MANUAL
ET-TFT43-EVE

ET- TFT43-EVE

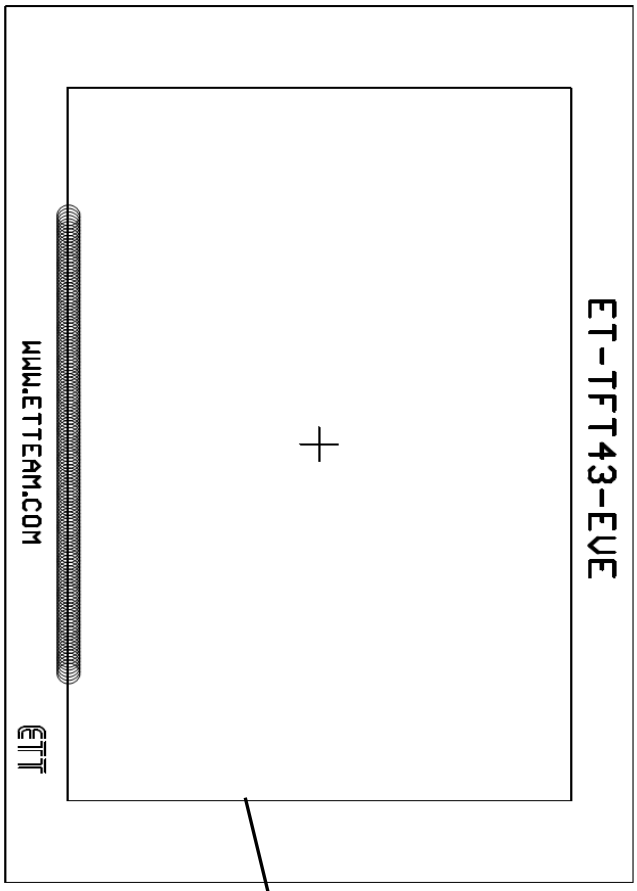
ET-TFT43-EVE เป็นบอร์ด ที่ประกอบด้วย Graphic-LCD, Resistive touch และ Mono Audio Output รวมไว้ด้วยกัน โดยมี Chip FT800 เป็นตัวกลางในการควบคุมการทำงานระหว่าง ผู้ใช้ กับตัวบอร์ด ซึ่ง Chip FT800 นี้ ได้รวมเอาคุณสมบัติของ Display, Audio และ Touch ไว้ใน Chip ตัวเดียว และยังรวมเอาฟังก์ชั่นทางด้าน Graphic Controller , Audio processing และ Resistive Touch Controller ไว้ให้ด้วยซึ่งทำให้ง่ายต่อการใช้งาน ในส่วนของการ Interface กับ MCU จะใช้การ Interface แบบ SPI โดย Pin SPI ของบอร์ดสามารถรองรับ TTL Level 3.3V-5V ได้ ดังนั้นจึงสามารถนำไปเชื่อมต่อกับ MCU 5 V หรือ 3.3 V ได้....

1. คุณสมบัติของบอร์ด ET-TFT43-EVE

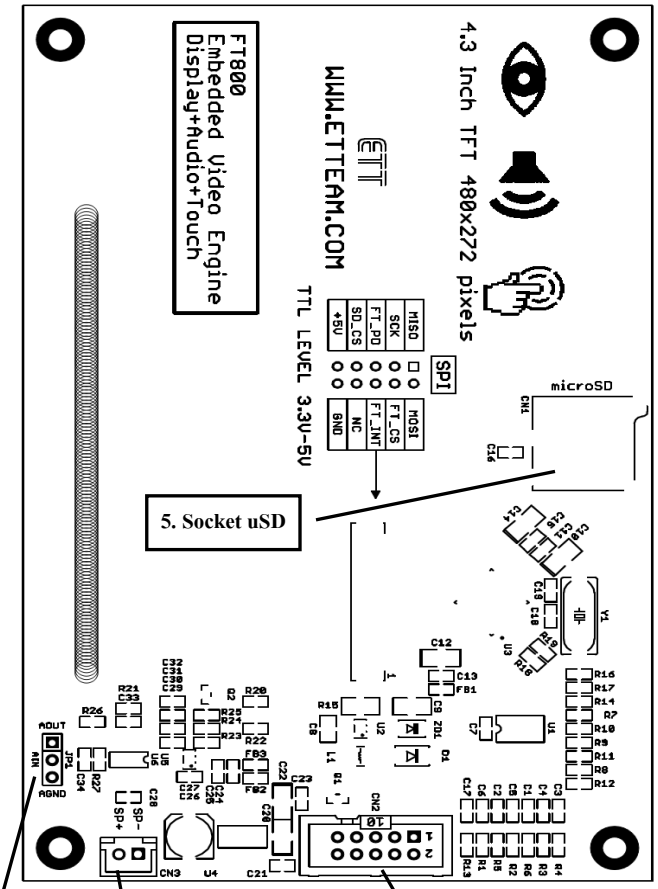
- ใช้ Single Chip Control เบอร์ FT800 ซึ่งได้รวมเอา Function การทำงานทางด้าน Graphic Control ,Audio processing และ Resistive Touch Controller เข้าไว้ใน Chip ตัวเดียว ทำให้ผู้ใช้สามารถควบคุมการทำงานต่างๆของบอร์ดได้ง่าย ผ่าน Function เหล่านี้
- Display 4.3” TFT แบบ WQVGA + Touch Screen แบบ Resistive Touch
- Display Dimension W x H x D = 105.60 x 67.3 x 4.0 mm
- ความละเอียดของหน้าจอ 480 x 272 Pixel
- ความละเอียดของสีแสดงได้ 262 K Color , 18 bit interface (RGB-6,6,6 : ใช้ตาราง code สี แบบ 24 bit ในการเขียนโปรแกรม)
- รองรับการเล่นเสียงจาก 2 Audio Source ได้แก่ 1) Sound Synthesizer : คือ Sound Effect ซึ่งถูก Built-in เก็บไว้ใน ROM ของ FT800 โดย มีให้เลือกเล่น 58 Sound เช่น เสียง Switch , Bell 2) Audio Playback : คือการเล่น Audio File แบบ Mono ใน Format 8-bit PCM , 8-bit uLaw และ 4-bit IMA-ADPCM โดย Audio File ที่จะเล่น จะต้องส่งมาจาก MCU ภายนอกและนำมาเก็บไว้ที่ Memory RAM_G ของ FT800 ก่อน
- สามารถ Control ปรับความดังของเสียงได้ด้วยการกำหนดค่า Volume ให้กับ Register ภายในของ FT800
- On Board Mono Audio Output และ Power Amp 1 W. พร้อมขั้วต่อ Speaker (ใช้กับลำโพงขนาด 8 Ω / 0.4W แยกขายเป็น Option)
- การ Interface ระหว่าง MCU และบอร์ดเป็นแบบ Serial SPI 4 สาย รองรับความเร็ว Clock สูงสุด 30 MHz
- ใช้จำนวน I/O ในการ Interface 5 Pin หรือ 7 Pin เมื่อใช้ INT และ Micro SD Card
- I/O Pin ที่ใช้ในการ Interface รองรับไฟ 3.3V และ 5V tolerant
- สามารถปรับ Dimming Control LED Backlight ได้ด้วย PWM โดยการกำหนดค่าความถี่ หรือ Duty ให้กับ Register ภายในของ FT800
- มี Socket สำหรับใส่ Micro SD Card เพื่อใช้เก็บข้อมูล File ต่างๆ โดยใช้ MCU จากภายนอกในการ Control ซึ่งจะแยกการ Control คนละส่วนกันกับ Display โดยใช้ขาสัญญาณ SPI ร่วมกัน แต่ใช้ขา CS แยกกันเพื่อเลือกการ Control ระหว่าง Micro SD กับ Display
- ไฟเลี้ยงบอร์ด DC +5V

2. ลักษณะและโครงสร้างของบอร์ด ET-TFT43-EVE

รูปด้านหน้าบอร์ด ET-TFT43-EVE



รูปด้านหลังบอร์ด ET-TFT43-EVE



1. LCD+Touch | 2. Jumper | 3. Con-Speaker | 4. SPI-I/O Interface

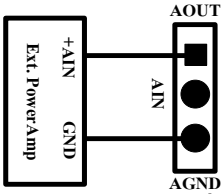
- 1. LCD+Touch : เป็นพื้นที่ของจอ LCD TFT 4.3” ความละเอียด 480 x 272 Pixel โดยด้านบนของจอจะถูกปิดทับด้วยแผ่น Touch Screen แบบ Resistance

- 2 . Jumper : ทำหน้าที่เชื่อมต่อระหว่างสัญญาณเสียง Analog output ของ FT800 (Audio_L) กับ Input ของภาค Power Amp เพื่อขยายสัญญาณเสียงออกไปที่ Con-Speaker โดย Set Jumper มาทางด้าน AOUT และถ้าต้องการตัดสัญญาณเสียงออกจาก Con-Speaker ต้อง Set Jumper มาทางด้าน AGND และอีกหน้าที่หนึ่งคือเมื่อถอด Jumper ออก ผู้ใช้สามารถดึงสัญญาณเสียง Output ที่ Pin AOUT และ AGND ไปต่อเข้าภาคขยายด้านนอกได้

ต่อสัญญาณเสียงไปที่ Con-Speaker

ตัดสัญญาณเสียงออกจาก Con-Speaker

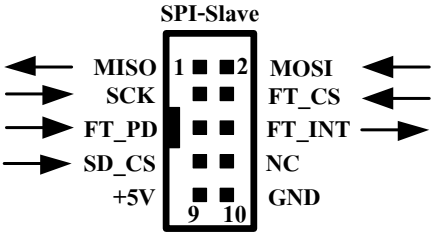
รูปแสดงการ Set Jumper



รูปแสดงการต่อสัญญาณเสียงไปเข้าเครื่องขยายภายนอก

- 3. Con-Speaker : เป็นขั้วต่อ Speaker แบบ Mono สามารถนำลำโพงขนาด 8 Ω / 0.4 W มาต่อได้โดยตรง ซึ่งทางบริษัทมีจำหน่ายเป็น Option เพื่อให้สัญญาณเสียงจากบอร์ดออกมาที่ลำโพงจะต้อง Set Jumper มาทางด้าน AOUT ด้วย

- 4. SPI-I/O Interface : เป็น Connector แบบ บล็อก 10 Pin ใช้สำหรับ Interface แบบ SPI ระหว่าง MCU กับ บอร์ด ET- TFT4.3EVE เพื่อให้ MCU ส่งคำสั่งในการควบคุมการทำงานของบอร์ด รวมทั้งเป็นขั้วต่อไฟเลี้ยง + 5V ให้กับบอร์ดด้วยโดยมีการจัดเรียงขาสัญญาณต่างๆดังนี้



รูปแสดง Port Pin สำหรับใช้ Interface

<u>หมายเหตุ</u> ขาสัญญาณ SPI , FT_ INT และ FT_PD รองรับไฟ 3.3V และ tolerant ไฟ 5V ได้

<u>ตารางที่2.1 แสดงรายละเอียด PIN สำหรับใช้ Control LCD</u>

PIN-No.	Name	Type	หน้าที่
1	MISO	O	ขา Serial Data Slave Out ใช้ส่งข้อมูลไปให้ MCU สำหรับ SPI LCD/micro SD
2	MOSI	I	ขา Serial Data Slave In ใช้รับข้อมูล หรือ คำสั่งจาก MCU สำหรับ SPI LCD/micro SD
3	SCK	I	ขา Clock Input ทำงานขอบขาขึ้น ปกติควรเป็น 0 ใช้สำหรับ SPI LCD/micro SD
4	FT_CS	O	ขา Chip-Select , Active Low (ปกติเป็น 1) ใช้งานเมื่อต้องการติดต่อกับ LCD
5	FT_PD	I	ขา Power Down Input , Active Low (ปกติเป็น 1) สำหรับ Control Power Down LCD
6	FT_INT	OP	ขา ส่งสัญญาณ Interrupt จาก LCD ไปยัง MCU ซึ่งเป็น Open Drain Output , Active Low ซึ่งปกติที่บอร์ด ได้ทำการ Pull Up ไว้ให้แล้ว สามารถต่อใช้งานได้เลย
7	SD_CS	I	ขา Chip-Select , Active Low (ปกติเป็น 1) ใช้งานเมื่อต้องการติดต่อกับ micro SD Card
8	NC	-	ไม่ใช้งาน
9	+5V	P	ขา Power Supply +5V สำหรับเลี้ยงบอร์ด
10	GND	P	ขา Ground ของบอร์ด

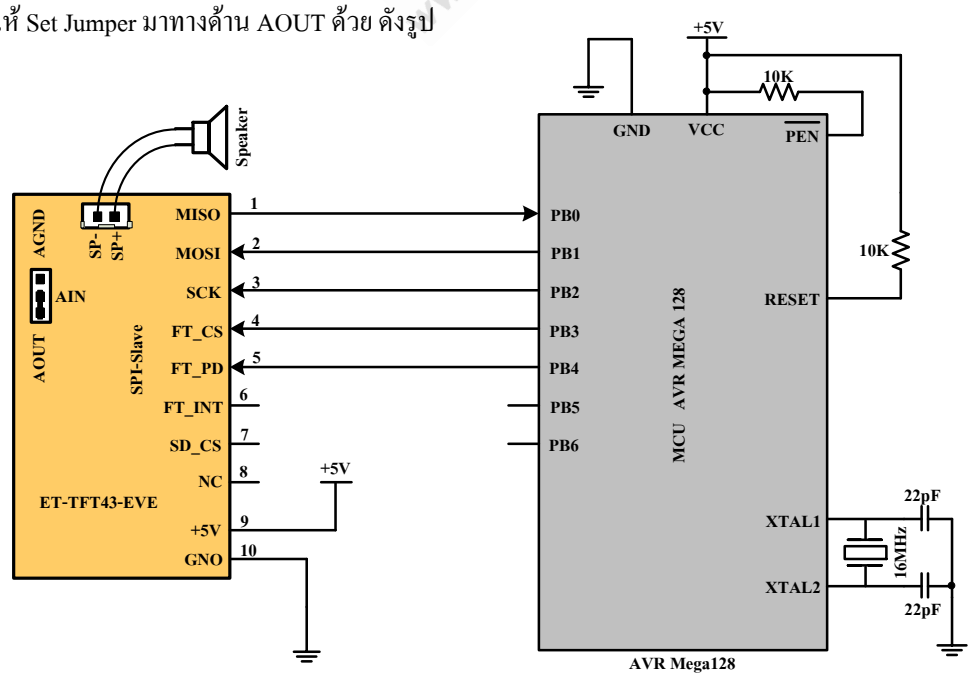
I = Input , O = Output , OP = Open Drain , P = Power

- 5. Socket uSD : เป็น Socket สำหรับใส่ micro SD Card เพื่อใช้เก็บไฟล์รูปภาพหรือไฟล์เสียงไว้สำหรับแสดงผล หรือ เล่นเสียง

3. การนำบอร์ด ET-TFT43-EVE ไปต่อใช้งานร่วมกับ MCU

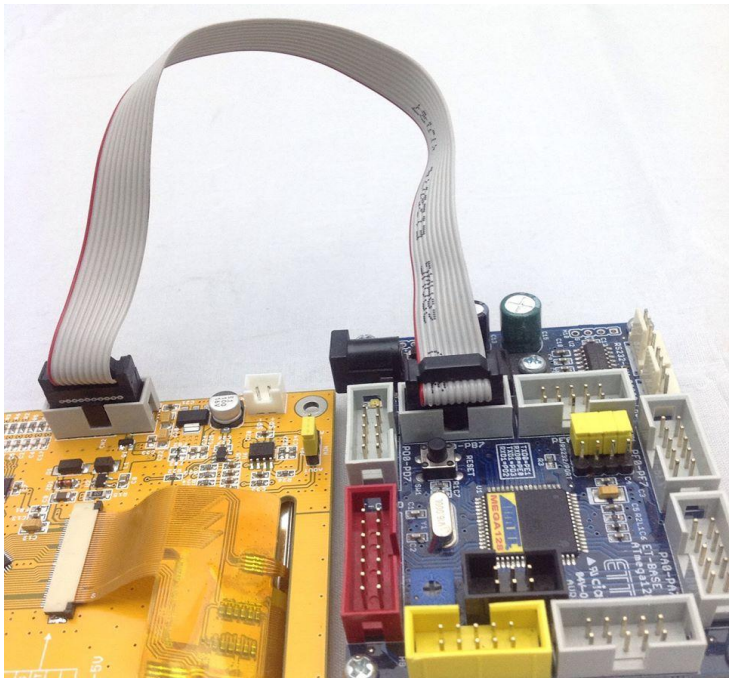
สำหรับการต่อใช้งานบอร์ด ET-TFT43-EVE ที่จะกล่าวถึงต่อจากนี้จะรองรับกับตัวอย่างที่ทางอีทีทีเขียนมาให้ โดยใช้ MCU AVR MEGA128, PIC 18F8722 ซึ่งวงจรการต่อ MCU ที่แสดงในรูปตัวอย่างด้านล่างนี้สามารถนำไปดัดแปลงเพื่อต่อกับ MCU เบอร์อื่น หรือ ตระกูลอื่นๆได้ โดยวงจรในการต่อใช้งานนี้ จะเป็นการ Interface แบบ Serial SPI ซึ่งจะไม่มีการต่อใช้งานในส่วนของ micro SD Card และ การใช้งาน Interrupt ดังนั้น เราจะไม่ใช้ขา SD_CS และ FT_INT

3.1) การต่อ ET-TFT43-EVE กับ MCU AVR Mega 128 แบบ SPI : ในส่วนของ Speaker ถ้าไม่ต้องการใช้งานเสียงก็ไม่จำเป็นต้องต่อ แต่ถ้าต้องการใช้งาน ให้ Set Jumper มาทางด้าน AOUT ด้วย ดังรูป

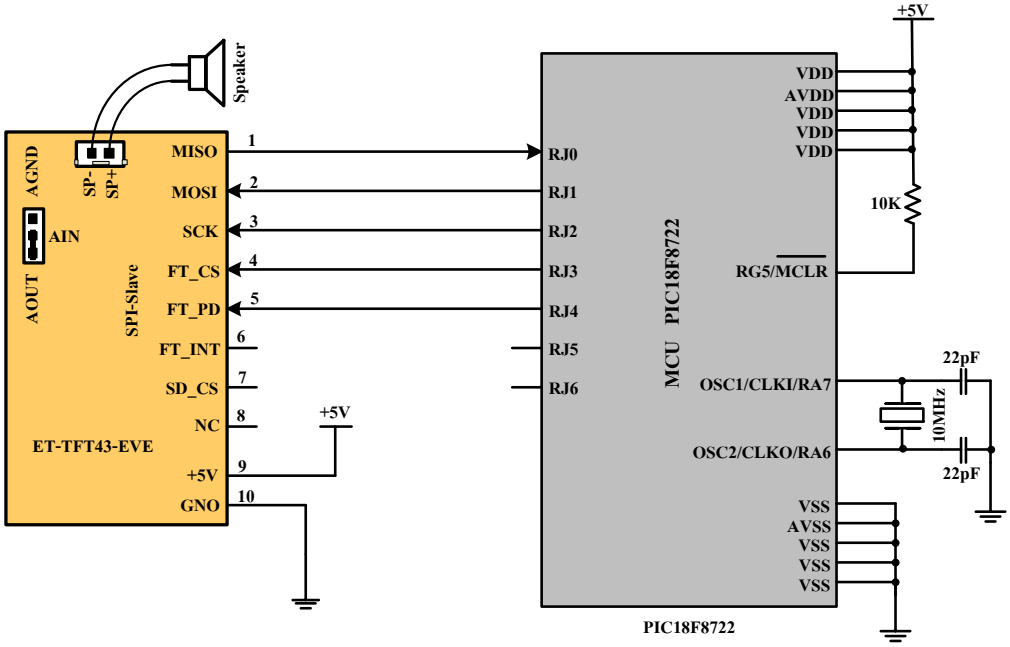


รูปแสดงการต่อ ET-TFT43-EVE กับ MCU AVR Mega 128 แบบ SPI

ในกรณีที่ผู้ใช้นำบอร์ด ET-TFT43-EVE ไปต่อใช้กับ บอร์ด MCU ของ ETT รุ่น ET-Base AVR ATmega128 r3 และใช้ขา Port B ตามรูปข้างต้น ผู้ใช้สามารถใช้สายแพร์แบบบล็อก 10 Pin เพื่อเชื่อมต่อระหว่างบอร์ดทั้งสองได้โดยตรง ดังรูปด้านล่าง

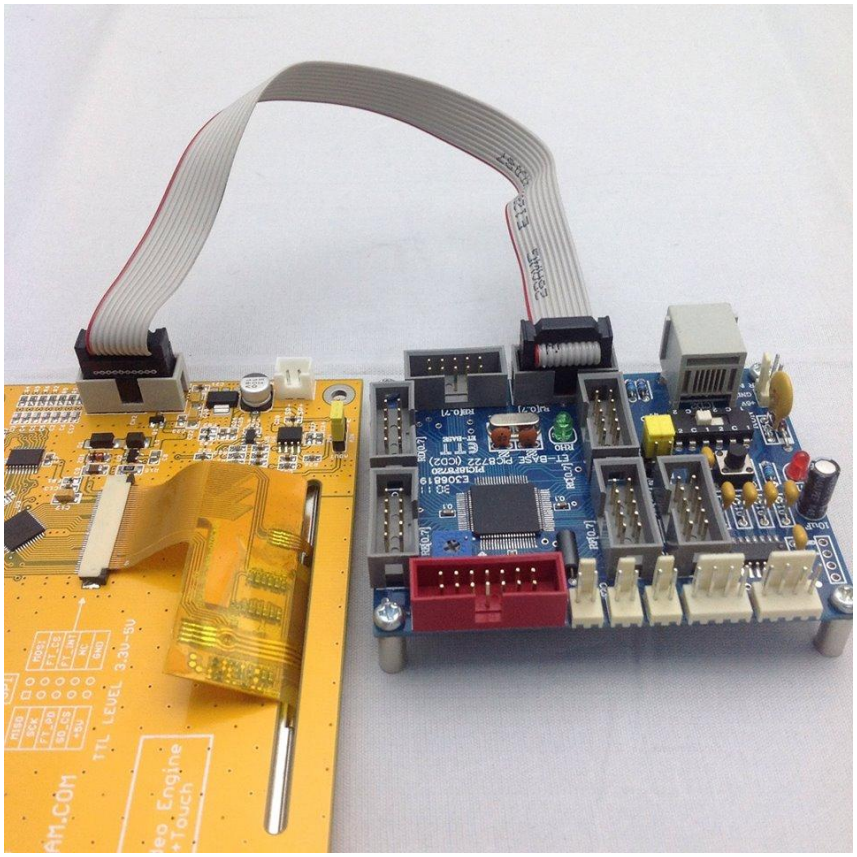


3.2) การต่อ ET-TFT43-EVE กับ MCU PIC18F8722 แบบ SPI : ในส่วนของ Speaker ถ้าไม่ต้องการใช้งานเสียงก็ไม่จำเป็นต้องต่อ แต่ถ้าต้องการใช้งาน ให้ Set Jumper มาทางด้าน AOUT ด้วยดังรูปด้านล่าง



รูปแสดงการต่อ ET-TFT43-EVE กับ MCU PIC18F8722 แบบ SPI

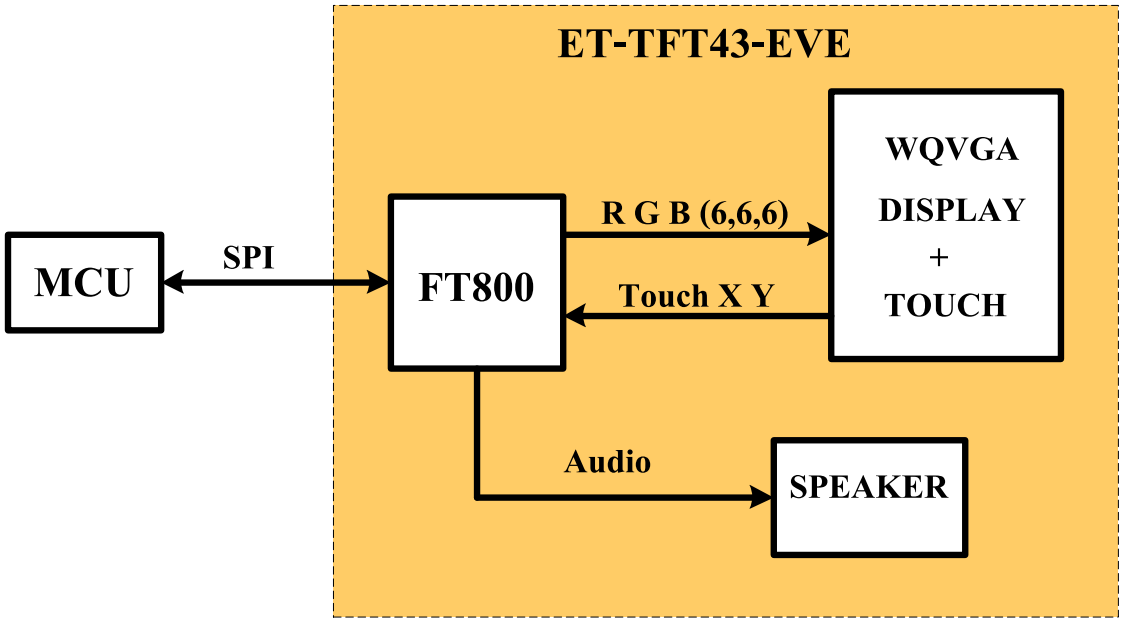
ในกรณีที่ผู้ใช้นำบอร์ด ET-TFT43-EVE ไปต่อใช้กับ บอร์ด MCU ของ ETT รุ่น ET-Base PIC8722(ICD2) และใช้ขา Port RJ ตามรูปข้างต้น ผู้ใช้สามารถใช้สายแพรแบบบล็อก 10 Pin เพื่อเชื่อมต่อระหว่างบอร์ดทั้งสองได้โดยตรง ดังรูปด้านล่าง



4. การทำงาน และการ Control บอร์ด ET-TFT43-EVE เบื้องต้น

4.1 การทำงานเบื้องต้น

สำหรับบอร์ด ET-TFT42-EVE นี้ประกอบไปด้วย 3 ส่วนหลักๆคือ Display ,Touch Screen และ Audio โดยมี Chip FT800 เป็นตัวกลางในการประมวลผลคำสั่งจากผู้ใช้และนำคำสั่งที่ได้ไปควบคุมการทำงานในส่วนต่างๆตามคำสั่ง ซึ่งสรุปเป็นบล็อกการติดต่อการทำงานได้ดังรูปด้านล่าง



รูปแสดงบล็อกการติดต่อการทำงานของ ET-TFT43-EVE

ภาพรวมการทำงาน โดยปกติแล้วสำหรับบอร์ด ET-TFT43-EVE นี้จะมีฟังชั่น Library สำเร็จรูปฝังอยู่ใน Chip FT800 แล้ว ซึ่งใน Chip FT800 นี้แบ่งเป็น 2 ส่วนใหญ่ๆคือ ส่วนที่ติดต่อกับผู้ใช้ซึ่งจะให้ผู้ใช้เข้าถึงได้ในส่วนที่เป็น Register และ Ram ของ FT800 ดังนั้นคำสั่ง หรือ Data ที่ผู้ใช้ส่งมาจะต้อง Write มาเก็บไว้ในพื้นที่ Register และ Ram ก่อน และอีกส่วนหนึ่งคือส่วน Control Display , Touch และ Audio โดยในส่วนนี้ ตัว FT800 จะเป็นตัว Control ให้เอง โดยจะนำคำสั่งหรือข้อมูลที่ได้จากผู้ใช้ซึ่งถูกเก็บไว้ที่ Register และ RAM มาทำการประมวลผล แล้วจึงนำผลที่ได้ไปควบคุมการทำงานของระบบอีกทีหนึ่ง

ดังนั้นในการเขียนโปรแกรมก็จะมี 2 ส่วนที่เราต้องศึกษาคือ การ อ่าน/เขียน data ไปยัง Register ของ FT800 และอีกส่วนหนึ่งคือ การ อ่าน/เขียน คำสั่ง ด้วยฟังก์ชันต่างๆ ที่มีให้ ไปเก็บไว้ยัง Ram ของ FT800 ซึ่งผู้ใช้สามารถดูการทำงานและการใช้งานของ Register และฟังก์ชั่นต่างๆ ได้จาก Data sheet “ FT800 Programmers Guide.pdf ” และ “ DS_FT800.pdf ” หน้า 37 สำหรับตารางสรุป Register FT800

สำหรับการ อ่าน/เขียน Register ต่างๆของ FT800 นั้น ผู้ใช้สามารถระบุตำแหน่ง Address ของ Register นั้นๆ และทำการ Read/Write Register นั้นๆได้โดยตรง ตัว FT800 จะอ่านค่าไปใช้งานทันที แต่ในส่วนของการ อ่าน/เขียน โดยใช้ฟังก์ชันคำสั่งต่างๆนั้น จะเป็นการเขียนคำสั่งในลักษณะ Routine ซึ่งใน Routine หนึ่งๆอาจประกอบไปด้วยหลายๆคำสั่ง โดยในแต่ละ Routine จะต้องมีคำสั่ง Start และคำสั่งจบ Routine ซึ่งคำสั่งต่างๆใน Routine ที่เขียนนี้จะต้องถูก Write เข้าไปเรียงเก็บไว้ยังหน่วยความจำ Ram ของ FT800 ดังนั้นเวลาเขียนคำสั่งต่างๆใน Routine ใดๆก็ตาม ผู้ใช้ต้องอ้าง Address Ram ที่จะเขียนคำสั่งไปเก็บก่อนแล้วจึงตามด้วยการเขียนคำสั่งหรือข้อมูลตามออกไป หลังจากที่คำสั่งต่างๆของ Routine นั้นถูกเขียนไปเก็บไว้ยัง Ram แล้ว เมื่อตัว FT800 พบคำสั่งจบของ Routine ก็จะทำการ Run Routine นั้นบน Ram อีกทีหนึ่ง โดยการ Run นั้นจะ Run เพียงรอบเดียว ถ้าโปรแกรมใน Routine ที่เขียนมีการ Up date ข้อมูลอยู่ตลอดเวลา ผู้ใช้ก็จะต้องทำการส่งคำสั่งของ Routine นั้น มายัง Ram เรื่อยๆ ตัว FT800 ก็จะ Run โปรแกรมนั้นเรื่อยๆเช่นกัน และสิ่งสำคัญในส่วนของ Display เวลา มีการ Update หน้าจอตัว FT800 จะทำการ Up Date ทั้งหน้าจอ ดังนั้นเวลาเขียนโปรแกรมผู้ใช้ต้องคำนึงเสมอว่าหน้าจอในส่วนที่ไม่มีการ Up Date แต่ต้องการให้แสดงคงไว้ ไม่เปลี่ยนแปลง ผู้ใช้จะต้องส่งคำสั่งเดิมที่ใช้ Set หน้าจอมาด้วย จะส่งเฉพาะคำสั่งที่ต้องการ Up Date ค่าหน้าจอมาอย่างเดียวไม่ได้ ดังนั้นเวลาเขียนโปรแกรม ควรทำฟังก์ชั่นสำหรับ Up Date Display ไว้ด้วย โดย

ลักษณะฟังก์ชั่นที่ใช้ Up date หน้าจอ ก็ควรประกอบไปด้วยคำสั่งต่างๆที่เราต้องการให้ Display แสดงออกมา และส่วนที่ต้องการ Up Date บนหน้าจอก็ให้ใช้การผ่านค่าตัวแปรเข้ามายังฟังก์ชัน เป็นต้น

4.2 หน่วยความจำ RAM ของ FT800

สำหรับในหัวข้อนี้จะกล่าวถึงหน่วยความจำของตัว FT800 ว่ามีกี่ส่วนทำหน้าที่อะไรบ้างและมีการอ้าง Address ใช้งานอย่างไร เนื่องจากมีความสำคัญเพราะเวลาเราเขียนโปรแกรม คำสั่งต่างๆ ที่ส่งไปจะต้องเขียนลงไปในหน่วยความจำ Ram เหล่านี้ แล้วตัว FT800 จะทำการ Run โปรแกรมที่เขียนบน Ram เหล่านี้อีกทีหนึ่ง ซึ่งจะทำให้โปรแกรมตอบสนองต่อผู้ใช้ได้อย่างรวดเร็ว สำหรับ Memory Map ของ FT800 แสดงดังตารางด้านล่าง

ตาราง FT800 Memory MAP

Start Address	End Address	Size	NAME	Description
0x000000	0x03FFFF	256 kB	RAM_G	Main graphics RAM
0x0C0000	0x0C0003	4 B	ROM_CHIPID	FT800 chip identification and revision information: Byte [0:1] Chip ID: "0800" Byte [2:3] Version ID: "0100"
0x0BB23C	0x0FFFFB	275 kB	ROM_FONT	Font table and bitmap
0x0FFFFC	0x0FFFFF	4 B	ROM_FONT_ADDR	Font table pointer address
0x100000	0x101FFF	8 kB	RAM_DL	Display List RAM
0x102000	0x1023FF	1 kB	RAM_PAL	Palette RAM
0x102400	0x10257F	380 B	REG_*	Registers
0x108000	0x108FFF	4 kB	RAM_CMD	Graphics Engine Command Buffer

สำหรับหน่วยความจำและ Register ทั้งหมดใน FT800 นั้น จะมี Address ขนาด 22 Bit หรือคิดเป็น 3 Byte โดยจะมีว่างอยู่ 2 Bit คือ Bit ที่ 22 กับ 23 จะใช้เป็นบิตในการกำหนดคำสั่ง อ่าน/เขียน หน่วยความจำของ Address นั้นๆ โดยถ้าเป็น

00b คือ คำสั่งสำหรับ Read Memory ใน FT800

10b คือ คำสั่งสำหรับ Write Memory ใน FT800

01b คือ คำสั่งสำหรับ Host Command กล่าวคือ ใช้ระบุว่าเป็นคำสั่ง Set up การทำงานของ ตัว FT800

11b คือ ไม่ถูกใช้งาน

โดยผู้ใช้สามารถดูการจัดเรียงบิตในการส่ง Address + Read/Write Memory ได้ในหัวข้อ 4.3 ด้านล่าง

จากตาราง Memory Map ด้านบนหน่วยความจำภายในของ FT800 จะแบ่งออกเป็น 8 ส่วนด้วยกัน ซึ่งแต่ละส่วนจะมีการใช้งานในการเก็บข้อมูลที่จะนำไปใช้งานต่างกันออกไป โดยในที่นี้จะขอกล่าวถึงในส่วนที่จำเป็นหลักๆที่เกี่ยวข้องกับการเขียนโปรแกรมที่ผู้ใช้สามารถเข้าถึงได้และถูกนำมาใช้งานในตัวอย่างของ ETTเท่านั้น หน่วยความจำที่ไม่กล่าวถึงผู้ใช้สามารถดูเพิ่มเติมได้จาก Data Sheetใน CD

- RAM_G : พื้นที่ของ RAM_G นี้จะมีขนาด 256 Kb โดยเริ่มจาก Address 0x000000 -0x03FFFF และใน 1 Address จะเก็บ Data ได้ 1Byte(8-bit) และในการ Write data แต่ละครั้ง Address จะเพิ่มค่าขึ้นเองอัตโนมัติ ดังนั้นการ Write Data ไปที่หน่วยความจำ RAM_G นี้ สามารถ Write Data ได้อย่างต่อเนื่องจนจบ Package ของข้อมูล โดยผู้ใช้อ้าง Address เริ่มต้นแค่ครั้งเดียวเท่านั้น ถ้าจะ Write ข้อมูล Package ต่อไปจึงค่อยอ้าง Address ใหม่ ที่ไม่ทับซ้อนกับ Address สุดท้ายของข้อมูล Package แรก

สำหรับพื้นที่ RAM_G นี้ จะใช้สำหรับเก็บ Data ประเภทไฟล์รูปภาพ ในรูปแบบ Bitmap (*.PNG,*.JPEG) หรือ ไฟล์ Audio ในรูปแบบ PCM , u-Law , IMA ADPCM กล่าวคือ เมื่อผู้ใช้ต้องการจะให้แสดงรูปภาพ หรือ เล่นเสียงใดๆที่ผู้ใช้สร้างขึ้นมาเอง หรือทำ Font ในแบบอื่นๆนอกเหนือจากที่มีให้ในตัว FT800 ผู้ใช้จะต้องนำไฟล์เหล่านั้นมา Convert ให้เป็น Hex Code เสียก่อน ด้วยโปรแกรม

img_cvt.exe(Picture) , fnt_cvt.exe(Font) , aud_cvt.exe(Audio) อย่างใดอย่างหนึ่งตามชนิดไฟล์ จากนั้นเวลาเขียนโปรแกรมก็ให้ MCU Write Data File เหล่านั้นไปเก็บไว้ที่ RAM_G ของ FT800 เสียก่อน โดยผู้ใช้จะต้องจำว่า ไฟล์ที่จะเรียกใช้งานแต่ละไฟล์ มี Address เริ่มต้นไฟล์อยู่ที่ Address เท่าใดใน RAM_G ด้วย จากนั้นถึงจะส่งคำสั่ง BEGIN(BITMAPS) เพื่อสั่งให้แสดงไฟล์ที่ Write มาเก็บไว้ใน RAM_G ออก Display เป็นต้น สำหรับตัวอย่างการ Write Data ไปเก็บยัง RAM_G สามารถดูได้ใน ตัวอย่างที่ 4 ของ ETT (EX4_TFT43_Bitmap) ซึ่งอยู่ในฟังก์ชัน FT800memWrBitmap(…) และการสั่งอ่าน Data ใน RAM_G ไปใช้งานดูได้ในฟังก์ชัน Bitmap_DL(...) เป็นต้น โดยการ Write Data ไปเก็บไว้ยัง RAM_G จะต้องเป็นไปตามกฎการเข้าถึงหน่วยความจำ RAM ตามหัวข้อด้านล่างด้วย

-RAM_DL : พื้นที่ของ RAM_DL นี้จะมีขนาด 8 Kb โดยเริ่มจาก Address 0x100000 - 0x101FFF โดยใน 1 Address จะเก็บ Data ได้ 1Byte(8-bit) และในการ Write data แต่ละครั้ง Address จะเพิ่มค่าขึ้นเองอัตโนมัติ โดย RAM_DL นี้จะใช้สำหรับเก็บคำสั่ง DL_CMD จากผู้ใช้ที่ส่งมาทาง MCU ซึ่งเป็นคำสั่งเกี่ยวกับ Display Graphics(ดูคำสั่งได้ใน Data Sheet “FT800 Programmers Guide.pdf ” หน้า 78) โดยคำสั่งที่ถูกเก็บใน RAM_DL จะถูกนำไปประมวลผลและแสดงผลบน Display โดยตรง สำหรับคำสั่ง DL_CMD นี้สามารถจะเก็บได้ทั้งใน RAM_DL และ RAM_CMD ดังนั้นเวลาส่งคำสั่งผู้ใช้ต้องอ้าง Address สำหรับเก็บคำสั่งด้วย โดยรูปแบบของคำสั่ง DL_CMD จะมีขนาด 4 Byte (32 bit) เสมอ ดังนั้นเวลา Write คำสั่งไปเก็บยัง RAM_DL แล้ว 1 คำสั่ง ก่อนที่จะส่งคำสั่งที่2 ผู้ใช้จะต้องบวก Offset Address เพิ่มไปอีก 4 byte สำหรับคำสั่งที่ 2 เช่น คำสั่งแรกเริ่มที่ Address 0x100000 คำสั่งที่ 2 ก็จะต้อง Write ไปเก็บไว้ที่ Address 0x100004 เป็นต้น โดยเวลาใช้งานคำสั่ง DL_CMD จะต้อง Write คำสั่งในลักษณะ Routine โดยจะต้องมีคำสั่งเริ่มต้นและจบ Routine เสมอ ภายใน Routine จะประกอบไปด้วยคำสั่ง DL_CMD กี่คำสั่งก็ได้

-REG_* : เป็นพื้นที่หน่วยความจำสำหรับ Register ต่างๆของ FT800 มีขนาด 380 Byte โดยเริ่มจาก Address 0x102400 - 0x10257F ซึ่ง Register ทั้งหมดของ FT800 จะแบ่งเป็น 5 กลุ่ม คือ Graphics Engine Registers, Audio Engine Registers, Touch Engine Registers, Co-processor Engine Registers และ Miscellaneous Registers

เวลาใช้งานต้องอ้างตำแหน่ง Address ของ Register นั้นๆ แล้วตามด้วยการ Write data เพื่อกำหนดค่าให้กับ Register นั้นๆได้เลย ซึ่งขนาด Byte Data ของ Register แต่ละตัวจะแตกต่างกันไปตามการใช้งาน (ดู Register ใช้งานได้ใน Data Sheet “FT800 Programmers Guide.pdf ” หน้า 33)

-RAM_CMD : พื้นที่ของ RAM_CMD นี้จะมีขนาด 4 Kb โดยเริ่มจาก Address 0x108000 - 0x108FFF และใน 1 Address จะเก็บ Data ได้ 1Byte(8-bit) และในการ Write data แต่ละครั้ง Address จะเพิ่มค่าขึ้นเองอัตโนมัติ โดย RAM_CMD นี้จะใช้สำหรับเก็บคำสั่ง DL_Command (DL_CMD) และ Co-processor Command (CO_CMD) ซึ่ง CO_CMD นี้จะเป็นคำสั่งในลักษณะของฟังชั่นสำเร็จรูป ซึ่งเป็นคำสั่งเกี่ยวกับ Display Graphics เช่นกัน (ดูคำสั่งได้ใน Data Sheet “FT800 Programmers Guide.pdf ” หน้า 139) เช่นคำสั่ง TEXT , BUTTON เป็นต้น ลักษณะขนาดของคำสั่งจะมีขนาด 4 Byte (32 bit) แล้วตามด้วย Data Parameter ของคำสั่งนั้นๆซึ่งจะไม่เท่ากัน ดังนั้นใน 1 คำสั่งของ CO_CMD อาจมีมากกว่า 4 Byte ซึ่งต่างจาก DL_CMD (คำสั่ง + Data) จะมีขนาดแค่ 4 Byte

โดยปกติแล้วเราจะเขียนโปรแกรมมาเก็บไว้ที่ RAM_CMD เนื่องจากมีคำสั่งที่เป็นฟังก์ชั่นสำเร็จรูปให้ใช้งานและมีความยืดหยุ่นในการใช้งานคือ รองรับทั้งคำสั่ง DL_CMD และ CO_CMD เราไม่สามารถใช้คำสั่งในส่วนของ CO_CMD เขียนไปเก็บไว้ที่ RAM_DL ได้ โดยคำสั่งที่เก็บใน RAM_CMD จะถูก Copy เข้าไปยัง RAM_DL อีกทีหนึ่ง ก่อนที่จะถูก Run แสดงผลที่ Display โดยเวลาใช้งานคำสั่ง CO_CMD จะต้อง Write คำสั่งในลักษณะ Routine เช่นกัน ซึ่งจะต้องมีคำสั่งเริ่มต้นและจบ Routine เสมอ โดยดู Routine การเขียนได้ในหัวข้อต่อไป

<u>กฎในการเข้าถึงหน่วยความจำ RAM จากตารางด้านบน มีดังนี้</u>

ในการที่ผู้ใช้จะ Write คำสั่ง และข้อมูลต่างๆ ไปเก็บไว้ในพื้นที่หน่วยความจำนั้นจะมีหลักเกณฑ์ที่ต้องปฏิบัติตามหัวข้อด้านล่างนี้มิเช่นนั้นแล้วโปรแกรมที่เขียนขึ้น เมื่อถูกเขียนเข้าไปไว้ยัง Ram แล้ว FT800 จะ Run โปรแกรมออกมาผิดพลาด หรือ โปรแกรมอาจไม่ทำงานได้ ยกเว้นในส่วนตำแหน่ง Address Register (REG_*) เวลา อ่านเขียน Register ผู้ใช้สามารถอ้าง Address ตำแหน่งของ Register ที่ต้องการ อ่านเขียนได้เลย ไม่จำเป็นต้องคำนึงถึงกฎ ที่จะกล่าวดังต่อไปนี้

❖ ในการเข้าถึง Address หน่วยความจำทั้งหมด ยกเว้น Register เมื่อจะ Write Data หรือ คำสั่งไปเก็บไว้ยัง หน่วยความจำเหล่านี้ เวลาอ้างAddress ในการเก็บคำสั่งของแต่ละคำสั่งจะต้องอ้าง Address ในตำแหน่งที่หารด้วย 4 ลงตัว เช่น ถ้าเป็นใน RAM_DL ก็ให้อ้าง Address 0x100000,0x100004,0x100008…12…16.. เป็นต้น ถ้าเป็นใน RAM_CMD ก็ให้อ้างที่ Address 0x108000 ,108004…8…12..เป็นต้น ในกรณีที่คำสั่งที่ Write ไปยังหน่วยความจำเมื่อจบคำสั่งนั้นแล้ว Address ที่ต่อจาก Address สุดท้ายของคำสั่งนั้นไม่ได้เริ่มต้น Address ที่หารด้วย 4 ลงตัว ให้ Write ค่า 0 ต่อจาก Address สุดท้ายของคำสั่งนั้นไปจนกระทั่ง Address เริ่มต้นสำหรับคำสั่งที่ 2 หารด้วย 4 ลงตัว เช่น Address ของคำสั่งจบลงที่ Address 0x108005 ผู้ใช้ก็ควร Write ค่า 0 ไปที่ Address 0x108006 และ 0x108007 เพื่อว่าคำสั่งที่สอง จะได้เริ่มที่ Address 0x108008 เป็นต้น ถ้าไม่ Write ค่า 0 เพิ่มเข้าไป เมื่อจะ Write คำสั่งที่ 2 ต่อจากคำสั่งแรกผู้ใช้ก็สามารถอ้าง Address เริ่มต้นสำหรับคำสั่งใหม่ได้โดยให้อ้าง Address เริ่มต้นที่ 0x108008 เลยก็ได้ ซึ่งหารด้วย 4 ลงตัว

ในกรณี Write data เข้าไปเก็บยัง RAM_G ก็เช่นกัน Data ของ ไฟล์ต่างๆ ก็จะต้องเริ่มต้นเก็บที่ Address ที่หารด้วย 4 ลงตัวเช่นกัน ยกตัวอย่างเช่น เริ่มต้นที่ Address 0x000000 และจบที่ Address 0x000502 เมื่อ Write ไฟล์แรกจบแล้ว Address ใน RAM_G จะจบลงที่เท่าไหร่ก็แล้วแต่ Data ไฟล์ที่ 2 ที่จะ Write ไปเก็บต้องเริ่มต้นที่ Address ที่ หารด้วย 4 ลงตัว เช่น เริ่มต้นที่ Address 0x001000 โดยอย่าให้ทับกับ Address สุดท้ายที่ใช้เก็บ Data ของไฟล์แรก

❖ สำหรับ Register จะเห็นว่า Bit Data ของ Register แต่ละตัว จะมีจำนวน Bit Data ในการอ่านเขียน ไม่เท่ากันอาจจะไม่เต็ม Byte หรือเกิน Byte เวลา Read/Write ให้อ่านมาเป็น Byte หรือ เป็น Word เลย เช่น Register REG_HSYNC มีขนาด 10-bit เวลา Read ก็ควร Read แบบ 16 bit Data Read เป็นต้น

4.3 การ Read/Write Data โดย Interface แบบ SPI

สำหรับหัวข้อนี้จะกล่าวถึงการ Interface แบบ SPI ระหว่าง MCU และ FT800 ว่ามีรูปแบบกระบวนการ Read/Write Data อย่างไรอะไรบ้าง ตามที่เคยกล่าวมาแล้วข้างต้นว่าการ Read/Write Data ของ MCU ไปยัง FT800 นั้นจะเป็นการ Read/Write ข้อมูลไปยังหน่วยความจำ หรือ Register ต่างๆของ FT800 โดยรูปแบบของการ Read/Write Data ไปยัง FT800 นั้นจะมี รูปแบบการ Read/Write อยู่ 3 รูปแบบดังต่อไปนี้

1) การ Read Memory : เป็นการส่งคำสั่งจาก MCU ไปยัง FT800 เพื่ออ่าน Data จาก Register ต่างๆ ของ FT800 โดยรูปแบบเริ่มต้น MCU จะต้อง Write 2 bit แรกไปเป็น 0 เพื่อระบุเป็นคำสั่งอ่าน แล้วตามด้วย 22-bit Address Register ที่ต้องการอ่าน และตามด้วย Dummy 1 Byte ซึ่งให้กำหนดเป็น 0 หลังจากส่ง Dummy Byte แล้ว FT800 ก็จะส่ง Data Byte0 กลับมาเป็น Byte แรก ซึ่งสรุปได้ดังตารางด้านล่าง

ตารางแสดง Memory Read Transaction (SPI)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
SPI Active	Set SPI_SS Low (active)								
Command/Address	0	0	Address[21:16]						Write Address Register
Address	Address[15:8]								
Address	Address[7:0]								
Dummy	Dummy Byte Set to 0								
Byte0	Read Byte 0 , MSB First								Read Data
...	Read Byte ... , MSB First								
Byte n	Read Byte n , MSB First								
SPI Activity	Set SPI_SS High (inactive)								

จากตาราง Read Memory เริ่มต้นเราต้องกำหนดให้ MCU Set Pin SPI_SS ไปเป็น Low ก่อน แล้วเริ่ม Write Byte แรก คือ Byte Command Read (bit 6,7 เป็น 0) + Address Register (bit 21:16) แล้วตามด้วย Address Register Byte2 (bit 15:8) และ Address Register

Byte3 (bit 7:0) และจบด้วย Dummy Byte (write ค่า 0) หลังจาก Dummy Byte ถูกส่งออกไป FT800 ก็จะส่ง Data Byte กลับมาให้ MCU โดย จะส่ง Byte 0 กลับมาเป็น Byte แรก (ส่ง Bit MSB ออกมาก่อน) โดยจำนวน Data Byte ที่ส่งกลับมาก็ขึ้นอยู่กับขนาดของ Register ที่อ่าน สุดท้ายกำหนดให้ MCU Set Pin SPI_SS ไปเป็น High เป็นอันสิ้นสุดกระบวนการอ่าน ซึ่งกระบวนการอ่านนี้ในตัวอย่างของ ETT ก็คือ ฟังก์ชัน FT800memRead8() , FT800memRead16() , FT800memRead32() ซึ่งพารามิเตอร์ที่ต้องผ่านค่าให้ฟังก์ชันคือค่า Address Ram และในแต่ละฟังก์ชันขนาดของ Data Bit ที่ทำการอ่านจะต่างกันไป ดังนั้นเวลาเลือกใช้ฟังก์ชันในการอ่านก็ควรเลือกตามขนาด Data ของ Register ที่จะอ่าน เช่น Register มีขนาด Data ที่ต้องการอ่าน 16 bit ก็ให้เลือกใช้ฟังก์ชัน FT800memRead16() หรือ FT800memRead32() เป็นต้น

2) การ Write Memory : เป็นการส่งคำสั่งจาก MCU ไปยัง FT800 เพื่อเขียน Data ไปยัง Register ต่างๆของ FT800 หรือ เขียนคำสั่ง ฟังก์ชั่นต่างๆ หรือ Data ไปเก็บไว้ยังหน่วยความจำ RAM_G ,RAM_DL หรือ RAM_CMD ของ FT800 โดยลักษณะการส่งคำสั่ง เริ่มต้น MCU จะต้อง Write บิตแรกไปเป็น 1 และบิตที่สองไปเป็น 0 เพื่อระบุเป็นคำสั่งเขียน แล้วตามด้วย 22-bit Address ของ Ram หรือ Register แล้วตามด้วย Data ที่ต้องการจะ Write โดยให้เริ่ม Write data byte0 เป็น Byte แรก ซึ่งสรุปได้ดังตารางด้านล่าง

ตารางแสดง Memory Write Transaction (SPI)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
SPI Active	Set SPI_SS Low (active)								
Command/Address	1	0	Address[21:16]						Write Address RAM ,Register
Address	Address[15:8]								
Address	Address[7:0]								
Byte0	Write Byte 0 , MSB First								Write Data
...	Write Bytes ... , MSB First								
Byte n	Write Byte n , MSB First								
SPI Activity	Set SPI_SS High (inactive)								

จากตาราง Write Memory เริ่มต้นเราต้องกำหนดให้ MCU Set Pin SPI_SS ไปเป็น Low ก่อน แล้ว Write Byte แรก คือ Byte Command Write (bit 7=1,bit6=0) + Address (bit 21:16) แล้วตามด้วย Address byte2 (bit 15:8) และ Address byte3 (bit 7:0) จากนั้นก็เริ่ม Write Data Byte0 (ส่ง bit MSB ออกมาก่อน) ออกไปเป็น Byte แรก และ Write Byte ต่อไปเรื่อยๆจน จบในส่วนของ Command นั้นๆ แล้วกำหนดให้ MCU Set Pin SPI_SS กลับมาเป็น High เป็นอันสิ้นสุดกระบวนการ Write คำสั่งนั้นๆไปยัง RAM หรือ Register สำหรับกระบวนการเขียนนี้ในตัวอย่างของ ETT ก็คือ ฟังก์ชัน FT800memWrite8(), FT800memWrite16(), FT800memWrite32() โดยสามารถเลือกใช้งานคำสั่ง Write ได้ตามขนาดของ Data หรือ Command ที่จะสั่ง Write ซึ่งพารามิเตอร์ที่ต้องผ่านให้ฟังก์ชันคือ Address Ram และ Data ตามขนาด Bit ของคำสั่ง

<u>หมายเหตุ</u> ในการ Write Data หรือคำสั่งต่างๆ ไปเก็บยังหน่วยความจำ RAM นั้น การกำหนด Address จะต้องเป็นไปตามกฎที่กล่าวไว้ข้างต้นคือ Address เริ่มต้นสำหรับเก็บคำสั่งใดๆจะต้องหารด้วย 4 ลงตัว จากฟังก์ชัน Write Memory ของอีทีที ที่เขียนขึ้น จะสามารถ Write Data หรือคำสั่งได้สูงสุด 4 Byte คือ ฟังก์ชั่น FT800memWrite32() ในกรณีคำสั่งที่จะ Write มีขนาดเกิน 4 Byte ปกติจะเป็นคำสั่งประเภท CO_CMD เช่นคำสั่ง CMD_BGCOLOR() คำสั่งนี้ จะมี Data ที่เป็นรหัสคำสั่ง 4 Byte คือ 0xFFFFFF09 และ Data ที่เป็น พารามิเตอร์ค่าสีอีก 4 Byte รวมคำสั่งนี้มีขนาด 8 byte เวลา จะ Write คำสั่งนี้ก็ให้ใช้ฟังก์ชั่น FT800memWrite32() Write คำสั่งนี้ 2 ครั้ง โดยครั้งแรกให้ Write รหัสคำสั่งไปก่อนโดย Address เริ่มต้นสำหรับเก็บคำสั่งจะต้องหารด้วย 4 ลงตัว สมมติ Write ไปเก็บที่ RAM_CMD Address เริ่มต้นก็อาจจะเป็น 0x108000 ซึ่งหารด้วย 4ลงตัว ก็จะได้ FT800memWrite32(0x108000,0xFFFFFF09) ครั้งที่2ให้ Write พารามิเตอร์ตามไป โดย Address ที่จะอ้างไม่จำเป็นต้องหารด้วย 4ลงตัว เนื่องจาก เป็นส่วนของ Data ที่อยู่ในชุดคำสั่งเดียวกัน เพียงแต่อ้าง Address ให้ต่อจากการ

Write ในครั้งแรกก็พอ ก็จะได้ FT800memWrite32(0x108004,ค่าพารามิเตอร์) เนื่องจาก Data ที่ Write ในครั้งแรกมีขนาด 4 Byte ดังนั้น Address สำหรับเก็บ Data ชุดที่ 2 ก็จะต้องเริ่มที่ Address 0x108004 ถ้ามีการส่งคำสั่ง Write ต่อจากคำสั่งชุดนี้อีก Address เริ่มต้นของคำสั่งใหม่ จะต้องกำหนด Address ที่หารด้วย 4 ลงตัวอีกเช่นเดิม ***จำไว้เสมอว่าทุกครั้งที่ Write คำสั่งไปยัง RAM Address เริ่มต้นสำหรับเก็บคำสั่งนั้นจะต้องหารด้วย 4 ลงตัวเสมอ ส่วน Address สำหรับพารามิเตอร์ของคำสั่งนั้นๆไม่จำเป็นต้องหารด้วย 4 ลงตัว แต่ต้อง Write ต่อจาก Address สุดท้ายของคำสั่งนั้นๆ

3) การ Write Host COMMAND : เป็นการส่งคำสั่งจาก MCU ไปยัง FT800 เพื่อ Set ค่าการทำงานให้กับ FT800 ได้แก่ Set Power Mod ,Set Clock . ให้กับ FT800 เป็นต้น สำหรับการส่งคำสั่ง ไปยัง Host Command นั้น จะใช้เฉพาะ ตอน Initial LCD เท่านั้น โดยลักษณะการส่งคำสั่ง เริ่มต้นใน Byte แรก MCU จะต้อง Write บิตแรกไปเป็น 0 และบิตที่สองไปเป็น 1 เพื่อระบุเป็นคำสั่ง Write host แล้วตามด้วยรหัสคำสั่งอีก 6 บิต จากนั้น Byteที่2 และ 3 ให้ Write ค่า 0x00 ตามไปอีก 2 Byte ซึ่งสรุปได้ดังตารางด้านล่าง

ตารางแสดง Write Host Command Transaction (SPI)

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPI Active	**Set SPI_SS Low (active)**							
Command	**0**	**1**	**Command [5:0]**					
Zero	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Zero	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
SPI Activity	**Set SPI_SS High (inactive)**							

จากตาราง Write Memory เริ่มต้นเราต้องกำหนดให้ MCU Set Pin SPI_SS เป็น Low ก่อน แล้ว Write Byte แรก คือ Byte Command ซึ่งประกอบด้วย bit7 = 0 , bit6 = 1 + Command Host[bit5:0] แล้วตามด้วยการ Write 0x00 ไปใน byte2 และ byte3 จากนั้นก็ให้ MCU Set Pin SPI_SS เป็น High เป็นอันสิ้นสุดกระบวนการ Write Host Command สำหรับกระบวนการเขียนนี้ในตัวอย่างของ ETT ก็คือ ฟังก์ชัน FT800_HostCmdWrite(cmd) ซึ่งพารามิเตอร์ที่ต้องผ่านให้ฟังก์ชันก็คือ Host Command เช่น 0x41 , 0x42 เป็นต้น

สำหรับการ Read/Write Data โดย Interface แบบ SPI นี้ รวมทั้งคำสั่งของ Host Command สามารถดูเพิ่มเติมได้ใน Data Sheet " AN_240 FT800 From the Ground Up.pdf " และ " DS_FT800.pdf."

4.4 การใช้งานคำสั่งและการ Initial Display เบื้องต้น

ในหัวข้อนี้เราจะมาทำความรู้จักเกี่ยวกับรูปแบบการใช้งานคำสั่งเสียก่อน ก่อนที่จะกล่าวถึงกระบวนการเขียนโปรแกรมว่ามีลำดับการเขียนอย่างไร โดยปกติแล้ว การอ่านเขียนในส่วนที่เป็น Register ของ FT800 นั้นจะไม่มีปัญหาอะไร ผู้ใช้สามารถใช้ฟังก์ชั่น Read/Write ตามที่กล่าวไปข้างต้น แล้วอ้าง Address ของ Register ที่ต้องการจะอ่านเขียนได้โดยตรง แต่การใช้งานคำสั่งหรือ ฟังก์ชั่นต่างๆของ FT800 ซึ่งจะเป็นคำสั่งเกี่ยวกับ Graphic Display นั้น จะต้อง Write ไปยังหน่วยความจำ RAM_DL หรือ RAM_CMD ส่วน Data ประเภทไฟล์รูปภาพ ไฟล์เสียง จากภายนอก จะต้อง Write ไปเก็บที่ RAM_G ก่อนจะถูกเรียกใช้งาน ก่อนอื่นเรามาทำความเข้าใจเกี่ยวกับคำสั่งที่ใช้สำหรับ Control Display กันก่อน ดังนี้

❖ ***คำสั่งที่ใช้ Control Display แบ่งเป็น 2 กลุ่ม ดังนี้***

1) Display list command (DL_CMD) : เป็นกลุ่มคำสั่งที่สามารถเขียนไปเก็บยังหน่วยความจำ RAM_DL หรือ RAM_CMD ได้ โดยคำสั่งในกลุ่มนี้จะมีขนาด 4 Byte (32-bit) ซึ่งจะแบ่งเป็น Byte บนสุด(bit 31..24) จะเป็นในส่วนของรหัสคำสั่ง ส่วนอีก 3 Byte ที่เหลือ(bit23..0) จะเป็นส่วนของพารามิเตอร์ของแต่ละคำสั่ง ดังแสดงในตารางด้านล่างซึ่งเป็นรูปแบบ ตัวอย่างของคำสั่ง "CLEAR_COLOR_RGB"

Example DL_CMD : CLEAR_COLOR_RGB

31 24	23 16	15 8	7 0
0x02	Data =Red Color	Data =Green Color	Data =Blue Color

รหัสคำสั่ง พารามิเตอร์

เวลาจะ Write คำสั่งไปยัง RAM ผู้ใช้สามารถเรียกใช้งานฟังก์ชัน FT800memWrite32(Address RAM , DL_CMD) ได้เลย และอย่าลืมว่า การอ้าง Address RAM เริ่มต้นของคำสั่งจะต้องเป็น Address ที่หารด้วย 4 ลงตัว

2) Co-Processor Engine command (CO_CMD) : เป็นกลุ่มคำสั่งที่สามารถเขียนไปเก็บยังหน่วยความจำ RAM_CMD ได้เท่านั้น โดยคำสั่งในกลุ่มนี้จะมีลักษณะเหมือนฟังก์ชั่นที่มีการผ่านค่าพารามิเตอร์เข้ามาในฟังก์ชัน โดย พารามิเตอร์จะมีกี่ตัวก็ขึ้นอยู่กับลักษณะของคำสั่งนั้นๆ จึงทำให้แต่ละคำสั่งในกลุ่มนี้มีจำนวน Byte ที่ไม่แน่นอน ซึ่งลักษณะของคำสั่งจะประกอบไปด้วยรหัสคำสั่ง 4 Byte (32 bit) ส่วนจำนวน Byte ของ พารามิเตอร์ต่างๆ ของแต่ละคำสั่งนั้นจะไม่เท่ากัน ตัวอย่างของคำสั่งแสดงดังตารางด้านล่าง ซึ่งเป็นตัวอย่างของคำสั่ง “CMD_SLIDER” เป็นคำสั่งสร้างปุ่มแบบเลื่อน

Example Co_CMD : SLIDERS

ฟังก์ชันสำหรับ Write คำสั่ง	RAM_CMD+offset Address	จำนวน Byte Data	ชนิด Data
FT800_HostCmdWrite32()	Ram_CMD + 0	**4**	**รหัส CMD_SLIDERS =0xFFFFFF10**
FT800_HostCmdWrite16()	Ram_CMD + 4	**2**	**พารามิเตอร์ X**
FT800_HostCmdWrite16()	Ram_CMD + 6	**2**	**พารามิเตอร์ Y**
FT800_HostCmdWrite16()	Ram_CMD + 8	**2**	**พารามิเตอร์ W**
FT800_HostCmdWrite16()	Ram_CMD + 10	**2**	**พารามิเตอร์ h**
FT800_HostCmdWrite16()	Ram_CMD + 12	**2**	**พารามิเตอร์ option**
FT800_HostCmdWrite16()	Ram_CMD + 14	**2**	**พารามิเตอร์ val**
FT800_HostCmdWrite16()	Ram_CMD + 16	**2**	**พารามิเตอร์ rang**

ให้ RAM_CMD Start = 0x108000 (ต้อง 4 หารลงตัว)

จากตารางตัวอย่างจะเห็นว่าในช่อง “ชนิด Data” จะประกอบไปด้วย รหัสคำสั่ง 4 Byte และพารามิเตอร์ของคำสั่งขนาด 2 Byte 7 ตัว ดังนั้นคำสั่งนี้จะมีขนาด 18 Byte ดังนั้น เวลาจะ Write คำสั่งไปยัง RAM_CMD ผู้ใช้สามารถแยก Write data ทีละส่วนโดยใช้ฟังก์ชั่นที่ทางอีทีทีเขียนไว้ให้ จากตารางในส่วนที่เป็นรหัสคำสั่งขนาด 4 Byte เราก็ใช้งานฟังก์ชัน FT800memWrite32() โดย Address Start คำสั่งจะต้องหารด้วย 4 ลงตัว จากนั้นก็ตามด้วยการ Write พารามิเตอร์ X ซึ่งมีขนาด 2 Byte เราก็ใช้ฟังก์ชัน FT800memWrite16() โดย Address สำหรับ Write พารามิเตอร์นี้ ไม่จำเป็นต้องหารด้วย 4 ลงตัว แต่จะต้องอ้าง Address ให้ต่อจาก Address สุดท้ายของรหัสคำสั่ง ดังนั้น Address ที่ใช้ก็คือ Ram_CMD+4 จากนั้นก็ Write พารามิเตอร์ตัวต่อไปโดยจะต้องบวก Address เพิ่มไปทีละ 2 Address เนื่องจากขนาดของพารามิเตอร์ที่ต่อจาก พารามิเตอร์ X มีขนาด 2 Byte เท่ากันหมด จากตารางเราอาจจะใช้ฟังก์ชั่น FT800memWrite32() อย่างเดียวก็ได้ โดยในส่วนของพารามิเตอร์ให้จับคู่ Write ทีละ 4 Byte ซึ่งการอ้าง Address สำหรับ Write ก็จะต้องอ้างเพิ่มขึ้นครั้งละ 4 Address สำหรับพารามิเตอร์คู่ต่อไป และในส่วนของการเรียงข้อมูลสำหรับ Write ก็จะต้องวางตำแหน่งให้ถูกต้องด้วย เนื่องจากการ Write data ฟังก์ชันจะ Write ในส่วน Data byte0 ออกไปก่อน เช่น Write พารามิเตอร์ X,Y จะต้องกำหนด Data ค่า X อยู่ใน 2 Byte Low (bit 15..0) ส่วน Data ค่า Y จะต้องวางไว้ที่ ตำแหน่ง 2 Byte High (Bit 31-16) เป็นต้น

*** คำสั่งทั้งสองกลุ่มที่กล่าวไปข้างต้นผู้ใช้สามารถดูรายละเอียดในแต่ละคำสั่งได้ใน Data Sheet “FT800 Programmers Guide.pdf” ***

❖ *รูปแบบ Routine ในการส่งคำสั่ง*

หลังจากเราทราบลักษณะของกลุ่มคำสั่งในแต่ละแบบแล้ว ต่อไปเราจะมาดูในส่วนของการนำคำสั่งมาเขียนโปรแกรมกันบ้างซึ่งแน่นอนว่าจากที่กล่าวไปข้างต้นเป็นเพียงหลักการในการส่งคำสั่งออกไปยัง FT800 แต่เวลาเขียนโปรแกรมจริงๆ ผู้ใช้จะไม่สามารถส่งคำสั่งเหล่านั้นออกไปเพียงคำสั่งเดียว หรือ หลายๆคำสั่งตามที่ผู้ใช้อยากส่งแล้วจะเห็นการแสดงผลที่ Display ตามที่ผู้ใช้ต้องการให้เกิดขึ้นได้ เนื่องจากโปรแกรมที่เขียนส่งไปเก็บยัง RAM ของ FT800 นั้นจะต้องเขียนในรูปแบบเป็น Routine ที่มีการจัดลำดับของคำสั่งก่อนหลัง และใน Routine หนึ่งๆ จะต้องเริ่มต้นด้วยคำสั่งอะไรจบด้วยคำสั่งอะไรด้วย โดยจากคำสั่งทั้ง 2 กลุ่มข้างต้นจะเห็นว่า RAM ที่ใช้สำหรับเก็บคำสั่งนั้นจะมีด้วยกัน 2 ส่วน คือ RAM_DL และ RAM_CMD ดังนั้นเราสามารถแยกรูปแบบ Routine ในการเขียนโปรแกรม Control Display ได้ 2 แบบดังนี้

1) แบบที่1 : จะเป็นการเขียนคำสั่งกลุ่ม Display list (DL_CMD) ไปยัง RAM_DL โดยตรง ซึ่งการเขียนไปยัง RAM_DL นี้ ปกติจะเขียนสำหรับ Initial และ Enable Display โดยคำสั่ง Initial Display ต่างๆที่เก็บอยู่ใน List ของ RAM_DL จะถูก FT800 สั่งให้ Display Active ด้วยคำสั่ง swap สำหรับคำสั่ง Display List นั้นจะมีขนาด 32 bit (4 byte) โดยคำสั่งแรกของ Display List ควรจะเริ่มต้นที่ Address RAM_DL + 0 และคำสั่งต่อมาใน Routine เดียวกันก็จะต้องเพิ่มค่า Address ขึ้นครั้งละ 4 Address ค่าที่เพิ่มขึ้นนี้ก็คือค่า Offset Address โดยอย่าลืมว่า Address เริ่มต้นสำหรับเก็บคำสั่งแต่ละคำสั่งนั้นต้องหารด้วย 4 ลงตัวเสมอ เมื่อเราจะ Start Routine Display ด้วยคำสั่ง Display List ใน RAM_DL เราก็ควรจะเริ่มต้น Routine ในการเขียนดังนี้

1. ส่งคำสั่ง CLEAR_COLOR_RGB เพื่อกำหนดสี Back Ground ที่ต้องการ
2. ส่งคำสั่ง CLEAR เพื่อ Clear color buffer ,Clear stencil buffer , Clear tag buffer
3. ส่งคำสั่ง Display List อื่นๆที่ผู้ใช้ต้องการเขียน กี่คำสั่งก็ได้
4. ส่งคำสั่ง DISPLAY เพื่อ End display List , FT800 จะไม่ทำคำสั่ง DL_CMD ที่ตามหลังคำสั่งนี้ จนกว่าจะมีการ Start DL ใหม่
5. Write Data 0x02 ไปยัง REG_DLSWAP เพื่อ Active คำสั่ง Display List ที่เรา Write ไปเก็บ ใน RAM _DL

Example (อ้างอิงการใช้ฟังก์ชัน และตัวแปร ต่างๆตามตัวอย่างของอีทีที)

```
FT800memWrite32(RAM_DL+ 0, DL_CLEAR_RGB | 0xFFFFFF)      ; //1. Clear color Back Ground to White
FT800memWrite32(RAM_DL+ 4, DL_CLEAR | 7)                  ; //2.Clear color,stencil,tag buffer
//... คำสั่ง DL_CMD สำหรับผู้ใช้...//
FT800memWrite32(RAM_DL + 8, DL_COLOR_RGB |  0xFF0000)     ; //3. Set Color Point to Red color
FT800memWrite32(RAM_DL+ 12, DL_POINT_SIZE | (5*16))       ; //Set Point Size 5 Pixel  (unit 1/16 )
//.....//
FT800memWrite32(RAM_DL+16 , DL_DISPLAY)                   ; //4. End of display list
FT800memWrite32(REG_DLSWAP,0x02)                          ; //5. Write data(0x02) to Reg : Display List Execute
```

นอกจากนี้ในกลุ่มคำสั่ง DL_CMD จะมีคำสั่งที่เกี่ยวกับพื้นฐานการวาดทางด้าน Graphic ให้ด้วย ซึ่งพื้นฐานในการวาดทางด้าน Graphic ที่มีให้จะแสดงดังตารางด้านล่าง เมื่อจะใช้คำสั่งวาดพื้นฐานเหล่านี้เวลาส่งคำสั่งก็จะต้องมี Routine ในการวาด ซึ่ง Routine จะถูกกำหนดด้วยคำสั่งของ DL_CMD คือคำสั่ง BEGIN และ END โดย

คำสั่ง BEGIN จะเป็นตัวเลือกชนิดของการวาดพื้นฐาน ดังตารางด้านล่าง
คำสั่ง END จะเป็นตัวบอกจุดจบของการวาดพื้นฐานนั้นๆ

ตารางแสดงคำสั่งวาด Graphic

Name	Value	Description
BITMAPS	1	Bitmap drawing
POINTS	2	Point drawing
LINES	3	Line drawing
LINE_STRIP	4	Line strip drawing
EDGE_STRIP_R	5	Edge strip right side drawing
EDGE_STRIP_L	6	Edge strip left side drawing
EDGE_STRIP_A	7	Edge strip above drawing
EDGE_STRIP_B	8	Edge strip below side drawing
RECTS	9	Rectangle drawing

ซึ่งสามารถสรุป Routine ในการเขียนคำสั่งวาด Graphic พื้นฐานได้ดังนี้

1. ส่งคำสั่ง BEGIN() เพื่อกำหนดรูปแบบการวาด
2. ส่งคำสั่งที่เกี่ยวกับการกำหนด พารามิเตอร์ ในการวาด ตามรูปแบบที่เลือกไว้เช่น คำสั่ง VERTEX2F หรือ คำสั่ง VERTEX2II ซึ่งเป็นคำสั่งการกำหนดพิกัด X,Y สำหรับการวาด หรือ คำสั่ง POINT_SIZE สำหรับกำหนดขนาด Pixel หรือคำสั่ง COLOR_RGB สำหรับกำหนดสีให้ Pixel ที่จะวาด ข้อสังเกตอย่างหนึ่งคือ การวาดจะเกิดขึ้นที่คำสั่งกำหนดพิกัด X,Y ดังนั้น เวลาใช้คำสั่งกำหนดขนาด Pixel หรือ สีของ Pixel ควรใช้ก่อนคำสั่งกำหนดพิกัด X,Y มิฉะนั้น รูปที่วาด จะมีสี หรือ ขนาด Pixel ไม่เป็นไปตามที่ผู้ใช้ต้องการได้
3. ส่งคำสั่ง END เพื่อจบ Routine การวาด

(รายละเอียดเกี่ยวกับการใช้คำสั่ง BEGIN,END สามารถดูได้ใน Data Sheet "FT800 Programmers Guide.pdf ")

สำหรับ Routine สั่งวาด Graphic พื้นฐานนี้ เวลาจะเขียนไปยัง RAM_DL ให้ผู้ใช้เขียนแทรกไว้ใน Routine หลักที่กล่าวไปในข้างต้น โดยแทรกไว้ในส่วน " DL_CMD สำหรับผู้ใช้ " ดังตัวอย่างด้านล่าง

Example -วาด Dot สีแดง รัศมี 10 Pixel ที่ตั้ง (X= 100 pixel ,Y=130 pixel)
-วาด Dot สีเขียว รัศมี 50 Pixel ที่ตั้ง(X=200 pixel ,Y=130 pixel)
(อ้างอิงการใช้ฟังก์ชันและตัวแปรต่างๆตามตัวอย่างของอีทีที)

```
FT800memWrite32(RAM_DL+ 0, DL_CLEAR_RGB | 0xFFFFFF)      ; // ***Clear color Back Ground to white
FT800memWrite32(RAM_DL+ 4, DL_CLEAR | 7)                  ; //***Clear color,stencil,tag buffer
//... คำสั่ง DL_CMD สำหรับผู้ใช้...//
FT800memWrite32(RAM_DL+8,DL_BEGIN | 0x02)                 ; //1.Start Point Drawing
FT800memWrite32(RAM_DL+12,DL_COLOR_RGB | 0xFF0000)        ; //2.Set Color Point to Red color
FT800memWrite32(RAM_DL+16,DL_POINT_SIZE | (10*16) )       ; //Set Point Size 10 Pixel (unit 1/16 pixel)
FT800memWrite32(RAM_DL+20,(DL_VERTEX2F | ((long)(100*16) <<15) |130*16))  ; //Set Position point in unit 1/16 pixel

FT800memWrite32(RAM_DL+24,DL_COLOR_RGB | 0x00FF00)        ; // Set Color Point to Green color
FT800memWrite32(RAM_DL+28,DL_POINT_SIZE | (50*16))        ; //Set Point Size 50 Pixel (unit 1/16 pixel)
FT800memWrite32(RAM_DL+32,DL_VERTEX2F | ((long)(200*16) <<15) | (130*16))  ; //Set Position point in unit 1/16 pixel
//.....//
```

```
FT800memWrite32(RAM_DL+36 , DL_END)                    ; //3.End Point Draw
FT800memWrite32(RAM_DL+40 , DL_DISPLAY)                ; // ***End of display list
FT800memWrite32(REG_DLSWAP,0x02)                       ; // ***Write data(0x02) to Reg : Display List Execute
```

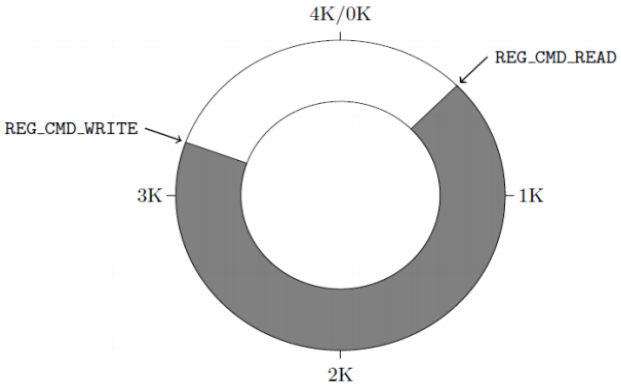
ในตัวอย่างนี้จะเห็นว่าในบรรทัดที่ Comment มี *** จะเป็นส่วนเริ่มต้นและส่วนจบของ Routine สำหรับการ Write DL_CMD ไปยัง RAM_DL และในส่วนคำสั่ง Point Draw จะอยู่ภายใน สำหรับคำสั่ง Point Draw ในตัวอย่าง เมื่อเราเลือกรูปแบบการวาดแล้วก็จะตามด้วยคำสั่ง Set สี และ Set ขนาด Point และเมื่อเจอคำสั่งกำหนดตำแหน่งการ Plot (VERTEX2F) จุดสีแดง ก็จะถูก Plot ขึ้นเป็นลำดับแรก เมื่อจะ Plot จุดสีเขียว ผู้ใช้ก็เพียง ส่งคำสั่ง Set สี และขนาด Point ใหม่ แล้วตามด้วยคำสั่งกำหนดตำแหน่งการ Plot อีกครั้ง ก็จะได้จุดที่สองออกมา ผู้ใช้ไม่จำเป็นต้อง เลือกโหมดการ Plot ใหม่ เนื่องจากใช้คำสั่ง Point Draw เหมือนกัน ยกเว้นผู้ใช้จะเปลี่ยนการวาดไปในโหมดอื่นก็จะต้องใช้คำสั่ง BEGIN เพื่อเปลี่ยนโหมดการวาดใหม่ ส่วนคำสั่ง End สำหรับจบโหมดการวาด สามารถเขียนในคำสั่งสุดท้ายครั้งเดียวก็ได้ถึงแม้ว่าก่อนหน้านั้นจะมีการเปลี่ยนโหมดการวาดกี่โหมดก็ตาม หรือ เขียนทุกครั้งเมื่อจบโหมดการวาดนั้นๆทุกครั้งก่อนที่จะเริ่มโหมดการวาดอื่นๆใหม่ก็ได้ สำหรับในคำสั่ง VERTEX2F นั้นการ Or (|) จะเป็นการรวม Data ระหว่าง รหัสคำสั่งและพารามิเตอร์เข้าไว้ใน 32 bit เดียวกัน ส่วนการ Shift ซ้าย (<<) จะเป็นการปรับตำแหน่งบิต Data ของ พารามิเตอร์ ให้ตรงกับตำแหน่ง(X,Y)ที่กำหนดไว้ในการจัดเรียงบิตของคำสั่ง ซึ่งดูได้ใน Data sheet และการคูณ Data ด้วย 16 ก็เพื่อแปลงค่า Pixel จากผู้ใช้ ให้อยู่ในค่าที่คำสั่งรองรับ เนื่องจากค่าที่คำสั่งรองรับจะมีค้าอยู่ที่ 1/16 Pixel ดังนั้น ถ้า 10 Pixel ค่าที่ต้องแทนเข้าไปจริงในคำสั่งก็คือ 10x16 = 160 นั่นเอง

ข้อระวัง *ในการส่งคำสั่งจะเห็นว่า ในส่วน Data 4 Byte คือ รหัสคำสั่ง+ค่าพารามิเตอร์ ของคำสั่ง DL_CMD นั้นๆ จะเป็น Data 32 bit ดังนั้น ในการจัดเรียง Bit จะต้องคำนึงเสมอว่า รหัสคำสั่งอยู่ที่ตำแหน่งบิตเท่าไหร่ และ พารามิเตอร์แต่ละค่าอยู่ที่ตำแหน่งบิตเท่าไหร่ด้วย เมื่อเวลารวมกันเป็น 32 บิต สำหรับส่ง จะต้องจัดเรียงบิตให้ตรงตำแหน่งตามรูปแบบของคำสั่งนั้นๆเสมอ*

2) แบบที่2 : จะเป็นการเขียนคำสั่งกลุ่ม Co-Processor Engine command (CO_CMD) และ Display list (DL_CMD) ไปยัง RAM_CMD โดยวิธีนี้ Co-Processor ภายใน FT800 จะแปลงและสร้าง คำสั่ง DL_CMD เข้าไปเก็บไว้ยัง RAM_DL ให้อีกต่อหนึ่ง ซึ่งต่างจากวิธีแรกที่เราใช้ MCU เขียนคำสั่ง เข้าไปยัง RAM_DL โดยตรง โดยวิธีนี้จะทำได้ง่ายในการเชื่อมต่อคำสั่งการวาด Graphic พื้นฐาน (เช่น line) เข้ากับอุปกรณ์(เช่น Button) บนหน้าจอเดียวกัน

จะเห็นความแตกต่างระหว่างวิธีที่ 1 กับที่ 2 คือ วิธีที่ 1 เราจะให้ MCU เขียนคำสั่งไปยัง RAM_DL โดยตรง ส่วนวิธีที่ 2 เราจะให้ MCU เขียนคำสั่งไปยัง RAM_CMD ก่อน แล้วให้ Co-Processor ภายในเขียนคำสั่งไปยัง RAM_DL อีกต่อหนึ่งเป็นต้น ซึ่งในตัวอย่างของอีทีที ก็จะใช้แบบที่สองเป็นหลัก โดย Routine ในการเขียนคำสั่ง CO_CMD และ DL_CMD ไปยัง RAM_CMD นั้นก็จะคล้ายๆกับแบบที่ 1 แต่จะมีในส่วนของการตรวจสอบตำแหน่งของหน่วยความจำเข้ามาเกี่ยวข้องด้วย เนื่องจากคำสั่งของ CO_CMD มีจำนวน Byte ไม่เท่ากันในแต่ละคำสั่ง อาจทำให้คำสั่งที่เขียนต่อจากคำสั่งแรกหรือคำสั่งต่อๆไป ไม่ได้เริ่มต้น Address ในตำแหน่งที่หารด้วย 4 ลงตัว เพราะเราต้องยึดหลักการเขียนคำสั่งไปยังหน่วยความจำตามที่กล่าวไปแล้วข้างต้นเสมอคือ Address เริ่มต้นสำหรับเก็บแต่ละคำสั่งต้องหารด้วย 4 ลงตัวเสมอ

ซึ่งในการเขียนคำสั่งไปยัง RAM_CMD สิ่งแรกที่เราต้องทำในการเริ่ม Routine เพื่อเขียนคำสั่งไปยัง RAM_CMD ก็คือตรวจสอบความพร้อมของ Co-Processor ภายใน ว่ามีความพร้อมในการรับคำสั่งจาก MCU หรือไม่ ด้วยการอ่านค่า Offset Address ของ RAM_CMD จาก REG_CMD_READ และ REG_CMD_WRITE ซึ่งปกติถ้าค่า Offset Address ที่อ่านได้จาก Register ทั้งสองตัวไม่เท่ากันแสดงว่า Co-Processor ภายในยังไม่พร้อมที่จะรับคำสั่งจาก MCU เนื่องจาก Co-processor ยังประมวลผลของ Routine คำสั่งก่อนหน้าที่ส่งมายังไม่เสร็จ ดังนั้นผู้ใช้จะต้องให้ MCU วนรออ่าน ค่า Offset Address จาก Register ทั้งสองตัวนี้ให้เท่ากันก่อนถึงจะเริ่มส่งคำสั่งไปยัง RAM_CMD ได้ โดยลักษณะ ของ RAM_CMD นั้นจะมีลักษณะเป็นวงแหวน มีขนาด 4096 Byte(4KB) โดย Address จะเริ่มที่ 0x108000 และ Offset Address ของ RAM_CMD จะมีค่าอยู่ที่ 0-4096 ดังรูปด้านล่าง



ซึ่งค่า Offset Address ใน REG_CMD_READ จะถูก Update โดย Co-processor ภายใน ในขณะที่ทำการอ่านคำสั่งที่เก็บอยู่ในหน่วยความจำ ที่ตำแหน่ง Address ปัจจุบันที่อ่านไปถึง โดย Co-processor จะเริ่มอ่านคำสั่งเมื่อคำสั่ง CMD_SWAP ถูกเขียนไปยังหน่วยความจำ ส่วน Offset Address ใน REG_CMD_WRITE จะถูก Update โดยให้ MCU เขียนค่า Offset Address ไปบอก Co-processor (Write Offset Address ไปยัง REG_CMD_WRITE) เมื่อสิ้นสุดคำสั่งสุดท้าย (CMD_SWAP)ของ Routine นั้นๆ ดังนั้น เมื่อจบ Routine ใดๆแล้วเราก็จะต้อง Write Offset Address สุดท้ายของ Routine นั้นๆไปยัง REG_CMD_WRITE จากนั้นเราก็มาวนรออ่านค่า REG_CMD_READ ว่ามีค่า Offset Address เท่ากับค่า ของ REG_CMD_WRITE หรือยัง ถ้าเท่ากันแสดงว่า Co-processor ภายในมีการอ่านค่าคำสั่งในหน่วยความจำมาถึง Offset Address สุดท้ายของ Routine แล้ว เราก็เริ่มส่งคำสั่งสำหรับ Routine ใหม่ได้จากที่กล่าวมาข้างต้นนี้ สามารถสรุป Routine ในการเขียนคำสั่งไปยัง RAM_CMD ได้ดังนี้

1. วนอ่านค่า Offset Address จาก REG_CMD_READ และ REG_CMD_WRITE แล้วเปรียบเทียบค่า Offset Address ว่าเท่ากันหรือไม่ เมื่อเท่ากัน แล้วให้นำค่า Offset Address ที่ได้ไปใช้งาน
2. ส่งคำสั่ง CMD_DLSTART (CO_CMD) เพื่อ Start Display List
3. ส่งคำสั่ง CLEAR_COLOR_RGB (DL_CMD) เพื่อกำหนดสี Back Ground ที่ต้องการ (**)
4. ส่งคำสั่ง CLEAR (DL_CMD) เพื่อ Clear color buffer ,Clear stencil buffer , Clear tag buffer (**)
5. ส่งคำสั่งกลุ่ม DL_CMD หรือ CO_CMD ตามที่ผู้ใช้ต้องการจะเขียนโปรแกรมให้แสดงที่หน้าจอ (**)
6. ส่งคำสั่ง DISPLAY (DL_CMD) เพื่อ End display (**)
7. ส่งคำสั่ง CMD_SWAP (CO_CMD) เพื่อทำการ Active Display list (**)
8. Write Offset Address สุดท้ายของคำสั่งใน Routine ไปยัง REG_CMD_WRITE เพื่อให้ Routine ใหม่ที่จะเขียนต่อตรวจสอบความพร้อมของ Co-processor ได้

(**) คือส่วนลำดับการส่งคำสั่งใน Routine ที่เหมือนกับ การเขียนคำสั่งไปยัง RAM_DL ที่กล่าวไว้ข้างต้น

เมื่อเทียบการเขียนคำสั่งไปยัง RAM_CMD กับการเขียนคำสั่งไปยัง RAM_DL จะเห็นว่าใน การเขียนคำสั่งไปยัง RAM_CMD จะมี Step 1,2,8 ที่เพิ่มเข้ามาเท่านั้น ส่วน Step 7 เราจะใช้คำสั่ง SWAP ของ CO_CMD ในการ Active Display list แทนการ Write ค่า 0x02 ไปยัง REG_DLSWAP

Example แสดงการ Plot ข้อความแสดงที่ Display ด้วยคำสั่ง CMD_TEXT( )

```
//-------------------------ประกาศตัวแปร-----------------------------
#define  RAM_CMD            0x108000          //Address Start Graphic Engine Command Buffer
unsigned int cmdOffset   = 0x0000;           //Offset Address Pointer of Command Buffer memory
unsigned int cmdBufferRd = 0x0000;           //Used to navigate command ring buffer
unsigned int cmdBufferWr = 0x0000;           //Used to navigate command ring buffer
```

```
//----------Step 1 Determine Pointer Offset for Ram_CMD Address and รอ Co-processor พร้อม ----------
  do{
      cmdBufferRd = FT800memRead16(REG_CMD_READ)     ;  //***Read Reg.the graphics processor read pointer
      cmdBufferWr = FT800memRead16(REG_CMD_WRITE)    ;  //***Read Reg.the graphics processor write pointer
   }while(cmdBufferWr != cmdBufferRd)                ;  //Wait until the two registers match
  cmdOffset = cmdBufferWr                            ;  //The new starting point the first location after the last command

  //-----------------  Step 2 Start Display List ------------------------
  FT800memWrite32(RAM_CMD+cmdOffset , CMD_DLSTART)   ;  //*** Sent Co-CMD Start the display list
  cmdOffset = incCMDOffset(cmdOffset,4)              ; //Update Offset pointer Ram-CMD 4 byte for Next CMD

  //----------------- Step 3 Clear Color RGB และ Set Back ground ----------------------
  FT800memWrite32(RAM_CMD + cmdOffset,(DL_CLEAR_RGB | 0x000000)) ; //*** Set Back Ground to black
  cmdOffset = incCMDOffset(cmdOffset,4)              ; //Update Offset pointer Ram-CMD 4 byte for Next CMD

//----------------- Step 4 Clear buffer Color, stencil, tag ----------------------
  FT800memWrite32(RAM_CMD+ cmdOffset , DL_CLEAR | 7 )  ;  //***Clear color,stencil,tag buffer
  cmdOffset = incCMDOffset(cmdOffset,4)              ;  //Update Offset pointer Ram-CMD 4 byte for Next CMD

  //---------------------- Step 5 User Program ------------------------
  FT800memWrite32(RAM_CMD+cmdOffset ,(DL_COLOR_RGB | 0xFFFFFF)  ; //***Sent DL-CMD Set Color Text-White
  cmdOffset = incCMDOffset(cmdOffset,4)              ;  //Update Offset pointer Ram-CMD 4 byte for Next CMD
   //------Start Draw Text------//
  FT800memWrite32(RAM_CMD+cmdOffset,CMD_TEXT)        ; //***Sent Co-CMD Draw TEXT = 0xFFFFFF0C
  cmdOffset = incCMDOffset(cmdOffset,4)              ; //Update Offset pointer Ram-CMD 4 byte for Next data
  FT800memWrite16(RAM_CMD + cmdOffset,x)             ; //---Sent data X-Point top-left (Pixel)
  cmdOffset = incCMDOffset(cmdOffset,2)              ; //Update Offset pointer Ram-CMD 2 byte for Next data
  FT800memWrite16(RAM_CMD + cmdOffset,y)             ; //---Sent data Y-Point top-left (Pixel)
  cmdOffset = incCMDOffset(cmdOffset,2)              ; //Update Offset pointer Ram-CMD 2 byte for Next data
 FT800memWrite16(RAM_CMD + cmdOffset,font)           ; //---Sent data Set format font(16,18,20,21,22,23,24...)
 cmdOffset = incCMDOffset(cmdOffset,2)               ; //Update Offset pointer Ram-CMD 2 byte for Next data
 FT800memWrite16(RAM_CMD + cmdOffset,opt)            ; //---Sent data Option : Position Draw  Text
 cmdOffset = incCMDOffset(cmdOffset,2)               ; //Update Offset pointer Ram-CMD 2 byte for Next data
 FT800memWriteStr(RAM_CMD + cmdOffset,str)           ; //---Sent data Text     String
 cmdOffset = End_incCMDOffset(cmdOffset,num_cha)     ; //Update Offset pointer Ram-CMD for End CMD-Text

  //-----------------Step6 End Display List  -------------
  FT800memWrite32(RAM_CMD + cmdOffset,DL_DISPLAY)    ;  //*** Sent DL-CMD End Display List
  cmdOffset = incCMDOffset(cmdOffset,4)              ;  //Update Offset pointer Ram-CMD 4 byte for Next CMD
```

```
//-----------------Step7 Active Display List Execute -------------
FT800memWrite32(RAM_CMD + cmdOffset,CMD_SWAP)     ; //*** Sent Co-CMD Requse Display List Swap
cmdOffset = incCMDOffset(cmdOffset,4)             ; //Update Offset pointer Ram-CMD 4 byte for Next CMD
//-----------------Step8 End Display List and Update Execute -------------
FT800memWrite16(REG_CMD_WRITE,cmdOffset)          ; //*** Sent Data cmdOffset to reg. CMD_Write for Update the
                                                  //ring buffer pointer so the graphics processor starts executing
```

จากตัวอย่างนี้จะเป็นส่วนตัดต่อมาจากตัวอย่างของอีทีที เพื่อให้มองเห็นภาพในแต่ละ Step ของ Routine ในการส่งคำสั่งเพื่อให้เข้าใจง่ายขึ้น โดยใน Step อื่นๆให้ผู้ใช้ส่งคำสั่งตามตัวอย่าง ยกเว้น Step 5 ผู้ใช้สามารถแก้ไขโปรแกรมให้แสดงผลบน Display ได้ตามที่ผู้ใช้ต้องการ จากตัวอย่างนี้จะมีฟังก์ชันในการคำนวณ Offset Address เพิ่มเข้ามา คือ incCMDOffset(cmdOffset,จำนวน Byte ของคำสั่งก่อนหน้า) ซึ่งฟังก์ชันนี้ จะต้องผ่านค่า Offset address และจำนวน Byte ของคำสั่งก่อนหน้า ให้กับฟังก์ชัน จากนั้นฟังก์ชันก็จะคำนวณ Offset Address สำหรับคำสั่งต่อไปให้โดยจะเอาค่า Offset ใหม่ที่คำนวณได้ไปบวกกับ ค่า Address เริ่มต้นของ RAM_CMD ก็จะได้เป็น Address เริ่มต้นของคำสั่งใหม่ (Address เริ่มต้นของคำสั่งใหม่ต้องหารด้วย 4 ลงตัว) โดยเราไม่ต้องคอยบวกเองเหมือนกับตัวอย่างของวิธีแรก จากตัวอย่างอีกส่วนหนึ่งที่ผู้ใช้ต้องสังเกตในการเขียนโปรแกรมส่งคำสั่งคือ ในส่วนที่เป็นคำสั่ง CO_CMD จะเห็นว่า คำสั่ง CO_CMD ที่ไม่มีการผ่านค่าพารามิเตอร์ จะมีขนาด 4 Byte ซึ่งเป็น Data ของรหัสคำสั่ง ผู้ใช้ก็สามารถส่งคำสั่งได้ตามปกติและเวลาคำนวณค่า Offset Address สำหรับคำสั่งต่อไปก็จะได้ค่า Address เริ่มต้นของคำสั่งใหม่ที่หารด้วย 4 ลงตัวพอดี แต่สำหรับ CO_CMD ที่มีการผ่านค่าพารามิเตอร์ในตัวอย่างคือคำสั่ง CMD_TEXT ซึ่ง Address เริ่มต้นสำหรับส่งรหัสคำสั่งจะต้องหารด้วย 4 ลงตัวเหมือนคำสั่งอื่นๆ แต่เวลาส่ง Data ในส่วนของพารามิเตอร์ Address สำหรับ Data พารามิเตอร์ไม่จำเป็นต้องหารด้วย 4 ลงตัว ให้เขียนต่อจาก Address สุดท้ายของรหัสคำสั่งได้เลยเนื่องจากอยู่ในชุดคำสั่งเดียวกัน ซึ่งในตัวอย่าง พารามิเตอร์ตัวแรกมีขนาด 2 Byte ก็ใช้ฟังก์ชัน FT800memWrite16() ในการส่ง , Data พารามิเตอร์ตัวที่ 2 มีขนาด 2 Byte เช่นกันก็ใช้ฟังก์ชัน FT800memWrite16() ในการส่งเช่นเดียวกัน เมื่อส่ง data พารามิเตอร์ออกไปแต่ละครั้งก็ให้เพิ่มค่า Offset address เท่ากับจำนวน Byte ของพารามิเตอร์นั้นๆ ซึ่งจะทำให้การส่ง Data ของ CMD_TEXT ถูกเก็บใน Address ที่ต่อเนื่องกัน และในส่วนสุดท้ายของ CMD_TEXT เราจะจบด้วยฟังก์ชัน End_incCMDOffset(cmdOffset,จำนวน Byte ของพารามิเตอร์ที่เขียนก่อนหน้า) ซึ่งฟังก์ชันนี้นอกจากทำหน้าที่คำนวณ Offset Address ให้คำสั่งต่อไปแล้ว ยังทำหน้าที่ปรับ Offset Address ให้หารด้วย 4 ลงตัวสำหรับคำสั่งต่อไปด้วย ถ้า Offset Address ที่คำนวณนั้นหารด้วย 4 ไม่ลงตัว

เนื่องจากจะเห็นว่าคำสั่ง CO_CMD ที่มีการผ่านค่าพารามิเตอร์นั้น จำนวน Byte data ทั้งหมดของคำสั่ง จะเป็นจำนวน Byte ที่หารด้วย 4 ลงตัวบ้างไม่ลงตัวบ้าง ทำให้เมื่อคำนวณค่า Offset Address ของคำสั่งต่อไปออกมาจะได้ค่า Offset address ทีหารด้วย 4 ลงตัวบ้างไม่ลงตัวบ้างเช่นกัน ดังนั้นฟังก์ชัน End_incCMDOffset() ก็จำทำการปรับค่า Offset Address ด้วยการเลื่อน Offset Address ไปข้างหน้าจนกว่าจะได้ตำแหน่ง Offset Address ที่หารด้วย 4 ลงตัว สำหรับ CO_CMD อื่นๆที่มีการผ่านค่าพารามิเตอร์ก็ให้ยึดหลักการเขียนในลักษณะนี้ด้วย

จากการเขียนคำสั่งไปยังหน่วยความจำทั้งสองวิธีเมื่อจบ Routineใดๆ ด้วยคำสั่ง CMD_SWAP หรือ เขียนค่า 0x02 ไปยัง REG_DLSWAP แล้ว ใน Routine ใหม่ผู้ใช้สามารถกลับไปใช้ Address เริ่มต้นซ้ำในการอ้างอิงตำแหน่งของหน่วยความจำสำหรับเก็บ Data ใหม่ได้ ไม่จำเป็นต้องอ้าง Address ต่อจาก Routine ก่อนหน้า

❖ *การ Initial MCU /FT800/Display Screen*

เมื่อทราบ Routine ในการเขียนคำสั่งไปยังหน่วยความจำทั้งสองแบบแล้ว ในหัวข้อนี้เราก็จะมาเริ่มต้นเขียนโปรแกรมกันโดยในการเขียนโปรแกรมเราจะต้องทำการส่งคำสั่งเพื่อ Initial ส่วนต่างๆของ FT800 เป็นอันดับแรก อันได้แก่ Initial ตัว FT800 , Audio ,Touch และ Display โดยการ Initial จะอ้างอิงตามตัวอย่างของอีทีทีซึ่งสารมารถสรุปเป็นขั้นตอนได้ดังนี้

MCU_SETUP

1) Setup MCU ให้ทำงาน Interface แบบ SPI 8 bit โดยจะใช้ Software SPI (ใช้ขา I/O) หรือ Hardware SPI(ใช้ Module SPI ภายใน) ก็ได้ซึ่งในตัวอย่างของอีทีทีจะใช้แบบ Software SPI โดยกำหนดขา SCK Default เป็น 0 และ SCK ทำงานที่ขอบขาขึ้น และในการส่งข้อมูล Bit 7 จะถูกส่งเป็นบิตแรก

Wake UP

2) Reset FT800 โดย Drive PIN FT_PD ไปเป็น Low แล้ว delay 20 ms จากนั้น Drive กลับมาเป็น High แล้ว delay อีก 20 ms

3) ส่งคำสั่ง Wake-up โดยใช้ฟังก์ชัน FT800_HostCmdWrite() , Write ค่า 0x00

4) ส่งคำสั่ง เลือก Source Clock ให้ FT800 โดยใช้ฟังก์ชัน FT800_HostCmdWrite() , Write ค่า 0x44 สำหรับ External Clock

5) ส่งคำสั่ง Set speed FT800 ไปที่ 48 MHZ โดยใช้ฟังก์ชัน FT800_HostCmdWrite() , Write ค่า 0x62 สำหรับ Speed 48 MHz PLL

6) อ่านค่า Device ID จาก Register REG_ID ด้วยคำสั่ง FT800memRead8(REG_ID) ซึ่งค่าที่อ่านได้จะต้องเท่ากับ 0x7C

Configure Display

7) เขียน 0x00 ไปยัง REG_PCLK และ REG_PWM_DUTY ด้วยคำสั่ง FT800memWrite8() เพื่อ Disable Clock และ Turn off Backlight

8) ทำการ Set ค่า Register ต่างๆสำหรับ Display ดังนี้

```
FT800memWrite16(REG_HSIZE,480)      ; //***Write data to Reg : Active display width =480 Pixel
FT800memWrite16(REG_HCYCLE,548)     ; //***Write data to Reg : Total number of clocks per line, incl front/back porch
FT800memWrite16(REG_HOFFSET,43)     ; //***Write data to Reg : start of active line
FT800memWrite16(REG_HSYNC0,0)       ; //***Write data to Reg : Start of horizontal sync pulse
FT800memWrite16(REG_HSYNC1,41)      ; //***Write data to Reg : End of horizontal sync pulse
FT800memWrite16(REG_VSIZE,272)      ; //***Write data to Reg : Active display height =272 Pixel
FT800memWrite16(REG_VCYCLE,292)     ; //***Write data to Reg : Total number of lines per screen, incl pre/post
FT800memWrite16(REG_VOFFSET,12)     ; //***Write data to Reg : Start of active screen
FT800memWrite16(REG_VSYNC0,0)       ; //***Write data to Reg : Start of vertical sync pulse
FT800memWrite16(REG_VSYNC1,10)      ; //***Write data to Reg : End of vertical sync pulse
FT800memWrite8(REG_SWIZZLE,0)       ; //***Write data to Reg : FT800 output to LCD - pin order
FT800memWrite8(REG_PCLK_POL,1)      ; //***Write data to Reg : LCD data is clocked in on this PCLK edge
                                      //***Write data to Reg : Don't set PCLK yet - wait for just after the first display list
```

Configure Touch Screen

9) เขียนค่า 0x03 ไปยัง REG_TOUCH_MOD ด้วยคำสั่ง FT800memWrite8() เพื่อเลือกโหมด Touch แบบ Continuous

10) เขียนค่า 1200 ไปยัง REG_TOUCH_RZTHRESH ด้วยคำสั่ง FT800memWrite8() เพื่อกำหนดค่าความต้านทานในการ Touch หรือ กำหนดค่าความไวในการ Touch นั่นเอง

Configure Audio

11) เขียนค่า 0x82 ไปยัง REG_GPIO_DIR ด้วยคำสั่ง FT800memWrite8() สำหรับกำหนดทิศทาง PORT GPIO Bit 7 กับ 1 ไปเป็น O/P เพื่อ ใช้ Control การทำงานของ Display และ Audio Amp

12) เขียนค่า 0x02 ไปยัง REG_GPOI ด้วยคำสั่ง FT800memWrite8() เพื่อ Enable Audio Amp (ควรอ่านค่าจาก REG_GPIO ก่อน แล้วนำค่า 0x02 ไป Or เพื่อให้ค่า ใน REG_GPIO บิตอื่นไม่เปลี่ยนแปลง)

13) เขียนค่า 0x00 ไปยัง REG_VOL_PB ด้วยคำสั่ง FT800memWrite8() เพื่อ Turn Off Volume สำหรับเล่นกลับไฟล์เสียง

14) เขียนค่า 0x80 ไปยัง REG_VOL_SOUND ด้วยคำสั่ง FT800memWrite8() เพื่อปรับ Volume ไปที่ 50 % สำหรับ เสียงสังเคราะห์

15) เขียนค่า 0x0000 ไปยัง REG_SOUND ด้วยคำสั่ง FT800memWrite8() เพื่อเลือกเสียงสังเคราะห์ (sound effect) Silence (No sound)

16) เขียนค่า 0x01 ไปยัง REG_PLAY ด้วยคำสั่ง FT800memWrite8() เพื่อสั่งเล่นเสียงสังเคราะห์ข้างต้น

Initial Display List & Enable Display

17) Initial Display List สำหรับ Clear Screen โดยเขียนคำสั่งในรูปแบบ Display list ตามที่กล่าวไปในหัวข้อข้างต้น ด้วยคำสั่งต่อไปนี้

```
FT800memWrite32(RAM_DL+ 0, DL_CLEAR_RGB | 0x000000)      ; //1. Clear color Back Ground to Black
FT800memWrite32(RAM_DL+ 4, DL_CLEAR | 7)                 ; //2.Clear color,stencil,tag buffer
```

```
FT800memWrite32(RAM_DL+8, DL_DISPLAY)                    ; //3. End of display list
FT800memWrite32(REG_DLSWAP,0x02)                         ; //4. Write data(0x02) to Reg : Display List Execute
```

18) เขียนค่า 0x80 ไปยัง REG_GPIO ด้วยคำสั่ง FT800memWrite8() เพื่อ Set บิต 7 ของ REG_GPIO ไปเป็น 1 สำหรับ Enable Display (ควรอ่านค่าจาก REG_GPIO ก่อน แล้วนำค่า 0x80 ไป Or เพื่อให้ค่า ใน REG_GPIO บิตอื่นไม่เปลี่ยนแปลง)

19) เขียนค่า 0x05 ไปยัง REG_PCLK ด้วยคำสั่ง FT800memWrite8() เพื่อ Start Clock

20) เขียนค่า 128 ไปยัง REG_PWM_DUTY ด้วยคำสั่ง FT800memWrite8() สำหรับ Turn on Back-Light

จากที่กล่าวข้างต้นนั้นเป็นเพียงขั้นตอนในการ Initial Display สำหรับการเขียนโปรแกรมใช้งานจริงนั้นให้ผู้ใช้ทำการ Copy ฟังก์ชัน FT800_Initial() จากตัวอย่างของ อีทีที ไปใช้งานได้เลย

4.5 การ Write Data ไปเก็บใน RAM_G จากหัวข้อที่กล่าวไปข้างต้นนั้นจะเป็นการเขียนคำสั่งเพื่อ Control Display โดยการ Write คำสั่งต่างๆไปเก็บยัง หน่วยความจำ RAM_DL และ RAM_CMD ส่วนในหัวข้อนี้จะเป็นการเขียนข้อมูลจากภายนอกไปเก็บไว้ยัง RAM_G โดยข้อมูลที่เก็บอยู่ใน RAM_G นี้จะถูกดึงไปใช้โดยผ่านคำสั่งต่างๆของ FT800 ตามโปรแกรมที่ผู้ใช้เขียน พูดง่ายๆก็คือ เราไม่สามารถส่ง Data จากภายนอกไปให้ FT800 นำไปแสดงผลได้โดยตรงแบบ Real Time เราจะต้อง Write Data ไปเก็บไว้ยัง RAM_G ก่อน จากนั้นจึงใช้คำสั่งที่มีให้ของ FT800 ไปอ่าน Data จาก RAM_G ไปแสดงที่ Display ซึ่ง Data ที่ผู้ใช้จะนำไปเก็บไว้ยัง RAM_G เช่น ไฟล์ภาพ , เสียง เป็นต้น โดยไฟล์เหล่านี้ผู้ใช้จะต้องทำการ Convert ออกมาเป็น Hex Code ก่อน ด้วยโปรแกรม Convert ไฟล์ที่ให้มาใน CD จากนั้นนำ Hex Code ที่ได้ไปวางในพื้นที่ Flash หรือ RAM ของ MCU โดยให้ประกาศตัวแปรสำหรับเก็บ Hex Code เป็นชนิด Array แล้วจึงใช้คำสั่ง FT800memWrite8() เขียนไปยังหน่วยความจำ RAM_G ซึ่ง RAM_G จะเริ่มที่ Address 0x000000 มีขนาด 256 Kb โดยหลักการ Write ก็เช่นเดิม คือ Address เริ่มต้นสำหรับ Write Data Byte แรก ของข้อมูลแต่ละชุดจะต้องหารด้วย 4 ลงตัว เช่น ผู้ใช้ต้องการ Write Data รูปภาพจำนวน 3 รูป ไปเก็บยัง RAM_G เพื่อใช้แสดงผลที่ Display ดังนั้น Data Byte แรกของแต่ละรูป จะต้องอ้าง Address เริ่มต้นที่หารด้วย 4 ลงตัว โดย Address เริ่มต้นของแต่ละไฟล์ภาพ ไม่ควรไปทับซ้อนกับ Address สุดท้ายของไฟล์ภาพนั้นๆ โดย Address เริ่มต้นของไฟล์ภาพไม่จำเป็นต้องต่อเนื่องกับ Address สุดท้ายของไฟล์ภาพก่อนหน้าก็ได้ และสุดท้ายผู้ใช้จะต้องจำ Address Start ของไฟล์แต่ละภาพไว้ด้วยเพื่อนำไปใช้กับคำสั่งแสดงภาพ โดยดูฟังก์ชันการเรียกแสดงภาพจาก RAM_G ได้จากฟังก์ชั่น Bitmap_DL() ซึ่งอยู่ในตัวอย่างที่ 4 ของอีทีที

สำหรับการ Write Data ไปยัง RAM_G จะอ้างอิงตามฟังก์ชัน FT800memWrBitmap() ที่ทางอีทีทีเขียนขึ้นดังนี้

```
unsigned long BMP_Add[50]       ;       //Keep Address Start of each Picture in Ram_G  Max 50 Pixture
unsigned char bmp_pt=0          ;
void FT800memWrBitmap(unsigned long addr, unsigned long length ,char *bitmap)
{
 unsigned long pt=0 ;
 unsigned char dat  ;
 BMP_Add[bmp_pt] = addr                  ; //Set begin Address in Ram_G for Write data Picture
 bmp_pt        = bmp_pt+1                ; //Pointer Number Picture
 spi_cs_lo()                             ; //Set CS# low
 spi_rw(((addr>>16)|MEM_WRITE))          ; //Send Memory Write plus high address byte
 spi_rw(addr>>8)                         ; //Send middle address byte
 spi_rw(addr)                            ; //Send low address byte
 while(length--)                         ; //Start Write data to RAM_G
 {
   dat  =  bitmap[pt])                   ; //Read data from Flash at data picture
   spi_rw(dat)                           ; //Write data byte to FT800 RAM_G
```

```
        pt++                                    ; //Increment Pointer for array
    }
    spi_cs_hi()                                 ; //Set CS# high
}
```

จากตัวอย่างในฟังก์ชันนี้เริ่มต้นตัวแปร BMP_Add[50] จะมีไว้สำหรับเก็บ Address ของ RAM_G ที่เป็น Address เริ่มต้นของข้อมูลแต่ละชุดที่ Write ไปยัง RAM_G โดยตั้งไว้ 50 ชุด และตัวแปร bmp_pt จะมีไว้สำหรับเก็บลำดับของชุดข้อมูล ข้อมูลทั้งสองตัวแปรนี้จะเอาไว้เรียกใช้เมื่อสั่งแสดงข้อมูลออก Display เมื่อจะใช้งานฟังก์ชันผู้ใช้จะต้องผ่านค่าให้กับฟังก์ชันนี้ 3 ค่า คือ 1) Address เริ่มต้นของ RAM_G ที่ต้องการจะ Write Data ไปเก็บ 2) จำนวน Byte ของ Data ที่จะ Write โดยจะระบุไว้ในข้อมูลที่ Convert ได้ (ดูได้จากหัวข้อต่อไป) 3) ตำแหน่ง Array Pointer ของ Data ที่จะให้ MCU มาอ่าน เพื่อเขียนข้อมูลไปยัง RAM_G เริ่มการทำงานของฟังก์ชันอันดับแรกฟังก์ชันก็จะทำการเก็บ Address ของ RAM_G ที่เป็น Address เริ่มต้นที่จะ Write ข้อมูลไปเก็บ และ เก็บลำดับชุดข้อมูลไว้ในตัวแปรที่กล่าวไว้ข้างต้นก่อน จากนั้น MCU ก็จะทำการ Write Address ของRAM_G ไปยัง FT800 เพื่อใช้เป็น Address เริ่มต้นในการเก็บข้อมูล ต่อมาใน Loop While ก็จะเป็นการเริ่มเขียนข้อมูลไปยัง RAM_G ทีละ Byte จนครบตามจำนวน Byte ที่ต้องการ Write เป็นอันจบการ Write Data ไปยัง RAM_G เมื่อจะเขียนข้อมูลชุดต่อไปผู้ใช้ก็เรียกใช้งานฟังก์ชันนี้ใหม่ โดยอย่าอ้าง Address ให้ทับกับ Address ที่มีข้อมูลชุดแรกเก็บอยู่ ข้อสำคัญ Address เริ่มต้นสำหรับเก็บข้อมูลแต่ละชุดจะต้องอ้างตำแหน่ง Address ที่หารด้วย 4 ลงตัว

5. การใช้โปรแกรม Convert ไฟล์รูปภาพไปเป็น hex code

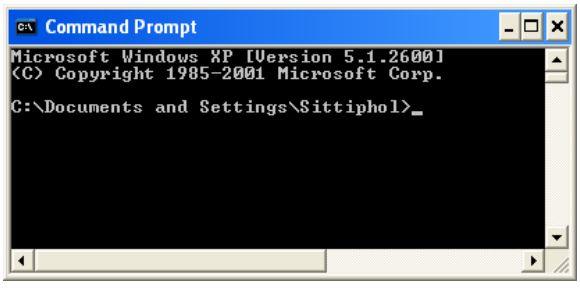
สำหรับโปรแกรม Convert ที่มีให้ใน CD จะมีอยู่ด้วยกัน 3 โปรแกรม โดยโปรแกรมเหล่านี้จะ Run ด้วย Command Dos ดังนั้นเวลาใช้งานผู้ใช้จะต้อง Run โปรแกรม Command Prompt () จาก Window ขึ้นมาก่อน แล้วจึงใช้ Command Dos เรียก Run โปรแกรม Convert ที่จะใช้งานขึ้นมา โปรแกรม Convert จะประกอบด้วย

1) img_cvt_0.3 เป็นโปรแกรม Convert รูปภาพ ที่มีนามสกุล *.PNG หรือ *.JPG
2) fnt_cvt_0.3.1 เป็นโปรแกรม Convert Character ในรูปแบบ UTF-8 encoder File หรือ ASCII printable Code
3) aud_cvt_0.2 เป็นโปรแกรม Convert ไฟล์เสียงที่อยู่ในรูปแบบ 8 Bit signed PCM(LINEAR) หรือ 8 Bit u-Law หรือ 4 Bit IMA ADPCM

โดยในหัวข้อนี้จะกล่าวถึงเฉพาะการใช้งานโปรแกรม img_cvt_0.3 เท่านั้นซึ่งเป็นการ Convert ไฟล์รูปภาพให้เป็น Hex Code แล้วนำ Hex Code ที่ได้ ไป Write ลงใน RAM_G ของ FT800 ดังที่กล่าวไปแล้วข้างต้น ส่วนโปรแกรม Convert อื่น ให้ผู้ใช้ศึกษาการใช้เองโดยสามารถดูรายละเอียดการใช้โปรแกรมได้จาก ไฟล์ Readme.txt ของโปรแกรมนั้นๆ

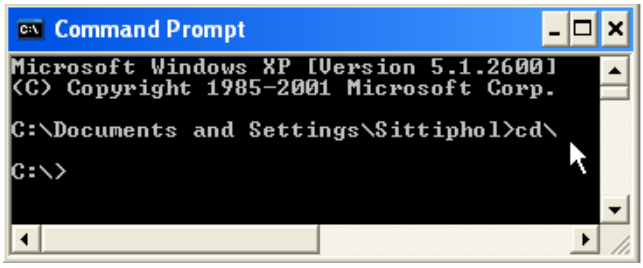
ขั้นตอนการใช้งานโปรแกรม img_cvt_0.3

1) Copy Folder “ img_cvt_0.3” ไปวางไว้ที่ Drive C หรือ Drive ใดๆก็ได้โดย พยายามวาง Folder ให้อยู่ชั้นนอกสุดเพื่อให้ง่ายในการพิมพ์คำสั่ง DOS
2) Copy File ภาพ(*.png , *.jpg) ที่ต้องการ Convert มาวางไว้ใน Folder “img_cvt_0.3” โดยในตัวอย่างจะให้ File รูปภาพมา 3 File คือ human130_95.jpg , audio105_105.jpg และ dog75_85.jpg
3) ทำการ Run Command Prompt () ขึ้นมาจะได้หน้าต่างดังรูปด้านล่าง



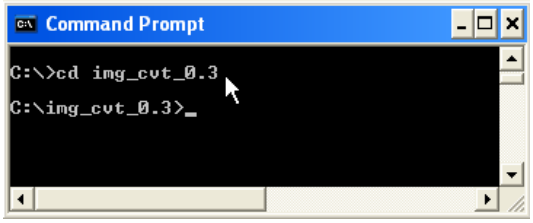
สำหรับโปรแกรม Command Prompt นั้น ถ้าเป็น -Window Xp ให้คลิกที่ Start (start) เลือก Programs และเลือก Accessories , -Window7 ให้คลิกที่ Start ( ) เลือก All Programs และเลือก Accessories และ - Window8 ให้คลิกขวาที่ Start ( ) และเลือกที่ Search แล้วพิมพ์คำว่า Command Prompt ก็จะพบ Icon

4) พิมพ์ cd\ แล้ว Enter เพื่อออกมาอยู่ที่ตำแหน่ง Drive C ซึ่งจะได้หน้าจอดังรูป

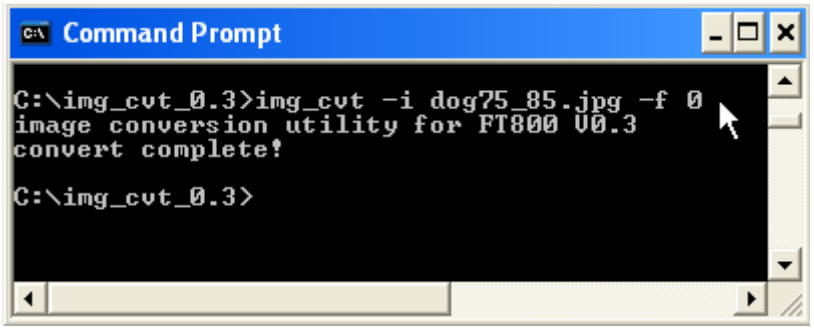


5) ในที่นี้สมมติเราวาง Folder “ img_cvt_0.3” ไว้ที่ Drive C ในกรณีวางไว้ที่ Drive อื่นๆ ก็ต้องทำการเปลี่ยน Drive ก่อน โดยไปที่ C-Prompt (C:\>) ให้พิมพ์ Drive ที่จะเปลี่ยนตามด้วย Colon (:) แล้ว Enter เช่นเปลี่ยนไป Drive D ก็ให้พิมพ์ C:\> D: แล้ว Enter ก็จะได้ D-Prompt (D:\>) เป็นต้น

6) ให้พิมพ์คำสั่ง cd img_cvt_0.3 แล้ว Enter ก็จะได้ข้อความดังรูปด้านล่าง คำสั่งนี้จะเข้าไปยัง Folder “img_cvt_0.3” ซึ่งเป็น Folder ที่เก็บโปรแกรม img_cvt.exe และไฟล์รูปอยู่



7) สมมติเราจะ Convert ไฟล์รูป ชื่อ “ dog75_85.jpg ” ก็ให้พิมพ์คำสั่งตามนี้ img_cvt -i dog75_85.jpg -f 0 แล้ว Enter รูปก็จะถูก Convert และจะได้หน้าต่างดังรูปด้านล่าง เป็นอัน Convert เสร็จสิ้น



จากชุดคำสั่งเขียนเป็นโครงสร้างคำสั่งได้ดังนี้ img_cvt -i inputfilename -f format

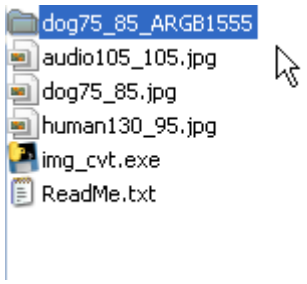
โดย Img_cvt = คำสั่ง Convert file รูปภาพ

inputfilename = ชื่อและนามสกุลของไฟล์รูปภาพที่จะใช้ Convert (เปลี่ยนไปตามชื่อไฟล์ที่นำมา Convert)

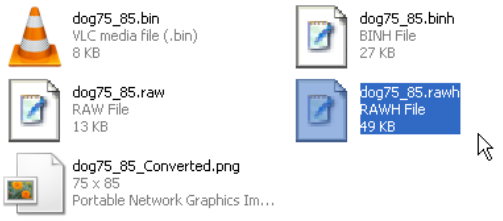
format = รูปแบบ Output File ที่จะทำการ Convert โดยกำหนดค่าได้ตั้งแต่ 0-8 ซึ่งมีรายละเอียดดังนี้

0 : ARGB1555 [default]	1 : L1
2 : L4	3 : L8
4 : RGB332	5 : ARGB2
6 : ARGB4	7 : RGB565
8 : PALETTEED	

8) เข้าไปที่ Folder “ img_cvt_0.3” ใน Drive ที่ผู้ใช้วางไว้ ก็จะพบ Folder “dog75_85_ARGB1555” ซึ่งเป็น Folder Output ที่ได้จากการ Convert ซึ่งใน Folder ก็จะพบ Output ไฟล์ 5ไฟล์ คือ dog75_85.bin , dog75_85.binh , dog75_85.raw, dog75_85.rawh และ dog75_85_Converted.png ให้ผู้ใช้สนใจเฉพาะไฟล์นามสกุล *.rawh ซึ่งจะเป็นไฟล์ที่เก็บ Hex Code

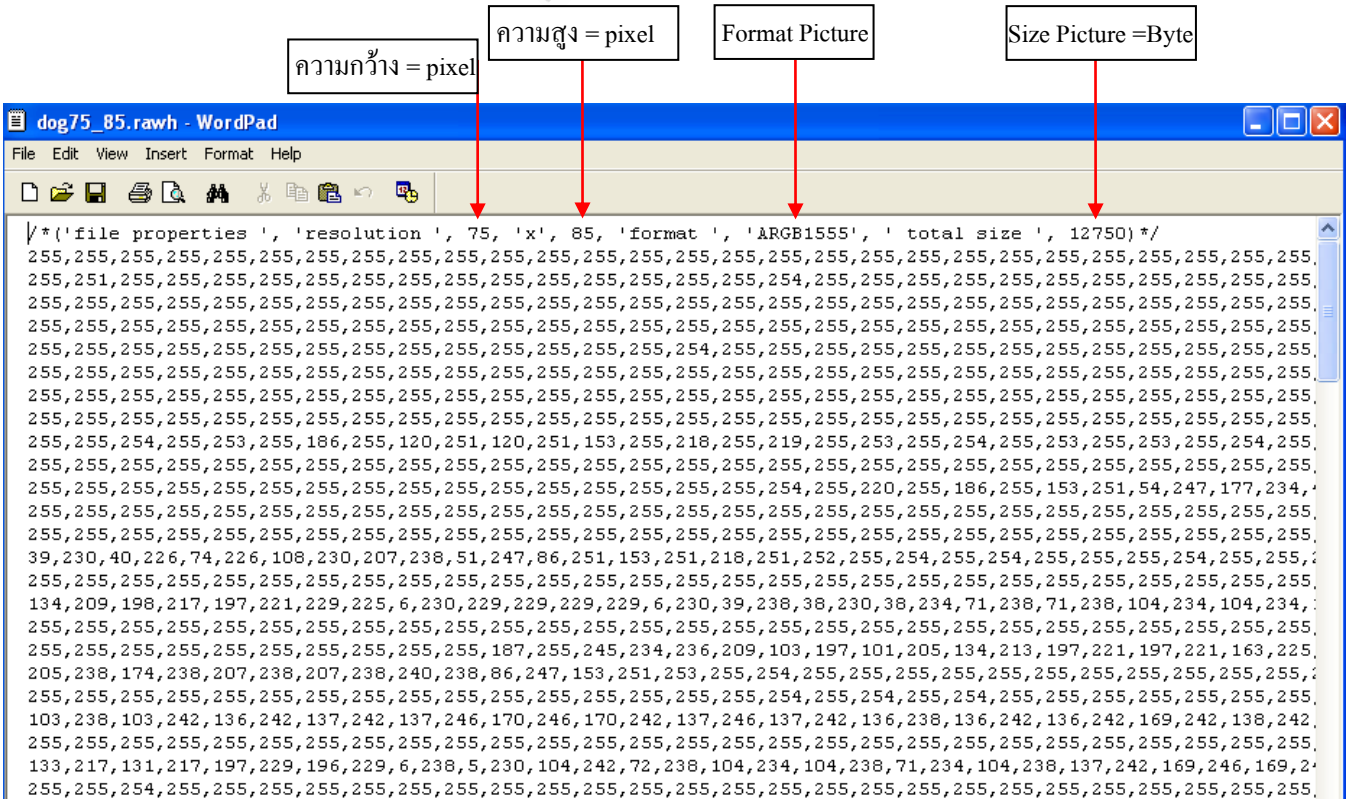


รูป Folder Output จากการ Convert



รูปไฟล์ที่อยู่ใน Folder Output

9) ทำการเปิดไฟล์ที่มีนามสกุล *.rawh ด้วย โปรแกรม WordPad ของ Window โดยดับเบิลคลิกที่ไฟล์ *.rawh จากนั้นก็จะพบ Hex Code ดังรูปด้านล่าง โดยให้ผู้ใช้ Copy Data ทั้งหมดไปวางในโปรแกรมที่ใช้เขียน Control FT800 โดยแนะนำให้วางไว้ในพื้นที่ Flash และกำหนดเป็นตัวแปรชนิด Array ซึ่ง Data นี้จะใช้สำหรับให้ MCU เขียนไปยัง RAM_G ของ FT800 อีกต่อหนึ่ง



```
/*('file properties ', 'resolution ', 75, 'x', 85, 'format ', 'ARGB1555', ' total size ', 12750)*/
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,251,255,255,255,255,255,255,255,255,255,255,255,255,255,255,254,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,254,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,254,255,253,255,186,255,120,251,120,251,153,255,218,255,219,255,253,255,254,255,253,255,253,255,254,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,254,255,220,255,186,255,153,251,54,247,177,234,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
39,230,40,226,74,226,108,230,207,238,51,247,86,251,153,251,218,251,252,255,254,255,254,255,255,255,254,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
134,209,198,217,197,221,229,225,6,230,229,229,229,229,6,230,39,238,38,230,38,234,71,238,71,238,104,234,104,234,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,187,255,245,234,236,209,103,197,101,205,134,213,197,221,197,221,163,225,
205,238,174,238,207,238,207,238,240,238,86,247,153,251,253,255,254,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,254,255,254,255,254,255,255,255,255,255,255,255,
103,238,103,242,136,242,137,242,137,246,170,246,170,242,137,246,137,242,136,238,136,242,136,242,169,242,138,242,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
133,217,131,217,197,229,196,229,6,238,5,230,104,242,72,238,104,234,104,238,71,234,104,238,137,242,169,246,169,
255,255,254,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
```

จากรูปด้านบนนี้ผู้ใช้จะต้องจำความกว้าง , ความสูง และ Format ของรูปที่ Convert มาได้ เพื่อนำไปผ่านค่าให้กับฟังก์ชันสำหรับ Plot รูปคือฟังก์ชัน `void Bitmap_DL(long x , long y , long addr , long format , long line_byte , long width , int height )` โดย

x ,y : คือพิกัด Pixel บนหน้าจอ เป็นตำแหน่งสำหรับเริ่ม Plot รูป โดยจะเริ่ม Plot รูปจาก มุมบนซ้ายมือของหน้าจอ

addr : คือ Address เริ่มต้นของ RAM_G ที่จะเข้าไปอ่าน Data Picture Byte แรก

format : คือรูปแบบของ Picture ที่ เราทำการ Convert มาได้ ในตัวอย่างคือ ARGB1555 ซึ่งแทนค่าด้วย 0

line_byte : คือจำนวน Byte Data ใน 1 Line โดยถ้าเรา Convert รูปมาใน Format RGB565/ARGB4/ARGB1555 ให้แทนด้วย ความกว้าง(Pixel) x 2 ในตัวอย่างคือ 75x2 = 150 ถ้าเป็น Format อื่นให้แทนความกว้างของรูปได้เลย

width , height : คือความกว้างและความสูงของรูป (Pixel) ในตัวอย่างคือ 75 Pixel และ 85 Pixel ตามลำดับ

สำหรับ Size Picture คือจำนวน Data Byte ทั้งหมดของรูป ซึ่งจะใช้ผ่านค่าให้กับฟังก์ชันสำหรับ Write Data ไปเก็บไว้ใน RAM_G โดยฟังก์ชั่น Write Data คือ

void FT800memWrBitmap(unsigned long addr , unsigned long length , char *bitmap) โดย

addr : คือ Address เริ่มต้นใน RAM_G ที่จะเขียน Data เข้าไปเก็บ ซึ่ง Address นี้จะต้องหารด้วย 4 ลงตัวด้วย

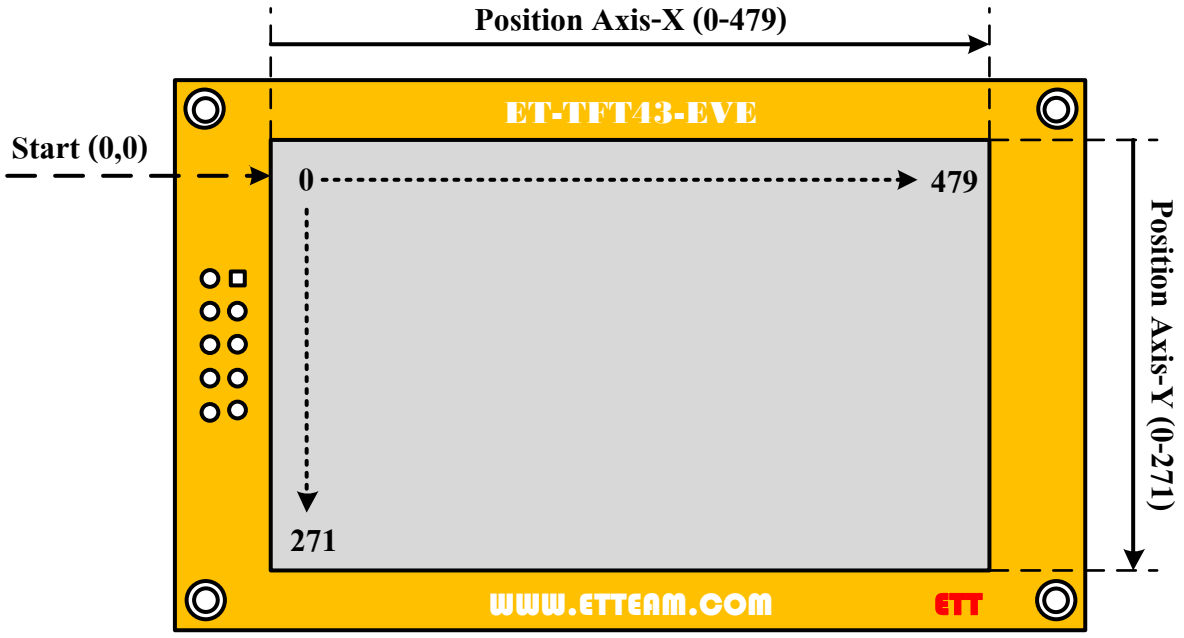
length : คือขนาดจำนวน Data Byte ทั้งหมดของไฟล์ Picture นั้นๆ ที่จะเขียนเข้าไปเก็บใน RAM_G

*bitmap : คือตำแหน่ง Array Pointer ใน MCU ที่ใช้ชี้ไปยัง ตำแหน่ง Array เริ่มต้นของ Data Picture ที่อยู่ใน MCU เพื่อจะใช้เขียนไปยัง RAM_G

สามารถศึกษาตัวอย่างโปรแกรมการนำรูปมาแสดงได้ในตัวอย่างที่4 ของ อีทีที คือ “Ex4_TFT43_Bitmap”

6. ตัวอย่างโปรแกรม

สำหรับตัวอย่างที่ให้มาใน CD นั้นจะมีอยู่ 2 ส่วน คือจะเป็นตัวอย่างที่โหลดมาจาก เว็บไซต์ ผู้ผลิต Chip FT800 ซึ่งจะเป็นตัวอย่างของ Arduino โดยจะไม่ขอกล่าวถึงในที่นี้ให้ผู้ใช้ศึกษาเอง อีกส่วนหนึ่งจะเป็นตัวอย่างของอีทีทีซึ่งเขียนด้วยภาษาซีโดยเขียนไว้กับ MCU ตระกูล AVR(Mega128) เขียนด้วย AVR Studio และ PIC(18F8722) เขียนด้วย CCS ซึ่งในตัวอย่างของอีทีที และตัวอย่างจาก เว็บไซต์ จะเป็นการ Control LCD แนวนอน เนื่องจากคำสั่งที่มีให้ใช้งานของ FT800 ในด้าน Graphic Control Display เป็นคำสั่งที่เกี่ยวกับการแสดงผลในแนวนอนทั้งสิ้น ซึ่งในตัวอย่างทั้งหมดจะอ้างแอดเดรสเริ่มต้น(0,0) ในแนวแกน X,Y ที่มุมบนซ้ายมือ ดังรูปด้านล่าง



รูปแสดงการอ้างตำแหน่ง Pixel เริ่มต้น บนหน้าจอ LCD ในแนวนอน

จากรูปด้านบนเมื่อผู้ใช้นำฟังก์ชันตัวอย่างที่ให้มาไปใช้งาน เวลากำหนดตำแหน่งแอดเดรส X,Y สำหรับเริ่ม Plot รูปหรือตัวอักษร ผู้ใช้ก็จะต้องมองตำแหน่งแอดเดรสอ้างอิงตามรูปด้านบน คือ แกน X = 0-479 pixel และ Y = 0-271 pixel

สำหรับค่าที่อ่านได้จากการ Touch ถ้าใช้ตัวอย่างของอีทีที ค่าในการ Touch ก็จะเริ่มต้นที่ตำแหน่งเดียวกันกับค่าตำแหน่ง Pixel บนหน้าจอ ดังนั้นผู้ใช้สามารถเปรียบเทียบค่าที่ได้จากการ Touch กับค่า ตำแหน่ง Pixel บนหน้าจอตรงๆได้เลย

สำหรับในตัวอย่างของอีทีทีค่า Code สีที่ประกาศในโปรแกรมจะอ้างอิงกับ ค่า Code สีที่แสดงอยู่ท้ายคู่มือ หรือ ใน Data Sheet “RGB_24bit_Color.pdf” สำหรับรายละเอียดของตัวอย่างอีทีทีมีดังนี้

- Ex1_TFT43_Clock ในตัวอย่างนี้เมื่อ Run โปรแกรม เริ่มต้นจะให้ผู้ใช้ทำการ Calibrate หน้าจอ Touch Screen เสียก่อน ด้วยการ Touch ที่จุดบนหน้าจอ 3 จุด เพื่อเวลาใช้งานหลังจาก Calibrate แล้ว เมื่อ Touch หน้าจอจะทำให้ MCU สามารถอ่านตำแหน่ง Touch ได้ถูกต้องตรงกับตำแหน่งพิกัด Pixel ของ LCD หลังจาก Calibrate เรียบร้อยแล้วก็จะปรากฏหน้าจอเป็น นาฬิกาดังรูป ซึ่งผู้ใช้สามารถ ตั้งเวลาได้ด้วยการ Touch ที่ปุ่ม SET ปุ่ม SET ก็จะเปลี่ยนเป็น HH คือตั้งเข็มชั่วโมง จากนั้น Touch ปุ่ม + หรือ - เพื่อทำการตั้งเข็มชั่วโมง จากนั้น Touch ที่ปุ่ม HH อีกครั้ง ปุ่มก็จะเปลี่ยนเป็น MM สำหรับตั้งเข็มนาที สุดท้ายคลิกปุ่ม MM อีกครั้ง ปุ่มก็จะเปลี่ยนเป็น SET นาฬิกาก็จะเริ่มเดิน ถ้าผู้ใช้ต่อลำโพงก็จะได้ยินเสียงนาฬิกาเดินด้วย ส่วนปุ่ม Color เมื่อ Touch ก็จะเป็นการเปลี่ยนสีบนหน้าปัดนาฬิกา สำหรับเวลาการเดินของนาฬิกาจะเป็นการใช้ Delay Timer ในการกำหนด เวลาการเดิน อาจจะไม่แม่นยำนัก

- Ex2_TFT43_KeyDTMF ในตัวอย่างนี้เมื่อเริ่มต้น Run โปรแกรม ก็จะให้ผู้ใช้ Calibrate Touch Screen เหมือนกับตัวอย่างที่ 1 จากนั้นที่หน้าจอก็จะแสดง Key DTMF ดังรูปด้านล่าง เมื่อผู้ใช้กด Key ใด ๆ ก็จะได้ยินเสียง DTMF ของ Keyนั้นเกิดขึ้น ตัวอักษรที่ Key ก็จะปรากฏในตำแหน่งแสดงผล ไฟแสดงสัญลักษณ์การกดด้านซ้ายมือก็จะกระพริบ โดยกำหนดให้ช่องแสดงผลแสดงอักษรได้ 10 ตัว เมื่อเต็มแล้วเวลากด Key ก็จะไม่มีผลใดๆ จะต้อง Touch ปุ่ม C เพื่อทำการ Clear ส่วนช่องแสดงผล

- Ex3_TFT43_RegTouch สำหรับตัวอย่างที่ 3 นี้จะเขียนขึ้นเพื่อแก้ปัญหาการ Calibrate Touch Screen เวลาใช้งานจริง กล่าวคือผู้ใช้จะเห็นว่าจาก 2 ตัวอย่างแรก ทุกครั้งที่เรา Run โปรแกรมขึ้นมาผู้ใช้จะต้องทำการ Calibrate Touch Screen ทุกครั้ง มิเช่นนั้นเราก็จะไม่สามารถใช้งานในส่วนของ Touch Screen ได้ คือ เวลา Touch จะไม่ตรงกับตำแหน่งที่กำหนดไว้บนหน้าจอถ้าไม่ทำการ Calibrate เสียก่อน

เพื่อแก้ปัญหาการ Calibrate ทุกครั้งที่ทำการ Run โปรแกรมที่ผู้ใช้เขียนขึ้น ก็สามารถทำได้โดย Run ตัวอย่างโปรแกรมที่ 3.1 เพื่ออ่านค่าสัมประสิทธิ์ ของ REG_TOUCH_TRANSFORM_A -F ออกมาหลังจากที่ทำการ Calibrate ซึ่งจะแสดงให้ผู้ใช้เห็นบน Display จากนั้นให้ผู้ใช้จดค่าสัมประสิทธิ์ที่ได้ของ Register ต่างๆไว้ ถ้าเป็นค่าติดลบก็ให้จดเครื่องหมายลบไว้ด้วย จากนั้นเวลาที่ผู้ใช้เขียนโปรแกรม Control Display ผู้ใช้ก็ไม่จำเป็นต้องเรียกฟังก์ชัน Calibrate Touch Screen มาใช้งานอีก ทำเพียงเขียนค่าสัมประสิทธิ์ที่อ่านได้จากตัวอย่างโปรแกรมที่ 3.1 ไปยัง Register REG_TOUCH_TRANSFORM_A-F ซึ่งได้แสดงตัวอย่างการนำค่าสัมประสิทธิ์ไปใช้งานไว้ในตัวอย่างโปรแกรมที่ 3.2 โดยจะเห็นว่าในตัวอย่างที่ 3.2 เมื่อ Run โปรแกรมผู้ใช้ไม่ต้อง Calibrate Touch Screen อีกแต่สามารถใช้งานส่วน Touch Screen ได้ตามปกติ โดยวิธีที่กล่าวมานี้จะต้องทำจอต่อจอเท่านั้น ถ้าเปลี่ยนจอใหม่ก็ต้องทำการอ่านค่าสัมประสิทธิ์ของจอใหม่ด้วย ไม่แนะนำให้ใช้ค่าสัมประสิทธิ์ที่อ่านได้จากจอใดจอหนึ่งไปใช้ร่วมกับทุกจอเพราะอาจทำให้เกิดความผิดพลาดในการ Touch ได้ โดยตัวอย่าง ที่กล่าวไปข้างต้นทั้งสองตัวอย่างมีการทำงานดังนี้

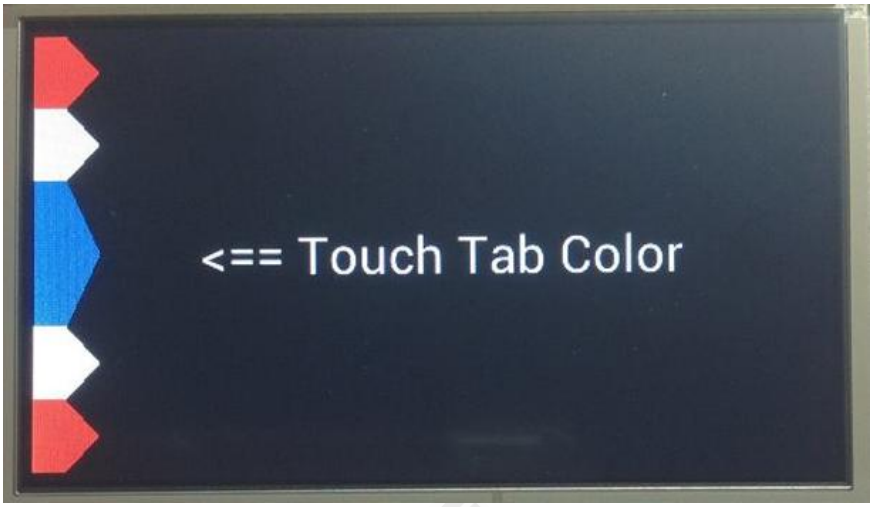
3_1_RdReg_Touch ในตัวอย่างนี้จะเป็นการอ่านค่าสัมประสิทธิ์จาก Register REG_TOUCH_TRANSFORM_A-F ซึ่งค่านี้จะได้หลังจากที่ผู้ใช้ทำการ Calibrate โดยเมื่อ Run โปรแกรม เริ่มต้นจะให้ผู้ใช้ทำการ Calibrate หน้าจอ Touch Screen ด้วยการ Touch ที่จุดบนหน้าจอ 3 จุด จากนั้นโปรแกรมก็จะเข้าไปอ่านค่าจาก Register ที่กล่าวไปข้างต้น 7 ตัว (A-F) และนำค่าจาก Register แต่ละตัวที่อ่านได้มาแสดงให้ผู้ใช้เห็นบนหน้าจอ ดังแสดงในรูป ให้ผู้ใช้จดค่าที่ได้นี้ไปใช้งาน (ไม่ใช่ค่าที่แสดงในตัวอย่าง ต้องเป็นค่าที่ผู้ใช้อ่านได้จากจอของผู้ใช้เอง)

โดยจากรูปจะเห็นว่าเวลาจะนำค่าสัมประสิทธิ์ไปใช้งานในตัวอย่างที่ 3.2 นั้น ค่าที่อ่านได้จะแบ่งเป็น 3 ส่วนคือ ส่วนที่ 1 คือเครื่องหมาย(Sig) โดย + ให้แทนด้วยเลข 0 ส่วนเครื่องหมาย - ให้แทนด้วยเลข 1 ส่วนที่2 คือ ส่วนตัวเลขที่เป็นจำนวนเต็มที่อยู่หน้าจุด(Int) และส่วนที่ 3 คือส่วนที่เป็นเลขทศนิยม(Frac) ที่อยู่หลังจุด โดยค่าที่แสดงนี้จะเป็นเลขฐานสิบ

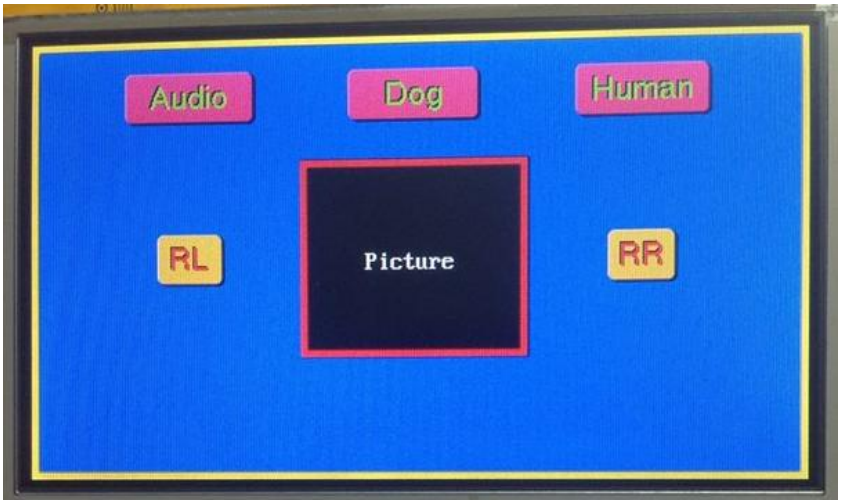
3_2_WrReg_Touch สำหรับตัวอย่างนี้จะเป็นการนำค่าสัมประสิทธิ์ที่ได้จากการ Calibrate Touch Screen ในตัวอย่างที่ 3.1 ไปใช้งาน ซึ่งการทำงานของตัวอย่างที่3.2 เมื่อ Run โปรแกรม จะไม่มีการให้ผู้ใช้ทำการ Calibrate Touch Screen อีก แต่จะให้โปรแกรมไปทำฟังก์ชัน void Initial_Calibrate (void) แทน ซึ่งเป็นฟังก์ชันสำหรับเขียนค่าสัมประสิทธิ์ที่อ่านมาได้ไปยัง REG_TOUCH_TRANSFORM_A-F เมื่อโปรแกรมเขียนค่าสัมประสิทธิ์ไปยัง Register เรียบร้อย ก็จะแสดงหน้าจอดังรูปด้านล่าง ให้ผู้ใช้ทำการ Touch ที่ Tab สีแต่ละ Tab จนครบก็จะได้สีออกมาเป็นลายธงชาติเต็มจอจากนั้นก็จะมีเสียง Melody เพลงชาติไทยดังขึ้น สำหรับค่าสัมประสิทธิ์ที่ได้ในตัวอย่างที่ 3.1 เราจะเอาไปแทนไว้ในตัวแปรที่ประกาศไว้เหนือ main() ดังนี้

```
unsigned char sigA=0 ,sigB=1 ,sigC=1 ,sigD=1 ,sigE=1 ,sigF=0  ;  //Set Signed + = 0  and  -  =  1
unsigned int  intA  = 0          ;      //Set value integer of Reg.Touch Transfrom A  *****Data Integer*****
unsigned int  intB  = 32767   ;      //Set value integer of Reg.Touch Transfrom B
unsigned int  intC  = 32756   ;      //Set value integer of Reg.Touch Transfrom C
unsigned int  intD  = 32767   ;      //Set value integer of Reg.Touch Transfrom D
unsigned int  intE  = 32767   ;      //Set value integer of Reg.Touch Transfrom E
```

```
unsigned int  intF  = 283      ;      //Set value integer of Reg.Touch Transfrom F
unsigned int  fracA = 32291    ;      //Set value fraction of Reg.Touch Transfrom A    *****Data Fraction*****
unsigned int  fracB = 65507    ;      //Set value fraction of Reg.Touch Transfrom B
unsigned int  fracC = 18121    ;      //Set value fraction of Reg.Touch Transfrom C
unsigned int  fracD = 65429    ;      //Set value fraction of Reg.Touch Transfrom D
unsigned int  fracE = 46365    ;      //Set value fraction of Reg.Touch Transfrom E
unsigned int  fracF = 49006    ;      //Set value fraction of Reg.Touch Transfrom F
```



- Ex4_TFT43_Bitmap ในตัวอย่างนี้จะเป็นการนำรูปภาพจากภายนอกเข้ามาแสดงที่ Display และสามารถกดหมุนรูปภาพซ้าย-ขวาได้ โดยรูปภาพที่นำมาแสดงจะต้องทำการ Convert ด้วยโปรแกรม " img_cvt_0.3" โดยเลือก Format ARGB1555 เพื่อให้ได้ Hex Code ออกมาก่อน เริ่มต้นเมื่อ Run โปรแกรม จะให้ผู้ใช้ทำการ Calibrate หน้าจอ Touch Screen เสียก่อน เหมือนตัวอย่างที่ 1 และ 2 จากนั้นโปรแกรมก็จะทำการ Write Data Picture จำนวน 3 รูป ไปเก็บไว้ยัง RAM_G ก่อน จากนั้นก็จะแสดงหน้าจอดังรูปด้านล่าง เมื่อผู้ใช้ Touch ปุ่มด้านบน ภาพของปุ่มนั้นๆก็จะแสดงในช่อง Picture จากนั้นลอง Touch ที่ปุ่ม RL หรือ RR ค้างไว้ รูปในช่อง Picture ก็จะหมุนตามปุ่มที่ Touch ขณะหมุนก็จะมีเสียง Melody ด้วย



THE -END ET-TFT43-EVE

รหัสสีที่ใช้กับตัวอย่างอีทีที

Snow #FFFAFA	GhostWhite #F8F8FF	WhiteSmoke #F5F5F5	Gainsboro #DCDCDC
FloralWhite #FFFAF0	OldLace #FDF5E6	Linen #FAF0E6	AntiqueWhite #FAEBD7
PapayaWhip #FFEFD5	BlanchedAlmond #FFEBCD	Bisque #FFFAF0	PeachPuff #FFDAB9
NavajoWhite #FFDEAD	Moccasin #FFE4B5	Cornsilk #FFF8DC	Ivory #FFFFF0
LemonChiffon #FFFACD	Seashell #FFF5EE	Honeydew #F0FFF0	MintCream #F5FFFA
Azure #F0FFFF	AliceBlue #F0F8FF	lavender #E6E6FA	Lavender Blush #FFF0F5
MistyRose #FFE4E1	White #FFFFFF	Black #000000	DarkSlateGray #2F4F4F
LightSlateGray #778899	Gray #BEBEBE	LightGray #D3D3D3	MidnightBlue #191970
CornflowerBlue #6495ED	DarkSlateBlue #483D8B	SlateBlue #6A5ACD	MediumSlateBlue #7B68EE
LightSlateBlue #8470FF	NavyBlue #000080	DimGrey #696969	SlateGrey #708090
MediumBlue #0000CD	RoyalBlue #4169E1	Blue #0000FF	DodgerBlue 1E90FF
DeepSkyBlue #00BFFF	SkyBlue #87CEEB	LightSkyBlue #87CEFA	SteelBlue #4682B4

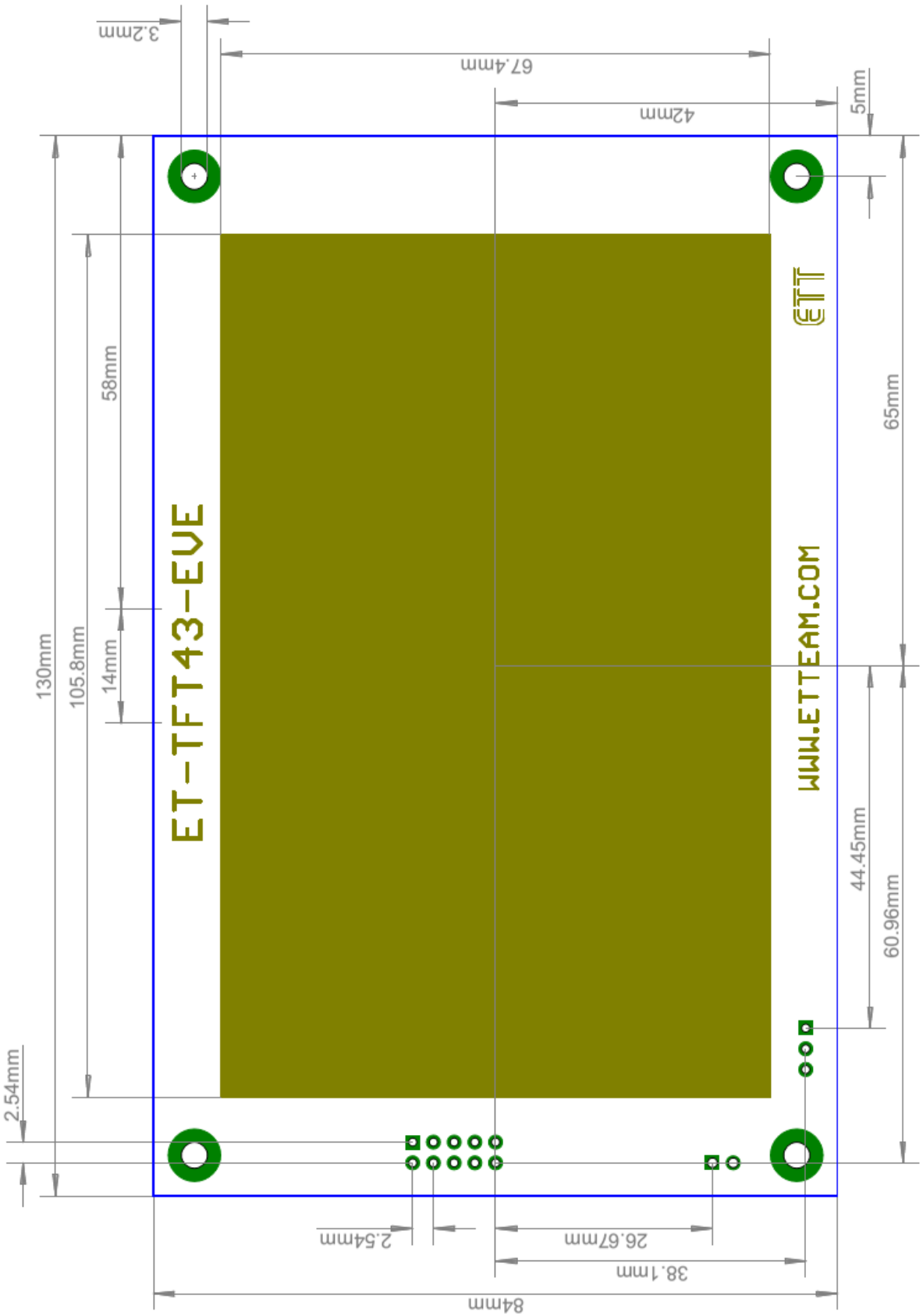
24 Bit Color RGB (1)

LightSteelBlue #B0C4DE	LightBlue #ADD8E6	PowderBlue #B0E0E6	PaleTurquoise #AFEEEE
DarkTurquoise #00CED1	MediumTurquoise #48D1CC	Turquoise #40E0D0	Cyan #00FFFF
LightCyan #E0FFFF	CadetBlue #5F9EA0	MediumAquamarine #66CDAA	Aquamarine #7FFFD4
DarkGreen #006400	DarkOliveGreen #556B2F	DarkSeaGreen #8FBC8F	SeaGreen #2E8B57
MediumSeaGreen #3CB371	LightSeaGreen #20B2AA	PaleGreen #98FB98	SpringGreen #00FF7F
LawnGreen #7CFC00	Green #00FF00	Chartreuse #7FFF00	MedSpringGreen #00FA9A
GreenYellow #ADFF2F	LimeGreen #32CD32	YellowGreen #9ACD32	ForestGreen #228B22
OliveDrab #6B8E23	DarkKhaki #BDB76B	PaleGoldenrod #EEE8AA	LtGoldenrodYello #FAFAD2
LightYellow #FFFFE0	Yellow #FFFF00	Gold #FFD700	LightGoldenrod #EEDD82
Goldenrod #DAA520	DarkGoldenrod #B8860B	RosyBrown #BC8F8F	IndianRed #CD5C5C
SaddleBrown #8B4513	Sienna #A0522D	Peru #CD853F	Burlywood #DEB887
Beige #F5F5DC	Wheat #F5DEB3	SandyBrown #F4A460	Tan #D2B48C

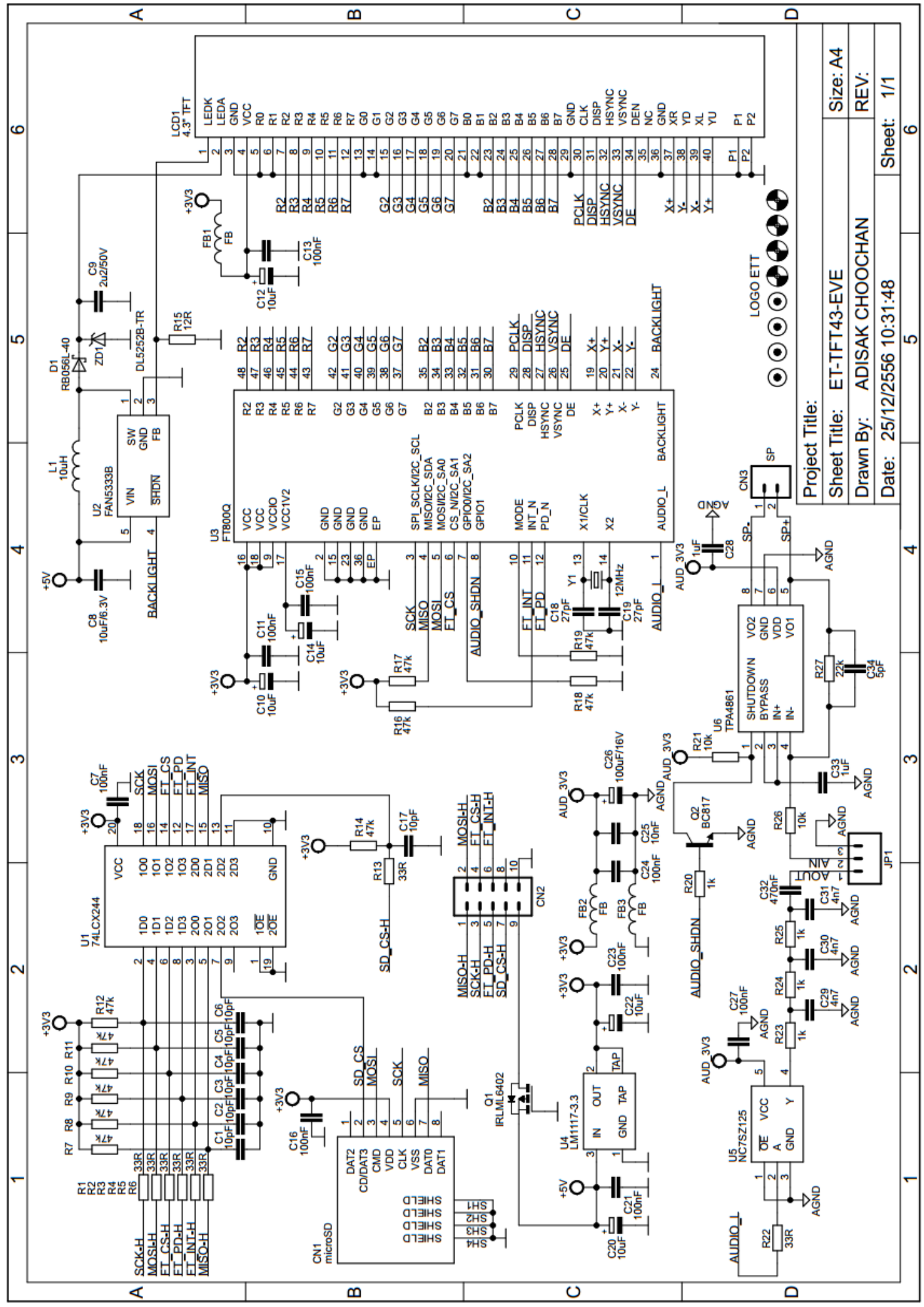
24 Bit Color RGB(2)

Chocolate #D2691E	Firebrick #B22222	Brown #A52A2A	DarkSalmon #E9967A
Salmon #FA8072	LightSalmon #FFA07A	Orange #FFA500	DarkOrange #FF8C00
Coral #FF7F50	LightCoral #F08080	Tomato #FF6347	OrangeRed #FF4500
Red #FF0000	HotPink #FF69B4	DeepPink #FF1493	Pink #FFC0CB
MediumVioletRed #C71585	VioletRed #D02090	Magenta #FF00FF	Violet #EE82EE
Plum #DDA0DD	LightPink #FFB6C1	PaleVioletRed #DB7093	Maroon #B03060
Orchid #DA70D6	MediumOrchid #BA55D3	DarkOrchid #9932CC	DarkViolet #9400D3
BlueViolet #8A2BE2	Purple #A020F0	MediumPurple #9370DB	Thistle #D8BFD8

24 Bit Color RGB(3)



รูปแสดงขนาดของจอและ PCB ET-TFT 43-EVE


Project Title:
Sheet Title: ET-TFT43-EVE
Size: A4
Drawn By: ADISAK CHOOCHAN
REV:
Date: 25/12/2556 10:31:48
Sheet: 1/1
LOGO ETT
LCD1 4.3" TFT
U3 FT800Q
U1 74LCX244
U2 FAN5333B
U4 LM1117-3.3
U5 NC7SZ125
U6 TPA4861
Q1 IRLML6402
Q2 BC817
CN1 microSD
CN2
CN3 SP
JP1
D1 RB056L-40
ZD1 DL5252B-TR
L1 10uH
FB1 FB
FB2 FB
FB3 FB
Y1 12MHz
+5V
+3V3
AUD_3V3
AGND
BACKLIGHT
AUDIO_L
AUDIO_SHDN
SCK
MISO
MOSI
FT_CS
FT_PD
FT_INT
SD_CS
SCK-H
MISO-H
MOSI-H
FT_CS-H
FT_PD-H
FT_INT-H
SD_CS-H