Годишник на секция "Информатика"
Съюз на учените в България
Том 6, 2013, 150–158

Annual of "Informatics" Section
Union of Scientists in Bulgaria
Volume 6, 2013, 150–158

СОФТУЕРНИ И ХАРДУЕРНИ РЕШЕНИЯ ЗА ИЗМЕРВАНЕ И ИЗСЛЕДВАНЕ НА ЯРКОСТ, ИНТЕНЗИТЕТ И НА НАЛИЧИЕ НА ШИРОЧИННО ИМПУЛСНА МОДУЛАЦИЯ НА СВЕТЛИНАТА

Апостол Маринов, Велиан Христов, Даниела Орозова

Бургаски свободен университет

Абстракт: В статията се разглеждат основни въпроси, свързани с измерване на светлинния интензитет и изследване за наличие или липса на широчинно импулсна модулация на светлинен източник чрез създаване на хардуерно и софтуерно решение.
Ключови думи: Светлинен интензитет, Lux метър, Светлинен поток, Светлинна ефективност, Широчинно импулсна модулация на светлината, Осцилоскоп.

1. Въведение

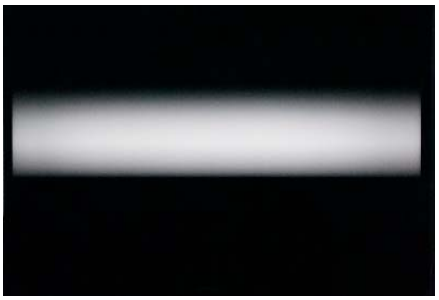
Невроните на ретината реагират при облъчване на светлина и тя се свива и разширява дори и при трептене, достигащо 120Hz. Въпреки това, възприятието за трептене не може да бъде установено от човек при тези честоти. Възможността за разграничаване на светлинно трептене се смята за ограничена, поради устоичивостта на зрението да запазва и наслагва послеобраза в ретината за приблизително 1/25 от секундата. Критичен праг на видимост на синтезирано светлинно трептене (CCT, *Critical flicker fusion, CCF*), е най-ниската честота на продължително трептене на светлината, която човек възприема за постоянна. Подобно трептене има при източници на светлина, използващи широчинно импулсна модулация (ШИМ, *Pulse Width Modulation, PWM*), за контрол на осветеността както и такива захранвани от AC мрежата. Такива източници могат да бъдат флуоресцентни CFL/CCFL и LED лампи, които се използват у дома и в офиса и при LCD мониторите.

Ако нивото на освеност на източник на светлина е 100cd/m^2 и се използва 100Hz ШИМ контрол за намаляне на видимата осветеност до 10% , то той ще се включи и изключи 100 пъти в секундата с продължителност на включване 1/1000 s (1 ms) и продължителност на изключване 9/1000 s (9 ms). Това ще доведе до постоянно свиване и разширяване на зеницата и в много случаи води до изморяване на очите, главоболия и при по-ниски честоти дори до епилептични пристъпи.

Неинертните и неизползващи баластри светлинни източници в дома и офиса са захранвани директно от 50Hz AC мрежата и поради това се генерира 100 Hz ШИМ. Мигането на CFL лампите с индуктивен баласт предизвиква и стробоскопичен ефект,

поради което е забранено в много страни за използване в производства, където има бързо движещи се обекти.

Използвайки фотоапарат с висока скорост на затвора – 1/3200 s, и визуализиране на бял образ върху LCD монитор можем да заснемем ШИМ цикъла на включване и изключване. На Фиг. 1 се вижда цикъла на включване и изключване на LED подсветката на Samsung LCD TV дисплей, използващ 133 Hz ШИМ, а на Фиг. 2 такъв на Sony, използващ аналогов DC контрол на LED подсветката (без ШИМ) или високочестотен WLED ШИМ драйвер (> 600 000 Hz). В този и много други случаи две различни марки телевизори или монитори използват LCD панел от един и същи производител, поради което ШИМ не зависи директно от панела, а от използваното решение за контрол на подсветката. Някои производители използват ШИМ дори и при най-скъпите/висококласни монитори и телевизори, като това никъде не се упоменава от производителя.



Фигура 1. LCD с ШИМ

Фигура 2. LCD без ШИМ

ШИМ се използва, защото е лесен и евтин начин за контрол на подсветката. Може да бъде реализиран чрез елементарен хардуер, PIC8/12/16 [2] контролер или чрез висококачествен драйвер като например TPS61195 WLED драйвер за LCD Backlighting With PWM and SMBus Control Interface, който дава възможност за използване на супер високи честоти от 600 kHz до 1 MHz. Съответно разликата в тази реализация може да коства от 2-3 $ и да достига до 100$. По тези причини е важна и възможността за изследване на осветеността за наличие или липса на ШИМ и нейната честота.

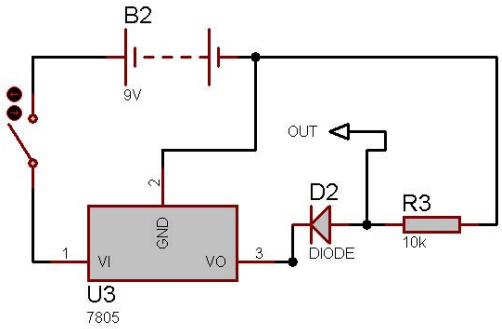
2. Хардуерна реализация

За детектор на светлина използваме силиконов фотодиод. Силиконовите фотодиоди са полупроводникови устройства, реагиращи на високоенергийни частици и фотони и генериращи поток на електричен ток към външна електрическа верига, пропорционален на съпътстващата сила. Фотодиодите могат се използват за откриване наличие или липса на минимални количества светлина и могат да се калибрират за екстремно точни измервания на интензивности дори под 1 pW/cm^2, както и до такива над

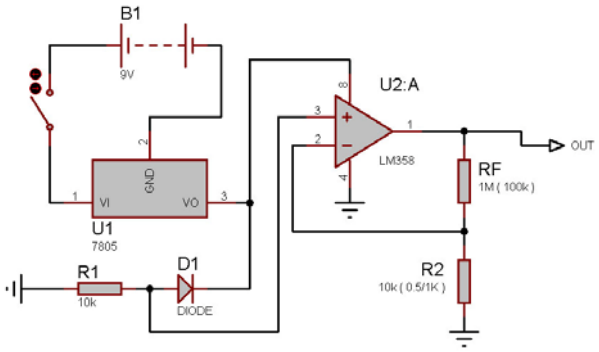
100 mW/cm^2. Силиконовите фотодиоди са широко употребявани в различни приложения като спектроскопия, фотография, аналитични контролно-измервателни уреди, оптично-позиционни сензори, подравняване на лъчи, характеризация на повърхности, измерване на обсега на лазери, оптични комуникации и образна медицинска техника, дори и Гайгерови броячи.

Избраният за целта фотодиод е BWP34 поради неговите характеристики: висока скорост и висока точка на чувствителност, чувствителен към видимата и близка до инфрачервената радиация, ъгъл на чувствителност 65 градуса (130 цялостен), дължина на вълната 40–110 nm, Rise Time – 100 ns (така можем да засечем честоти на трептене на светлината достигащи 10 000 Hz).

Фотодиодите могат да бъдат свързани фотоволтатично или фотопроводящо. Те имат високо съпротивление, когато са вързани фотопроводящо. При осветяване на диода с светлина това съпротивление намалява, следователно диодът може да бъде използван за детектор на светлина чрез следене на тока, минаващ през него, както е показано на Фиг. 3а. За да усилим сигнала, можем да използваме схема с операционен усилвател и така в съответствие с RF, R2 можем да контролираме коефициента на усилване: Af = 1 + RF/R2 пъти. На мястото на единия резистор можем да използваме потенциометър и така ще имаме възможност за контрол на степента на усилване – Фиг 3.б.



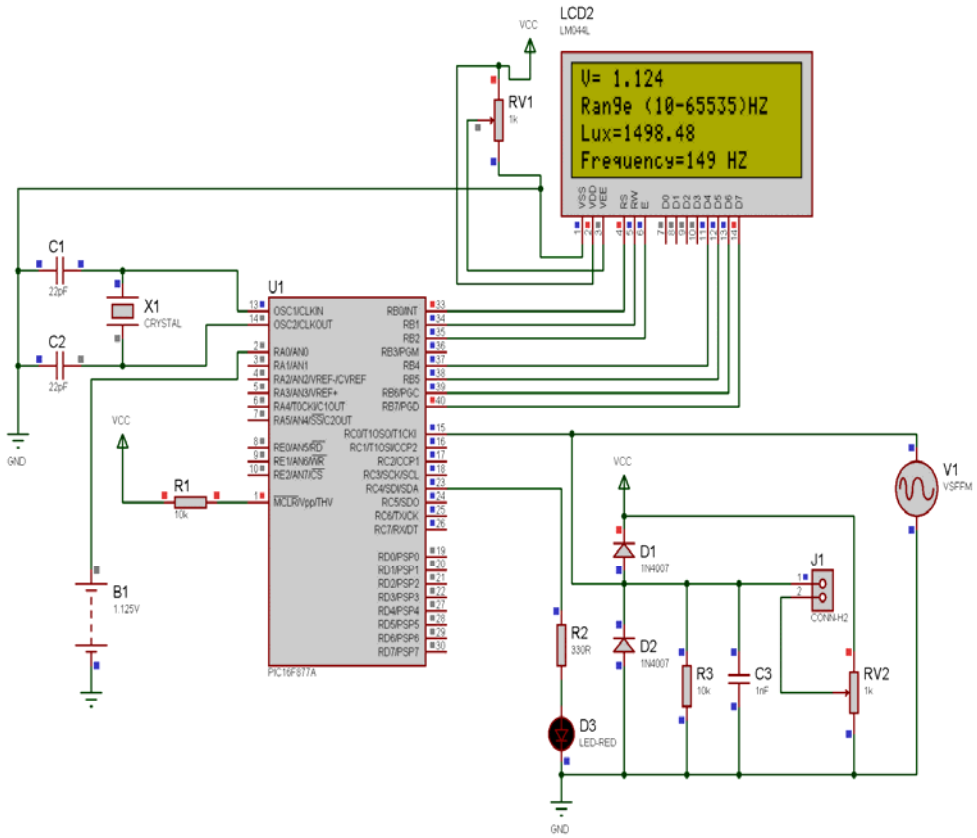
Фиг. 3а. Фотодиод без усилвател



Фиг. 3 б. Фотодиод с усилвател

Съществуват различни варианти за реализация, наблюдение и визуализация на сигнала. Тук разглеждаме следните варианти:

Вариант 1: Използване на PIC16F877A микроконтролер и 4x20 LCD дисплей за показване на осветеност в LUX и честота на трептенето, показани на Фиг. 4.



Фигура 4. Визуализация на примерна схема за измерване на lux и честота.

Тук е нужно да се добави обработка и разделяне на сигнала от усилвателя:

- да се стабилизира (при наличие на трептене) за RA0/AN0, като бъде свързан на мястото на B1.
- да се използва тригер на Шмит, за да получим чисти TTL (0/5 V) нива на таймера T1 (свързан на мястото на V1).

Тази схема може се опрости за измерване на lux, като използваме оптоелектронен сензор (TSL235R-LF), преобразуващ светлината в честота с TTL нива (0 и 5V) и брояч/таймер RA4/T0.

Фигурите 3а, 3б и 4 са разработени чрез продукта ***Protheus*** – софтуер за моделиране и симулация на схеми.

Следва програмен код на езика за програмно осигуряване C и компилатор на CSS – PIC C, използван за генериране на .hex във PIC контролера от фигура 4.

```
   int16 adc_value;
   float volts,lux;
   setup_adc_ports(AN0);
   setup_adc(ADC_CLOCK_DIV_64);   //  тъй като честотата е 20Mhz и трябва да имаTAd time 1.6us
   set_adc_channel(0);
   delay_us(20);   //слага се преди първото четене
   adc_value = read_adc();

   while(1)
   {
      volts = (float)(adc_value * 5)/1023.0;
      lux=(float)(volts*1333);  //1333 е стойност която се променя за калибриране на lux
       lcd_gotoxy(4,1);
       printf(lcd_putc, "%3.3f", volts);
       lcd_gotoxy(5,3);
       printf(lcd_putc,"%4.2f",lux);

       set_timer1(0);
       setup_timer_1(t1_external | T1_DIV_BY_1);

       delay_ms(1000);

      setup_timer_1(T1_DISABLED);
      value=get_timer1();
      lcd_gotoxy(11,4);
      printf(lcd_putc,"%LU HZ   ",value);
      led=!led;
   }
}
```

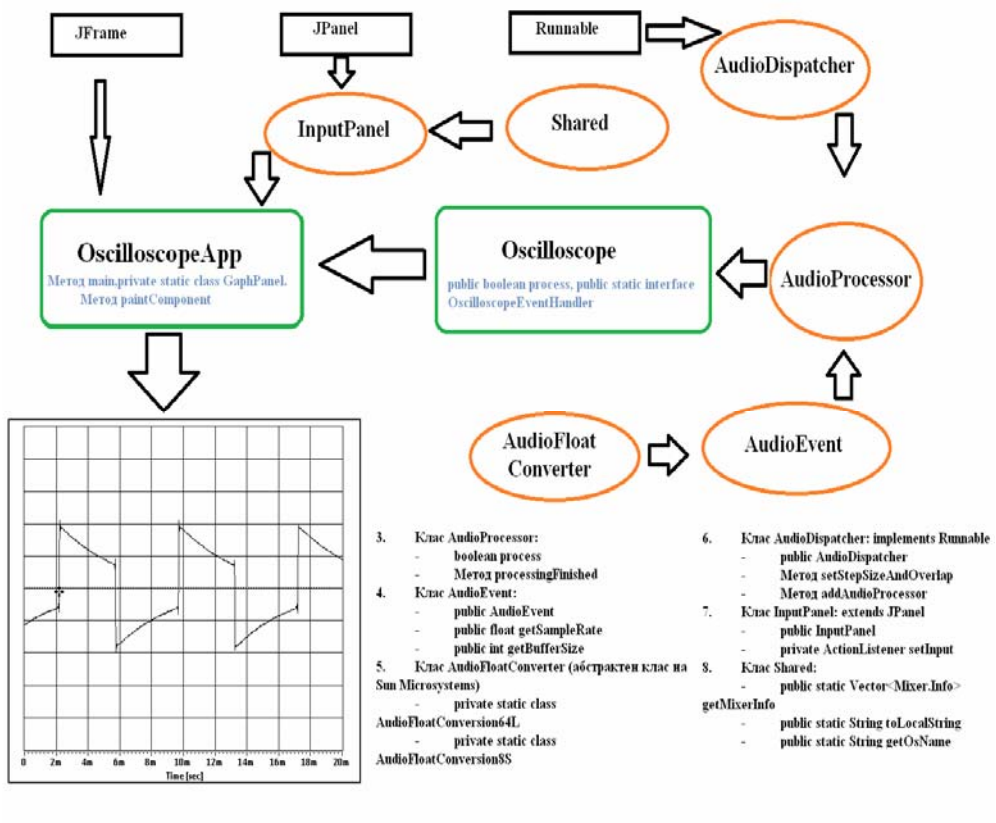
Вариант 2: Друго решение е чрез конструиране на осцилоскоп с използване на PIC микроконтролер, графичен дисплей и езици за програмно осигуряване – асемблер и C. Подобно решение е по-непрактично и доста сложно за потребителя, а закупуването на такъв осцилоскоп е скъпо. Именно затова фокусът на тази разработка е в третия предложен вариант.

Вариант 3: Разработване на софтуерно решение, използвайки език за програмно осигуряване Java и персонален компютър, лаптоп, таблет или смартфон, като използваме сигнала, представен на Фиг. 3а или 3б, и аудио входа към звуковата карта.

За целта използваме готови JAVA библиотеки [1] TarsosDSP за обработка на аудио поток, чиято цел е да осигури лесен за употреба инферфейс с алгоритми за практическа обработка на аудио сигнал чрез Java код без използване на външни библиотеки. TarsosDSP включва алгоритми за откриване на внезапни нива: YIN, Mcleod Pitch метод. Също е включен и Goertzel DTMF декодиращ алгоритъм, и алгоритъм за разтягане времето (WSOLA), пресемплиране, филтри, синтез, аудио ефекти и *pitch shifting* алгоритъм.

За това изследване създадохме Java-пакет ***Osc***. В него дефинирахме класовете: *AudioDispatcher*, *AudioEvent*, *AudioFloatConverter*, *AudioProcessor*, *InputPanel*, *Oscilloscope*, *Shared* и главен клас *OscilloscopeApp*.

На Фиг. 5 е дадена блокова схема, която показва схематично компонентите на софтуерния осцилоскоп, визуализиращ графично аудио-потока за определен период от време. За програмната реализация е използван език за програмиране Java.



Фигура 5. Блокова схема на софтуерен осцилоскоп

OscilloscopeApp реализира интерфейса *OscilloscopeEventHandler* от класа *Oscilloscope.java*. Следва фрагмент от кода, чрез който се създава обект от класа *InputPanel*;

```
JPanel inputPanel = new InputPanel();
//add(inputPanel);
inputPanel.addPropertyChangeListener("mixer",
        new PropertyChangeListener() {
            @Override
            public void propertyChange(PropertyChangeEvent arg0) {
                try {
                    setNewMixer((Mixer) arg0.getNewValue());
                } catch (LineUnavailableException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (UnsupportedAudioFileException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
```

GaphPanel наследява класа JPanel и описва формулата за рисуване на графиката със съответна дължината и височината:

```
private static class GaphPanel extends JPanel{

    private static final long serialVersionUID = 4969781241442094359L;

    float data[];

    public GaphPanel(){
        setMinimumSize(new Dimension(80,60));
    }

    public void paintComponent(Graphics g) {

        super.paintComponent(g); //paint background
        g.setColor(Color.BLACK);
        g.fillRect(0, 0,getWidth(), getHeight());
        g.setColor(Color.WHITE);
        if(data != null){
            float width = getWidth();
            float height = getHeight();
            float halfHeight = height / 2;
            for(int i=0; i < data.length ; i+=4){
                 g.drawLine((int)(data[i]* width),(int)( halfHeight - data[i+1]* height),
                         (int)( data[i+2]*width),(int)( halfHeight - data[i+3]*height));
            }
        }
    }

    public void paint(float[] data, AudioEvent event){
        this.data = data;
    }
}
```

В метода *setNewMixer* се задават параметрите на *sampleRate* и размера на буфера. Ако се използва буфер 4410, то *sampleRate* ще се визуализира 10 пъти в секундата, като всяко визуализиране ще показва 4410 стойности от всичките 44100.

```
private void setNewMixer(Mixer mixer) throws LineUnavailableException,
        UnsupportedAudioFileException {

    if(dispatcher!= null){
        dispatcher.stop();
    }
    currentMixer = mixer;

    float sampleRate = 44100;
    int bufferSize = 2048*4;
    int overlap = 0;

    final AudioFormat format = new AudioFormat(sampleRate, 16, 1, true,
            true);
    final DataLine.Info dataLineInfo = new DataLine.Info(
            TargetDataLine.class, format);
    TargetDataLine line;
    line = (TargetDataLine) mixer.getLine(dataLineInfo);
    final int numberOfSamples = bufferSize;
    line.open(format, numberOfSamples);
    line.start();
    final AudioInputStream stream = new AudioInputStream(line);
```

```
public InputPanel(){
    super(new BorderLayout());
    this.setBorder(new TitledBorder("1. Choose a microphone input"));
    JPanel buttonPanel = new JPanel(new GridLayout(0,1));
    ButtonGroup group = new ButtonGroup();
    for(Mixer.Info info : Shared.getMixerInfo(false, true)){
        JRadioButton button = new JRadioButton();
        button.setText(Shared.toLocalString(info));
        buttonPanel.add(button);
        group.add(button);
        button.setActionCommand(info.toString());
        button.addActionListener(setInput);
    }
    this.add(new JScrollPane(buttonPanel,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER),BorderLayout.CENTER);
    this.setMaximumSize(new Dimension(300,150));
    this.setPreferredSize(new Dimension(300,150));
}
```

Oscilloscope.java наследява интерфейса *AudioProcessor*. Генерира масив със следните елементи: array[i] е x координата в проценти; array[i+1] е стойността на амплитудата в аудиобуфера; array[i+2] е другата x координата в проценти, а array [i+3] следващата амплитуда.

AudioProcessor използва булева функция *process(AudioEvent audioEvent)*, за да обработва аудио-събитията и актуалния аудио-сигнал. Ако върне *False*, то няма сигнал и може да се използва като детектор за тишина и прекъсва. В противен случай връща *True*.

AudioEvent е клас описващ методите за протичане на аудио-сигнала през конвейера за обработка. Класът използва *AudioFloatConverter* и библиотеката на Java *javax.sound.sampled.AudioFormat*.

AudioFloatConverter се използва за конвертиране между 8, 16, 24, 32, 32+ битов голям/малък ендиан, фиксирана и плаваща запетая, байтов буфер и плаващ буфер.

AudioDispatcher класа изпраща масивите до регистрираните *AudioProcessor* имплементатори, в резултат на което е възможно синхронизираното изпълнение на аудио-процесори и звук. *InputPanel* класа наследява *JPanel* и описва графичния интерфейс на приложението, като дава възможност за избор на входен сигнал.

Shared класа проверява какви устройства има инсталирани на машината и дава вътрешна информация за операционната система и хардуера .

3. Заключение

В заключение можем да обобщим, че най-малко дразнене на очите ще се наблюдава от източници на светлина, неизползващи ШИМ или такива, чиято ШИМ честота е над 10 000 Hz.

Такъв тип изследвания дават възможност за правилен избор при закупуване на даден светлинен източник, независимо от неговия вид и естество. При излагане на светлинно излъчване от източник, генериращ рязко изменящи се пулсации чрез широчинно импулсна модулиция, както и на прекалено ниски или високи нива на осветеност, може да се получи преждевременна умора и увреждане на зрението, главоболие и дискомфорт.

Литература

[1] TarsosDSP, `http://tarsos.0110.be/tag/TarsosDSP.`

[2] PWM Sets Output of LCD/LED Driver, Application Note 3326, Sep 20, 2004. `http://www.maximintegrated.com/app-notes/index.mvp/id/3326`.

[3] `http://www.cssinfo.com`

[4] Proteus Design Suite, `http://www.labcenter.com/`.