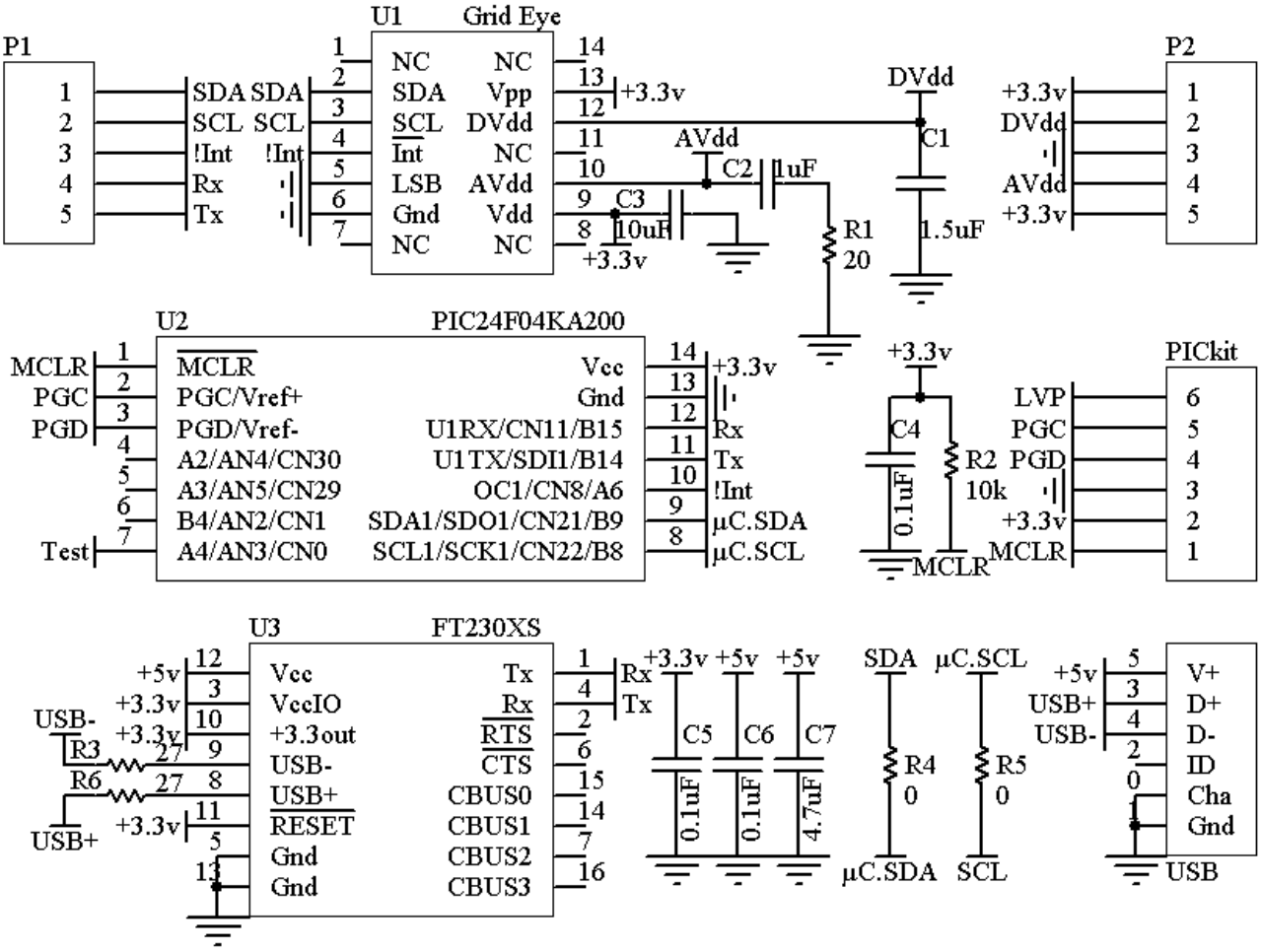
GridEye-Breakout Board.

The Grid-EYE is a high precision infrared array sensor based on advanced MEMS technology. It is capable of measuring temperatures across a grid of 8x8 (64 pixels) up to ten times per second over the following temperature ranges:

High gain: +32F to +176F
Low gain: -4F to +212F

Schematics:



This PIR is connected to a PIC24F04KA200 micro-controller. The micro-controller operates within a 100ms timing. Pseudo Code for firmware:

```
// Thermistor reading
I2C Start
I2C Send Address + Write
I2C Send Byte 0x0E
I2C Stop
```

```
Delay 15 µS
I2C Start
I2C Send Address + Read
I2C Read Thermistor Byte 1
I2C Send Ack
I2C Read Thermistor Byte 2
I2C Send NoAck
I2C Stop
// End of Thermistor reading


Delay 800 µS

I2C Start
I2C Send Address + Write
I2C Send Byte 0x80 // Set up to read temperature Array
I2C Stop
Delay 15 µS


Loop 4 times:
{
  I2C Start
  I2C Send Address + Read


  Loop 31 times:
  {
    I2C Read Pixel Byte
    I2C Send Ack
    Delay 300 uS
  }

  I2C Read Pixel Byte
  I2C Send NoAck
  I2C Stop


  Delay 2 mS
}
```

The Embedded code for PIC24F04KA200 is written in C by Digi-Key. Credits go where they are due. The source code is pasted below:

```
#include <p24F04KA200.h>

#define COMM_QUEUE_SIZE 150
#define UART1_TX(x) ( U1TXREG = x )
#define ADDRESS 104

_FOSCSEL( FNOSC_FRCPLL & IESO_OFF )
_FOSC( POSCMOD_NONE & OSCIOFNC_ON & POSCFREQ_HS & FCKSM_CSDCMD )
_FWDT( WINDIS_OFF & FWDTEN_OFF )
_FPOR( BOREN_BOR0 & PWRTEN_OFF & BORV_18V & MCLRE_ON )
```

```
void I2C_start()
{
    _SEN = 1;
    Nop();
    while(_SEN);
}

void I2C_stop()
{
    _PEN = 1;
    Nop();
    while(_PEN);
}

void I2C_restart()
{
    _RSEN = 1;
    Nop();
    while(_RSEN);
}

void I2C_send_ack()
{
    _ACKDT = 0;
    _ACKEN = 1;
    Nop();
    while (_ACKEN);
}

void I2C_send_nack()
{
    _ACKDT = 1;
    _ACKEN = 1;
    Nop();
    while (_ACKEN);
}

char I2C_send_address(unsigned char addr, char read)
{
    I2C1TRN = ((addr << 1) | (read != 0));
    Nop();
    while (_TRSTAT || _TBF);
    return _ACKSTAT;
}

unsigned char I2C_send_byte(unsigned char b)
{
    I2C1TRN = b;
```

```
    Nop();
    while (_TRSTAT || _TBF);
    return _ACKSTAT;
}

unsigned char I2C_read_byte()
{
    unsigned char b;
    _RCEN = 1;
    Nop();
    while (_RCEN);
    b = I2C1RCV;
    return b;
}

unsigned char * I2C_read_bytes(unsigned char * bytes, int length)
{
    unsigned int i;
    for(i = 0; i < length; ++i)
    {
        bytes[i] = I2C_read_byte();
        if (i == length - 1)
        {
            I2C_send_nack();
        } else
        {
            I2C_send_ack();
        }
    }
    return bytes;
}

void delay_us(unsigned int delay)
{
    if( delay == 0 ) return;
    delay = delay << 1;
    while(--delay);
}

void delay_ms(unsigned int delay)
{
    while(delay--) delay_us(1000);
}

int main(  )
{
    // Hardware setup
    _RCDIV = 0;            // CLKDIV
    _CN8PUE = 1;        // Interrupt input pull-up enable
```

```
    _CN21PUE = 1;        // SDA input pull-up enable
    _CN22PUE = 1;        // SCL input pull-up enable
    _PCFG3 = 1;            // ADC pin to digital mode
    // I2C1 to GridEye
    I2C1BRG = 100;    // I2C1 Baud = 100 kHz @ 30 MIPS
    _I2CEN = 1;            // Enable I2C Module
    // UART1 to USB
    U1BRG = 8;            // UART1 BAUD = 115,200 bps when FCY =
30MHz
    _UARTEN = 1;        // Enable UART Module
    _UTXEN = 1;            // Enables UART TX hardware
    _U1RXIF = 0;        // Clear interrupt flag
    _U1RXIP = 4;        // Interrupt Priority
    _U1RXIE = 1;        // Enable interrupt

    // Timer 1 - Send data packet upon overflow
    T1CONbits.TCKPS = 3;    // Prescalar
    PR1 = 6240;                        // Target value
    _T1IF = 0;                        // Clear interrupt flag
    _T1IP = 5;                        // Interrupt priority
    _T1IE = 1;                        // Enable interrupt

    while(1)
    {
        Idle(); // Magic happens in timer and UART interrupts below.
    }
}

void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void)
{
    char data = 0;
    _U1RXIF = 0;                                                    /
/ Clear interrupt flag
    data = U1RXREG;                                                //
Read RX register
    if( data == '*') T1CONbits.TON = 1;        // Turns timer on
    if( data == '~') T1CONbits.TON = 0;        // Turns timer off
}

void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void)
{
    _T1IF = 0;
    int i,j;
    unsigned char data, checksum;
    UART1_TX('*');
    UART1_TX('*');
    UART1_TX('*');
    checksum = 0;
    // Thermistor reading
```

```
    I2C_start();
    I2C_send_address( ADDRESS, 0 );
    I2C_send_byte(14);
    I2C_stop();
    delay_us(15);
    I2C_start();
    I2C_send_address( ADDRESS, 1 );
    data = I2C_read_byte(); // Thermistor Byte 1
    I2C_send_ack();
    UART1_TX(data);
    checksum += data;
    data = I2C_read_byte(); // Thermistor Byte 1
    I2C_send_nack();
    UART1_TX(data);
    checksum += data;
    I2C_stop();
    // End of Thermistor reading
    delay_us(800);
    I2C_start();
    I2C_send_address( ADDRESS, 0 );
    I2C_send_byte(128); // Set up to read temperature Array
    I2C_stop();
    delay_us(15);
    for( i=0; i<4; i++ )
    {
        I2C_start();
        I2C_send_address( ADDRESS, 1 );
        for( j=0; j<31; j++ )
        {
            data = I2C_read_byte();
            I2C_send_ack();
            UART1_TX(data);
            checksum += data;
            delay_us(300); // 1100
        }
        data = I2C_read_byte();
        I2C_send_nack();
        I2C_stop();
        UART1_TX(data);
        checksum += data;
        delay_ms(2);
    }
    UART1_TX(checksum);
}
```

From the micro-controller to the USB hub, the data bus format is :

```
'*' \
'*' >-- Packet start designator
'*' /
Thermistor High Byte
Thermistor Low Byte
64 High Byte, Low Byte pairs
```

The important part to note from the PIC24 controller firmware is that :

```
if( data == '*') T1CONbits.TON = 1;        // Turns timer on
if( data == '~') T1CONbits.TON = 0;        // Turns timer off
```

So, when we write a '*', we enable the timer, and when we write a '~' we disable the timer.

CODE:

```
"""
Author: Biswaranjan Das
WSU Pullman, WA.

(c) WSU CASAS 2014.

PANASONIC GRIDEYE BREAKOUT BOARD WITH PIC24F04KA200 Microcontroller
Interfacing with x86,x64,ARM based machines with Python 2.7.

Dependencies:
        PySerial
        Python Imaging Library (can be substituted for pygame, opencv, tkinter)

"""

#headers and imports.
from serial import *
import atexit
from threading import Thread
from struct import unpack
import Image


#load 8px,8px image with white background.
img = Image.new( 'RGB', (8,8), "white")
pixels = img.load()

#global variables to keep track of event frame.
count=0
```

```
"""
input parameters: ser -> serial device in usage. by
this time, we have enabled the timer. Processing info.

output : no returns. saves image from PIR to gg#.jpg for all # in int.

"""
def receiving(ser):
    global count
    while True:

        #read the first 5 bytes. {42,42,42,ThermistorL,ThermistorH}
        ser.read(5)

        #increment out frame count. we are ready to stream.
        count+=1

        #range=8*8=64. for each we separate low bytes, print to image.
        for i in range(64):
            try:
                buf1 = ser.read()
                vg= unpack('B', buf1)
                #this will read high byte. to be ignored.
                buf1 = ser.read()

                #vg[0] has the temperature value in F.
                        #insert custom color defs for prettier printing please.
                if(int(vg[0])<80):
                    color=(0,0,vg[0])
                elif(int(vg[0])>80 and int(vg[0])<100):
                    color=((vg[0]-80)*50/20,(100-vg[0])*255/20,0)
                else:
                    color=((100-vg[0])*255/20,(vg[0]-100)*50/20,0)
                pixels[7-(i%8),7-(i/8)]=color
            except:
                print()

        #pprint for images basically. resizing antialias to 500x500
        #from 8x8
        newim = img.resize((500,500), Image.ANTIALIAS)
        newim.save('gg'+str(count)+'.jpg')

"""
input parameters: ser -> serial device in usage.
output : N/A

This function executes when we exit/terminate our python script.
This will make sure that out serial device (PIC24F04KA200) is not
continuing to run after we terminate the appication. Write a '~'!!
"""
def exitSafe(ser):
        try:
                ser.write('~')
                ser.close();
        except:
                pass
```

```
"""
main function.
no input/output.

config serial device : with baudrate=115200,Data=8Bits,Timeout=0.1

"""
if __name__ ==  '__main__':
    ser = Serial(
        port='/dev/ttyUSB0',
        baudrate=115200,
        bytesize=EIGHTBITS,
        parity=PARITY_NONE,
        stopbits=STOPBITS_ONE,
        timeout=0.1,
        rtscts=0,
        xonxoff=0
    )

    #close to make sure we have no trouble opening.
    ser.close()

    #open serial device.
    ser.open()

    #write '~' to close if the timer is pre-enabled.
    ser.write('~')

    #write '*' to enable timer.
    ser.write('*')

            #dynamic thread to process received data.
    Thread(target=receiving, args=(ser,)).start()

    #function to make sure we have a termination procedure in place.
            atexit.register(exitSafe, ser)
```