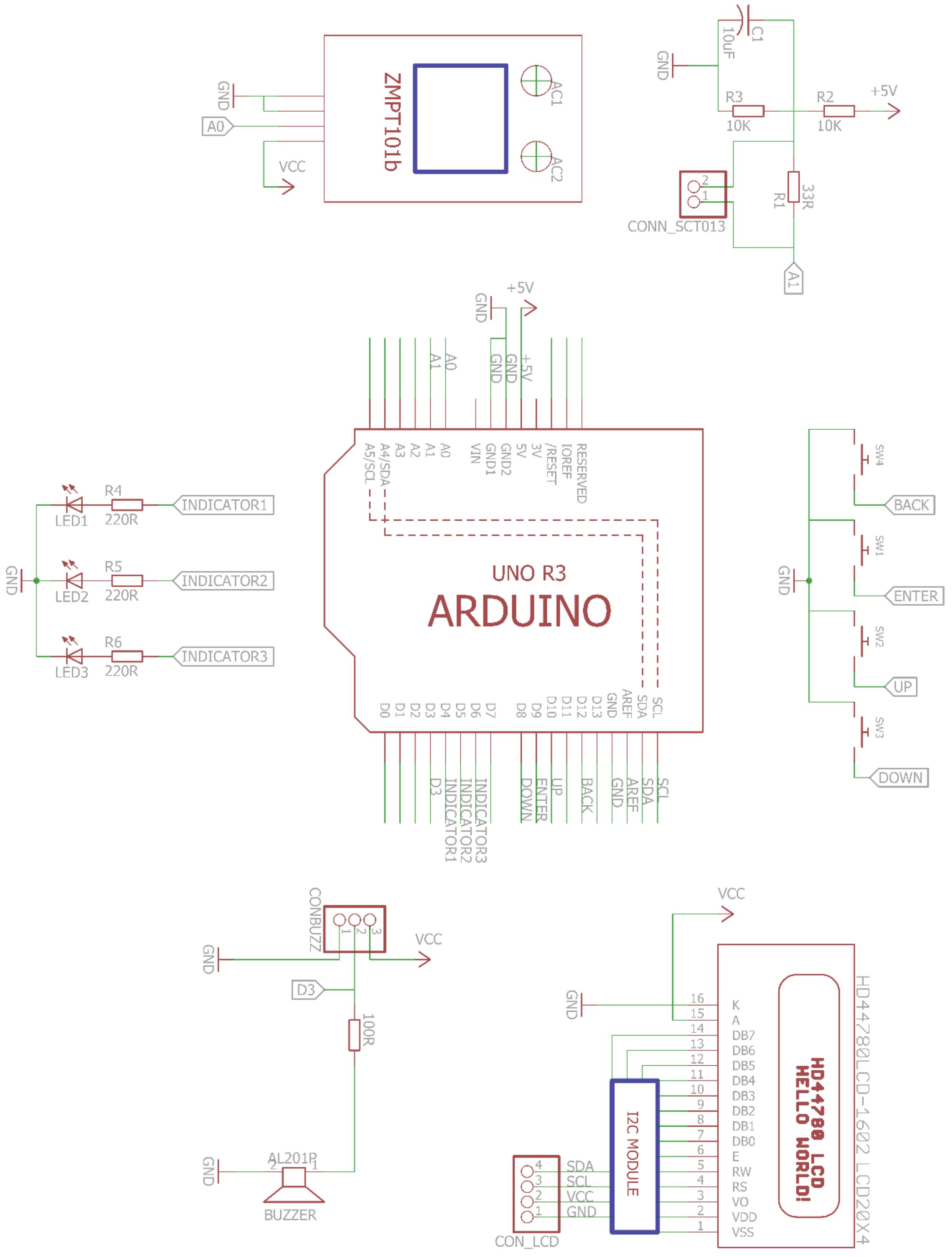
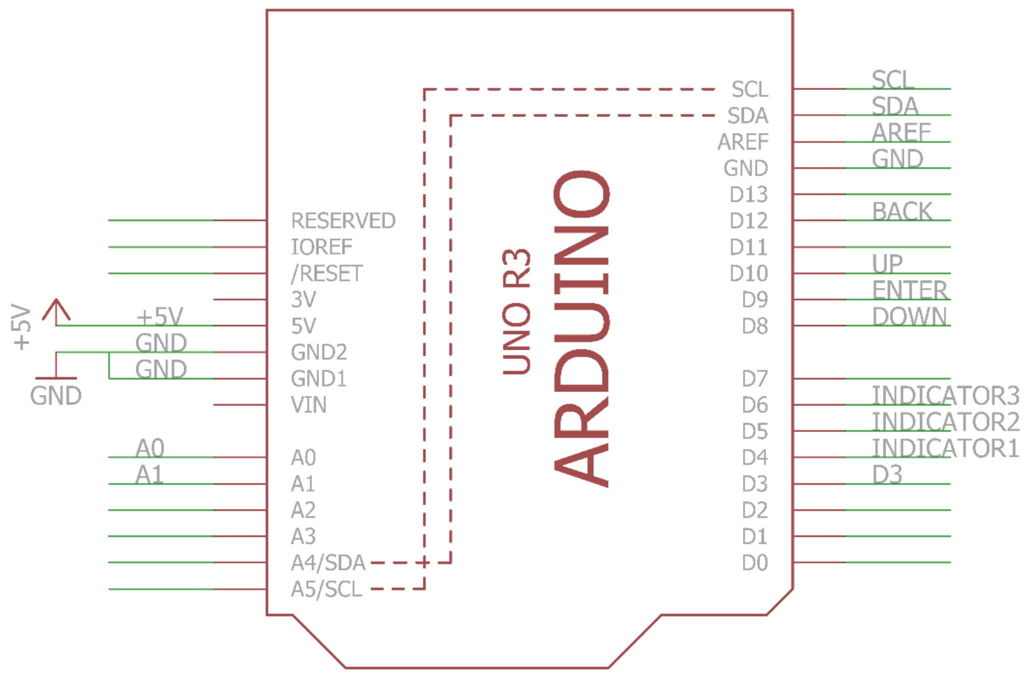
LAMPIRAN SKEMATIK ALAT

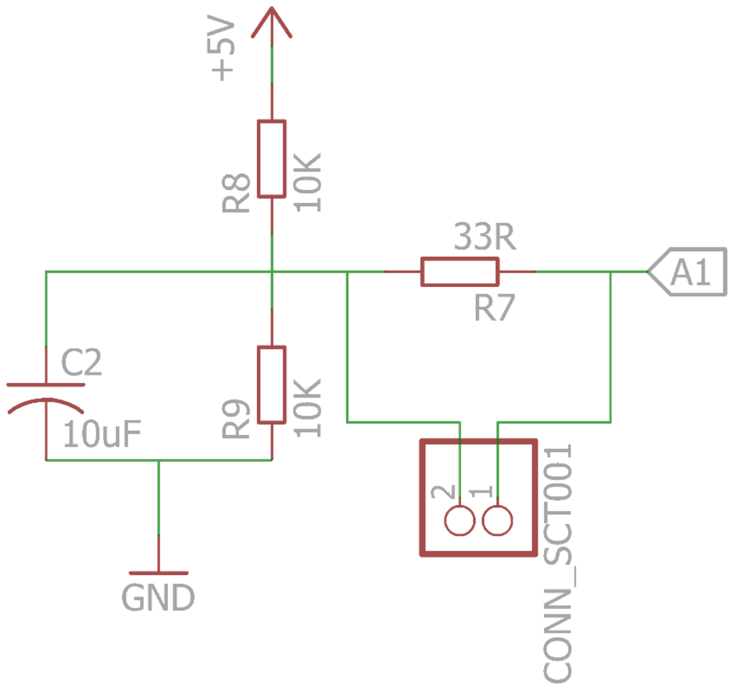
1. Skematik Keseluruhan



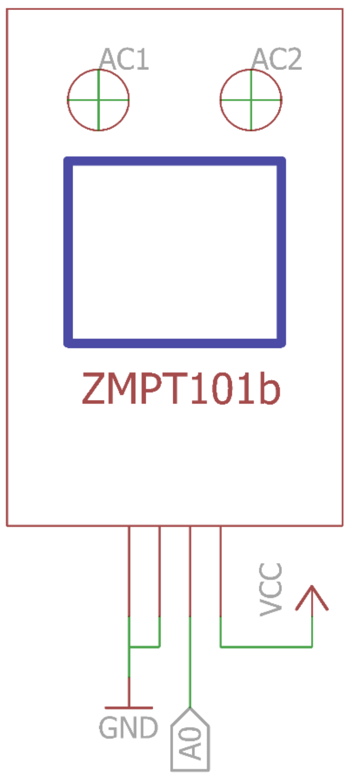
2. **Sistem *Board* Arduino**



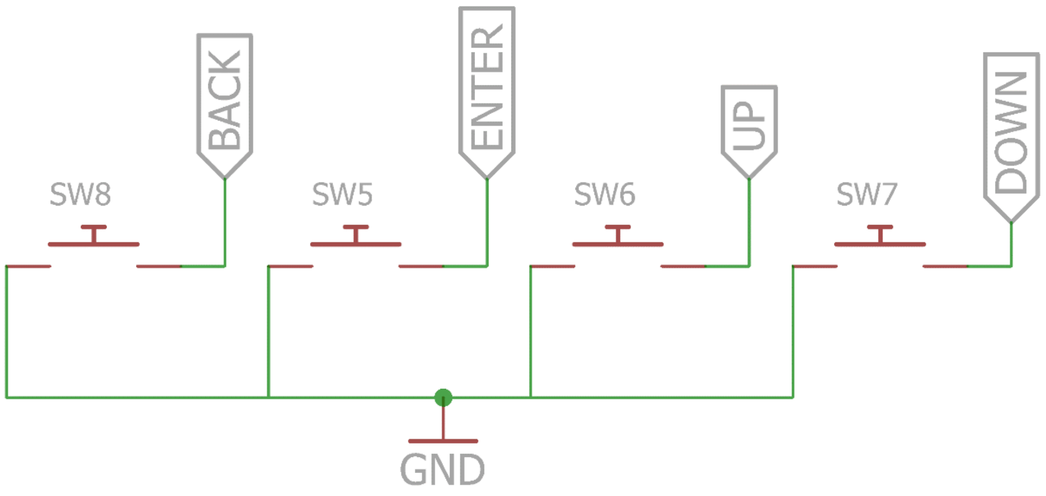
3. **Sistem Konverter SCT 013-000**



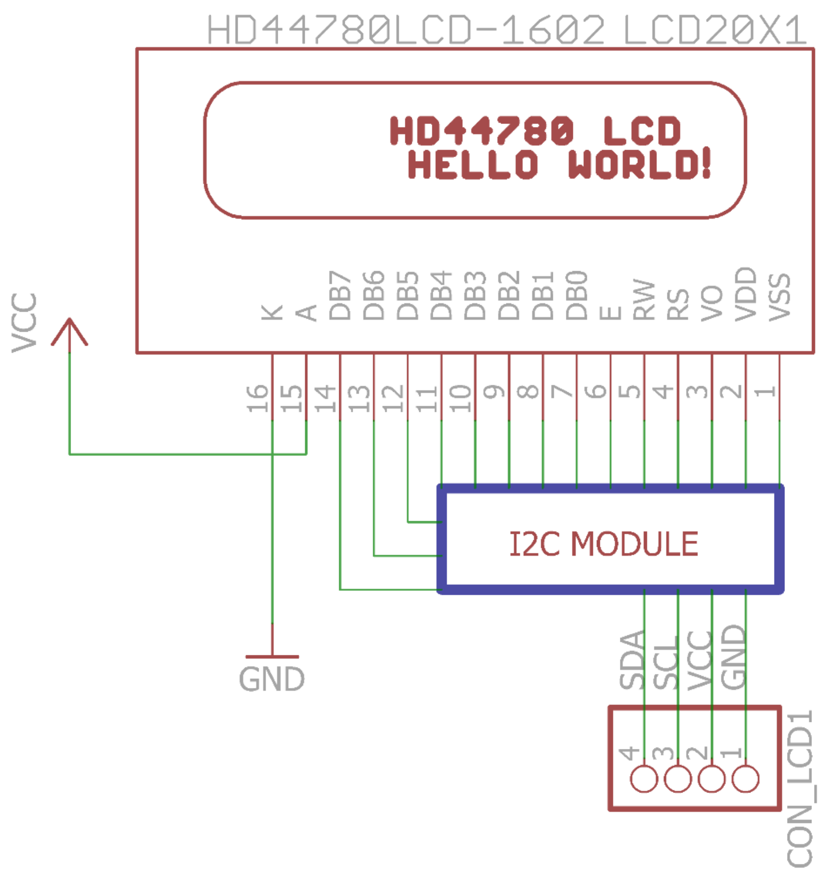
4. Sistem Sensor ZMPT101b



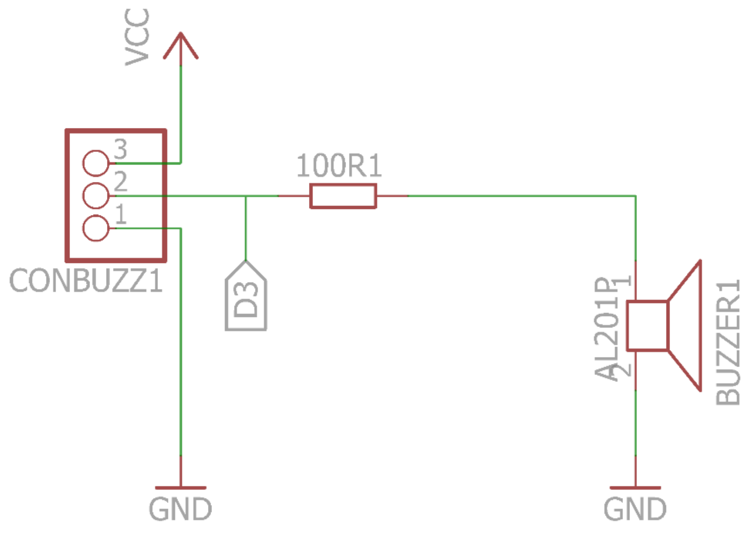
5. Sistem *Push Button*



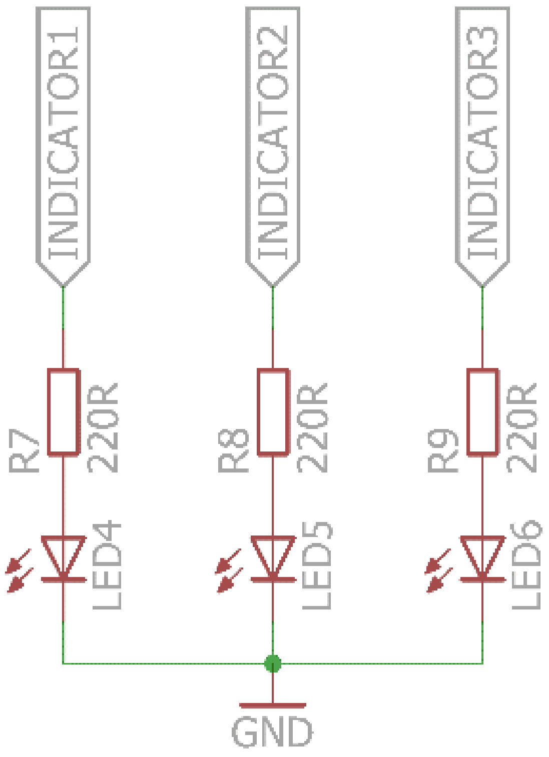
6. Sistem *Alphanumeric* LCD



7. Sistem *Buzzer*



8. Sistem Indikator LED



LAMPIRAN SCRIPT PROGRAM

```
//-------------------File ke-1 LCDML_002_lcd_i2c---------------------

#include "EmonLib.h"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <LCDMenuLib.h>
#include <LcdBarGraphX.h>


#define _LCDML_DISP_cfg_button_press_time          200
#define _LCDML_DISP_cfg_scrollbar                  1
#define _LCDML_DISP_cfg_cursor                     0x7E

#define led1    4
#define led2    5
#define led3    6
#define buzzer  3

byte lcdNumCols = 20;
EnergyMonitor emon1;

#define _LCDML_DISP_cols             20
#define _LCDML_DISP_rows             4

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LcdBarGraphX lbg            (&lcd, 20, 0, 3);

const uint8_t scroll_bar[5][8] = {
  {B10001, B10001, B10001, B10001, B10001, B10001, B10001, B10001},
  {B11111, B11111, B10001, B10001, B10001, B10001, B10001, B10001},
  {B10001, B10001, B11111, B11111, B10001, B10001, B10001, B10001},
  {B10001, B10001, B10001, B10001, B11111, B11111, B10001, B10001},
  {B10001, B10001, B10001, B10001, B10001, B10001, B11111, B11111}
};

#define _LCDML_DISP_cnt    12

 LCDML_DISP_init(_LCDML_DISP_cnt);
  LCDML_DISP_add      (0  , _LCDML_G1  , LCDML_root        , 1  , "About
This"          , LCDML_FUNC_about_this);
  LCDML_DISP_add      (1  , _LCDML_G1  , LCDML_root        , 2  , "Alarm
Test"          , LCDML_FUNC_alarm_test);
  LCDML_DISP_add      (2  , _LCDML_G1  , LCDML_root        , 3  ,
"Calibration"         , LCDML_FUNC_auto_calibrate);
  LCDML_DISP_add      (3  , _LCDML_G1  , LCDML_root        , 4  ,
"Measure"             , LCDML_FUNC);
  LCDML_DISP_add      (4  , _LCDML_G1  , LCDML_root_4      , 1  , "AC
Current"          , LCDML_FUNC_current);
  LCDML_DISP_add      (5  , _LCDML_G1  , LCDML_root_4      , 2  , "AC
Voltage"          , LCDML_FUNC_voltage);
  LCDML_DISP_add      (6  , _LCDML_G1  , LCDML_root_4      , 3  ,
"Apparent Power"      , LCDML_FUNC_apparent_power);
```

```
  LCDML_DISP_add      (7  , _LCDML_G1  , LCDML_root_4      , 4  , "Real
Power"          , LCDML_FUNC_real_power);
  LCDML_DISP_add      (8  , _LCDML_G1  , LCDML_root_4      , 5  , "Power
Factor"        , LCDML_FUNC_power_factor);
  LCDML_DISP_add      (9  , _LCDML_G1  , LCDML_root        , 5  ,
"Select Monitor"      , LCDML_FUNC);
  LCDML_DISP_add      (10 , _LCDML_G1  , LCDML_root_5      , 1  , "Power
450  VA"       , LCDML_FUNC_450);
  LCDML_DISP_add      (11 , _LCDML_G1  , LCDML_root_5      , 2  , "Power
900  VA"       , LCDML_FUNC_900);
  LCDML_DISP_add      (12 , _LCDML_G1  , LCDML_root_5      , 3  , "Power
1300 VA"       , LCDML_FUNC_1300);
  LCDML_DISP_createMenu(_LCDML_DISP_cnt);


  #define _LCDML_BACK_cnt    1

  LCDML_BACK_init(_LCDML_BACK_cnt);
  LCDML_BACK_new_timebased_dynamic (0  , ( 20UL )         , _LCDML_start
, LCDML_BACKEND_control);
  LCDML_BACK_new_timebased_dynamic (1  , ( 1000UL )       , _LCDML_stop
, LCDML_BACKEND_menu);
  LCDML_BACK_create();



  void setup()
  {
    emon1.current(A1, 60.7);
    emon1.voltage(A0, 212, 1.2);

    lcd.begin(4, lcdNumCols);
    lcd.clear();
    delay(100);

    pinMode (led1, OUTPUT);
    pinMode (led2, OUTPUT);
    pinMode (led3, OUTPUT);
    pinMode (buzzer, OUTPUT);

    lcd.home ();

    lcd.createChar(0, (uint8_t*)scroll_bar[0]);
    lcd.createChar(1, (uint8_t*)scroll_bar[1]);
    lcd.createChar(2, (uint8_t*)scroll_bar[2]);
    lcd.createChar(3, (uint8_t*)scroll_bar[3]);
    lcd.createChar(4, (uint8_t*)scroll_bar[4]);
    lcd.setCursor(0,0);

    LCDML_DISP_groupEnable(_LCDML_G1);
    LCDML_setup(_LCDML_BACK_cnt);

  }
  void loop()
  {
    LCDML_run(_LCDML_priority);
  }

# if(_LCDML_DISP_rows > _LCDML_DISP_cfg_max_rows)
# error change value of _LCDML_DISP_cfg_max_rows in LCDMenuLib.h
# endif
# if(_LCDML_DISP_cols > _LCDML_DISP_cfg_max_string_length)
# error change value of _LCDML_DISP_cfg_max_string_length in
LCDMenuLib.h
# endif
```

//-----------------------File ke-2 LCDML_CONTROL-----------------------

```
#define _LCDML_CONTROL_cfg    2
void LCDML_BACK_setup(LCDML_BACKEND_control)
{
  LCDML_CONTROL_setup();
}
boolean LCDML_BACK_loop(LCDML_BACKEND_control)
{
  LCDML_CONTROL_loop();
  return true;
}
void LCDML_BACK_stable(LCDML_BACKEND_control)
{
}


#if(_LCDML_CONTROL_cfg == 2)

  #define _LCDML_CONTROL_digital_low_active      1
  #define _LCDML_CONTROL_digital_enable_quit     1
  #define _LCDML_CONTROL_digital_enter           9
  #define _LCDML_CONTROL_digital_up              10
  #define _LCDML_CONTROL_digital_down            8
  #define _LCDML_CONTROL_digital_quit            12

void LCDML_CONTROL_setup()
{

  pinMode(_LCDML_CONTROL_digital_enter      , INPUT_PULLUP);
  pinMode(_LCDML_CONTROL_digital_up         , INPUT_PULLUP);
  pinMode(_LCDML_CONTROL_digital_down       , INPUT_PULLUP);
  # if(_LCDML_CONTROL_digital_enable_quit == 1)
    pinMode(_LCDML_CONTROL_digital_quit     , INPUT_PULLUP);
  # endif
}

void LCDML_CONTROL_loop()
{
  #if(_LCDML_CONTROL_digital_low_active == 1)
  #  define _LCDML_CONTROL_digital_a !
  #else
  #  define _LCDML_CONTROL_digital_a
  #endif

  uint8_t but_stat = 0x00;

  bitWrite(but_stat, 0,
_LCDML_CONTROL_digital_a(digitalRead(_LCDML_CONTROL_digital_enter)));
  bitWrite(but_stat, 1,
_LCDML_CONTROL_digital_a(digitalRead(_LCDML_CONTROL_digital_up)));
  bitWrite(but_stat, 2,
_LCDML_CONTROL_digital_a(digitalRead(_LCDML_CONTROL_digital_down)));
  #if(_LCDML_CONTROL_digital_enable_quit == 1)
  bitWrite(but_stat, 3,
_LCDML_CONTROL_digital_a(digitalRead(_LCDML_CONTROL_digital_quit)));
  #endif


  if (but_stat > 0) {
    if((millis() - g_LCDML_DISP_press_time) >=
_LCDML_DISP_cfg_button_press_time) {
      g_LCDML_DISP_press_time = millis();

      if (bitRead(but_stat, 0)) { LCDML_BUTTON_enter(); }
      if (bitRead(but_stat, 1)) { LCDML_BUTTON_up();    }
      if (bitRead(but_stat, 2)) { LCDML_BUTTON_down();  }
      if (bitRead(but_stat, 3)) { LCDML_BUTTON_quit();  }

    }
  }
}


#else
  #error _LCDML_CONTROL_cfg is not defined or not in range
#endif
```

//-----------------------File ke-3 LCDML_DISP--------------------------

```
void LCDML_lcd_menu_display()

{
  lcd.home ();


  if (LCDML_DISP_update()) {
    uint8_t n_max             = (LCDML.getChilds() >= _LCDML_DISP_rows)
? _LCDML_DISP_rows : (LCDML.getChilds());
    uint8_t scrollbar_min     = 0;
    uint8_t scrollbar_max     = LCDML.getChilds();
    uint8_t scrollbar_cur_pos = LCDML.getCursorPosAbs();
    uint8_t scroll_pos        = ((1.*n_max * _LCDML_DISP_rows) /
(scrollbar_max - 1) * scrollbar_cur_pos);



    if (LCDML_DISP_update_content()) {

      LCDML_lcd_menu_clear();

      for (uint8_t n = 0; n < n_max; n++)
      {
        lcd.setCursor(1, n);
        lcd.print(LCDML.content[n]);
      }
    }

    if (LCDML_DISP_update_cursor()) {
      for (uint8_t n = 0; n < n_max; n++)
      {
        lcd.setCursor(0, n);

        if (n == LCDML.getCursorPos()) {
          lcd.write(_LCDML_DISP_cfg_cursor);
        } else {
          lcd.write(' ');
        }

        if (_LCDML_DISP_cfg_scrollbar == 1) {
          if (scrollbar_max > n_max) {
            lcd.setCursor((_LCDML_DISP_cols - 1), n);
            lcd.write((uint8_t)0);
          }
          else {
            lcd.setCursor((_LCDML_DISP_cols - 1), n);
            lcd.print(' ');
          }
        }
      }

      if (_LCDML_DISP_cfg_scrollbar == 1) {
        if (scrollbar_max > n_max) {
          if (scrollbar_cur_pos == scrollbar_min) {
            lcd.setCursor((_LCDML_DISP_cols - 1), 0);
            lcd.write((uint8_t)1);
          } else if (scrollbar_cur_pos == (scrollbar_max - 1)) {
            lcd.setCursor((_LCDML_DISP_cols - 1), (n_max - 1));
            lcd.write((uint8_t)4);
          } else {
            lcd.setCursor((_LCDML_DISP_cols - 1), scroll_pos / n_max);
            lcd.write((uint8_t)(scroll_pos % n_max) + 1);
          }
        }
      }
    }
  }
  LCDML_DISP_update_end();
}

void LCDML_lcd_menu_clear()
{
  lcd.clear();
  lcd.setCursor(0, 0);
}




//-----------------------File ke-4 LCDML_FUNC_DISP----------------------
void LCDML_DISP_setup(LCDML_FUNC_about_this)

{
  lcd.setCursor(4,0);
  lcd.print("Home  Energy");
  lcd.setCursor(5,1);
  lcd.print("Monitoring");
  lcd.setCursor(2,3);
  lcd.print("450 | 900 | 1300");

}

void LCDML_DISP_loop(LCDML_FUNC_about_this)
  {
  lcd.setCursor(0,2);
  lcd.print("....................");
  delay(300);

  lcd.setCursor(0,2);
  lcd.print("                    ");
  delay(300);

  }


void LCDML_DISP_loop_end(LCDML_FUNC_about_this)
  {

  }


void LCDML_DISP_setup(LCDML_FUNC_alarm_test)
{
 pinMode (led1, OUTPUT);
 pinMode (led2, OUTPUT);
 pinMode (led3, OUTPUT);
 pinMode (buzzer, OUTPUT);
}

void LCDML_DISP_loop(LCDML_FUNC_alarm_test)
{
  alarm();
}

void LCDML_DISP_loop_end(LCDML_FUNC_alarm_test)
{

}



//----------------------------------------------------------------------
------------------------------------------//MEASURE CURRENT

void LCDML_DISP_setup(LCDML_FUNC_current)
{
 lcd.print(F("**MEASURE  CURRENT**"));
 lcd.setCursor(11,2);
 lcd.print(F("Amp"));
}

void LCDML_DISP_loop(LCDML_FUNC_current)
{
  measure_current();
}

void LCDML_DISP_loop_end(LCDML_FUNC_current)
{

}

//----------------------------------------------------------------------
------------------------------------------//MEASURE VOLTAGE

void LCDML_DISP_setup(LCDML_FUNC_voltage)
{
 lcd.print(F("**MEASURE  VOLTAGE**"));
 lcd.setCursor(12,2);
 lcd.print(F("Volt"));
}

void LCDML_DISP_loop(LCDML_FUNC_voltage)
{
 measure_voltage();
}

void LCDML_DISP_loop_end(LCDML_FUNC_voltage)
{
}

//------------------------------------------------------------------------
-----------------------------------------//MEASURE APPARENT POWER
void LCDML_DISP_setup(LCDML_FUNC_apparent_power)
{
 lcd.print(F("***APPARENT POWER***"));
 lcd.setCursor(12,2);
 lcd.print(F("VA"));
}

void LCDML_DISP_loop(LCDML_FUNC_apparent_power)
{
  measure_apparent_power();
}


void LCDML_DISP_loop_end(LCDML_FUNC_apparent_power)
{

}


//------------------------------------------------------------------------
-----------------------------------------//MEASURE REAL POWER
void LCDML_DISP_setup(LCDML_FUNC_real_power)
{
 lcd.print(F("*****REAL POWER*****"));
 lcd.setCursor(11,2);
 lcd.print(F("Watt"));
}

void LCDML_DISP_loop(LCDML_FUNC_real_power)
{
  measure_real_power();
}


void LCDML_DISP_loop_end(LCDML_FUNC_real_power)
{

}


//------------------------------------------------------------------------
-----------------------------------------//MEASURE POWER FACTOR
void LCDML_DISP_setup(LCDML_FUNC_power_factor)
{
 lcd.print(F("****POWER FACTOR****"));
 lcd.setCursor(6,2);
 lcd.print(F("Pf="));

}
```

```
void LCDML_DISP_loop(LCDML_FUNC_power_factor)
{
  measure_power_factor();
}


void LCDML_DISP_loop_end(LCDML_FUNC_power_factor)
{

}

void LCDML_DISP_setup(LCDML_FUNC_450)
{
    lcd.setCursor (0,0);
    lcd.print("V:");
    lcd.setCursor (8,0);
    lcd.print("v");
    lcd.setCursor (13,0);
    lcd.print("I:");
    lcd.setCursor (19,0);
    lcd.print("A");
    lcd.setCursor(6,2);
    lcd.print("S:");
    lcd.setCursor(14,2);
    lcd.print("VA");
}

void LCDML_DISP_loop(LCDML_FUNC_450)
{ range_450();
}

void LCDML_DISP_loop_end(LCDML_FUNC_450)
{

}


void LCDML_DISP_setup(LCDML_FUNC_900)
{
    lcd.setCursor (0,0);
    lcd.print("V:");
    lcd.setCursor (8,0);
    lcd.print("v");
    lcd.setCursor (13,0);
    lcd.print("I:");
    lcd.setCursor (19,0);
    lcd.print("A");
    lcd.setCursor(6,2);
    lcd.print("S:");
    lcd.setCursor(14,2);
    lcd.print("VA");
}

void LCDML_DISP_loop(LCDML_FUNC_900)
```

```
{
  range_900();
}

void LCDML_DISP_loop_end(LCDML_FUNC_900)
{

}



void LCDML_DISP_setup(LCDML_FUNC_1300)
{
    lcd.setCursor (0,0);
    lcd.print("V:");
    lcd.setCursor (8,0);
    lcd.print("v");
    lcd.setCursor (13,0);
    lcd.print("I:");
    lcd.setCursor (19,0);
    lcd.print("A");
    lcd.setCursor(6,2);
    lcd.print("S:");
    lcd.setCursor(14,2);
    lcd.print("VA");
}

void LCDML_DISP_loop(LCDML_FUNC_1300)
{ emon1.calcVI(20,150);
  range_1300();
}

void LCDML_DISP_loop_end(LCDML_FUNC_1300)
{

}


void LCDML_DISP_setup(LCDML_FUNC_auto_calibrate)
{
  lcd.setCursor(0,1);
  lcd.print("   Calibrating");
  delay(1000);
  lcd.setCursor(0,1);
  lcd.print("   Calibrating.");
  delay(3000);
  lcd.setCursor(0,1);
  lcd.print("   Calibrating..");
  delay(2000);
  lcd.setCursor(0,1);
  lcd.print("   Calibrating...");
  delay(2000);

  lcd.clear();
}

void LCDML_DISP_loop(LCDML_FUNC_auto_calibrate)
{
  emon1.calcVI(20,2000);
  double Irms = emon1.calcVrms(1480);
  double Vrms = emon1.calcVrms(1480);

  lcd.setCursor(6,1);
  lcd.print("Complete");
}

void LCDML_DISP_loop_end(LCDML_FUNC_auto_calibrate)
  {

  }
```

```
//---------------------File ke-4 LCDML_FUNC_MEASURE---------------------
void alarm()

{
  lcd.setCursor(6,1);
  lcd.print("Alarm ON");
  digitalWrite (led1    , HIGH);
  digitalWrite (led2    , HIGH);
  digitalWrite (led3    , HIGH);
  digitalWrite (buzzer  , HIGH);
  delay(1000);

  lcd.setCursor(6,1);
  lcd.print("        ");
  digitalWrite (led1    , LOW);
  digitalWrite (led2    , LOW);
  digitalWrite (led3    , LOW);
  digitalWrite (buzzer  , LOW);
  delay(300);
}

void alarm_level_1()
{
  digitalWrite (led3    , HIGH);
  digitalWrite (led2    , HIGH);
  digitalWrite (buzzer  , HIGH);
  delay(1000);

  digitalWrite (led3    , LOW);
  digitalWrite (led2    , LOW);
  digitalWrite (buzzer  , LOW);
  delay(300);
}

void alarm_level_2()
```

```
{
  digitalWrite (led3    , HIGH);
  digitalWrite (led2    , HIGH);
  digitalWrite (buzzer  , HIGH);
  delay(1000);

  digitalWrite (led3    , LOW);
  digitalWrite (led2    , LOW);
  digitalWrite (buzzer  , LOW);
  delay(300);
}

void alarm_level_3()
{
  digitalWrite (led1    , HIGH);
  digitalWrite (led2    , HIGH);
  digitalWrite (led3    , HIGH);
  digitalWrite (buzzer  , HIGH);
  delay(1000);

  digitalWrite (led1    , LOW);
  digitalWrite (led2    , LOW);
  digitalWrite (led3    , LOW);
  digitalWrite (buzzer  , LOW);
  delay(300);
}

void alarm_off()
{
    digitalWrite  (led3 ,LOW);
    digitalWrite  (led2 ,LOW);
    digitalWrite  (led1 ,LOW);
    digitalWrite  (buzzer  ,LOW);
}

void measure_current()
{
  double Irms = emon1.calcIrms(1480);
  lcd.setCursor(6,2);lcd.print(Irms);
  lbg.drawValue(Irms, 7);
  delay(7);
}

void measure_voltage()
{
  double Vrms = emon1.calcVrms(1480);
  lcd.setCursor(5,2);lcd.print(Vrms);
  lbg.drawValue(Vrms, 280);
  delay(7);
}

void measure_apparent_power()
{
  emon1.calcVI(20,2000);
  double apparentPower  = emon1.apparentPower;
  lcd.setCursor(7,2);lcd.print(apparentPower);
  lbg.drawValue(apparentPower, 900);
  delay(7);
}

void measure_real_power()
{
  emon1.calcVI(20,2000);
  double realPower  = emon1.realPower;
  lcd.setCursor(6,2);lcd.print(realPower);
  lbg.drawValue(realPower, 900);
  delay(7);
}

void measure_power_factor()
{
  emon1.calcVI(20,2000);
  double powerFActor  = emon1.powerFactor;
  lcd.setCursor(9,2);lcd.print(powerFActor);
}
void range_450()
{
  emon1.calcVI(20,2000);
  double apparentPower  = emon1.apparentPower;

  lcd.setCursor(0,0);lcd.print("***Monitor  450VA***");
  lcd.setCursor(8,2);lcd.print(apparentPower);
  lbg.drawValue(apparentPower, 450);delay(7);

  if (apparentPower >= 150 && apparentPower <= 200)
  {
   alarm_level_3();
  }

   if (apparentPower >= 200 && apparentPower <= 250)
  {
    alarm_level_2();
  }

   if (apparentPower >= 250 && apparentPower <= 450)
  {
    alarm_level_3();
  }

  else
  {
    alarm_off();
  }
}
void range_900()
{
  emon1.calcVI(20,2000);
  double apparentPower  = emon1.apparentPower;
  lcd.setCursor(0,0);lcd.print("***Monitor  900VA***");
  lcd.setCursor(8,2);lcd.print(apparentPower);
  lbg.drawValue(apparentPower, 900);delay(7);

  if (apparentPower >= 650 && apparentPower <= 750)
  {
    alarm_level_1();
  }

   if (apparentPower >= 750 && apparentPower <= 800)
  {
    alarm_level_2();
  }

   if (apparentPower >= 800 && apparentPower <= 900)
  {
    alarm_level_3();
  }

  else
  {
    alarm_off();
  }
}
void range_1300()
{
  emon1.calcVI(20,2000);
  double apparentPower  = emon1.apparentPower;
  lcd.setCursor(0,0);lcd.print("***Monitor 1300VA***");
  lcd.setCursor(8,2);lcd.print(apparentPower);
  lbg.drawValue(apparentPower, 1300);delay(7);

  if (apparentPower >= 850 && apparentPower <= 950)
  {
    alarm_level_1();
  }

   if (apparentPower >= 950 && apparentPower <= 1100)
  {
    alarm_level_2();
  }

   if (apparentPower >= 1100 && apparentPower <= 1300)
  {
    alarm_level_3();
  }

  else
  {
    alarm_off();
  }
}
```

LAMPIRAN DATA SHEET

1. Sensor Arus SCT 013-000

SPECIFICATION

Customer Title : XiDi Technology

Product Name : Split-core current transformer

Manufacture Model : SCT-013-000

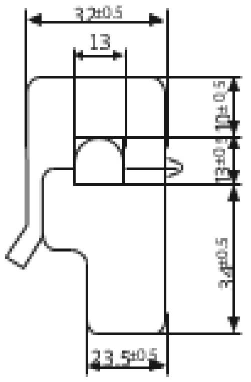
Charateristics: open size:13mm×13mm 1m leading wire

Core material:Ferrite

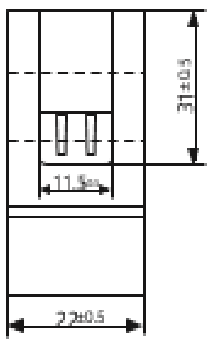
Fire resistance property:in accordance with UL 94-V0

Dielectric strength: 1000V AC/1min 5mA (between shell and output)

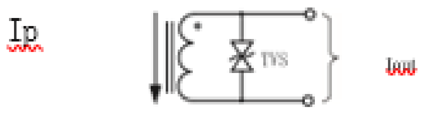
Outline size diagram:(in mm)



Front View



Side View



TVS: Transient-voltage Suppressor

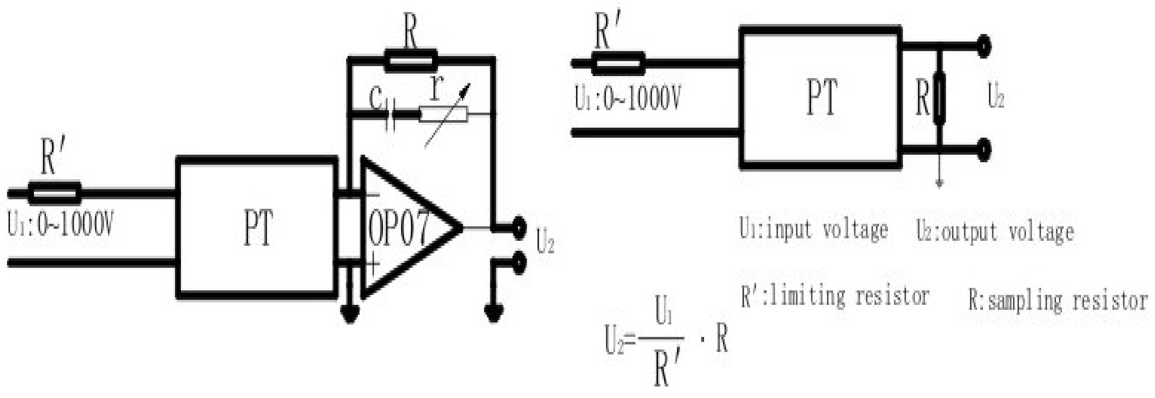
Current output type

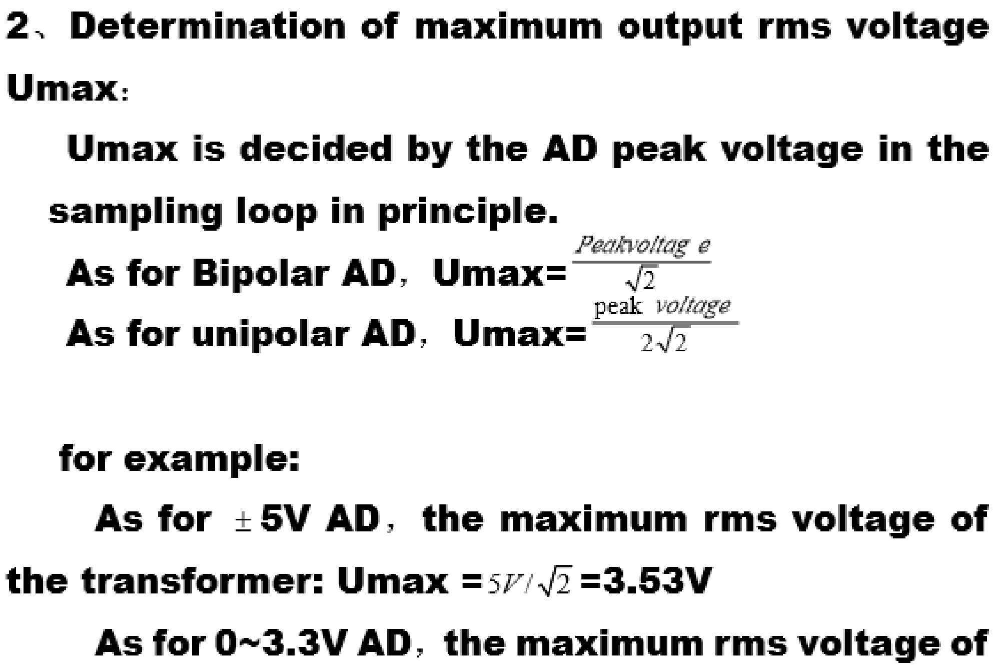
Schematic Diagram

Typical table of technical parameters:

input current	output voltage	non-linearity	build-in sampling resistance (RL)
0-100A	0-50mV	±3%	Ω
turn ratio	resistance grade	work temperature	dielectric strength(between shell and output)
100A:0.05A	Grade B	-25℃~+70℃	1000V AC/1min 5mA

2. **Sensor Tegangan ZMPT101b**





2、Determination of maximum output rms voltage Umax：

Umax is decided by the AD peak voltage in the sampling loop in principle.

As for Bipolar AD，Umax=$\frac{Peakvoltag\ e}{\sqrt{2}}$

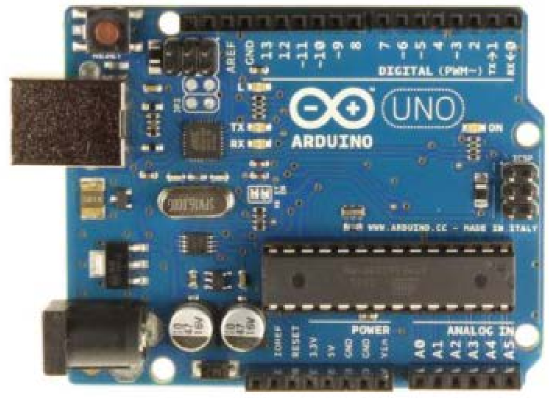
As for unipolar AD，Umax=$\frac{\text{peak } voltage}{2\sqrt{2}}$

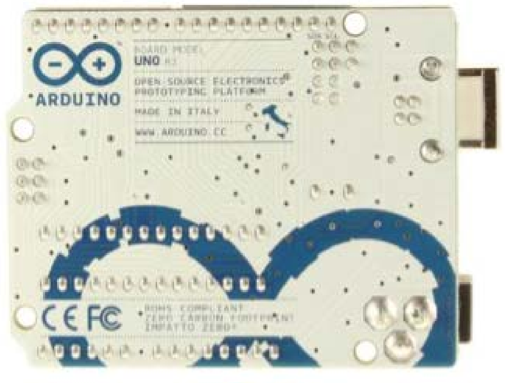
for example:

As for ±5V AD，the maximum rms voltage of the transformer: Umax =$5V/\sqrt{2}$**=3.53V**

As for 0~3.3V AD，the maximum rms voltage of

3. Arduino Uno R3

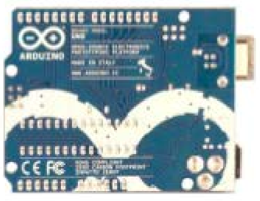

Arduino Uno R3 Front


Arduino Uno R3 Back

Arduino Uno R2 Front

Arduino Uno SMD

Arduino Uno Front


Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter. Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode. Revision 3 of the board has the following new features:

1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.

2.0 Stronger RESET circuit.

Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

Microcontroller ATmega328 Operating Voltage 5V Input Voltage (recommended) 7-12V

Input Voltage (limits) 6-20V Digital I/O Pins 14 (of which 6 provide PWM output) Analog Input Pins 6 DC Current per I/O Pin 40 mA DC Current for 3.3V Pin 50 mA Flash Memory 32 KB (ATmega328) of which 0.5 KB used by bootloader SRAM 2 KB (ATmega328) EEPROM 1 KB (ATmega328) Clock Speed 16 MHz

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino-uno-Rev3-schematic.pdf Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- 5V.This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA. □GND. Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials. The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files). You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details. The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. □On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following halfsecond or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data. The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

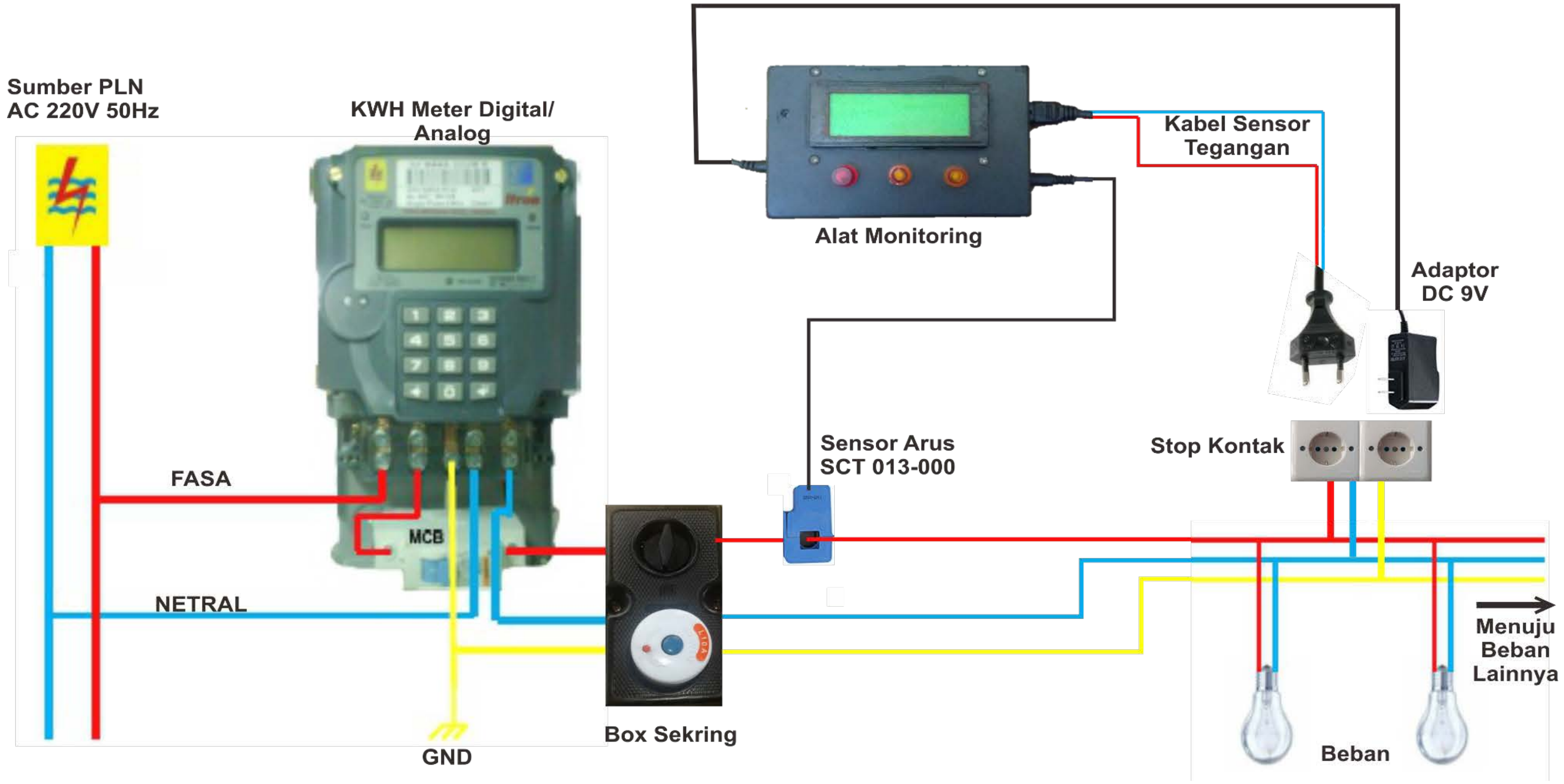
USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

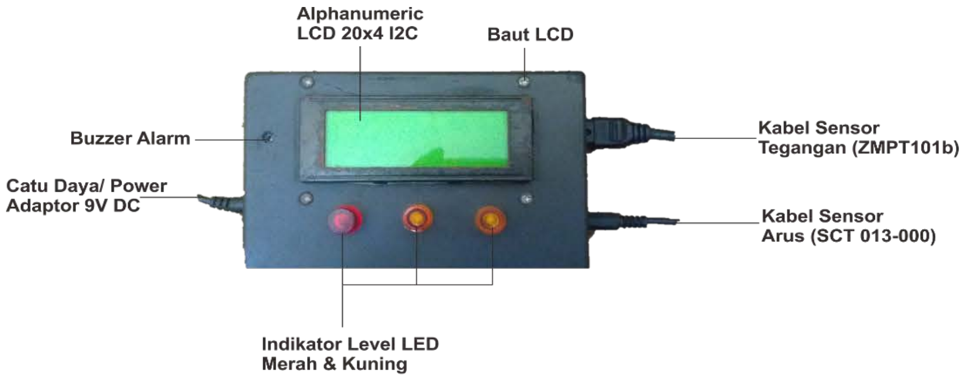
Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

LAMPIRAN PEMASANGAN ALAT MONITORING

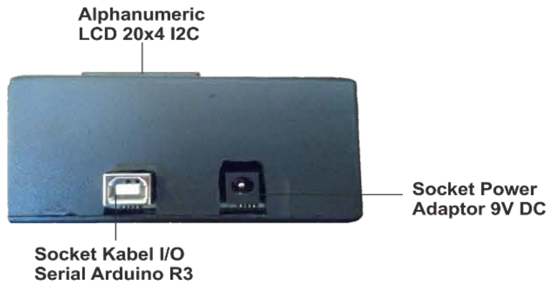


LAMPIRAN DETAIL ALAT MONITORING



Alphanumeric LCD 20x4 I2C
Baut LCD
Buzzer Alarm
Catu Daya/ Power Adaptor 9V DC
Kabel Sensor Tegangan (ZMPT101b)
Kabel Sensor Arus (SCT 013-000)
Indikator Level LED Merah & Kuning

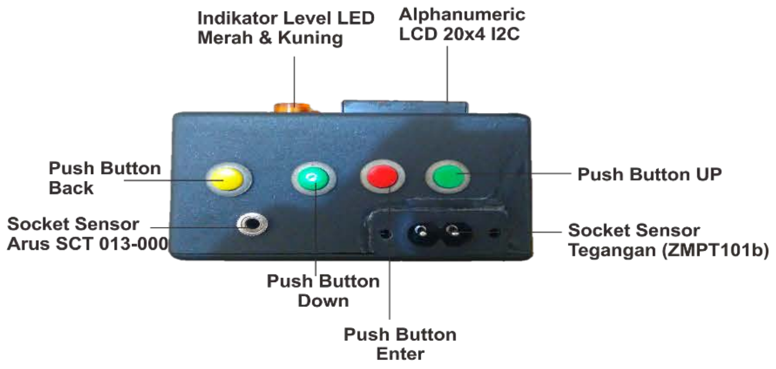
TAMPAK DEPAN



Alphanumeric LCD 20x4 I2C
Socket Power Adaptor 9V DC
Socket Kabel I/O Serial Arduino R3

TAMPAK KIRI

TAMPAK BELAKANG



Indikator Level LED Merah & Kuning
Alphanumeric LCD 20x4 I2C
Push Button Back
Push Button UP
Socket Sensor Arus SCT 013-000
Socket Sensor Tegangan (ZMPT101b)
Push Button Down
Push Button Enter

TAMPAK KANAN