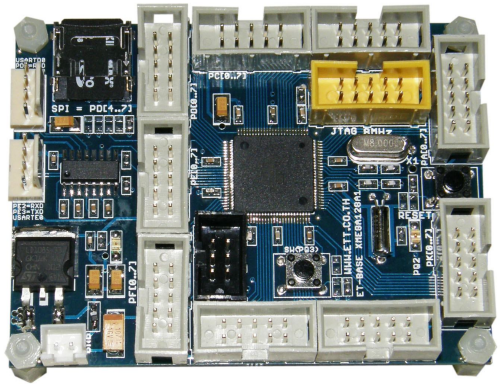
ET-BASE-xMEGA128A1



ไมโครคอนโทรลเลอร์ตระกูล AVR ของ ATMEL เป็นไมโครคอนโทรลเลอร์อีกตระกูลหนึ่งซึ่งได้รับความนิยมอย่างแพร่หลายจากผู้ใช้งานทั่วไป ซึ่งทาง ATMEL เองก็ได้มีการปรับปรุง พัฒนาขีดความสามารถของ MCU เพื่อตอบสนองความต้องการใช้งานในลักษณะต่างๆ มีการผลิตชิพ MCU ออกมาจำหน่ายเป็นจำนวนมากมายหลายเบอร์ เพื่อให้ผู้ใช้งานสามารถเลือก MCU ไปประยุกต์ใช้งานให้เหมาะสมกับงานได้ง่ายและสะดวกมากยิ่งขึ้น

มาถึงวันนี้ทาง ATMEL ได้ปรับปรุงพัฒนาขีดความสามารถของ MCU ตระกูล AVR ขึ้นมาอีกขั้นหนึ่ง โดยให้ชื่อว่าตระกูล “xMEGA” ซึ่งเป็น MCU ตระกูล AVR ใหม่ล่าสุด ที่มีความโดดเด่นและมีขีดความสามารถสูงขึ้นกว่าเดิม AVR รุ่นเก่าๆที่ผ่านมา แต่ยังคงใช้ชุดคำสั่งต่างๆเหมือนเดิมกับ AVR เพียงแต่มุ่งเน้นปรับปรุงขีดจำกัดทางฮาร์ดแวร์และเพิ่มขีดความสามารถให้มากยิ่งขึ้น ทั้งเรื่องความเร็วในการประมวลผล ระบบทรัพยากรภายใน และการประหยัดพลังงาน จึงเหมาะอย่างยิ่งสำหรับนักพัฒนาโดยเฉพาะผู้ที่มีความคุ้นเคยกับการใช้งาน MCU ตระกูล AVR อยู่แล้วที่จะสามารถต่อยอดใช้งาน MCU ตระกูล xMEGA จากพื้นฐานของ AVR รุ่นเก่าๆได้โดยง่ายดาย

โดย xMEGA มีความโดดเด่นในหลายๆด้าน เช่น ระบบสัญญาณนาฬิกาได้รับการพัฒนาให้มีเสถียรภาพมากขึ้นกว่า AVR MEGA รุ่นเก่า สามารถเลือกกำหนดสัญญาณนาฬิกา สามารถสลับเปลี่ยนค่าความถี่และเลือกแหล่งกำเนิดสัญญาณนาฬิกาได้จากโปรแกรมในขณะทำงานได้ตลอดเวลา ที่สำคัญคือมีระบบตรวจสอบความผิดพลาด เมื่อสัญญาณนาฬิกาภายนอกทำงานผิดพลาด MCU จะสลับไปทำงานด้วยระบบสัญญาณนาฬิกาภายในได้เองโดยอัตโนมัติ ทำให้ระบบมีเสถียรภาพมากยิ่งขึ้น และในส่วนของ GPIO ต่างๆก็ได้รับการปรับปรุงให้สามารถเข้าถึงได้อย่างรวดเร็ว สามารถโปรแกรมฟังก์ชั่นการทำงานได้มากมายหลายแบบ เช่น ใช้เป็น Input ตรวจจับการเปลี่ยนแปลง Input และกระตุ้น Interrupt หรือใช้เป็น Output โดยสามารถเลือกโปรแกรมการ Pull-Up, Pull-Down, Push Pull, Bus Keeper ได้ ทำให้ระบบมีขนาดเล็กลง และประหยัดพลังงานมากยิ่งขึ้น นอกจากนี้แล้วยังมีความสามารถพิเศษอื่นๆอีกหลายอย่างที่ xMEGA ได้รับการพัฒนาขึ้น เช่น Interrupt และ DMA เป็นต้น

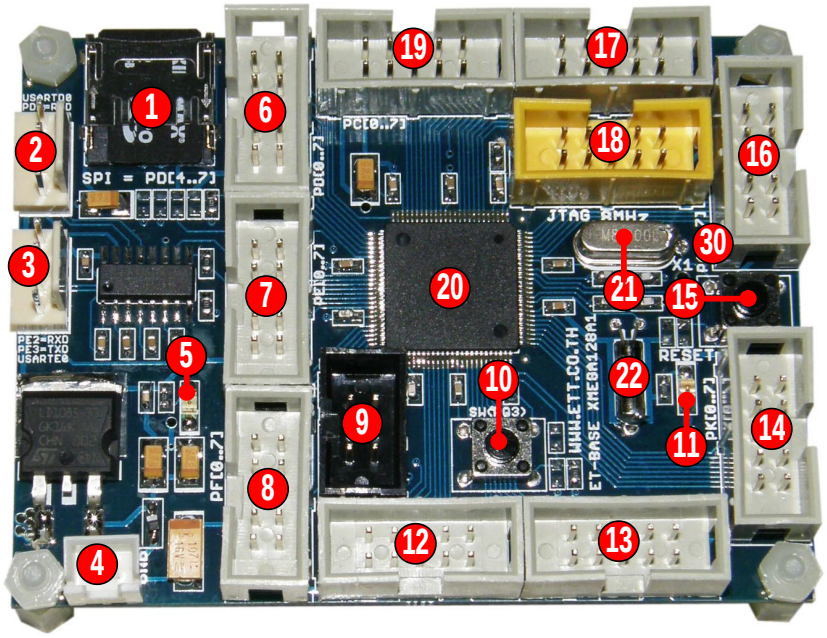
คุณสมบัติของ MCU ATxMEGA128A1

- 128KB Flash / 8KB SRAM / 2KB EEPROM
- ทำงานที่แรงดัน 1.6V-3.6V (Run 32MHz ที่ 2.7V-3.6V, Run 12MHz ที่ 1.6V)
- มีวงจร Internal Clock ภายใน พร้อม PLL สามารถโปรแกรมค่าได้สูงสุด 32MHz
- 4 Channel DMA
- 8 ช่อง 16Bit Timer/Counter
- 8 ช่อง USART และสามารถโปรแกรมเป็น IrDA modulation/demodulation ได้ 1 ช่อง
- 4 ช่อง I2C สามารถโปรแกรมสัญญาณเป็นแบบ SM-Bus compatible ได้
- 4 ช่อง SPI
- 16 Bit RTC
- 16 ช่อง 12Bit ADC(2MSPS)
- 4 ช่อง 12 Bit DAC(1MSPS)
- 4 ช่อง Analog Comparator
- มีวงจรเข้ารหัส ถอดรหัสข้อมูลแบบ AES(Advanced Encryption Standard) และ DES(Data Encryption Standard)
- สามารถโปรแกรม ระดับความสำคัญของการเกิด Interrupt ได้
- สามารถสร้าง External Interrupt ผ่าน GPIO Pin ได้ทุก Pin
- มีระบบ JTAG(IEEE 1149.1 Compliant) สำหรับ Program และ Debug
- มี PDI (Program and Debug Interface) สำหรับ program และ Debug

คุณสมบัติของบอร์ด ET-BASE-XMEGA128A1

1. ใช้ MCU ตระกูล xMEGA AVR เบอร์ ATxMEGA128A1 ของ ATMEL
2. มีหน่วยความจำ Flash 128KB, 8KB Boot loader, Static RAM 8KB และ EEPROM 2KB
3. ใช้ Crystal 8.00 MHz โดย MCU สามารถประมวลผลด้วยความเร็วสูงสุดที่ 32MHz เมื่อใช้งานร่วมกับ Phase-Locked Loop (PLL) ภายในตัว MCU เอง
4. มีวงจร RTC(Real Time Clock) พร้อม XTAL ค่า 32.768KHz
5. รองรับการโปรแกรมแบบ In-System Programming แบบ PDI
6. มีวงจรเชื่อมต่อกับ AVR-JTAG ขนาด 10 Pin เพื่อทำการ Debug แบบ Real Time ได้
7. Power Supply ใช้แรงดันไฟฟ้า +5VDC พร้อมวงจร Regulate +3V3/3A ภายในบอร์ด
8. มีวงจรเชื่อมต่อการ์ดหน่วยความจำแบบ SD Card(Micro SD) เชื่อมต่อแบบ SPI จำนวน 1 ช่อง
9. มีวงจรสื่อสาร RS232 โดยใช้ขั้วต่อแบบ 4-PIN มาตรฐาน ETT จำนวน 2 ช่อง
10. มีวงจร Push Button Switch(PQ3) จำนวน 1 ชุด พร้อมสวิตช์ RESET
11. มีวงจร LED แสดงสถานะเพื่อทดลอง Output(PQ2) จำนวน 1 ชุด
12. มี 72 Bit GPIO อิสระ สำหรับประยุกต์ต่างๆ เช่น A/D,D/A,I2C,SPI,USART และ Input / Output แบบต่างๆ โดยมีการจัดสรรใช้งานภายในบอร์ดไว้แล้ว จำนวน 8 เส้นสัญญาณ คือ PD[4..7] สำหรับ micro SD Card, PD[2..3] และ PE[2..3] สำหรับ RS232 แต่สัญญาณทั้ง 8 เส้นดังกล่าวยังมีการเชื่อมต่อสัญญาณออกมาไว้ที่ขั้วต่อ 10PIN IDE ของพอร์ตด้วย โดยใช้ขั้วต่อสัญญาณแบบ 10PIN IDE จำนวน 9 ชุด มีดังนี้
    a. Header 10Pin IDE (PA[0..7]) สำหรับ GPIO
    b. Header 10Pin IDE (PB[0..7]) สำหรับ GPIO(PB[4..7] ถูกใช้สำหรับ AVR-JTAG)
    c. Header 10Pin IDE (PC[0..7]) สำหรับ GPIO
    d. Header 10Pin IDE (PD[0..7]) สำหรับ GPIO(PD[2..3] ถูกใช้สำหรับ USARTD0 และ PD[4..7] ถูกใช้สำหรับ SPID ในการเชื่อมต่อกับ micro SD Card
    e. Header 10Pin IDE (PE[0..7]) สำหรับ GPIO (PE[2..3] ถูกใช้สำหรับ USARTE0)
    f. Header 10Pin IDE (PF[0..7]) สำหรับ GPIO
    g. Header 10Pin IDE (PH[0..7]) สำหรับ GPIO
    h. Header 10Pin IDE (PJ[0..7]) สำหรับ GPIO
    i. Header 10Pin IDE (PK[0..7]) สำหรับ GPIO

โครงสร้างบอร์ด ET-BASE xMEGA128A1

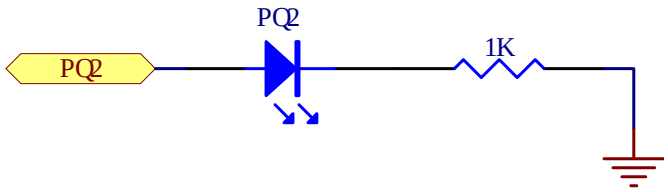


รูปแสดง โครงสร้างของบอร์ด ET-BASE xMEGA128A1

- **หมายเลข 1 คือ** ช่องเสียบการ์ดหน่วยความจำสามารถใช้ได้กับ SD Card แบบ Micro-SD
- **หมายเลข 2,3 คือ** ขั้วต่อ UARTD0(RS232) และ USARTE0(RS232) สำหรับใช้งาน
- **หมายเลข 4,5 คือ** ขั้วต่อแหล่งจ่ายไฟเลี้ยงวงจรของบอร์ดใช้ได้กับไฟ +5VDC และ LED Power
- **หมายเลข 6,7,8 คือ** ขั้วต่อ GPIO(PD[0..7]) , GPIO(PE[0..7]) และ GPIO(PF[0..7])
- **หมายเลข 9 คือ** ขั้วต่อ PDI สำหรับ Download โปรแกรม
- **หมายเลข 10 คือ** SW Push Button เชื่อมต่อกับสัญญาณ PQ3
- **หมายเลข 11 คือ** LED ใช้ทดสอบ Logic Output ของ PQ2
- **หมายเลข 12,13,14 คือ** ขั้วต่อ GPIO(PH[0..7]), GPIO(PJ[0..7]) และ GPIO(PK[0..7])
- **หมายเลข 15 คือ** SW RESET
- **หมายเลข 16,17 คือ** ขั้วต่อ GPIO(PA[0..7]) และ GPIO(PB[0..7])
- **หมายเลข 18 คือ** ขั้วต่อ AVR-JTAG สำหรับ Debug แบบ Real Time
- **หมายเลข 19 คือ** ขั้วต่อ GPIO(PC[0..7])
- **หมายเลข 20 คือ** MCU เบอร์ ATxMEGA128A1 (100Pin TQFP)
- **หมายเลข 21 คือ** Crystal ค่า 8 MHz สำหรับใช้เป็นฐานเวลาระบบให้ MCU
- **หมายเลข 22 คือ** Crystal ค่า 32.768KHz สำหรับฐานเวลาให้ RTC ภายในตัว MCU

การใช้งานวงจรขับ LED แสดงผล

LED แสดงผลของบอร์ด จะต่อวงจรแบบขับกระแส (Source Current) โดยใช้กับแหล่งจ่าย +3.3V ทำงานด้วยโลจิก “1” (+3V3) และหยุดทำงานด้วยโลจิก “0” (0V) โดยควบคุมการทำงานจากขาสัญญาณ PQ2 โดยวงจรในส่วนนี้จะใช้สำหรับทดสอบการทำงานของ Output จากขาสัญญาณ PQ2



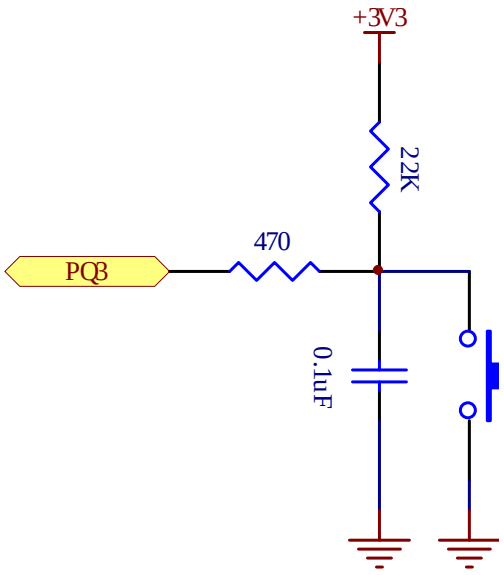
โดยเมื่อต้องการใช้งานผู้ใช้ต้องกำหนดให้PQ[2] ทำหน้าที่เป็น GPIO Output Port เสียก่อนแล้วจึงควบคุม Logic ให้กับ PQ[2] ตามต้องการ ดังตัวอย่าง

```
#include "port_driver.h"                  //Port Library
.
.
.
int main(void)
{
  PORT_SetPinsAsOutput(&PORTQ,(1<<2)); //PQ2 = Output
   .
   .
   .
  PORT_SetPins(&PORTQ,(1<<2));         //PQ2= High
   .
   .
   .
  PORT_ClearPins(&PORTQ,(1<<2));       //PQ2 = Low
   .
   .
   .
  PORT_TogglePins(&PORTQ,(1<<2));      //Toggle PQ2
   .
   .
   .
}
```

ตัวอย่าง การกำหนดค่าการใช้งาน PQ2 เป็น Output LED

การใช้งานวงจร Push Button Switch

วงจร Push Button Switch จะใช้วงจร Switch แบบ กดติด-ปล่อยดับ (Push Button) พร้อมวงจร Pull-Up ใช้กับแหล่งจ่าย +3.3V โดยในขณะที่สวิตช์ยังไม่ถูกกดจะให้ค่าสถานะเป็นโลจิก “1” แต่เมื่อสวิตช์ถูกกดอยู่จะให้สถานะเป็นโลจิก “0” ใช้สำหรับทดสอบการทำงานของ Input Logic โดยวงจรส่วนนี้จะใช้ PQ3 ในการเชื่อมต่อเพื่ออ่านสถานะจากสวิตช์




```
#include "port_driver.h"                  //Port Library
.
.
.
int main(void)
{
  PORT_SetPinsAsInput (&PORTQ,(1<<3));  //PQ3 = Input

  if((PORTQ.IN&(1<<3))==(1<<3))         //If PQ3 = High
  {
    …
  }
  else                                  //If PQ3 = Low
  {
    …
  }
  .
  .
  .
}
```

ตัวอย่าง การกำหนดค่าการใช้งาน **PQ3** เป็น **Input Switch**

นอกจากจะใช้วิธีการอ่านค่า Input Logic จาก Switch ด้วยวิธีการ Polling อ่านค่าสถานะจาก Pin โดยตรงแล้ว ความสามารถของ GPIO ของ ATxMEGA128A1 ยังสามารถกำหนดหน้าที่ให้เป็น Input Pin แบบ Interrupt ได้อีกด้วย

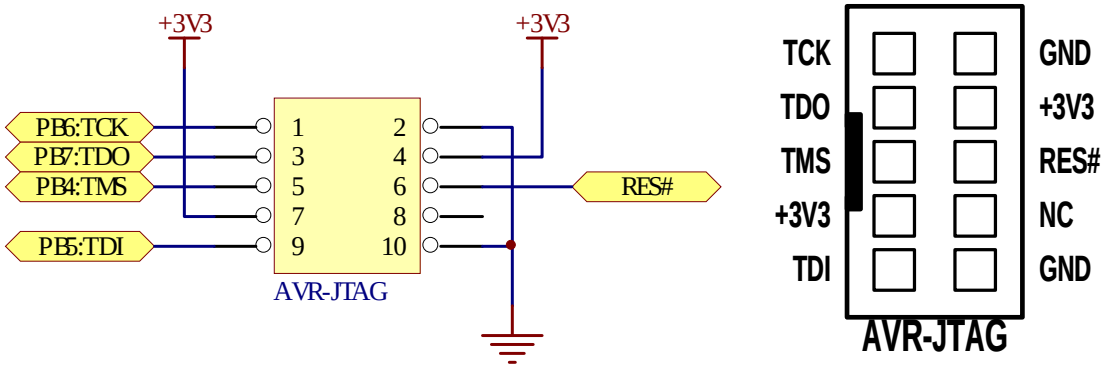
```
#include "avr_compiler.h"
#include "port_driver.h"

int main(void)
{
  /* PQ3 = Input, Triggered on Rising Edge. */
  PORT_ConfigurePins( &PORTQ,                          //Port = PQ
                      (1<<3),                          //Pin  = PQ3
                      false,                           //Disable slewRate
                      false,                           //Disable invert
                      PORT_OPC_TOTEM_gc,               //Push-Pull Output
                      PORT_ISC_RISING_gc );            //Rising Edge

  /* PQ3 = Input SW */
  PORT_SetPinsAsInput( &PORTQ,(1<<3));                 //PQ3 = Input

  /* PQ2 = Output LED */
  PORT_SetPinsAsOutput(&PORTQ,(1<<2));                 //PQ2 = Output

  /* ON LED */
  PORT_SetPins(&PORTQ,(1<<2));                         //PQ2 = High

  /* INT0 = Medium Level, Triggered by PQ3 */
  PORT_ConfigureInterrupt0(&PORTQ,PORT_INT0LVL_MED_gc,(1<<3));

  /* Enable Medium Level Interrupts in the PMIC. */
  PMIC.CTRL |= PMIC_MEDLVLEN_bm;

  /* Enable the Global Interrupt flag. */
  sei();

  //Loop Continue Wait Interrupt
  while(1);
}

/*********************************************/
/*  PQ3 Interrupt on every Rising Edge on PQ3.*/
/*********************************************/
ISR(PORTQ_INT0_vect)
{
  PORT_TogglePins( &PORTQ, (1<<2));                    //Toggle LED(PQ2)
}
```

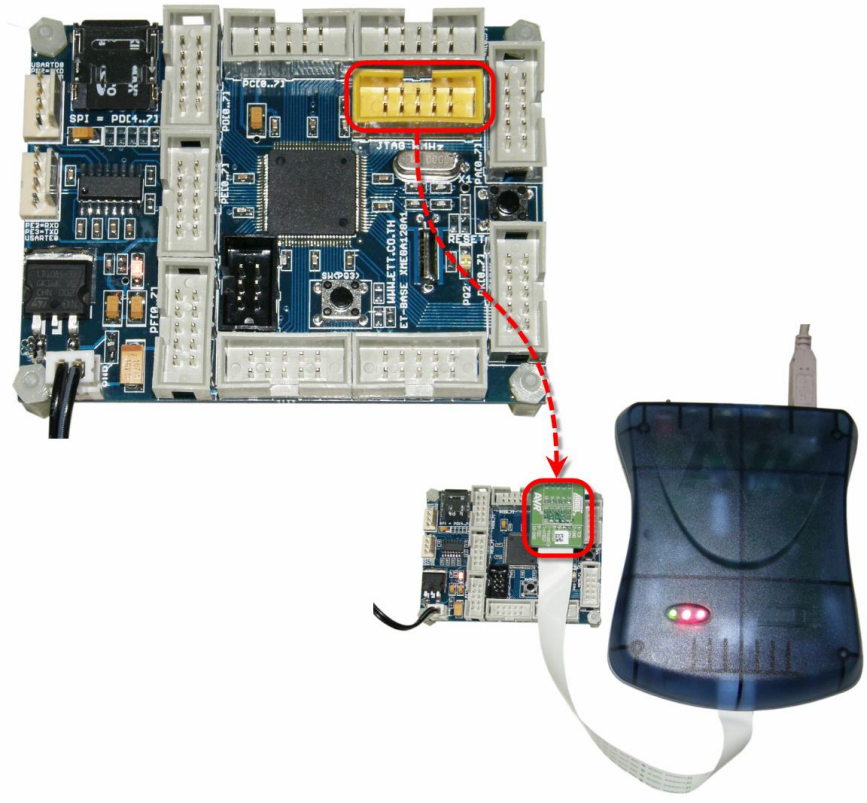
ตัวอย่างการใช้ SW(PQ3) แบบ Interrupt Trigger สำหรับ ON/OFF LED จาก PQ2

โดยจากตัวอย่างเมื่อกดสวิตช์ PQ3 จะทำให้เกิดการ Interrupt ส่งผลให้เกิดการสลับสถานะของ Output PQ2 ทำให้ LED(PQ2) เกิดการ ON/OFF สลับกันไปมา ตามการกดสวิตช์ PQ3 ในแต่ละครั้ง

การใช้งาน AVR-JTAG

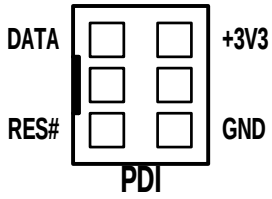
AVR-JTAG จะเป็น Connector แบบ IDE 10 Pin สำหรับ สำหรับเชื่อมต่อกับเครื่อง โปรแกรม และ Debug ภายนอกที่ใช้มาตรฐาน AVR-JTAG ซึ่งรองรับการใช้งานร่วมกับ MCU เบอร์ ATxMEGA128A1 เช่น AVR DRAGON หรือ AVR JTAGICE mkII หรือ เทียบเท่า โดยมีการจัดวงจรและสัญญาณตามแบบ AVR-JTAG ตามมาตรฐานของ ATMEL ไว้ดังนี้

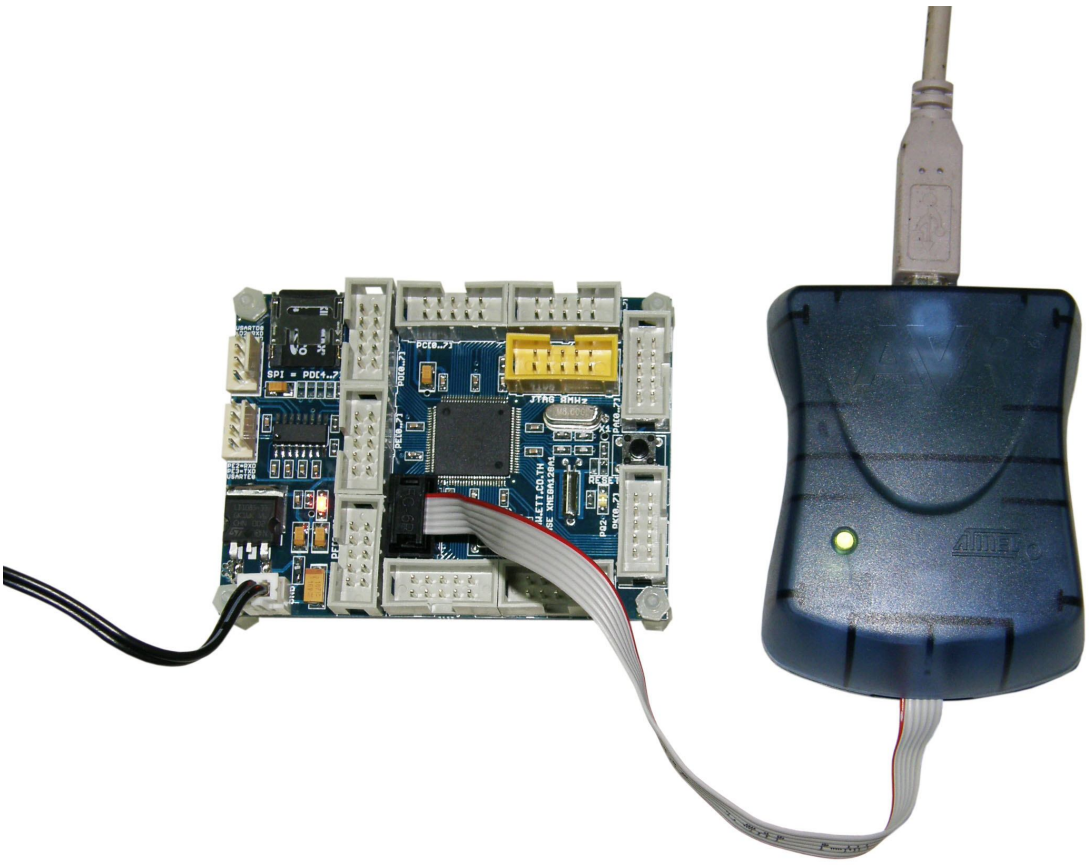




การใช้งาน PDI Program

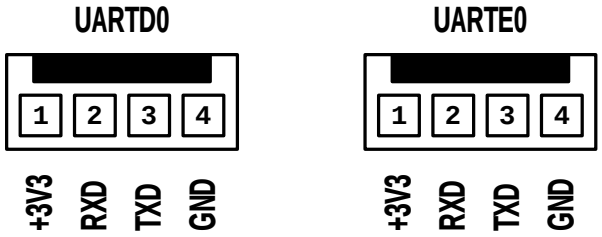
PDI จะเป็น Connector แบบ IDE 6 Pin สำหรับ สำหรับเชื่อมต่อกับเครื่อง โปรแกรมภายนอกที่ใช้มาตรฐาน AVR PDI ซึ่งรองรับการใช้งานร่วมกับ MCU เบอร์ ATxMEGA128A1 เช่น AVRISP mkII หรือเทียบเท่า เช่น ET-AVRISP mkII โดยมีการจัดวงจรและสัญญาณตามแบบ AVR PDI ตามมาตรฐานของ ATMEL ไว้ดังนี้



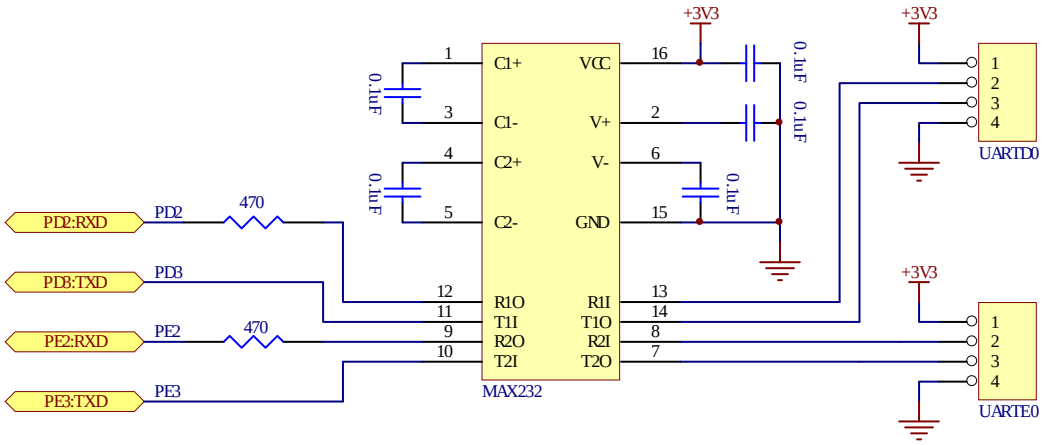


พอร์ต RS232

เป็นสัญญาณ RS232 ซึ่งผ่านวงจรแปลงระดับสัญญาณ MAX3232 เรียบร้อยแล้ว โดยมีจำนวน 2 ช่อง ด้วยกันคือ UARTD0 และ UARTE0 โดยทั้ง 2 ช่องสามารถใช้เชื่อมต่อกับสัญญาณRS232 เพื่อรับส่งข้อมูลได้



- UARTD0 ใช้ขาสัญญาณจาก PD2(RXD) และ PD3(TXD)
- UARTE0 ใช้ขาสัญญาณจาก PE2(RXD) และ PE3(TXD)



เนื่องจากระบบ Hardware USART ของ ATxMEGA128A1 นั้นจะมี USART ไว้ใช้งานมากถึง 8 ชุด คือ PC2,PC3 และ PC6,PC7 และ PD2,PD3 และ PD6,PD7 และ PE2,PE3 และ PE6,PE7 และ PF2,PF3 และ PF6,PF7 โดยในกรณีของบอร์ด ET-BASE xMEGA128A1 จะจัดวงจร USART ร่วมกับวงจร Line Driver ของ RS232 ไว้ให้จำนวน 2 ช่อง คือ PD2,PD3 และ PE2,PE3 ดังวงจร

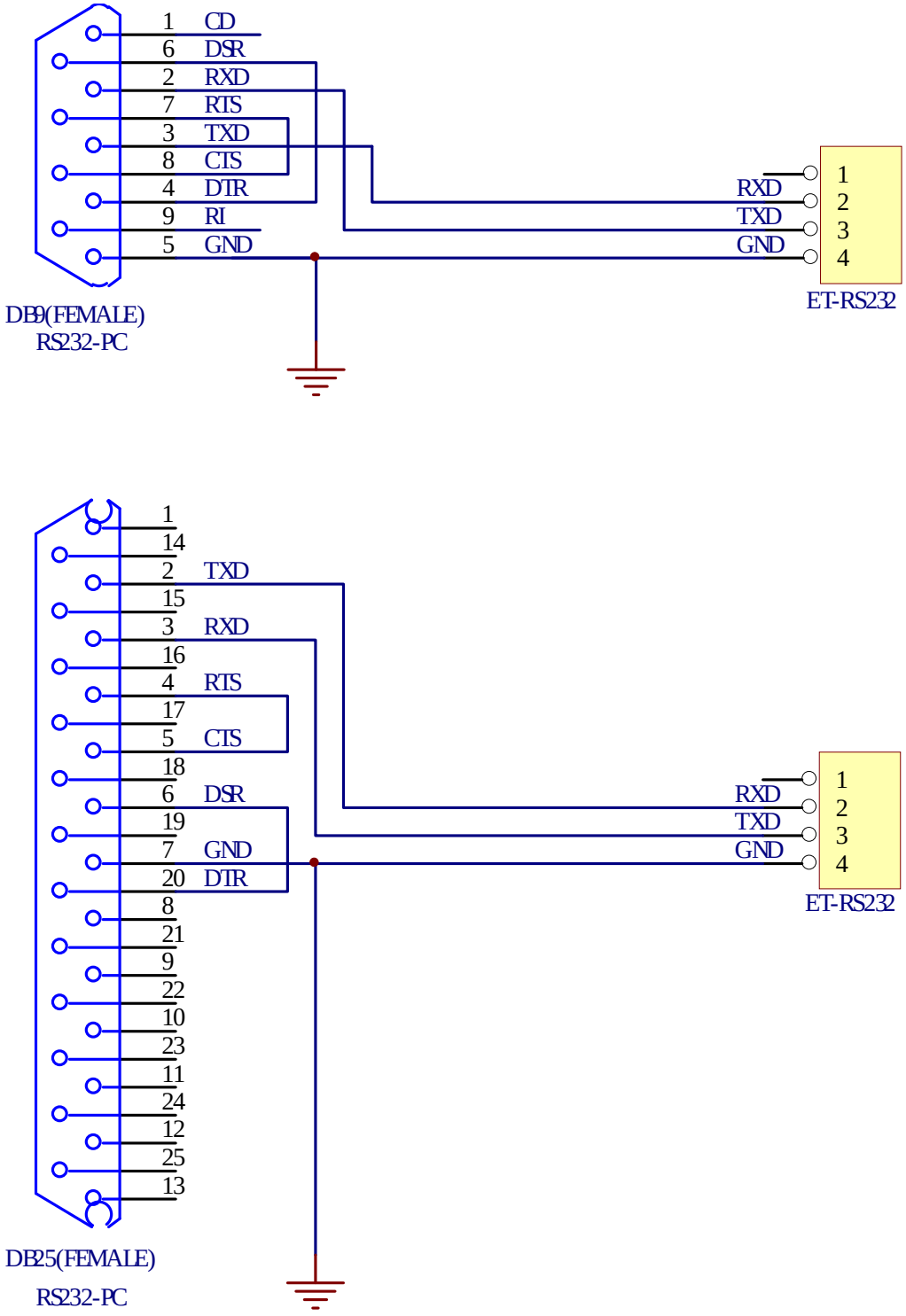
```
PORTD.DIRCLR = (1<<2);                  //RXD0(PD2) = Input
PORTD.DIRSET = (1<<3);                  //TXD0(PD3) = Output
USART_Rx_Enable(&USARTD0);              //Enable RX(UARTD0)
USART_Tx_Enable(&USARTD0);              //Enable TX(UARTD0)

PORTE.DIRCLR = (1<<2);                  //RXE0(PE2) = Input
PORTE.DIRSET = (1<<3);                  //TXE0(PE3) = Output
USART_Rx_Enable(&USARTE0);              //Enable RX(UARTE0)
USART_Tx_Enable(&USARTE0);              //Enable TX(UARTE0)
```

ตัวอย่าง การกำหนดค่า Pin สำหรับใช้งาน UARTD0 และ UARTE0

```
#include "avr_compiler.h"
#include "usart_driver.h"
#include <stdio.h>

/* pototype section */
int   my_putchar(char c, FILE *stream);
int   my_getchar(FILE *stream);

/* Retarget STDIO(putchar,getchar of printf) to My Function */
FILE uart_str = FDEV_SETUP_STREAM(my_putchar, my_getchar, _FDEV_SETUP_RW);

int main(void)
{
  stdout = stdin = &uart_str;                   //Retarget UART Function
  PORTE.DIRCLR = (1<<2);                        //RXE0(PE2) = Input
  PORTE.DIRSET = (1<<3);                        //TXE0(PE3) = Output

  /* USART, 8 Data bits, No Parity, 1 Stop bit. */
  USART_Format_Set(&USARE0,                     //USARTE0
                   USART_CHSIZE_8BIT_gc,        //8Bit Data
                   USART_PMODE_DISABLED_gc,     //Non Parity
                   false);                      //1 Stop Bit

  /* Set Baudrate to 9600 bps / 2MHz
   * Baudrate select = (1/(16*(((I/O clock frequency)/Baudrate)-1)
   * BSEL = ((I/O clock frequency)/(2^(ScaleFactor)*16*Baudrate))-1
   *      = ((2MHz)/(2^(0)*16*Baudrate))-1
   *      = (13.02)-1
   *      = 12
   */
  USART_Baudrate_Set(&USARTE0, 12 , 0);         //BSEL=12,Scale Factor=0
  USART_Rx_Enable(&USARTE0);                    //Enable RX(UARTE0)
  USART_Tx_Enable(&USARTE0);                    //Enable TX(UARTE0)

  printf("Hello World\n\r");
  while(1)
  {
    putchar(getchar());                         //Get & Echo Character
  }
}

/************************************/
/* Write 1 Character To UART(stdio) */
/************************************/
int my_putchar(char c, FILE *stream)
{
  while(!USART_IsTXDataRegisterEmpty(&USARTE0));
  USART_PutChar(&USARTE0, c);
  return 0;
}

/************************************/
/* Get 1 Character From UART(stdio) */
/************************************/
int my_getchar(FILE *stream)
{
  while(!USART_IsRXComplete(&USARTE0));
  return(USART_GetChar(&USARTE0));
}
```

ตัวอย่างการใช้งาน UARTE0 รับส่งข้อมูล

สำหรับ Cable ที่จะใช้ในการเชื่อมต่อ RS232 ระหว่าง Comport ของเครื่องคอมพิวเตอร์ PC เข้ากับขั้วต่อ UARTD0 และ UARTE0 ของบอร์ด ET-BASE xMEGA128A1 นั้น เป็นดังนี้

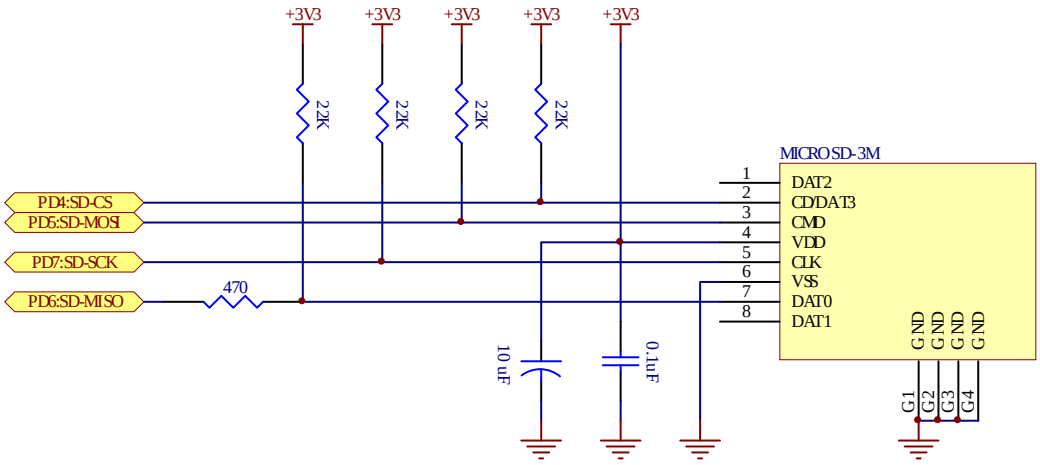


รูป แสดงวงจรสาย **Cable** สำหรับ **RS232**

การ์ดหน่วยความจำ SD Card แบบ Micro-SD

บอร์ด ET-BASE xMEGA128A1 รองรับการเชื่อมต่อกับการ์ดหน่วยความจำ SD Card แบบ Micro-SD โดยใช้การเชื่อมต่อแบบ SPI โดยใช้ขาสัญญาณ PD[4..7] ในการเชื่อมต่อกับการ์ด ซึ่งในการติดต่อสั่งงาน การ์ดนั้น สามารถโปรแกรม Pin I/O ของ PD[4..7] ให้ทำงานในโหมด SPI โดยต้องกำหนดหน้าที่ของขาสัญญาณ PD[4..7] ของ MCU เป็นดังนี้

- CD/DAT3 ใช้ PD4 ในหน้าที่ของ GPIO Output
- CMD ใช้ PD5 ในหน้าที่ MOSI ของ SPI
- DAT0 ใช้ PD6 ในหน้าที่ MISO ของ SPI
- CLK ใช้ PD7 ในหน้าที่ SCK ของ SPI

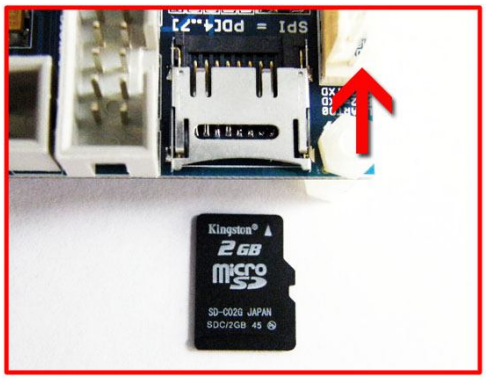



```
#include "spi_driver.h"

SPI_Master_t spiMasterD;  //ET-BASE xMEGA128A1 SPI=PD
PORT_t *ssPort = &PORTD;  //Instantiate pointer to ssPort

PORTD.DIRSET = PIN4_bm;                          //PD4 = CS#
PORTD.PIN4CTRL = PORT_OPC_WIREDANDPULL_gc;       //PD4 = wired + Pullup Out
PORTD.OUTSET = PIN4_bm;                          //SS# = High(Disable Card)
SPI_MasterSSHigh(ssPort, PIN4_bm);

/* Initialize SPI master on port-D */
// SPI = Low Speed(Max. 400 kBit used in Card Initialization)
SPI_MasterInit(&spiMasterD,                      //SPI Master of Port-PD
               &SPID,                            //Used SPI Port-PD
               &PORTD,                           //Port PD
               false,                            //MSB First
               SPI_MODE_0_gc,                    //Clock Low,Rising Edge
               SPI_INTLVL_OFF_gc,                //Disable Interrupt
               false,                            //Disable Clock2X
               SPI_PRESCALER_DIV128_gc);         //System Clock(32MHz)/128
```

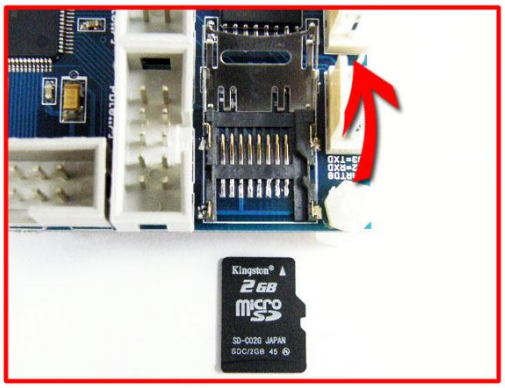
ตัวอย่าง การกำหนดค่า Pin สำหรับใช้งาน SD Card

การใส่การ์ดหน่วยความจำใน Socket

สำหรับ Socket สำหรับติดตั้งการ์ดหน่วยความจำของบอร์ด ET-BASE xMEGA128A1 นั้น จะใช้ Socket หน่วยความจำแบบใส่การ์ดจากด้านบน แบบเดียวกันกับซิมการ์ดของโทรศัพท์มือถือ โดยเมื่อต้องการจะใส่หรือถอดการ์ดหน่วยความจำจะต้องทำการเปิดฝาครอบ Socket ออกเสียก่อน จากนั้นจึงจะสามารถใส่หรือถอดการ์ดหน่วยความจำได้ โดยการเปิด หรือ ปิด ฝาครอบ Socket จะใช้การกดเลื่อนฝาครอบเข้าหรือออก ซึ่งถ้าเลื่อนฝาครอบเข้าด้านในจะเป็นการเลื่อนเพื่อเปิดฝา แต่ถ้าเลื่อนออกด้านนอกจะเป็นการเลื่อนเพื่อล๊อคฝาครอบดังรูป

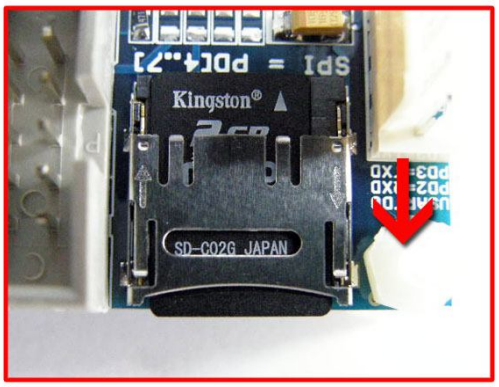


1. กดฝาครอบและเลื่อนเข้าเพื่อเปิดล๊อค



2. เปิดฝาครอบ Socket ออก

3. ใสการ์ดหน่วยความจำใน Socket

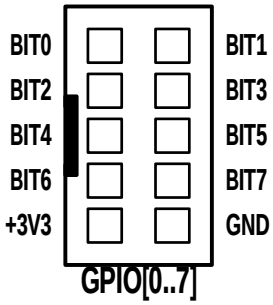


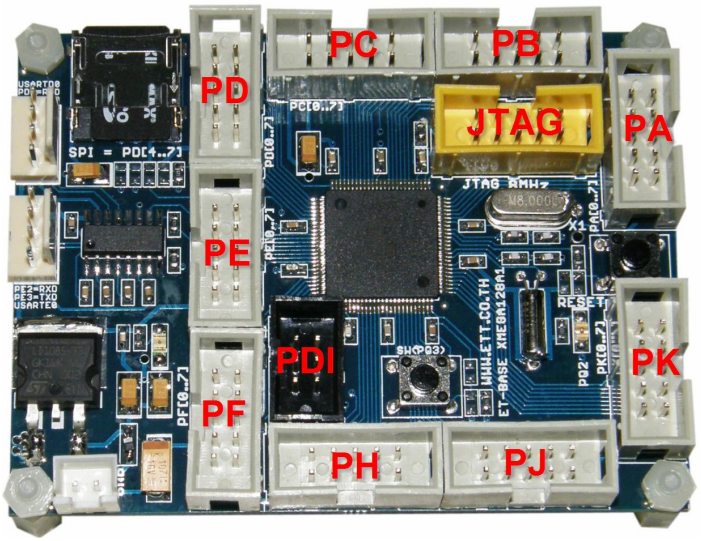
4. ปิดฝาครอบและกดเลื่อนออกเพื่อล๊อค

แสดงลำดับขั้นตอนการใส่การ์ดหน่วยความจำ

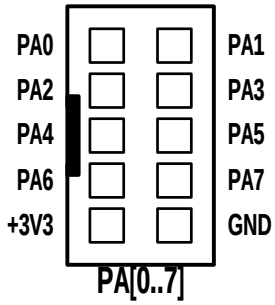
ขั้วต่อ Port I/O ต่าง ๆของบอร์ด

สำหรับขั้วต่อ Port I/O ของ MCU ของบอร์ด ET-BASE xMEGA128A1 นั้น จะจัดเรียงออกมารอไว้ยังขั้วต่อแบบ IDE 10 Pin จำนวน 9 ชุดๆละ 8บิต สำหรับให้ผู้ใช้เลือกต่อออกไปใช้งานตามต้องการ โดยรูปแบบการจัดเรียงสัญญาณของแต่ละพอร์ต จะเรียงลำดับตำแหน่งบิต I/O เหมือนกัน โดยมีรายละเอียดของสัญญาณแต่ละชุดดังนี้





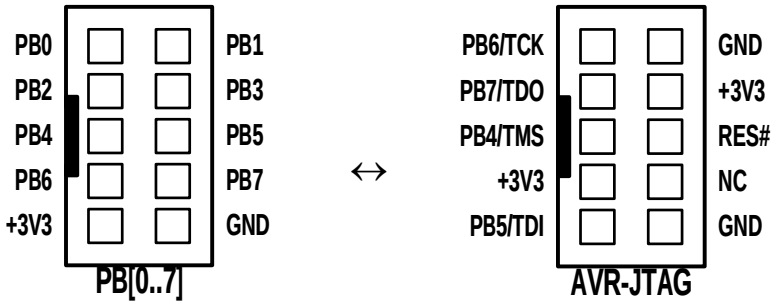
- ขั้วต่อ IDE 10 Pin ของ PA[0..7] โดย Port PA ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 เองนั้นสัญญาณทั้ง 8 เส้น จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	ADCA POS	ADCA NEG	ADCA GAIN POS	ADCA GAIN NEG	ACA POS	ACA NEG	ACA OUT	DACA	REFA
PA0	SYNC	ADC0	ADC0	ADC0		AC0	AC0			AREF
PA1	SYNC	ADC1	ADC1	ADC1		AC1	AC1			
PA2	SYNC/ASYNC	ADC2	ADC2	ADC2		AC2			DAC0	
PA3	SYNC	ADC3	ADC3	ADC3		AC3	AC3		DAC1	
PA4	SYNC	ADC4		ADC4	ADC4	AC4				
PA5	SYNC	ADC5		ADC5	ADC5	AC5	AC5			
PA6	SYNC	ADC6		ADC6	ADC6	AC6				
PA7	SYNC	ADC7		ADC7	ADC7		AC7	AC0OUT		

Port A Alternate Function

- ขั้วต่อ IDE 10 Pin ของ PB[0..7] โดย Port PB ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 นั้น จะมีการเชื่อมต่อ PB[4...7] ไปยังขั้วต่อ AVRJTAG ด้วย ส่วน PB[0...3] จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ ซึ่งถ้าต้องการใช้งาน PB[4..7] ต้องไปสั่ง Disable Fuse Bit ของ JTAGEN ออกเสียก่อนด้วยเครื่องโปรแกรมทาง PDI Port โดย PB[0..7] มีการจัดเรียงสัญญาณไว้ดังนี้

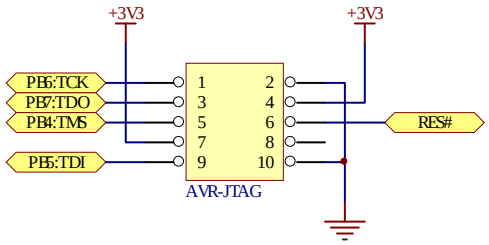


Pin	Interrupt	ADCB POS	ADCB NEG	ADCB GAIN POS	ADCB GAIN NEG	ACB POS	ACB NEG	ACB OUT	DACB	REFB
PB0	SYNC	ADC0	ADC0	ADC0		AC0	AC0			AREF
PB1	SYNC	ADC1	ADC1	ADC1		AC1	AC1			
PB2	SYNC/ASYNC	ADC2	ADC2	ADC2		AC2			DAC0	
PB3	SYNC	ADC3	ADC3	ADC3		AC3	AC3		DAC1	
PB4	SYNC	ADC4		ADC4	ADC4	AC4				
PB5	SYNC	ADC5		ADC5	ADC5	AC5	AC5			
PB6	SYNC	ADC6		ADC6	ADC6	AC6				
PB7	SYNC	ADC7		ADC7	ADC7		AC7	AC0OUT		

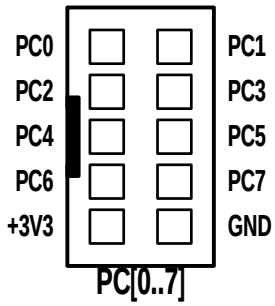
Port B Alternate Function

*****หมายเหตุ***** PB4...PB7 จะซ้อนทับกับ JTAG Function ด้วย ซึ่งขาสัญญาณ PB4...PB7 ของบอร์ดจะถูกเชื่อมต่อไปยังขั้วต่อ AVRJTAG ด้วย ซึ่งถ้าผู้ใช้ได้เชื่อมต่อ AVRJTAG ไว้ด้วย จะไม่สามารถใช้งานขาสัญญาณ PB4...PB7 ได้ โดยขาสัญญาณ PB4...PB7 เมื่อทำหน้าที่เป็น JTAG จะมีหน้าที่ดังนี้

- PB4 = TMS JTAG
- PB5 = TDI JTAG
- PB6 = TCK JTAG
- PB7 = TDO JTAG



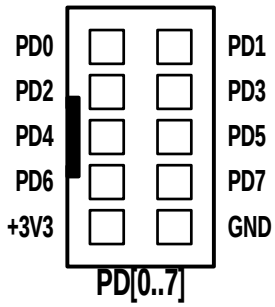
- ขั้วต่อ IDE 10 Pin ของ PC[0..7] โดย Port PC ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 เองนั้นสัญญาณทั้ง 8 เส้น จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	TCC0	AWEXC	TCC1	USARTC0	USARTC1	SPIC	TWIC	CLOCKOUT	EVENOUT
PC0	SYNC	OC0A	OC0ALS					SDA		
PC1	SYNC	OC0B	OC0AHS		XCK0			SCL		
PC2	SYNC/ASYNC	OC0C	OC0BLS		RXD0					
PC3	SYNC	OC0D	OC0BHS		TXD0					
PC4	SYNC		OC0CLS	OC1A			SS#			
PC5	SYNC		OC0CHS	OC1B		XCK1	MOSI			
PC6	SYNC		OC0DLS			RXD1	MISO			
PC7	SYNC		OC0DHS			TXD1	SCK		CKOUT	EVOUT

Port C Alternate Function

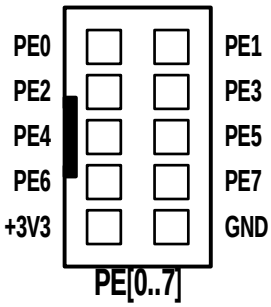
- ขั้วต่อ IDE 10 Pin ของ PD[0..7] โดย Port PD ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 นั้น จะมีการเชื่อมต่อ PD2...PD3 ไปยังวงจร USARTD0 และเชื่อมต่อสัญญาณ PD4...PD7 ไปยังวงจร SD Card ด้วย ส่วนสัญญาณ PD0...PD1 จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	TCD0	TCD1	USARTD0	USARTD1	SPID	TWID	CLOCKOUT	EVENOUT
PD0	SYNC	OC0A					SDA		
PD1	SYNC	OC0B		XCK0			SCL		
PD2	SYNC/ASYNC	OC0C		RXD0					
PD3	SYNC	OC0D		TXD0					
PD4	SYNC		OC1A			SS#			
PD5	SYNC		OC1B		XCK1	MOSI			
PD6	SYNC				RXD1	MISO			
PD7	SYNC				TXD1	SCK		CKOUT	EVOUT

Port D Alternate Function

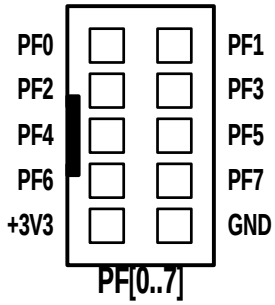
- ขั้วต่อ IDE 10 Pin ของ PE[0..7] โดย Port PB ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 นั้น จะมีการเชื่อมต่อ PE2...PE3 ไปยังวงจร USARTE0 ด้วย ส่วน PE0...PE1 และ PE4...PE7 จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	TCE0	AWEXE	TCE1	USARTE0	USARTE1	SPIE	TWIE	CLOCKOUT	EVENOUT
PE0	SYNC	OC0A	OC0ALS					SDA		
PE1	SYNC	OC0B	OC0AHS		XCK0			SCA		
PE2	SYNC/ASYNC	OC0C	OC0BLS		RXD0					
PE3	SYNC	OC0D	OC0BHS		TXD0					
PE4	SYNC		OC0CLS	OC1A			SS#			
PE5	SYNC		OC0CHS	OC1B		XCK1	MOSI			
PE6	SYNC		OC0DLS			RXD1	MISO			
PE7	SYNC		OC0DHS			TXD1	SCK		CKOUT	EVOUT

Port E Alternate Function

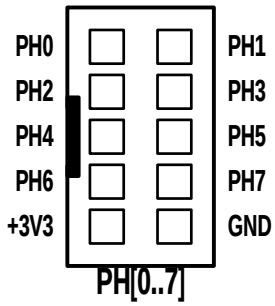
- ขั้วต่อ IDE 10 Pin ของ PF[0..7] โดย Port PF ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 เองนั้นสัญญาณทั้ง 8 เส้น จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	TCF0	TCF1	USARTF0	USARTF1	SPIF	TWIF
PF0	SYNC	OC0A					SDA
PF1	SYNC	OC0B		XCK0			SCL
PF2	SYNC/ASYNC	OC0C		RXD0			
PF3	SYNC	OC0D		TXD0			
PF4	SYNC		OC1A			SS#	
PF5	SYNC		OC1B		XCK1	MOSI	
PF6	SYNC				RXD1	MISO	
PF7	SYNC				TXD1	SCK	

Port F Alternate Function

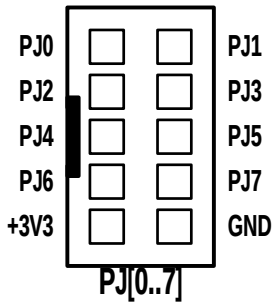
- ขั้วต่อ IDE 10 Pin ของ PH[0..7] โดย Port PH ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 เองนั้นสัญญาณทั้ง 8 เส้น จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	SDRAM 3P	SRAM ALE1 3P	SRAM ALE12 3P	LPC ALE1 3P	LPC ALE1 2P	LPC ALE12 2P
PH0	SYNC	WE#	WE#	WE#	WE#	WE#	WE#
PH1	SYNC	CAS#	RE#	RE#	RE#	RE#	RE#
PH2	SYNC/ASYNC	RAS#	ALE1	ALE1	ALE1	ALE1	ALE1
PH3	SYNC	DQM#	-	ALE2	-	-	ALE2
PH4	SYNC	BA0	CS0#/A16	CS0#	CS0#/A16	CS0#	CS0#/A16
PH5	SYNC	BA1	CS1#/A17	CS1#	CS1#/A17	CS1#	CS1#/A17
PH6	SYNC	CKE	CS2#/A18	CS2#	CS2#/A18	CS2#	CS2#/A18
PH7	SYNC	CLK	CS3#/A19	CS3#	CS3#/A19	CS3#	CS3#/A19

Port H Alternate Function

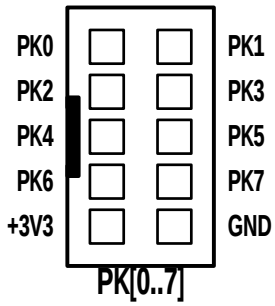
- ขั้วต่อ IDE 10 Pin ของ PJ[0..7] โดย Port PJ ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 เองนั้นสัญญาณทั้ง 8 เส้น จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



Pin	Interrupt	SDRAM 3P	SRAM ALE1 3P	SRAM ALE12 3P	LPC ALE1 3P	LPC ALE1 2P	LPC ALE12 2P
PJ0	SYNC	D0	D0	D0	D0/A0	D0/A0	D0/A0/A8
PJ1	SYNC	D1	D1	D1	D1/A1	D1/A1	D1/A1A9
PJ2	SYNC/ASYNC	D2	D2	D2	D2/A2	D2/A2	D2/A2/A10
PJ3	SYNC	D3	D3	D3	D3/A3	D3/A3	D3/A3/A11
PJ4	SYNC	A8	D4	D4	D4/A4	D4/A4	D4/A4/A12
PJ5	SYNC	A9	D5	D5	D5/A5	D5/A5	D5/A5/A13
PJ6	SYNC	A10	D6	D6	D6/A6	D6/A6	D6/A6/A14
PJ7	SYNC	A11	D7	D7	D7/A7	D7/A7	D7/A7/A15

Port J Alternate Function

- ขั้วต่อ IDE 10 Pin ของ PK[0..7] โดย Port PK ของ ATxMEGA128A1 จะมีทั้งหมดจำนวน 8 บิต ซึ่งในบอร์ด ET-BASE xMEGA128A1 เองนั้นสัญญาณทั้ง 8 เส้น จะยังคงอิสระปล่อยว่างไว้ให้ผู้ใช้นำไปต่อประยุกต์ใช้งานได้เองตามความต้องการ โดยมีการจัดเรียงสัญญาณไว้ดังนี้



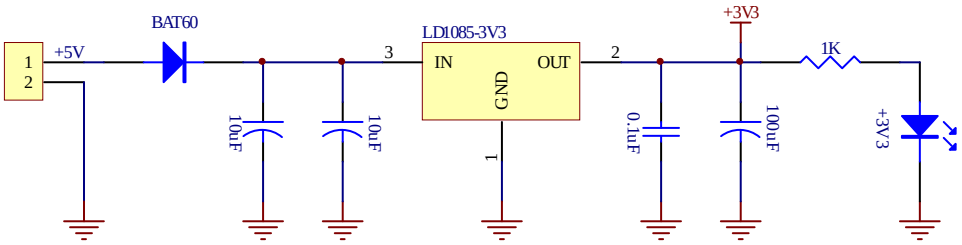
Pin	Interrupt	SDRAM 3P	SRAM ALE1 3P	SRAM ALE12 3P	LPC ALE1 3P	LPC ALE1 2P	LPC ALE12 2P
PK0	SYNC	A0	A0/A8	A0/A8/A16	A8	-	-
PK1	SYNC	A1	A1/A9	A1/A9/A17	A9	-	-
PK2	SYNC/ASYNC	A2	A2/A10	A2/A10/A18	A10	-	-
PK3	SYNC	A3	A3/A11	A3/A11/A19	A11	-	-
PK4	SYNC	A4	A4/A12	A4/A12/A20	A12	-	-
PK5	SYNC	A5	A5/A13	A5/A13/A21	A13	-	-
PK6	SYNC	A6	A6/A14	A6/A14/A22	A14	-	-
PK7	SYNC	A7	A7/A15	A7/A15/A23	A15	-	-

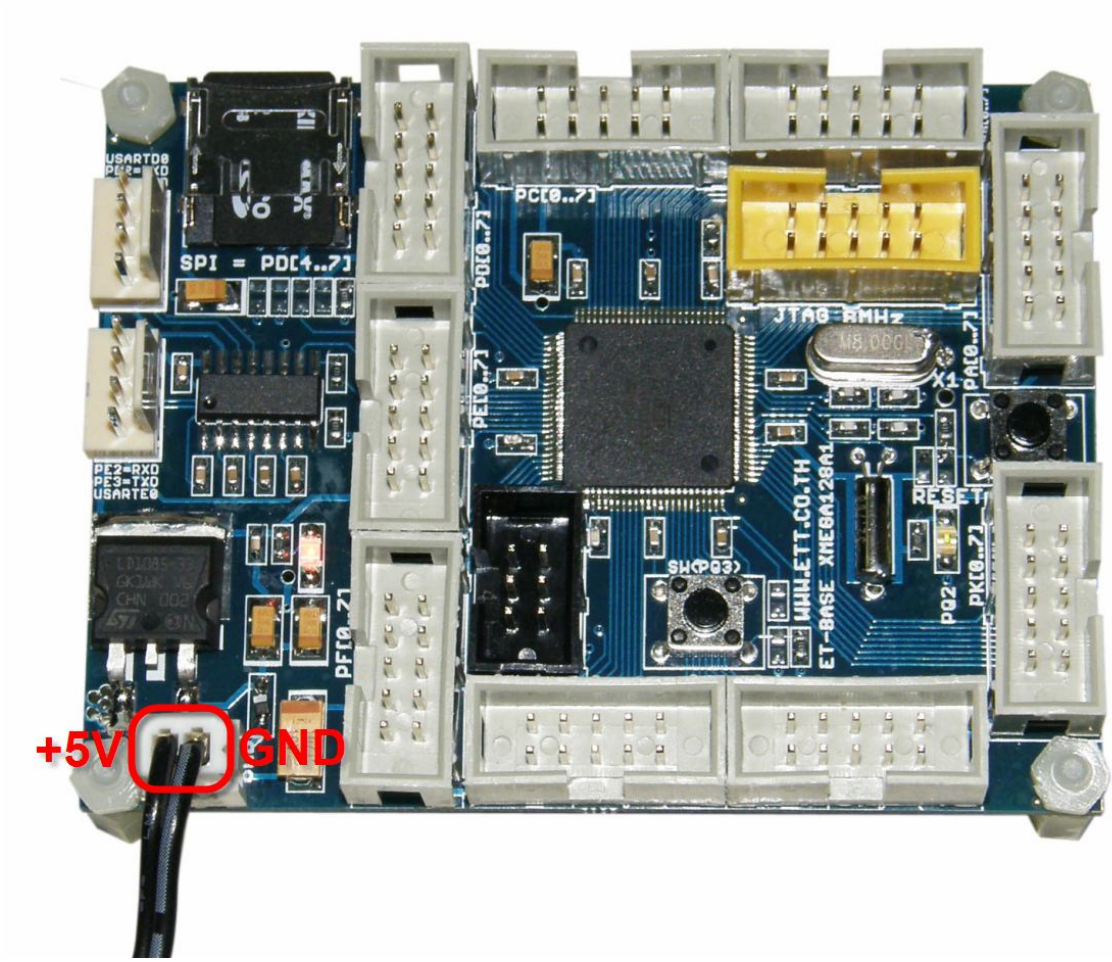
Port K Alternate Function

วงจรแหล่งจ่ายไฟ

วงจรแหล่งจ่ายไฟของบอร์ดใช้งานได้กับไฟ DC ขนาด +5V โดยใช้ขั้วต่อแบบ 2 Pin Block ป้องกันการเสียบสายกลับขั้ว พร้อมวงจร Regulate ขนาด +3V3/3A

โดยวงจรภาคแหล่งจ่ายไฟในส่วนที่เป็นวงจร Regulate ขนาด 3.3V นั้นจะจ่ายให้กับ CPU และวงจร I/O ของบอร์ดทั้งหมด

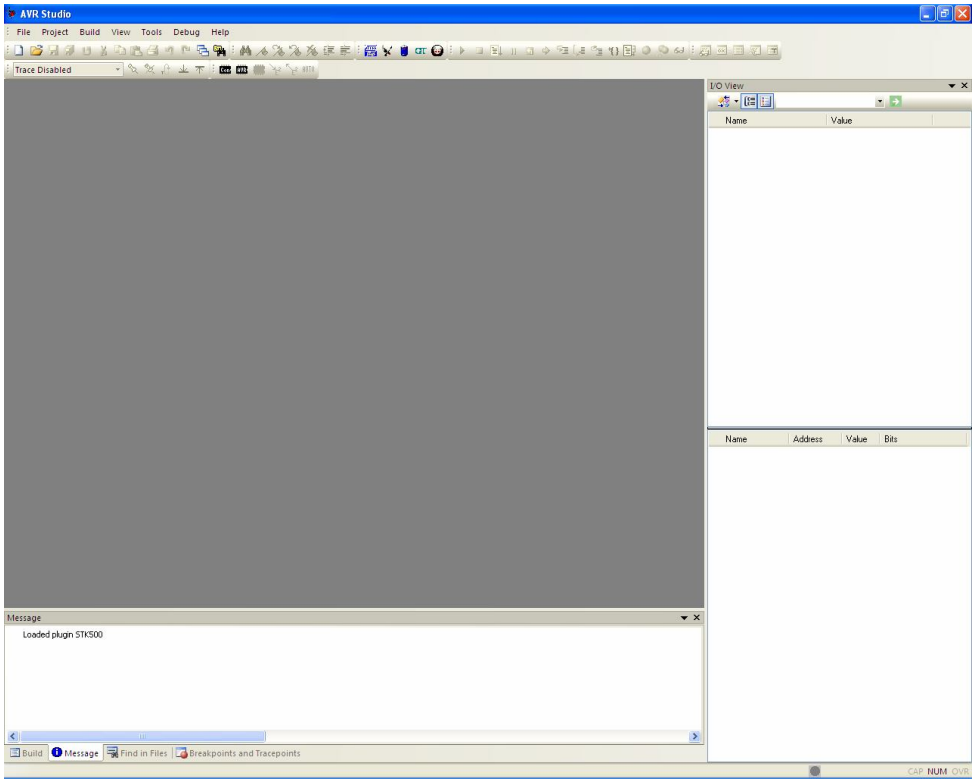




ตัวอย่างการพัฒนาโปรแกรมด้วย WinAVR ร่วมกับ AVR Studio4

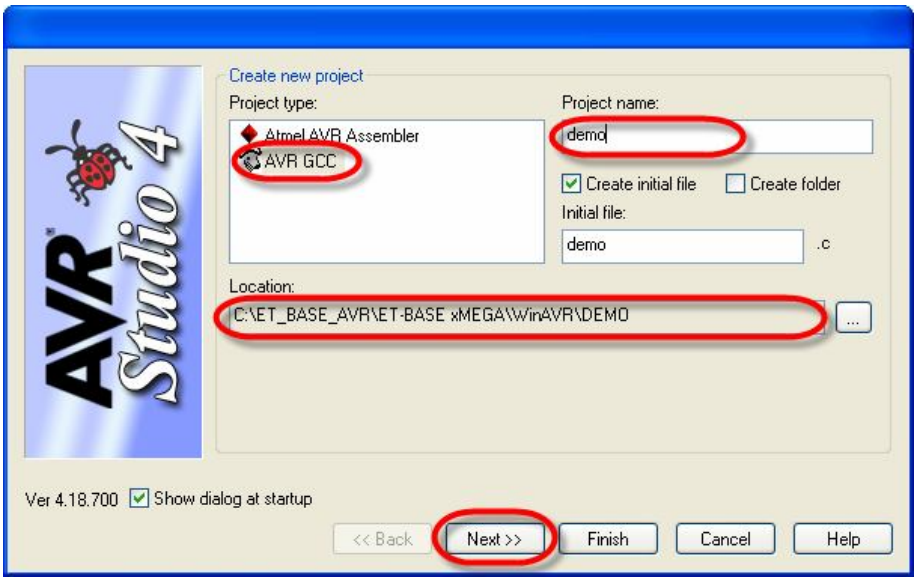
ในการพัฒนาโปรแกรมของ AVR เบอร์ ATxMEGA128A1 นั้น ตามปรกติแล้วจะสามารถเลือกใช้ Compiler ต่างๆที่สนับสนุน MCU เบอร์นี้อยู่ได้ทั้งหมด แต่ในที่นี้จะขอแนะนำให้ใช้โปรแกรม AVR Studio4 ร่วมกับ WinAVR ซึ่งเป็นชุดโปรแกรมที่แจกจ่ายให้ใช้ได้ฟรีๆไม่เสียค่าใช้จ่าย มีการพัฒนาปรับปรุงความสามารถของโปรแกรมกันอย่างต่อเนื่องและมีผู้ใช้งานกันอย่างแพร่หลายทั่วโลก สามารถค้นหาตัวอย่างโปรแกรม และ Library ต่างๆที่ผู้ใช้ต่างๆจำนวนมากได้สร้างและเผยแพร่ไว้มาเป็นแนวทางในการศึกษาได้มากมายซึ่งปัจจุบัน (สิงหาคม 2553) ทาง ATMEL ได้ทำการปรับปรุงโปรแกรม AVR Studion4 เป็นรุ่น V4.18 แล้ว ส่วนโปรแกรม WinAVR สามารถ Download ได้จาก http://winavr.sourceforge.net/ จะปรับปรุงเป็น WINAVR-20100110 แล้ว ซึ่งผู้ใช้สามารถไป Download มาติดตั้งใช้งานได้ฟรี โดยในที่นี้จะขอแนะนำแนวทางการพัฒนาโปรแกรมแบบพอสังเขป เพื่อเป็นแนวทางสำหรับผู้เริ่มต้น ซึ่งรายละเอียดต่างๆสามารถศึกษาได้จากคู่มือของโปรแกรมได้ โดยแนวทางการพัฒนาโปรแกรมของ ATxMEGA128A1 โดยใช้โปรแกรม AVR Studio4 ร่วมกับ WinAVR มีลำดับขั้นตอนดังนี้

1. สั่ง Run Program AVR Studio4 ซึ่งจะได้ผลดังรูป

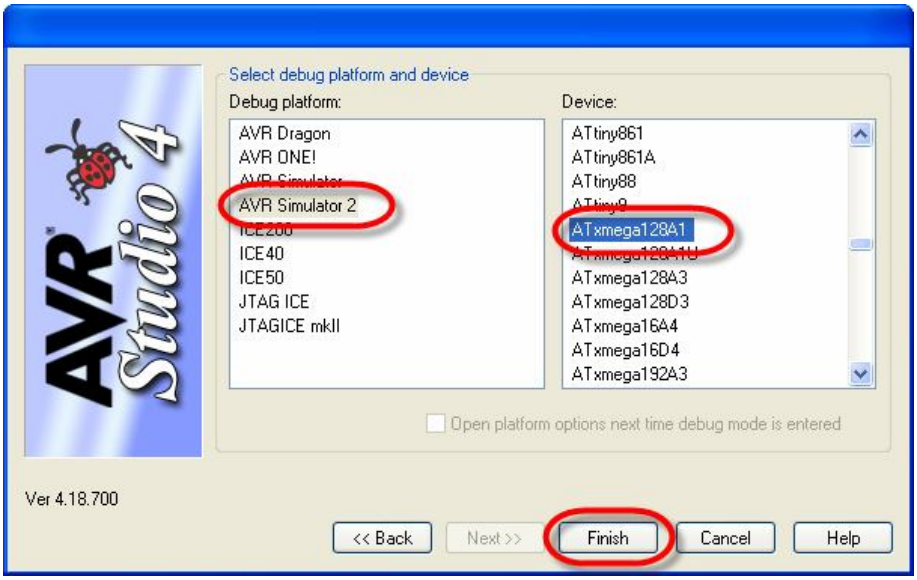


2. สั่งสร้าง project ใหม่ โดยเลือกที่ project → New project จากนั้นเลือกกำหนดตัวเลือกต่างๆ ให้กับโปรแกรมดังนี้

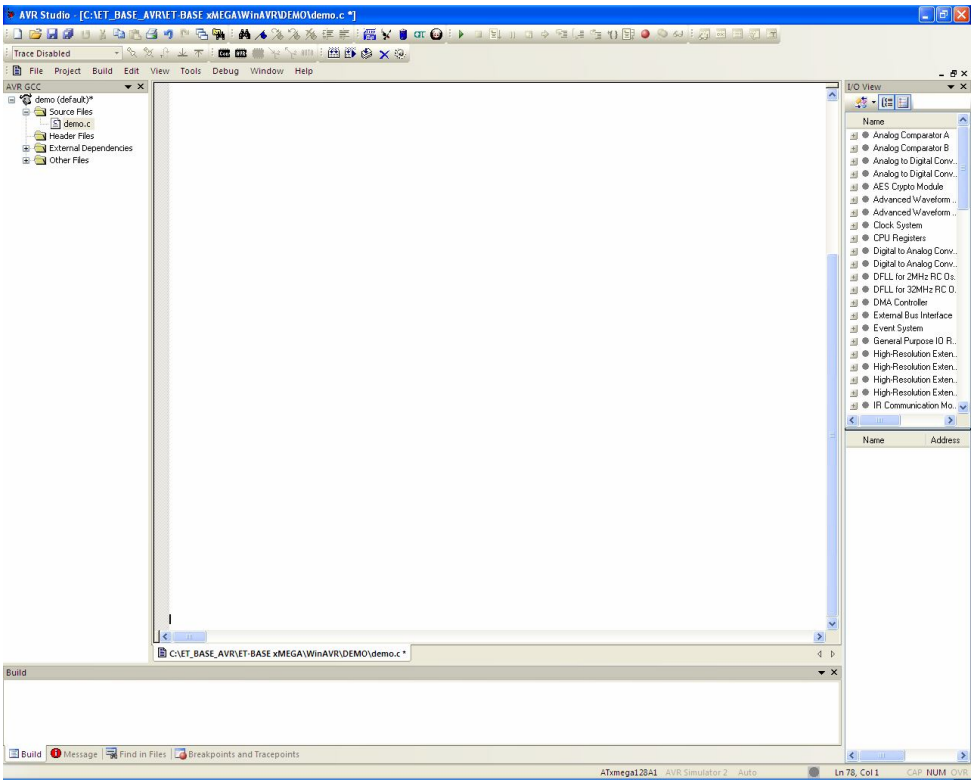
   - Project type เลือกกำหนดเป็น AVR GCC
   - Location สำหรับบันทึก project ให้ระบุตำแหน่ง Folder ที่ต้องการใช้บันทึกไฟล์ และ Codeต่างของ project
   - Project name ให้กำหนดชื่อ project ตามต้องการในตัวอย่างกำหนดเป็น “demo” และให้เลือก Create initial file ไว้ด้วย ซึ่งเมื่อเราทำการกำหนดชื่อ project name เสร็จแล้ว โปรแกรมจะสร้างไฟล์ ที่มีชื่อเดียวกันกับ project name ให้เองโดยอัตโนมัติ ซึ่งถ้าต้องการกำหนดชื่อไฟล์เป็นชื่ออื่น ก็ไม่ต้องเลือก Create initial file



3. เมื่อกำหนดค่าตัวเลือกต่างๆ ให้กับโปรแกรมเรียบร้อยแล้ว ให้เลือกที่ Next แล้วกำหนดค่าใน Debug platform เป็น AVR Simulator2 และเลือก Device เป็น ATxmega128A1 ซึ่งเมื่อสร้าง project เสร็จโปรแกรมจะสร้างไฟล์ภาษาซีให้ โดยมีชื่อเดียวกับ project ไฟล์ ซึ่งในที่นี้จะเป็นไฟล์ชื่อ demo.c ให้เองโดยอัตโนมัติ เพียงแต่ไฟล์ดังกล่าวจะยังไม่มี code ใดๆบรรจุไว้ให้ เป็นเพียงหน้ากระดาษเปล่าๆ ซึ่งต้องรอให้เราเขียน code เพิ่มเข้าไปเอง ดังรูป



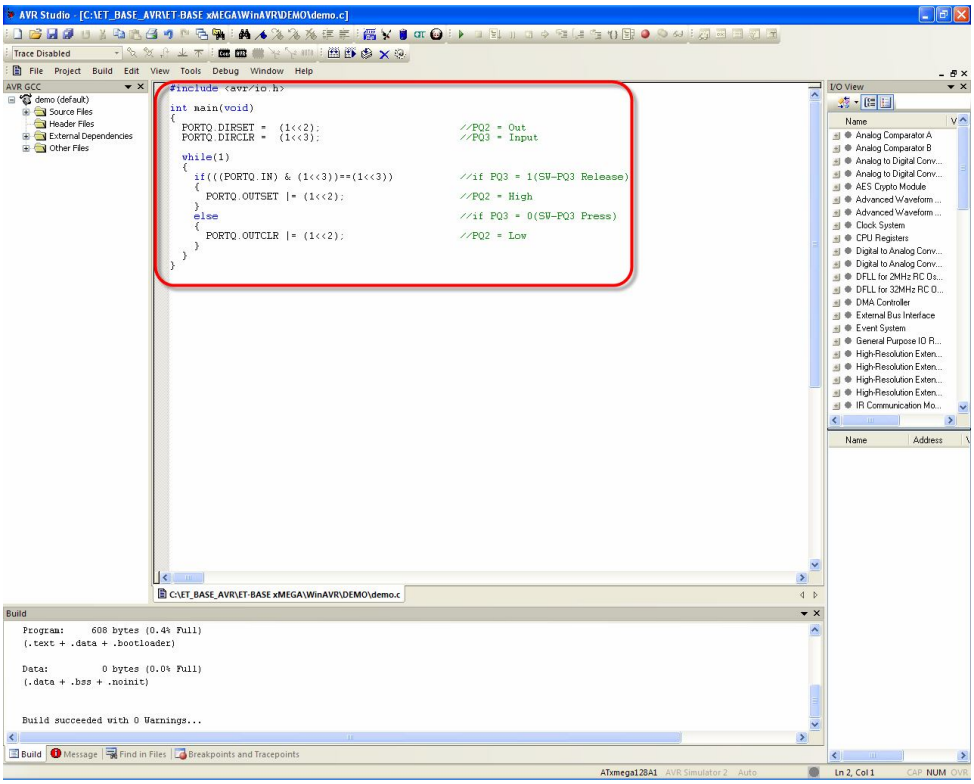
↓



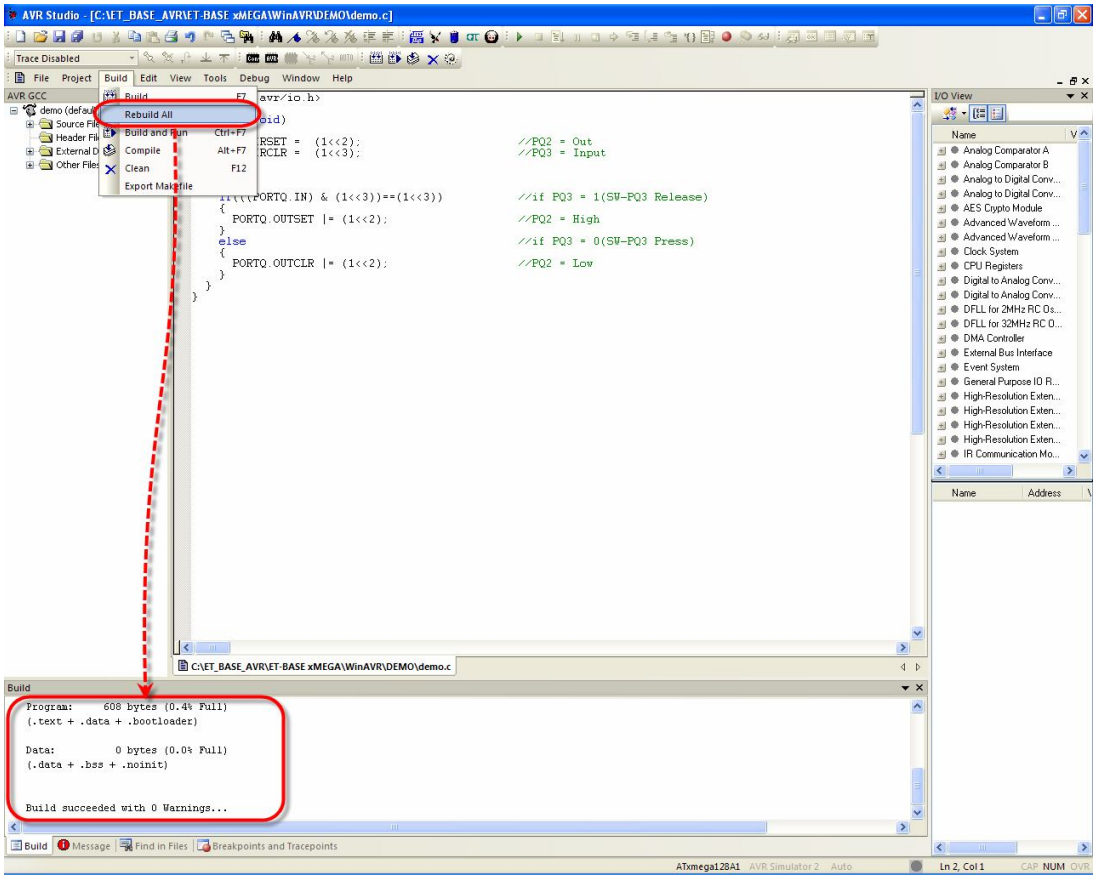
4. ให้พิมพ์คำสั่งของโปรแกรมสำหรับทดสอบการทำงาน ในหน้าต่าง Text Editor ของโปรแกรม โดยในที่นี้จะทดสอบด้วย Codeโปรแกรม สำหรับทำหน้าที่ทดสอบการทำงานของบอร์ดในเบื้องต้น โดยทำหน้าที่อ่านสถานการกดสวิตช์ที่ต่ออยู่กับขา PQ3 แล้วแสดงค่าการทำงานที่ LED ซึ่งต่อควบคุมจากขา PQ2 ดังตัวอย่าง

```
#include <avr/io.h>

int main(void)
{
  PORTQ.DIRSET =  (1<<2);                  //PQ2 = Out
  PORTQ.DIRCLR =  (1<<3);                  //PQ3 = Input

  while(1)
  {
    if(((PORTQ.IN) & (1<<3))==(1<<3))      //if PQ3 = 1
    {
      PORTQ.OUTSET |= (1<<2);              //PQ2 = High
    }
    else                                   //if PQ3 = 0
    {
      PORTQ.OUTCLR |= (1<<2);              //PQ2 = Low
    }
  }
}
```

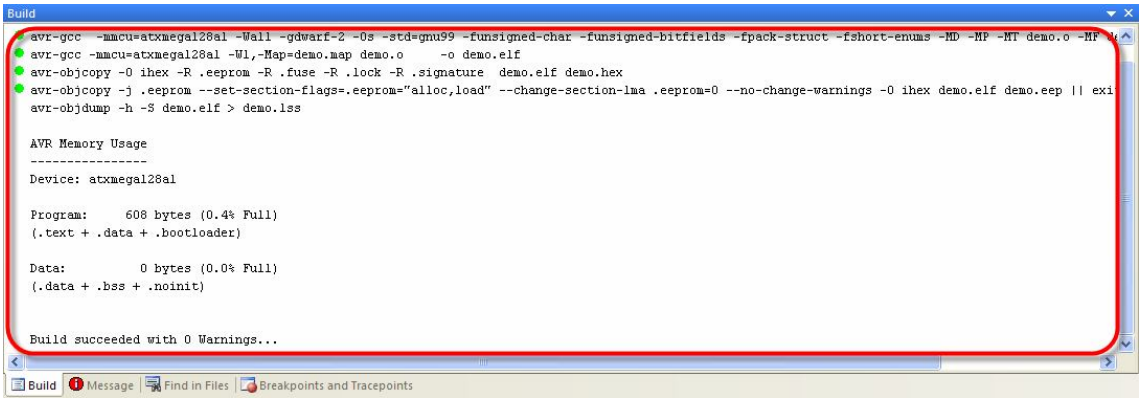
↓



5. หลังจากพิมพ์ Code โปรแกรมเสร็จแล้วให้สั่งแปลโปรแกรม โดยเลือกที่ build →rebuild all ซึ่งถ้าทุกอย่างถูกต้อง ผลการแปลคำสั่งจะได้ผลลัพธ์เป็น “Build succeeded with 0 Warnings... “ และจะรายงานผลการแปลพร้อมขนาดหน่วยความจำที่ใช้ไป และจะได้ Output เป็น HEX File ที่มีชื่อเดียวกันกับ project ที่สร้างไว้ โดยจะบรรจุอยู่ในDirectory ย่อยชื่อ default ดังรูป

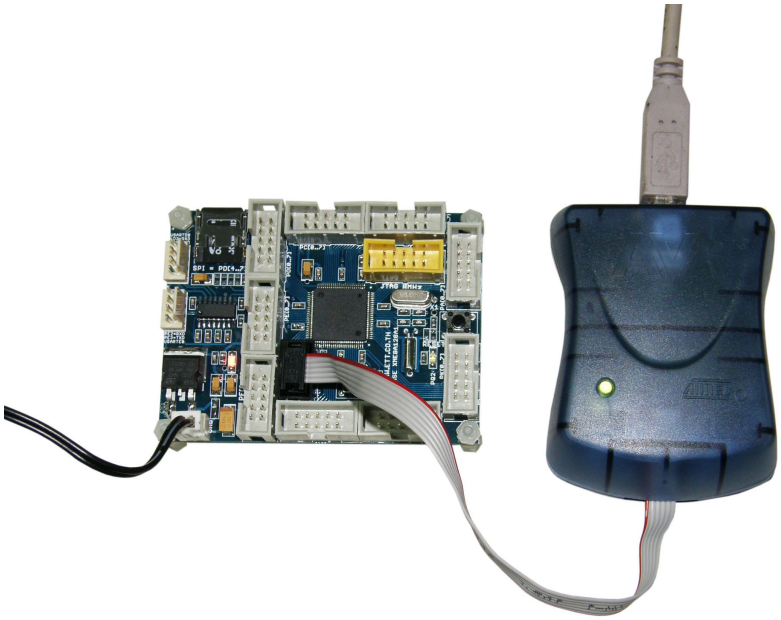


↓

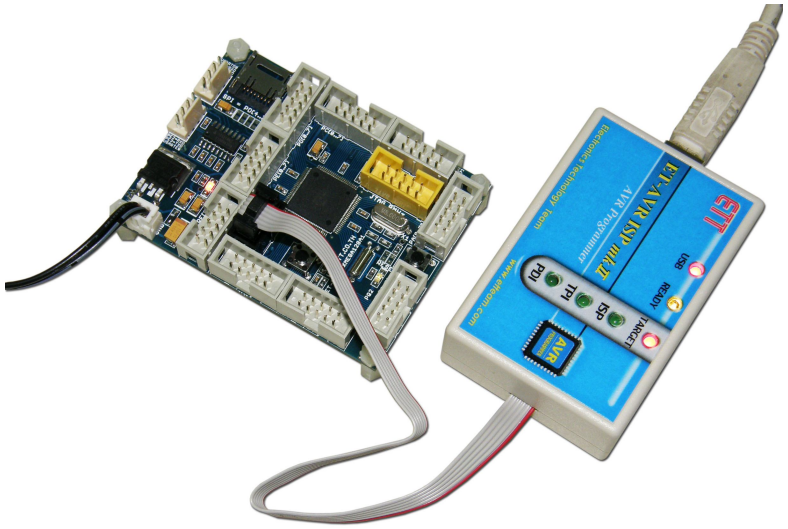


การโปรแกรม Hex File ให้บอร์ดโดยใช้ AVRISP mkII

ซึ่งเมื่อผ่านกระบวนการ Compile Code จนได้ Hex File มาแล้ว ขั้นตอนต่อไป จะเป็นขั้นตอนของการนำข้อมูล Code ใน Hex File เขียนเข้าไปในหน่วยความจำของ MCU ซึ่งขั้นตอนนี้สามารถทำได้หลายแนวทาง แต่ในที่นี้จะแนะนำให้ใช้เครื่องโปรแกรม AVRISP mkII หรือ อุปกรณ์อื่นที่มีความสามารถเทียบเท่ากัน เช่นเครื่องโปรแกรมรุ่น ET-AVRISP mkII ของ อีทีที โดยสามารถสั่งงานผ่านโปรแกรม AVR Studio4 ได้เลย ซึ่งในการเชื่อมต่อกับเครื่องโปรแกรมจะใช้ขั้วต่อPDI ดังตัวอย่าง



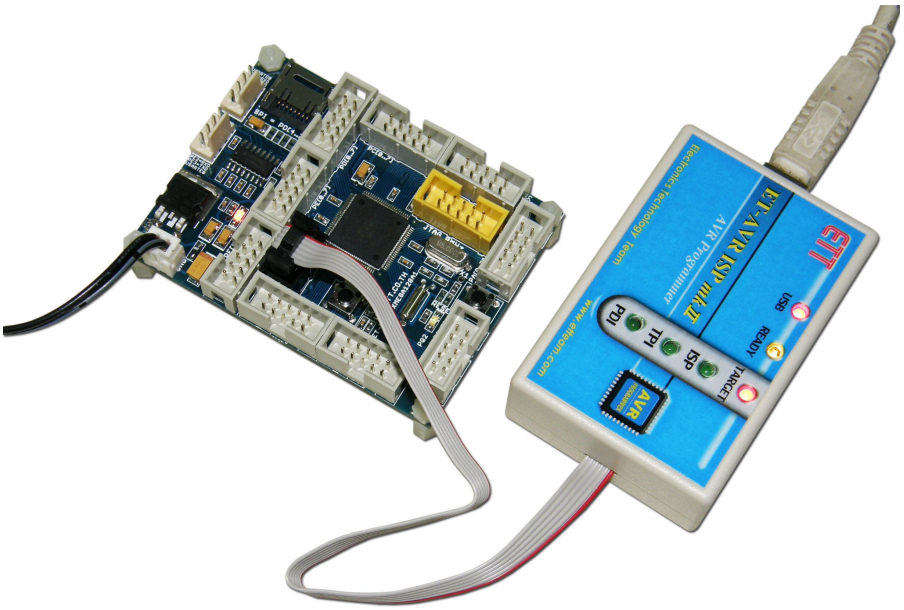
รูปแสดงตัวอย่างการต่อ AVRISP mkII ของ ATMEL



รูปแสดงตัวอย่างการต่อ ET-AVRISP mkII ของ อีทีที

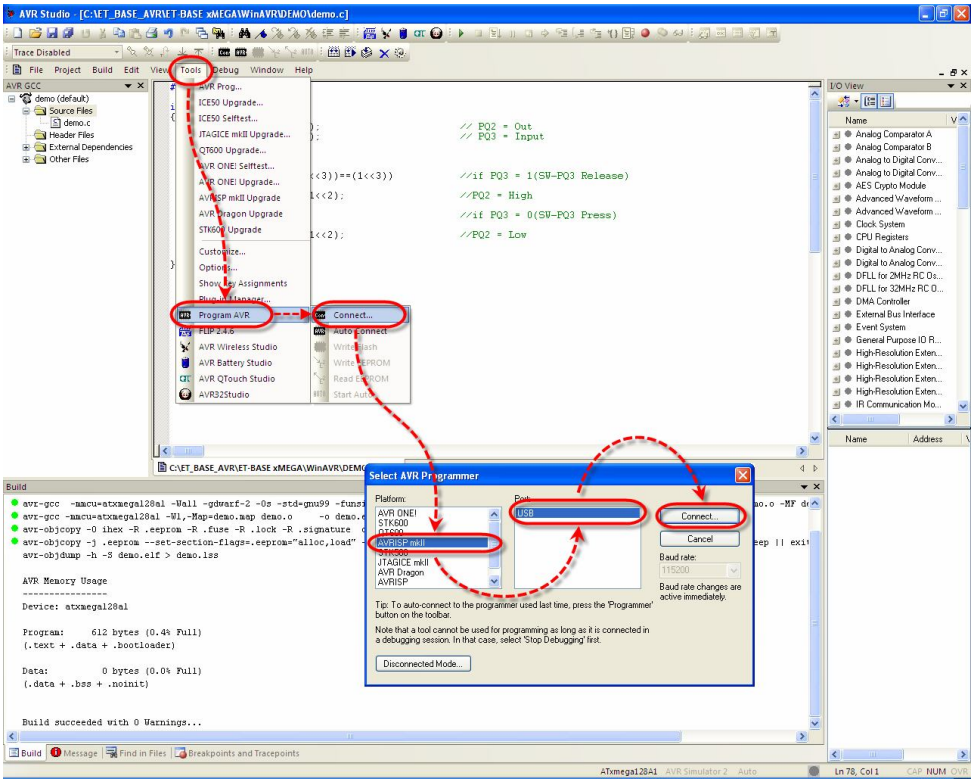
สำหรับลำดับขั้นตอนการโปรแกรม Hex File ด้วย VARISP mkII มีดังนี้

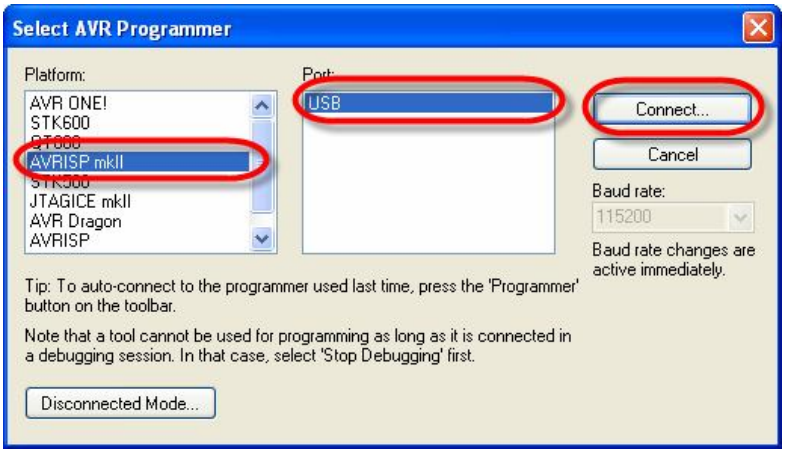
1. จ่ายไฟให้บอร์ดโดยใช้ Adapter จ่ายไฟขนาด 5VDC โดยให้ระมัดระวัง และ ตรวจสอบขั้วของแหล่งจ่ายไฟให้ถูกต้องด้วย ซึ่งถ้าถูกต้องควรจะเห็น LED Power ติดสว่างให้เห็น
2. ต่อสาย PDI ระหว่างเครื่อง AVRISP mkII เข้ากับขั้วต่อ PDI ของบอร์ด ET-BASE xMEGA128A1 โดยให้ตรวจตำแหน่งของขาสัญญาณให้ดี ระวังอย่าเสียบสายกลับด้าน ซึ่งถ้าเป็นเครื่องโปรแกรมและบอร์ดของอีทีที จะเลือกใช้ Connector IDE แบบ 6Pin ชนิดที่ป้องกันการเสียบสายกลับด้านเพื่อป้องกันไว้อยู่แล้ว ถ้าพบผิดความผิดปรกติเช่น LED Power ดับขณะเสียบสายให้รีบถอดสายออกและตรวจสอบสาเหตุความผิดพลาดทันที



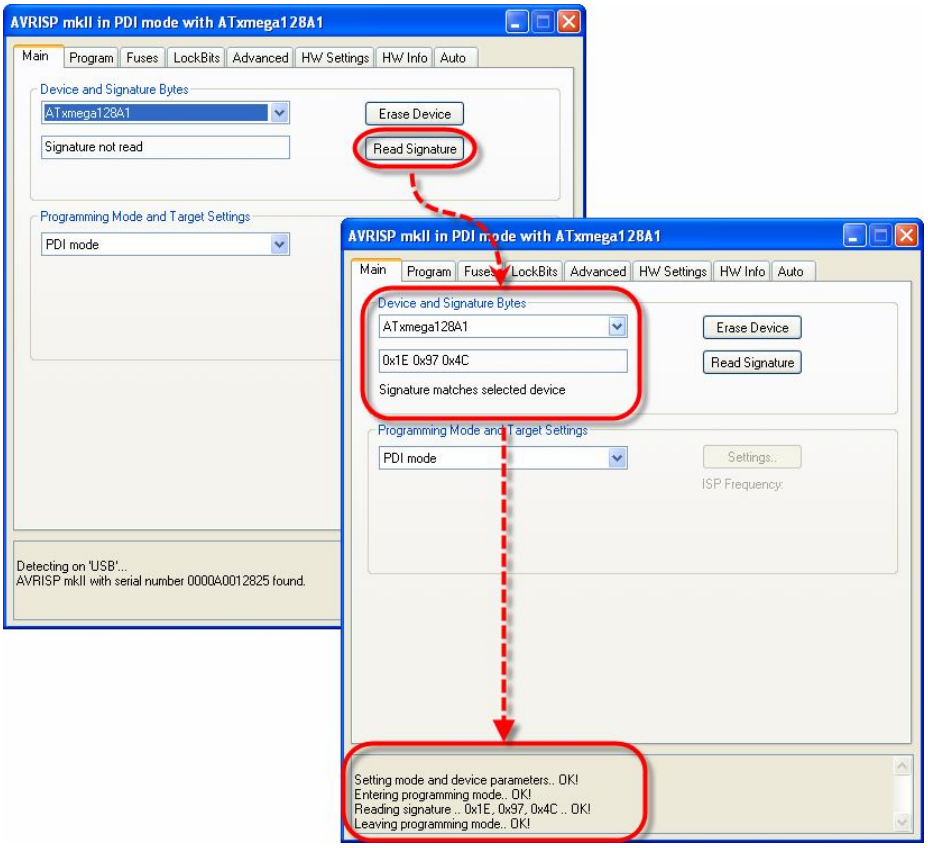
3. เสียบสาย USB ของเครื่อง AVRISP mkII เข้ากับเครื่องคอมพิวเตอร์ PC ซึ่งถ้าเป็นการใช้งานครั้งแรก Windows จะแจ้งว่าพบ new hardware และถามหาการติดตั้ง Driver ให้สั่งติดตั้ง Driver ให้เรียบร้อย โดยใน AVR Studio จะมีไฟล์ Driver ของ AVRISP mkII เตรียมไว้ให้ด้วยแล้ว โดยจะอยู่ใน C:\Program Files\Atmel\AVR Tools\usb ให้ทำการติดตั้ง Driver ให้เรียบร้อย (รายละเอียดศึกษาได้จากคู่มือการใช้งานของเครื่องโปรแกรม ET-AVRISP mkII)

4. เลือกคลิกเมาส์ที่ Tools → Program AVR → Connect.. → AVRISP mkII จากนั้นก็ให้เลือก port เป็น USB พร้อมกับเลือก Connect ดังรูป





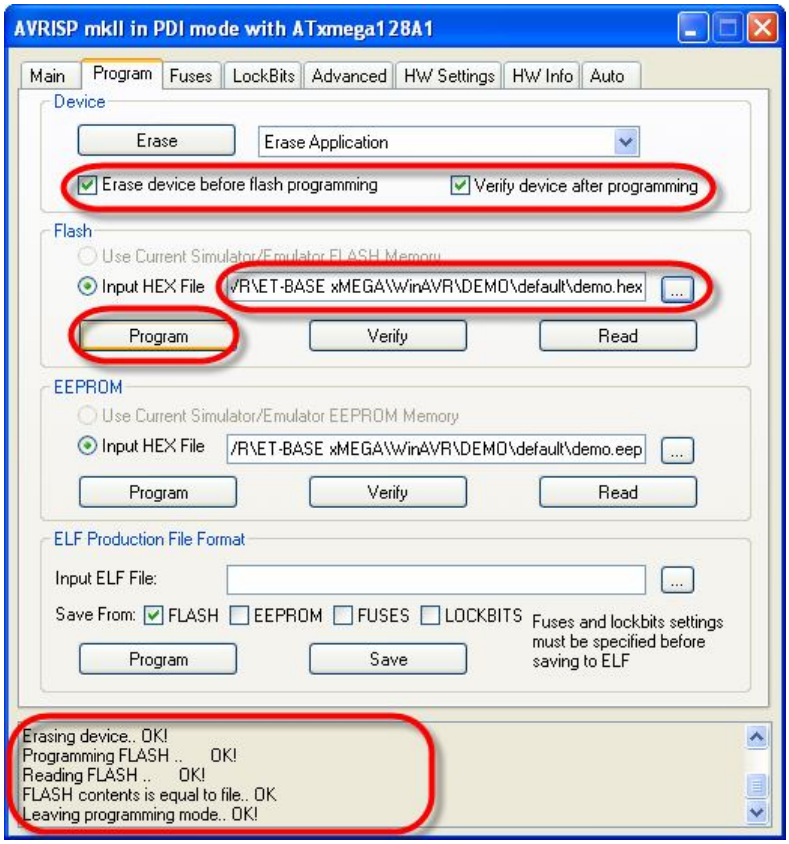
5. ถ้าทุกอย่างถูกต้องโปรแกรมจะเข้าสู่หน้าต่างโปรแกรมของ AVRISP mkII ให้ลองทดสอบการเชื่อมต่อระหว่าง ATxMEGA128A1 กับ AVRISP mkII ดู ว่าสามารถสื่อสารกันได้เรียบร้อยหรือยัง โดยให้เลือกที่ tab ของ Main แล้วเลือกกำหนดเบอร์ MCU เป็น ATxmega128A1 พร้อมกับเลือกการเชื่อมต่อเป็น PDI mode แล้วลองเลือก Read Signature ดู ซึ่งถ้าทุกอย่างถูกต้องโปรแกรมควรต้องอ่านค่า Signature ของ ATxMEGA128A1 ได้อย่างถูกต้อง ดังรูป



6. ให้เลือกไปที่ tab ของ Program พร้อมทั้งเลือก ตัวเลือกต่างๆดังนี้

- Device ให้เลือก Erase device before flash programming และ Verify device after programming
- Flash ให้เลือก Input HEX File ที่ต้องการจะโปรแกรมให้กับ MCU บนบอร์ด
- Fuses และ Lock Bits สามารถเลือกกำหนด และสั่งโปรแกรมค่าได้ตามต้องการ ซึ่งก่อนจะสั่ง Program ค่าของ Fuse Bit ผู้ใช้ควรต้องศึกษารายละเอียดในการกำหนดค่าให้เข้าใจ ซึ่งจะต้องสัมพันธ์สอดคล้องกับความต้องการของระบบ Hardware ที่ใช้อยู่ด้วย ถ้า

ยังไม่แน่ใจในรายละเอียดไม่ควรไปสั่งโปรแกรมค่า ของ Fuse Bit เหล่านี้ เพราะถ้ามีการโปรแกรมค่าของ Fuse Bit ผิดไปอาจส่งผลให้ MCU ทำงานผิดพลาดไปจากความต้องการ ซึ่งในเบื้องต้น ค่า Fuse Bit และ Lock Bit แนะนำให้ข้ามไปก่อนยังไม่ต้องสั่ง โปรแกรมค่าทั้ง สองนี้ ให้จัดการเฉพาะส่วนของการโปรแกรม Flash ด้วย Hex ดังรูป



7. ซึ่งหลังจากสั่งโปรแกรมเสร็จ MCU จะเริ่มต้นทำงานตามคำสั่งใน Hex ที่ได้สั่งโปรแกรมไปแล้วทันที ถ้าใช้เครื่องโปรแกรม ET-AVRISP mkII แต่ถ้าเป็นเครื่องโปรแกรมรุ่นอื่นอาจต้องทำการกดสวิตช์รีเซ็ต 1 ครั้ง เพื่อสั่งให้ MCU เริ่มต้นทำงานหลังจากโปรแกรมเสร็จ