($9.81 m/s^2$) และค่า degree per second ดังรูปที่ 4.5 ถึง 4.6

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

รูปที่ 4.5 register ในการกำหนดช่วงการรับค่าของ gyroscope

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

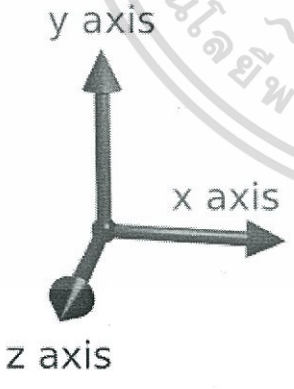
รูปที่ 4.6 register ในการกำหนดช่วงการรับค่าของ accelerometer

โดยทางกลุ่มผู้จัดทำโครงงานจะเลือกใช้ range ของ accelerometer ในช่วง ±4g และช่วงของการรับค่า gyroscope ช่วง ±500 dps ซึ่งช่วงที่เลือกจะมี LSB Sensitivity ต่างกัน ซึ่งค่า LSB Sensitivity ของ accelerometer จะมีหน่วยเป็น LSB/g ซึ่งถ้านำไปหารค่าดิบ (LSB) จะได้หน่วย g ออกมา ซึ่งคือความเร่งที่โมดูลทำการวัดได้ เช่นเดียวกับกรณีของ gyroscope นำไปหารค่าดิบ (LSB) จะได้หน่วย Degree per sec ออกมา โดยการเลือกช่วงของการเก็บค่านั้นไม่สามารถนำ LSB Sensitivity ไปหารค่าดิบได้โดยตรง จะต้องนำไปกำหนดช่วงใน Library ให้ register ที่ 28 ทำการอ่านค่าตามช่วงที่กำหนดแล้วโมดูลจะทำการแปลง 16 bit ADC เก็บค่าลงใน register ดังรูปที่ 4.3 ถึง 4.4 ตามลำดับ

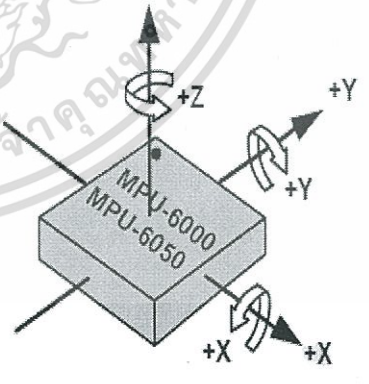
4.2 อัลกอลิทึมในการตรวจจับการล้ม

สำหรับการสร้างอัลกอลิทึมในการตรวจจับการล้ม จะต้องดำเนินการศึกษาทั้งความเร่งเชิงเส้นและความเร็วเชิงมุม เพื่อดูทั้งความเร่งและทิศทางที่เปลี่ยนแปลงไป ซึ่งจะทำการศึกษาทั้งหมด 4 กรณี คือ ล้มไปทางซ้าย ล้มไปทางขวา ล้มไปด้านหน้า และล้มไปด้านหลัง

สำหรับกรณีแรกที่ศึกษาคือการล้มไปทางซ้าย โดยในการทำการทดลองจะทำการหันทิศทางของโมดูลให้สอดคล้องดังรูปที่ 4.7 โดยจะทำให้ทิศที่แรงโน้มถ่วงของโลกกระทำกับแกน y และทิศของการหมุนรอบแกนเป็นไปดังรูปที่ 4.8



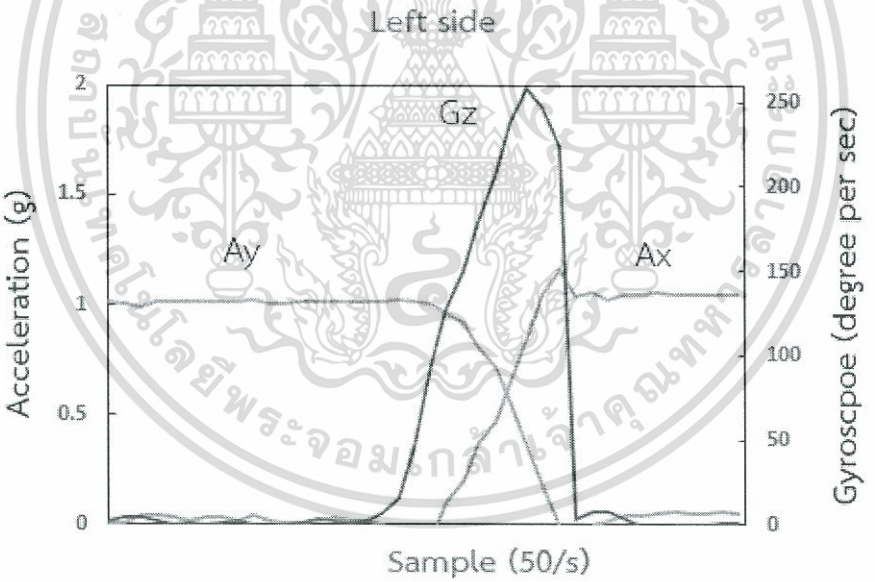
รูปที่ 4.7 ทิศของโมดูลที่อ้างอิงในการทำการทดลอง



รูปที่ 4.8 ทิศการหมุนของแกน

โดยในการทดลองเมื่อมีการล้มไปทางซ้ายนั้นจะเป็นการหมุนรอบแกน y และทำให้ค่าที่อ่านได้จาก Gyroscope ในแกน Z (Gz) มีค่าเป็นบวกตามกฎมือขวา โดยจากกราฟแสดงความสัมพันธ์ของ

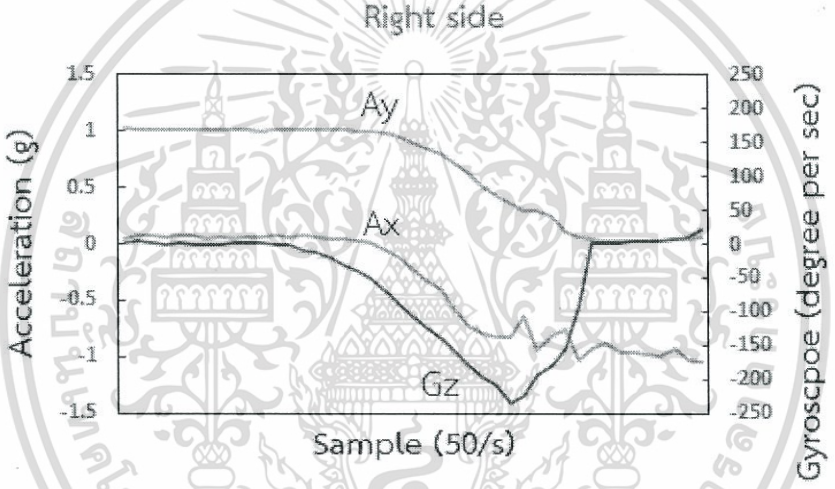
ความเร่งและความเร็วเชิงมุมรูปที่ 4.9 เมื่อทดลองล้มไปทางซ้าย ค่าของ Gz หรือความเร็วเชิงมุมรอบแกน z จะมีการเปลี่ยนแปลงจาก 0 ขึ้นไปถึงประมาณ 250 Degree per sec และค่อยๆลดลงจนมีค่าเป็น 0 ในขณะนี้โมดูลจะไม่สามารถตรวจจับความเร็วเชิงมุมได้แล้วซึ่งเป็นสถานะที่ล้มไปทางซ้ายแล้ว และในขณะเดียวกันกรณีที่ล้มไปทางซ้ายจากเดิมนั้นแกน y นั้นจะมีความเร่งเนื่องจากแรงโน้มถ่วงของโลกกระทำอยู่ เป็นแรง 1g (Ay) เมื่อเอียงไปทางซ้ายทำให้แกนหมุนไป 90 องศา ทำให้ความเร่งเนื่องจากแรงโน้มถ่วงที่กระทำกับแกน y นั้นค่อยๆลดลง ส่งผลให้ Ay ลดลงจนเป็น 0 เนื่องจากแรงที่กระทำในทิศตั้งฉากนั้นเท่ากับ 0 และ Ax มีค่าเพิ่มขึ้นเนื่องจากแกน x นั้นหมุนมา 90 องศา ในทิศของความเร่งเนื่องจากแรงโน้มถ่วงของโลกแทนแกน y ทำให้ Ax มีค่า 1g โดยความชันของการลดลงของ Ay นั้นจะมีแนวโน้มเดียวกับความชันในการเพิ่มขึ้นของ Ax เนื่องจากมีความเร่งเนื่องจากแรงโน้มถ่วงของโลกเป็นแรงอ้างอิง ส่วนค่าของความชันว่าจะมากหรือน้อยนั้น จะขึ้นอยู่กับความเร็วในการล้มที่หมุนแกนทั้ง 2 และเนื่องจากแกน z เป็นแกนอ้างอิงในการหมุนไปทางซ้าย จึงไม่มีความเร่งที่กระทำกับแกน z จึงไม่ได้นำความเร่งในแกน z มาพิจารณาในการทดลองล้มไปทางซ้าย



รูปที่ 4.9 ความสัมพันธ์ของความเร่งกับความเร็วเชิงมุมของการล้มไปทางซ้าย

ในกรณีที่สองจะทำการศึกษาการล้มไปทางขวา โดยในการทดลองในการล้มไปทางขวานั้นจะเป็นการหมุนรอบแกน z ในทิศตามเข็ม และทำให้ความเร็วเชิงมุมรอบแกน z (Gz) มีค่าเป็นลบดังรูปที่ 4.8 โดยจากกราฟแสดงความสัมพันธ์ของความเร่งและความเร็วเชิงมุมรูปที่ 4.10 เมื่อทดลองการล้มไปทางขวา ค่าของ Gz หรือความเร็วเชิงมุมรอบแกน z จะมีการเปลี่ยนแปลงจาก 0 ลดลงไปถึงประมาณ -250 Degree per sec แล้วค่อยๆกลับมามีค่าเป็น 0 เนื่องจากล้มเรียบแล้วจึงไม่มีการ

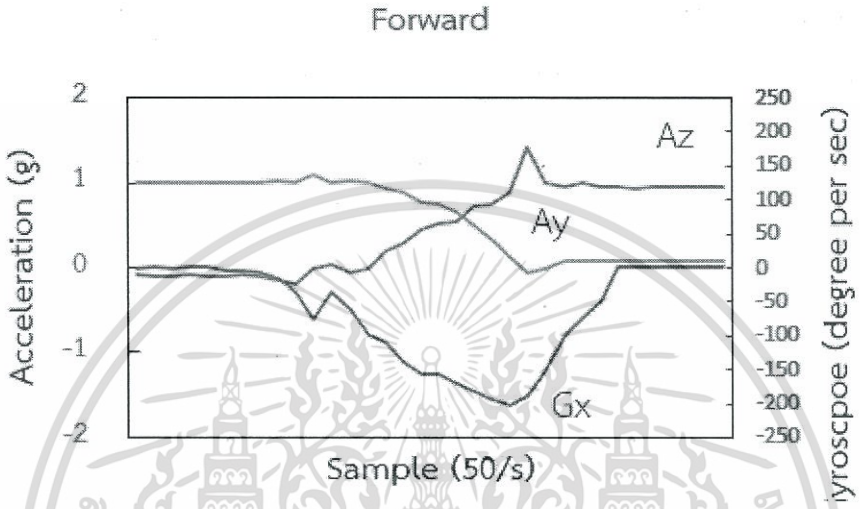
เปลี่ยนแปลงเชิงมุม และในขณะเดียวกันที่ล้มไปทางขวา จากเดิมนั้นแกน y นั้นมีความเร่งเนื่องจากแรงโน้มถ่วงของโลกกระทำอยู่ เป็นแรง 1g เอียงไปทางขวาทำให้แกนหมุน 90 องศา ทำให้ความเร่งจากแรงโน้มถ่วง ที่กระทำกับแกน y นั้นค่อยๆลดลง ส่งผลให้ Ay ลดลงจนเป็น 0 ในขณะเดียวกัน Ax มีค่าลดลงเรื่อยๆเนื่องจากแกน x นั้นหมุนไป 90 องศา ในทิศทางตรงข้ามกับความเร่งเนื่องจากแรงโน้มถ่วงของโลก ทำให้ Ax ค่าค่อยๆลดลงจนมีค่า -1g และเมื่อพิจารณาจากรูปที่ 4.10 ทิศของแกน y จะทำมุม 90 องศากับแรงโน้มถ่วง ซึ่งคือระนาบของพื้นทำให้ขณะนั้น Ay มีค่าเป็น 0 และ Gz มีค่าเป็น 0 เพราะไม่มีการเปลี่ยนแปลงความเร็วเชิงมุมแล้ว จึงสรุปได้ว่าในขณะที่ Ay และ Gz มีค่าเป็น 0 นั้นได้มีการล้มไปถึงพื้นเรียบร้อยแล้ว



รูปที่ 4.10 ความสัมพันธ์ของความเร่งกับความเร็วเชิงมุมของการล้มไปทางขวา

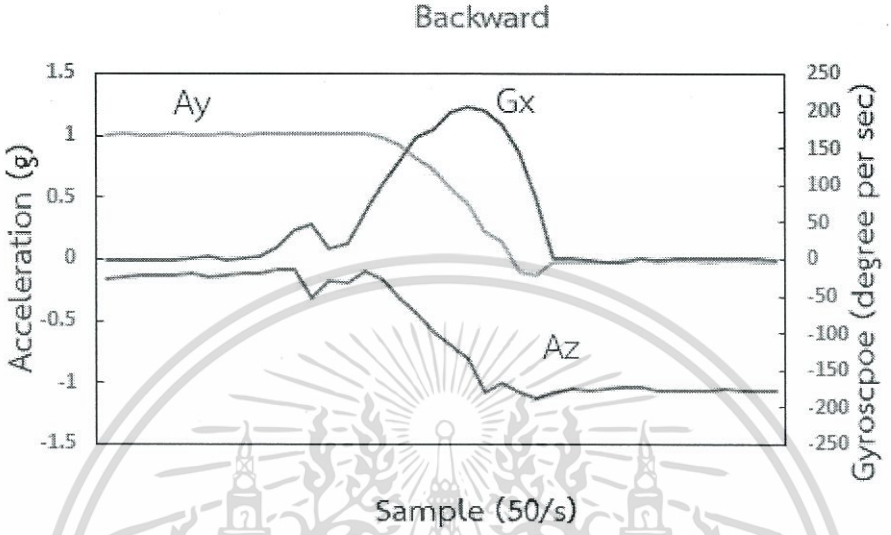
ในกรณีที่สามจะทำการศึกษาการล้มไปข้างหน้า โดยในการทดลองในการล้มไปข้างหน้านั้นจะเป็นการหมุนรอบแกน x ในทิศตามเข็ม และทำให้ความเร็วเชิงมุมรอบแกน x (Gx) มีค่าเป็นลบดังรูปที่ 4.8 โดยจากกราฟแสดงความสัมพันธ์ของความเร่งและความเร็วเชิงมุมรูปที่ 4.11 เมื่อทดลองล้มไปข้างหน้า ค่าของ Gx จะมีการเปลี่ยนแปลงจาก 0 ลดลงไปถึงประมาณ -250 Degree per sec แล้วค่อยๆกลับมามีค่าเป็น 0 เนื่องจากล้มเรียบแล้วจึงไม่มีการเปลี่ยนแปลงความเร็วเชิงมุม ในขณะเดียวกันจากเดิมนั้นแกน y นั้นมีความเร่งเนื่องจากแรงโน้มถ่วงของโลกกระทำอยู่ เป็นแรง 1g เมื่อมีการเอียงไปข้างหน้า ทำให้แกนหมุน 90องศาไปทิศหน้า ทำให้ความเร่งเนื่องจากแรงโน้มถ่วงที่กระทำกับแกน y นั้นค่อยๆลดลง ส่งผลให้ Ay ลดลงจนเป็น 0 เนื่องจากแรงที่กระทำในทิศตั้งฉากนั้นเท่ากับ 0 และ Az มีค่าเพิ่มขึ้นเนื่องจากแกน z นั้นหมุนมา 90องศา ตรงกับทิศของความเร่งเนื่องจากแรงโน้มถ่วงของโลกแทนแกน y ทำให้ Az มีค่า 1g และเมื่อพิจารณาจากรูปที่ 4.11 พบว่าทิศของแกน y ทำมุม 90 องศา

กับแรงโน้มถ่วงของโลก ซึ่งคือระนาบของพื้นทำให้ขณะนั้น Ay มีค่าเป็น 0 และ Gx มีค่าเป็น 0 เพราะไม่มีการเปลี่ยนแปลงความเร็วเชิงมุมแล้ว จึงสรุปได้ว่าในขณะที่ Ay และ Gx เป็น 0 นั้นได้มีการล้มไปถึงพื้นแล้ว



รูปที่ 4.11 ความสัมพันธ์ของความเร่งกับความเร็วเชิงมุมของการล้มไปข้างหน้า

ในกรณีที่จะทำการศึกษาการล้มไปข้างหลัง โดยในการทดลองในการล้มไปข้างหลังนั้นจะเป็นการหมุนรอบแกน x ในทิศทวนเข็ม และทำให้ความเร็วเชิงมุมรอบแกน x (Gx) มีค่าเป็นบวกดังรูปที่ 4.8 โดยจากกราฟแสดงความสัมพันธ์ของความเร่งและความเร็วเชิงมุมรูปที่ 4.12 เมื่อทดลองล้มไปข้างหน้า ค่าของ Gx จะมีการเปลี่ยนแปลงจาก 0 เพิ่มขึ้นไปถึงประมาณ 250 Degree per sec แล้วค่อยๆกลับมามีค่าเป็น 0 เนื่องจากล้มเรียบแล้วจึงไม่มีการเปลี่ยนแปลงเชิงมุม แล้วจากเดิมนั้น Ay นั้นมีความเร่งเนื่องจากแรงโน้มถ่วงของโลกกระทำอยู่ เป็นแรง 1g เอียงไปข้างหน้าทำให้แกนหมุนไป 90 องศา ทำให้ความเร่งเนื่องจากแรงโน้มถ่วงที่กระทำกับแกน y นั้นค่อยๆลดลง ส่งผลให้ Ay ลดลงจนเป็น 0 ในขณะเดียวกัน Az มีค่าลดลงเรื่อยๆเนื่องจากแกน x นั้นหมุนไป 90 องศา ในทิศทางตรงข้ามกับความเร่งเนื่องจากแรงโน้มถ่วงของโลก ทำให้ Ax ค่าค่อยๆลดลงจนมีค่า -1g และเมื่อพิจารณาจากรูปที่ 4.12 ทิศของแกน y ทำมุม 90 องศากับแรง g ซึ่งคือระนาบของพื้นทำให้ขณะนั้น Ay เป็น 0 และ Gx เป็น 0 เพราะไม่มีการเปลี่ยนแปลงความเร็วเชิงมุมแล้ว จึงสรุปได้ว่าในขณะที่ Ay และ Gx เป็น 0 นั้นได้มีการล้มไปถึงพื้นเรียบร้อยแล้ว



รูปที่ 4.12 ความสัมพันธ์ของความเร่งกับความเร็วเชิงมุมของการล้มไปข้างหลัง

หลังจากนั้นนำข้อมูลจากการล้มในทิศทางต่างๆมาเขียนอัลกอริทึม ซึ่งจากการทดลองจะเห็นได้ว่าความสัมพันธ์ของความเร่งในทิศที่ความเร่ง 1g กระทำกับความเร็วเชิงมุมนั้นสัมพันธ์กันเมื่อทำการล้มถึงพื้นแล้ว ความเร่งเนื่องจากแรงโน้มถ่วงของโลกจะเหลือ 0 g เพราะทิศของแกนจะตั้งฉากกับแรง g และเมื่อล้มถึงพื้นแล้วค่าของความเร็วเชิงมุมจะเป็น 0 เพราะไม่มีการเปลี่ยนแปลงองศาต่อวินาทีแล้ว จะนำค่าทั้ง 2 ค่าใน 4 กรณีไปเขียน algorithm ในการตรวจจับการล้มดังนี้

Algorithm fall detecting in different direction

Input : Ax , Ay , Az , Gx , Gy , Gz

Output : Fall detect

(1) fall left side : if gz <threshold (while rotating) and if ay && gz == 0 then fall detect

(2) fall right side : if gz > threshold (while rotating) and if ay && gz == 0 then fall detect

(3) fall forward : if gx > threshold (while rotating) and if ay && gx == 0 then fall detect

(4) fall backward : if gx < threshold (while rotating) and if ay && gx == 0 then fall detect

โดยในการเขียนโปรแกรมตรวจจับแต่ละทิศทางจะแบ่งเป็น 2 เงื่อนไขหลัก เงื่อนไขแรกเป็นการตรวจจับว่าค่าของความเร็วเชิงมุมนั้นมีค่าพุ่งสูง (ในกรณีล้มทวนเข็ม) หรือลดลงติดลบ (ในกรณีทิศตามเข็ม) หรือไม่ อาทิเช่น กรณีแรก เป็นการล้มไปทางซ้าย รอบแกน y ในทิศทวนเข็ม จะให้ค่า Gz เป็นบวก และโปรแกรมจะทำการเก็บค่า Gz ลงในตัวแปร int ทุกขณะ และนำค่าต่อไปไปเปรียบเทียบกับค่าเดิม เมื่อ Gz มีค่าเพิ่มขึ้นนั้นแสดงว่ามีการหมุน ซึ่งจะมีค่าเพิ่มขึ้นถึงจุดสูงสุดขณะหนึ่ง แล้วค่าจะเริ่มลดลง เมื่อโปรแกรมตรวจจับได้ว่า Gz มีค่าลดลงแสดงว่าเกิดการหมุน จะเข้าเงื่อนไขแรก แล้วทำการตรวจสอบต่อไปว่าเกิดการล้มจริงๆหรือไม่ หรือเพียงแค่การเคลื่อนไหวลักษณะโน้มไปทางซ้าย และจะทำการตรวจสอบต่อมาว่า เมื่อ Ay และ Gz เป็น 0 แสดงว่าทิศของแกน y ระนาบไปกับพื้น และ Gz เป็น 0 เงื่อนไขแรกที่ตรวจสอบ Gz ค่อยๆลดลง แสดงว่า เกิดการล้มและนอนราบไปกับพื้น จึงเข้าเงื่อนไขที่สอง จะสรุปได้ว่าเกิดการล้ม และสำหรับการล้มทิศทางอื่นๆก็ใช้ algorithm เดียวกัน แต่จะต้องทำการเปลี่ยนแกนการหมุนและเปลี่ยนทิศในการตัดสินใจดังที่แสดงใน algorithm 4 ข้อด้านบน

สำหรับ Threshold ในการตัดสินใจของส่วน Gyroscope จากการทดลองซ้ำโดยพิจารณาเพียง Magnitude ไม่คิดทิศทาง ค่าเฉลี่ยของ Gyroscope ขณะเปลี่ยนแปลงจะมีค่าประมาณ 200 degrees/s แต่ทางผู้จัดทำจะใช้ค่า Threshold เพียง 150 degrees/s ครอบคลุมกรณีล้มที่มีการพยุงตัวได้ทัน ส่งผลให้ค่าความเร่งเชิงมุมอ่านได้น้อยและอาจจับการล้มไม่ได้ทั้งที่มีการล้ม

4.3 ส่งข้อมูลไปยังฐานข้อมูล

สำหรับการส่งข้อมูลระหว่างไมโครคอนโทรลเลอร์ไปยังฐานข้อมูล โดยจะใช้ฐานข้อมูลของ Google Firebase เนื่องจากมีทั้งบริการ Cloud Messaging ที่สามารถเชื่อมต่อการแจ้งเตือนไปยังแอพพลิเคชั่นได้โดยตรงผ่าน Api key และมีบริการฐานข้อมูลให้ใช้ โดยข้อมูลที่ทำการส่งจากไมโครคอนโทรลเลอร์ไม่ใช่ข้อมูลที่มีจำนวนมาก เป็นเพียงการส่ง ข้อมูลเป็นที่ตรงกับเงื่อนไขการล้มแล้วทำการแจ้งเตือนไปยังแอพพลิเคชั่นเท่านั้น ไม่ได้มีข้อมูลที่มีหลาย table มากแล้วไม่ต้องเชื่อมโยงข้อมูลถึงกันจึงไม่ได้เลือกทำฐานข้อมูลเอง โดยโฟลว์ชาร์ตการทำงานของการส่งข้อมูลแสดง ดังรูปที่ 4.13

รูปที่ 4.13 โฟลว์ชาร์ตการส่งข้อมูลไปยังฐานข้อมูล Firebase

สำหรับโปรแกรมการทำงานในส่วนของไมโครคอนโทรลเลอร์นั้นจะเริ่มจากการสร้างโปรเจ็คใน Firebase แล้วมีส่วนของ API Key เพื่อนำมาติดต่อสำหรับการส่งข้อมูลจากแหล่งใดๆมายังฐานข้อมูลของ Firebase ดังรูปที่ 4.14

รูปที่ 4.14 คีย์ API ในการเชื่อมต่อเข้าฐานข้อมูล

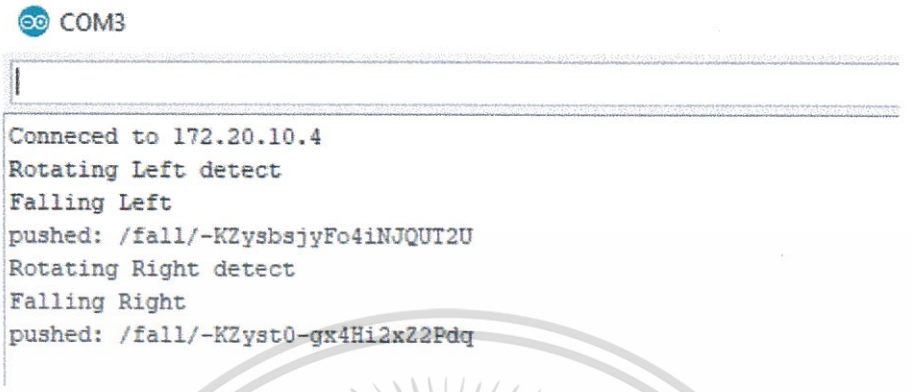
หลังจากนั้นทำการตั้งค่าในส่วนของไมโครคอนโทรลเลอร์ทำการกำหนดตัวแปรคีย์ API และ**ตัวโปรแกรม URL ของ Firebase จากนั้นเขียนเงื่อนไขในการส่งข้อมูล** โดยอ้างอิงถึงสัปดาห์ที่แล้ว เมื่อ**เกิดการล้ม จะเพิ่ม** Statement **ให้ส่งข้อมูลไปยังฐานข้อมูลของ** Firebase โดยมี statement ดังนี้

If (Fall detecting){
Send data to Firebase server
}

หลังจากนั้นลองรันโปรแกรมลงไมโครคอนโทรลเลอร์แล้วส่งข้อมูลโดยมีโค้ดในการทำงานดังนี้

```
void setup() {
  Serial.begin(9600);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("connecting");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());

  FirebaseCloudMessaging fcm(SERVER_KEY);
  FirebaseCloudMessage message =
    FirebaseCloudMessage::SimpleNotification("Hello World!", "What's happening?");
  FirebaseError error = fcm.SendMessageToUser(CLIENT_REGISTRATION_ID, message);
  if (error) {
    Serial.print("Error:");
    Serial.print(error.code());
    Serial.print(" :: ");
    Serial.println(error.message().c_str());
  } else {
    Serial.println("Sent OK!");
  }
}
```

ผลลัพธ์ที่ได้ใน Serial Monitor จะเริ่มจากการที่ไมโครคอนโทรลเลอร์เชื่อมต่อกับ WIFI และทำการแสดง IP Address แล้วหลังจากนั้นโมดูลสามารถตรวจจับการล้มจากอัลกอลิทึมในการตรวจจับการล้ม หากมีการตรวจจับได้ว่าเกิดการล้ม ผลลัพธ์ที่ได้คือไมโครคอนโทรลเลอร์จะส่งข้อมูล ไปยังเซิร์ฟเวอร์ของ Firebase ดังรูปที่ 4.15



รูปที่ 4.15 Serial Monitor ของการส่งข้อมูลไปยัง Firebase

เมื่อตรวจสอบในส่วนของฐานข้อมูลใน Firebase จะพบว่ามีข้อมูลถูกส่งมายังที่ฐานข้อมูล ดังรูปที่ 4.16

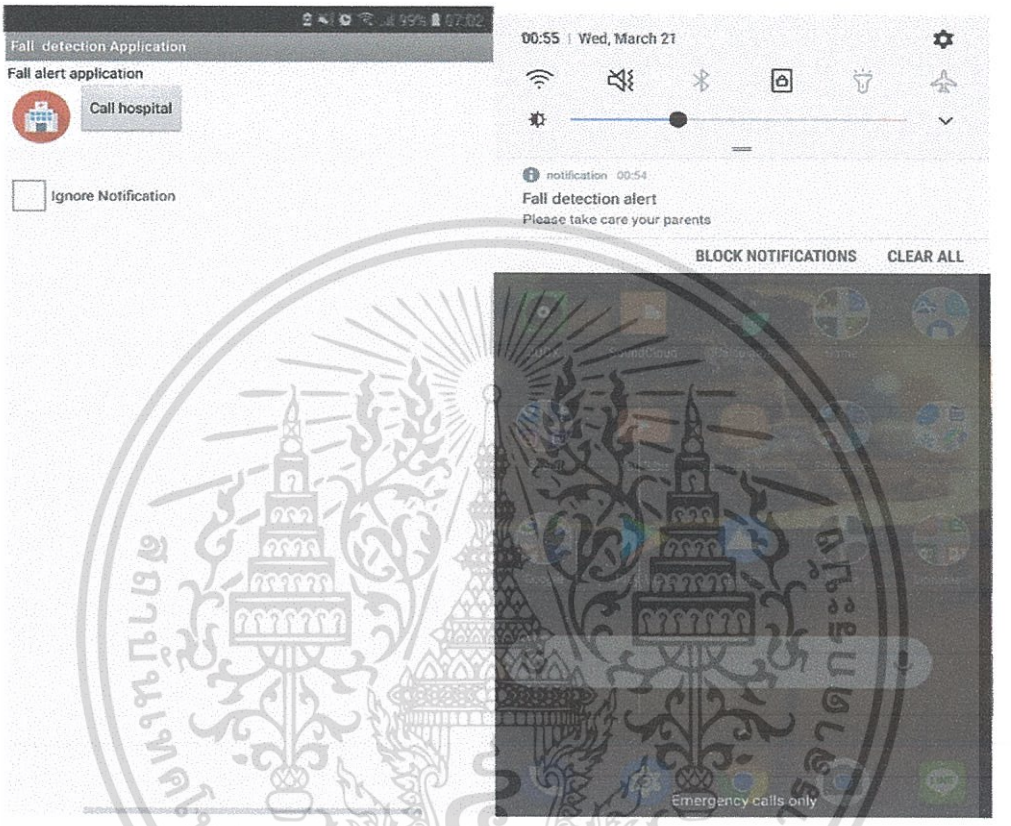
รูปที่ 4.16 ข้อมูลที่ถูกส่งมายังฐานข้อมูล

4.4 การแจ้งเตือนไปยังแอพพลิเคชั่น

สำหรับการพัฒนาจะใช้เครื่องมือที่มีชื่อว่า MIT App Inventor ซึ่งเป็นเครื่องมือที่ใช้สำหรับสร้างแอพพลิเคชั่นสำหรับสมาร์ทโฟนและแท็บเล็ตที่เป็นระบบปฏิบัติการแอนดรอยด์โดยที่โฟลวชาร์ตการทำงานของแอพพลิเคชั่นแสดงดังรูปที่ 4.13 ในหัวข้อที่ 4.3

โดยหน้าตาของแอพพลิเคชั่นที่ออกแบบแล้วเมื่อเกิดการล้มจะทำการแจ้งเตือนยังแถบ Notification และมีปุ่ม Call hospital ไว้สำหรับการกดโทรเรียกพยาบาลได้ทันที โดยเบอร์โทรที่กำหนดจะเป็นเบอร์โทรของโรงพยาบาลที่กำหนดไว้ ซึ่งสามารถกำหนดโรงพยาบาลใกล้บ้าน เพื่อที่

สามารถเข้าช่วยเหลือได้ทันท่วงที โดยหน้าตาของแอพพลิเคชั่น และขณะที่มีการแจ้งเตือน จะแสดงได้ดังรูปที่ 4.17

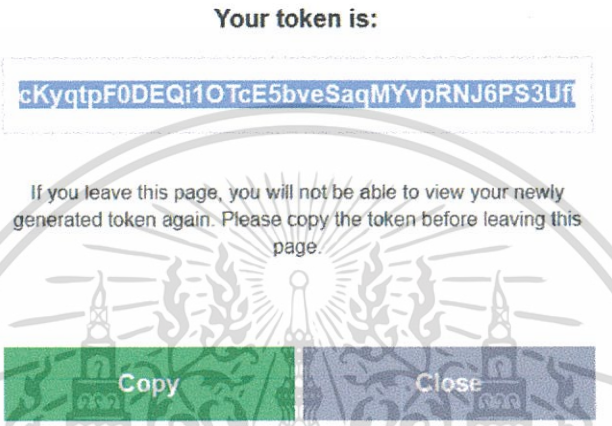


รูปที่ 4.17 ลักษณะของแอพพลิเคชั่นและการแจ้งเตือน

4.5 ส่งการแจ้งเตือนผ่านแอพพลิเคชั่น LINE

ในการทำงานของแอพพลิเคชั่นบนโทรศัพท์แอนดรอยด์นั้น การจะทำให้โปรแกรมแจ้งเตือนทำการแจ้งเตือน ผู้ใช้จำเป็นต้องเปิดหน้าจอหรือ active เครื่องไว้ ซึ่งในกรณีที่ทางญาติหรือผู้ดูแลนั้นอาจไม่ได้เปิดหน้าจอโทรศัพท์ไว้ ระบบจะใช้แอพพลิเคชั่น LINE ในการช่วย active หน้าจอ โดยเมื่อส่งการแจ้งเตือนผ่านทาง LINE แล้ว หน้าจอโทรศัพท์จะแสดงผลการแจ้งเตือนของ LINE ซึ่งทำให้หน้าจอ active และเป็นการแจ้งเตือนไปในตัวด้วย ซึ่งเป็นอีกหนึ่งทางเลือกในกรณีที่การแจ้งเตือนผ่านทางแอพพลิเคชั่นของทางผู้จัดทำโครงงานเกิดขัดข้อง

โดยจะใช้ NodeMCU ในการรับข้อมูลจากเซ็นเซอร์ที่ทำการตรวจจับการล้มแล้วส่งข้อมูลผ่านทาง LINE ไปยังแอคเคาท์ที่ทำการลงทะเบียนไว้ ผ่านการใช้ API Token ดังรูปที่ 4.18 ซึ่งเรียกผ่าน HTTP POST ซึ่งข้อจำกัดของ LINE Notify คือ สามารถส่งแจ้งเตือนได้เฉพาะผู้ที่ขอใช้ หรือกลุ่มที่ผู้ขอใช้เป็นสมาชิกเท่านั้น โดยมีคำสั่งการทำงานดังรูปที่ 4.19

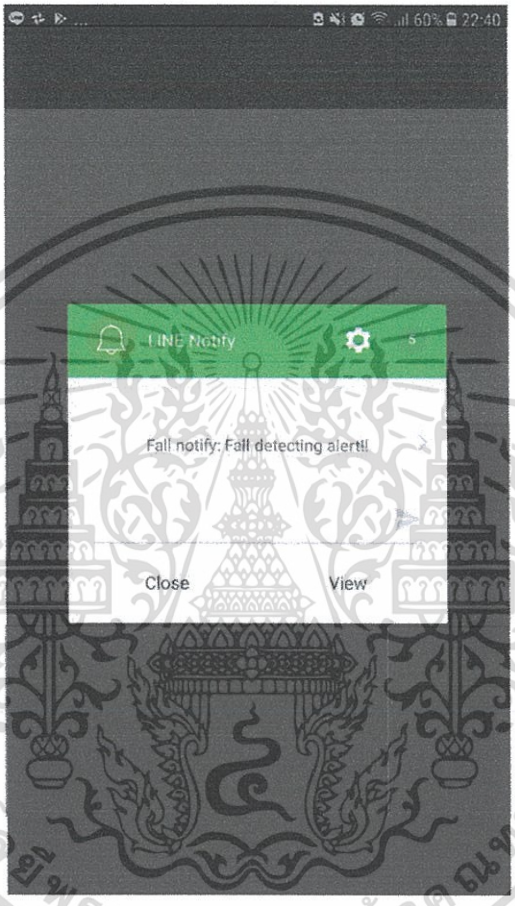


รูปที่ 4.18 Token ที่ทำการลงทะเบียนไว้กับทางไลน์

```
void Line_Notify(String message) {
  WiFiClientSecure client;

  if (!client.connect("notify-api.line.me", 443)) {
    Serial.println("connection failed");
    return;
  }

  String req = "";
  req += "POST /api/notify HTTP/1.1\r\n";
  req += "Host: notify-api.line.me\r\n";
  req += "Authorization: Bearer " + String(LINE_TOKEN) + "\r\n";
  req += "Cache-Control: no-cache\r\n";
  req += "User-Agent: ESP8266\r\n";
  req += "Content-Type: application/x-www-form-urlencoded\r\n";
  req += "Content-Length: " + String(String("message=" + message).length()) + "\r\n";
  req += "\r\n";
  req += "message=" + message;
  // Serial.println(req);
  client.print(req);

  delay(20);
```

รูปที่ 4.19 คำสั่งการทำงานของการแจ้งเตือนผ่านทางไลน์

ซึ่งเมื่อมีการตรวจจับการล้ม ทางไมโครคอนโทรลเลอร์จะทำการส่งการแจ้งเตือนมาดังรูปที่ 4.20 ซึ่งข้อความในการแจ้งเตือนสามารถเปลี่ยนได้ โดยจะต้องทำการแปลงข้อความเป็น ASCII แล้วนำไปใส่ในคำสั่งของไมโครคอนโทรลเลอร์ แล้วจึงจะสามารถแจ้งเตือนเป็นภาษาไทยได้



รูปที่ 4.20 หน้าจอแจ้งเตือนจาก Line เมื่อตรวจจับการล้ม

4.6 ส่งการแจ้งเตือนทาง SMS

นอกเหนือจากการแจ้งเตือนด้วยแอพพลิเคชั่นในหัวข้อที่ 4.3 แล้วสามารถทำการแจ้งเตือนโดยการใช้ SMS อีกด้วย ในกรณีที่ญาติหรือผู้ดูแลนั้นอาจไม่ได้เชื่อมต่ออินเทอร์เน็ตอยู่ตลอดเวลา หรือสัญญาณอินเทอร์เน็ตเกิดขัดข้อง แต่โดยพื้นฐานของระบบ GSM แล้วจะยังสามารถทำงานได้อย่างครอบคลุมกว่า จึงเลือก SMS เป็นอีกหนึ่งทางเลือกในการแจ้งเตือน

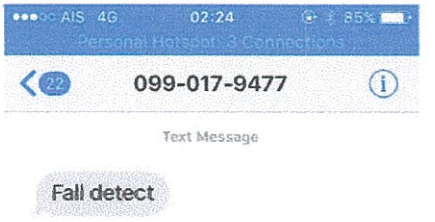
โดยโมดูล GSM A6 นั้นเป็นโมดูลการเชื่อมต่อ GSM ขนาดเล็ก มีขนาด 35x45 mm รอบรับการทำงาน 4 ความถี่ คือ 850,900,1800 และ 1900 MHz ตัวโมดูลทำงานที่ 3.3 V ถึง 4.2 V รองรับ

การโทรด้วยเสียง และรองรับการส่ง SMS โดยมีการทำงานผ่านคำสั่ง AT Command เชื่อมต่อกับไมโครคอนโทรลเลอร์แบบอนุกรม Serial Communication โดยการทำงานจะทำการติดตั้งไว้พร้อมกับไมโครคอนโทรลเลอร์ที่ติดตั้งเพื่อตรวจจับการล้มกับผู้สูงอายุ และหากมีการล้มเกิดขึ้นจะทำการส่งการแจ้งเตือนไปยังญาติหรือผู้ดูแลอีกทางหนึ่งโดยการสั่งให้โมดูลทำงานจะทำการเชื่อมต่อแบบอนุกรม Serial Communication ผ่านทางขา RX , TX ของไมโครคอนโทรลเลอร์และตัวโมดูล แล้วทำการอัพโหลดคำสั่งการทำงาน ซึ่งมีคำสั่งการทำงานดังรูปที่ 4.21

```
char phone_no[]="0982691136";

void setup() {
Serial.begin(9600);
delay(300);
Serial.println("AT");
Serial.println("AT+CMGF=1");
delay(2000);
Serial.print("AT+CMGS=\"");
Serial.print(phone_no);
Serial.write(0x22);
Serial.write(0x0D);
Serial.write(0x0A);
delay(2000);
Serial.print("Fall detect!");
delay(500);
Serial.println (char(26));
```

รูปที่ 4.21 คำสั่งการทำงานของการส่ง SMS

จะเห็นได้ว่าเมื่อกำหนดเบอร์ปลายทางเป็นอันดับแรก แล้วสั่งการทำงานโมดูลผ่าน AT Command ซึ่งสามารถทำได้โดยการ Serial Print คำสั่งได้เลย โดยคำสั่ง AT+CMGF เป็นคำสั่งที่สั่งให้โมดูลทำงานในโหมด SMS และสั่งให้ส่ง SMS ผ่านคำสั่ง AT+CMGS ตัวโมดูลจะทำการส่ง SMS ไปยังหมายเลขปลายทางได้ดังรูปที่ 4.22



รูปที่ 4.22 SMS ที่ถูกส่งมาจากโมดูล

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

สำหรับปริญญานิพนธ์เรื่องระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุ โดยจะแบ่งการทำงานออกเป็น 2 ส่วนคือ ส่วนแรกเป็นส่วนของการตรวจจับการล้มจากโมดูลวัดความเร่ง โดยใช้เงื่อนไขหรืออัลกอลิทึมในการตรวจจับ ทั้งความเร่งเชิงเส้นและความเร่งเชิงมุมภายใต้เงื่อนไขที่กำหนด เมื่อมีการตรวจจับการล้มได้จะส่งต่อไปยังส่วนที่สองคือ ส่วนของการแจ้งเตือน โดยจะทำการแจ้งเตือนทั้งหมด 3 ทาง ประกอบด้วยทางแอพพลิเคชั่นที่ได้สร้างขึ้นมา ทางการแจ้งเตือนทาง LINE ซึ่งเป็นการแจ้งเตือนอีกทางในกรณีที่แอพพลิเคชั่นเกิดปัญหา และสุดท้าย ทางการแจ้งเตือนผ่าน SMS ในกรณีที่สัญญาณของทางฝั่งใดฝั่งหนึ่งมีปัญหาเพราะการส่ง SMS ผ่านระบบ GSM นั้นครอบคลุมกว่า

5.2 ข้อเสนอแนะ

ระบบตรวจจับการล้มและแจ้งเหตุสำหรับผู้สูงอายุสามารถเลือกวิธีการแจ้งเตือนตามที่ผู้ใช้งานต้องการได้

บรรณานุกรม

[1] Farhad Hossain & Md. Liakot Ali & Md. Zahurul Islam and Hossen Mustafa , "A Direction-Sensitive Fall Detection System Using Single 3D Accelerometer and Learning Classifier" [Online]. Available: http://ieeexplore.ieee.org/document/7835372/

[2] https://www.bumrungrad.com/th/arrhythmia-treatment-center-bangkok-thailand/conditions/arrhythmia

[3] http://thainurseclub.blogspot.com/2014/06/blog-post_5022.html

ภาคผนวก ก

โปรแกรมคำสั่งการทำงาน

โค้ดส่วนอ่านค่า MPU6050 จาก NodeMCU

```
#include <Wire.h>
// MPU6050 Slave Device Address
const uint8_t MPU6050SlaveAddress = 0x68;

// Select SDA and SCL pins for I2C communication
const uint8_t scl = D6;
const uint8_t sda = D7;

// sensitivity scale factor respective to full scale setting provided in datasheet
const uint16_t AccelScaleFactor = 16384;
const uint16_t GyroScaleFactor = 131;

// MPU6050 few configuration register addresses
const uint8_t MPU6050_REGISTER_SMPLRT_DIV   = 0x19;
const uint8_t MPU6050_REGISTER_USER_CTRL    = 0x6A;
const uint8_t MPU6050_REGISTER_PWR_MGMT_1   = 0x6B;
const uint8_t MPU6050_REGISTER_PWR_MGMT_2   = 0x6C;
const uint8_t MPU6050_REGISTER_CONFIG       = 0x1A;
const uint8_t MPU6050_REGISTER_GYRO_CONFIG  = 0x1B;
const uint8_t MPU6050_REGISTER_ACCEL_CONFIG = 0x1C;
const uint8_t MPU6050_REGISTER_FIFO_EN      = 0x23;
const uint8_t MPU6050_REGISTER_INT_ENABLE   = 0x38;
const uint8_t MPU6050_REGISTER_ACCEL_XOUT_H = 0x3B;
const uint8_t MPU6050_REGISTER_SIGNAL_PATH_RESET  = 0x68;

int16_t AccelX, AccelY, AccelZ, Temperature, GyroX, GyroY, GyroZ;

void setup() {
  Serial.begin(9600);
  Wire.begin(sda, scl);
```

```
  MPU6050_Init();
}

void loop() {
  double Ax, Ay, Az, T, Gx, Gy, Gz;

  Read_RawValue(MPU6050SlaveAddress, MPU6050_REGISTER_ACCEL_XOUT_H);

  //divide each with their sensitivity scale factor
  Ax = (double)AccelX/AccelScaleFactor;
  Ay = (double)AccelY/AccelScaleFactor;
  Az = (double)AccelZ/AccelScaleFactor;
  T = (double)Temperature/340+36.53; //temperature formula
  Gx = (double)GyroX/GyroScaleFactor;
  Gy = (double)GyroY/GyroScaleFactor;
  Gz = (double)GyroZ/GyroScaleFactor;

  Serial.print("Ax: "); Serial.print(Ax);
  Serial.print(" Ay: "); Serial.print(Ay);
  Serial.print(" Az: "); Serial.print(Az);
  Serial.print(" T: "); Serial.print(T);
  Serial.print(" Gx: "); Serial.print(Gx);
  Serial.print(" Gy: "); Serial.print(Gy);
  Serial.print(" Gz: "); Serial.println(Gz);

  delay(100);
}

void I2C_Write(uint8_t deviceAddress, uint8_t regAddress, uint8_t data){
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
```

```
  Wire.write(data);
  Wire.endTransmission();
}

// read all 14 register
void Read_RawValue(uint8_t deviceAddress, uint8_t regAddress){
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.endTransmission();
  Wire.requestFrom(deviceAddress, (uint8_t)14);
  AccelX = (((int16_t)Wire.read()<<8) | Wire.read());
  AccelY = (((int16_t)Wire.read()<<8) | Wire.read());
  AccelZ = (((int16_t)Wire.read()<<8) | Wire.read());
  Temperature = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroX = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroY = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroZ = (((int16_t)Wire.read()<<8) | Wire.read());
}

//configure MPU6050
void MPU6050_Init(){
  delay(150);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_SMPLRT_DIV, 0x07);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_PWR_MGMT_1, 0x01);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_PWR_MGMT_2, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_CONFIG, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_GYRO_CONFIG, 0x00);//set +/-
250 degree/second full scale
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_ACCEL_CONFIG, 0x00);// set
+/- 2g full scale
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_FIFO_EN, 0x00);
```

```
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_INT_ENABLE, 0x01);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_SIGNAL_PATH_RESET, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_USER_CTRL, 0x00);
}
```

โค้ดการส่งข้อมูลไปยังฐานข้อมูล

```
#include <ArduinoJson.h>

#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

#define FIREBASE_HOST "falldetect-6ed35.firebaseio.com"

#define FIREBASE_AUTH "TnvqfNi3TGEY5ZUUM4Q8lC1aahYSw8kcFE6oHSkN"

#define WIFI_SSID "Ais wifi"

#define WIFI_PASSWORD "04022538"

void setup() {


Serial.begin(9600);

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

Serial.print("connecting");

while (WiFi.status() != WL_CONNECTED) {

Serial.print(".");
```

```
delay(500);

}

Serial.println();

Serial.print("connected: ");

Serial.println(WiFi.localIP());

Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}

void loop() {
// set value
Firebase.push("fall", 1);
}
```

โค้ดส่งการแจ้งเตือนผ่านทาง Line

```
void Line_Notify(String message) ;

#include <ESP8266WiFi.h>

// Config connect WiFi
#define WIFI_SSID "Ais wifi"
#define WIFI_PASSWORD "04022538"

// Line config
#define LINE_TOKEN "cKyqtpF0DEQi1OTcE5bveSaqMYvpRNJ6PS3UfPNw76D"
```

```
String message = "Fall detecting alert!!";

void setup() {

  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  // connect to wifi.
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("connecting");

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());
}
void loop() {

     Line_Notify(message);

  delay(1000);
}

 void Line_Notify(String message) {
  WiFiClientSecure client;
```

```
if (!client.connect("notify-api.line.me", 443)) {
  Serial.println("connection failed");
  return;
}

String req = "";
req += "POST /api/notify HTTP/1.1\r\n";
req += "Host: notify-api.line.me\r\n";
req += "Authorization: Bearer " + String(LINE_TOKEN) + "\r\n";
req += "Cache-Control: no-cache\r\n";
req += "User-Agent: ESP8266\r\n";
req += "Content-Type: application/x-www-form-urlencoded\r\n";
req += "Content-Length: " + String(String("message=" + message).length()) + "\r\n";
req += "\r\n";
req += "message=" + message;
// Serial.println(req);
client.print(req);

delay(20);

// Serial.println("-----------");
while(client.connected()) {
  String line = client.readStringUntil('\n');
  if (line == "\r") {
    break;
  }
  //Serial.println(line);
}
// Serial.println("-----------");
}
```

โค้ดส่งการแจ้งเตือนไปยังแอพพลิเคชั่น , LINE และ SMS

```
#include <Wire.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>
#define FIREBASE_HOST "falldetect-6ed35.firebaseio.com"
#define FIREBASE_AUTH "TnvqfNi3TGEY5ZUUM4Q8IC1aahYSw8kcFE6oHSkN"
#define WIFI_SSID "Ais wifi"
#define WIFI_PASSWORD "04022538"
#define LINE_TOKEN "cKyqtpF0DEQi1OTcE5bveSaqMYvpRNJ6PS3UfPNw76D"
String message = "Fall detecting alert!!";
int oldG =0;
int fall=0;
// MPU6050 Slave Device Address
const uint8_t MPU6050SlaveAddress = 0x68;
// Select SDA and SCL pins for I2C communication
const uint8_t scl = D6;
const uint8_t sda = D7;
// sensitivity scale factor respective to full scale setting provided in datasheet
const uint16_t AccelScaleFactor = 16384;
const uint16_t GyroScaleFactor = 131;
```

```
// MPU6050 few configuration register addresses
const uint8_t MPU6050_REGISTER_SMPLRT_DIV   =  0x19;
const uint8_t MPU6050_REGISTER_USER_CTRL    =  0x6A;
const uint8_t MPU6050_REGISTER_PWR_MGMT_1   =  0x6B;
const uint8_t MPU6050_REGISTER_PWR_MGMT_2   =  0x6C;
const uint8_t MPU6050_REGISTER_CONFIG       =  0x1A;
const uint8_t MPU6050_REGISTER_GYRO_CONFIG  =  0x1B;
const uint8_t MPU6050_REGISTER_ACCEL_CONFIG =  0x1C;
const uint8_t MPU6050_REGISTER_FIFO_EN      =  0x23;
const uint8_t MPU6050_REGISTER_INT_ENABLE   =  0x38;
const uint8_t MPU6050_REGISTER_ACCEL_XOUT_H =  0x3B;
const uint8_t MPU6050_REGISTER_SIGNAL_PATH_RESET  = 0x68;
int16_t AccelX, AccelY, AccelZ, Temperature, GyroX, GyroY, GyroZ;
void setup() {
  Serial.begin(9600);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("connecting");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");

delay(500);
```

```
}
Serial.println();
Serial.print("connected: ");
Serial.println(WiFi.localIP());
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  Wire.begin(sda, scl);
  MPU6050_Init();
}
void loop() {
  double Ax, Ay, Az, T, Gx, Gy, Gz;
  Read_RawValue(MPU6050SlaveAddress, MPU6050_REGISTER_ACCEL_XOUT_H);
  //divide each with their sensitivity scale factor
  Ax = (double)AccelX/AccelScaleFactor;
  Ay = (double)AccelY/AccelScaleFactor;
  Az = (double)AccelZ/AccelScaleFactor;
  T = (double)Temperature/340+36.53; //temperature formula
  Gx = (double)GyroX/GyroScaleFactor;
  Gy = (double)GyroY/GyroScaleFactor;
  Gz = (double)GyroZ/GyroScaleFactor;

 if(Gz < -150 && Gz > oldG)
```

```
    {
      Serial.println(Gz);
      fall =1;
    }
if(Gz != oldG)
    {
      oldG = Gz;
    }
 if(Ax< -0.5 && Ay<0.5 && fall == 1) {
      Serial.print("-");
      Firebase.push("fall", 1);
      Line_Notify(message);
      Serial.println("AT+CMGF=1");    //ตั้งค่าโมดูลให้เป็น text mode
      delay(1000);
      Serial.println("AT+CMGS=\"");  // คำสั่งในการส่ง sms โดยจะมีpatameterดังนี้
      //AT+CMGS=<number><CR><message><CTRL-Z>
      Serial.print("0982691136");  //เบอร์ที่จะทำการส่ง
      Serial.println(char(13));  //CR เป็น parameterที่ใช้สำหรับ AT Commands ของโมดูล
      delay(2000);
      Serial.print ("Fall detecting"); //
      Serial.println(char(26)); // CTRL-Z เป็น parameterที่ใช้สำหรับ AT Commands ของ
โมดูล

 }
  delay(100);
```

```
}
void I2C_Write(uint8_t deviceAddress, uint8_t regAddress, uint8_t data){
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.write(data);
  Wire.endTransmission();
}
// read all 14 register
void Read_RawValue(uint8_t deviceAddress, uint8_t regAddress){
  Wire.beginTransmission(deviceAddress);
  Wire.write(regAddress);
  Wire.endTransmission();
  Wire.requestFrom(deviceAddress, (uint8_t)14);
  AccelX = (((int16_t)Wire.read()<<8) | Wire.read());
  AccelY = (((int16_t)Wire.read()<<8) | Wire.read());
  AccelZ = (((int16_t)Wire.read()<<8) | Wire.read());
  Temperature = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroX = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroY = (((int16_t)Wire.read()<<8) | Wire.read());
  GyroZ = (((int16_t)Wire.read()<<8) | Wire.read());
 }
```

```
//configure MPU6050
void MPU6050_Init(){
  delay(150);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_SMPLRT_DIV, 0x07);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_PWR_MGMT_1, 0x01);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_PWR_MGMT_2, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_CONFIG, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_GYRO_CONFIG, 0x01);//set +/-
500 degree/second
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_ACCEL_CONFIG, 0x00);// set
+/- 2g full scale
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_FIFO_EN, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_INT_ENABLE, 0x01);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_SIGNAL_PATH_RESET, 0x00);
  I2C_Write(MPU6050SlaveAddress, MPU6050_REGISTER_USER_CTRL, 0x00);
}
void Line_Notify(String message) {
  WiFiClientSecure client;

  if (!client.connect("notify-api.line.me", 443)) {
    Serial.println("connection failed");
    return;
  }
```

```
String req = "";
req += "POST /api/notify HTTP/1.1\r\n";
req += "Host: notify-api.line.me\r\n";
req += "Authorization: Bearer " + String(LINE_TOKEN) + "\r\n";
req += "Cache-Control: no-cache\r\n";
req += "User-Agent: ESP8266\r\n";
req += "Content-Type: application/x-www-form-urlencoded\r\n";
req += "Content-Length: " + String(String("message=" + message).length()) + "\r\n";
req += "\r\n";
req += "message=" + message;
// Serial.println(req);
client.print(req);

delay(20);

// Serial.println("-----------");
while(client.connected()) {
  String line = client.readStringUntil('\n');
  if (line == "\r") {
    break;
  }
  //Serial.println(line);
 }
 // Serial.println("-----------");
}
```

ภาคผนวก ข

MPU6050 DATASHEET (ส่วนของการรับค่า ACC และ GY)

MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2

CONTENTS

1 Revision History

Revision Date	Revision	Description
11/29/2010	1.0	Initial Release
04/20/2011	1.1	Updated register map and descriptions to reflect enhanced register functionality.
05/19/2011	2.0	Updates for Rev C silicon: Edits for readability (section 2.1) Edits for changes in functionality (section 3, 4.4, 4.6, 4.7, 4.8, 4.21, 4.22, 4.23, 4.37)
10/07/2011	3.0	Updates for Rev D silicon: Updated accelerometer sensitivity specifications (sections 4.6, 4.8, 4.10, 4.23)
10/24/2011	3.1	Edits for clarity
11/14/2011	3.2	Updated reset value for register 107 (section 3) Updated register 27 with gyro self-test bits (section 4.4) Provided gyro self-test instructions and register bits (section 4.4) Provided accel self-test instructions (section 4.5)
3/9/2012	4.0	Updated register map to include Self-Test registers (section 3) Added description of Self-Test registers (section 4.1) Revised temperature register section (section 4.19) Corrections in registers 107 and 108 (section 4.30)
2/11/2013	4.1	Added reset clarification for SPI interface (section 4.3)
8/19/2013	4.2	Updated sections 6, 7, 8, 10

2 Purpose and Scope

This document provides preliminary information regarding the register map and descriptions for the Motion Processing Units™ MPU-6000™ and MPU-6050™, collectively called the MPU-60X0™ or MPU™.

The MPU devices provide the world's first integrated 6-axis motion processor solution that eliminates the package-level gyroscope and accelerometer cross-axis misalignment associated with discrete solutions. The devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor™ (DMP™) capable of processing complex 9-axis sensor fusion algorithms using the field-proven and proprietary MotionFusion™ engine.

The MPU-6000 and MPU-6050's integrated 9-axis MotionFusion algorithms access external magnetometers or other sensors through an auxiliary master I^2C bus, allowing the devices to gather a full set of sensor data without intervention from the system processor. The devices are offered in the same 4x4x0.9 mm QFN footprint and pinout as the current MPU-3000™ family of integrated 3-axis gyroscopes, providing a simple upgrade path and facilitating placement on already space constrained circuit boards.

For precision tracking of both fast and slow motions, the MPU-60X0 features a user-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps). The parts also have a user-programmable accelerometer full-scale range of ±2*g*, ±4*g*, ±8*g*, and ±16*g*.

The MPU-6000 family is comprised of two parts, the MPU-6000 and MPU-6050. These parts are identical to each other with two exceptions. The MPU-6050 supports I^2C communications at up to 400kHz and has a VLOGIC pin that defines its interface voltage levels; the MPU-6000 supports SPI at up to 20MHz in addition to I^2C, and has a single supply pin, VDD, which is both the device's logic reference supply and the analog supply for the part.

For more detailed information for the MPU-60X0 devices, please refer to the "MPU-6000 and MPU-6050 Product Specification".

3 Register Map

The register map for the MPU-60X0 is listed below.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0D	13	SELF_TEST_X	R/W	XA_TEST[4-2]			XG_TEST[4-0]				
0E	14	SELF_TEST_Y	R/W	YA_TEST[4-2]			YG_TEST[4-0]				
0F	15	SELF_TEST_Z	R/W	ZA_TEST[4-2]			ZG_TEST[4-0]				
10	16	SELF_TEST_A	R/W	RESERVED		XA_TEST[1-0]		YA_TEST[1-0]		ZA_TEST[1-0]	
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	-	-	-	FS_SEL [1:0]		-	-	-
1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]				
23	35	FIFO_EN	R/W	TEMP _FIFO_EN	XG _FIFO_EN	YG _FIFO_EN	ZG _FIFO_EN	ACCEL _FIFO_EN	SLV2 _FIFO_EN	SLV1 _FIFO_EN	SLV0 _FIFO_EN
24	36	I2C_MST_CTRL	R/W	MULT _MST_EN	WAIT _FOR_ES	SLV_3 _FIFO_EN	I2C_MST _P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	R/W	I2C_SLV0 _RW	I2C_SLV0_ADDR[6:0]						
26	38	I2C_SLV0_REG	R/W	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	R/W	I2C_SLV0 _EN	I2C_SLV0 _BYTE_SW	I2C_SLV0 _REG_DIS	I2C_SLV0 _GRP	I2C_SLV0_LEN[3:0]			
28	40	I2C_SLV1_ADDR	R/W	I2C_SLV1 _RW	I2C_SLV1_ADDR[6:0]						
29	41	I2C_SLV1_REG	R/W	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	R/W	I2C_SLV1 _EN	I2C_SLV1 _BYTE_SW	I2C_SLV1 _REG_DIS	I2C_SLV1 _GRP	I2C_SLV1_LEN[3:0]			
2B	43	I2C_SLV2_ADDR	R/W	I2C_SLV2 _RW	I2C_SLV2_ADDR[6:0]						
2C	44	I2C_SLV2_REG	R/W	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	R/W	I2C_SLV2 _EN	I2C_SLV2 _BYTE_SW	I2C_SLV2 _REG_DIS	I2C_SLV2 _GRP	I2C_SLV2_LEN[3:0]			
2E	46	I2C_SLV3_ADDR	R/W	I2C_SLV3 _RW	I2C_SLV3_ADDR[6:0]						
2F	47	I2C_SLV3_REG	R/W	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	R/W	I2C_SLV3 _EN	I2C_SLV3 _BYTE_SW	I2C_SLV3 _REG_DIS	I2C_SLV3 _GRP	I2C_SLV3_LEN[3:0]			
31	49	I2C_SLV4_ADDR	R/W	I2C_SLV4 _RW	I2C_SLV4_ADDR[6:0]						
32	50	I2C_SLV4_REG	R/W	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	R/W	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	R/W	I2C_SLV4 _EN	I2C_SLV4 _INT_EN	I2C_SLV4 _REG_DIS	I2C_MST_DLY[4:0]				
35	53	I2C_SLV4_DI	R	I2C_SLV4_DI[7:0]							
36	54	I2C_MST_STATUS	R	PASS_ THROUGH	I2C_SLV4 _DONE	I2C_LOST _ARB	I2C_SLV4 _NACK	I2C_SLV3 _NACK	I2C_SLV2 _NACK	I2C_SLV1 _NACK	I2C_SLV0 _NACK
37	55	INT_PIN_CFG	R/W	INT_LEVEL	INT_OPEN	LATCH _INT_EN	INT_RD _CLEAR	FSYNC_ INT_LEVEL	FSYNC _INT_EN	I2C _BYPASS _EN	-
38	56	INT_ENABLE	R/W	-	-	-	FIFO _OFLOW _EN	I2C_MST _INT_EN	-	-	DATA _RDY_EN
3A	58	INT_STATUS	R	-	-	-	FIFO _OFLOW _INT	I2C_MST _INT	-	-	DATA _RDY_INT

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							
49	73	EXT_SENS_DATA_00	R	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01	R	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02	R	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03	R	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04	R	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05	R	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06	R	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07	R	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08	R	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09	R	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10	R	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11	R	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12	R	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13	R	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14	R	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15	R	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16	R	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17	R	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18	R	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19	R	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20	R	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21	R	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22	R	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23	R	EXT_SENS_DATA_23[7:0]							
63	99	I2C_SLV0_DO	R/W	I2C_SLV0_DO[7:0]							
64	100	I2C_SLV1_DO	R/W	I2C_SLV1_DO[7:0]							
65	101	I2C_SLV2_DO	R/W	I2C_SLV2_DO[7:0]							
66	102	I2C_SLV3_DO	R/W	I2C_SLV3_DO[7:0]							

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
67	103	I2C_MST_DELAY_CTRL	R/W	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	R/W	-	-	-	-	-	GYRO_RESET	ACCEL_RESET	TEMP_RESET
6A	106	USER_CTRL	R/W	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RESET	I2C_MST_RESET	SIG_COND_RESET
6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	LP_WAKE_CTRL[1:0]		STBY_XA	STBY_YA	STBY_ZA	STBY_XG	STBY_YG	STBY_ZG
72	114	FIFO_COUNTH	R/W	FIFO_COUNT[15:8]							
73	115	FIFO_COUNTL	R/W	FIFO_COUNT[7:0]							
74	116	FIFO_R_W	R/W	FIFO_DATA[7:0]							
75	117	WHO_AM_I	R	-	WHO_AM_I[6:1]						-

Note: Register Names ending in _H and _L contain the high and low bytes, respectively, of an internal register value.

In the detailed register tables that follow, register names are in capital letters, while register values are in capital letters and italicized. For example, the ACCEL_XOUT_H register (Register 59) contains the 8 most significant bits, *ACCEL_XOUT*[15:8], of the 16-bit X-Axis accelerometer measurement, *ACCEL_XOUT*.

The reset value is 0x00 for all registers other than the registers below.

- Register 107: 0x40.
- Register 117: 0x68.

4 Register Descriptions

This section describes the function and contents of each register within the MPU-60X0.

Note: The device will come up in sleep mode upon power-up.

4.1 Registers 13 to 16 – Self Test Registers SELF_TEST_X, SELF_TEST_Y, SELF_TEST_Z, and SELF_TEST_A

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0D	13	XA_TEST[4-2]			XG_TEST[4-0]				
0E	14	YA_TEST[4-2]			YG_TEST[4-0]				
0F	15	ZA_TEST[4-2]			ZG_TEST[4-0]				
10	16	RESERVED		XA_TEST[1-0]		YA_TEST[1-0]		ZA_TEST[1-0]	

Description:

These registers are used for gyroscope and accelerometer self-tests that permit the user to test the mechanical and electrical portions of the gyroscope and the accelerometer. The following sections describe the self-test process.

1. Gyroscope Hardware Self-Test: Relative Method

Gyroscope self-test permits users to test the mechanical and electrical portions of the gyroscope. Code for operating self-test is included within the MotionApps™ software provided by InvenSense. Please refer to the next section (*Obtaining the Gyroscope Factory Trim (FT) Value*) if not using MotionApps software.

When self-test is activated, the on-board electronics will actuate the appropriate sensor. This actuation will move the sensor's proof masses over a distance equivalent to a pre-defined Coriolis force. This proof mass displacement results in a change in the sensor output, which is reflected in the output signal. The output signal is used to observe the self-test response.

The self-test response (STR) is defined as follows:

$$SelfTest\ Response = Gyroscope\ Output\ with\ Self\text{-}Test\ Enabled - Gyroscope\ Output\ with\ Self\text{-}Test\ Disabled$$

This self test-response is used to determine whether the part has passed or failed self-test by finding the change from factory trim of the self-test response as follows:

$$Change\ from\ Factory\ Trim\ of\ the\ Self\text{-}Test\ Response(\%) = \frac{(STR - FT)}{FT}$$

where,

$FT = Factory\ trim\ value\ of\ selftest\ response, available\ via\ MotionApps\ software$

This change from factory trim of the self-test response must be within the limits provided in the MPU-6000/MPU-6050 Product Specification document for the part to pass self-test. Otherwise, the part is deemed to have failed self-test.

Obtaining the Gyroscope Factory Trim (FT) Value

If InvenSense MotionApps software is not used, the procedure detailed below should be followed to obtain the Factory trim value of the self test response (FT) mentioned above. For the specific registers mentioned below, please refer to registers 13-15.

The Factory trim value of the self test response (FT) is calculated as shown below. FT[Xg], FT[Yg], and FT[Zg] refer to the factory trim (FT) values for the gyroscope X, Y, and Z axes, respectively. XG_TEST is the decimal version of XG_TEST[4-0], YG_TEST is the decimal version of YG_TEST[4-0], and ZG_TEST is the decimal version of ZG_TEST[4-0].

When performing self test for the gyroscope, the full-scale range should be set to ±250dps.

$$\begin{cases} \text{FT [Xg]} = 25 * 131 * 1.046^{(XG\_TEST-1)} & \text{if XG\_TEST} \neq 0 \\ \text{FT [Xg]} = 0 & \text{if XG\_TEST = 0} \end{cases}$$

$$\begin{cases} \text{FT [Yg]} = -25 * 131 * 1.046^{(YG\_TEST-1)} & \text{if YG\_TEST} \neq 0 \\ \text{FT [Yg]} = 0 & \text{if YG\_TEST = 0} \end{cases}$$

$$\begin{cases} \text{FT [Zg]} = 25 * 131 * 1.046^{(ZG\_TEST-1)} & \text{if ZG\_TEST} \neq 0 \\ \text{FT [Zg]} = 0 & \text{if ZG\_TEST = 0} \end{cases}$$

2. Accelerometer Hardware Self-Test: Relative Method

Accelerometer self-test permits users to test the mechanical and electrical portions of the accelerometer. Code for operating self-test is included within the MotionApps software provided by InvenSense. Please refer to the next section (titled Obtaining the Accelerometer Factory Trim (FT) Value) if not using MotionApps software.

When self-test is activated, the on-board electronics will actuate the appropriate sensor. This actuation simulates an external force. The actuated sensor, in turn, will produce a corresponding output signal. The output signal is used to observe the self-test response.

The self-test response (STR) is defined as follows:

$$SelfTest\ Response = Accelerometer\ Output\ with\ Self\text{-}Test\ Enabled - Accelerometer\ Output\ with\ Self\text{-}Test\ Disabled$$

This self test-response is used to determine whether the part has passed or failed self-test by finding the change from factory trim of the self-test response as follows:

$$Change\ from\ Factory\ Trim\ of\ the\ Self\text{-}Test\ Response(\%) = \frac{(STR - FT)}{FT}$$

where,

$FT = Factory\ trim\ value\ of\ self test\ response, available\ via\ MotionApps\ software$

This change from factory trim of the self-test response must be within the limits provided in the MPU-6000/MPU-6050 Product Specification document for the part to pass self-test. Otherwise, the part is deemed to have failed self-test.

Obtaining the Accelerometer Factory Trim (FT) Value

If InvenSense MotionApps software is not used, the procedure detailed below should be followed to obtain the Factory trim value of the self test response (FT) mentioned above. For the specific registers mentioned below, please refer to registers 13-16.

The Factory trim value of the self test response (FT) is calculated as shown below. FT[Xa], FT[Ya], and FT[Za] refer to the factory trim (FT) values for the accelerometer X, Y, and Z axes, respectively. In the equations below, the factory trim values for the accel should be in decimal format, and they are determined by concatenating the upper accelerometer self test bits (bits 4-2) with the lower accelerometer self test bits (bits 1-0).

When performing accelerometer self test, the full-scale range should be set to ±8*g*.

$$\begin{cases} FT[Xa] = 4096 * 0.34 * \frac{0.92}{0.34}^{\left(\frac{XA\_TEST-1}{2^5-2}\right)} & \text{if XA\_TEST} \neq 0. \\ FT[Xa] = 0 & \text{if XA\_TEST} = 0. \end{cases}$$

$$\begin{cases} FT[Ya] = 4096 * 0.34 * \frac{0.92}{0.34}^{\left(\frac{YA\_TEST-1}{2^5-2}\right)} & \text{if YA\_TEST} \neq 0. \\ FT[Ya] = 0 & \text{if YA\_TEST} = 0. \end{cases}$$

$$\begin{cases} FT[Za] = 4096 * 0.34 * \frac{0.92}{0.34}^{\left(\frac{ZA\_TEST-1}{2^5-2}\right)} & \text{if ZA\_TEST} \neq 0. \\ FT[Za] = 0 & \text{if ZA\_TEST} = 0. \end{cases}$$

Parameters:

XA_TEST	5-bit unsigned value. FT[Xa] is determined by using this value as explained above.
XG_TEST	5-bit unsigned value. FT[Xg] is determined by using this value as explained above.
YA_TEST	5-bit unsigned value. FT[Ya] is determined by using this value as explained above.
YG_TEST	5-bit unsigned value. FT[Yg] is determined by using this value as explained above.
ZA_TEST	5-bit unsigned value. FT[Za] is determined by using this value as explained above.
ZG_TEST	5-bit unsigned value. FT[Zg] is determined by using this value as explained above.

4.2 Register 25 – Sample Rate Divider SMPRT_DIV

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25	SMPLRT_DIV[7:0]							

Description:

This register specifies the divider from the gyroscope output rate used to generate the Sample Rate for the MPU-60X0.

The sensor register output, FIFO output, and DMP sampling are all based on the Sample Rate.

The Sample Rate is generated by dividing the gyroscope output rate by *SMPLRT_DIV*:

Sample Rate = Gyroscope Output Rate / (1 + *SMPLRT_DIV*)

where Gyroscope Output Rate = 8kHz when the DLPF is disabled (*DLPF_CFG* = 0 or 7), and 1kHz when the DLPF is enabled (see Register 26).

Note: The accelerometer output rate is 1kHz. This means that for a Sample Rate greater than 1kHz, the same accelerometer sample may be output to the FIFO, DMP, and sensor registers more than once.

For a diagram of the gyroscope and accelerometer signal paths, see Section 8 of the MPU-6000/MPU-6050 Product Specification document.

Parameters:

SMPLRT_DIV	8-bit unsigned value. The Sample Rate is determined by dividing the gyroscope output rate by this value.

4.3 Register 26 – Configuration
CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Description:

This register configures the external Frame Synchronization (FSYNC) pin sampling and the Digital Low Pass Filter (DLPF) setting for both the gyroscopes and accelerometers.

An external signal connected to the FSYNC pin can be sampled by configuring *EXT_SYNC_SET*.

Signal changes to the FSYNC pin are latched so that short strobes may be captured. The latched FSYNC signal will be sampled at the Sampling Rate, as defined in register 25. After sampling, the latch will reset to the current FSYNC signal state.

The sampled value will be reported in place of the least significant bit in a sensor data register determined by the value of *EXT_SYNC_SET* according to the following table.

EXT_SYNC_SET	FSYNC Bit Location
0	Input disabled
1	TEMP_OUT_L[0]
2	GYRO_XOUT_L[0]
3	GYRO_YOUT_L[0]
4	GYRO_ZOUT_L[0]
5	ACCEL_XOUT_L[0]
6	ACCEL_YOUT_L[0]
7	ACCEL_ZOUT_L[0]

The DLPF is configured by *DLPF_CFG*. The accelerometer and gyroscope are filtered according to the value of *DLPF_CFG* as shown in the table below.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Bit 7 and bit 6 are reserved.

Parameters:

EXT_SYNC_SET 3-bit unsigned value. Configures the FSYNC pin sampling.
DLPF_CFG 3-bit unsigned value. Configures the DLPF setting.

4.4 Register 27 – Gyroscope Configuration
GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Description:

This register is used to trigger gyroscope self-test and configure the gyroscopes' full scale range.

Gyroscope self-test permits users to test the mechanical and electrical portions of the gyroscope. The self-test for each gyroscope axis can be activated by controlling the *XG_ST*, *YG_ST*, and *ZG_ST* bits of this register. Self-test for each axis may be performed independently or all at the same time.

When self-test is activated, the on-board electronics will actuate the appropriate sensor. This actuation will move the sensor's proof masses over a distance equivalent to a pre-defined Coriolis force. This proof mass displacement results in a change in the sensor output, which is reflected in the output signal. The output signal is used to observe the self-test response.

The self-test response is defined as follows:

Self-test response = Sensor output with self-test enabled – Sensor output without self-test enabled

The self-test limits for each gyroscope axis is provided in the electrical characteristics tables of the MPU-6000/MPU-6050 Product Specification document. When the value of the self-test response is within the min/max limits of the product specification, the part has passed self test. When the self-test response exceeds the min/max values specified in the document, the part is deemed to have failed self-test.

FS_SEL selects the full scale range of the gyroscope outputs according to the following table.

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

Bits 2 through 0 are reserved.

Parameters:

XG_ST	Setting this bit causes the X axis gyroscope to perform self test.
YG_ST	Setting this bit causes the Y axis gyroscope to perform self test.
ZG_ST	Setting this bit causes the Z axis gyroscope to perform self test.
FS_SEL	2-bit unsigned value. Selects the full scale range of gyroscopes.

4.5 Register 28 – Accelerometer Configuration ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Description:

This register is used to trigger accelerometer self test and configure the accelerometer full scale range. This register also configures the Digital High Pass Filter (DHPF).

Accelerometer self-test permits users to test the mechanical and electrical portions of the accelerometer. The self-test for each accelerometer axis can be activated by controlling the *XA_ST*, *YA_ST*, and *ZA_ST* bits of this register. Self-test for each axis may be performed independently or all at the same time.

When self-test is activated, the on-board electronics will actuate the appropriate sensor. This actuation simulates an external force. The actuated sensor, in turn, will produce a corresponding output signal. The output signal is used to observe the self-test response.

The self-test response is defined as follows:

Self-test response = Sensor output with self-test enabled – Sensor output without self-test enabled

The self-test limits for each accelerometer axis is provided in the electrical characteristics tables of the MPU-6000/MPU-6050 Product Specification document. When the value of the self-test response is within the min/max limits of the product specification, the part has passed self test. When the self-test response exceeds the min/max values specified in the document, the part is deemed to have failed self-test.

AFS_SEL selects the full scale range of the accelerometer outputs according to the following table.

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

Parameters:

XA_ST	When set to 1, the X- Axis accelerometer performs self test.
YA_ST	When set to 1, the Y- Axis accelerometer performs self test.
ZA_ST	When set to 1, the Z- Axis accelerometer performs self test.
AFS_SEL	2-bit unsigned value. Selects the full scale range of accelerometers.

4.6 Register 35 – FIFO Enable
FIFO_EN

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
23	35	TEMP_ FIFO_EN	XG_ FIFO_EN	YG_ FIFO_EN	ZG_ FIFO_EN	ACCEL _FIFO_EN	SLV2 _FIFO_EN	SLV1 _FIFO_EN	SLV0 _FIFO_EN

Description:

This register determines which sensor measurements are loaded into the FIFO buffer.

Data stored inside the sensor data registers (Registers 59 to 96) will be loaded into the FIFO buffer if a sensor's respective FIFO_EN bit is set to 1 in this register.

When a sensor's FIFO_EN bit is enabled in this register, data from the sensor data registers will be loaded into the FIFO buffer. The sensors are sampled at the Sample Rate as defined in Register 25. For further information regarding sensor data registers, please refer to Registers 59 to 96

When an external Slave's corresponding FIFO_EN bit (*SLVx_FIFO_EN*, where x=0, 1, or 2) is set to 1, the data stored in its corresponding data registers (EXT_SENS_DATA registers, Registers 73 to 96) will be written into the FIFO buffer at the Sample Rate. EXT_SENS_DATA register association with I^2C Slaves is determined by the I2C_SLVx_CTRL registers (where x=0, 1, or 2; Registers 39, 42, and 45). For information regarding EXT_SENS_DATA registers, please refer to Registers 73 to 96.

Note that the corresponding FIFO_EN bit (*SLV3_FIFO_EN)* is found in I2C_MST_CTRL (Register 36). Also note that Slave 4 behaves in a different manner compared to Slaves 0-3. Please refer to Registers 49 to 53 for further information regarding Slave 4 usage.

Parameters:

TEMP_FIFO_EN	When set to 1, this bit enables TEMP_OUT_H and TEMP_OUT_L (Registers 65 and 66) to be written into the FIFO buffer.
XG_ FIFO_EN	When set to 1, this bit enables GYRO_XOUT_H and GYRO_XOUT_L (Registers 67 and 68) to be written into the FIFO buffer.
YG_ FIFO_EN	When set to 1, this bit enables GYRO_YOUT_H and GYRO_YOUT_L (Registers 69 and 70) to be written into the FIFO buffer.
ZG_ FIFO_EN	When set to 1, this bit enables GYRO_ZOUT_H and GYRO_ZOUT_L (Registers 71 and 72) to be written into the FIFO buffer.
ACCEL_ FIFO_EN	When set to 1, this bit enables ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L (Registers 59 to 64) to be written into the FIFO buffer.

SLV2_ FIFO_EN	When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 2 to be written into the FIFO buffer.
SLV1_ FIFO_EN	When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 1 to be written into the FIFO buffer.
SLV0_ FIFO_EN	When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 0 to be written into the FIFO buffer.

Note: For further information regarding the association of EXT_SENS_DATA registers to particular slave devices, please refer to Registers 73 to 96.

4.7 Register 36 – I^2C Master Control I2C_MST_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
24	36	MULT_MST_EN	WAIT_FOR_ES	SLV_3_FIFO_EN	I2C_MST_P_NSR	I2C_MST_CLK[3:0]			

Description:

This register configures the auxiliary I^2C bus for single-master or multi-master control. In addition, the register is used to delay the Data Ready interrupt, and also enables the writing of Slave 3 data into the FIFO buffer. The register also configures the auxiliary I^2C Master's transition from one slave read to the next, as well as the MPU-60X0's 8MHz internal clock.

Multi-master capability allows multiple I^2C masters to operate on the same bus. In circuits where multi-master capability is required, set *MULT_MST_EN* to 1. This will increase current drawn by approximately 30μA.

In circuits where multi-master capability is required, the state of the I^2C bus must always be monitored by each separate I^2C Master. Before an I^2C Master can assume arbitration of the bus, it must first confirm that no other I^2C Master has arbitration of the bus. When *MULT_MST_EN* is set to 1, the MPU-60X0's bus arbitration detection logic is turned on, enabling it to detect when the bus is available.

When the *WAIT_FOR_ES* bit is set to 1, the Data Ready interrupt will be delayed until External Sensor data from the Slave Devices are loaded into the EXT_SENS_DATA registers. This is used to ensure that both the internal sensor data (i.e. from gyro and accel) and external sensor data have been loaded to their respective data registers (i.e. the data is synced) when the Data Ready interrupt is triggered.

When the Slave 3 FIFO enable bit (*SLV_3_FIFO_EN*) is set to 1, Slave 3 sensor measurement data will be loaded into the FIFO buffer each time. EXT_SENS_DATA register association with I^2C Slaves is determined by I2C_SLV3_CTRL (Register 48).

For further information regarding EXT_SENS_DATA registers, please refer to Registers 73 to 96.

The corresponding FIFO_EN bits for Slave 0, Slave 1, and Slave 2 can be found in Register 35.

The *I2C_MST_P_NSR* bit configures the I^2C Master's transition from one slave read to the next slave read. If the bit equals 0, there will be a restart between reads. If the bit equals 1, there will be a stop followed by a start of the following read. When a write transaction follows a read transaction, the stop followed by a start of the successive write will be always used.

I2C_MST_CLK is a 4 bit unsigned value which configures a divider on the MPU-60X0 internal 8MHz clock. It sets the I^2C master clock speed according to the following table:

I2C_MST_CLK	I^2C Master Clock Speed	8MHz Clock Divider
0	348 kHz	23
1	333 kHz	24
2	320 kHz	25
3	308 kHz	26
4	296 kHz	27
5	286 kHz	28
6	276 kHz	29
7	267 kHz	30
8	258 kHz	31
9	500 kHz	16
10	471 kHz	17
11	444 kHz	18
12	421 kHz	19
13	400 kHz	20
14	381 kHz	21
15	364 kHz	22

Parameters:

MUL_MST_EN — When set to 1, this bit enables multi-master capability.

WAIT_FOR_ES — When set to 1, this bit delays the Data Ready interrupt until External Sensor data from the Slave devices have been loaded into the EXT_SENS_DATA registers.

SLV3_FIFO_EN — When set to 1, this bit enables EXT_SENS_DATA registers associated with Slave 3 to be written into the FIFO. The corresponding bits for Slaves 0-2 can be found in Register 35.

I2C_MST_P_NSR — Controls the I^2C Master's transition from one slave read to the next slave read.
When this bit equals 0, there is a restart between reads.
When this bit equals 1, there is a stop and start marking the beginning of the next read.

When a write follows a read, a stop and start is always enforced.

I2C_MST_CLK — 4 bit unsigned value. Configures the I^2C master clock speed divider.

Note: For further information regarding the association of EXT_SENS_DATA registers to particular slave devices, please refer to Registers 73 to 96.

4.8 Registers 37 to 39 – I²C Slave 0 Control I2C_SLV0_ADDR, I2C_SLV0_REG, and I2C_SLV0_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
25	37	I2C_SLV0_RW	I2C_SLV0_ADDR[6:0]						
26	38	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_EN	I2C_SLV0_BYTE_SW	I2C_SLV0_REG_DIS	I2C_SLV0_GRP	I2C_SLV0_LEN[3:0]			

Description:

These registers configure the data transfer sequence for Slave 0. Slaves 1, 2, and 3 also behave in a similar manner to Slave 0. However, Slave 4's characteristics differ greatly from those of Slaves 0-3. For further information regarding Slave 4, please refer to registers 49 to 53.

I²C slave data transactions between the MPU-60X0 and Slave 0 are set as either read or write operations by the *I2C_SLV0_RW* bit. When this bit is 1, the transfer is a read operation. When the bit is 0, the transfer is a write operation.

I2C_SLV0_ADDR is used to specify the I²C slave address of Slave 0.

Data transfer starts at an internal register within Slave 0. This address of this register is specified by *I2C_SLV0_REG*.

The number of bytes transferred is specified by *I2C_SLV0_LEN*. When more than 1 byte is transferred (*I2C_SLV0_LEN* > 1), data is read from (written to) sequential addresses starting from *I2C_SLV0_REG*.

In read mode, the result of the read is placed in the lowest available EXT_SENS_DATA register. For further information regarding the allocation of read results, please refer to the EXT_SENS_DATA register description (Registers 73 – 96).

In write mode, the contents of *I2C_SLV0_DO* (Register 99) will be written to the slave device.

I2C_SLV0_EN enables Slave 0 for I²C data transaction. A data transaction is performed only if more than zero bytes are to be transferred (*I2C_SLV0_LEN* > 0) between an enabled slave device (*I2C_SLV0_EN* = 1).

I2C_SLV0_BYTE_SW configures byte swapping of word pairs. When byte swapping is enabled, the high and low bytes of a word pair are swapped. Please refer to *I2C_SLV0_GRP* for the pairing convention of the word pairs. When this bit is cleared to 0, bytes transferred to and from Slave 0 will be written to EXT_SENS_DATA registers in the order they were transferred.

When *I2C_SLV0_REG_DIS* is set to 1, the transaction will read or write data only. When cleared to 0, the transaction will write a register address prior to reading or writing data. This bit should equal 0 when specifying the register address within the Slave device to/from which the ensuing data transaction will take place.

I2C_SLV0_GRP specifies the grouping order of word pairs received from registers. When cleared to 0, bytes from register addresses 0 and 1, 2 and 3, etc (even, then odd register addresses) are paired to form a word. When set to 1, bytes from register addresses are paired 1 and 2, 3 and 4, etc. (odd, then even register addresses) are paired to form a word.

I^2C data transactions are performed at the Sample Rate, as defined in Register 25. The user is responsible for ensuring that I^2C data transactions to and from each enabled Slave can be completed within a single period of the Sample Rate.

The I^2C slave access rate can be reduced relative to the Sample Rate. This reduced access rate is determined by *I2C_MST_DLY* (Register 52). Whether a slave's access rate is reduced relative to the Sample Rate is determined by I2C_MST_DELAY_CTRL (Register 103).

The processing order for the slaves is fixed. The sequence followed for processing the slaves is Slave 0, Slave 1, Slave 2, Slave 3 and Slave 4. If a particular Slave is disabled it will be skipped.

Each slave can either be accessed at the sample rate or at a reduced sample rate. In a case where some slaves are accessed at the Sample Rate and some slaves are accessed at the reduced rate, the sequence of accessing the slaves (Slave 0 to Slave 4) is still followed. However, the reduced rate slaves will be skipped if their access rate dictates that they should not be accessed during that particular cycle. For further information regarding the reduced access rate, please refer to Register 52. Whether a slave is accessed at the Sample Rate or at the reduced rate is determined by the Delay Enable bits in Register 103.

Parameters:

I2C_SLV0_RW	When set to 1, this bit configures the data transfer as a read operation. When cleared to 0, this bit configures the data transfer as a write operation.
I2C_SLV0_ADDR	7-bit I^2C address of Slave 0.
I2C_SLV0_REG	8-bit address of the Slave 0 register to/from which data transfer starts.
I2C_SLV0_EN	When set to 1, this bit enables Slave 0 for data transfer operations. When cleared to 0, this bit disables Slave 0 from data transfer operations.
I2C_SLV0_BYTE_SW	When set to 1, this bit enables byte swapping. When byte swapping is enabled, the high and low bytes of a word pair are swapped. Please refer to *I2C_SLV0_GRP* for the pairing convention of the word pairs. When cleared to 0, bytes transferred to and from Slave 0 will be written to EXT_SENS_DATA registers in the order they were transferred.
I2C_SLV0_REG_DIS	When set to 1, the transaction will read or write data only. When cleared to 0, the transaction will write a register address prior to reading or writing data.
I2C_SLV0_GRP	1-bit value specifying the grouping order of word pairs received from registers. When cleared to 0, bytes from register addresses 0 and 1, 2 and 3, etc (even, then odd register addresses) are paired to form a word. When set to 1, bytes from register addresses are paired 1 and 2, 3 and 4, etc. (odd, then even register addresses) are paired to form a word.
I2C_SLV0_LEN	4-bit unsigned value. Specifies the number of bytes transferred to and from Slave 0. Clearing this bit to 0 is equivalent to disabling the register by writing 0 to *I2C_SLV0_EN*.

Byte Swapping Example

The following example demonstrates byte swapping for *I2C_SLV0_BYTE_SW* = 1, *I2C_SLV0_GRP* = 0, *I2C_SLV0_REG* = 0x01, and *I2C_SLV0_LEN* = 0x4:

1. The first byte, read from Slave 0 register 0x01, will be stored at EXT_SENS_DATA_00. Because *I2C_SLV0_GRP* = 0, bytes from even, then odd register addresses will be paired together as word pairs. Since the read operation started from an odd register address instead of an even address, only one byte is read.
2. The second and third bytes will be swapped, since *I2C_SLV0_BYTE_SW* = 1 and *I2C_SLV0_REG[0]* = 1. The data read from 0x02 will be stored at EXT_SENS_DATA_02, and the data read from 0x03 will be stored at EXT_SENS_DATA_01.
3. The last byte, read from address 0x04, will be stored at EXT_SENS_DATA_03. Because there is only one byte remaining in the read operation, byte swapping will not occur.

Slave Access Example

Slave 0 is accessed at the Sample Rate, while Slave 1 is accessed at half the Sample Rate. The other slaves are disabled. In the first cycle, both Slave 0 and Slave 1 will be accessed. However, in the second cycle, only Slave 0 will be accessed. In the third cycle, both Slave 0 and Slave 1 will be accessed. In the fourth cycle, only Slave 0 will be accessed. This pattern continues.

4.9 Registers 40 to 42 – I²C Slave 1 Control I2C_SLV1_ADDR, I2C_SLV1_REG, and I2C_SLV1_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
28	40	I2C_SLV1_RW	I2C_SLV1_ADDR[6:0]						
29	41	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_EN	I2C_SLV1_BYTE_SW	I2C_SLV1_REG_DIS	I2C_SLV1_GRP	I2C_SLV1_LEN[3:0]			

Description:

These registers describe the data transfer sequence for Slave 1. Their functions correspond to those described for the Slave 0 registers (Registers 37 to 39).

4.10 Registers 43 to 45 – I²C Slave 2 Control I2C_SLV2_ADDR, I2C_SLV2_REG, and I2C_SLV2_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
2B	43	I2C_SLV2_RW	I2C_SLV2_ADDR[6:0]						
2C	44	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_EN	I2C_SLV2_BYTE_SW	I2C_SLV2_REG_DIS	I2C_SLV2_GRP	I2C_SLV2_LEN[3:0]			

Description:

These registers describe the data transfer sequence for Slave 2. Their functions correspond to those described for the Slave 0 registers (Registers 37 to 39).

4.11 Registers 46 to 48 – I²C Slave 3 Control I2C_SLV3_ADDR, I2C_SLV3_REG, and I2C_SLV3_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
2E	46	I2C_SLV3_RW	I2C_SLV3_ADDR[6:0]						
2F	47	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_EN	I2C_SLV3_BYTE_SW	I2C_SLV3_REG_DIS	I2C_SLV3_GRP	I2C_SLV3_LEN[3:0]			

Description:

These registers describe the data transfer sequence for Slave 3. Their functions correspond to those described for the Slave 0 registers (Registers 37 to 39).

4.12 Registers 49 to 53 – I²C Slave 4 Control
I2C_SLV4_ADDR, I2C_SLV4_REG, I2C_SLV4_DO, I2C_SLV4_CTRL, and I2C_SLV4_DI

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
31	49	I2C_SLV4_RW	I2C_SLV4_ADDR[6:0]						
32	50	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_EN	I2C_SLV4_INT_EN	I2C_SLV4_REG_DIS	I2C_MST_DLY[4:0]				
35	53	I2C_SLV4_DI[7:0]							

Description:

These registers describe the data transfer sequence for Slave 4. The characteristics of Slave 4 differ greatly from those of Slaves 0-3. For further information regarding the characteristics of Slaves 0-3, please refer to Registers 37 to 48.

I²C slave data transactions between the MPU-60X0 and Slave 4 are set as either read or write operations by the *I2C_SLV4_RW* bit. When this bit is 1, the transfer is a read operation. When the bit is 0, the transfer is a write operation.

I2C_SLV4_ADDR is used to specify the I²C slave address of Slave 4.

Data transfer starts at an internal register within Slave 4. This register address is specified by *I2C_SLV4_REG*.

In read mode, the result of the read will be available in *I2C_SLV4_DI*. In write mode, the contents of *I2C_SLV4_DO* will be written into the slave device.

A data transaction is performed only if the *I2C_SLV4_EN* bit is set to 1. The data transaction should be enabled once its parameters are configured in the _ADDR and _REG registers. For write, the _DO register is also required. *I2C_SLV4_EN* will be cleared after the transaction is performed once.

An interrupt is triggered at the completion of a Slave 4 data transaction if the interrupt is enabled . The status of this interrupt can be observed in Register 54.

When *I2C_SLV4_REG_DIS* is set to 1, the transaction will read or write data instead of writing a register address. This bit should equal 0 when specifying the register address within the Slave device to/from which the ensuing data transaction will take place.

I2C_MST_DLY configures the reduced access rate of I²C slaves relative to the Sample Rate. When a slave's access rate is decreased relative to the Sample Rate, the slave is accessed every

1 / (1 + *I2C_MST_DLY*) samples

This base Sample Rate in turn is determined by *SMPLRT_DIV* (register 25) and *DLPF_CFG* (register 26). Whether a slave's access rate is reduced relative to the Sample Rate is determined by I2C_MST_DELAY_CTRL (register 103).

For further information regarding the Sample Rate, please refer to register 25.

Slave 4 transactions are performed after Slave 0, 1, 2 and 3 transactions have been completed. Thus the maximum rate for Slave 4 transactions is determined by the Sample Rate as defined in Register 25.

Parameters:

I2C_SLV4_RW	When set to 1, this bit configures the data transfer as a read operation. When cleared to 0, this bit configures the data transfer as a write operation.
I2C_SLV4_ADDR	7-bit I^2C address for Slave 4.
I2C_SLV4_REG	8-bit address of the Slave 4 register to/from which data transfer starts.
I2C_SLV4_DO	This register stores the data to be written into the Slave 4. If *I2C_SLV4_RW* is set 1 (set to read), this register has no effect.
I2C_SLV4_EN	When set to 1, this bit enables Slave 4 for data transfer operations. When cleared to 0, this bit disables Slave 4 from data transfer operations.
I2C_SLV4_INT_EN	When set to 1, this bit enables the generation of an interrupt signal upon completion of a Slave 4 transaction. When cleared to 0, this bit disables the generation of an interrupt signal upon completion of a Slave 4 transaction. The interrupt status can be observed in Register 54.
I2C_SLV4_REG_DIS	When set to 1, the transaction will read or write data. When cleared to 0, the transaction will read or write a register address.
I2C_MST_DLY	Configures the decreased access rate of slave devices relative to the Sample Rate.
I2C_SLV4_DI	This register stores the data read from Slave 4. This field is populated after a read transaction.

4.13 Register 54 – I²C Master Status I2C_MST_STATUS

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
36	54	PASS_ THROUGH	I2C_SLV4 _DONE	I2C_LOST _ARB	I2C_SLV4 _NACK	I2C_SLV3 _NACK	I2C_SLV2 _NACK	I2C_SLV1 _NACK	I2C_SLV0 _NACK

Description:

This register shows the status of the interrupt generating signals in the I²C Master within the MPU-60X0. This register also communicates the status of the FSYNC interrupt to the host processor.

Reading this register will clear all the status bits in the register.

Parameters:

PASS_THROUGH This bit reflects the status of the FSYNC interrupt from an external device into the MPU-60X0. This is used as a way to pass an external interrupt through the MPU-60X0 to the host application processor. When set to 1, this bit will cause an interrupt if *FSYNC_INT_EN* is asserted in INT_PIN_CFG (Register 55).

I2C_SLV4_DONE Automatically sets to 1 when a Slave 4 transaction has completed. This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted and if the *SLV_4_DONE_INT* bit is asserted in the I2C_SLV4_CTRL register (Register 52).

I2C_LOST_ARB This bit automatically sets to 1 when the I²C Master has lost arbitration of the auxiliary I²C bus (an error condition). This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted.

I2C_SLV4_NACK This bit automatically sets to 1 when the I²C Master receives a NACK in a transaction with Slave 4. This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted.

I2C_SLV3_NACK This bit automatically sets to 1 when the I²C Master receives a NACK in a transaction with Slave 3. This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted.

I2C_SLV2_NACK This bit automatically sets to 1 when the I²C Master receives a NACK in a transaction with Slave 2. This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted.

I2C_SLV1_NACK This bit automatically sets to 1 when the I²C Master receives a NACK in a transaction with Slave 1. This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted.

I2C_SLV0_NACK This bit automatically sets to 1 when the I²C Master receives a NACK in a transaction with Slave 0. This triggers an interrupt if the *I2C_MST_INT_EN* bit in the INT_ENABLE register (Register 56) is asserted.

4.14 Register 55 – INT Pin / Bypass Enable Configuration INT_PIN_CFG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
37	55	INT_LEVEL	INT_OPEN	LATCH_INT_EN	INT_RD_CLEAR	FSYNC_INT_LEVEL	FSYNC_INT_EN	I2C_BYPASS_EN	-

Description:

This register configures the behavior of the interrupt signals at the INT pins. This register is also used to enable the FSYNC Pin to be used as an interrupt to the host application processor, as well as to enable Bypass Mode on the I^2C Master. This bit also enables the clock output.

FSYNC_INT_EN enables the FSYNC pin to be used as an interrupt to the host application processor. A transition to the active level specified in *FSYNC_INT_LEVEL* will trigger an interrupt. The status of this interrupt is read from the *PASS_THROUGH* bit in the I^2C Master Status Register (Register 54).

When *I2C_BYPASS_EN* is equal to 1 and *I2C_MST_EN* (Register 106 bit[5]) is equal to 0, the host application processor will be able to directly access the auxiliary I^2C bus of the MPU-60X0. When this bit is equal to 0, the host application processor will not be able to directly access the auxiliary I^2C bus of the MPU-60X0 regardless of the state of *I2C_MST_EN*.

For further information regarding Bypass Mode, please refer to Section 7.11 and 7.13 of the MPU-6000/MPU-6050 Product Specification document.

Parameters:

INT_LEVEL
When this bit is equal to 0, the logic level for the INT pin is active high.
When this bit is equal to 1, the logic level for the INT pin is active low.

INT_OPEN
When this bit is equal to 0, the INT pin is configured as push-pull.
When this bit is equal to 1, the INT pin is configured as open drain.

LATCH_INT_EN
When this bit is equal to 0, the INT pin emits a 50us long pulse.
When this bit is equal to 1, the INT pin is held high until the interrupt is cleared.

INT_RD_CLEAR
When this bit is equal to 0, interrupt status bits are cleared only by reading INT_STATUS (Register 58)

When this bit is equal to 1, interrupt status bits are cleared on any read operation.

FSYNC_INT_LEVEL
When this bit is equal to 0, the logic level for the FSYNC pin (when used as an interrupt to the host processor) is active high.
When this bit is equal to 1, the logic level for the FSYNC pin (when used as an interrupt to the host processor) is active low.

FSYNC_INT_EN
When equal to 0, this bit disables the FSYNC pin from causing an interrupt to the host processor.

When equal to 1, this bit enables the FSYNC pin to be used as an interrupt to the host processor.

I2C_BYPASS_EN When this bit is equal to 1 and *I2C_MST_EN* (Register 106 bit[5]) is equal to 0, the host application processor will be able to directly access the auxiliary I^2C bus of the MPU-60X0.

When this bit is equal to 0, the host application processor will not be able to directly access the auxiliary I^2C bus of the MPU-60X0 regardless of the state of *I2C_MST_EN* (Register 106 bit[5]).

4.15 Register 56 – Interrupt Enable
INT_ENABLE

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
38	56		-		FIFO_OFLOW_EN	I2C_MST_INT_EN	-	-	DATA_RDY_EN

Description:

This register enables interrupt generation by interrupt sources.

For information regarding the interrupt status for each interrupt generation source, please refer to Register 58. Further information regarding I^2C Master interrupt generation can be found in Register 54.

Bits 2 and 1 are reserved.

Parameters:

FIFO_OFLOW_EN When set to 1, this bit enables a FIFO buffer overflow to generate an interrupt.

I2C_MST_INT_EN When set to 1, this bit enables any of the I^2C Master interrupt sources to generate an interrupt.

DATA_RDY_EN When set to 1, this bit enables the Data Ready interrupt, which occurs each time a write operation to all of the sensor registers has been completed.

4.16 Register 58 – Interrupt Status
INT_STATUS

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3A	58	-	-	-	FIFO_OFLOW_INT	I2C_MST_INT	-	-	DATA_RDY_INT

Description:

This register shows the interrupt status of each interrupt generation source. Each bit will clear after the register is read.

For information regarding the corresponding interrupt enable bits, please refer to Register 56.

For a list of I^2C Master interrupts, please refer to Register 54.

Bits 2 and 1 are reserved.

Parameters:

FIFO_OFLOW_INT This bit automatically sets to 1 when a FIFO buffer overflow interrupt has been generated.

The bit clears to 0 after the register has been read.

I2C_MST_INT This bit automatically sets to 1 when an I^2C Master interrupt has been generated. For a list of I^2C Master interrupts, please refer to Register 54.

The bit clears to 0 after the register has been read.

DATA_RDY_INT This bit automatically sets to 1 when a Data Ready interrupt is generated.

The bit clears to 0 after the register has been read.

4.17 Registers 59 to 64 – Accelerometer Measurements ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Description:

These registers store the most recent accelerometer measurements.

Accelerometer measurements are written to these registers at the Sample Rate as defined in Register 25.

The accelerometer measurement registers, along with the temperature measurement registers, gyroscope measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the accelerometer sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit accelerometer measurement has a full scale defined in *ACCEL_FS* (Register 28). For each full scale setting, the accelerometers' sensitivity per LSB in *ACCEL_xOUT* is shown in the table below.

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Parameters:

ACCEL_XOUT 16-bit 2's complement value.

Stores the most recent X axis accelerometer measurement.

ACCEL_YOUT 16-bit 2's complement value.

Stores the most recent Y axis accelerometer measurement.

ACCEL_ZOUT 16-bit 2's complement value.

Stores the most recent Z axis accelerometer measurement.

4.18 Registers 65 and 66 – Temperature Measurement TEMP_OUT_H and TEMP_OUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT[15:8]							
42	66	TEMP_OUT[7:0]							

Description:

These registers store the most recent temperature sensor measurement.

Temperature measurements are written to these registers at the Sample Rate as defined in Register 25.

These temperature measurement registers, along with the accelerometer measurement registers, gyroscope measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the temperature sensor's internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

The scale factor and offset for the temperature sensor are found in the Electrical Specifications table (Section 6.4 of the MPU-6000/MPU-6050 Product Specification document).

The temperature in degrees C for a given register value may be computed as:

Temperature in degrees C = (TEMP_OUT Register Value as a signed quantity)/340 + 36.53

Please note that the math in the above equation is in decimal.

Parameters:

TEMP_OUT 16-bit signed value.

Stores the most recent temperature sensor measurement.

4.19 Registers 67 to 72 – Gyroscope Measurements GYRO_XOUT_H, GYRO_XOUT_L, GYRO_YOUT_H, GYRO_YOUT_L, GYRO_ZOUT_H, and GYRO_ZOUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Description:

These registers store the most recent gyroscope measurements.

Gyroscope measurements are written to these registers at the Sample Rate as defined in Register 25.

These gyroscope measurement registers, along with the accelerometer measurement registers, temperature measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the gyroscope sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit gyroscope measurement has a full scale defined in *FS_SEL* (Register 27). For each full scale setting, the gyroscopes' sensitivity per LSB in *GYRO_xOUT* is shown in the table below:

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Parameters:

GYRO_XOUT 16-bit 2's complement value.

Stores the most recent X axis gyroscope measurement.

GYRO_YOUT 16-bit 2's complement value.

Stores the most recent Y axis gyroscope measurement.

GYRO_ZOUT 16-bit 2's complement value.

Stores the most recent Z axis gyroscope measurement.

4.20 Registers 73 to 96 – External Sensor Data
EXT_SENS_DATA_00 through EXT_SENS_DATA_23

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
49	73	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23[7:0]							

Description:

These registers store data read from external sensors by the Slave 0, 1, 2, and 3 on the auxiliary I^2C interface. Data read by Slave 4 is stored in I2C_SLV4_DI (Register 53).

External sensor data is written to these registers at the Sample Rate as defined in Register 25. This access rate can be reduced by using the Slave Delay Enable registers (Register 103).

External sensor data registers, along with the gyroscope measurement registers, accelerometer measurement registers, and temperature measurement registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the external sensors' internal register set is always updated at the Sample Rate (or the reduced access rate) whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Data is placed in these external sensor data registers according to I2C_SLV0_CTRL, I2C_SLV1_CTRL, I2C_SLV2_CTRL, and I2C_SLV3_CTRL (Registers 39, 42, 45, and 48). When more than zero bytes are read (*I2C_SLVx_LEN* > 0) from an enabled slave (*I2C_SLVx_EN* = 1), the slave is read at the Sample Rate (as defined in Register 25) or delayed rate (if specified in Register 52 and 103). During each Sample cycle, slave reads are performed in order of Slave number. If all slaves are enabled with more than zero bytes to be read, the order will be Slave 0, followed by Slave 1, Slave 2, and Slave 3.

Each enabled slave will have EXT_SENS_DATA registers associated with it by number of bytes read (*I2C_SLVx_LEN)* in order of slave number, starting from EXT_SENS_DATA_00. Note that this means enabling or disabling a slave may change the higher numbered slaves' associated registers. Furthermore, if fewer total bytes are being read from the external sensors as a result of such a change, then the data remaining in the registers which no longer have an associated slave device (i.e. high numbered registers) will remain in these previously allocated registers unless reset.

If the sum of the read lengths of all SLVx transactions exceed the number of available EXT_SENS_DATA registers , the excess bytes will be dropped. There are 24 EXT_SENS_DATA registers and hence the total read lengths between all the slaves cannot be greater than 24 or some bytes will be lost.

Note: Slave 4's behavior is distinct from that of Slaves 0-3. For further information regarding the characteristics of Slave 4, please refer to Registers 49 to 53.

Example:

Suppose that Slave 0 is enabled with 4 bytes to be read (*I2C_SLV0_EN* = 1 and *I2C_SLV0_LEN* = 4) while Slave 1 is enabled with 2 bytes to be read, (*I2C_SLV1_EN*=1 and *I2C_SLV1_LEN* = 2). In such a situation, EXT_SENS_DATA _00 through _03 will be associated with Slave 0, while EXT_SENS_DATA _04 and 05 will be associated with Slave 1.

If Slave 2 is enabled as well, registers starting from EXT_SENS_DATA_06 will be allocated to Slave 2.

If Slave 2 is disabled while Slave 3 is enabled in this same situation, then registers starting from EXT_SENS_DATA_06 will be allocated to Slave 3 instead.

Register Allocation for Dynamic Disable vs. Normal Disable

If a slave is disabled at any time, the space initially allocated to the slave in the EXT_SENS_DATA register, will remain associated with that slave. This is to avoid dynamic adjustment of the register allocation.

The allocation of the EXT_SENS_DATA registers is recomputed only when (1) all slaves are disabled, or (2) the *I2C_MST_RST* bit is set (Register 106).

This above is also true if one of the slaves gets NACKed and stops functioning.

4.21 Register 99 – I²C Slave 0 Data Out I2C_SLV0_DO

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
63	99	I2C_SLV0_DO[7:0]							

Description:

This register holds the output data written into Slave 0 when Slave 0 is set to write mode.

For further information regarding Slave 0 control, please refer to Registers 37 to 39.

Parameters:

I2C_SLV0_DO 8 bit unsigned value that is written into Slave 0 when Slave 0 is set to write mode.

4.22 Register 100 – I²C Slave 1 Data Out I2C_SLV1_DO

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
64	100	I2C_SLV1_DO[7:0]							

Description:

This register holds the output data written into Slave 1 when Slave 1 is set to write mode.

For further information regarding Slave 1 control, please refer to Registers 40 to 42.

Parameters:

I2C_SLV1_DO 8 bit unsigned value that is written into Slave 1 when Slave 1 is set to write mode.

4.23 Register 101 – I²C Slave 2 Data Out I2C_SLV2_DO

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
65	101	I2C_SLV2_DO[7:0]							

Description:

This register holds the output data written into Slave 2 when Slave 2 is set to write mode.

For further information regarding Slave 2 control, please refer to Registers 43 to 45.

Parameters:

I2C_SLV2_DO 8 bit unsigned value that is written into Slave 2 when Slave 2 is set to write mode.

4.24 Register 102 – I²C Slave 3 Data Out I2C_SLV3_DO

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
66	102	I2C_SLV3_DO[7:0]							

Description:

This register holds the output data written into Slave 3 when Slave 3 is set to write mode.

For further information regarding Slave 3 control, please refer to Registers 46 to 48.

Parameters:

I2C_SLV3_DO 8 bit unsigned value that is written into Slave 3 when Slave 3 is set to write mode.

4.25 Register 103 – I^2C Master Delay Control I2C_MST_DELAY_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
67	103	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN

Description:

This register is used to specify the timing of external sensor data shadowing. The register is also used to decrease the access rate of slave devices relative to the Sample Rate.

When *DELAY_ES_SHADOW* is set to 1, shadowing of external sensor data is delayed until all data has been received.

When *I2C_SLV4_DLY_EN, I2C_SLV3_DLY_EN, I2C_SLV2_DLY_EN, I2C_SLV1_DLY_EN,* and *I2C_SLV0_DLY_EN* are enabled, the rate of access for the corresponding slave devices is reduced.

When a slave's access rate is decreased relative to the Sample Rate, the slave is accessed every

1 / (1 + *I2C_MST_DLY*) samples.

This base Sample Rate in turn is determined by *SMPLRT_DIV* (register 25) and *DLPF_CFG* (register 26).

For further information regarding *I2C_MST_DLY*, please refer to register 52.

For further information regarding the Sample Rate, please refer to register 25.

Bits 6 and 5 are reserved.

Parameters:

DELAY_ES_SHADOW	When set, delays shadowing of external sensor data until all data has been received.
I2C_SLV4_DLY_EN	When enabled, slave 4 will only be accessed at a decreased rate.
I2C_SLV3_DLY_EN	When enabled, slave 3 will only be accessed at a decreased rate.
I2C_SLV2_DLY_EN	When enabled, slave 2 will only be accessed at a decreased rate.
I2C_SLV1_DLY_EN	When enabled, slave 1 will only be accessed at a decreased rate.
I2C_SLV0_DLY_EN	When enabled, slave 0 will only be accessed at a decreased rate.

4.26 Register 104 – Signal Path Reset
SIGNAL_PATH_RESET

Type: Write Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
68	104	-	-	-	-	-	GYRO_RESET	ACCEL_RESET	TEMP_RESET

Description:

This register is used to reset the analog and digital signal paths of the gyroscope, accelerometer, and temperature sensors.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

Note: This register does not clear the sensor registers. The reset initializes the serial interface as well.

Bits 7 to 3 are reserved.

Parameters:

GYRO_RESET When set to 1, this bit resets the gyroscope analog and digital signal paths.

ACCEL_RESET When set to 1, this bit resets the accelerometer analog and digital signal paths.

TEMP_RESET When set to 1, this bit resets the temperature sensor analog and digital signal paths.

4.27 Register 106 – User Control
USER_CTRL

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6A	106	-	FIFO_EN	I2C_MST _EN	I2C_IF _DIS	-	FIFO _RESET	I2C_MST _RESET	SIG_COND _RESET

Description:

This register allows the user to enable and disable the FIFO buffer, I^2C Master Mode, and primary I^2C interface. The FIFO buffer, I^2C Master, sensor signal paths and sensor registers can also be reset using this register.

When *I2C_MST_EN* is set to 1, I^2C Master Mode is enabled. In this mode, the MPU-60X0 acts as the I^2C Master to the external sensor slave devices on the auxiliary I^2C bus. When this bit is cleared to 0, the auxiliary I^2C bus lines (AUX_DA and AUX_CL) are logically driven by the primary I^2C bus (SDA and SCL). This is a precondition to enabling Bypass Mode. For further information regarding Bypass Mode, please refer to Register 55.

MPU-6000: The primary SPI interface will be enabled in place of the disabled primary I^2C interface when *I2C_IF_DIS* is set to 1.

MPU-6050: Always write 0 to *I2C_IF_DIS*.

When the reset bits (*FIFO_RESET, I2C_MST_RESET, and SIG_COND_RESET*) are set to 1, these reset bits will trigger a reset and then clear to 0.

Bits 7 and 3 are reserved.

Parameters:

FIFO_EN — When set to 1, this bit enables FIFO operations.

When this bit is cleared to 0, the FIFO buffer is disabled. The FIFO buffer cannot be written to or read from while disabled.

The FIFO buffer's state does not change unless the MPU-60X0 is power cycled.

I2C_MST_EN — When set to 1, this bit enables I^2C Master Mode.

When this bit is cleared to 0, the auxiliary I^2C bus lines (AUX_DA and AUX_CL) are logically driven by the primary I^2C bus (SDA and SCL).

I2C_IF_DIS — MPU-6000: When set to 1, this bit disables the primary I^2C interface and enables the SPI interface instead.

MPU-6050: Always write this bit as zero.

FIFO_RESET — This bit resets the FIFO buffer when set to 1 while *FIFO_EN* equals 0. This bit automatically clears to 0 after the reset has been triggered.

I2C_MST_RESET — This bit resets the I^2C Master when set to 1 while *I2C_MST_EN* equals 0. This bit automatically clears to 0 after the reset has been triggered.

SIG_COND_RESET When set to 1, this bit resets the signal paths for all sensors (gyroscopes, accelerometers, and temperature sensor). This operation will also clear the sensor registers. This bit automatically clears to 0 after the reset has been triggered.

When resetting only the signal path (and not the sensor registers), please use Register 104, SIGNAL_PATH_RESET.

4.28 Register 107 – Power Management 1
PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Description:

This register allows the user to configure the power mode and clock source. It also provides a bit for resetting the entire device, and a bit for disabling the temperature sensor.

By setting *SLEEP* to 1, the MPU-60X0 can be put into low power sleep mode. When *CYCLE* is set to 1 while *SLEEP* is disabled, the MPU-60X0 will be put into Cycle Mode. In Cycle Mode, the device cycles between sleep mode and waking up to take a single sample of data from accelerometer at a rate determined by *LP_WAKE_CTRL* (register 108). To configure the wake frequency, use *LP_WAKE_CTRL* within the Power Management 2 register (Register 108).

An internal 8MHz oscillator, gyroscope based clock, or external sources can be selected as the MPU-60X0 clock source. When the internal 8 MHz oscillator or an external source is chosen as the clock source, the MPU-60X0 can operate in low power modes with the gyroscopes disabled.

Upon power up, the MPU-60X0 clock source defaults to the internal oscillator. However, it is highly recommended that the device be configured to use one of the gyroscopes (or an external clock source) as the clock reference for improved stability. The clock source can be selected according to the following table.

CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

For further information regarding the MPU-60X0 clock source, please refer to the MPU-6000/MPU-6050 Product Specification document.

Bit 4 is reserved.

Parameters:

DEVICE_RESET	When set to 1, this bit resets all internal registers to their default values. The bit automatically clears to 0 once the reset is done. The default values for each register can be found in Section 3.
SLEEP	When set to 1, this bit puts the MPU-60X0 into sleep mode.
CYCLE	When this bit is set to 1 and *SLEEP* is disabled, the MPU-60X0 will cycle between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by *LP_WAKE_CTRL* (register 108).
TEMP_DIS	When set to 1, this bit disables the temperature sensor.
CLKSEL	3-bit unsigned value. Specifies the clock source of the device.

Note:

When using SPI interface, user should use DEVICE_RESET (register 107) as well as SIGNAL_PATH_RESET (register 104) to ensure the reset is performed properly. The sequence used should be:

1. Set DEVICE_RESET = 1 (register PWR_MGMT_1)
2. Wait 100ms
3. Set GYRO_RESET = ACCEL_RESET = TEMP_RESET = 1 (register SIGNAL_PATH_RESET)
4. Wait 100ms

4.29 Register 108 – Power Management 2
PWR_MGMT_2

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6C	108	LP_WAKE_CTRL[1:0]		STBY_XA	STBY_YA	STBY_ZA	STBY_XG	STBY_YG	STBY_ZG

Description:

This register allows the user to configure the frequency of wake-ups in Accelerometer Only Low Power Mode. This register also allows the user to put individual axes of the accelerometer and gyroscope into standby mode.

The MPU-60X0 can be put into Accelerometer Only Low Power Mode using the following steps:

(i) Set CYCLE bit to 1
(ii) Set SLEEP bit to 0
(iii) Set TEMP_DIS bit to 1
(iv) Set STBY_XG, STBY_YG, STBY_ZG bits to 1

All of the above bits can be found in Power Management 1 register (Register 107).

In this mode, the device will power off all devices except for the primary I^2C interface, waking only the accelerometer at fixed intervals to take a single measurement. The frequency of wake-ups can be configured with *LP_WAKE_CTRL* as shown below.

LP_WAKE_CTRL	Wake-up Frequency
0	1.25 Hz
1	5 Hz
2	20 Hz
3	40 Hz

For further information regarding the MPU-6050's power modes, please refer to Register 107.

The user can put individual accelerometer and gyroscopes axes into standby mode by using this register. If the device is using a gyroscope axis as the clock source and this axis is put into standby mode, the clock source will automatically be changed to the internal 8MHz oscillator.

Parameters:

LP_WAKE_CTRL	2-bit unsigned value. Specifies the frequency of wake-ups during Accelerometer Only Low Power Mode.
STBY_XA	When set to 1, this bit puts the X axis accelerometer into standby mode.
STBY_YA	When set to 1, this bit puts the Y axis accelerometer into standby mode.
STBY_ZA	When set to 1, this bit puts the Z axis accelerometer into standby mode.
STBY_XG	When set to 1, this bit puts the X axis gyroscope into standby mode.
STBY_YG	When set to 1, this bit puts the Y axis gyroscope into standby mode.
STBY_ZG	When set to 1, this bit puts the Z axis gyroscope into standby mode.

4.30 Register 114 and 115 – FIFO Count Registers FIFO_COUNT_H and FIFO_COUNT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
72	114	FIFO_COUNT[15:8]							
73	115	FIFO_COUNT[7:0]							

Description:

These registers keep track of the number of samples currently in the FIFO buffer.

These registers shadow the FIFO Count value. Both registers are loaded with the current sample count when FIFO_COUNT_H (Register 72) is read.

Note: Reading only FIFO_COUNT_L will not update the registers to the current sample count. FIFO_COUNT_H must be accessed first to update the contents of both these registers.

FIFO_COUNT should always be read in high-low order in order to guarantee that the most current FIFO Count value is read.

Parameters:

FIFO_COUNT 16-bit unsigned value. Indicates the number of bytes stored in the FIFO buffer. This number is in turn the number of bytes that can be read from the FIFO buffer and it is directly proportional to the number of samples available given the set of sensor data bound to be stored in the FIFO (register 35 and 36).

4.31 Register 116 – FIFO Read Write
FIFO_R_W

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
74	116	FIFO_DATA[7:0]							

Description:

This register is used to read and write data from the FIFO buffer.

Data is written to the FIFO in order of register number (from lowest to highest). If all the FIFO enable flags (see below) are enabled and all External Sensor Data registers (Registers 73 to 96) are associated with a Slave device, the contents of registers 59 through 96 will be written in order at the Sample Rate.

The contents of the sensor data registers (Registers 59 to 96) are written into the FIFO buffer when their corresponding FIFO enable flags are set to 1 in FIFO_EN (Register 35). An additional flag for the sensor data registers associated with I^2C Slave 3 can be found in I2C_MST_CTRL (Register 36).

If the FIFO buffer has overflowed, the status bit *FIFO_OFLOW_INT* is automatically set to 1. This bit is located in INT_STATUS (Register 58). When the FIFO buffer has overflowed, the oldest data will be lost and new data will be written to the FIFO.

If the FIFO buffer is empty, reading this register will return the last byte that was previously read from the FIFO until new data is available. The user should check *FIFO_COUNT* to ensure that the FIFO buffer is not read when empty.

Parameters:

FIFO_DATA 8-bit data transferred to and from the FIFO buffer.

4.32 Register 117 – Who Am I
WHO_AM_I

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Description:

This register is used to verify the identity of the device. The contents of *WHO_AM_I* are the upper 6 bits of the MPU-60X0's 7-bit I^2C address. The least significant bit of the MPU-60X0's I^2C address is determined by the value of the AD0 pin. The value of the AD0 pin is not reflected in this register.

The default value of the register is 0x68.

Bits 0 and 7 are reserved. (Hard coded to 0)

Parameters:

WHO_AM_I Contains the 6-bit I^2C address of the MPU-60X0.

The Power-On-Reset value of Bit6:Bit1 is 110 100.

InvenSense