PNI Application Note:

SENtral Sample Code

1 Introduction

This application note provides sample code for using the SENtral Sensor Fusion Coprocessor. These sample codes illustrate how to load the SENtral Configuration File for the host or a dedicated EEPROM, and describes the necessary steps to place SENtral into Normal Operation. The code is written in C++.

2 Sample Code

2.1 Firmware File Description and Structures

```
/******************** (C) COPYRIGHT 2014 PNI Sensor Corp ************************
* File Name      : EEPROMImage.h
* Date           : 28-Jan-2014
* Description    : Firmware file description and structures
********************************************************************************/

   Typedef      UInt16       EEPROMMagic;
   #define EEPROM_MAGIC_VALUE       0x652A
      /** EEPROM boot flags. */
   Typedef union {
      /** Direct access to all flags. */
      UInt16 value;
      Struct {
         /** Do not execute the EEPROM image immediately after upload. */
         UInt16       EEPROMNoExec:1;
         /** Reserved */
         UInt16       Reserved:7;
         /** The clock speed for uploading the firmware. */
```

```
        UInt16       I2CClockSpeed:3;
        /** The Expected Rom Version for the image. */
        UInt16       ROMVerExp:4;
        /** Reserved */
        UInt16       reserved1:1;
    } bits;

} EEPROMFlags;

#define EXP_ROM_VERSION_ANY      0x00
#define EXP_ROM_VERSION_DI01     0x01
#define EXP_ROM_VERSION_DI02     0x02

Typedef      UInt32       EEPROMTextCRC;
Typedef      UInt32       EEPROMDataCRC;
Typedef      UInt16       EEPROMTextLength;
Typedef      UInt16       EEPROMDataLength;
Typedef      UInt8*       EEPROMText;
Typedef      UInt8*       EEPROMData;

/** EEPROM header format
* NOTE: a ROM version may also be useful to ensure an incorrect ram binary is
    not used.
* This is currently not implimented, however the RAM / EEPROM start code can
    double check this before it starts if needed.
*/
Typedef struct {
    /** The firmware magic number */
    UInt16    Magic; // Already read
        /** Flags used to notify and control the boot process */
    UInt16    EEPROMFlags;
        /** The CRC32-CCITT of the firmware text segment */
    UInt32    EEPROMTextCRC; // CRC32-CCITT
        /** The CRC32-CCITT of the firmware data segment */
    UInt32    EEPROMDataCRC; // CRC32-CCITT
        /** The number of program bytes to upload */
    UInt16    EEPROMTextLength;
        /** The number of data bytes to upload */
    UInt16    EEPROMDataLength;
} EEPROMHeader;
```

2.2 Sentral Setup after upload of Configuration File

```
/******************** (C) COPYRIGHT 2014 PNI Sensor Corp ***************************
* Function Name  : Setup_Sentral
* Date           : 28-Jan-2014
* Description    : Enables events, sets sensor rates, and commands CPU to run.
* Return         : None
*******************************************************************************/

    SInt32 Setup_Sentral()
    {
    UInt8 I2CTransactionStatus = 0x00;

    I2CTransactionStatus |= I2CWrite(SENTRAL_ADDRESS,ENABLE_EVENTS_REG, 0x20);// enable gyro
                                        // event only (for data logging function)
    I2CTransactionStatus |= I2CWrite(SENTRAL_ADDRESS,MAG_RATE_REG, 0x64);// set mag rate 100Hz
    I2CTransactionStatus |= I2CWrite(SENTRAL_ADDRESS,ACCEL_RATE_REG, 0x0a);// set accel rate 100H
    I2CTransactionStatus |= I2CWrite(SENTRAL_ADDRESS,GYRO_RATE_REG, 0x0f);// set gyro rate 150Hz
    I2CTransactionStatus |= I2CWrite(SENTRAL_ADDRESS,ALGORITHM_CONTROL, 0x02);// update
    I2CTransactionStatus |= I2CWrite(SENTRAL_ADDRESS,HOST_CONTROL_REG, 0x01);// Request
                                                                    // CPU to run

        if (I2CTransactionStatus)
            return RETURN_FAILURE;
        else
            return RETURN_SUCCESS;

    }
```

2.3 Reset SENtral – with an EEPROM

```
/******************** (C) COPYRIGHT 2014 PNI Sensor Corp ***************************
* Function Name  : ResetSentral
* Date           : 28-Jan-2014
* Description    : Sends a Reset command and checks Sentral's status.
*                  This function calls Setup_Sentral after Sentral successfully
*                  boots from the EEPROM
* Return         : status
*******************************************************************************/

    SInt32 ResetSentral(UInt8 command)
    {

        UInt8 ReturnedByte = 0x00, temp[1], boot_timeout = FALSE, count = 0x00;
        SInt8 ret_status = 0x00;
```

```
    ret_status = SentralRead(REVISION_ID_REG, &ReturnedByte);//read back sentral
    ROM revision, todo display over uart

    if (ret_status == TRUE) {

    I2CWrite(SENTRAL_ADDRESS,RESET_REQ_REG, 0x01);

    // Check sentral's status register to see if it has booted successfully.
    // Times out after 3 seconds.

    while (((ReturnedByte & 0x06) != 2) && (boot_timeout == FALSE)) {
        SentralRead(SENTRAL_STATUS_REG, &ReturnedByte);
        count++;
        if (count == 30)
        boot_timeout = TRUE;

        Clock_Wait(100);

    }

    if (boot_timeout) {
        PrintChars("Timeout occurred, sentral not present or took too long to
    boot from the EEPROMn");

        return RETURN_FAILURE;
    }

    else {
        PrintChars("Boot from EEPROM successful n");

        return Setup_Sentral();

    }

    }

    PrintChars("Sentral not detectd! n");

    return RETURN_FAILURE;
}
```

2.4 Uploading Configuration File from a Host

```
/******************** (C) COPYRIGHT 2014 PNI Sensor Corp ***************************
* File Name      : upload_fw.c
* Date           : 28-Jan-2014
* Description    : Example function for uploading configuration file from a host.
*******************************************************************************/

    int upload_fw(uint32_t* numericArgs, uint8_t numNumeric, float* floatArgs,
        uint8_t numFloat, char** charArgs, uint8_t numChar)
    {
      UInt32 numBytes = 0;

      FILE* fw = fopen(charArgs[0], "rb");
      if(!fw)
      {
       printf("Unable to open firmware image '%s'\n", charArgs[0]);
       return -1;
      }

      char serialPort[] = "COM%d";
      sprintf(serialPort, "COM%d", numericArgs[0]);

    Serial* port = new Serial(serialPort, BAUD_RATE);
      if(!port->isOpened())
      {
       printf("Unable to open serial port %s\n", serialPort);
       fclose(fw);
       return -1;
      }

      fseek(fw, 0, SEEK_END);
      numBytes = ftell(fw);
      fseek(fw, 0, SEEK_SET);

      EEPROMHeader header;
      fread(&header, sizeof(header), 1, fw);

      // validate EEPROM image

      if(header.Magic != EEPROM_MAGIC_VALUE)
      {
       // magic number does not match eeprom magic.
       printf("Invalid firmware image: magic number does not match.\n");
       return -1;
      }
      if(header.EEPROMDataLength)
      {
       /* Unable to upload data via i2c, bootloader must upload by reading from
```

```
      eeprom.
     NOTE: if needed, a helper uploader can be used to perform a two stage
      upload. */
  printf("Unable to upload firmware image. Firmware may only be loaded from
  EEPROM.");
  return -1;
}

if(header.EEPROMTextLength != numBytes - sizeof(EEPROMHeader))
{
  /* Number of bytes remaining in file does not match the number of bytes in
  the header. */
  printf("Firmware image is incomplete. Expected %d bytes, found %d.\n",
  header.EEPROMTextLength, numBytes - sizeof(EEPROMHeader));
  return -1;
}

printf("Preparing to upload %d bytes...\n", numBytes - sizeof(EEPROMHeader));

UInt32 value = i2c_read(I2C_SLAVE_ADDR, 0x37, NULL, 1, port);
printf("SentralStatus = 0x%0.2X\n", value);
if(value & 0x02)
{
  printf("Sentral already has eeprom firmware loaded.\n");
}

/* Write value 0x01 to the ResetReq register, address 0x9B.  This will result
  in a hard reset of the Sentral.  This is unnecessary if the prior event was
  a Reset. */

if(!(value & 0x08))
{
  printf("CPU is not in standby, issuing a shutdown request.\n");
  uint32_t data[] = { 0x00 };
  i2c_write(I2C_SLAVE_ADDR, 0x34, data, 1, port);

  UInt32 value = i2c_read(I2C_SLAVE_ADDR, 0x34, NULL, 1, port);
  printf("HostControl = 0x%0.2X\n", value);
  do {
    value = i2c_read(I2C_SLAVE_ADDR, 0x37, NULL, 1, port);
    printf("SentralStatus = 0x%0.2X\n", value);
    Sleep(100);
  } while(!(value & 0x08));
}

printf("Enabling upload mode...\n");

/* Write value 0x02 to the HostControl register, address 0x34.  This will
  enable an upload of the Configuration File. */
```

```
    uint32_t data[] = { 0x02 };
    i2c_write(I2C_SLAVE_ADDR, 0x34, data, 1, port);
    value = i2c_read(I2C_SLAVE_ADDR, 0x34, NULL, 1, port);
    printf("HostControl = 0x%0.2X\n", value);

    printf("Uploading data...\n");

    #define TRASACTION_SIZE  3
    for(int i = 0; i < header.EEPROMTextLength; i += TRASACTION_SIZE * 4)
    {

     uint32_t* data = new uint32_t[TRASACTION_SIZE * 4];
     for(int j = 0; j < TRASACTION_SIZE; j++)
     {
       uint32_t value;
       fread(&value, 4, 1, fw);
       data[j * 4 + 0] = (value >> 24) & 0xFF;
       data[j * 4 + 1] = (value >> 16) & 0xFF;
       data[j * 4 + 2] = (value >>  8) & 0xFF;
       data[j * 4 + 3] = (value >>  0) & 0xFF;
     }
     if(header.EEPROMTextLength < (i + (TRASACTION_SIZE * 4)))
     {
       uint32_t bytes = header.EEPROMTextLength - i;
       i2c_write(I2C_SLAVE_ADDR, 0x96, data, bytes, port);
     }
     else
     {
     /* Write the Configuration File to Sentral's program RAM.  The file is sent
     one byte at a time, using the UploadData register, register address 0x96. */

       i2c_write(I2C_SLAVE_ADDR, 0x96, data, TRASACTION_SIZE * 4, port);
     }

     delete data;
    }
    uint32_t crc[4];

     /* Read the CRC-32 register, address 0x97 - 0x9A.  Compare this to the Host
     calculated CRC-32 to confirm a successful upload. */

  i2c_read(I2C_SLAVE_ADDR, 0x97, crc, 4, port);
    uint32_t actualCRC =  crc[0] << 0 | crc[1] << 8 | crc[2] << 16 | crc[3] <<
     24;

    if(actualCRC != header.EEPROMTextCRC)
    {
```

```
    printf("Program crc (0x%.8X) does not match CRC reported by Sentral
    (0x%0.8X)\n", header.EEPROMTextCRC, actualCRC);
  }
  else
  {
    printf("Firmware Upload Complete.\n");
  }

  fclose(fw);

  port->close();
  delete port;
  return 0;
}
```