DAQ-1000 Data Acquisition and Control System Application

Based on the DAQ-1000 Arduino UNO Data Acquisition shield

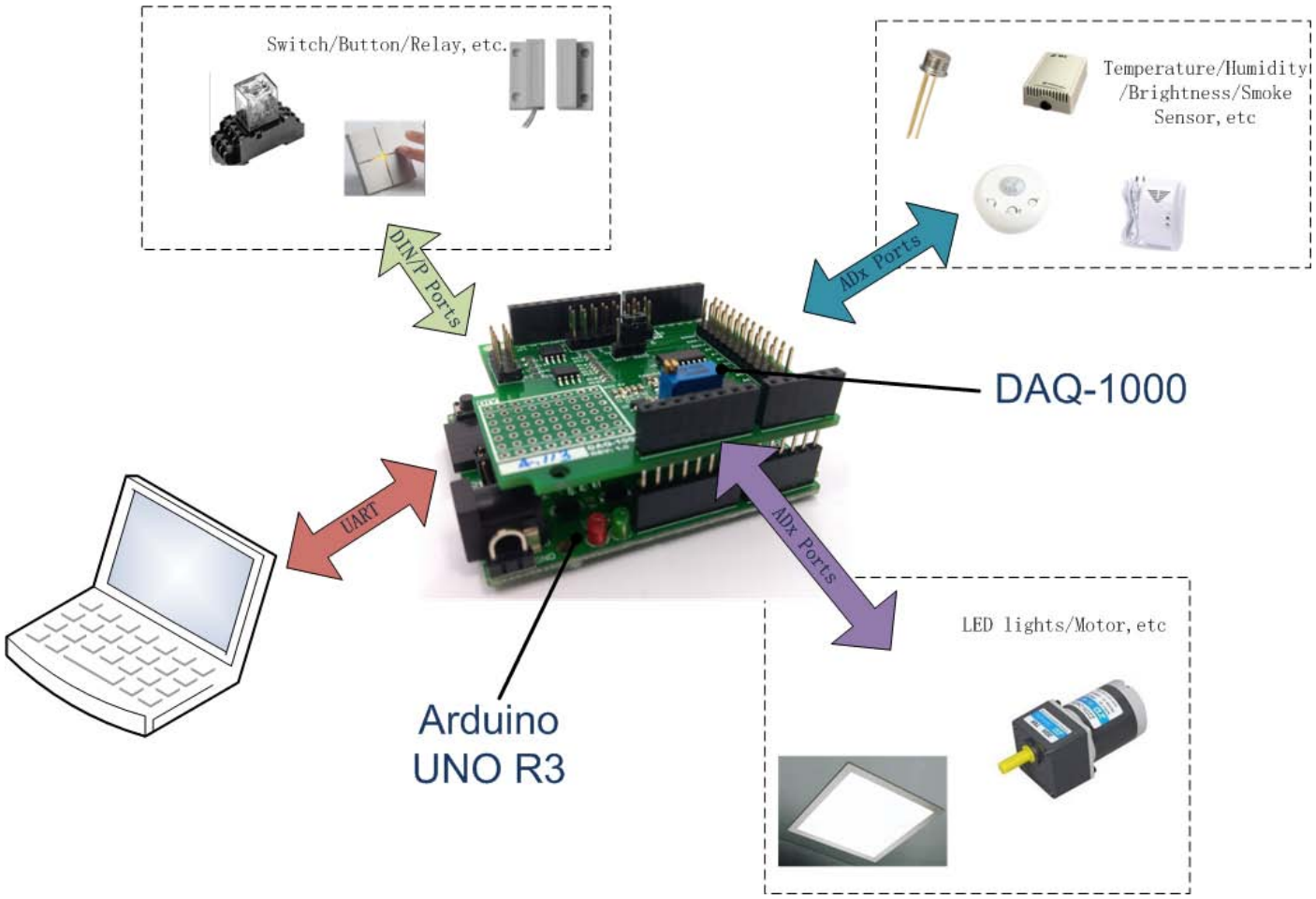
Tony Tan

2015/11/10

1. Summary

This document gives an example of how to build a data acquisition and control system using the DAQ-1000. It Mainly involves the following contents: The use of PC software, the protocol with arduino board, coding arduino and functions testing with peripherals sensors connected.
For more detail about the DAQ-1000, please see the datasheet: *DAQ-1000 Datasheet *pdf*, or download it from the url: http://www.inhaos.com/uploadfile/otherpic/DOC-DAQ-1000-V02-20151110.pdf

2. System structure



The system consists of PC, the Arduino UNO R3 motherboard, DAQ-1000 shield board and other peripheral devices.
The DAQ-1000 directly plugged-into Arduino motherboard, and then connected to PC via a USB cable. And other peripherals connected to DAQ-1000 according its pins.
Peripheral sensors is roughly divided into the following categories:

- Digital IO sensors, such as switches, buttons and relays, etc, which are controlled or collected through digital IO's HIGH/LOW state
- Analog Input sensors, such as Temperature Sensors, Humidity Sensors, Brightness Sensors, Smoke Sensors ,etc, those are all collected via analog quantity

- Analog Output sensors, such as LED lights, Motors, etc, which are controlled through the analog level.

By programming the arduino board following the protocol with PC software, user can achieve any functions they are want.

3. UART Data Communication Protocol

Before coding let's look at the protocol between PC and Arduino motherboard.

For the system every communication are initiated by the PC. The arduino board as the passive side make corresponding responses while requested by PC.

The data format using the ASCII coding and ending with '\r\n' (Hex value: 0x0d 0x0a). And the detail protocol list as following:

3.1. Get Device Info

PC REQ: ***IDN?**

ARDUINO RES:

DAQ-1000 (Success)
The DAQ-1000 device name

ERROR (Fail)

3.2. Read VREF

PC REQ: **REF?**

ARDUINO RES:

VREF_VAL (Success)
VREF_VAL is the voltage of VREF, value range: 0 to 40960, Unit:0.1mV, (e.g.) : 40851=4.0851V

ERROR (Fail)

3.3. Set VREF

PC REQ: **REF= VREF_VAL**
VREF_VAL is the voltage of VREF, value range: 0 to 40960, Unit:0.1mV, (e.g.) : 40851=4.0851V

ARDUINO RES:

TRUE (Success)

ERROR (Fail)

3.4. Read Comparator Bias

PC REQ: **CMP?**

ARDUINO RES:

	BIAS_VOL	(Success)
		BIAS_VOL is the c**omparator bias voltage, value** range: 0 to 40960, Unit:0.1mV, (e.g.) : 40851=4.0851V
	ERROR	(Fail)

3.5. Set Comparator Bias

PC REQ:	**CMP= BIAS_VOL**	
		BIAS_VOL is the c**omparator bias voltage, value** range: 0 to 40960, Unit:0.1mV, (e.g.) : 40851=4.0851V
ARDUINO RES:		
	TRUE	(Success)
	ERROR	(Fail)

3.6. Read Comparator Bias Enable

PC REQ:	**CMPEN**?	
ARDUINO RES:		
	1 or 0	(Success)
		1: Enabled
		0: Disabled
	ERROR	(Fail)

3.7. Set Comparator Bias Enable

PC REQ:	**CMPEN**= 1 or 0	
		1: Enabled
		0: Disabled
ARDUINO RES:		
	TRUE	(Success)
	ERROR	(Fail)

3.8. Set DAC CHx

PC REQ:	**DAx=VOL_VAL**	
		x select the DAC channel, value range: 0 to 2;
		VOL_VAL is the output level for the DAC channel, its value range: 0 to 40960, Unit:0.1mV,(e.g.):40851=4.0851V
ARDUINO RES:		
	TRUE	(Success)
	ERROR	(Fail)

3.9. Read ADC CHx

PC REQ: **ADx?**

x select the DAC channel, value range: 0 to 7

ARDUINO RES:

VOL_VAL (Success)

VOL_VAL is the input level for the ADC channel, its value range: 0 to 40960,

Unit:0.1mV,(e.g.):40851=4.0851V

ERROR (Fail)

3.10. Read DIN Port

PC REQ: **PI?**

ARDUINO RES:

PORT_STATE (Success)

PORT_STATE expresses all the DI pins state, value range: 0 to 255;

Each bit corresponding to a DI pin:

- Bit0: DI_CH0, (1=High,0=Low)
- Bit1: DI_CH1, (1=High,0=Low)
- Bit2: DI_CH2, (1=High,0=Low)
- Bit3: DI_CH3, (1=High,0=Low)

ERROR (Fail)

3.11. Read P Port

PC REQ: **P?**

ARDUINO RES:

PORT_STATE (Success)

PORT_STATE expresses all the pins state of P port, value range: 0 to 255;

Each bit corresponding to a pin of P port:

Bit0: Reserved

Bit1: Reserved

Bit2: D2 (1=High,0=Low)

Bit3: D8 (1=High,0=Low)

Bit4: MOSI (1=High,0=Low)

Bit5: MISO (1=High,0=Low)

Bit6: SCK (1=High,0=Low)

ERROR (Fail)

3.12. Read Px Port

PC REQ: **Px?**

x select the pin of P port, value range: 2 to 6:

0: Reserved
1: Reserved
2: D2
3: D8
4: MOSI
5: MISO
6: SCK

ARDUINO RES:

1 or 0 (Success)
1: HIGH level
0: LOW level

ERROR (Fail)

3.13. Set P Port

PC REQ: **P= PORT_STATE**

PORT_STATE expresses all the pins state of P port, value range: 0 to 255;
Each bit corresponding to a pin of P port:

Bit0: Reserved
Bit1: Reserved
Bit2: D2 (1=High,0=Low)
Bit3: D8 (1=High,0=Low)
Bit4: MOSI (1=High,0=Low)
Bit5: MISO (1=High,0=Low)
Bit6: SCK (1=High,0=Low)

ARDUINO RES:

TRUE (Success)
ERROR (Fail)

3.14. Set Px Port

PC REQ: **Px=1** (HIGH)
Px=0 (LOW)
x select the pin of P port, value range: 2 to 6:

0: Reserved
1: Reserved
2: D2
3: D8
4: MOSI
5: MISO
6: SCK

ARDUINO RES:

TRUE (Success)
ERROR (Fail)

3.15. Set P Port Mode

PC REQ: **PM= MODE_STATE**

MODE _STATE expresses all the pins mode of P port, value range: 0 to 255;
Each bit corresponding to a pin of P port:

- Bit0: Reserved
- Bit1: Reserved
- Bit2: D2 (1=Output,0=Input)
- Bit3: D8 (1= Output,0= Input)
- Bit4: MOSI (1= Output,0= Input)
- Bit5: MISO (1= Output,0= Input)
- Bit6: SCK (1= Output,0= Input)

ARDUINO RES:

TRUE (Success)
ERROR (Fail)

3.16. Set Px Mode

PC REQ: **PMx= 1** (OUTPUT)
PMx= 0 (INPUT)
x select the pin of P port, value range: 2 to 6:

- 0: Reserved
- 1: Reserved
- 2: D2
- 3: D8
- 4: MOSI
- 5: MISO
- 6: SCK

ARDUINO RES:

TRUE (Success)
ERROR (Fail)

3.17. Read P Port Mode

PC REQ: **PM?**

ARDUINO RES:

MODE_STATE (Success)
MODE _STATE expresses all the pins mode of P port, value range: 0 to 255;
Each bit corresponding to a pin of P port:

- Bit0: Reserved
- Bit1: Reserved
- Bit2: D2 (1=Output,0=Input)

Bit3: D8 (1= Output,0= Input)
Bit4: MOSI (1= Output,0= Input)
Bit5: MISO (1= Output,0= Input)
Bit6: SCK (1= Output,0= Input)

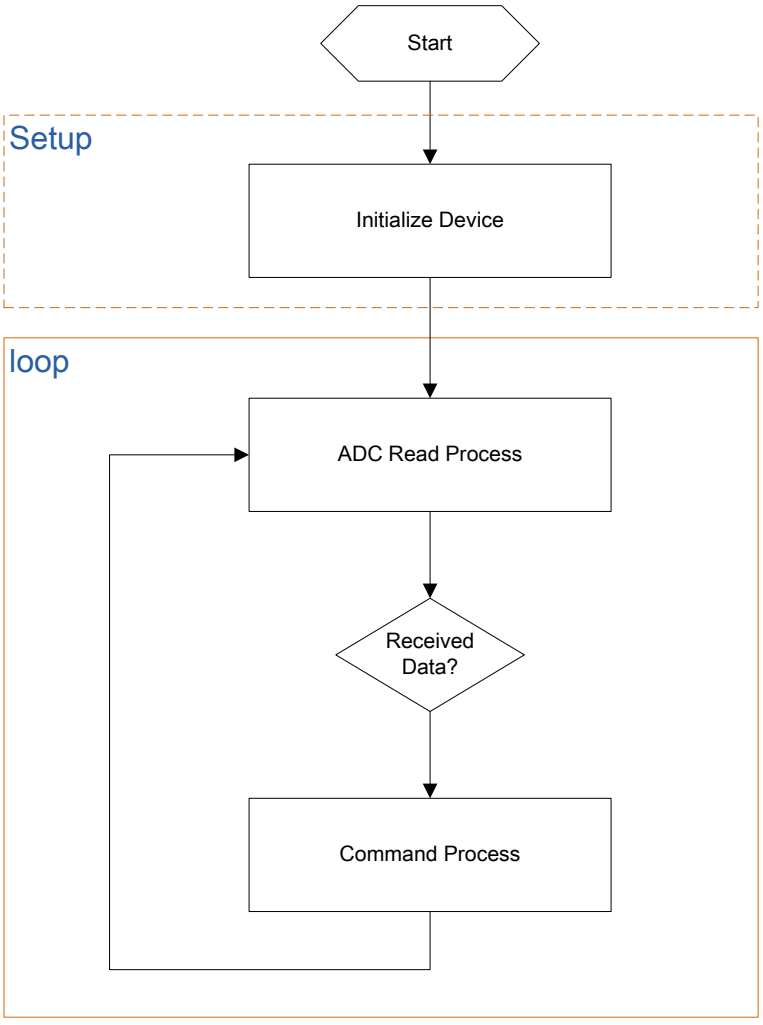
ERROR (Fail)

4. Arduino Code Descriptions

The arduino board mainly do those tasks:

- Continuous acquisition of external sensor latest status, using ADC
- Monitor and receive the request of the PC, through the serial port
- Parsing the PC request, and execute, then response results to PC

And the main routine flow chart as following:

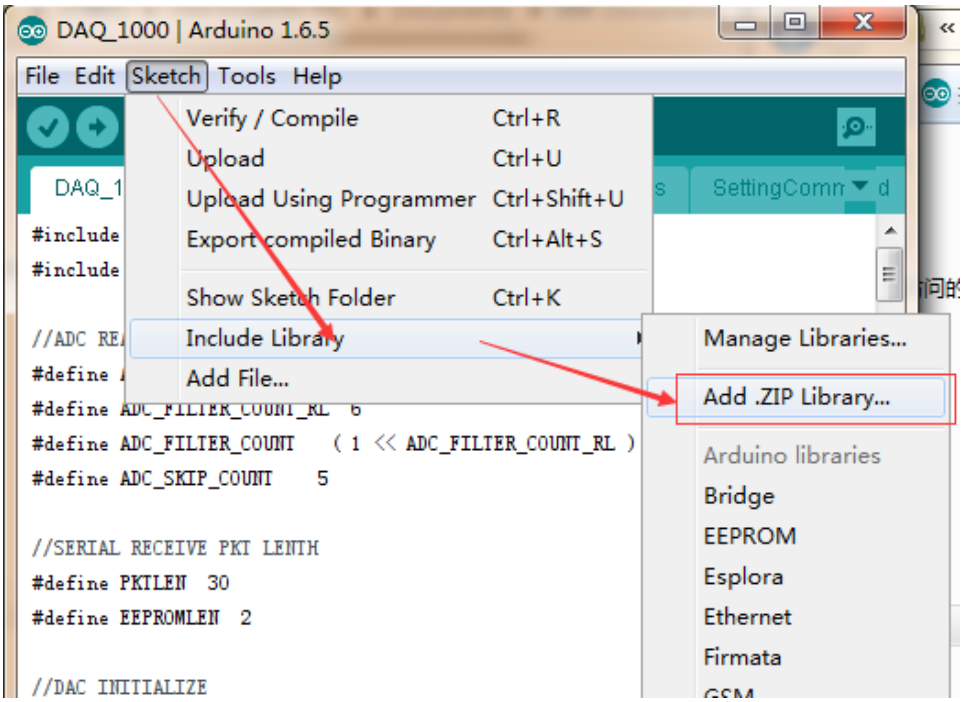


4.1. Library Preparations

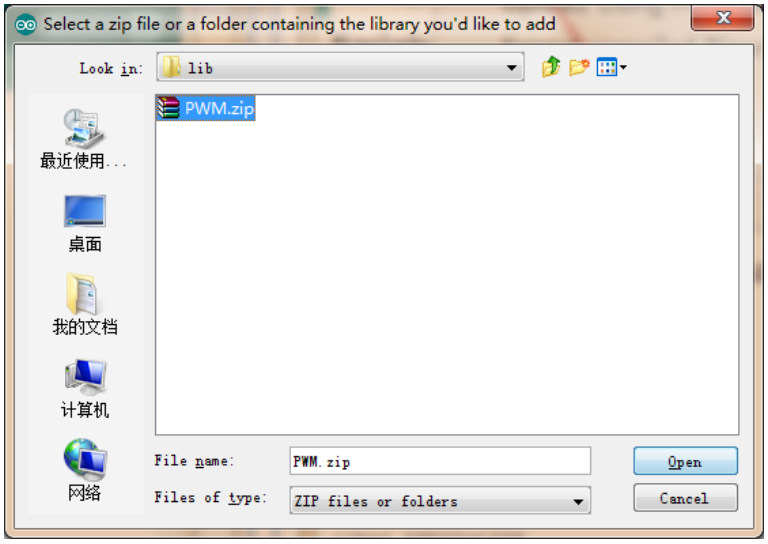
The PWM library needed in this project, which used for output high resolution PWM via DAC pins, and its resolution up to 16 bits.

Following the steps to add PWM library into this projects:

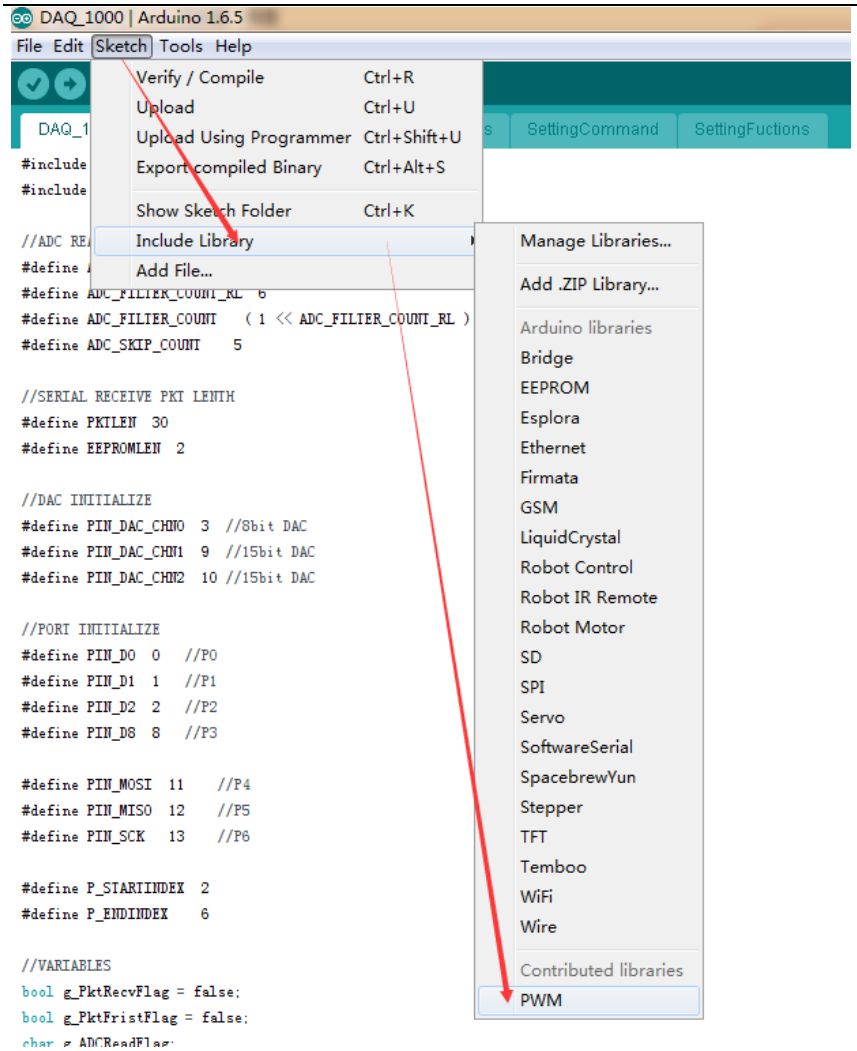
Click the 'Sketch'->'Include Library'->'Add .ZIP Library'



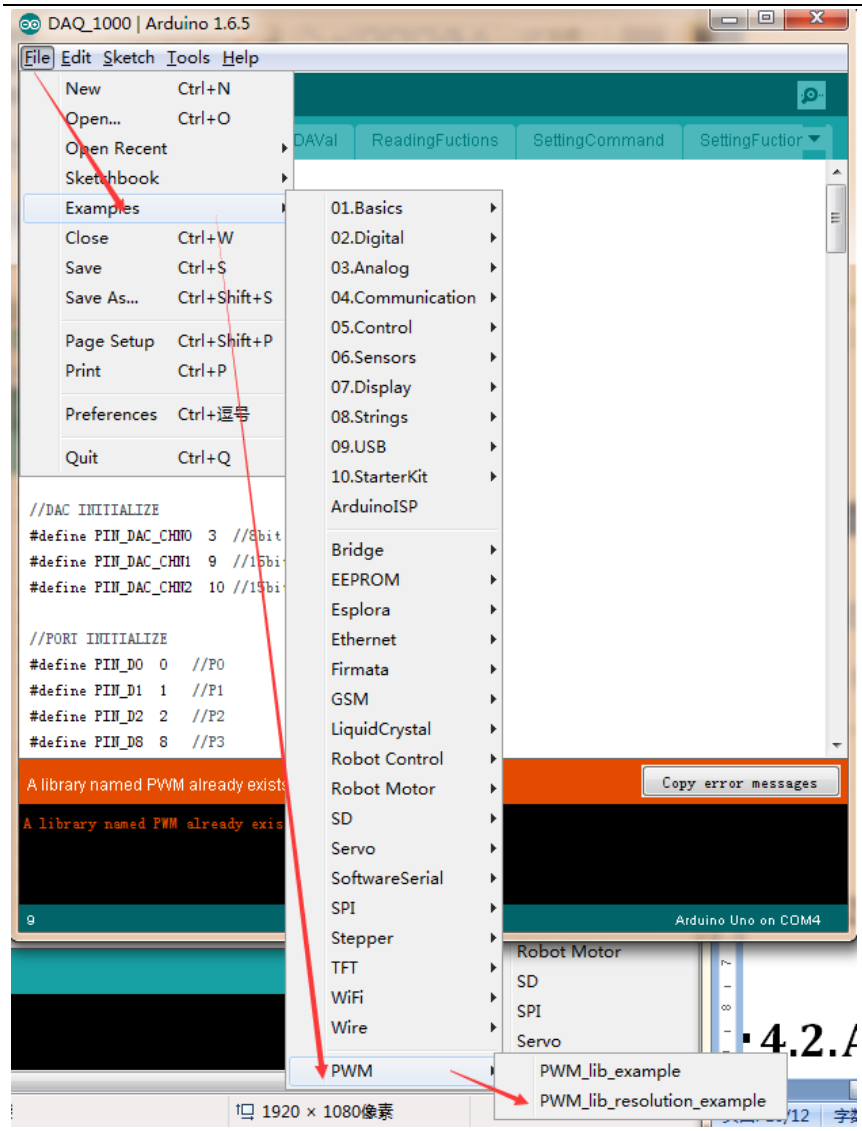
Select the PWM.zip under this project lib folder



Then include the PWM library into the project as follow:



So we can use the 16bits PWM in the project now. For more detail about the PWM library please see its demo project, by click 'File'->'Examples'->'PWM', then choose one of the example projects.



4.2. Setup Function

The setup() function mainly do the device initializations:

1) Initialize the P port.

   Set all pins of P port as input mode.

```
//Initilize P port mode
IOchn[0] = PIN_D0;
IOchn[1] = PIN_D1;
IOchn[2] = PIN_D2;
IOchn[3] = PIN_D8;
IOchn[4] = PIN_MOSI;
IOchn[5] = PIN_MISO;
IOchn[6] = PIN_SCK;
SetPortMode(0);
```

2) Initialize the DI ports

   Set DI_CH0 to DI_CH3 to INPUT mode

```
for ( char pinNum = 4; pinNum <= 7; pinNum++ ) {
```

```
  pinMode( pinNum, INPUT );
}
```

3) Initialize the AD ports

Set AD0 to AD7 as input mode, and disable the pull-up by write pin state to LOW

```
pinMode( A0, INPUT );
pinMode( A1, INPUT );
pinMode( A2, INPUT );
pinMode( A3, INPUT );
pinMode( A4, INPUT );
pinMode( A5, INPUT );
pinMode( A6, INPUT );
pinMode( A7, INPUT );
digitalWrite( A0, LOW );
digitalWrite( A1, LOW );
digitalWrite( A2, LOW );
digitalWrite( A3, LOW );
digitalWrite( A4, LOW );
digitalWrite( A5, LOW );
digitalWrite( A6, LOW );
digitalWrite( A7, LOW );
```

4) Initialize the DAC ports

Set DAC_CH0 to DAC_CH as OUTPUT mode, and HIGH level.

Initialize the PWM, for the DAC function used the PWM

```
 pinMode( PIN_DAC_CHN0, OUTPUT );
digitalWrite( PIN_DAC_CHN0, HIGH );
pinMode( PIN_DAC_CHN1, OUTPUT );
digitalWrite( PIN_DAC_CHN1, HIGH );
pinMode( PIN_DAC_CHN2, OUTPUT );
```

5) Initialize the PWM

Set the DAC_CH1 and DAC_CH2 's Frequency to 221 Hz and Number of Possible Duties is 36200, Resolution: 15 bit, for more detail, please see the PWM demo project. For the DAC_CH0 we only need 8bits resolution, so it doesn't be initialized here.

```
//initialize all timers except for 0, to save time keeping functions
InitTimersSafe();
//setting frequency
SetPinFrequency(PIN_DAC_CHN1, 221); //setting the frequency to 221 Hz
SetPinFrequency(PIN_DAC_CHN2, 221); //setting the frequency to 221 Hz
```

6) Initialize serial port

```
Serial.begin(115200);
```

7) Initialize ADC reference

```
analogReference(EXTERNAL);// use ecternal reference 4.096 V
```

4.3. loop function(main task)

As the flow chart shows, the loop function does two major tasks:

Read all ADC pin analog levels through call the function AdcReadProcess();

And invoke the PcRequestParsingProcess() after any request from PC received by serial port.

```
void loop() {
    // put your main code here, to run repeatedly:
    AdcReadProcess();//AD read process
    if (g_PktRecvFlag) {
        g_PktRecvFlag = false;
        g_PktFristFlag = false;
        PcRequestParsingProcess();
        g_RecvCount = 0;
    }
}
```

4.4. ADC Sample Task

The ADC sample task continuous make acquisition to ADC pin channels one by one. And for each channel, make multiple acquisitions and get the average value as its result. Then Call the CalcVoltageVal() function convert the average adc value to voltage. The formula for calculate the voltage from ADC value is: $\text{Voltage} = \frac{\text{adc}}{4096} \times \text{VREF}$, 4096 is the maximum value of 12bits number, and the VREF is the reference voltage value.

```
void AdcReadProcess() {
    //Read current ADC Channel, and add to the sum value
    g_ADCReadValSum += analogRead( g_AdcIndex + A0 );
    if ( ++g_ADCReadCount >= ADC_FILTER_COUNT ) {
        //Calc the average value
        g_ADCVoltage = g_ADCReadValSum / g_ADCReadCount;
        //Translate the ADC value to Voltage
        g_ADCVoltage = CalcVoltageVal( g_ADCVoltage );
        //Save the Voltage the g_ADCVal array
        g_ADCVal[g_AdcIndex] = g_ADCVoltage;
        //Clear the count and sum variable
        g_ADCReadCount = 0;
        g_ADCReadValSum = 0;
        //Loop the channel
        if (++g_AdcIndex > 7) {
            g_AdcIndex = 0;
        }
    }
}

//calculate voltage value
float CalcVoltageVal(float val) {
    float voltage;
    voltage = eepromvalue.g_Reference / 4096;
    voltage *= val;
    return voltage;
}
```

4.5. Serial Receive Task

In order to improve the efficiency of serial receive using interrupt function for the serial receiving process.

For arduino, the serial interrupt function is serialEvent(), so we just add the receiving data code into it. As we defined before, the serial request data end with '\r' and '\n', we can check them to decide whether the request data receive completed. Store the valid request data into a global buffer.

```
//serial peceive interrupt
```

```
void serialEvent() {
    while ( Serial.available() ) {  //Check serial data available
        char inChar = Serial.read();
        if ( inChar == '\r' ) {       //check whether the first end char
            g_PktFristFlag = true;
        } else if ( inChar == '\n' ) { //Check whether the second end char
            if ( g_PktFristFlag ) {     //If two end char received, then the request data valid
                g_RecvBuff[g_RecvCount] = '\0';
                g_PktRecvFlag = true;
            }
        } else {
            g_RecvBuff[g_RecvCount++] = inChar; //Put request data into buffer
        }
    }
}
```

4.6. Request Parsing & Process Task

The function PcRequestParsingProcess() parsing the request data, process, and then feedback the response.

For the sake of simplicity, we just compare the request data with each of strings in the request list we defined before. For example: we compare request data with "*IDN?" firstly, if is same then call ReadVer() function print back to PC the device name "DAQ-1000"; Otherwise, we compare it with "REF?" secondly, if is same then call ReadVoltageReference() function send back the voltage reference value; Otherwise, we compare it again with ...

For more detail, please reference the detail function codes.

```
void PcRequestParsingProcess() {
    char *pString = (char *)&g_RecvBuff[0];
    char *indexStr;
    char g_temp;
    Char_Convertion();

    //READ DEVICE Name
    if ( 0 == strncmp( pString, "*IDN?", 5 ) ) {
        if ( g_RecvCount == 5 ) {
            ReadVer();
        } else {
            Error();
        }
    }
    //READ REFERENCE VOLTAGE
    else if ( 0 == strncmp( pString, "REF?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadVoltageReference();
        } else {
            Error();
        }
    }
    //READ COMPARATOR OFFSET
    else if ( 0 == strncmp( pString, "CMP?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadCmpVal();
        } else {
            Error();
        }
    }
    //READ COMPARATOR OFFSET ENABLE
    else if ( 0 == strncmp( pString, "CMPEN?", 6 ) ) {
        if (  g_RecvCount == 6 ) {
            ReadCmpEnState();
        } else {
            Error();
        }
    }
    //READ AD0
    else if ( 0 == strncmp( pString, "AD0?", 4 ) ) {
```

```
        if ( g_RecvCount == 4 ) {
            ReadADVal( 0 );
        } else {
            Error();
        }
    }
    //READ AD1
    else if ( 0 == strncmp( pString, "AD1?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 1 );
        } else {
            Error();
        }
    }
    //READ ADC2
    else if ( 0 == strncmp( pString, "AD2?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 2 );
        } else {
            Error();
        }
    }
    //READ ADC3
    else if ( 0 == strncmp( pString, "AD3?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 3 );
        } else {
            Error();
        }
    }
    //READ ADC4
    else if ( 0 == strncmp( pString, "AD4?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 4 );
        } else {
            Error();
        }
    }
    //READ ADC5
    else if ( 0 == strncmp( pString, "AD5?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 5 );
        } else {
            Error();
        }
    }
    //READ ADC6
    else if ( 0 == strncmp( pString, "AD6?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 6 );
        } else {
            Error();
        }
    }
    //READ ADC7
    else if ( 0 == strncmp( pString, "AD7?", 4 ) ) {
        if ( g_RecvCount == 4 ) {
            ReadADVal( 7 );
        } else {
            Error();
        }
    }
    //READ DIN PORT STATE
    else if ( 0 == strncmp( pString, "PI?", 3 ) ) {
        if ( g_RecvCount == 3 ) {
            ReadPI();
        } else {
            Error();
        }
    }
    //READ Px PORT STATE
```

```
111     else if ( 0 == strncmp( pString, "P?", 2 ) ) {
112         if ( g_RecvCount == 2 ) {
113             ReadPortState();
114         } else {
115             Error();
116         }
117     }
118     //READ PORT MODE
119     else if ( 0 == strncmp( pString, "PM?", 3 ) ) {
120         if ( g_RecvCount == 3 ) {
121             ReadPinMode();
122         } else {
123             Error();
124         }
125     }
126     //SET REFERENCE VOLTAGE : REF = xxxxx
127     else if ( 0 == strncmp( pString, "REF=", 4 ) ) {
128         if ( g_RecvCount >= 5 &&
129                 ( g_RecvCount <= 9 ) ) {
130             sscanf((char *)&g_RecvBuff[4], "%d", &g_Para);  // 读取参数值
131             SettingReference( g_Para );
132             Right();//返回 RIGHT
133         } else {
134             Error();
135         }
136     }
137     //SET COMPARATOR OFFSET: CMP=xxxxx
138     else if ( 0 == strncmp( pString, "CMP=", 4 ) ) {
139         if ( ( g_RecvCount >= 5 ) &&
140                 ( g_RecvCount <= 9 ) ) {
141             sscanf((char *)&g_RecvBuff[4], "%d", &g_Para);
142             SettingCMP(g_Para);
143             Right();
144         } else {
145             Error();
146         }
147     }
148     //SET COMPARATOR OFFSET ENABLE: CMPEN=0/1
149     else if ( 0 == strncmp( pString, "CMPEN=", 6 ) ) {
150         if ( g_RecvCount == 7 ) {
151             if ( g_RecvBuff[6] == '1' ) {
152                 SettingCmpEnAble();
153                 Right();
154             } else if ( g_RecvBuff[6] == '0' ) {
155                 SettingCmpEnDisable();
156                 Right();
157             } else {
158                 Error();
159             }
160         } else {
161             Error();
162         }
163     }
164     //SET DAC CHN0
165     else if ( 0 == strncmp( pString, "DA0=", 4 ) ) {
166         if ( (g_RecvCount >= 5) &&
167                 (g_RecvCount <= 9) ) {
168             sscanf((char *)&g_RecvBuff[4], "%d", &g_Para);
169             SettingDAVal(PIN_DAC_CHN0, g_Para );
170             Right();
171         } else {
172             Error();
173         }
174     }
175     //SET DAC CHN1
176     else if ( 0 == strncmp( pString, "DA1=", 4 ) ) {
177         if ( (g_RecvCount >= 5) &&
178                 (g_RecvCount <= 9) ) {
179             sscanf((char *)&g_RecvBuff[4], "%d", &g_Para);
180             SettingDAVal( PIN_DAC_CHN1, g_Para );
181             Right();
```

```
        } else {
            Error();
        }
    }
    //SET DAC CHN2
    else if ( 0 == strncmp( pString, "DA2=", 4 ) ) {
        if ( (g_RecvCount >= 5) &&
                (g_RecvCount <= 9) ) {
            sscanf((char *)&g_RecvBuff[4], "%d", &g_Para);
            SettingDAVal( PIN_DAC_CHN2, g_Para );
            Right();
        } else {
            Error();
        }
    }
    //SET P PORT STATE
    else if ( 0 == strncmp( pString, "P=", 2 ) ) {
        if ( (g_RecvCount >= 3) &&
                (g_RecvCount <= 5) ) {
            sscanf( (char *)&g_RecvBuff[2], "%d", &g_Para);
            SettingPortState(g_Para);
            Right();
        } else {
            Error();
        }
    }
    //SET Px PORT STATE
    else if ( 0 == strncmp( pString, "P0=", 3 ) ) {
        if ( (g_RecvCount == 4) ) {
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[0] );
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[0] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "P1=", 3 ) ) {
        if ( g_RecvCount == 4 ) {
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[1] );
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[1] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "P2=", 3 ) ) {
        if ( g_RecvCount == 4 ) {
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[2] );
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[2] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "P3=", 3 ) ) {
        if ( g_RecvCount == 4 ) {
```

```
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[3] );
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[3] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "P4=", 3 ) ) {
        if ( g_RecvCount == 4 ) {
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[4]);
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[4] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "P5=", 3 ) ) {
        if ( g_RecvCount == 4 ) {
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[5] );
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[5] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "P6=", 3 ) ) {
        if ( g_RecvCount == 4 ) {
            if ( g_RecvBuff[3] == '1' ) {
                SetIOState( IOchn[6] );
                Right();
            } else if ( g_RecvBuff[3] == '0' ) {
                ClrIOState( IOchn[6] );
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    }
    //SET P PORT MODE
    else if ( 0 == strncmp( pString, "PM=", 3 ) ) {
        if ( (g_RecvCount >= 4) &&
                (g_RecvCount <= 6) ) {
            sscanf((char *)&g_RecvBuff[3], "%d", &g_Para);
            SetPortMode(g_Para);
            Right();
        } else {
            Error();
        }
    }
    //SET Px PORT MODE
    else if ( 0 == strncmp( pString, "PM0=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 0;
            if ( g_RecvBuff[4] == '1' ) {
```

```
                SetIOMode(IOchn[0]);
                Right();
            } else if ( g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[0]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "PM1=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 1;
            if ( g_RecvBuff[4] == '1' ) {
                SetIOMode(IOchn[1]);
                Right();
            } else if ( g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[1]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "PM2=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 2;
            if ( g_RecvBuff[4] == '1' ) {
                SetIOMode(IOchn[2]);
                Right();
            } else if (g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[2]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "PM3=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 3;
            if ( g_RecvBuff[4] == '1' ) {
                SetIOMode(IOchn[3]);
                Right();
            } else if (g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[3]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "PM4=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 4;
            if ( g_RecvBuff[4] == '1' ) {
                SetIOMode(IOchn[4]);
                Right();
            } else if (g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[4]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
```

```
    } else if ( 0 == strncmp( pString, "PM5=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 5;
            if ( g_RecvBuff[4] == '1' ) {
                SetIOMode(IOchn[5]);
                Right();
            } else if (g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[5]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else if ( 0 == strncmp( pString, "PM6=", 4 ) ) {
        if ( g_RecvCount == 5 ) {
            g_PinModeIndex = 6;
            if ( g_RecvBuff[4] == '1' ) {
                SetIOMode(IOchn[6]);
                Right();
            } else if (g_RecvBuff[4] == '0' ) {
                ClrIOMode(IOchn[6]);
                Right();
            } else {
                Error();
            }
        } else {
            Error();
        }
    } else {
        Error();
    }
    memset( (char *)&g_RecvBuff , 0 , PKTLEN );
}

void Error() {
    Serial.println( "ERROR" );
}
void Right() {
    Serial.println( "TRUE" );
}

void Char_Convertion( void ) {
    uint8_t i   = 0;
    char *pString = (char *)&g_RecvBuff[0];
    for ( i = 0; (*pString) != '\0'; i++ ) {
        if ( *pString >= 'a' && *pString <= 'z'  ) {
            *pString -= 32;
        }
        pString++;
    }
}
```

Till now, we have finished all the arduino board coding work.

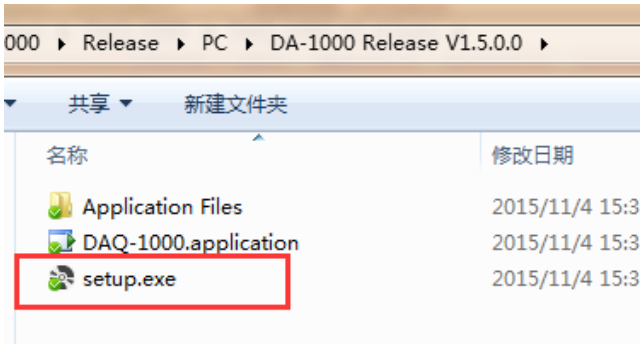
We can use a Serial Tool to making a test. The following picture shows the test result for request: Get Device Info.

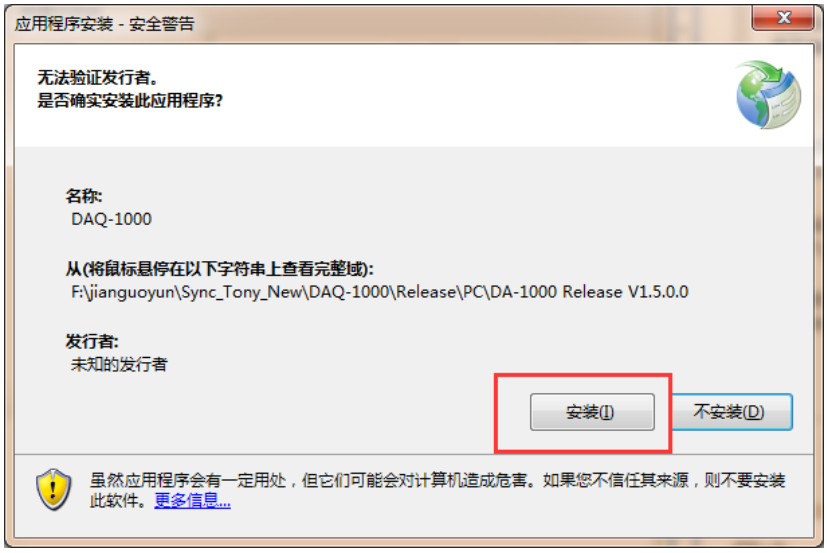
5. DAQ-1000 PC software using guide

INHAOS has provided a PC software DAQ-1000 for debugging the DAQ-1000 board. This chapter will guide you how to using the DAQ-1000 PC software.

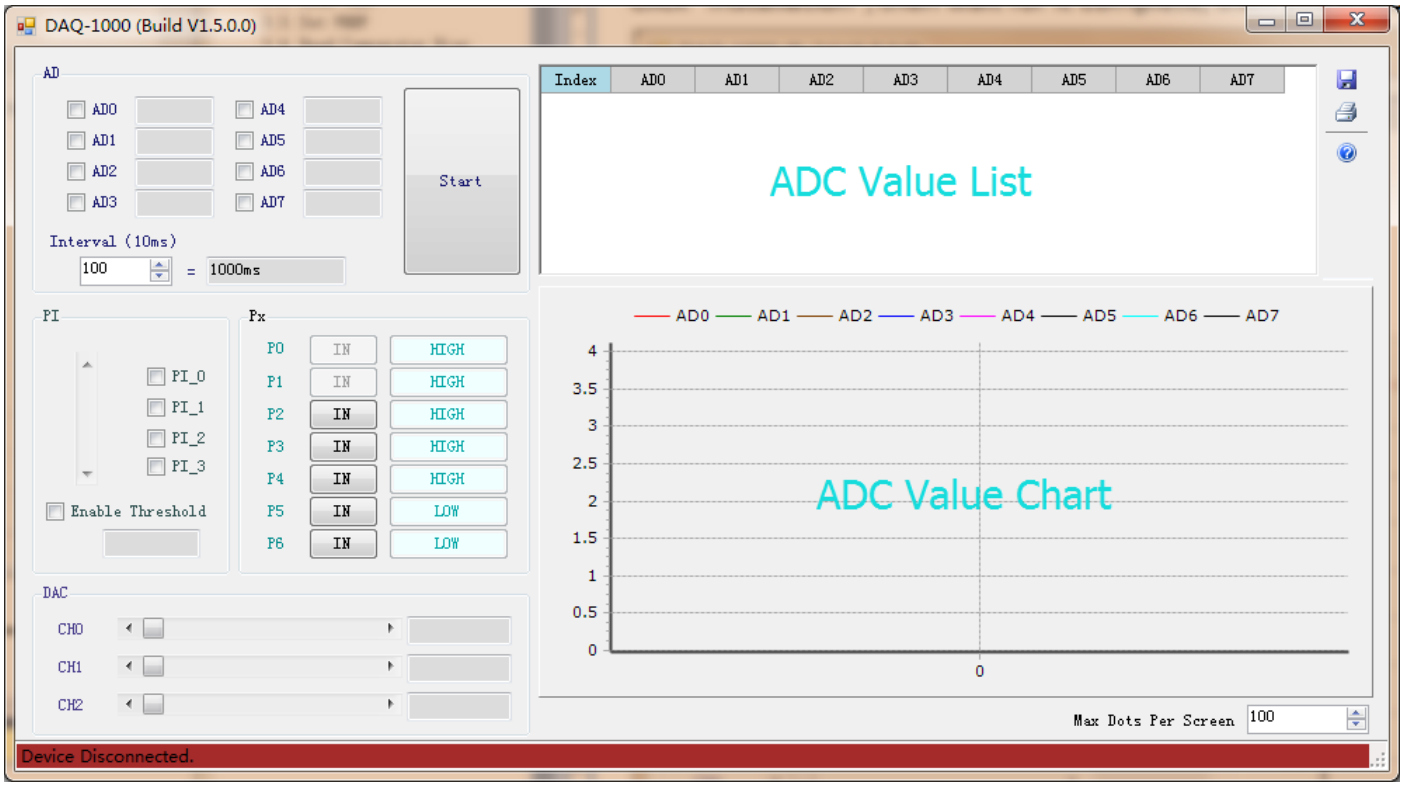
5.1. Installation

Browse to folder *DA-1000 Release V1.5.0.0* and find the *setup.exe*, double click it then start-up the installation.





Click "Installation", then wait for it complete, then you'll see the main window.

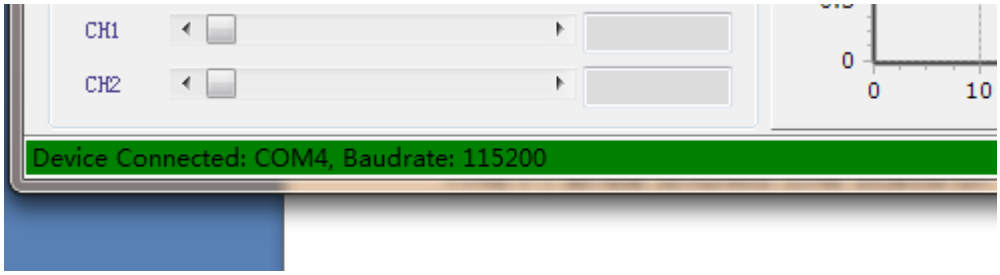


5.2. Using DAQ-1000

5.2.1. Connect DAQ-1000 board

Since the software started, it will automatically search for the DAQ-1000 board through the existed COM ports on computer port by port, till find the right one.

So now, plugged the board (which already loaded the codes previous chapters printed) into computer's USB port. Then the software will connected to it immediately. And the status bar shows the connected message, like below:

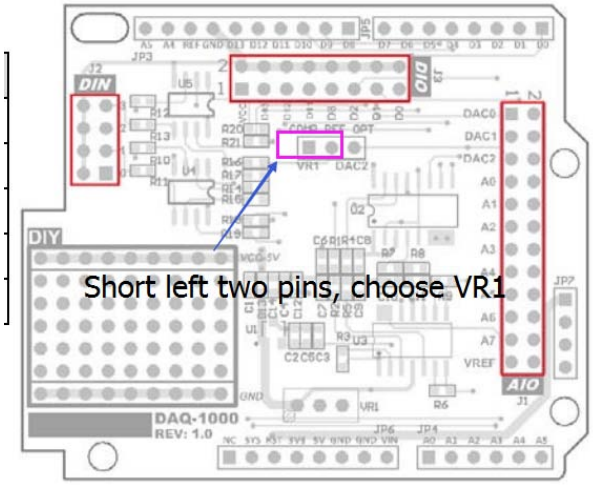


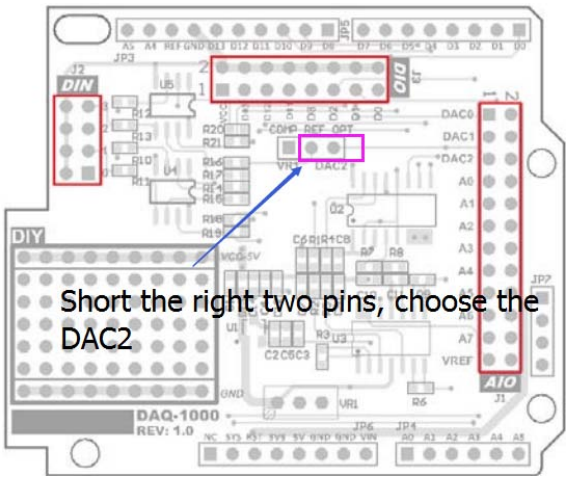
After device connected, the DIN Ports and P ports state will update every 10ms.

5.2.2. DIN Ports

The PI area shows the state of the DIN pins, you can test the DIN pins by pull-up to VCC or pull-down to GND, and the DAQ-1000 will update its latest state.

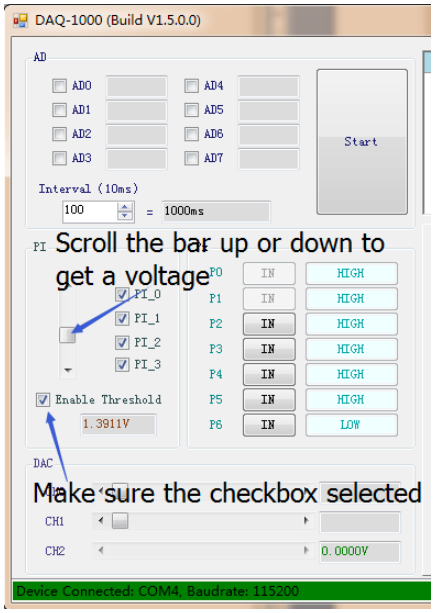
There are two ref source which user can select for the DIN comparator, the VR1 and the DAC_CH2. You can select one of them via change the short-jumper.





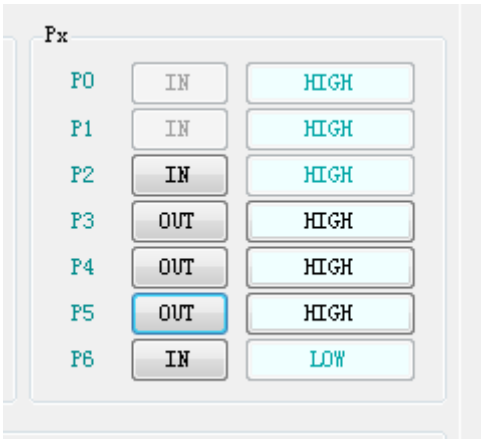
When choose VR1 as the comparator reference, you can tune the VR1 to get the appropriate compare voltage you needed.

And when choose the DAC2, you can tune the compare voltage by set the DAC_CH2's output level.



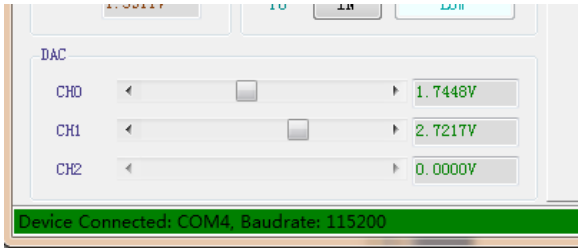
5.2.3. P Ports

Each pin of the P port can be configured as input or output. When set the Px mode to IN, the state button will be disabled, and the Px state will be updated every 10ms automatically to consistent it with the arduino board; And when set the Px mode to OUTPUT, the state button will be enabled. If click the state button to HIGH, the pin on arduino board becomes HIGH; Otherwise, the pin on arduino board becomes LOW.



5.2.4. DAC

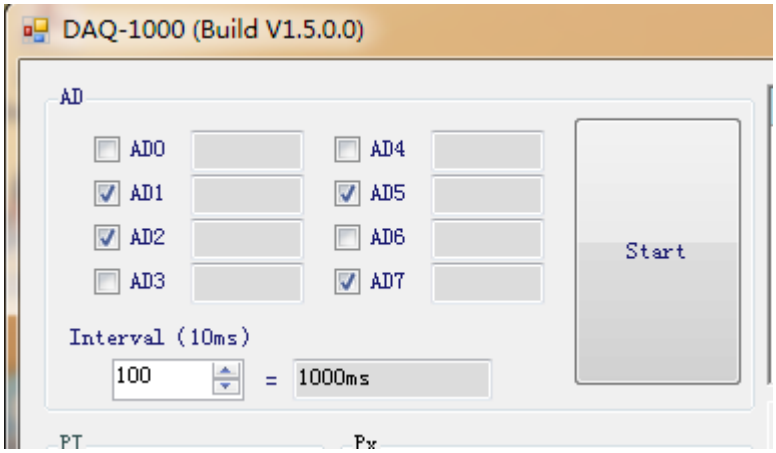
The DAC area controls the DAC output analog voltage level for each DAC channel.



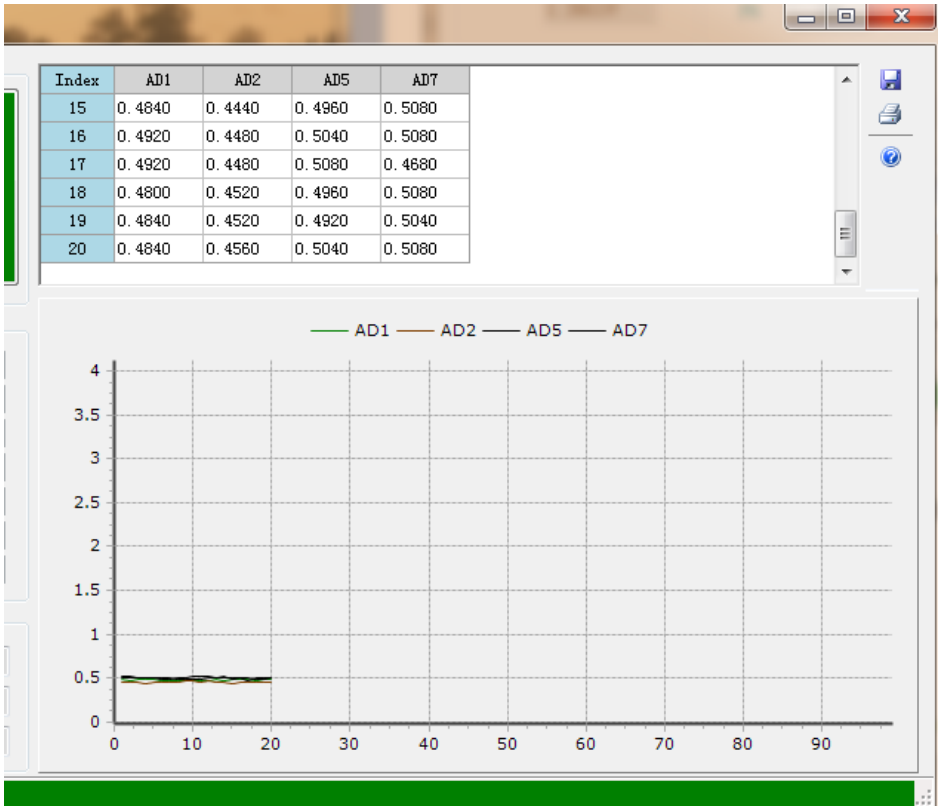
Please note, the resolution of the DAC_CH0/DAC_CH1 is 15bits. and DAC_CH2 is 8 bits.

5.2.5. ADC

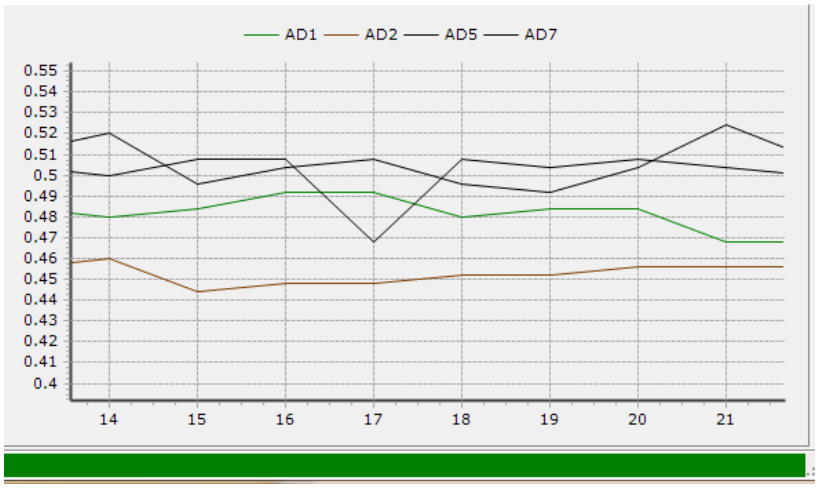
You can choose the ADC channels you need for periodic data acquisition. And the Interval decides the time period for acquisition, and the base time unit is 10ms.



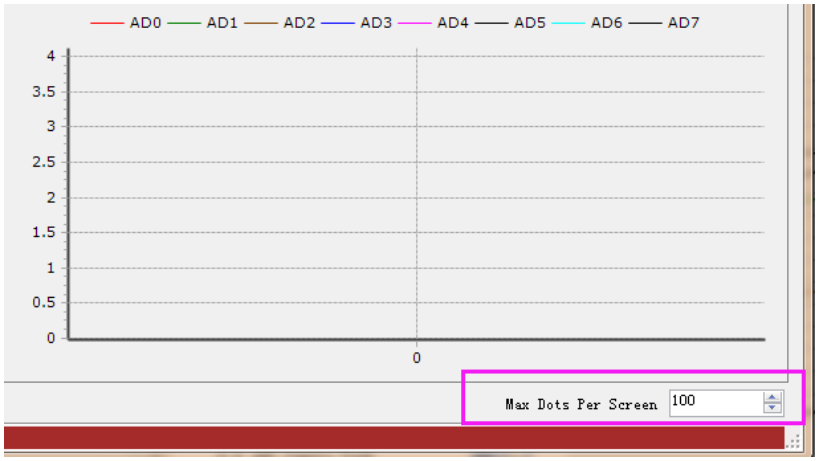
After channels selected, press the start button, then the periodic data acquisition starting. The data of ADC channels will be filled in the ADC Value list, meanwhile, chart lines are showing in the ADC value chart.



The chart lines can be either zoomed select an area by mouse left button or scrolled by mouse right button. The following picture shows a zoomed example:



The sample dots will automatically pan left once the dots count exceeds the setting which decided by the "Max Dot Per Screen". For example, if we set the Max Dot Per Screen = 100, and the chart lines will pan left when sample dots reached 100.



The tool bars provides you a way to save the ADC data list to excel files or to print the list.