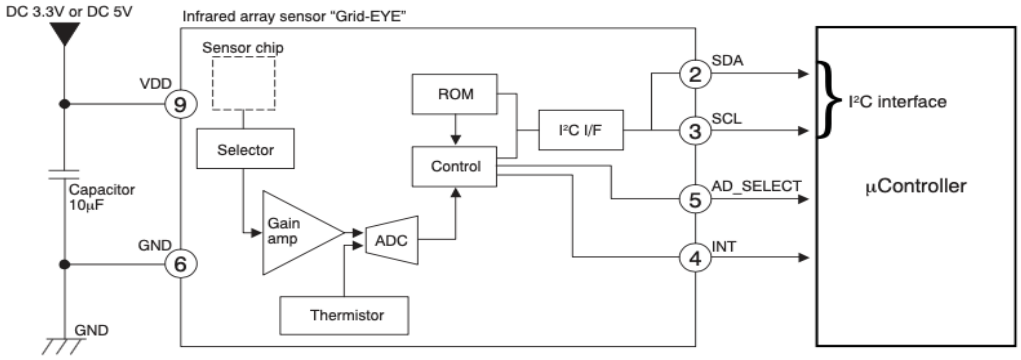
Panasonic

C-Source Code for General Use

Design Guidelines Grid-EYE

INTRODUCTION

Grid-EYE is a high precision, infrared array sensor based on advanced MEMS technology (Micro-Electro-Mechanical-System), featuring 64 thermopile elements in an 8x8 grid format. 64 pixels yield accurate temperature measurement over a viewing angle of 60° (horizontally and vertically) provided by a silicon lens. Grid-EYE uses an I²C communication interface, enabling temperature measurements at speeds of 1 or 10 frames/sec. Using this interface it is very uncomplicated to integrate the sensor into your development environment, read out raw sensor data and depending on your needs to configure it via corresponding registers. Thus a high degree of freedom is guaranteed to the development team.



This documentation has been written for users of the GridEYE infrared array sensor. The intention of this documentation is to provide an overview of the C-source code for the general use. It should make yourself familiar to read out first raw data and give you all the relevant information, for getting the Grid-EYE into operation quickly.
This code is written for an arduino uno development board. For reference please refer to www.arduino.cc.

The example code should give an insight into how to read out thermistor and pixel data via I²C. Furthermore it describes how to process this information and how to print it on serial monitor. To get familiar how to work with different registers the code also describes how to read them out and/or how to write into them.

These commands can be used via serial monitor with this example code:

SR	Read out status register
F1	Framerate set to 1 Hz
F2	F2 Framerate set to 10 Hz
RF	Flag Reset
RI	Initial Reset

NOTE

Panasonic Electric Works Europe AG points out that information and functionality in this example code may be subject to technical changes, as the products are being developed by Panasonic constantly. Panasonic assumes no responsibility for the operation of this free code, possible printing errors or other inaccuracies. Panasonic reserves all rights.

EXAMPLE CODE

Initialization Procedure

Define Constants

Global Variables

Setup

```
/* GridEye Arduino Demonstration Software */
/*-------------------------------------------------------------------------*/
/* This is meant to show the measurement capabilities of the GridEye sensor.   */
/* The following code may be used as example.                                  */
/* Further features (e.g. interrupts) are not part of this demonstration.      */

#include "twi.h"
#include <TimerOne.h>

// GridEye I2C slave address of the sensor 1101 000, AD_SELECT is connected to GND
// GridEye I2C slave address of the sensor 1101 001, AD_SELECT is connected to VDD
#define AD_SELECT_IS_GND 0b1101000
#define AD_SELECT_IS_VDD 0b1101001
const uint8_t Address = AD_SELECT_IS_GND;

// set update period in use. GridEye supports 1 Hz and 10 Hz
#define REFRESH_TIME_1HZ 1000000
#define REFRESH_TIME_10HZ 100000

#define BUFFER_LENGTH 128

uint8_t rxBuffer[BUFFER_LENGTH];
uint8_t txBuffer[6];

volatile char trigger;


void setup()
{
  // Setup I2C bus, transfer speed is set to 100kHz (in twi.h)
  twi_init();

  // Setup serial terminal connection, baudrate is 115200 b/s
  Serial.begin(115200);

  // Setup timer to fire an event each second
  Timer1.initialize(REFRESH_TIME_1HZ);

  // Attach timer interrupt service routine to the timer event
  Timer1.attachInterrupt(timer, 0, 1);
}
```

```
void loop()
{


  register unsigned char i, length;
  register char tmp;
  register int raw;
  register float result;

if (trigger)
  {
    // reset trigger
    trigger = 0;
    // set register pointer of GridEye to 0x0E in order to read the thermistor value
    txBuffer[0]=0x0E;
    i = twi_writeTo(Address, txBuffer, 1, 0);

    // perform block read of 2 bytes into rxBuffer
    length = twi_readFrom(Address, rxBuffer, 2);

    // put upper byte into a register variable
    tmp = rxBuffer[1];

    // convert the upper byte of the 12bit signed value into proper 16bit signed
    // format, if value is negative
    if (tmp & 0b00001000) tmp |= 0b11111000;

    // construct the 16bit signed value, shifting the upper byte 8 bits up and adding
    // the lower byte
    raw = (tmp << 8) | rxBuffer[0];

    // calculate actual thermistor temperature in °C
    result = raw * 0.0625;

    // print value to serial terminal
    Serial.print("\tthermistor value:\t");
    Serial.println(result);
```

Register Variables

Thermistor Data

Get Data

Process Data

Print Data

Pixel Data

Get Data

Process & Print Data

```
  // set register pointer of GridEye to 0x80, the first temperature value register
  txBuffer[0]=0x80;
  i = twi_writeTo(Address, txBuffer, 1, 0);

  // perform block read of 128 bytes into rxBuffer
  // only a block read ensures all given pixel data is out of the same frame
  length = twi_readFrom(Address, rxBuffer, 128);

  // loop through all pixel temperature data
    for(i=0;i<=length;i++)
    {
      // insert a new line every 8 pixel values
      if (!(i % 16)) Serial.println();

      // put temperature data into register variable
      tmp = rxBuffer[i];

      // if we are processing the upper byte
      if (i % 2)
      {
        // convert the upper byte of the 12 bit signed value into proper 16 bit
        // signed format, if value is negative
        if (tmp & 0b00001000) tmp |= 0b11111000;

        // add the upper byte into 16 bit register by shifting it 8 bits up
        raw |= tmp << 8;

        // calculate actual pixel temperature in °C
        result = raw * 0.25;

        Serial.print("\t");

        // if temperature reading is out of specification, don't print it
        if ((result >= -20) && (result <= 100))
          Serial.print(result);
        else
          Serial.print("X.XX");
       }
       // if the lower byte is processed
       else
       {
         // erase raw value by setting it to the lower byte value
         raw = 0xFF & tmp;
       }
    }

  // print new line after each frame
  Serial.println();
}
```

Work with Register

Get Serial Data

Read & Print Status Register

Write into Frameratee Register

Perform Reset

```
// if at least 2 bytes of serial data have been received
if (Serial.available() > 1)
{
  i = 0;
  // get serial buffer data
  while (Serial.available()) txBuffer[i++] = Serial.read();

  // search the command
  switch (txBuffer[0])
  {
  case 'S':
    if (txBuffer[1] == 'R')
    {
      // input the status register address into buffer
      txBuffer[0] = 0x04;
      // send status register address
      i = twi_writeTo(Address, txBuffer, 1, 0);
      // read the status register value
      length = twi_readFrom(Address, rxBuffer, 1);
      // print the status register value
      Serial.print("\t\tStatus register value:\t0b");
      Serial.print(rxBuffer[0], BIN);
      Serial.println();
      Serial.println();
    }
    break;

  case 'F':
    // input the frame rate register address into buffer
    txBuffer[0] = 0x02;
    if (txBuffer[1] == '1')
    {
      // input value to be set into frame rate register
      txBuffer[1] = 0x01;
      // send frame rate register address and its new value
      i = twi_writeTo(Address, txBuffer, 2, 0);
      // match data transfer trigger to sensor data refresh rate
      Timer1.setPeriod(REFRESH_TIME_1HZ);
      Serial.println("\t\tFrame rate set to 1Hz");
      Serial.println();
    }
    if (txBuffer[1] == '2')
    {
      txBuffer[1] = 0x00;
      i = twi_writeTo(Address, txBuffer, 2, 0);
      Timer1.setPeriod(REFRESH_TIME_10HZ);
      Serial.println("\t\tFrame rate set to 10Hz");
      Serial.println();
    }
    break;
  case 'R':
    txBuffer[0] = 0x01;
    if (txBuffer[1] == 'F')
    {
      txBuffer[1] = 0x30;
      i = twi_writeTo(Address, txBuffer, 2, 0);
      Serial.println("\t\tFlag reset done!");
      Serial.println();
    }
    if (txBuffer[1] == 'I')
    {
      txBuffer[1] = 0x3F;
      i = twi_writeTo(Address, txBuffer, 2, 0);
      Serial.println("\t\tInitial reset done!");
      Serial.println();
    }
    break;
  }
 }
}
```

Interrupt Service Routine

```
// timer interrupt service routine
void timer()
{
  // set trigger
  trigger |= 1;
}
```

See also Grid-EYE data sheet.