# X702/X703

Hardware User’s Manual

Table of Contents

# 1. General Introduction

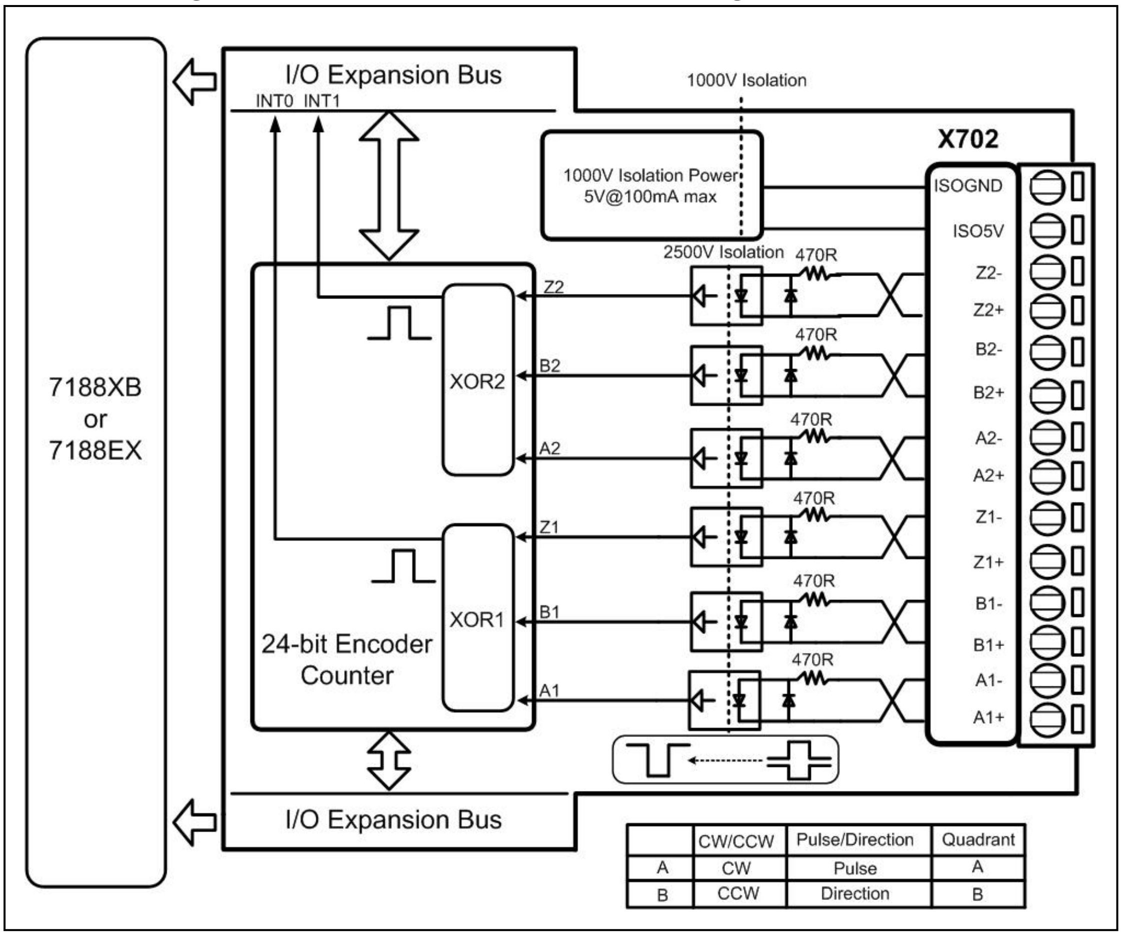
The X702/X703 is the 2-axis/3-axis expansion board designed for encoder applications. The specifications of X702/X703 are given as follows:

1. Support 7188XB or 7188EX
2. X702: 2-axis encoder counter
   X703: 3-axis encoder counter
3. 24-bit encoder counter
4. Encoder counting mode: Quadrant , CW/CCW , Pulse/Direction
5. Maximum counting rate: 1MHz
6. Encoder Input:
   X702: A1+,A1-,B1+,B1-,Z1+Z1- ,A2+,A2-,B2+,B2-,Z2+,Z2-
   X703:A1, B1, Z1, A2, B2, Z2, A3, B3, Z3
7. Input Level:
   Input 5V
      Logic High: 3.5~5V
      Logic Low: 0~2.0V

   Input 12V with external resistor, 1K ohm, 1/4W
      Logic High: 5~12V
      Logic Low: 0~2.0V

   Input 24V with external resistor, 2K ohm, 1/2W
      Logic High: 7~24V
      Logic Low: 0~2.0V
8. A/B/Z signal isolation voltage: 2500V optical isolation
9. Built-in isolated voltage output: +5V, 100mA max., 1000V isolation
10. Built-in XOR logic for active high or active low encoder input

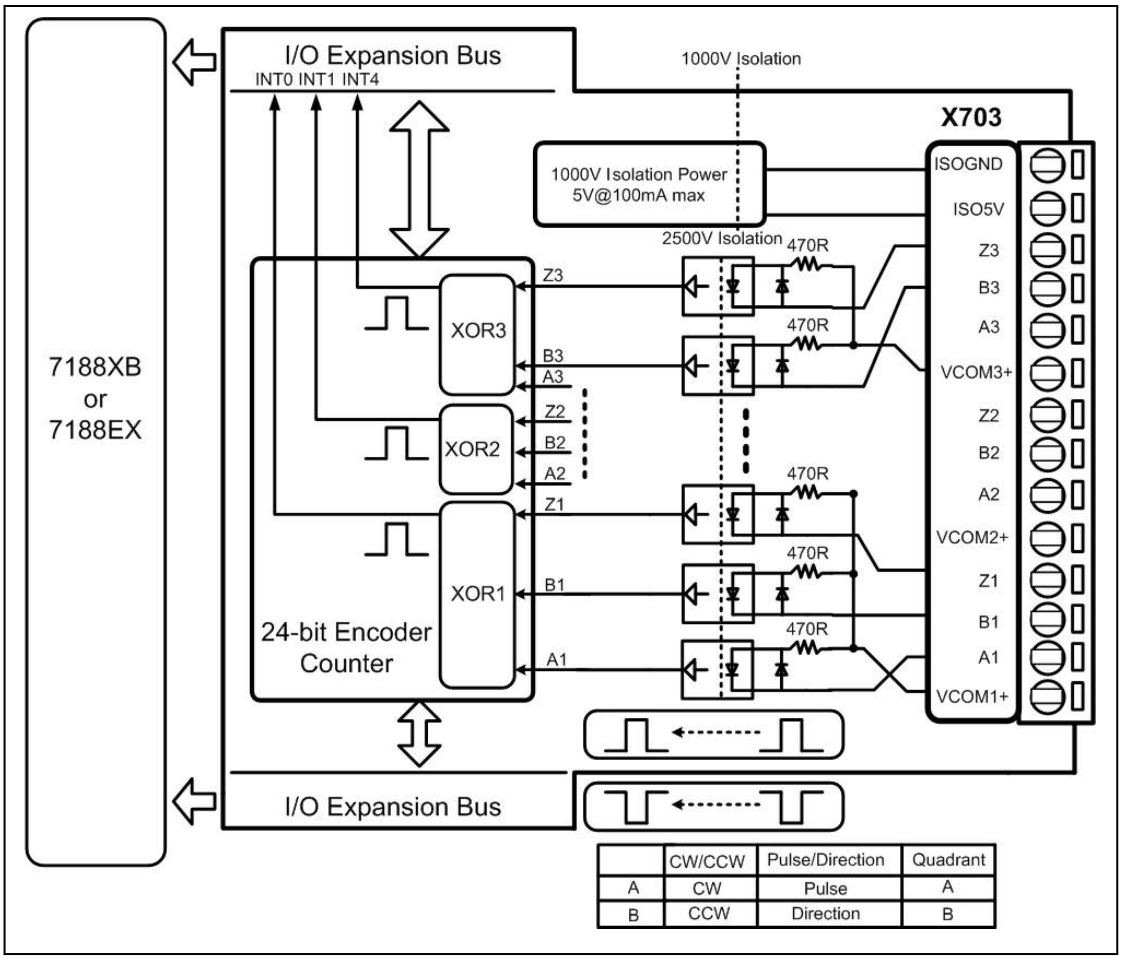
The block diagram of X702 with differential encoder is given as follows:



| | CW/CCW | Pulse/Direction | Quadrant |
|---|---|---|---|
| A | CW | Pulse | A |
| B | CCW | Direction | B |

- X702 can be plugged into 7188XB or 7188EX.
- X702 equips a isolated 5V power for 5V encoder.
  Refer to Sec. 1.2 for 12V encoder & Sec. 1.3 for 24V encoder.
- (A1+,A1-)(B1+,B1-)(Z1+, Z1-) is for encoder1 and
  (A2+,A2-)(B2+,B2-)(Z2+,Z2-) is for encoder2.
  These signals are connected to photo-couple devices for signal isolation.
- The internal logic is active high.
- The Z1 after XOR1 logic is connected to int0 of I/O Expansion Bus
  The Z2 after XOR2 logic is connected to int1 of I/O Expansion Bus
  Both int0 & int1 are initial low & active high.
  Refer to Sec. 3.1.3 for more information
- For differential encoder
  (A1+,A1-)(B1+,B1-)(Z1+,Z1-) are active high, XOR1/XOR2 must be set to 1.
- Refer to Sec. 1.1.1, Sec. 1.1.3, Sec. 1.2.1, Sec. 1.3.1 for more information about this configuration, X702+differential encoder.

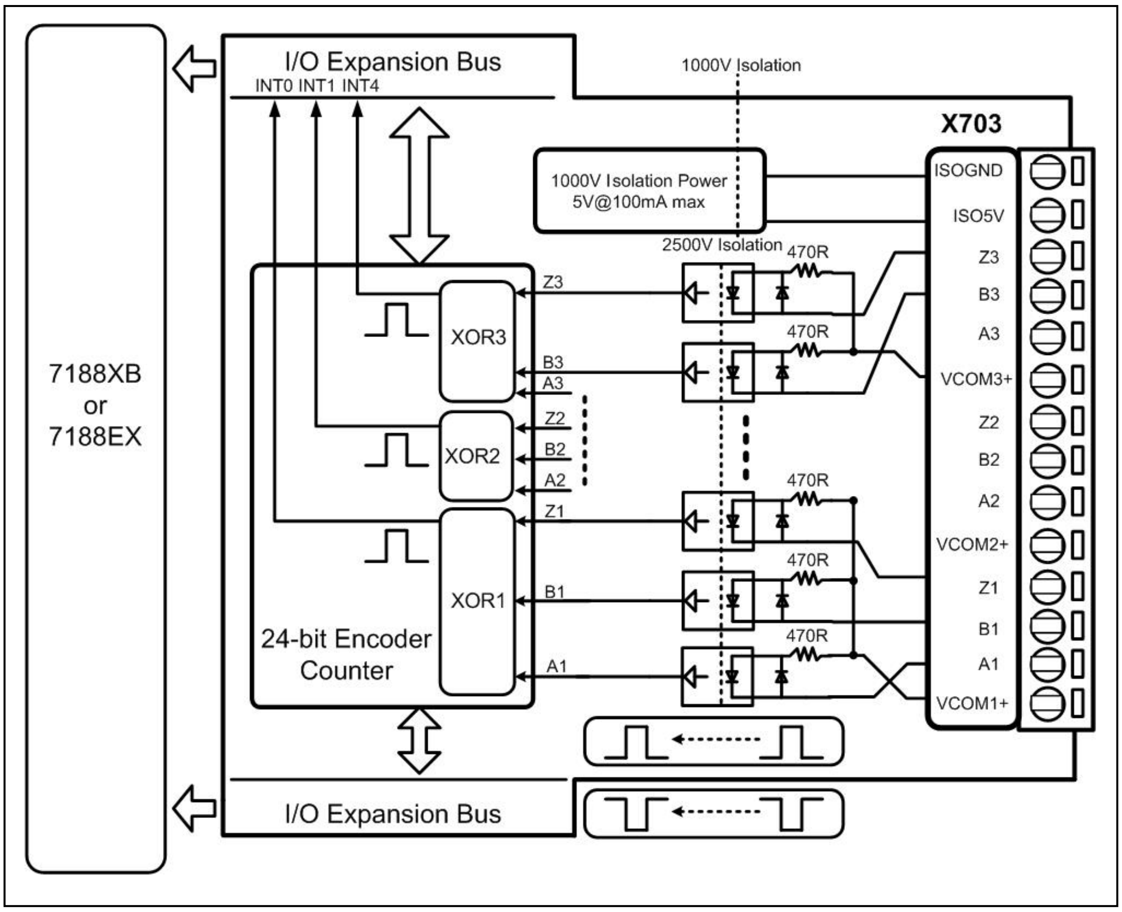
The block diagram of X702 with single-ended encoder is given as follows:



| | CW/CCW | Pulse/Direction | Quadrant |
|---|---|---|---|
| A | CW | Pulse | A |
| B | CCW | Direction | B |

- For single-ended encoder
  If (A,B,Z) is active low, XOR1/XOR2 must be set to 1
  If (A,B,Z) is active high, XOR1/XOR2 must be set to 0
- Refer to Sec. 1.1.2, Sec. 1.1.4, Sec. 1.2.2, Sec. 1.3.2 for more information about this configuration, X702+single-ended encoder

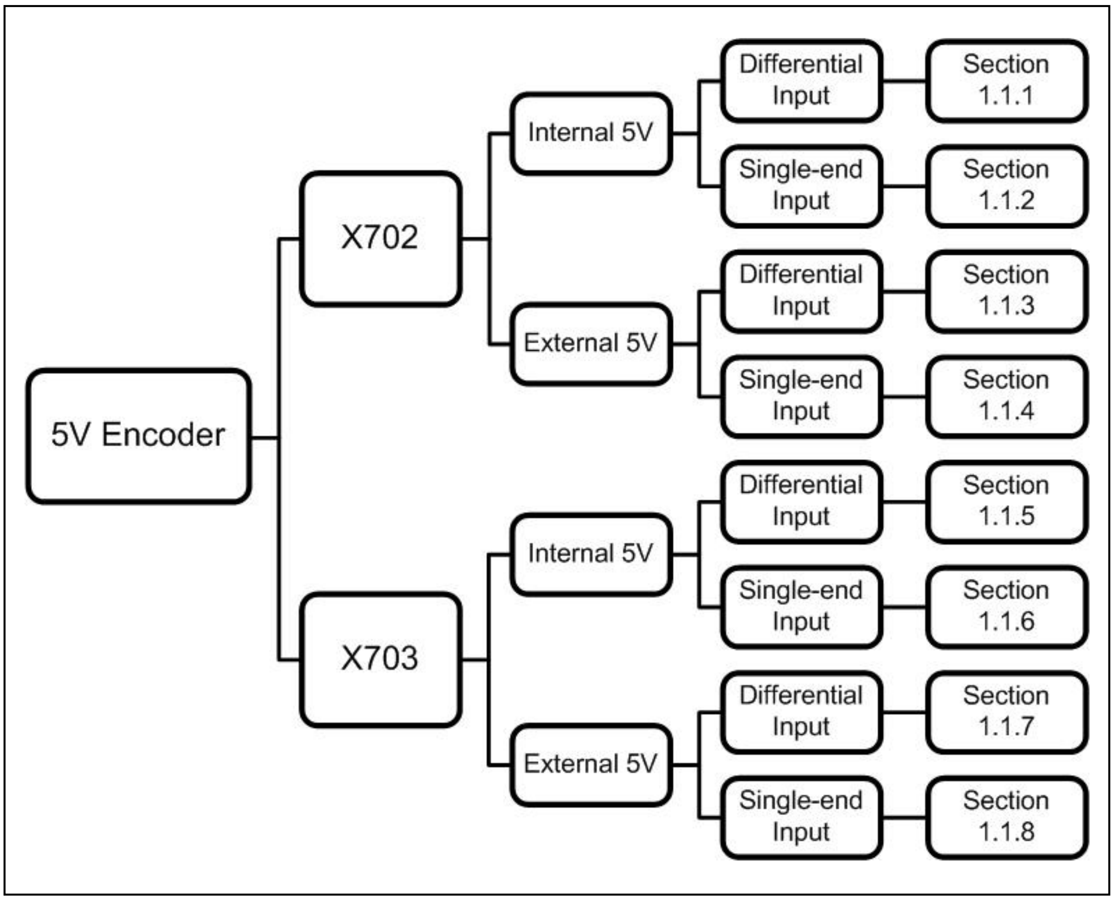
The block diagram of X703 is given as follows:



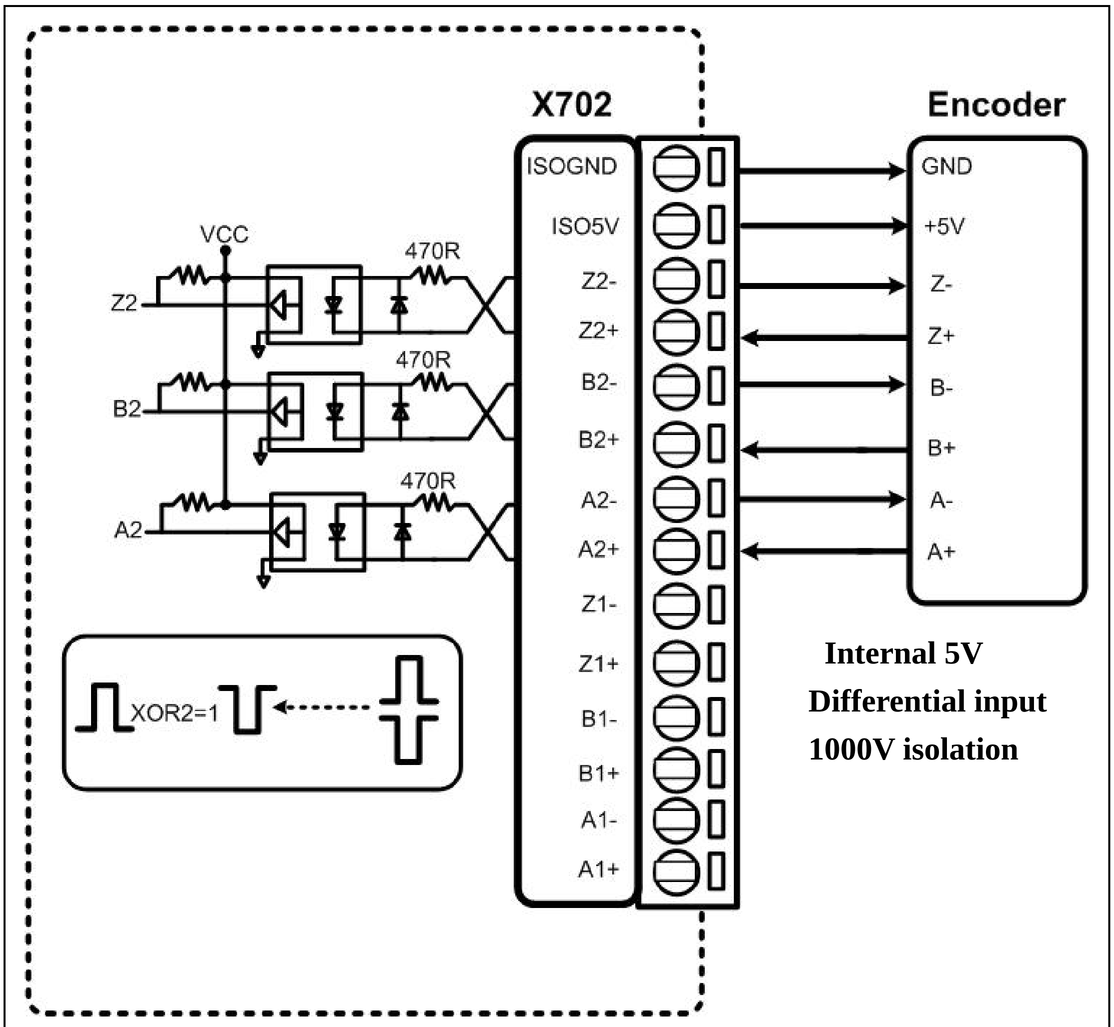
- X703 can be plugged into 7188XB or 7188EX.
- X703 equips a isolated 5V power for 5V encoder.
  Refer to Sec. 1.2 for 12V encoder & Sec. 1.3 for 24V encoder.
- (VCOM1+,A1,B1,Z1) is for encoder1, (VCOM2+, A2,B2,Z2) is for encoder2 and (VCOM3+,A3,B3,Z3)is for encoder3.
  These signals are connected to photo-couple devices for signal isolation.
- The internal logic is active high.
  If (VCOM1,A1,B1,Z1) is active high, XOR1 must be set to 0
  If (VCOM1,A1,B1,Z1) is active low, XOR1 must be set to 1
  If (VCOM2+,A2,B2,Z2) is active high, XOR2 must be set to 0
  If (VCOM2+,A2,B2,Z2) is active low, XOR2 must be set to 1
  If (VCOM3+,A3,B3,Z3) is active high, XOR3 must be set to 0
  If (VCOM3+,A3,B3,Z3) is active low, XOR3 must be set to 1
- The Z1 after XOR1 logic is connected to int0 of I/O Expansion Bus
  The Z2 after XOR2 logic is connected to int1 of I/O Expansion Bus
  The Z3 after XOR3 logic is connected to int1 of I/O Expansion Bus
  Refer to Sec. 3.2.3 for more information

## 5V Encoder Signal Wiring

User can use internal 5V or external 5V to drive 5V encoder. It is recommended to use internal 5V for 5V encoder. The isolation voltage of internal 5V power is 1000V and the isolation voltage of photo-couple is 2500V. Using internal 5V power, the system isolation voltage will be 1000V. This configuration can fit most of applications. User can use external 5V power supply to increase the system isolation voltage to 2500V. The signal wiring diagram are given in Sec. 1.1.1 ~ Sec. 1.1.8 as follows:

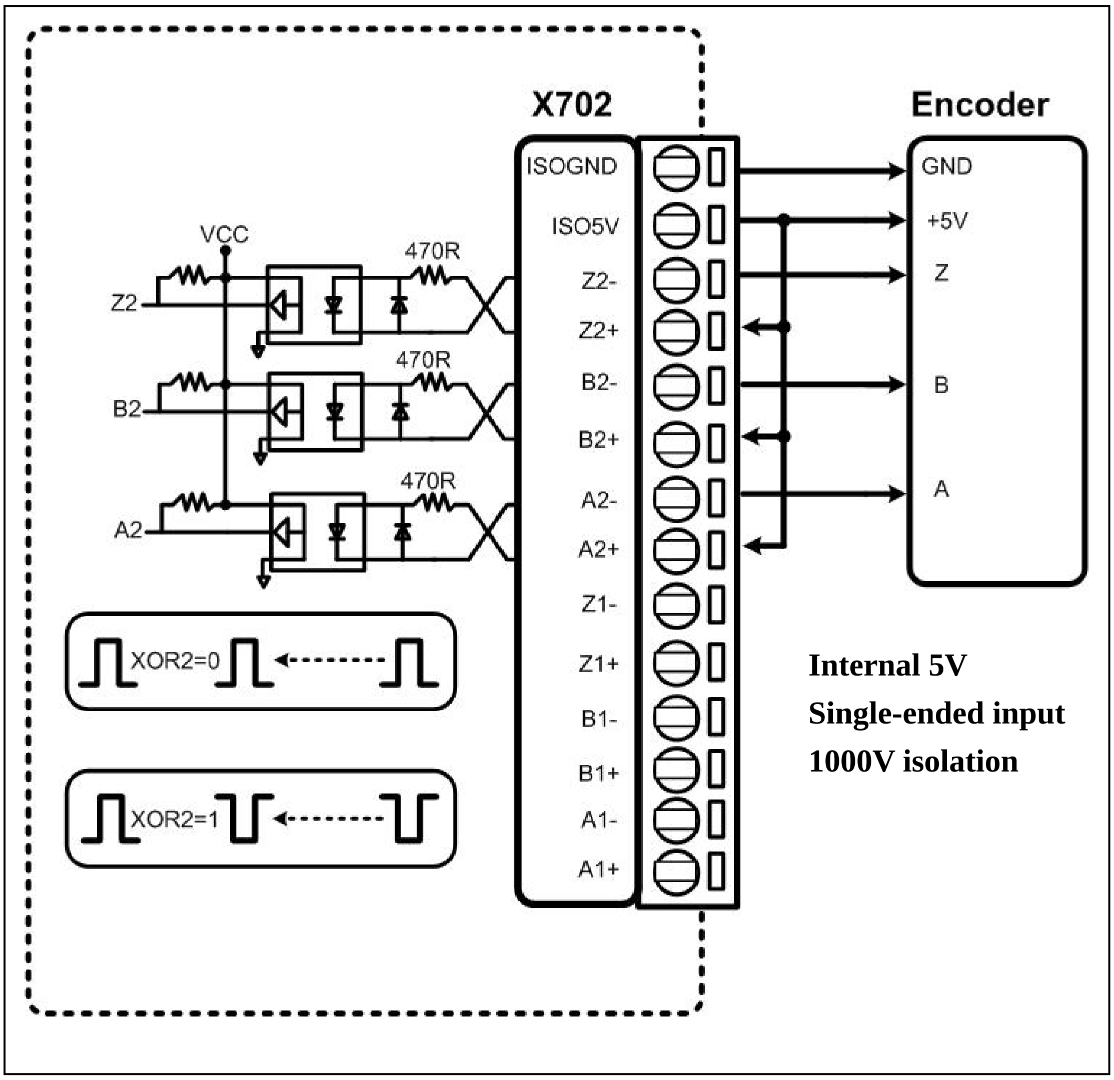


## 1.1.1. X702: Differential Encoder & Int_5V



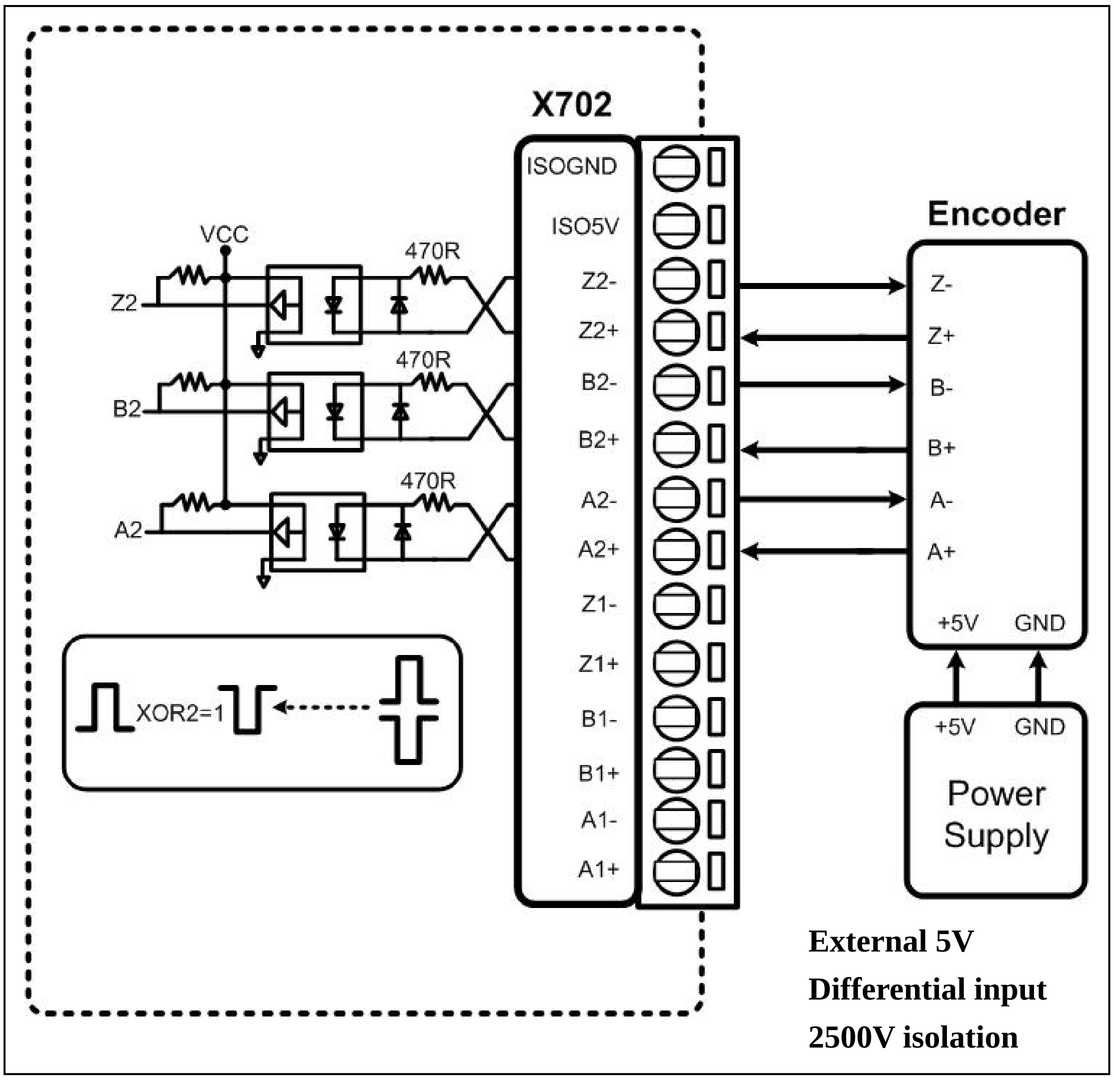
- The isolation voltage of 5V power is 1000V. So the isolation voltage of this configuration is also 1000V.
- The XOR2 bit must be set to 1 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

## 1.1.2. X702: Single-ended Encoder + Int_5V



- The isolation voltage of 5V power is 1000V. So the isolation voltage of this configuration is also 1000V.
- If A/B/Z signals of encoder2 are active low, the XOR2 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active high, the XOR2 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

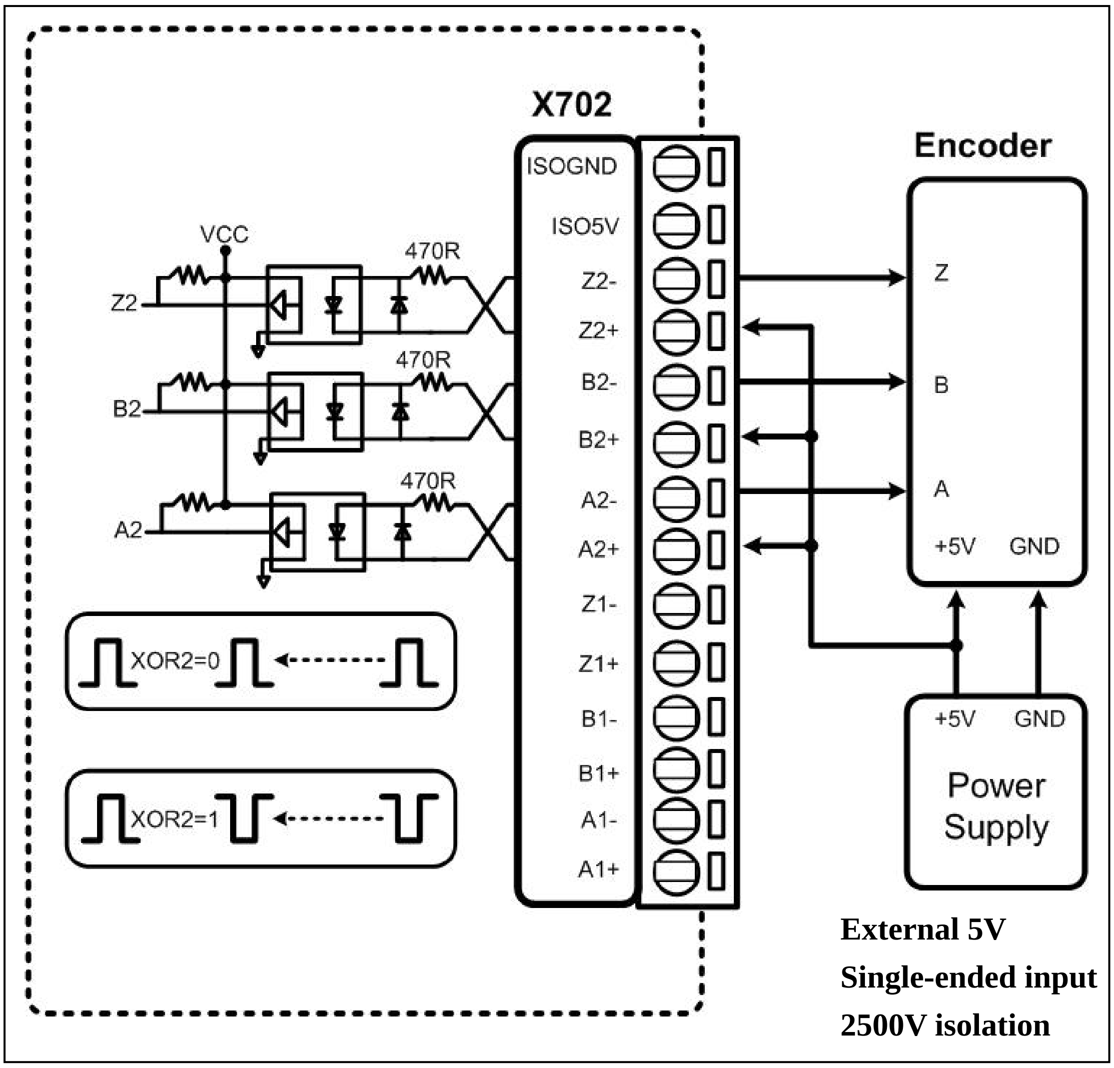
## 1.1.3. X702: Differential Encoder + Ext_5V



**External 5V**
**Differential input**
**2500V isolation**

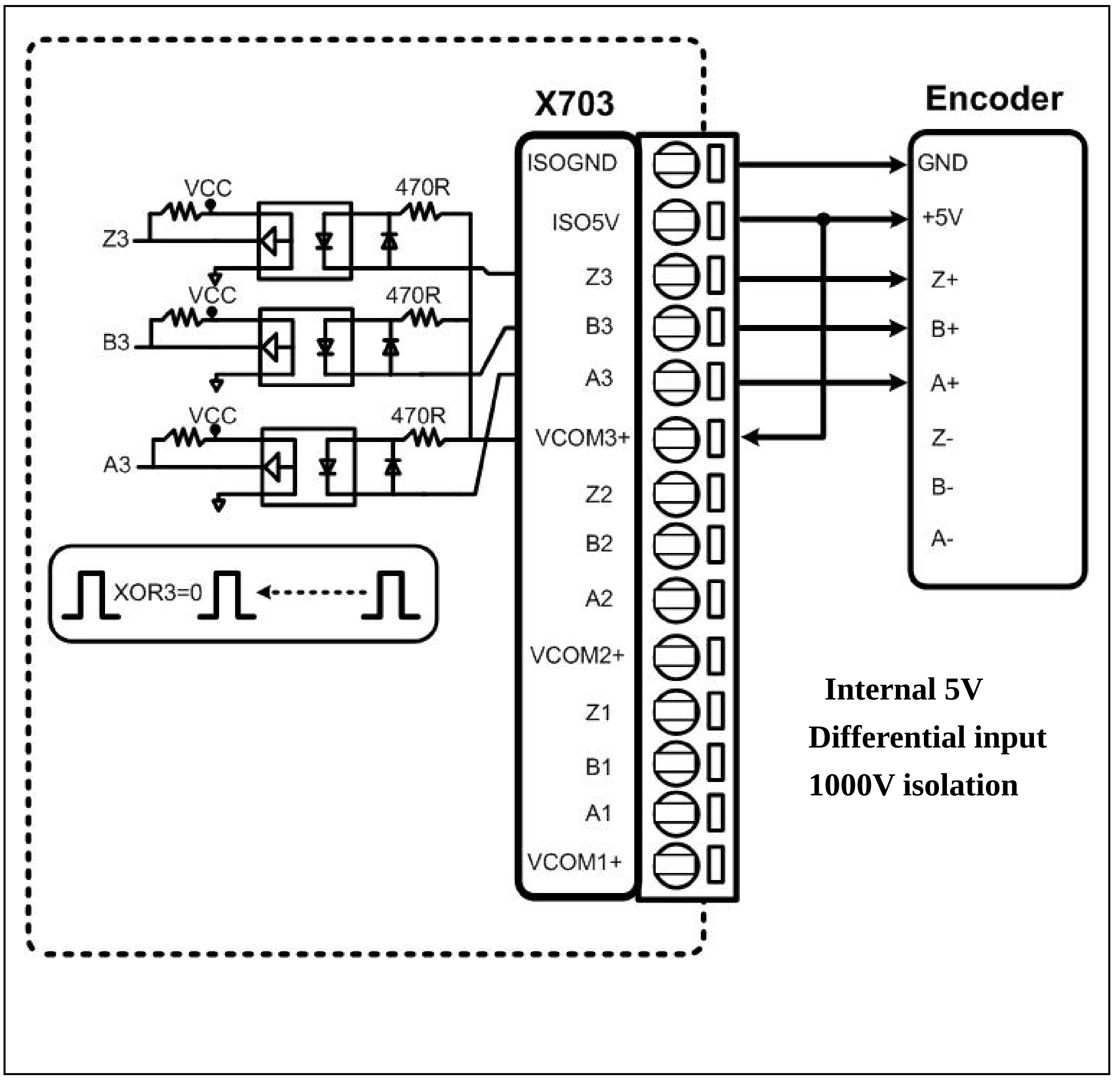
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- The XOR2 bit must be set to 1 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

## 1.1.4. X702: Single-ended Encoder + Ext_5V



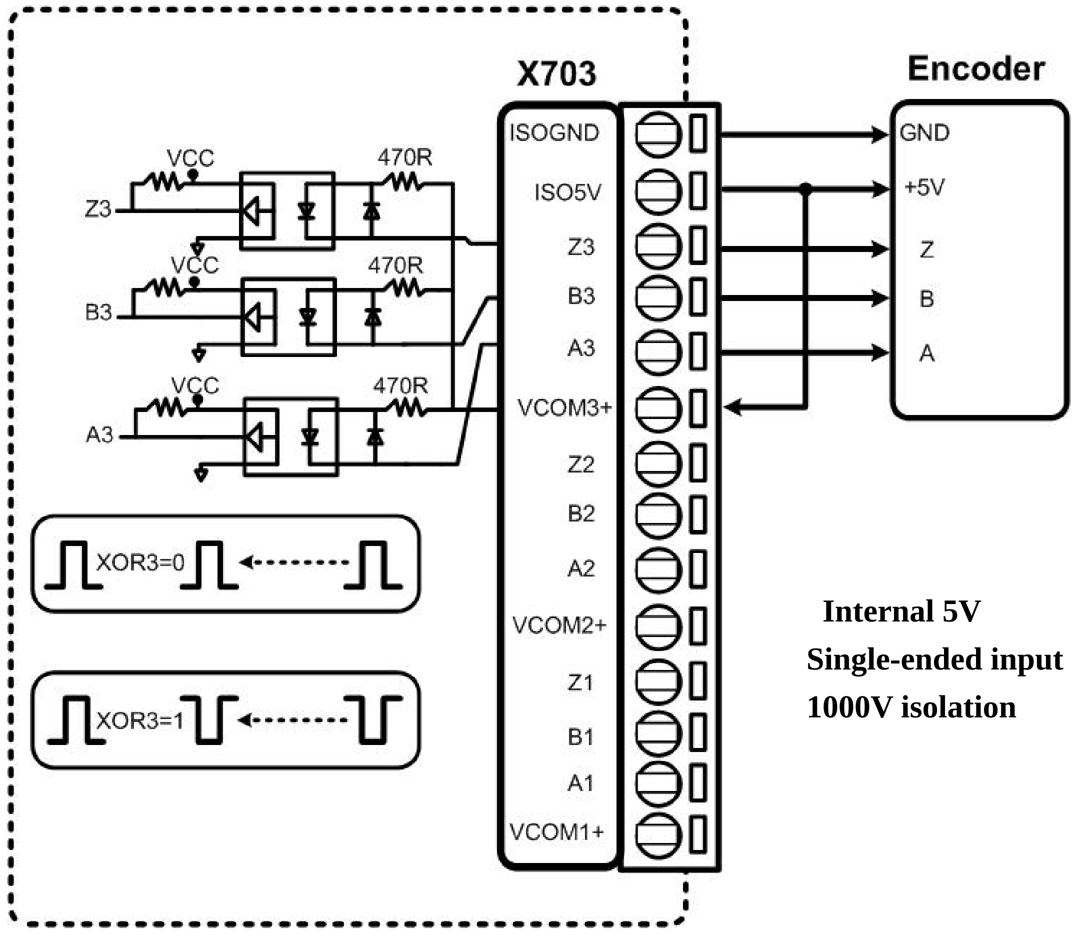
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- If A/B/Z signals of encoder2 are active low, the XOR2 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active high, the XOR2 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

## 1.1.5. X703: Differential Encoder + Int_5V



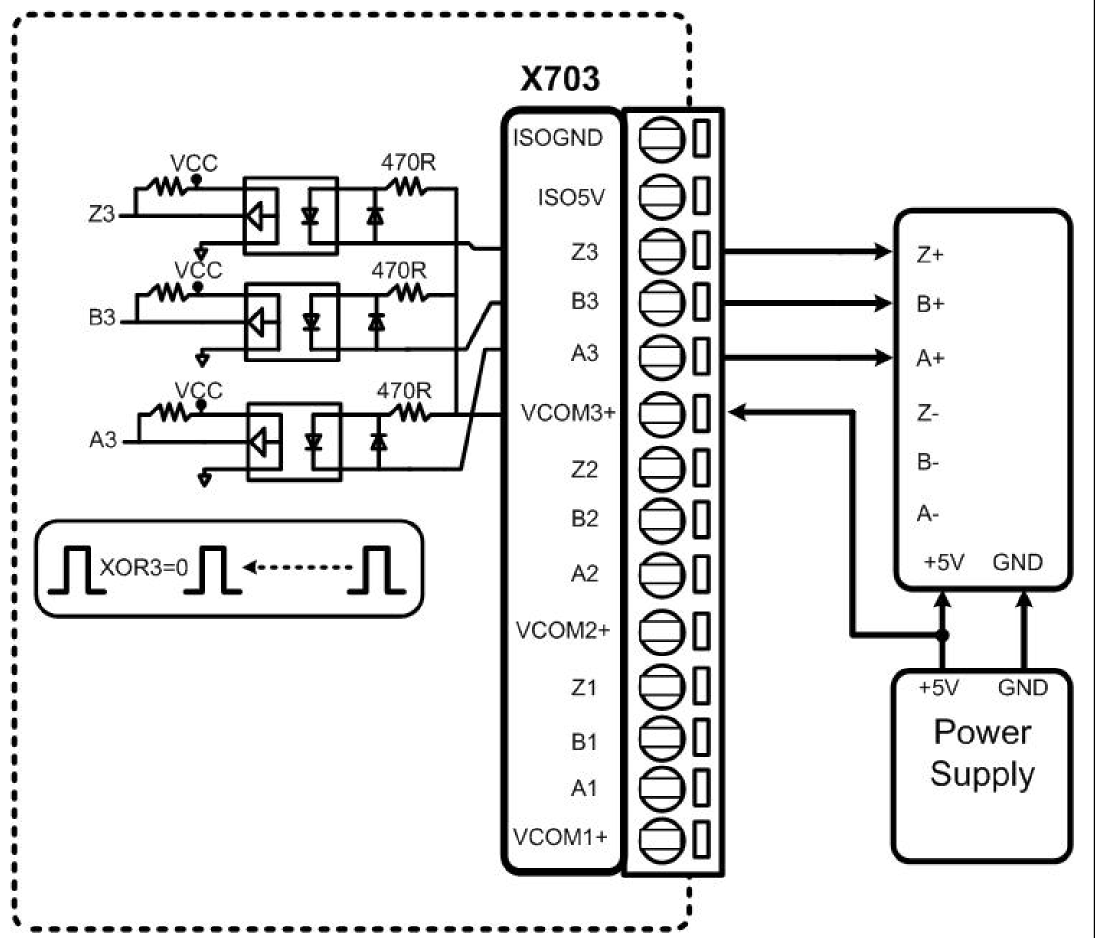
- The isolation voltage of 5V power is 1000V. So the isolation voltage of this configuration is also 1000V.
- The XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.1.6. X703: Single-ended Encoder + Int_5V



- The isolation voltage of 5V power is 1000V. So the isolation voltage of this configuration is also 1000V.
- If A/B/Z signals of encoder3 are active low, the XOR3 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder3 are active high, the XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

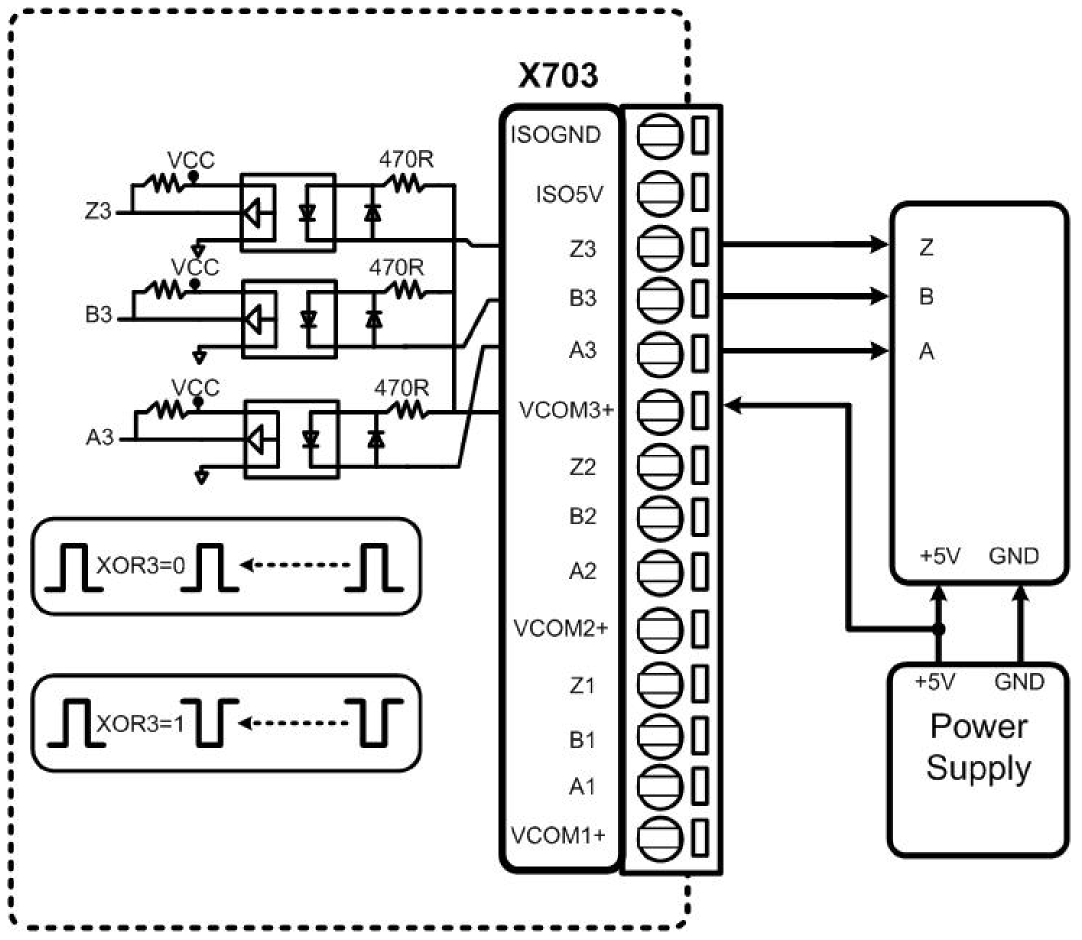
## 1.1.7. X703: Differential Encoder + Ext_5V



**External 5V**
**Differential input**
**2500V isolation**

- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- The XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.1.8. X703: Single-ended Encoder + Ext_5V

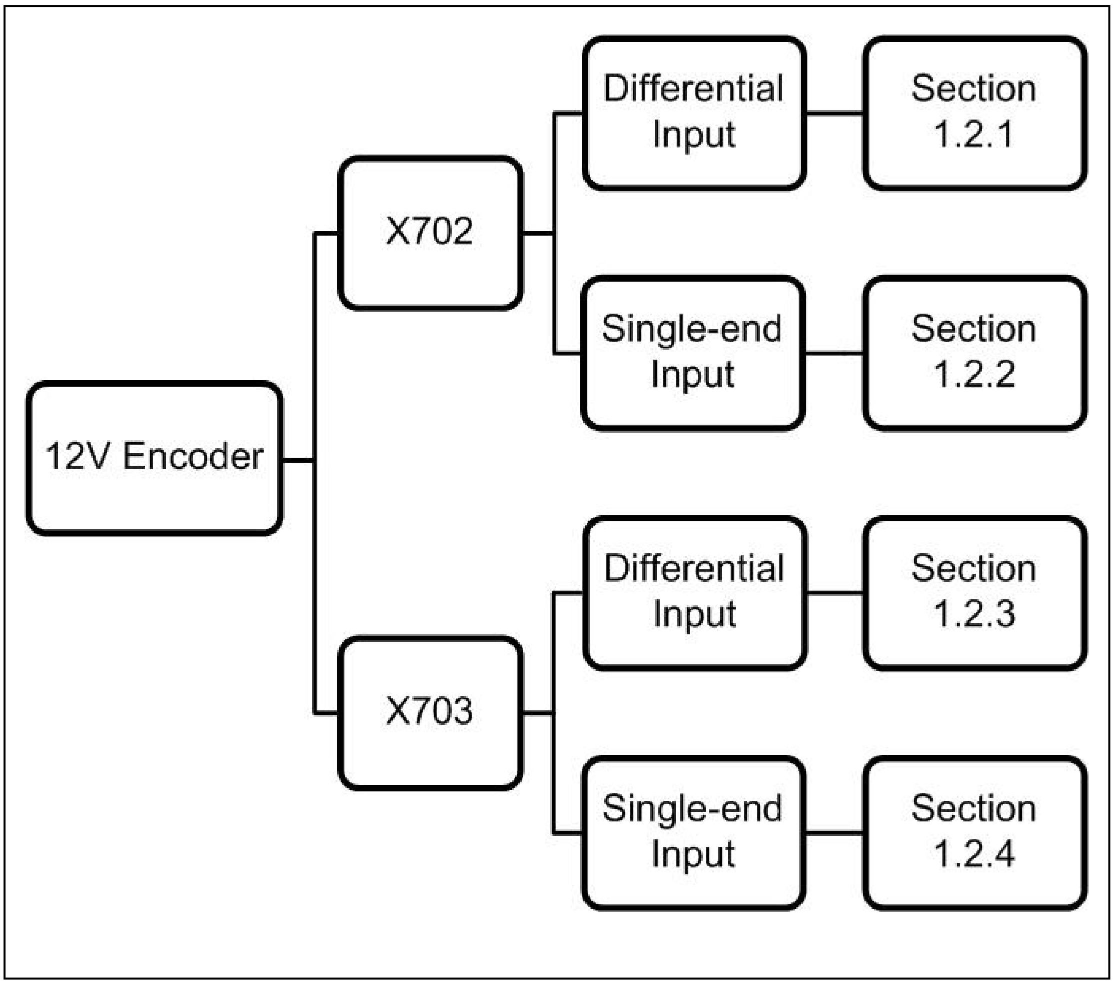


**External 5V**
**Single-ended input**
**2500V isolation**

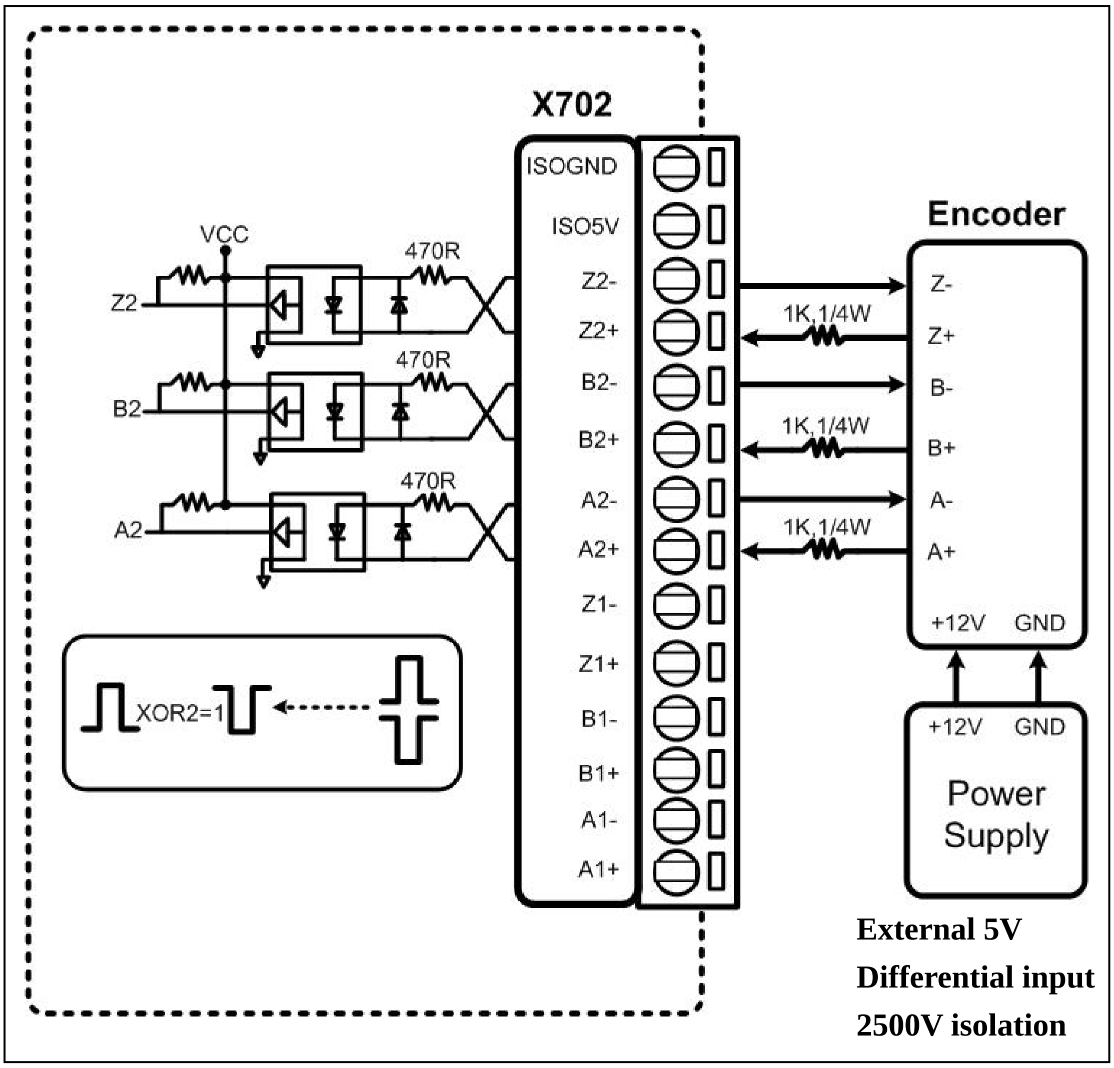
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- If A/B/Z signals of encoder3 are active low, the XOR3 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder3 are active high, the XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.2. 12V Encoder Signal

The internal logic of X702/X703 is designed for 5V logic. A current limit resistor must be added to X702/X703 for 12V encoder. The signal wiring diagram are given in Sec. 1.2.1 ~ Sec. 1.2.4 as follows:

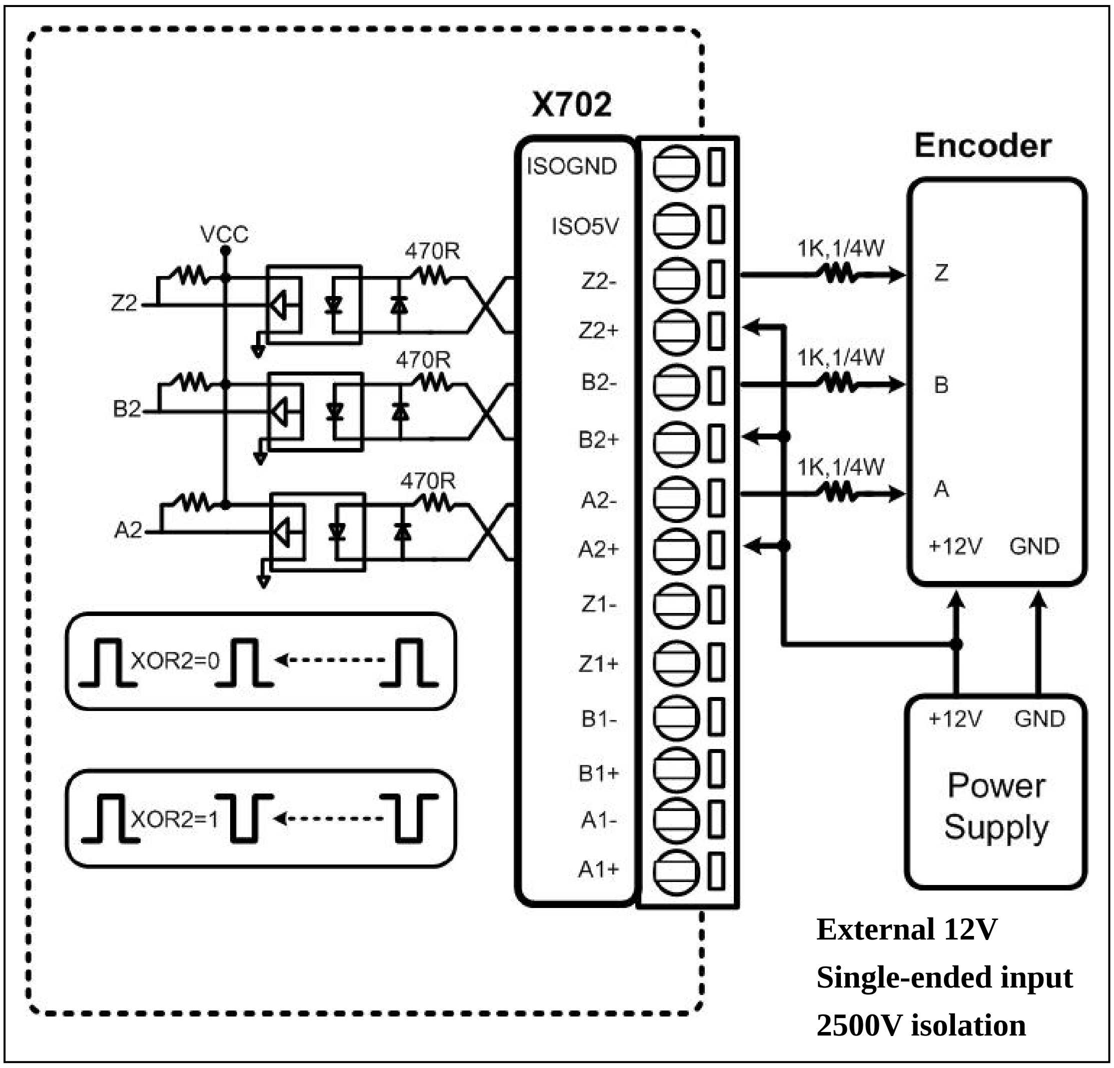


## 1.2.1. X702: Differential Encoder + Ext_12V



- **Note**: some current limit resistors, **1K, 1/4W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- The XOR2 bit must be set to 1 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

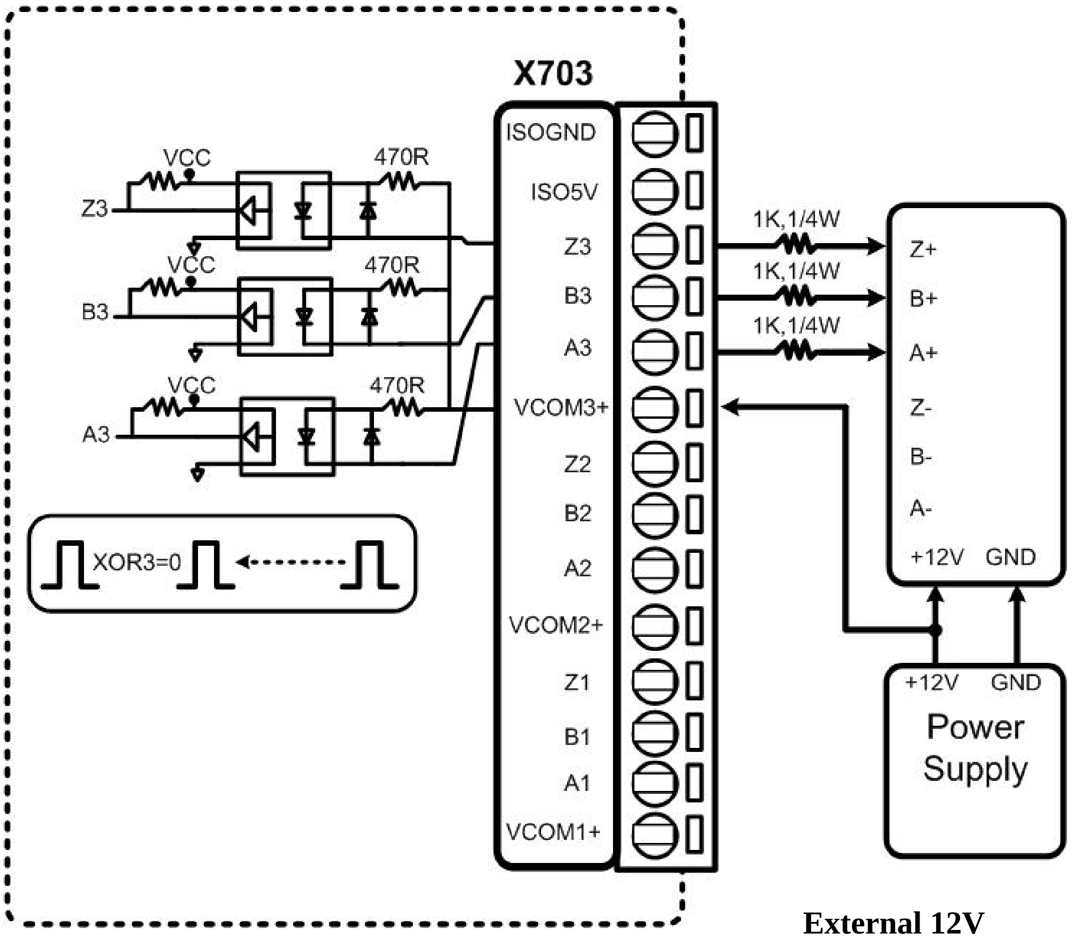
## 1.2.2. X702: Single-ended Encoder + Ext_12V



**External 12V**
**Single-ended input**
**2500V isolation**

- **Note**: some current limit resistors, **1K, 1/4W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- If A/B/Z signals of encoder2 are active low, the XOR2 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active high, the XOR2 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

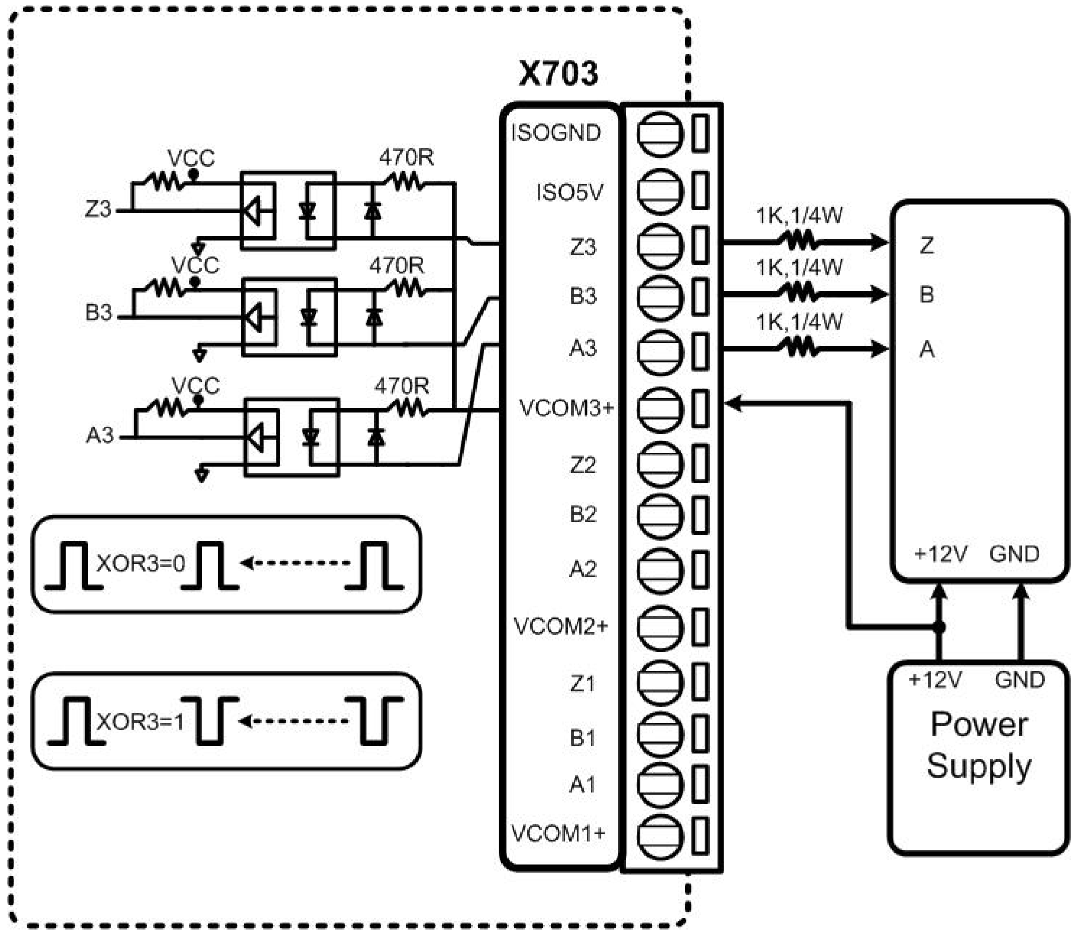
## 1.2.3. X703: Differential Encoder + Ext_12V



**External 12V**
**Differential input**
**2500V isolation**

- **Note**: some current limit resistors, **1K, 1/4W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- The XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.2.4. X703: Single-ended Encoder + Ext_12V

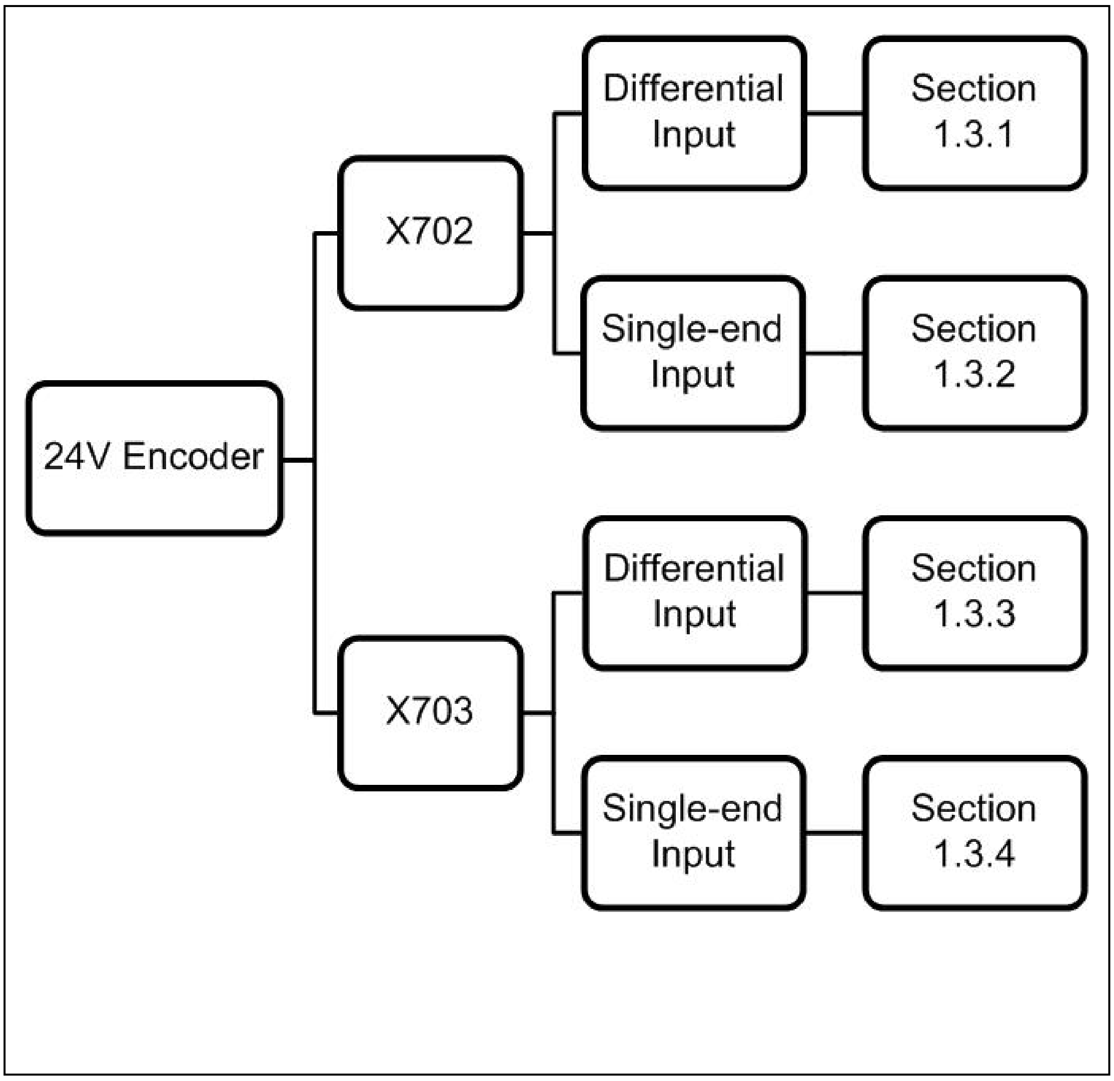


**External 12V**
**Single-ended input**
**2500V isolation**

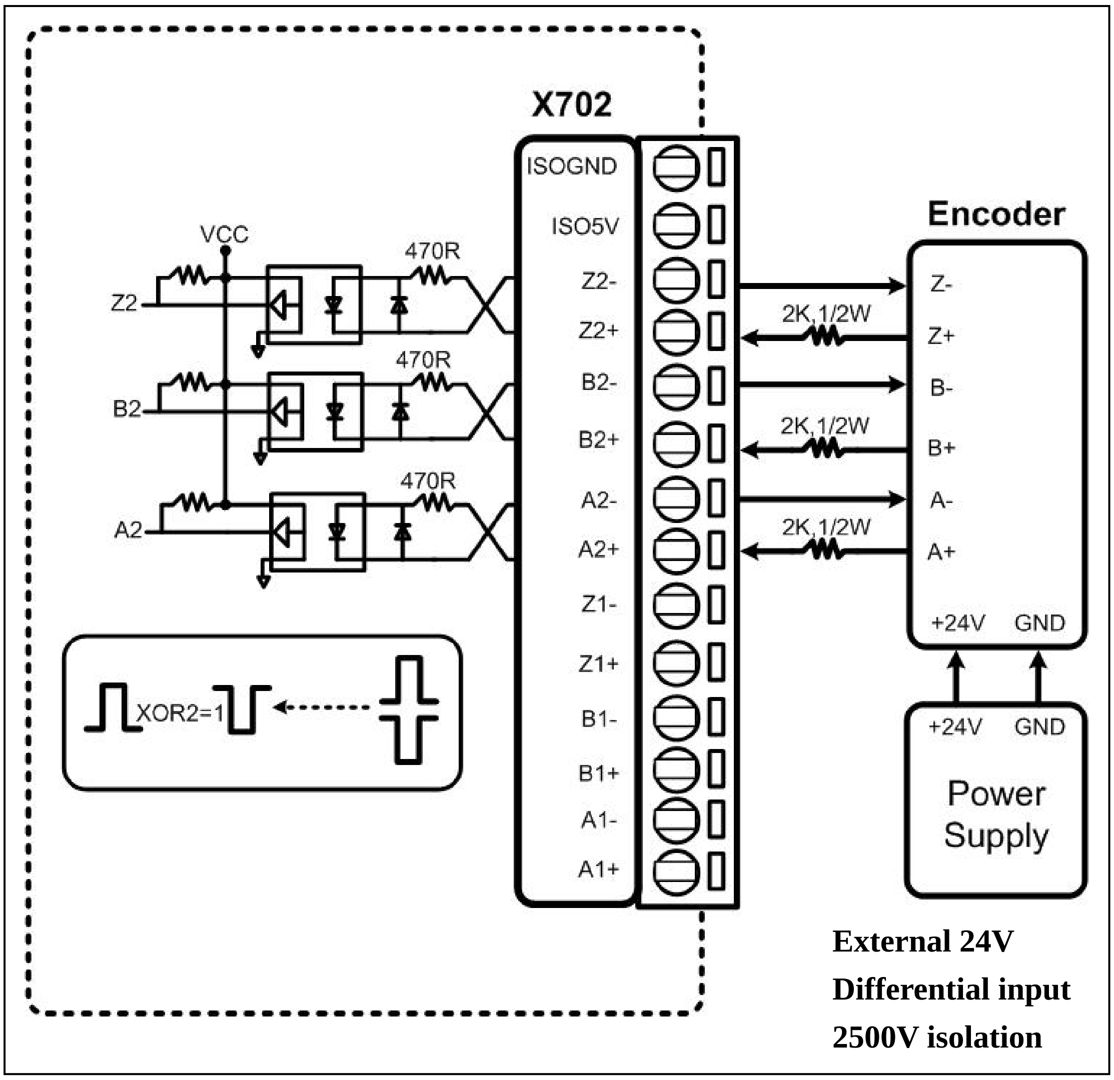
- **Note**: some current limit resistors, **1K, 1/4W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- If A/B/Z signals of encoder3 are active low, the XOR3 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder3 are active high, the XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.3. 24V Encoder Signal

The internal logic of X702/X703 is designed for 5V logic. A current limit resistor must be added to X702/X703 for 24V encoder. The signal wiring diagram are given in Sec. 1.3.1 ~ Sec. 1.3.4 as follows:



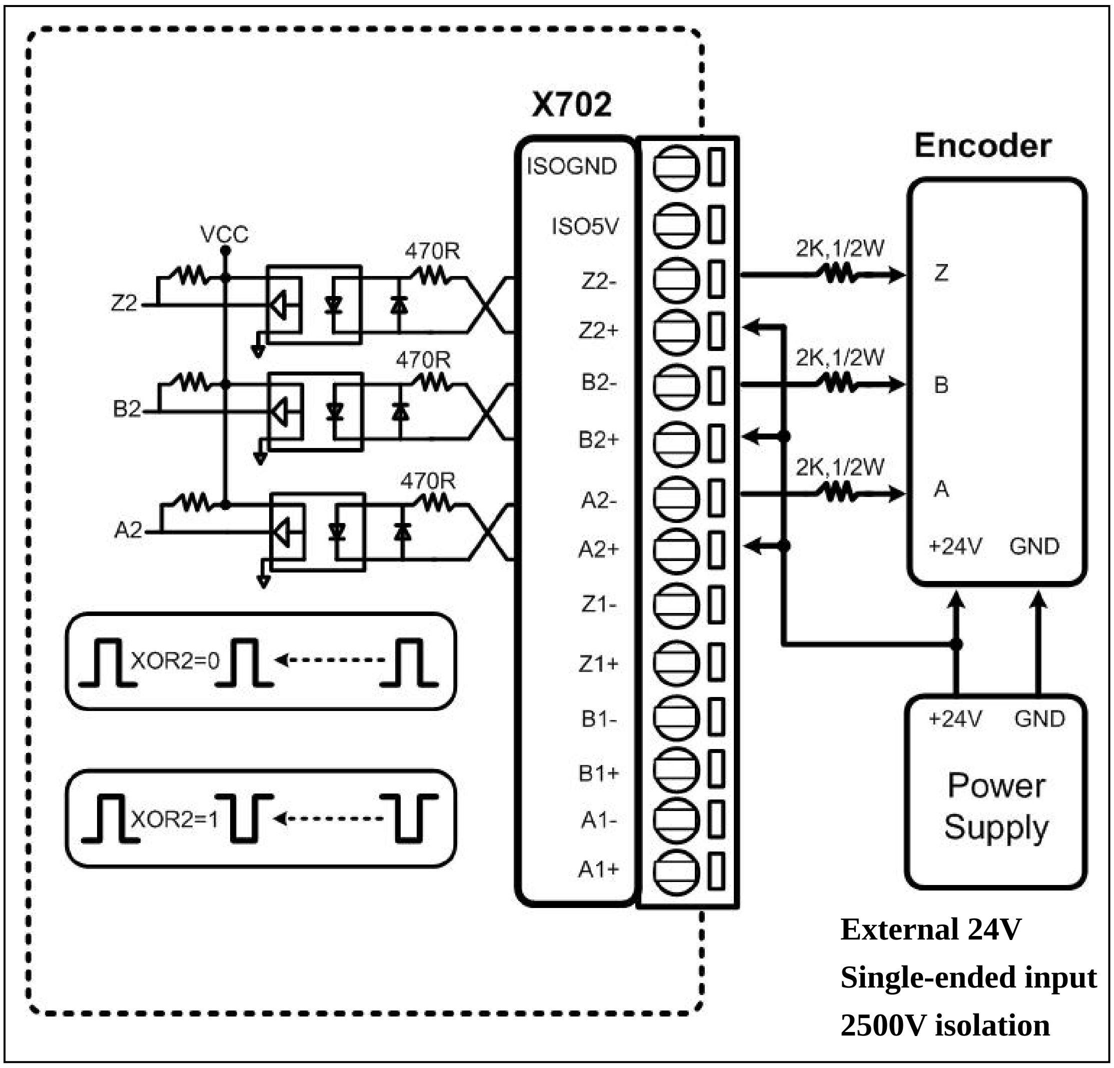
## 1.3.1. X702: Differential Encoder + Ext_24V



**External 24V**
**Differential input**
**2500V isolation**

- **Note**: some current limit resistors, **2K, 1/2W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- The XOR2 bit must be set to 1 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

## 1.3.2. X702: Single-ended Encoder + Ext_24V

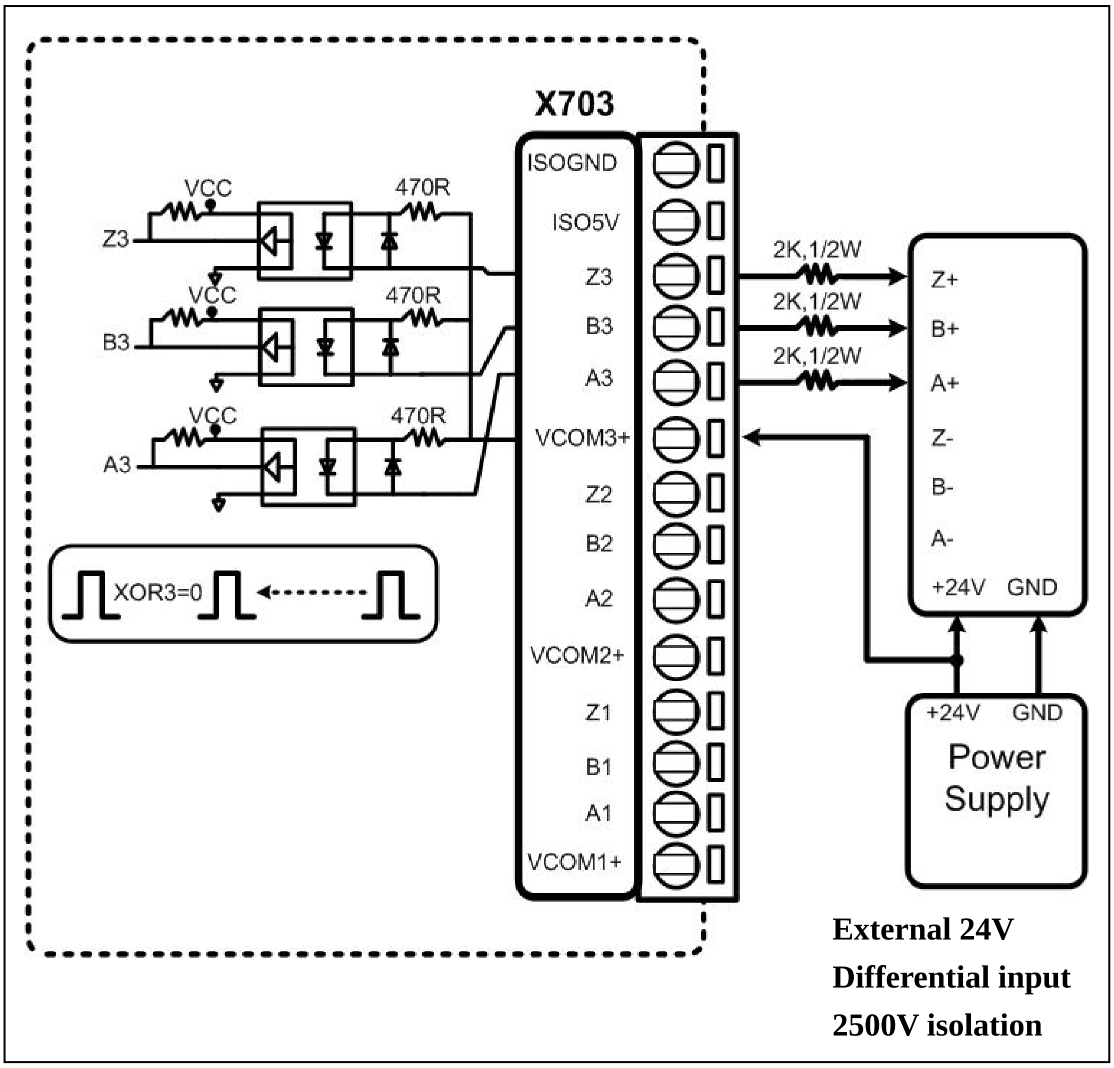


**External 24V**
**Single-ended input**
**2500V isolation**

- **Note**: some current limit resistors, **2K, 1/2W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- If A/B/Z signals of encoder2 are active low, the XOR2 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active high, the XOR2 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1 and XOR2 will apply to all A/B/Z signals of encoder2. Refer to Sec. 2.1.1 for XOR1/XOR2 control bits

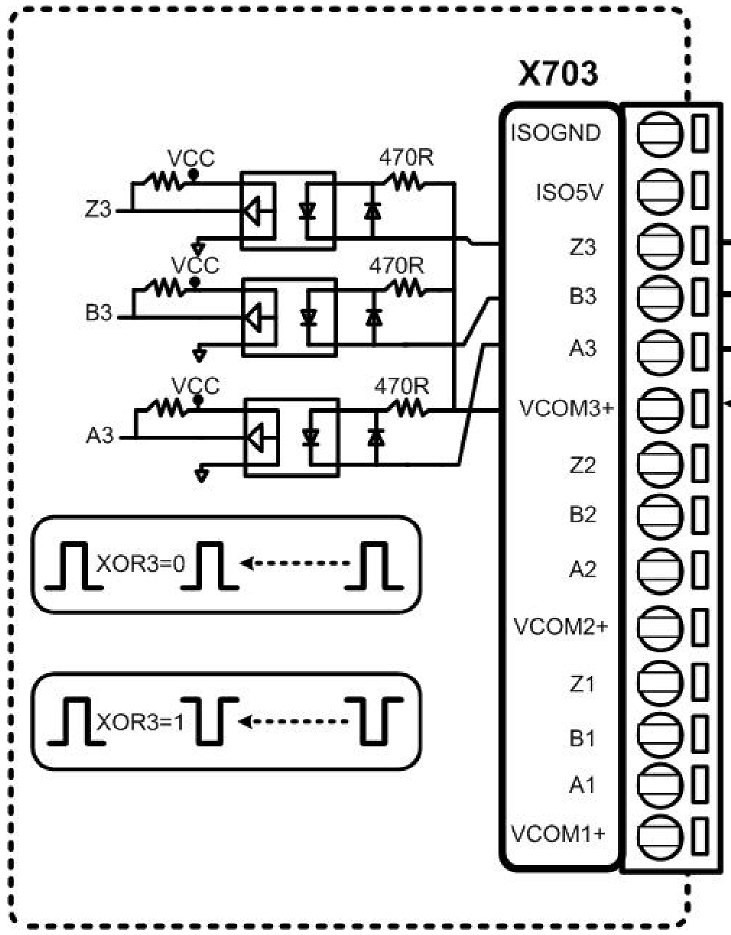
## 1.3.3. X703: Differential Encoder + Ext_24V

:



- **Note**: some current limit resistors, **2K, 1/2W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- The XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.3.4. X703: Single-ended Encoder + Ext_24V

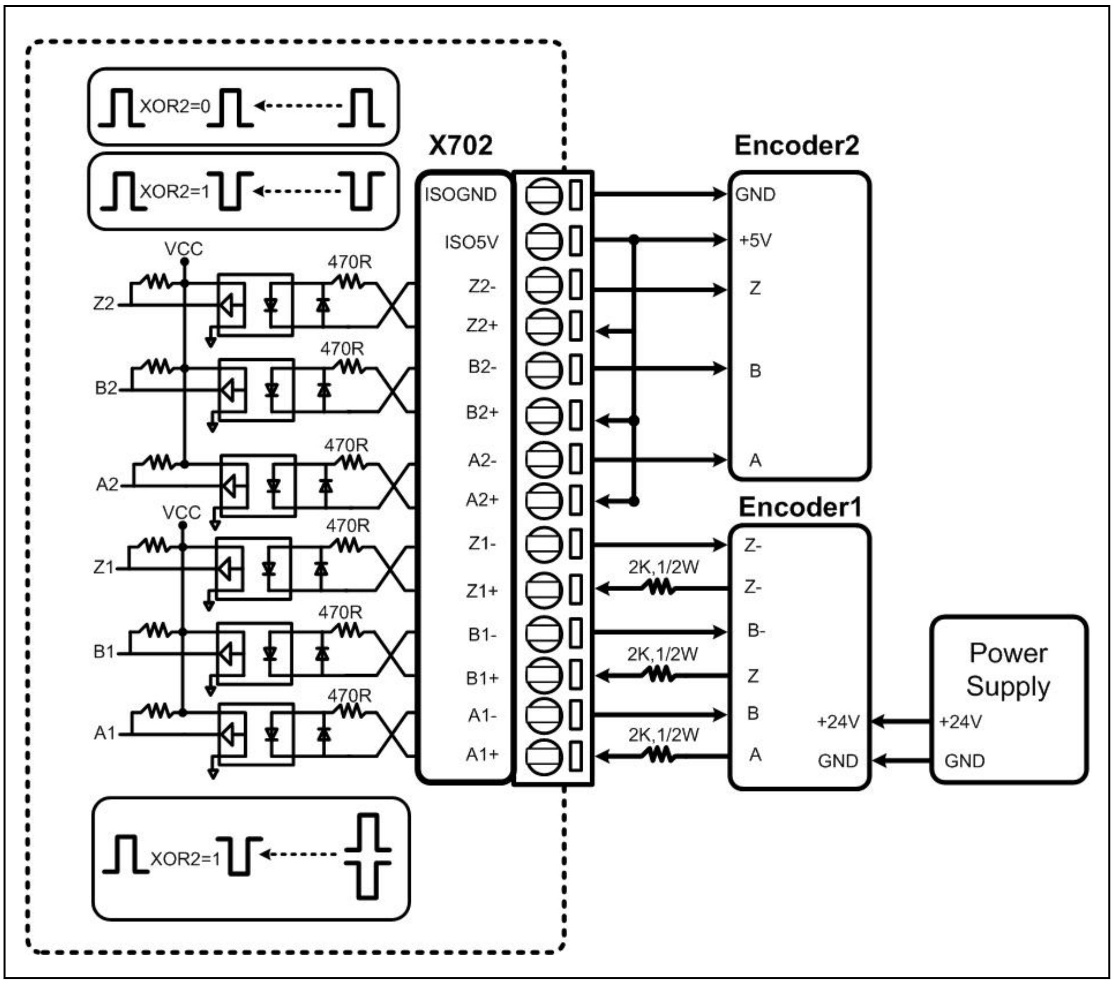


**External 24V**
**Single-ended input**
**2500V isolation**

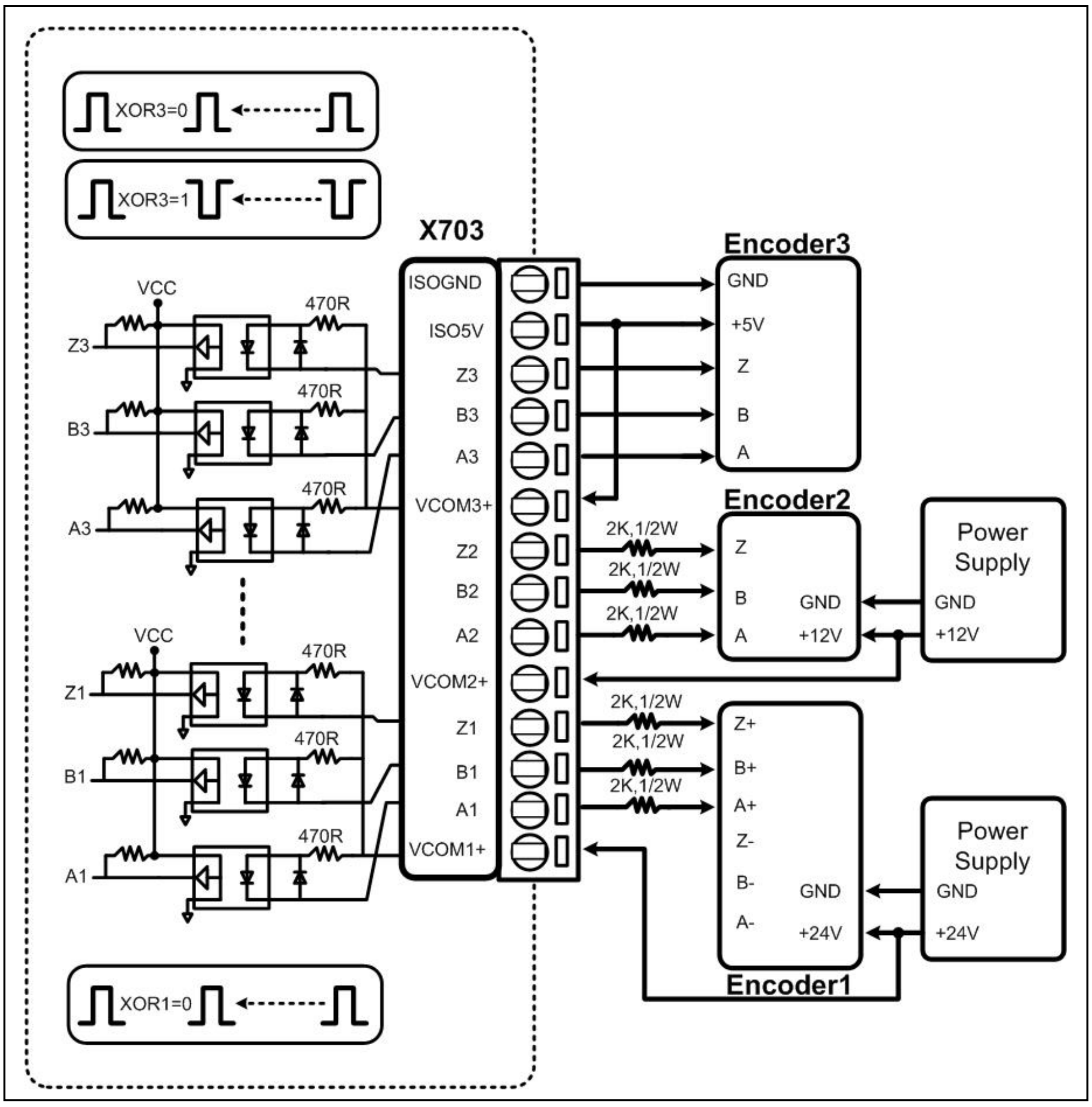
- **Note**: some current limit resistors, **2K, 1/2W** must be added as above diagram
- The isolation voltage of photo-couple is 2500V. Assume the isolation voltage of above power supply is greater than 2500V, the isolation voltage of this configuration will be 2500V.
- If A/B/Z signals of encoder3 are active low, the XOR3 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder3 are active high, the XOR3 bit must be set to 0 for internal active high logic
- The XOR1 will apply to all A/B/Z signals of encoder1, XOR2 will apply to all A/B/Z signals of encoder2 and XOR3 will apply to all A/B/Z signals of encoder3. Refer to Sec. 2.2.1 for XOR1/XOR2/XOR3 control bits

## 1.4. Mixed Configuration

- power can be internal 5V, external 5V, extern 12V or external 24V
- encoder signal can be single-ended or differential
- Any different type of encoders can used together as follows:



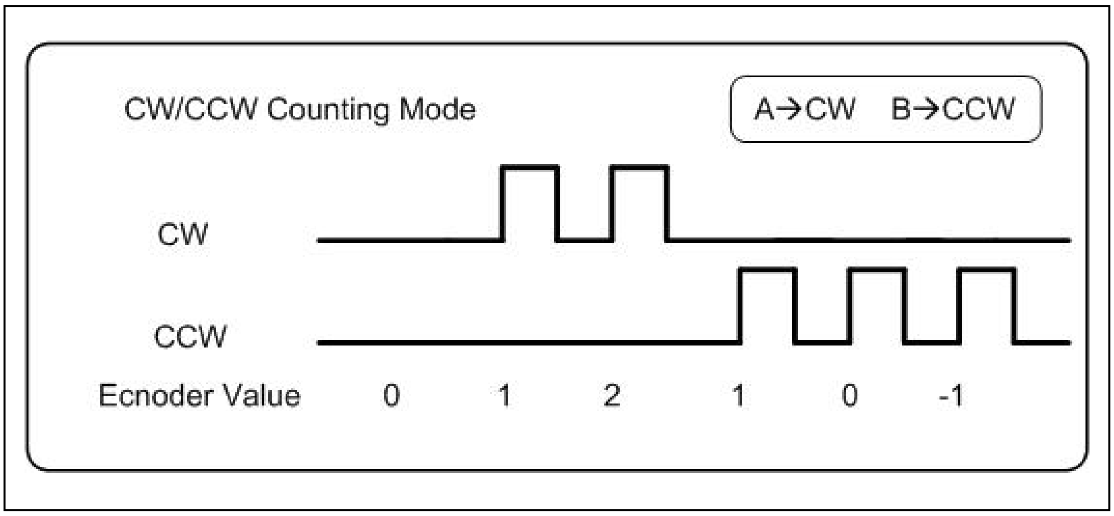
- There are two different type of encoders used in this configuration.
- Encoder1 is a 24V & differential type encoder
- Encoder2 is a 5V & single-ended type encoder
- XOR1 must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active low, the XOR2 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active high, the XOR2 bit must be set to 0 for internal active high logic

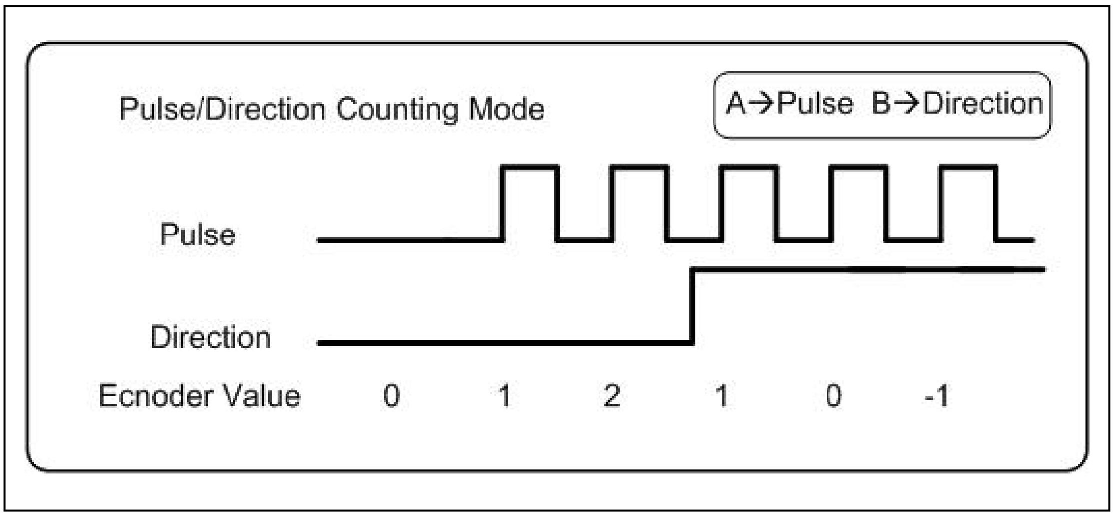


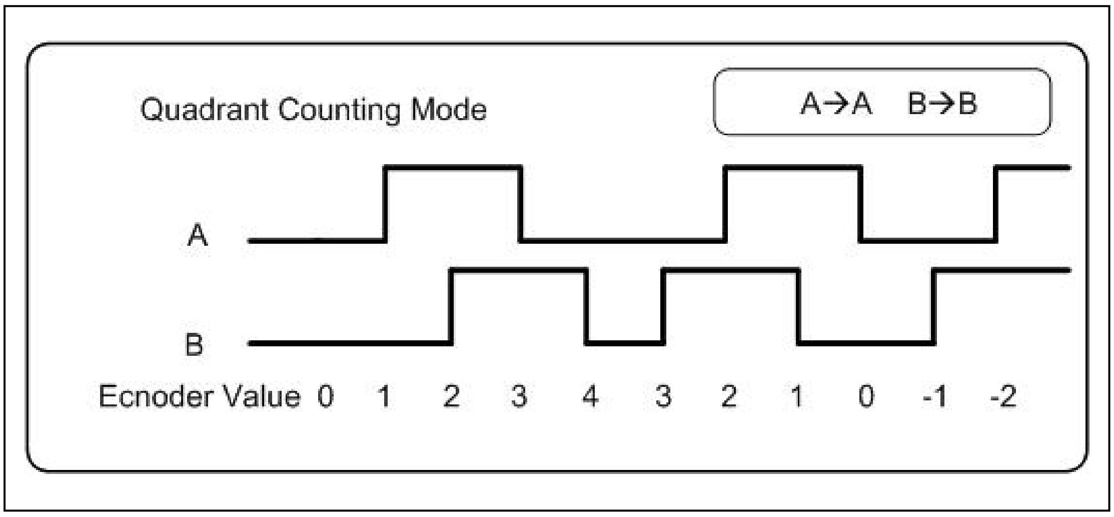
- There are three different type of encoders used in this configuration
- Encoder1 is a 24V & differential type encoder
- Encoder2 is a 12V & single-ended type encoder
- Encoder3 is a 5V & single-ended type encoder
- XOR1 must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active low, the XOR2 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder2 are active high, the XOR2 bit must be set to 0 for internal active high logic
- If A/B/Z signals of encoder3 are active low, the XOR3 bit must be set to 1 for internal active high logic
- If A/B/Z signals of encoder3 are active high, the XOR3 bit must be set to 0 for internal active high logic

## 1.5. Encoder Counting Mode

The internal counting logic is **active high**. Refer to Sec. 1.6 to use XOR control bits for active high logic. There are 3 different counting modes given as follows:



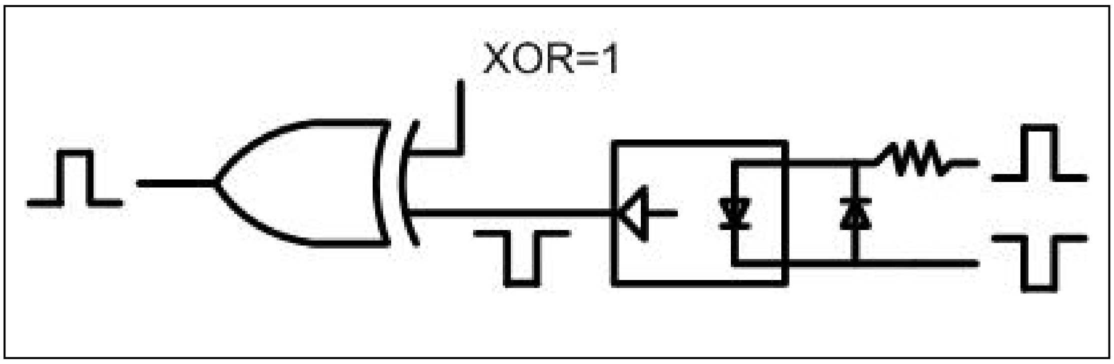




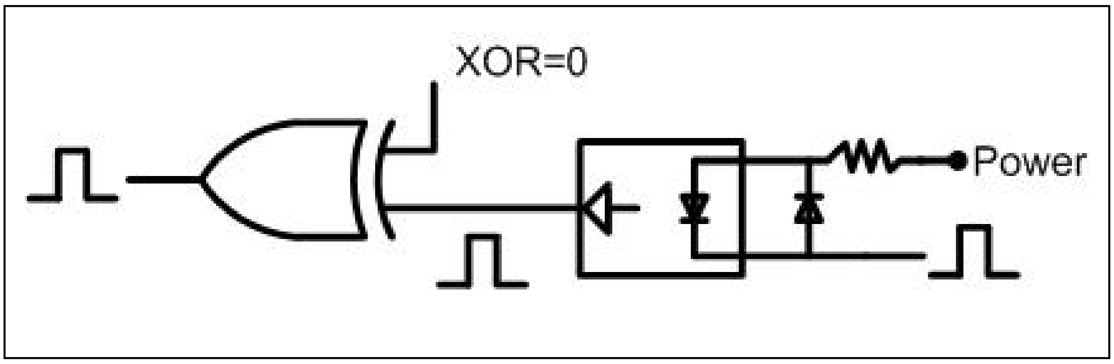
## 1.6. XOR Control

The expected waveform of internal logic is active high as given in Sec. 1.5. User can use XOR control bits to select the proper waveform as follows:

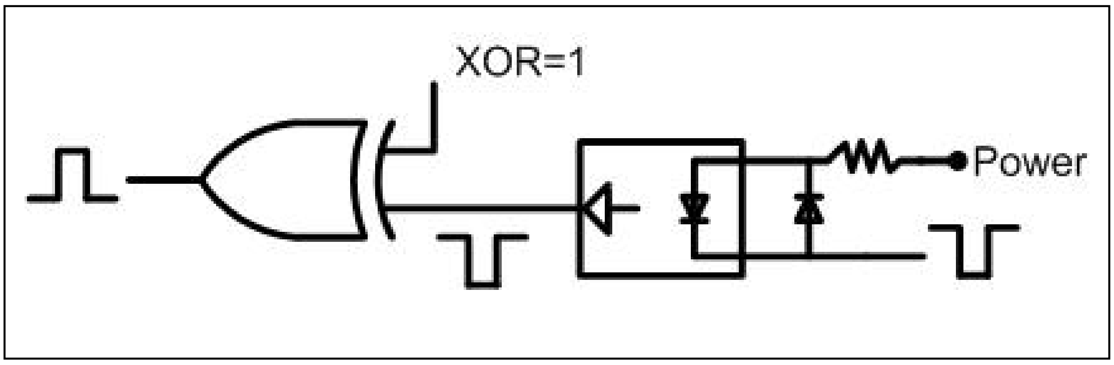
Case 1: differential input of X702, set XOR=1



Case 2: active high single-ended input of X702/X703, set XOR=0

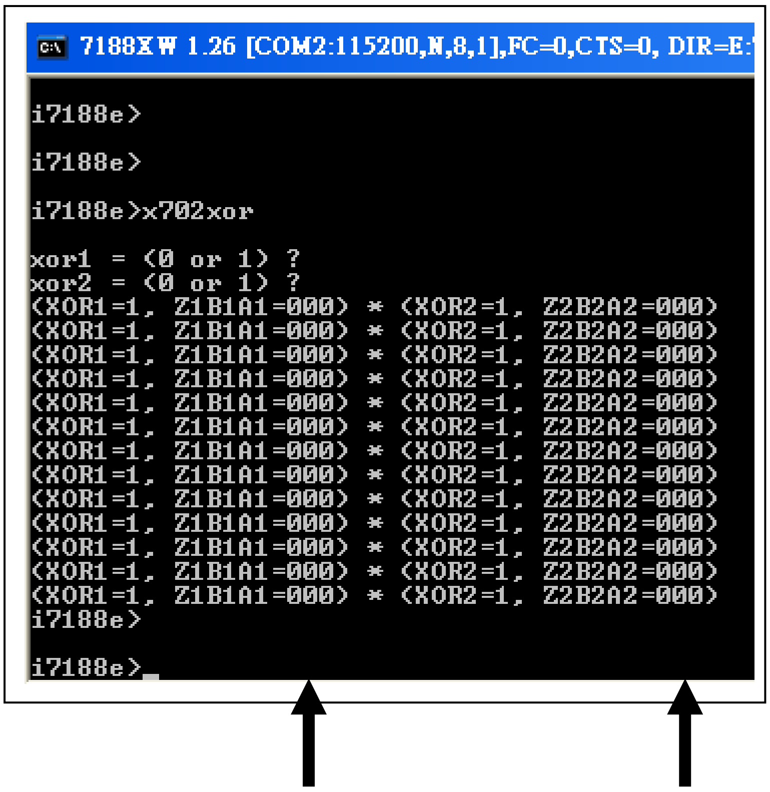


Case3: active low single-ended input of X702/X703, set XOR=1



The demo program, X702XOR.EXE, is design for X702 and X703XOR.EXE is designed for X703. User can run X702XOR.EXE or X703XOR.EXE first to check the initial value must be 0 as follows:

1. Run X702XOR.EXE
2. Press 1 ← User set XOR1=1
3. Press 1 ← User set XOR2=1
4. Press Any Key to Stop

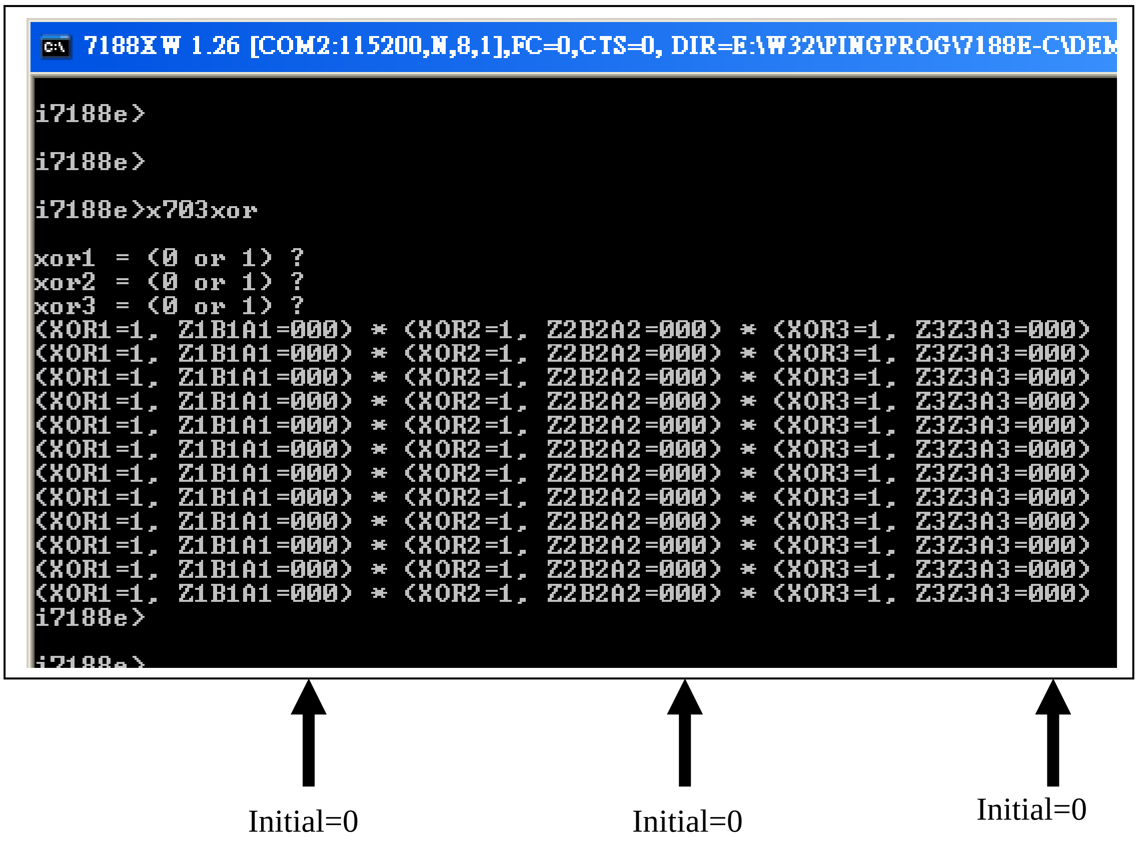


```
i7188e>

i7188e>

i7188e>x702xor

xor1 = (0 or 1) ?
xor2 = (0 or 1) ?
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
i7188e>

i7188e>_
```

Initial=0 Initial=0

Note: user can set XOR1/XOR2 to 0/0, 0/1, 1/0 or 1/1. The key point is → **all ABZ signals must be initial 0 & active high.**

1. Run X703XOR.EXE
2. Press 1 ← User set XOR1=1
3. Press 1 ← User set XOR2=1
4. Press 1 ← User set XOR3=1
5. Press Any Key to Stop



```
7188XW 1.26 [COM2:115200,N,8,1],FC=0,CTS=0, DIR=E:\W32\PINGPROG\7188E-C\DEM

i7188e>

i7188e>

i7188e>x703xor

xor1  = (0 or 1) ?
xor2  = (0 or 1) ?
xor3  = (0 or 1) ?
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
i7188e>

i7188e>
```

Note: user can set XOR1/XOR2/XOR3 to 0/0/0, 0/0/1, 0/1/0, 0/1/1,1/0/0,1/0/1,1/1/0 or 1/1/1. The key point is → **all ABZ signals must be initial 0 & active high.**

# 2. I/O Control Register

## 2.1. I/O Control Registers of X702

| I/O Address | Read | Write |
|---|---|---|
| 0 | Read bit7 ~ bit0 of encoder1 (Sec. 2.1.3) | Clear encoder1 (Sec. 2.1.3) |
| 1 | Read bit15 ~ bit8 of encoder1 (Sec. 2.1.3) **(latch by read 0)** | N/A |
| 2 | Read bit23 ~ bit16 of encoder1 (Sec. 2.1.3) **(no latch)** | Clear encoder2 (Sec. 2.1.4) |
| 3 | Read A/B/Z & encoder1 counting mode. (Sec. 2.1.2) | Set encoder1 counting mode (Sec. 2.1.1) |
| 4 | Read bit7 ~ bit0 of encoder2 (Sec. 2.1.4) | N/A |
| 5 | Read bit15 ~ bit8 of encoder2 (Sec. 2.1.4) **(latch by read 4)** | N/A |
| 6 | Read bit23 ~ bit16 of encoder2 (Sec. 2.1.4) **(no latch)**) | N/A |
| 7 | Read A/B/Z & encoder2 counting mode. (Sec. 2.1.2) | Set encoder2 counting mode (Sec. 2.1.1) |

### 2.1.1. Set Encoder Counting Mode

Write address=3

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | M1A | M1B | XOR1 |

Write address=7

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | M2A | M2B | XOR2 |

- (M1A,M1B) & XOR1 for encoder 1
- (M2A,M2B) & XOR2 for encoder 2
- XOR1=0 → select the normal logic of A1,B1,Z1
  XOR1=1 → select the inverse logic of A1,B1,Z1
- XOR2=0 → select normal logic of A2,B2,Z2
  XOR2=1 → select inverse logic of A2,B2,Z2
- (MxA, MxB)=(0,0) → stop mode, no counting
- (MxA, MxB)=(0,1) → select CW/CCW counting mode
- (MxA, MxB)=(1,0) → select Pulse/Direction counting mode
- (MxA, MxB)=(1,1) → select quadrant counting mode
- Refer to Sec. 1.5 for more information about CW/CCW, Pulse/Direction & quadrant counting mode
- Refer to Sec. 1.6 for more information about XOR1 & XOR2

## 2.1.2. Read A/B/Z & Encoder Counting Mode

Read address=3

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Z1 | B1 | A1 | M1A | M1B | XOR1 |

Read address=7

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Z2 | B2 | A2 | M2A | M2B | XOR2 |

- (Z1,B1,A1) is the logic status after XOR1, must be active high
- (Z2,B2,A2) is the logic status after XOR2, must be active high
- Refer to Sec. 2.1.1 for more information about MxA, MxB, XORx
- Refer to Sec. 1.5 for more information about CW/CCW, Pulse/Direction & quadrant counting mode
- Refer to Sec. 1.6 for more information about XOR1, XOR2 & XOR3

## 2.1.3. Encoder1 24-bit Value

Read address=0

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| ENA7 | ENA6 | ENA5 | ENA4 | ENA3 | ENA2 | ENA1 | ENA0 |

Read Address=1

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| ENA15 | ENA14 | ENA13 | ENA12 | ENA11 | ENA10 | ENA9 | ENA8 |

Read address=2

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| ENA23 | ENA22 | ENA21 | ENA20 | ENA19 | ENA18 | ENA17 | ENA16 |

- **24-bit encoder1 → [EnA23 ~ EnA0], EnA23 is MSB, EnA0 is LSB**

When program read address 0, the ENA15 ~ ENA8 will also be latched, but ENA23 ~ ENA16 will not be latched. The correct codes to read 24-bit encoder1 value are given as follows:

```
read_enc1(unsigned long *enc1)
{
unsigned int HighByte,MiddleByte,LowByte,Check;
HighByte=inportb(2);     // no latch
try_again:
LowByte=inportb(0);      // latch MiddleByte
MiddleByte=inportb(1);
Check=inportb(2);  // no latch
if (Check !=HighByte)    // HighByte is changed
    {
    HighByte=Check;
    goto try_again;
    }
(*enc1) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

- **Write any value to address 0 will clear EnA23 ~ EnA0 to 0 as follows:**

  ```
  Outportb(0,0);                //clear [EnA23 ~ EnA0] to 0
  ```

## 2.1.4. Encoder2 24-bit Value

Read address=4

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| ENB7 | ENB6 | ENB5 | ENB4 | ENB3 | ENB2 | ENB1 | ENB0 |

Read address=5

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| ENB15 | ENB14 | ENB13 | ENB12 | ENB11 | ENB10 | ENB9 | ENB8 |

Read address=6

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| ENB23 | ENB22 | ENB21 | ENB20 | ENB19 | ENB18 | ENB17 | ENB16 |

- **24-bit encoder2 → [EnB23 ~ EnB0], EnB23 is MSB, EnB0 is LSB**

When program read address 4, the ENB15 ~ ENB8 will also be latched, but ENB23 ~ ENB16 will not be latched. The correct codes to read 24-bit encoder2 value are given as follows:

```
read_enc2(unsigned long *enc2)
{
unsigned int HighByte,MiddleByte,LowByte,Check;
HighByte=inportb(6);     // no latch
try_again2:
LowByte=inportb(4);      // latch MiddleByte
MiddleByte=inportb(5);
Check=inportb(6);  // no latch
if (Check !=HighByte)    // HighByte is changed
    {
    HighByte=Check;
    goto try_again2;
    }
(*enc2) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

- **Write any value to address 4 will clear EnB23 ~ EnB0 to 0 as follows:**

```
Outportb(0,0);                //clear [EnB23 ~ EnB0] to 0
```

## 2.2. I/O Control Registers of X703

| I/O Address | Read | Write |
|---|---|---|
| 0 | Read bit7 ~ bit0 of encoder1 (Sec. 2.2.3) | Clear encoder1 (Sec. 2.2.3) |
| 1 | Read bit15 ~ bit8 of encoder1 (Sec. 2.2.3), **(latch by read 0)** | N/A |
| 2 | Read bit23 ~ bit16 of encoder1 (Sec. 2.2.3), **(latch by read 0)** | N/A |
| 3 | Read A/B/Z & encoder1 counting mode (Sec 2.2.2) | Set encoder1 counting mode (Sec. 2.2.1) |
| 4 | Read bit7 ~ bit0 of encoder2 (Sec. 2.2.4) | Clear encoder2 (Sec. 2.2.4) |
| 5 | Read bit15 ~ bit8 of encoder2 (Sec. 2.2.4), **(latch by read 4)** | N/A |
| 6 | Read bit23 ~ bit16 of encoder2 (Sec. 2.2.4), **(latch by read 4)** | N/A |
| 7 | Read A/B/Z & encoder2 counting mode (Sec 2.2.2) | Set encoder2 counting m mode (Sec. 2.2.1) |
| 8 | Read bit7 ~ bit0 of encoder3 (Sec. 2.2.5) | Clear encoder3 (Sec. 2.2.5) |
| 9 | Read bit15 ~ bit8 of encoder3 (Sec. 2.2.5), **(latch by read 8)** | N/A |
| 10 | Read bit23 ~ bit16 of encoder3 (Sec. 2.2.5), **(latch by read 8)** | N/A |
| 11 | Read A/B/Z & encoder3 counting mode (Sec 2.2.2) | Set encoder3 counting mode (Sec. 2.2.1) |
| 12 | N/A | N/A |
| 13 | N/A | N/A |
| 14 | N/A | N/A |
| 15 | N/A | N/A |

## 2.2.1. Set Encoder Counting Mode

Write address=3

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | M1A | M1B | XOR1 |

Write address=7

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | M2A | M2B | XOR2 |

Write address=11

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | M3A | M3B | XOR3 |

- (M1A,M1B) & XOR1 for encoder 1
- (M2A,M2B) & XOR2 for encoder 2
- (M2A,M2B) & XOR2 for encoder 2
- XOR1=0 → select the normal logic of A1,B1,Z1
  XOR1=1 → select the inverse logic of A1,B1,Z1
- XOR2=0 → select normal logic of A2,B2,Z2
  XOR2=1 → select inverse logic of A2,B2,Z2
- XOR3=0 → select normal logic of A3,B3,Z3
  XOR3=1 → select inverse logic of A3,B3,Z3
- (MxA, MxB)=(0,0) → stop mode, no counting
- (MxA, MxB)=(0,1) → select CW/CCW counting mode
- (MxA, MxB)=(1,0) → select Pulse/Direction counting mode
- (MxA, MxB)=(1,1) → select quadrant counting mode
- Refer to Sec. 1.5 for more information about CW/CCW, Pulse/Direction & quadrant counting mode
- Refer to Sec. 1.6 for more information about XOR1, XOR2 & XOR3

## 2.2.2. Read A/B/Z & Encoder Counting Mode

Read address=3

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Z1 | B1 | A1 | M1A | M1B | XOR1 |

Read address=7

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Z2 | B2 | A2 | M2A | M2B | XOR2 |

Read address=11

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Z3 | B3 | A3 | M3A | M3B | XOR3 |

- (Z1,B1,A1) is the logic status after XOR1, must be active high
- (Z2,B2,A2) is the logic status after XOR2, must be active high
- (Z3,B3,A3) is the logic status after XOR3, must be active high
- Refer to Sec. 2.2.1 for more information about MxA, MxB, XORx
- Refer to Sec. 1.5 for more information about CW/CCW, Pulse/Direction & quadrant counting mode
- Refer to Sec. 1.6 for more information about XOR1, XOR2 & XOR3

## 2.2.3. Encoder1 24–bit Value

Read address=0

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnA7 | EnA6 | EnA5 | EnA4 | EnA3 | EnA2 | EnA1 | EnA0 |

Read Address=1

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnA15 | EnA14 | EnA13 | EnA12 | EnA11 | EnA10 | EnA9 | EnA8 |

Read address=2

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnA23 | EnA22 | EnA21 | EnA20 | EnA19 | EnA18 | EnA17 | EnA16 |

- **24-bit encoder1 → [EnA23 ~ EnA0], EnA23 is MSB, EnA0 is LSB**

When program read address 0, the EnA23 ~ EnA8 will also be latched. The correct codes to read 24-bit encoder1 value are given as follows:

```
read_enc1(unsigned long *enc1)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

LowByte=inportb(0);     // latch MiddleByte & HighByte
MiddleByte=inportb(1);
HighByte=inportb(2);
(*enc1) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

- **Write any value to address 0 will clear EnA23 ~ EnA0 to 0 as follows:**
  ```
  Outportb(0,0);                    //clear [EnA23 ~ EnA0] to 0
  ```

## 2.2.4. Encoder2 24–bit Value

Read address=4

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnB7 | EnB6 | EnB5 | EnB4 | EnB3 | EnB2 | EnB1 | EnB0 |

Read Address=5

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnB15 | EnB14 | EnB13 | EnB12 | EnB11 | EnB10 | EnB9 | EnB8 |

Read address=6

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnB23 | EnB22 | EnB21 | EnB20 | EnB19 | EnB18 | EnB17 | EnB16 |

- **24-bit encoder2 → [EnB23 ~ EnB0], EnB23 is MSB, EnB0 is LSB**

When program read address 4, the EnB23 ~ EnB8 will also be latched. The correct codes to read 24-bit encoder1 value are given as follows:

```
read_enc2(unsigned long *enc2)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

LowByte=inportb(4);      // latch MiddleByte & HighByte
MiddleByte=inportb(5);
HighByte=inportb(6);
(*enc1) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

- **Write any value to address 4 will clear EnB23 ~ EnB0 to 0 as follows:**
  Outportb(4,0);                    //clear [EnB23 ~ EnB0] to 0

## 2.2.5. Encoder3 24–bit Value

Read address=8

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnC7 | EnC6 | EnC5 | EnC4 | EnC3 | EnC2 | EnC1 | EnC0 |

Read Address=9++

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnC15 | EnC14 | EnC13 | EnC12 | EnC11 | EnC10 | EnC9 | EnC8 |

Read address=10

| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|
| EnC23 | EnC22 | EnC21 | EnC20 | EnC19 | EnC18 | EnC17 | EnC16 |

- **24-bit encoder3 → [EnC23 ~ EnC0], EnC23 is MSB, EnC0 is LSB**

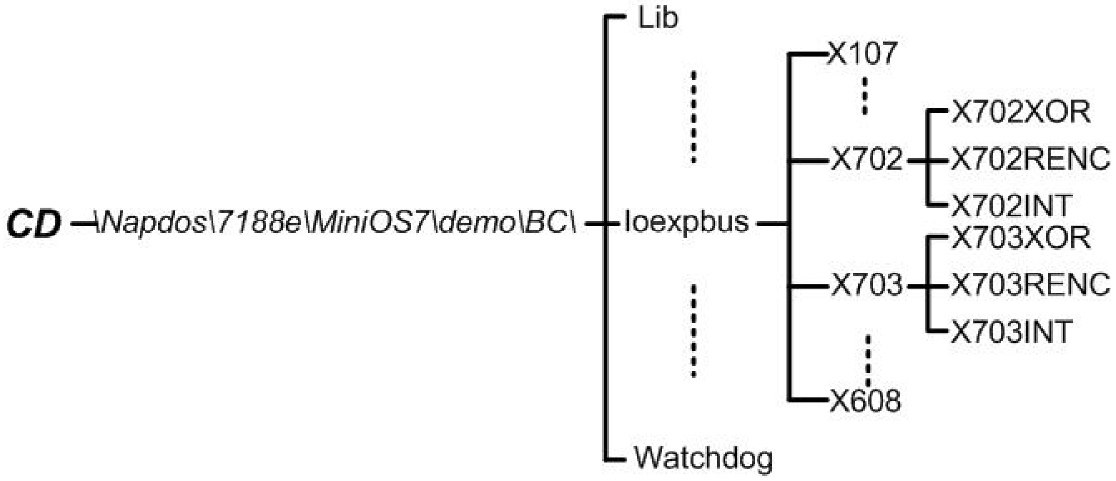
When program read address 8, the EnC23 ~ EnC8 will also be latched. The correct codes to read 24-bit encoder1 value are given as follows:

```
read_enc3(unsigned long *enc3)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

LowByte=inportb(8);     // latch MiddleByte & HighByte
MiddleByte=inportb(9);
HighByte=inportb(10);
(*enc3) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

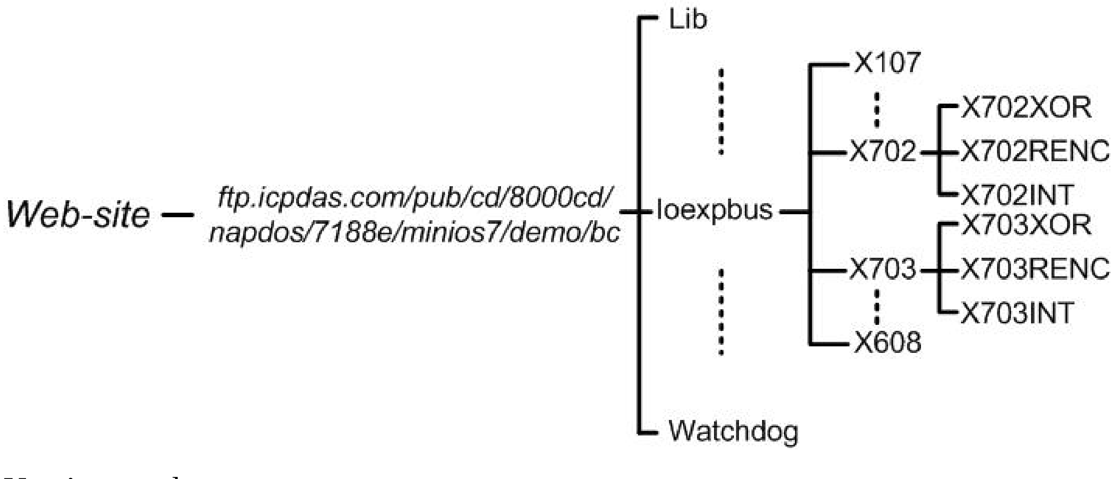
- **Write any value to address 8 will clear EnC23 ~ EnC0 to 0 as follows:**
  Outportb(8,0);                    //clear [EnC23 ~ EnC0] to 0

# 3. Software Examples

The demo program of X702/X703 are given in the companion CD as follows:



User’s manual : CD\Napdos\7188x\manual\hardware\x702x703.pdf
User can download these demo programs from our web site as follows:



User’smanual : **ftp.icpdas.com/pub/cd/8000cd/napdos/7188x/hardware/X702X703.pdf**

All demo program are coded as follows:

- Using BC, Large model
- Using 7188EL.LIB
- Main program → MAIN.C
- One project file for BC
- One execution file

## 3.1. X702 Demo Program

### 3.1.1. Read A/B/Z Demo, X702XOR.EXE

- Main program → MAIN.C
- Project file → X702XOR.PRJ

```
#include <stdio.h>
#include <stdlib.h>
#include "..\..\lib\7188E.h"
int xor1,xor2;
main()
{
Print("\nxor1 = (0 or 1) ?");
xor1=Getch()-'0'; if (xor1) xor1=1;
Print("\nxor2 = (0 or 1) ?");
xor2=Getch()-'0'; if (xor2) xor2=1;

outportb(3,xor1); // set xor1 & counting mode1=stop
outportb(7,xor2); // set xor2 & counting mode2=stop
```

**Set Xor & Counting Mode**

```
for (;;)
{
if (Kbhit())
    {
    Getch(); return;
    }
show_abz();
DelayMs(100);
}
}
show_abz()
{
int zba1,zba2,i;
```

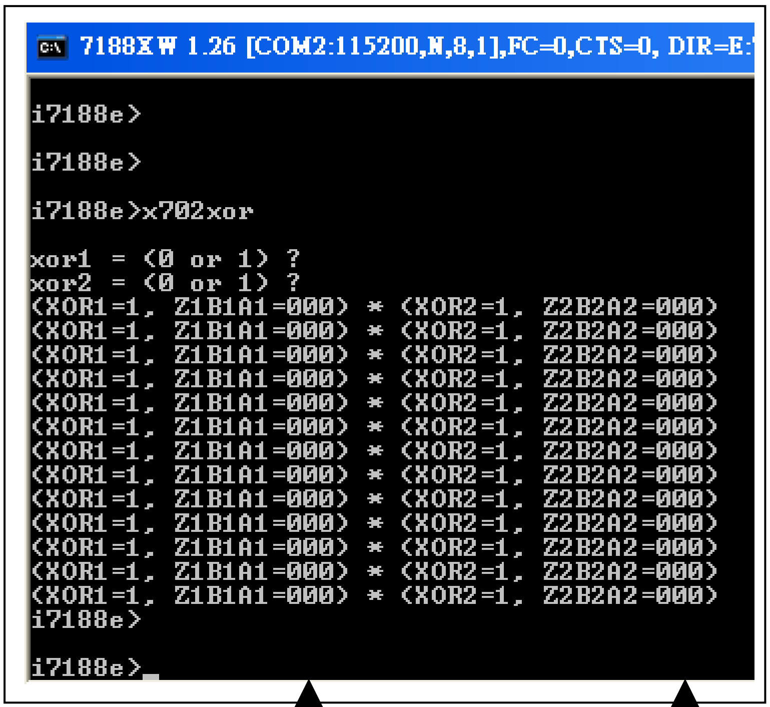
**Read ABZ status**

```
zba1=inport(3); // read ABZ status & counting mode of encoder1
zba2=inport(7); // read ABZ status & counting mode of encoder2
```

```
Print("\n(XOR1=%d, Z1B1A1=",xor1);
for (i=0; i<3; i++)
        {
        if (zba1&0x20) Print("1"); else Print("0");
        zba1 = (zba1<<1);
        }

Print(") * (XOR2=%d, Z2B2A2=",xor2);
for (i=0; i<3; i++)
        {
        if (zba2&0x20) Print("1"); else Print("0");
        zba2 = (zba2<<1);
        }
Print(")");
}
```

**Show ABZ status**

1. Run X702XOR.EXE
2. Press 1 ← User set XOR1=1
3. Press 1 ← User set XOR2=1
4. Press Any Key to Stop



```
7188XW 1.26 [COM2:115200,N,8,1],FC=0,CTS=0, DIR=E:

i7188e>

i7188e>

i7188e>x702xor

xor1 = (0 or 1) ?
xor2 = (0 or 1) ?
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000)
i7188e>

i7188e>_
```

Initial=0 Initial=0

## 3.1.2. Read Encoder Demo, X702RENC.EXE

- Main program → MAIN.C
- Project file → X702RENC.PRJ

```
#include <stdio.h>
#include <stdlib.h>
#include "..\..\lib\7188E.h"
int xor1,xor2,mode1,mode2;
unsigned long EncVal1,EncVal2;
main()
{
Print("\nxor1 = (0 or 1) ?");
xor1=Getch()-'0'; if (xor1) xor1=1;
try_again1:
Print("\nmode1 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?");
mode1=Getch();
switch(mode1)
    {
    case '0' : break;                   // stop
    case '1' : xor1+=0x02; break;       // cw/ccw
    case '2' : xor1+=0x04; break;       // pulse/dir
    case '3' : xor1+=0x06; break;       // a/b
    default   : goto try_again1;
    }
Print("\nxor2 = (0 or 1) ?");
xor2=Getch()-'0'; if (xor2) xor2=1;
try_again2:
Print("\nmode2 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?");
mode2=Getch();
switch(mode2)
    {
    case '0' : break;                   // stop
    case '1' : xor2+=0x02; break;       // cw/ccw
    case '2' : xor2+=0x04; break;       // pulse/dir
    case '3' : xor2+=0x06; break;       // a/b
    default   : goto try_again2;
    }
```

**Set XOR1 & Counting Mode1**

**Set XOR2 & Counting Mode2**

```
outportb(3,xor1); // set xor1 & counting mode1
outportb(7,xor2); // set xor2 & counting mode2
outportb(0,0);     // clear encoder1 to 0
outportb(4,0);     // clear encoder2 to 0
```

**Clear encoder1 & encoder2**

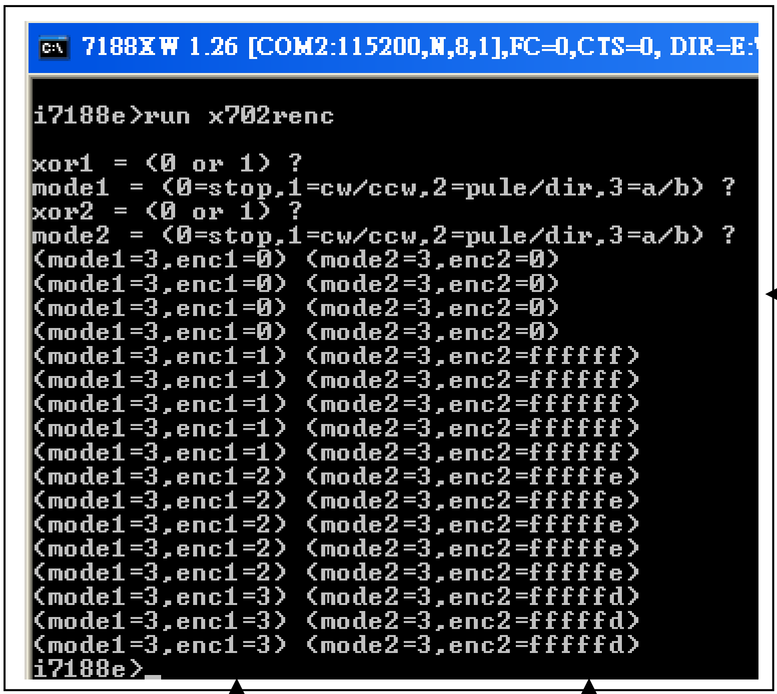
```
for (;;)
{
if (Kbhit())
    {
    Getch(); return;
    }

read_enc1(&EncVal1);
read_enc2(&EncVal2);
Print("\n(mode1=%x,enc1=%lx)
(mode2=%x,enc2=%lx)",xor1,EncVal1,xor2,EncVal2);
DelayMs(100);
}
}

read_enc1(unsigned long *enc1)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

HighByte=inportb(2);     // no latch
try_again:
LowByte=inportb(0);      // latch MiddleByte
MiddleByte=inportb(1);
Check=inportb(2);  // no latch
if (Check !=HighByte)    // HighByte is changed
    {
    HighByte=Check;
    goto try_again;
    }
(*enc1) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

**Get 24-bit encoder1**

```
read_enc2(unsigned long *enc2)
{
unsigned int HighByte,MiddleByte,LowByte,Check;
HighByte=inportb(6);     // no latch
try_again:
LowByte=inportb(4);      // latch MiddleByte
MiddleByte=inportb(5);
Check=inportb(6);  // no latch
if (Check !=HighByte)    // HighByte is changed
      {
      HighByte=Check;
      goto try_again;
      }
(*enc2) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

**Get 24-bit encoder2**

1. Run X702RENC.EXE
2. Press 1 — User set XOR1=1
3. Press 1 — User set counting mode1 = cw/ccw
4. Press 1 — User set XOR2=1
5. Press 1 — User set counting mode2 = cw/ccw
6. Press Any Key to Stop



```
i7188e>run x702renc

xor1  = (0 or 1) ?
mode1  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
xor2  = (0 or 1) ?
mode2  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
(mode1=3,enc1=0) (mode2=3,enc2=0)
(mode1=3,enc1=0) (mode2=3,enc2=0)
(mode1=3,enc1=0) (mode2=3,enc2=0)
(mode1=3,enc1=0) (mode2=3,enc2=0)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)
(mode1=3,enc1=3) (mode2=3,enc2=fffffd)
(mode1=3,enc1=3) (mode2=3,enc2=fffffd)
(mode1=3,enc1=3) (mode2=3,enc2=fffffd)
i7188e>_
```

Initial count=0

Up counting        Down counting

## 3.1.3. Z Interrupt Demo, X702INT.EXE

- Main program → MAIN.C
- Interrupt service routine → INT.C
- Project file → X702INT.PRJ
- Z1 → to int0 of 7188Xb or 7188EX
- Z2 → to int1 of 7188XB or 7188EX

The main.c is given as follows:

```
=============================================================
#include <stdio.h>
#include <stdlib.h>
#include "..\..\lib\7188E.h"

int xor1,xor2,mode1,mode2;
unsigned long EncVal1,EncVal2;
extern unsigned long EncLatch1,EncLatch2;//defined in int.c

main()
{
Print("\nxor1 = (0 or 1) ?");
xor1=Getch()-'0'; if (xor1) xor1=1;
try_again1:
Print("\nmode1 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?");
mode1=Getch();
switch(mode1)
    {
    case '0' : break;                    // stop
    case '1' : xor1+=0x02; break;        // cw/ccw
    case '2' : xor1+=0x04; break;        // pulse/dir
    case '3' : xor1+=0x06; break;        // a/b
    default  : goto try_again1;
    }
```

```
Print("\nxor2 = (0 or 1) ?");
xor2=Getch()-'0'; if (xor2) xor2=1;
try_again2:
Print("\nmode2 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?");
mode2=Getch();
switch(mode2)
    {
    case '0' : break;                   // stop
    case '1' : xor2+=0x02; break;     // cw/ccw
    case '2' : xor2+=0x04; break;     // pulse/dir
    case '3' : xor2+=0x06; break;     // a/b
    default   : goto try_again2;
    }

outportb(3,xor1); // set xor1 & counting mode1
outportb(7,xor2); // set xor2 & counting mode2
outportb(0,0);     // clear encoder1 to 0
outportb(4,0);     // clear encoder2 to 0

InstallInt0Isr();
InstallInt1Isr();
```

Install interrupt service routine for Z1 & Z2

```
for (;;)
{
if (Kbhit())
   {
   RestoreInt0Isr();
   RestoreInt1Isr();
   Getch(); return;
   }

read_enc1(&EncVal1);
read_enc2(&EncVal2);
Print("\n(mode1=%x,enc1=%lx)mode2=%x,enc2=%lx)"
     ,xor1,EncVal1,xor2,EncVal2);
Print("(Latch1=%lx,Latch2=%lx)",EncLatch1,EncLatch2);
DelayMs(100);
}
}
```

Show values latched by Z1 & Z2

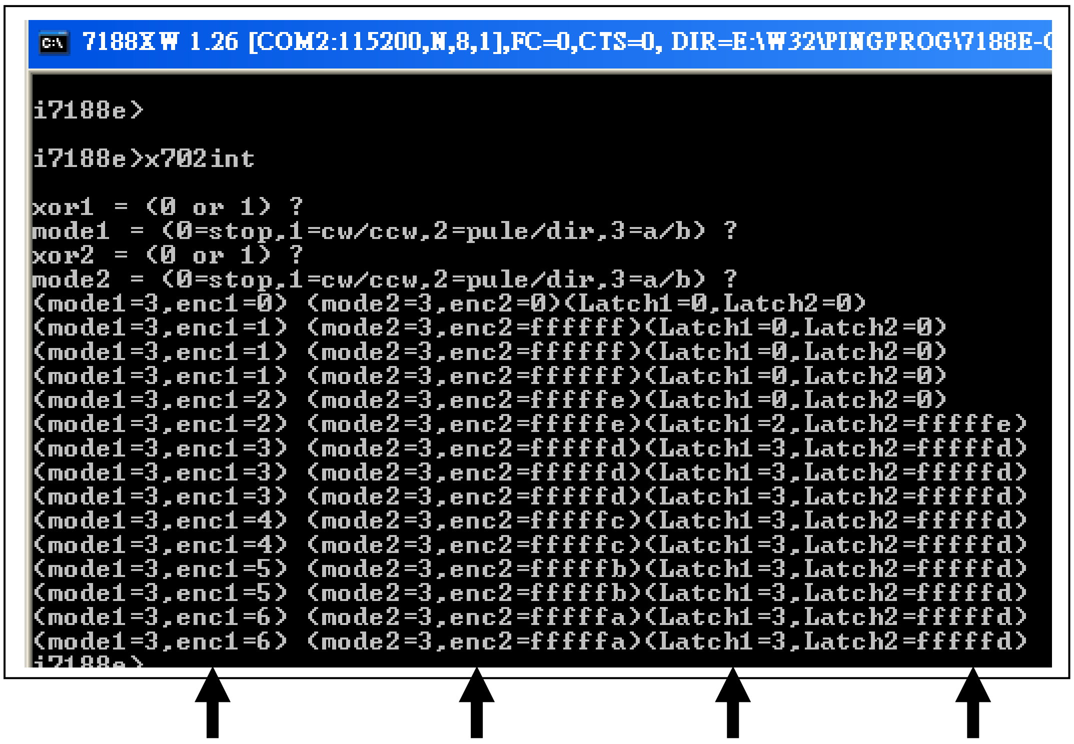
```
read_enc1(unsigned long *enc1)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

HighByte=inportb(2);     // no latch
try_again:
LowByte=inportb(0);      // latch MiddleByte
MiddleByte=inportb(1);
Check=inportb(2);  // no latch
if (Check !=HighByte)    // HighByte is changed
     {
     HighByte=Check;
     goto try_again;
     }

(*enc1) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}

read_enc2(unsigned long *enc2)
{
unsigned int HighByte,MiddleByte,LowByte,Check;
unsigned long enc;

HighByte=inportb(6);     // no latch
try_again:
LowByte=inportb(4);      // latch MiddleByte
MiddleByte=inportb(5);
Check=inportb(6);  // no latch
if (Check !=HighByte)    // HighByte is changed
     {
     HighByte=Check;
     goto try_again;
     }

(*enc2) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

The int.c is given as follows:

```
============================================================
#include <stdio.h>
#include <stdlib.h>
#include "..\..\lib\7188e.h"

#define EOITYPE_INT0          0x0c
#define EOITYPE_INT1          0x0d
#define INT_EOI               0xff22   // End-of-interrupt register
#define INT_MASK              0xff28   // Interrupt mask register

unsigned long EncLatch1,EncLatch2;
//-----------------------------------------
unsigned long OldIntVect0=0,OldIntVect1=0,OldIntVect4=0;
unsigned long far *IntVect=(unsigned long far *)0L;
void interrupt Int0Isr(void)
{
   read_enc1(&EncLatch1);                    Encoder1 latched by Z1
   outpw(INT_EOI,EOITYPE_INT0);
}
void InstallInt0Isr(void)
{
   OldIntVect0=IntVect[0x0C];             // save old ISR
   IntVect[0x0C]=(unsigned long)Int0Isr;   // install new ISR
   outpw(INT_MASK, inpw(INT_MASK)&0xffef);// enable INT-0
}
void RestoreInt0Isr(void)
{
   if(OldIntVect0){
        IntVect[0x0C]=OldIntVect0;          // restore OLD ISR
        outpw(INT_MASK,inpw(INT_MASK)|0x0010);// disable INT-0
   }
}
void interrupt Int1Isr(void)
{
   read_enc2(&EncLatch2);                    Encoder2 latched by Z2
   outpw(INT_EOI,EOITYPE_INT1);
}
```

```
void InstallInt1Isr(void)
{
    OldIntVect1=IntVect[0x0D];              // save old ISR
    IntVect[0x0D]=(unsigned long)Int1Isr;   // install new ISR
    outpw(INT_MASK, inpw(INT_MASK)&0xffdf);// enable INT-1
}

void RestoreInt1Isr(void)
{
    if(OldIntVect1){
        IntVect[0x0D]=OldIntVect1;            //   restore OLD ISR
        outpw(INT_MASK,inpw(INT_MASK)|0x0020);//   disable INT-1
    }
}
```

1. Run X702INT.EXE
2. Press 1 ← User set XOR1=1
3. Press 1 ← User set counting mode1 = cw/ccw
4. Press 1 ← User set XOR2=1
5. Press 1 ← User set counting mode2 = cw/ccw
6. Press Any Key to Stop



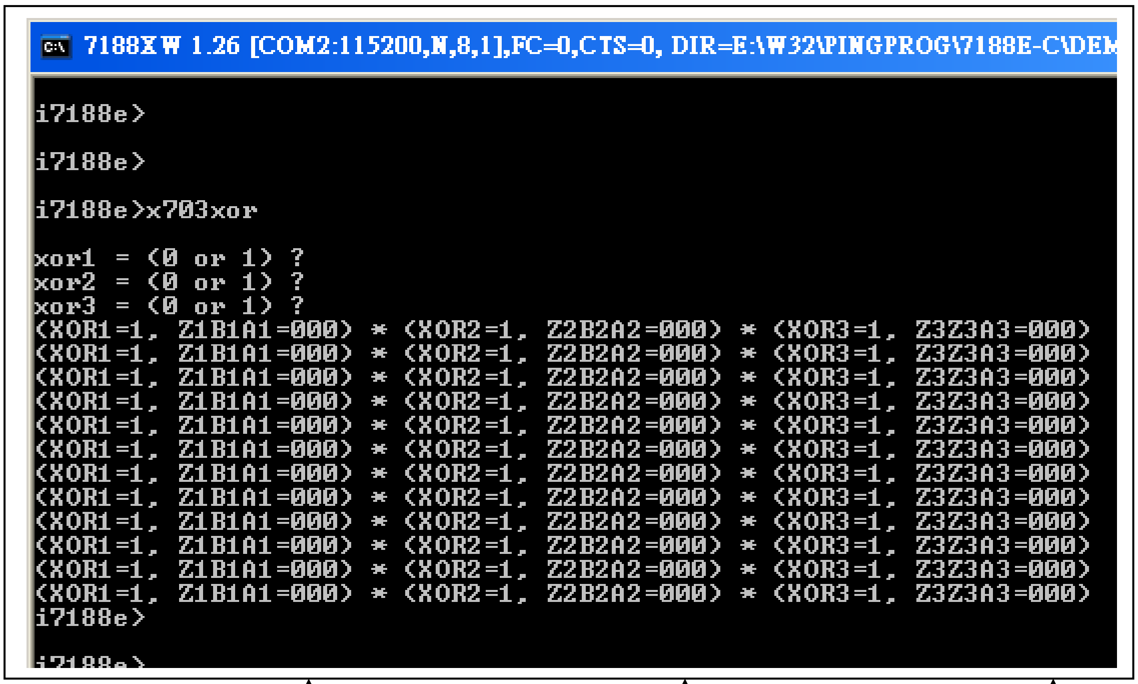
```
7188XW 1.26 [COM2:115200,N,8,1],FC=0,CTS=0, DIR=E:\W32\PINGPROG\7188E-(

i7188e>

i7188e>x702int

xor1  = (0 or 1) ?
mode1  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
xor2  = (0 or 1) ?
mode2  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
(mode1=3,enc1=0) (mode2=3,enc2=0)(Latch1=0,Latch2=0)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)(Latch1=0,Latch2=0)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)(Latch1=0,Latch2=0)
(mode1=3,enc1=1) (mode2=3,enc2=ffffff)(Latch1=0,Latch2=0)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)(Latch1=0,Latch2=0)
(mode1=3,enc1=2) (mode2=3,enc2=fffffe)(Latch1=2,Latch2=fffffe)
(mode1=3,enc1=3) (mode2=3,enc2=fffffd)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=3) (mode2=3,enc2=fffffd)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=3) (mode2=3,enc2=fffffd)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=4) (mode2=3,enc2=fffffc)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=4) (mode2=3,enc2=fffffc)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=5) (mode2=3,enc2=fffffb)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=5) (mode2=3,enc2=fffffb)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=6) (mode2=3,enc2=fffffa)(Latch1=3,Latch2=fffffd)
(mode1=3,enc1=6) (mode2=3,enc2=fffffa)(Latch1=3,Latch2=fffffd)
i7188e>
```

## 3.2. X703 Demo Program

### 3.2.1. Read A/B/Z Demo, X703XOR.EXE

- Main program → MAIN.C
- Project file → X702XOR.PRJ
- MAIN.C is similar to Sec. 3.1.1.

1. Run X703XOR.EXE
2. Press 1 ←
3. Press 1 ← User set XOR2=1
4. Press 1 ← User set XOR3=1
5. Press Any Key to Stop



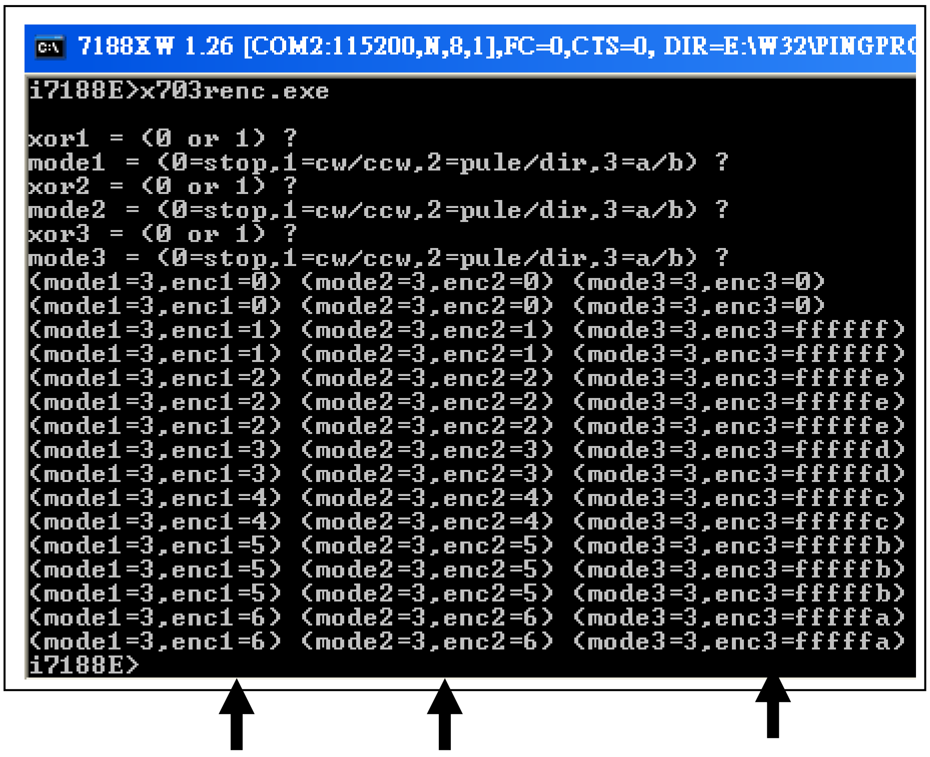
```
7188XW 1.26 [COM2:115200,N,8,1],FC=0,CTS=0, DIR=E:\W32\PINGPROG\7188E-C\DEM

i7188e>

i7188e>

i7188e>x703xor

xor1  = (0 or 1) ?
xor2  = (0 or 1) ?
xor3  = (0 or 1) ?
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
(XOR1=1, Z1B1A1=000) * (XOR2=1, Z2B2A2=000) * (XOR3=1, Z3Z3A3=000)
i7188e>

i7188e>
```

↑ Initial=0 ↑ Initial=0 ↑ Initial=0

## 3.2.2. Read Encoder Demo, X703RENC.EXE

- Main program → MAIN.C
- Project file → X702RENC.PRJ
- MAIN.C is similar to Sec. 3.1.2.

1. Run X703RENC.EXE
2. Press 1 ← User set XOR1=1
3. Press 1 ← User set counting mode1 = cw/ccw
4. Press 1 ← User set XOR2=1
5. Press 1 ← User set counting mode2 = cw/ccw
6. Press 1 ← User set XOR3=1
7. Press 1 ← User set counting mode3 = cw/ccw
8. Press Any Key to Stop



```
7188XW 1.26 [COM2:115200,N,8,1],FC=0,CTS=0, DIR=E:\W32\PINGPR
i7188E>x703renc.exe

xor1 = (0 or 1) ?
mode1 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
xor2 = (0 or 1) ?
mode2 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
xor3 = (0 or 1) ?
mode3 = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
(mode1=3,enc1=0) (mode2=3,enc2=0) (mode3=3,enc3=0)
(mode1=3,enc1=0) (mode2=3,enc2=0) (mode3=3,enc3=0)
(mode1=3,enc1=1) (mode2=3,enc2=1) (mode3=3,enc3=ffffff)
(mode1=3,enc1=1) (mode2=3,enc2=1) (mode3=3,enc3=ffffff)
(mode1=3,enc1=2) (mode2=3,enc2=2) (mode3=3,enc3=fffffe)
(mode1=3,enc1=2) (mode2=3,enc2=2) (mode3=3,enc3=fffffe)
(mode1=3,enc1=2) (mode2=3,enc2=2) (mode3=3,enc3=fffffe)
(mode1=3,enc1=3) (mode2=3,enc2=3) (mode3=3,enc3=fffffd)
(mode1=3,enc1=3) (mode2=3,enc2=3) (mode3=3,enc3=fffffd)
(mode1=3,enc1=4) (mode2=3,enc2=4) (mode3=3,enc3=fffffc)
(mode1=3,enc1=4) (mode2=3,enc2=4) (mode3=3,enc3=fffffc)
(mode1=3,enc1=5) (mode2=3,enc2=5) (mode3=3,enc3=fffffb)
(mode1=3,enc1=5) (mode2=3,enc2=5) (mode3=3,enc3=fffffb)
(mode1=3,enc1=5) (mode2=3,enc2=5) (mode3=3,enc3=fffffb)
(mode1=3,enc1=6) (mode2=3,enc2=6) (mode3=3,enc3=fffffa)
(mode1=3,enc1=6) (mode2=3,enc2=6) (mode3=3,enc3=fffffa)
i7188E>
```

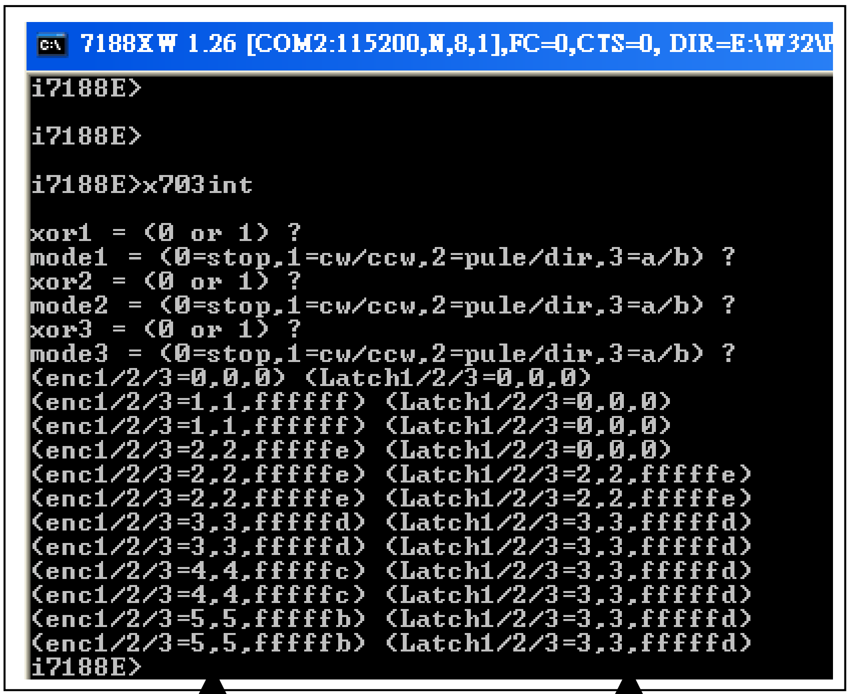
Read to 0/4/8 will latch the high bytes of encoder 1/2/3 as follows:

```
read_enc1(unsigned long *enc1)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

LowByte=inportb(0);     // latch MiddleByte & HighByte
MiddleByte=inportb(1);
HighByte=inportb(2);
(*enc1) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}

read_enc2(unsigned long *enc2)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

LowByte=inportb(4);     // latch MiddleByte & HighByte
MiddleByte=inportb(5);
HighByte=inportb(6);
(*enc2) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}

read_enc3(unsigned long *enc3)
{
unsigned int HighByte,MiddleByte,LowByte,Check;

LowByte=inportb(8);     // latch MiddleByte & HighByte
MiddleByte=inportb(9);
HighByte=inportb(10);
(*enc3) = (((long)HighByte)<<16) + (MiddleByte<<8) + LowByte;
}
```

## 3.2.3. Z Interrupt Demo, X703INT.EXE

- Main program → MAIN.C
- Interrupt service routine → INT.C
- Project file → X703INT.PRJ
- MAIN.C & INT.C are similar to Sec. 3.1.3.
- Z1 → to int0 of 7188Xb or 7188EX
- Z2 → to int1 of 7188XB or 7188EX
- Z3 → to int4 of 7188XB or 7188EX

1. Run X703INT.EXE
2. Press 1 ← User set XOR1=1
3. Press 1 ← User set counting mode1 = cw/ccw
4. Press 1 ← User set XOR2=1
5. Press 1 ← User set counting mode2 = cw/ccw
6. Press 1 ← User set XOR3=1
7. Press 1 ← User set counting mode3 = cw/ccw
8. Press Any Key to Stop



```
7188XW 1.26 [COM2:115200,N,8,1],FC=0,CTS=0, DIR=E:\W32\F
i7188E>

i7188E>

i7188E>x703int

xor1  = (0 or 1) ?
mode1  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
xor2  = (0 or 1) ?
mode2  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
xor3  = (0 or 1) ?
mode3  = (0=stop,1=cw/ccw,2=pule/dir,3=a/b) ?
(enc1/2/3=0,0,0) (Latch1/2/3=0,0,0)
(enc1/2/3=1,1,ffffff) (Latch1/2/3=0,0,0)
(enc1/2/3=1,1,ffffff) (Latch1/2/3=0,0,0)
(enc1/2/3=2,2,fffffe) (Latch1/2/3=0,0,0)
(enc1/2/3=2,2,fffffe) (Latch1/2/3=2,2,fffffe)
(enc1/2/3=2,2,fffffe) (Latch1/2/3=2,2,fffffe)
(enc1/2/3=3,3,fffffd) (Latch1/2/3=3,3,fffffd)
(enc1/2/3=3,3,fffffd) (Latch1/2/3=3,3,fffffd)
(enc1/2/3=4,4,fffffc) (Latch1/2/3=3,3,fffffd)
(enc1/2/3=4,4,fffffc) (Latch1/2/3=3,3,fffffd)
(enc1/2/3=5,5,fffffb) (Latch1/2/3=3,3,fffffd)
(enc1/2/3=5,5,fffffb) (Latch1/2/3=3,3,fffffd)
i7188E>
```

Encoder 1/2/3 Latch encoders by Z1/Z2/Z3

The interrupt service routines for Z3 are given as follows:

```
#define EOITYPE_INT0       0x0c
#define EOITYPE_INT1       0x0d
#define EOITYPE_INT4       0x10
#define INT_EOI            0xff22    // End-of-interrupt register
#define INT_MASK           0xff28    // Interrupt mask register

unsigned long EncLatch1,EncLatch2,EncLatch3;
//-----------------------------------------

unsigned long OldIntVect0=0,OldIntVect1=0,OldIntVect4=0;
unsigned long far *IntVect=(unsigned long far *)0L;

void interrupt Int4Isr(void)
{
   read_enc3(&EncLatch3);
   outpw(INT_EOI,EOITYPE_INT4);
}

void InstallInt4Isr(void)
{
   OldIntVect0=IntVect[0x10];               // save old ISR
   IntVect[0x10]=(unsigned long)Int4Isr;    // install new ISR
   outpw(INT_MASK, inpw(INT_MASK)&0xfeff);// enable INT-4
}

void RestoreInt4Isr(void)
{
   if(OldIntVect4){
       IntVect[0x10]=OldIntVect4;            // restore OLD ISR
       outpw(INT_MASK,inpw(INT_MASK)|0x0100);// disable INT-4
   }
}
```