# BAB IV
# IMPLEMENTASI DATA DAN ANALISIS

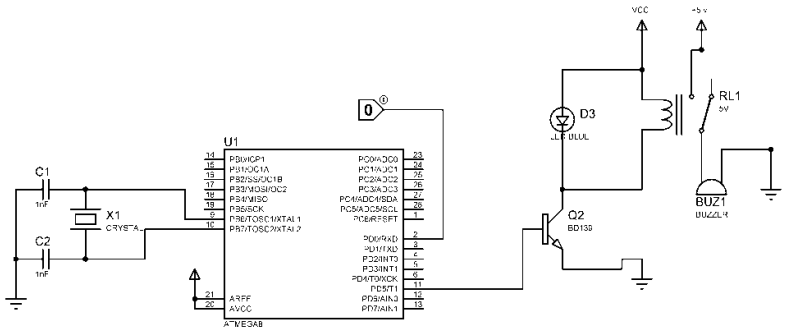
## 4.1 Pengujian

Dalam bab ini akan dibahas mengenai pengujian dan analisa dari simulasi sistem perancangan program. Tujuan simulasi adalah untuk mengetahui kebenaran suatu program rangkaian dan komponen yang akan diuji. dalam simulasi proteus pengujian program ini sangat penting karena bila ada salah satu rangkaian yang tidak bekerja sesuai dengan fungsinya dapat di ketahui dari awal sehingga memudahkan dalam menganalisis.

Dengan adanya pengujian-pengujian diharapkan kemungkinan terjadi kesalahan atau kelemahan suatu program yang masih ada masalah dalam program yang masih terdapat tiap-tiap bagian rangkaian dapat diketahui lebih detail.

### 4.1.1 Rangkaian Keseluruhan Program

Rangkaian secara keseluruhan dari bagian elektronik dapat dilihat pada gambar 4.11.

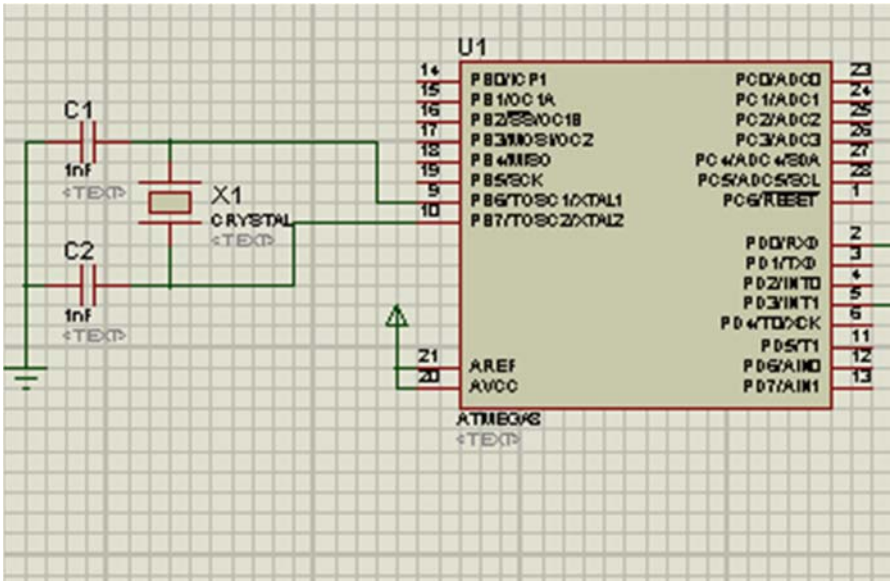



**Gambar 4.1 Rangkaian keseluruhan**

Rangkaian pada sistem dibuat mengunakan mikrokontroler dimaksutkan untuk menjadikan otak pengerak dan pengatur suatu syistem alat

tesebut karena dalam suatusistem tanpa ada mikrokontroler sebagai otak pengera sistem tersebut tidak bisa berjalan dengan lancar atau mungkin tida jalan dengan semestinya.

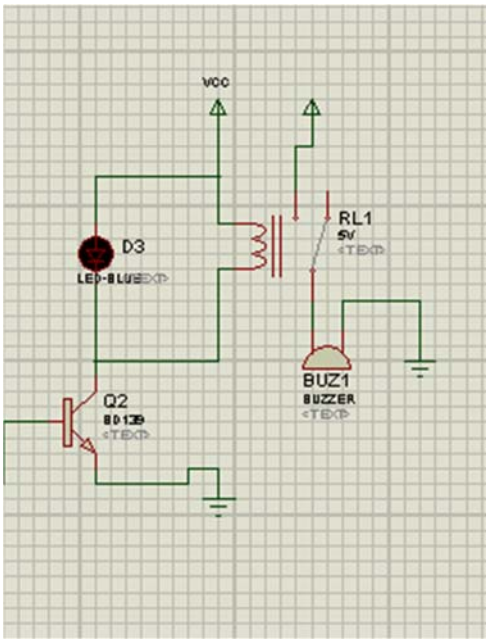
### 4.1.2 Rangkaian Dasar Mikrokontroler




**Gambar 4.2 Rangkaian dasar mikrokontroler**

Bagian minmum sistem mikrokontroler ATMega8 memerlukan daya sebesar 5Vdc. Sumber clock diperoleh dari sebuah kristal (XTAL) 12MHz di pasang pada kaki 9dan 10 Kapasitor C1 dan C2 yang dipasang pada kristal berfungsi sebaga penghilang tegangan rippil yang dihasilkan dari pembentukan osilator oleh kristal.

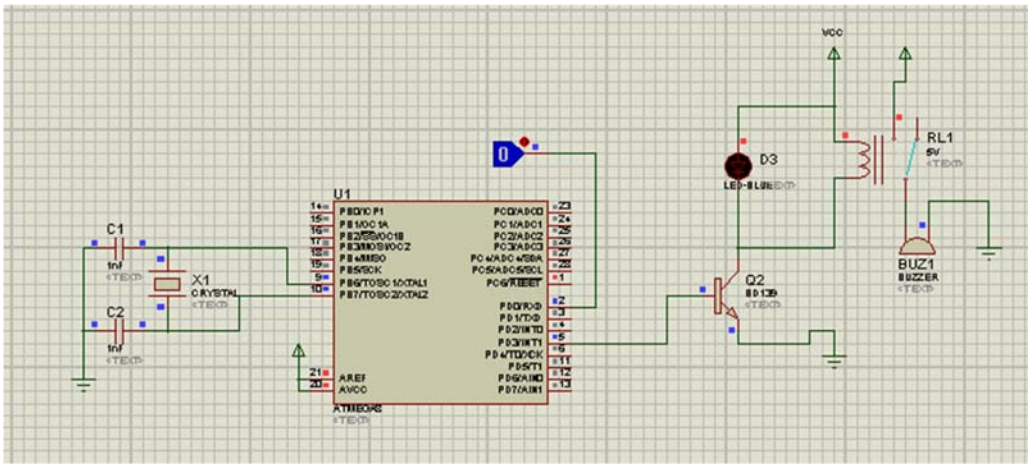
### 4.1.3 Rangkaian Dasar Sensor Infus




**Gambar 4.3 Rangkaian dasar sensor**

Rangkaian sensor infus rangkaian utama yang akan digunakan sebagai bahan ujicoba mendeteksi infus ketika cairan infus akan habis. dan memberi respon terhadap sensor yang mendeteksi cairan dan membaca perinta dari mikrokontroler yang akan menerima dan menyalurkan ke *buzzer* agar memberi tanda dengan cara membunyikan suara peringatan ketika cairan dalam infus pasien mulai habis.

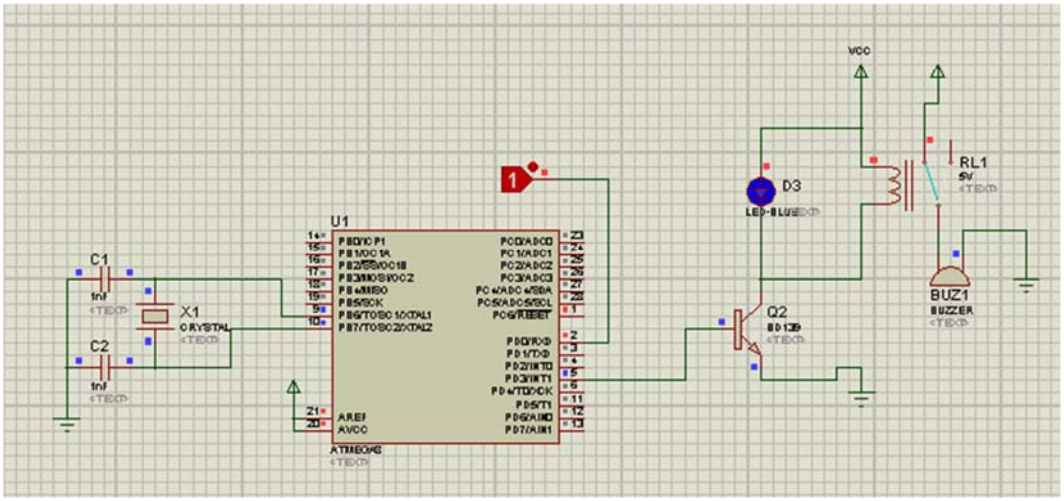
### 4.1.4 Rangkaian Sensor Infus Tidak Terhubung




**Gambar 4.4 Rangkaian sensor infus tidak terhubung**

Di atas menunjukan bahwa Rangkaian Sensor Infus terdiri dari pothodioda kapasitor transistor LED *relay buzzer* dan mikrokontroler apa bila rangkaian diatas tidak bekerja dan menunjukan bahwa sensor itu menunjukan angka nol karna rangkaian tidak mendapat *suplay* daya komponen dan sensor infus tidak dapat membaca mikro hanya bisa bekerja ketika sensor itu mendapat suplay daya dari listrik yang akan masuk lewat input dan apa bila daya tidak di hidupkan atau listrik maka tidak ada perintah dari mikrokontroler sebagai otak pengerak atau perintah menjalankan sensor dari photodioda diatas yang sementara berbentuk angka nol apa bila tidak diberi tegangan dan satu apa bila mendapat *suplay* tegangan dari listrik dan di bawah meunjukan dimana terhubung ke tegangan listrik dan rangkaian mendapat *suplay* daya dari listrik dan sensor membaca perintah dari mikro dan mikro melanjutkan ke bazer agar bazer dapat membunyikan suara peringatan di bawah menunjukan gambar rangkaian sensor terhubung.

### 4.1.5 Rangkaian Sensor Infus Terhubung



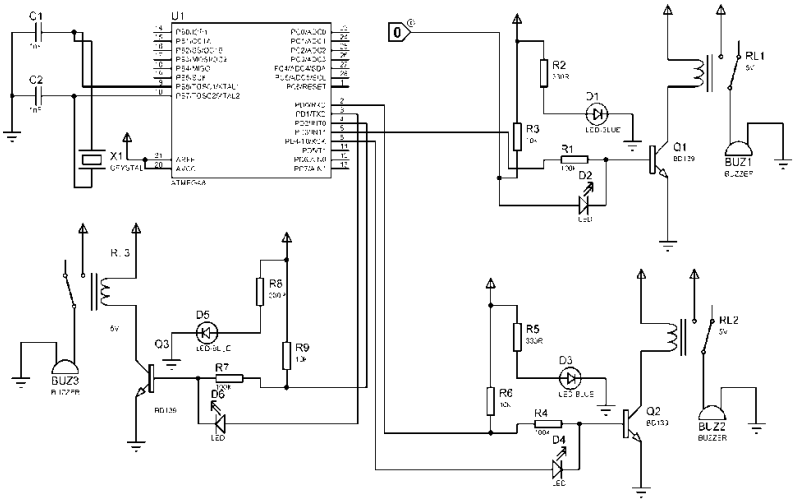

**Gambar 4.5 Rangkaian sensor infus terhubung**

Rangkaian Sensor Infus diatas menunjukan bahwa sensor berjalan karna sensor menunjukan angka satu berarti terhubung karna sensor mendapat perintah dari mikrokontroler untuk membaca apa yang di perintahkan oleh mikrokontroler yang sudah di program untuk dijalankan sedangkan gambar 4.1.4 diatas menunjukan bawah gambar tidak berjalan karna sensor tidak mendapat perintah dari mikro yang tidak mendapat *suplay* daya sedangkan gambar 4.1.5 gambar diatas terlihat bahwah relay terhubung dan bazzer mendapat perintah bahawa sensor membaca infus pasien mulai habis dan membunyikan suara peringatan agar diganti dengan yang baru karna terlambat saja darah akan naik ke permukaan melalui selang dan ketika itu akan mengakibatkan fatal.

Diatas gambar dimana menunjukan rangkaian untuk satu pasien inap sedangkan di bawah

Sekema rangkaian sensor infus untuk banyak ruangan yang bisa digunakan untuk berapa ruangan inap pasien yang lain hanya menggunakan satu mikro bisa digunakan banyak rangkaian sensor infus karna mikro kontroler memiiki seribu program untuk menyimpan data.

### 4.1.6 Rangkaian Sensor Infus Tiga Ruangan




**Gambar 4.6 Rangkaian sensor infus tiga ruangan**

Diatas menunjukan sekema rangkaian untuk banyak ruangan dimana setiap *port* memiliki *output* untuk setiap sensor infus karna mikrokontroler memiliki banyak kaki setiap kaki memiliki input dan output sendiri-sendiri dan tidak perlu membuat kembali sekema maupun aplikasi untuk sensor infus karna di mikrokontroler menyimpan seribu program diatas adalah contoh gambar sekema sensor infus untuk ruang inap pasien untuk banyak ruangan memper mudah perawat dalam mengentahui diamana infus yang akan habis dan untuk memrogram mikro kontroler bisa menyimpan data yang diprogramkan di mikro dan ambar diatas hanya sebagai contoh memperbanyak sensor infus bila di tambah sensor untuk ruangan yang lain tidak perlu menambah mikro melainkan hanya memrogramnya saja dan menambah rangkaian sensor infus karna mikrokontroler memiliki seribu data yang bisa di tampung di dalam mikro nya.

### 4.1.7 Program Sensor Infus

Berikut adalah program sensor infus :

```
#include <mega8.h>
#include <delay.h>
```

```
// Declare your global variables here
// Voltage Reference: AREF pin
#define ADC_VREF_TYPE ((0<<REFS1) | (0<<REFS0) | (0<<ADLAR))
// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | ADC_VREF_TYPE;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=(1<<ADSC);
// Wait for the AD conversion to complete
while ((ADCSRA & (1<<ADIF))==0);
ADCSRA|=(1<<ADIF);
return ADCW;
}
// TWI functions
#include <twi.h>
void main(void)
{
// Declare your local variables here
// Input/Output Ports initialization
// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) |
(0<<DDB3) | (0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4)
| (0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);
// Port C initialization
```

```
// Function: Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRC=(0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
(0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3)
| (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);
// Port D initialization
// Function: Bit7=In Bit6=In Bit5=Out Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (1<<DDD5) | (0<<DDD4) |
(0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=0 Bit4=T Bit3=P Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) |
(0<<PORTD4) | (1<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) |
(0<<PORTD0);
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=(0<<CS02) | (0<<CS01) | (0<<CS00);
TCNT0=0x00;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Disconnected
// OC1B output: Disconnected
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
```

```
// Compare B Match Interrupt: Off
TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) |
(0<<COM1B0) | (0<<WGM11) | (0<<WGM10);
TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) |
(0<<CS12) | (0<<CS11) | (0<<CS10);
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0<<AS2;
TCCR2=(0<<PWM2) | (0<<COM21) | (0<<COM20) | (0<<CTC2) |
(0<<CS22) | (0<<CS21) | (0<<CS20);
TCNT2=0x00;
OCR2=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) |
(0<<OCIE1B) | (0<<TOIE1) | (0<<TOIE0);
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
```

```
MCUCR=(0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);
// USART initialization
// USART disabled
UCSRB=(0<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (0<<RXEN) |
(0<<TXEN) | (0<<UCSZ2) | (0<<RXB8) | (0<<TXB8);
// Analog Comparator initialization
// Analog Comparator: Off
// The Analog Comparator's positive input is
// connected to the AIN0 pin
// The Analog Comparator's negative input is
// connected to the AIN1 pin
ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) |
(0<<ACIC) | (0<<ACIS1) | (0<<ACIS0);
// ADC initialization
// ADC Clock frequency: 750,000 kHz
// ADC Voltage Reference: AREF pin
ADMUX=ADC_VREF_TYPE;
ADCSRA=(1<<ADEN) | (0<<ADSC) | (0<<ADFR) | (0<<ADIF) |
(0<<ADIE) | (1<<ADPS2) | (0<<ADPS1) | (0<<ADPS0);
SFIOR=(0<<ACME);

// SPI initialization
// SPI disabled
SPCR=(0<<SPIE) | (0<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) |
(0<<CPHA) | (0<<SPR1) | (0<<SPR0);
// TWI initialization
// Mode: TWI Master
// Bit Rate: 100 kHz
twi_master_init(100);
// Global enable interrupts
#asm("sei")
```

```
while (1)
      {
      // Place your code here
      if(PIND.0==1)PORTD.3=1;
      delay_ms(500);
      }
```