Getting Started with AVR Microcontroller

AN17644: Getting Started with AVR® Microcontroller

Prerequisites

- **Hardware Prerequisites**
  - ATmega328PB Xplained Mini board
  - Two Micro-USB Cables (Type-A/Micro-B)
  - Atmel Power Debugger kit
  - Three female to male wires. One male to male wire.
- **Software Prerequisites**
  - Atmel Studio 7.0
- **Estimated Completion Time: 60 minutes**

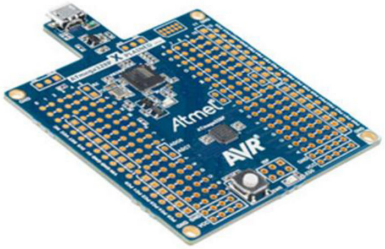
Introduction

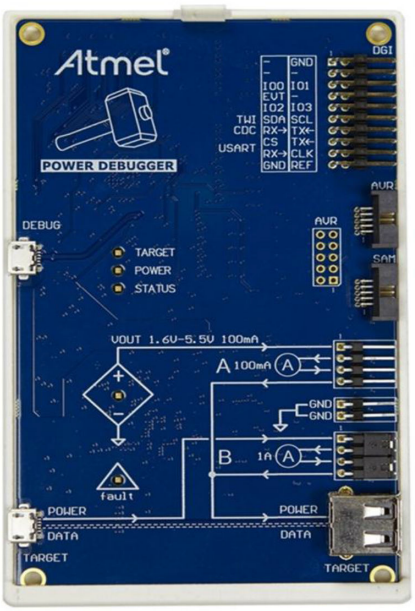
This hands-on will demonstrate how to develop AVR® applications in Atmel Studio 7 along with the rich user interface and other great development tools that it provides.

The ATmega328PB Xplained Mini evaluation kit is a hardware platform for evaluating the Atmel ATmega328PB microcontroller. A fully integrated embedded debugger is included in the kit, which provides seamless integration with Atmel Studio 6.2 or later. Easy access to the features of the ATmega328PB is enabled by the kit, facilitating easy integration of the device in a custom design.

This training module also demonstrates how to save power in applications. The power consumption will be measured using the power debugger board and the ‘Data Visualizer’, which is a new feature in Atmel Studio 7.

The training will start with creating an application on the ATmega328PBXplained Mini board. The initial power consumption of the microcontroller will be measured. After this, different peripherals will be turned off, and the power consumption will be measured to show how much power is saved.





The following topics are covered:

- Creating basic application with GPIO, timer, and ADC
- Debugging and programming with Atmel Studio 7, breakpoints, single stepping, and I/O view
- Measuring the power consumption with Power Debugger and Data Visualizer
- Optimizing the power consumption, applying power saving techniques

Table of Contents

1. Icon Key Identifiers

The following icons are used in this document to identify different assignment sections and to reduce complexity.

Info: Delivers contextual information about a specific topic.

Tip: Highlights useful tips and techniques.

To do: Highlights objectives to be completed.

Result: Highlights the expected result of an assignment step.

Warning: Indicates important information.

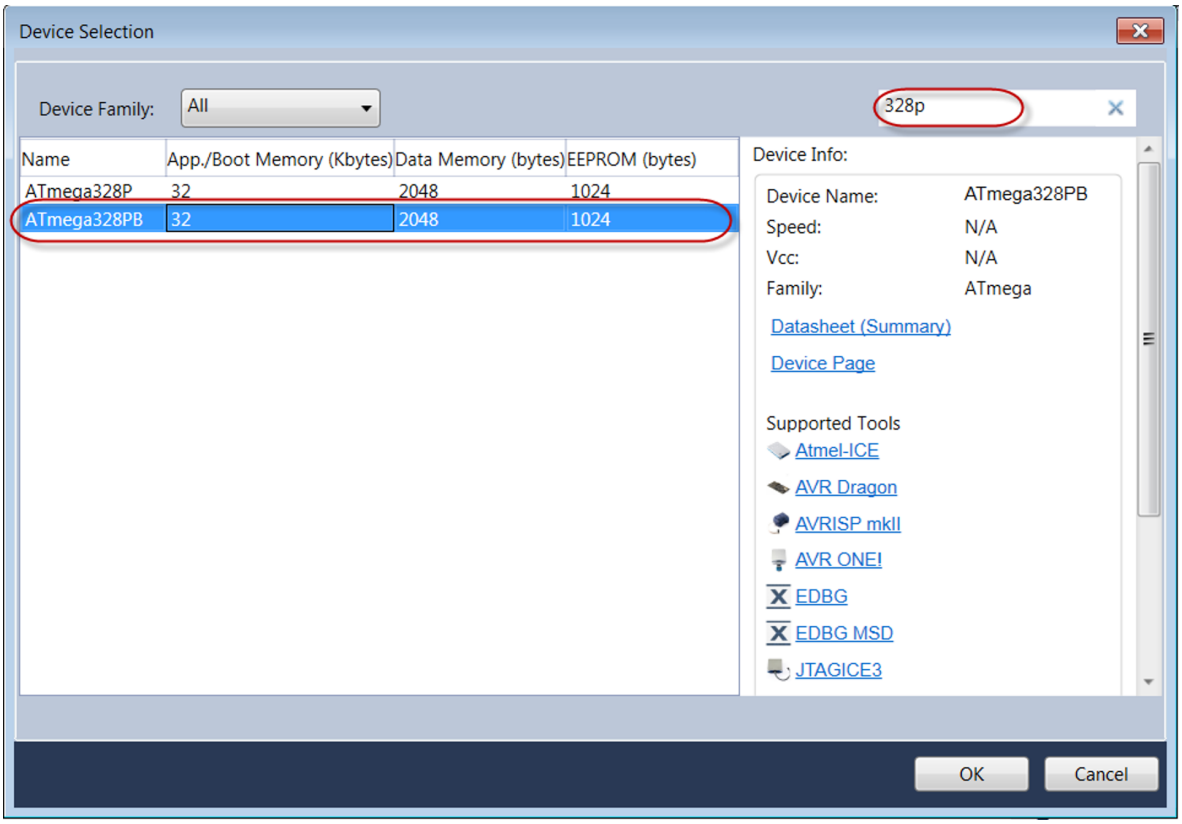
Execute: Highlights actions to be executed out of the target when necessary.

2. Assignment 1: Project Creation

To do: Create a new Project.

1. Open **Atmel Studio 7**.
2. Select **File** → **New** → **Project**.
3. The **New project** window appears. Select **GCC C executable Project** and give the project 'Name' “HANDS-ON_Assignment1_LED_TOGGLE”. Set the 'Location' (any path on the PC) and click **OK**.
4. Now the **Device Selection** window appears as shown in the following image. In the search bar, type **328p** then select the device Atmega328PB and click **OK**.

Figure 2-1. Device Selection



Result: The AVR project is created.

2.1 Application Development: GPIO

An application that controls the LED using the push-button on the board will be created here. The LED will be OFF on pressing the button. Default state is LED ON.

On the ATmega328PB Xplained Mini board, LED0 is connected to pin PB5 and the pushbutton (SW0) is connected to pin PB7.

The data sheet section ***Configuring the pins*** under ***I/O ports*** describes the pin configuration.

1. Each I/O port has three registers: DDRx, PORTx, and PINx where x is I/O port B,C,D,E. The DDRx register is used to configure the port pin direction. 1 - Output; 0 - Input.
2. If the I/O pin is configured as an output pin and if the respective bit in PORTx is written to logic one, the respective port pin is driven high. If the same bit is written to logic zero, the pin will be driven low.
3. The PINx register is used to return the logic level available on the pin.
4. In this code, the direction of PB7 (SW0) should be configured as input and the direction of pin PB5 (LED0) should be configured as output.
5. Here the LED is controlled by the push-button status. As long as SW0 is pushed down - state (0): LED0 will be OFF (0). As long as SW0 is released - state (1): LED0 will be ON.

To do: Refer to the **Register Description** section in the data sheet as shown in the following image to set the necessary port pin configurations.

Figure 2-2. Data sheet: Register Description

18.4.2. Port B Data Register

Name: PORTB

Bit	7	6	5	4	3	2	1	0
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

18.4.3. Port B Data Direction Register

Name: DDRB

Bit	7	6	5	4	3	2	1	0
	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

18.4.4. Port B Input Pins Address

Name: PINB

Bit	7	6	5	4	3	2	1	0
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

To do: Add missing code in the main.c file, in function main ( ).

1. For configuring the direction bit of pin PB5, write the DDB5 bit to logic 1 in the DDRB register before `while(1)`, in the main.c file, in function `main ( )`.

Tip: Example: How to configure bits to logic 1 or logic 0 in register is shown below. Here, direction bit of PB5 is written to 1 and status of pin PB5 is written to 0 (LED0 is turned off):

```
DDRB |= 1<<DDRB5; // direction bit of pin PB5 is written to 1.
PORTB &= ~(1<<PORTB5); // pin PB5 is 0
```

Info: PORTB5 is defined as 5 in iom328pb.h. All the register bit definitions can be found in iom328pb.h. In Atmel Studio, right-click on PORTB5 and select **Goto Implementation**, to check the PORTB5 definition.

2. Read the PB7 bit from the PINB register in while(1) and add the code to turn LED0 ON and OFF as per the status mentioned below.
   When Pin PB7 =0, Button (SW0) is pushed down and LED0 should be OFF.

   When Pin PB7 =1, Button (SW0) is released and LED0 should be ON.
3. After completion of the code, in the file menu, select **Build → Build Solution**. The build should finish successfully with no errors.

Result: The application 'LED control on pushbutton' is created.

The code should look like the image shown below.

Figure 2-3. Code: LED Control on Pushbutton

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


int main(void)
{
    DDRB |= 1<<DDRB5;   // Direction of pin PB5 set as Output
    while(1)
    {
        if(!(PINB & (1<<PINB7)))        // read PIN PB7
        {
            PORTB &= ~(1<<PORTB5);     // PB7 is low so LED OFF
        }
        else
        {

            PORTB |= 1<<PORTB5;       // By default PB7 is high, so LED ON
        }
    }
}
```

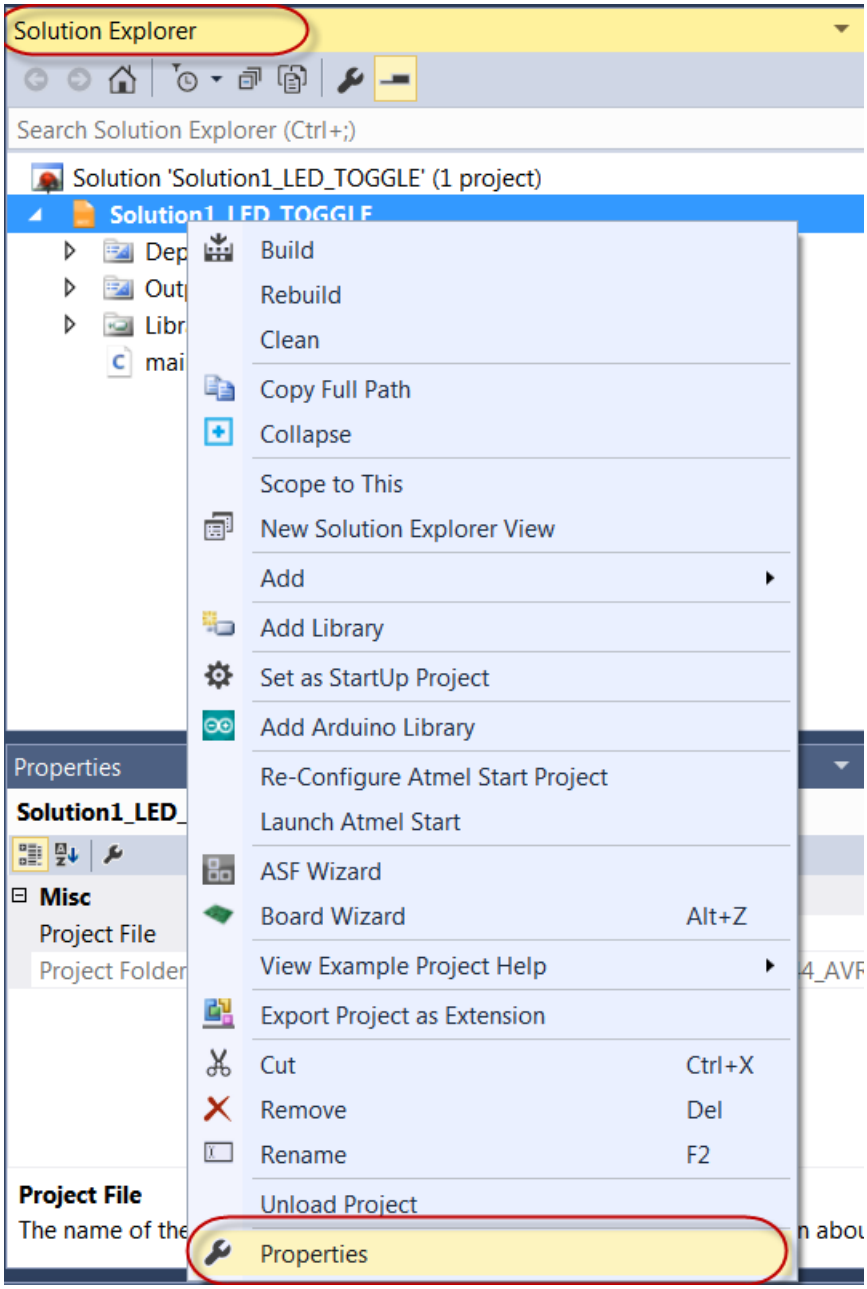
2.2 Debugging the Application

To do: Debug the application 'LED control on pushbutton'.

Warning: If any error message is displayed while debugging, refer the Appendix B: Troubleshooting Guide.

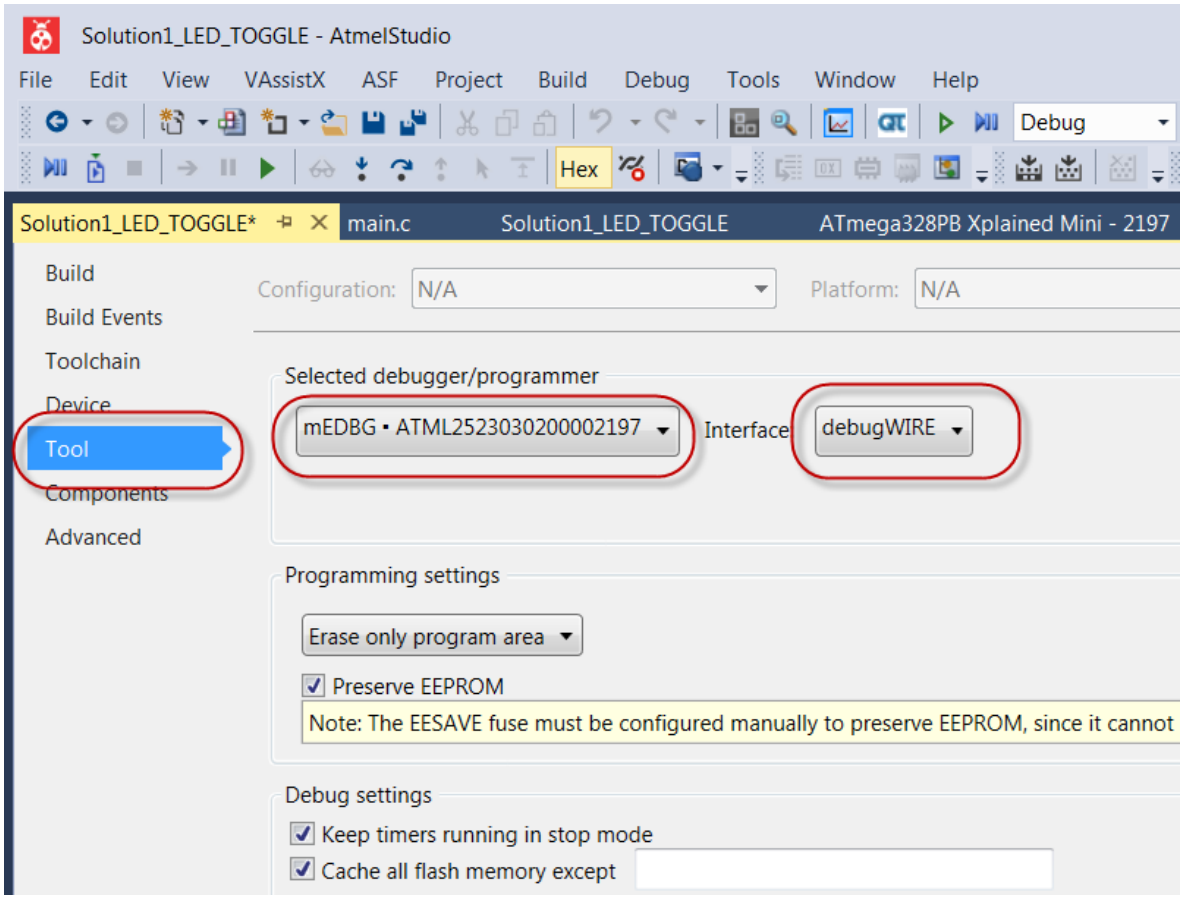
1. Power the ATmega328PB Xplained Mini board by connecting Micro-USB cable to the PC.
2. In the **Solution Explorer**, right-click the project and select **Properties**.

Figure 2-4. Solution Explorer



3. Under **Tool**, select **mEDBG** and **debugWIRE** as interface as shown below.

Figure 2-5. Project Properties



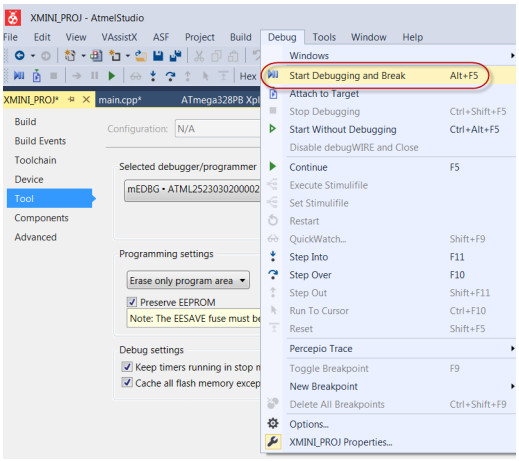
Info: Here the embedded debugger is used on the ATmega328PB Xplained Mini board to debug the ATmega328PB via debugWIRE.

Info: The debugWIRE interface uses a one-wire, bi-directional interface to control the program flow, execute AVR instructions in the CPU, and to program the different non-volatile memories. When the debugWIRE Enable (DWEN) fuse is programmed and Lock bits are unprogrammed, the debugWIRE interface within the target device is activated.

Info: Limitations of debugWIRE: debugWIRE communication pin (dW) is physically located on the same pin as external reset (RESET). An external reset source is therefore not supported when the debugWIRE is enabled.

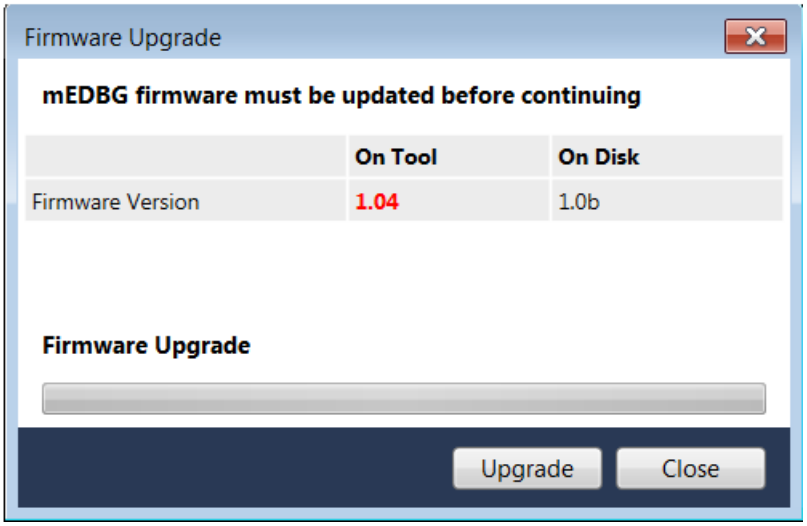
4. Select **Debug** → **Start Debugging and Break**.

Figure 2-6. Debug Menu



Info: If the firmware on the on-board debugger is lower than the one in Atmel Studio installation, it will be asked to upgrade the firmware.

Figure 2-7. Firmware Upgrade

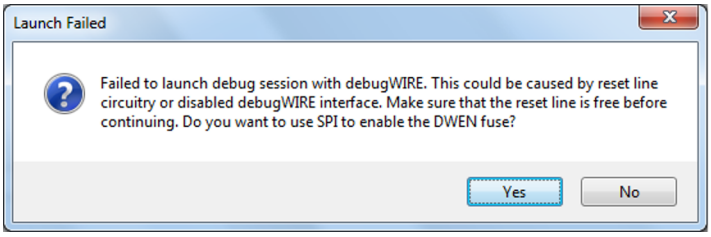


Select **Upgrade** and when the progress bar is complete, select **Close**.

Now, start debugging by selecting **Debug** → **Start Debugging and break**.

Warning: If the DWEN fuse is not enabled, an error message is displayed. Click **Yes** and Atmel Studio 7 will use the SPI to set the fuse as shown below.

Figure 2-8. Enable the DWEN Fuse



5. Wait until Atmel Studio 7 completes the programming and the status is **Ready** (at the bottom left corner of Atmel Studio 7 window).

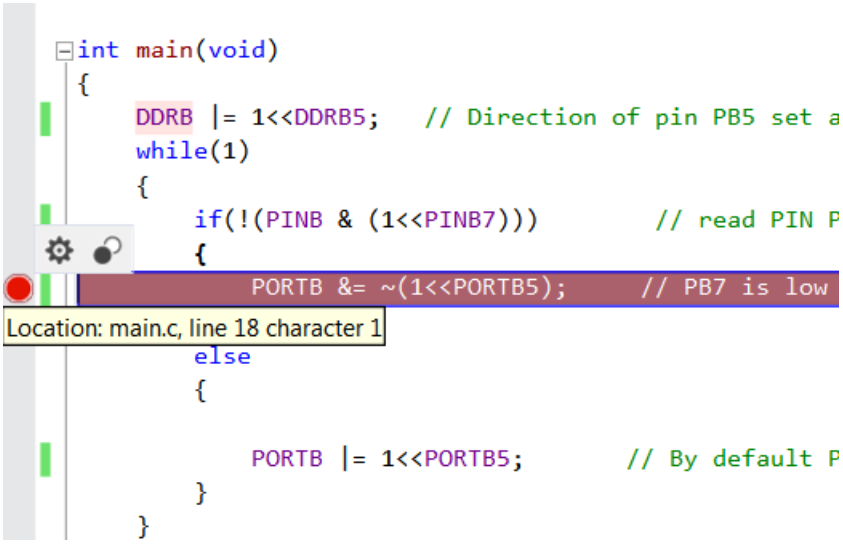
Figure 2-9. Programming Status

Result: The debugger is started and breaks in main. Debugging can now be started.

Info: Different debug options are available in the **Debug** menu.

6. Insert the breakpoint where LED goes OFF. Go to the line in the source code where LED goes OFF and to insert a breakpoint, click on the margin, as shown below.

Figure 2-10. Breakpoint Inserted



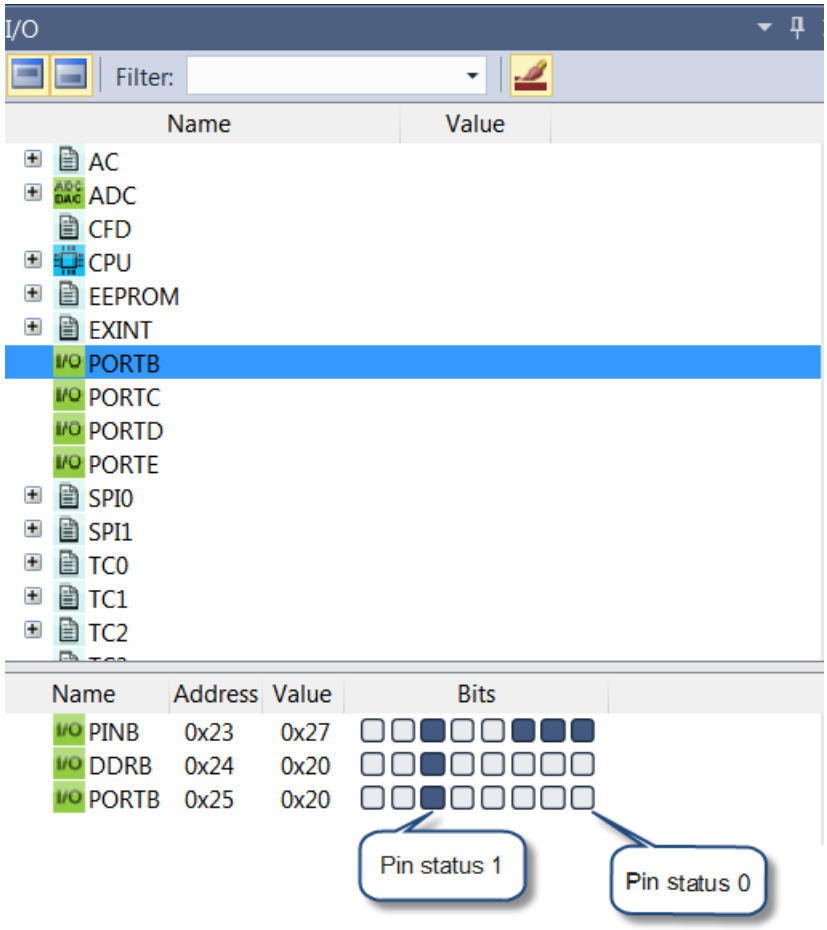
7. Run to Breakpoint by clicking **Debug** → **Continue**.

Tip: Find the Play button on top of the Atmel Studio 7 or find it in the **Debug** menu.

8. Press the SW0 pushbutton and observe that the execution stops at the breakpoint.
9. To check the status of all the PORTB pins, open the I/O view window by selecting **Debug** → **Windows** → **I/O** and click on the PORTB register group, as shown in the image below.

Info: In the I/O view, the status of all the peripherals can be observed. Pin status is indicated for PB0 to PB7 from right to left under the **Bits** column. As shown in the image below, pin PB5 status is 1 (filled square). The empty square indicates that the bit status is zero.

Figure 2-11. I/O View



10. Do single-step debugging by pressing F10 and observe the status of PB7 and PB5.

Info: PB7 is leftmost bit and PB0 is rightmost bit in the PORTB register.

11. In the I/O view, select bit 5 and click on it. When the status of the bit gets changed from high to low, LED0 on the board is turned OFF. Click bit 5 once more to toggle the status and observe the LED0 status on the board.

12. Remove the breakpoint by clicking on the breakpoint and run the code by clicking **Debug** → **Continue**.

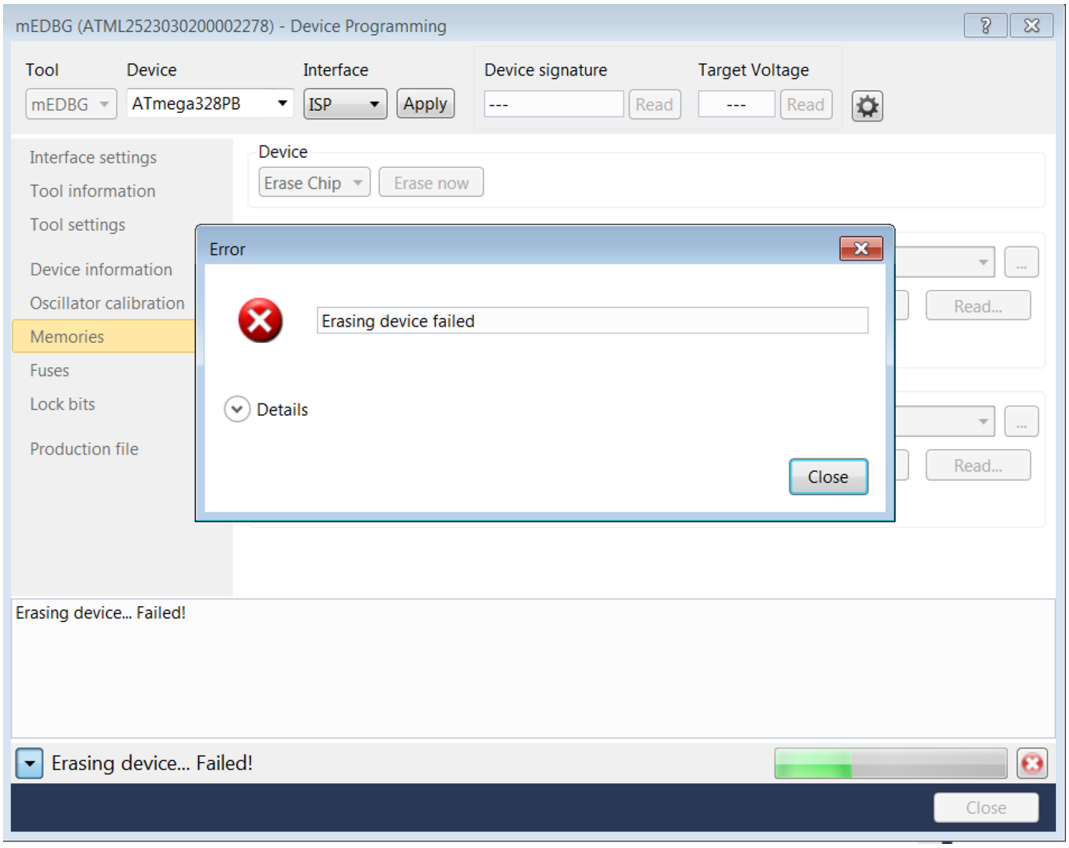
Result: The application is successfully programmed to the ATmega328PB Xplained Mini board.

13. Press the SW0 pushbutton and observe the LED0 toggle.
14. Exit safely from debug mode by disabling debugWIRE. Select **Debug** → **Disable debugWIRE and Close**.

Result: The AVR project is successfully completed.

Warning: It is important to disable debugWIRE. If debug mode is not exited by selecting "Disable debugWIRE and Close" in the Debug menu, the DWEN fuse will be enabled and the target will still be in debug mode, i.e. it will not be possible to program the target using the ISP interface. An error message will be displayed while programming, as shown below. In this case, the user will has to open the project and select **Debug → Start Debugging and Break** or **Debug → Continue**. Then, exit the debug mode by selecting **Debug → Disable debugWIRE and Close**.

Figure 2-12. Device Programming



3. Assignment 2: Application Development: Timer

The ATmega328PB has three 16-bit Timer/Counter instances: TC1, TC3, and TC4. Here, an application will be developed to generate the PWM using TC1. It will be verified with the LED dimming application.

The mode of operation, i.e. the behavior of Timer/Counter and Output Compare pins, is defined by the combination of the Waveform Generation mode and Compare Output mode. The double buffered Output Compare Registers (OCRnA/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OCnA/B).

In this application, Fast PWM Mode will be used to generate PWM.

In the data sheet, from section ***Pin Configurations***, it can be seen that pin PB1 is the OC1A pin. The TC1 PWM output will be generated on pin PB1, which is for Compare Output Mode for Channel A.

To do: Create an application for LED dimming using PWM with TC1.

1. Open the Assignment 2 project from **Assignment2_TC1_PWM\Assignment2_TC1_PWM.atsln**, provided with this training.
   In the main.c file some of the code is already completed. Initialization of TC1 is done in the function `init_TC1_pwm()`. Some missing code needs to be added in the function `init_TC1_pwm()`.
2. Refer to the data sheet section ***Register Description*** and take a look at the registers **TCn Control Register A** and **TCn Control Register B**.
3. Take a look at the `init_TC1_pwm()` function and verify the compare output mode configuration, as described below.
   Compare output mode:

   For the TCCR1A register, bits COMA[1:0] and COMB[1:0] control the Output Compare pins (OC1A and OC1B respectively). Here, the output should be generated on the OC1A pin only. So, for the register TCCR1A, the bits are COMA1=1 and COMA0=0 (non-inverting mode from data sheet table **Compare Output Mode, Fast PWM**).

   The TCCR1A register is configured as below,

   ```
   TCCR1A = (1<<COM1A1);
   ```

4. **Waveform Generation mode**:
   Add the missing code in the `init_TC1_pwm()` function for configuration of 'Bits 1:0 – WGM[1:0]: Waveform Generation Mode'. Refer to table **Waveform Generation Mode Bit Description** in the data sheet, and configure the bits accordingly for Fast PWM, 8-bit mode.

Tip: Configure the WGM10 and WGM11 in the TCCR1A register and WGM12 in the TCCR1B register according to table **Waveform Generation Mode Bit Description** in the data sheet.

5. Take a look at the `init_TC1_pwm()` function and verify the configuration for the clock source and prescaler to be used by the Timer/Counter, which decides the frequency of the PWM. The internal clock source divided by 1024 is configured as below.
   This is done by writing 1 to 'Bits 2:0 – CS[2:0]: Clock Select' in the TCCR1B register.

```
TCCR1B |= (1 << CS10)|(1 << CS12); // 1024 prescaler
```

Info: The frequency of the PWM is calculated by $F_{PWM} = \frac{F_{CLK}}{N \cdot (1 + TOP)}$. Here N = 1024, TOP = 0xFF (Fast PWM 8-bit mode), and F_{CLK} = 16 MHz.

6. Take a look at the `init_TC1_pwm()` function and verify the enabling of the Timer/Counter Output Compare A Match interrupt.
   This is done by configuring the Timer/Counter 1 Interrupt Mask Register (TIMSK1) as below.

```
TIMSK1 |= (1 << OCIE1A);
```

7. The OC1RA register decides the duty cycle of PWM. For LED dimming, gradually decrease in duty cycle is needed. Now, add the missing code in interrupt ISR to assign the duty cycle to the OC1RA register.
8. Configure the direction bit for PB1 as output, which is done in the DDRB direction register by writing 1 to bit DDRB1. Add the missing code in the `main()`.
9. On the ATmega328PB Xplained Mini board, LED0 is connected to PB5 and PWM is generated on PB1. To view the LED dimming, connect the wire from PB5 to PB1 on the ATmega328PB Xplained Mini board, as shown below.

Info: PB5 and PB1 are connected as shown with a red arrow in the figure below.

Figure 3-1. Connection PB5 to PB1

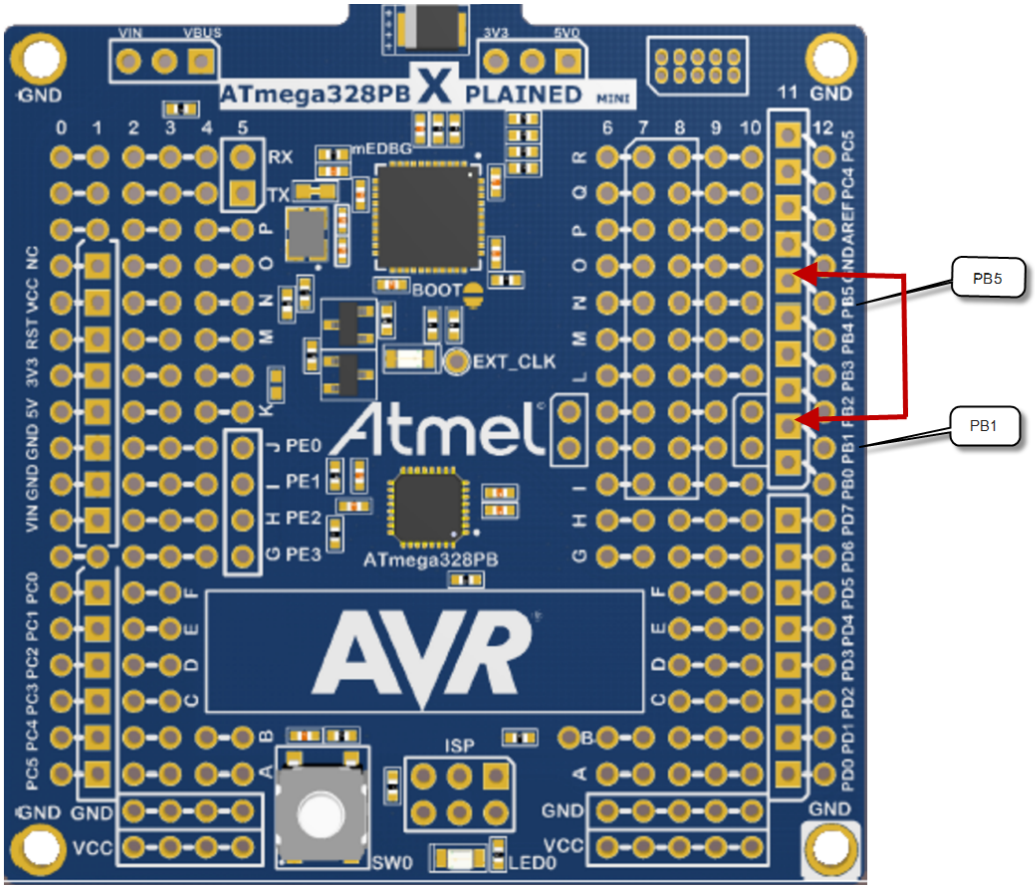
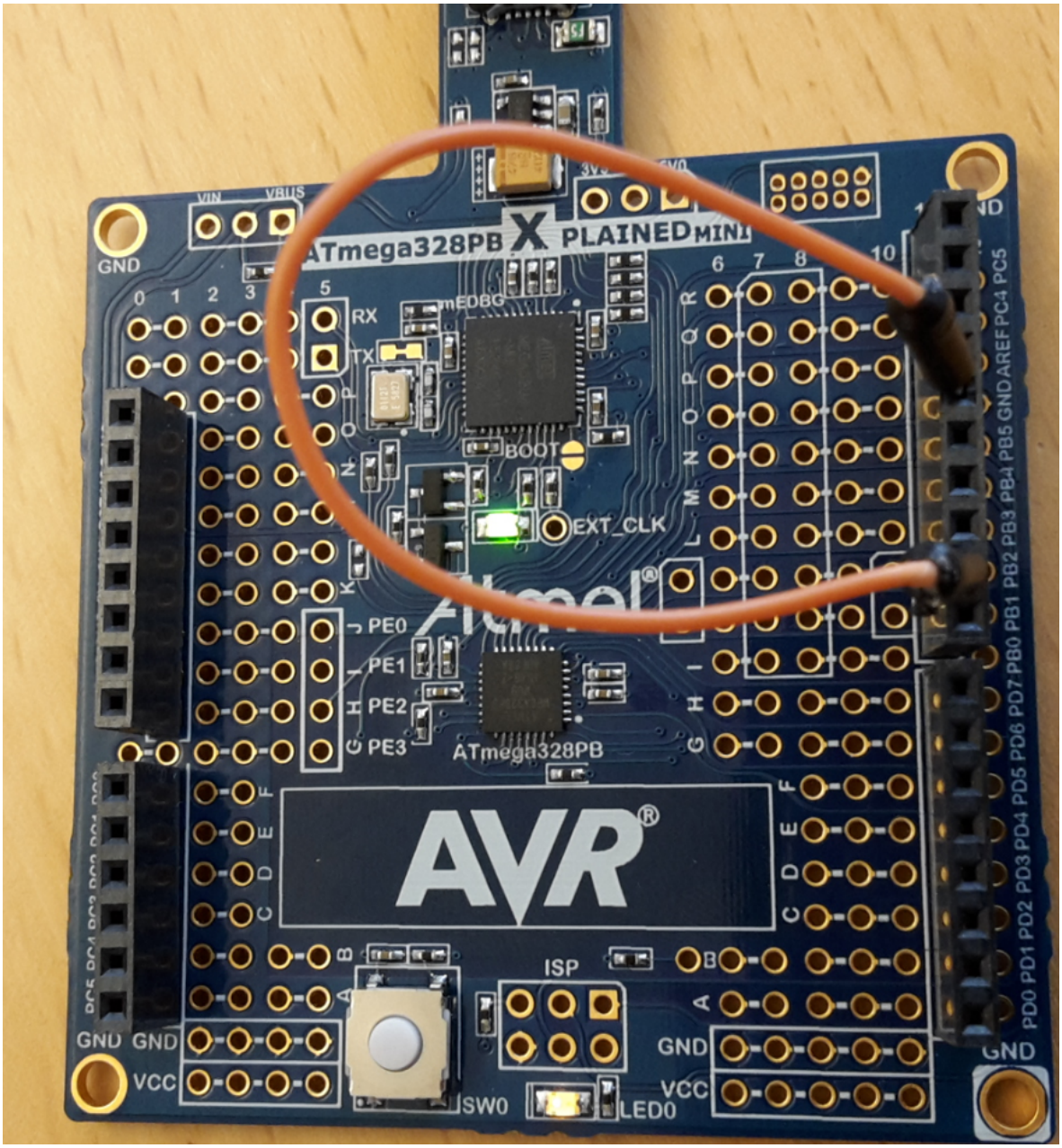


Figure 3-2. Visual Representation of HW Setup for Assignment 2



10. After completion of the code, in the file menu, select **Build** → **Build Solution** or press the **F7** key. The build should finish successfully with no errors.

Result: The code should look like the image shown below.

Figure 3-3. Code:LED Dimming

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

unsigned int duty_cycle=0xff;


ISR(TIMER1_COMPA_vect)
{

    duty_cycle--;
    if (duty_cycle==0)
    {
        duty_cycle=0xff;
    }
    OCR1A = duty_cycle;

}

//Timer1 as PWM
void init_TC1_pwm(void)
{
    TCCR1A = (1<<COM1A1) ;
    TCCR1A |= (1<<WGM10);

    TCCR1B |= (1 << WGM12);
    TCCR1B |= (1 << CS10)|(1 << CS12);    // 1024 prescaler

    TIMSK1 |= (1 << OCIE1A);  // enable interrupt
}

int main(void)
{

    DDRB |= (1<< DDRB1 ); // Set direction of PB1 as OUTPUT

    init_TC1_pwm();
    sei();  //enable global interrupts

    while(1)
    {

    }
}
```

11. Program the application by selecting **Debug** → **Continue** and observe the dimming of LED0.
12. Exit from the debug mode by disabling debugWIRE. Select **Debug** → **Disable debugWIRE and Close**.

Result: Assignment 2: 'LED dimming using PWM' is successfully completed.

4. Assignment 3: Application Development: ADC

In this assignment, an application will be created to read ADC channel 0 after every 500 ms. ADC channel 0 is pin PC0.

The ATmega328PB device features a 10-bit successive approximation ADC. The ADC contains a Sample and Hold circuit, which ensures that the input voltage to the ADC is held at a constant level during conversion. By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution.

The ADC has a separate analog supply voltage pin, AV_{CC}. AV_{CC} or an internal 1.1V reference voltage may be connected to the A_{REF} pin by writing to the REFSn bits in the ADMUX register.

The analog input channel is selected by writing to the MUX bits in the ADC Multiplexer Selection register ADMUX.MUX[3:0].

The ADC is enabled by writing a '1' to the ADC Enable bit in the ADC Control and Status Register A (ADCSRA.ADEN).

The ADC generates a 10-bit result, which is presented in the ADC Data registers; ADCH and ADCL.

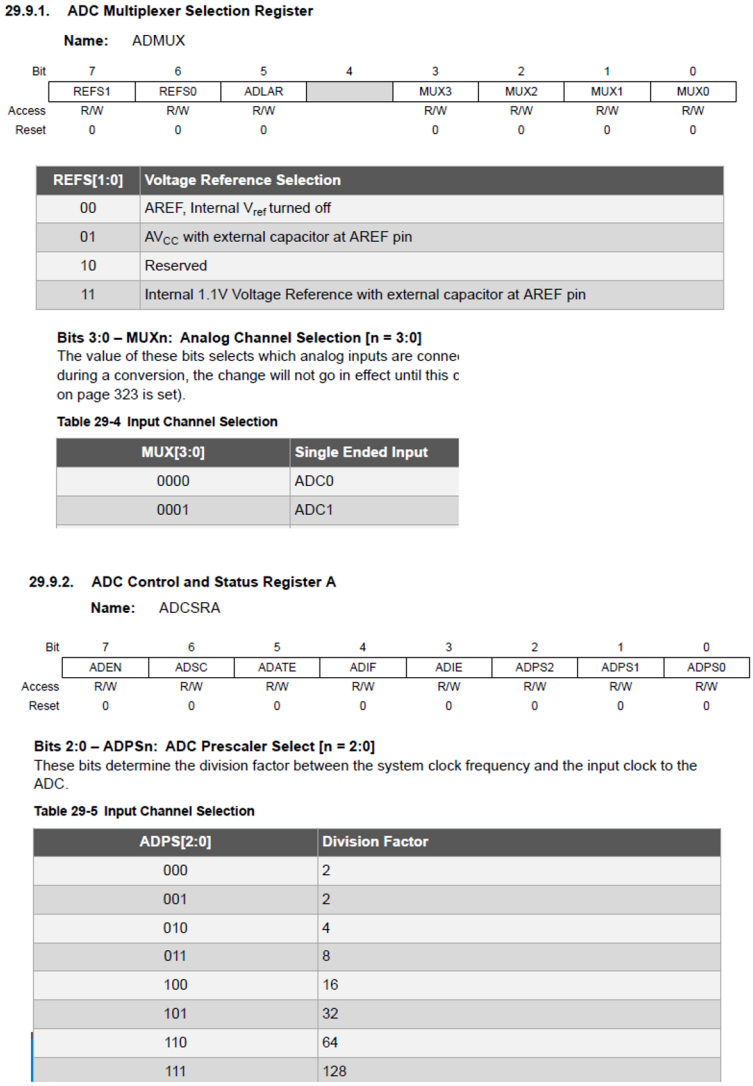
A single conversion is started by writing a '1' to the ADC Start Conversion bit in the ADC Control and Status Register A (ADCSRA.ADSC). ADCS will stay high as long as the conversion is in progress, and will be cleared by hardware when the conversion is completed.

To do: Create an application to read the ADC after every 500 ms.

1. Open the Assignment 3 project from **Assignment3_ADC/Assignment3_ADC.atsln**, provided with this training.
   In the main.c file some of the code is already completed. Initialization of ADC is done in the `initADC()` function. Some missing code needs to be added in the `initADC()` function.
2. Refer to the data sheet section ***Register Description*** and take a look at the registers **ADC Multiplexer Selection Register** and **ADC Control and Status Register A**.

Info: ADC Multiplexer Selection Register and **ADC Control and Status Register A** are shown in the image below.

Figure 4-1. Data sheet: ADC Registers



29.9.1. ADC Multiplexer Selection Register

Name: ADMUX

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Bits 3:0 – MUXn: Analog Channel Selection [n = 3:0]
The value of these bits selects which analog inputs are conne
during a conversion, the change will not go in effect until this c
on page 323 is set).

Table 29-4 Input Channel Selection

MUX[3:0]	Single Ended Input
0000	ADC0
0001	ADC1

29.9.2. ADC Control and Status Register A

Name: ADCSRA

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 2:0 – ADPSn: ADC Prescaler Select [n = 2:0]
These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 29-5 Input Channel Selection

ADPS[2:0]	Division Factor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Bit 7 – ADEN: ADC Enable
Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

3. Add missing code to configure ADC Voltage Reference Selection to $V_{REF} = AV_{CC}$, in `initADC()`. Refer to the ADCMUX register in the image above and complete the code below. Select $V_{REF}=AV_{CC}$.

```
ADMUX |= (1<xxxx);
```

Tip: Configure the REFS0 to 1 in the ADCSRA register.

4. Refer to the ADCSRA register in the image above and configure the prescaler to 128 and enable the ADC. Complete the code below:

```
ADCSRA |= (1<<xxxx) | (1<<xxxx) | (1<<xxxx) |(1<<xxxx);
```

Tip: Configure ADPS2, ADPS1, ADPS0, and ADEN to 1 in the ADCSRA register.

5. Take a look at the `ReadADC()` function and verify the ADC channel selection, as described below. The ADC channel is selected with safety mask.

```
ADMUX = (ADMUX & 0xF0) | (ADCchannel & 0x0F);
```

6. Take a look at the `ReadADC()` function and verify how the ADC start of conversion and ADC read is done, as described below.

```
//single conversion mode
ADCSRA |= (1<<ADSC);
// wait until ADC conversion is complete
while( ADCSRA & (1<<ADSC) );
return ADC;
```

7. Now, build the solution by pressing the **F7** key. The build should finish successfully with no errors. Now, the final code should look like the image shown below.

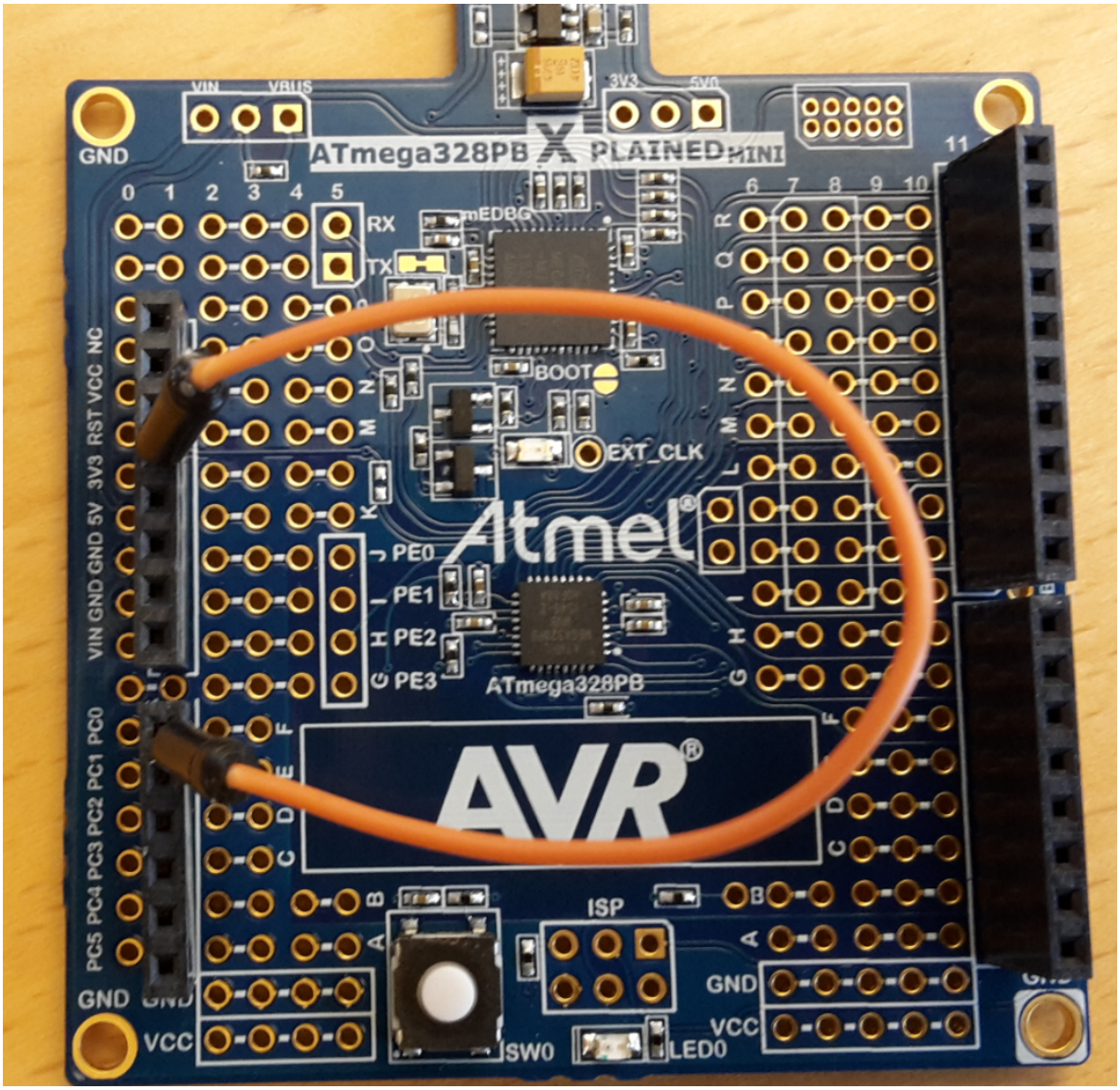
Info: Code for 500 ms delay and LED toggle is already added in `while(1)`.

Figure 4-2. Code: ADC Read

```
#define F_CPU 16000000UL //16 MHz
#include <avr/io.h>
#include <util/delay.h>

uint16_t adc_res=0;

void InitADC()
{
    // Select Vref=AVcc
    ADMUX |= (1<<REFS0);
    //set prescaler to 128 and enable ADC
    ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADEN);
}

uint16_t ReadADC(uint8_t ADCchannel)
{
    //select ADC channel with safety mask
    ADMUX = (ADMUX & 0xF0) | (ADCchannel & 0x0F);
    //single conversion mode
    ADCSRA |= (1<<ADSC);
    // wait until ADC conversion is complete
    while( ADCSRA & (1<<ADSC) );
    return ADC;
}


int main(void)
{
    DDRB &= ~(1<<DDB7); //Set PORTB7 as input
    DDRC &= ~(1<<DDC0); //Set PORTC0 as input
    DDRB |= (1<<DDB5);  //Set PORTB5 as output

    InitADC();

    while (1)
    {
        _delay_ms(500);
        PINB |= 1<<PINB5 ;  // toggle LED
        adc_res=ReadADC(0)  ;
    }
}
```

8. Connect a wire from PC0 to 3V3 on the ATmega328PB Xplained Mini board.

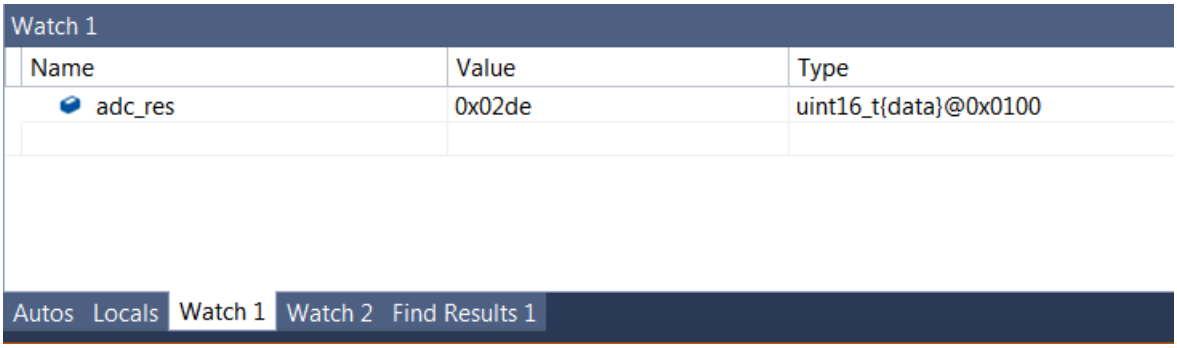
Figure 4-3. Visual Representation of HW Setup for Assignment 3



9. Insert the breakpoint at `adc_res=ReadADC(0);` by clicking on the margin on the line of the code.
10. Program the code by selecting **Debug** → **Continue** and observe the execution stop at the breakpoint.
11. Do the single step debugging by pressing **F10**.
12. Observe the ADC result in the variable 'adc_res' by placing the cursor on 'adc_res'.
13. Add a watch window by right-clicking on 'adc_res' and select **Add Watch**.

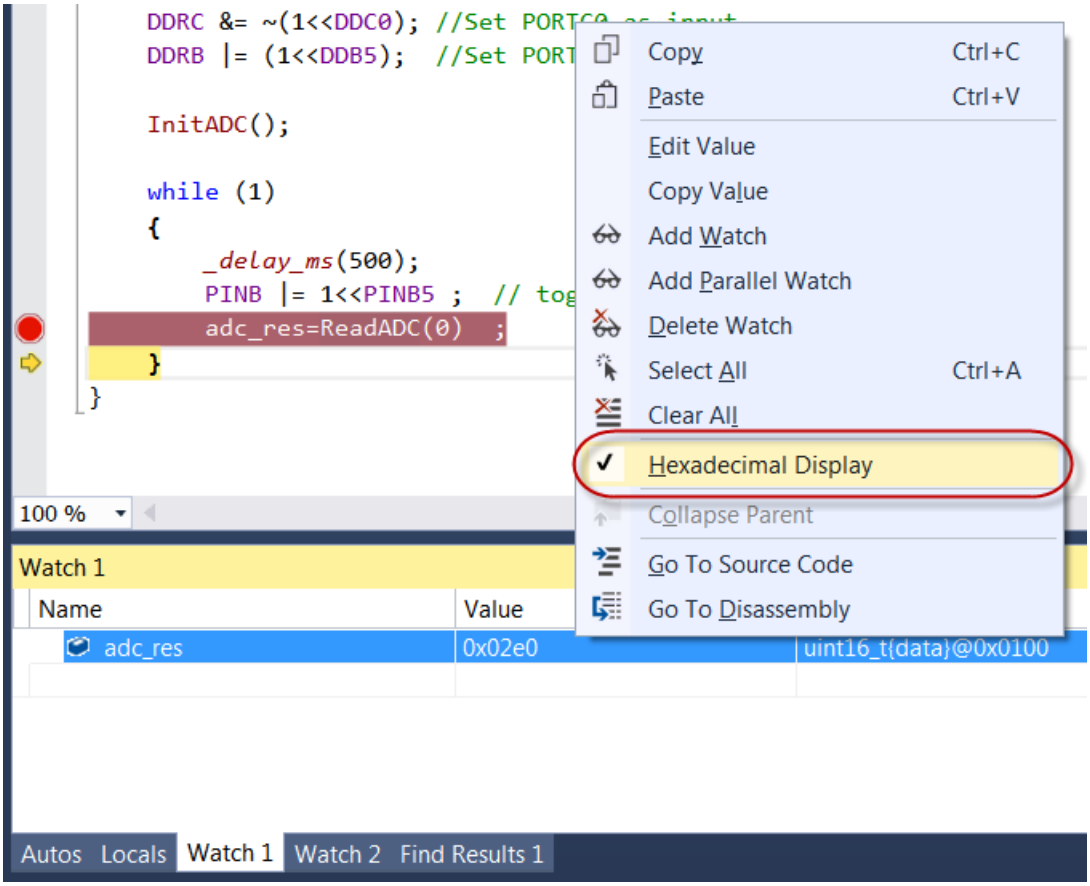
Result: The **Watch1** window will be displayed as below.

Figure 4-4. Watch1 Window



14. To display the 'Value' in decimal or hexadecimal, select 'adc_res' in the **Watch1** window and **Right click → Hexadecimal Display**. Click to remove or add the check mark as shown below.

Figure 4-5. Hexadecimal Display



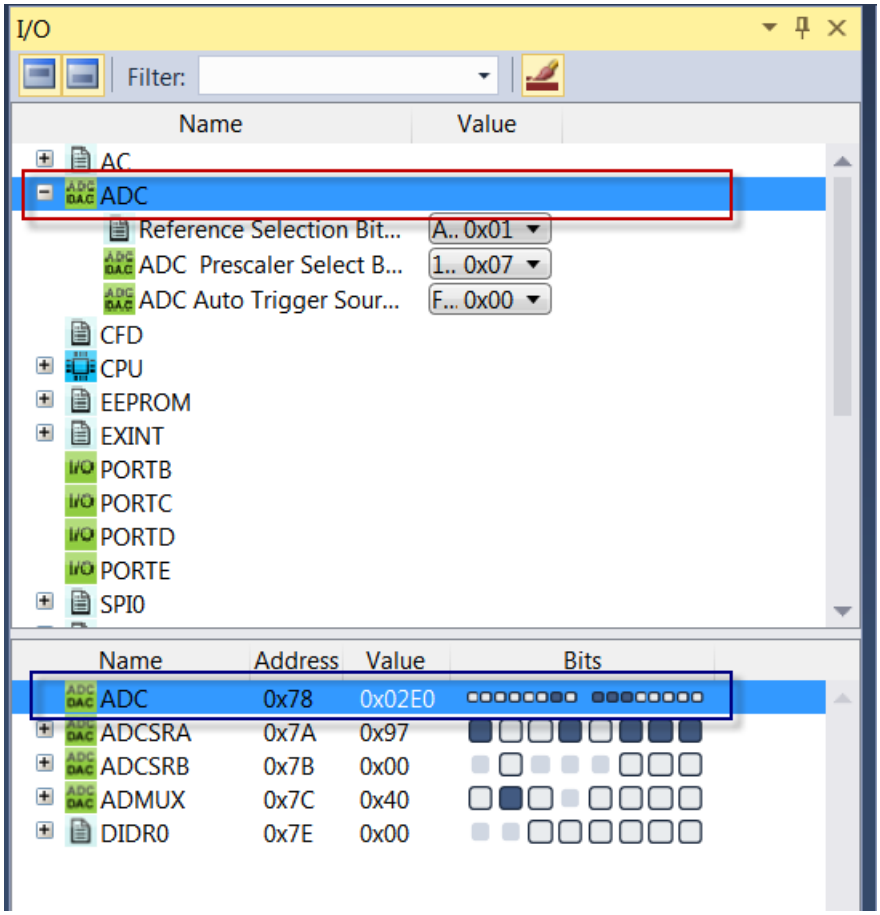
Result: The check mark will be removed and the value will be displayed in decimal in the **Watch1** window.

Note: Here, V_{REF} = AV_{CC} = 5V and ADC0 (PC0) is connected to 3.3V.

15. Open the I/O view by selecting **Debug → Windows → I/O**. Click on the ADC module as shown with red marked square.

Result: All the registers in the ADC module will be listed in the bottom window. Register 'ADC' with blue marked square shows the ADC result.

Figure 4-6. I/O View



16. Remove the breakpoint (by clicking on the breakpoint) and continue the execution by selecting **Debug → Continue**.
17. Exit the debugging by selecting **Debug → Disable debugWIRE and Close**.

Result: Assignment 3: ADC read with 500 ms delay is completed successfully.

5. Assignment 4: Measuring Power Consumption using Power Debugger

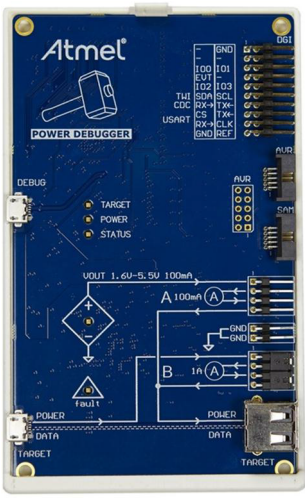
The Power Debugger is a CMSIS-DAP compatible debugger, which works with Atmel Studio 7.0 or later. The Power Debugger streams power measurements and the application debug the data to Atmel Data Visualizer for real-time analysis.

The Power Debugger has two independent current sensing channels for measuring and optimizing the power consumption of a design.

The 'A' channel is the recommended channel for accurately measuring low currents.

The 'B' channel is the recommended channel for measuring higher currents with lower resolution.

Info: A quick start guide is included in Power Debugger kit for convenience.



Info: Frequent updates of the Power Debugger firmware are provided in order to improve its features but also to give support to new MCUs. If required, look at Appendix A to upgrade the Power Debugger firmware.

Here, the Power Debugger board needs to be connected to the ATmega328PB Xplained Mini board.

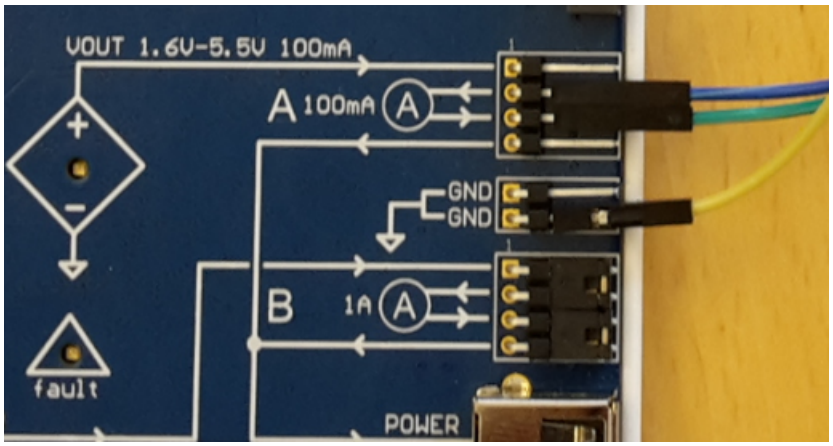
The ‘A’ and 'B' channel current measurement ports on the ‘Power Debugger’ board depicted with ammeter symbols on the silkscreen. The voltage supply is connected to the input of the ammeter, and the load (target) is connected to the output. With the following setup, the power debugger board measures the consumption on the AVR core.

To do: Refer to the following table and image, and make the required connections to connect the Power Debugger board and the ATmega328PB Xplained Mini board.

Table 5-1. Power Debugger and ATmega328PB Xplained Mini Board Connections

Power Debugger	ATmega328PB Xplained Mini
Port A input: Blue wire	5V
Port A output: Green wire	V_{CC}
GND: Yellow wire	GND

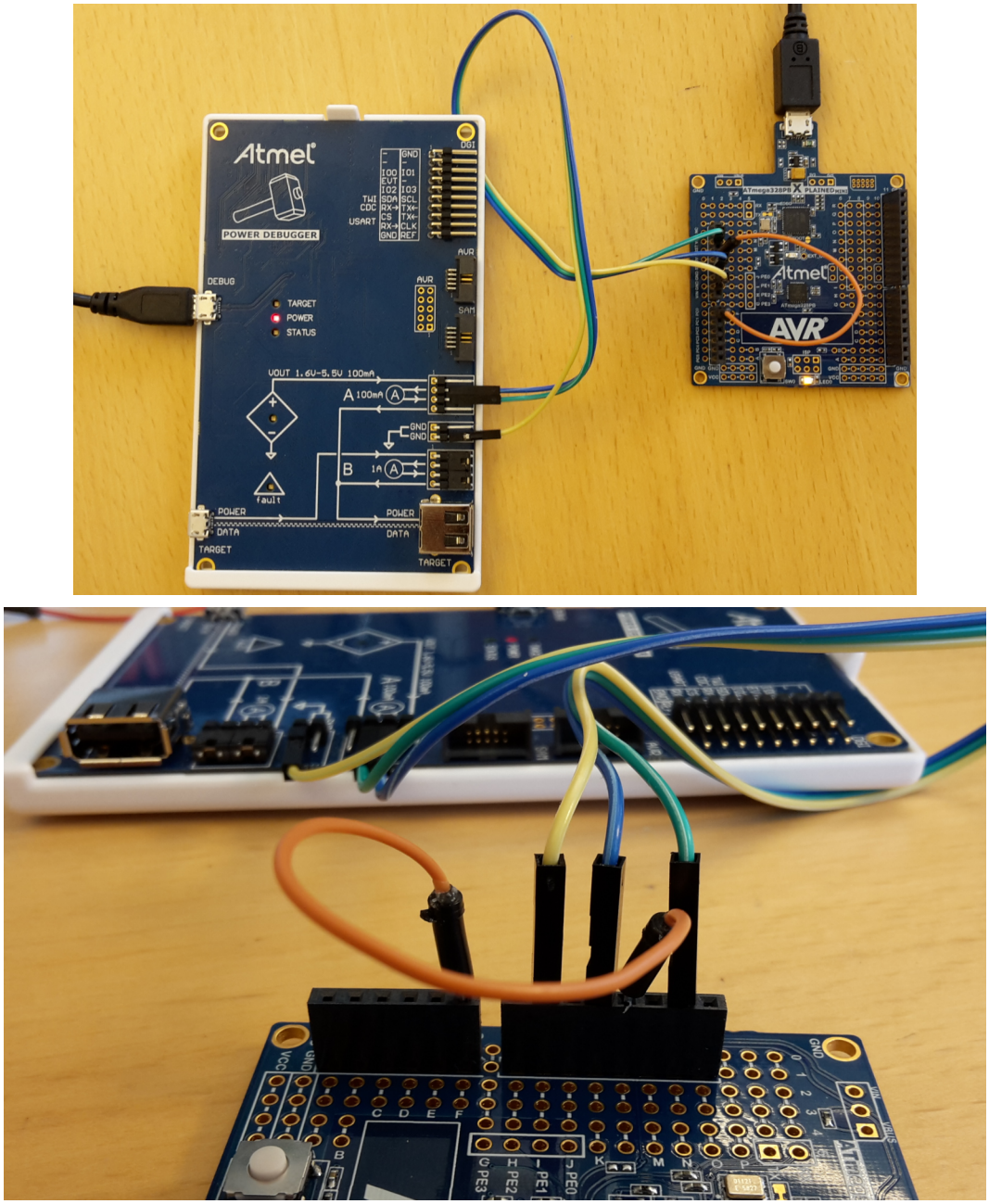
Figure 5-1. Power Debugger and ATmega328PB Xplained Mini Board Connections



To do: Connect the Power Debugger board to the PC using a Micro-USB cable on the Debug port. Also, connect the ATmega328PB Xplained Mini board to the PC using a Micro-USB cable.

Result: Visual representation of the HW setup is as below.

Figure 5-2. Visual Representation of HW Setup for Assignment 4



5.1 Measuring Power Consumption

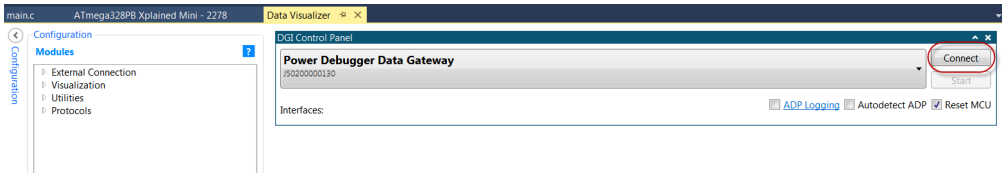
To do: Measure the power consumption for the application 'Read ADC' from Assignment 3.

1. Make sure that the debugging for the application 'Read ADC' from Assignment 3 is exited by selecting **Debug → Disable debugWire and Close**.

Info: If debugWIRE is not disabled, a programmed DWEN Fuse enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption while in sleep. Thus, the DWEN Fuse should be disabled when debugWIRE is not used.

2. Power up both boards; the ATmega328PB Xplained Mini and Power Debugger.
3. In Atmel Studio 7, open the menu **Tools** → **Data Visualizer**.
4. The **Power Debugger Data Gateway** should be selected by default in the **DGI Control Panel**. Select it if not. Click on the **Connect**.

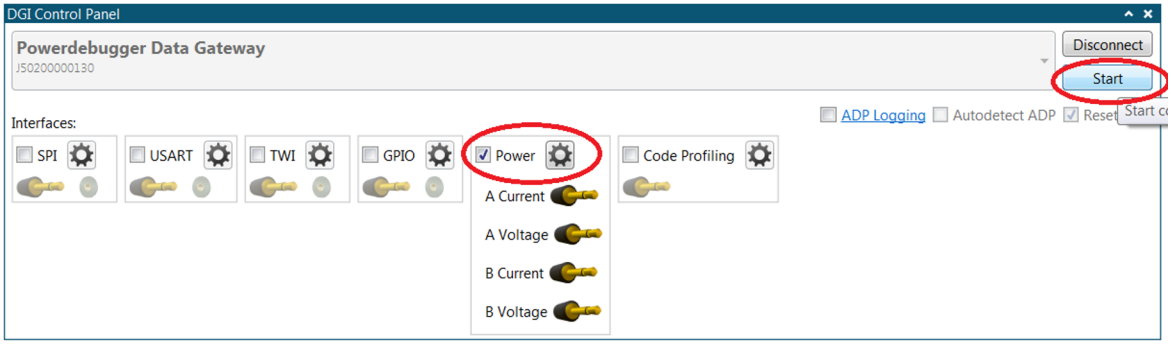
Figure 5-3. Data Visualizer



Info: If the **DGI Control panel** tab has not appeared, it can be opened from the **Configuration** tab at the left-hand side under **Modules** by selecting **External Connection** → **DGI Control panel**.

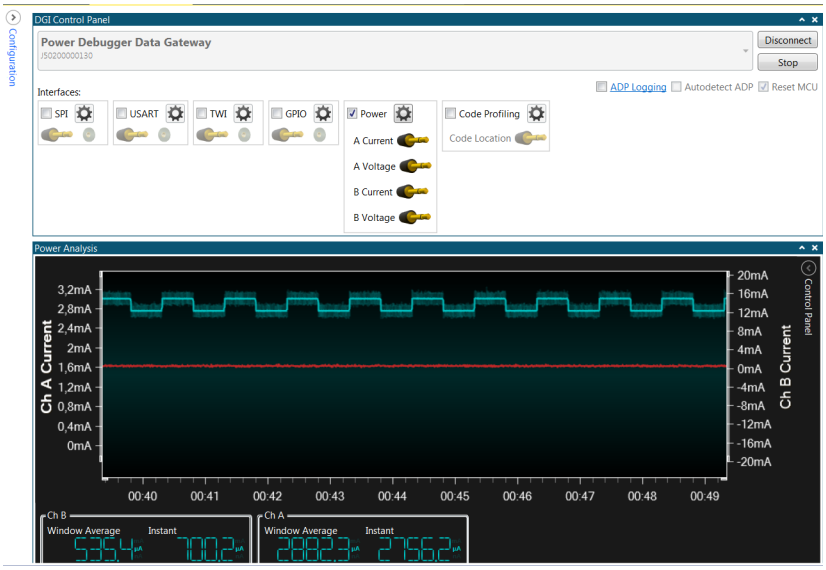
5. Select **Power** and then **Start**.

Figure 5-4. DGI Control Panel



Result: The Data Visualizer will now show real-time power analysis of the microcontroller (MCU), as shown below.

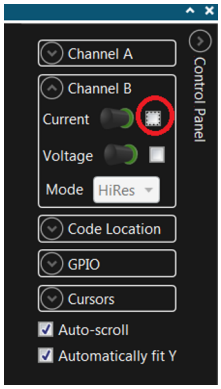
Figure 5-5. Power Analysis



Info: The red color graph is the B channel current measurement of the power debugger. We can disable it here. Channel B is additional current measurement channel with range up to 1A, but with a lower accuracy.

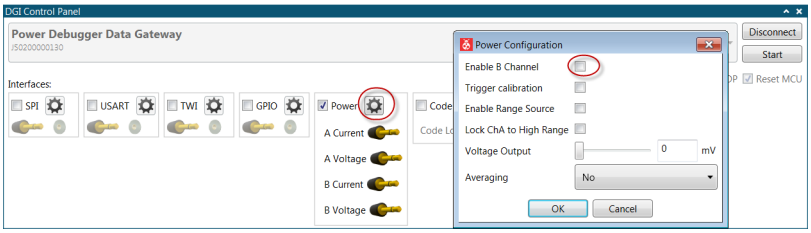
6. Open the **Control Panel** on the right-hand side of **Power Analysis** and uncheck the **Channel B Current**.

Figure 5-6. Control Panel



Info: Alternatively, it can be disabled in the configuration panel for the DGI before starting the measurement. Press the cogwheel next to the power interface and uncheck B channel from there. Press **OK**.

Figure 5-7. DGI Control Panel



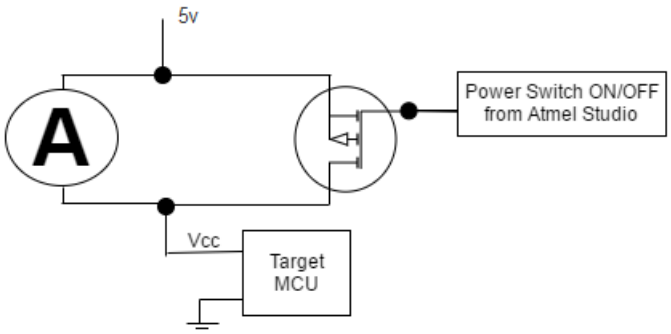
5.2 Enabling Current Measurement on the Xplained Mini Board

The mEDBG debugger on the Xplained Mini board has control of the target device's V_{CC} so that it can toggle its power.

By opening the switch on the Xplained Mini board the user can reroute power to the target device through an external current measuring probe. It enables to feed the target voltage externally and provide accurate power measurements.

Basically, the block diagram is as shown below.

Figure 5-8. Block Diagram Power Switch



By default, on power-up, the switch is closed (power is on).

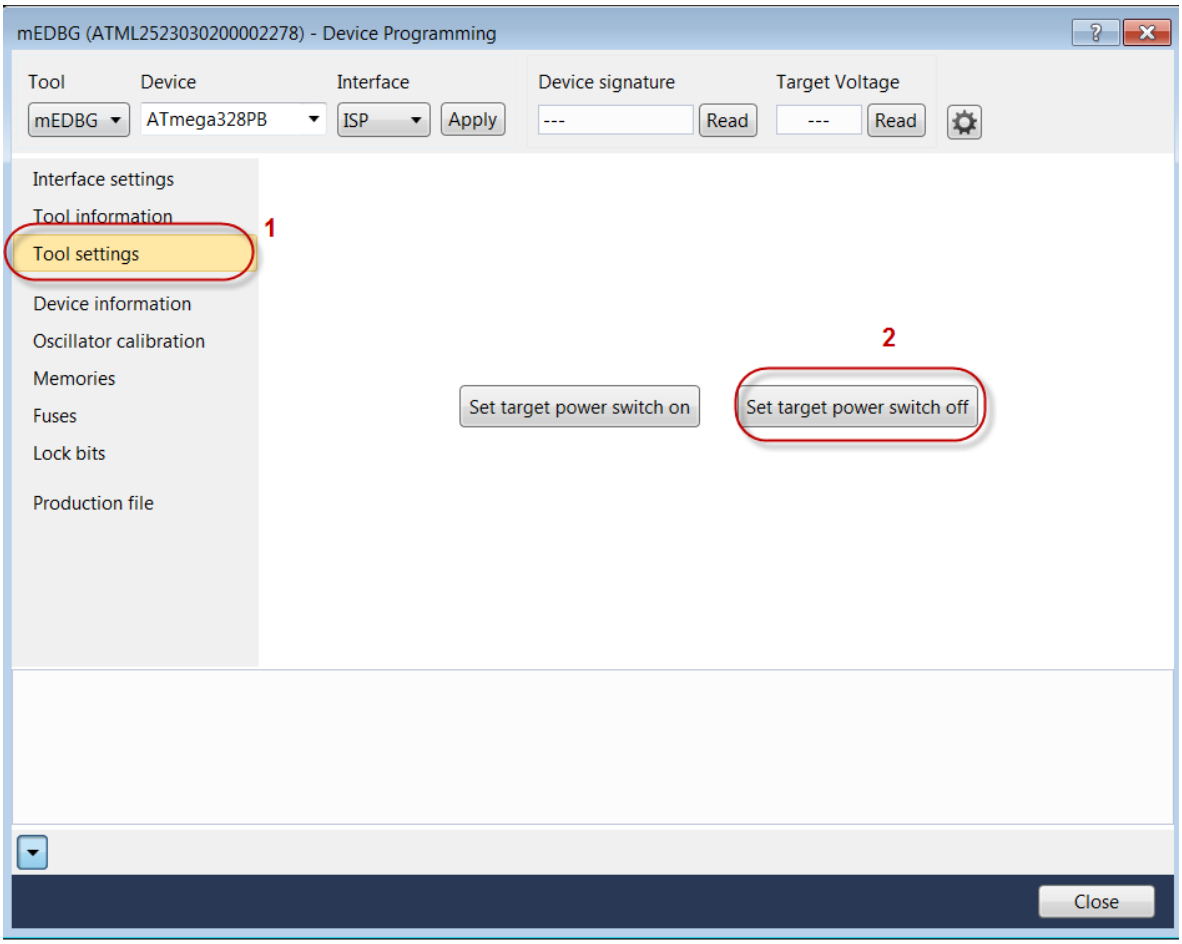
To do: Use Atmel Studio 7 to open the switch.

1. In Atmel Studio 7 select **Tools** → **Device Programming**.
2. Select **Tool**, **Device**, and **Interface** as shown below and click **Apply**.

Figure 5-9. Device Programming

3. Select **Tool settings** and click on **Set target power switch off**, as shown below.

Figure 5-10. Set Target Power Switch Off



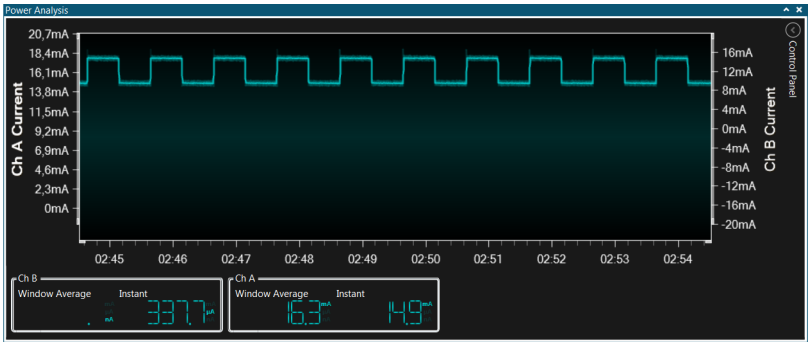
4. Close the **Device Programming** window.

Info: Note that this is required to run at every power-cycle of the board. (Run it every time the board has lost its power and is powered up again.)

Result: The Data Visualizer will show the Window Average and the instant power consumption for Channel A.

From the figure below it can be seen that the average power consumption is approximately 16.3 mA for the setup. The power consumption is high when LED0 is ON and power consumption is low when LED0 is OFF.

Info: Power consumption of the board may differ from board to board. It also depends upon the temperature.



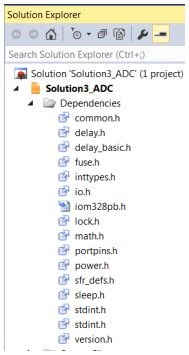
6. Assignment 5: Reducing the Power Consumption

There are several clever methods one can apply to reduce the power consumption on the AVR microcontroller. Some of the methods will be implemented here.

Power consumption optimization techniques will be implemented in the function `optimize_power_consumption()` and then the function `optimize_power_consumption()` will be called from `main()`.

Info: In this assignment it is required to configure some of the registers, which are defined in the header files: `#include <avr/power.h>` and `#include <avr/sleep.h>`. These files will be included from the Atmel Studio installation folder C:\Program Files (x86)\Atmel\Studio \7.0\toolchain\avr8\avr8-gnu-toolchain\avr\include\avr. Atmel Studio 7 project includes all of them and other dependency files. These are listed under **Dependencies** in **Solution Explorer**.

Figure 6-1. Solution Explorer



To do: Add function `void optimize_power_consumption()` before `main()`.

```
void optimize_power_consumption()
{

}
```

6.1 Disable Digital Input Buffers and Analog Comparator

For analog input pins, the digital input buffer should be disabled.

The digital input buffer measures the voltage on the pin to check if it should be interpreted as a logical one or zero. An analog signal level close to $V_{CC}/2$ on an input pin can cause significant additional current consumption, even in active mode. This is due to the frequent switching of the state of the pin between being a logical one and zero. If the pin is used as an analog pin, there is no need to know if the analog signal's digital value would be a one or a zero, and this conversion should, therefore, be disabled.

The only time digital input buffers enabled are needed, is when the pin is used as a digital pin. A digital input buffer on pins AIN0D and AIN1D also needs to be disabled. These are connected to an analog comparator.

Info: Digital input buffers can be disabled by writing to the Digital Input Disable Registers (DIDR1 and DIDR0).

To do: Add mentioned code described in the steps below in function `optimize_power_consumption`.

1. Disable the digital input buffer on the Analog to Digital Converter (ADC) pins.
   This is done by configuring the corresponding bits in the DIDR0 register to logic 1.

```
/* Disable digital input buffer on ADC pins */
DIDR0 = (1 << ADC5D) | (1 << ADC4D) | (1 << ADC3D) | (1 << ADC2D) | (1 << ADC1D) | (1 << ADC0D);
DIDR0 |= 0xC0 ;   /*ADC7D and ADC6D are undefined in header file so set bits this way*/
```

2. Disable digital input buffer on AIN0D and AIN1D pins. This is done by configuring the corresponding bits to logic 1 in the DIDR1 register.

```
/* Disable digital input buffer on Analog comparator pins */
DIDR1 |= (1 << AIN1D) | (1 << AIN0D);
```

3. Disable the analog comparator. This is done by writing 1 to the ACD bit of the ACSR register.

```
/* Disable Analog Comparator */
ACSR |= (1 << ACD);
```

6.2 Turning Off Unused Peripherals

The next step is to take a look at the peripherals, and disable those which are not used in the application.

The Power Reduction Register (PRR0) and (PRR1) provides a method to stop the clock to individual peripherals to reduce the power consumption. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral when stopping the clock will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock.

Waking up a module, which is done by clearing the bit in PRR, puts the module in the same state as before shutdown. Module shutdown can be used in Idle mode and Active mode to significantly reduce the overall power consumption.

In all other sleep modes, the clock is already stopped.

Below is the image from the datasheet for PRR0. Writing a logic one to this bit shuts down the respective peripheral by stopping the clock to the module.

14.11.3. Power Reduction Register 0

Name: PRR0
Offset: 0x64
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	PRTWI0	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI0	PRUSART0	PRADC
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

To do: Add the mentioned code described in the steps below.

1. Add the following file inclusion at the top of the file. The header file from the AVR library, <avr/power.h> gives us easy access to disabling and enabling peripherals.

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

2. Add the below code in `optimize_power_consumption ()`.

Info: In this application, the ADC module is used. The clock for the rest of the modules can be disabled to reduce the power consumption.

```
/*Power shutdown to unused peripherals*/
PRR0 = 0xFE;
PRR1 = 0x3F;
```

3. Add the following file inclusion at the top of the file. The header file from the AVR library, `<avr/wdt.h>`, gives easy access to functions related to WDT.

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

4. Turning off the Watchdog Timer:

Info: If the watchdog timer is not needed in the application, the module should be turned off. If the watchdog timer is enabled, it will be enabled in all sleep modes and hence always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

Add the following code in function `optimize_power_consumption()` to turn off the Watchdog Timer.

```
    /*Watchdog Timer OFF*/

    /* Disable interrupts */
    cli();
    /* Reset watchdog timer */
    wdt_reset();
    /* Clear WDRF in MCUSR */
    MCUSR &= ~(1<<WDRF);
    /* Turn off WDT */
     WDTCSR = 0x00;
```

6.3 Applying Pull-Up Resistors

One thing to notice is that when the pins of the microcontroller are not in use and not connected to anything, they are floating.

This means that the pins are not set to a specific voltage, and if one measures the voltage over the pins, it can vary a lot.

This consumes a lot of extra power that can be avoided by setting all unused pins as inputs with pull-up resistors. This will keep the power consumption to a minimum.

Port pin can be set to input pull-up by setting DDxn bit to 0 and PORTxn bit to 1. (x is PORT B,C,D,E, and n is 0 to 7.)

Info: Pins are needed to be set as pull-up only when they are floating pins. On the ATmega328P Xplained Mini board some of the pins are connected, e.g. PTC pins, UART, SPI, LED0, and SW0 pins. For more information, refer to schematic 'ATmega328PB_Xplained_Mini_design_Documentation'.
Pins are needed to be configured with direction input and pull-up resistor activated when they are floating pins. On the ATmega328P Xplained Mini board some of the pins are connected, e.g. PTC pins, UART, SPI, LED0, and SW0 pins. For more information, refer to schematic 'ATmega328PB_Xplained_Mini_design_Documentation'. The other pins are configured with direction input and pull-up resistor activated

.

To do: Add the below code in `optimize_power_consumption ()` to configure unused port pins as input with pull-up resistors.

```
/*Unused pins set as input pull up*/

 DDRB &= 0xE0;
 DDRC &= 0xC9;
 DDRD &= 0x03;
 DDRE &=0xF3;

 PORTB |=~(0xE0);
 PORTC |=~(0xC9);
 PORTD |=~(0x03);
 PORTE |=~(0xf3);
```

6.4 Use Sleep Function

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

To do: Add the code below.

1. Include the <avr/sleep.h> header file from the AVR library at the top of the file.

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

2. Set the desired sleep mode in the `optimize_power_consumption ()` function by configuring the microcontroller to use the sleep mode SLEEP_MODE_PWR_DOWN for minimum power consumption.

   ```
   /* Set sleep mode */
   set_sleep_mode(SLEEP_MODE_PWR_DOWN);
   ```

3. Now, call the function `sleep_mode()` in `main()`, in `while(1)` so that the microcontroller enters sleep. To minimize the power consumption, disable the ADC and shut down the power to the ADC before entering sleep and after wake-up from sleep, enable the module again.
   Add the code below in `while(1)`

```
ADCSRA &= ~(1 << ADEN);     //disable ADC
power_adc_disable();        //shutdown power to ADC
sleep_mode();
ADCSRA |= (1 << ADEN);     //after wake up enable ADC
power_adc_enable();        //enable the module
```

4. Comment out the delay function and LED toggle. The code should look as shown below.

Figure 6-2. Code Block: Sleep mode

```
while (1)
{
    ADCSRA &= ~(1 << ADEN);
    power_adc_disable();
    sleep_mode();
    ADCSRA |= (1 << ADEN);
    power_adc_enable();

//  _delay_ms(500);
//  PINB |= 1<<PINB5 ;  // toggle LED
    adc_res=ReadADC(0)  ;
}
```

5. And finally, call the `optimize_power_consumption ()` function before the `while(1)` loop.

```
optimize_power_consumption();
```

6.5 Using Pin Change Interrupt

To wake up the microcontroller from SLEEP_MODE_PWR_DOWN sleep mode, Pin Change interrupt is used.

On the ATmega328PB Xplained Mini board, the switch (SW0) is connected to PB7. Pin PB7 will be used as the source for the pin change interrupt.

1. File inclusion to access interrupt related functions:

To do: Include the <avr/interrupt.h> header file from the AVR library at the top of the file.

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

2. Configuring bit PCINT7 and PCICR register:
   The PCINT7 bit is part of the PCMSK0 (Pin change Mask Register 0) register.

Info: In the data sheet, the section "***Alternate Functions of Port B***", describes the alternate functions of PORTB. Here it is mentioned that PB7 has the alternate function of “Pin change interrupt 7”, as shown in the following figure.

Figure 6-3. Data Sheet: PORT B Pins Alternate Functions

Table 18-3 Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	XTAL2 (Chip Clock Oscillator pin 2) TOSC2 (Timer Oscillator pin 2) PCINT7 (Pin Change Interrupt 7)

Also, the pin change interrupt is enabled on the corresponding I/O pin if PCICR (Pin Change Interrupt Control Register) is configured accordingly.

To do: Enable the PCINT7 by including the following lines in the code in `main()` before `while(1)` and enable the global interrupt.

```
PCMSK0 |= (1<<PCINT7); //Enable Pin Change Interrupt 7
PCICR |= (1<<PCIE0); //Enable the interrupt enable bit for PCINT
sei();
```

Info: Each PCINT[7:0] bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] is set and the PCIE0 bit in PCICR is set, the pin change interrupt is enabled on the corresponding I/O pin.

3. Interrupt service routine:
In order to enable the ISR routines, the vector name for the PCINT7 is ISR (PCINT0_vect), defined in the device header file.

To do: Add the code below before the `main()` to turn LED0 OFF and ON with switch (SW0) pressed and released.

```
ISR (PCINT0_vect)
{
    if (!(PINB & (1<<PINB7))) // if PINB7 is low (Switch pressed)
    {
        PORTB |= (1<<PORTB5); // Turn ON LED
    }
    else
    {
        PORTB &= ~(1<<PORTB5); // Turn OFF LED
    }
}
```

Result: Implementing the different low-power techniques in the application is completed. Now the code should look as shown below.

Figure 6-4. Code: Assignment 5

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/sleep.h>
#include <avr/power.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>

uint16_t adc_res=0;
void InitADC()
{
    // Select Vref=AVcc
    ADMUX |= (1<<REFS0);
    //set prescaler to 128 and enable ADC
    ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADEN);
}
uint16_t ReadADC(uint8_t ADCchannel)
{
    //select ADC channel with safety mask
    ADMUX = (ADMUX & 0xF0) | (ADCchannel & 0x0F);
    //single conversion mode
    ADCSRA |= (1<<ADSC);
    // wait until ADC conversion is complete
    while( ADCSRA & (1<<ADSC) );
    return ADC;
}
```

```
void optimize_power_consumption()
{
    /* Disable digital input buffer on ADC pins */
    DIDR0 = (1 << ADC5D) | (1 << ADC4D) | (1 << ADC3D) | (1 << ADC2D) | (1 << ADC1D) | (1 << ADC0D);
    DIDR0 |= 0xC0 ;   /*ADC7D and ADC6D are undefined in header file so set bits this way*/

    /* Disable digital input buffer on Analog comparator pins */
    DIDR1 |= (1 << AIN1D) | (1 << AIN0D);
    /* Disable Analog Comparator */
    ACSR |= (1 << ACD);

    /*Power shutdown to unused peripherals*/
    PRR0 = 0xFE;
    PRR1 = 0x3F;
    /*Unused pins set as input pull up*/
    DDRB &= 0xE0;
    DDRC &= 0xC9;
    DDRD &= 0x03;
    DDRE &= 0xF3;

    PORTB |=~(0xE0);
    PORTC |=~(0xC9);
    PORTD |=~(0x03);
    PORTE |=~(0xf3);

    /*Watchdog Timer OFF*/

    /* Disable interrupts */
    cli();
    /* Reset watchdog timer */
    wdt_reset();
    /* Clear WDRF in MCUSR */
    MCUSR &= ~(1<<WDRF);
    /* Turn off WDT */
    WDTCSR = 0x00;
    /* Set sleep mode */
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
}
```

```
ISR (PCINT0_vect)
{
    if (!(PINB & (1<<PINB7))) // if PINB7 is low (Switch pressed)
    {
        PORTB |= (1<<PORTB5); // Turn ON LED
    }
    else
    {
        PORTB &= ~(1<<PORTB5); // Turn OFF LED
    }
}
int main(void)
{
    DDRB &= ~(1<<DDB7); //Set PORTB7 as input
    DDRC &= ~(1<<DDC0); //Set PORTC0 as input
    DDRB |= (1<<DDB5);  //Set PORTB5 as output

    InitADC();
    optimize_power_consumption();

    PCMSK0 |= (1<<PCINT7); //Enable Pin Change Interrupt 7
    PCICR |= (1<<PCIE0); //Enable the interrupt enable bit for PCINT
    sei();

    while (1)
    {
        ADCSRA &= ~(1 << ADEN);
        power_adc_disable();
        sleep_mode();
        ADCSRA |= (1 << ADEN);
        power_adc_enable();

    //  _delay_ms(500);
    //  PINB |= 1<<PINB5 ;  // toggle LED
        adc_res=ReadADC(0)  ;
    }
}
```

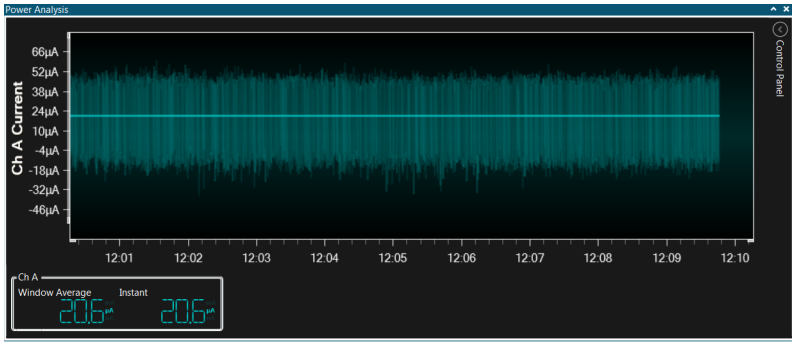
6.6 Program and Measure Power Consumption

To do: Program the implemented code from Assignment 5 and measure the power consumption.

1. Program the code by selecting **Debug** → **Continue**.
2. Wait till the application gets programmed. The message at the bottom of the window appears as **Running**.
3. Exit the debugging by selecting **Debug** → **Disable debugWire and Close**.

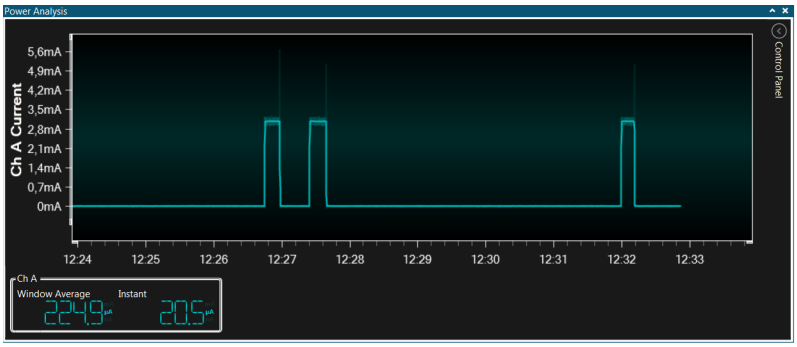
4. Open the window **Data Visualizer** and check the power consumption as shown below.

Figure 6-5. Power Analysis



5. Press the SW0 switch on the ATmega328PB board and observe that the device will wake up, power consumption will be increased, and again the device will enter sleep().

Figure 6-6. Power Analysis



Info: It is required to enable current measurement on the ATmega328P Xplained Mini if the board has lost its power and is powered up again; look at section Enabling Current Measurement on the Xplained Mini Board.

Result: After applying the different low-power techniques, it can be seen that the power consumption of the application has been significantly reduced.

7. Conclusion

This training demonstrated many different techniques, which can be applied to an application to lower the power consumption. The power consumption can be significantly reduced by:

- Intelligent design
- Using sleep modes
- Switching off unused peripherals

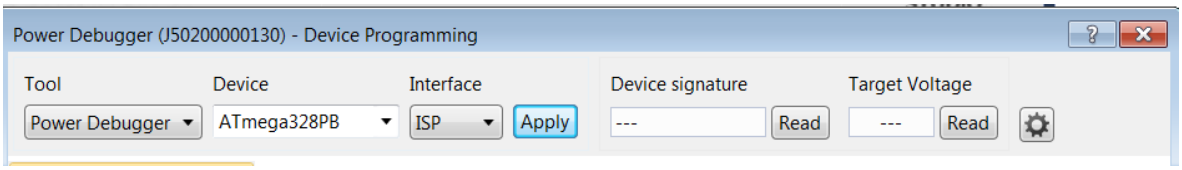
With Atmel Studio 7, it is easier to run real-time debugging of an application and use I/O view, which provides registers view and allows to modify the microcontroller's registers in real-time. It is possible to debug the application using various debugging methods such as:

- Switching off unneeded peripherals
- Breakpoints
- Single Stepping
- I/O view

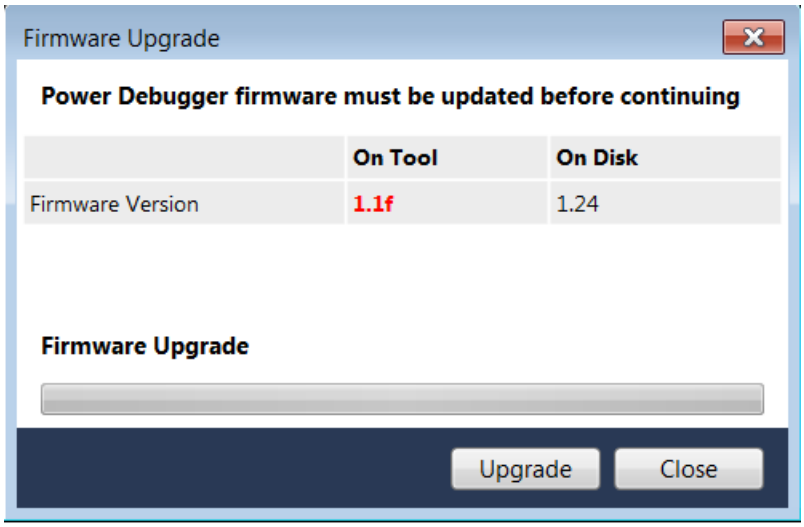
With the new feature of Atmel Studio 7: Data Visualizer and using Power debugger board, it is very easy to measure the power consumption of the application.

8. Appendix A: Firmware Upgrade on Power Debugger Board

- Connect the Power Debugger board to the PC by using a Micro-USB cable on the **DEBUG** port on the board
- In Atmel Studio 7, from the menu, select **Tools** → **Device Programming**
- From the **Tool** tab, select **Power Debugger**, then select **Device** as **ATmega328PB**, and click **Apply**



Info: The user will be asked to upgrade the firmware if it is not the latest.



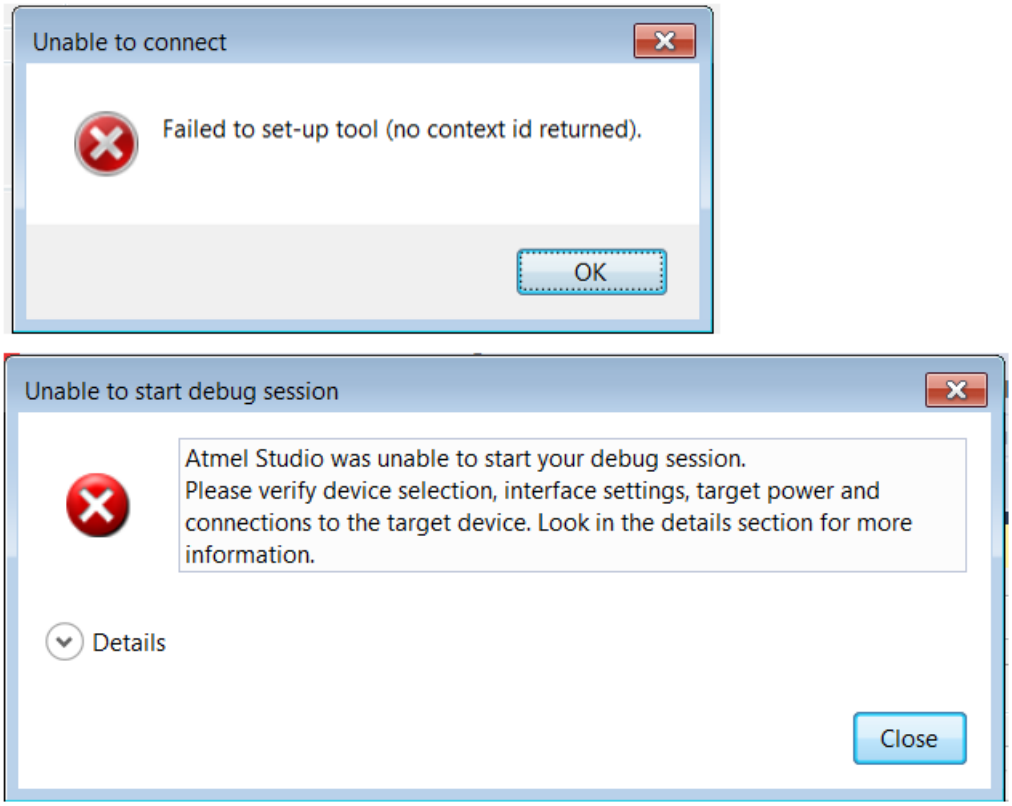
- Select **Upgrade**. Once the progress bar is completed, click **Close**. Also, close the **Device Programming** window.

Result: The power debugger firmware upgrade is successful.

9. Appendix B: Troubleshooting Guide

1. **Error message: Unable to Connect**. If the connected tool is not recognized by Atmel Studio, the following error message appears.

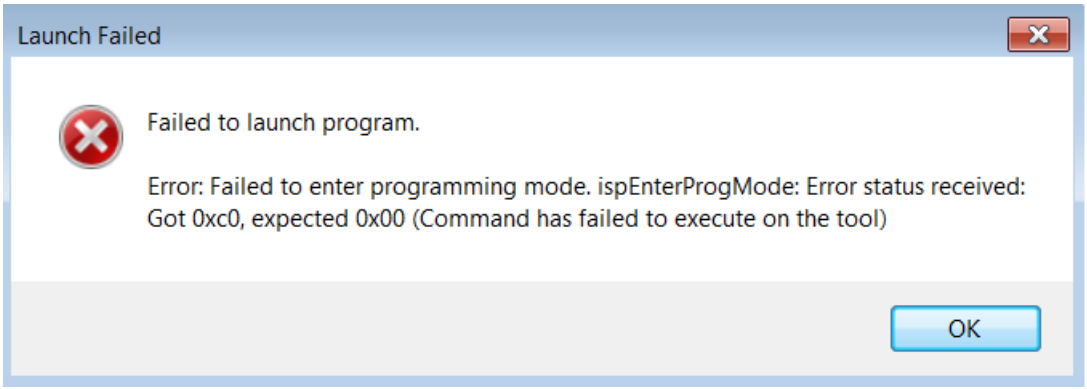
   Figure 9-1. Unable to Connect



   Workaround: Disconnect and re-connect the USB cable from Xplained Mini board and try to program again.

2. **Error message: Launch Failed.**
   The following error message is displayed if the DWEN FUSE is not disabled. That means the Xplained Mini board is left in 'debugWIRE' mode (programming/debugging interface as 'debugWIRE') and if an attempt is made to program the board using 'ISP' interface.

   Figure 9-2. Launch Failed



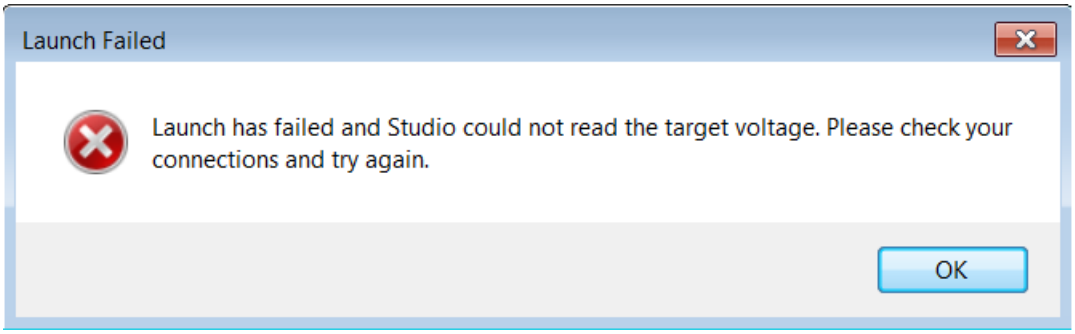
Info: Refer Figure 2-5 and check the **Interface**.

Workaround: Open the project. Check the interface. Change it to **DebugWIRE** and do programming by selecting **Debug->Start Debugging and break** or **Debug->Continue**. Don't forget to exit debugwire by selecting **Debug -> Disable debugWIRE and Close**.

Tip: Above Workaround is also applicable for error message displayed in Figure 2-12.

3. **Error message: Launch Failed.**
The following error message is displayed if Atmel Studio could not read the target voltage.

Figure 9-3. Launch Failed



Workaround: Disconnect and re-connect the USB cable from Xplained Mini board and try to program again.

10. Revision History

Doc Rev.	Date	Comments
A	09/2017	Converted to Microchip format and replaced the Atmel document number 42800A.
42800A	10/2016	Initial document release

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-2161-0

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Asia Pacific Office** Suites 3707-14, 37th Floor Tower 6, The Gateway Harbour City, Kowloon **Hong Kong** Tel: 852-2943-5100 Fax: 852-2401-3431 **Australia - Sydney** Tel: 61-2-9868-6733 Fax: 61-2-9868-6755 **China - Beijing** Tel: 86-10-8569-7000 Fax: 86-10-8528-2104 **China - Chengdu** Tel: 86-28-8665-5511 Fax: 86-28-8665-7889 **China - Chongqing** Tel: 86-23-8980-9588 Fax: 86-23-8980-9500 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 Fax: 86-571-8792-8116 **China - Hong Kong SAR** Tel: 852-2943-5100 Fax: 852-2401-3431 **China - Nanjing** Tel: 86-25-8473-2460 Fax: 86-25-8473-2470 **China - Qingdao** Tel: 86-532-8502-7355 Fax: 86-532-8502-7205 **China - Shanghai** Tel: 86-21-3326-8000 Fax: 86-21-3326-8021 **China - Shenyang** Tel: 86-24-2334-2829 Fax: 86-24-2334-2393 **China - Shenzhen** Tel: 86-755-8864-2200 Fax: 86-755-8203-1760 **China - Wuhan** Tel: 86-27-5980-5300 Fax: 86-27-5980-5118 **China - Xian** Tel: 86-29-8833-7252 Fax: 86-29-8833-7256	**China - Xiamen** Tel: 86-592-2388138 Fax: 86-592-2388130 **China - Zhuhai** Tel: 86-756-3210040 Fax: 86-756-3210049 **India - Bangalore** Tel: 91-80-3090-4444 Fax: 91-80-3090-4123 **India - New Delhi** Tel: 91-11-4160-8631 Fax: 91-11-4160-8632 **India - Pune** Tel: 91-20-3019-1500 **Japan - Osaka** Tel: 81-6-6152-7160 Fax: 81-6-6152-9310 **Japan - Tokyo** Tel: 81-3-6880- 3770 Fax: 81-3-6880-3771 **Korea - Daegu** Tel: 82-53-744-4301 Fax: 82-53-744-4302 **Korea - Seoul** Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934 **Malaysia - Kuala Lumpur** Tel: 60-3-6201-9857 Fax: 60-3-6201-9859 **Malaysia - Penang** Tel: 60-4-227-8870 Fax: 60-4-227-4068 **Philippines - Manila** Tel: 63-2-634-9065 Fax: 63-2-634-9069 **Singapore** Tel: 65-6334-8870 Fax: 65-6334-8850 **Taiwan - Hsin Chu** Tel: 886-3-5778-366 Fax: 886-3-5770-955 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 Fax: 886-2-2508-0102 **Thailand - Bangkok** Tel: 66-2-694-1351 Fax: 66-2-694-1350	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **France - Saint Cloud** Tel: 33-1-30-60-70-00 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820