Daftar Pustaka

[1] Kaur, Singh, Ubhi, & Rani. 2011. "*Digital Filteration of ECG Signal for Removal of Baseline Drift*". International Conference on Telecommunication Technology and Applications

[2] Kugelstadt, T. 2005. "*Getting the most out of your instrumentation amplifier design*". Texas Instrument Analog Applications Journal

[3] M.W. Hann. "*Ultra Low Power, 18 bit Precision ECG Data Acquistion System*". Texas Instrument Precision Design

[4] Merck, "*Electrical Injuries*". 1997. Pennsylvania : Merck.

[5] Ogata, K. 1970. "*Modern Control Engineering*". New Jersey : Prentice Hall

[6] Pan, J. Tompkins, W.J. 1985. "*A Real-Time QRS Detection Algorithm*". IEEE Transactions on Biomedical Engineering

[7] Townsend, Neil. 2001. "*Medical Electronics*"

[8] USB.org. 2000. "*Universal Serial Bus Spesification*". USB.org

[9] USB.org. 2001. "*Device Class Definition for Human Interface Devices (HID)*". USB.org

[10] Van Valkenburg, M.E. 1982. "*Analog Filter Design*". Oxford University Press

[11] Winter, B.B. Webster, J.G. 1983. "*Driven-Right-Leg Circuit Design*". IEEE Transactions on Biomedical Engineering

Lampiran A : USB *Device Descriptor*

Device Descriptor

Item	Keterangan
bLength	Panjang *device descriptor*, selalu 18 *byte*
bDescriptorType	Tipe 1 (*device descriptor*)
bcdUSB	Versi USB (1.0, 1.1, atau 2.0) dala m BCD
bDeviceClass	Nilai tergantung pada kelas *device*
bDeviceSubClass	Perincian kelas *device*
bDeviceProtocol	Kualifikasi kelas *device*
bMaxPacketSize	Jumlah data paket maksimum
idVendor	Identifikasi pembuat USB
idProduct	Identifikasi produk USB berdasarkan pembuat
bcdDevice	Seri *device* dalam BCD
iManufacturer	*String descriptor* pembuat USB
iProduct	*String descriptor* produk USB
iSerialNumber	*String descriptor* seri *device*
bNumConfigurations	Jumlah *configuration descriptor*

Configuration Descriptor

Item	Keterangan
bLength	Panjang *configuration descriptor*, selalu 9 *byte*
bDescriptorType	Tipe 2 (*configuration descriptor*)
wTotallength	Panjang keseluruhan *device descriptor*
bNumInterfaces	Jumlah antarmuka yang didukung
bConfigurationValue	Nilai untuk argumen *request* SET_CONFIG
iConfiguration	Indeks untuk *string descriptor*
bmAttributes	Sifat khusus konfigurasi
bMaxPower	Arus maksimum pada USB *bus*

Interface Descriptor

Item	Keterangan
bLength	Panjang *interface descriptor*, selalu 9 *byte*
bDescriptorType	Tipe 4 (*interface descriptor*)
bInterfaceNumber	Indeks antarmuka
bAlternateSetting	Aturan khusus pada antarmuka
bNumEndpoints	Jumlah *endpoint* yang digunakan antarmuka
bInterfaceClass	Serupa dengan *device descriptor*
bInterfaceSubClass	Serupa dengan *device descriptor*
bInterfaceProtocol	Serupa dengan *device descriptor*
iInterface	Indeks *string descriptor* untuk antarmuka

Lampiran B : HID *Report Descriptor*

Report Descriptor disusun berdasarkan aturan USB.org, yang tersusun atas tiga tipe *item* : *Main, Global*, dan *Local*. *Main item* meliputi *Input*, *Output*, *Feature*, *Collection*, dan End Collection. *Global* dan *Local item* menjelaskan properti *Main item*.

Contoh di bawah adalah HID *Report Descriptor* dari USB-ECG. Mengingat ada banyak sekali konten *report descriptor*, maka pada bagian ini hanya diberikan satu contoh saja. Rincian lengkap dari HID *report descriptor* dapat dilihat pada http://www.usb.org/developers/hidpage.

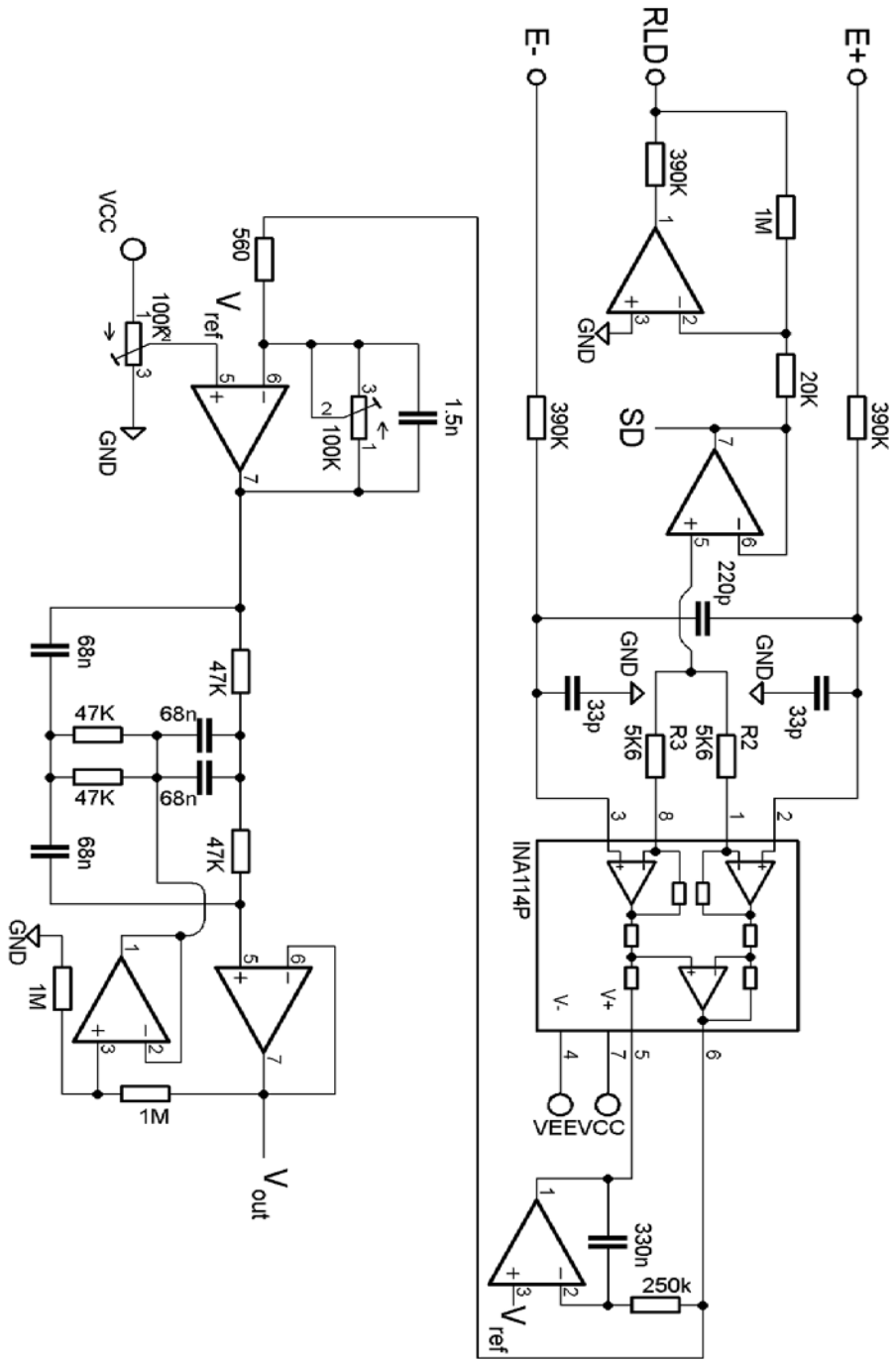
```
0x06, 0x00, 0xFF,       // USAGE_PAGE (Vendor Defined Page 1); Global Item
0x09, 0x01,             // USAGE (Vendor Usage 1); Local Item
0xA1, 0x01,             // COLLECTION (Application); Main Item
    0x15, 0x00,         // LOGICAL_MINIMUM (0); Global Item
    0x26, 0xFF, 0x00,   // LOGICAL_MAXIMUM (255); Global Item
    0x75, 0x08,         // REPORT_SIZE (8); Global Item
    0x95, 0x02,         // REPORT_COUNT (2); Global Item
    0x09, 0x00,         // USAGE (Undefined) ; Local Item
    0xB2, 0x02, 0x01,   // FEATURE (Data,Var,Abs,Buf); Main Item
    0xC0                // END_COLLECTION (Application); Main Item
```

Karena USB-ECG adalah *custom device*, maka digunakan *USAGE* dan *USAGE PAGE* yang bersifat *Vendor Defined*. *COLLECTION* menandakan awal dari suatu *report data*. Nilai *report* dibatasi dari 0 hingga 255.

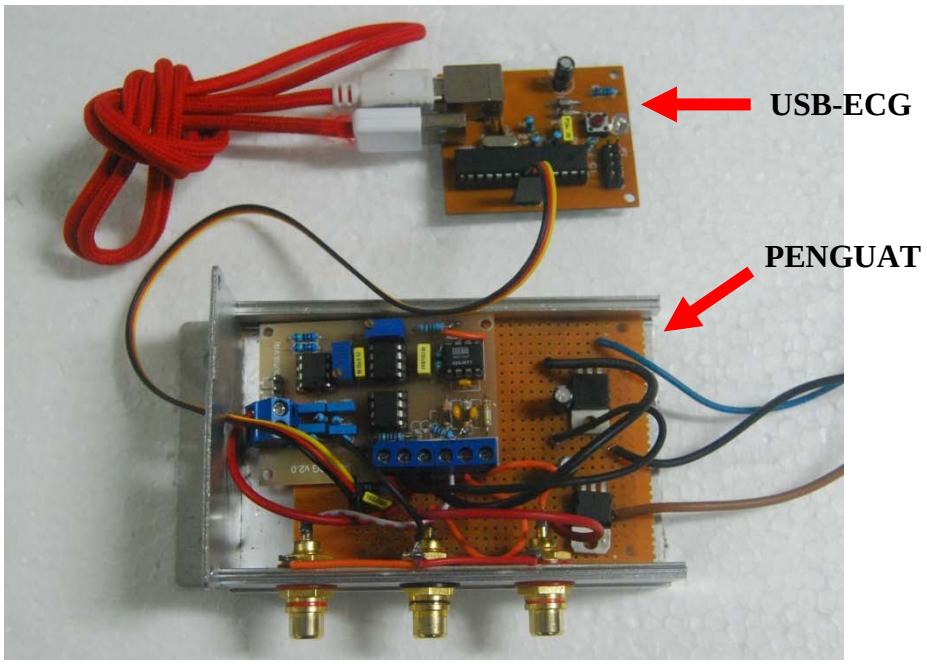
Panjang *report* 8 bit, dengan *count* 2. Artinya dalam satu kali transmisi, *report buffer* berisi 16 bit data. Karena fungsinya tidak spesifik, maka diberi *USAGE Undefined*. Tipe *report* adalah *FEATURE* (*bidirectional*). *END COLLECTION* menandakan akhir dari *report data*.

Lampiran C : Skematik dan PCB USB-ECG

1) Skematik Penguat



2) Realisasi Alat



3) Tes Alat

Hasil monitoring pada subyek, dilihat pada osiloskop dan GUI

Lampiran D : *Listing* Program

1) *Device-side Software*

```
#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include "usbdrv.h"
#include "func_init.h"

/*-----------------------USB INTERFACE--------------------------*/
PROGMEM const char usbHidReportDescriptor[22] = { //22
      0x06, 0x00, 0xff,              // USAGE_PAGE (Vendor Defined
Page 1)
    0x09, 0x01,                     // USAGE (Vendor Usage 1)
    0xa1, 0x01,                     // COLLECTION (Application)
    0x15, 0x00,                     //   LOGICAL_MINIMUM (0)
    0x26, 0xff, 0x00,               //   LOGICAL_MAXIMUM (255)
    0x75, 0x08,                     //   REPORT_SIZE (8)
    0x95, 0x02,                     //   REPORT_COUNT (2)
    0x09, 0x00,                     //   USAGE (Undefined)
    0xb2, 0x02, 0x01,               //   FEATURE (Data,Var,Abs,Buf)
    0xc0                            // END_COLLECTION
};

static uchar bytesRemaining;

static int adc_w;
static uchar adc_h;
static uchar adc_l;
```

```
usbMsgLen_t usbFunctionSetup(uchar data[8])
{
usbRequest_t *rq = (void *)data;
    if((rq->bmRequestType & USBRQ_TYPE_MASK) == USBRQ_TYPE_CLASS)
    {    /* HID class request */
        if(rq->bRequest == USBRQ_HID_GET_REPORT) {
            bytesRemaining = 2;
            return USB_NO_MSG;
        }
    }
    return 0;
}

uchar usbFunctionRead(uchar *data, uchar len)
{
      if(len > bytesRemaining)
            len = bytesRemaining;
      bytesRemaining -= len;
      *(data) = adc_h;
      *(data+1) = adc_l;

      return len;

}

/*---------------------------MAIN LOOP--------------------------*/
int main(void) {
uchar i;
    wdt_enable(WDTO_1S);
    usbInit();
    usbDeviceDisconnect();
    i = 0;
    while(--i) {
        wdt_reset();
        _delay_ms(1);
    }
    usbDeviceConnect();
```

```
    adc_init();
    sei();
    for(;;) {
        wdt_reset();
        usbPoll();
        adc_w = read_adc();
        adc_h = (adc_w >> 8);
        adc_l = adc_w & 0xff;
    }
    return 0;
}

/* ------------------------------------------------------------ */
```

2) *Host-side Software*

```
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "hiddata.h"
#include "../firmware/usbconfig.h"
#include "stdafx.h"

/*Kalau ada Error*/
static char *usbErrorMessage(int errCode)
{
static char buffer[80];
switch(errCode) {
case USBOPEN_ERR_ACCESS:      return "Access to device denied";
case USBOPEN_ERR_NOTFOUND:    return "Specified device not found";
case USBOPEN_ERR_IO:          return "Communication error with
                                      device";
default:    sprintf(buffer, "Unknown USB error %d", errCode);
return buffer;
}
return NULL;    /* not reached */
}
```

```
/*Cari USB Device dengan VID dan PID yang sesuai*/
static usbDevice_t *openDevice(void)
{
usbDevice_t     *dev = NULL;
unsigned char   rawVid[2] = {USB_CFG_VENDOR_ID}, rawPid[2] =
{USB_CFG_DEVICE_ID};
char            vendorName[] = {USB_CFG_VENDOR_NAME, 0},
productName[] = {USB_CFG_DEVICE_NAME, 0};
int             vid = rawVid[0] + 256 * rawVid[1];
int             pid = rawPid[0] + 256 * rawPid[1];
int             err;

if((err = usbhidOpenDevice(&dev, vid, vendorName, pid,
    productName, 0)) != 0) {
fprintf(stderr, "1. Error finding %s: %s\n", productName,
         usbErrorMessage(err));
return NULL;
}
else {
      fprintf(stderr, "2. No Error\n");
      return dev;
}
}

static int data_buff, bpm_buffer;

static void DataDump(char *buffer)
{
data_buff = (unsigned int)buffer[0]*256 + (unsigned int)buffer[1];
data_buff = Filter(data_buff);
bpm_buffer = PanTom(data_buff);
fprintf(stdout,"%d\n",data_buff);
}
```

```
int main(int argc, char **argv) {
clock_t start;
usbDevice_t *dev;
char buffer[3];
int err;

if(argc < 2) {
      exit(1);
}

if((dev = openDevice()) == NULL) exit(1);

int len = sizeof(buffer);

if(strcasecmp(argv[1], "read") == 0) {
   start = clock();

   while(1) {
      if (difftime(clock(),start) >= 5) {
         if((err = usbhidGetReport(dev, 0, buffer, &len)) == 0) {
            DataDump(buffer + 1);
            BPM_count(bpm_buffer);
         }
      start = clock();
      }
   }
}

usbhidCloseDevice(dev);
return 0;

}
```

3) *Graphical User Interface*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO;
using System.Diagnostics;
using System.Windows.Forms;

namespace USB_ECG_v10
{

    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
        }

        public Pen drawPen;
              public Bitmap imgGraph;
              public Graphics gpcGraph;
              public GraphicsPath gpcPath;
              public bool IsStarted, IsRec;
              public int x0, y0, xt, yt, x_null, y_null;
              public int x_scale, y_scale, bpm;
              public String bpm_buffer;

        private void Form1_Load(object sender, EventArgs e)
        {
              drawPen = new Pen(Color.Yellow);
              drawPen.Width= 2;
              imgGraph = new Bitmap(pb_Graph.Width,pb_Graph.Height);
              pb_Graph.Image = imgGraph
              gpcGraph = Graphics.FromImage(imgGraph);
              gpcPath = new GraphicsPath();

              x0 = 0;
              xt = 0;
              y0 = 0;
              yt = 0;

              x_null = pb_Graph.Width;
              y_null = pb_Graph.Height/2;
              x_scale = 0;
              y_scale = 3;

        }
```

```
private void bt_Start_Click(object sender, EventArgs e)
{
    if (!IsStarted)
    {
        if (proc.Start())
        {
            IsStarted = true;
            proc.BeginErrorReadLine();
            proc.BeginOutputReadLine();
        }
    }

    if (!IsRec)
    {
        bt_Start.Text = "STOP MONITORING";
        bt_Start.ForeColor = Color.Red;
        IsRec = true;
    }

    else
    {
        bt_Start.Text = "START MONITORING";
        bt_Start.ForeColor = Color.Black;
        IsRec = false;
    }

}

private void outputData(object sender, DataReceivedEventArgs e)
{
    if (IsRec)
    {
        if (String.IsNullOrEmpty(e.Data) == false)
        {
            new Thread(() =>
            {
            this.label1.Invoke(new Action(() =>
            {
                if (e.Data.StartsWith("#") == false)
                {
                    if (xt == pb_Graph.Width)
                    {
                        x0 = 0;
                        xt = 1;
                        gpcPath.Reset();
                    }
                    else
                    {
                        x0 = xt;
                        xt++;
                    }

                    y0 = yt;
                    yt = Convert.ToInt16(e.Data);
```

```
                    gpcPath.AddLine(x0 >> x_scale,
                                    y_null - (y0 >> y_scale),
                                    xt >> x_scale,
                                    y_null - (yt >> y_scale));
                    gpcGraph.Clear(Color.Transparent);
                    gpcGraph.DrawPath(drawPen, gpcPath);
                    pb_Graph.Refresh();
                }

                else
                {
                    bpm_buffer = e.Data.Substring(1);
                    lbl_BpmCount.Text = bpm_buffer;
                    bpm = Convert.ToInt16(bpm_buffer);

                    if (bpm > 110) {
                           lbl_Brady.Visible = false;
                           lbl_Tachy.Visible = true;
                           lbl_Asystole.Visible = false;
                    } else if (bpm < 60) {
                           lbl_Brady.Visible = true;
                           lbl_Tachy.Visible = false;
                           lbl_Asystole.Visible = false;
                    } else if (bpm == 0) {
                           lbl_Brady.Visible = false;
                           lbl_Tachy.Visible = false;
                           lbl_Asystole.Visible = true;
                    } else {
                           lbl_Brady.Visible = false;
                           lbl_Tachy.Visible = false;
                           lbl_Asystole.Visible = false;
                    }
                    }
                }));
            }).Start();
        }
    }
}

private void errorData(object sender, DataReceivedEventArgs e)
{
    if (e.Data.StartsWith("1"))
    {
        if (MessageBox.Show("An error has occured. Reconnect USB-
                            ECG and restart the application!",
                            "Error",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Error) ==
                            DialogResult.OK) IsStarted = false;

        proc.CancelErrorRead();
        bt_Start.Text = "START MONITORING";
        bt_Start.ForeColor = Color.Black;
        IsRec = false;
        proc.CancelOutputRead();
    }
```

```
            else
            {
                proc.CancelErrorRead();
                IsRec = true;
            }
        }

      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
              if (IsStarted && proc.HasExited == false)
              {
                     proc.CancelOutputRead();
                     proc.Kill();
              }
      }
    }
}
```

Source code lengkap dengan *linker dependencies* dapat diakses pada CD Tugas Akhir.