PX318J
Application Note
006

Example Code

Version	Date	Description	Author
1.00	2019/08/06	Create the file	Rocky Hsiao
1.01	2020/07/20	#define PsPuw (0x20) revise (0x10)	Robert Chan
1.02	2020/7/23	The calibration process has a headroom of 10cm changed to 100mm	Robert Chan
1.03	2020/7/27	Modify the initialization PsPuc Register address 0x61 to 0x62	Robert Chan
1.04	2022/03/02	Modify the Function-AutoDac() and remove Fixed-point	Brian Chiu

px318j.h

```
//Function form MCU
extern uint32_t MCU_I2C_Write(uint8_t devid, uint8_t reg, uint8_t* data, uint8_t num);
extern uint32_t MCU_I2C_Read(uint8_t devid, uint8_t reg, uint8_t* data, uint8_t num);
extern void MCU_Delay_ms(uint32_t millisecond);

//#define FIXEDPT_BITS 32
//I2C device address (7bits)
//SEL PIN = Float,      address = 0x1D
//SEL PIN = VDD,        address = 0x1F
//SEL PIN = GND,        address = 0x1C
#define PX318J_ID                      (0x1C)
#define PsBits                         (0x01)    //ADC output = 10-bits
#define PsMean                         (0x00)    //Mean = 1 time
#define PsCtrl                         ((PsBits << 4) | (PsMean << 6) | (0x05))

#define PsPuw                          (0x10)    // VSCEL pulses width, 0x10 width = 32 us
#define PsPuc                          (0x02)    // VSECL pulses count, 0x02 = 2 counts

#define PsDrv                          (0x0B)    // VSCEL driving current, 0x0b = 12 mA
#define PsDrvCtrl                      (PsDrv)

#define WaitTime                       (0x11)    // Sensor waiting time 0x11 = 170 ms

#define PsWaitAlgo                     (0x01)
#define PsIntAlgo                      (0x01)
#define PsPers                         (0x04)
//PsInt asserted after 4 consecutive PsData meets the PsInt criteria
#define PsAlgoCtrl                     ((PsWaitAlgo << 5) | (PsIntAlgo << 4) | (PsPers))

#define DefaultThreshold               1         //1 = fixed threshold, 0 = factory threshold
#define PsDefaultThresholdHigh         600
#define PsDefaultThresholdLow          400
#define LoadCtCalibrationSetting       0         //1 = load close talk calibration setting

#define PXY_FULL_RANGE                 ((1 << (PsBits + 9)) - 1)
#define TARGET_PXY                     ((PXY_FULL_RANGE + 1) >> 2)

void PX318J_enable(uint8_t addr);
uint8_t PX318J_auto_dac(uint8_t addr);
```

px318j.c

Function list：

```
//*******************************************************************
void PX318J_I2C_Write(uint8_t addr,uint8_t reg, uint8_t data)
{
     MCU_I2C_Write(addr, reg, &data, 1);
}


//*******************************************************************
void PX318J_I2C_Write_Word(uint8_t addr, uint8_t reg, uint16_t data)
{
     uint8_t value[2];

     value[0] = data & 0x00FF;
     value[1] = data >> 8;

     MCU_I2C_Write(addr, reg, value, 2);
}


//*******************************************************************
void PX318J_I2C_Read(uint8_t addr, uint8_t reg, uint8_t* data)
{
     MCU_I2C_Read(addr, reg, data, 1);
}

//*******************************************************************
uint16_t PX318J_I2C_Read_Word(uint8_t addr, uint8_t reg, uint8_t* data, uint16_t mask)
{
     uint16_t value;

     MCU_I2C_Read(addr, reg, data, 2);

     value = data[1];
     value <<= 8;
     value |= data[0];

     value &= mask;

     return value;
}


//*******************************************************************
void PX318J_enable(uint8_t addr, uint8_t enable)
{
     if (px318j_enable)
     {
          PX318J_I2C_Write(addr, 0xF0, 0x02);
          MCU_Delay_ms(10);
     }
     else
     {
          PX318J_I2C_Write(addr, 0xF0, 0x00);
          MCU_Delay_ms(5);
     }
}
```

Initial Sensor

```
#if LoadCtCalibrationSetting
uint8_t PsDacCtrl = 0;
uint8_t PsCtDac = 0;
uint8_t PsCalL = 0;
uint8_t PsCalH = 0;
#endif

PX318J_I2C_Write(PX318J_ID, 0xF4, 0xEE);     // soft reset
DelayMs(30);                                 // waiting for soft reset
PX318J_I2C_Write(PX318J_ID, 0x60, PsCtrl);   // ADC output = 10-bits, Mean = 1
PX318J_I2C_Write(PX318J_ID, 0x61, PsPuw);    // VSCEL pulses width, 0x10 width = 32 us
PX318J_I2C_Write(PX318J_ID, 0x62, PsPuc);    // VSECL pulses count, 0x02 = 2 counts
PX318J_I2C_Write(PX318J_ID, 0x64, PsDrv);    // VSCEL driving current, 0x0b = 12 mA
PX318J_I2C_Write(PX318J_ID, 0x4F, WaitTime); // Sensor waiting time 0x11 = 170 ms

//High, Low Threshold setting path. Form default or factory calibration value
#if DefaultThreshold
PX318J_I2C_Write_Word(PX318J_ID, 0x6C, PsDefaultThresholdLow);
PX318J_I2C_Write_Word(PX318J_ID, 0x6E, PsDefaultThresholdHigh);
#else
// load threshold value from flash memory (add by customer)

// PX318J_I2C_Write_Word(PX318J_ID, 0x6C, FACOTRY_L_THRESHOLD);
// PX318J_I2C_Write_Word (PX318J_ID, 0x6E, FACOTRY_H_THRESHOLD);
#endif

#if LoadCtCalibrationSetting
// load calibration value from flash memory (add by customer)

PX318J_I2C_Write(PX318J_ID, 0x65, PsDacCtrl);    //set PsDacCtrl
PX318J_I2C_Write(PX318J_ID, 0x67, PsCtDac); //set PsCtDac
PX318J_I2C_Write(PX318J_ID, 0x69, PsCalL);  //set PsCal
PX318J_I2C_Write(PX318J_ID, 0x6A, PsCalH);  //set PsCal
#endif

PX318J_I2C_Write(PX318J_ID, 0xFE, 0x00);    // clear status flag
PX318J_enable(PX318J_ID, 1);                //PX318J Enable
```

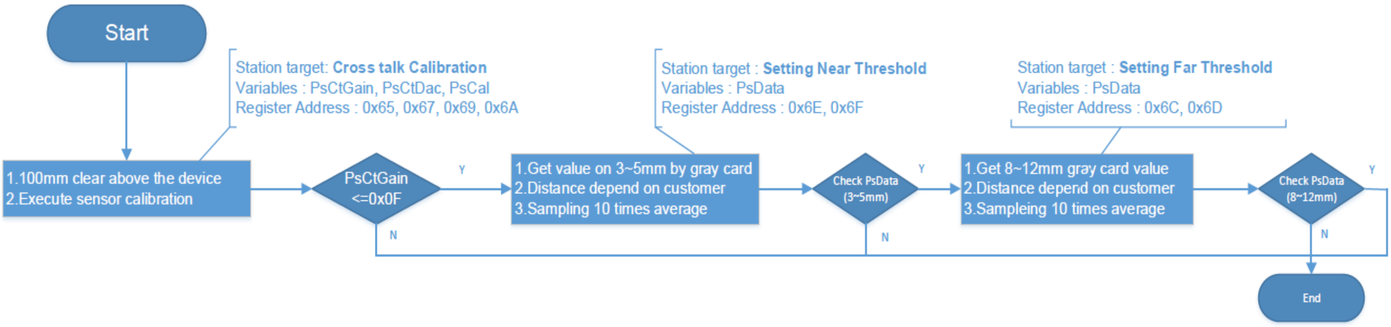
Polling

```
while(1)
{
     uint8_t near_far_flag = 0;
     uint8_t buf = 0;
     PX318J_I2C_Read(PX318J_ID, 0xFF, &buf)              //read near / far status
     if (buf & 0x80)
         near_far_flag = 1; //far event
     else
         near_far_flag = 0; //near event
     DelayMs(30); // waiting for sensor output ready
}
```

ISR

```
PX318J_ISR()
{
     uint8_t near_far_flag = 0;
     uint8_t buf = 0;
     PX318J_I2C_Read(PX318J_ID, 0xFE, &IntFlag)        //Check Interrupt status
     if (IntFlag & 0x02) {// checking PsInt, make sure the ISR is from sensor
          PX318J_I2C_Read(PX318J_ID, 0xFF, &buf)       //read near / far status
          if (buf & 0x80)
               near_far_flag = 1;            //far event
          else
               near_far_flag = 0;            //near event
          PX318J_I2C_Write(PX318J_ID, 0xFE, 0x00);     //release interrupt pin and flag
     }
}
```

Crosstalk calibration and High / Low threshold calibration flow


Start
Station target: Cross talk Calibration
Variables : PsCtGain, PsCtDac, PsCal
Register Address : 0x65, 0x67, 0x69, 0x6A
Station target : Setting Near Threshold
Variables : PsData
Register Address : 0x6E, 0x6F
Station target : Setting Far Threshold
Variables : PsData
Register Address : 0x6C, 0x6D
1.100mm clear above the device
2.Execute sensor calibration
PsCtGain
<=0x0F
Y
N
1.Get value on 3~5mm by gray card
2.Distance depend on customer
3.Sampling 10 times average
Check PsData
(3~5mm)
Y
N
1.Get 8~12mm gray card value
2.Distance depend on customer
3.Sampleing 10 times average
Check PsData
(8~12mm)
Y
N
End

How to do crosstalk calibration

1. Using the assembled machine, face the sensor of the device in a direction without any cover.
2. Executing the calibration program.
3. Write the relevant calibration values into the MCU flash memory.

Sensor crosstalk calibration：

```
uint8_t PX318J_auto_dac(void)
{
    uint8_t addr = PX318J_ID;
    uint8_t buff[5] = {0};
    uint16_t PsData = 0;
    bool first_data = true;

    //Setting Variable
    uint8_t PsCtDac = 0;
    uint8_t PsDacCtrl = 0;
    uint8_t PsCtGain = 0;
    int16_t Dac_temp = 0;

    //PI Control variable
    bool PI_Control = true;
    int32_t dp = 0;
    int32_t di = 0;
    uint8_t last_try = 0;

    //Bisection method
    uint8_t DacMax = 96;
    uint8_t DacMin = 1;

    //Sensor Initial
    PX318J_I2C_Write(addr, 0x60, PsCtrl);
    PX318J_I2C_Write(addr, 0x61, PsPuw);
    PX318J_I2C_Write(addr, 0x62, PsPuc);
    PX318J_I2C_Write(addr, 0x64, PsDrvCtrl);
    PX318J_I2C_Write(addr, 0x4F, 0x00);              //WaitTime = 0
    PX318J_I2C_Write(addr, 0x65, 0x01);              //Reset PsDacCtrl
    PX318J_I2C_Write(addr, 0x67, 0x00);              //Reset PsCtDac
    PX318J_I2C_Write(addr, 0x69, 0x00);              //Reset PsCal
    PX318J_I2C_Write(addr, 0x6A, 0x00);              //Reset PsCal
    PX318J_I2C_Write(addr, 0xF1, 0x01);              //Close INT pin output
    PX318J_I2C_Write(addr, 0xF2, 0x10);              //Enable Data Ready Interrupt Halt
    PX318J_I2C_Write(addr, 0xFE, 0x00);              //Clear Interrupt Flag
    PX318J_I2C_Write(addr, 0x80, 0x08);              //Enable Fast-En(Factor function)

    px318j_enable(addr, 1);                          //Enable Sensor

    PsCtGain = 0x01;
    PsDacCtrl = PsCtGain;
    PsCtDac = 0x00;

    //First Step
    while (1)
    {
        if (MCU_I2C_Read(addr, 0xFE, buff, 4) != STATUS_OK)  //Get Interrupt flag and PS Data.
            return 0;

        if ((buff[0] & 0x10) == 0x10) //Data Ready flag
        {
            PsData = (uint16_t)buff[2] + ((uint16_t)buff[3] << 8);

            if (first_data)          //Ignore the first data.
            {
                first_data = false;
                PX318J_I2C_Write(addr, 0xFE, 0x00);    //Clear Interrupt Flag
                continue;
            }

            //With last try and PS Data > 0, finish the calibration else keep going.
            if (last_try == 1 && PsData > 0)
                break;
```

```
else
    last_try = 0;

if (PsCtDac > 0)
{
    //The PsCtDac is over spec, try to use the bisection method to get the right
    setting.
    if (PsData == 0)
    {
        DacMax = (uint8_t)PsCtDac;
        PI_Control = false;
    }
    //PS Data <= target value, finish the calibration.
    else if (PsData <= TARGET_PXY)
        break;
    //With the bisection method, we get the last value. finish calibration.
    else if (PsCtDac == DacMin || PsCtDac == DacMax)
        break;
}
//PS Data <= target value, finish the calibration.
else if (PsData <= TARGET_PXY)
    break;

if (PI_Control)     //Get the setting with PI control.
{
    dp = PsData - TARGET_PXY;
    di += dp;


    Dac_temp = (int16_t)PsCtDac
    + (int16_t)((dp >> 6) + ((di >> 6) + (di >>8))) + (dp >=0 ? 1 : -1);

    if (Dac_temp >= 96)
    {
        if (PsCtGain == 0x0F)
        {
            last_try = 1;
            Dac_temp = 96;
        }
        else
        {
            if (dp > (TARGET_PXY <<1))  //If PS Data > (target value) x 2
            {
               PsCtGain <<=1;          //New PsCtGain = PsCtGain x2
            }
            else if (dp > TARGET_PXY) //If PS Data > target value
            {
               PsCtGain +=2;           //New PsCtGain = PsCtGain + 2
            }
            else
            {
               PSCtGain++;             // New PsCtGain = PsCtGain + 1
            }

            if (PsCtGain == 0x00)
                PsCtGain = 1;
            else if (PsCtGain > 0x0F)
                PsCtGain = 0x0F;

            Dac_temp = 48;

            PsDacCtrl = (PsDacCtrl & 0xF0) | PsCtGain;
            PX318J_I2C_Write(addr, 0x65, PsDacCtrl);
        }
    }
}
```

```
            else
            {
                if (PsData > TARGET_PXY)
                    DacMin = (uint8_t)Dac_temp;

                if (PsData < PXY_FULL_RANGE && PsData > TARGET_PXY)  //Reduce calculate time.
                    Dac_temp += 1;
                else
                    Dac_temp = (uint16_t)(DacMin + DacMax) >> 1;
            }

            PsCtDac = (uint8_t)Dac_temp;

            PX318J_I2C_Write(addr, 0x67, PsCtDac);
            PX318J_I2C_Write(addr, 0xFE, 0x00);    //Clear Interrupt Flag
        }
    }

    PX318J_enable(addr, 0);                 //Shutdown sensor
    PX318J_I2C_Write(addr, 0xFE, 0x00);    //Clear IntFlag
    PX318J_I2C_Write(addr, 0xF2, 0x00);    //DataHalt Disable
    PX318J_I2C_Write(addr, 0x80, 0x00);    //FastEn Disable

    //Second Step
    PX318J_enable(addr, 1);                 //Enable sensor

    uint8_t index = 0;
    uint32_t Sum = 0;
    do
    {
        MCU_I2C_Read(addr, 0xFE, buff, 4);
        if ((buff[0] & 0x10) == 0x10)
        {
            PsData = (uint16_t)buff[2] + ((uint16_t)buff[3] << 8);

            buff[0] = 0x00;
            PX318J_I2C_Write(addr, 0xFE, 0x00);
            if(index > 1)  //Ignore the first two data
                Sum += PsData;
            index++;
        }
    }while (index < 10);

    PX318J_enable(addr, 0);                 //Shutdown sensor
    PsData = (uint16_t)(Sum >> 3) + 20;

    PX318J_I2C_Write_Word(addr, 0x69, PsData);
    PX318J_I2C_Write(addr, 0xF1, 0x03);         // Open INT pin output
    PX318J_I2C_Write(addr, 0x4F, WaitTime);     //WaitTime = 170ms

    //Save calibration value to flash memory (this function have to add by customer)
    //reg 0x65 (PsDacCtrl)
    //reg 0x67 (PsCtDac)
    //reg 0x69, 0x6a (PsData)


    PX318J_enable(addr, 1);                      //Enable sensor

    return 1;
}
```

Important Notice and Disclaimer

DI reserves the right to make changes to this document and its products and specifications at any time without notice. Customers should obtain and confirm the latest product information and specifications before final design, purchase or use.

DI makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ***DI*** assume any liability for application assistance or customer product design. ***DI*** does not warrant or accept any liability with products which are purchased or used for any unintended or unauthorized application.

No license is granted by implication or otherwise under any intellectual property rights of ***DI***.

DI products are not authorized for use as critical components in life support devices or systems without express written approval of *DI*.