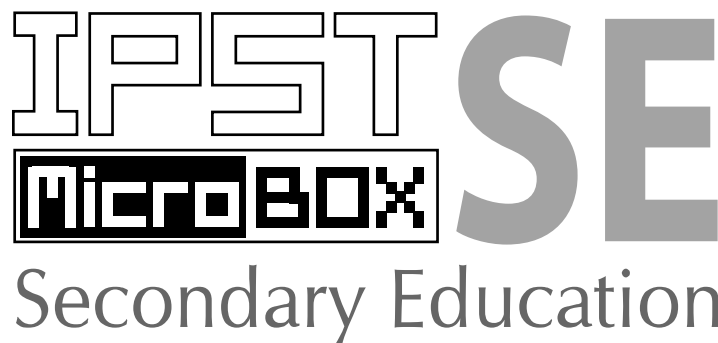


คู่มือเริ่มต้นใช้งานกล่องสมองกล



กับ **Arduino IDE**

IPST-MicroBOX Secondary Education (SE)
Starter manual with Arduino IDE

IPST-MicoBOX (SE) Starter Manual

คู่มือเริ่มต้นใช้งานกล่องสมองกล IPST-MicoBOX (SE) กับ Arduino IDE

สงวนลิขสิทธิ์ตาม พ.ร.บ. ลิขสิทธิ์ พ.ศ. 2521

ห้ามการลอกเลียนไม่ว่าส่วนหนึ่งส่วนใดของหนังสือเล่มนี้ นอกจากจะได้รับอนุญาต

ใครควรใช้หนังสือเล่มนี้

1. นักเรียน นิสิต นักศึกษา และบุคคลทั่วไปที่มีความสนใจในการนำไมโครคอนโทรลเลอร์ไปประยุกต์ใช้ในการทดลองทางวิทยาศาสตร์ หรือสนใจในการเรียนรู้และทดลองวิทยาศาสตร์ในแนวทางใหม่ที่ใช้กิจกรรมเป็นสื่อ โดยมีไมโครคอนโทรลเลอร์เป็นส่วนประกอบ
2. สถาบันการศึกษา โรงเรียน วิทยาลัย มหาวิทยาลัย ที่มีการเปิดการเรียนการสอนวิชาอิเล็กทรอนิกส์หรือภาควิชาวิศวกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์
3. คณาจารย์ที่มีความต้องการศึกษา และเตรียมการเรียนการสอนวิชาไมโครคอนโทรลเลอร์ รวมถึงวิทยาศาสตร์ประยุกต์ที่ต้องการบูรณาการความรู้ทางอิเล็กทรอนิกส์-ไมโครคอนโทรลเลอร์-การเขียนโปรแกรมคอมพิวเตอร์-การทดลองทางวิทยาศาสตร์ ในระดับมัธยมศึกษา อาชีวศึกษา และปริญญาตรี

ดำเนินการจัดพิมพ์และจำหน่ายโดย

บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด

108 ซ. สุขุมวิท 101/2 ถ. สุขุมวิท แขวงบางนา เขตบางนา กรุงเทพฯ 10260

โทรศัพท์ 0-2747-7001-4

โทรสาร 0-2747-7005

รายละเอียดที่ปรากฏในคู่มือเริ่มต้นใช้งานกล่องสมองกล IPST-MicoBOX (SE) กับ Arduino IDE ผ่านการตรวจทานอย่างละเอียดและถี่ถ้วน เพื่อให้มีความสมบูรณ์และถูกต้องมากที่สุดภายใต้เงื่อนไขและเวลาที่พึงมีก่อนการจัดพิมพ์เผยแพร่ ความเสียหายอันอาจเกิดจากการนำข้อมูลในหนังสือเล่มนี้ไปใช้ ทางบริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด มิได้มีภาระในการรับผิดชอบแต่ประการใด ความผิดพลาดคลาดเคลื่อนที่อาจมีและได้รับการจัดพิมพ์เผยแพร่ออกไปนั้น ทางบริษัทฯ จะพยายามชี้แจงและแก้ไขในการจัดพิมพ์ครั้งต่อไป

ที่มาของชุดกล่องสมองกล IPST-MicroBOX (SE)

จากการเริ่มต้นพัฒนาชุดกล่องสมองกล IPST-MicroBOX โดยสาขาคอมพิวเตอร์ สถาบันส่งเสริมการสอนวิทยาศาสตร์และเทคโนโลยีหรือ สสวท. ที่สามารถนำไปบูรณาการกับวิชาอื่นๆ ได้ ได้รับการตอบรับและมีการนำไปใช้ในการเรียนการสอนด้านวิทยาศาสตร์ประยุกต์ ด้านการเขียนและพัฒนาโปรแกรมภาษา C รวมถึงในวิชาโครงงาน เพื่อให้ผู้เรียนสามารถนำองค์ความรู้ไปใช้และต่อยอดเพื่อสร้างโครงงานวิทยาศาสตร์สมัยใหม่

IPST-MicroBOX เพื่อเป็นสื่อทางเลือกหนึ่งสำหรับครูผู้สอนในการจัดการเรียนการสอนวิชาการโปรแกรม วิชาโครงงาน ในระดับมัธยมศึกษา ชุดการเรียนการสอนนี้จะเน้นการจัดกิจกรรมการเรียนรู้แบบบูรณาการ นักเรียนได้รู้เกี่ยวกับอุปกรณ์และอิเล็กทรอนิกส์เบื้องต้น การเขียนโปรแกรมเพื่อควบคุมไมโครคอนโทรลเลอร์ การทำโครงงานซึ่งต้องบูรณาการกับวิชา ฟิสิกส์ เคมี ชีววิทยา คณิตศาสตร์ และคอมพิวเตอร์เข้าด้วยกัน ซึ่งจะทำให้การเรียนการสอนมีความน่าสนใจ และเป็นอีกแนวทางหนึ่งในการสอนเพื่อให้นักเรียนรักการเขียนโปรแกรม รู้จักคิดวิเคราะห์และแก้ปัญหาทั้งในวิชาที่เรียนและในชีวิตประจำวัน

จนกระทั่งในปี พ.ศ. 2556 สสวท. ได้มีการจัดตั้งโครงการห้องเรียนวิทยาศาสตร์ในระดับมัธยมศึกษาตอนต้นขึ้น โดยมีชุดกล่องสมองกล IPST-MicroBOX เป็นสื่อการเรียนรู้หนึ่งที่ควรมีในห้องเรียนวิทยาศาสตร์ เนื่องจาก IPST-MicroBOX เดิมออกแบบมาเพื่อใช้ในระดับมัธยมศึกษาตอนปลายเป็นหลัก ดังนั้นเมื่อนำมาใช้ในห้องเรียนวิทยาศาสตร์ในระดับมัธยมศึกษาตอนต้น จึงต้องมีการปรับปรุงใหม่ เพื่อให้เหมาะกับนักเรียนในระดับนี้ กอปรกับการเปลี่ยนแปลงของเทคโนโลยีอุปกรณ์อิเล็กทรอนิกส์ คอมพิวเตอร์สมัยใหม่ที่มีระบบปฏิบัติการที่หลากหลายทั้ง วินโดวส์, ลินุกซ์ หรือกระทั่ง MAC OS พอร์ตเชื่อมต่อของคอมพิวเตอร์ที่เน้นไปยังพอร์ต USB ส่งผลให้การปรับปรุง IPST-MicroBOX ครั้งนี้ จึงต้องเลือกฮาร์ดแวร์ที่สามารถรองรับกับพอร์ต USB เลือกซอฟต์แวร์ที่ใช้ในการพัฒนาโปรแกรมที่รองรับกับความหลากหลายของระบบปฏิบัติการ และยังต้องมีการพัฒนาชุดคำสั่งต่างๆ ที่ทำให้นักเรียนในระดับมัธยมต้นสามารถเรียนรู้และทำความเข้าใจได้

IPST-MicroBOX Secondary Education หรือ IPST-MicroBOX (SE) จึงเกิดขึ้น โดยในชุดจะมีอุปกรณ์ที่เพียงพอสำหรับการเรียนรู้ในเบื้องต้น ต่อยอดไปทำโครงงานอย่างง่ายและขั้นกลางได้ ทั้งยังมีชิ้นส่วนในให้นำไปสร้างเป็นหุ่นยนต์อัตโนมัติขนาดเล็กได้ด้วย ภายใต้งบประมาณรวมที่ถูกกลง ทางด้านซอฟต์แวร์เลือกใช้ ซอฟต์แวร์ Wiring IDE (www.wiring.org.co) อันเป็นซอฟต์แวร์สำหรับพัฒนาโปรแกรมด้วยภาษา C/C++ ที่ใช้งานได้กับระบบปฏิบัติการวินโดวส์, ลินุกซ์ และ MAC OS ทั้งยังเป็นซอฟต์แวร์แบบซอร์สเปิด ใช้งานได้โดยไม่มีค่าใช้จ่าย และไม่จำกัดระยะเวลาใช้งาน รวมถึงมีการปรับปรุงอย่างต่อเนื่องเพื่อให้ได้ซอฟต์แวร์ที่มีประสิทธิภาพสูง

ชุดกล่องสมองกล IPST-MicroBOX Secondary Education หรือ IPST-MicroBOX (SE) เป็นสื่อการเรียนรู้ทางเลือกสำหรับครู, อาจารย์ และนักเรียนที่มีความประสงค์ในการต่อยอดหรือประยุกต์ใช้กล่องสมองกลที่มีไมโครคอนโทรลเลอร์เป็นตัวควบคุมหลักในการเรียนรู้และพัฒนาโครงงานด้านวิทยาศาสตร์และเทคโนโลยี การจัดหาสื่อการเรียนรู้นี้เป็นไปในรูปแบบสมัครใจ การบริการเกี่ยวกับการจัดหาและซ่อมแซมอุปกรณ์อยู่ภายใต้ความรับผิดชอบของบริษัท อินโนเวตีฟ แอ็กเพอริเมนต์ จำกัด (www.inex.co.th หรือ www.ipst-microbox.com)

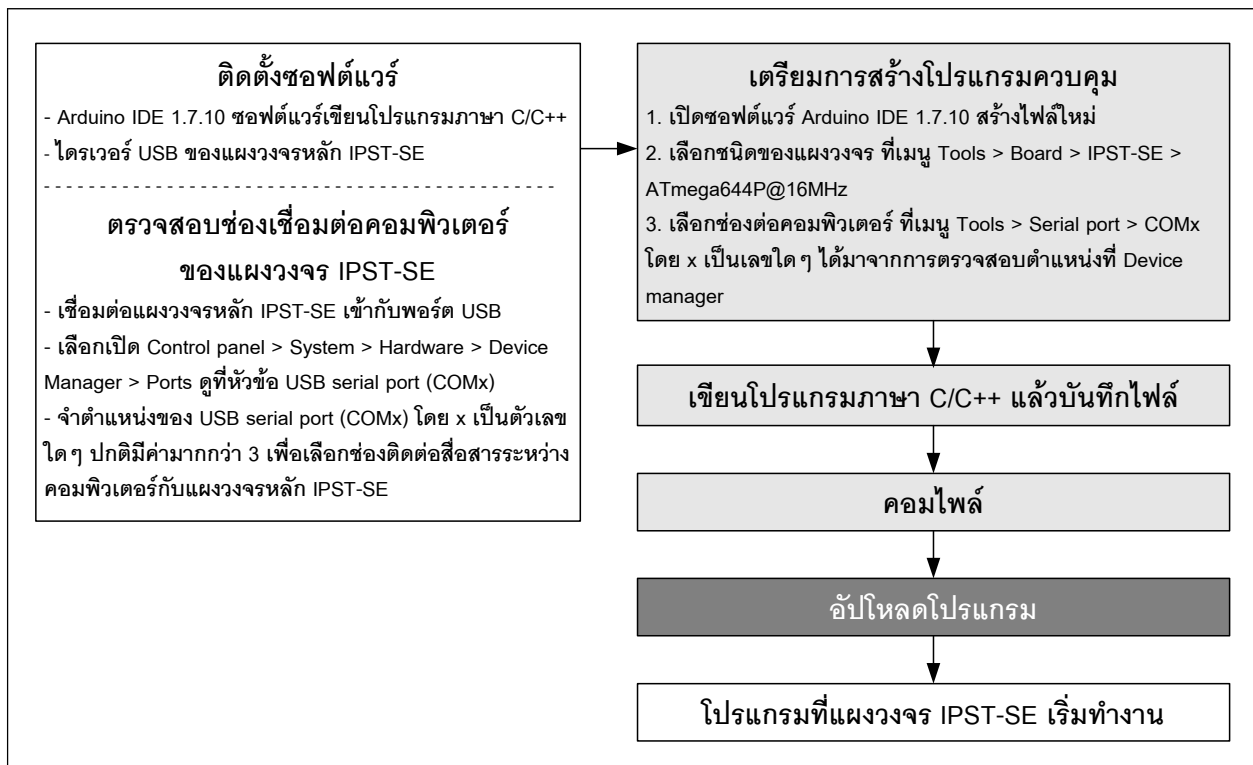
สารบัญ

บทที่ 1 เริ่มต้นใช้งานชุดกล่องสมองกล IPST-MicroBOX (SE).....	5
บทที่ 2 แนะนำชุดกล่องสมองกล IPST-MicroBOX (SE).....	19
บทที่ 3 รู้จักกับ Arduino IDE ซอฟต์แวร์พัฒนาโปรแกรมภาษา C/C++ สำหรับชุดกล่องสมองกล IPST-MicroBOX (SE).....	43
บทที่ 4 ทดสอบการควบคุมอุปกรณ์เบื้องต้นของชุดกล่องสมองกล IPST-MicroBOX (SE).....	53
บทที่ 5 ความรู้เบื้องต้นเกี่ยวกับไฟล์ไลบรารีของชุดกล่องสมองกล IPST-MicroBOX (SE).....	71
บทที่ 6 การแสดงผลด้วยจอกราฟิก LCD สีของชุดกล่องสมองกล IPST-MicroBOX (SE).....	97
บทที่ 7 ควบคุมการติดดับของ LED ด้วยซอฟต์แวร์.....	119
บทที่ 8 การควบคุม LED หลายดวงของชุดกล่องสมองกล IPST-MicroBOX (SE).....	127
บทที่ 9 ติดต่อกับสวิทช์เพื่ออ่านค่าและนำไปใช้งาน.....	147
บทที่ 10 การอ่านค่าสัญญาณอะนาลอกอย่างง่าย.....	161

บทที่ 1

เริ่มต้นใช้งานกล่องสมองกล **IPST-MicroBOX SE**

การใช้งานชุดกล่องสมองกล IPST-MicroBOX Secondary Education (SE) มีขั้นตอนโดยสรุปดังแผนภาพในรูปที่ 1-1 ในบทนี้จะอธิบายถึงขั้นตอนต่างๆ ในการเริ่มต้นใช้งานชุดกล่องสมองกล IPST-MicroBOX (SE) เป็นลำดับไป



รูปที่ 1-1 แผนภาพแสดงขั้นตอนและกระบวนการเรียนรู้เพื่อใช้งานชุดกล่องสมองกล IPST-MicroBOX(SE) เริ่มจากด้านซ้าย ตั้งแต่การติดตั้งโปรแกรม และการตรวจสอบการเชื่อมต่อ ระหว่างแผงวงจรควบคุมกับคอมพิวเตอร์ ไล่มาทางขวา เริ่มจากขั้นตอนเตรียมการสร้างโปรแกรมควบคุม, เขียนโปรแกรม, คอมไพล์หรือการแปลโปรแกรมภาษา C เป็นภาษาเครื่อง, อัปโหลดหรือส่งโปรแกรมไปยังแผงวงจรหลัก **IPST-SE** จากนั้นจึงทำการรันโปรแกรมเพื่อตรวจสอบการทำงาน

1.1 ติดตั้งโปรแกรมซอฟต์แวร์และไดรเวอร์

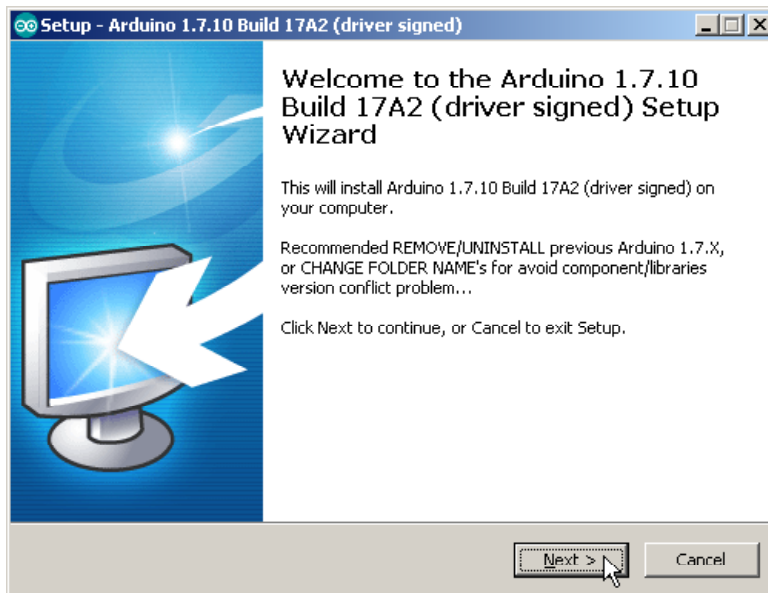
1.1.1 ระบบปฏิบัติการที่รองรับ

ซอฟต์แวร์สำหรับพัฒนาโปรแกรมคือ **Arduino IDE 1.7.10** ทำงานได้กับระบบปฏิบัติการหรือแพลตฟอร์ม (platform) ดังนี้

- Mac OS X 10.6 (ทั้งรุ่นที่ใช้ซีพียูเพาเวอร์พีซีและอินเทล)
- วินโดวส์ 7 ขึ้นไป

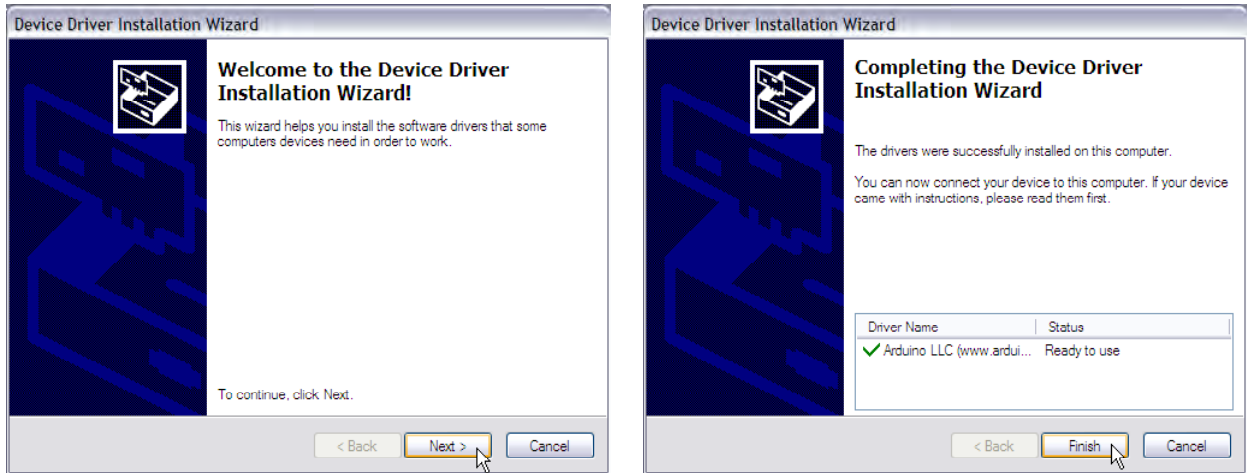
1.1.2 ขั้นตอนติดตั้งซอฟต์แวร์และไดรเวอร์ USB

(1.1.2.1) นำแผ่นซีดีรอมที่มากับชุดกล่องสมองกล IPST-MicroBOX (SE) ใส่เข้าไปในซีดีรอมไดรฟ์ของคอมพิวเตอร์ ค้นหาและดับเบิลคลิกที่ไฟล์ *Arduino1.7.10_Setup160603.exe* (หมายเลขรุ่นของซอฟต์แวร์เปลี่ยนแปลงได้) จะปรากฏหน้าต่างต้อนรับสู่การติดตั้งให้คลิก **Next**

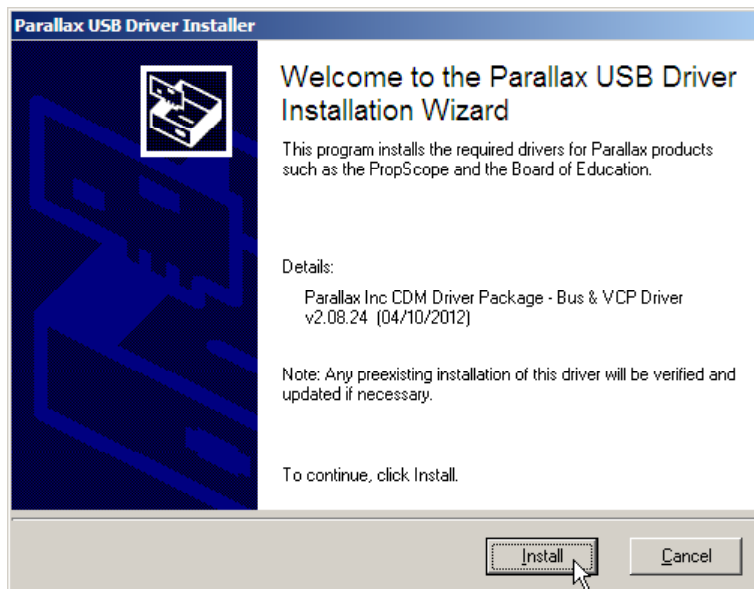


(1.1.2.2) คลิกตอบรับการติดตั้งในแต่ละขั้นตอน จนกระทั่งการติดตั้งเสร็จสิ้น

(1.1.2.3) จากนั้นจะเข้าสู่การติดตั้งไดรเวอร์ USB ตัวที่ 1 ให้คลิกปุ่ม **Next** เพื่อตอบรับ เมื่อการติดตั้งเสร็จสิ้น คลิกปุ่ม **Finish** เพื่อเข้าสู่ขั้นตอนถัดไป

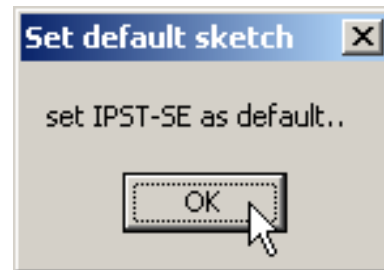
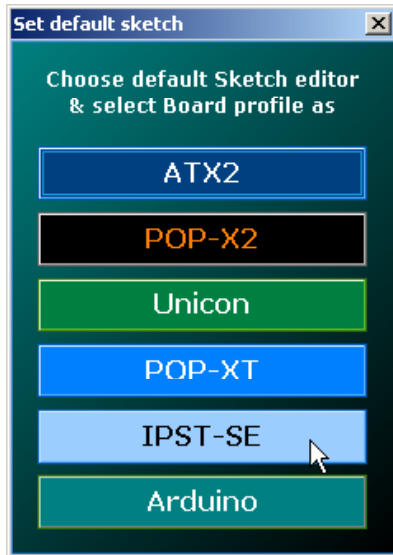


(1.1.2.4) ต่อไปเป็นการติดตั้งไดรเวอร์ USB ตัวที่สอง ซึ่งเป็นไดรเวอร์สำหรับแผงวงจร IPST-SE คลิกปุ่ม **Install** เพื่อทำการติดตั้งไดรเวอร์ และคลิก **Finish** เมื่อการติดตั้งเสร็จสมบูรณ์

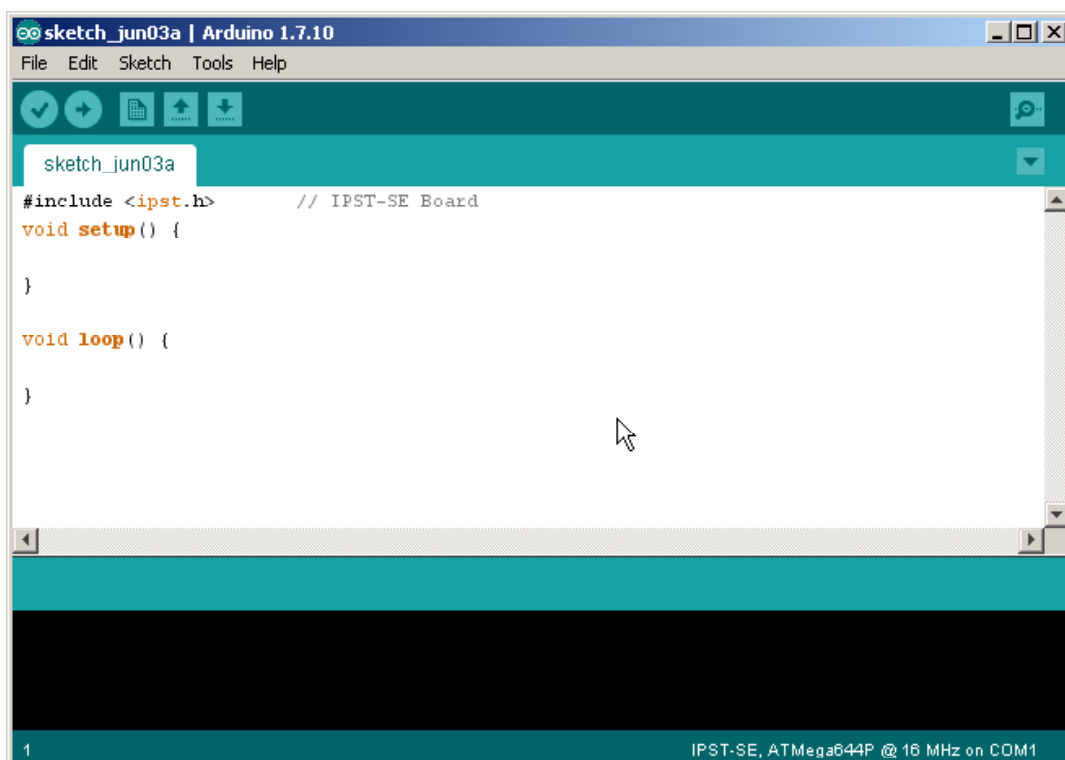


8 • คู่มือเริ่มต้นใช้งานกล่องสมองกล SE กับ Arduino IDE

(1.1.2.5) เมื่อติดตั้งโปรแกรมเสร็จ จะมีการเรียกซอฟต์แวร์ Arduino IDE ให้ทำงานทันที มีหน้าต่างเลือกฮาร์ดแวร์ที่ต้องการใช้งานปรากฏขึ้นมา ในที่นี้ให้เลือก **IPST-SE** จากนั้นจะปรากฏหน้าต่างให้ยืนยัน คลิกปุ่ม **OK** เพื่อตอบรับ



(1.1.2.6) รอสักครู่ซอฟต์แวร์ Arduino IDE เวอร์ชัน 1.7.10 จะทำงาน แสดงหน้าต่างหลักพร้อมกับโค้ดเริ่มต้นของโปรแกรม ซึ่งก็คือ คำสั่ง `#include <ipst.h>` เพื่อเรียกใช้งานไลบรารี ipst.h ตามด้วยส่วนประกอบหลักของโปรแกรมคือ ฟังก์ชัน `setup()` และ `loop()`

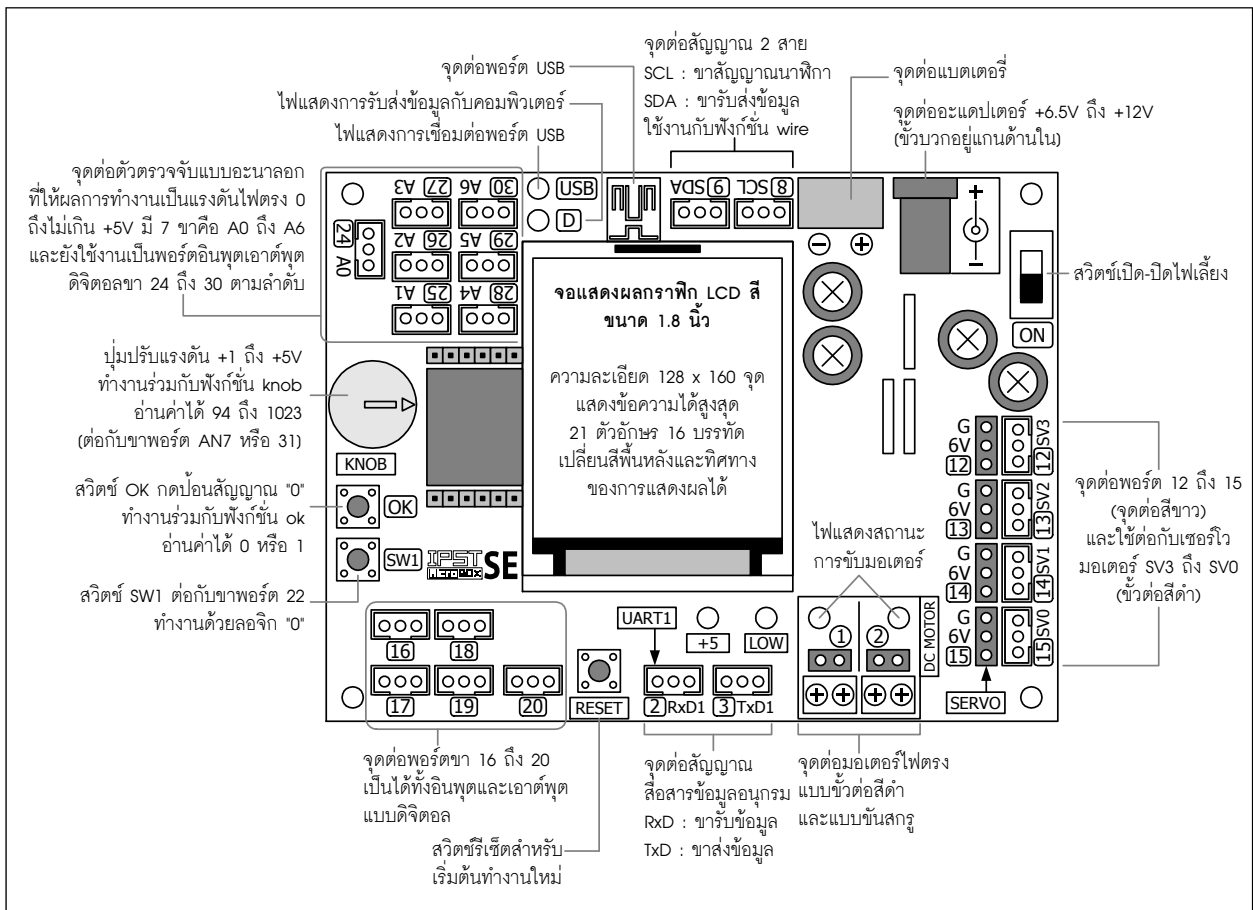


(1.1.2.7) เมื่อได้ตามนี้ แสดงว่า การติดตั้งซอฟต์แวร์ Arduino IDE และไดรเวอร์ต่างๆ เสร็จสมบูรณ์ พร้อมสำหรับการพัฒนาโปรแกรมต่อไป หากต้องการเรียกให้ Arduino IDE 1.7.10 ทำงานในครั้งต่อไป ให้คลิกที่ **Start > All Programs > Arduino 1.7.x > Arduino 1.7.10**

จากนี้ซอฟต์แวร์ Arduino IDE (เวอร์ชันสำหรับกล่องสมองกล IPST- MicroBOX(SE) พร้อมสำหรับการพัฒนาโปรแกรมแล้ว

1.2 แผงวงจรหลัก IPST-SE ของชุดกล่องสมองกล IPST-MicroBOX (SE)

อุปกรณ์หลักที่ใช้ในการเรียนรู้กล่องสมองกลคือ ชุดกล่องสมองกล IPST-MicroBOX (SE) ที่มีแผงวงจรหลักชื่อ **IPST-SE** มีหน้าตาแสดงดังรูปที่ 1-2 พร้อมคำอธิบายของส่วนประกอบต่างๆ แผงวงจร IPST-SE เป็นแผงวงจรขนาดเล็กที่มีไมโครคอนโทรลเลอร์เบอร์ ATmega644P เป็นหัวใจหลักในการควบคุมการทำงาน โดยตัวควบคุมหลักหรือไมโครคอนโทรลเลอร์จะได้รับการโปรแกรมผ่านทางพอร์ต USB ด้วยซอฟต์แวร์ Arduino IDE 1.7.10



รูปที่ 1-2 แสดงส่วนประกอบและหน้าที่ของแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE)

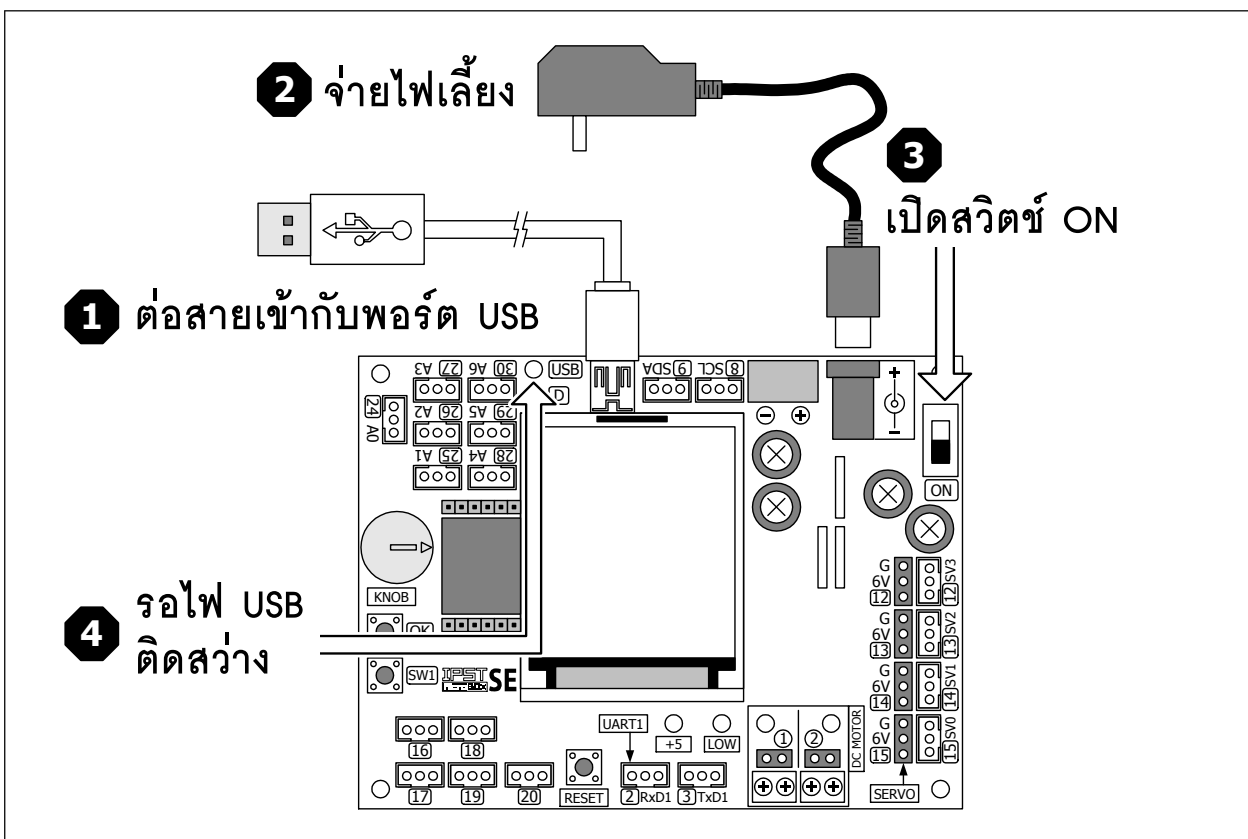
บนแผงวงจรควบคุมนี้มีจุดต่อเพื่อรับสัญญาณจากตัวตรวจจับภายนอกทั้งแบบอะนาล็อกและดิจิทัลเพื่อช่วยให้แผงวงจรสามารถรับข้อมูลจากสิ่งแวดล้อม เช่น แสง, อุณหภูมิ, ระยะห่างจากวัตถุของตัวตรวจจับ เป็นต้น นอกจากนี้ยังมีจุดต่อเพื่อส่งสัญญาณออกไปควบคุมอุปกรณ์ภายนอก อาทิ ไดโอดเปล่งแสง ลำโพง มอเตอร์ไฟตรง และเซอร์โวมอเตอร์

ด้านการแสดงผล แผงวงจร IPST-SE มีจอแสดงผลแบบกราฟิก LCD สีขนาด 1.8 นิ้วในตัว มีความละเอียด 128 x 160 จุด แสดงตัวอักษรได้สูงสุด 21 ตัวอักษร 16 บรรทัด แสดงภาพกราฟิกสีได้ (ไม่รองรับภาพถ่ายที่มีความละเอียดสูง) และเลือกทิศทางการแสดงผลได้

1.3 ทดสอบการอัปโหลดโปรแกรม

สำหรับการเขียนโปรแกรมลงไปในแผงวงจร IPST-SE ครั้งแรก จะเรียกว่า การอัปโหลด (upload) ปกติแล้วจะใช้คำว่า “ดาวน์โหลด” แต่สำหรับการทำงานกับซอฟต์แวร์ Arduino IDE 1.7.10 จะเรียกกระบวนการนี้ว่า อัปโหลด

ขั้นตอนการอัปโหลดโปรแกรมครั้งแรก มี 2 ขั้นตอนหลักๆ คือ การตรวจสอบตำแหน่งของพอร์ตที่ใช้ในการติดต่อระหว่างแผงวงจรหลัก IPST-SE กับซอฟต์แวร์ Arduino IDE 1.7.10 บนคอมพิวเตอร์ และขั้นตอนการตั้งค่าเพื่อใช้ในการอัปโหลดโปรแกรม



รูปที่ 1-3 การเชื่อมต่อแผงวงจร IPST-SE กับคอมพิวเตอร์เพื่อเตรียมใช้งานกับซอฟต์แวร์ Arduino IDE

1.3.1 การตรวจสอบตำแหน่งของพอร์ตอนุกรมเสมือน หรือ USB Serial port สำหรับแผงวงจร IPST-SE

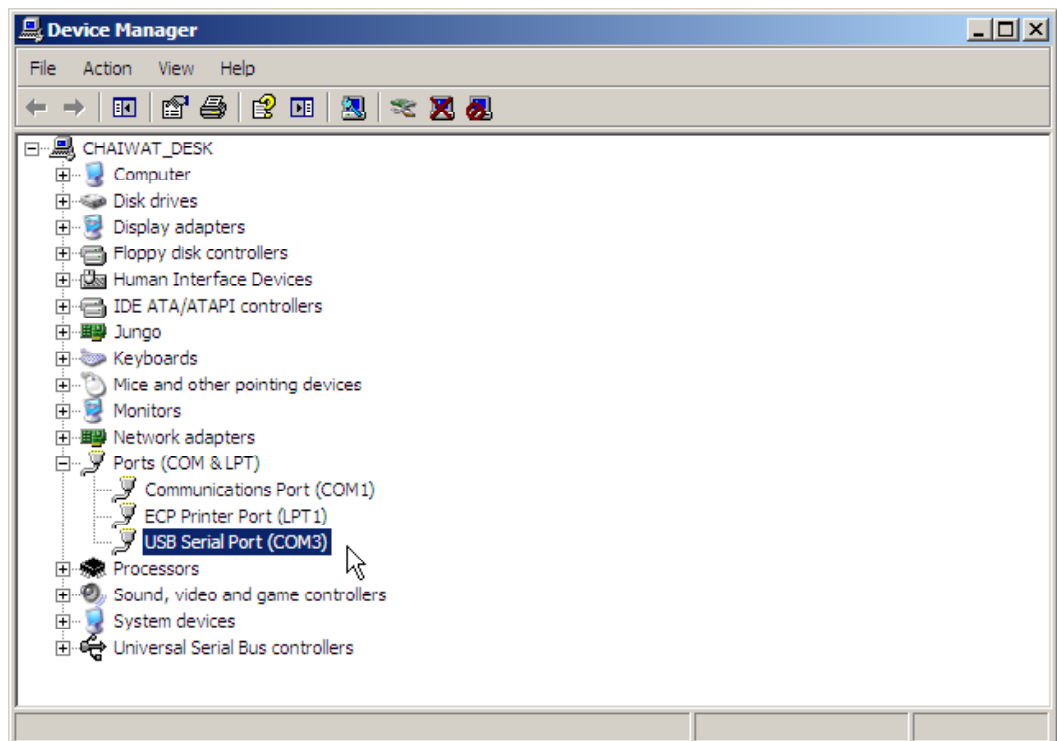
(1) เสียบสาย USB เชื่อมต่อระหว่างแผงวงจร IPST-SE กับพอร์ต USB ของคอมพิวเตอร์ เปิดสวิตช์เพื่อจ่ายไฟ รอจนกระทั่งไฟสีน้ำเงินที่ตำแหน่ง USB บนแผงวงจรควบคุมติดสว่าง ดังรูปที่ 1-3

(2) คลิกที่ปุ่ม **Start** แล้วเลือกไปที่ **Control Panel**

(3) จากนั้นดับเบิลคลิกเลือกที่ **System**

(4) เลือกไปที่แท็บ **Hardware** แล้วคลิกที่ **Device Manager**

(5) ตรวจสอบรายการฮาร์ดแวร์ที่หัวข้อ **Port** จะพบ **USB Serial port** ให้ดูว่ามีการเลือกตำแหน่งของพอร์ตอนุกรม USB Serial port ไว้ที่ตำแหน่งใด ปกติจะเป็น COM3 ขึ้นไป ให้ใช้ค่าของตำแหน่งของพอร์ตอนุกรมนี้ในการกำหนดการเชื่อมต่อกับ โปรแกรมต่อไป ตามรูปตัวอย่างจะเป็น **COM3**

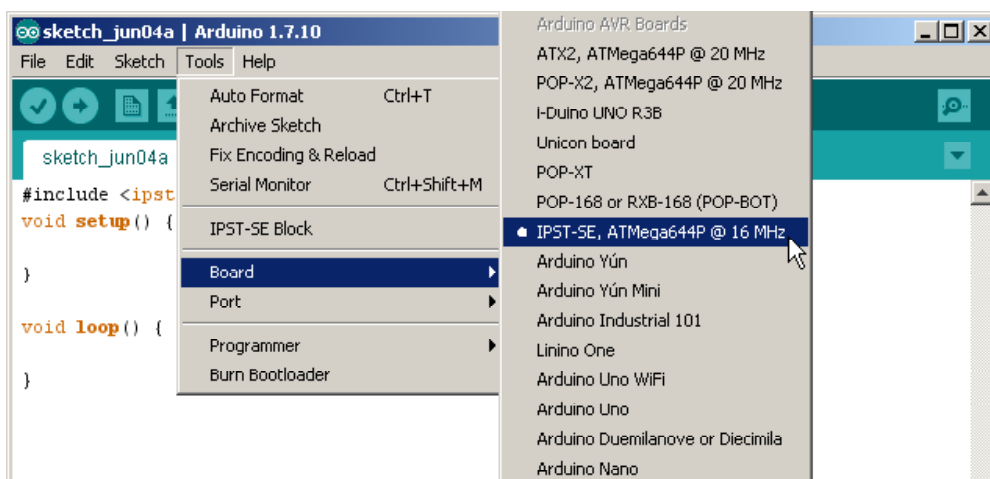


1.3.2 เชื่อมต่อแผงวงจรหลัก IPST-SE กับซอฟต์แวร์ Arduino IDE

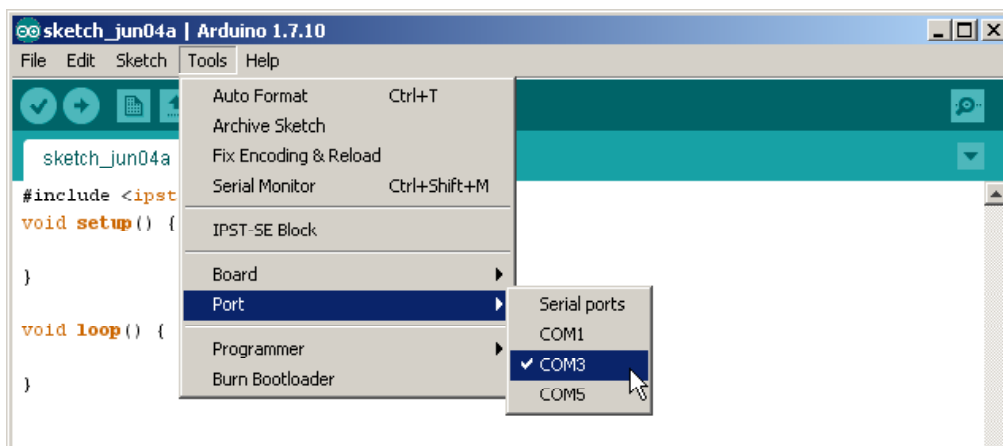
หลังจากทราบถึงตำแหน่งของพอร์ตที่เชื่อมต่อกับคอมพิวเตอร์แล้ว ในลำดับต่อไปเป็นการเชื่อมต่อเข้ากับซอฟต์แวร์ Arduino IDE 1.7.10

(1.3.2.1) เปิดโปรแกรม **Arduino IDE** รอสักครู่หนึ่ง หน้าต่างหลักของโปรแกรมจะปรากฏขึ้น การเปิดใช้งาน **Arduino IDE** ในครั้งแรกอาจใช้เวลาพอสมควร (ขึ้นอยู่กับคอมพิวเตอร์แต่ละเครื่อง)

(1.3.2.2) เลือกฮาร์ดแวร์ที่ใช้ โดยเลือกเมนู **Tools > Board > IPST-SE > ATmega644P @16MHz**



(1.3.2.3) เลือกพอร์ตติดต่อ โดยไปที่เมนู **Tools > Serial Port** เลือกตำแหน่งของพอร์ตคอมพิวเตอร์ที่ใช้ในการเชื่อมต่อกับแผงวงจร IPST-SE ในที่นี้คือ **COM3**



ขั้นตอนนี้ควรทำทุกครั้งที่เชื่อมต่อแผงวงจร IPST-SE กับพอร์ต USB ของคอมพิวเตอร์ใหม่ เพียงเท่านี้แผงวงจร IPST-SE พร้อมสำหรับการติดต่อกับซอฟต์แวร์ Arduino IDE 1.7.10 แล้ว

1.3.3 ขั้นตอนการพัฒนาโปรแกรม

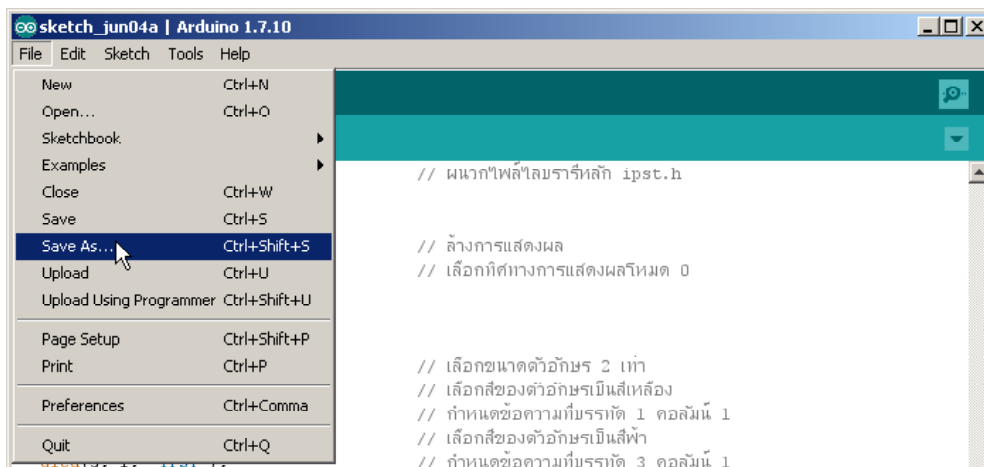
(1.3.3.1) สร้างไฟล์ใหม่ด้วยการคลิกที่ปุ่ม **New** บนแถบเครื่องมือหรือเลือกจากเมนู **File > New**

(1.3.3.2) พิมพ์โค้ดโปรแกรมต่อไปนี้

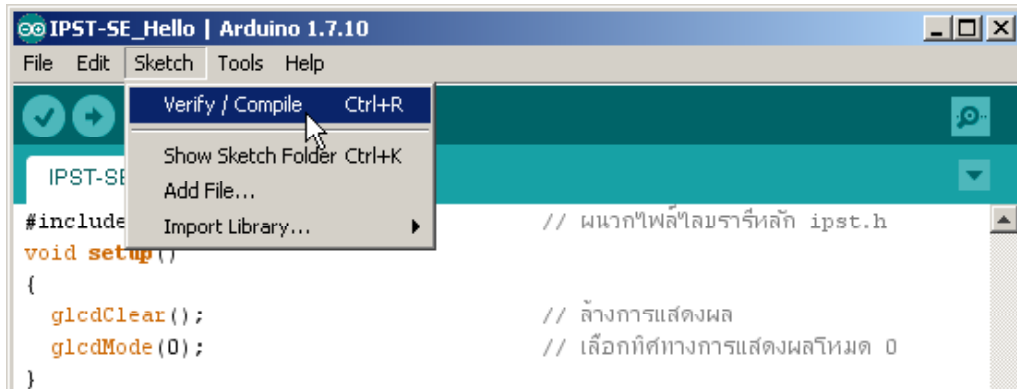
```
#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก ipst.h
void setup()
{
  glcdClear(); // ล้างการแสดงผล
  glcdMode(0); // เลือกทิศทางการแสดงผลโหมด 0
}
void loop()
{
  setTextSize(2); // เลือกขนาดตัวอักษร 2 เท่า
  setTextColor(GLCD_YELLOW); // เลือกสีของตัวอักษรเป็นสีเหลือง
  glcd(1,1,"Hello"); // กำหนดข้อความที่บรรทัด 1 คอลัมน์ 1
  setTextColor(GLCD_SKY); // เลือกสีของตัวอักษรเป็นสีฟ้า
  glcd(3,1,"IPST"); // กำหนดข้อความที่บรรทัด 3 คอลัมน์ 1
  glcd(4,1,"MicroBOX"); // กำหนดข้อความที่บรรทัด 4 คอลัมน์ 1
  setTextSize(1); // เลือกขนาดตัวอักษรปกติ
  glcd(10,2,"Secondary Education"); // กำหนดข้อความที่บรรทัด 10 คอลัมน์ 2
}
```

โปรแกรมนี้ใช้ทดสอบการทำงานเบื้องต้นของแผงวงจร *IPST-SE* โดยกำหนดให้แสดงข้อความที่จอแสดงผล ด้วยขนาดและสีของตัวอักษรที่ต่างกัน

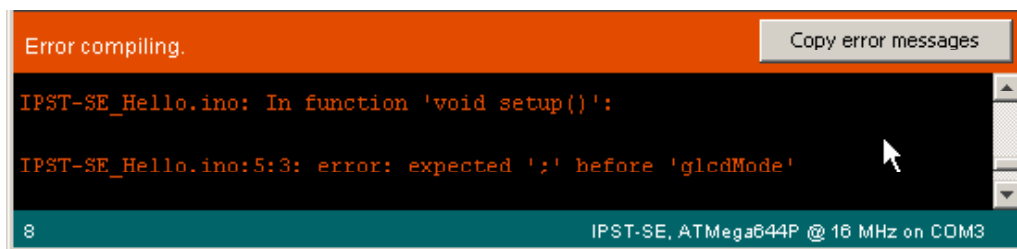
(1.3.3.3) ไปที่เมนู **File** เลือกคำสั่ง **Save As** เพื่อบันทึกไฟล์ในชื่อ **microbox_Hello** ตอนนี้จะมีไฟล์ **microbox_Hello.ino** เกิดขึ้นในโฟลเดอร์ชื่อว่า **microbox_Hello**



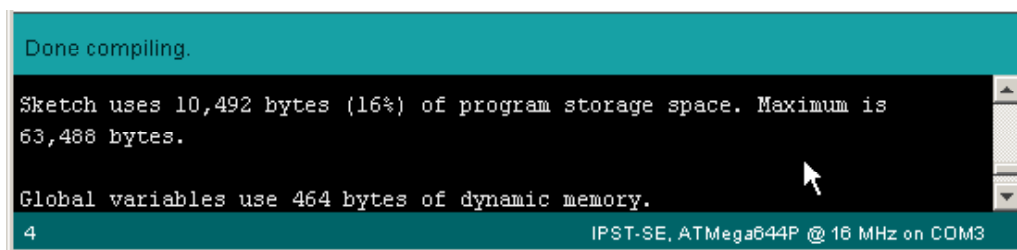
(1.3.3.4) ตรวจสอบการเขียนโปรแกรมด้วยการคลิกที่ปุ่ม **Run** ที่แถบเครื่องมือ หรือเลือกคำสั่งจากเมนู **Sketch > Verify/Compile**



หากมีความผิดพลาดเกิดขึ้นจากการคอมไพล์ จะปรากฏข้อความแจ้งความผิดพลาดในช่องแสดงสถานะและพื้นที่แสดงข้อความ ต้องทำการแก้ไขโปรแกรม




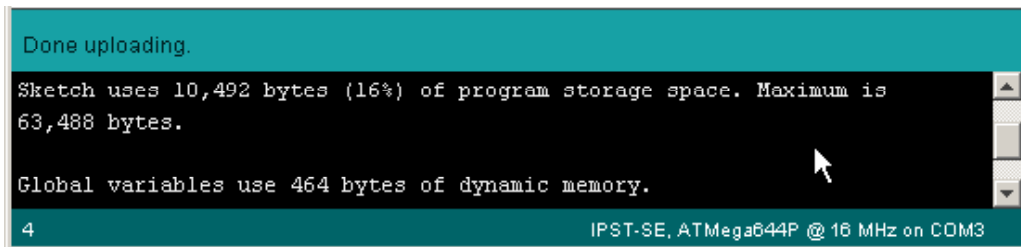
หากการคอมไพล์ถูกต้อง ที่ช่องแสดงสถานะจะแจ้งแสดงข้อความ **Done compiling**



(1.3.3.5) ต่อสาย USB เข้ากับแผงวงจร IPST-SE จากนั้นเปิดสวิตช์จ่ายไฟเลี้ยง แล้วรอให้การเชื่อมต่อกับคอมพิวเตอร์เสร็จสมบูรณ์ (ดูจากไฟแสดงผลสีน้ำเงินที่ตำแหน่ง USB ติดสว่าง)

(1.3.3.6) ตรวจสอบตำแหน่งพอร์ต แล้วเลือกฮาร์ดแวร์และตำแหน่งพอร์ตที่ทำการเชื่อมต่อให้ถูกต้องตามขั้นตอนในหัวข้อ 1.3.2

(1.3.3.7) คลิกที่ปุ่ม  **Upload to Wiring Hardware** บนแถบเครื่องมือ ถ้าทุกอย่างเป็นปกติ เมื่อทำการอัปโหลดเสร็จ จะมีข้อความแจ้งที่ช่องแสดงสถานะว่า **Done uploading**. และที่พื้นที่แสดงข้อความจะแจ้งกระบวนการและผลคอมพิวเตอร์ รวมถึงขนาดของไฟล์ผลลัพธ์ที่เกิดขึ้น



```
Done uploading.
Sketch uses 10,492 bytes (16%) of program storage space. Maximum is 63,488 bytes.
Global variables use 464 bytes of dynamic memory.
4 IPST-SE, ATmega644P @ 16 MHz on COM3
```

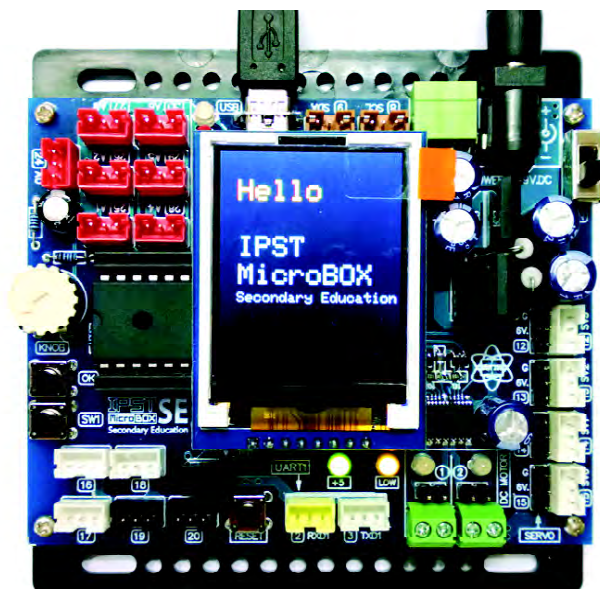
ถ้ามีข้อผิดพลาดเกิดขึ้นจะมีข้อความแจ้งเตือนในพื้นที่แสดงข้อความด้านล่าง ของหน้าต่างโปรแกรมหลัก



```
Problem uploading to board. See http://www.arduino.cc/en/Guide/... Copy error messages
avrdude: stk500v2_disable(): failed to leave programming mode
17 IPST-SE, ATmega644P @ 16 MHz on COM3
```

ซึ่งส่วนใหญ่แล้วมักเกิดจากการเลือกพอร์ตอนุกรมไม่ถูกต้อง หรือไม่ได้ต่อแผงวงจรไว้ หรือไม่ได้เปิดสวิตช์จ่ายไฟให้แก่แผงวงจร IPST-SE

(1.3.3.8) หลังจากอัปโหลดโปรแกรมแล้ว แผงวงจรหลัก IPST-SE จะทำงานทันที ได้ผลการทำงานตามรูป



1.4 การแก้ปัญหาในกรณีที่อัปโหลดโปรแกรมไม่ได้

สาเหตุ :

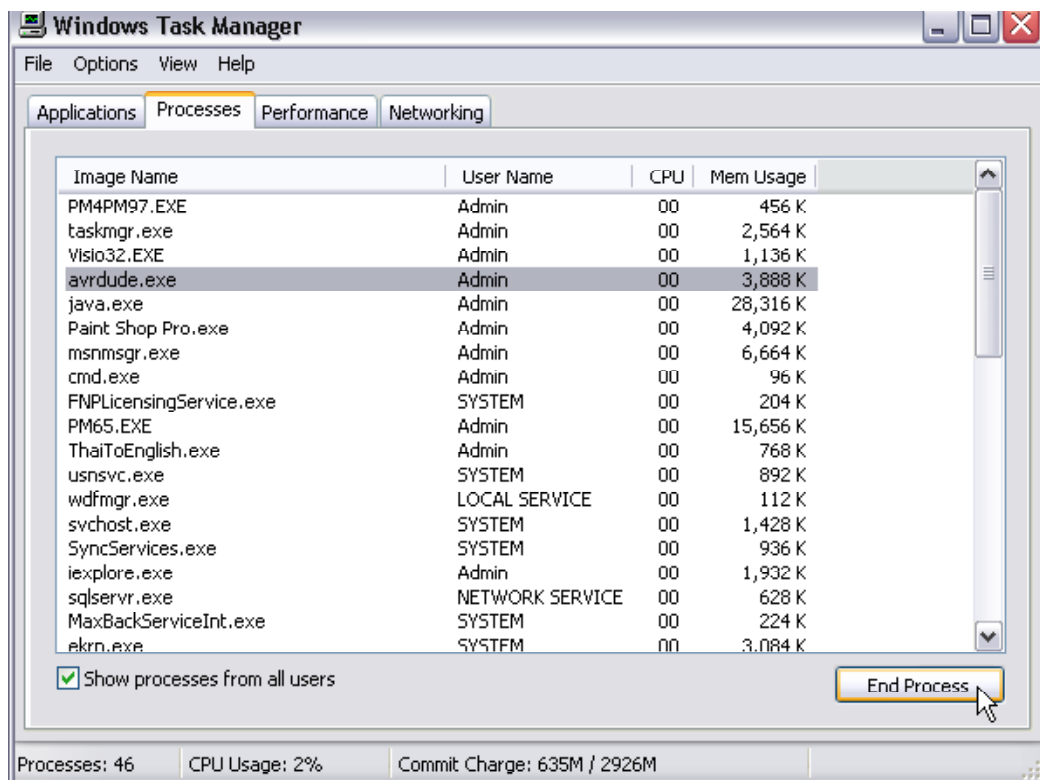
ซอฟต์แวร์ Arduino IDE ไม่สามารถติดต่อกับแผงวงจรหลัก IPST-SE ได้

ทางแก้ไข :

(1) ตรวจสอบการต่อสาย miniB-USB

(2) ตรวจสอบการเลือกพอร์ตหรือช่องเชื่อมต่อว่า ถูกต้องหรือไม่

(3) เกิดความผิดปกติขึ้นในระบบคอมพิวเตอร์ จึงต้องหยุดการทำงานในส่วนของการอัปโหลดโปรแกรม โดยกดคีย์ **Ctrl, Alt** และ **Delete** พร้อมกัน หน้าต่าง **Window Security** ปรากฏขึ้น แล้วคลิกเลือก **Task Manager** หรือในคอมพิวเตอร์บางเครื่องอาจเข้าสู่หน้าต่าง **Window Task manager** ทันที ให้เลือกแท็บ **Processes** แล้วหาชื่อไฟล์ **avrdude.exe** คลิกเลือกที่ไฟล์นั้น แล้วคลิกที่ปุ่ม **End Process**



จากนั้นซอฟต์แวร์ Arduino IDE จะกลับมาทำงานในสถานะปกติได้ ทำการจ่ายไฟให้กับบอร์ดอีกครั้ง เลือกพอร์ตเชื่อมต่อให้ถูกต้อง แล้วทำการอัปโหลดโปรแกรมอีกครั้ง

1.5 การเปิดไฟล์ตัวอย่าง

เพื่ออำนวยความสะดวกและลดการผิดพลาดในการพัฒนาโปรแกรมสำหรับผู้เริ่มต้นใช้งาน Arduino IDE จึงได้เตรียมไฟล์ตัวอย่างไว้พอสมควร

การเปิดไฟล์ตัวอย่างทำได้ง่ายมาก โดยไปที่เมนู **Help > Example > IPST-SE** จะเห็นชื่อไฟล์ให้เลือกเปิดใช้งานตามต้องการ

หรือเลือกเปิดผ่านทางคำสั่ง **Open** ซึ่งจะปรากฏหน้าต่าง Explorer ขึ้นมาเพื่อให้นักศึกษาเลือกไฟล์ เลือกไปที่ `C:/Wiring/Examples/IPST-SE` จะพบโฟลเดอร์ที่ใช้เก็บไฟล์สเก็ตจำนวนมาก

เมื่อเลือกเปิดโฟลเดอร์ที่ต้องการ จะพบไฟล์ `.ino` ซึ่งก็คือไฟล์ที่ใช้งานกับ Arduino IDE จากนั้นจะแก้ไข, คอมไพล์ รวมทั้งอัปโหลดโปรแกรมก็ทำได้ตามต้องการ

1.6 ข้อกำหนดในการแก้ไขและบันทึกไฟล์

ในกรณีที่ต้องการแก้ไขไฟล์ตัวอย่างเดิม ต้องเปิดไฟล์นั้นๆ ขึ้นมาทำการแก้ไขโค้ดโปรแกรม ตรวจสอบไวยากรณ์ด้วยการคอมไพล์ เมื่อเรียบร้อยแล้ว มีทางเลือกในการบันทึกไฟล์ 2 ทางคือ

1. บันทึกในชื่อเดิม ให้ใช้คำสั่ง **Save**

2. บันทึกในชื่อใหม่ด้วยคำสั่ง **Save As** แต่ไม่ควรบันทึกทับไฟล์เดิมที่ไม่ได้ถูกเปิดขึ้นมา เพราะจะทำให้การเชื่อมโยงไฟล์สับสน และทำให้เกิดความผิดพลาดในการเปิดใช้งานครั้งต่อไปได้ **ถ้าหากมีความต้องการบันทึกทับไฟล์เดิมที่ไม่ได้ถูกเปิดขึ้นมา จะต้องทำการลบโฟลเดอร์ของไฟล์เดิมนั้นออกไปเสียก่อน**

บทที่ 2

แนะนำ

ชุดกล่องสมองกลสำหรับเรียนรู้-ทดลองและพัฒนา โครงการวิทยาศาสตร์ด้วยไมโครคอนโทรลเลอร์

IPST-MicroBOX (SE) เป็นชุดแผงวงจรเนกประสงค์ที่ใช้อุปกรณ์ควบคุมแบบโปรแกรมได้ขนาดเล็กที่เรียกว่า “ไมโครคอนโทรลเลอร์” (microcontroller) ทำงานร่วมกับวงจรเชื่อมต่อกอมพิวเตอร์เพื่อการโปรแกรมและสื่อสารข้อมูล โดยในชุดประกอบด้วยแผงวงจรควบคุมหลัก IPST-SE ซึ่งมีไมโครคอนโทรลเลอร์เป็นอุปกรณ์หลัก, กลุ่มของแผงวงจรอุปกรณ์แสดงผลการทำงานหรืออุปกรณ์เอาต์พุต อาทิ แผงวงจรแสดงผลด้วยไดโอดเปล่งแสง 8 ดวง และแผงวงจรแสดงผลด้วยไดโอดเปล่งแสงแบบตัวเดียว รวมถึงแผงวงจรอุปกรณ์ตรวจจับสัญญาณหรือเซนเซอร์ (sensor) ซึ่งมีด้วยกันหลากหลายรูปแบบ จึงทำให้ผู้ใช้งานสามารถนำชุดกล่องสมองกล IPST-MicroBOX(SE) นี้มาใช้ในการเรียนรู้, ทดลองและพัฒนาโครงการทางวิทยาศาสตร์ที่เกี่ยวข้องกับระบบควบคุมอัตโนมัติได้อย่างสะดวกและมีประสิทธิภาพสูง

IPST-MicroBOX (SE) เริ่มต้นมีด้วยกัน 2 รุ่นคือ รุ่นมาตรฐาน 1 และ 2 ต่อมาได้มีการพัฒนาโดยบรรจุอุปกรณ์ตัวตรวจจับและอุปกรณ์แสดงผลการทำงานหรืออุปกรณ์เอาต์พุตเพิ่มเติม กลายมาเป็นรุ่นมาตรฐาน 3 ดังมีรายละเอียดขั้นต้นต่อไปนี้

1. **รุ่นมาตรฐาน 1** ในชุดนี้ประกอบด้วยแผงวงจรควบคุมหลัก IPST-SE เป็นอุปกรณ์หลักที่มีโมดูลแสดงผลกราฟิก LCD สีในตัว, แผงวงจร LED, แผงวงจรลำโพง, แผงวงจรตรวจจับสัญญาณหรือเซนเซอร์ (sensor) พื้นฐาน, และเครื่องจ่ายไฟ ทำให้นำชุด IPST-MicroBOX (SE) นี้ไปใช้ในการเรียนรู้และเขียนโปรแกรมเพื่อพัฒนาเป็นโครงการทางวิทยาศาสตร์ที่มีการควบคุมด้วยระบบอัตโนมัติโดยใช้โปรแกรมภาษา C/C++ ในเบื้องต้นได้ภายใต้งบประมาณที่เหมาะสม

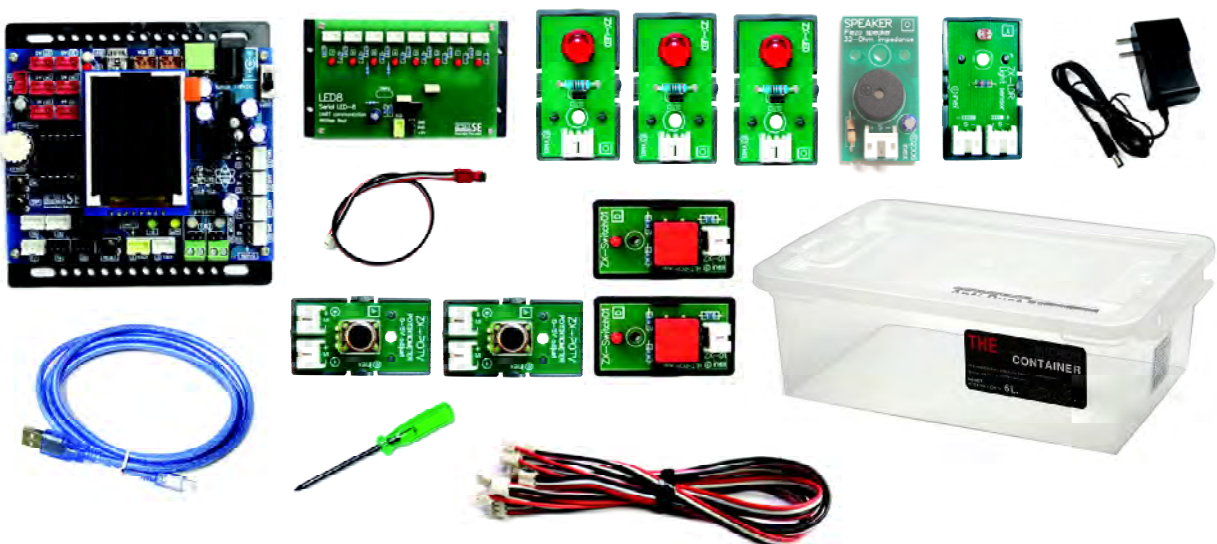
2. **รุ่นมาตรฐาน 2** ในชุดประกอบด้วยอุปกรณ์หลักเหมือนกับชุด IPST-MicroBOX (SE) รุ่นมาตรฐาน 1 มีการเพิ่มตัวตรวจจับแสงสะท้อนอินฟราเรดอีก 2 ตัว, มอเตอร์ไฟตรงพร้อมชุดเฟืองขับและชิ้นส่วนทางกลที่จำเป็น เพื่อให้ต่อยอดการเรียนการสอนและการใช้งาน IPST-MicroBOX (SE) นี้ไปสร้างเป็นหุ่นยนต์อัตโนมัติแบบโปรแกรมได้ ทั้งยังรองรับกิจกรรมการแข่งขันได้เป็นอย่างดี

3. รุ่นมาตรฐาน 3 ประกอบด้วย อุปกรณ์หลักเหมือนกับชุด IPST-MicroBOX (SE) รุ่นมาตรฐาน 2 มีการเพิ่มตัวตรวจจับอีกหลายรายการ อาทิ ตัวตรวจจับและวัดระยะทาง, ตัวตรวจจับแสงอินฟราเรด, ตัวตรวจจับสัญญาณรีโมตคอนโทรล และแผงวงจรตรวจจับเสียง นอกจากนี้ ยังเพิ่มแผงวงจรแสดงการทำงานและขับโหลดกระแสไฟฟ้าสูง อาทิ แผงวงจรแสดงผลตัวเลข 7 ส่วน 4 หลัก, แผงวงจรขับแสงอินฟราเรด และแผงวงจรขับรีเลย์ 4 ช่อง จึงทำให้ผู้ใช้งานต่อยอดการนำชุดกล่องสมองกล IPST-MicroBOX SE นี้ไปทำโครงการวิทยาศาสตร์ประยุกต์ได้หลากหลายเพิ่มมากขึ้น สอดคล้องกับแนวคิดการเรียนการสอนสมัยใหม่ที่อ้างอิงกับ STEM ศึกษาได้เป็นอย่างดีและมีประสิทธิภาพ

3.1 รายการอุปกรณ์ของชุดกล่องสมองกล IPST-MicroBOX (SE)

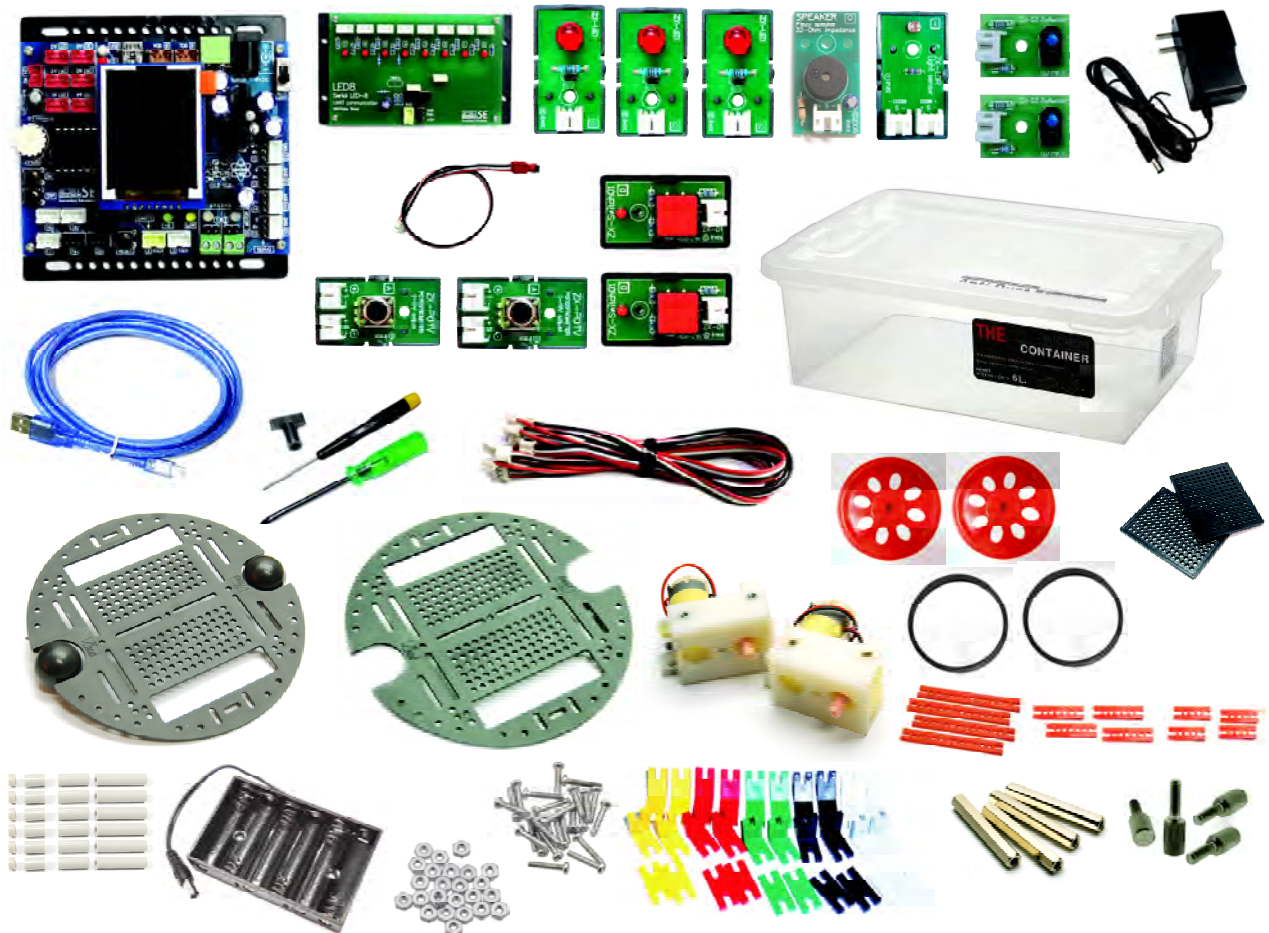
รุ่นมาตรฐาน 1 ประกอบด้วย

1. แผงวงจรควบคุมหลัก IPST-SE
2. แผงวงจร LED เดี่ยวพร้อมสายสัญญาณ 3 ชุด
3. แผงวงจร LED 8 ดวงพร้อมสายสัญญาณ
4. แผงวงจรลำโพงเปียโซพร้อมสายสัญญาณ
5. แผงวงจรสวิทช์พร้อมสายสัญญาณ 2 ชุด
6. แผงวงจรตรวจจับแสงพร้อมสายสัญญาณ
7. แผงวงจรตัวต้านทานปรับค่าได้พร้อมสายสัญญาณ 2 ชุด
8. ไอซีวัดอุณหภูมิพร้อมสายต่อ
9. อะแดปเตอร์ไฟตรง +9V 1A
10. สายเชื่อมต่อ USB-miniB
11. ซีดีรอม
12. คู่มือการทดลอง
13. กล่องบรรจุ
14. ไขควง



รุ่นมาตรฐาน 2 ประกอบด้วย รายการที่ 1 ถึง 13 ของรุ่นมาตรฐาน 1 และอุปกรณ์เพิ่มเติมดังนี้

14. แผงวงจรตรวจจับแสงสะท้อนอินฟราเรดพร้อมสายสัญญาณ 2 ชุด
15. มอเตอร์ไฟตรงพร้อมชุดเฟืองขับรุ่น BO2 อัตราทด 48:1 พร้อมสายเชื่อมต่อ 2 ตัว
16. ล้อพลาสติกกลมสำหรับชุดเฟืองขับมอเตอร์และยาง จำนวน 2 ชุด
17. แผ่นกริดขนาด 80 x 60 เซนติเมตรและ 80 x 80 เซนติเมตร จำนวน 2 ชุด
18. แผ่นฐานกลมพร้อมล้ออิสระ 1 แผ่น
19. แผ่นฐานกลมสำหรับทำโครงหุ่นยนต์ 1 แผ่น
20. ชิ้นต่อ/แท่งต่อพลาสติกและเสารองพลาสติก
21. ชุดเสารองโลหะ, นอตและสกรู
22. กะบะถ่าน AA 6 ก้อน พร้อมสายและหัวต่อป้องกันการกลับขั้วสำหรับต่อกับแผงวงจรหลัก
23. แผ่นทดสอบการเคลื่อนที่ตามเส้นของหุ่นยนต์



ดังนี้

รุ่นมาตรฐาน 3 ประกอบด้วย รายการที่ 1 ถึง 23 ของรุ่นมาตรฐาน 1 และ 2 มีอุปกรณ์เพิ่มเติม

24. แผงวงจร LED ตัวเลข 7 ส่วน 4 หลักพร้อมสายสัญญาณ
25. แผงวงจร LED อินฟราเรดพร้อมสายสัญญาณ 2 ชุด
26. แผงวงจรขับรีเลย์ 4 ช่องพร้อมสายสัญญาณ 4 เส้น
27. แผงวงจรโฟโต้ทรานซิสเตอร์สำหรับตรวจจับแสงอินฟราเรดพร้อมสายสัญญาณ 2 ชุด
28. แผงวงจรตรวจจับเสียงพร้อมสายสัญญาณ
29. แผงวงจรโมดูลรับแสงอินฟราเรด 38kHz พร้อมสายสัญญาณ
30. โมดูลตรวจจับและวัดระยะทางด้วยแสงอินฟราเรดพร้อมสายสัญญาณ
31. รีโมทคอนโทรลอินฟราเรดที่ใช้รหัสข้อมูลในรูปแบบของโซนนี่
32. อะแดปเตอร์ไฟตรง +12V 1A สำหรับแผงวงจรขับรีเลย์

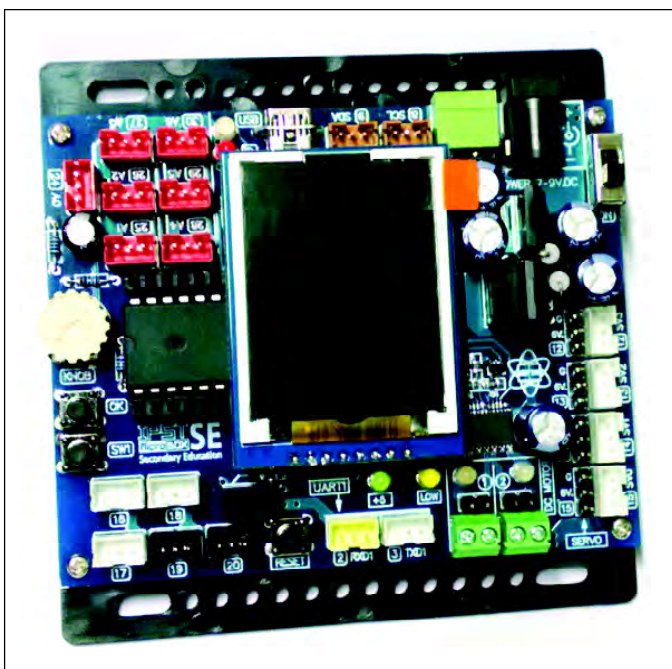


2.2 คุณสมบัติของแผงวงจรหลัก IPST-SE

มีหน้าตาของแผงวงจรแสดงในรูปที่ 2-1 ส่วนรูปที่ 2-2 แสดงรายละเอียดที่สำคัญของแผงวงจร IPST-SE ที่ควรทราบเพื่อใช้ประโยชน์ในการอ้างอิงเมื่อทำการทดลอง ส่วนคุณสมบัติโดยสรุปของ IPST-SE เป็นดังนี้

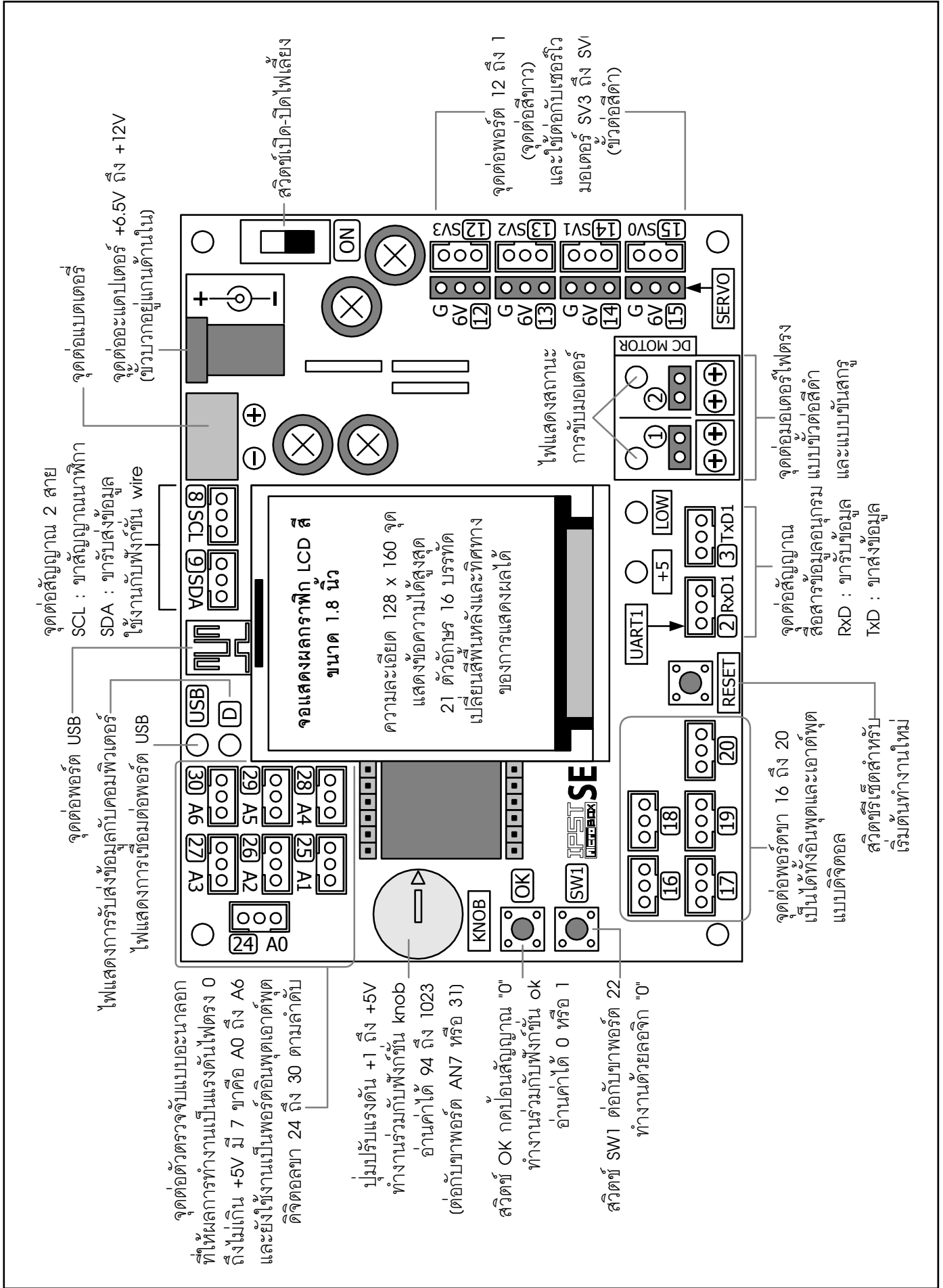
- ใช้ไมโครคอนโทรลเลอร์ 8 บิตเบอร์ ATmega644P ของ Atmel รองรับโปรแกรมควบคุมที่พัฒนาจากภาษาแอสเซมบลี, เบสิก และ C *โดยในที่นี้จะเน้นไปที่โปรแกรมภาษา C/C++* โดยภายในมีโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิทัล ความละเอียด 10 บิต ให้ค่าของข้อมูลในช่วง 0 ถึง 1,023 จึงนำมาต่อกับแผงวงจรตรวจจับที่ให้ผลการทำงานเป็นแรงดันไฟฟ้าได้ง่าย มีหน่วยความจำโปรแกรมแบบแฟลชมากถึง 64 กิโลไบต์ โปรแกรมใหม่ได้ 10,000 ครั้ง มีหน่วยความจำข้อมูลอีพ롬 512 ไบต์ และหน่วยความจำข้อมูลแรม 1 กิโลไบต์

- สัญญาณนาฬิกาหลัก 16MHz จากคริสตอล
- มีจุดต่อพอร์ต USB สำหรับดาวน์โหลดโปรแกรมและสื่อสารข้อมูลกับคอมพิวเตอร์
- มีสวิตช์ RESET การทำงาน
- มีจุดต่อพอร์ตแบบ 3 ขา (ขาไฟเลี้ยง, สัญญาณ และกราวด์) จำนวน 20 จุด แบ่งเป็นขาพอร์ตดิจิทัล 13 จุด (ขาพอร์ตหมายเลข 2, 3, 8, 9, 12 ถึง 20) และขาพอร์ตแบบดิจิทัลหรืออะนาลอก (กำหนดได้) 7 จุด (หากใช้เป็นขาอินพุตอะนาลอกเป็นขา A0 ถึง A6 และถ้าใช้เป็นขาพอร์ตดิจิทัลเป็นขาพอร์ตหมายเลข 24 ถึง 30)



รูปที่ 2-1 แผงวงจรหลัก IPST-SE มีจอแสดงผลกราฟิกสีความละเอียด 128 x 160 จุด แสดงตัวอักษรขนาดมาตรฐานได้มากถึง 21 ตัวอักษร 16 บรรทัด อัปโหลดโปรแกรมผ่านพอร์ต USB

รูปที่ 2-2 แสดงส่วนประกอบที่ควรทราบของแผงวงจรหลัก IPST-SE

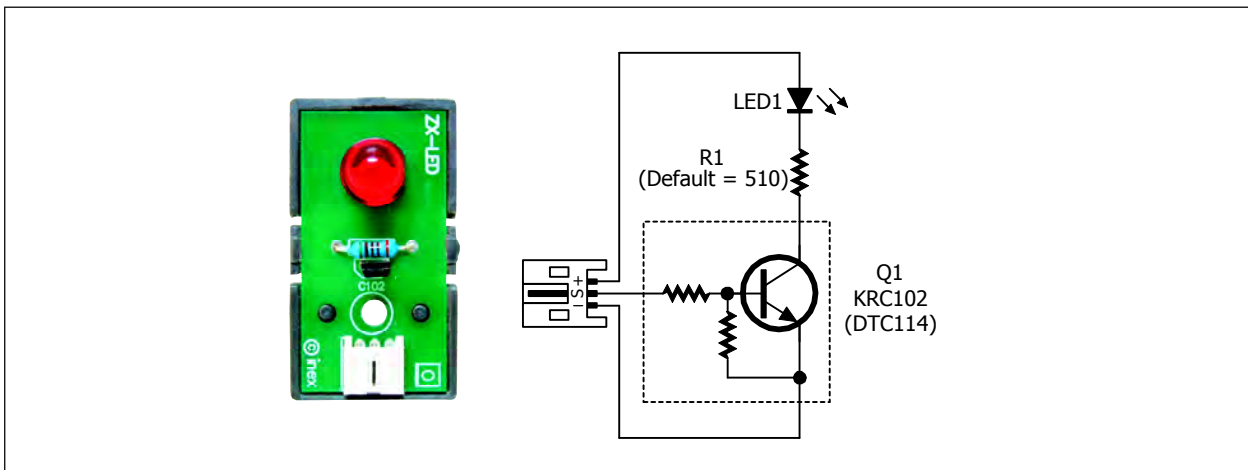


- มีจุดต่อระบบบัส 2 สาย (I²C) เพื่อขยายระบบ
- มีจุดต่อสำหรับสื่อสารข้อมูลอนุกรมหรือ UART (Universal Asynchronous Receiver Transmitter) เพื่อขยายระบบ
- ใช้ไฟเลี้ยงในย่าน +6.5 ถึง +12V กระแสไฟฟ้า 1,500mA กรณีต่อใช้งานมอเตอร์ไฟตรง และเซอร์โวมอเตอร์ร่วมด้วย หรือ 500mA กรณีไม่ใช้งานมอเตอร์ บนแผงวงจรหลัก IPST-SE มีวงจรควบคุมแรงดันคงที่ +5V จึงนำไปจ่ายให้กับแผงวงจรต่อพ่วงอื่นๆ รวมทั้งแผงวงจรตัวตรวจจับด้วย
- มีจุดต่อไฟเลี้ยง (DC INPUT) ผ่านทางจุดต่อแบบหัวเสียบป้องกันการต่อกลับขั้วและแจ๊กอะแดปเตอร์ รับไฟเลี้ยงได้ตั้งแต่ 7.2V ถึง +12V โดยมีสวิทช์เปิด-ปิดเพื่อตัดต่อไฟเลี้ยงแก่แผงวงจร พร้อมไฟแสดงสถานะไฟเลี้ยง +5V และมีวงจรแจ้งสถานะแบตเตอรี่อ่อน (LOW) ด้วย LED สีเหลือง ในกรณีที่ใช้แหล่งจ่ายไฟเป็นแบตเตอรี่ โดยกำหนดระดับแรงดันไว้ที่ +7V
- มีวงจรควบคุมไฟเลี้ยงคงที่ +5V 2A แบบสวิทช์สำหรับรักษาระดับไฟเลี้ยงให้แก่ไมโครคอนโทรลเลอร์
- มีวงจรขับมอเตอร์ไฟตรง 2 ช่อง พร้อมไฟแสดงผล
- มีจุดต่อขาพอร์ตของไมโครคอนโทรลเลอร์สำหรับขับเซอร์โวมอเตอร์ 4 ช่องคือ จุดต่อ 15, 14, 13 และ 12 (เรียงตามลำดับ SERVO0, SERVO1, SERVO2 และ SERVO3)
- มีโมดูลแสดงผลแบบกราฟิกสี ความละเอียด 128 x 160 จุด แสดงภาพกราฟิกลายเส้นและพื้นสี (ไม่รองรับไฟล์รูปภาพใดๆ) พร้อมไฟส่องหลัง แสดงผลเป็นตัวอักษรขนาดปกติ (5x7 จุด) ได้ 21 ตัวอักษร 16 บรรทัด (21 x 16)
- มีสวิทช์กดติดปล่อยดับใช้งานอิสระ 1 ตัว คือ SW1 ซึ่งต่อกับขาพอร์ตหมายเลข 22
- มีสวิทช์กดติดปล่อยดับชื่อ สวิทช์ OK ซึ่งต่อร่วมกับตัวต้านทานปรับค่าได้ชื่อ KNOB ซึ่งเชื่อมต่อไปยังขาพอร์ตดิจิทัลหมายเลข 31 (หรืออินพุตอะนาลอก A7) ทำให้อ่านค่าสัญญาณดิจิทัลและอะนาลอกได้ในขาพอร์ตเดียวกัน
- มีจุดต่อ ISP สำหรับอัปเกรดหรือกู้เฟิร์มแวร์ โดยใช้ชุดโปรแกรมแบบ ISP เพิ่มเติม (แนะนำเครื่องโปรแกรม AVR-ISP mark II ของ Atmel)

2.3 คุณสมบัติของชุดอุปกรณ์เอาต์พุต

2.3.1 แผงวงจรไฟแสดงผล : ZX-LED (มีในชุดมาตรฐาน 1 ถึง 3)

ใช้ LED ขนาด 8 มิลลิเมตร ต้องการลอจิก “1” ในการขับให้สว่าง มีวงจรแสดงในรูปที่ 2-3



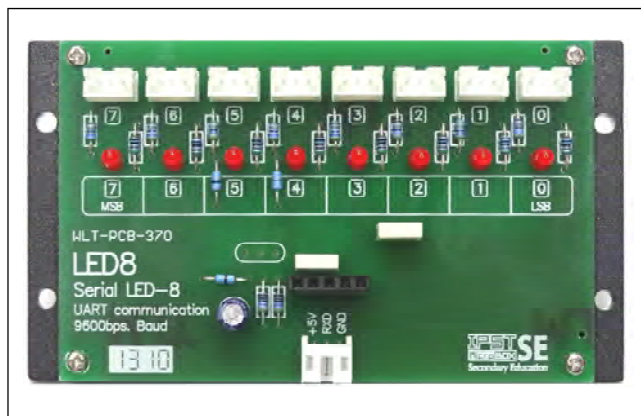
รูปที่ 2-3 รูปร่างและวงจรของแผงวงจรไฟแสดงผล ZX-LED ที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX(SE)

2.3.2 แผงวงจรไฟแสดงผล 8 ดวง : ZX-LED8 (มีในชุดมาตรฐาน 1 ถึง 3)

เป็นแผงวงจรที่มี LED ขนาด 3 มิลลิเมตรสำหรับแสดงผล 8 ดวง พร้อมจุดต่อพ่วงเอาต์พุตเพื่อนำไปใช้ในการขับรีเลย์ได้ด้วย โดยแผงวงจร ZX-LED8 นี้จะต่อเข้ากับขาพอร์ตโคของแผงวงจร IPST-SE ก็ได้ โดยใช้ขาพอร์ตเพียงขาเดียวในการควบคุมและขับ LED ให้ติดดับตามที่ต้องการ ได้พร้อมกันถึง 8 ดวง มีหน้าตาของแผงวงจรแสดงในรูปที่ 2-4

ในแผงวงจร ZX-LED8 ใช้การติดต่อกับแผงวงจรหลัก IPST-SE ในแบบอนุกรม ร่วมกับคำสั่งทางซอฟต์แวร์ ผู้พัฒนาโปรแกรมสามารถเขียนโปรแกรมให้ ZX-LED8 ติดดับได้ตั้งแต่ 1 ถึง 8 ตัว หรือจะเขียนโปรแกรมให้ทำงานเป็นไฟวิ่งได้ตั้งแต่ 1 ถึง 8 ดวงเช่นกัน

ที่ด้านบนของแผงวงจร ZX-LED8 มีจุดต่อซึ่งต่อพ่วงมาจาก LED ทำงานที่ลอจิก “1” มีระดับแรงดันไฟตรงขาออกประมาณ +5V จึงใช้สัญญาณจากจุดนี้ไปต่อกับวงจรขับโหลดกระแสไฟฟ้าสูง อาทิ แผงวงจรขับรีเลย์ ได้ทันทีโดยไม่ต้องเขียนโปรแกรมควบคุมเพิ่มเติม



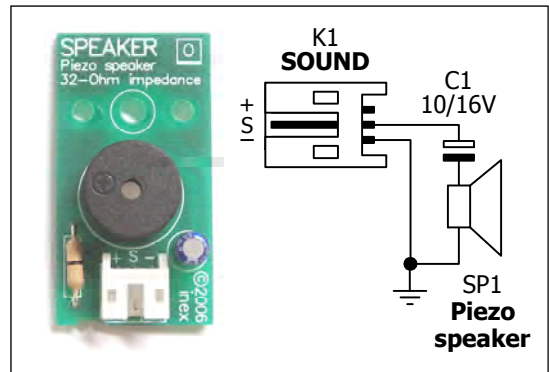
รูปที่ 2-4 รูปร่างของแผงวงจรไฟแสดงผล ZX-LED ที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX(SE)

2.3.3 แผงวงจรลำโพงเปียโซ :

ZX-SPEAKER (มีในชุดมาตรฐาน 1 ถึง 3)

มีวงจรและหน้าตาของแผงวงจรแสดงในรูปแบบที่ 2-5 คุณสมบัติทางเทคนิคที่สำคัญมีดังนี้

- ใช้ลำโพงเปียโซ มีอิมพีแดนซ์ 32Ω
- มีค่าความถี่เรโซแนนซ์ในย่าน 1 ถึง 3kHz

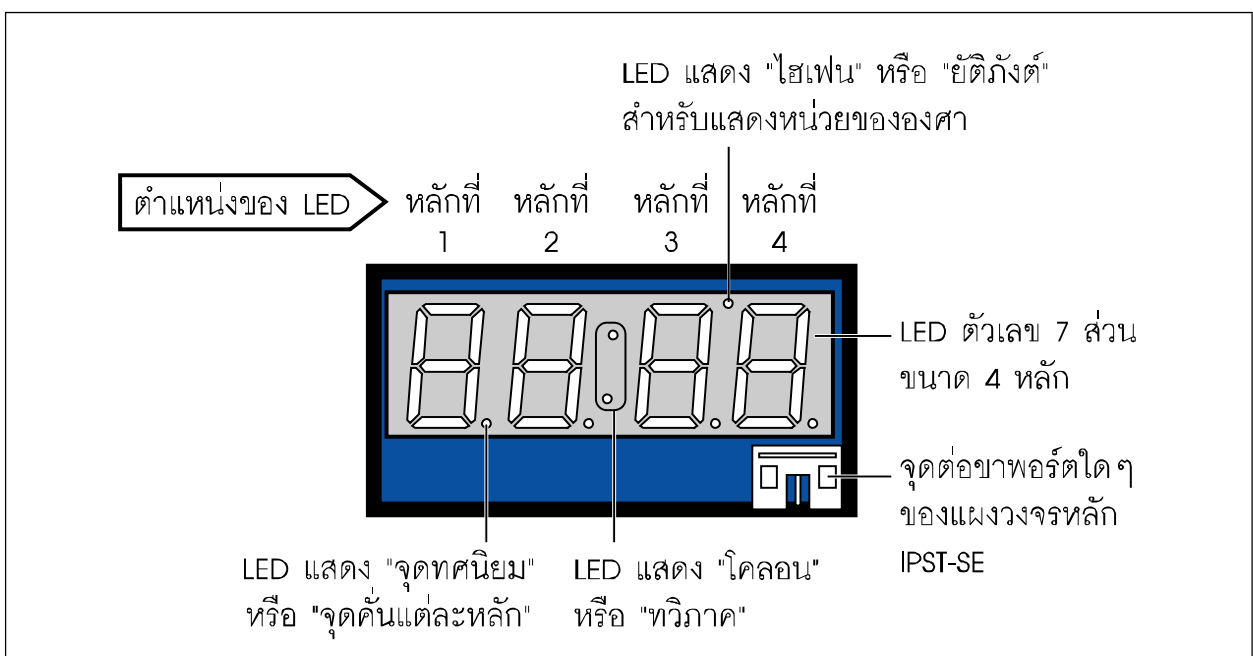


รูปที่ 2-5 วงจรของแผงวงจรลำโพงเปียโซ ZX-SPEAKER

2.3.4 แผงวงจร LED ตัวเลข 7 ส่วน 4 หลัก : DSP-4S (มีเฉพาะในชุดมาตรฐาน 3)

มีหน้าตาของแผงวงจรแสดงในรูปแบบที่ 2-6 คุณสมบัติทางเทคนิคที่สำคัญมีดังนี้

- ใช้แสดงผลในรูปแบบของตัวเลข 4 หลัก, สัญลักษณ์ และเครื่องหมายวรรคตอน เช่น โคลอน (สำหรับแสดงเวลา), ไฮเฟน (สำหรับแสดงค่าของหน่วยของศา)
- ใช้ LED ตัวเลข 7 ส่วน 4 หลักแบบแคโทดร่วม เลือกให้แสดงผลแยกเป็นหลักหรือรวมกันก็ได้ด้วยการเขียนคำสั่งสำหรับควบคุมการทำงาน
- มีจุดต่อขาพอร์ต 1 จุด เพื่อติดต่อกับแผงวงจรหลัก IPST-SE ในแบบอนุกรม
- เชื่อมต่อกับแผงวงจรหลัก IPST-SE ได้พร้อมกันสูงสุด 16 แผง



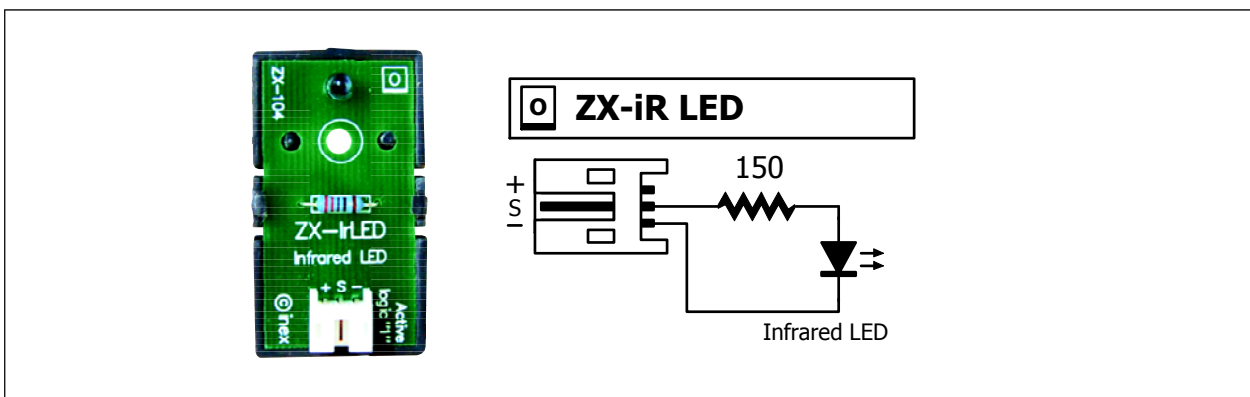
รูปที่ 2-6 รูปร่างของ DSP-4S แผงวงจรแสดงผล LED ตัวเลข 7 ส่วน 4 หลัก

2.3.5 แผงวงจร LED อินฟราเรด : ZX-IrLED (มีเฉพาะในชุดมาตรฐาน 3)

ใช้ LED เปล่งแสงอินฟราเรด 3 มม. มีวงจรและหน้าตาของแผงวงจรแสดงในรูปที่ 2-7 ใช้งาน
ได้ 2 แบบคือ

1. **ส่งแบบต่อเนื่อง** ทำงานเมื่อได้รับลอจิก "1" ใช้กับแผงวงจรตรวจจับแสงอินฟราเรดที่ใช้โฟโตทรานซิสเตอร์เพื่อวัดระดับความเข้มของแสงอินฟราเรดที่ส่งออกไป

2. **ส่งแบบสัญญาณความถี่** โดยผสมสัญญาณพาห้ความถี่ 38kHz ในกรณีนี้จะใช้งานร่วมกับแผงวงจร โมดูลรับแสงอินฟราเรด 38kHz (ZX-IRM) เพื่อตรวจสอบการรับสัญญาณ

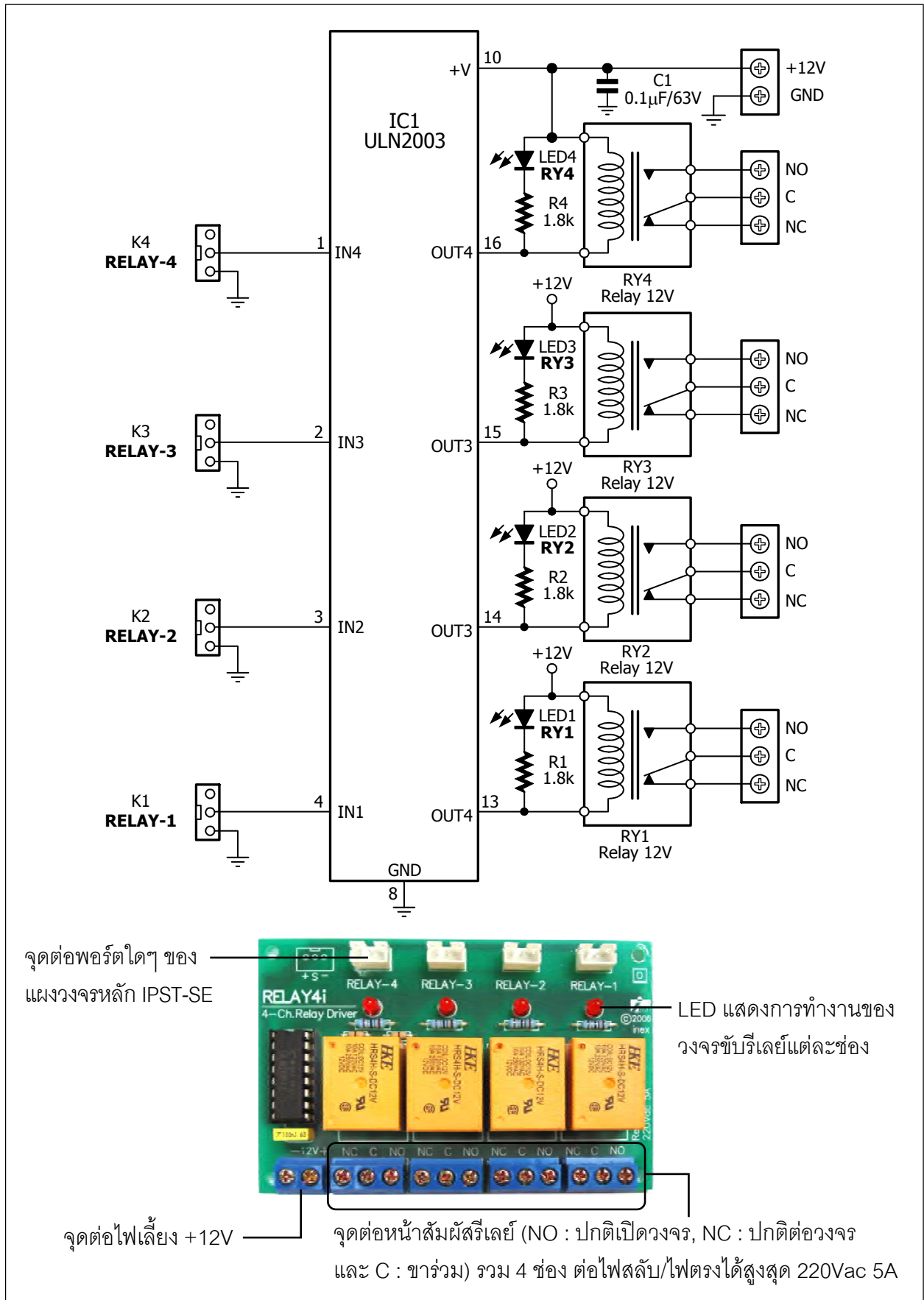


รูปที่ 2-7 รูปร่างและวงจรของแผงวงจรถูกำเนิดแสงอินฟราเรดที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX (SE) มาตรฐาน 3

2.3.6 แผงวงจรขั้วรีเลย์ 4 ช่อง : Relay-4i (มีเฉพาะในชุดมาตรฐาน 3)

มีวงจรและหน้าตาของบอร์ดแสดงในรูปที่ 2-8 มีคุณสมบัติโดยสรุปดังนี้

- ใช้ไอซีขับโวลตกระแสสูงเบอร์ ULN2003 บนบอร์ดจัดวงจรเพื่อขั้วรีเลย์ 12V 4 ช่อง
- ใช้ไฟเลี้ยง +12V แยกต่างหาก
- รับสัญญาณลอจิก "1" จากไมโครคอนโทรลเลอร์หรือวงจรขับจากภายนอกในการกระตุ้นให้รีเลย์ทำงาน
- มีไฟแสดงการทำงานของรีเลย์
- จุดต่อหน้าสัมผัสรีเลย์เป็นแบบขันสกรู ทำให้สามารถต่อใช้งานได้อย่างสะดวก
- อัตราทนไฟของหน้าสัมผัสรีเลย์ 220Vac 5A สามารถรองรับโหลดได้ไม่เกิน 300 วัตต์



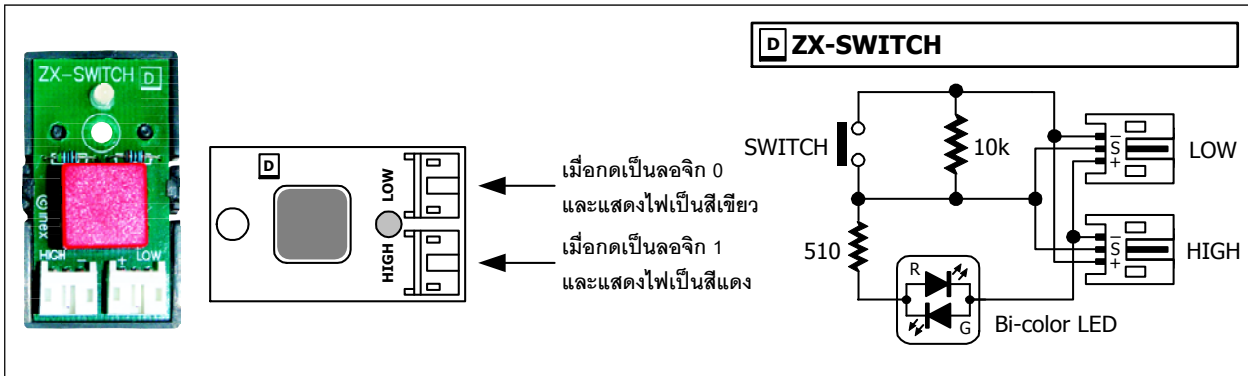
รูปที่ 2-8 วงจรสมบูรณของแผงวงจรขั้วรีเลย์ 4 ช่อง

2.4 คุณสมบัติของชุดอุปกรณ์ตรวจจับสัญญาณ

2.4.1 แผงวงจรสวิตช์ : ZX-SWITCH01 (มีในชุดมาตรฐาน 1 ถึง 3)

มีวงจรแสดงในรูปที่ 2-9 ประกอบด้วยสวิตช์พร้อมไฟแสดงผล

ให้เอาต์พุตคือ หากมีการกดสวิตช์ จะส่งลอจิก “0” (ระดับแรงดัน 0V) และไฟสีเขียวติด



รูปที่ 2-9 รูปร่างและวงจรของแผงวงจรสวิตช์ที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX (SE)

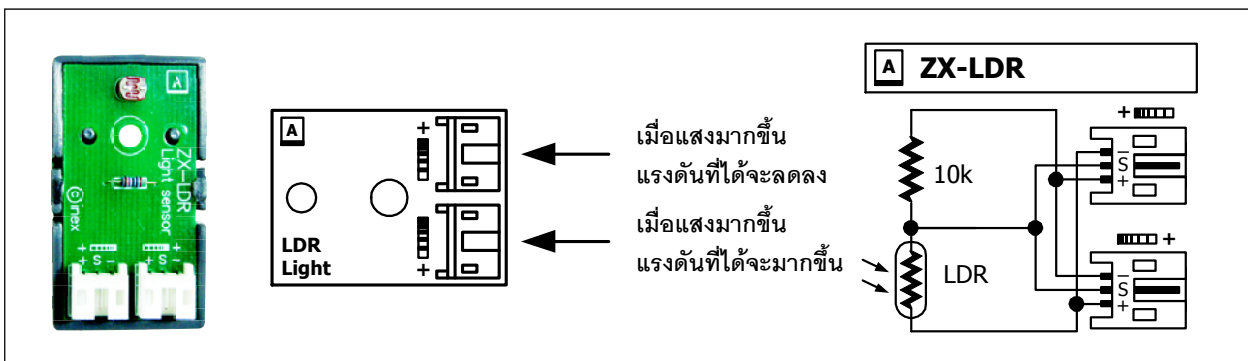
2.4.2 แผงวงจรตรวจจับแสง : ZX-LDR (มีในชุดมาตรฐาน 1 ถึง 3)

ใช้ตรวจจับแสงสว่าง เลือกเอาต์พุตได้ 2 แบบคือ

+ แรงแดันเอาต์พุตเพิ่ม เมื่อแสงตกกระทบ

แรงแดันเอาต์พุตลดลง เมื่อแสงตกกระทบ

มีวงจรและรูปร่างของแผงวงจรแสดงในรูปที่ 2-10

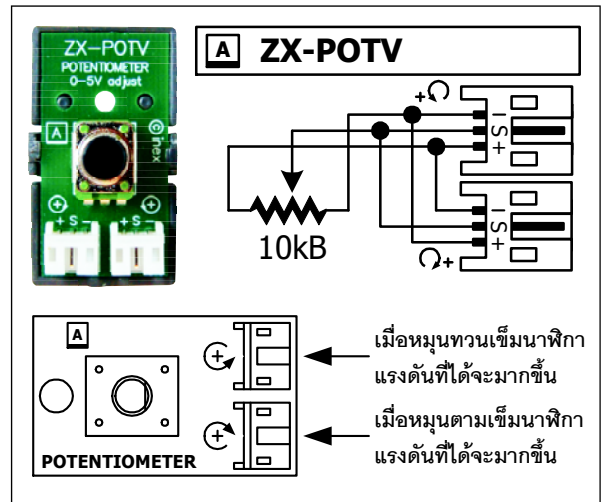


รูปที่ 2-10 รูปร่างและวงจรของแผงวงจรตรวจจับแสงที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX (SE)

2.4.3 แผงวงจรตัวต้านทานปรับค่าได้แบบ แกนหมุน: ZX-POTV POTENTIOMETER (มีในชุดมาตรฐาน 1 ถึง 3)

ใช้กำหนดแรงดัน 0 ถึง +5V ตามการหมุนแกนนำไปใช้วัดค่ามุมและระยะทางได้ มีเอาต์พุต 2 แบบคือ แรงดันมากขึ้นเมื่อหมุนทวนเข็มนาฬิกาหรือ ตามเข็มนาฬิกา

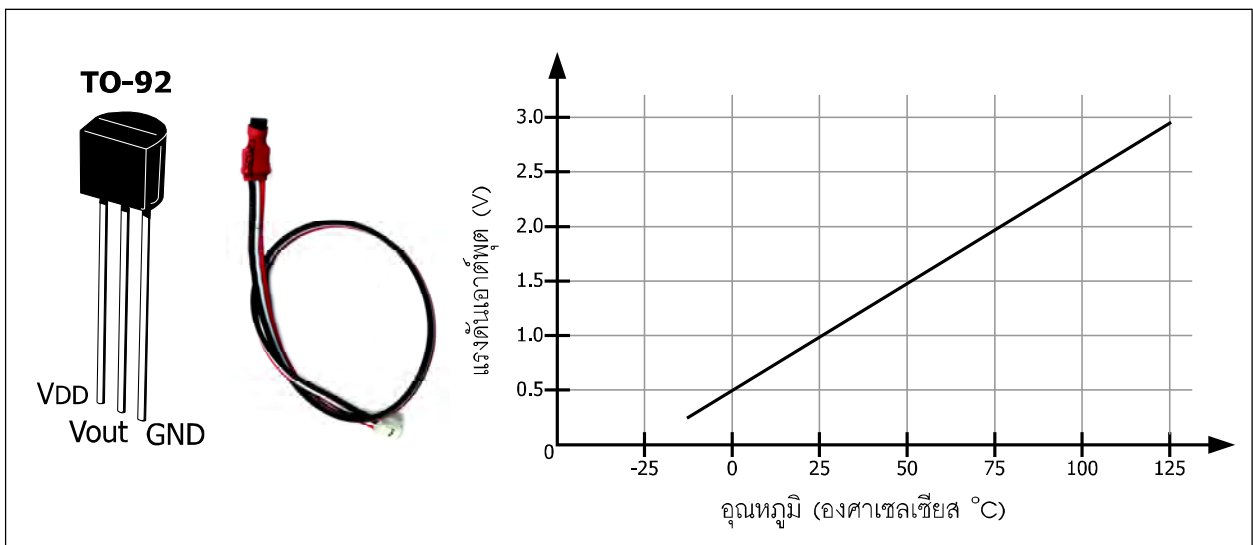
มีวงจรและหน้าตาของแผงวงจรแสดงในรูปที่ 2-11



รูปที่ 1-11 แสดงรูปร่าง, วงจร และการทำงานของแผงวงจรตัวต้านทานปรับค่าได้แบบแกนหมุน

2.4.4 ไอซีวัดอุณหภูมิ MCP9701 (มีในชุดมาตรฐาน 1 ถึง 3)

เป็นอุปกรณ์ตรวจจับและวัดอุณหภูมิที่ให้ผลการทำงานเป็นแรงดันไฟฟ้าแบบเชิงเส้น รับรู้การเปลี่ยนแปลงของอุณหภูมิภายในเวลาไม่ถึง 2 วินาที เชื่อมต่อกับอินพุตอะนาล็อก A0 ถึง A6 ของแผงวงจรหลัก IPST-SE ได้ทันที ในรูปที่ 2-12 แสดงการจัดขาและกราฟคุณสมบัติของไอซีวัดอุณหภูมิเบอร์ MCP9701



รูปที่ 2-12 การจัดขาของ MCP9701, หน้าตาเมื่อต่อสายสัญญาณพร้อมใช้งานและกราฟคุณสมบัติ

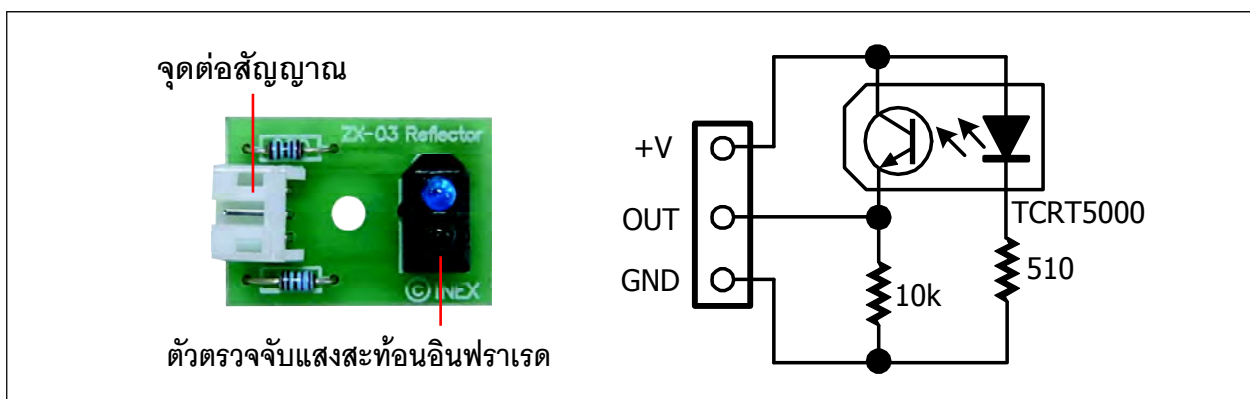
คุณสมบัติทางเทคนิคของ MCP9701 ที่ควรทราบ

- เป็น ไอซีวัดอุณหภูมิในกลุ่มเทอร์มิสเตอร์แบบแอคทีฟที่ให้ผลการทำงานแบบเชิงเส้น
- ย่านวัด -40 ถึง +125 องศาเซลเซียส
- ผลการวัดอ้างอิงกับหน่วยขององศาเซลเซียสโดยตรง
- ความผิดพลาดเฉลี่ย ± 2 องศาเซลเซียส
- ย่านไฟเลี้ยง +3.1 ถึง +5.5V กินกระแสไฟฟ้าเพียง 6 μ A ใช้แบตเตอรี่เป็นแหล่งจ่ายไฟได้
- ค่าแรงดันเอาต์พุต 500mV (ที่ 0°C) ถึง 2.9375V (ที่ 125°C)
- ค่าแรงดันเอาต์พุตต่อการเปลี่ยนแปลงอุณหภูมิ 19.5mV/°C ใช้งานกับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัลความละเอียดตั้งแต่ 8 บิตได้ โดยมีความคลาดเคลื่อนต่ำ
- ไม่ต้องการอุปกรณ์ภายนอกต่อเพิ่มเติมเพื่อชดเชยการทำงาน

2.4.5 แผงวงจรตรวจจับแสงสะท้อนอินฟราเรด : ZX-03 (มีในชุดมาตรฐาน 2 และ 3)

มีวงจรและหน้าตาของแผงวงจรแสดงในรูปที่ 2-13 เป็นแผงวงจรที่ใช้ในการตรวจสอบการสะท้อนของแสงอินฟราเรดของตัวตรวจจับแสงสะท้อนอินฟราเรดซึ่งรวมตัวส่งและตัวรับไว้ในตัวเดียวกัน โดยตัวตรวจจับแสงสะท้อนอินฟราเรดที่นำมาใช้คือ TCRT5000

เมื่อจ่ายไฟเลี้ยง LED อินฟราเรดภายในตัวโมดูล TCRT5000 จะเปล่งแสงออกมาตลอดเวลา ส่วนตัวรับซึ่งเป็นโฟโตทรานซิสเตอร์จะได้รับแสงอินฟราเรดจากการสะท้อนกลับ โดยปริมาณของแสงที่ได้รับจะมากหรือน้อยขึ้นอยู่กับว่ามีวัตถุมาทาบขวางหรือไม่ และวัตถุนั้นมีความสามารถในการ



รูปที่ 2-13 หน้าตาและวงจรของแผงวงจรตรวจจับแสงสะท้อนอินฟราเรดที่ใช้ในชุด IPST-MicroBOX (SE) รุ่นมาตรฐาน 2 และ 3

สะท้อนแสงอินฟราเรดได้ดีเพียงไร ซึ่งขึ้นกับลักษณะพื้นผิวและสีของวัตถุ โดยวัตถุสีขาวผิวเรียบจะสะท้อนแสงอินฟราเรดได้ดี ทำให้ตัวรับแสงอินฟราเรดได้รับแสงสะท้อนมาก ส่งผลให้แรงดันที่เอาต์พุตของวงจรสูงตามไปด้วย ในขณะที่วัตถุสีดำสะท้อนแสงอินฟราเรดได้น้อย ทำให้ตัวรับอินฟราเรดส่งแรงดันออกมาต่ำ ด้วยคุณสมบัติดังกล่าวจึงนิยมนำแผงวงจรตรวจจับแสงสะท้อนอินฟราเรดนี้มาใช้ในการตรวจจับพื้นหรือเส้น โดยต้องติดตั้งไว้ด้านล่างของโครงหุ่นยนต์

เนื่องจากแผงวงจรตรวจจับแสงสะท้อนอินฟราเรด ZX-03 ให้ผลการทำงานเป็นแรงดันไฟตรง ดังนั้นในการใช้งานกับแผงวงจรหลัก IPST-SE จึงต้องต่อเข้ากับช่องอินพุตอะนาล็อก (A0 ถึง A6) ของแผงวงจรหลัก จากนั้นใช้ความรู้จากการอ่านค่าสัญญาณอะนาล็อกเพื่ออ่านค่าจากแผงวงจรตรวจจับแสงสะท้อนอินฟราเรดนำไปสู่การตรวจจับเส้นต่อไป

2.4.5 แผงวงจรตรวจจับเสียง : ZX-SOUND (มีในชุดมาตรฐาน 3)

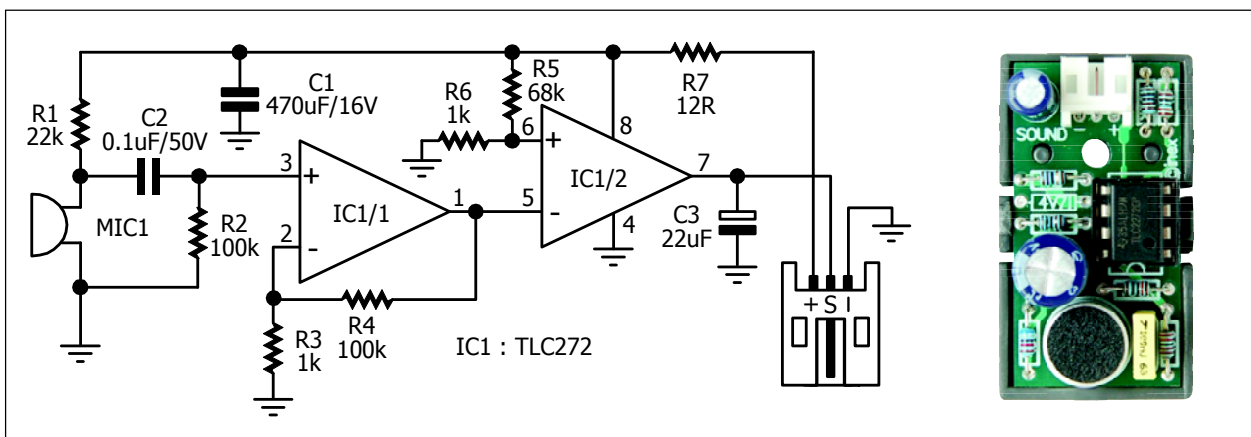
ใช้ตรวจจับการเปลี่ยนแปลงระดับเสียง เช่น เสียงปรบมือ เสียงพูด ใช้งานเป็นได้ทั้งตัวตรวจจับดิจิทัลและอะนาล็อก

กรณีทำงานกับอินพุตอะนาล็อก (A1 ถึง A7) ของแผงวงจรหลัก IPST-SE

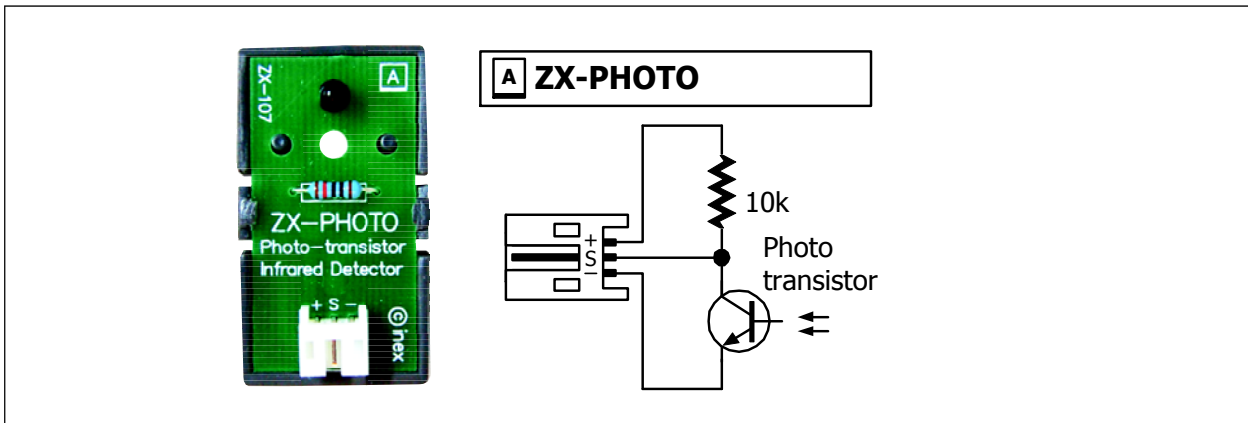
ในภาวะปกติ แรงดันเอาต์พุต 0V

เมื่อมีเสียงเข้ามา แรงดันเปลี่ยนแปลงในช่วงมากกว่า 0V ถึง +5V

แสดงวงจรและลักษณะของแผงวงจรในรูปแบบที่ 2-14



รูปที่ 2-14 วงจรสมบูรณ์ของแผงวงจรตรวจจับเสียงและหน้าตาของแผงวงจรที่ใช้งานจริง



รูปที่ 2-15 รูปร่างและวงจรของแผงวงจรตรวจจับแสงอินฟราเรดโดยใช้โฟโตทรานซิสเตอร์ที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX

2.4.6 แผงวงจรตรวจจับแสงอินฟราเรดโดยใช้โฟโตทรานซิสเตอร์ :

ZX-Photo Transistor (มีในชุดมาตรฐาน 3)

มีวงจรและรูปร่างของแผงวงจรแสดงในรูปที่ 2-15

ใช้ตรวจจับแสงที่มีความยาวคลื่นอยู่ในช่วงอินฟราเรด ซึ่งมีค่าระหว่าง 1 ไมโครเมตร (μm) ถึง 1 มิลลิเมตร (mm) ใช้งานได้ 2 ลักษณะคือ

1. อ่านค่าเป็นระดับความเข้มแสงแบบอนาล็อก โดยแรงดันเอาต์พุตที่ได้จะลดลงเมื่อได้รับแสงอินฟราเรดที่มีความเข้มเพิ่มขึ้น (ถ้าเลือกการทำงานแบบนี้ให้ต่อสัญญาณเข้ากับจุดต่อพอร์ต A0 ถึง A6 ของแผงวงจรหลัก IPST-SE)

2. ตรวจสอบว่าตรวจจับแสงอินฟราเรดได้หรือไม่ ให้เอาต์พุตเป็นสัญญาณดิจิทัลแบบลอจิก "0" เมื่อตรวจจับแสงอินฟราเรดได้ (ถ้าเลือกการทำงานแบบนี้ให้ต่อสัญญาณเข้ากับพอร์ตใดก็ได้ของแผงวงจรหลัก IPST-SE)

ควรใช้งานแผงวงจรตรวจจับแสงอินฟราเรดตัวนี้กับแผงวงจรกำเนิดแสงอินฟราเรด ZX-IRLED

นอกจากนั้น ยังใช้แผงวงจรตรวจจับแสงอินฟราเรดในการตรวจจับเปลวเทียนได้ด้วยเนื่องจากเมื่อเทียนถูกจุด เกิดการเผาไหม้ไส้เทียน ทำให้เกิดแสงสว่าง และรังสีอินฟราเรดออกจากเปลวเทียน

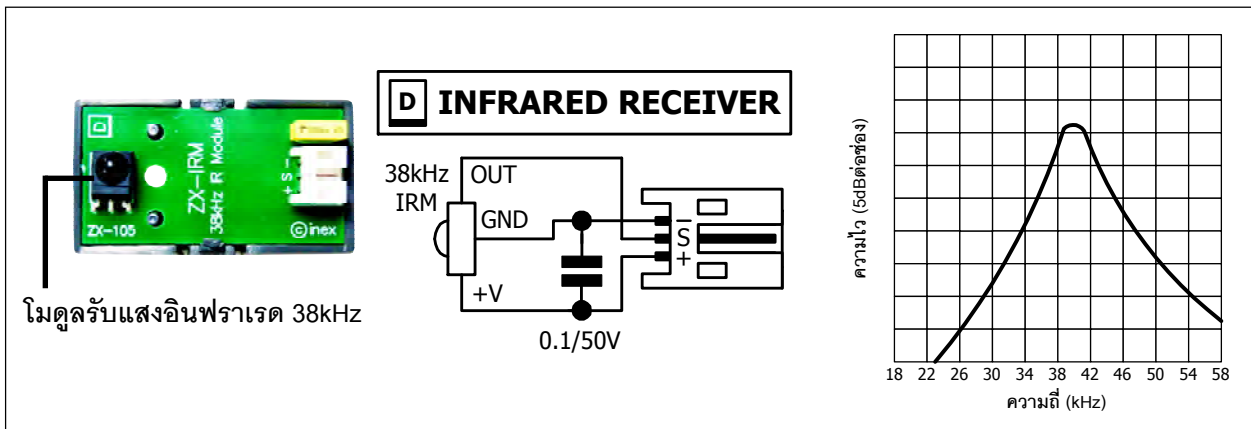
2.4.7 แผงวงจรโมดูลรับแสงอินฟราเรด 38kHz : ZX-IRM (มีในชุดมาตรฐาน 3)

มีวงจรและหน้าตาของบอร์ดแสดงในรูปที่ 2-16 ใช้ตรวจจับแสงอินฟราเรดที่ผสมสัญญาณพาห้ความถี่ 38 kHz

ให้ผลเป็นลอจิก “1” เมื่อตรวจจับสัญญาณไม่ได้

ให้ผลเป็นลอจิก “0” เมื่อตรวจจับสัญญาณแสงได้

โดยปกติแล้วโมดูลรับแสงอินฟราเรดทำงานได้ดีที่สุดที่ความถี่ 38.5kHz แต่ในความเป็นจริงโมดูลรับแสงอินฟราเรด 38kHz สามารถรับสัญญาณที่มีความถี่ใกล้เคียงเข้ามาได้ แต่การตอบสนองหรือความไวจะลดลงอย่างมาก จากกราฟคุณสมบัติในการทำงานของโมดูลรับแสงอินฟราเรด 38kHz ในรูปที่ 2-16 แสดงให้เห็นถึงความไวในการรับสัญญาณของโมดูลรับแสงอินฟราเรดที่ความถี่ต่างๆ ที่ความถี่ 38.5kHz จะเป็นจุดที่ให้ความแรงของสัญญาณสูงสุด



รูปที่ 2-16 หน้าตา, วงจร และกราฟคุณสมบัติในการทำงานของแผงวงจรโมดูลรับอินฟราเรด 38kHz ที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX (SE) รุ่นมาตรฐาน 3

2.4.8 รีโมทคอนโทรลอินฟราเรด (มีในชุดมาตรฐาน 3)

เป็นรีโมทคอนโทรลที่ใช้รูปแบบข้อมูลของ SONY ซึ่งเป็นรหัสข้อมูลแบบอนุกรมที่เรียกว่า โปรโตคอล SIRCS (serial infrared remote control system) สำหรับรีโมทคอนโทรลของโทรทัศน์มีชื่อรหัสคือ Sony D7C5 มีหน้าตาแสดงในรูปที่ 2-17 ใช้งานร่วมกับ ZX-IRM แผงวงจรโมดูลรับแสงอินฟราเรด 38kHz

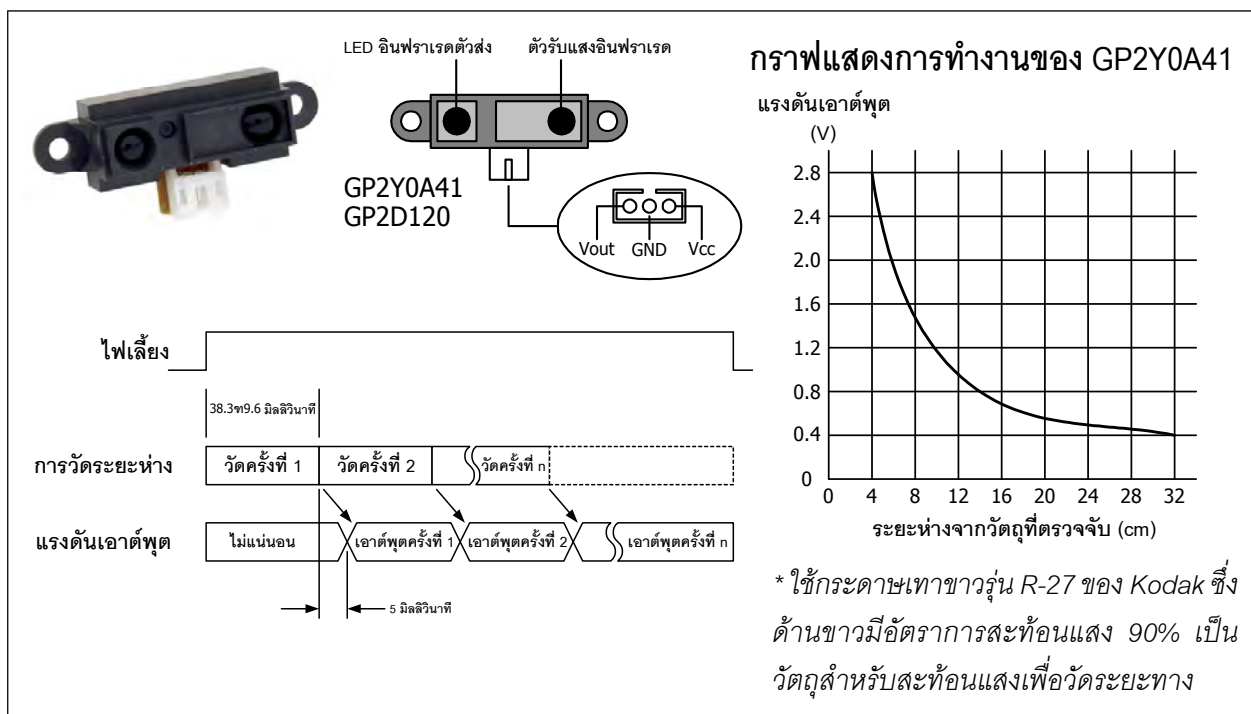


รูปที่ 2-17 รีโมทคอนโทรลอินฟราเรดที่ใช้รหัสข้อมูลของ SONY

2.4.9 GP2Y0A41 โมดูลตรวจจับระยะทางแบบอินฟราเรด (มีในชุดมาตรฐาน 3)

GPY0A41 (หรือ GP2D120) เป็นโมดูลตรวจจับระยะทางแบบอินฟราเรดมีขาต่อใช้งาน 3 ขาคือ ขาต่อไฟเลี้ยง (Vcc), ขากราวด์ (GND) และขาแรงดันเอาต์พุต (Vout) การอ่านค่าแรงดันจาก GP2D120 จะต้องรอให้พื้นช่วงเตรียมความพร้อมของโมดูลก่อน ซึ่งใช้เวลาประมาณ 32.7-52.9 มิลลิวินาที ดังนั้นในการอ่านค่าแรงดันจึงควรรอให้พื้นช่วงเวลาดังกล่าวไปก่อน ดังแสดงข้อมูลเบื้องต้นในรูปที่ 2-18

ค่าแรงดันเอาต์พุตของ GP2Y0A41 ที่ระยะทาง 30 เซนติเมตรที่ไฟเลี้ยง +5V อยู่ในช่วง 0.25 ถึง 0.55V โดยค่ากลางคือ 0.4V ช่วงของการเปลี่ยนแปลงแรงดันเอาต์พุตที่ระยะทาง 4 เซนติเมตรคือ $2.25V \pm 0.3V$



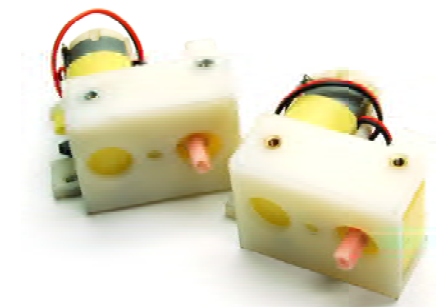
รูปที่ 2-18 แสดงรูปร่าง การจัดขา โดยแอมป์เวลาจังหวะการทำงาน และกราฟแสดงการทำงานของ GP2Y0A41 (หรือ GP2D120)

2.5 ข้อมูลของอุปกรณ์ทางกลที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX (SE) รุ่นมาตรฐาน 2 และ 3

2.5.1 มอเตอร์ไฟตรงพร้อมชุดเฟืองขับ

เป็นชุดมอเตอร์พร้อมเฟืองขับรุ่น BO-2 อัตราทด 48:1 มีสายต่อ 2 เส้น คุณสมบัติทางเทคนิคที่สำคัญมีดังนี้

- ต้องการไฟเลี้ยงในย่าน +3 ถึง +9Vdc
- กินกระแสไฟฟ้า 130mA (ที่ไฟเลี้ยง +6V และไม่มีโหลด)
- ความเร็วเฉลี่ย 170 ถึง 250 รอบต่อนาที (ที่ไฟเลี้ยง +6V และไม่มีโหลด)
- น้ำหนัก 30 กรัม
- แรงบิดต่ำสุด 0.5 กิโลกรัม-เซนติเมตร
- ติดตั้งเข้ากับตัวยึดพลาสติกแบบมีพุกทองเหลืองสำหรับยึดในตัว
- ขนาด (กว้าง x ยาว x สูง) 42 x 45 x 22.7 มิลลิเมตร



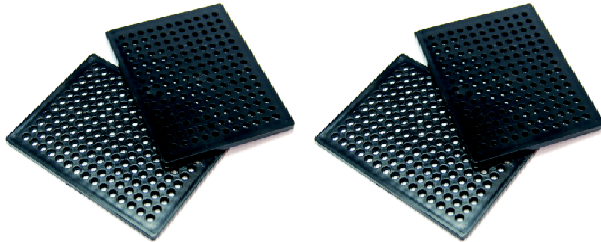
2.5.2 ล้อพลาสติกสำหรับชุดเฟืองขับมอเตอร์และยาง

เป็นล้อกลม มีเส้นผ่านศูนย์กลาง 65 มิลลิเมตร สามารถสวมเข้ากับแกนของชุดเฟืองขับมอเตอร์ได้ทันทีโดยไม่ต้องตัดแปลงเพิ่มเติม ขันยึดด้วยสกรูเกลียวปละลาย 2 มิลลิเมตร ส่วนยางขอบล้อที่ใช้ร่วมด้วยผลิตจากยางพารา ผิวมีดอกยางเพื่อช่วยเพิ่มสมรรถนะในการเกาะพื้นผิว



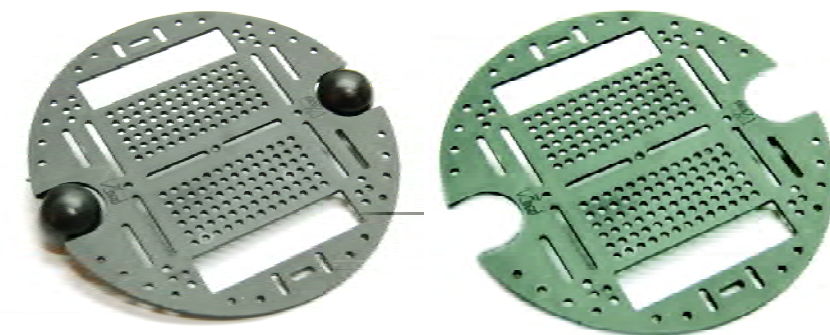
2.5.3 แผ่นกริด

เป็นแผ่นพลาสติกที่ผลิตจากวัสดุ ABS ขนาด 80 x 60 มิลลิเมตร และ 80 x 80 มิลลิเมตร อย่างละ 1 แผ่น ในแต่ละมีรูขนาด 3 มิลลิเมตรที่มีระยะห่างกัน 5 มิลลิเมตร



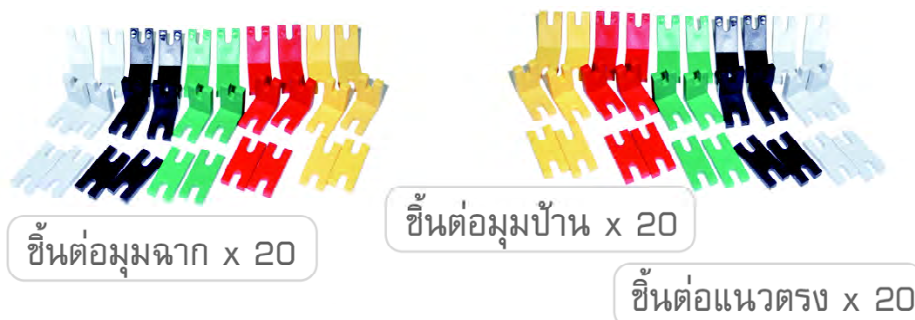
2.5.4 แผ่นฐานกลม

เป็นแผ่นพลาสติกที่ผลิตจากวัสดุ ABS ขนาดเส้นผ่านศูนย์กลาง 250 มิลลิเมตร เป็นแผ่นฐานสำหรับยึดอุปกรณ์ต่างๆ ที่แผ่นฐานมีรูขนาด 3 มิลลิเมตรสำหรับติดตั้งอุปกรณ์หรือโครงสร้างกลไกเพิ่มเติม มีรูปร่างเป็นแผ่นกลม ในชุดมีให้ 2 แบบ แบบมีล้อกลมกึ่งอิสระทั้งด้านหน้าและหลัง และแบบมาตรฐานไม่มีล้อกลมกึ่งอิสระ



2.5.5 ชิ้นต่อพลาสติก

เป็นชิ้นส่วนพลาสติกแข็งเหนียว มี 3 แบบคือ ชิ้นต่อแนวตรง, ชิ้นต่อมุมฉาก และชิ้นต่อมุมป้าน สามารถเสียบต่อกันได้ ใช้ต่อกันเป็นโครงสร้างหรือตกแต่ง บรรจุ 3 แบบ รวม 30 ชิ้น



2.5.6 แท่งต่อพลาสติก

เป็นชิ้นส่วนพลาสติกแข็งเหนียวในแต่ละชั้นจะมีรูขนาด 3 มิลลิเมตรสำหรับร้อยสกรูเพื่อติดตั้งหรือต่อกับชิ้นส่วนโครงสร้างอื่นๆ ที่ปลายของแท่งต่อสามารถเสียบเข้ากับชิ้นต่อพลาสติกได้ ในชุดมี 3 ขนาด คือ 3, 5 และ 12 รู แต่ละขนาดมี 4 ชั้น



2.5.7 สกรูและนอต

เป็นอุปกรณ์สำหรับยึดชิ้นส่วนต่างๆ เข้าด้วยกัน ในชุดประกอบด้วย สกรูมือหมุน (4 ตัว), สกรูเกลียวป้อย 2 มิลลิเมตร (2 ตัว), สกรู 3 x 8 มิลลิเมตร (4 ตัว), 3 x 10 มิลลิเมตร (30 ตัว), 3 x 15 มิลลิเมตร (4 ตัว), 3 x 40 มิลลิเมตร (4 ตัว), สกรูหัวตัด 3 x 8 มิลลิเมตร (10 ตัว) และนอต 3 มิลลิเมตร (30 ตัว)



2.5.8 เสาองโลหะ

เป็นอุปกรณ์ช่วยยึดชิ้นส่วนต่างๆ และรองรับแผงวงจร, แผ่นกริดและแผ่นฐาน ทำจากโลหะชุบนิเกิลกันสนิม มีลักษณะเป็นแท่งทรงกระบอกยาว 50 มิลลิเมตร ภายในมีรูเกลียวตลอดตัวสำหรับขันสกรู 3 มิลลิเมตร ในชุดมี 4 ตัว



2.5.9 เสารองพลาสติก

เป็นอุปกรณ์ช่วยยึดชิ้นส่วนต่างๆ และประคอง รองรับแผงวงจร, แผ่นกริดและแผ่นฐาน ทำจากพลาสติก ABS เหนียว สามารถตัดได้ มีลักษณะเป็นแท่งทรงกระบอก ภายในมีรูตลอดตัวสำหรับร้อยสกรู 3 มิลลิเมตร ในชุดประกอบด้วย เสารองขนาด 3 มิลลิเมตร (4 ตัว), 10 มิลลิเมตร (4 ตัว), 15 มิลลิเมตร (4 ตัว) และ 25 มิลลิเมตร (4 ตัว)



2.5.10 กะบะถ่าน

ใช้บรรจุแบตเตอรี่ขนาด AA จำนวน 6 ก้อน มีหัว ต่อแบบเดียวกับปลั๊กอะแดปเตอร์สำหรับต่อกับแผงวงจร ควบคุมหลัก IPST-SE ได้ทันที



2.5.11 อะแดปเตอร์ไฟตรง

เป็นแหล่งจ่ายไฟตรงแบบสวิตซิ่ง มี 2 ตัว

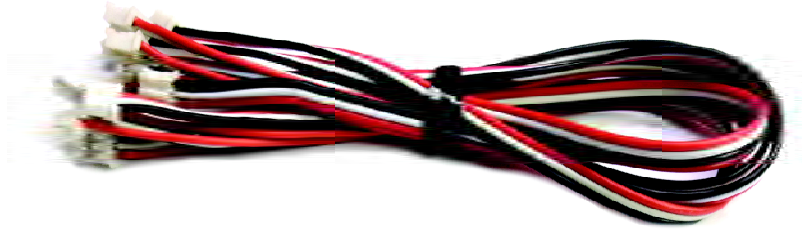
1. อะแดปเตอร์ +9V 2A ให้แรงดันขาออก +9V จ่ายกระแสไฟฟ้าได้สูงสุด 2A ปลายสายเป็นหัวปลั๊ก แบบบาร์เรล (barrel) ซึ่งเป็นมาตรฐานที่พบโดยทั่วไป

2. อะแดปเตอร์ +12V 1A ให้แรงดันขาออก +12V จ่ายกระแสไฟฟ้าได้สูงสุด 1A ปลายสายต่อเป็นแบบขั้วต่อที่ป้องกันการต่อกลับขั้ว มีแถบสีแดงระบุขั้ว บวกไว้อย่างชัดเจน

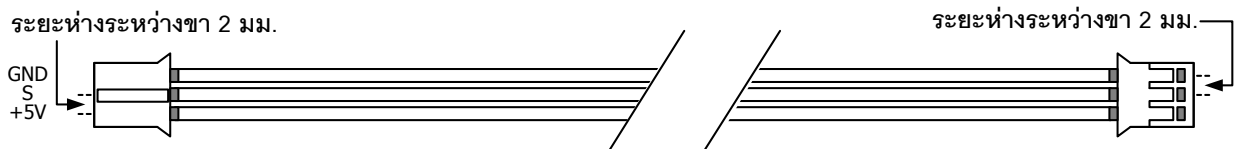


2.6 ข้อมูลของสายสัญญาณที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX(SE)

2.6.1 สาย JST3AA-8 : สายเชื่อมต่อระหว่างแผงวงจร



สาย JST3AA-8 ใช้เชื่อมต่อระหว่างแผงวงจรควบคุม IPST-SE กับแผงวงจรตรวจจับและแผงวงจรอุปกรณ์ต่างๆ เป็นสายแพ 3 เส้น ยาว 8 นิ้ว ปลายสายทั้งสองด้านติดตั้งคอนเน็กเตอร์แบบ JST 3 ขา ตัวเมีย ระยะห่างระหว่างขา 2 มิลลิเมตร มีการจัดขาดังนี้



2.6.2 สาย USB-miniB

เป็นสายสัญญาณสำหรับเชื่อมต่อระหว่างพอร์ต USB ของคอมพิวเตอร์กับแผงวงจร IPST-SE



บทที่ 3

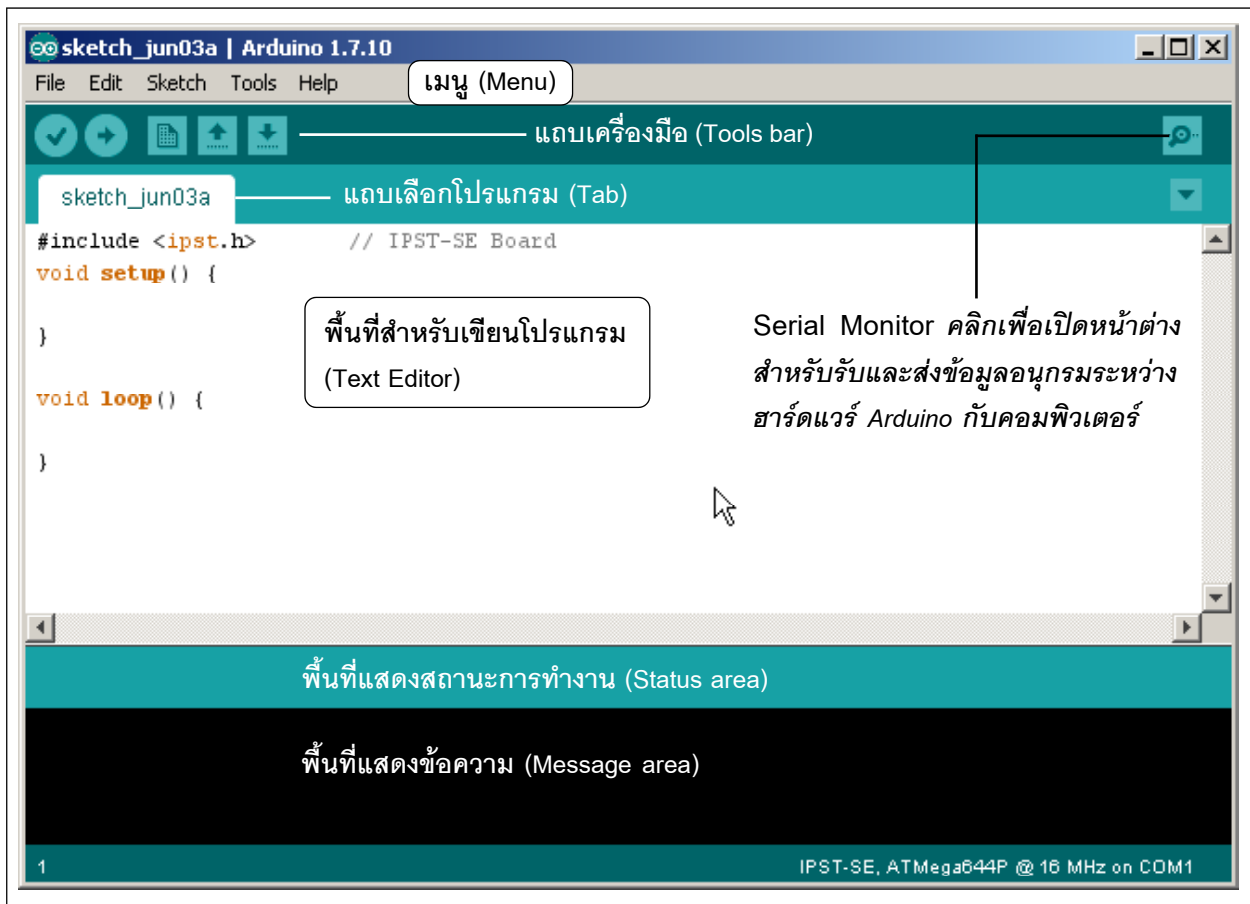
รู้จักกับ Arduino IDE ซอฟต์แวร์ พัฒนาโปรแกรมภาษา C/C++ สำหรับ ชุดกล่องสมองกล SE

ในบทนี้จะอธิบายส่วนประกอบและรายละเอียดของซอฟต์แวร์ Arduino IDE 1.7.10 ที่ใช้ในการเขียนโปรแกรม คอมไพเลอร์โปรแกรม และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยซอฟต์แวร์ Arduino IDE 1.7.10 ทำงานบนระบบปฏิบัติการได้ทุกแบบ ไม่ว่าจะเป็นวินโดวส์ที่รองรับตั้งแต่วินโดวส์ XP ขึ้นไป (แนะนำ 8 ขึ้นไป), MAC OS และ Linux

3.1 ส่วนประกอบของหน้าจอโปรแกรม Arduino1.7.10

เมื่อเรียกให้โปรแกรมทำงาน จะมีหน้าต่างดังรูปที่ 3-1 ตัวโปรแกรมประกอบด้วยส่วนต่างๆ ดังนี้

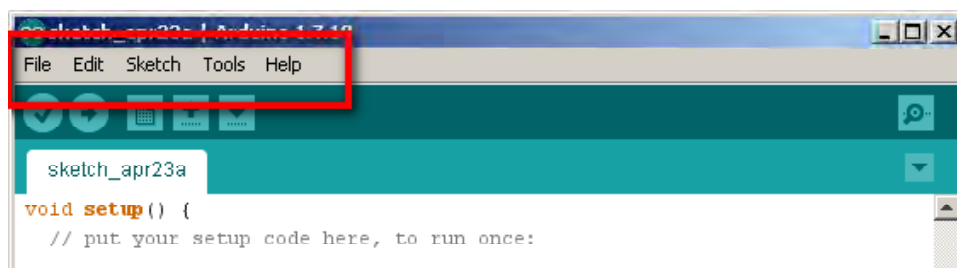
- **เมนู (Menu)** ใช้เลือกคำสั่งต่างๆ ในการใช้งานโปรแกรม
- **แถบเครื่องมือ (Toolbar)** เป็นการนำคำสั่งที่ใช้งานบ่อยมาสร้างเป็นปุ่ม เพื่อเรียกใช้ได้เร็วขึ้น
- **แถบเลือกโปรแกรม (Tabs)** เป็นแถบที่ใช้เลือกไฟล์โปรแกรมแต่ละตัว (กรณีที่เขียนโปรแกรมขนาดใหญ่ประกอบด้วยไฟล์หลายตัว)
- **พื้นที่เขียนโปรแกรม (Text editor)** เป็นพื้นที่สำหรับเขียนโปรแกรมภาษา C/C++
- **พื้นที่แสดงสถานะการทำงาน (Status area)** เป็นพื้นที่ใช้แจ้งสถานะการทำงานของโปรแกรม เช่น ผลการคอมไพล์โปรแกรม
- **พื้นที่แสดงข้อความ (Message area)** ใช้แจ้งว่าโปรแกรมที่ผ่านการคอมไพล์แล้วมีขนาดกี่ไบต์ และแจ้งตำแหน่งหรือรายละเอียดของข้อผิดพลาดในการทำงานของโปรแกรม
- **ปุ่มสำหรับเปิดหน้าต่าง Serial Monitor** ปุ่มนี้จะอยู่ทางมุมบนด้านขวามือ คลิกปุ่มนี้เมื่อต้องการเปิดหน้าต่างสื่อสารและแสดงข้อมูลอนุกรม โดยต้องมีการต่อฮาร์ดแวร์ Arduino และเลือกพอร์ตการเชื่อมต่อให้ถูกต้องก่อน



รูปที่ 3-1 แสดงส่วนประกอบของโปรแกรม Arduino IDE 1.7.10

3.2 รายละเอียดแถบคำสั่งหรือเมนู (Menu)

เป็นส่วนที่แสดงรายการ (เมนู) ของคำสั่งต่างๆ ของโปรแกรม ประกอบด้วย



1. เมนู File

ใน Arduino เรียกโปรแกรมที่พัฒนาขึ้นว่า **สเก็ตช์ (Sketch)** และในโปรแกรมของผู้ใช้งานอาจมีไฟล์โปรแกรมหลายตัว จึงเรียกรวมว่าเป็น **สเก็ตช์บุ๊ก (Sketchbook)** ในเมนูนี้เกี่ยวข้องกับการเปิด-บันทึก-ปิดไฟล์ดังนี้

- **New** : ใช้สร้างไฟล์สเก็ทซ์ตัวใหม่ เพื่อเริ่มเขียนโปรแกรมใหม่
- **Open** ใช้เปิดสเก็ทซ์ที่บันทึกไว้ก่อนหน้านี้
- **Sketchbook** : ใช้เปิดไฟล์สเก็ทซ์ล่าสุดที่เปิดใช้งานเสมอ
- **Example** : ใช้เปิดไฟล์สเก็ทซ์ตัวอย่างในโฟลเดอร์ Arduino
- **Close** : ใช้ปิดไฟล์สเก็ทซ์ที่เปิดอยู่
- **Save** : ใช้ในการบันทึกไฟล์สเก็ทซ์ปัจจุบัน
- **Save as** : ใช้บันทึกไฟล์สเก็ทซ์โดยเปลี่ยนชื่อไฟล์
- **Upload** : ใช้อัปโหลดโปรแกรมไปยังฮาร์ดแวร์ของ Arduino
- **Preference** : ใช้กำหนดค่าการทำงานของโปรแกรม
- **Quit** : ใช้จบการทำงานและออกจากโปรแกรม

2. เมนู Edit

ในขณะที่พิมพ์โปรแกรมให้ใช้คำสั่งในเมนูนี้ในการแก้ไขโปรแกรม รวมถึงยกเลิกคำสั่งที่แล้ว ทำซ้ำ ตัด คัดลอก วางข้อความ ค้นหา เป็นต้น มีเมนูต่างๆ ดังนี้

- **Undo** : ยกเลิกคำสั่งหรือการพิมพ์ครั้งสุดท้าย
- **Redo** : ทำซ้ำคำสั่งหรือการพิมพ์ครั้งสุดท้าย
- **Cut** : ตัดข้อความที่เลือกไว้ไปเก็บในคลิปบอร์ดของโปรแกรม
- **Copy** : คัดลอกข้อความที่เลือกไว้มาเก็บในคลิปบอร์ด
- **Paste** : นำข้อความที่อยู่ในคลิปบอร์ดมาแปะลงในตำแหน่งที่เคอร์เซอร์ชี้อยู่
- **Select All** : เลือกข้อความทั้งหมด
- **Comment/Uncomment** : ใช้เติมเครื่องหมาย // เพื่อสร้างหมายเหตุหรือคำอธิบายลงในโปรแกรมหรือยกเลิกหมายเหตุด้วยการนำเครื่องหมาย // ออก
- **Find** : ค้นหาข้อความ
- **Find Next** : ค้นหาข้อความถัดไป

3. เมนู Sketch

เป็นเมนูที่บรรจุคำสั่งที่ใช้คอมไพล์โปรแกรม เพิ่มไฟล์ไลบรารี ฯลฯ มีเมนูย่อยดังนี้

- **Verify/Compile** : ใช้คอมไพล์แปลโปรแกรมภาษาซีให้เป็นภาษาเครื่อง
- **Import Library** : เป็นคำสั่งเรียกใช้ไลบรารีเพิ่มเติม เมื่อเลือกคำสั่งนี้แล้ว โปรแกรม Arduino IDE จะแสดงไลบรารีให้เลือก เมื่อเลือกแล้ว ต้องแทรกบรรทัดคำสั่ง `#include` ที่ส่วนต้นของโปรแกรม
- **Add file** : เพิ่มไฟล์ให้กับสเก็ทซ์ปัจจุบัน เมื่อใช้คำสั่งนี้โปรแกรม Arduino จะคัดลอกไฟล์ที่เลือกไว้มาเก็บไว้ในโฟลเดอร์เดียวกันกับโปรแกรมที่กำลังพัฒนา
- **Show Sketch folder** : สั่งเปิดโฟลเดอร์ที่เก็บโปรแกรมของผู้ใช้

4. เมนู Tools

ใช้จัดรูปแบบของโค้ดโปรแกรม, เลือกฮาร์ดแวร์ไมโครคอนโทรลเลอร์ Arduino หรือเลือกพอร์ตอนุกรม มีเมนูพื้นฐานดังนี้

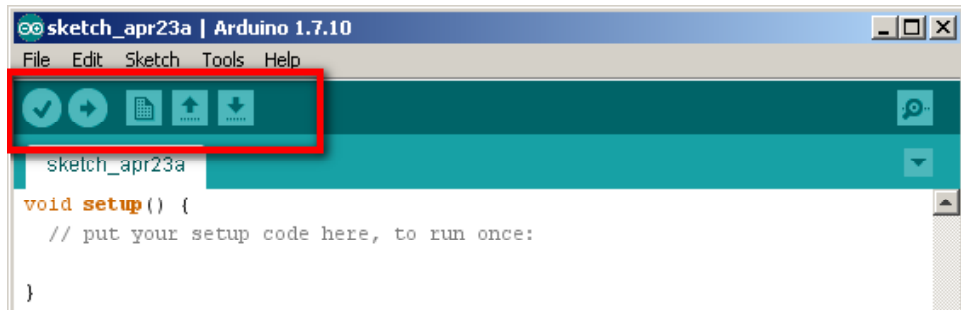
- **Auto Format** : จัดรูปแบบของโค้ดโปรแกรมให้สวยงาม เช่น กั้นหน้าเยื้องขวา จัดตำแหน่งวงเล็บปีกกาปิดให้ตรงกับปีกกาเปิด ถ้าเป็นคำสั่งที่อยู่ภายในวงเล็บปีกกาเปิดและปิดจะถูกกั้นหน้าเยื้องไปทางขวามากขึ้น
- **Archive Sketch** : สั่งบีบอัดไฟล์โปรแกรมทั้งโฟลเดอร์หลักและโฟลเดอร์ย่อยของไฟล์สเก็ทซ์ปัจจุบัน ไฟล์ที่สร้างใหม่จะมีชื่อเดียวกับไฟล์สเก็ทซ์ปัจจุบันและต่อท้ายด้วยวันเดือนปีที่สร้างไฟล์สเก็ทซ์ขึ้น เช่น `-510123.zip`
- **Board** : เลือกฮาร์ดแวร์ของบอร์ดไมโครคอนโทรลเลอร์ Arduino สำหรับแผงวงจร IPST-SE ให้เลือก **IPST-SE, Atmega644P@16MHz**
- **Serial Port** : เลือกหมายเลขพอร์ตอนุกรมเสมือน (COM) ของคอมพิวเตอร์ที่ใช้ติดต่อกับฮาร์ดแวร์ Arduino

5. เมนู Help


เมื่อต้องการความช่วยเหลือ หรือข้อมูลเกี่ยวกับโปรแกรมให้เลือกเมนูนี้ เมื่อเลือกเมนูย่อยตัวโปรแกรมจะเปิดไฟล์เว็บเพจ (ไฟล์นามสกุล `.html`) ที่เกี่ยวข้องกับหัวข้อนั้นๆ โดยไฟล์จะเก็บในคอมพิวเตอร์ของผู้ใช้งาน ภายในโฟลเดอร์ที่เก็บ Arduino IDE


3.3 แถบเครื่องมือ (Toolbar)


เป็นการนำคำสั่งที่ใช้งานบ่อยๆ มาสร้างเป็นปุ่ม เพื่อให้เรียกใช้คำสั่งได้รวดเร็วขึ้น




สำหรับคำสั่งที่มีการใช้บ่อยๆ ตัวโปรแกรม Arduino จะนำมาสร้างเป็นปุ่มบนแถบเครื่องมือ เพื่อให้คลิกเลือกได้ทันที ปุ่มต่างๆ บนแถบเครื่องมือมีดังนี้

 **Verify/Compile** ใช้ตรวจสอบการเขียนคำสั่งในโปรแกรมว่า ถูกต้องตามหลักไวยากรณ์หรือไม่ และคอมไพล์โปรแกรม

 **Upload to I/O Board** ใช้อัปโหลดโปรแกรมที่เขียนขึ้นไปยังบอร์ดหรือฮาร์ดแวร์ Arduino ก่อนจะอัปโหลดไฟล์ ต้องแน่ใจว่าได้บันทึกไฟล์และคอมไพล์ไฟล์สเก็ทซ์เรียบร้อยแล้ว

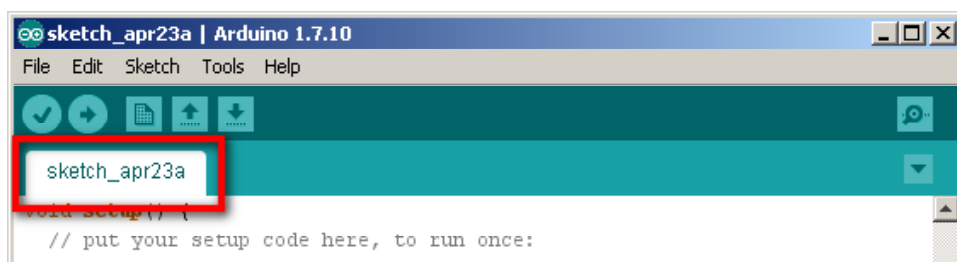
 **New** ใช้สร้างสเก็ทไฟล์ (ไฟล์โปรแกรม) ตัวใหม่

 **Open** ใช้แทนเมนู **File > Sketchbook** เพื่อเปิดสเก็ทซ์ (ไฟล์โปรแกรม) ที่มีในเครื่อง

 **Save** ใช้บันทึกไฟล์สเก็ทซ์บุ๊กที่เขียนขึ้น

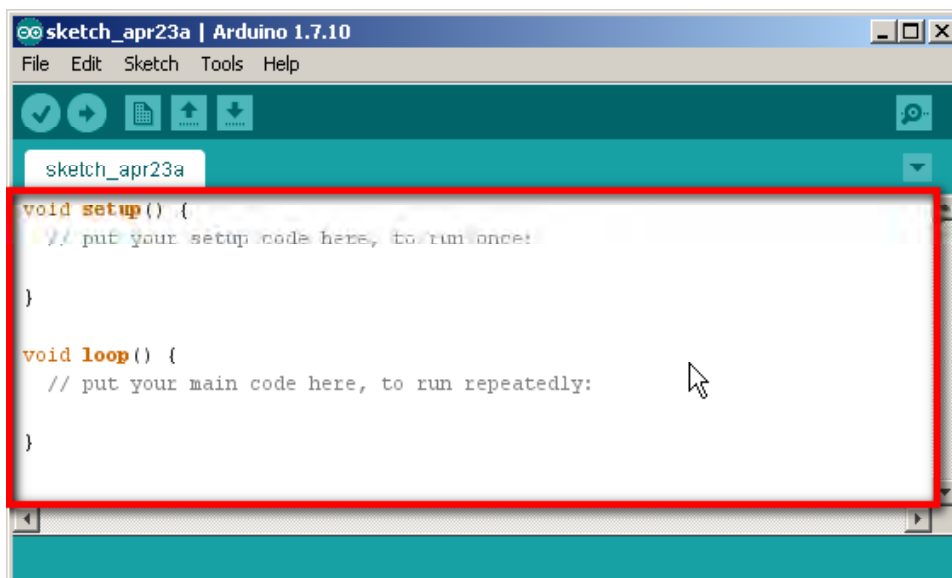
3.4 แถบเลือกโปรแกรม (Tabs)

เป็นแถบที่ใช้เลือกไฟล์โปรแกรมแต่ละตัว (กรณี que เขียนโปรแกรมขนาดใหญ่ประกอบด้วยไฟล์หลายตัว)



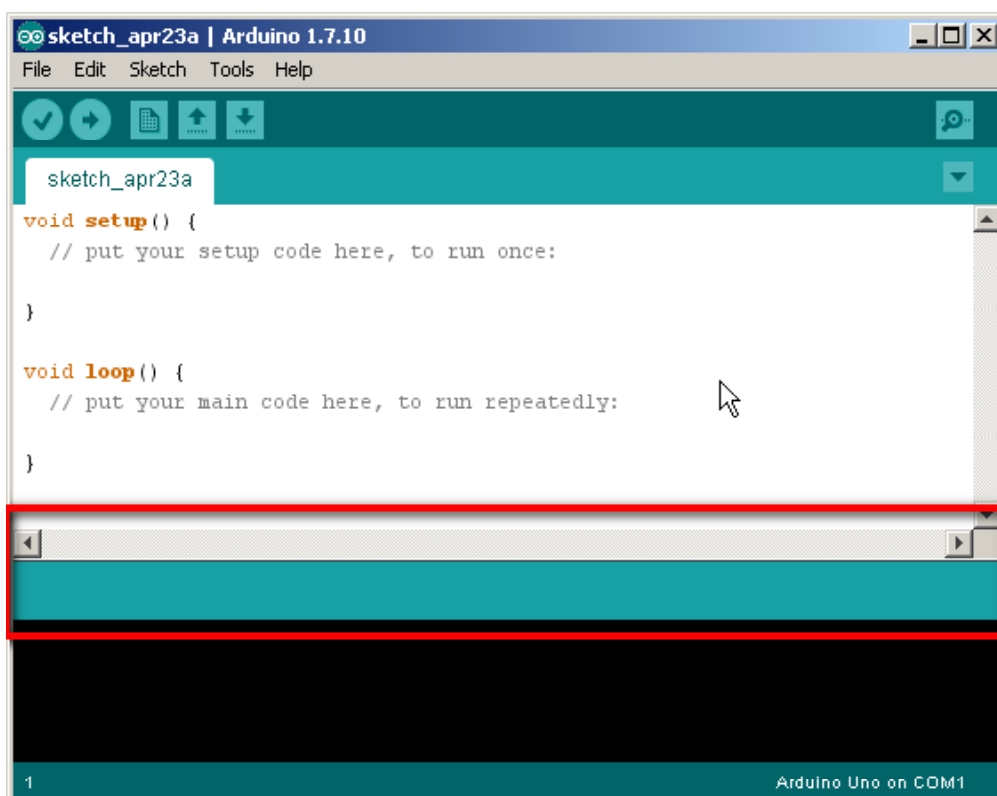
3.5 พื้นที่เขียนโปรแกรม (Text editor)

เป็นพื้นที่สำหรับเขียนโปรแกรมภาษา C/C++



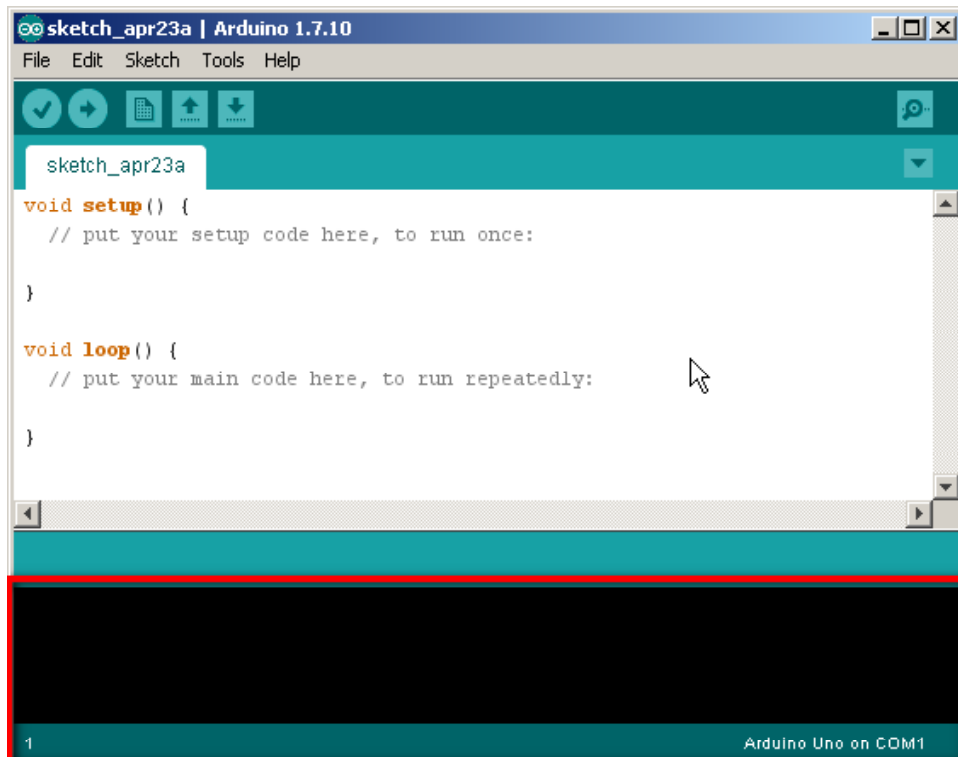
3.6 พื้นที่แสดงสถานะการทำงาน (Status area)

เป็นพื้นที่โปรแกรมใช้แจ้งสถานะการทำงานของโปรแกรม เช่น ผลการคอมไพล์โปรแกรม



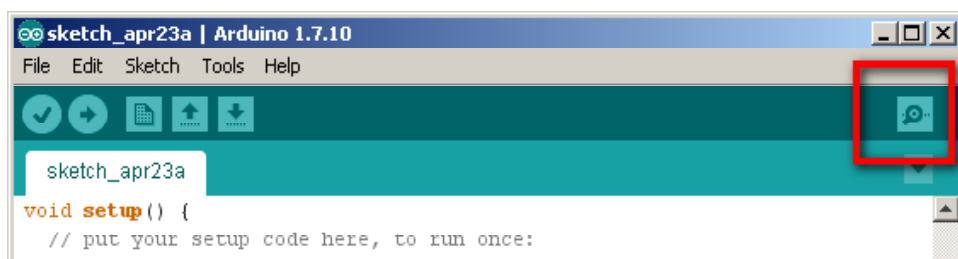
3.7 พื้นที่แสดงข้อมูล (Message area)

ใช้แจ้งว่าโปรแกรมที่ผ่านการคอมไพล์แล้วมีขนาดกี่ไบต์และแจ้งสาเหตุของการคอมไพล์โปรแกรมไม่ผ่าน ตำแหน่งหรือคำสั่งที่ผิดพลาด

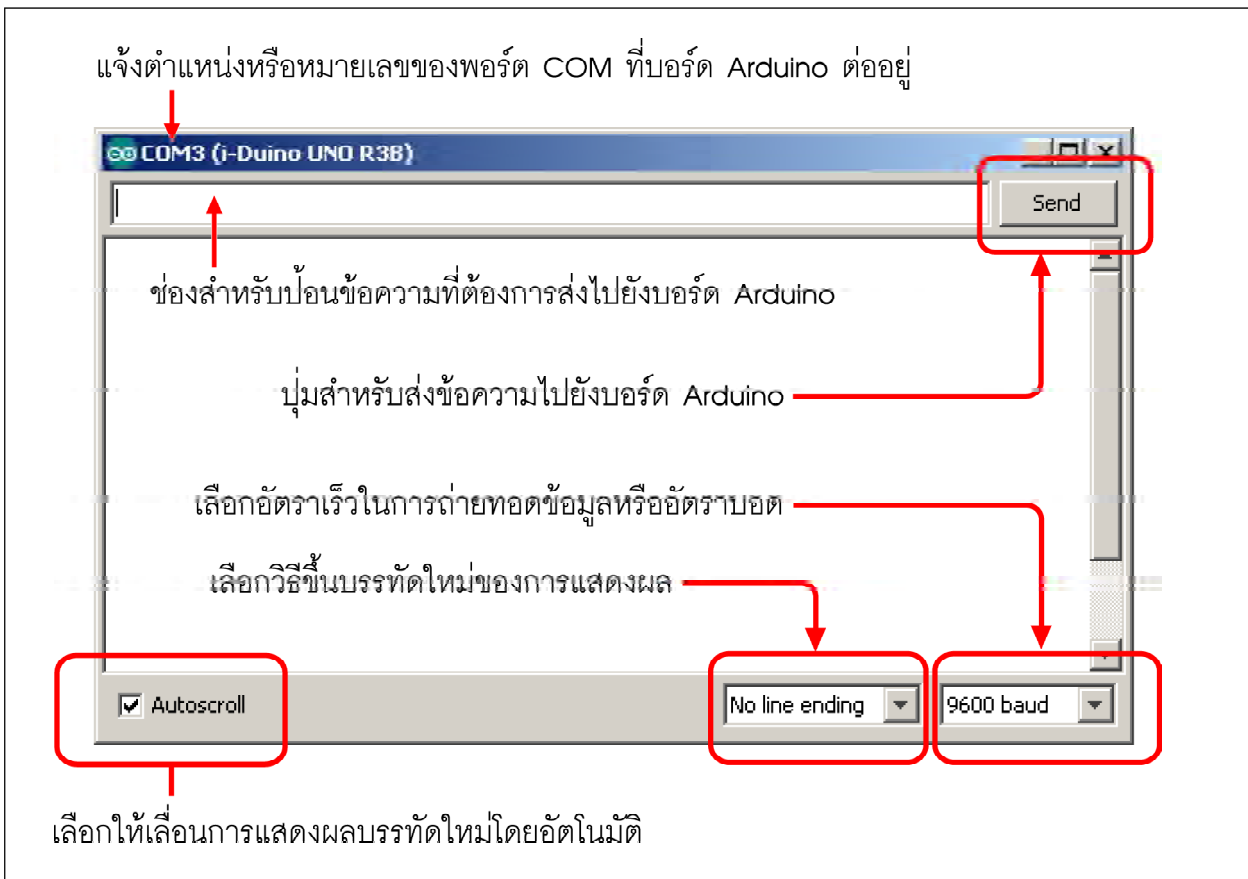


3.8 ปุ่มเปิดหน้าต่าง Serial Monitor

ปุ่มนี้จะอยู่ทางมุมบนด้านขวามือ คลิกปุ่มนี้เมื่อต้องการเปิดหน้าต่างสื่อสารและแสดงข้อมูลอนุกรม โดยต้องมีการต่อฮาร์ดแวร์ Arduino และเลือกพอร์ต การเชื่อมต่อให้ถูกต้องก่อน



เมื่อคลิกเปิดหน้าต่าง Serial Monitor ขึ้นมา จะปรากฏหน้าต่างดังรูปที่ 3-2



รูปที่ 3-2 แสดงส่วนประกอบของหน้าต่าง Serial Monitor ของซอฟต์แวร์ Arduino IDE

หน้าต่าง Serial Monitor มีบทบาทมากในการใช้แสดงผลการทำงานของโปรแกรมแทนการใช้อุปกรณ์แสดงผลอื่นๆ เนื่องจาก Arduino ได้เตรียมคำสั่งสำหรับใช้แสดงค่าของตัวแปรที่ต้องการดูผลการทำงานไว้แล้ว นั่นคือ `Serial.print()`

ส่วนการส่งข้อมูลจากคอมพิวเตอร์ไปยังฮาร์ดแวร์ Arduino หรือแผงวงจรรควบคุม IPST-SE ให้พิมพ์ข้อความและคลิกปุ่ม **Send**

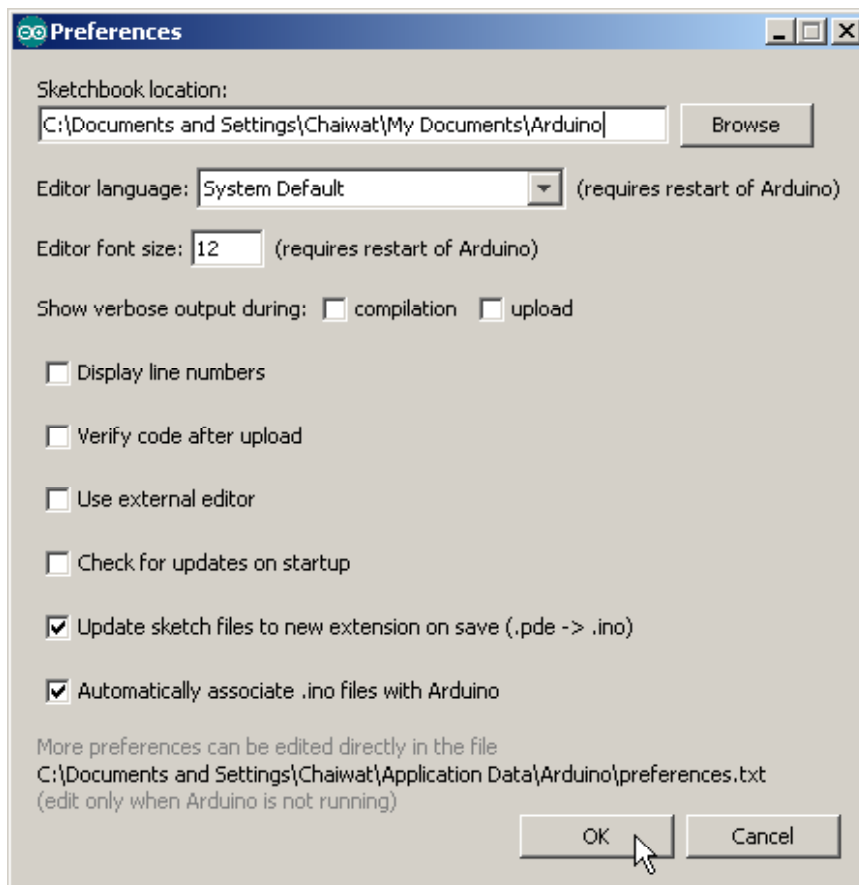
ในการรับส่งข้อมูลต้องกำหนดอัตราเร็วในการถ่ายทอดข้อมูลหรือบอดเรต (baud rate) ให้กับโปรแกรมในคำสั่ง `Serial.begin()`

ตัวฮาร์ดแวร์ของ Arduino จะรีเซตเมื่อเริ่มเปิดใช้งานหน้าต่าง Serial monitor

3.9 ตำแหน่งการเก็บไฟล์สเก็ทซ์ของ Arduino IDE

ผู้พัฒนาโปรแกรมบันทึกไฟล์สเก็ทซ์ไว้ที่ใดในพื้นที่จัดเก็บเพิ่มข้อมูลของคอมพิวเตอร์ก็ได้ แต่เพื่ออำนวยความสะดวกและจดจำง่าย Arduino IDE จึงกำหนดค่าตั้งต้น (default) ของตำแหน่งจัดเก็บไฟล์สเก็ทซ์ที่พัฒนาขึ้นไว้ที่ *C:\Documents and Settings\Admin (หรือชื่อผู้ใช้) \My Documents\Arduino* สามารถตรวจสอบรวมถึงกำหนดตำแหน่งใหม่ได้ โดยไปที่เมนู **File > Preference**

จะปรากฏหน้าต่าง **Preference** ขึ้นมาดังรูป หากต้องการเปลี่ยนแปลงตำแหน่งไฟล์สเก็ทซ์ที่จัดเก็บไฟล์สเก็ทซ์ที่ช่อง **Sketchbook location** โดยใช้เมาส์คลิกที่ปุ่มค้นหาทางด้านขวามือ หรือจะพิมพ์ตำแหน่งลงไปในช่วง **Sketchbook location** โดยตรงก็ได้ จากนั้นคลิกปุ่ม **OK** เพื่อตอบตกลงและยืนยันการเปลี่ยนแปลง



ส่วนพารามิเตอร์หรือข้อกำหนดอื่นๆ แนะนำให้ใช้ตามค่าที่กำหนดมา ทั้งนี้เพื่อลดขั้นตอนการพัฒนาโปรแกรมและความผิดพลาดที่อาจเกิดขึ้นได้สำหรับนักพัฒนาโปรแกรมมือใหม่หรือผู้เริ่มต้น

บทที่ 4

ทดสอบการควบคุมอุปกรณ์เบื้องต้น ของชุดกล่องสมองกล

หลังจากการแนะนำขั้นตอนแนวทางการพัฒนาโปรแกรมภาษา C/C++ ด้วยซอฟต์แวร์ Arduino IDE 1.7.10 ไปแล้วในบทที่ 3 เพื่อให้การเรียนรู้เป็นไปอย่างต่อเนื่อง ในบทนี้จึงนำเสนอตัวอย่างการทดสอบการทำงานในส่วนต่างๆ ที่สำคัญของแผงวงจร IPST-SE อันเป็นแผงวงจรควบคุมหลักของชุดกล่องสมองกล IPST-MicroBOX (SE) เพื่อเป็นแนวทางในการต่อยอดสู่การพัฒนาโปรแกรมเพื่อสร้างโครงงานและหุ่นยนต์อัตโนมัติอย่างเต็มรูปแบบต่อไป

หัวข้อกิจกรรมสำหรับทดสอบเบื้องต้นมีทั้งสิ้น 4 กิจกรรม ประกอบด้วย

กิจกรรมที่ 1 แสดงผลข้อความที่หน้าจอภาพกราฟิก LCD (มีกิจกรรมย่อย 5 กิจกรรม)

กิจกรรมที่ 2 อ่านค่าจากปุ่ม KNOB และสวิตช์ OK บนแผงวงจร IPST-SE

กิจกรรมที่ 3 ขับอุปกรณ์เอาต์พุตอย่างง่าย

กิจกรรมที่ 4 ขับเสียงออกลำโพงเปียโซ

ขั้นตอนการพัฒนาโปรแกรมในแต่ละกิจกรรมจะเหมือนกัน นั่นคือ เปิดซอฟต์แวร์ Arduino IDE 1.7.10 ทำการเขียนโปรแกรม คอมไพล์ และอัปโหลดลงบนแผงวงจร IPST-SE ของชุดกล่องสมองกล IPST-MicroBOX (SE) จากนั้นทดสอบการทำงาน

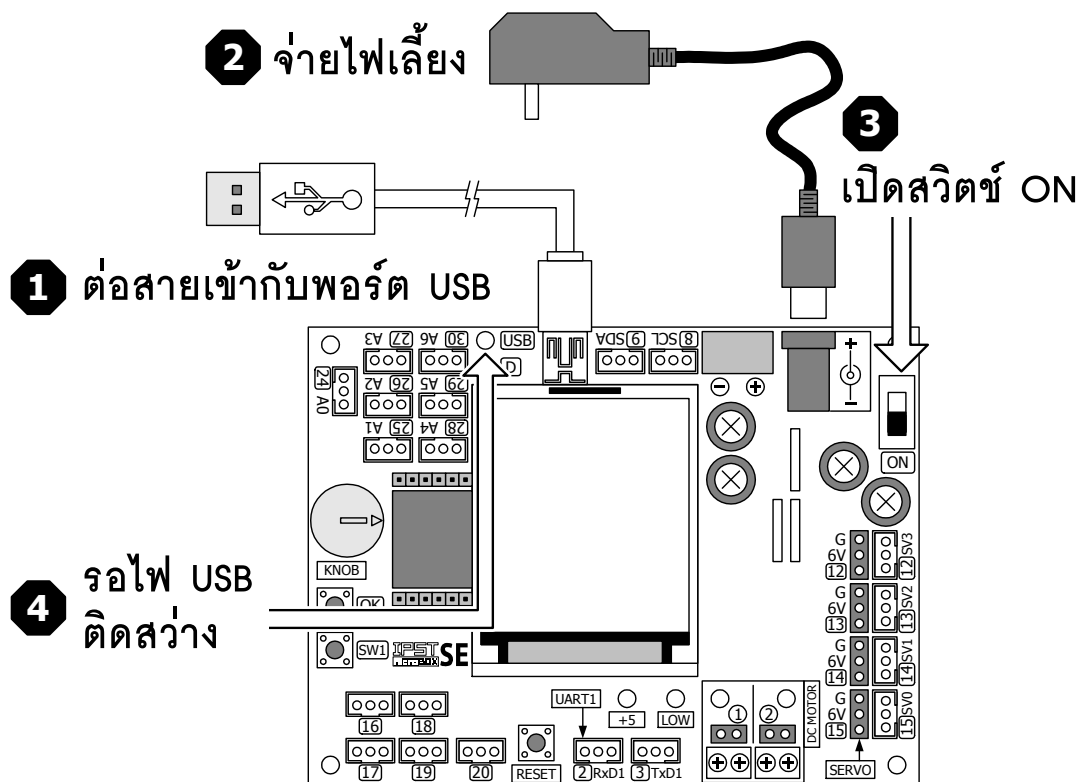
สิ่งสำคัญที่ต้องเน้นย้ำคือ ทุกครั้งที่เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE ต้องรอให้ตัวควบคุมพร้อมทำงานเสียก่อน ซึ่งใช้เวลาประมาณ 3 ถึง 5 วินาทีหลังจากเปิดไฟเลี้ยงหรือหลังจากการกดสวิตช์ RESET หากมีการอัปโหลดก่อนที่แผงวงจร IPST-SE จะพร้อมทำงาน อาจทำให้เกิดความผิดพลาดในการเชื่อมต่อ หรือ โค้ดที่อัปโหลดลงไปไม่ทำงานตามที่ควรจะเป็น แต่จะไม่ส่งผลกระทบทำให้แผงวงจรเกิดความเสียหาย สิ่งที่เกิดขึ้นมีเพียงแผงวงจรไม่ทำงานหรือทำงานไม่ถูกต้องเท่านั้น

กิจกรรมที่ 1 แสดงผลข้อความที่หน้าจอภาพกราฟิก LCD

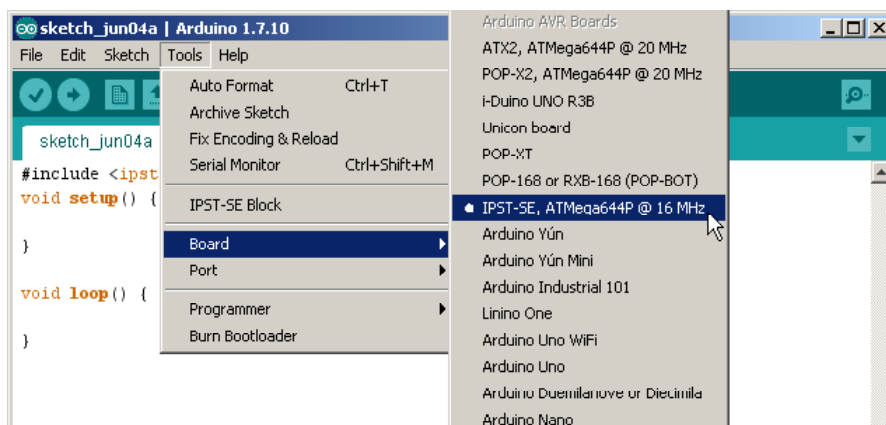
กิจกรรมที่ 1-1 Hello World

(A1.1.1) เปิดซอฟต์แวร์ Arduino IDE 1.7.10 พิมพ์โปรแกรมที่ A1-1 แล้วบันทึกไฟล์

(A1.1.2) เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์



(A1.1.3) เลือกชนิดหรือรุ่นของฮาร์ดแวร์ให้ถูกต้อง โดยเลือกที่เมนู Tools > Board > IPST-SE > ATmega644P @16MHz ดังรูป



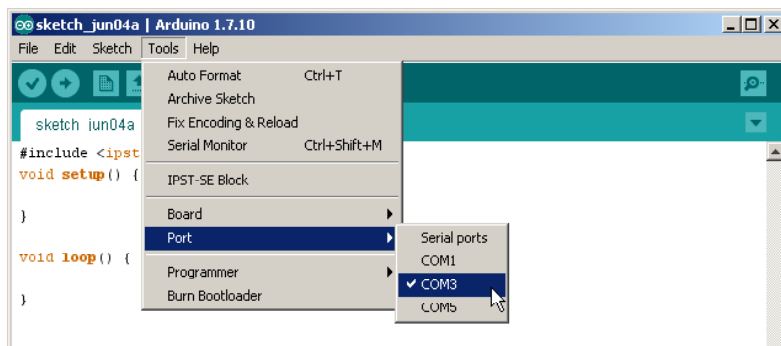
```
#include <ipst.h>           // ผนวกไลบรารีหลัก
void setup()
{
  glcd(1,0,"Hello World"); // แสดงข้อความบนจอแสดงผล
}
void loop()
{}
```


คำอธิบายโปรแกรม

โปรแกรมนี้จะทำงานโดยส่งข้อความ Hello World ออกไปแสดงผลที่บรรทัด 1 คอลัมน์ 0 ของจอแสดงผล จะทำงานเพียงครั้งเดียว จึงเขียนโปรแกรมไว้ที่ตำแหน่งของ void setup() เท่านั้น

โปรแกรมที่ A1-1 ไฟล์ microbox_HelloGLCD.ino สำหรับทดสอบการแสดงผลของแผงวงจร IPST-SE

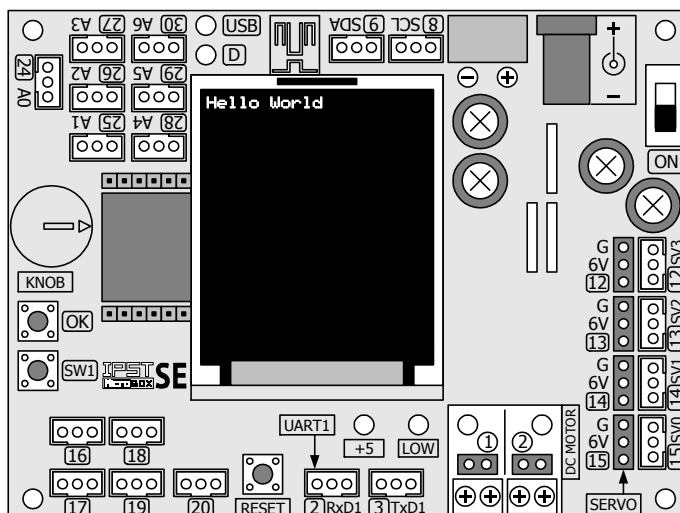
(A1.1.4) เลือกพอร์ตสำหรับติดต่อกับแผงวงจร IPST-SE โดยเลือกที่เมนู Tools > Serial Port ดังรูป (ตำแหน่งของพอร์ตที่ใช้เชื่อมต่ออาจแตกต่างกันในคอมพิวเตอร์แต่ละเครื่อง)



(A1.1.5) คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม  หรือเลือกที่เมนู

File > Upload to Wiring Hardware

ที่หน้าจอแสดงผลกราฟิก LCD ของแผงวงจร IPST-SE แสดงข้อความ Hello World



กิจกรรมที่ 1-2 การแสดงข้อความหลายบรรทัด

จอแสดงผลของแผงวงจร IPST-SE มีขนาด 128 x 160 พิกเซล แสดงตัวอักษรความละเอียด 5 x 7 จุด จำนวน 21 ตัวอักษร 16 บรรทัด ผู้ใช้งานสามารถระบุตำแหน่งบรรทัดและตำแหน่งคอลัมน์ที่ต้องการแสดงผลได้ โดยกำหนดผ่านคำสั่ง `glcd` ซึ่งมีอยู่ในไฟล์ไลบรารี `ipst.h`

นอกจากนั้นคำสั่ง `glcd` ยังมีอักขระพิเศษเพื่อระบุตำแหน่งแทนการใช้ค่าตัวเลข ดังแสดงในโปรแกรมที่ A1-2

```
#include <ipst.h> // มนวกไฟล์ไลบรารีหลัก
int i,j;
void setup()
{
  glcdFillScreen(GLCD_WHITE); // กำหนดให้สีของพื้นหลังของจอแสดงผลเป็นสีขาว
  setTextColor(GLCD_BLACK); // กำหนดสีตัวอักษรเป็นสีดำ
  setTextBackgroundColor(GLCD_WHITE); // กำหนดสีพื้นหลังของตัวอักษรเป็นสีขาว
  for (i=0;i<16;i++) // วนลูป 16 รอบเพื่อแสดงข้อความ
  {
    glcd(i,i,"Row %d ",i); // แสดงข้อความที่จอแสดงผล
  }
}
void loop()
{}

```

คำอธิบายโปรแกรม

ในโปรแกรมนี้เพิ่มเติมคำสั่งสำหรับการใช้งานจอแสดงผลอีก 3 คำสั่งคือ


1. `glcdFillScreen` เป็นคำสั่งกำหนดสีพื้นหลังของจอแสดงผล
2. `setTextColor` สำหรับกำหนดสีให้แก่ตัวอักษร
3. `setTextBackground` สำหรับกำหนดสีพื้นหลังของตัวอักษร

เมื่อตั้งค่าของจอแสดงผลแล้ว จึงทำการส่งข้อความ Row ตามด้วยหมายเลขบรรทัดซึ่งมาจากการเพิ่มค่าของตัวแปร `i` และมีการเลื่อนตำแหน่งตามค่าของ `i` ด้วย ดังนั้นที่บรรทัดแรก ข้อความ Row0 ถูกแสดงที่คอลัมน์ 0 ที่บรรทัด 2 แสดงข้อความ Row 1 ที่คอลัมน์ 1 ไปตามลำดับจนถึงบรรทัด 15 (บรรทัดที่ 16) จะแสดงเป็น Row 15 ที่คอลัมน์ 15

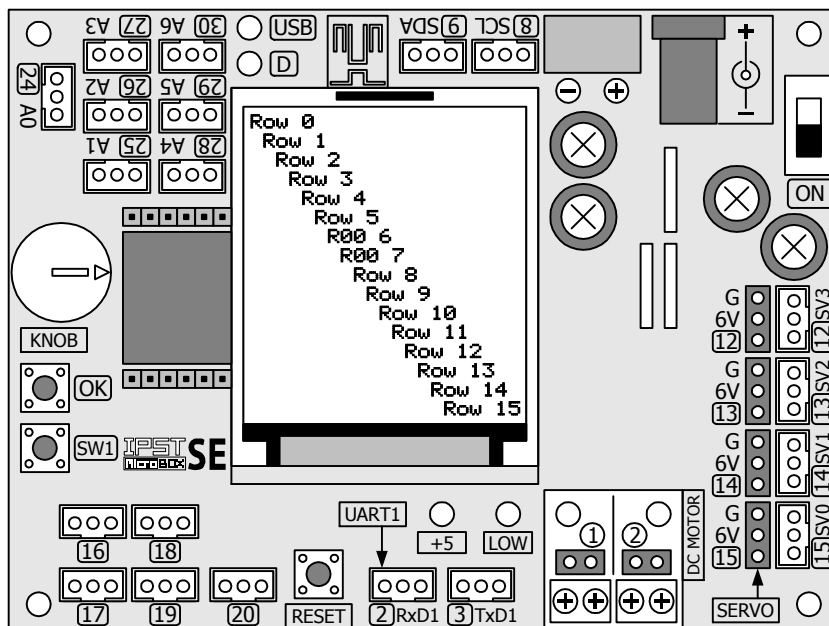
โปรแกรมที่ A1-2 ไฟล์ `microbox_GLCDmultipleLine.ino` สำหรับทดสอบการแสดงผลของแผงวงจร IPST-SE

(A1.2.1) เปิดซอฟต์แวร์ Arduino IDE 1.7.10 พิมพ์โปรแกรมที่ A1-2 แล้วบันทึกไฟล์

(A1.2.2) เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์

(A1.2.3) คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม  หรือเลือกที่เมนู File > Upload to Wiring Hardware

ที่หน้าจอแสดงผลกราฟิก LCD ของแผงวงจร IPST-SE แสดงข้อความ Row 0 ถึง Row 15 เรียงไปบรรทัดละข้อความ



กิจกรรมที่ 1-3 กำหนดขนาดตัวอักษรและทิศทางการแสดงผล

ขนาดตัวอักษรปกติที่แสดงบนจอแสดงผลของแผงวงจร IPST-SE เมื่อเริ่มต้นทำงานเป็นขนาดเล็กสุด ใช้จำนวนจุดต่อตัวอักษรคือ 6 x 10 จุด (อักษรจริงมีขนาด 5 x 7 จุด) ถ้าต้องการปรับขนาดตัวอักษรให้ใหญ่ขึ้น จะมีคำสั่ง `setTextSize` ไว้สำหรับปรับขนาด โดยค่าที่กำหนดจะเป็นจำนวนเท่าของตัวอักษรปกติ เช่น

`setTextSize(2)` หมายถึงขนาดตัวอักษรใหญ่ขึ้นเป็น 2 เท่า ใช้ 12 x 20 พิกเซลต่อ 1 ตัวอักษร

`setTextSize(3)` หมายถึงขนาดตัวอักษรใหญ่ขึ้นเป็น 3 เท่า ใช้ 18 x 30 พิกเซลต่อ 1 ตัวอักษร

เมื่อปรับขนาดตัวอักษรมีขนาดใหญ่ขึ้น จำนวนตัวอักษรต่อบรรทัดก็ต้องลดลง จากเดิม 21 ตัวอักษร 16 บรรทัด เมื่อขนาดของตัวอักษรเพิ่มขึ้นเป็นสองเท่า ก็จะทำให้แสดงได้ 10 ตัวอักษร 8 บรรทัดแทน ดังนั้นเมื่อเขียนโปรแกรมจะต้องคำนึงถึงค่าเหล่านี้ด้วย


```
#include <ipst.h>
int x,m;
void setup()
{
  //glcdSetColorWordRGB();           // หากสีของการแสดงผลมิต ให้เปิดใช้ฟังก์ชันนี้
  setTextColor(GLCD_RED);           // กำหนดสีตัวอักษรเป็นสีแดง
}
void loop()
{
  for (x=1;x<6;x++)
  {
    setTextSize(x);                 // กำหนดขนาดตัวอักษร
    for(m=0;m<4;m++)
    {
      glcdClear();                  // เคลียร์หน้าจอ
      glcdMode(m);                  // กำหนดทิศทาง
      glcd(0,0,"%dX",x);            // แสดงขนาดตัวอักษร
      glcd(1,0,"M=%d",m);          // แสดงโหมดทิศทาง
      sleep(500);
    }
  }
}
```

โปรแกรมที่ A1-3 ไฟล์ `microbox_GLCDtextFlip.ino` สำหรับทดสอบการเพิ่มขนาดตัวอักษรในการแสดงผล และการเปลี่ยนทิศทางของการแสดงผลของแผงวงจร IPST-SE

นอกจากขนาดตัวอักษรแล้ว ยังกำหนดทิศทางการแสดงผลของจอแสดงผลได้ โดยใช้คำสั่ง `glcdMode()` โดยมีค่าตั้งต้นคือ โหมด 0 (`glcdMode(0)`) นั่นคือ แสดงผลในแนวตั้ง สำหรับอีก 3 โหมดคือ โหมด 1, 2 และ 3 ใช้ปรับให้การแสดงผลหมุนไปโหมดละ 90 องศา นั่นคือ โหมด 1 หมุนไป 90 องศา, โหมด 2 หมุนไป 180 องศา และโหมด 3 หมุนไป 270 องศา

(A1.3.1) เปิดซอฟต์แวร์ Arduino IDE 1.7.10 พิมพ์โปรแกรมที่ A1-3 แล้วบันทึกไฟล์

(A1.3.2) เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์

(A1.3.3) คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม  หรือเลือกที่เมนู File > Upload to Wiring Hardware

ที่หน้าจแสดงผลของแผงวงจร IPST-SE แสดงข้อความแจ้งขนาดของตัวอักษรและโหมดการแสดงผลในทิศทางที่ต่างกัน เริ่มจากมุมบนซ้าย, มุมบนขวา, มุมล่างขวา และมุมล่างซ้าย โดยรอบการแสดงผลจะเริ่มจาก 1X, 2X, 3X, 4X และ 5X แต่ละรอบมีการแสดงผล 4 ทิศทาง โดยดูจากค่า M

$M = 0$ จอแสดงข้อความแนวตั้ง
ตัวอักษรขนาด 3 เท่า



$M = 1$ หมุนการแสดงผลไป 90 องศาทางขวา
ตัวอักษรขนาด 4 เท่า



$M = 2$ หมุนการแสดงผลไป 180 องศา
จะได้ภาพที่กลับหัวเมื่อเทียบกับ $M = 0$
ตัวอักษรขนาด 4 เท่า



$M = 3$ หมุนการแสดงผลไป 270 องศา
ตัวอักษรขนาด 5 เท่า



กิจกรรมที่ 1-4 แสดงลายเส้นกราฟิก

ฟังก์ชัน `glcd` เป็นฟังก์ชันหลักในการติดต่อกับจอแสดงผลกราฟิก LCD นอกจากนี้มีคำสั่งแสดงข้อความแล้ว ยังมีคำสั่งในการวาดลายเส้นกราฟิกอีกหลายคำสั่ง ประกอบด้วย

`glcdRect(int x1,int y1,int width,int height,uint color)` เป็นคำสั่งสร้างรูปสี่เหลี่ยม

`glcdFillRect(int x1,int y1,int width,int height,uint color)` เป็นคำสั่งสร้างพื้นที่สี่เหลี่ยม

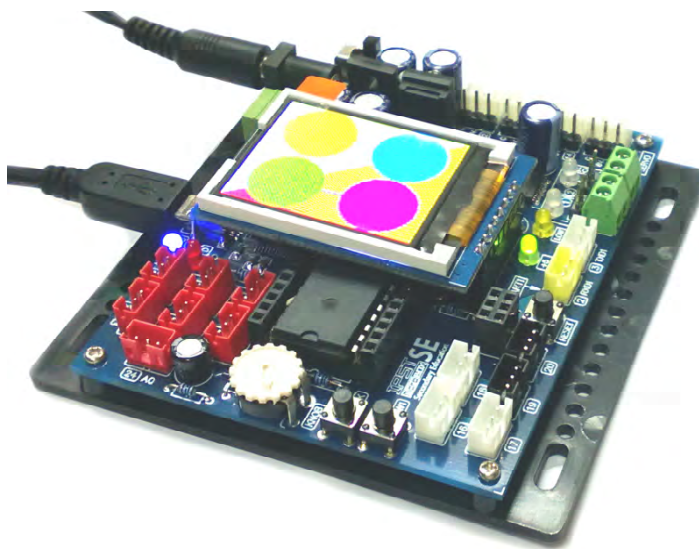
`glcdLine(int x1, int y1, int x2, int y2,uint color)` เป็นคำสั่งลากเส้น

`glcdCircle(int x, int y, int radius,uint color)` เป็นคำสั่งวาดเส้นวงกลม

`glcdFillCircle(int x, int y, int radius,uint color)` เป็นคำสั่งสร้างพื้นที่วงกลม

`glcdClear(uint color)` เป็นการเคลียร์หน้าจอแสดงผล

โดยทดสอบเขียนโปรแกรมได้ตั้งโปรแกรมที่ A1-4 แล้วอัปโหลดเพื่อทดสอบการทำงานไปยังแผงวงจร IPST-SE จะได้ผลดังรูป



ถ้าหากสีของการแสดงผลไม่ถูกต้อง อาจเกิดจากรุ่นของจอแสดงผลเนื่องจากจอภาพแบบนี้มี 2 รุ่น มีการเรียงข้อมูลที่สร้างสีแตกต่างกันเล็กน้อย ทางแก้ไขคือ นำเครื่องหมาย // ที่หน้าฟังก์ชัน `glcdSetColorWordRGB()`; ออก เพื่อเปลี่ยนรูปแบบการเรียงข้อมูลสี


```

#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
int i,j;
void setup()
{
  //glcdSetColorWordRGB(); // หากสีของการแสดงผลผิด ให้เปิดใช้ฟังก์ชันนี้
}
void loop()
{
  glcdClear; // เคลียร์หน้าจอและพื้นหลังเป็นสีดำ
  sleep(300);
  for (i=0;i<160;i+=4)
  {
    glcdLine(0,0,128,i,GLCD_WHITE); // วาดเส้นสีขาวจากพิกัด 0,0 ไปยังจุดที่กำหนด
  }
  for (i=0;i<128;i+=4)
  {
    glcdLine(0,0,i,160,GLCD_RED); // วาดเส้นสีแดงจากพิกัด 0,0 ไปยังจุดที่กำหนด
  }
  delay(2000);
  glcdRect(32,40,64,80,GLCD_BLUE); // วาดเส้นกรอบสี่เหลี่ยมสีน้ำเงิน
  delay(300);
  glcdFillCircle(32,40,31,GLCD_GREEN); // สร้างวงกลมพื้นสีเขียว
  glcdFillCircle(96,40,31,GLCD_YELLOW); // สร้างวงกลมพื้นสีเหลือง
  glcdFillCircle(32,120,31,GLCD_MAGENTA); // สร้างวงกลมพื้นสีบานเย็น
  glcdFillCircle(96,120,31,GLCD_SKY); // สร้างวงกลมพื้นสีฟ้า
  delay(1000);
  glcdCircle(64,40,31,GLCD_GREEN); // วาดเส้นรอบวงกลมสีเขียว
  glcdCircle(32,80,31,GLCD_BLUE); // วาดเส้นรอบวงกลมสีน้ำเงิน
  glcdCircle(64,120,31,GLCD_YELLOW); // วาดเส้นรอบวงกลมสีเหลือง
  glcdCircle(96,80,31,GLCD_SKY); // วาดเส้นรอบวงกลมสีฟ้า
  delay(1000);
  glcdFillRect(0,0,128,160,GLCD_YELLOW); // สร้างรูปสี่เหลี่ยมสีเหลือง
  delay(1000);
}

```

กิจกรรมที่ 1-5 ลากเส้นโค้ง


นอกจากวงกลมและสี่เหลี่ยมแล้ว เส้นโค้งก็เป็นส่วนประกอบสำคัญในการสร้างภาพกราฟิก ในชุดคำสั่งเกี่ยวกับการแสดงผลจอภาพแบบกราฟิกสีของแผงวงจร IPST-SE ยังมีคำสั่ง `glcdArc()` สำหรับสร้างเส้นโค้ง โดยมีพารามิเตอร์หรือตัวแปรที่ต้องกำหนดอยู่พอสมควร ดูรายละเอียดเพิ่มเติมในบทที่ 6 เกี่ยวกับการใช้งานจอแสดงผลแบบกราฟิก LCD สีตัวนี้

(A1.5.1) เปิดซอฟต์แวร์ Arduino IDE 1.7.10 พิมพ์โปรแกรมที่ A1-5 แล้วบันทึกไฟล์

(A1.5.2) เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์

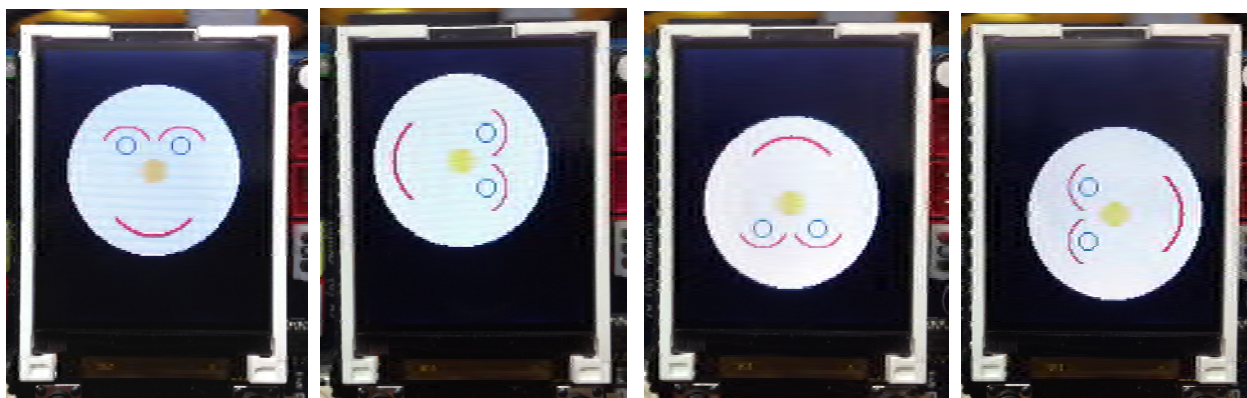
```
#include <ipst.h>
int i;
// ฟังก์ชันสร้างรูปหน้ายิ้ม
void face()
{
  glcdFillCircle(64,70,50,GLCD_WHITE);
  glcdArc(48,60,16,30,150,GLCD_RED);
  glcdCircle(48,55,5,GLCD_BLUE);
  glcdCircle(80,55,5,GLCD_BLUE);
  glcdArc(80,60,16,30,150,GLCD_RED);
  glcdFillCircle(64,70,7,GLCD_YELLOW);
  glcdArc(64,80,30,220,320,GLCD_RED);
  glcdArc(64,80,29,220,320,GLCD_RED);
}
void setup()
{
  //glcdSetColorWordRGB(); // หากสีของการแสดงผลผิดพลาด ให้เปิดใช้ฟังก์ชันนี้
}
void loop()
{
  for(i=0;i<4;i++)
  {
    glcdClear();
    glcdMode(i); // สุ่มมุมการแสดงผล
    face();
    sleep(1000);
  }
}
```

โปรแกรมที่ A1-5 ไฟล์ `microbox_GLCDarcTest.ino` สำหรับทดสอบการวาดเส้นโค้งของแผงวงจร IPST-SE

(A1.5.3) คอมไฟล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม  หรือเลือกที่เมนู File > Upload to Wiring Hardware

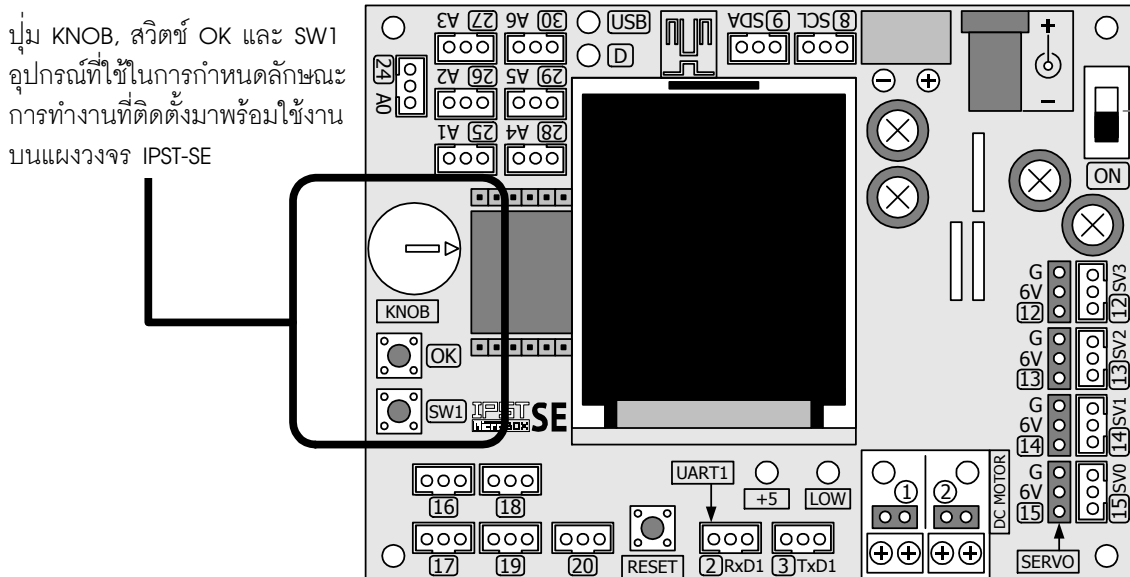
(A1.5.4) รันโปรแกรม ดูผลการทำงานที่จอแสดงผลของแผงวงจร IPST-SE

ที่จอแสดงผลแสดงเป็นรูปการ์ตูนหน้ายิ้มนาน 1 วินาที แล้วหมุนไปครั้งละ 90 องศา แล้ววนกลับมาที่หน้าเริ่มต้น จะวนแสดงผลไปตลอดเวลา



กิจกรรมที่ 2 อ่านค่าจากปุ่ม KNOB และสวิตช์ OK ของแผงวงจร IPST-SE

ในระบบควบคุมพื้นฐานจะต้องมีการปรับตั้งค่า มีเมนู มีสวิตช์ในการสั่งงานต่างๆ บนแผงวงจร IPST-SE ก็มีส่วนติดต่อกับผู้ใช้งานด้วยเช่นกัน ประกอบด้วยปุ่ม KNOB สำหรับปรับเลือกรายการ และสวิตช์ OK กับ SW1 สำหรับยืนยันการเข้าสู่รายการทางเลือกรุ่นๆ



เขียนโปรแกรมที่ A2-1 แล้วบันทึกไฟล์ชื่อ `microbox_KnobSwitchTest.ino` ทำการคอมไพล์และอัปโหลดไปยังแผงวงจร IPST-SE แล้วรันโปรแกรมทดสอบการทำงาน

เมื่อโปรแกรมเริ่มทำงาน ที่หน้าจอแสดงผลของแผงวงจร IPST-SE แสดงข้อความ

Press OK (ขนาดตัวอักษรใหญ่ขนาด 2x)

ให้กดสวิตช์ OK เพื่อทำงานต่อ

หน้าจอแสดงรูปร่างกลมสีเหลือง 1 วินาที จากนั้นแสดงข้อความ

Knob value (ขนาดตัวอักษรใหญ่ขนาด 2x)

XXXX (ขนาดตัวอักษรใหญ่ขึ้นเป็นขนาด 3x)

โดยที่ xxxx มีค่าได้ตั้งแต่ 94 ถึง 1023

ทดลองปรับปุ่ม KNOB บนแผงวงจร IPST-SE

ค่าของ Knob ที่จอแสดงผลจะต้องเปลี่ยนแปลงตามการปรับที่ปุ่ม KNOB

จากนั้นกดสวิตช์ OK แล้วปล่อย

หน้าจอแสดงรูปร่างกลมสีเขียว 1 วินาที แล้วกลับไปแสดงข้อความและค่าของ KNOB

ทดลองกดสวิตช์ SW1 แล้วปล่อย

หน้าจอแสดงรูปวงกลมสีแดง 1 วินาที แล้วกลับไปแสดงข้อความและค่าของ KNOB

```
#include <ipst.h> // มนากไฟล์ไลบรารีหลัก
void setup()
{
  //glcdSetColorWordRGB(); // หากสีของการแสดงผลให้ใช้ฟังก์ชันนี้
  glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
  glcd(1,1,"Press OK"); // แสดงข้อความออกหน้าจอแสดงผล
  sw_OK_press(); // วนรอจนกระทั่งกดสวิตช์ OK
  glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
  glcdFillCircle(64,70,50,GLCD_YELLOW); // วาดวงกลมสีเหลือง
  delay(1000); // แสดงผลวงกลมสีเหลือง 1 วินาที
  glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
}
void loop()
{
  if (sw_OK()) // ตรวจสอบการกดสวิตช์ OK
  {
    glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
    glcdFillCircle(64,70,50,GLCD_GREEN); // วาดวงกลมสีเขียว
    delay(1000); // แสดงผลวงกลมสีเขียว 1 วินาที
    glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
  }
  if(sw1()) // ตรวจสอบสวิตช์ SW1 ว่ามีการกดหรือไม่
  {
    glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
    glcdFillCircle(64,70,50,GLCD_RED); // วาดวงกลมสีแดง
    delay(1000); // แสดงผลวงกลมสีแดง 1 วินาที
    glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
  }
  glcd(1,0,"Knob value"); // แสดงข้อมูลที่จอแสดงผล
  setTextSize(3); // เลือกขนาดตัวอักษรใหญ่เป็น 3 เท่าจากขนาดปกติ
  glcd(2,2,"%d ",knob()); // แสดงค่าที่อ่านได้จากการปรับปุ่ม KNOB ที่หน้าจอแสดงผล
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}
```

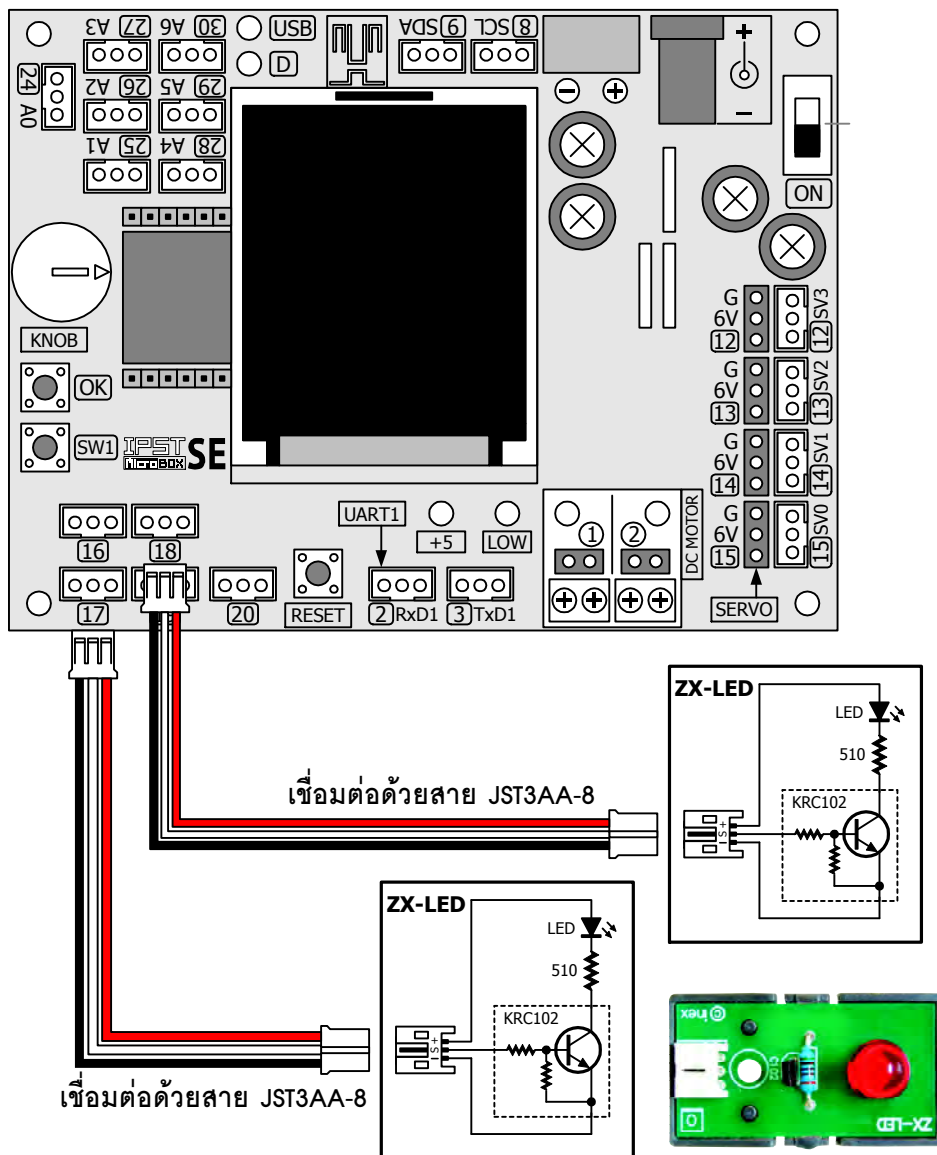
โปรแกรมที่ A2-1 ไฟล์ microbox_KnobSwitchTest.ino สำหรับทดสอบอ่านค่าจากปุ่ม KNOB, สวิตช์ OK และ SW1

กิจกรรมที่ 3 ขับอุปกรณ์เอาต์พุตอย่างง่าย

ในไฟล์ไลบรารี ipst.h มีคำสั่ง `out(int num, int _dat)` ซึ่งช่วยให้สามารถส่งลอจิก “0” หรือ “1” ออกไปยังขาพอร์ตที่ต้องการได้ ทำให้นำแผงวงจร IPST-SE ไปใช้ขั้วอุปกรณ์เอาต์พุตพื้นฐานได้ง่ายขึ้น ยกตัวอย่างง่ายที่สุดคือ ไดโอดเปล่งแสงหรือ LED

ในกิจกรรมนี้จึงนำแผงวงจร ZX-LED อันเป็นแผงวงจร LED แสดงผลแบบเดี่ยวที่จะติดสว่างเมื่อได้รับลอจิก “1” และดับลงเมื่อได้รับลอจิก “0” มาต่อกับแผงวงจร IPST-SE เพื่อทดลองใช้งาน

(A3.1) นำ ZX-LED ชุดที่ 1 ต่อเข้ากับจุดต่อพอร์ต 17 และชุดที่ 2 ต่อกับจุดต่อพอร์ต 18



```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
  setTextSize(2);          // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
  lcd(1,1,"Press OK");    // แสดงข้อความออกหน้าจอ GLCD
  sw_ok_press();          // วนรอกการกดสวิตช์ OK
}
void loop()
{
  out(17,1);               // ทำให้ LED ที่ต่ออยู่กับพอร์ต 17 ติดสว่าง
  out(18,0);               // ทำให้ LED ที่ต่ออยู่กับพอร์ต 18 ดับ
  sleep(400);
  out(17,0);               // ทำให้ LED ที่ต่ออยู่กับพอร์ต 17 ดับ
  out(18,1);               // ทำให้ LED ที่ต่ออยู่กับพอร์ต 18 ติดสว่าง
  delay(400);
}
```

โปรแกรมที่ A3-1 ไฟล์ microbox_LEDtest.ino สำหรับทดสอบการขับอุปกรณ์เอาต์พุตอย่างง่าย

(A3.2) เขียนโปรแกรมที่ A3-1 บันทึกไฟล์ชื่อ microbox_LEDtest.ino

(A3.3) เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE จากนั้นทำการคอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE

(A3.4) รันโปรแกรม สังเกตการทำงานของ ZX-LED

เมื่อรันโปรแกรม ที่หน้าจอแสดงผลของแผงวงจร IPST-SE แสดงข้อความ

Press OK

ให้กดสวิตช์ OK เพื่อเริ่มการทำงาน จะเห็น LED บนแผงวงจร ZX-LED ทั้งสองชุดติดและดับสลับกันไปอย่างต่อเนื่อง

กิจกรรมที่ 4 ขับเสียงออกลำโพงเป็ยไซ

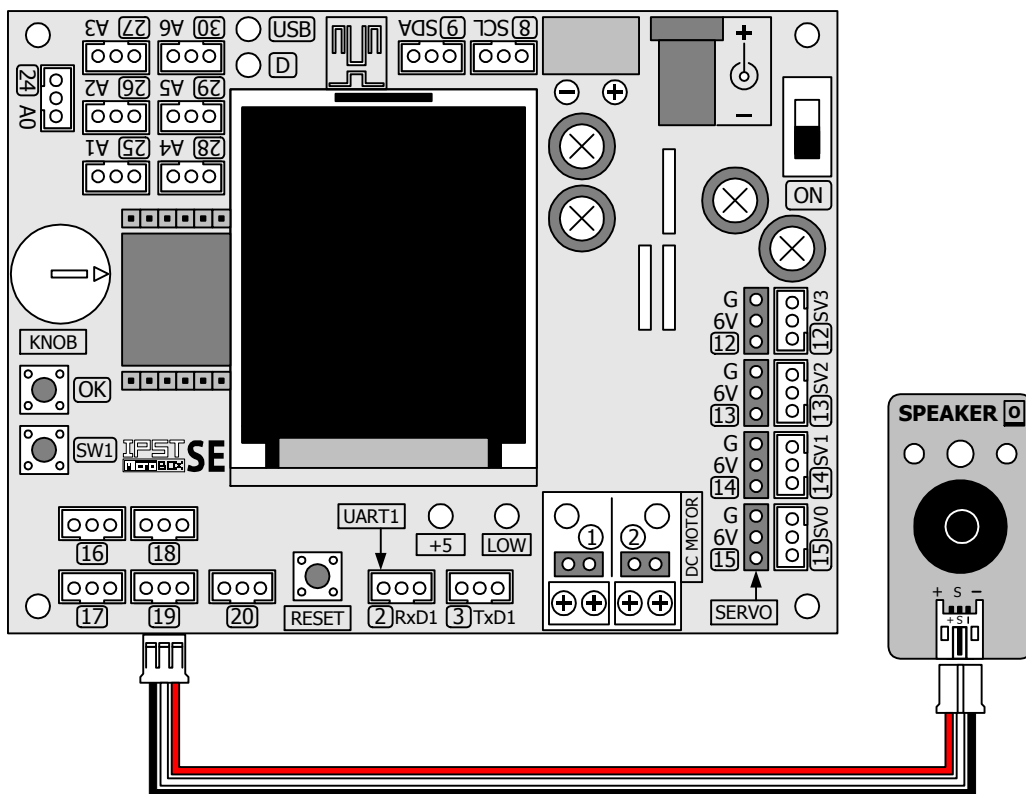
ในชุดกล่องสมองกล IPST-MicroBOX (SE) มีแผงวงจรขับเสียงโดยใช้ลำโพงเป็ยไซ (ZX-SPEAKER) โดยตัวลำโพงเป็ยไซขนาดเล็กนี้ตอบสนองความถี่เสียงในช่วงความถี่ประมาณ 300 ถึง 3,000Hz ในการเขียนโปรแกรมเพื่อสั่งงานแผงวงจร IPST-SE ให้ขับเสียงออกทางแผงวงจร ZX-SPEAKER นี้ทำได้โดยใช้คำสั่ง `beep()` และ `sound()`



ในโปรแกรมที่ A4-1 เป็นตัวอย่างการใช้คำสั่ง `beep()` เพื่อขับเสียง “ติ๊ด” ความถี่เดียวออกทางลำโพงทุกๆ 1 วินาที

ส่วนในโปรแกรมที่ A4-2 เป็นตัวอย่างการใช้คำสั่ง `sound()` เพื่อขับเสียงที่มีความถี่ตามที่กำหนดออกทางลำโพงเป็ยไซ ตามเวลาที่กำหนดในโปรแกรม

(A4.1) นำแผงวงจรขับเสียง ZX-SPEAKER ต่อเข้ากับจุดต่อพอร์ต 19 ของแผงวงจร IPST-SE




```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
}
void loop()
{
  beep(19);                 // ขับเสียง “ดีด” ออกลำโพงผ่านทางจุดต่อพอร์ต 19
  delay(1000);
}
```

โปรแกรมที่ A4-1 ไฟล์ microbox_BeepTest.ino สำหรับทดสอบขับเสียงออกลำโพงของแผงวงจร IPST-SE

```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
}
void loop()
{
  sound(19,500,500);       // ขับเสียงความถี่ 500Hz นาน 0.5 วินาที ออกลำโพงผ่านทางจุดต่อพอร์ต 19
  sound(19,2500,500);     // ขับเสียงความถี่ 2500Hz นาน 0.5 วินาที ออกลำโพงผ่านทางจุดต่อพอร์ต 19
}
```

โปรแกรมที่ A4-2 ไฟล์ microbox_SoundTest.ino สำหรับทดสอบการขับเสียงออกลำโพงของแผงวงจร IPST-SE แบบกำหนดความถี่และเวลาได้

(A4.2) เขียนโปรแกรมที่ A4-1 บันทึกไฟล์ชื่อ microbox_BeepTest.pde

(A4.3) เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE จากนั้นทำการคอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE

(A4.4) รันโปรแกรม

จะได้ยินเสียง “ดีด” ดังเป็นจังหวะในทุกๆ 1 วินาทีจากลำโพงของแผงวงจร ZX-SPEAKER

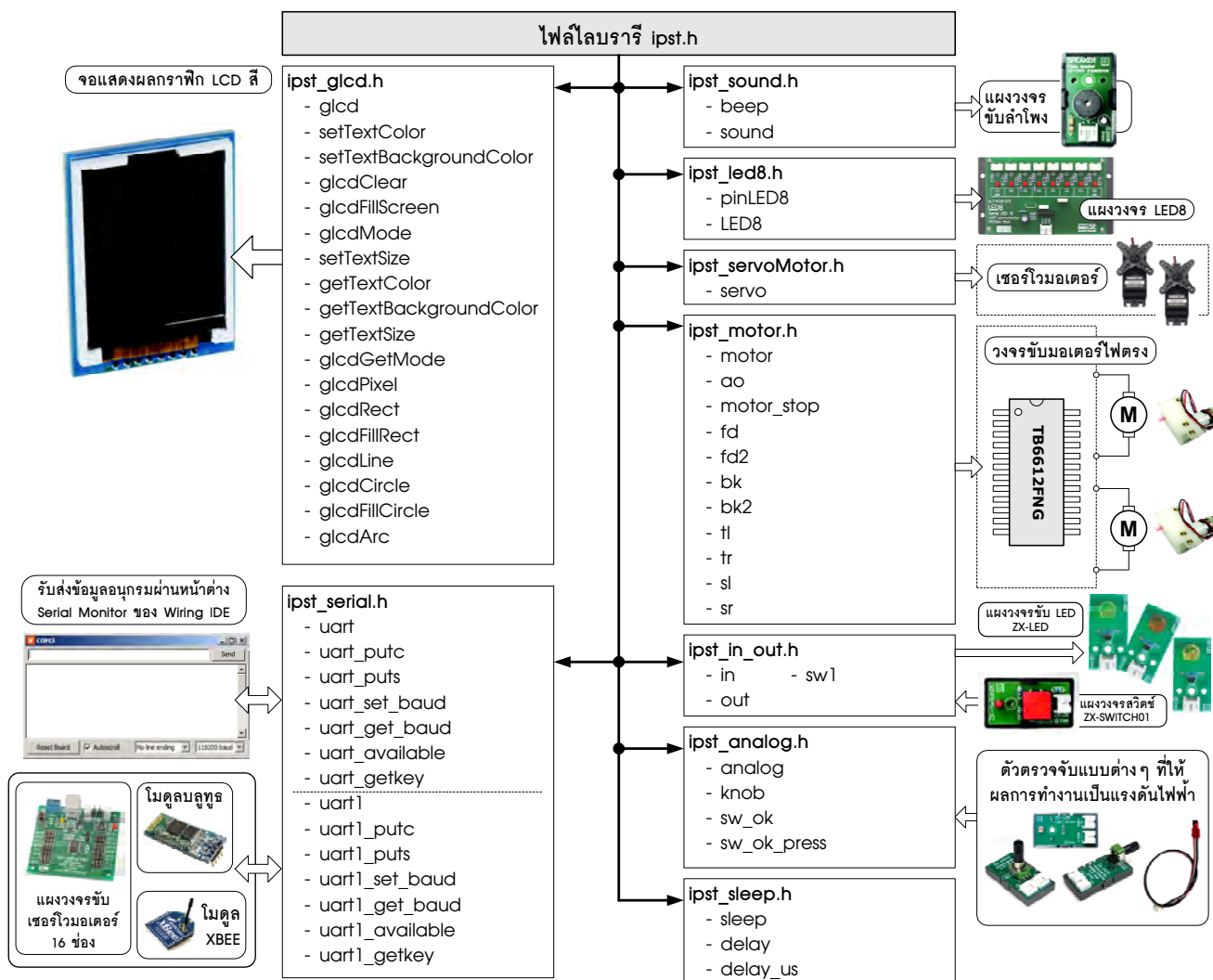
(A4.5) เขียนโปรแกรมที่ A4-2 บันทึกไฟล์ชื่อ microbox_SoundTest.ino แล้วอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE เพื่อทดสอบการทำงานอีกครั้ง

จะได้ยินสัญญาณเสียง 2 ความถี่ดังสลับกันออกจากลำโพงของแผงวงจร ZX-SPEAKER

บทที่ 5

ความรู้เบื้องต้นเกี่ยวกับไฟล์ไลบรารี ของชุดกล่องสมองกล IPST MicroBOX SE

การพัฒนาโปรแกรมภาษา C/C++ สำหรับชุดกล่องสมองกล IPST-MicroBOX (SE) ดำเนินการภายใต้การสนับสนุนของไฟล์ไลบรารี ipst.h ทั้งนี้เพื่อช่วยลดขั้นตอนและความซับซ้อนในการเขียนโปรแกรมเพื่อความคุมส่วนต่างๆ ของฮาร์ดแวร์ลง โครงสร้างของไฟล์ไลบรารี ipst.h แสดงดังรูป



5.1 ไฟล์ไลบรารี ipst.h

ในการเรียกใช้งานชุดคำสั่งย่อยต่างๆ เพื่อการพัฒนาโปรแกรมควบคุมสำหรับชุดกล่องสมองกล IPST-MicroBOX (SE) ผู้พัฒนาต้องผนวกไฟล์ไลบรารีหลัก ipst.h ไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst.h>
```

เพื่อประกาศให้ให้ตัวแปลภาษา C หรือคอมไพเลอร์รู้จักชุดคำสั่งย่อยที่กำลังจะถูกเรียกใช้งานจากไฟล์ไลบรารี ipst.h

ไลบรารีย่อยของไฟล์ไลบรารี ipst.h ประกอบด้วย

- **ipst_glcd.h** บรรจุฟังก์ชันและคำสั่งสำหรับแสดงผลข้อความ, ตัวเลข และสร้างภาพกราฟิกสีที่จอแสดงผลแบบกราฟิก LCD สีบนแผงวงจร IPST-SE (ยังไม่รองรับการทำงานกับไฟล์รูปภาพ) ฟังก์ชันนี้มีการกำหนดการใช้งานที่เฉพาะเจาะจง (มีรายละเอียดอธิบายในบทที่ 6)

- **ipst_sleep.h** บรรจุฟังก์ชันและคำสั่งสำหรับการหน่วงเวลา

- **ipst_in_out.h** บรรจุฟังก์ชันและคำสั่งสำหรับอ่านค่าอินพุตดิจิทัลและส่งค่าออกทางขาพอร์ตเอาต์พุตดิจิทัล

- **ipst_analog.h** บรรจุฟังก์ชันและคำสั่งอ่านค่าจากอินพุตอะนาล็อกช่อง A0 ถึง A6 ที่ต่อกับตัวตรวจจับและปุ่ม KNOB กับสวิตช์ OK

- **ipst_sound.h** บรรจุฟังก์ชันและคำสั่งสำหรับสร้างเสียงเพื่อขับออกลำโพง

- **ipst_motor.h** บรรจุฟังก์ชันและคำสั่งสำหรับขับมอเตอร์ไฟตรง 2 ช่อง ต้องทำงานร่วมกับวงจรขับมอเตอร์ที่ใช้ไอซี TB6612 และใช้ไฟเลี้ยงอีกส่วนหนึ่งสำหรับเลี้ยงมอเตอร์ไฟตรง

- **ipst_servoMotor.h** บรรจุฟังก์ชันและคำสั่งสำหรับขับเซอร์โวมอเตอร์ ต้องทำงานร่วมกับเซอร์โวมอเตอร์ และต้องใช้ไฟเลี้ยง +6V อีกส่วนหนึ่งสำหรับเซอร์โวมอเตอร์ ฟังก์ชันนี้มีการกำหนดการใช้งานที่เฉพาะเจาะจง

- **ipst_serial.h** บรรจุฟังก์ชันและคำสั่งสำหรับสื่อสารข้อมูลอนุกรมผ่านทางพอร์ต USB และผ่านทางจุดต่อพอร์ต TxD1 และ RxD1 ของแผงวงจร IPST-SE

- **ipst_led8.h** บรรจุฟังก์ชันและคำสั่งสำหรับแสดงผล LED 8 ดวงบนแผงวงจร ZX-LED8 โดยสั่งให้ LED ติดทีละดวง หรือหลายดวงก็ได้

5.2 ไฟล์ไลบรารีของ Arduino 1.x

Arduino 1.x มีไลบรารีจำนวนมากเพื่อช่วยให้การเขียนโปรแกรมเพื่อติดต่อและใช้งานอุปกรณ์ฟังก์ชันพิเศษ ตลอดจนตัวตรวจจับแบบต่างๆ ทำให้สะดวกและง่ายขึ้น สรุปได้ดังนี้

1. **EEPROM.h** ไลบรารีสำหรับการจัดการหน่วยความจำข้อมูลอีอีพรอมภายในไมโครคอนโทรลเลอร์ ATmega644P ที่ใช้ในแผงวงจร IPST-SE
2. **LiquidCrystal.h** ไลบรารีติดต่อกับโมดูล LCD แบบตัวอักษร
3. **Encoder.h** ไลบรารีสำหรับการใช้งานวงจรเข้ารหัสขั้นพื้นฐาน (ต้องการอุปกรณ์เฉพาะ)
4. **Wire.h** ไลบรารีสำหรับติดต่อกับอุปกรณ์ระบบบัส 2 สายและบัส I²C
5. **SoftwareSerial.h** ไลบรารีสำหรับสื่อสารข้อมูลอนุกรมกับอุปกรณ์ภายนอกผ่านทางขาพอร์ตดิจิทัลปกติ
6. **OneWire.h** ไลบรารีสำหรับติดต่อกับอุปกรณ์บัสหนึ่งสาย (เป็นไลบรารีเสริม)

ในการเรียนรู้เพื่อใช้งานแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE) จะใช้ไฟล์ไลบรารีทั้งแบบมาตรฐานที่มากับซอฟต์แวร์ Arduino 1.x และไฟล์ไลบรารี ipst.h ร่วมกันเพื่อช่วยให้การพัฒนาโปรแกรมสำหรับการใช้งานมีประสิทธิภาพสูงสุด และทำความเข้าใจได้ง่าย ทั้งนี้เพื่อประโยชน์ในการต่อยอดการเรียนรู้ของผู้ใช้งานในวงกว้าง

5.3 การเรียกใช้ไลบรารี ipst.h และไลบรารีย่อย

ไฟล์ไลบรารี ipst.h ประกอบด้วยไลบรารีหลายไฟล์ การเรียกใช้งานทำได้ 2 วิธีคือ

1. เรียกใช้ทั้งหมดผ่านทาง **#include <ipst.h>** วิธีนี้ง่าย สะดวก และลดโอกาสที่ผู้เขียนโปรแกรมจะลืมผนวกไฟล์ไลบรารีที่ต้องใช้งาน
2. เรียกใช้เฉพาะไลบรารีที่ต้องการ ด้วยคำสั่ง **#include <ipst_XXXX.h>** ตัวอักษร XXXX คือชื่อใดๆ ของไฟล์ไลบรารีภายในไฟล์ ipst.h ดูได้จากหัวข้อ 5.1 ในบทนี้ วิธีนี้มีข้อดีคือ ลดขนาดของไฟล์โปรแกรมภาษา C/C++ ลง เนื่องจากมีการเรียกใช้งานฟังก์ชันหรือชุดคำสั่งเท่าที่จำเป็น

ในหัวข้อต่อไปนี้เป็นารอธิบายถึงข้อมูลสำหรับผู้พัฒนาโปรแกรมสำหรับชุดกล่องสมองกล IPST-MicroBOX (SE) ควรทราบเกี่ยวกับฟังก์ชันสำคัญที่อยู่ในไฟล์ไลบรารีย่อยของไลบรารีหลัก ipst.h

5.4 ipst_sleep.h ไฟล์ไลบรารีย่อการหน่วงเวลา

เป็นไฟล์ไลบรารีบรรจุชุดคำสั่งเกี่ยวกับการหน่วงเวลา ก่อนเรียกใช้งานควรผนวกไฟล์ไลบรารีไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_sleep.h> หรือเรียก #include <ipst.h>
```

5.4.1 sleep และ delay

sleep และ **delay** เป็นฟังก์ชันหน่วงเวลาโดยประมาณในหน่วยมิลลิวินาที มีรายละเอียดดังนี้

รูปแบบ

```
void sleep(unsigned int ms)
```

```
void delay(unsigned int ms)
```

พารามิเตอร์

ms - กำหนดค่าเวลาที่ต้องการหน่วงในหน่วยมิลลิวินาที มีค่า 0 ถึง 65,535

ตัวอย่างที่ 5-1

```
sleep(20); // หน่วงเวลาประมาณ 20 มิลลิวินาที
```

```
delay(1000); // หน่วงเวลาประมาณ 1 วินาที
```

5.4.2 delay_us

เป็นฟังก์ชันหน่วงเวลาโดยประมาณภายในโปรแกรมในหน่วยไมโครวินาที

รูปแบบ

```
void delay_us(unsigned int us)
```

พารามิเตอร์

us - กำหนดค่าเวลาที่ต้องการหน่วงในหน่วยไมโครวินาที มีค่า 0 ถึง 65,535

ตัวอย่างที่ 5-2

```
delay_us(100); // หน่วงเวลาประมาณ 100 ไมโครวินาที
```

5.5 ipst_sound.h ไฟล์ไลบรารีสร้างสัญญาณเสียง

เป็นไฟล์ไลบรารีบรรจุชุดคำสั่งเพื่อสร้างและขับสัญญาณเสียงไปยังลำโพงหรือแผงวงจรลำโพงเปียโซ ก่อนเรียกใช้งานควรผนวกไฟล์ไลบรารีไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_sound.h> หรือเรียก #include <ipst.h>
```

5.5.1 beep

เป็นฟังก์ชันกำเนิดเสียง “ตืด” มีความถี่ 500Hz นาน 100 มิลลิวินาที เพื่อขับออกลำโพงเปียโซ ต้องต่อวงจรขับลำโพงเปียโซหรือแผงวงจร ZX-SPEAKER เข้าที่จุดต่อพอร์ตใดๆ ของแผงวงจร IPST-SE

รูปแบบ

```
void beep(int pin)
```

พารามิเตอร์

pin - ขาพอร์ตใดๆ ของแผงวงจร IPST-SE มีค่า 0 ถึง 30 (แนะนำให้ใช้จุดต่อพอร์ต 19 หรือ 20)

ตัวอย่างที่ 5-3

```
beep(19); // กำเนิดเสียงความถี่ 500Hz นาน 100 มิลลิวินาทีออกทางจุดต่อพอร์ต 19
```

5.5.2 sound

เป็นฟังก์ชันกำเนิดสัญญาณเสียงที่กำหนดความถี่, ระยะเวลาในการกำเนิดสัญญาณ และจุดต่อพอร์ตที่เลือกใช้งานได้

รูปแบบ

```
void sound(int pin, int freq, int time)
```

พารามิเตอร์

pin - ขาพอร์ตใดๆ ของแผงวงจร IPST-SE มีค่า 0 ถึง 30 (แนะนำให้ใช้จุดต่อพอร์ต 19 หรือ 20)

freq - กำหนดความถี่สัญญาณเสียง มีค่า 0 ถึง 32,767

time - กำหนดค่าเวลาในการกำเนิดสัญญาณเสียงในหน่วย 1 มิลลิวินาที มีค่า 0 ถึง 32,767

ตัวอย่างที่ 5-4

```
sound(19,1200,500); // กำเนิดสัญญาณเสียงออกทางจุดต่อพอร์ต 19
// ด้วยความถี่ 1200Hz นาน 500 มิลลิวินาที
```

5.6 ipst_in_out.h ไฟล์ไลบรารีย่อยสำหรับติดต่อพอร์ต

เป็นไลบรารีที่มีคำสั่งเกี่ยวกับการอ่านและเขียนค่ากับพอร์ตของแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE) ก่อนเรียกใช้งานฟังก์ชันต้องผนวกไฟล์ไลบรารีนี้ไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_in_out.h> หรือเรียก #include <atx.h>
```

ฟังก์ชันที่สำคัญของไฟล์ไลบรารีนี้ประกอบด้วย

5.6.1 in

เป็นฟังก์ชันอ่านค่าสถานะลอจิกของพอร์ตที่กำหนด

รูปแบบ

```
char in(x)
```

พารามิเตอร์

x - กำหนดขาพอร์ตที่ต้องการอ่านค่า มีค่าตั้งแต่ 0 ถึง 50 สำหรับ IPST-SE ใช้ได้ถึง 30

หมายเหตุ : ไม่แนะนำให้ใช้ฟังก์ชันนี้กับจุดต่อ 19 และ 20 บนแผงวงจร IPST-SE

การคืนค่า

เป็น 0 หรือ 1

ตัวอย่างที่ 5-5

```
char x;           // ประกาศตัวแปร x เพื่อเก็บค่าผลลัพธ์จากการอ่านค่าระดับสัญญาณ
x = in(16);       // อ่านค่าดิจิตอลจากพอร์ตหมายเลข 16 แล้วเก็บค่าไว้ที่ตัวแปร x
```

5.6.2 out

เป็นฟังก์ชันกำหนดระดับสัญญาณหรือข้อมูลดิจิตอลไปยังขาพอร์ตที่กำหนด

รูปแบบ

```
out(char _bit,char _dat)
```

พารามิเตอร์

_bit - กำหนดขาพอร์ตที่ต้องการ มีค่า 0 ถึง 50 สำหรับ IPST-SE ใช้ได้ถึง 30

_dat - กำหนดข้อมูลที่ต้องการส่งออก มีค่าเป็น 0 หรือ 1

ตัวอย่างที่ 5-6

```
out(17,1);       // กำหนดให้ขาพอร์ต 17 เป็น "1"
out(18,0);       // กำหนดให้ขาพอร์ต 18 เป็น "0"
```


5.5.3 sw1_press

เป็นฟังก์ชันตรวจสอบการกดสวิตช์ SW1 บนแผงวงจร IPST-SE ต้องรอจนกระทั่ง SW1 ถูกปล่อยหลังจากมีการกดสวิตช์ จึงจะผ่านขั้นตอนการทำงานของฟังก์ชันนี้ไป

รูปแบบ

```
void sw1_press()
```

ตัวอย่างที่ 5-7

```
.....
sw1_press();    // รอจนกระทั่งสวิตช์ SW1 ถูกกดและปล่อย
.....
```

5.5.4 sw1

เป็นฟังก์ชันตรวจสอบการกดสวิตช์ SW1 ในขณะใดๆ

รูปแบบ

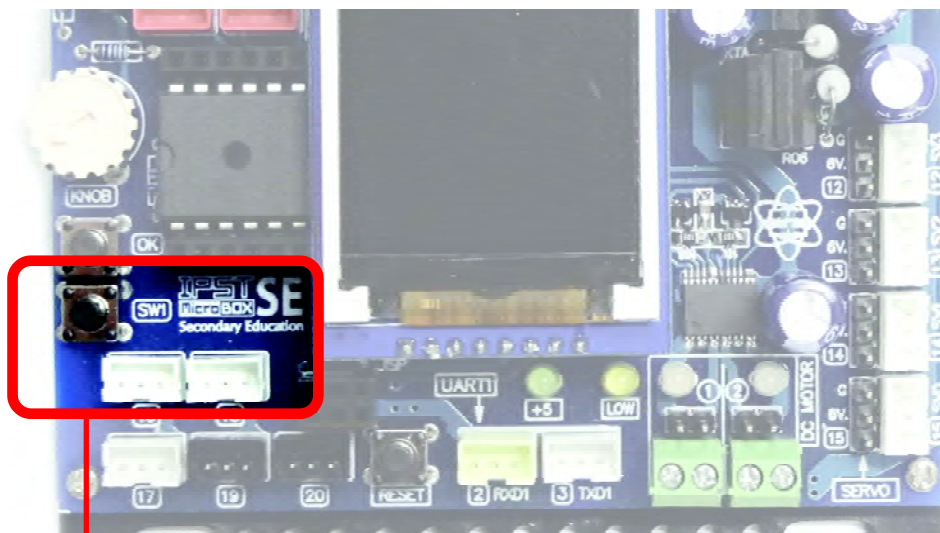
```
char sw1()
```

การคืนค่า

เป็น “0” เมื่อสวิตช์ SW1 ถูกกด และ เป็น “1” เมื่อไม่มีการกดสวิตช์ SW1

ตัวอย่างที่ 5-8

```
char x;          // ประกาศตัวแปร x เพื่อเก็บค่าผลลัพธ์จากการอ่านค่าดิจิตอล
x = sw1();      // อ่านค่าสถานะของสวิตช์ SW1 มาเก็บไว้ที่ตัวแปร x
```

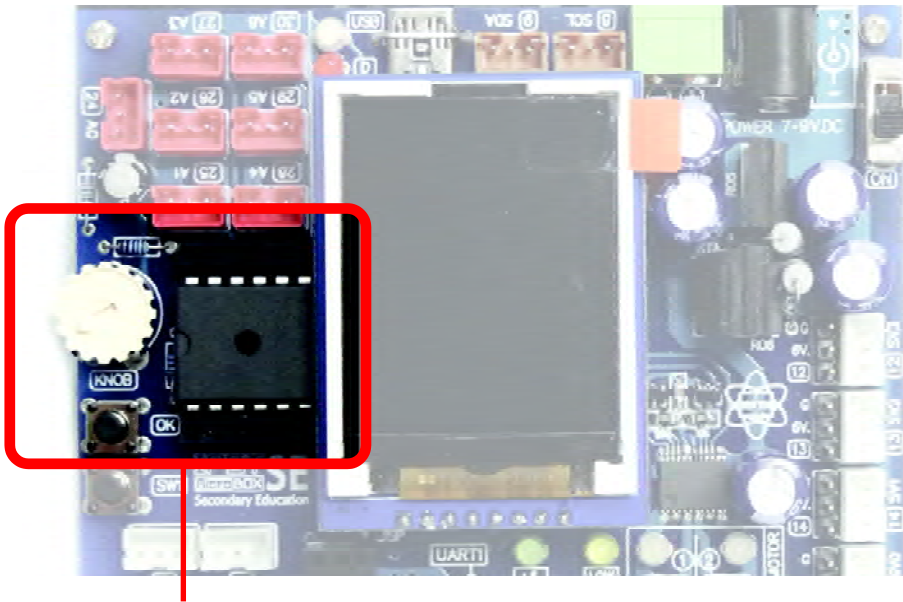


สวิตช์ SW1 อ่านค่าด้วยฟังก์ชัน sw1() และ sw1_press()

5.6 ipst_analog.h ไฟล์ไลบรารีจัดการสัญญาณอะนาล็อก

เป็นไฟล์ไลบรารีที่บรรจุชุดคำสั่งที่เกี่ยวกับการอ่านค่าอินพุตอะนาล็อก (A0 ถึง A6 และปุ่ม KNOB) ของแผงวงจร IPST-SE ซึ่งใช้ในการเชื่อมต่อกับตัวตรวจจับที่ให้ผลการทำงานในรูปแบบแรงดันไฟฟ้า ในย่าน 0 ถึง +5V ก่อนเรียกใช้งานต้องผนวกไฟล์ไลบรารีไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_analog.h> หรือ #inlue <ipst.h>
```



ปุ่ม KNOB และสวิตช์ OK บนแผงวงจร IPST-SE ที่อ่านค่าด้วยฟังก์ชัน knob() และ sw_OK() กับ sw_OK_press()

5.6.1 analog

เป็นฟังก์ชันอ่านค่าจากการแปลงสัญญาณอะนาล็อกของแผงวงจร IPST-SE ที่จุดต่อ A0 ถึง A6

รูปแบบ

```
unsigned int analog(unsigned char channel)
```

พารามิเตอร์

channel - กำหนดช่องอินพุตที่ต้องการ มีค่า 0 ถึง 6 ซึ่งตรงกับขาพอร์ต A0 ถึง A6

การคืนค่า

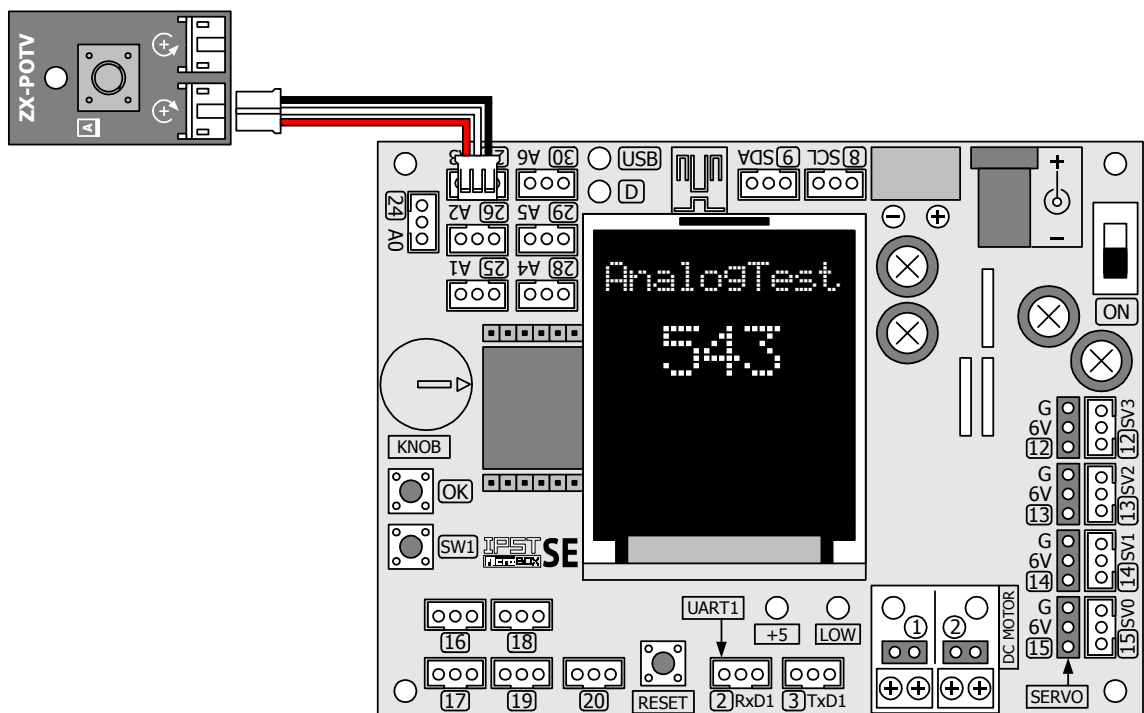
เป็นข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้า 0 ถึง +5V จากช่องอินพุตที่กำหนด มีค่า 0 ถึง 1,023

ตัวอย่างที่ 5-9

```

#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
int val=0; // กำหนดตัวแปรสำหรับเก็บค่าที่ได้จากการแปลงสัญญาณแล้ว
void setup()
{
  glcdClear();
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}
void loop()
{
  glcd(1,0,"AnalogTest"); // แสดงข้อความที่จอแสดงผล
  val = analog(2); // อ่านค่าของสัญญาณช่อง A2 มาเก็บไว้ที่ตัวแปร val
  setTextSize(3); // เลือกขนาดตัวอักษรใหญ่เป็น 3 เท่าจากขนาดปกติ
  glcd(2,2,"%d ",val); // แสดงค่าที่อ่านได้จากจุดต่อ A2 ที่หน้าจอแสดงผล
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}

```



5.6.2 knob()

เป็นฟังก์ชันอ่านค่าข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้าที่ขาพอร์ต A7 ซึ่งต่อกับตัวต้านทานปรับค่าได้ที่ตำแหน่ง KNOB

รูปแบบ

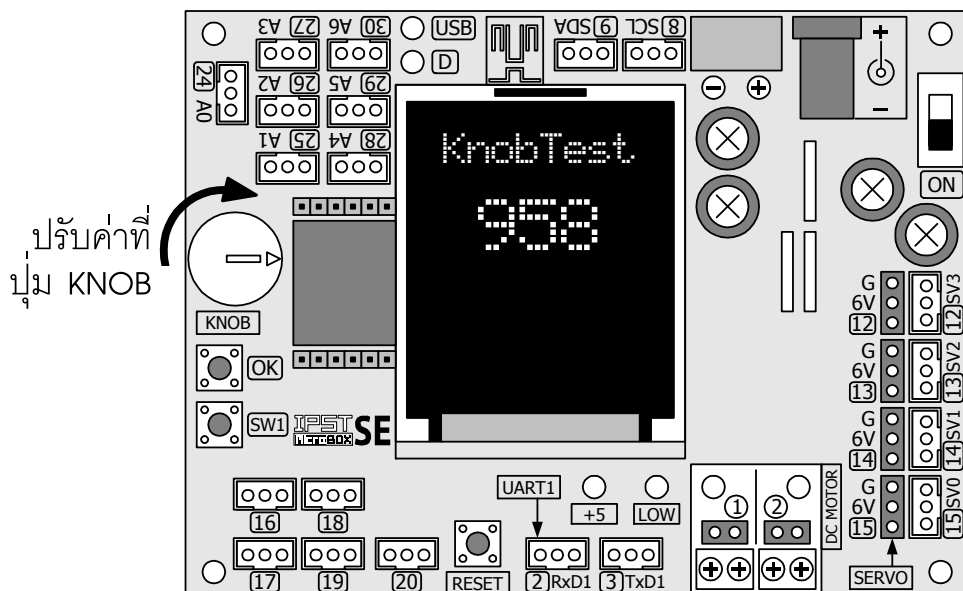
unsigned int knob()

การคืนค่า

เป็นข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้าที่มาจากค่าที่ปุ่ม KNOB บนแผงวงจร IPST-SE มีค่า 95 ถึง 1,023

ตัวอย่างที่ 5-10

```
#include <ipst.h> // มนวกไฟล์ไลบรารีหลัก
void setup()
{
  lcdClear();
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}
void loop()
{
  lcd(1,0," KnobTest"); // แสดงข้อความที่จอแสดงผล
  setTextSize(3); // เลือกขนาดตัวอักษรใหญ่เป็น 3 เท่าจากขนาดปกติ
  lcd(2,2,"%d ",knob()); // แสดงค่าที่อ่านได้จากปุ่ม KNOB ที่จอแสดงผล
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}
```



5.6.3 sw_OK()

เป็นฟังก์ชันตรวจสอบสถานะสวิตช์ OK บนแผงวงจร IPST-SE โดยให้สถานะ “เป็นจริง” เมื่อมีการกดสวิตช์และ “เป็นเท็จ” เมื่อไม่มีการกดสวิตช์

รูปแบบ

```
unsigned char sw_ok()
```

การคืนค่า

1 (เป็นจริง) เมื่อมีการกดสวิตช์ 0 (เป็นเท็จ) เมื่อไม่มีการกดสวิตช์

หมายเหตุ การกดสวิตช์ OK มีผลทำให้ค่าที่อ่านได้จาก knob() เป็น 0 ด้วย

ตัวอย่างที่ 5-11

```
#include <ipst.h>                                     // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
    glcdClear();
}
void loop()
{
    if (sw_OK())                                       // ตรวจสอบการกดสวิตช์ OK
    {
        glcdFillScreen(GLCD_YELLOW);                // เปลี่ยนสีพื้นเป็นสีเหลือง
        delay(3000);                                 // แสดงสีพื้นใหม่นาน 3 วินาที
    }
    glcdClear();                                       // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
}

```

5.6.4 sw_OK_press()

เป็นฟังก์ชันตรวจสอบการกดสวิตช์ OK บนแผงวงจร IPST-SE ต้องรอจนกระทั่งสวิตช์ถูกปล่อยหลังจากการกดสวิตช์จึงจะผ่านฟังก์ชันนี้ไปกระทำคำสั่งอื่นๆ

ตัวอย่างที่ 5-12

```
.....
sw_OK_press();   // รอจนกระทั่งเกิดกดสวิตช์ OK
.....
```

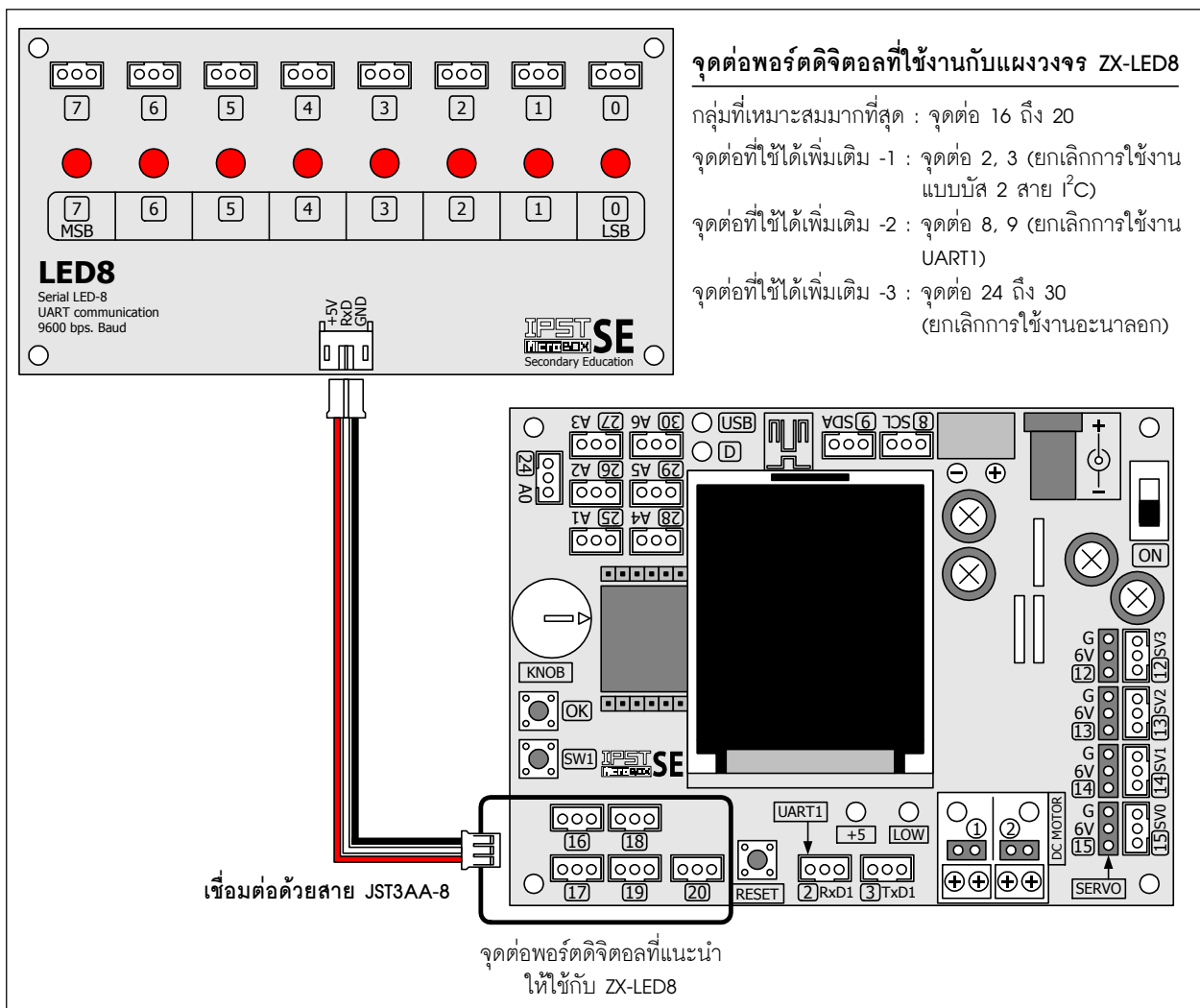
5.7 ipst_led8.h ไฟล์ไลบรารีขับ LED 8 ดวง

เป็นไฟล์ไลบรารีบรรจุชุดคำสั่งที่เกี่ยวกับการติดต่อกับแผงวงจร ZX-LED8 เพื่อขับ LED 8 ดวง ให้แสดงผลตามต้องการ ก่อนเรียกใช้งานต้องผนวกไฟล์ไลบรารีไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_led8.h> หรือ #includ <ipst.h>
```

5.7.1 การเชื่อมต่อทางฮาร์ดแวร์

ไฟล์ไลบรารีนี้ใช้งานกับจุดต่อพอร์ตดิจิทัลทั้งหมดของแผงวงจร IPST-SE โดยใช้เพียง 1 จุดต่อพอร์ตในการติดต่อกับแผงวงจร ZX-LED8 ดังในรูปที่ 5-1



รูปที่ 5-1 การเชื่อมต่อเพื่อใช้งานแผงวงจร ZX-LED8 ของแผงวงจร IPST-SE เพื่อขับ LED 8 ดวง

5.7.2 pinLED8

เป็นฟังก์ชันกำหนดจุดต่อพอร์ตของแผงวงจร IPST-SE ที่ต้องการเชื่อมต่อกับแผงวงจร ZX-LED8

รูปแบบ

```
void pinLED8(int pin)
```

พารามิเตอร์

pin - ขาพอร์ตใดๆ ของแผงวงจร IPST-SE มีค่า 0 ถึง 30 (แนะนำให้ใช้จุดต่อพอร์ต 16 ถึง 20)

ตัวอย่างที่ 5-13

```
pinLED8(20);           // เลือกจุดต่อพอร์ต 20 ของแผงวงจร IPST-SE ในการต่อกับแผงวงจร ZX-LED8
```

5.7.3 LED8

เป็นฟังก์ชันกำหนดข้อมูลสำหรับแสดงผลบนแผงวงจร ZX-LED8

รูปแบบ

```
void LED8(unsigned char _dat) (ใช้ในกรณีที่มีการกำหนดขาเชื่อมต่อด้วย pinLED8)
```

```
void LED8(int pin, unsigned char _dat)
```

พารามิเตอร์

pin - ขาพอร์ตใดๆ ของแผงวงจร IPST-SE มีค่า 0 ถึง 30 (แนะนำให้ใช้จุดต่อพอร์ต 16 ถึง 20)

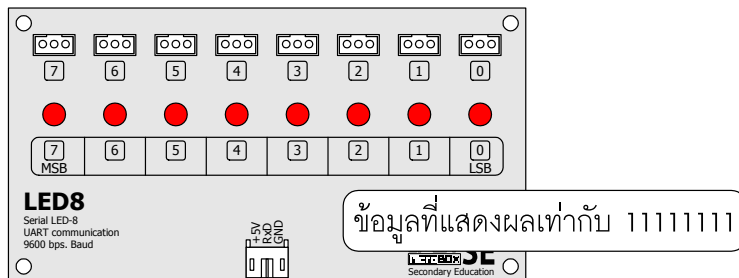
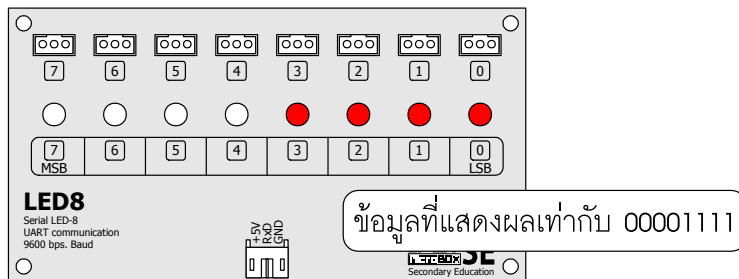
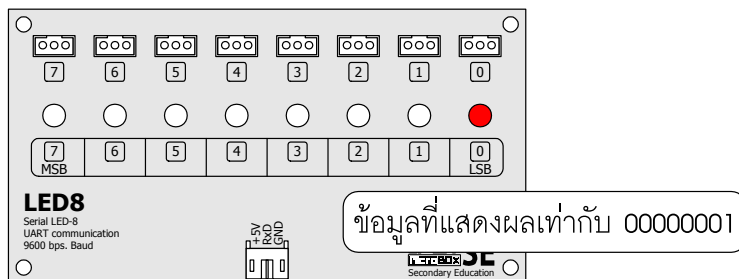
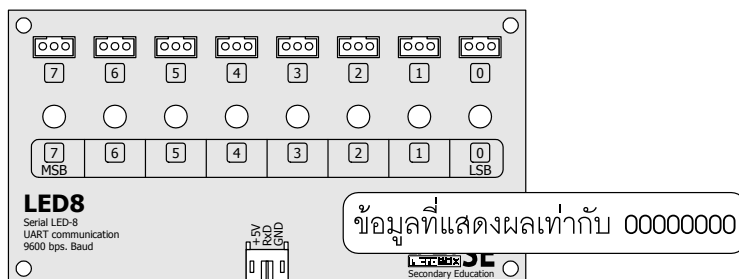
_dat - ข้อมูลสำหรับนำไปแสดงผล มีค่า 0 ถึง 255

ตัวอย่างที่ 5-14

```
#include <ipst.h>           // มนากไฟล์ไลบรารีหลัก
void setup()
{
  pinLED8(20);             // ใช้จุดต่อพอร์ต 20 ในการต่อกับแผงวงจร ZX-LED8
}
void loop()
{
  unsigned char i=0;
  while(1)
  {
    LED8(i++);             // แสดงค่าเลขฐานสอง 8 บิต (00000000 ถึง 11111111)
    delay(500);           // หน่วงเวลา 0.5 วินาที
  }
}
```

ตัวอย่างที่ 5-15

```
#include <ipst.h>           // มนวกไฟล์ไลบรารีหลัก
void setup()
{
void loop()
{
  unsigned char i=0;
  while(1)
  {
    LED8(20,i++);          // เลือกจุดต่อพอร์ต 20 ในการเชื่อมต่อกับแผงวงจร ZX-LED8
                           // แล้วแสดงค่าเลขฐานสอง 8 บิต (00000000 ถึง 11111111)
    delay(500);           // หน่วงเวลา 0.5 วินาที
  }
}
```



5.8 ipst_motor.h ไลบรารีขับเคลื่อนมอเตอร์ไฟตรง

แผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE) ขับมอเตอร์ไฟตรงได้ 2 ตัว โดยต่อมอเตอร์ไฟตรงขนาด 3 ถึง 9V เข้าที่จุดต่อมอเตอร์ช่อง 1 และ 2 ฟังก์ชันที่ใช้ในการขับมอเตอร์ไฟตรงมีดังนี้

5.8.1 motor

เป็นฟังก์ชันขับเคลื่อนมอเตอร์ไฟตรง

รูปแบบ

```
void motor(char _channel,int _power)
```

พารามิเตอร์

_channel - กำหนดช่องเอาต์พุตมอเตอร์ไฟตรง มีค่า 1 และ 2

_power - กำหนดกำลังขับเคลื่อนมอเตอร์ มีค่าในช่วง -100 ถึง 100

ถ้ากำหนดค่า _power เป็นบวก (1 ถึง 100) ทำให้มอเตอร์หมุนไปในทิศทางหนึ่ง

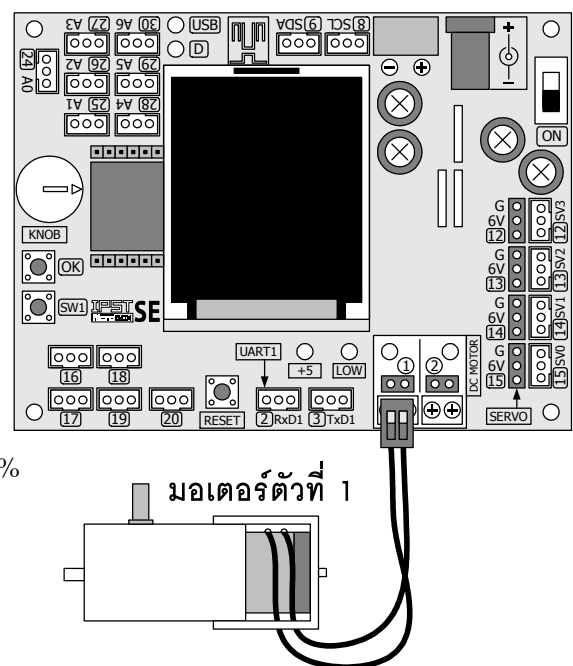
ถ้ากำหนดค่า _power เป็นลบ (-1 ถึง -100) มอเตอร์จะถูกขับให้หมุนไปในทิศทางตรงข้าม

ถ้ากำหนดค่า _power เป็น 0 มอเตอร์หยุดหมุน ไม่แนะนำให้กำหนดค่าเป็น 0 หากต้อง

การให้มอเตอร์หยุดหมุนควรเรียกใช้ฟังก์ชัน motor_stop

ตัวอย่างที่ 5-16

```
#include <ipst.h> // ผนวกไลบรารีหลัก
void setup()
{
}
void loop()
{
  motor(1,60); // ขับมอเตอร์ช่องที่ 1 ด้วย
                // กำลัง 60%
  delay(500); // ขับนาน 0.5 วินาที
  motor(1,-60); // ขับมอเตอร์ช่องที่ 1
                // กลับทิศทางด้วยกำลัง 60%
  delay(500); // ขับนาน 0.5 วินาที
}
```



5.8.2 motor_stop

เป็นฟังก์ชันหยุดขับเคลื่อนมอเตอร์

รูปแบบ

```
void motor_stop(char _channel)
```

พารามิเตอร์

_channel - กำหนดช่องเอาต์พุตมอเตอร์ไฟตรง มีค่า 1, 2 และ ALL
โดย ALLเป็นการเลือกให้มอเตอร์ทั้งหมดหยุดทำงาน

ตัวอย่างที่ 5-17

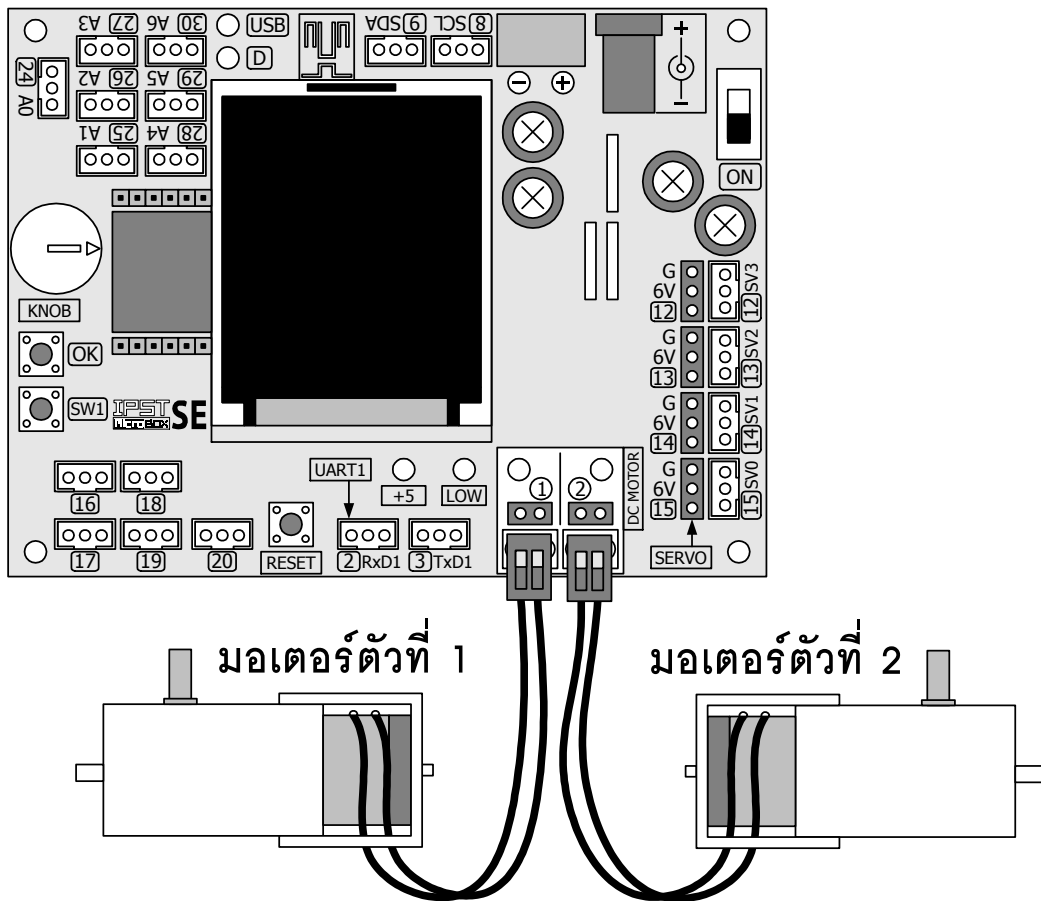
```
#include <ipst.h>           // มนวกไลบรารีหลัก
void setup()
{
    sw_OK_press();         // ตรวจสอบการกดสวิตช์ OK
}
void loop()
{
    motor(1,60);           // มอเตอร์ 1 หมุนด้วยกำลังไฟฟ้า 60%
    delay(500);            // หน่วงเวลา 0.5 วินาที
    motor(1,-60);          // มอเตอร์ 1 หมุนกลับทิศด้วยกำลังไฟฟ้า 60%
    delay(500);            // หน่วงเวลา 0.5 วินาที
    if (sw1())              // ตรวจสอบการกดสวิตช์ SW1
    {
        motor_stop(1);     // ถ้าสวิตช์ SW1 ถูกกด มอเตอร์ช่อง 1 หยุดหมุน
        while (1);
    }
}
```

ตัวอย่างที่ 5-18

```

#include <ipst.h>           // มนวกไฟล์ไลบรารีหลัก
void setup()
{
}
void loop()
{
  motor(1,50);             // ขับมอเตอร์ช่อง 1 ด้วยกำลัง 50% ของกำลังสูงสุด
  motor(2,50);             // ขับมอเตอร์ช่อง 2 ด้วยกำลัง 50% ของกำลังสูงสุด
  sleep(3000);             // หน่วงเวลา 3 วินาที
  motor(1,-50);            // ขับมอเตอร์ช่อง 1 กลับทิศทางด้วยกำลัง 50% ของกำลังสูงสุด
  motor(2,-50);            // ขับมอเตอร์ช่อง 2 กลับทิศทางด้วยกำลัง 50% ของกำลังสูงสุด
  sleep(3000);             // หน่วงเวลา 3 วินาที
  motor_stop(ALL);         // หยุดขับมอเตอร์ทั้งสองช่อง
  sleep(3000);             // หน่วงเวลา 3 วินาที
}

```



5.9 ipst_servoMotor.h ไฟล์ไลบรารีขับเคลื่อนเซอร์โวมอเตอร์

เป็นไฟล์ไลบรารีควบคุมเซอร์โวมอเตอร์ มีฟังก์ชันสนับสนุนการควบคุมตำแหน่งแกนหมุนของเซอร์โวมอเตอร์ โดยแผงวงจร IPST-SE ควบคุมเซอร์โวมอเตอร์ได้ 4 ตัวในเวลาเดียวกัน ก่อนใช้งานต้องผนวกไลบรารีไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_servoMotor.h> หรือ #include <atx.h>
```

ในไฟล์ไลบรารีนี้มี 1 ฟังก์ชันคือ **servo** เป็นฟังก์ชันกำหนดตำแหน่งแกนหมุนของเซอร์โวมอเตอร์

รูปแบบ

```
void servo(unsigned char _ch, int _angle)
```

พารามิเตอร์

_ch - ช่องเอาต์พุตเซอร์โวมอเตอร์ มีค่า 0 ถึง 3

_pos - กำหนดตำแหน่งแกนหมุนของเซอร์โวมอเตอร์ มีค่าในช่วง 0 ถึง 180 และ -1

ถ้ากำหนดเป็น -1 หมายถึง ไม่ใช้งานเซอร์โวมอเตอร์ที่ช่องนั้นๆ

ตัวอย่างที่ 5-19

```
#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
void loop()
{
servo(0,60); // ขับเซอร์โวมอเตอร์ช่อง 0 ไปยังตำแหน่ง 60 องศา
delay(5000); // หน่วงเวลา 5 วินาที
servo(0,120); // ขับเซอร์โวมอเตอร์ช่อง 0 ไปยังตำแหน่ง 120 องศา
delay(5000); // หน่วงเวลา 5 วินาที
}
```

5.10 ipst_serial.h ไลบรารีรับส่งข้อมูลอนุกรม

เป็นไฟล์ไลบรารีที่บรรจุชุดคำสั่งเกี่ยวกับการรับส่งข้อมูลอนุกรมผ่านวงจรสื่อสารข้อมูลอนุกรม (UART) ของไมโครคอนโทรลเลอร์บนแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE) ก่อนเรียกใช้งานต้องผนวกไฟล์ไลบรารีไว้ในตอนต้นของโปรแกรมด้วยคำสั่ง

```
#include <ipst_serial.h> หรือ #include <atx.h>
```

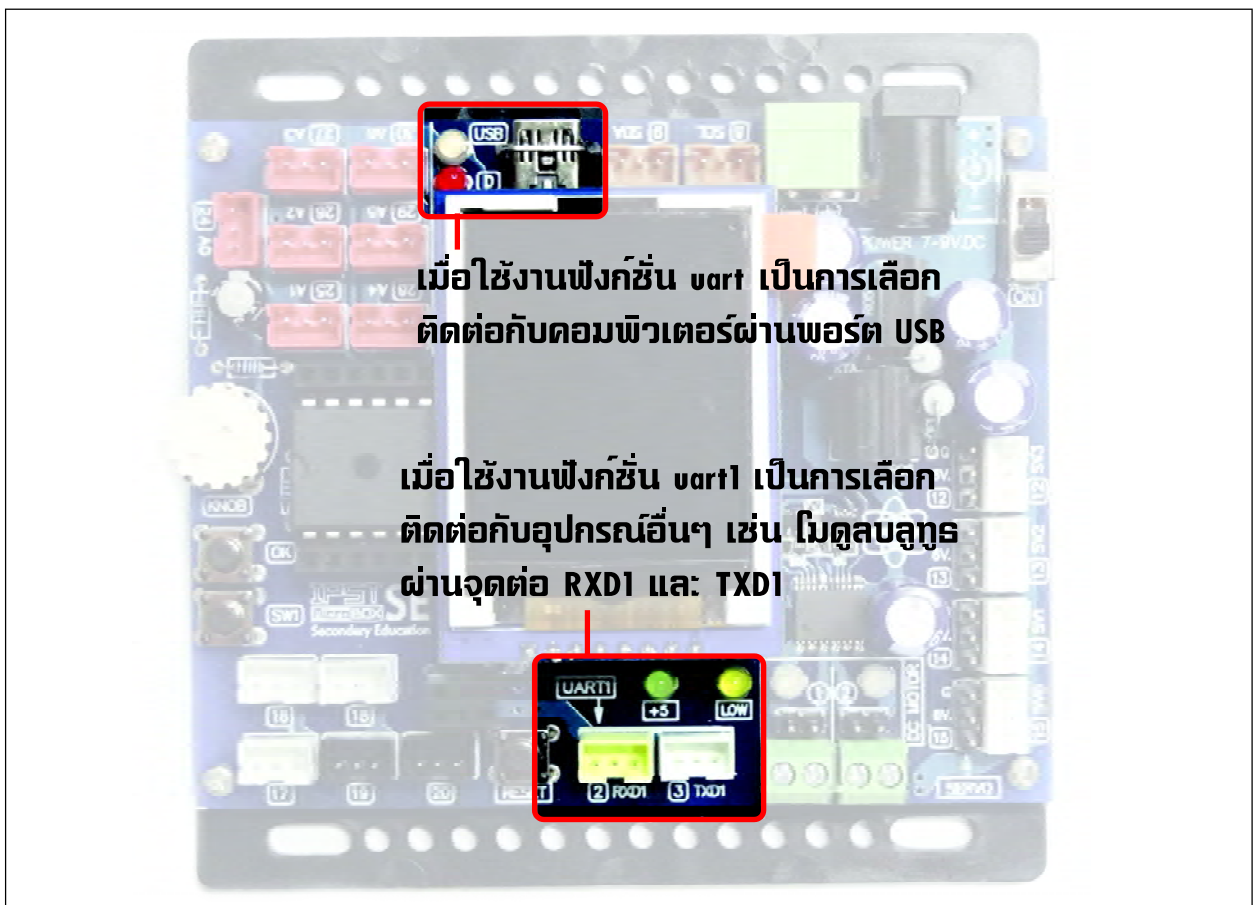
5.10.1 การเชื่อมต่อทางฮาร์ดแวร์

เมื่อต้องการใช้งานช่อง UART0 (ทำงานผ่านพอร์ต USB)

ให้ต่อสายจากจุดต่อพอร์ต USB บนแผงวงจร IPST-SE (เป็นจุดต่อเดียวกับที่ใช้ในการอัปโหลด) เข้ากับพอร์ต USB ของคอมพิวเตอร์

เมื่อต้องการใช้งานช่อง UART1

ต่อสายสัญญาณเข้ากับจุดต่อ RXD1 (จุดต่อพอร์ต 2) และ TXD1 (จุดต่อพอร์ต 3) บนแผงวงจร IPST-SE



รูปที่ 5-2 แสดงจุดต่อสัญญาณของแผงวงจร IPST-SE เมื่อมีการใช้งานไลบรารี ipst_serial.h

5.10.2 uart

เป็นฟังก์ชันสำหรับส่งข้อมูลกลุ่มอักษรหรือสายอักขระหรือสตริง (string) ออกจากโมดูล UART0 ไปยังคอมพิวเตอร์ผ่านพอร์ต USB มีอัตราการถ่ายทอข้อมูลเริ่มต้นที่ 115,200 บิตต่อวินาที

รูปแบบ

```
void uart(char *p,...)
```

พารามิเตอร์


p - รหัสสับของลุ่มข้อความที่ต้องการส่งออกจากภาคส่งของโมดูล UART0 กำหนดรูปแบบการแทรกสัญลักษณ์พิเศษเพื่อใช้ร่วมในการแสดงผลได้ดังนี้

รหัสบังคับ	การทำงาน
%c หรือ %C	แสดงผลตัวอักษร 1 ตัว
%d หรือ %D	แสดงผลตัวเลขฐานสิบช่วงตั้งแต่ -32,768 ถึง +32,767
%l หรือ %L	แสดงผลตัวเลขฐานสิบช่วงตั้งแต่ -2,147,483,648 ถึง +2,147,483,647
%f หรือ %F	แสดงผลข้อมูลแบบจำนวนจริง(แสดงทศนิยม 3 หลัก)
\r	กำหนดให้ข้อความชิดไปทางด้านซ้ายของบรรทัด
\n	กำหนดให้ข้อความขึ้นบรรทัดใหม่

ตัวอย่างที่ 5-20

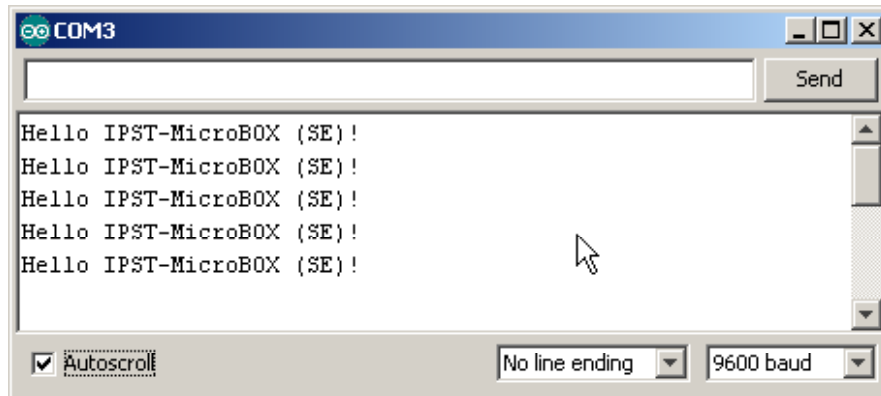
```
#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
}
void loop()
{
  uart("Hello IPST-MicroBOX (SE)!\r\n"); // ส่งข้อมูลไปยังคอมพิวเตอร์แบบขึ้นบรรทัดใหม่
  delay(2000); // หน่วงเวลา 2 วินาที
}

```

ในการรันโปรแกรมนี้ต้องต่อสาย USB ระหว่างแผงวงจร IPST-SE กับพอร์ต USB ของคอมพิวเตอร์ไว้ตลอดเวลา จากนั้นเปิดหน้าต่าง Serial Monitor โดยคลิกที่ปุ่ม  หรือเลือกเมนู Tools > Serial Monitor



จากนั้นรอสักครู่ หน้าต่าง Serial Monitor จะปรากฏขึ้นมา พร้อมกับแสดงข้อความตามรูป ให้ทำการคลิกทำเครื่องหมายที่ช่อง Auto Scroll และเลือก No line ending ส่วนช่องสุดท้ายเลือกเป็น 9600 Baud



5.10.3 uart_set_baud

เป็นฟังก์ชันกำหนดอัตราหรือความเร็วในการสื่อสารข้อมูลของโมดูล UART0 กับคอมพิวเตอร์

รูปแบบ

```
void uart_set_baud(unsigned int baud)
```

พารามิเตอร์

baud - อัตราบอดในการสื่อสารของโมดูล UART0 กับคอมพิวเตอร์ มีค่า 2400 ถึง 115,200

ตัวอย่างที่ 5-21

```
uart_set_baud(4800); // กำหนดอัตราบอดในการสื่อสารข้อมูลเป็น 4,800 บิตต่อวินาที
```

5.10.4 uart_available

เป็นฟังก์ชันตรวจสอบการรับข้อมูลเข้ามาของโมดูล UART0 เมื่อติดต่อกับคอมพิวเตอร์

รูปแบบ

```
unsigned char uart_available(void)
```

การคืนค่า

- เป็น "0" เมื่อยังไม่มีข้อมูลเข้ามา
- มากกว่า 0 เมื่อมีข้อมูลเข้ามา โดยมีค่าเท่ากับจำนวนของอักขระที่ได้รับ

ตัวอย่างที่ 5-22

```
char x =uart_available();
// ตรวจสอบว่า มีข้อมูลเข้ามาทางภาครับของโมดูล UART0 หรือไม่ ถ้า x มีค่ามากกว่า 0 แสดงว่า
// มีข้อมูลเข้ามายังภาครับแล้ว ควรอ่านข้อมูลออกด้วยฟังก์ชัน uart_getkey ในลำดับถัดไปทันที
```

5.10.5 uart_getkey

เป็นฟังก์ชันอ่านข้อมูลจากบัฟเฟอร์ตัวรับของโมดูล UART0

รูปแบบ

```
char uart_getkey(void)
```


การคืนค่า

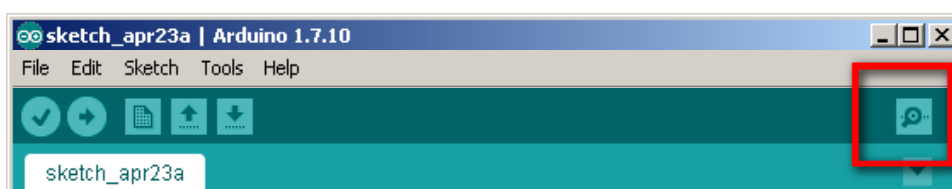
- เป็น “0” เมื่อไม่มีการรับอักขระใดๆ เข้ามายังวงจรรับของโมดูล UART
- เป็นค่าของอักขระที่รับได้ในรูปแบบของรหัสแอสกี

ตัวอย่างที่ 5-23

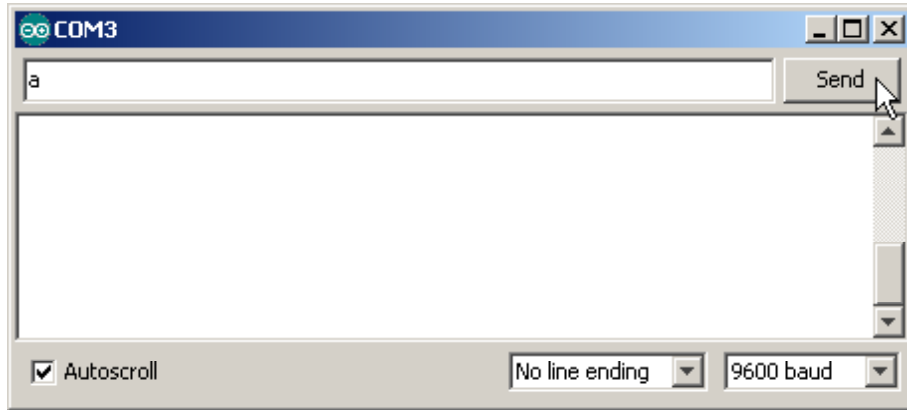
```
#include <ipst.h>
void setup()
{
void loop()
{
// ลูปการทำงานหลัก
if(uart_available())
// ตรวจสอบว่ามีข้อมูลเข้ามาหรือไม่
{
if(uart_getkey()=='a')
// ตรวจสอบการกดคีย์ a ว่า ถูกกดหรือไม่
{
glcd(3,1,“Key a Active!”); // แสดงข้อความเพื่อตอบสนองต่อการตรวจพบว่ามีคีย์ a
sleep(2000); // หน่วงเวลาแสดงข้อความประมาณ 2 วินาที
}
else
{
glcdClear(); // เคลียร์ข้อความที่หน้าจอแสดงผล
}
}
}
}
```

หมายเหตุ เมื่อเรียกใช้ฟังก์ชัน `uart` เพื่อส่งข้อมูลออกทางโมดูลพอร์ตอนุกรมหรือ UART และ `uart_getkey` เพื่อตรวจรับอักขระใดๆ นั้น อัตราการสื่อสารข้อมูลจะถูกกำหนดเป็น 115,200 บิตต่อวินาที ข้อมูล 8 บิต และไม่มีการตรวจสอบพาริตีโดยอัตโนมัติ และเป็นค่าตั้งต้น เพื่อลดความซับซ้อนในการเขียนโปรแกรมลง หากต้องการเปลี่ยนอัตราเร็วในการสื่อสารข้อมูลต้องใช้คำสั่ง `uart_set_baud` อย่างไรก็ตาม ต้องคำนึงด้วยว่า เมื่ออัตราการสื่อสารสูงขึ้นอาจส่งผลกระทบต่อความถูกต้องในการสื่อสารข้อมูลได้

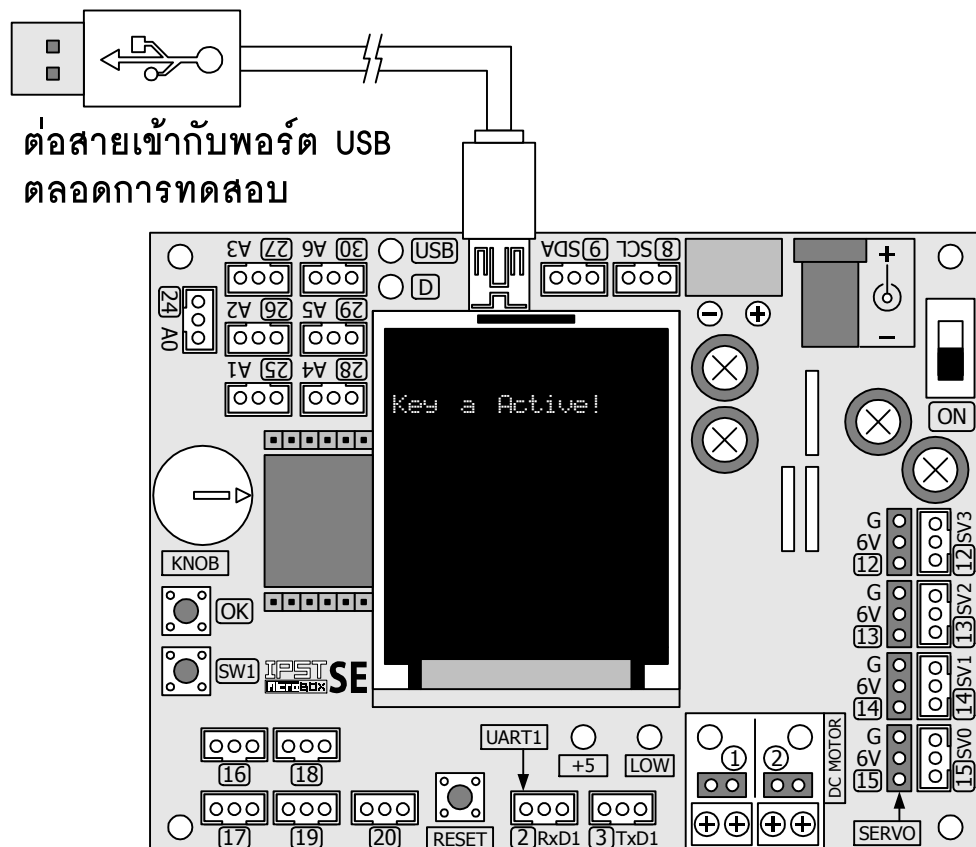
ในการรันโปรแกรมนี้อาจต้องต่อสาย USB ระหว่างแผงวงจร IPST-SE กับพอร์ต USB ของคอมพิวเตอร์ไว้ตลอดเวลา จากนั้นเปิดหน้าต่าง Serial Monitor โดยคลิกที่ปุ่ม  หรือเลือกเมนู Tools > Serial Monitor



จากนั้นรอสักครู่ หน้าต่าง Serial Monitor จะปรากฏขึ้นมา ให้กดปุ่มตัวอักษร a ที่ช่องบนสุด แล้วคลิก กดปุ่ม Send ทางขวามือ โดยที่ช่องด้านล่างเลือก No line ending ส่วนช่องสุดท้ายด้านล่างขวามือเลือกเป็น 9600 Baud



เมื่อส่งตัวอักษร a จากคอมพิวเตอร์ไปยังแผงวงจร IPST-SE แล้ว ที่แผงวงจร IPST-SE จะตรวจสอบการเข้ามาของข้อมูล หากถูกต้อง จะแสดงข้อความ Key a Active !



5.10.6 uart1

เป็นฟังก์ชันส่งข้อมูลสายอักขระออกทางภาคส่งของโมดูล UART1 มีอัตราการรับส่งข้อมูลเริ่มต้นที่ 9,600 บิตต่อวินาที โดยการใช้งานต้องต่อสายสัญญาณเข้าที่จุด TXD1

รูปแบบ

```
void uart1(char *p,...)
```

พารามิเตอร์

p - รหัสของกลุ่มข้อความที่ต้องการส่งออกจากภาคส่งของโมดูล UART1 โดยสามารถกำหนดรูปแบบการแทรกสัญลักษณ์พิเศษเพื่อใช้ร่วมในการแสดงผลเหมือนกับฟังก์ชัน uart1

5.10.7 uart1_set_baud

เป็นฟังก์ชันกำหนดอัตราการรับส่งข้อมูลของ โมดูล UART1 กับอุปกรณ์ภายนอก

รูปแบบ

```
void uart1_set_baud(unsigned int baud)
```

พารามิเตอร์

baud - กำหนดค่าอัตราการรับส่งข้อมูลในการสื่อสารข้อมูลของโมดูล UART1

ตัวอย่างที่ 5-24

```
uart1_set_baud(19200); // กำหนดอัตราบอดในการสื่อสารเป็น 19,200 บิตต่อวินาที
```

5.10.8 uart1_available

เป็นฟังก์ชันตรวจสอบการรับข้อมูลเข้ามาของโมดูล UART1 เมื่อติดต่อกับอุปกรณ์ภายนอก โดยการใช้งานต้องต่อสายสัญญาณเข้าที่จุด RXD1 และ TXD1

รูปแบบ

```
unsigned char uart1_available(void)
```

การคืนค่า

- เป็น 0 เมื่อไม่มีข้อมูลเข้ามา
- มากกว่า 0 โดยมีค่าเท่ากับจำนวนของอักขระที่ได้รับ

ตัวอย่างที่ 5-25

```
char x =uart1_available();
// ตรวจสอบว่ามีข้อมูลเข้ามาทางภาครับของโมดูล UART1 หรือไม่
// ถ้า x มีค่ามากกว่า 0 แสดงว่ามีข้อมูลเข้ามาแล้ว ควรอ่านออกไปด้วยฟังก์ชัน uart1_getkey ทันที
```

5.10.9 uart1_getkey

เป็นฟังก์ชันอ่านข้อมูลจากบัฟเฟอร์ตัวรับของโมดูล UART1

รูปแบบ

char uart1_getkey(void)

การคืนค่า

- เป็น 0 เมื่อยังไม่มีการรับอักขระใดๆ
- เป็นค่าของอักขระที่รับได้ในรูปแบบของรหัสแอสกี

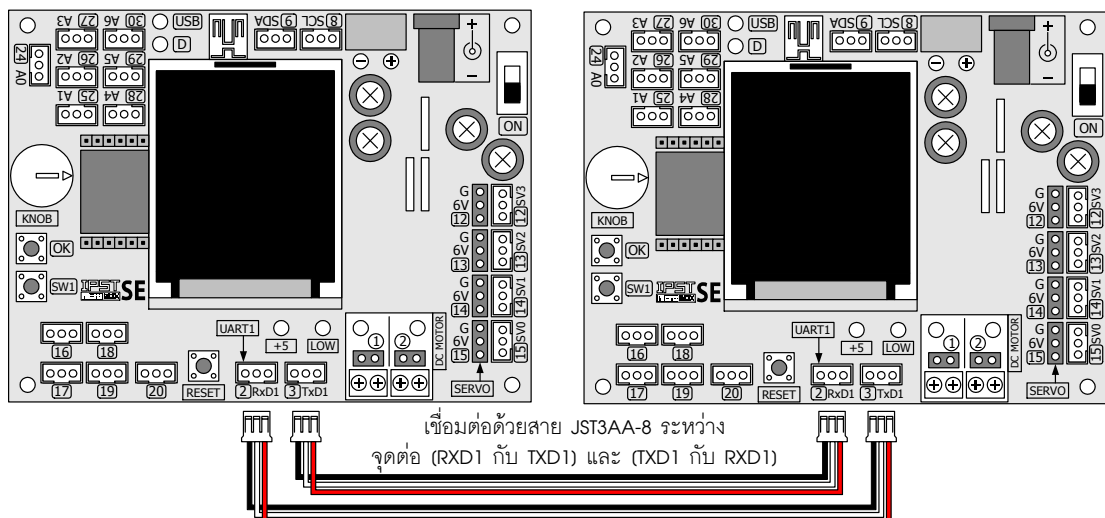
จะใช้งาน UART1 เมื่อใด

1. เมื่อต้องการติดต่อกับอุปกรณ์สื่อสารข้อมูลไร้สาย อาทิ โมดูลบลูทูธ, โมดูล XBEE เป็นต้น



แนวทางเชื่อมต่อแผงวงจร IPST-SE ติดต่อกับสมาร์ทโฟน/แท็บเล็ต (แอนดรอยด์) หรือคอมพิวเตอร์แบบไร้สายผ่านระบบบลูทูธ

2. เมื่อต้องการสื่อสารข้อมูลระหว่างแผงวงจร IPST-SE 2 ชุด



3. เมื่อต้องการติดต่อกับอุปกรณ์ภายนอกที่ต้องการรูปแบบข้อมูลอนุกรมในการสื่อสาร อาทิ แผงวงจรขับเคลื่อนมอเตอร์ 16 ช่อง (ZX-SERVO16i) , แผงวงจรโมดูล LCD แบบอนุกรม (SLCD16x2) , แผงวงจรฐานเวลานาฬิกาจริงแบบอนุกรม (ZX-17)

บทที่ 6

การแสดงผลด้วยจอกราฟิก LCD สี ของชุดกล่องสมองกล

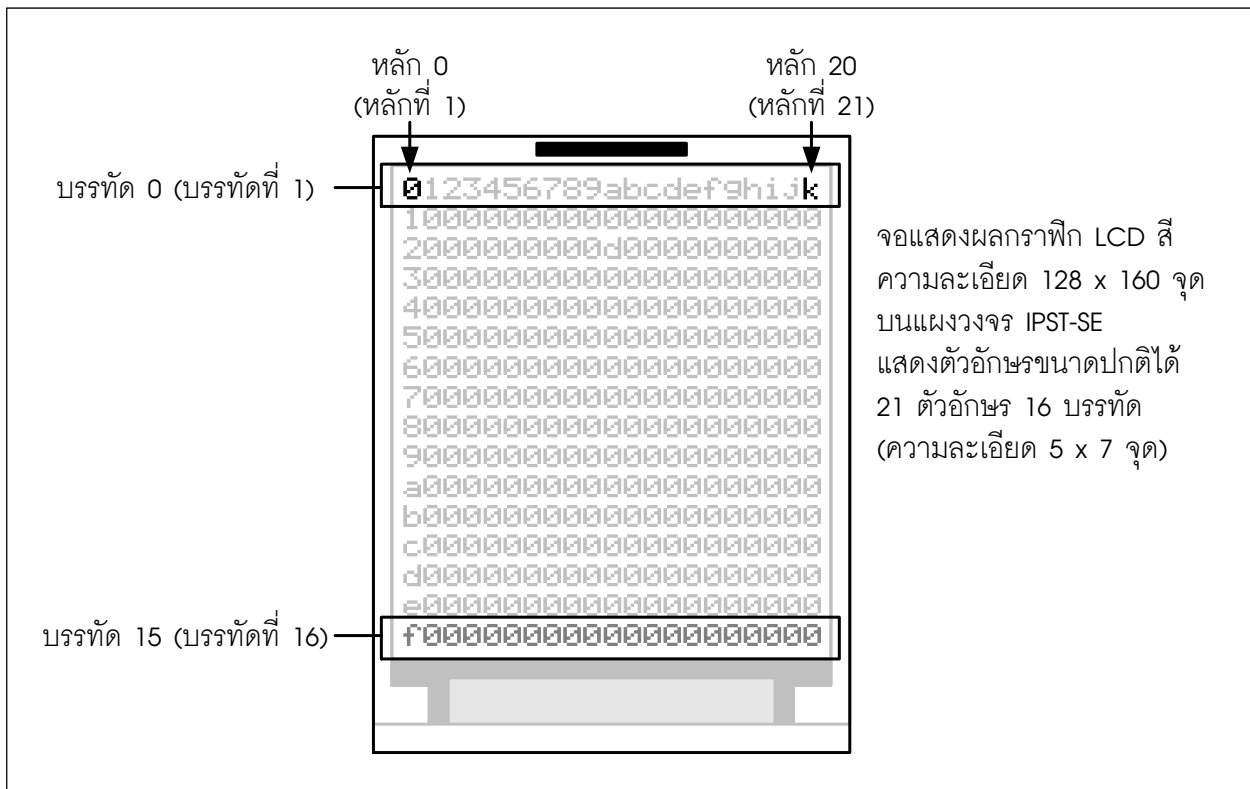
ในการพัฒนาโครงงานวิทยาศาสตร์สมัยใหม่ ที่เกี่ยวข้องกับการเขียนโปรแกรมและต้องมีระบบควบคุมอัตโนมัติเข้ามาเกี่ยวข้อง รวมถึงต้องมีการติดต่อกับตัวตรวจจับ ซึ่งผู้ใช้งานต้องการทราบถึงค่าที่ตัวตรวจจับวัดหรือตรวจจับได้ อุปกรณ์ตัวหนึ่งที่มีความสำคัญคือ **อุปกรณ์แสดงผล** อุปกรณ์พื้นฐานที่สุดที่ใช้ในการแสดงผลคือ LED หากต้องการให้มีการแสดงผลเป็นตัวเลข ก็จะสามารถเปลี่ยนมาใช้ LED ตัวเลข 7 ส่วน หากมีความต้องการเพิ่มขึ้นไปอีก นั่นคือ ต้องการแสดงเป็นข้อความเป็นตัวอักษรจำนวนมากขึ้น โมดูล LCD แบบอักขระ (character LCD module) ก็จะเข้ามาทำหน้าที่นี้

ในชุดกล่องสมองกล IPST-MicroBOX Secondary Education (SE) รองรับทั้งการแสดงผลด้วยอุปกรณ์พื้นฐานอย่าง LED และจอแสดงผลแบบกราฟิก LCD สี ทำให้ชุดกล่องสมองกล IPST-MicroBOX (SE) นี้ความสมบูรณ์พร้อมในการแสดงผลการทำงานอย่างเพียงพอ ในบทนี้เป็นการนำเสนอข้อมูลสำหรับการใช้งานจอแสดงผลแบบกราฟิก LCD สีที่เป็นอุปกรณ์แสดงผลหลักของชุดกล่องสมองกล IPST-MicroBOX (SE) เพื่อให้นำไปใช้งานได้อย่างเกิดประโยชน์สูงสุด

6.1 คุณสมบัติของจอแสดงผลแบบกราฟิก LCD สีของแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE)

คุณสมบัติที่สำคัญโดยสรุปของจอแสดงผลแบบกราฟิก LCD สีของแผงวงจร IPST-SE มีดังนี้

- ขนาดจอแสดงผล 1.8 นิ้ว ความละเอียด 128 x 160 จุด
- แสดงภาพกราฟิกลายเส้นและพื้นสี 65,536 สี (ไม่รองรับไฟล์รูปภาพ) พร้อมไฟส่องหลัง
- แสดงผลเป็นตัวอักษรขนาดปกติ (5 x 7 จุด) ได้ 21 ตัวอักษร 16 บรรทัด (หรือ 21 x 16)
- มีไฟล์ไลบรารี ipst_glcd.h รองรับเพื่อช่วยให้การเขียนโปรแกรมเพื่อกำหนดรูปแบบการแสดงผลทำได้หลากหลายแบบและสะดวกขึ้นอย่างมาก



รูปที่ 6-1 แสดงการกำหนดตำแหน่งของการแสดงผลสำหรับจอแสดงผลกราฟิก LCD สีที่ใช้บนแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE)

6.2 การเรียกใช้ไฟล์ไลบรารีสำหรับจอแสดงผลแบบกราฟิก LCD สี

ไฟล์ไลบรารีที่ใช้ในการติดต่อและควบคุมการทำงานของจอแสดงผลกราฟิก LCD สีจะได้รับการติดตั้งไปพร้อมกับการติดตั้งซอฟต์แวร์ Arduino IDE 1.7.10 แล้ว ไฟล์ไลบรารีชื่อ ipst_glcd.h ได้รับการคัดลอกลงในโฟลเดอร์ C:\Arduino17\libraries\IPST

การประกาศเพื่อผนวกไฟล์ไลบรารีนี้ในโปรแกรมหรือสเก็ทซ์ของ Arduino 1.7.10 ทำได้ดังนี้

```
#include <ipst_glcd.h> หรือ #include <ipst.h>
```

6.3 ฟังก์ชันเกี่ยวกับการแสดงผลจอภาพแบบกราฟิก LCD สี

6.3.1 glcd

เป็นฟังก์ชันแสดงข้อความที่หน้าจอแสดงผลกราฟิก LCD สี โดยแสดงตัวอักษรขนาดปกติได้ 21 ตัวอักษร 16 บรรทัด

รูปแบบ

```
void glcd(unsigned char x, unsigned char y ,char *p,...)
```

พารามิเตอร์

x คือตำแหน่งบรรทัดมีค่าตั้งแต่ 0 ถึง 15

y คือตำแหน่งตัวอักษรมีค่าตั้งแต่ 0 ถึง 20

*p คือข้อความที่ต้องการนำมาแสดงรวมถึงรหัสที่ใช้กำหนดรูปแบบพิเศษเพื่อร่วมแสดงผลข้อมูลตัวเลขในรูปแบบอื่นๆ ประกอบด้วย

%c หรือ %C - รับค่าแสดงผลตัวอักษร 1 ตัวอักษร

%d หรือ %D - รับค่าแสดงผลตัวเลขจำนวนเต็มในช่วง -32,768 ถึง 32,767

%l หรือ %L - รับค่าแสดงผลตัวเลขจำนวนเต็มในช่วง -2,147,483,648 ถึง 2,147,483,647

%f หรือ %F - รับค่าเพื่อแสดงผลตัวเลขจำนวนจริง (แสดงทศนิยม 3 หลัก)

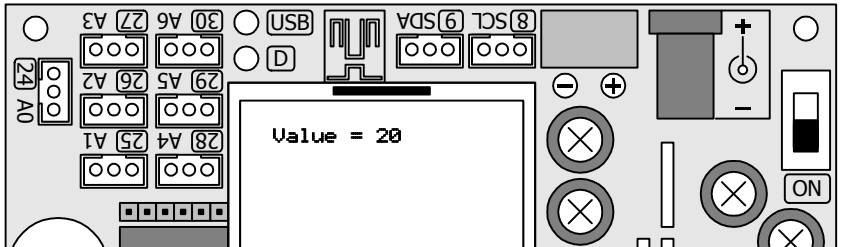
ตัวอย่างที่ 6-1

```
glcd(2,0,"Hello World"); // แสดงข้อความ Hello World ที่ตำแหน่งซ้ายสุดของบรรทัด 2 (บรรทัดที่ 3)
```



ตัวอย่างที่ 6-2

```
int x=20;
lcd(1,2,"Value = %d",x);
// แสดงตัวอักษรและตัวเลขบนบรรทัดเดียวกัน เริ่มต้นที่คอลัมน์ 2 ของบรรทัด 1 (บรรทัดที่ 2)
```



6.3.2 colorRGB

เป็นฟังก์ชันเปลี่ยนค่าสีในรูปแบบ RGB (แดง เขียว น้ำเงิน) ให้อยู่ในรูปแบบของตัวเลข 16 บิต โดยแบ่งเป็นค่าของสีแดง 5 บิต ต่อด้วยสีเขียว 6 บิต และปิดท้ายด้วยค่าของสีน้ำเงิน 5 บิต

รูปแบบ

```
unsigned int colorRGB(uint red,uint green,uint blue)
```

พารามิเตอร์

- red - เป็นค่าของสีแดง มีค่า 0 ถึง 31 ถ้าค่าที่ป้อนมากกว่า 31 จะปรับลดให้เท่ากับ 31
- green - ค่าของสีเขียว มีค่า 0 ถึง 63 ถ้าค่าที่ป้อนมากกว่า 63 จะถูกปรับลดให้เท่ากับ 63
- blue - ค่าของสีน้ำเงิน มีค่า 0 ถึง 31 ถ้าค่าที่ป้อนมากกว่า 31 จะปรับลดให้เท่ากับ 31

ตัวอย่างที่ 6-3

```
#include <ipst.h>
int colors;
void setup()
{
    int colors;
    colors=colorRGB(31,0,0); // ส่งค่าสี 16 บิตของสีแดงให้ตัวแปร colors
    lcdFillScreen(colors); // นำค่าไปแสดงเป็นสีพื้นของจอแสดงผล
}
void loop()
{}
```


6.3.3 color[]

เป็นตัวแปรอะเรย์ที่ใช้กำหนดสีจำนวน 8 สีที่เป็นสีพื้นฐาน ผู้พัฒนาโปรแกรมสามารถเรียกใช้ตัวแปร color[] หรือเรียกใช้ชื่อสีตรงๆ ก็ได้

รูปแบบ

```
unsigned int color[] = { GLCD_RED,
  GLCD_GREEN,
  GLCD_BLUE,
  GLCD_YELLOW,
  GLCD_BLACK,
  GLCD_WHITE,
  GLCD_SKY,
  GLCD_MAGENTA};
```

พารามิเตอร์

GLCD_RED - ใช้กำหนดสีแดง
 GLCD_GREEN - ใช้กำหนดสีเขียว
 GLCD_BLUE - ใช้กำหนดสีน้ำเงิน
 GLCD_YELLOW - ใช้กำหนดสีเหลือง
 GLCD_BLACK - ใช้กำหนดสีดำ
 GLCD_WHITE - ใช้กำหนดสีขาว
 GLCD_SKY - ใช้กำหนดสีฟ้า
 GLCD_MAGENTA - ใช้กำหนดสีบานเย็น

ตัวอย่างที่ 6-4

```
glcdFillScreen(color[5]); // กำหนดให้พื้นหลังเป็นสีขาว
```

ตัวอย่างที่ 6-5

```
glcdFillScreen(GLCD_BLUE); // กำหนดให้พื้นหลังเป็นสีน้ำเงิน
```

6.3.4 glcdSetColorWordBGR

เป็นฟังก์ชันกำหนดการเรียงบิตข้อมูลสีให้เป็นแบบ BGR (5-6-5) นั่นคือ ค่าของสีน้ำเงิน 5 บิต ต่อด้วยสีเขียว 6 บิต และปิดท้ายด้วยค่าของสีแดง 5 บิต ทั้งนี้เนื่องจากผู้ผลิตจอแสดงผลกราฟิก LCD สีมีการผลิตจอแสดงผลที่มีคุณสมบัติเหมือนกันออกมา 2 รุ่น แต่มีการเรียงบิตข้อมูลสีต่างกัน คือ เรียงแบบ BGR และเรียงแบบ RGB

อย่างไรก็ตาม ค่าตั้งต้นของไฟล์ไลบรารี ipst_glcd.h เลือกใช้การเรียงบิตข้อมูลสีให้เป็นแบบ BGR นั่นคือ มีการเรียกใช้ฟังก์ชันนี้ตั้งแต่ต้น จึงไม่ต้องเขียนฟังก์ชันนี้ลงในโปรแกรม

รูปแบบ

```
glcdSetColorWordBGR()
```

ตัวอย่างที่ 6-6

```
#include <ipst.h>
void setup()
{
    glcdSetColorWordBGR(); // เขียนฟังก์ชันนี้ลงไปหรือไม่ก็ได้
}
void loop()
{
}
```

6.3.5 glcdSetColorWordRGB();

เป็นฟังก์ชันกำหนดการเรียงบิตข้อมูลสีให้เป็นแบบ RGB (5-6-5) นั่นคือ ค่าของสีแดง 5 บิต ต่อด้วยสีเขียว 6 บิต และปิดท้ายด้วยค่าของสีน้ำเงิน 5 บิต ทั้งนี้เนื่องจากผู้ผลิตจอแสดงผลกราฟิก LCD สีมีการผลิตจอแสดงผลแบบนี้เป็น 2 รุ่น โดยมีการเรียงบิตข้อมูลสีแบบ BGR และแบบ RGB

หากผู้ใช้งานแผงวงจร IPST-SE และทดลองกำหนดสีของภาพหรือตัวอักษรแล้วพบว่า สีที่ได้ไม่ถูกต้อง จะต้องเรียกใช้ฟังก์ชันนี้ โดยบรรจุไว้ใน setup() ที่ตอนต้นของโปรแกรม

รูปแบบ

```
glcdSetColorWordRGB()
```

ตัวอย่างที่ 6-7

```
#include <ipst.h>
void setup()
{
    glcdSetColorWordBGR(); // เลือกรูปแบบการเรียงบิตสีเป็นแบบ RGB
}
void loop()
{
}
```

6.3.6 setTextColor

เป็นการกำหนดค่าสีของตัวอักษรที่แสดงด้วยฟังก์ชัน `gLCD()` โดยค่าตั้งต้นกำหนดเป็นสีขาว

รูปแบบ

```
void setTextColor(unsigned int newColor)
```

พารามิเตอร์

`newColor` คือสีที่ต้องการ เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร `color[]`

ตัวอย่างที่ 6-8

```
#include <ipst.h>
int colors;
void setup()
{
  int colors;
  colors=colorRGB(31,0,0); // ส่งค่าสี 16 บิตของสีแดงให้ตัวแปร colors
  setTextColor(colors); // กำหนดให้สีของตัวอักษรเป็นสีแดง
  gLCD(1,0,"Text color is Red"); // กำหนดข้อความ
}
void loop()
{
  setTextColor(GLCD_YELLOW); // กำหนดให้สีของตัวอักษรเป็นสีเหลือง
  gLCD(3,0,"Text color is Yellow"); // กำหนดข้อความ
}
```



6.3.7 setTextBackgroundColor

เป็นฟังก์ชันกำหนดสีของพื้นหลังตัวอักษร โดยค่าตั้งต้นเป็นสีดำ สีของพื้นหลังตัวอักษรจะเป็นคนละส่วนกับสีของพื้นจอภาพ (screen background) ซึ่งต้องกำหนดค่าผ่านฟังก์ชัน `glcdFillScreen`

รูปแบบ

```
void setTextBackgroundColor(unsigned int newColor)
```

พารามิเตอร์

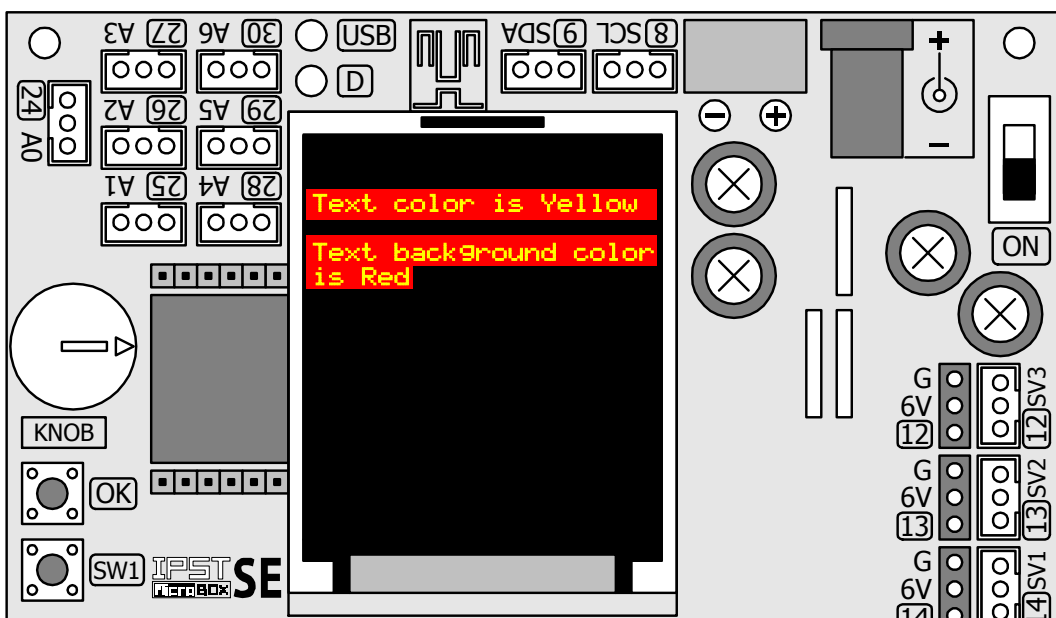
`newColor` คือสีที่ต้องการ เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร `color[]`

ตัวอย่างที่ 6-9

```
#include <ipst.h>

void setup()
{
    setTextBackgroundColor(GLCD_RED); // กำหนดให้สีพื้นหลังตัวอักษรเป็นสีแดง
}

void loop()
{
    setTextColor(GLCD_YELLOW); // กำหนดให้สีของตัวอักษรเป็นสีเหลือง
    glcd(3,0,"Text color is Yellow."); // กำหนดข้อความ
    glcd(5,0,"Text background color"); // กำหนดข้อความ
    glcd(6,0,"is Red."); // กำหนดข้อความ
}
```



6.3.8 glcdClear

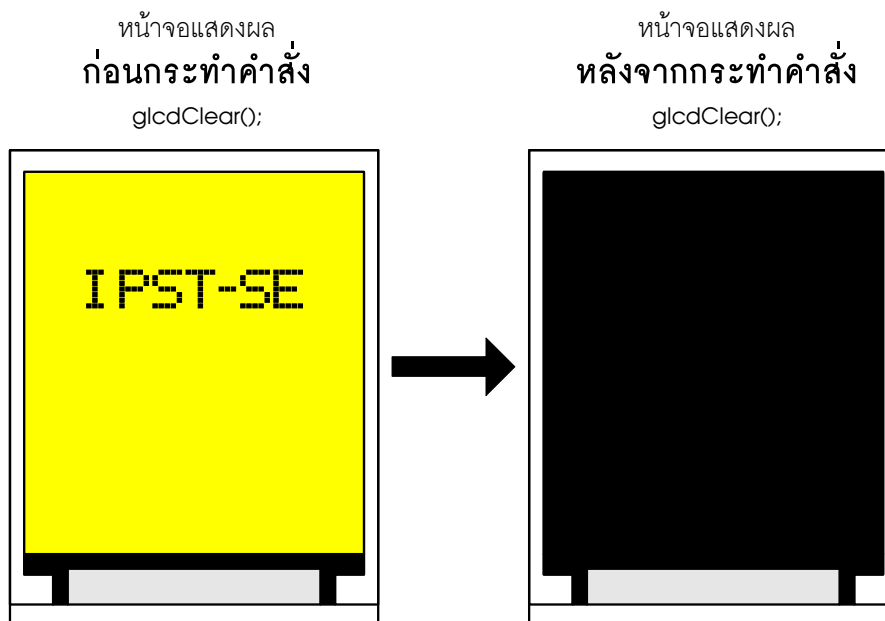
เป็นการเคลียร์หน้าจอแสดงผล โดยสีของพื้นหลังจะเป็นสีพื้นหลังของตัวอักษรล่าสุด โดยถ้าไม่ได้กำหนดด้วยคำสั่ง `setTextBackGroundColor()` มาก่อนหน้านี้ หลังจากทำคำสั่ง `glcdClear()` แล้ว พื้นหลังจะเป็นสีดำ

รูปแบบ

```
void glcdClear()
```

ตัวอย่างที่ 6-10

```
glcdClear(); // เคลียร์หน้าจอแสดงผล
```



6.3.9 glcdFillScreen

เป็นการเคลียร์หน้าจอแสดงผล แล้วเปลี่ยนสีพื้นหลังของจอแสดงผลด้วยสีที่ระบุ

รูปแบบ

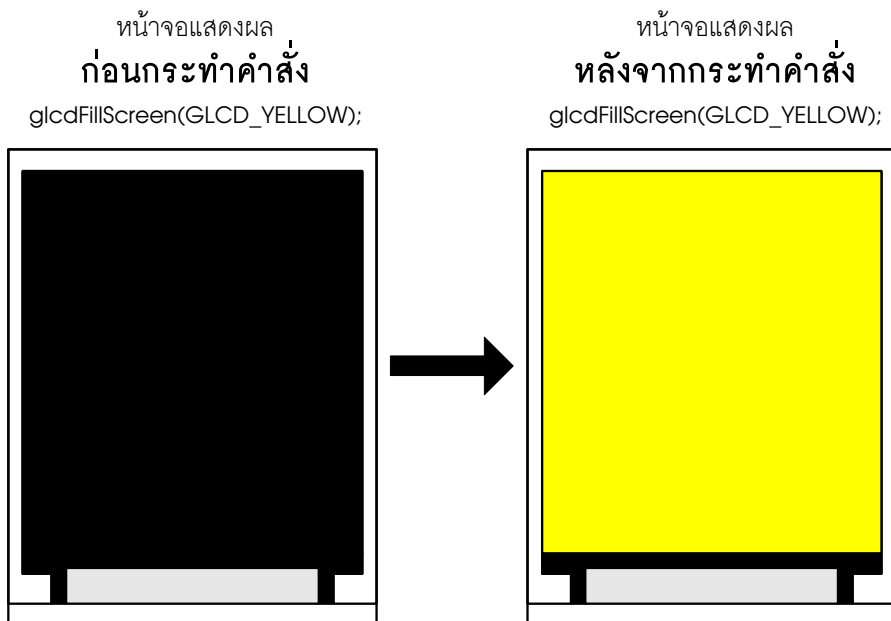
```
void glcdFillScreen(unsigned int color)
```

พารามิเตอร์

Color คือสีที่ต้องการ เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร color[]

ตัวอย่างที่ 6-11

```
glcdFillScreen(GLCD_YELLOW); // กำหนดสีพื้นหลังของจอภาพเป็นสีเหลือง
```



6.3.10 glcdMode

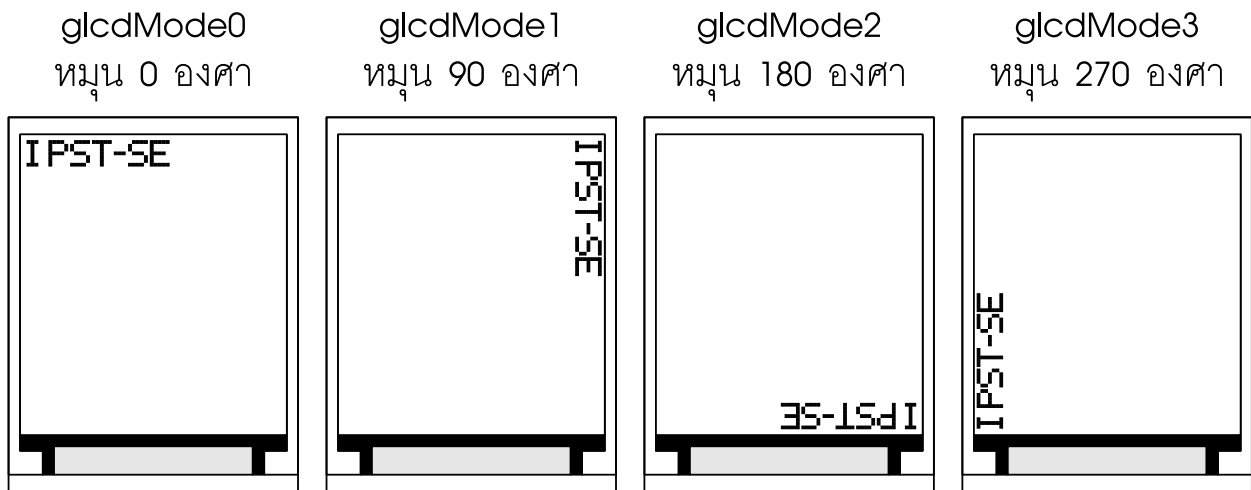
เป็นการกำหนดทิศทางแสดงผลของจอกราฟิก LCD สี โดยกำหนดให้ข้อความหรือภาพหน้าจอให้แสดงภาพตั้งฉากตรงหน้า (โหมด 0), หมุนขวา 90 องศา (โหมด 1), หมุน 180 องศาหรือกลับหัว (โหมด 2) และหมุน 270 องศา (โหมด 3)

รูปแบบ

```
glcdMode(unsigned int modeset)
```

พารามิเตอร์

modeset คือค่าทิศทางของการหมุนมีค่า 0 ถึง 3 โดยใช้แทนทิศทาง 0, 90, 180 หรือ 270 องศา โดยมีค่าเริ่มต้นคือ 0 องศา ทำงานอยู่ในแนวตั้ง



ตัวอย่างที่ 6-12

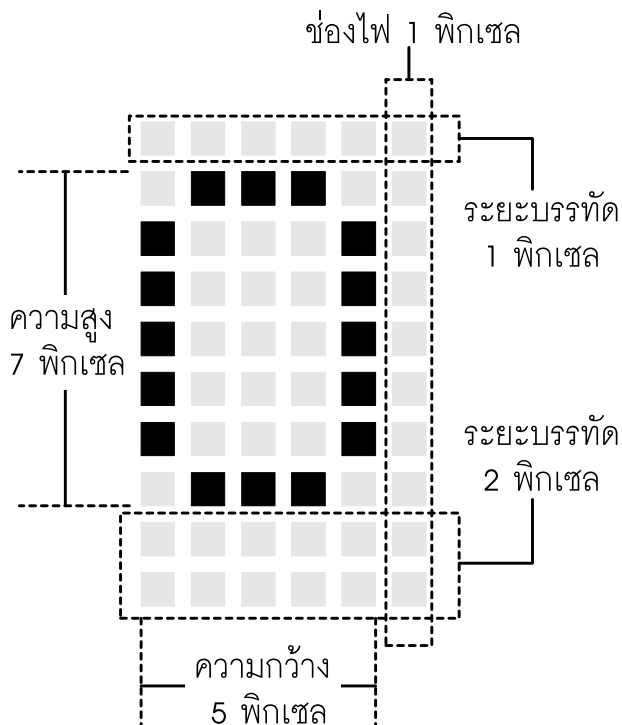
```

#include <ipst.h>
void setup()
{
    setTextSize(2);    // ขนาดตัวอักษร 2 เท่า
}
void loop()
{
    glcdClear();      // เคลียร์หน้าจอ
    glcdMode(0);      // โหมด 0 องศา
    glcd(0,0,"IPST-SE"); // แสดงข้อความ
    sw_ok_press();    // รอกดสวิตช์เข้าโหมดต่อไป
    glcdClear();
    glcdMode(1);
    glcd(0,0,"IPST-SE");
    sw_ok_press();
    glcdClear();
    glcdMode(2);
    glcd(0,0,"IPST-SE");
    sw_ok_press();
    glcdClear();
    glcdMode(3);
    glcd(0,0,"IPST-SE");
    sw_ok_press();
}

```

6.3.11 setTextSize

เป็นการกำหนดขนาดตัวอักษร โดยระบุเป็นจำนวนเท่าของขนาดปกติ ค่าตั้งต้นเมื่อเริ่มทำงานทุกครั้งคือ ขนาดตัวอักษรปกติ ใช้พื้นที่รวมระยะช่องไฟคือ 6 x 10 พิกเซลต่อ 1 ตัวอักษร จึงแสดงได้ 21 ตัวอักษร 16 บรรทัดในแนวตั้ง



รูปแบบ

```
setTextSize(unsigned int newSize)
```

พารามิเตอร์

newSize คือค่าขนาดจำนวนเท่าของขนาดปกติ มีค่า 1 ถึง 16 เพื่อให้ตัวอักษรที่แสดงไม่ล้นหน้าจอ

ถ้าหากกำหนดเป็น 1 เท่า แสดงตัวอักษรได้ 21 ตัว 16 บรรทัด

ถ้าหากกำหนดเป็น 2 เท่า แสดงตัวอักษรได้ 10 ตัว 8 บรรทัด

ถ้าหากกำหนดเป็น 3 เท่า แสดงตัวอักษรได้ 7 ตัว 5 บรรทัด

ถ้าหากกำหนดเป็น 4 เท่า แสดงตัวอักษรได้ 5 ตัว 4 บรรทัด

ถ้าหากกำหนดเป็น 5 เท่า แสดงตัวอักษรได้ 4 ตัว 3 บรรทัด

ถ้าหากกำหนดเป็น 6 และ 7 เท่า แสดงตัวอักษรได้ 3 ตัว 2 บรรทัด

ถ้าหากกำหนดเป็น 8 เท่า แสดงตัวอักษรได้ 2 ตัว 2 บรรทัด

ถ้าหากกำหนดเป็น 9 และ 10 เท่า แสดงตัวอักษรได้ 2 ตัว 1 บรรทัด

ถ้าหากกำหนดเป็น 11 ถึง 16 เท่า แสดงตัวอักษรได้ 1 ตัว 1 บรรทัด (แบบไม่ล้นจอแสดงผล)

ตัวอย่างที่ 6-13

```
#include <ipst.h>
void setup()
{
  setTextSize(1);           // กำหนดขนาดข้อความ 1 เท่า
  setTextColor(GLCD_GREEN); // สีตัวอักษรเป็นสีเขียว
  glcd(0,0,"Size1");       // แสดงข้อความ
  setTextSize(2);
  glcd(1,0,"Size2");       // กำหนดขนาดข้อความ 2 เท่า
  setTextSize(3);
  glcd(2,0,"Size3");       // กำหนดขนาดข้อความ 3 เท่า
  setTextSize(4);
  glcd(3,0,"Size4");       // กำหนดขนาดข้อความ 4 เท่า
}
void loop()
{}
```



6.3.12 getTextColor

เป็นคำสั่งคืนค่าสีปัจจุบันของตัวอักษร

รูปแบบ

```
unsigned int getTextColor()
```

การคืนค่า

textColor เป็นค่าสีแสดงอยู่ในรูปของตัวเลข 16 บิต ดูรูปแบบได้จากฟังก์ชัน colorRGB()

ตัวอย่างที่ 6-14

```
unsigned int color;
color=getTextColor(); // นำค่าสีของตัวอักษรเก็บไว้ที่ตัวแปร color
```

6.3.13 getTextBackgroundColor

เป็นคำสั่งคืนค่าสีพื้นหลังของตัวอักษรในปัจจุบัน

รูปแบบ

```
unsigned int getTextBackgroundColor()
```

การคืนค่า

textBackgroundColor เป็นค่าสีแสดงอยู่ในรูปของตัวเลข 16 บิต ดูรูปแบบได้จากฟังก์ชัน colorRGB ()

ตัวอย่างที่ 6-15

```
unsigned int color;
color=getTextBackgroundColor(); // นำค่าสีพื้นหลังของตัวอักษรเก็บในตัวแปร color
```

6.3.14 getTextSize

คืนค่าขนาดของตัวอักษรออกมาเป็นจำนวนเท่าของค่าปกติ

รูปแบบ

```
unsigned int getTextSize()
```

การคืนค่า

textSize เป็นค่าจำนวนเท่าของขนาดตัวอักษร

ตัวอย่างที่ 6-16

```
unsigned int textSize;
textSize=getTextSize(); // นำค่าจำนวนเท่าของขนาดของตัวอักษรเก็บในตัวแปร textSize
```

6.3.15 glcdGetMode

เป็นคำสั่งคืนค่าของโหมดทิศทางการแสดงผลในปัจจุบัน

รูปแบบ

```
unsigned int glcdGetMode()
```

การคืนค่า

mode เป็นค่าของโหมดทิศทางการแสดงผล เป็นตัวเลข 0 ถึง 3 เพื่อแสดงผลในทิศทาง 0 องศา, หมุน 90 องศา, หมุน 180 องศา และหมุน 270 องศาตามลำดับ

ตัวอย่างที่ 6-17

```
unsigned int Mode;
Mode=glcdGetMode(); // คืนค่าทิศทางการแสดงผลของหน้าจอ GLCD
```

6.3.16 glcdPixel

เป็นคำสั่งพล็อตจุดบนจอภาพตามพิกัดที่กำหนด โดยอ้างอิงถึงตำแหน่งของพิกเซลหรือจุดของจอแสดงผลที่มีความละเอียด 128 x 160 จุด ดังรูปที่ 6-2

รูปแบบ

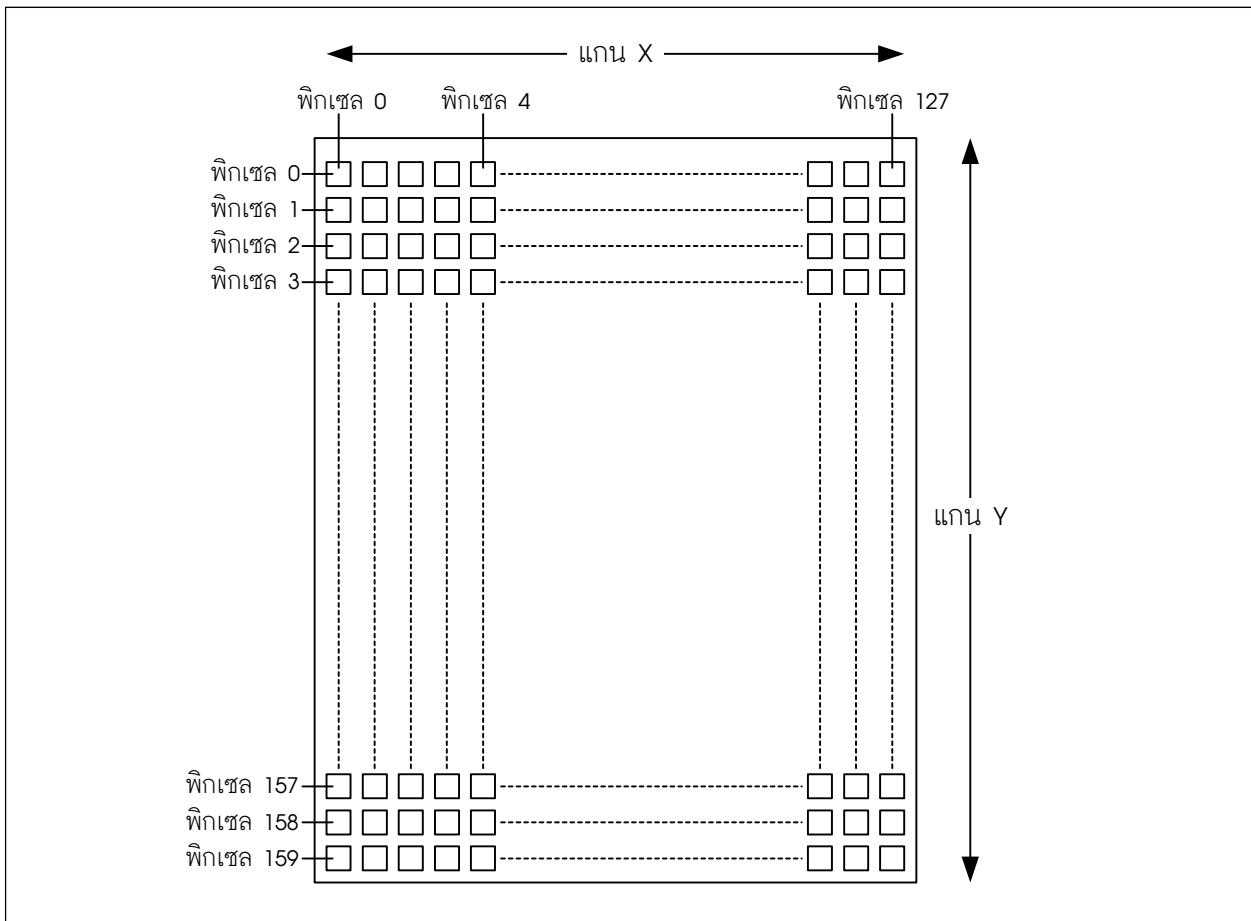
```
void glcdPixel(unsigned int x,unsigned int y,unsigned int color)
```

พารามิเตอร์

x คือค่าพิกัดในแนวนอนหรือแกน x มีค่าระหว่าง 0 ถึง 127

y คือค่าพิกัดในแนวตั้งหรือแกน y มีค่าระหว่าง 0 ถึง 159

color คือค่าของสีที่ต้องการ เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร color[]



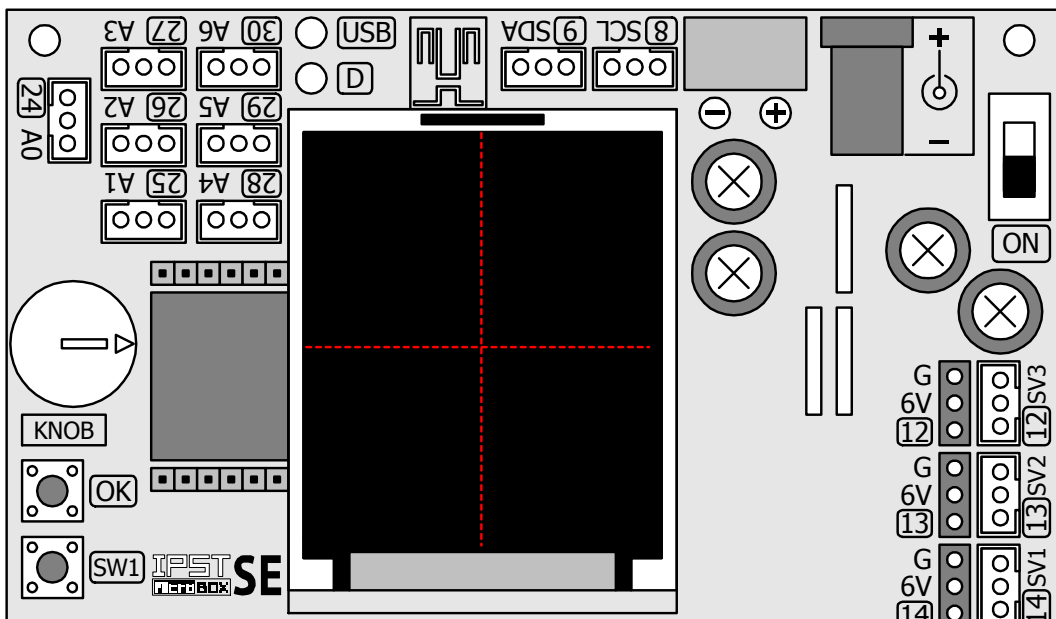
รูปที่ 6-2 แสดงการกำหนดตำแหน่งของพิกเซลหรือจุดของจอแสดงผลกราฟิก LCD สีที่ใช้บนแผงวงจร IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE)

ตัวอย่างที่ 6-18

```

#include <ipst.h>
int i;
void setup()
{
  for (i=0;i<128;i+=4)
  {
    glcdPixel(i,80,GLCD_RED);    // พล็อตจุดทุกๆ 4 พิกเซลในแนวแกน x กลางจอ
  }
  for (i=0;i<160;i+=4)
  {
    glcdPixel(64,i,GLCD_RED);    // พล็อตจุดทุกๆ 4 พิกเซลในแนวแกน y กลางจอ
  }
}
void loop()
{}

```



6.3.1.17 glcdRect

เป็นฟังก์ชันลากเส้นจากพิกัดที่กำหนดมายังพิกัดปลายทาง โดยอ้างอิงถึงตำแหน่งของพิกเซลหรือจุดของจอแสดงผลที่มีความละเอียด 128 x 160 จุด ตามรูปที่ 6-2

รูปแบบ

```
void glcdRect(unsigned int x1,unsigned int y1,unsigned int width,unsigned int height,unsigned int color)
```

พารามิเตอร์

x1 คือ ค่าตำแหน่งเริ่มต้นของรูปสี่เหลี่ยมในแกน x มีค่าระหว่าง 0 ถึง 127

y1 คือ ค่าตำแหน่งเริ่มต้นของรูปสี่เหลี่ยมในแกน y มีค่าระหว่าง 0 ถึง 159

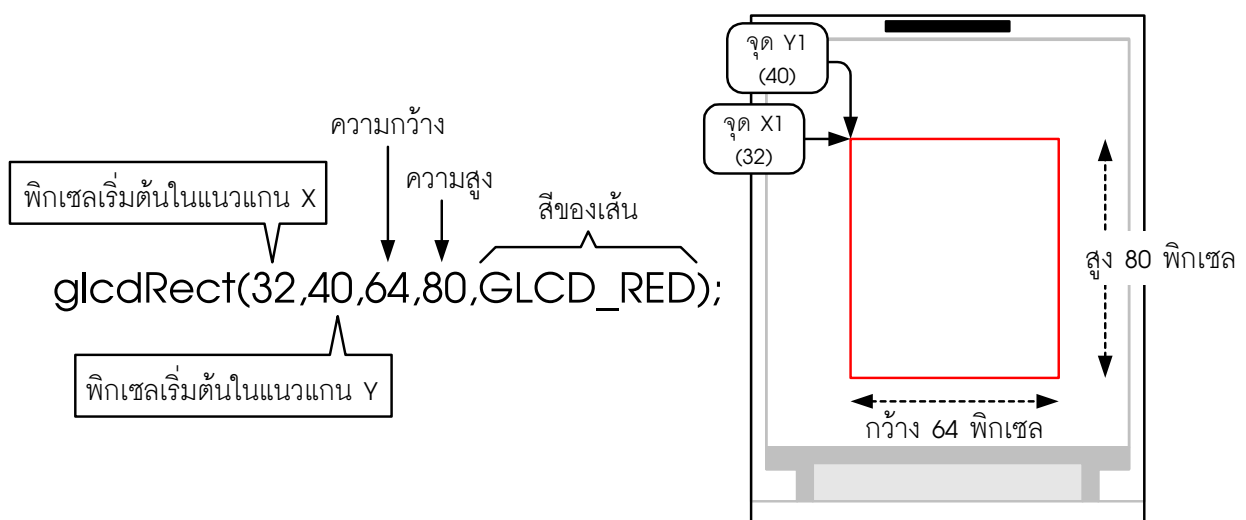
width คือ ค่าความกว้างของรูปสี่เหลี่ยม (แนวนอน) มีค่าระหว่าง 1 ถึง 128

height คือ ค่าความสูงของรูปสี่เหลี่ยม (แนวตั้ง) มีค่าระหว่าง 1 ถึง 158

color คือ สีของเส้น เป็นค่าตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร color[]

ตัวอย่างที่ 6-19

```
#include <ipst.h>
void setup()
{
  glcdRect(32,40,64,80,GLCD_RED); // วาดรูปสี่เหลี่ยมสีแดง ขนาด 64 x 80 พิกเซล
}
void loop()
{
}
```



6.3.18 glcdFillRect

เป็นการระบายสีพื้นของรูปสี่เหลี่ยม โดยกำหนดจุดเริ่มต้นและความกว้างยาวของรูปสี่เหลี่ยมที่ต้องการ ฟังก์ชันนี้เป็นการสร้างรูปสี่เหลี่ยมที่มีสีพื้นแต่ไม่มีเส้นกรอบ ในขณะที่ฟังก์ชัน glcdRect เป็นการวาดรูปกรอบสี่เหลี่ยมที่กำหนดสีของเส้นกรอบได้ แต่ภายในกรอบไม่มีสี

รูปแบบ

```
void glcdFillRect(unsigned int x1, unsigned int y1, unsigned int width, unsigned int height, unsigned int color)
```

พารามิเตอร์

x1 คือ ค่าตำแหน่งเริ่มต้นของรูปสี่เหลี่ยมในแกน x มีค่าระหว่าง 0 ถึง 127

y1 คือ ค่าตำแหน่งเริ่มต้นของรูปสี่เหลี่ยมในแกน y มีค่าระหว่าง 0 ถึง 159

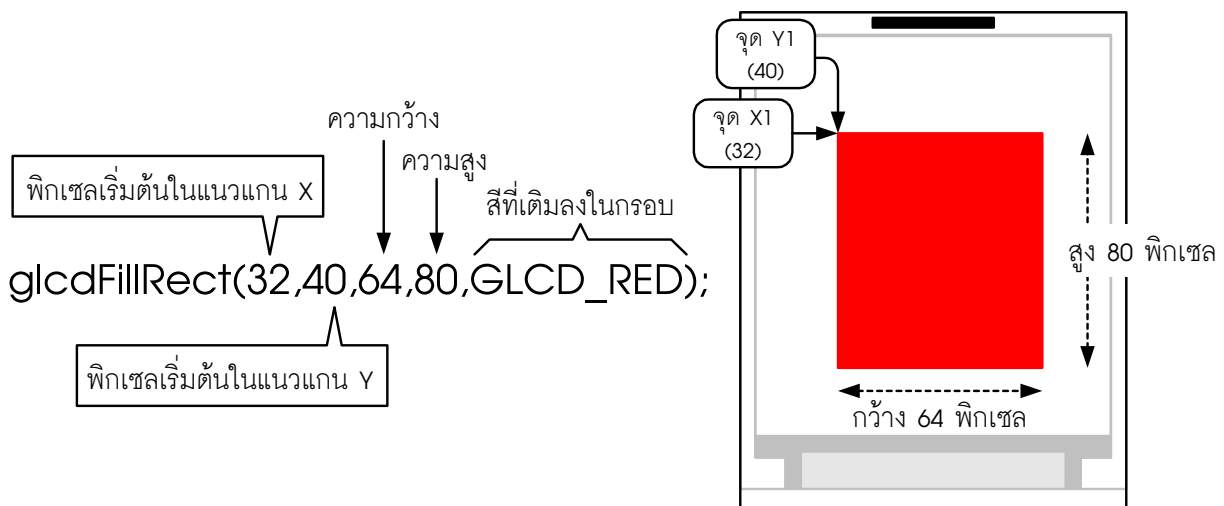
width คือ ค่าความกว้างของรูปสี่เหลี่ยม (แนวนอน) มีค่าระหว่าง 1 ถึง 128

height คือ ค่าความสูงของรูปสี่เหลี่ยม (แนวตั้ง) มีค่าระหว่าง 1 ถึง 158

color คือ สีของเส้น เป็นค่าตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร color[]

ตัวอย่างที่ 6-20

```
#include <ipst.h>
void setup()
{
    glcdFillRect(32,40,64,80,GLCD_RED);
    // สร้างภาพสี่เหลี่ยมพื้นสีแดง ขนาด 64 x 80 พิกเซล
}
void loop()
{
}
```



6.3.19 glcdLine

เป็นฟังก์ชันลากเส้นจากจุดหนึ่งไปยังอีกจุดหนึ่ง กำหนดเป็นพิกัดในแนวแกนอน (x) และ แกนตั้ง (y)

รูปแบบ

```
void glcdLine(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned int color)
```

พารามิเตอร์

x1 คือค่าตำแหน่งเริ่มต้นของเส้นบนแกน x มีค่าระหว่าง 0 ถึง 127

y1 คือค่าตำแหน่งเริ่มต้นของเส้นบนแกน y มีค่าระหว่าง 0 ถึง 159

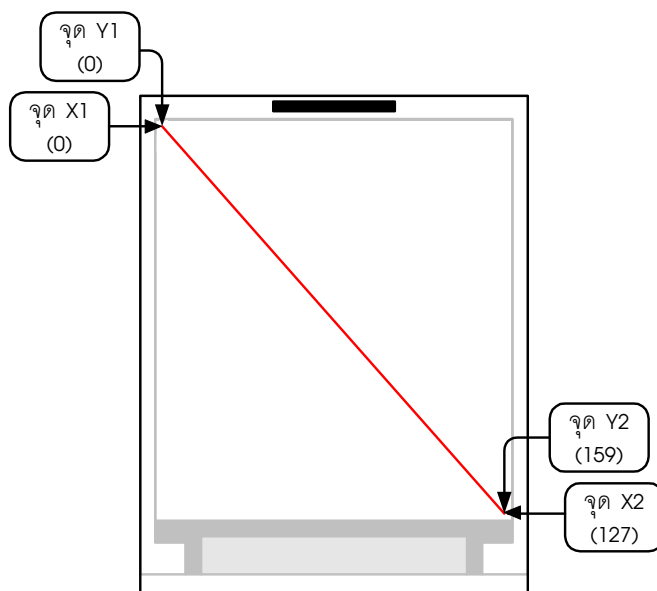
x2 คือค่าตำแหน่งสิ้นสุดของเส้นบนแกน x มีค่าระหว่าง 0 ถึง 127

y2 คือค่าตำแหน่งสิ้นสุดของเส้นบนแกน y มีค่าระหว่าง 0 ถึง 159

color คือ ค่าสีของเส้น เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร color[]

ตัวอย่างที่ 6-21

```
#include <ipst.h>
void setup()
{
    glcdLine(0,0,127,159,GLCD_RED); // ลากเส้นสีแดงทแยงมุมจากด้านบนซ้ายลงมาด้านล่างขวา
}
void loop()
{
}
```



6.3.20 glcdCircle

เป็นฟังก์ชันวาดเส้นรูปวงกลมจากการกำหนดจุดกึ่งกลางของวงกลมและความยาวของรัศมี

รูปแบบ

```
void glcdCircle(unsigned int x, unsigned int y, unsigned int radius, unsigned int color)
```

พารามิเตอร์

x คือ พิกัดจุดศูนย์กลางของวงกลมบนแกน x มีค่าระหว่าง 0 ถึง 127

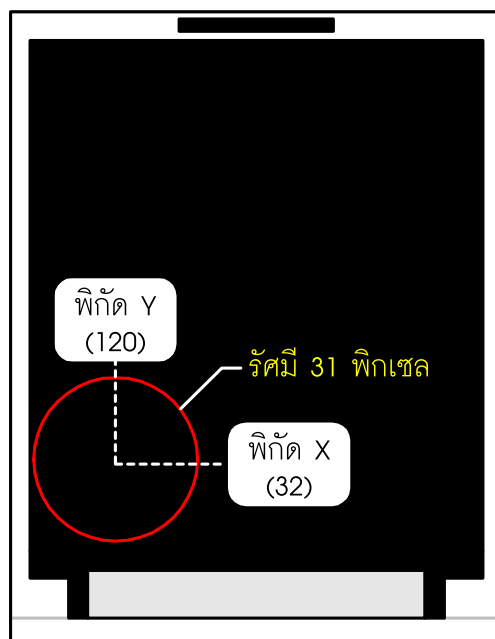
y คือ พิกัดจุดศูนย์กลางของวงกลมบนแกน y มีค่าระหว่าง 0 ถึง 159

radius คือ ค่ารัศมีของวงกลม

color คือ ค่าสีของเส้น เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร color[]

ตัวอย่างที่ 6-22

```
#include <ipst.h>
void setup()
{
    glcdCircle(32,120,31,GLCD_RED);// สร้างเส้นวงกลมสีแดง มีรัศมี 31 พิกเซล
}
void loop()
{
}
```



6.3.21 glcdFillCircle

เป็นฟังก์ชันวาดรูปวงกลมที่มีสีพื้นจากการกำหนดจุดศูนย์กลางของวงกลม และความยาวของรัศมี ฟังก์ชันนี้เป็นการสร้างรูปวงกลมที่มีสีพื้นแต่ไม่มีเส้นกรอบ ในขณะที่ฟังก์ชัน `glcdCircle` เป็นการวาดรูปวงกลมที่กำหนดสีของเส้นรอบวงได้ แต่ภายในวงกลมไม่มีสี

รูปแบบ

```
void glcdFillCircle(unsigned int x,unsigned int y,unsigned int radius,unsigned int color)
```

พารามิเตอร์

x คือ พิกัดจุดศูนย์กลางของวงกลมบนแกน x มีค่าระหว่าง 0 ถึง 127

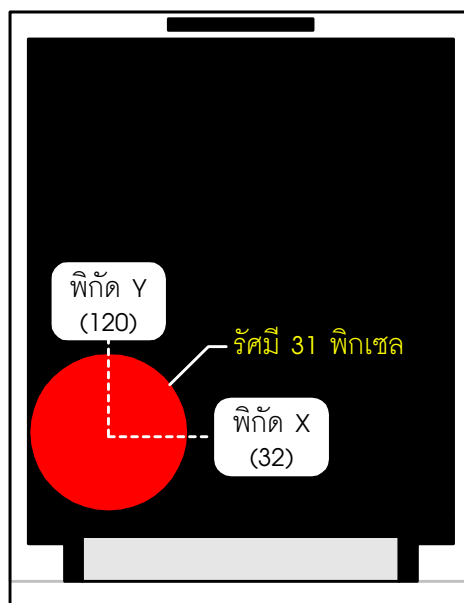
y คือ พิกัดจุดศูนย์กลางของวงกลมบนแกน y มีค่าระหว่าง 0 ถึง 159

radius คือ ค่ารัศมีของวงกลม

color คือ ค่าสีของเส้น เป็นตัวเลข 16 บิต หรือเป็นค่าตัวแปรที่กำหนดค่าไว้แล้วจากตัวแปร `color[]`

ตัวอย่างที่ 6-23

```
#include <ipst.h>
void setup()
{
  glcdFillCircle(32,120,31,GLCD_RED); // สร้างรูปวงกลมพื้นสีแดง รัศมี 31 พิกเซล
}
void loop()
{}
```



6.3.22 glcdArc

เป็นฟังก์ชันวาดส่วนโค้งของวงกลม โดยระบุตำแหน่งจุดกึ่งกลาง รัศมี ตำแหน่งจุดเริ่ม จุดสิ้นสุดและสีของเส้น

รูปแบบ

```
void glcdArc(unsigned int x,unsigned int y,unsigned int r,int start_angle,int end_angle,uint color)
```

พารามิเตอร์

x คือตำแหน่งจุดกึ่งกลางในแนวแกน x

y คือตำแหน่งจุดกึ่งกลางในแนวแกน y

r คือรัศมีของเส้นโค้ง

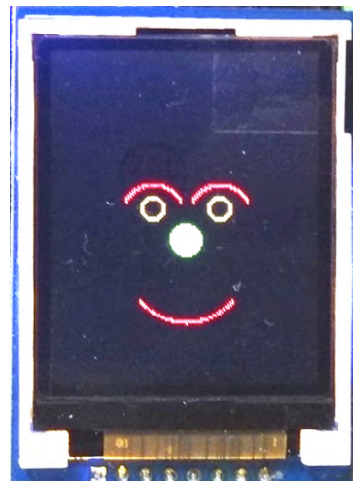
start_angle คือตำแหน่งมุมของจุดเริ่มต้นของวงกลม

end_angle คือตำแหน่งมุมจุดสิ้นสุดของวงกลม

color คือสีของเส้นวงกลม

ตัวอย่างที่ 6-24

```
#include <ipst.h>
void setup()
{
  glcdArc(48,80,16,30,150,GLCD_RED);
  glcdCircle(48,75,5,GLCD_YELLOW);
  glcdCircle(80,75,5,GLCD_YELLOW);
  glcdArc(80,80,16,30,150,GLCD_RED);
  glcdFillCircle(64,90,7,GLCD_GREEN);
  glcdArc(64,100,30,220,320,GLCD_RED);
}
void loop()
{
}
```



บทที่ 7

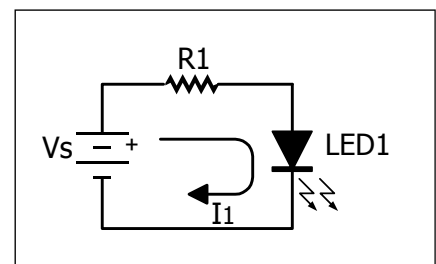
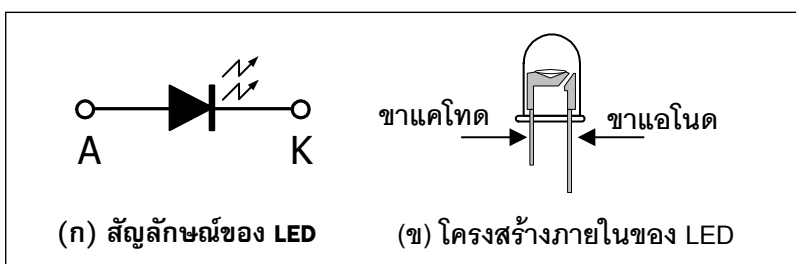
ควบคุมการติดดับ ของ LED ด้วยซอฟต์แวร์

7.1 LED คืออะไร

LED (Light Emitting Diode) หรือไดโอดเปล่งแสงเป็นอุปกรณ์เอาต์พุตสำหรับการแสดงผล ซึ่งสามารถติดสว่างได้เมื่อได้รับแรงดันกระตุ้นอย่างเหมาะสม โดย LED มีให้เลือกใช้งานได้หลายสี อาทิ สีแดง, เหลือง, เขียว, น้ำเงิน, ขาว, ส้ม, ม่วง เป็นต้น

LED สามารถกำเนิดแสงออกมาได้เมื่อได้รับจ่ายไฟอย่างถูกต้อง การจ่ายไฟให้ LED ทำงาน เรียกว่า การไบแอส (bias) และการไบแอสที่ทำให้ LED ทำงาน เรียกว่า การไบแอสตรง (forward bias) โดยปกติแรงดันที่ใช้ในการขับหรือไบแอสให้ LED ทำงานจะมีค่าอยู่ระหว่าง 1-4.5 V ขึ้นอยู่กับสารที่นำมาใช้ทำ LED และขึ้นอยู่กับปริมาณกระแสที่ไหลผ่าน กล่าวคือ ถ้ากระแสไหลผ่าน LED มาก มีผลทำให้แรงดันที่ตกคร่อม LED มีค่ามากและแสงที่ LED กำเนิดออกมาก็จะสว่างมากขึ้นด้วย โดยปกติมักจะกำหนดให้แรงดันไบแอสตรงของ LED เท่ากับ 2V ในรูปที่ 7-1 เป็นสัญลักษณ์, โครงสร้าง และการจัดขาของ LED

LED สามารถเปล่งแสงออกได้หลายสีขึ้นอยู่กับว่า นำสารกึ่งตัวนำชนิดใดมาสร้างเป็น LED ถ้าหากเป็นสีแดงและเหลืองทำมาจากแกลเลียมอาร์เซไนด์ฟอสไฟด์ (GaAsP) ส่วนสีเขียวทำมาจาก แกลเลียมฟอสไฟด์ (GaP) และ LED ที่ให้แสงอินฟราเรดซึ่งทำมาจากแกลเลียมอาร์เซไนด์ (GaAs)



รูปที่ 7-1 แสดงสัญลักษณ์, โครงสร้างและการจัดขาของไดโอดเปล่งแสงหรือ LED

รูปที่ 7-2 การต่อตัวต้านทานเพื่อจำกัดกระแสให้แก่ LED

7.2 การใช้งาน LED

LED ต้องการแรงดันไบแอสตรงประมาณ 2V และยอมให้กระแสไฟฟ้าไหลผ่านได้ไม่เกิน 40mA แต่ปริมาณกระแสไฟฟ้าที่เหมาะสมคือ 10 ถึง 20mA ดังนั้นการใช้งาน LED จึงต้องมีตัวต้านทานจำกัดกระแสไฟฟ้าต่ออนุกรมร่วมอยู่ด้วย ดังในรูปที่ 7-2

การหาค่าของตัวต้านทานที่ใช้ในการจำกัดกระแสไฟฟ้าให้ LED ทำได้โดยใช้สูตร

$$R_S = \frac{V_{CC} - V_F}{I_F}$$

โดยที่ V_{CC} คือไฟเลี้ยง

V_F คือแรงดันไบแอสตรงที่ตกคร่อม LED

I_F คือกระแสไฟฟ้าที่ไหลผ่าน LED เมื่อได้รับไบแอสตรง

ในทางตรงกันข้าม หากจ่ายแรงดันไบแอสกลับแก่ LED นอกจาก LED จะไม่ทำงานแล้ว อาจทำให้ LED เสียหายเนื่องจาก LED มีอัตราความทนแรงดันย้อนกลับได้ไม่สูงนัก เพียง 3 ถึง 10V เท่านั้น

7.3 การควบคุม LED ด้วยไมโครคอนโทรลเลอร์และโปรแกรมภาษา C

การควบคุมการติด/ดับของ LED นั้นผู้พัฒนาสามารถใช้สัญญาณจากจุดต่อพอร์ตใดๆ ของแผงวงจร IPST-SE ก็ได้ โดยในการเขียนโปรแกรมควบคุมการติด/ดับของ LED นั้นใช้กลุ่มคำสั่งเอาต์พุตเพื่อควบคุมสถานะของจุดต่อพอร์ตที่เชื่อมต่อกับ LED

ฟังก์ชันหรือคำสั่งหลักของโปรแกรมภาษา C/C++ สำหรับชุดกล่องสมองกล IPST-MicroBOX (SE) ที่นำมาใช้ควบคุมการทำงานของ LED คือ `out` ซึ่งบรรจุอยู่ในไลบรารี `ipst_in_out.h` ในการใช้งานจึงต้องผนวกไฟล์ไลบรารี `ipst_in_out.h` หรือ `ipst.h` ด้วยคำสั่ง `#include` ที่ส่วนหัวของโปรแกรมก่อนเสมอ ฟังก์ชัน `out` มีรูปแบบและการเรียกใช้ดังนี้

`out` เป็นฟังก์ชันกำหนดระดับสัญญาณหรือข้อมูลดิจิทัลไปยังขาพอร์ตที่กำหนด

รูปแบบ

```
out(char _bit, char _dat)
```

พารามิเตอร์

`_bit` - กำหนดขาพอร์ตที่ต้องการ มีค่า 0 ถึง 50 สำหรับ IPST-SE ใช้ได้ถึง 30

`_dat` - กำหนดข้อมูลที่ต้องการส่งออก มีค่าเป็น 0 หรือ 1

ตัวอย่างที่ 7-1

```
out(17,1); // กำหนดให้ขาพอร์ต 17 เป็น "1"
out(18,0); // กำหนดให้ขาพอร์ต 18 เป็น "0"
```

ข้อควรปฏิบัติในการทดลองทางฮาร์ดแวร์ของชุดกล่องสมองกล



เพื่อให้เครื่องมือและอุปกรณ์อยู่ในสภาพที่พร้อมทำงานตลอดเวลา สิ่งที่ต้องระวังทุกครั้งที่ใช้งานชุดกล่องสมองกล IPST-MicroBOX คือ

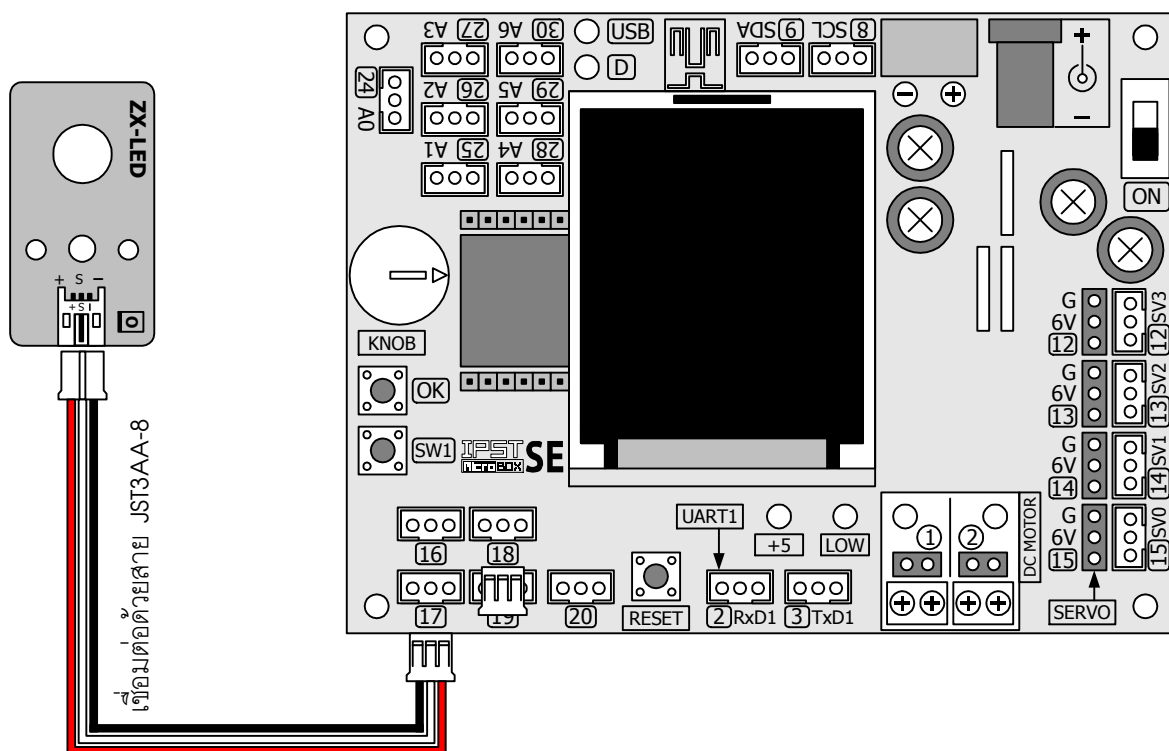
1. ปิดสวิตซ์ไฟเลี้ยงทุกครั้งที่มีการถอดหรือต่อสายเข้ากับคอมพิวเตอร์และชุดโปรแกรม
2. ปิดสวิตซ์ไฟเลี้ยงทุกครั้งที่มีการต่อหรือปลดสายของแผงวงจรตรวจจับสัญญาณหรืออุปกรณ์ใดๆ เข้ากับแผงวงจรควบคุม IPST-SE
3. หลังจากทดลองเสร็จในแต่ละการทดลอง ควรปิดสวิตซ์ไฟเลี้ยงก่อนที่จะทำการปลดสายสัญญาณเพื่อต่อแผงวงจรใหม่เข้าไปเพื่อทำการทดลองในหัวข้อใหม่
4. ไม่ควรปลดหรือต่อสายสัญญาณของแผงวงจรใดๆ เข้ากับแผงวงจร IPST-SE ในขณะที่แผงวงจรกำลังทำงาน เว้นแต่มีขั้นตอนการปฏิบัติอื่นใดที่ระบุเจาะจงว่าต้องสายสัญญาณในขณะทำงานของการทดลองนั้นๆ
5. หากมีความผิดพลาดใดๆ เกิดขึ้น ต้องปิดสวิตซ์ไฟเลี้ยงทันที
6. ไม่ใช้อะแดปเตอร์ไฟตรงที่มีแรงดันขาออกเกิน +9V กับแผงวงจร IPST-SE
7. หลังจากเสร็จสิ้นการทดลอง ให้ปลดสายเชื่อมต่อกับคอมพิวเตอร์และสายของอะแดปเตอร์หรือแหล่งจ่ายไฟออกจากแผงวงจร IPST-SE เสมอ

ปฏิบัติการที่ 1 ควบคุมการติด/ดับของ LED

ปฏิบัติการที่ 1-1 สั่งการให้ LED ติดสว่าง

การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อแผงวงจร ZX-LED เข้ากับจุดต่อพอร์ต 17 ของแผงวงจรหลัก IPST-SE



```
#include <ipst.h> // มนวกไฟล์ไลบรารีหลัก
void setup()
{}
void loop()
{
  out(17,1); // สั่งให้ LED ที่จุดต่อพอร์ต 17 ติดสว่าง
  delay(500); // หน่วงเวลา 0.5 วินาที
  out(17,0); // สั่งให้ LED ที่จุดต่อพอร์ต 17 ดับ
  delay(500); // หน่วงเวลา 0.5 วินาที
}
```

คำอธิบายโปรแกรม

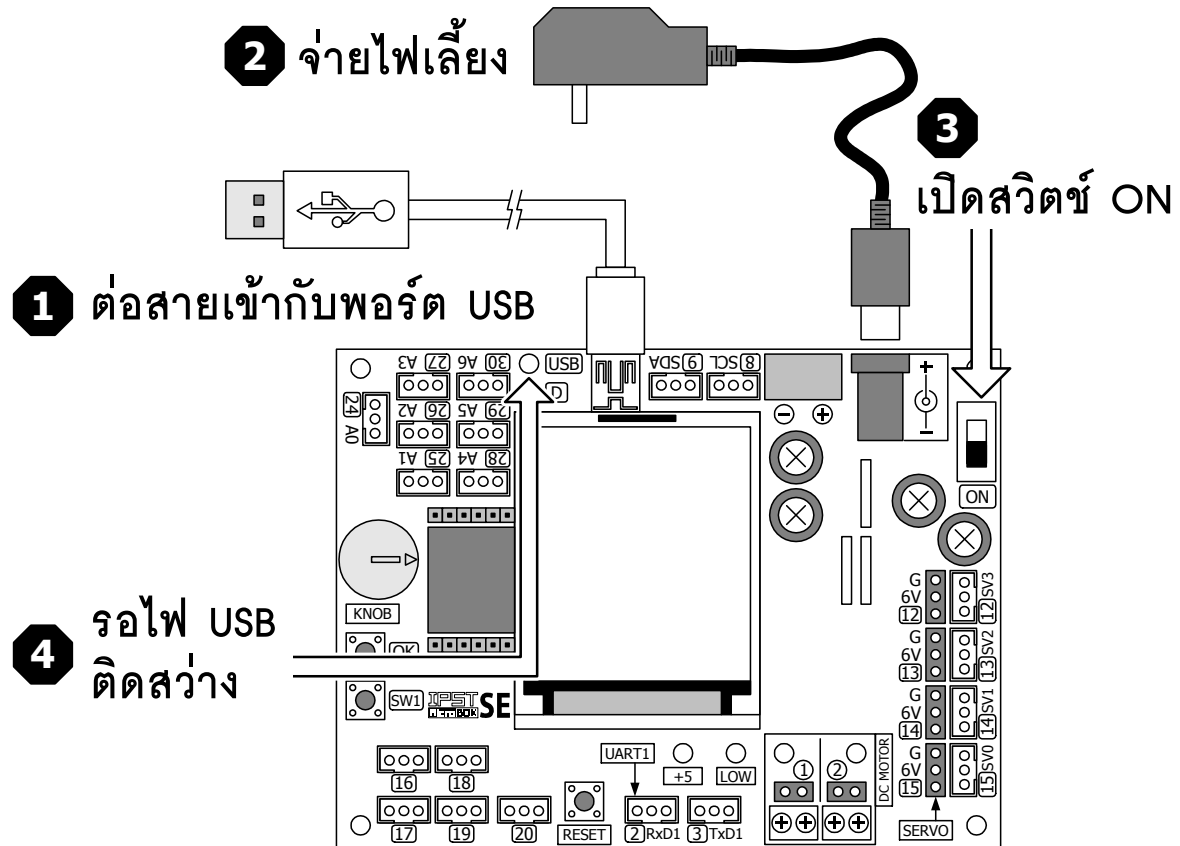
โปรแกรมจะสั่งให้ส่งสัญญาณ “1” ออกมาทางจุดต่อพอร์ต 17 สัญญาณ “1” มีแรงดันไฟฟ้า +5V จึงทำให้ LED ติดสว่างได้ และกำหนดให้ติดนาน 0.5 วินาที จากนั้นจึงสั่งให้ดับด้วยการส่งสัญญาณ “0” ซึ่งมีแรงดันไฟฟ้า 0V ทำให้ไม่มีแรงดันและกระแสไฟฟ้าส่งไปยัง LED ได้ LED ที่แผงวงจร ZX-LED จึงดับลง

โปรแกรมที่ L1-1 : ไฟล์ microbox_LedTest01.ino โปรแกรมภาษา C สำหรับทดลองขับ LED เบื้องต้น

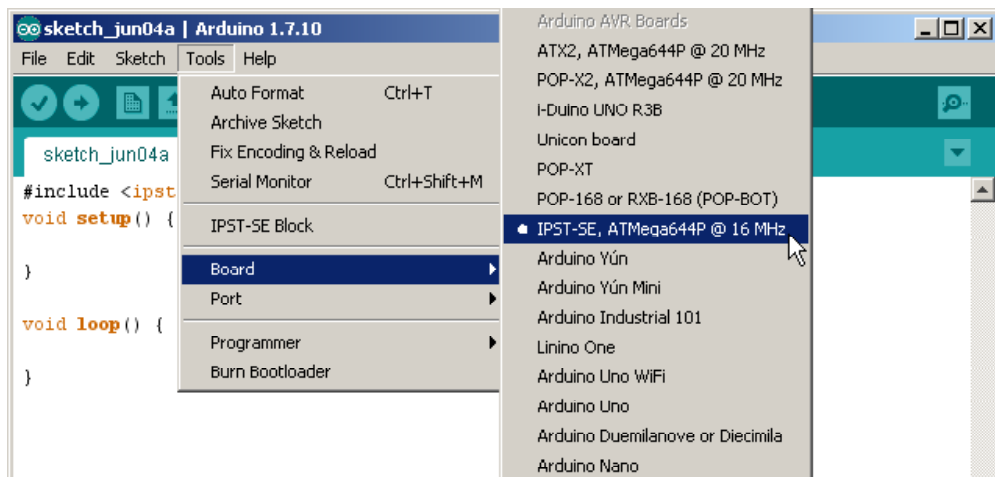
ขั้นตอนการทดลอง

1.1.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L1-1 บันทึกในชื่อ microbox_LEDtest01

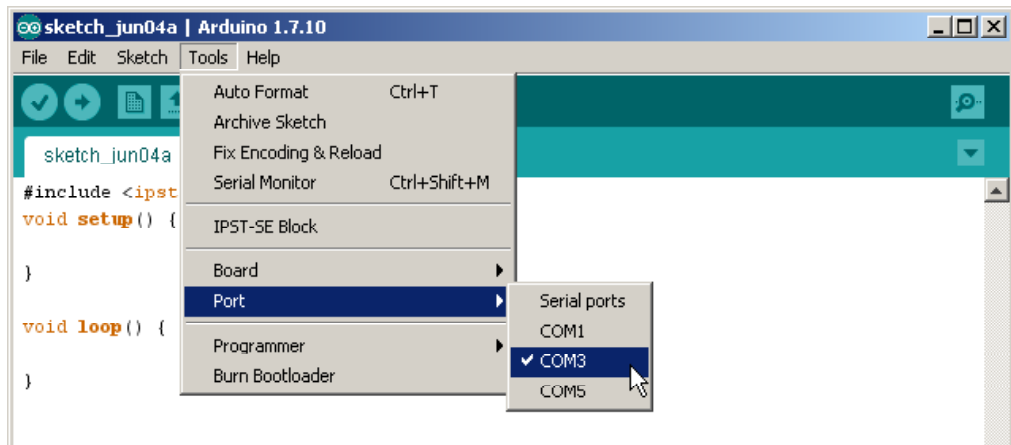
1.1.2 เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์




1.1.3 เลือกชนิดหรือรุ่นของฮาร์ดแวร์ให้ถูกต้อง โดยเลือกที่เมนู Tools > Board > IPST-SE > ATmega644P ดังรูป



1.1.4 เลือกพอร์ตที่ติดต่อกับแผงวงจร IPST-SE โดยเลือกที่เมนู Tools > Serial Port ดังรูป (ตำแหน่งของพอร์ตที่ใช้เชื่อมต่ออาจแตกต่างกันในคอมพิวเตอร์แต่ละเครื่อง)



1.1.5 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม  หรือเลือกที่เมนู File > Upload to Wiring Hardware

1.1.6 รันโปรแกรม สังเกตการทำงานของ LED

LED ที่ต่อกับจุดต่อพอร์ต 17 ติด-ดับทุกๆ 0.5 วินาที

เพิ่มเติม ผู้พัฒนาสามารถกำหนดอัตราในการติด-ดับหรืออัตราการกะพริบของ LED ได้ ด้วยการปรับค่าเวลาในฟังก์ชัน `delay()`; ภายในโปรแกรมที่ L1-1

1.1.7 ทดลองแก้ไขโปรแกรมที่ L1-1 ให้ LED มีความเร็วในการติด-ดับช้าลง

1.1.8 ทดลองแก้ไขโปรแกรมที่ L1-1 ให้ LED มีความเร็วในการติด-ดับเร็วขึ้น

1.1.9 ทดลองแก้ไขโปรแกรมที่ L1-1 เพื่อเปลี่ยนจุดต่อพอร์ตจากเดิมพอร์ตหมายเลข 17 เป็นพอร์ตอื่นๆ เช่น 18, 19 และ 20

ปฏิบัติการที่ 1-2 ควบคุม LED ด้วยเวลา

การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อแผงวงจร ZX-LED กับจุดต่อพอร์ต 17 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

1.2.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L1-2 บันทึกในชื่อ microbox_LEDtimer

1.2.2  หรือเลือกที่เมนู File > Upload to Wiring Hardware

1.2.3 รันโปรแกรม สังเกตการทำงานของ LED

LED จะติดนาน 4 วินาที หลังจากนั้นจึงดับลง

1.2.4 แก้ไขโปรแกรมที่ L1-2 ให้ LED ติดสว่างนานขึ้น

1.2.5 แก้ไขโปรแกรมที่ L1-2 ให้ LED ดับเร็วขึ้น

```
#include <ipst.h>    // มนวกไฟล์ไลบรารีหลัก
void setup()
{
}
void loop()
{
  out(17,1);        // สั่งให้ LED ที่จุดต่อพอร์ต 17 ติดสว่าง
  delay(4000);      // หน่วงเวลา 4 วินาที
  out(17,0);        // สั่งให้ LED ที่จุดต่อพอร์ต 17 ดับ
  while(1);         // วนทำงานที่คำสั่งนี้
}
```

คำอธิบายโปรแกรม

ในโปรแกรมจะกำหนดให้เขียนข้อมูล “1” ไปยังขาพอร์ต 17 ซึ่งต่อกับ ZX-LED ทำให้ LED ติดสว่างนานประมาณ 4 วินาทีก่อนที่จะดับลง เนื่องจากเมื่อสั่งให้ LED ติดสว่างจากการส่งขั้วสัญญาณเอาต์พุตเป็น “1” แล้ว ต่อด้วยการหน่วงเวลาด้วยฟังก์ชัน delay หรือ sleep โปรแกรมจะทำงานอยู่ที่คำสั่งนี้นานเป็นเวลาประมาณ 4000 มิลลิวินาที หรือ 4 วินาที หลังจากครบรอบเวลา จึงส่งสัญญาณ “0” เพื่อควบคุมให้ LED ดับ และวนอยู่ที่คำสั่ง while(); ทำให้ไม่มีการทำงานใดๆ LED จึงดับจนกว่าจะมีการกดสวิทช์ RESET หรือปิดเปิดไฟเลี้ยงใหม่

โปรแกรมที่ L1-2 : ไฟล์ microbox_Ledtimer.c โปรแกรมภาษา C สำหรับทดลอง
ควบคุมการทำงานของ LED ด้วยเวลา

บทที่ 8

การควบคุม LED หลายดวง ของชุดกล่องสมองกล

ในบทที่ผ่านมาเป็นการนำเสนอการควบคุม LED ด้วยกระบวนการทางซอฟต์แวร์ แต่ LED ที่ควบคุมนั้นมีเพียงช่องเดียว ในบทนี้นำเสนอการควบคุม LED ในจำนวนมากช่องขึ้นถึง 8 ช่องในเวลาเดียวกัน

อุปกรณ์เอาต์พุตที่นำมาใช้ในการเรียนรู้คือ แผงวงจร ZX-LED8 ซึ่งมี LED 8 ดวงพร้อมวงจรประกอบ การควบคุม LED ทั้ง 8 ดวงให้ทำงานพร้อมกัน หรือทำงานแยกกันต้องใช้ความรู้เกี่ยวกับระบบเลขฐานสองและโครงสร้างของข้อมูลทั้งแบบบิตและไบต์ในการกำหนดรูปแบบของข้อมูลที่นำมาแสดงผล

8.1 เลขฐานสอง

ในระบบตัวเลขฐานสองนี้มีตัวเลขเพียง 2 ตัวคือ “0” และ “1” ซึ่งสามารถใช้แทนสถานะต่ำสูง, เปิด-ปิด, ไม่ต่อ-ต่อ, ดับ-ติด เป็นต้น แต่ถ้าหากนำตัวเลขฐานสองมากกว่า 1 หลักมาพิจารณา เช่น 2 หลัก จะทำให้เกิดจำนวนของการเปลี่ยนแปลงเป็น 4 สถานะ หากแทนด้วยการติด-ดับของหลอดไฟ จะได้ ดับ-ดับ, ดับ-ติด, ติด-ดับ และติด-ติด ถ้าหากมี 3 หลักก็จะเกิดการเปลี่ยนแปลง 8 สถานะ จึงสามารถสรุปเป็นสมการคณิตศาสตร์และความสัมพันธ์ของจำนวนหลักและสถานะของการเปลี่ยนแปลงได้ดังนี้

จำนวนของการเปลี่ยนแปลง = 2 จำนวนหลัก

ถ้ามี 2 หลักจะได้จำนวนของการเปลี่ยนแปลง $2^2 = 4$

ถ้ามี 3 หลักจะได้จำนวนของการเปลี่ยนแปลง $2^3 = 8$

ถ้ามี 4 หลักจะได้จำนวนของการเปลี่ยนแปลง $2^4 = 16$

8.1.1 การนับจำนวนของระบบเลขฐานสอง

เนื่องจากเลขฐานสองมีตัวเลขเพียง 2 ตัวคือ 0 และ 1 เมื่อมีการนับจำนวนขึ้น จึงต้องมีการเพิ่มจำนวนหลัก ดังนั้นเพื่อให้เห็นการเปลี่ยนแปลงอย่างชัดเจนจะใช้เลขฐานสิบเป็นตัวเปรียบเทียบดังนี้

เลขฐานสอง	เลขฐานสิบ
00	0
01	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

8.1.2 ตัวแปรของเลขฐานสอง (bit variables)

เมื่อเลขฐานสองถูกนำมาใช้งานมากขึ้นจาก 1 หลักเป็น 2, 3 จนถึง 8 หลัก ทำให้เกิดตัวแปรใหม่ๆ ดังนี้

- (1) **บิต (bit)** หมายถึง หนึ่งหลักของเลขฐานสอง (binary digit) มีเลข 0 กับ 1 เท่านั้น
- (2) **ไบต์ (byte)** หมายถึง เลขฐานสองจำนวน 8 หลักหรือเท่ากับ 8 บิต ไบต์มีความสำคัญมากเพราะในระบบคอมพิวเตอร์จะประมวลข้อมูลเลขฐานสองครั้งละ 8 บิตหรือ 1 ไบต์เป็นอย่างน้อยเสมอ

(3) **LSB : Least Significant Bit หรือบิตนัยสำคัญต่ำสุด** หมายถึง บิตที่อยู่ในตำแหน่งขวาสุดของเลขฐานสอง มีค่านำหนักประจำหลักต่ำสุดคือ 2^0 ถ้าเป็น “1” ค่าของหลักสุดท้ายเท่ากับ $1 \times 2^0 = 1 \times 1 = 1$ แต่ถ้าบิตสุดท้ายนี้เป็น “0” ค่าของหลักสุดท้ายจะเท่ากับ $0 \times 2^0 = 0 \times 1 = 0$

(4) การกำหนดชื่อหลักของเลขฐานสอง บิตที่อยู่ขวามือสุดจะถูกเรียกว่า บิตศูนย์ (bit 0 : b_0) หรือบิต LSB บิตถัดมาเรียกว่า บิตหนึ่ง (bit 1 : b_1) ไล่ไปทางซ้ายจนครบ สามารถสรุปชื่อหลักของเลขฐานสองได้คือ b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 โดยตัวเลขแสดงตำแหน่ง 0-7 ต้องเขียนเป็นตัวห้อยเสมอ แต่เพื่อความสะดวกจึงขอเขียนในระดับเดียวกันเป็น b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0

(5) **MSB : Most Significant Bit** หรือบิตนัยสำคัญสูงสุด หมายถึงบิตที่อยู่ในตำแหน่งซ้ายมือสุดของเลขฐานสองจำนวนนั้นๆ หากเลขฐานสองมีจำนวน 8 บิต บิต MSB คือบิต 7 (bit 7 : b_7) มีค่าน้ำหนักประจำหลักเท่ากับ 2^7 หรือ 128 แต่ถ้าหากจำนวนบิตมีน้อยกว่านั้น เช่น 6 บิต, 5 บิต หรือ 4 บิต บิต MSB จะมีค่าน้ำหนักประจำหลักเปลี่ยนเป็น 2^5 , 2^4 และ 2^3 ตามลำดับ

8.1.3 ค่าน้ำหนักประจำหลัก

ในเลขฐานสิบจะมีค่าน้ำหนักประจำหลัก โดยคิดจากจำนวนสิบยกกำลัง โดยในหลักหน่วยมีค่าน้ำหนักประจำหลักเป็น 10^0 หรือ 1 หลักสิบมีค่าน้ำหนักประจำหลักเป็น 10^1 หรือ 10 ในหลักร้อยมีค่าน้ำหนักประจำหลักเป็น 10^2 หรือ 100 เป็นต้น

ในเลขฐานสองก็มีค่าน้ำหนักประจำหลักเช่นกัน แต่จะคิดจากจำนวนสองยกกำลัง โดยในหลักขวาสุดคือ บิต 0 หรือบิต LSB มีค่าน้ำหนักประจำหลักเป็น 2^0 หรือเท่ากับ 1 หลักถัดมาคือบิต 1 มีค่าน้ำหนักเป็น 2^1 หรือ 2 ถัดมาเป็นบิต 2 มีค่าน้ำหนักเป็น 2^2 หรือ 4 เมื่อพิจารณาที่เลขฐานสอง 8 บิต ค่าน้ำหนักประจำหลักสามารถสรุปได้ดังนี้

บิต	ค่าน้ำหนักประจำหลัก	เลขฐานสิบ
0	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128

จากค่าน้ำหนักประจำหลักจึงสามารถแปลงเลขฐานสองเป็นฐานสิบ หรือแปลงฐานสิบเป็นฐานสองได้

8.2 การแปลงเลขฐานสองและฐานสิบ

8.2.1 การแปลงเลขฐานสองเป็นฐานสิบ

เริ่มต้นด้วยการกำหนดค่าน้ำหนักประจำหลักของเลขฐานสองแต่ละหลัก แล้วคูณด้วยค่าของเลขฐานสองในหลักนั้นๆ สุดท้ายนำผลคูณทั้งหมดมารวมกัน ก็จะได้เป็นเลขฐานสิบที่ต้องการ

ตัวอย่างที่ 8-1

จงแปลงเลขฐานสอง 1011 เป็นฐานสิบ

(1) กำหนดค่าน้ำหนักประจำหลัก

หลัก	b_3	b_2	b_1	b_0
ค่าน้ำหนักประจำหลักคือ	2^3	2^2	2^1	2^0
เลขฐานสอง	1	0	1	1

(2) จากนั้นนำค่าน้ำหนักประจำหลักคูณกับค่าของเลขฐานสองประจำบิตนั้นแล้ว นำผลคูณของทุกหลักมารวมกัน

$$\begin{aligned}
 \text{เลขฐานสิบ} &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
 &= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) \\
 &= 8 + 0 + 2 + 1 \\
 &= 11
 \end{aligned}$$

8.2.2 การแปลงเลขฐานสิบเป็นเลขฐานสอง

จะใช้วิธีการหารเลขฐานสิบจำนวนนั้นด้วย 2 แล้วเก็บค่าของเศษที่ได้จากการหารเป็นเลขฐานสองในแต่ละหลัก โดยเศษที่ได้จากการหารครั้งแรกไม่ว่าจะเป็น “0” หรือ “1” จะเป็นหลักที่มีนัยสำคัญต่ำสุดหรือบิต LSB หรือบิต 0 (b_0) และเศษตัวสุดท้ายจะเป็นเลขฐานสองหลักที่มีนัยสำคัญสูงสุดหรือบิต MSB

ตัวอย่างที่ 8-2

จงแปลงเลขฐานสิบ 13 เป็นเลขฐานสอง

(1) หาร 13 ด้วย 2 ได้ 6 เศษ 1 เศษที่ได้จะเป็นบิตศูนย์ (b_0) หรือบิต LSB นั่นคือ บิต LSB = 1

(2) หาร 6 ด้วย 2 ได้ 3 เศษ 0 เศษที่ได้จะเป็นบิตหนึ่ง (b_1) ซึ่งก็คือ 0

(3) หาร 3 ด้วย 2 ได้ 1 เศษ 1 เศษที่ได้จะเป็นบิตสอง (b_2) ซึ่งก็คือ 1

(4) หาร 1 ด้วย 2 ได้ 0 เศษ 1 เศษที่ได้จะเป็นบิตสาม (b_3) และเป็นบิต MSB ซึ่งก็คือ 1

ดังนั้นจะได้เลขฐานสองเท่ากับ 1101

8.3 เครื่องหมายของเลขฐานสอง

ในเลขฐานสองสามารถที่จะมีทั้งค่าตัวเลขที่เป็นบวกและลบเช่นเดียวกับเลขฐานอื่นๆ โดยใช้บิต MSB เป็นตัวกำหนดเครื่องหมายของเลขฐานสอง ถ้ากำหนดบิต MSB เป็น “0” เลขจำนวนนั้นจะมีค่าเป็นบวก และหากกำหนดบิต MSB เป็น “1” เลขจำนวนนั้นจะมีค่าเป็นลบ

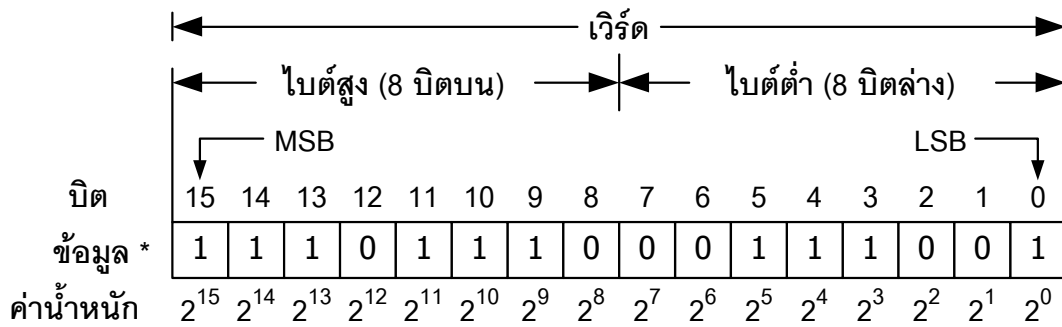
ข้อมูลต่อไปนี้จะเป็นการแสดงค่าของจำนวนเลขฐานสอง เมื่อคิดเครื่องหมายและไม่คิดเครื่องหมาย โดยได้ทำการแปลงเป็นเลขฐานสิบเปรียบเทียบให้เห็นความแตกต่างอย่างชัดเจน

เลขฐานสอง	เลขฐานสิบ	
	คิดเครื่องหมาย	ไม่คิดเครื่องหมาย
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

ในกรณีคิดเครื่องหมาย เมื่อนับถอยหลังจาก 0000 ก็จะเป็น 1111 นั่นคือเกิดการถอยหลังหนึ่งจำนวนหรือ -1 นับถอยหลังต่อไปจะเป็น 1110 ซึ่งก็คือ -2 เมื่อเป็นเช่นนี้การแปลงเลขฐานสองที่คิดเครื่องหมายเป็นฐานสิบจึงไม่สามารถใช้วิธีการแปลงแบบเดิมได้แต่ก็พอมีเทคนิคในการพิจารณา โดยใช้หลักเกณฑ์ค่าน้ำหนักประจำหลัก ยกตัวอย่าง เลขฐานสอง 1000 เลข 1 ที่อยู่หน้าสุด มีค่าน้ำหนักประจำหลักเท่ากับ 2^3 หรือ 8 จากการกำหนดว่า ถ้าคิดเครื่องหมาย เมื่อบิต MSB เป็น “1” จะต้องเป็นค่าลบ ดังนั้นจึงเป็น -8 ส่วนอีก 3 หลักที่เหลือจะเป็นเลขบวกจึงกลายเป็น $-8+0 = -8$ มาพิจารณาที่เลขฐานสอง 1101 บิตแรกเป็นลบเท่ากับ -8 ส่วน 3 บิตหลังเป็นบวกมีค่า +5 จึงได้ $-8+5 = -3$ เป็นต้น

8.4 ส่วนประกอบของข้อมูล

ข้อมูลที่ใช้ในการประมวลผลของไมโครคอนโทรลเลอร์นั้นสามารถกระทำได้ตั้งแต่ 1 บิตขึ้นไป สำหรับไมโครคอนโทรลเลอร์ ATmega644 ในชุด IPST-MicroBOX จะทำงานกับข้อมูล 1 ถึง 16 บิต โดยมีการกำหนดโครงสร้างส่วนประกอบของข้อมูลที่เป็นมาตรฐานเดียวกับไมโครคอนโทรลเลอร์ตัวอื่นทั่วโลกและเหมือนกับในคอมพิวเตอร์ด้วย ดังนี้



* สามารถเปลี่ยนแปลงได้

บิต-นิบิล-ไบต์-เวิร์ด เป็นชื่อหน่วยของข้อมูลที่ใช้ในการประมวลผลของไมโครคอนโทรลเลอร์

บิต (bit) เป็นขนาดของข้อมูลเลขฐานสองที่เล็กที่สุด เท่ากับ 1 หลักของเลขฐานสอง

นิบิล (nibble) มีขนาดเท่ากับ 4 บิต

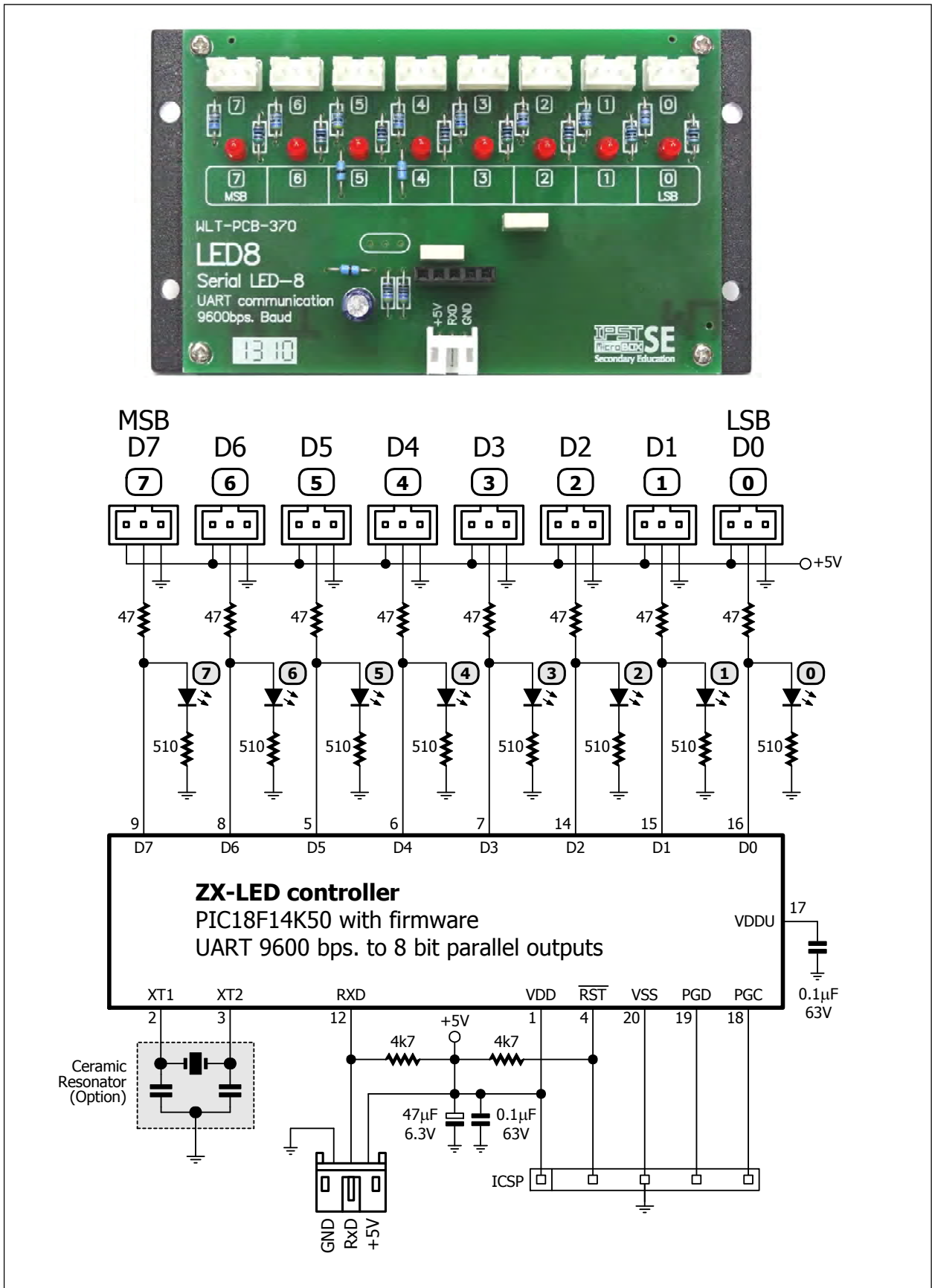
ไบต์ (byte) มีขนาดเท่ากับ 8 บิต

เวิร์ด (word) มีขนาดเท่ากับ 16 บิต หรือ 2 ไบต์

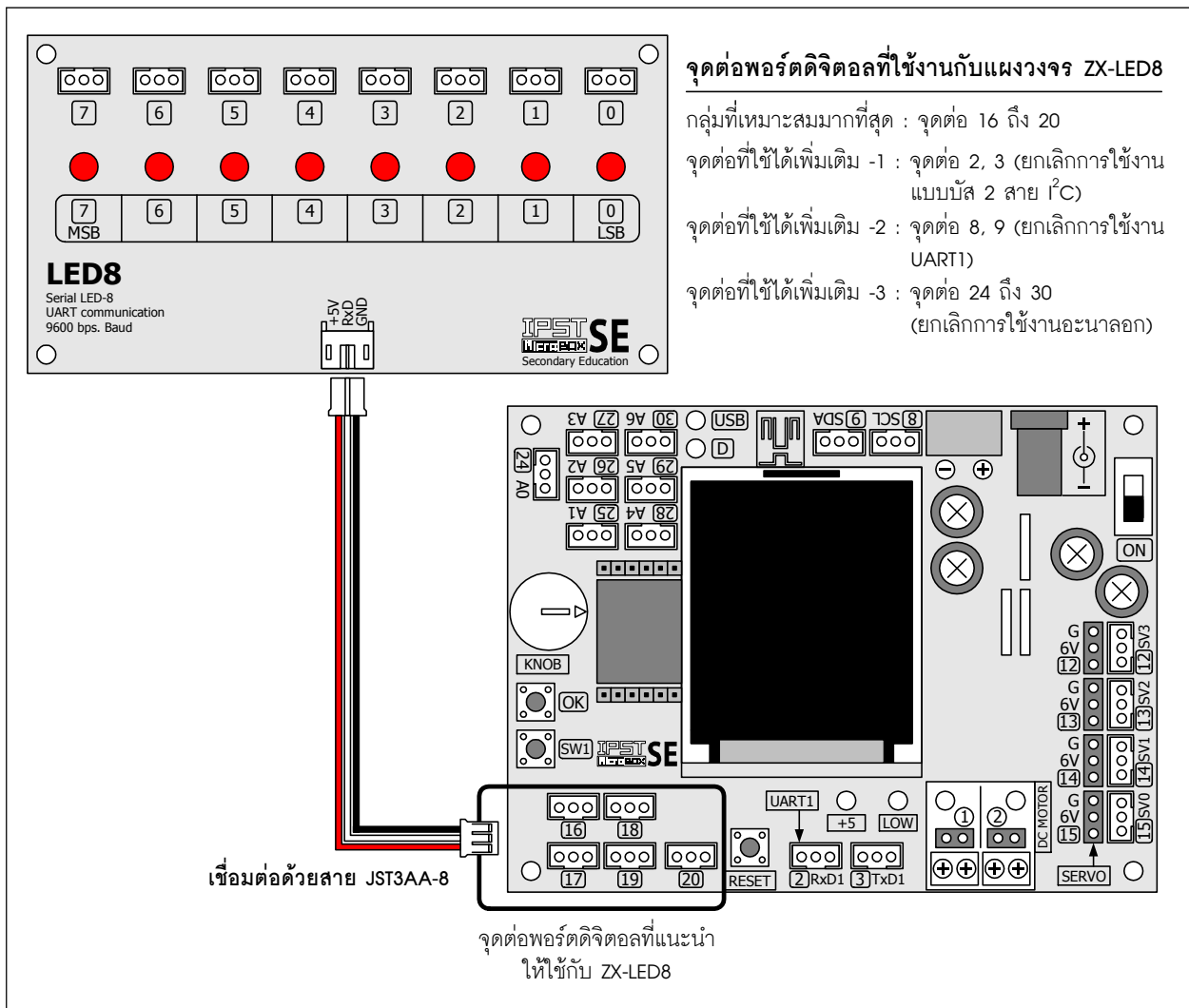
8.5 แผงวงจรไฟแสดงผล 8 ดวง : ZX-LED8

เป็นแผงวงจรที่มี LED ขนาด 3 มิลลิเมตรสำหรับแสดงผล 8 ดวง พร้อมจุดต่อพ่วงเอาต์พุตเพื่อนำไปใช้ในการขับรีเลย์ได้ด้วย โดยแผงวงจร ZX-LED8 นี้จะต่อเข้ากับขาพอร์ตใดของแผงวงจร IPST-SE ก็ได้ โดยใช้ขาพอร์ตเพียงขาเดียวในการควบคุมและขับ LED ให้ติดดับตามที่ต้องการได้พร้อมกันถึง 8 ดวง มีหน้าตาและวงจรทางไฟฟ้าของแผงวงจร ZX-LED8 แสดงในรูปที่ 8-1

ในแผงวงจร ZX-LED8 ใช้การติดต่อกับแผงวงจรหลัก IPST-SE ในแบบสื่อสารข้อมูลอนุกรมร่วมกับคำสั่งทางซอฟต์แวร์ ผู้พัฒนาโปรแกรมสามารถเขียนโปรแกรมให้ ZX-LED8 ติดดับได้ตั้งแต่ 1 ถึง 8 ตัว หรือจะเขียนโปรแกรมให้ทำงานเป็นไฟวิ่งได้ตั้งแต่ 1 ถึง 8 ดวงเช่นกัน



รูปที่ 8-1 ลักษณะและวงจรทางไฟฟ้าของแผงวงจรไฟแสดงผล ZX-LED8 ที่ใช้ในชุดกล่องสมองกล IPST-MicroBOX (SE)

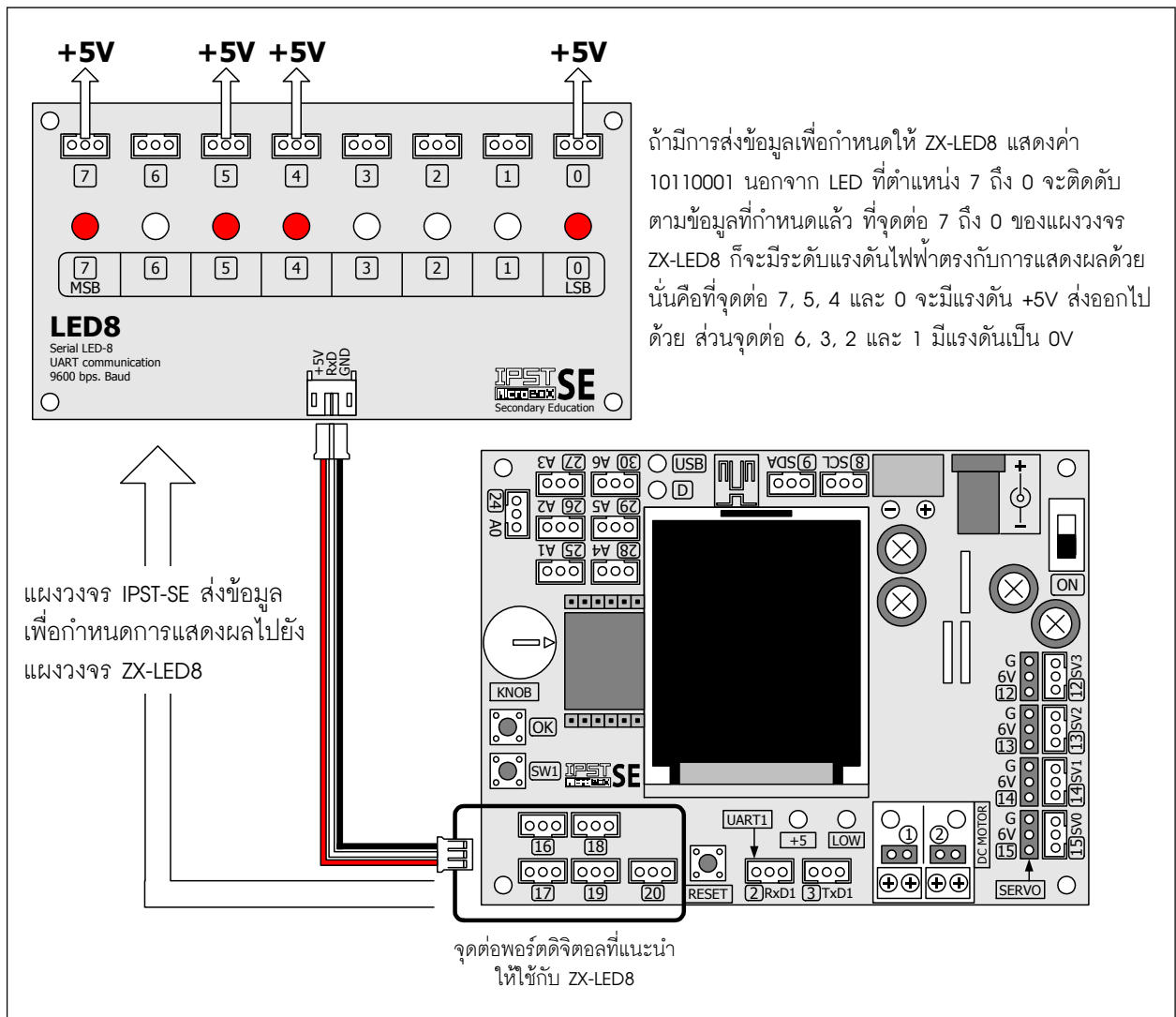


รูปที่ 8-2 แนวทางการต่อใช้งานแผงวงจรไฟแสดงผล ZX-LED8 กับแผงวงจรหลัก IPST-SE ในชุดกล่องสมองกล IPST-MicroBOX (SE)

ที่ด้านบนของแผงวงจร ZX-LED8 มีจุดต่อ JST ซึ่งต่อพ่วงมาจาก LED ทำงานที่ลอจิก “1” มีระดับแรงดันไฟตรงขาออกประมาณ +5V ดังแสดงในรูปที่ 8-3 จึงสามารถใช้สัญญาณเอาต์พุตจากจุดนี้ไปต่อกับวงจรขับโหลดกระแสไฟฟ้าสูง อาทิ แผงวงจรขับรีเลย์ ได้ทันที โดยไม่ต้องเขียนโปรแกรมควบคุมเพิ่มเติม

8.6 กลุ่มคำสั่งโปรแกรมภาษา C/C++ ของไลบรารี ipst.h ที่ใช้ควบคุม LED 8 ดวงบนแผงวงจร ZX-LED8

- pinLED8(); ใช้กำหนดจุดต่อพอร์ตของแผงวงจร IPST-SE ที่ต่อกับแผงวงจร ZX-LED8
- LED8 ใช้กำหนดค่าการแสดงผลของ LED 8 ดวงบนแผงวงจร ZX-LED8



รูปที่ 8-3 แสดงความสัมพันธ์ของการติดดับของ LED กับระดับแรงดันไฟฟ้าที่จุดต่อเอาต์พุต 0 ถึง 7 ของแผงวงจรไฟแสดงผล ZX-LED8

ปฏิบัติการที่ 2 ควบคุม LED บนแผงวงจร ZX-LED8

ปฏิบัติการที่ 2-1 ควบคุม LED ด้วยข้อมูลเลขฐานสอง


ในการทดลองนี้เป็นการเขียนโปรแกรมภาษา C เพื่อควบคุมให้ LED 8 ดวงบนแผงวงจร ZX-LED8 ติดหรือดับด้วยข้อมูลที่กำหนดไว้

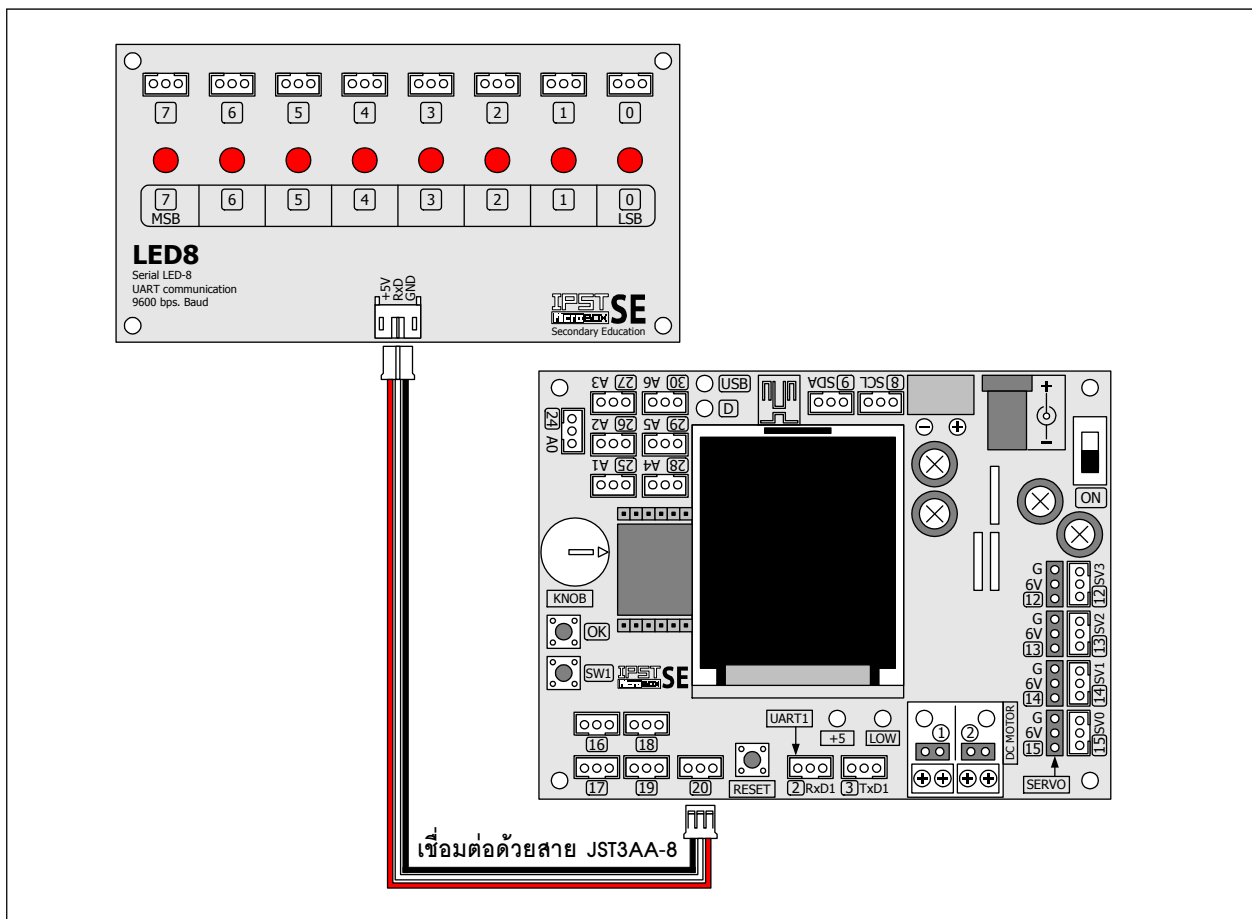
การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อสัญญาณจากจุดต่อ 20 ของแผงวงจรหลัก IPST-SE เข้ากับจุดต่อ RXD ของแผงวงจร ZX-LED8 ด้วยสายสัญญาณ JST3AA-8

ขั้นตอนการทดลอง

2.1.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L2-1 บันทึกในชื่อ microbox_LED8test.ino

2.1.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 



รูปที่ L2-1 การต่อวงจรเพื่อทำการทดลองสำหรับปฏิบัติการที่ 2 และ 3

```
#include <ipst.h>           // ผนวกไลบรารีหลัก
void setup()
{
  pinLED8(20);             // เชื่อมต่อ ZX-LED8 ผ่านทางจุดต่อพอร์ต 20
}

void loop()
{
  LED8(0b10000001);       // LED บิต 7 และ 0 ติดสว่าง ที่เหลือดับหมด
}
```

คำอธิบายโปรแกรม

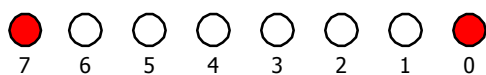
การทำงานของโปรแกรมนี้นี้ ข้อมูลเลขฐานสอง 10000001 ถูกส่งไปยังแผงวงจร ZX-LED8 ผ่านฟังก์ชัน LED8 โดยใช้จุดต่อพอร์ต 20 ซึ่งประกาศไว้ด้วยฟังก์ชัน pinLED8 ที่ส่วนของ setup()

ข้อมูล 8 บิตที่ส่งไปในรูปของเลขฐานสองสามารถเทียบเคียงได้กับตำแหน่งหลักของ LED ทั้ง 8 ดวงนั่นเอง โดยค่าของหลักข้อมูลใดเป็น “1” LED ประจำหลักนั้นจะติดสว่าง ในทางกลับกันถ้ามีค่าเป็น “0” LED ก็จะไม่ติด ดังนั้นถ้าทำการแก้ไขโปรแกรมเพื่อส่งค่าใหม่จะได้ผลลัพธ์ที่แตกต่างกันไป

โปรแกรมที่ L2-1 : ไฟล์ microbox_LED8test.ino โปรแกรมภาษา C สำหรับทดลองควบคุม LED 8 ดวง

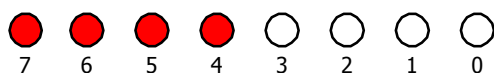
2.1.3 สังเกตผลการทำงานที่แผงวงจร ZX-LED8

LED ที่หลัก 0 และ 7 ของแผงวงจร ZX-LED8 ติดสว่าง จากผลของข้อมูล 10000001 ที่กำหนดในโปรแกรม



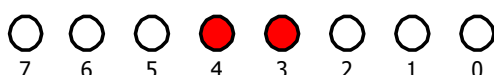
2.1.4 ทดลองแก้ไขข้อมูลแสดงผลเป็น LED8(0b11110000); ทำการคอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE

ได้ผลการทำงานดังนี้



2.1.5 ทดลองแก้ไขข้อมูลแสดงผลเป็น led8(0b00011000); ทำการคอมไพล์ไฟล์โปรแกรมโปรเจกต์ใหม่ แล้วดาวน์โหลดโปรแกรมไปยังแผงวงจร IPST-SE

ได้ผลการทำงานดังนี้



```
#include <ipst.h> // ผนวกไลบรารีหลัก
void setup()
{}
void loop()
{
  unsigned char i=0;
  while(1)
  {
    LED8(20,i++); // กำหนดให้ใช้จุดต่อพอร์ต 20 ในการควบคุม ZX-LED8
                  // โดยแสดงผลด้วยค่าของ i ที่เปลี่ยนแปลงจาก 00000000 ถึง 11111111
    delay(300);
  }
}
```

คำอธิบายโปรแกรม

การทำงานของโปรแกรมนี้อาศัยข้อมูลเลขฐานสองที่กำหนดโดยตัวแปร i ถูกส่งออกไปยังแผงวงจร ZX-LED8 ผ่านฟังก์ชัน LED8 ที่เขียนในแบบใหม่ ให้เหลือเพียงบรรทัดเดียว โดยกำหนดพอร์ตที่ใช้ในการติดต่อแล้วต่อด้วยข้อมูลที่ต้องการแสดงผล ทำให้โปรแกรมกระชับขึ้น

โดยข้อมูล 8 บิตที่ส่งไปในรูปของเลขฐานสองนั้นมีค่า 00000000 ถึง 11111111 ตามชนิดของตัวแปร i ที่กำหนดให้เป็นแบบ char ซึ่งมีขนาด 8 บิต

โปรแกรมที่ L2-2 : ไฟล์ microbox_LED8binary.ino โปรแกรมภาษา C/C++ สำหรับทดลองควบคุม LED 8 ดวง ให้แสดงผลตามค่าของเลขฐานสองขนาด 8 บิต จาก 00000000 ถึง 11111111

2.1.6 เขียนโปรแกรมที่ L2-2 บันทึกไฟล์เป็น microbox_LED8binary.ino จากนั้นทำการคอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

2.1.6 สังเกตผลการทำงานที่แผงวงจร ZX-LED8

LED ทั้ง 8 ดวงแสดงเป็นรหัสเลขฐานสอง 8 บิต เริ่มจาก 00000000 ถึง 11111111 แล้วกลับมาเริ่มต้นที่ 00000000 ใหม่อีกครั้ง โดยมีอัตราการเปลี่ยนข้อมูลทุกๆ 0.3 วินาที

ปฏิบัติการที่ 2-2 ไฟกะพริบ LED 8 ดวง


ในการทดลองนี้เป็นการเขียนโปรแกรมควบคุมให้ LED 8 ดวงบนแผงวงจร ZX-LED8 ติดหรือดับด้วยข้อมูลที่กำหนด โดยในการทดลองนี้จะส่งข้อมูล 11111111 ไปแสดงผลในรูปแบบไฟกะพริบ

การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อสัญญาณจากจุดต่อ 20 ของแผงวงจรหลัก IPST-SE เข้ากับจุดต่อ RXD ของแผงวงจร ZX-LED8 ด้วยสายสัญญาณ JST3AA-8

ขั้นตอนการทดลอง

2.2.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L2-3 บันทึกในชื่อ microbox_LED8blink.ino

2.2.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

2.2.3 รันโปรแกรม สังเกตการทำงานของ LED 8 ดวงบนแผงวงจร ZX-LED8

LED ทั้ง 8 ดวงของแผงวงจร ZX-LED8 ติดกะพริบในทุกๆ 1 วินาที โดยประมาณ

```
#include <ipst.h>           // ผนวกไลบรารีหลัก
void setup()
{
}
void loop()
{
  LED8(20,0b11111111);     // กำหนดให้ LED ทั้ง 8 ดวงติดหมด
  sleep(500);              // หน่วงเวลา 0.5 วินาที
  LED8(20,0b00000000);     // กำหนดให้ LED ทั้ง 8 ดวงดับหมด
  sleep(500);              // หน่วงเวลา 0.5 วินาที
}
```

คำอธิบายโปรแกรม

การทำงานของโปรแกรมนี มีข้อมูลเลขฐานสอง 2 ชุดที่ส่งออกไปยังแผงวงจร ZX-LED8 ด้วยฟังก์ชัน LED8 ที่เขียนในรูปแบบกำหนดพอร์ตที่ใช้ในการติดต่อแล้วต่อด้วยข้อมูลที่ต้องการแสดงผล โดยชุดแรกคือ 11111111 ทำให้ LED 8 ดวงติดสว่างทั้งหมด ส่วนชุดที่ 2 คือ 00000000 ทำให้ LED ทั้งหมดดับ

อัตราการติดดับหรือกะพริบถูกกำหนดโดยฟังก์ชัน sleep() ซึ่งมีการทำงานเหมือนกับ delay() โดยในโปรแกรมนี้กำหนดให้เท่ากับ 500 มิลลิวินาทีหรือ 0.5 วินาที

โปรแกรมที่ L2-3 : ไฟล์ microbox_LED8blink.ino โปรแกรมภาษา C/C++ สำหรับควบคุม LED กะพริบพร้อมกัน 8 ดวง

ปฏิบัติการที่ 3 ควบคุมการแสดงผล LED บนแผงวงจร ZX-LED8 แบบพิเศษ

ปฏิบัติการที่ 3-1 ไฟวิ่ง LED 8 ดวง

ในการทดลองนี้เป็นการเขียนโปรแกรมภาษา C เพื่อควบคุมให้ LED 8 ดวงบนแผงวงจร ZX-LED8 ทำงานในลักษณะไฟวิ่งทางซ้ายและขวา

การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อสัญญาณจากจุดต่อ 20 ของแผงวงจรหลัก IPST-SE เข้ากับจุดต่อ RXD ของแผงวงจร ZX-LED8 ด้วยสายสัญญาณ JST3AA-8 (รูปที่ L2-1 ในปฏิบัติการที่ 2 ของบทที่ 8)

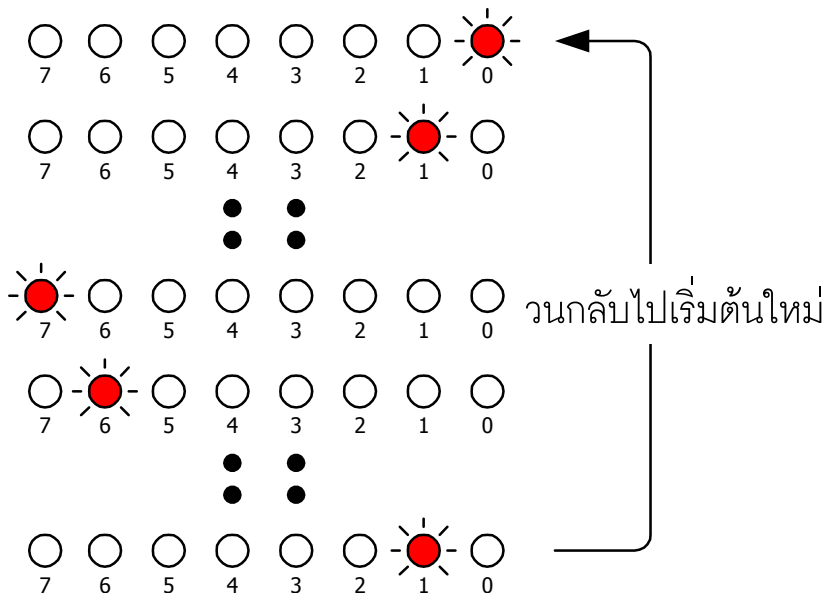
ขั้นตอนการทดลอง

3.1.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L3-1 บันทึกในชื่อ microbox_LED8running.ino

3.1.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

3.1.3 รันโปรแกรม สังเกตการทำงานของ LED 8 ดวงบนแผงวงจร DSP-4

LED จะติดครั้งละ 1 ดวงจากตำแหน่งที่อยู่ทางขวาสุด (บิต 0) ไถ่มาตามลำดับในลักษณะไฟวิ่งจากขวาไปซ้ายจนสุดทางขวา (บิต 7) จากนั้นจะไถ่กลับทิศทางเป็นติดจากดวงซ้ายสุด (บิต 7) กลับมายังดวงขวาสุด (บิต 0) ในลักษณะไฟวิ่งจากซ้ายมาขวา วนทำงานเช่นนี้ไปตลอด




```
#include <ipst.h>          // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
  // แสดงข้อความของการทดลอง
  glcdClear();
  glcd(0,0,"Example: LED8_02");
  glcd(2,0,"connect LED8 to (20)");
  glcd(4,0,"LED running LEFT");
  glcd(5,0,"          and RIGHT");

  pinLED8(20);           // ใช้จุดต่อพอร์ต 20 ในการเชื่อมต่อ
}

void loop()
{
  int i;
  for (i=0;i<8;i++)      // กำหนดให้เลื่อนข้อมูลไปทางซ้าย
  {
    LED8(0x01<<i);      // เลื่อนบิตไปทางซ้ายแล้วส่งไปแสดงที่ ZX-LED8 โดยมีค่าตั้งต้นที่ 00000001
    delay(300);         // หน่วงเวลา 0.3 วินาที
  }
  for (i=6;i>0;i--)      // กำหนดให้เลื่อนข้อมูลไปทางขวา
  {
    LED8(0x01<<i);      // เลื่อนบิตไปทางขวาแล้วส่งไปแสดงที่ ZX-LED8
    delay(300);         // หน่วงเวลา 0.3 วินาที
  }
}
}
```

คำอธิบายโปรแกรม

โปรแกรมนีทำงานโดยอาศัยการส่งข้อมูลผ่านฟังก์ชัน LED8 เพื่อให้ LED ติดครั้งละ 1 ดวงนานประมาณ 0.3 วินาที โดยไล่ลำดับจากบิต 0 ไปยังบิต 7 และวนกลับจากบิต 7 กลับมายังบิต 0 เป็นเช่นนี้อย่างต่อเนื่อง

การทำงานของ LED ในลักษณะไฟวิ่งนี้ของโปรแกรมนี้เกิดจากการเลื่อนบิตข้อมูลทั้งทางซ้ายและขวา โดยหลังจากเลื่อนข้อมูล 1 ครั้งก็จะส่งไปแสดงผลนาน 0.3 วินาที จากนั้นจึงเลื่อนข้อมูลในครั้งต่อไป

ด้านอัตราเร็วในการแสดงผลของ LED จะขึ้นอยู่กับค่าหน่วงเวลาของฟังก์ชัน delay (หรืออาจใช้ sleep ก็ได้) ถ้ากำหนดให้มีค่าน้อยกว่า 300 ก็จะทำให้ LED แสดงผลการเปลี่ยนแปลงได้เร็วมากขึ้น นั่นคือ ได้เห็นไฟวิ่งเร็วขึ้นนั่นเอง

โปรแกรมที่ L3-1 : ไฟล์ microbox_LED8running.ino โปรแกรมภาษา C/C++ สำหรับทดลองควบคุมไฟวิ่ง LED 8 ดวง

ปฏิบัติการที่ 3-2 ขับ LED 8 ดวง ด้วยรูปแบบข้อมูลที่กำหนดล่วงหน้า

ในการทดลองนี้เป็นการเขียนโปรแกรมภาษา C เพื่อควบคุมให้ LED 8 ดวงบนแผงวงจร ZX-LED8 ทำงานในลักษณะไฟวิ่งตามรูปแบบข้อมูลที่กำหนดไว้ล่วงหน้า โดยใช้ตัวแปรแบบอะเรย์ในการเก็บตารางรูปแบบข้อมูล

การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อสัญญาณจากจุดต่อ 20 ของแผงวงจรหลัก IPST-SE เข้ากับจุดต่อ RXD ของแผงวงจร ZX-LED8 ด้วยสายสัญญาณ JST3AA-8 (รูปที่ L2-1 ในปฏิบัติการที่ 2 ของบทที่ 8)

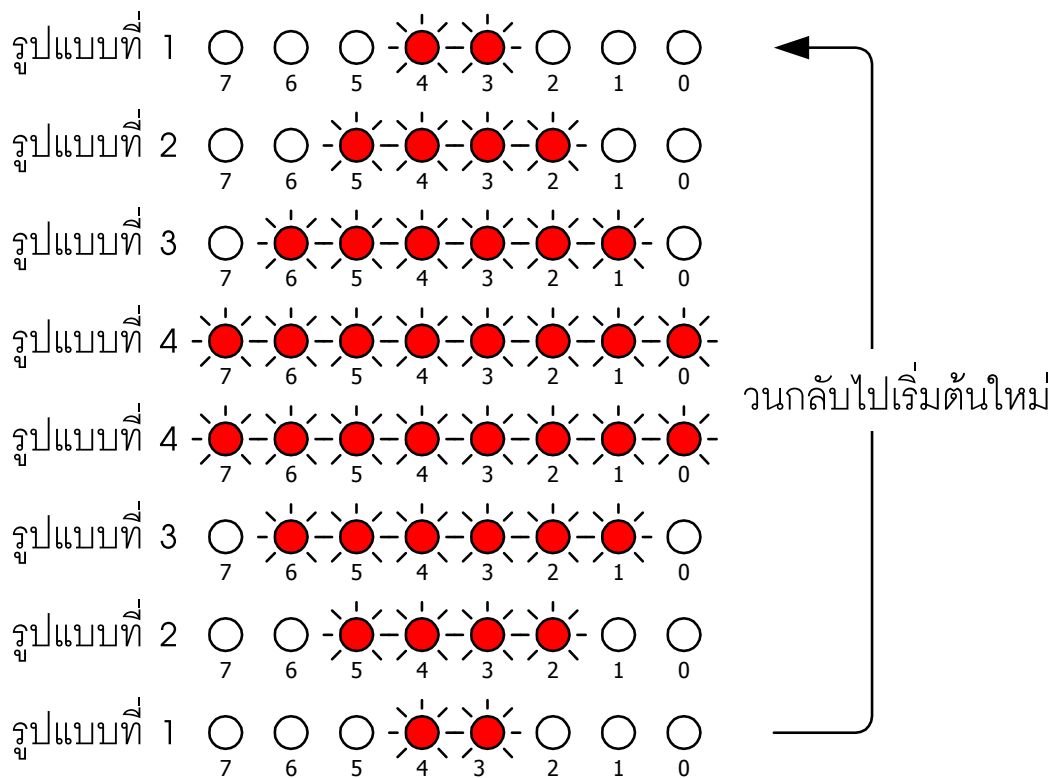
ขั้นตอนการทดลอง

3.2.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L3-2 บันทึกในชื่อ microbox_LED8pattern.ino

3.2.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

3.2.3 รันโปรแกรม สังเกตการทำงานของ LED 8 ดวงบนแผงวงจร ZX-LED8

LED จะติดตามข้อมูลที่กำหนด ซึ่งมีด้วยกัน 4 แบบ โดยเริ่มจากแบบที่ 1 ไปถึง 4 แล้วทำงานย้อนกลับจากแบบที่ 4 มายังแบบที่ 1 วนทำงานเช่นนี้ไปตลอด



```
#include <ipst.h>           // ผนวกไลบรารีหลัก

int pattern[]={             // กำหนดรูปแบบข้อมูลแสดงผล
  0b00011000,              // รูปแบบข้อมูล #1
  0b00111100,              // รูปแบบข้อมูล #2
  0b01111110,              // รูปแบบข้อมูล #3
  0b11111111};             // รูปแบบข้อมูล #4

void setup()
{
  // แสดงข้อความของการทดลอง
  glcdClear();
  glcd(0,0,"Example: LED8_pattern");
  glcd(2,0,"connect LED8 to (20)");
  glcd(4,0,"Pattern LED demo");

  pinLED8(20);              // ใช้จุดต่อพอร์ต 20 ในการเชื่อมต่อ
}

void loop()
{
  int i;
  for (i=0;i<4;i++)         // เปลี่ยนรูปแบบข้อมูลแสดงผล
  {
    LED8(pattern[i]);       // แสดงผลตามรูปแบบข้อมูลจาก 1 ถึง 4
    delay(300);
  }
  for (i=2;i>0;i--)         // เปลี่ยนรูปแบบข้อมูลแสดงผล
  {
    LED8(pattern[i]);       // แสดงผลตามรูปแบบข้อมูลจาก 4 ถึง 1
    delay(300);
  }
}
```

คำอธิบายโปรแกรม

ข้อมูลสำหรับการแสดงผลของโปรแกรมจะแตกต่างกันไปจากตัวอย่างที่ผ่านๆ มา ในโปรแกรมมีการกำหนดตารางรูปแบบข้อมูลไว้ล่วงหน้าเก็บไว้ในตัวแปร pattern ที่เป็นตัวแปรแบบอะเรย์ 4 ชุด จากนั้นโปรแกรมจะใช้ตัวแปร i เป็นตัวชี้ข้อมูลของ pattern ออกมาแสดงผลโดยผ่านฟังก์ชัน LED8 ข้อมูลแต่ละชุดจะได้รับการแสดงผลนานประมาณ 0.3 วินาที

โปรแกรมที่ L3-2 : ไฟล์ microbox_LED8pattern.ino โปรแกรมภาษา C/C++ สำหรับควบคุมไฟรั้ง LED 8 ดวง ในแบบกำหนดรูปแบบการแสดงผลด้วยตัวแปรชนิดอะเรย์

ปฏิบัติการที่ 3-3 ขับ LED 8 ดวง แสดงผลในแบบกราฟแท่ง

ในการทดลองนี้เป็นการเขียนโปรแกรมภาษา C เพื่อควบคุมให้ LED 8 ดวงบนแผงวงจร ZX-LED8 ทำงานในลักษณะไฟวิ่งตามรูปแบบข้อมูลที่กำหนดไว้ล่วงหน้า โดยใช้ตัวแปรแบบอะเรย์ในการเก็บตารางรูปแบบข้อมูล

การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อสัญญาณจากจุดต่อ 20 ของแผงวงจรหลัก IPST-SE เข้ากับจุดต่อ RXD ของแผงวงจร ZX-LED8 ด้วยสายสัญญาณ JST3AA-8 (รูปที่ L2-1 ในปฏิบัติการที่ 2 ของบทที่ 8)

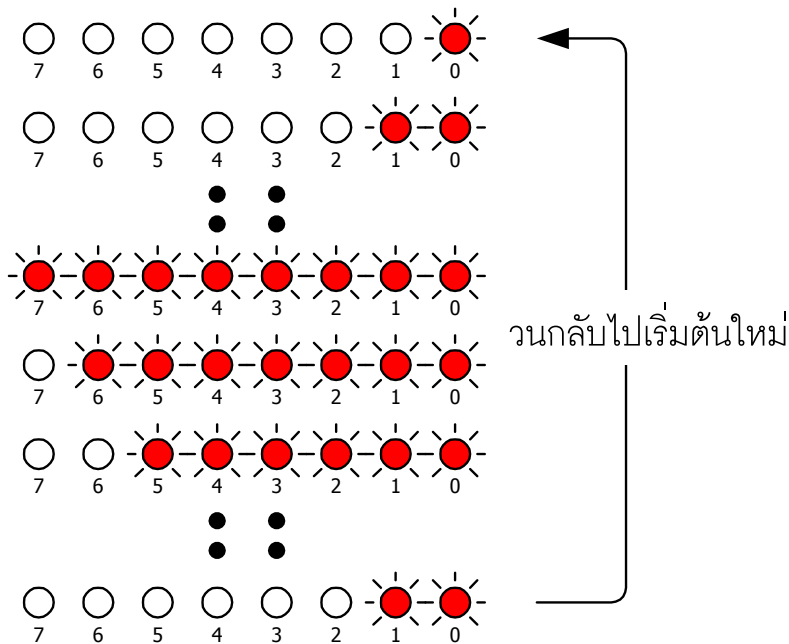
ขั้นตอนการทดลอง

3.3.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L3-3 บันทึกในชื่อ microbox_LED8bargraph.ino

3.3.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

3.3.3 รันโปรแกรม สังเกตการทำงานของ LED 8 ดวงบนแผงวงจร ZX-LED8

LED จะเริ่มติดจากดวงซ้ายสุด (บิต 0) ตามด้วยบิต 1 โดยที่ LED ของบิต 0 ยังติดอยู่ จากนั้นจะทยอยติดสว่างเพิ่มขึ้นทีละดวง จนครบ 8 ดวง จากนั้นจะเริ่มดับลงทีละดวงจาก LED บิต 7 ไล่ตามลำดับจนถึงบิต 1 จากนั้นวนกลับ LED บิต 1 ติดสว่าง ตามด้วยบิต 2 วนทำงานเช่นนี้ไปตลอด ทำให้ดูแล้วคล้ายกราฟแท่งที่มีการเพิ่มค่าและลดค่าตลอดเวลา



ต่อยอด

ผู้พัฒนาโปรแกรมสามารถนำเทคนิคนี้ไปใช้ในการเขียนโปรแกรมเพื่อแสดงค่าของสัญญาณไฟฟ้าที่ได้จากตัวตรวจจับแบบอะนาล็อก โดยเมื่อค่าของสัญญาณไฟฟ้ามีค่าต่ำ LED จะติดดวงเดียว และคิดเพิ่มขึ้นเมื่อระดับสัญญาณไฟฟ้ามีค่าสูงขึ้น

```
#include <ipst.h>           // มนวกไลบรารีหลัก
void setup()
{
  // แสดงข้อความของการทดลอง
  glcdClear();
  glcd(0,0,"Example: LED8_bar");
  glcd(2,0,"connect LED8 to (20)");
  glcd(4,0,"Solid bargraph demo");

  pinLED8(20);             // ใช้จุดต่อพอร์ต 20 ในการเชื่อมต่อ
}

void loop()
{
  int i,n;
  n=0;
  for (i=0;i<8;i++)
  {
    n1=(0x01<<i);          // สร้างข้อมูลแสดงผลด้วยการออร์บิต
    LED8(n);
    delay(300);
  }

  for (i=7;i>0;i--)
  {
    n&=0xff^(0x01<<i);     // สร้างข้อมูลแสดงผลด้วยการแอนด์บิต
    LED8(n);
    delay(300);
  }
}
```

คำอธิบายโปรแกรม

ข้อมูลสำหรับการแสดงผลของโปรแกรมนี้อาจใช้ตัวดำเนินการตรรกะหรือลอจิกในระดับบิตทั้งการออร์ (&&) และการแอนด์ (&) โดยการออร์บิตในโปรแกรมทำให้บิตข้อมูลที่เป็น "1" อยู่แต่เดิมไม่เปลี่ยน จึงนำมาใช้ในการเพิ่มจำนวนบิตที่ทำให้ LED ติดสว่าง ส่วนการแอนด์บิตในโปรแกรมนี้อาจทำให้บิตที่มีนัยสำคัญสูงเปลี่ยนเป็น "0" ที่ละบิต ทำให้จำนวนของ LED ที่ติดสว่างลดลง

โปรแกรมที่ L3-3 : ไฟล์ microbox_LED8bargraph.ino โปรแกรมภาษา C/C++ สำหรับทดลองควบคุมไฟรั้ง LED 8 ดวง ในแบบกราฟแท่ง

บทที่ 9

ติดต่อกับสวิตช์ เพื่ออ่านค่าและนำไปใช้งาน

ในบทนี้เป็นการนำแผงวงจรสวิตช์เข้ามาต่อทำงานร่วมกับไมโครคอนโทรลเลอร์ของแผงวงจร IPST-SE อันเป็นการเรียนรู้การอ่านค่าจากอินพุตมาประมวลผลเพื่อส่งสัญญาณออกไปควบคุมอุปกรณ์ทางเอาต์พุตอย่างง่ายนั่นเอง

ปฏิบัติการทั้งหมดที่นำเสนอในบทนี้ผู้พัฒนาสามารถนำความรู้ไปใช้สร้างระบบควบคุมอย่างง่ายที่มีการตรวจจับอินพุตจากการกดสวิตช์ แล้วกำหนดให้โปรแกรมที่ออกแบบนั้นมีการตอบสนองอะไรบ้างอย่างออกมา เช่น เมื่อมีการกดสวิตช์ ระบบตอบสนองด้วยการเปิด/ปิดไฟ, ควบคุมมอเตอร์ หรือส่งข้อมูลอุณหภูมิไปแสดงผลยังคอมพิวเตอร์ที่เชื่อมต่ออยู่ เป็นต้น

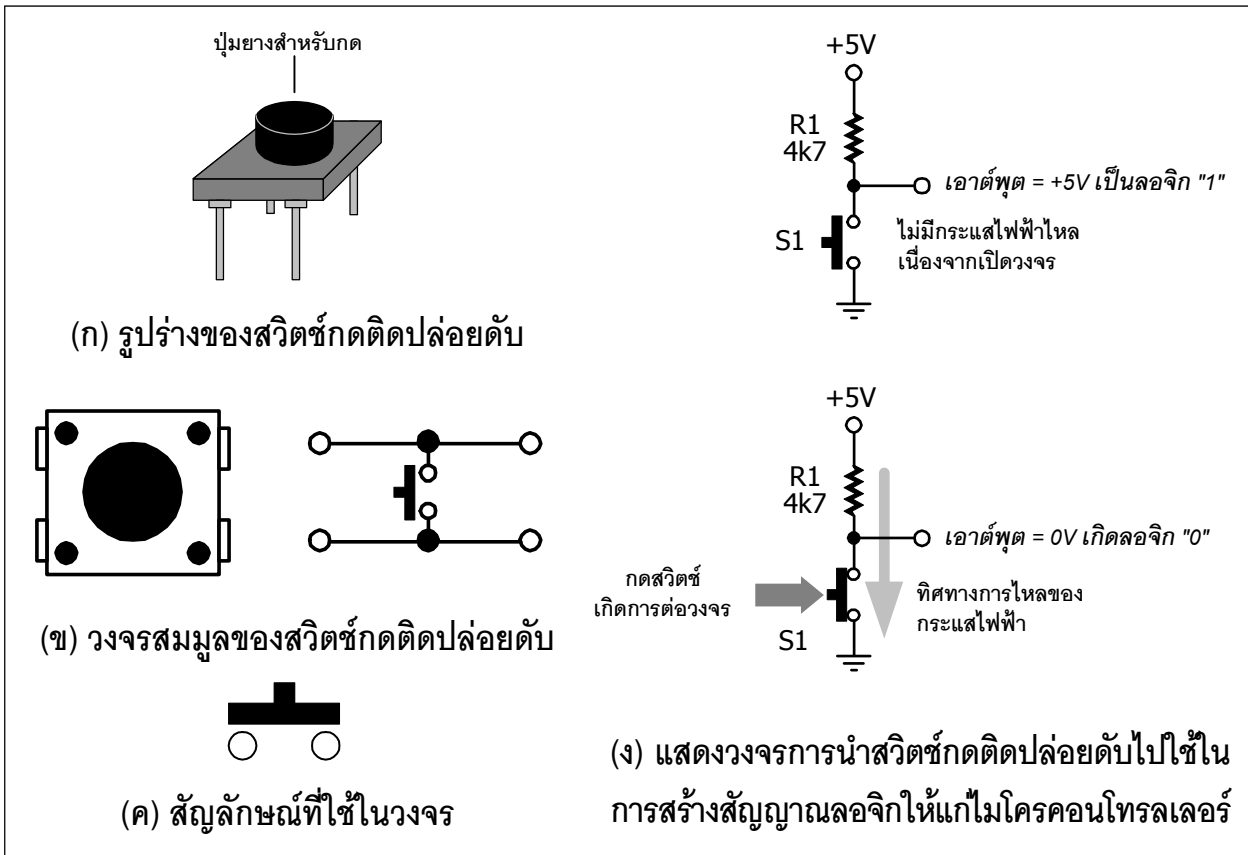
9.1 ความรู้เบื้องต้นเกี่ยวกับการทำงานของสวิตช์

สวิตช์เป็นอุปกรณ์พื้นฐานที่มีบทบาทและใช้ประโยชน์อย่างมากในวงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งในการใช้งานเป็นอุปกรณ์ตัดต่อหรืออุปกรณ์สำหรับสร้างสัญญาณอินพุตให้แก่วงจร โดยเฉพาะอย่างยิ่งกับวงจรดิจิทัล

9.1.1 สวิตช์กดติดปล่อยดับ (Push-button switch/Tact switch)

เป็นอุปกรณ์สำหรับตัดต่อวงจรไฟฟ้าที่นิยมใช้อย่างมากในระบบไมโครคอนโทรลเลอร์ โดยสวิตช์แบบนี้ปกติเมื่อไม่มีการกด หน้าสัมผัสของสวิตช์จะแยกออกจากกันหรือเรียกว่า เปิดวงจร เมื่อมีการกดลงบนปุ่มด้านบนซึ่งทำมาจากยางสังเคราะห์หรือพลาสติก ทำให้หน้าสัมผัสตัวนำภายในสวิตช์ต่อกัน กระแสไฟฟ้าก็จะสามารถไหลผ่านไปได้

รูปร่างของสวิตช์กดติดปล่อยดับมีด้วยกันหลายแบบ ทั้งแบบบัดกรีต่อสาย แบบลงแผ่นวงจรพิมพ์ แบบติดหน้าปัด และในบางแบบมีไฟแสดงในตัว ส่วนขาต่อใช้งานมีตั้งแต่ 2 ขาขึ้นไป ในการใช้งานกับไมโครคอนโทรลเลอร์นั้นมักใช้สวิตช์เป็นอุปกรณ์สร้างสัญญาณอินพุตแบบหนึ่ง โดยต่อปลายข้างหนึ่งกับตัวต้านทาน และปลายของตัวต้านทานนั้นต่อกับไฟเลี้ยงหรือที่เรียกว่า “พูลอัพ” ที่จุด

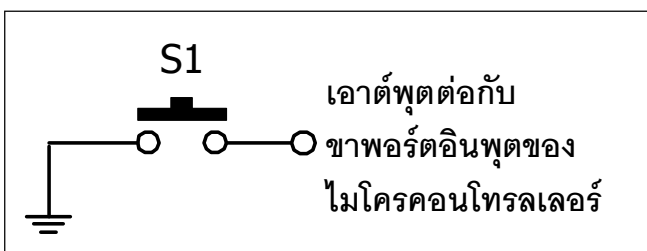


รูปที่ 9-1 แสดงรูปร่างและสัญลักษณ์ของสวิตช์แบบกดติดปล่อยดับที่ใช้ในการทดลอง

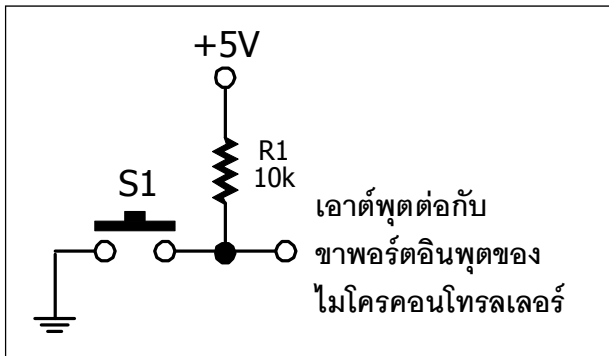
ต่อวงจรระหว่างสวิตช์กับตัวต้านทานจะเป็นจุดที่ต่อเข้ากับไมโครคอนโทรลเลอร์เพื่ออ่านค่าลอจิกของการกดสวิตช์ดังแสดงในรูปที่ 2-1 (ง) ถ้าหากไม่มีการกดสวิตช์ สถานะลอจิกของสวิตช์ตัวนั้นจะเป็น “1” อันเนื่องจากการต่อตัวต้านทาน पुलอัป และเมื่อมีการกดสวิตช์ จะเกิดสถานะลอจิกเป็น “0” เนื่องจากจุดต่อสัญญาณนั้นถูกต่อลงกราวด์

9.1.2 การต่อสวิตช์เพื่อกำหนดลอจิกทางอินพุต

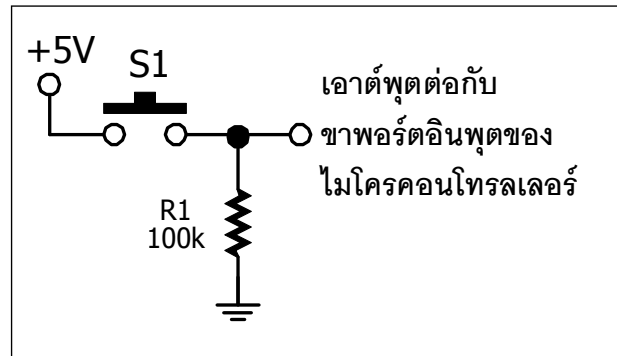
ในรูปที่ 9-2 เป็นการต่อสวิตช์กดติดปล่อยดับเพื่อสร้างสัญญาณอินพุตลอจิก “0” ให้แก่วงจร โดยเมื่อกดสวิตช์ S1 จะเป็นการต่อขาอินพุตลงกราวด์ อย่างไรก็ตาม การต่อสวิตช์เพียงตัวเดียวอาจทำให้สถานะลอจิกทางอินพุตในขณะที่ไม่มีการกดสวิตช์มีความไม่แน่นอน จึงควรต่อตัวต้านทานเข้ากับไฟเลี้ยงหรือลงกราวด์เพื่อกำหนดสถานะทางลอจิกในขณะที่ไม่มีการกดสวิตช์ให้แก่วงจร



รูปที่ 9-2 การเชื่อมต่อสวิตช์กดติดปล่อยดับเข้ากับอินพุตของไมโครคอนโทรลเลอร์อย่างง่าย



รูปที่ 9-3 การเชื่อมต่อสวิทช์กดติดปล่อยดับเข้ากับอินพุตของไมโครคอนโทรลเลอร์แบบมีตัวต้านทานต่อพูลอัป ทำให้สถานะลอจิกเมื่อไม่มีการกดสวิทช์เป็นลอจิก "1"



รูปที่ 9-4 การเชื่อมต่อสวิทช์กดติดปล่อยดับเข้ากับอินพุตของไมโครคอนโทรลเลอร์แบบมีตัวต้านทานต่อพูลดาวน์ ทำให้สถานะลอจิกเมื่อไม่กดสวิทช์เป็นลอจิก "0"

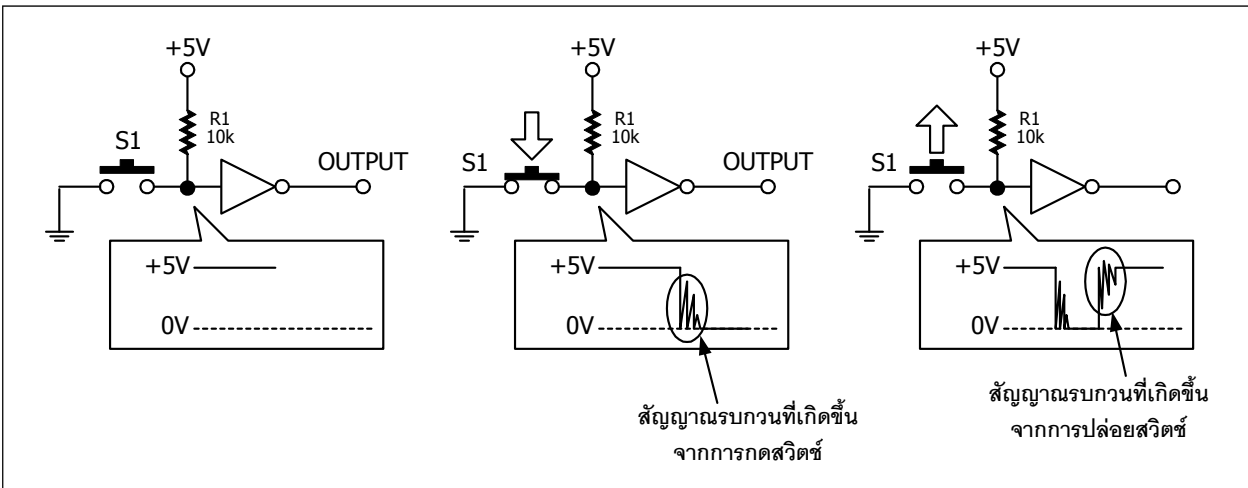
ในรูปที่ 9-3 เป็นการต่อสวิทช์เข้ากับอินพุตของไมโครคอนโทรลเลอร์ โดยมีการต่อตัวต้านทานเข้าที่ขาอินพุตของวงจรและไฟเลี้ยง +5V เรียกการต่อตัวต้านทานแบบนี้ว่า *การต่อตัวต้านทานพูลอัป (pull-up)* ด้วยการต่อตัวต้านทานในลักษณะนี้ทำให้เมื่อยังไม่มีการกดสวิทช์ สถานะลอจิกที่อินพุตต้องเป็นลอจิก "1" อย่างแน่นอน

ส่วนในรูปที่ 9-4 เป็นการต่อสวิทช์เข้ากับอินพุตในอีกลักษณะหนึ่งที่มีการต่อตัวต้านทานเข้าที่ขาอินพุตของวงจรและกราวด์ จะเรียกการต่อตัวต้านทานแบบนี้ว่า *การต่อตัวต้านทานพูลดาวน์ (pull-down)* ด้วยการต่อตัวต้านทานในลักษณะนี้ทำให้เมื่อยังไม่มีการกดสวิทช์ สถานะลอจิกที่ขาอินพุตเป็นลอจิก "0"

9.1.3 สัญญาณรบกวนของการกดสวิทช์

ในทางทฤษฎี เมื่อสวิทช์มีการเปิดปิดวงจร สัญญาณไฟฟ้าจะถูกปลดหรือต่อเข้าไปในวงจรและสามารถวิเคราะห์ผลการทำงานได้ แต่ในความเป็นจริง เมื่อมีการกดและปล่อยสวิทช์ หน้าสัมผัสของสวิทช์จะเกิดการสั่นและกว่าที่จะต่อหรือเปิดวงจรอย่างสมบูรณ์นั้นจะต้องใช้เวลาชั่วขณะหนึ่ง ซึ่งมนุษย์ไม่สามารถมองเห็นภาวะนั้นได้ แต่วงจรอิเล็กทรอนิกส์จะตรวจจับความไม่คงที่นั้นได้ และมองเป็นสัญญาณรบกวน

สัญญาณรบกวนที่เกิดขึ้นนั้นเรียกว่า *สัญญาณรบกวนจากการสั่นของหน้าสัมผัสสวิทช์ หรือ การเบ้า (bounce)* ถ้าหากนำสวิทช์นั้นไปใช้เป็นอุปกรณ์เพื่อป้อนสัญญาณอินพุตให้แก่วงจรนับ ในทางอุดมคติเมื่อกดสวิทช์ 1 ครั้ง จะได้สัญญาณพัลส์เพียง 1 ลูกเพื่อส่งไปยังวงจรเพื่อเปลี่ยนค่าหนึ่งค่า แต่ทางปฏิบัติ จะมีสัญญาณพัลส์ส่งออกไปอาจจะเพียง 1 ลูกตามต้องการหรือมากกว่านั้นก็ได้ ซึ่งไม่อาจคาดเดาได้ โดยสัญญาณพัลส์ที่เกิดขึ้นนั้นจะเกิดขึ้นเมื่อสวิทช์เริ่มต่อวงจร และเมื่อสวิทช์กำลังเปิดวงจรเนื่องจากการปล่อยสวิทช์ ในรูปที่ 9-5 แสดงปรากฏการณ์ดังกล่าว

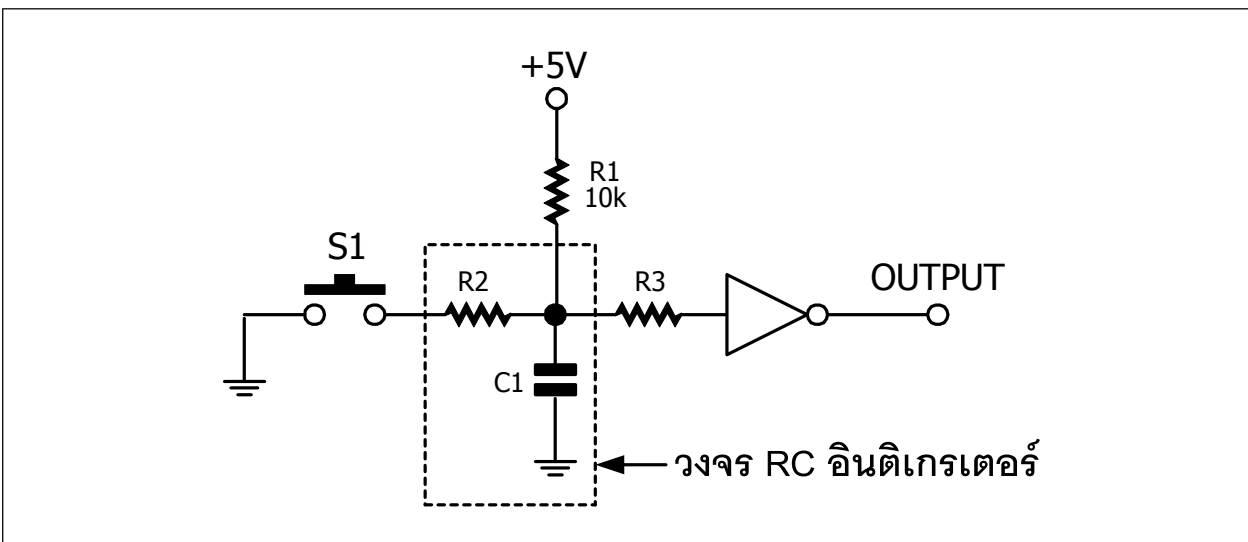


รูปที่ 9-5 แสดงการเกิดสัญญาณรบกวนเมื่อมีการกดและปล่อยสวิตช์ในวงจรดิจิทัล

9.1.4 การแก้ไขสัญญาณรบกวนของการกดสวิตช์เมื่อใช้งานกับไมโครคอนโทรลเลอร์

หลักการแก้ไขสัญญาณรบกวนแบบนี้คือ หน่วงเวลาการเกิดขึ้นของสัญญาณพัลส์เล็กน้อย เพื่อให้วงจรไม่สนใจสัญญาณที่เกิดขึ้นในช่วงเริ่มต้นกดสวิตช์เรียกการแก้ไขสัญญาณรบกวนนี้ว่า ดีเบายซ์ (debounce)

วิธีการแรกทำได้โดยใช้อุปกรณ์อิเล็กทรอนิกส์พื้นฐานอย่างตัวต้านทานและตัวเก็บประจุ โดยต่อกันในลักษณะวงจร RC อินทิเกรเตอร์ ดังในรูปที่ 9-6 ด้วยวิธีการนี้จะช่วยลดผลของสัญญาณรบกวนที่เกิดขึ้นจากการกดสวิตช์ได้ในระดับหนึ่ง โดยประสิทธิภาพของวงจรจะขึ้นกับการเลือกค่าของตัวต้านทานและตัวเก็บประจุ หากเลือกค่าของตัวเก็บประจุน้อยเกินไป อาจไม่สามารถลดสัญญาณรบกวนได้ แต่ถ้าเลือกค่ามากเกินไป จะทำให้ความไวในการตรวจจับการกดสวิตช์ลดลง นั่นคือ อาจต้องกดสวิตช์มากกว่า 1 ครั้งเพื่อให้ได้สัญญาณที่ต้องการ



รูปที่ 9-6 การต่อวงจร RC อินทิเกรเตอร์เพื่อแก้ไขปัญหาสัญญาณรบกวนจากการกดสวิตช์

วิธีการที่สองคือ *ใช้กระบวนการทางซอฟต์แวร์เข้ามาช่วย* ซึ่งมีขั้นตอนโดยสรุปดังนี้

- (1) อ่านค่าการกดสวิตช์ครั้งแรกเข้ามาก่อน
- (2) หน่วงเวลาประมาณ 0.1 ถึง 1 วินาที
- (3) อ่านค่าของการกดสวิตช์อีกครั้ง

ถ้าหากค่าที่อ่านได้เหมือนกับการอ่านครั้งแรก

แสดงว่า มีการกดสวิตช์เกิดขึ้นจริง

ถ้าค่าที่อ่านได้ไม่เหมือนกับการอ่านครั้งแรก

แสดงว่า สัญญาณที่เกิดขึ้นอาจเป็นเพียงพัลส์แคบๆ อาจตีความได้ว่า เป็นสัญญาณรบกวน จึงยังไม่มีอาการกดสวิตช์เกิดขึ้นจริง

9.2 ฟังก์ชันโปรแกรมภาษา C/C++ ในไลบรารี ipst.h ที่ใช้ในการทดลอง

สำหรับคำสั่งหรือฟังก์ชันของโปรแกรมภาษา C/C++ ที่ใช้ในการทดลองทั้งหมดในบทนี้ ได้รับการบรรจุไว้ในไฟล์ไลบรารี ipst.h คือ คำสั่ง in, sw_OK และ sw1

9.2.1 in

เป็นฟังก์ชันอ่านค่าสถานะลอจิกของพอร์ตที่กำหนด

รูปแบบ

char in(x)

พารามิเตอร์

x - กำหนดขาพอร์ตที่ต้องการอ่านค่า มีค่าตั้งแต่ 0 ถึง 50 สำหรับ IPST-SE ใช้ได้ถึง 30

หมายเหตุ : ไม่แนะนำให้ใช้ฟังก์ชันนี้กับจุดต่อ 19 และ 20 บนแผงวงจร IPST-SE

การคืนค่า

เป็น 0 หรือ 1

ตัวอย่างที่ 9-1

```
char x;           // ประกาศตัวแปร x เพื่อเก็บค่าผลลัพธ์จากการอ่านค่าระดับสัญญาณ
x = in(16);       // อ่านค่าดิจิตอลจากพอร์ตหมายเลข 16 แล้วเก็บค่าไว้ที่ตัวแปร x
```

9.2.2 sw_OK()

เป็นฟังก์ชันตรวจสอบสถานะสวิตช์ OK บนแผงวงจร IPST-SE โดยให้สถานะ “เป็นจริง” เมื่อมีการกดสวิตช์และ “เป็นเท็จ” เมื่อไม่มีการกดสวิตช์

รูปแบบ

```
unsigned char sw_ok()
```

การคืนค่า

1 (เป็นจริง) เมื่อมีการกดสวิตช์
0 (เป็นเท็จ) เมื่อไม่มีการกดสวิตช์

ตัวอย่างที่ 9-2

```
#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
  glcdClear();
}
void loop()
{
  if (sw_OK()) // ตรวจสอบการกดสวิตช์ OK
  {
    glcdFillScreen(GLCD_YELLOW); // เปลี่ยนสีพื้นเป็นสีเหลือง
    delay(3000); // แสดงสีพื้นใหม่นาน 3 วินาที
  }
  glcdClear(); // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
}

```

9.2.3 sw_OK_press()

เป็นฟังก์ชันตรวจสอบการกดสวิตช์ OK บนแผงวงจร IPST-SE ต้องรอจนกระทั่งสวิตช์ถูกปล่อยหลังจากการกดสวิตช์จึงจะผ่านฟังก์ชันนี้ไปกระทำคำสั่งอื่นๆ

ตัวอย่างที่ 9-3

```
.....
sw_OK_press(); // รอจนกระทั่งเกิดกดสวิตช์ OK
.....
```

9.2.4 sw1_press

เป็นฟังก์ชันตรวจสอบการกดสวิตช์ SW1 บนแผงวงจร IPST-SE ต้องรอจนกระทั่ง SW1 ถูกปล่อยหลังจากมีการกดสวิตช์ จึงจะผ่านขั้นตอนการทำงานของฟังก์ชันนี้ไป

รูปแบบ

```
void sw1_press()
```

ตัวอย่างที่ 9-4

```
.....
sw1_press();      // รอจนกระทั่งสวิตช์ SW1 ถูกกดและปล่อย
.....
```

9.2.5 sw1

เป็นฟังก์ชันตรวจสอบการกดสวิตช์ SW1 ในขณะใดๆ

รูปแบบ

```
char sw1()
```

การคืนค่า

เป็น “0” เมื่อสวิตช์ SW1 ถูกกด และ เป็น “1” เมื่อไม่มีการกดสวิตช์ SW1

ตัวอย่างที่ 9-5

```
char x;           // ประกาศตัวแปร x เพื่อเก็บค่าผลลัพธ์จากการอ่านค่าดิจิตอล
x = sw1();       // อ่านค่าสถานะของสวิตช์ SW1 มาเก็บไว้ที่ตัวแปร x
```

ปฏิบัติการที่ 4 ควบคุม LED ด้วยสวิตช์

ปฏิบัติการที่ 4-1 ควบคุม LED ด้วยการกดสวิตช์ OK


การเชื่อมต่อทางฮาร์ดแวร์

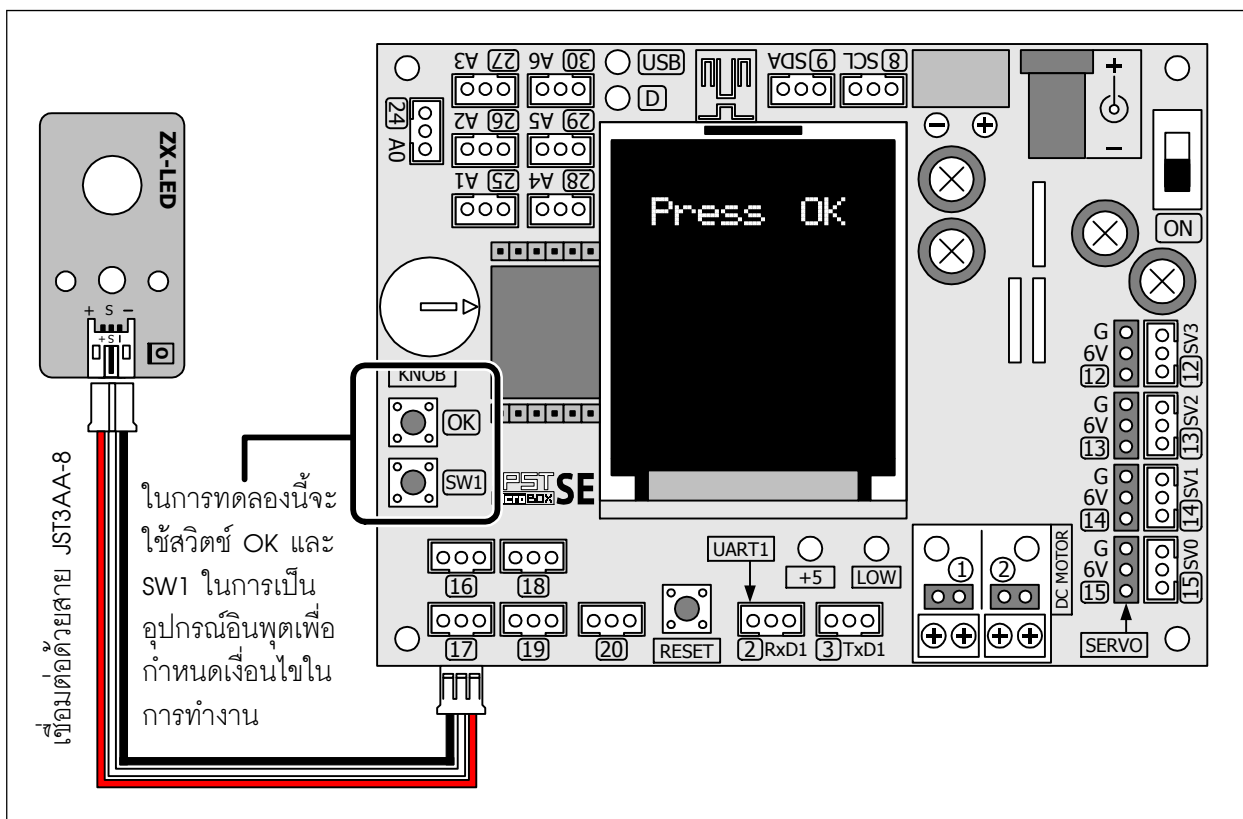
- ต่อแผงวงจร ZX-LED กับจุดต่อพอร์ต 17 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

4.1.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L1-1 บันทึกในชื่อ microbox_OKtest.ino

4.1.2 เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์

4.1.3 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 



รูปที่ L4-1 การต่อวงจรเพื่อทำการทดลองสำหรับปฏิบัติการที่ 4

```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
void setup()
{
  setTextSize(2);          // กำหนดขนาดตัวอักษร 2 เท่า
  glcd(1,1,"Press OK");    // แสดงข้อความออกหน้าจอแสดงผล
  sw_OK_press();           // วนรอจนกระทั่งกดสวิตช์ OK
  glcdClear();             // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
}
void loop()
{
  if (sw_OK())             // ตรวจสอบการกดสวิตช์ OK
  {
    out(17,0);             // ดับ LED ที่จุดต่อพอร์ต 17
    delay(2000);           // นาน 2 วินาที
  }
  out(17,1);              // ขยับ LED ที่จุดต่อพอร์ต 17 ให้ติดสว่าง
}
}
```

คำอธิบายโปรแกรม

การทำงานของโปรแกรมเริ่มต้นด้วยการแสดงข้อความแจ้งให้กดสวิตช์ OK บนแผงวงจร IPST-SE เมื่อ กดแล้ว จะเข้าสู่ลูการทำงานหลักในฟังก์ชัน loop() เพื่อตรวจสอบต่อไปว่า มีการกดสวิตช์ OK หรือไม่ ถ้าไม่มี จะส่งข้อมูล “1” ออกไปยังจุดต่อพอร์ต 17 ทำให้ LED ที่ต่ออยู่ติดสว่าง และยังคงสว่างอยู่เช่นนั้นจนกว่าจะมีการกดสวิตช์ OK

เมื่อสวิตช์ OK ถูกกด จะทำให้เงื่อนไขการตรวจสอบเป็นจริง เกิดการตอบสนองด้วยการส่งข้อมูล “0” ไปที่จุดต่อพอร์ต 17 ทำให้ LED ที่ต่ออยู่ดับลงเป็นเวลา 2 วินาที ตามการทำงานของฟังก์ชัน delay จากนั้นก็จะหลุดออกจากการตรวจสอบ มาพบฟังก์ชัน out เพื่อขยับให้ LED กลับมาติดอีกครั้ง แล้ววนรอการกดสวิตช์ OK ในรอบใหม่

โปรแกรมที่ L4-1 : ไฟล์ microbox_OKtest.ino โปรแกรมภาษา C/C++ สำหรับทดลองควบคุม LED ด้วยสวิตช์

4.1.4 รันโปรแกรม

ที่จอแสดงผลกราฟิก LCD จะแจ้งให้กดสวิตช์ OK ทันทีที่กดสวิตช์ OK บนแผงวงจร IPST-SE จะทำให้ LED ของแผงวงจร ZX-LED ที่ต่อกับจุดต่อพอร์ต 17 ติดสว่าง

4.15 กดสวิตช์ OK อีก 1 ครั้ง แล้วปล่อย สังเกตการทำงานของ LED

LED ของแผงวงจร ZX-LED จะดับลงนาน 2 วินาที จากนั้นจะกลับมาติดสว่างใหม่อีกครั้ง และจะทำงานในลักษณะนี้ไปตลอด จนกว่าจะมีการรีเซ็ตระบบ หรือปิดเปิดจ่ายไฟเลี้ยงใหม่

ปฏิบัติการที่ 4-2 ควบคุม LED ด้วยการกดสวิตช์ SW1

ในการทดลองนี้จะเพิ่มเติมสวิตช์เข้ามาอีก 1 ตัวคือ SW1 โดยสวิตช์ SW1 ถูกติดตั้งไว้พร้อมใช้งานบนแผงวงจร IPST-SE อยู่แล้ว โดยในปฏิบัติการนี้ต้องการนำสวิตช์ SW1 มาควบคุมการเปิดปิด LED ในแบบที่อกเกิด (toggle) นั่นคือ เมื่อกดสวิตช์หนึ่งครั้ง LED ติด และเมื่อกดซ้ำ LED จะดับ สลับกันเช่นนี้

การเชื่อมต่อทางฮาร์ดแวร์

- ต่อแผงวงจร ZX-LED กับจุดต่อพอร์ต 17 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

4.2.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L1-1 บันทึกในชื่อ microbox_SW1test.ino

4.2.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

4.2.3 รันโปรแกรม

ที่จอแสดงผลกราฟิก LCD จะแจ้งให้กดสวิตช์ OK

4.2.4 กดสวิตช์ OK อีก 1 ครั้ง แล้วปล่อย

4.2.5 ทดลองกดสวิตช์ SW1 แล้วปล่อย 3 ครั้ง สังเกตการทำงานของ ZX-LED

LED ของแผงวงจร ZX-LED จะติดและดับ กลับสถานะกันในทุกครั้งที่มีการกดสวิตช์ SW1 นั่นคือ จากเดิมดับจะกลายเป็นติดสว่าง และจากติดสว่างจะกลายเป็นดับ

4.2.6 กดสวิตช์ SW1 ค้างไว้ครู่หนึ่ง แล้วจึงปล่อย สังเกตการทำงานของ LED

เมื่อสวิตช์ยังถูกกดค้าง สถานะของ LED จะไม่มีการเปลี่ยนแปลง จนกว่าสวิตช์จะถูกปล่อย


```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
int i=0;
void setup()
{
  setTextSize(2);          // กำหนดขนาดตัวอักษร 2 เท่า
  glcd(1,1,"Press OK");    // แสดงข้อความออกหน้าจอแสดงผล
  sw_OK_press();           // วนรอกจนกระทั่งกดสวิตช์ OK
  glcdClear();             // เคลียร์หน้าจอแสดงผล กำหนดพื้นหลังเป็นสีดำ
}
void loop()
{
  if (sw1())               // ตรวจสอบการกดสวิตช์ OK
  {
    out(17,i^=1);          // ดับ LED ที่จุดต่อพอร์ต 17
    while(sw1())           //
      delay(5);
  }
}
```

คำอธิบายโปรแกรม

โปรแกรมจะมีตัวอย่างการรับค่าสวิตช์ 2 ตัวคือ สวิตช์ OK (ใช้ฟังก์ชัน sw_OK_press()) และสวิตช์ SW1 (ใช้ฟังก์ชัน sw1()) โดยตัวแรกจะอยู่ในฟังก์ชัน setup เพื่อวนรอการกดสวิตช์ OK เพื่อเริ่มต้นการทำงาน

หลังจากมีการกดสวิตช์ OK จะเข้าสู่รูปแบบการทำงานของฟังก์ชัน loop เพื่อวนรอการกดสวิตช์ SW1 เมื่อสวิตช์ SW1 ถูกกด จะทำการส่งสัญญาณออกมายังพอร์ต 17 โดยค่าที่ส่งออกมาจะได้มาจากการเอ็กคลูซีฟออร์ค่าของตัวแปร i กับ 1 จากคำสั่ง $i^=1$ ทำให้ค่าที่ได้มีการกลับสถานะทุกครั้งที่มีการเรียกให้ทำงาน ซึ่งการทำงานในลักษณะนี้สามารถใช้คำสั่ง $i=~i$ ก็ได้ แต่เนื่องจากในคอมพิวเตอร์บางเครื่อง (โดยเฉพาะเครื่องคอมพิวเตอร์ Macintosh) ไม่มีคีย์ ~ ให้ใช้ จึงต้องเขียนคำสั่งด้วยการทำเอ็กคลูซีฟ-ออร์แทน

ดังนั้นเมื่อมีการกดสวิตช์ SW1 ทุกครั้ง ก็จะมีการกลับสถานะลอจิกเดิมที่จุดต่อพอร์ต 17 ทำให้ LED ของแผงวงจร ZX-LED ที่ต่ออยู่เกิดการติดและดับสลับกันในทุกครั้งที่กดสวิตช์

สำหรับคำสั่ง while(sw1()) และ dealy(5); ทำหน้าที่ลดสัญญาณรบกวนที่เกิดจากการกดสวิตช์ ทำให้การกดสวิตช์ SW1 ในแต่ละครั้งมีความแน่นอนมากขึ้น

โปรแกรมที่ L4-2 : ไฟล์ microbox_SW1test.ino โปรแกรมภาษา C/C++ สำหรับทดลองควบคุม LED ด้วยสวิตช์ SW1 ในลักษณะกลับสถานะการทำงานในทุกครั้งที่มีการกดสวิตช์ SW1

ปฏิบัติการที่ 5 สวิตช์นับจำนวน

การเชื่อมต่อทางฮาร์ดแวร์

- ต่อแผงวงจร ZX-SWITCH01 กับจุดต่อพอร์ต 16 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

5.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L1-1 บันทึกในชื่อ microbox_CounterSwitch.ino

5.2 เปิดสวิตช์จ่ายไฟแก่แผงวงจร IPST-SE แล้ว เชื่อมต่อสาย USB เข้ากับคอมพิวเตอร์

5.3 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

5.4 รันโปรแกรม

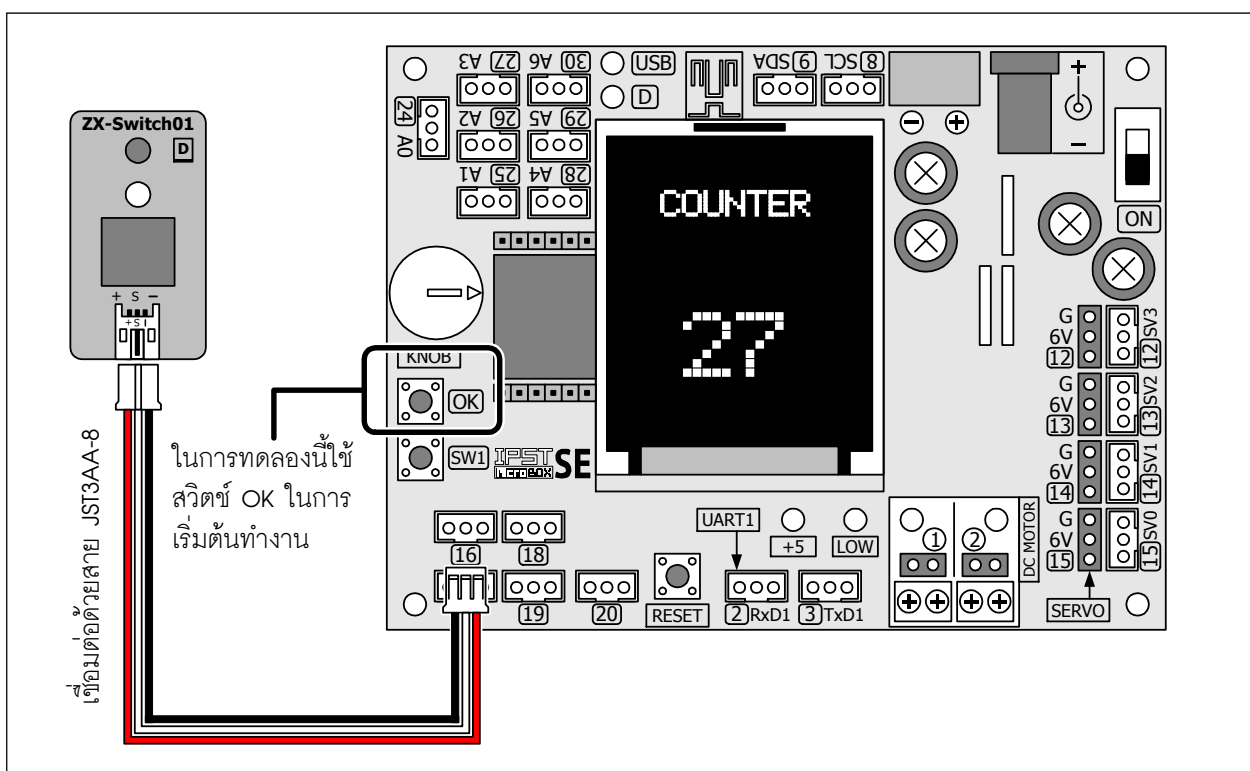
ที่จอแสดงผลกราฟิก LCD แสดงข้อความ Start ให้กดสวิตช์ OK

5.5 กดสวิตช์ OK 1 ครั้ง แล้วปล่อย

ที่จอแสดงผลกราฟิก LCD แสดงข้อความ COUNTER ให้เริ่มการนับด้วยการกดสวิตช์ที่ต่อกับพอร์ต 16

5.6 กดสวิตช์ที่แผงวงจร ZX-SWITCH01 ที่ต่อกับพอร์ต 16 สังเกตการทำงานที่จอแสดงผลของแผงวงจร IPST-SE

ที่จอแสดงผลจะเริ่มต้นแสดงค่าการนับจำนวนการกดสวิตช์ โดยเริ่มจาก 0 ค่าจะเปลี่ยนทันทีที่มีการกดสวิตช์ที่พอร์ต 16 และจะทำการรับค่าต่อไปได้ ก็ต่อเมื่อมีการปล่อยสวิตช์ แล้วกดใหม่ หากสวิตช์ยังถูกกดค้างค่าการนับจะไม่มีเปลี่ยนแปลง จนกว่าสวิตช์จะถูกปล่อย และกดเข้ามาใหม่



รูปที่ L5-1 การต่อวงจรเพื่อทำการทดลองสำหรับปฏิบัติการที่ 5

```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
int i=0;                   // ประกาศตัวแปรเก็บค่าการนับ
void setup()
{
  setTextSize(2);          // กำหนดขนาดตัวอักษร 2 เท่า
  lcd(1,3,"Start");        // แสดงข้อความ Start ออกหน้าจอแสดงผล
  sw_OK_press();          // วนรอนจนกระทั่งกดสวิตช์ OK
  lcdClear();              // เคลียร์หน้าจอแสดงผล กำหนดพื้นที่หลังเป็นสีดำ
  lcd(1,2,"COUNTER");      // แสดงข้อความ COUNTER เพื่อแจ้งชื่อการทดลอง
  setTextSize(3);          // กำหนดขนาดตัวอักษร 3 เท่า
  lcd(3,3,"0");            // กำหนดค่าเริ่มต้นเป็น 0
}
void loop()
{
  if (in(16)==0)           // ตรวจสอบการกดสวิตช์ที่พอร์ต 16
  {
    i++;                   // เพิ่มค่าตัวนับ
    lcd(3,3,"%d",i);       // แสดงค่าการนับ
    while(in(16)==0)       // ตรวจสอบการปล่อยสวิตช์
      delay(5);
  }
}
```

คำอธิบายโปรแกรม

โปรแกรมนี้ใช้ฟังก์ชัน in ในการตรวจจับและอ่านค่าจากการกดสวิตช์ที่พอร์ต 16 โดยตรงสอบว่าที่พอร์ต 16 เป็นลอจิก “0” หรือไม่ ถ้าใช่แสดงว่า มีการกดสวิตช์เปิดขึ้น จากนั้นจะทำการเพิ่มค่าตัวนับ แล้วนำมาแสดงผลที่จอแสดงผลกราฟิก LCD ของแผงวงจร IPST-SE

คำสั่ง while(in(16)==0) และ delay(5); ทำหน้าที่ลดสัญญาณรบกวนที่เกิดจากการกดสวิตช์ โดยจะตรวจสอบว่า มีการปล่อยสวิตช์แล้วหรือไม่ ถ้าไม่ ก็จะวนทำงานอยู่ที่คำสั่งนั้น ช่วยให้ไม่เกิดการนับค่าโดยไม่ตั้งใจขึ้น ดังนั้นการกดสวิตช์ที่พอร์ต 16 ในแต่ละครั้งจึงมีความแน่นอนสูง

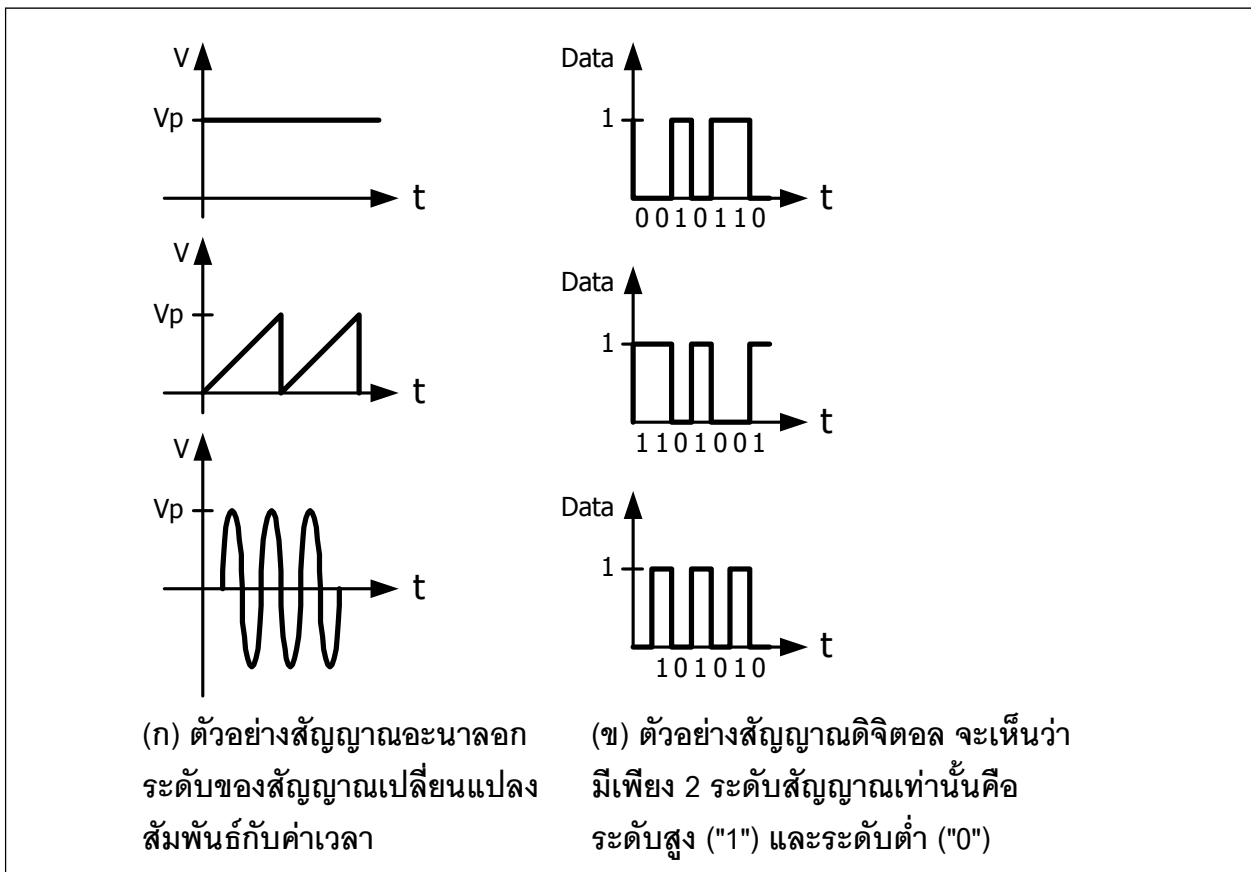
โปรแกรมที่ L5-1 : ไฟล์ microbox_CounterSwitch.ino โปรแกรมภาษา C/C++ สำหรับทดลองใช้งานสวิตช์ ในการสร้างเครื่องนับจำนวนอย่างง่าย

จากทั้งสองปฏิบัติการผู้พัฒนาสามารถนำไปประยุกต์ใช้งานสวิตช์หลายๆ ตัวพร้อมกันและมีความสามารถที่แตกต่างกันไปแล้วแต่จุดประสงค์ เช่น สวิตช์บางตัวสามารถกำหนดให้เมื่อกดสวิตช์ค้างแล้วสามารถเพิ่มค่าหรือลดค่าข้อมูลที่กำหนดได้ ในขณะที่สวิตช์บางตัวอาจกำหนดให้ไม่สามารถกดค้างได้ เป็นต้น

บทที่ 10

การอ่านค่าสัญญาณอะนาลอกอย่างง่าย

สัญญาณทางไฟฟ้าแบ่งออก 2 แบบหลักคือ สัญญาณอะนาลอก (analog) และดิจิทัล (digital) สัญญาณอะนาลอกก็คือ สัญญาณไฟฟ้าที่มีการเปลี่ยนแปลงสัมพันธ์กับค่าเวลา โดยการเปลี่ยนแปลงนั้นจะเพิ่มขึ้นหรือลดลงก็ได้ และมีระดับแรงดันเท่าใดก็ได้ ส่วนสัญญาณดิจิทัลเป็นสัญญาณที่มีการเปลี่ยนแปลงเพียง 2 ระดับที่แตกต่างกัน กล่าวคือ มีระดับสัญญาณสูง และต่ำ (เกือบหรือเท่ากับ 0V) โดยปกติจะมีระดับแรงดันเท่ากับ +5V สำหรับระดับแรงดันสูง หรือเรียกว่า ลอจิก “1” และ 0V สำหรับระดับแรงดันต่ำ หรือเรียกว่า ลอจิก “0” แต่ในปัจจุบันระดับแรงดันของลอจิก “1” อาจเท่ากับ +3.3V หรือ 1.8V ขึ้นอยู่กับเทคโนโลยีของอุปกรณ์ดิจิทัล อย่างไรก็ตาม สำหรับการเรียนรู้โดยพื้นฐานนี้จะอธิบายระดับลอจิก “1” ด้วยค่าแรงดัน +5V เป็นหลัก ในรูปที่ 10-1 แสดงความแตกต่างระหว่างสัญญาณอะนาลอกกับดิจิทัล



รูปที่ 10-1 ตัวอย่างของสัญญาณอะนาลอกและดิจิทัล

10.1 สัญญาณอะนาลอก

แบ่งได้ 3 แบบ คือ **แบบสัญญาณไฟตรง (analog DC signals)**, **แบบเปลี่ยนค่าตามเวลา (time-domain)** และ **แบบเปลี่ยนค่าตามความถี่ (frequency-domain)**

สัญญาณอะนาลอกไฟตรง มักเป็นค่าที่ได้จากการวัดขนาดหรือระดับของสัญญาณ ซึ่งมีการเปลี่ยนแปลงค่าสัญญาณในเวลาที่ไม่เร็วมากนัก อาทิ ค่าอุณหภูมิ, ระดับของไหล, ความดัน, อัตราการไหล, น้ำหนัก เป็นต้น สามารถใช้วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC : Analog to Digital Converter) ที่มีอัตราการสุ่มสัญญาณ ไม่เร็วมากได้

สัญญาณอะนาลอกแบบเปลี่ยนค่าตามเวลา เป็นสัญญาณที่วัดเพื่อพิจารณาลักษณะรูปสัญญาณเป็นหลัก อาทิ สัญญาณคลื่นหัวใจมนุษย์ (ECG) ซึ่งมีความจำเป็นต้องใช้วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลที่มีความเร็วในการสุ่มสัญญาณสูง

สัญญาณอะนาลอกแบบเปลี่ยนค่าตามความถี่ ได้แก่ สัญญาณความถี่วิทยุ (radio frequency : RF) และสัญญาณคลื่นเสียง เป็นต้น ในการวิเคราะห์จำเป็นต้องมีฮาร์ดแวร์พิเศษเพื่อช่วยวิเคราะห์อย่าง DSP (digital signal processing) ทำงานร่วมกับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลด้วย

10.2 ทำไมไมโครคอนโทรลเลอร์ต้องอ่านค่าสัญญาณอะนาลอก

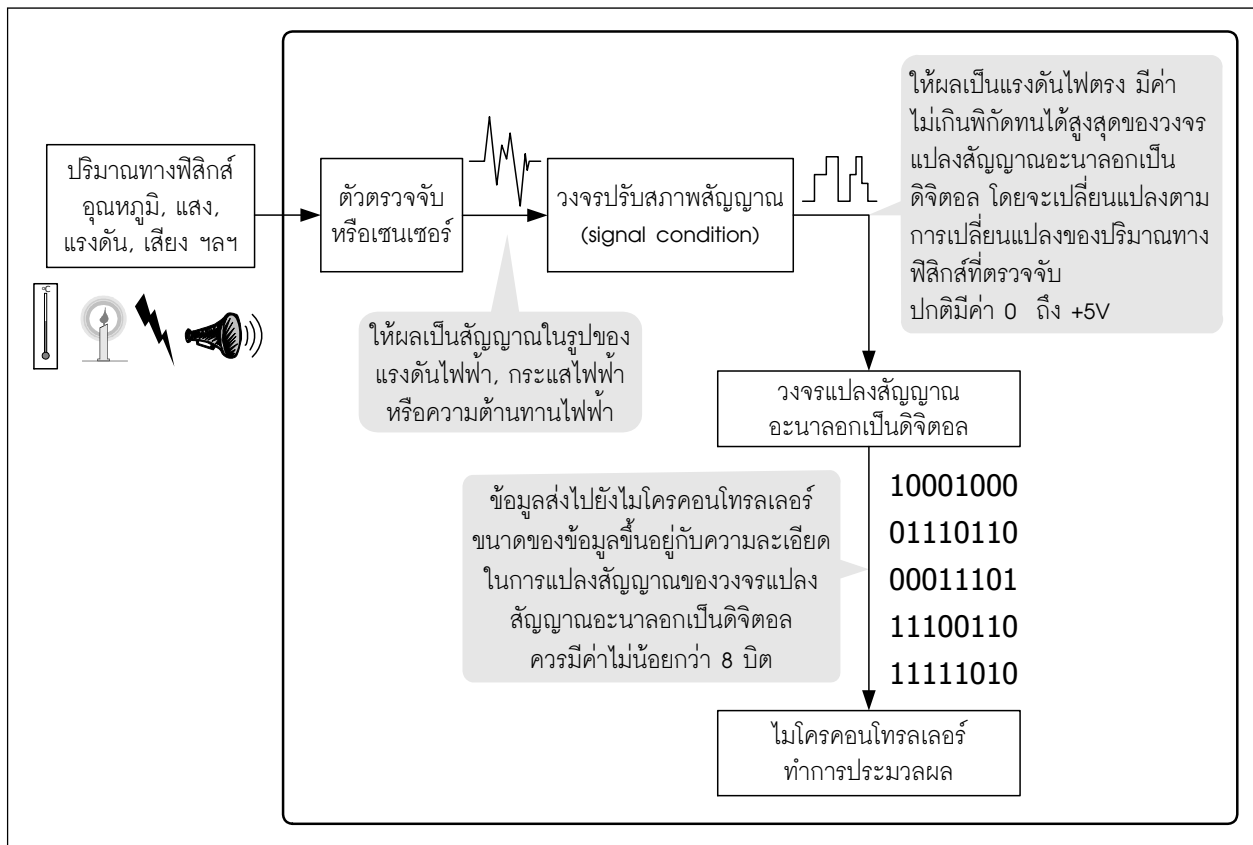
สาเหตุหลักที่ไมโครคอนโทรลเลอร์ต้องติดต่อกับสัญญาณอะนาลอกคือ **ต้องการอ่านค่าปริมาณทางฟิสิกส์ในรูปของสัญญาณไฟฟ้า เพื่อนำไปประมวลผลและควบคุมระบบต่อไป**

ในรูปที่ 10-2 แสดงไดอะแกรมการทำงานเบื้องต้นของการอ่านค่าสัญญาณอะนาลอกของไมโครคอนโทรลเลอร์ มีส่วนประกอบสำคัญ 4 ส่วนคือ

1. ส่วนตรวจจับสัญญาณกายภาพ (Transducer/Sensor/Detector)
2. วงจรปรับสภาพสัญญาณ (Signal conditioning)
3. วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล (Analog to Digital Converter : ADC)
4. ไมโครคอนโทรลเลอร์

10.2.1 ส่วนชุดตรวจจับสัญญาณกายภาพ

ชุดตรวจจับสัญญาณกายภาพ หรือ **ทรานสดิวเซอร์ (transducer)** คือตัวแปลงสัญญาณทางกายภาพ เช่น อุณหภูมิ ความดัน ระดับของไหล ความยาว ตำแหน่งการเคลื่อนที่ ฯลฯ ให้เป็นในรูปสัญญาณทางไฟฟ้า นั่นเอง หรือบางครั้งเรียกอุปกรณ์เหล่านี้ว่า **ตัวตรวจจับ (sensor)** ซึ่งมีด้วยกันหลากหลายรูปแบบ อาทิ เทอร์โมคัปเปิล (thermocouple) , เทอร์มิสเตอร์ (thermistor), ตัวตรวจจับการไหล (flow sensor) เป็นต้น



รูปที่ 10-2 แสดงกระบวนการอ่านค่าสัญญาณอนุบาลอกของไมโครคอนโทรลเลอร์

10.1.2 ส่วนปรับสภาพสัญญาณ

หน้าที่หลักของส่วนปรับสภาพสัญญาณนี้คือ **ปรับปรุงคุณภาพของสัญญาณไฟฟ้า** ที่ได้จากชุดตรวจจับก่อนส่งสัญญาณต่อไปยังฮาร์ดแวร์ที่เชื่อมต่อกับไมโครคอนโทรลเลอร์ ซึ่งอาจมีความจำเป็นต้องปรับสเกลสัญญาณ, ขยายขนาดสัญญาณ, แปลงรูปสัญญาณให้เป็นเชิงเส้น (*linearization*), กรองคลื่นสัญญาณและแยกกราวด์ของสัญญาณ (*common-mode rejection*)

หน้าที่เด่นของส่วนปรับสภาพสัญญาณคือ **ขยายขนาดสัญญาณ (*amplify*)** เพราะโดยส่วนใหญ่สัญญาณที่ได้จากชุดตรวจจับจะมีขนาดสัญญาณที่ต่ำมาก มีขนาดแรงดันไฟฟ้าในหน่วยมิลลิโวลต์ (*millivolt : mV*) หรือ $1/1000V$ และมักมีสัญญาณรบกวนจากแหล่งจ่ายไฟปะปนมา ซึ่งอาจรบกวนสัญญาณด้านอินพุตในขณะที่สัญญาณเข้าสู่ระบบ ทำให้ค่าสัญญาณที่วัดไม่ถูกต้องและไม่เที่ยงตรง

นอกจากนั้นวงจรปรับสภาพสัญญาณยังใช้ในการแปลงสัญญาณไฟฟ้าที่ไม่ได้อยู่ในรูปแบบของแรงดันไฟฟ้า เช่น กระแสไฟฟ้าหรือความต้านทานไฟฟ้ามาอยู่ในรูปของแรงดันไฟฟ้าให้เหมาะสมกับวงจรแปลงสัญญาณอนุบาลอกเป็นดิจิตอล อาทิ แปลงค่ากระแสไฟฟ้า 4 ถึง 20mA จากตัวตรวจจับที่ให้ผลแบบกระแสไฟฟ้าเป็นแรงดันไฟตรง 0 ถึง +5V เป็นต้น

10.1.3 วงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัล

วงจรนี้ทำหน้าที่รับสัญญาณอะนาลอกที่ผ่านมาจากวงจรปรับสภาพสัญญาณ เพื่อทำการแปลงเป็นข้อมูลทางดิจิทัลเพื่อส่งไปยังประมวลผลยังไม่โครคอนโทรลเลอร์ต่อไป จุดที่ต้องให้ความสนใจในส่วนนี้คือ ความละเอียดในการแปลงสัญญาณ วงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัลที่เหมาะสมในการนำมาใช้ในการทดลองวิทยาศาสตร์ควรมีความละเอียดไม่น้อยกว่า 8 บิต ซึ่งให้ความแตกต่างของข้อมูล 256 ค่า และถ้ายังมีความละเอียดสูงเท่าใดยิ่งดี เพราะจะทำให้ผลการแปลงที่แม่นยำมากขึ้น แต่นั่นจะทำให้ต้นทุนของระบบสูงขึ้นตามไปด้วย

10.1.4 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์เป็นหนึ่งในอุปกรณ์ของระบบดิจิทัล ดังนั้นการอ่านค่าสัญญาณอะนาลอกโดยตรงจึงต้องใช้อุปกรณ์ช่วยเพิ่มเติม นั่นคือ วงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัล (analog to digital converter) ไมโครคอนโทรลเลอร์ในอดีตจะไม่มีวงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัลบรรจุอยู่ในตัวชิป จึงต้องใช้ไอซีแปลงสัญญาณอะนาลอกเป็นดิจิทัลเข้ามาช่วย โดยไอซีแปลงสัญญาณนี้จะให้ผลลัพธ์ออกมาเป็นข้อมูลดิจิทัลที่มีความละเอียดของข้อมูลต่างกันไปแล้วแต่ความสามารถของไอซี โดยจะเริ่มตั้งแต่ 8, 10, 12, 16 บิตหรือสูงกว่า

ในปัจจุบัน ไมโครคอนโทรลเลอร์ได้รวมเอาวงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัลเข้ามาไว้ภายในชิป ทั้งนี้เพื่อลดขนาดของระบบโดยรวมลง ทำให้การประมวลผลสัญญาณทำได้เร็วขึ้น และต้นทุนรวมของระบบลดลงตามไปด้วย

ข้อมูลดิจิทัลที่ได้มาจากวงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัล ไม่ว่าจะ เป็นแบบไอซีแปลงสัญญาณภายนอก หรือจากโมดูลที่อยู่ภายในไมโครคอนโทรลเลอร์จะถูกส่งเข้ามาในระบบบัสข้อมูลเพื่อทำการประมวลผลและนำไปใช้ในการตัดสินใจเพื่อควบคุมการทำงานของระบบต่อไป

สำหรับในชุด IPST-MicroBOX (SE) ได้เลือกใช้ไมโครคอนโทรลเลอร์ ATmega644P อันเป็นไมโครคอนโทรลเลอร์ที่มีวงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัล 8 ช่องอยู่ภายในชิปแล้ว จึงทำให้การนำแผงวงจรควบคุมไปใช้ในการเชื่อมต่อกับตัวตรวจจับแบบต่างๆ เพื่ออ่านค่ากระทำได้ง่ายและสะดวกขึ้น

10.2 การแปลงสัญญาณอะนาลอกเป็นดิจิทัล (ADC)

การแปลงสัญญาณอะนาลอกเป็นดิจิทัล สัญญาณจะได้รับการแปลงเป็นจำนวนทางดิจิทัล โดยการสุ่มหรือแซมปลิง (sampling) ดังในรูปที่ 10-3 ถ้าหากวงจรแปลงสัญญาณอะนาลอกเป็นดิจิทัล มีความละเอียด 8 บิต จะมีความแตกต่างทางผลลัพธ์เลขฐานสองทั้งหมด 2^8 หรือ 256 ค่า และถ้าหาก มีความละเอียด 10 บิต ก็จะทำให้ผลลัพธ์ของข้อมูลเลขฐานสองสูงถึง 2^{10} หรือ 1,024 ค่า

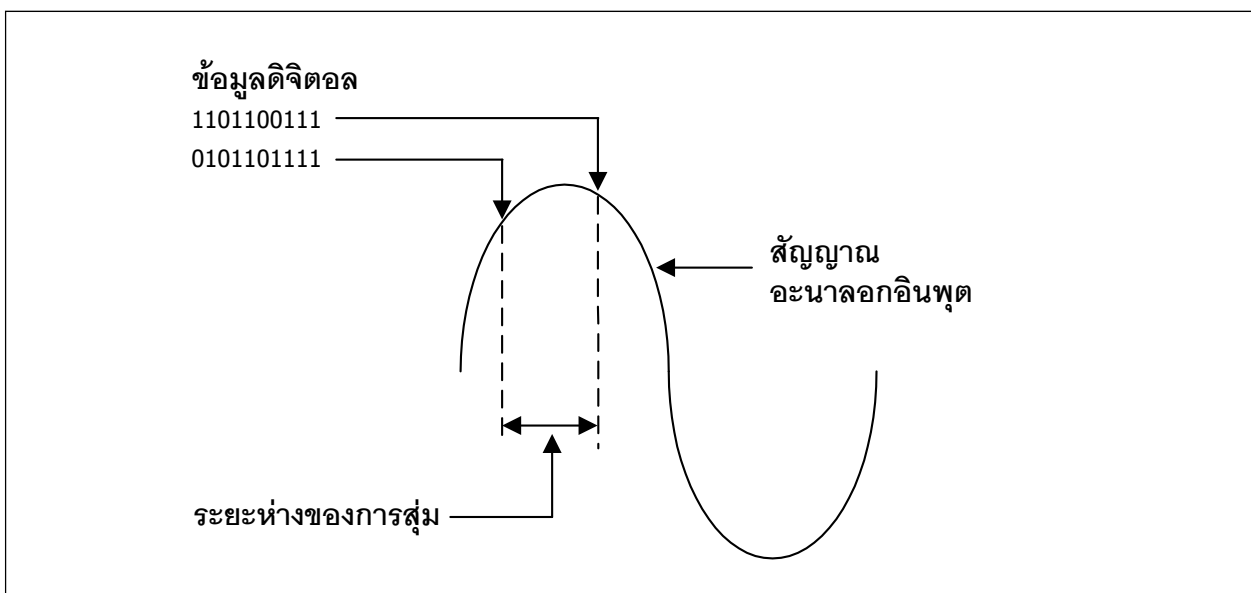
ค่าความละเอียดของตัวแปลงสัญญาณอะนาลอกเป็นดิจิทัลถูกอธิบายเป็นระยะห่างที่น้อยที่สุดของค่าแรงดันทางอินพุตที่เพิ่มขึ้น ซึ่งถูกกำหนดโดยตัวแปลงสัญญาณนั้น ระยะห่างยิ่งน้อยเท่าไร ค่าความละเอียดยิ่งสูงขึ้น โดยค่าความละเอียดแปรผันตรงกับจำนวนของบิตเอาต์พุต ยกตัวอย่างเช่น ถ้าวงจรแปลงสัญญาณมีความละเอียด 10 บิต ทำให้เกิดผลลัพธ์ที่แทนค่าสัญญาณมากถึง 1,024 ระดับสัญญาณอินพุตถูกแทนเป็นรหัสเลขฐานสองจาก 0000000000 ถึง 1111111111 ถ้าย่านอินพุตเริ่มต้นจาก 0 ถึง +5 V ดังนั้นความละเอียดเท่ากับ (ค่าโดยประมาณ)

$$\frac{5}{1024} = 0.005 \text{ V}$$

ถ้าเอาต์พุตรหัสเลขฐานสองเป็น 0000000001 แทนแรงดัน 0.005 V ดังนั้นข้อมูลของแรงดัน 3V จะมีค่าเท่ากับ

$$\frac{3}{0.005} = 600_{10}$$

ทำการแปลงเป็นเลขฐานสองจะได้ค่าเท่ากับ 1001011000_2



รูปที่ 10-3 การสุ่มสัญญาณอะนาลอกเพื่อกำหนดข้อมูลดิจิทัล

10.3 ฟังก์ชันของโปรแกรมภาษา C/C++ ที่ใช้อ่านค่าสัญญาณอะนาลอกของ IPST-MicroBOX (SE)

เพื่ออำนวยความสะดวกในการเขียนโปรแกรมภาษา C/C++ เพื่อควบคุมกล่องสมองกล IPST-MicroBOX (SE) เพื่อให้อ่านค่าสัญญาณอะนาลอกจากตัวตรวจจับต่างๆ ในไฟล์ไลบรารี ipst.h จึงได้บรรจุฟังก์ชันสำหรับอ่านค่าสัญญาณอะนาลอกโดยเฉพาะ 2 ฟังก์ชันคือ analog() และ knob();

การอ่านค่าผ่านฟังก์ชัน analog() และ knob() จะได้ผลลัพธ์คืนค่ากลับมาในช่วง 0 ถึง 1,023 ของเลขฐานสิบ หรือ 0x0000 ถึง 0x03FF ของเลขฐานสิบหก เนื่องจากความละเอียดในการแปลงสัญญาณอะนาลอกเป็นดิจิตอลเท่ากับ 10 บิต (เกิดค่าได้ 1,024 ค่า)

10.3.1 analog

เป็นฟังก์ชันอ่านค่าจากการแปลงสัญญาณอะนาลอกของแผงวงจร IPST-SE ที่จุดต่อ A0 ถึง A6

รูปแบบ

```
unsigned int analog(unsigned char channel)
```

พารามิเตอร์

channel - กำหนดช่องอินพุตที่ต้องการ มีค่า 0 ถึง 6 ซึ่งตรงกับขาพอร์ต A0 ถึง A6

การคืนค่า

เป็นข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้า 0 ถึง +5V จากช่องอินพุตที่กำหนด มีค่า 0 ถึง 1,023

10.3.2 knob()

เป็นฟังก์ชันอ่านค่าข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้าที่ขาพอร์ต A7 ซึ่งต่อกับตัวต้านทานปรับค่าได้ที่ตำแหน่ง KNOB

รูปแบบ

```
unsigned int knob()
```

การคืนค่า

เป็นข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้าที่มาจากปรับค่าที่ปุ่ม KNOB บนแผงวงจร IPST-SE มีค่า 95 ถึง 1,023

ปฏิบัติการที่ 6 อ่านค่าตัวตรวจจับอะนาลอกอย่างง่าย

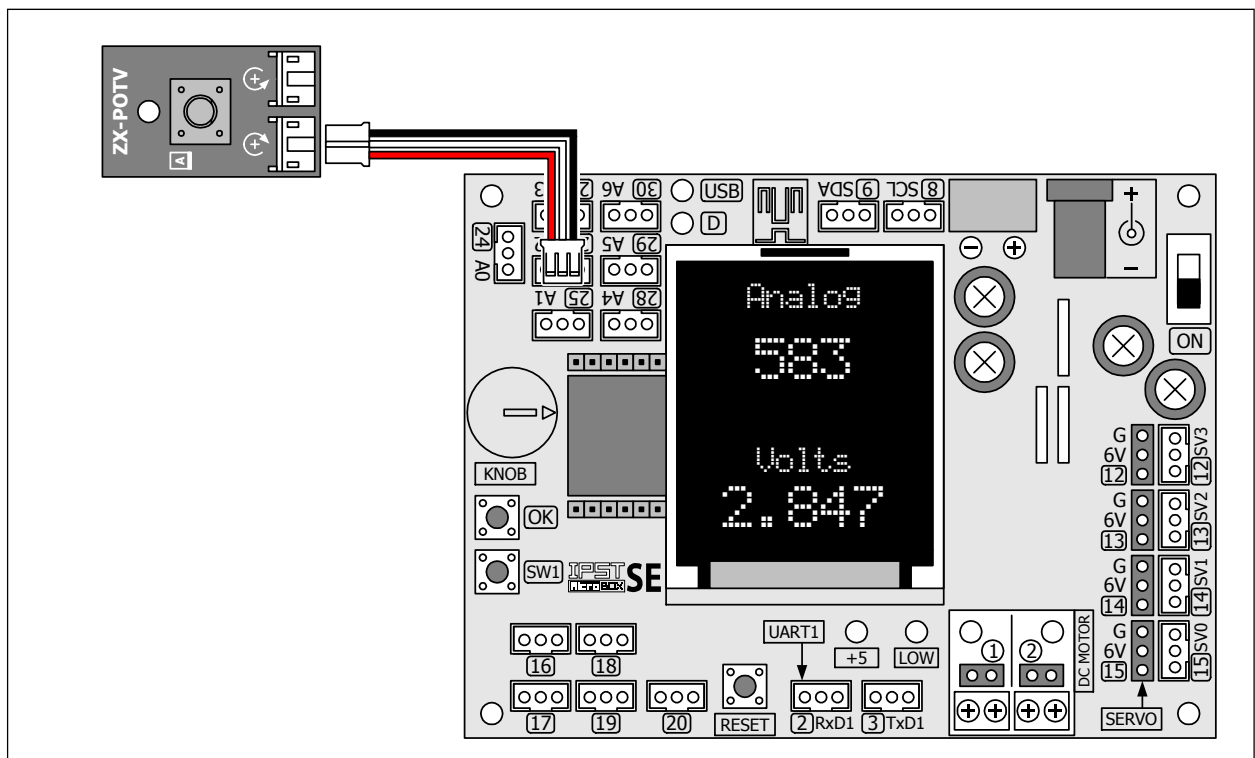
ในปฏิบัติการนี้เป็นการแนะนำการเชื่อมต่อชุดกล่องสมองกล IPST-MicroBOX (SE) กับอุปกรณ์ที่ให้ผลการทำงานเป็นแรงดันไฟตรง เพื่อทดสอบการอ่านค่าสัญญาณอะนาลอกหรือสัญญาณไฟตรงมาแสดงผลเป็นตัวเลขที่เข้าใจได้ง่ายขึ้น ซึ่งการทำงานของอุปกรณ์แบบนี้จะแตกต่างกันไปตามลักษณะและจุดประสงค์ เช่น ตัวตรวจจับอุณหภูมิที่ให้ผลการเปลี่ยนแปลงอุณหภูมิสัมพันธ์กับแรงดันทางเอาต์พุต, ตัวต้านทานแปรค่าตามแสงที่สามารถให้แรงดันไฟตรงเอาต์พุตเปลี่ยนแปลงตามความเข้มของแสงที่มาตกกระทบตัวมัน เป็นต้น ตัวแทนของอุปกรณ์ในลักษณะนี้ในขั้นพื้นฐานที่สุดคือ ZX-POTV แผงวงจรตัวต้านทานปรับค่าได้ (Potentiometer) โดยในปฏิบัติการนี้จะแสดงให้เห็นถึงการอ่านค่าและนำผลลัพธ์มาประยุกต์ใช้งานร่วมกับแผงวงจรหลัก IPST-SE

ปฏิบัติการที่ 6-1 อ่านค่าตัวตรวจจับมาแสดงผล

ในปฏิบัติการนี้แนะนำการเขียนโปรแกรมภาษา C/C++ เพื่ออ่านค่าสัญญาณไฟฟ้าแบบอะนาลอกจากแผงวงจรตัวต้านทานปรับค่าได้ที่ต่อกับจุดต่อพอร์ต A1 มาแสดงผลที่จอแสดงผลกราฟิก LCD สีบนแผงวงจรหลัก IPST-SE

การเชื่อมต่อทางฮาร์ดแวร์

- ต่อเอาต์พุต \oplus หรือเอาต์พุตปรับแรงดันเพิ่มเมื่อหมุนตามเข็มนาฬิกาของแผงวงจร ZX-POTV กับจุดต่อ A1 ของแผงวงจรหลัก IPST-SE



รูปที่ L6-1 การต่อวงจรเพื่อทำการทดลองสำหรับปฏิบัติการที่ 6

ขั้นตอนการทดลอง

6.1.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L6-1 บันทึกในชื่อ microbox_AnalogTest.ino

6.1.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

```
#include <ipst.h>           // ผนวกไฟล์ไลบรารีหลัก
int val = 0;
float volts = 0.0;
void setup()
{
  glcdClear();
  setTextSize(2);          // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}
void loop()
{
  glcd(0,2,"Analog");      // แสดงข้อความที่จอแสดงผล
  val = analog(1);        // อ่านค่าของสัญญาณช่อง A1 มาเก็บไว้ที่ตัวแปร val
  setTextSize(3);         // เลือกขนาดตัวอักษรใหญ่เป็น 3 เท่าจากขนาดปกติ
  glcd(1,2,"%d ",val);    // แสดงค่าที่อ่านได้จากจุดต่อ A1 ที่หน้าจอแสดงผล
  setTextSize(2);         // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
  glcd(5,3,"Volts");      // แสดงข้อความ Volts
  volts = (float(val)*5)/1024; // แปลงข้อมูลเป็นหน่วยแรงดัน
  setTextSize(3);         // เลือกขนาดตัวอักษรใหญ่เป็น 3 เท่าจากขนาดปกติ
  glcd(4,1,"%f",volts);  // แสดงค่าแรงดันความละเอียดทศนิยม 3 ตำแหน่ง
  setTextSize(2);         // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}
```

คำอธิบายโปรแกรม

ค่าของแรงดันไฟตรงที่ได้จากตัวตรวจจับสนหรือตัวต้านทานปรับค่าได้ที่จุดต่อ A1 ของแผงวงจร IPST-SE จะถูกแปลงเป็นข้อมูลดิจิทัลและจัดรูปแบบเป็นเลขฐานสิบได้ค่าในช่วง 0 ถึง 1023 จากการทำงานของฟังก์ชัน analog() จากนั้นข้อมูลนั้นได้รับการส่งต่อไปแสดงที่จอแสดงผลกราฟิก LCD สีด้วยฟังก์ชัน glcd อย่างต่อเนื่อง

นอกจากนั้นในโปรแกรมยังนำข้อมูลที่ได้จากการแปลงสัญญาณมาคำนวณกลับ เพื่อให้ได้เป็นค่าแรงดันไฟตรง โดยใช้สมการ $volts = (val \times 5) / 1024$ แล้วใช้ตัวแปร volts ที่เป็นตัวแปรแบบทศนิยมมารับค่าที่ได้จากคำนวณเพื่อนำไปแสดงผลที่จอกราฟิก LCD สี โดยแสดงเป็นค่าแรงดันในหน่วยโวลต์ ด้วยความละเอียดทศนิยม 3 ตำแหน่ง

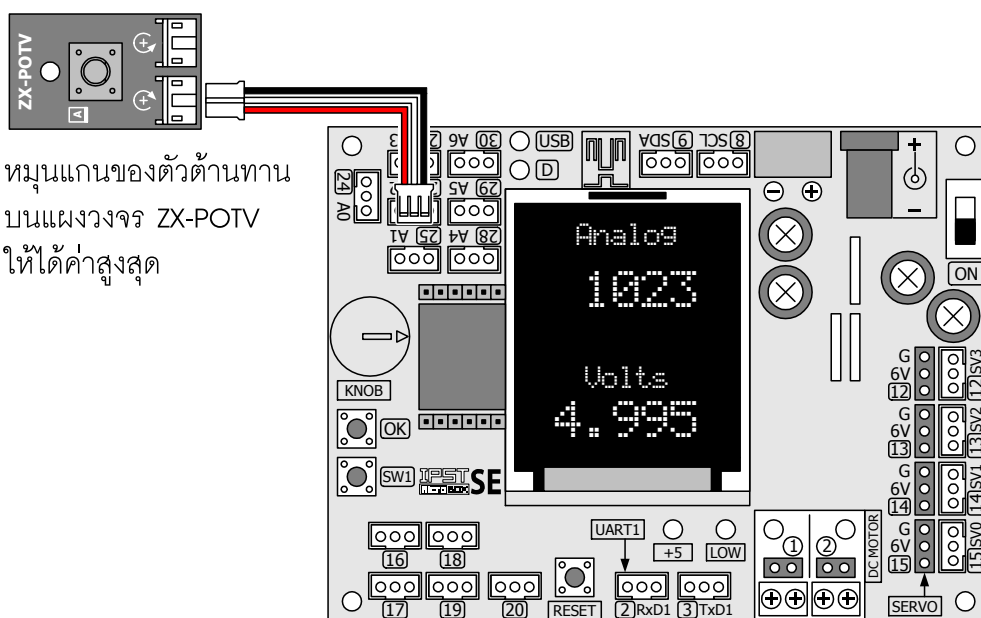
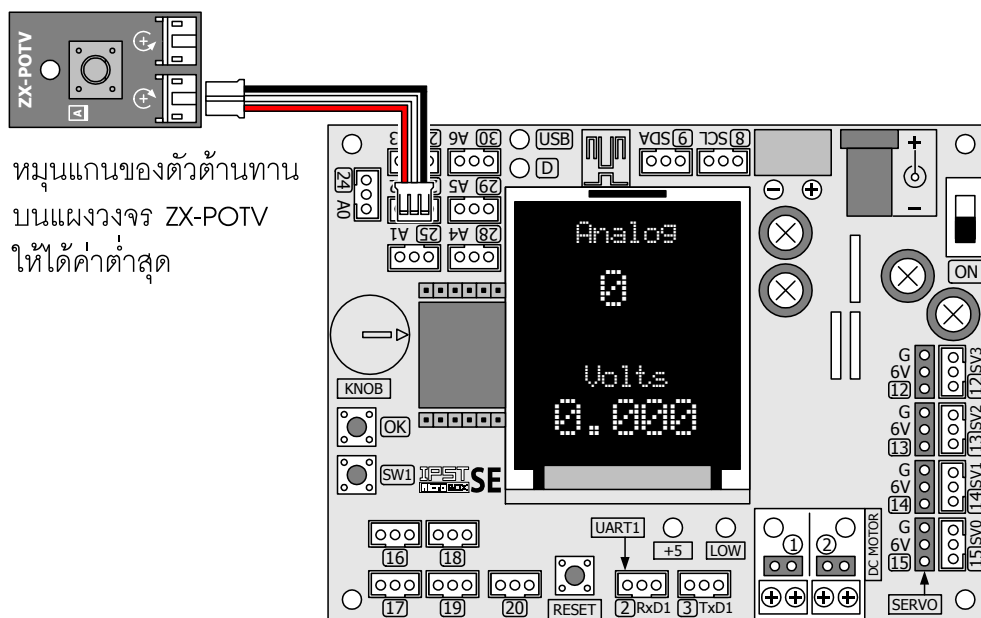
ในโปรแกรมมีการกำหนดขนาดของตัวอักษรที่แสดงผลต่างกัน เนื่องจากต้องการแยกข้อความและข้อมูลออกจากกันให้ชัดเจน

โปรแกรมที่ L6-1 : ไฟล์ microbox_AnalogTest.ino โปรแกรมภาษา C/C++ สำหรับอ่านค่าสัญญาณอะนาลอกของแผงวงจร IPST-SE

6.1.3 รันโปรแกรม ทดลองหมุนแกนของตัวต้านทานบนแผงวงจรตัวต้านทานปรับค่าได้ ZX-POTV สังเกตผลการทำงานผ่านทางจอแสดงผลผลของแผงวงจร IPST-SE

ที่จอแสดงผลกราฟิก LCD ช่วงบนแสดงค่าข้อมูลที่ได้จากการแปลงสัญญาณที่จุดต่อ A1 ซึ่งต่อกับแผงวงจรตัวต้านทานปรับค่าได้ ZX-POTV โดยมีค่าระหว่าง 0 ถึง 1023 (เทียบกับแรงดัน 0 ถึง +5V)

ที่ช่วงล่างของจอแสดงผลกราฟิก LCD สีแสดงค่าแรงดันไฟตรงในหน่วย โวลต์ (Volts) ที่ได้จากการปรับค่าของตัวต้านทานบนแผงวงจรตัวต้านทานปรับค่าได้ ZX-POTV โดยมีค่าระหว่าง 0.000 ถึง 4.995 (เทียบกับข้อมูล 0 ถึง 1023)



ปฏิบัติการที่ 6.2 ควบคุมการเปิด/ปิด LED ด้วยแผงวงจรตัวต้านทานปรับค่าได้

ในปฏิบัติการนี้เป็นการนำข้อมูลที่ได้จากการแปลงสัญญาณไฟฟ้าซึ่งมาจากการปรับค่าของ ZX-POTV แผงวงจรตัวต้านทานปรับค่าได้มากำหนดเงื่อนไขในการเปิด/ปิด LED เพื่อให้เห็นแนวทางในการประยุกต์ใช้งานเบื้องต้น

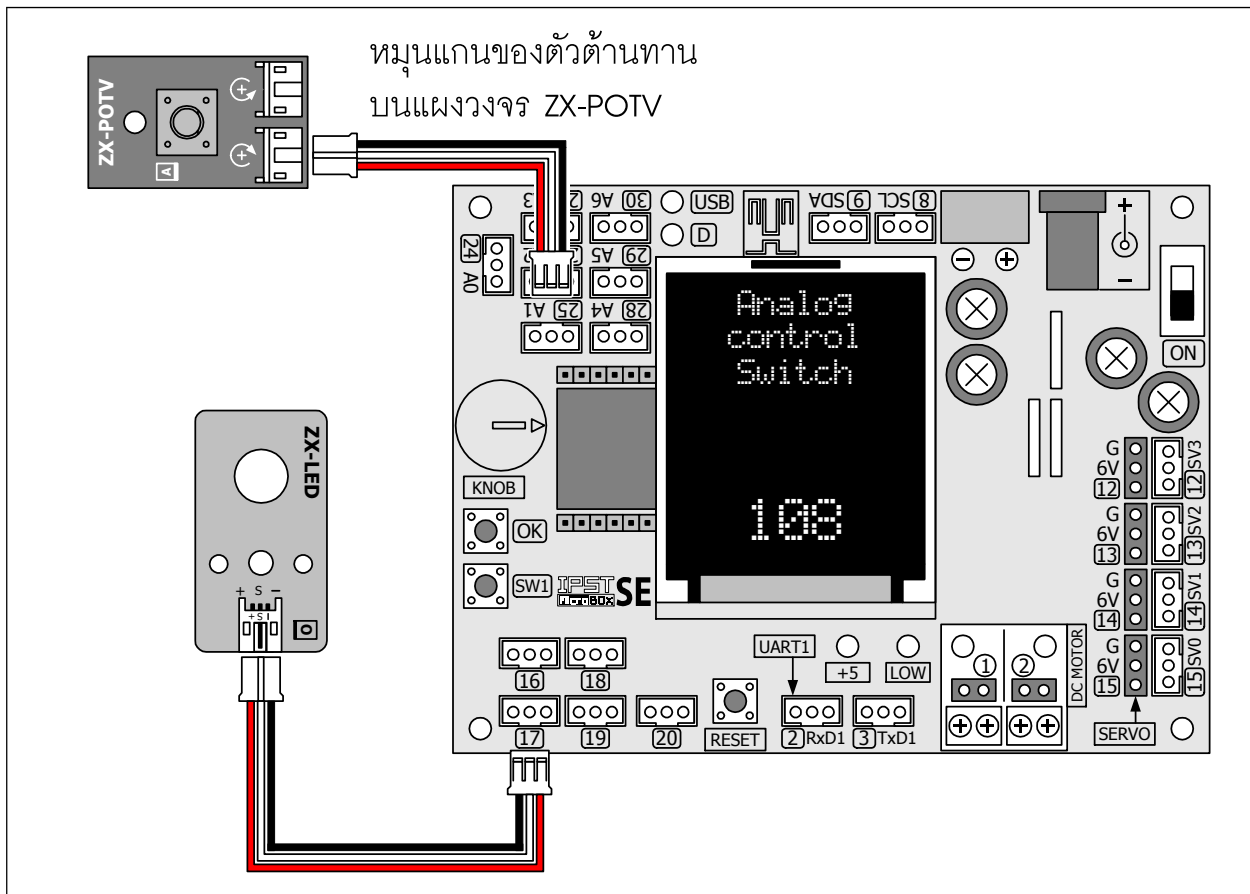
การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อแผงวงจร ZX-LED เข้ากับจุดต่อพอร์ต 17 ของแผงวงจรหลัก IPST-SE
- ต่อเอาต์พุต ⊕ หรือเอาต์พุตปรับแรงดันเพิ่มเมื่อหมุนตามตามเข็มนาฬิกาของแผงวงจร ZX-POTV กับจุดต่อพอร์ต A1 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

6.2.1 เปิดซอฟต์แวร์ Wiring 1.0 สร้างไฟล์ใหม่ พิมพ์โปรแกรมที่ L6-1 บันทึกในชื่อ microbox_AnalogSwitch.ino

6.2.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 



รูปที่ L6-2 การต่อวงจรเพื่อทำการทดลองสำหรับปฏิบัติการที่ 6-2

```

#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
int val=0; // กำหนดตัวแปรเก็บค่าที่ได้จากการแปลงสัญญาณแล้ว
void setup()
{
  lcdClear();
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
  lcd(1,2,"Analog"); // แสดงข้อความที่จอแสดงผล
  lcd(2,2,"control");
  lcd(3,2,"Switch");
}
void loop()
{
  val = analog(1); // อ่านค่าของสัญญาณช่อง A1 มาเก็บไว้ที่ตัวแปร val
  if(val>512) // ตรวจสอบว่าค่าที่อ่านได้มากกว่า 512 หรือไม่
  {
    setTextSize(4); // เลือกขนาดตัวอักษรใหญ่เป็น 4 เท่าจากขนาดปกติ
    setTextColor(GLCD_RED); // เปลี่ยนเป็นสีแดง
    lcd(3,1,"%d ",val); // แสดงค่าที่อ่านได้จากจุดต่อ A1 ที่หน้าจอแสดงผล
    out(17,1); // ถ้าค่า val มากกว่า 512 ทำการขับ LED ที่พอร์ต 17
  }
  else
  {
    setTextSize(4); // เลือกขนาดตัวอักษรใหญ่เป็น 4 เท่าจากขนาดปกติ
    setTextColor(GLCD_WHITE); // แสดงตัวอักษรสีขาว
    lcd(3,1,"%d ",val); // แสดงค่าที่อ่านได้จากจุดต่อ A2 ที่หน้าจอแสดงผล
    out(17,0); // ถ้าค่า val น้อยกว่า 512 ทำการปิด LED ที่พอร์ต 17
  }
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}

```

คำอธิบายโปรแกรม

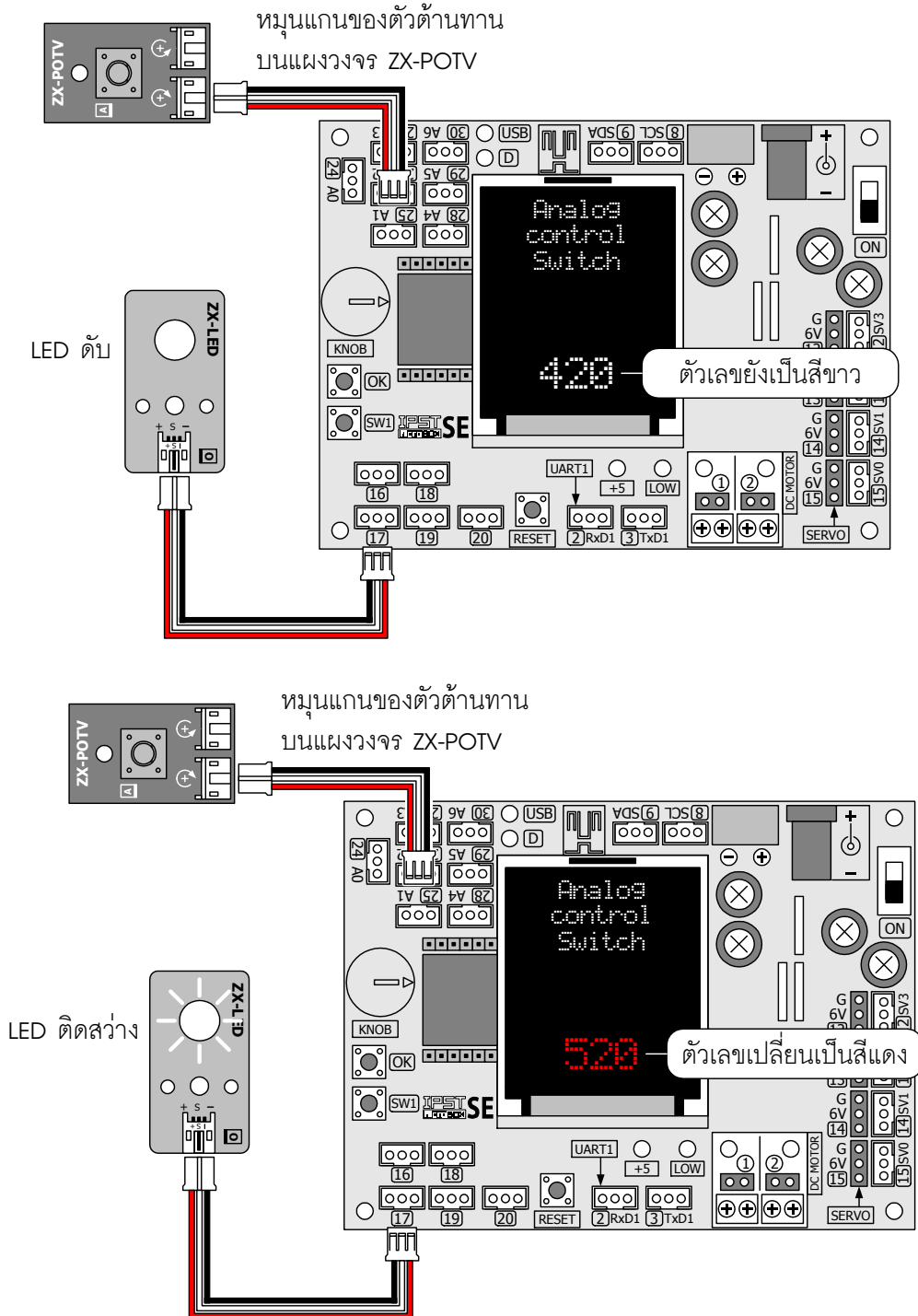
สัญญาณไฟฟ้าจากแผงวงจรตัวต้านทานปรับค่าได้จะถูกอ่านด้วยคำสั่ง analog() เก็บไว้ที่ตัวแปร val เพื่อนำไปตรวจสอบและส่งไปแสดงผลยังจอแสดงผลกราฟิก LCD สี หากค่าที่ได้ น้อยกว่า 512 ตัวเลขที่แสดงผลยังเป็นสีขาว และส่งข้อมูล "0" ไปยังพอร์ต 17 ทำให้ LED ที่ต่ออยู่ไม่ทำงาน

เมื่อค่าของ val มากกว่า 512 ตัวเลขแสดงผลจะเปลี่ยนเป็นสีแดง และมีการส่งข้อมูล "1" ไปยังพอร์ต 17 ทำให้ LED ที่ต่ออยู่ติดสว่าง

โปรแกรมที่ L6-2 : ไฟล์ microbox_AnalogSwitch.ino โปรแกรมภาษา C/C++ สำหรับอ่านค่าสัญญาณไฟฟ้า เพื่อนำมาควบคุมอุปกรณ์เอาต์พุต

6.2.3 รันโปรแกรม ทดลองปรับค่าที่แกนหมุนของตัวต้านทานปรับค่าได้บนแผงวงจร ZX-POTV สังเกตการทำงานของจอแสดงผลบนแผงวงจร IPST-SE และ LED บนแผงวงจร ZX-LED

เมื่อปรับค่าที่แกนของตัวต้านทาน สังเกตผลลัพธ์ที่จอแสดงผล มันจะแสดงค่า 0 ถึง 1023 เมื่อปรับค่าที่แผงวงจร ZX-POTV ถ้ามีค่าน้อยกว่า 512 ตัวเลขจะเป็นสีเขียว และ LED ดับ เมื่อได้ปรับค่าจนได้มากกว่า 512 ค่าตัวเลขที่จอแสดงผลจะเปลี่ยนเป็นสีแดง และ LED ที่ต่อกับพอร์ต 17 ติดสว่าง



ปฏิบัติการที่ 6.3 ควบคุมการเปิด/ปิด LED ด้วยแผงวงจรตรวจจับแสง

ในปฏิบัติการนี้เป็นการต่อยอดจากปฏิบัติการที่ 6.2 โดยเปลี่ยนจากการปรับค่าของ ZX-POTV แผงวงจรตัวต้านทานปรับค่าได้เป็นการตรวจจับแสงโดยใช้แผงวงจรตัวต้านทานแปรค่าตามแสงหรือ ZX-LDR โดยยังคงใช้เงื่อนไขในการเปิด/ปิด LED ในแบบเดียวกัน

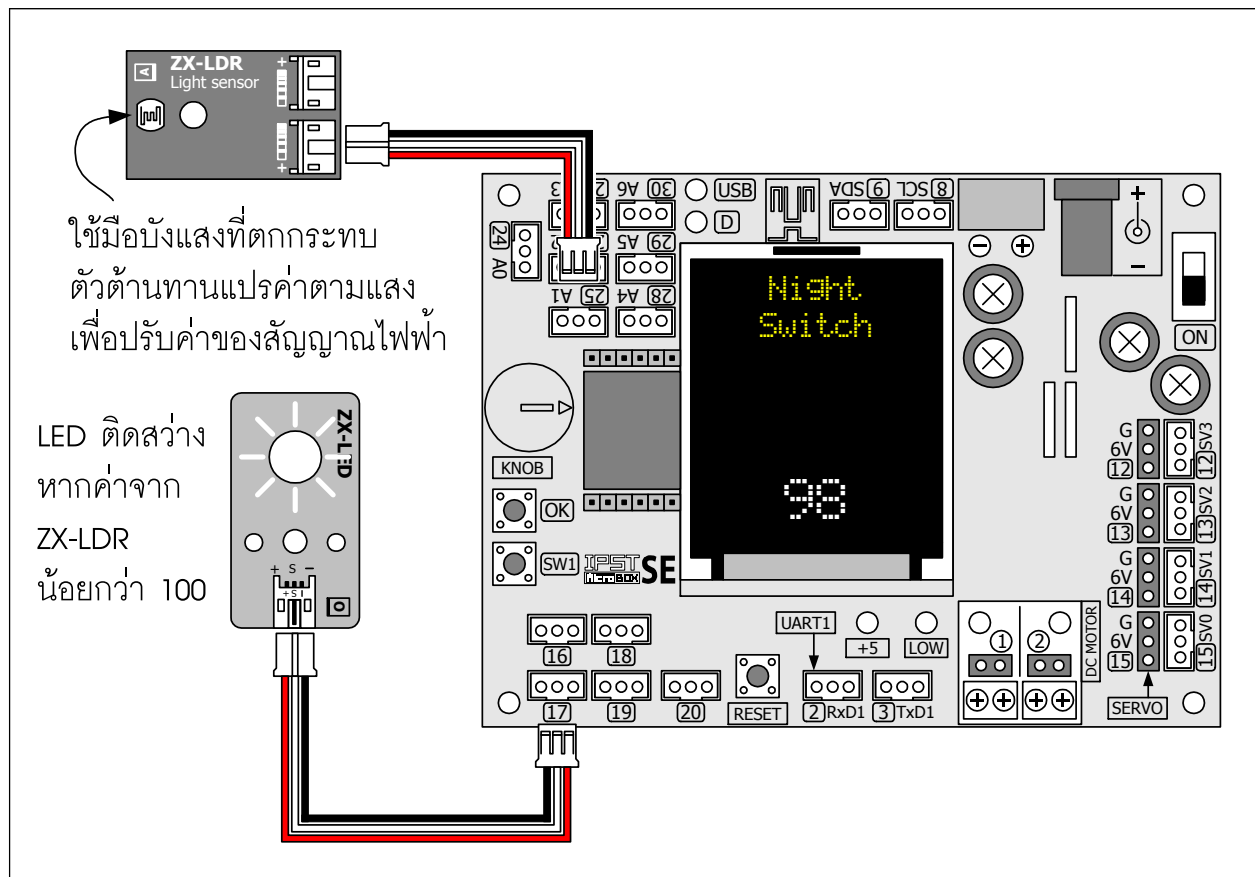
การเชื่อมต่อทางฮาร์ดแวร์

- เชื่อมต่อแผงวงจร ZX-LED เข้ากับจุดต่อพอร์ต 17 ของแผงวงจรหลัก IPST-SE
- ต่อเอาต์พุต + **XXXX** หรือเอาต์พุตแรงดันแปรค่าตามแสงของแผงวงจร ZX-LDR กับจุดต่อพอร์ต A1 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

6.3.1 เขียนโปรแกรมที่ L6-3 บันทึกในชื่อ microbox_NightSwitch.ino

6.3.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 



รูปที่ L6-3 การต่อวงจรเพื่อทดสอบการทำงานของแผงวงจรตรวจจับแสงเพื่อใช้กำหนดเงื่อนไขในการเปิด-ปิดอุปกรณ์เอาต์พุต

```

#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
int val=0; // กำหนดตัวแปรสำหรับเก็บค่าที่ได้จากการแปลงสัญญาณ
void setup()
{
  lcdClear();
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
  setTextColor(GLCD_YELLOW);
  lcd(1,2,"Night"); // แสดงข้อความที่จอแสดงผล
  lcd(2,2,"Switch");
}
void loop()
{
  val = analog(1); // อ่านค่าของสัญญาณช่อง A1 มาเก็บไว้ที่ตัวแปร val
  if(val<100) // ตรวจสอบว่าค่าที่อ่านได้มากกว่า 512 หรือไม่
  {
    setTextSize(4); // เลือกขนาดตัวอักษรใหญ่เป็น 4 เท่าจากขนาดปกติ
    setTextColor(GLCD_WHITE); // เปลี่ยนเป็นสีขาว
    lcd(3,1,"%d ",val); // แสดงค่าที่อ่านได้จากจุดต่อ A1 ที่หน้าจอแสดงผล
    out(17,1); // ถ้าค่า val น้อยกว่า 100 ทำการขับ LED ที่พอร์ต 17
  }
  else
  {
    setTextSize(4); // เลือกขนาดตัวอักษรใหญ่เป็น 4 เท่าจากขนาดปกติ
    setTextColor(GLCD_BLUE); // แสดงตัวอักษรสีน้ำเงิน
    lcd(3,1,"%d ",val); // แสดงค่าที่อ่านได้จากจุดต่อ A1 ที่หน้าจอแสดงผล
    out(17,0); // ถ้าค่า val มากกว่า 100 ทำการปิด LED ที่พอร์ต 17
  }
  setTextSize(2); // เลือกขนาดตัวอักษรใหญ่เป็น 2 เท่าจากขนาดปกติ
}

```

คำอธิบายโปรแกรม

ในโปรแกรมนี้อ่านค่าจากแผงวงจรตรวจจับแสงที่อ่านด้วยคำสั่ง `analog(1)`; เก็บไว้ที่ตัวแปร `val` หากค่าที่ได้มากกว่า 100 ตัวเลขที่แสดงผลเป็นสีน้ำเงิน และส่งข้อมูล "0" ไปยังพอร์ต 17 ทำให้ LED ที่ต่ออยู่ไม่ทำงาน

เมื่อค่าของ `val` น้อยกว่า 100 ตัวเลขแสดงผลจะเปลี่ยนเป็นสีขาว และมีการส่งข้อมูล "1" ไปยังพอร์ต 17 ทำให้ LED ที่ต่ออยู่ติดสว่าง

**โปรแกรมที่ L6-3 : ไฟล์ `microbox_NightSwitch.ino` โปรแกรมภาษา C/C++ สำหรับควบคุมการเปิดปิดอุปกรณ์
เอาต์พุตด้วยแสง**

6.3.3 รั้นโปรแกรม ทดลองใช้มือหรือแป้นกระดาษโปร่งแสงบังแสงที่ส่องมายังตัวต้านทานแปรค่าตามแสงหรือ LDR บนแผงวงจร ZX-LDR สังเกตการทำงานของจอแสดงผลบนแผงวงจร IPST-SE และ LED บนแผงวงจร ZX-LED

หาก ZX-LDR ได้รับแสงมาก สังเกตได้จากค่าที่แสดงบนจอแสดงผล นั่นคือ มีค่ามากกว่า 100 (ตัวเลขเป็นสีน้ำเงิน) จะสมมติสถานการณ์ว่า เป็นตอนกลางวัน จึงไม่มีเปิดไฟส่องสว่าง ซึ่งในที่นี้ใช้ LED บนแผงวงจร ZX-LED ทำหน้าที่แทน

แต่ถ้าหาก ZX-LDR ได้รับแสงลดลงจนต่ำกว่า 100 จะถือว่า เป็นตอนกลางคืน ระบบจะทำงานสั่งให้ LED ที่พอร์ต 17 ติดสว่าง จนกว่า ZX-LDR จะได้รับแสงมากเพียงพอ ซึ่งอาจตีความว่า เป็นตอนเช้าแล้ว วงจรขับ LED หยุดทำงาน ทำให้ LED ที่พอร์ต 17ดับ

ดังนั้น จึงอาจเรียกการทำงานของปฏิบัติการนี้ว่า สวิตช์สนธยา (Twilight Switch) หรือสวิตช์กลางคืน (Night Switch) ก็ได้

ปฏิบัติการที่ 7 เครื่องวัดอุณหภูมิระบบตัวเลขอย่างง่าย

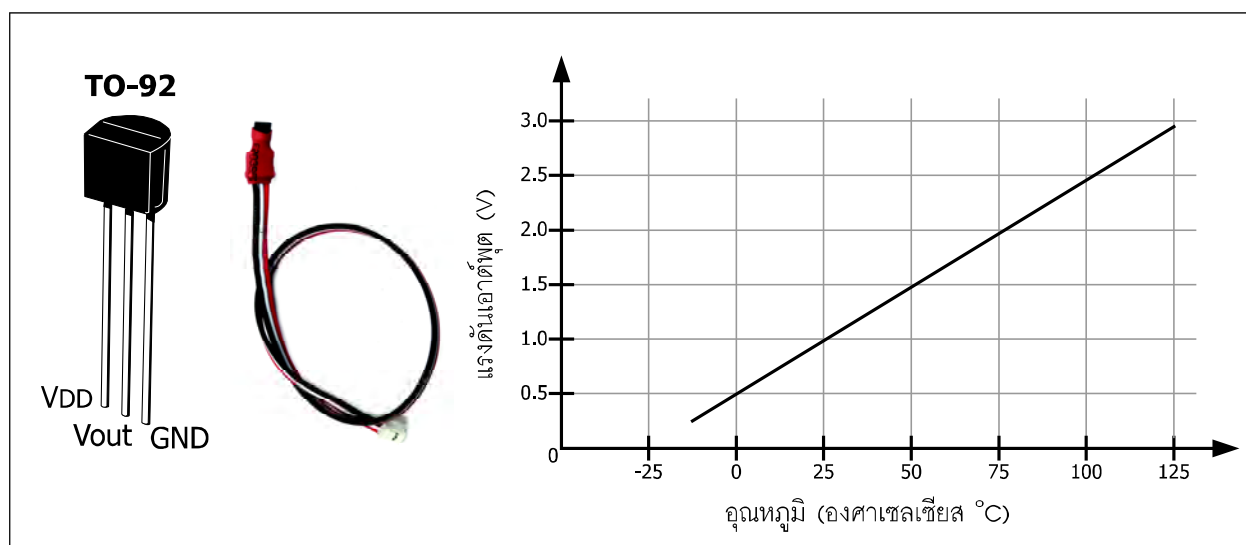
ในปฏิบัติการนี้เป็นการนำไอซีวัดอุณหภูมิที่ให้ผลการทำงานเป็นแรงดันไฟฟ้ามาเชื่อมต่อกับชุดกล่องสมองกล IPST-MicroBOX (SE) เพื่อสร้างเป็นเครื่องวัดอุณหภูมิระบบตัวเลขอย่างง่าย

รู้จักกับ MCP9701 ไอซีวัดอุณหภูมิ

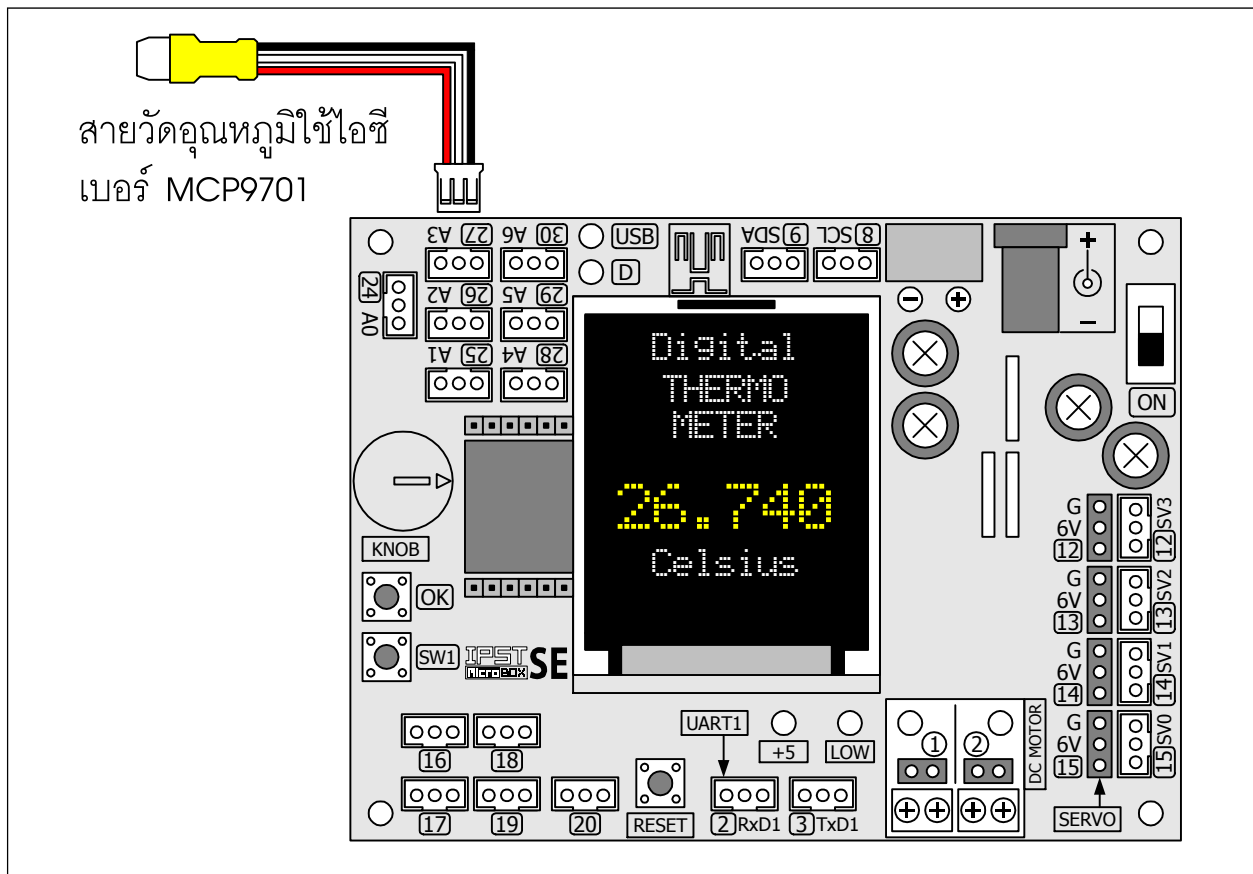
เป็นอุปกรณ์ตรวจจับและวัดอุณหภูมิที่ให้ผลการทำงานเป็นแรงดันไฟฟ้าแบบเชิงเส้น รับรู้การเปลี่ยนแปลงของอุณหภูมิภายในเวลาไม่ถึง 2 วินาที เชื่อมต่อกับอินพุตอะนาล็อก A0 ถึง A6 ของแผงวงจรหลัก IPST-SE ได้

คุณสมบัติทางเทคนิคของ MCP9701 ที่ควรทราบ

- เป็นไอซีวัดอุณหภูมิในกลุ่มเทอร์มิสเตอร์แบบแอคทีฟที่ให้ผลการทำงานแบบเชิงเส้น
- ย่านวัด -40 ถึง +125 องศาเซลเซียส
- ผลการวัดอ้างอิงกับหน่วยขององศาเซลเซียสโดยตรง
- ความผิดพลาดเฉลี่ย ± 2 องศาเซลเซียส
- ย่านไฟเลี้ยง +3.1 ถึง +5.5V กินกระแสไฟฟ้าเพียง 6 μ A ใช้แบตเตอรี่เป็นแหล่งจ่ายไฟได้
- ค่าแรงดันเอาต์พุต 500mV (ที่ 0°C) ถึง 2.9375V (ที่ 125°C)
- ค่าแรงดันเอาต์พุตต่อการเปลี่ยนแปลงอุณหภูมิ 19.5mV/°C ใช้งานกับวงจรแปลงสัญญาณอะนาล็อกเป็นดิจิทัลความละเอียดตั้งแต่ 8 บิตได้ โดยมีความคลาดเคลื่อนต่ำ



รูปที่ L7-1 การจัดขาของ MCP9701, หน้าตาเมื่อต่อสายสัญญาณพร้อมใช้งานและกราฟคุณสมบัติ




รูปที่ L7-2 การต่อวงจรเพื่อใช้งานชุดกล่องสมองกล IPST-MicroBOX (SE) กับไอซีวัดอุณหภูมิ MCP9701

การเชื่อมต่อทางฮาร์ดแวร์

- ต่อสายวัดอุณหภูมิที่เชื่อมต่อกับไอซี MCP9701 เข้ากับจุดต่อ A3 ของแผงวงจรหลัก IPST-SE

ขั้นตอนการทดลอง

7.1 เขียนโปรแกรมที่ L7-1 บันทึกในชื่อ microbox_Thermometer.ino

7.2 คอมไพล์และอัปโหลดโปรแกรมไปยังแผงวงจร IPST-SE โดยคลิกที่ปุ่ม 

7.3 รันโปรแกรม ทดลองใช้มือจับที่ตัวไอซีวัดอุณหภูมิ หรือนำหัววัดอุณหภูมิไปแช่ในน้ำแข็ง สังเกตการทำงานที่จอแสดงผลบนแผงวงจร IPST-SE

แผงวงจร IPST-SE แสดงข้อความแจ้งหน้าที่การทำงาน และแสดงค่าอุณหภูมิในหน่วยองศาเซลเซียส (Celsius) ด้วยความละเอียดทศนิยม 3 ตำแหน่ง โดยที่ค่าของอุณหภูมิจะแสดงด้วยตัวเลขสีเหลืองขนาดใหญ่ (3x)

```

#include <ipst.h> // ผนวกไฟล์ไลบรารีหลัก
int val,i; // กำหนดตัวแปรเก็บค่าที่อ่านได้จาก MCP9701
float Temp; // ประกาศตัวแปรค่าอุณหภูมิในแบบทศนิยม
void setup()
{
  glcdClear(); // เคลียร์จอแสดงผล
  setTextSize(2); // เลือกขนาดตัวอักษร 2 เท่า
}

void loop()
{
  glcd(1,2,"Digital"); // แสดงข้อความเริ่มต้น
  glcd(2,2,"THERMO");
  glcd(3,3,"METER");
  val=0;
  for (i=0;i<20;i++) // กำหนดรอบการอ่านค่าจาก MCP9701 รวม 20 ครั้ง
  {
    val = val+analog(3); // อ่านค่าจากอินพุต A3
  }
  val = val/20; // หาค่าเฉลี่ยจากการอ่านค่า 20 ครั้ง

  Temp = (float(val)*0.25) - 20.51 ; // แปลงค่าเป็นอุณหภูมิในหน่วยองศาเซลเซียส
  setTextSize(3); // เปลี่ยนขนาดตัวอักษรเป็น 3 เท่า
  setTextColor(GLCD_YELLOW); // เปลี่ยนสีตัวอักษรเป็นสีเหลือง
  glcd(3,1,"%f",Temp); // แสดงค่าอุณหภูมิด้วยความละเอียดทศนิยม 3 ตำแหน่ง
  setTextColor(GLCD_WHITE); // เปลี่ยนสีตัวอักษรเป็นสีขาว
  setTextSize(2); // เปลี่ยนขนาดตัวอักษรเป็น 2 เท่า
  glcd(6,2,"Celsius"); // แสดงหน่วยองศาเซลเซียส
  delay(500); // หน่วงเวลาก่อนเริ่มต้นการอ่านค่าในรอบใหม่
}

```

คำอธิบายโปรแกรม

ในโปรแกรมนี้หัวใจสำคัญคือ การคำนวณเพื่อเปลี่ยนข้อมูลดิจิทัลที่ได้จากการแปลงแรงดันเอาต์พุตของไอซี MCP9701 เป็นค่าอุณหภูมิในหน่วยองศาเซลเซียส กระบวนการจะเริ่มจากการอ่านและแปลงค่าของแรงดันไฟตรงที่จุดต่อ A3 ซึ่งได้มาจากการทำงานของไอซี MCP9701 มาเก็บไว้ในตัวแปร val จากนั้นนำข้อมูลที่ได้อ่านมาคำนวณด้วยสูตร $Temp = (val \times 0.25) - 20.51$ จากนั้นนำค่าอุณหภูมิที่ได้มาแสดงผลด้วยความละเอียดทศนิยม 3 ตำแหน่ง

โปรแกรมที่ L7-1 : ไฟล์ `microbox_Thermometer.ino` โปรแกรมภาษา C/C++ สำหรับพัฒนากล่องสมองกล IPST-MicroBOX (SE) เป็นเครื่องวัดอุณหภูมิระบบตัวเลขอย่างง่าย