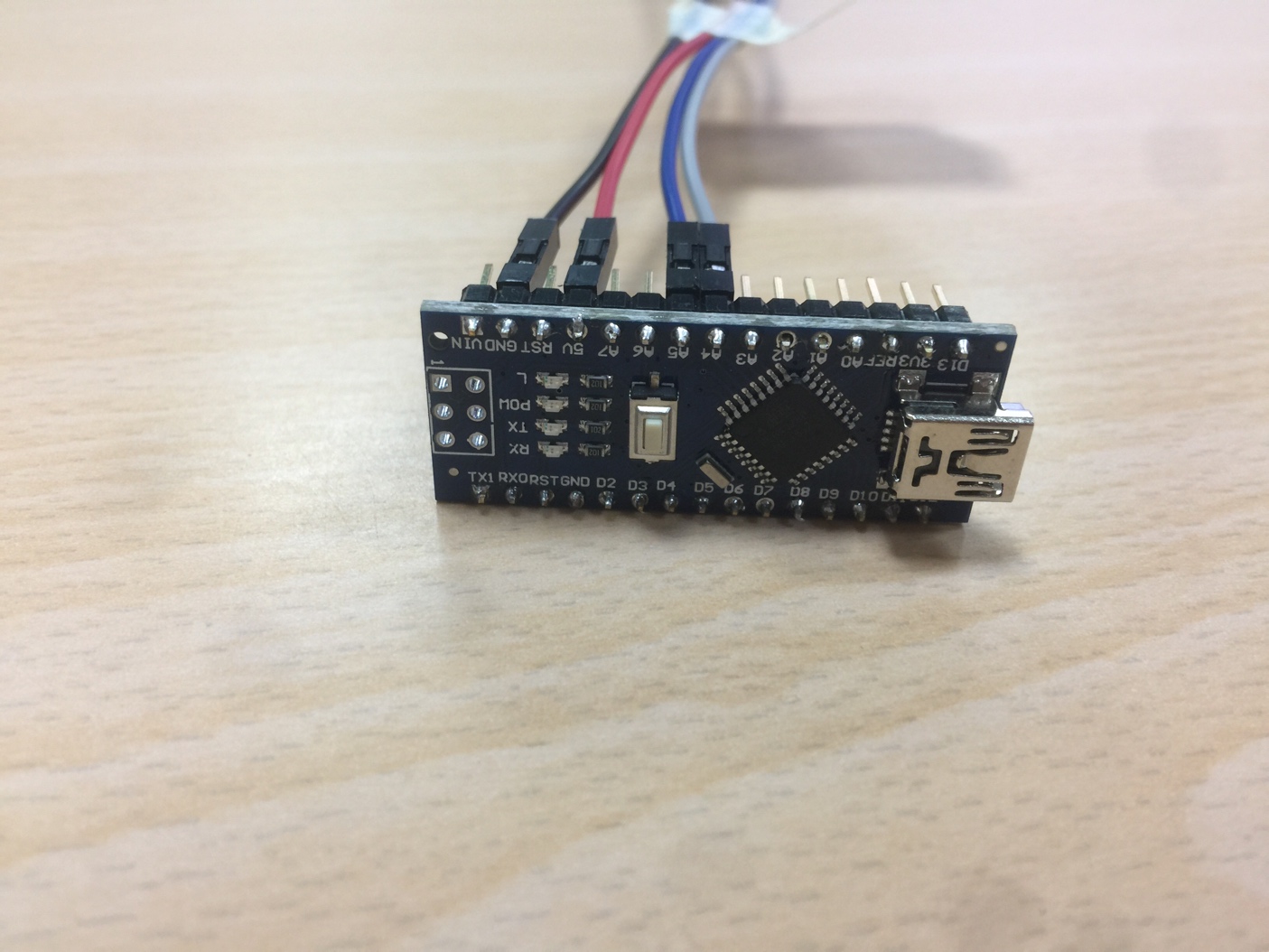
ES Report

# Lux parameter



Names: Deandra Arends

Sharilusca Thomas

Crisleidy Lopez de la Cruz

Stephany Wong

# Description of the Lux parameter and how it works

The Lux parameter is a very-high sensitivity light-to-digital converter that transforms light intensity into a digital signal output capable of direct I2C interface. The device combines one broadband photodiode (visible plus infrared) and one infrared-responding photodiode on a single CMOS integrated circuit. Two integrating ADCs convert the photodiode currents into a digital output that represents the irradiance measured on each channel. This digital output can be input to a microprocessor where illuminance (ambient light level) in lux is derived using an empirical formula to approximate the human eye response. It supports a traditional level style interrupt that remains asserted until the firmware clears it.

When the future is dazzlingly-bright, this ultra-high-range luminosity sensor will help you measure it. The lux’s luminosity sensor is an advanced digital light sensor, ideal for use in a wide range of light situations. Compared to low cost CdS cells, this sensor is more precise, allowing for exact lux calculations and can be configured for different gain/timing ranges to detect light ranges from up to 188uLux up to 88,000 Lux on the fly. The best part of this sensor is that it contains both infrared and full spectrum diodes! That means you can separately measure infrared, full-spectrum or human-visible light. Most sensors can only detect one or the other, which does not accurately represent what human eyes see (since we cannot perceive the IR light that is detected by most photo diodes)

Visible: the amount of light that we receive when the sun shines in our eye.

Lux: the intensity of the light that our eyes can see.

MS: LED light

IR: infrared radiation extends from the nominal red edge from visible spectrum

# Standard Value of the Lux

We use lux meters to measure light intensity. It is important to measure the light intensity for different reasons. For example, plants depends on the light to survive and to perform photosynthesis. By measuring the light intensity you can be aware of the amount of light the plant is receiving and provide them with ideal growing conditions to increase their health. It can also be used to ensure efficiency and safety for example by measure the amount of light in office spaces for emergency exits, and by enhancing the visibility and utility in outdoor spaces like improving visibility in streets and parking lots. Lighting is not always created effectively, because there are different factors that influence lighting. This is why light meters are important.

# How we used the sensor

First we plugged in the sensor the computer, after we ran the program so the parameter could start working. So we went outside to test it and we noticed that when we are in the shade (less sun) the ms was 5 numbers, the IR, Full and Visible were 3 numbers, and the lux 2 numbers. What we could notice is, that at the moment we went outside all of them increased, and something that captured our attention was that Full, increased to the amount of 37889 and did not change any more, while we were in the sun. And the visible before increasing, was 0, and after a few seconds started to show the increased amount.

# Values at UA

Inside

(48875 ms) IR: 231 Full: 707 Visible: 476 Lux: 53

(49499 ms) IR: 233 Full: 693 Visible: 460 Lux: 50

(50121 ms) IR: 229 Full: 693 Visible: 464 Lux: 51

(50744 ms) IR: 229 Full: 666 Visible: 437 Lux: 47

(51367 ms) IR: 225 Full: 747 Visible: 522 Lux: 61

Outside

(114974 ms) IR: 37889 Full: 37889 Visible: 0 Lux: 4294965627

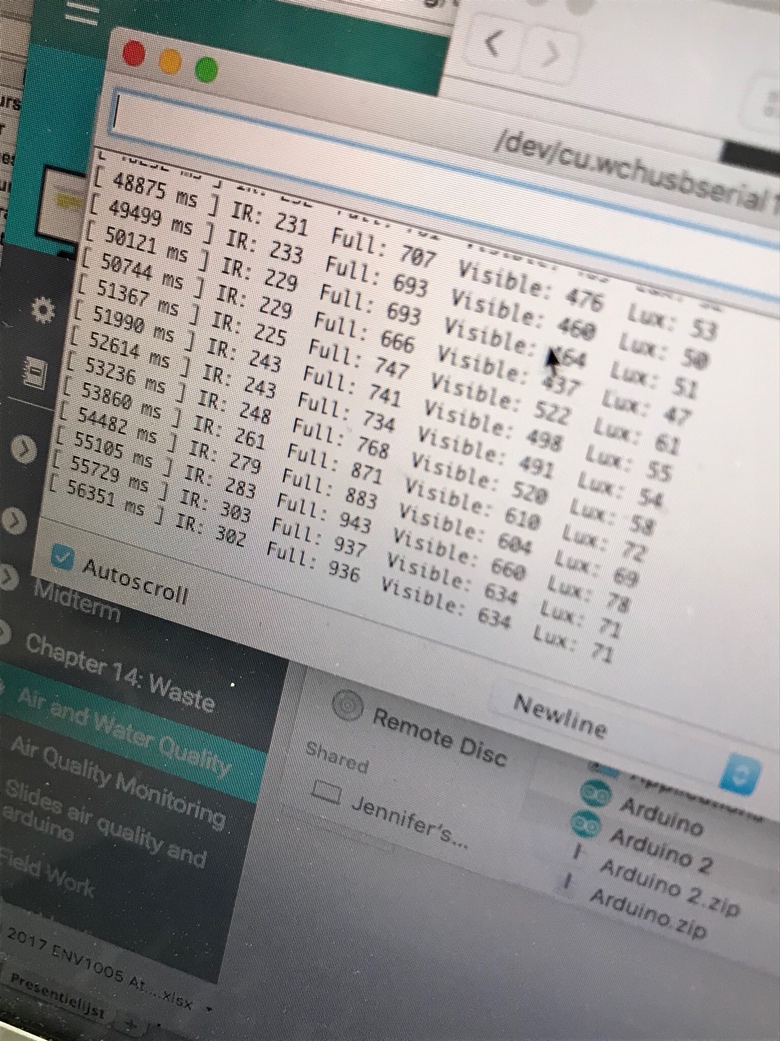
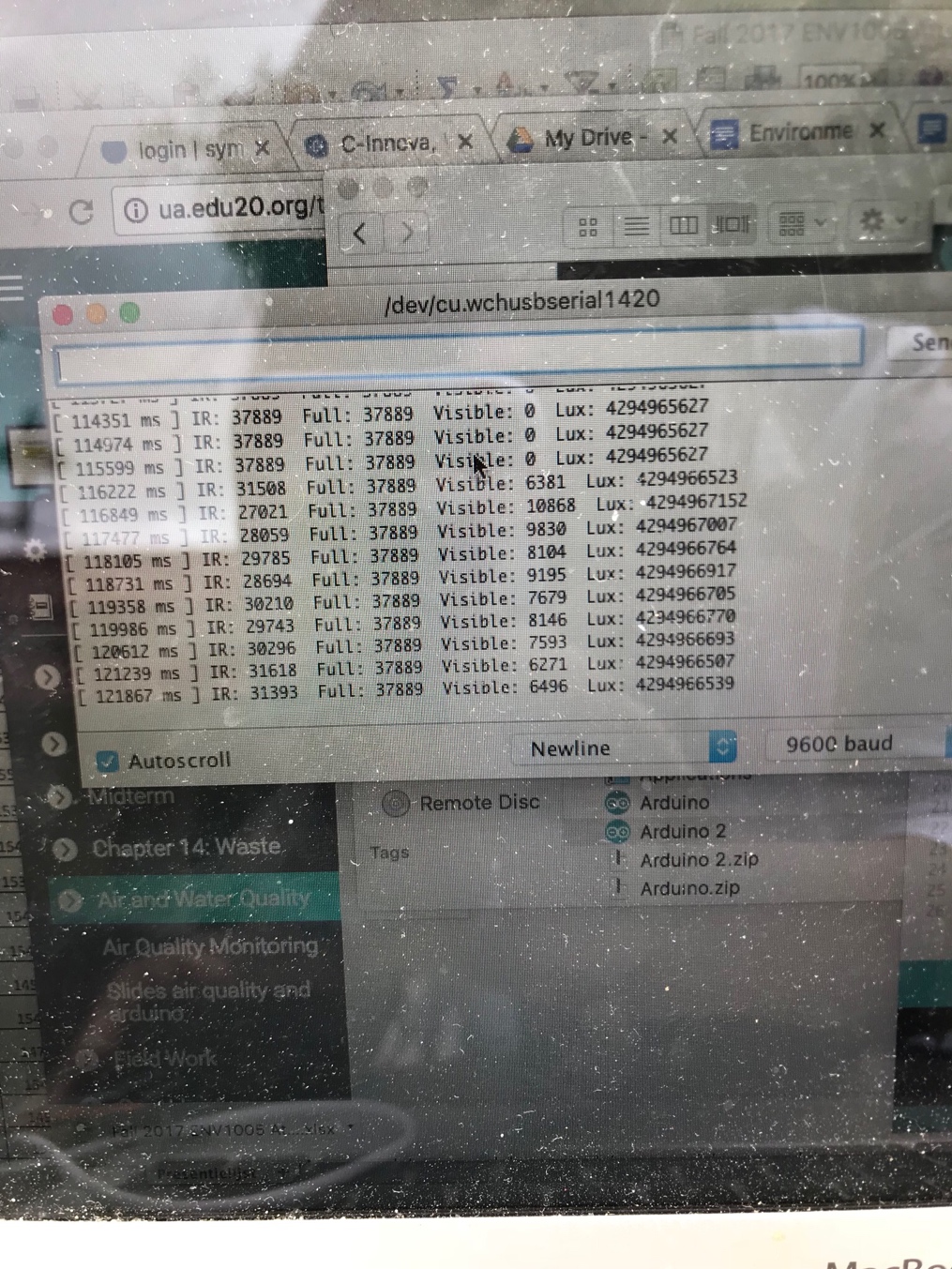
(115599 ms) IR: 37889 Full: 37889 Visible: 0 Lux: 4294965627

(116222 ms) IR: 31508 Full: 37889 Visible: 6381 Lux: 4294966523

(116849 ms) IR: 27021 Full: 37889 Visible: 10868 Lux: 4294967152

(117477 ms) IR: 28059 Full 37889 Visible: 9830 Lux: 4294967007

# Pictures



Code

#ifndef \_TSL2591\_H\_

#define \_TSL2591\_H\_

#if ARDUINO >= 100

#include <Arduino.h>d

#else

#include <WProgram.h>

#endif

#include <Adafruit\_Sensor.h>

#include <Wire.h>

#define TSL2591\_VISIBLE (2) // channel 0 - channel 1

#define TSL2591\_INFRARED (1) // channel 1

#define TSL2591\_FULLSPECTRUM (0) // channel 0

#define TSL2591\_ADDR (0x29)

#define TSL2591\_READBIT (0x01)

#define TSL2591\_COMMAND\_BIT (0xA0) // 1010 0000: bits 7 and 5 for 'command normal'

#define TSL2591\_CLEAR\_INT (0xE7)

#define TSL2591\_TEST\_INT (0xE4)

#define TSL2591\_WORD\_BIT (0x20) // 1 = read/write word (rather than byte)

#define TSL2591\_BLOCK\_BIT (0x10) // 1 = using block read/write

#define TSL2591\_ENABLE\_POWEROFF (0x00)

#define TSL2591\_ENABLE\_POWERON (0x01)

#define TSL2591\_ENABLE\_AEN (0x02) // ALS Enable. This field activates ALS function. Writing a one activates the ALS. Writing a zero disables the ALS.

#define TSL2591\_ENABLE\_AIEN (0x10) // ALS Interrupt Enable. When asserted permits ALS interrupts to be generated, subject to the persist filter.

#define TSL2591\_ENABLE\_NPIEN (0x80) // No Persist Interrupt Enable. When asserted NP Threshold conditions will generate an interrupt, bypassing the persist filter

#define TSL2591\_LUX\_DF (408.0F)

#define TSL2591\_LUX\_COEFB (1.64F) // CH0 coefficient

#define TSL2591\_LUX\_COEFC (0.59F) // CH1 coefficient A

#define TSL2591\_LUX\_COEFD (0.86F) // CH2 coefficient B

enum

{

TSL2591\_REGISTER\_ENABLE = 0x00,

TSL2591\_REGISTER\_CONTROL = 0x01,

TSL2591\_REGISTER\_THRESHOLD\_AILTL = 0x04, // ALS low threshold lower byte

TSL2591\_REGISTER\_THRESHOLD\_AILTH = 0x05, // ALS low threshold upper byte

TSL2591\_REGISTER\_THRESHOLD\_AIHTL = 0x06, // ALS high threshold lower byte

TSL2591\_REGISTER\_THRESHOLD\_AIHTH = 0x07, // ALS high threshold upper byte

TSL2591\_REGISTER\_THRESHOLD\_NPAILTL = 0x08, // No Persist ALS low threshold lower byte

TSL2591\_REGISTER\_THRESHOLD\_NPAILTH = 0x09, // etc

TSL2591\_REGISTER\_THRESHOLD\_NPAIHTL = 0x0A,

TSL2591\_REGISTER\_THRESHOLD\_NPAIHTH = 0x0B,

TSL2591\_REGISTER\_PERSIST\_FILTER = 0x0C,

TSL2591\_REGISTER\_PACKAGE\_PID = 0x11,

TSL2591\_REGISTER\_DEVICE\_ID = 0x12,

TSL2591\_REGISTER\_DEVICE\_STATUS = 0x13,

TSL2591\_REGISTER\_CHAN0\_LOW = 0x14,

TSL2591\_REGISTER\_CHAN0\_HIGH = 0x15,

TSL2591\_REGISTER\_CHAN1\_LOW = 0x16,

TSL2591\_REGISTER\_CHAN1\_HIGH = 0x17

};

typedef enum

{

TSL2591\_INTEGRATIONTIME\_100MS = 0x00,

TSL2591\_INTEGRATIONTIME\_200MS = 0x01,

TSL2591\_INTEGRATIONTIME\_300MS = 0x02,

TSL2591\_INTEGRATIONTIME\_400MS = 0x03,

TSL2591\_INTEGRATIONTIME\_500MS = 0x04,

TSL2591\_INTEGRATIONTIME\_600MS = 0x05,

}

tsl2591IntegrationTime\_t;

typedef enum

{

// bit 7:4: 0

TSL2591\_PERSIST\_EVERY = 0x00, // Every ALS cycle generates an interrupt

TSL2591\_PERSIST\_ANY = 0x01, // Any value outside of threshold range

TSL2591\_PERSIST\_2 = 0x02, // 2 consecutive values out of range

TSL2591\_PERSIST\_3 = 0x03, // 3 consecutive values out of range

TSL2591\_PERSIST\_5 = 0x04, // 5 consecutive values out of range

TSL2591\_PERSIST\_10 = 0x05, // 10 consecutive values out of range

TSL2591\_PERSIST\_15 = 0x06, // 15 consecutive values out of range

TSL2591\_PERSIST\_20 = 0x07, // 20 consecutive values out of range

TSL2591\_PERSIST\_25 = 0x08, // 25 consecutive values out of range

TSL2591\_PERSIST\_30 = 0x09, // 30 consecutive values out of range

TSL2591\_PERSIST\_35 = 0x0A, // 35 consecutive values out of range

TSL2591\_PERSIST\_40 = 0x0B, // 40 consecutive values out of range

TSL2591\_PERSIST\_45 = 0x0C, // 45 consecutive values out of range

TSL2591\_PERSIST\_50 = 0x0D, // 50 consecutive values out of range

TSL2591\_PERSIST\_55 = 0x0E, // 55 consecutive values out of range

TSL2591\_PERSIST\_60 = 0x0F, // 60 consecutive values out of range

}

tsl2591Persist\_t;

typedef enum

{

TSL2591\_GAIN\_LOW = 0x00, // low gain (1x)

TSL2591\_GAIN\_MED = 0x10, // medium gain (25x)

TSL2591\_GAIN\_HIGH = 0x20, // medium gain (428x)

TSL2591\_GAIN\_MAX = 0x30, // max gain (9876x)

}

tsl2591Gain\_t;

class Adafruit\_TSL2591 : public Adafruit\_Sensor

{

public:

Adafruit\_TSL2591(int32\_t sensorID = -1);

boolean begin ( void );

void enable ( void );

void disable ( void );

void write8 ( uint8\_t r);

void write8 ( uint8\_t r, uint8\_t v );

uint16\_t read16 ( uint8\_t reg );

uint8\_t read8 ( uint8\_t reg );

uint32\_t calculateLux ( uint16\_t ch0, uint16\_t ch1 );

void setGain ( tsl2591Gain\_t gain );

void setTiming ( tsl2591IntegrationTime\_t integration );

uint16\_t getLuminosity (uint8\_t channel );

uint32\_t getFullLuminosity ( );

tsl2591IntegrationTime\_t getTiming();

tsl2591Gain\_t getGain();

// Interrupt

void clearInterrupt(void);

void registerInterrupt(uint16\_t lowerThreshold, uint16\_t upperThreshold);

void registerInterrupt(uint16\_t lowerThreshold, uint16\_t upperThreshold, tsl2591Persist\_t persist);

uint8\_t getStatus();

/\* Unified Sensor API Functions \*/

bool getEvent ( sensors\_event\_t\* );

void getSensor ( sensor\_t\* );

private:

tsl2591IntegrationTime\_t \_integration;

tsl2591Gain\_t \_gain;

int32\_t \_sensorID;

boolean \_initialized;

};#endif