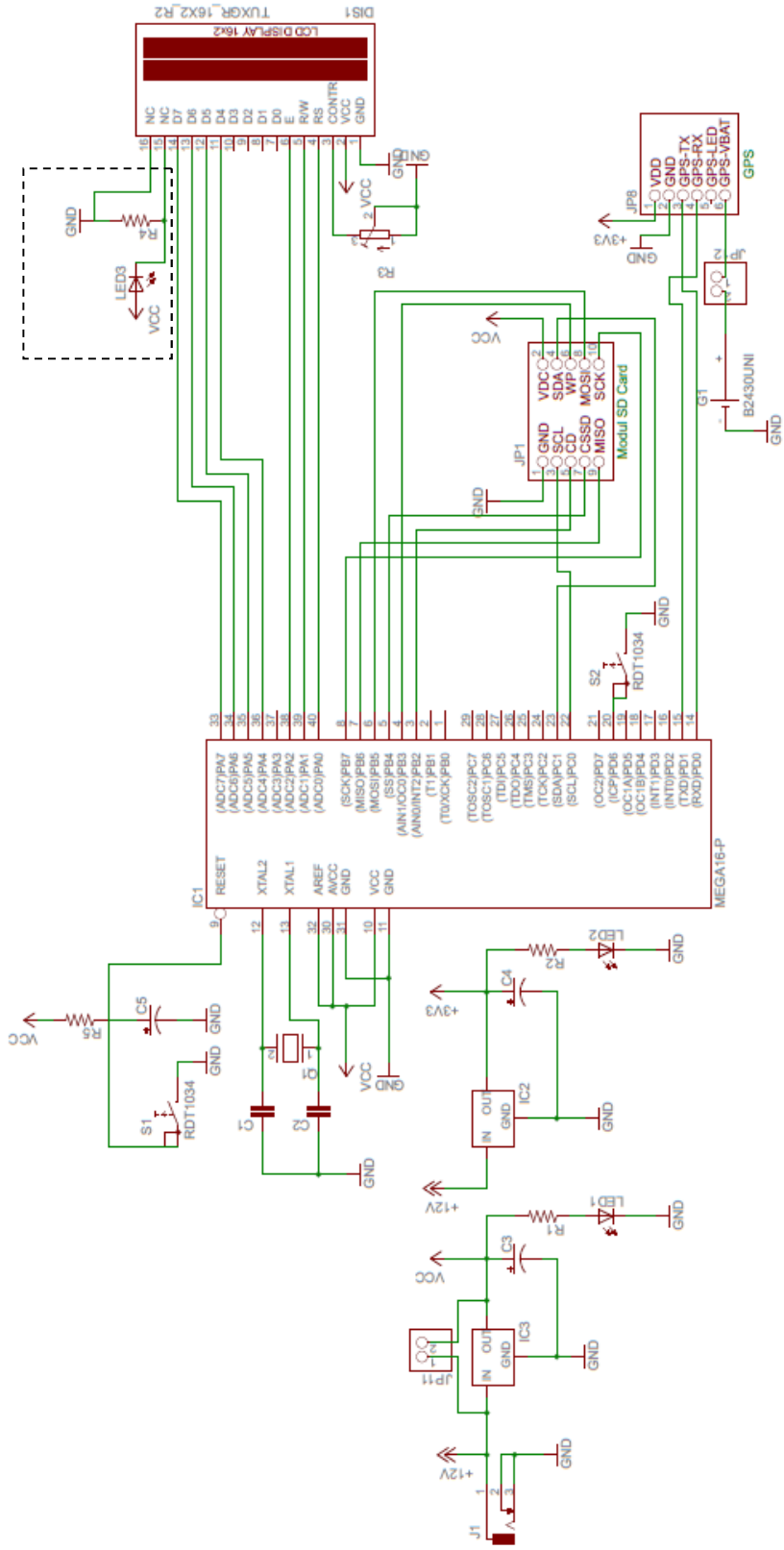
LAMPIRAN A

FOTO ALAT


GPS
ANT
Adapter 1.0GB

LAMPIRAN B

SKEMATIK PERANCANGAN GPS DAN MODUL SD CARD BERBASIS PENGONTROL ATMEGA16


DIS1
TUXGR_16X2_R2
LCD DISPLAY 16x2
NC
NC
D7
D6
D5
D4
D3
D2
D1
D0
E
R/W
RS
CONTR
VCC
GND
R4
LED3
VCC
GND
R3
VCC
GND
JP8
VDD
GND
GPS-TX
GPS-RX
GPS-LED
GPS-VBAT
GPS
+3V3
GND
JP
G1
B2430UNI
GND
JP1
GND
SCL
CD
CSSD
MISO
VDC
SDA
WP
MOSI
SCK
Modul SD Card
VCC
GND
S2
RDT1034
GND
IC1
RESET
XTAL2
XTAL1
AREF
AVCC
GND
VCC
GND
MEGA16-P
R2
LED2
GND
C4
+3V3
IC2
OUT
GND
IN
GND
+12V
R1
LED1
GND
C3
VCC
IC3
OUT
GND
IN
GND
JP11
+12V
GND
J1
R5
C5
GND
VCC
S1
RDT1034
GND
C1
C2
GND

LAMPIRAN C

PROGRAM PADA PENGONTROL MIKRO ATMEGA16

PROGRAM READ

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.3 Standard
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 3/22/2011
Author  : AN LIE
Company :
Comments:

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 11.059200 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <delay.h>
#include <math.h>
#include <stdio.h>

unsigned long int
FATtype,SectorCluster,BootSector,RootDir,FATregion,DataRegion,SectorPoint,SectorStart,FATemp,TotalS
ector;
unsigned int FATsector, BytePoint;
 unsigned char a[128];//,b[32],c[32],d[32],e[32],f[32];
unsigned char ee[512];
bit i2ced;
unsigned char lastmo;
// I2C Bus functions
#asm
   .equ __i2c_port=0x15 ;PORTC
   .equ __sda_bit=0
```

```
   .equ __scl_bit=1
#endasm
#include <i2c.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index]=data;
   if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
```

```
  if (++rx_counter == RX_BUFFER_SIZE)
    {
    rx_counter=0;
    rx_buffer_overflow=1;
    };
  };
}
#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// USART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE<256
unsigned char tx_wr_index,tx_rd_index,tx_counter;
#else
unsigned int tx_wr_index,tx_rd_index,tx_counter;
#endif

// USART Transmitter interrupt service routine
interrupt [USART_TXC] void usart_tx_isr(void)
{
if (tx_counter)
  {
  --tx_counter;
```

```
   UDR=tx_buffer[tx_rd_index];
   if (++tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
   };
}

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY)==0))
   {
   tx_buffer[tx_wr_index]=c;
   if (++tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
   ++tx_counter;
   }
else
   UDR=c;
#asm("sei")
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

// SPI functions
#include <spi.h>

// Declare your global variables here
#define card PINB.2
#define wprotect PINB.3
#define cs PORTB.4

void delayer(void)
   {
            delay_us(100);
   }
```

```
void i2cstart(void)
        {
        unsigned char imulti;
        delayer();
        i2ced=0;
  if (lastmo==0x38)
    {
    TWCR=0x04;//80;         //stop
    }
        TWCR=0xA4;      //start
        imulti=0;
        while(!(TWCR&0x80))
                {
                imulti++;
        delayer();
    if ((TWSR==0)||((imulti>5)&&(TWSR==0xF8)))
      {
      TWCR=0x04;//80;       //stop
      TWCR=0xA4;            //start
      imulti=0;
      }
                }
        if ((TWSR==0x08)||(TWSR==0x10))
    {
    i2ced=1;
    }
        }

void i2cwbyte(unsigned char wb)
        {
        unsigned char imulti;
        delayer();
        if (i2ced==1)
                {
                TWDR=wb;
                TWCR=0x84;      //send
                imulti=0;
                while(!(TWCR&0x80))
                        {
                        imulti++;
```

```
            delayer();
        if ((TWSR==0)||((imulti>5)&&(TWSR==0xF8)))
          {
          i2ced=0;
          break;
          }
                                    }
      if ((TWSR==0)||(TWSR==0x20)||(TWSR==0x30)||(TWSR==0x38)||(TWSR==0x48)) i2ced=0;
      lastmo=TWSR;
      if ((TWSR==0x0)||(TWSR==0x38)||(TWSR==0xF8))
        {
        TWCR=0x04;//80;
        }
                        }
            }

unsigned char i2crbyte(unsigned char rb)
            {
            unsigned char imulti;
            delayer();
            TWDR=0;
            if (i2ced==1)
                        {
                        rb<<=6;
                        TWCR = 0x84|rb;
                        imulti=0;
                        while(!(TWCR&0x80))
                                    {
                                    imulti++;
            delayer();
        if ((TWSR==0)||((imulti>5)&&(TWSR==0xF8)))
          {
          i2ced=0;
          break;
          }
                                    }
      if ((TWSR==0)||(TWSR==0x38)||(TWSR==0x48)||(TWSR==0x58)) i2ced=0;
      lastmo=TWSR;
      if (TWSR==0x38)
        {
        TWCR=0x04;//80;
```

```
        }
                        }
            return TWDR;
            }

void i2cstop(void)
            {
            delayer();
            if (i2ced==1) TWCR=0x94;   //stop
            i2ced=0;
            }

void linefeed(void)
{
putchar(0x0D);
putchar(0x0A);
}

void delayus5(void)
{
delay_us(5);
}

void write_SPI (unsigned char dataspi)
{
SPDR=dataspi;
while ((SPSR&0x80)!=0x80);
delayus5();
}

void prints(unsigned long int nominal)
  {
  unsigned char xsdprints,nolish;
  unsigned long int divider=1000000000;

  xsdprints=0x30+(nominal/divider);
  if (xsdprints!=0x30)
    {
    putchar(xsdprints);
    nolish=1;
    }
```

```
  else nolish=0;
  do
    {
    xsdprints=0x30+((nominal%divider)/(divider/10));
    if ((xsdprints!=0x30)||(nolish==1))
      {
      putchar(xsdprints);
      nolish=1;
      }
    divider=divider/10;
    } while (divider>10);
  xsdprints=0x30+(nominal%10);
  putchar(xsdprints);
  linefeed();
  }

void FMread(unsigned char pager)
  {
  unsigned char FMcount=0, slad;

  slad=0xA0+((pager&0x07)<<1);
  i2cstart();
  i2cwbyte(slad);
  i2cwbyte(0);

  slad=slad|0x01;
  i2cstart();
  i2cwbyte(slad);
  do
    {
    ee[FMcount]=i2crbyte(1);
    FMcount++;
    } while (FMcount<255);
  ee[FMcount]=i2crbyte(0);
  i2cstop();
  }

unsigned char SDreset(void)
{
      unsigned char xsdreset;
      xsdreset=0;
```

```
            do
            {
                write_SPI(0xFF);
                xsdreset++;
            }while (xsdreset<10);

        cs=0;
        delayus5();
        write_SPI(0x40);
        xsdreset=0;
            do
            {
                write_SPI(0x00);
                xsdreset++;
            }while (xsdreset<4);

        write_SPI(0x95);
        SPDR=0xFF;
        xsdreset=0;
            do
            {
                if (SPDR==0xFF) write_SPI(0xFF);
                else break;
                 xsdreset++;
            }while (xsdreset<9);

        delayus5();
        cs=1;

        putsf("Reset - ");
        if (SPDR==1) putsf("OK");
        else putsf("Error");
        linefeed();
        //delay_ms(1000);
return SPDR;
}

unsigned char SDinit(void)
  {
  unsigned char xsdinit, ysdinit;
```

```
 ysdinit=0;
 do
   {
   write_SPI(0xFF);
   delayus5();
   cs=0;
   delayus5();
   write_SPI(0x41);
   xsdinit=0;
   do
     {
     write_SPI(0x00);
     xsdinit++;
     } while (xsdinit<4);
   write_SPI(0xFF);
   SPDR=0xFF;
   xsdinit=0;
   do
     {
     if (SPDR==0xFF) write_SPI(0xFF); else break;
     xsdinit++;
     } while (xsdinit<9);
   delayus5();
   cs=1;
   if (SPDR==0) break;
   ysdinit++;
   } while (ysdinit<250);

 putsf("Init - ");
 if (SPDR==0) putsf("OK"); else putsf("Error");
 linefeed();

 return SPDR;
 }

void SDread(unsigned long int sectoraddress)
 {
 unsigned char xsdread, ysdread, slad;
 unsigned int xsdrcount;

 sectoraddress*=0x200;
```

```
ysdread=0;
do
  {
  write_SPI(0xFF);
  delayus5();
  cs=0;
  delayus5();
  write_SPI(0x51);
  xsdread=(sectoraddress>>24)&0xFF;
  write_SPI(xsdread);
  xsdread=(sectoraddress>>16)&0xFF;
  write_SPI(xsdread);
  xsdread=(sectoraddress>>8)&0xFF;
  write_SPI(xsdread);
  xsdread=sectoraddress&0xFF;
  write_SPI(xsdread);
  write_SPI(0xFF);
  SPDR=0xFF;
  xsdread=0;
  do
    {
    if (SPDR==0xFF) write_SPI(0xFF); else break;
    xsdread++;
    } while (xsdread<9);
  if (SPDR==0) break;
  ysdread++;
  } while (ysdread<250);

ysdread=0;
do
  {
  SPDR=0xFF;
  xsdread=0;
  do
    {
    if (SPDR==0xFF) write_SPI(0xFF); else break;
    xsdread++;
    } while (xsdread<9);
  if (SPDR==0xFE) break;
  ysdread++;
  } while (ysdread<250);
```

```
  i2cstart();
  slad=0xA0;
  i2cwbyte(slad);
  i2cwbyte(0);

  xsdrcount=0;
  do
    {
    write_SPI(0xFF);
    ee[xsdrcount%256]=SPDR;
    i2cwbyte(ee[xsdrcount%256]);
    xsdrcount++;
    if (xsdrcount==256)
      {
      i2cstop();
      i2cstart();
      slad=0xA2;
      i2cwbyte(slad);
      i2cwbyte(0);
      }
    } while (xsdrcount<512);

  i2cstop();

  write_SPI(0xFF);
  write_SPI(0xFF);
  delayus5();
  cs=1;
  }

void SDwrite(unsigned long int sectoraddress)
  {
  unsigned char xsdwrite, ysdwrite;
  unsigned int xsdwcount;

  sectoraddress*=0x200;
  ysdwrite=0;
  do
    {
    write_SPI(0xFF);
    delayus5();
```

```
  cs=0;
  delayus5();
  write_SPI(0x58);
  xsdwrite=(sectoraddress>>24)&0xFF;
  write_SPI(xsdwrite);
  xsdwrite=(sectoraddress>>16)&0xFF;
  write_SPI(xsdwrite);
  xsdwrite=(sectoraddress>>8)&0xFF;
  write_SPI(xsdwrite);
  xsdwrite=sectoraddress&0xFF;
  write_SPI(xsdwrite);
  write_SPI(0xFF);
  SPDR=0xFF;
  xsdwrite=0;
  do
    {
    if (SPDR==0xFF) write_SPI(0xFF); else break;
    xsdwrite++;
    } while (xsdwrite<9);
  if (SPDR==0) break;
  ysdwrite++;
  } while (ysdwrite<250);

xsdwrite=0;
do
  {
  write_SPI(0xFF);
  xsdwrite++;
  } while (xsdwrite<9);

write_SPI(0xFE);

xsdwcount=0;
do
  {
  write_SPI(ee[xsdwcount]);
  xsdwcount++;
  } while (xsdwcount<512);

write_SPI(0xFF);
write_SPI(0xFF);
```

```
  ysdwrite=0;
  do
    {
    SPDR=0xFF;
    xsdwrite=0;
    do
      {
      if (SPDR==0xFF) write_SPI(0xFF); else break;
      xsdwrite++;
      } while (xsdwrite<9);
    xsdwrite=SPDR&0x0F;
    if (xsdwrite==0x05) break;
    ysdwrite++;
    } while (ysdwrite<250);

  SPDR=0xFF;
  ysdwrite=0;
  do
    {
    write_SPI(0xFF);
    if (SPDR==0x00) break;
    ysdwrite++;
    } while (ysdwrite<250);

  SPDR=0xFF;
  ysdwrite=0;
  do
    {
    write_SPI(0xFF);
    if (SPDR!=0x00) break;
    ysdwrite++;
    } while (ysdwrite<250);
  cs=1;
  }

 void SDFirst(void)
  {
  SDreset();
  SDinit();
//=================
//Locate Bootsector
```

```
//-----------------
  SDread(0x00);
  FMread(1);
  BootSector=((unsigned long int) ee[457]*0x1000000)+((unsigned long int) ee[456]*0x10000)+((unsigned
int) ee[455]*0x100)+ee[454];
//==================

//==================
//Locate FAT Region
//-----------------
  SDread(0x00);
  FMread(0);
  FATregion=BootSector+((unsigned int) ee[15]*0x100)+ee[14];
  putsf("FATregion: ");
  prints(FATregion);
//==================

//======================
//Locate Root Directory
//----------------------
  FATsector=(((unsigned int) ee[23]*0x100)+ee[22]);
  RootDir=FATregion+((unsigned long int) ee[16]*FATsector);
  putsf("RootDir: ");
  prints(RootDir);
//======================

//===================
//Locate Data Region
//-------------------
  DataRegion=RootDir+ceil( (float) ((unsigned long int) (((unsigned long int)
ee[18]*0x100)+ee[17])*0x20)/(((unsigned int) ee[12]*0x100)+ee[11]) );
  putsf("DataRegion: ");
  prints(DataRegion);
//===================

//=========================
//Calculate Sector/Cluster
//-------------------------
  SectorCluster=ee[13];
  putsf("SectorCluster: ");
  prints(SectorCluster);
```

```
//=========================
//===============
//Locate FAT Type
//---------------
  FATemp=(((unsigned int) ee[20]*0x100)+ee[19]);
  if (FATemp==0) FATemp=((unsigned long int) ee[35]*0x1000000)+((unsigned long int)
ee[34]*0x10000)+((unsigned int) ee[33]*0x100)+ee[32];
  TotalSector=FATemp;
  FATemp=floor( (float) (TotalSector-(DataRegion-BootSector))/SectorCluster);
  if (FATemp<4085) FATtype=12;
  else if (FATemp<65525) FATtype=16;
  else FATtype=32;

  putsf("Type: FAT");
  prints(FATtype);
//===============

//=========================
//Calculate Card Capacity
//-------------------------
  putsf("Size (MB): ");
  FATemp=((unsigned long int) TotalSector*(((unsigned int) ee[12]*0x100)+ee[11]))/1048576;
  prints(FATemp);
//=========================
  }

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=Out Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTB=0x00;
```

```
DDRB=0xB0;
// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
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: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0xD8;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x47;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
```

```
// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 2*86.400 kHz
// SPI Clock Phase: Cycle Half
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=0x53;
SPSR=0x01;

// I2C Bus initialization
i2c_init();

// Global enable interrupts
#asm("sei")

while (1)
      {
      // Place your code here
   putsf("Insert card");
   linefeed();
   while (card==1);
   delay_ms(100);
   putsf("Card is ");
   if (wprotect==0) putsf("write enabled"); else putsf("read only");
   linefeed();
   SDFirst();
   putsf("Complete");
   linefeed();
   putsf("Eject card");
   linefeed();
   linefeed();
   while (card==0);
      };
}
```

PROGRAM AKUISISI DATA GPS PADA SD CARD

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.3 Standard
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 6/24/2011
Author  : AN LIE
Company :
Comments:

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 11.059200 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <delay.h>
#include <math.h>

char buffer[32],timer[32];
int a,b,c,x,y,z,q,u;

int k,n,p;

int w;

int t;

unsigned long int BootSector,SectorPoint,SectorStart,FATemp,TotalSector;
unsigned int FATsector, BytePoint;
unsigned char ee[475];
```

```
unsigned long int FATregion=8;
unsigned long int RootDir=496;
unsigned long int DataRegion=528;
unsigned char SectorCluster=32;
unsigned char FATtype=16;

bit i2ced;
unsigned char lastmo;

#define card PINB.2
#define wprotect PINB.3
#define cs PORTB.4

// I2C Bus functions
#asm
  .equ __i2c_port=0x15 ;PORTC
  .equ __sda_bit=1
  .equ __scl_bit=0
#endasm
#include <i2c.h>

// Alphanumeric LCD Module functions
#asm
  .equ __lcd_port=0x1B ;PORTA
#endasm
#include <lcd.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
```

```
// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

char index = 0;
char text[128];

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
  {
  rx_buffer[rx_wr_index]=data;
  if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
  if (++rx_counter == RX_BUFFER_SIZE)
    {
    rx_counter=0;
    rx_buffer_overflow=1;
    };
  };

  index++;
  text[index]=data;
  if(text[index]==36)index=0;
  }

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
```

```
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// USART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE<256
unsigned char tx_wr_index,tx_rd_index,tx_counter;
#else
unsigned int tx_wr_index,tx_rd_index,tx_counter;
#endif

// USART Transmitter interrupt service routine
interrupt [USART_TXC] void usart_tx_isr(void)
{
if (tx_counter)
   {
   --tx_counter;
   UDR=tx_buffer[tx_rd_index];
   if (++tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
   };
}

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
```

```
{
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY)==0))
   {
   tx_buffer[tx_wr_index]=c;
   if (++tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
   ++tx_counter;
   }
else
   UDR=c;
#asm("sei")
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here
}

// SPI functions
#include <spi.h>

// Declare your global variables here

void delayer(void)
      {
            delay_us(100);
      }

void i2cstart(void)
         {
            unsigned char imulti;
            delayer();
            i2ced=0;
```

```
        if (lastmo==0x38)
            {
                TWCR=0x04;//80;         //stop
            }

        TWCR=0xA4;//start
        imulti=0;

            while(!(TWCR&0x80))
                    {
                        imulti++;
                    delayer();
                if ((TWSR==0)||((imulti>5)&&(TWSR==0xF8)))
                    {
                        TWCR=0x04;//80;         //stop
                        TWCR=0xA4;//start
                        imulti=0;
                    }
                    }

            if ((TWSR==0x08)||(TWSR==0x10))
            {
                i2ced=1;
            }
    }

void i2cwbyte(unsigned char wb)
     {
        unsigned char imulti;
        delayer();

            if (i2ced==1)
                    {
                        TWDR=wb;
                        TWCR=0x84; //send
                        imulti=0;

                        while(!(TWCR&0x80))
                            {
                                imulti++;
                        delayer();
```

```
                    if ((TWSR==0)||((imulti>5)&&(TWSR==0xF8)))
                    {
                        i2ced=0;
                        break;
                    }
                            }

            if ((TWSR==0)||(TWSR==0x20)||(TWSR==0x30)||(TWSR==0x38)||(TWSR==0x48))
i2ced=0;
            lastmo=TWSR;
            if ((TWSR==0x0)||(TWSR==0x38)||(TWSR==0xF8))
                {
                    TWCR=0x04;//80;
                }
                    }
        }

unsigned char i2crbyte(unsigned char rb)
        {
            unsigned char imulti;
            delayer();
            TWDR=0;
                if (i2ced==1)
                        {
                            rb<<=6;
                            TWCR = 0x84|rb;
                            imulti=0;

                                while(!(TWCR&0x80))
                                    {
                                        imulti++;
                                delayer();
                                if ((TWSR==0)||((imulti>5)&&(TWSR==0xF8)))
                                    {
                                        i2ced=0;
                                        break;
                                    }
                                        }

                    if ((TWSR==0)||(TWSR==0x38)||(TWSR==0x48)||(TWSR==0x58)) i2ced=0;
                    lastmo=TWSR;
```

```
                    if (TWSR==0x38)
                        {
                            TWCR=0x04;//80;
                        }
                        }
            return TWDR;
        }

void i2cstop(void)
        {
            delayer();
            if (i2ced==1) TWCR=0x94;          //stop
            i2ced=0;
        }

void delayus5(void)
    {
        delay_us(5);
    }

void write_SPI (unsigned char dataspi)
    {
        SPDR=dataspi;
        while ((SPSR&0x80)!=0x80);
        delayus5();
    }


void FMread(unsigned char pager)
    {
        unsigned char FMcount=0, slad;
        slad=0xA0+((pager&0x07)<<1);
        i2cstart();
        i2cwbyte(slad);
        i2cwbyte(0);
        slad=slad|0x01;
        i2cstart();
        i2cwbyte(slad);
```

```
            do
            {
                ee[FMcount]=i2crbyte(1);
                FMcount++;
            } while (FMcount<255);

        ee[FMcount]=i2crbyte(0);
        i2cstop();
    }

void FMwrite(unsigned char pager)
    {
        unsigned char FMcount=0, slad;
        slad=0xA0+((pager&0x07)<<1);
        i2cstart();
        i2cwbyte(slad);
        i2cwbyte(0);

            do
            {
                i2cwbyte(ee[FMcount]);
                FMcount++;
            } while (FMcount<255);

        i2cwbyte(ee[FMcount]);
        i2cstop();
    }

unsigned char SDreset(void)
    {
        unsigned char xsdreset;
        xsdreset=0;

            do
            {
                write_SPI(0xFF);
                xsdreset++;
            }while (xsdreset<10);

        cs=0;
        delayus5();
```

```
    write_SPI(0x40);
    xsdreset=0;

        do
        {
            write_SPI(0x00);
            xsdreset++;
        }while (xsdreset<4);

    write_SPI(0x95);
    SPDR=0xFF;
    xsdreset=0;

        do
        {
            if (SPDR==0xFF) write_SPI(0xFF);
            else break;
            xsdreset++;
        }while (xsdreset<9);

    delayus5();
    cs=1;

    if (SPDR==1)
        {
            lcd_clear();
            lcd_putsf("Reset - OK");
            delay_ms(1000);
        }
    else
        {
            lcd_clear();
            lcd_putsf("Reset - Error");
            delay_ms(1000);
        }

    return SPDR;
}
```

```
unsigned char SDinit(void)
    {
        unsigned char xsdinit, ysdinit;
        ysdinit=0;

        do
        {
            write_SPI(0xFF);
            delayus5();
            cs=0;
            delayus5();
            write_SPI(0x41);
            xsdinit=0;

            do
            {
                write_SPI(0x00);
                xsdinit++;
            } while (xsdinit<4);

        write_SPI(0xFF);
        SPDR=0xFF;
        xsdinit=0;

            do
            {
                if (SPDR==0xFF) write_SPI(0xFF); else break;
                xsdinit++;
            } while (xsdinit<9);


            delayus5();
            cs=1;
            if (SPDR==0) break;
            ysdinit++;
        } while (ysdinit<250);

        if (SPDR==0)
            {
                lcd_clear();
                lcd_putsf("Init - OK");
```

```
                delay_ms(1000);
            }
        else
            {
                lcd_clear();
                putsf("Init - Error");
                delay_ms(1000);
            }

        return SPDR;
    }

void SDread(unsigned long int sectoraddress)
    {
        unsigned char xsdread, ysdread, slad;
        unsigned int xsdrcount;
        sectoraddress*=0x200;
        ysdread=0;

        do
        {
            write_SPI(0xFF);
            delayus5();
            cs=0;
            delayus5();
            write_SPI(0x51);
            xsdread=(sectoraddress>>24)&0xFF;
            write_SPI(xsdread);
            xsdread=(sectoraddress>>16)&0xFF;
            write_SPI(xsdread);
            xsdread=(sectoraddress>>8)&0xFF;
            write_SPI(xsdread);
            xsdread=sectoraddress&0xFF;
            write_SPI(xsdread);
            write_SPI(0xFF);
            SPDR=0xFF;
            xsdread=0;
```

```
        do
        {
            if (SPDR==0xFF) write_SPI(0xFF); else break;
            xsdread++;
        } while (xsdread<9);

        if (SPDR==0) break;
        ysdread++;

} while (ysdread<250);

ysdread=0;
do
{
        SPDR=0xFF;
        xsdread=0;

         do
         {
            if (SPDR==0xFF) write_SPI(0xFF); else break;
            xsdread++;
         } while (xsdread<9);

        if (SPDR==0xFE) break;
        ysdread++;
}while (ysdread<250);

i2cstart();
slad=0xA0;
i2cwbyte(slad);
i2cwbyte(0);

xsdrcount=0;
        do
        {
            write_SPI(0xFF);
            ee[xsdrcount%256]=SPDR;
            i2cwbyte(ee[xsdrcount%256]);
            xsdrcount++;
```

```
                    if (xsdrcount==256)
                        {
                            i2cstop();
                            i2cstart();
                            slad=0xA2;
                            i2cwbyte(slad);
                            i2cwbyte(0);
                        }
                } while (xsdrcount<512);

            i2cstop();
            write_SPI(0xFF);
            write_SPI(0xFF);
            delayus5();
            cs=1;
        }

void SDwrite(unsigned long int sectoraddress)
        {
            unsigned char xsdwrite, ysdwrite;
            unsigned int xsdwcount;
            sectoraddress*=0x200;
            ysdwrite=0;
                do
                {
                    write_SPI(0xFF);
                    delayus5();
                    cs=0;
                    delayus5();
                    write_SPI(0x58);
                    xsdwrite=(sectoraddress>>24)&0xFF;
                    write_SPI(xsdwrite);
                    xsdwrite=(sectoraddress>>16)&0xFF;
                    write_SPI(xsdwrite);
                    xsdwrite=(sectoraddress>>8)&0xFF;
                    write_SPI(xsdwrite);
                    xsdwrite=sectoraddress&0xFF;
                    write_SPI(xsdwrite);
                    write_SPI(0xFF);
                    SPDR=0xFF;
                    xsdwrite=0;
```

```
            do
            {
                if (SPDR==0xFF) write_SPI(0xFF); else break;
                xsdwrite++;
            } while (xsdwrite<9);

            if (SPDR==0) break;
            ysdwrite++;
        } while (ysdwrite<250);

xsdwrite=0;
do
{
        write_SPI(0xFF);
        xsdwrite++;
} while (xsdwrite<9);

write_SPI(0xFE);
xsdwcount=0;

do
{
        write_SPI(ee[xsdwcount]);
        xsdwcount++;
} while (xsdwcount<512);

write_SPI(0xFF);
write_SPI(0xFF);
ysdwrite=0;

do
{
        SPDR=0xFF;
        xsdwrite=0;
        do
        {
            if (SPDR==0xFF) write_SPI(0xFF); else break;
            xsdwrite++;
} while (xsdwrite<9);

xsdwrite=SPDR&0x0F;
```

```
            if (xsdwrite==0x05) break;
            ysdwrite++;
        } while (ysdwrite<250);

        SPDR=0xFF;
        ysdwrite=0;
            do
                {
                    write_SPI(0xFF);
                    if (SPDR==0x00) break;
                    ysdwrite++;
                } while (ysdwrite<250);

        SPDR=0xFF;
        ysdwrite=0;
            do
                {
                    write_SPI(0xFF);
                    if (SPDR!=0x00) break;
                    ysdwrite++;
                } while (ysdwrite<250);
        cs=1;
        }

void SDFirst(void)
        {
            unsigned char xsdfirst;

            SDreset();
            SDinit();
//================
//Write FAT Table
//---------------
         lcd_clear();
         lcd_putsf("Fat Table...");
         delay_ms(1000);
         ee[0]=0xF8;
         ee[1]=0xFF;
         ee[2]=0xFF;
         ee[3]=0xFF;
         ee[4]=0x03;
```

```
        ee[5]=0x00;
        ee[6]=0xFF;
        ee[7]=0xFF;

        FATemp=8;
        do
          {
          ee[FATemp%256]=0x00;
          FATemp++;
             if (FATemp==256)
             {
                   FMwrite(2);
             }
          } while (FATemp<512);
        FMwrite(3);
        SDwrite(FATregion);

//===============
//Card Name Entry
//---------------
        lcd_clear();
        lcd_putsf("Card Name...");
        delay_ms(1000);
        FMread(2);
        FATemp=0;
        do
          {
          ee[FATemp]=0x00;
          FATemp++;
          } while (FATemp<0x80);

        ee[0]=0x4D;
        ee[1]=0x45;
        ee[2]=0x4D;
        ee[3]=0x4F;
        ee[4]=0x52;
        ee[5]=0x59;
        ee[6]=0x5F;
        ee[7]=0x43;
        ee[8]=0x41;
        ee[9]=0x52;
```

```
        ee[10]=0x44;
        ee[11]=0x08;
        ee[22]=0x8C;
        ee[23]=0x86;
        ee[24]=0x4B;
        ee[25]=0x36;

//================
//File Name Entry
//---------------
        lcd_clear();
        lcd_putsf("File Name...");
        delay_ms(1000);
        ee[32]=0x54;
        ee[33]=0x45;
        ee[34]=0x53;
        ee[35]=0x54;
        ee[36]=0x49;
        ee[37]=0x4E;
        ee[38]=0x47;
        ee[39]=0x53;
        ee[40]=0x54;
        ee[41]=0x58;
        ee[42]=0x54;
        ee[43]=0x20;
        ee[58]=0x02;
    FMwrite(2);
    SDwrite(RootDir);

//====================
//Write File Contents
//-------------------
    lcd_clear();
    lcd_putsf("File Contents...");
    delay_ms(1000);
    FMread(2);
    FATemp=0;
    SectorStart=DataRegion+(((((unsigned int) ee[0x20+27]*0x100)+ee[0x20+26])-2)*SectorCluster);
    SectorPoint=SectorStart;

    FATemp=0;
```

```
      do
        {
          ee[FATemp]=0x00;
          FATemp++;
        } while (FATemp<475);

    t=0;
    q=0;
    u=1;
    k=0;
    p=0;
    while(u==1)
    {

      if(text[index] == 71 && text[index+1] == 80 && text[index+2] == 71 && text[index+3] == 71 &&
text[index+4] == 65)a=1;
      else a=0;

      while(a==1)
      {
      n=k*48;
      ee[n]=0x24;
      ee[n+1]=text[1];
      ee[n+2]=text[2];
      ee[n+3]=text[3];
      ee[n+4]=text[4];
      ee[n+5]=text[5];
      ee[n+6]=text[6];
      ee[n+7]=text[7];
      ee[n+8]=text[8];
      ee[n+9]=text[9];
      ee[n+10]=text[10];
      ee[n+11]=text[11];
      ee[n+12]=text[12];
      ee[n+13]=text[13];
      ee[n+14]=text[14];
      ee[n+15]=text[15];
      ee[n+16]=text[16];
      ee[n+17]=text[17];
      ee[n+18]=text[18];
      ee[n+19]=text[19];
```

```
ee[n+20]=text[20];
ee[n+21]=text[21];
ee[n+22]=text[22];
ee[n+23]=text[23];
ee[n+24]=text[24];
ee[n+25]=text[25];
ee[n+26]=text[26];
ee[n+27]=text[27];
ee[n+28]=text[28];
ee[n+29]=text[29];
ee[n+30]=text[30];
ee[n+31]=text[31];
ee[n+32]=text[32];
ee[n+33]=text[33];
ee[n+34]=text[34];
ee[n+35]=text[35];
ee[n+36]=text[36];
ee[n+37]=text[37];
ee[n+38]=text[38];
ee[n+39]=text[39];
ee[n+40]=text[40];
ee[n+41]=text[41];
ee[n+42]=text[42];
ee[n+43]=text[43];
ee[n+44]=0x00;
ee[n+45]=0x00;
ee[n+46]=0x0D;
ee[n+47]=0x0A;
FMwrite(2);
SDwrite(SectorPoint+p);
delay_ms(100);
k++;
lcd_clear();
lcd_gotoxy(0,0);
sprintf(buffer,"k= %i - p= %i ",k,p);
lcd_puts(buffer);
delay_ms(15000);
```

```
if(k%8==0)
      {
            p=p+1;
            k=0;
       }
if (p==16)u=2;
a=0;
}
}

while(u==2)
{

      //===============
      //Write File Size
      //---------------
            lcd_clear();
            lcd_putsf("File Size...");
            delay_ms(1000);
            SDread(RootDir);
            FMread(1);
            FMwrite(3);
            FMread(0);
            FATemp=p*512;
            ee[63]=FATemp>>24;
            ee[62]=FATemp>>16;
            ee[61]=FATemp>>8;
            ee[60]=FATemp;
            FMwrite(2);
            SDwrite(RootDir);

      //===============
      //Setting
      //---------------
            lcd_clear();
            lcd_putsf("Setting...");
            delay_ms(1000);
            FMread(2);
            FATemp=0;
```

```
                do
                {
                    ee[FATemp]=0x00;
                    FATemp++;
                } while (FATemp<256);

                ee[0]=0x00;
                ee[1]=0x00;
                ee[2]=0xFF;
                ee[3]=0xFF;
                ee[4]=0xFF;
                ee[5]=0xFF;

                FMwrite(2);
                SDwrite(8);
                u=0;
        }
}

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=Out Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0xB0;

// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=In Func0=In
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=T State0=T
PORTC=0x00;
DDRC=0xFC;
```

```
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=P State6=P State5=P State4=P State3=T State2=T State1=T State0=T
PORTD=0xF0;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 10.800 kHz
// Mode: CTC top=OCR0
// OC0 output: Disconnected
TCCR0=0x0D;
TCNT0=0x00;
OCR0=0xFF;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 10.800 kHz
// Mode: CTC top=OCR1A
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x2A;
OCR1AL=0x30;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
```

```
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x10;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 4800
UCSRA=0x00;
UCSRB=0xD8;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x8F;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 2*86.400 kHz
// SPI Clock Phase: Cycle Half
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
```

```
SPCR=0x53;
SPSR=0x01;

// I2C Bus initialization
i2c_init();

// LCD module initialization
lcd_init(16);

// Global enable interrupts
#asm("sei")

a=0;
x=0;
w=0;
t=0;
while (1)
    {
    // Place your code here

while (w==0)
{
      lcd_clear();
      lcd_putsf("Push The Button");
      delay_ms(500);
      if(PIND.6==0)w=6;
}

while(w==6)
      {
            if(text[index] == 71 && text[index+1] == 80 && text[index+2] == 71 && text[index+3] == 71
&& text[index+4] == 65)a=1;      //Jika GPGGA benar
            else a=0;

            if(a==1 && text[index+42]==48)
            {
                  lcd_clear();
                  lcd_gotoxy(5,0);
                  lcd_putsf("SEARCH");
                  lcd_gotoxy(2,1);
                  lcd_putsf("GPS SATELITE");
```

```
            delay_ms(1000);
        }

        if(a==1 && text[index+42]==49 && (text[index+27]==78||text[index+27]==83))
        {
            lcd_clear();
            lcd_gotoxy(6,0);
            lcd_putsf("FIND");
            lcd_gotoxy(2,1);
            lcd_putsf("GPS SATELITE");
            delay_ms(2000);
            x=5;
        }

while(x==5)
    {
        if(text[index] == 71 && text[index+1] == 80 && text[index+2] == 71 && text[index+3] == 71
&& text[index+4] == 65)a=1;     //Jika GPGGA benar
        else a=0;

        if(a==1)
        {
            lcd_clear();
            lcd_gotoxy(0,0);
            sprintf(buffer,"Lat
=%c%c%c%c%c%c%c%c%c",text[18],text[19],text[20],text[21],text[22],text[23],text[24],text[25],text[26]);
            lcd_puts(buffer);
            lcd_gotoxy(0,1);
            sprintf(buffer,"Long
=%c%c%c%c%c%c%c%c%c%c",text[30],text[31],text[32],text[33],text[34],text[35],text[36],text[37],text[3
8],text[39]);
            lcd_puts(buffer);
            delay_ms(3000);

            lcd_clear();
            lcd_putsf("Insert card");
            delay_ms(1000);
            while (card==1);
            delay_ms(100);
```

```
                if ((wprotect==0)&&(FATtype==16))
                {
                    SDFirst();
                    lcd_clear();
                    lcd_putsf("Write complete");
                    delay_ms(1000);
                }
            else
                {
                    lcd_clear();
                    lcd_putsf("Card is read only");
                    delay_ms(1000);
                }

            lcd_clear();
            lcd_putsf("Eject card");
            delay_ms(3000);
            lcd_clear();
            x=0;
            delay_ms(100);
            w=0;

        }

    }
  }

    }
}
```

LAMPIRAN D

DATASHEET

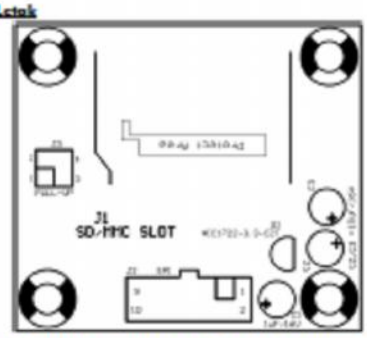
EMS (Embedded Module Series)
SD/MMC/FRAM

EMS SD/MMC/FRAM merupakan suatu modul untuk mempermudah antarmuka antara SD Card (atau MMC) dan mikrokontroler dengan tegangan kerja +5 VDC. SD Card (atau MMC) dapat digunakan sebagai memori yang dapat diganti dengan mudah sehingga memudahkan dalam ekspansi ke kapasitas memori yang lebih besar. Tersedia Ferroelectric Nonvolatile RAM (FRAM) yang dapat digunakan sebagai buffer sementara dalam mengakses SD Card (atau MMC) atau sebagai tempat penyimpan data lain. Modul ini dapat digunakan antara lain sebagai penyimpan data pada sistem absensi, sistem antrian, atau aplikasi datalogging lainnya.

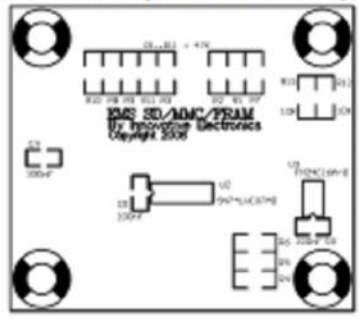
Spesifikasi Hardware

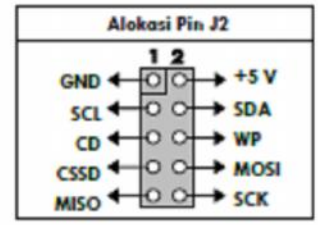
1. Tegangan supply +5 VDC.
2. Jenis kartu yang didukung: SD Card (dan MMC).
3. Antarmuka SD Card (dan MMC) dengan mikrokontroler secara SPI.
4. Tersedia 2 KByte Ferroelectric Nonvolatile RAM FM24C16.
5. Antarmuka FRAM dengan mikrokontroler secara Two-Wire Interface.
6. Tersedia contoh aplikasi untuk DT-51™ Low Cost Series dan DT-AVR Low Cost Series dalam bahasa BASIC untuk MCS-51® (BASCOM-8051®) dan bahasa C untuk AVR® (CodeVisionAVR®).
7. Kompatibel dengan DT-51™ Low Cost Series dan DT-AVR Low Cost Series. Mendukung DT-51™ Minimum System (MinSys) ver 3.0, DT-51™ PetraFux, dan lain-lain.

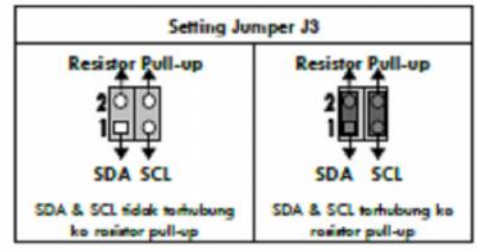
Tata Letak



Terdapat kesalahan cetak pada PCB: J2 ISP seharusnya J2 SPI.







Jumper J3 digunakan untuk resistor pull-up SDA dan SCL. Apabila modul terhubung ke jaringan Two-Wire Interface, maka dalam satu jaringan tersebut hanya perlu memasang pull-up pada salah satu modul saja.

Pin	Nama	Fungsi pada Modul	Keterangan
1	GND	Input	Referensi Ground
2	+5 V	Input	Terhubung ke Sumber Tegangan +5 VDC
3	SCL	Input	Serial Clock untuk akses FRAM
4	SDA	Input/ Output	Serial Data untuk transaksi data dari/ke FRAM
5	CD	Output	Card Detect, berlogika 0 jika ada kartu yang dimasukkan, berlogika 1 jika tidak ada kartu
6	WP	Output	Write Protect, berlogika 0 jika saklar pada SD Card tidak berada pada posisi dikunci, berlogika 1 jika SD Card dalam posisi dikunci
7	CSSD	Input	Chip Select, dibari logika 0 untuk mengakses SD Card, dibari logika 1 jika tidak mengakses SD Card
8	MOSI	Input	Jalur data masuk ke SD Card
9	MISO	Output	Jalur data keluar dari SD Card
10	SCK	Input	Jalur clock dari mikrokontroler untuk mengakses SD Card

Isi CD

1. Contoh Aplikasi dengan CodeVisionAVR® dan BASCOM-8051®.
2. Datasheet.
3. Manual EMS SD/MMC/FRAM.
4. Website Innovative Electronics.

Prosedur Testing

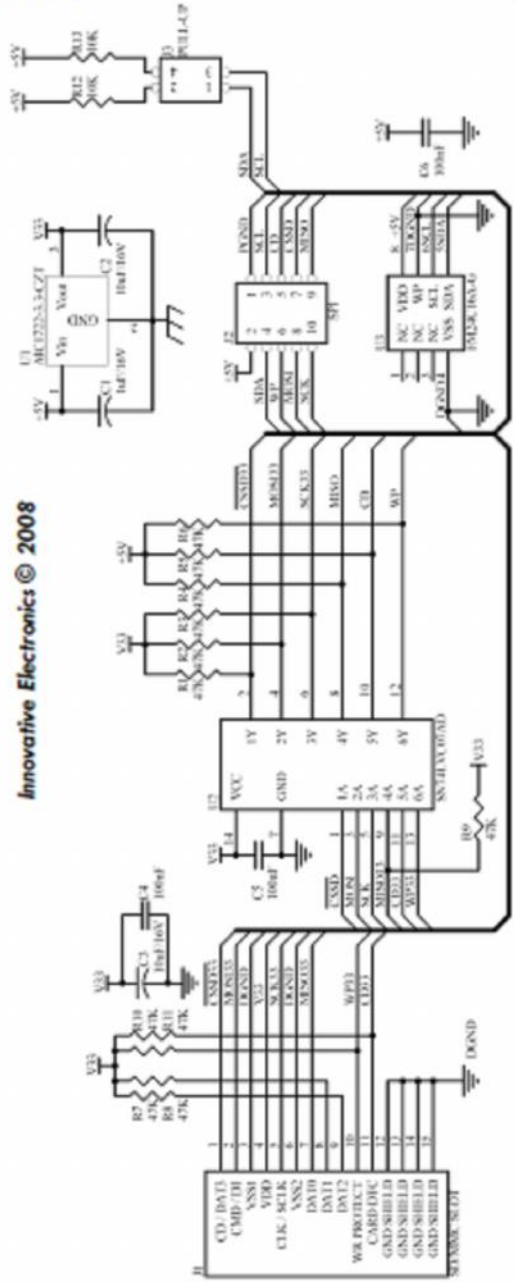
Ikuti petunjuk pada contoh aplikasi **AN SD BAS51.PDF** untuk MCS-51® (BASCOM-8051®) dan **AN SD CVAVR.PDF** untuk AVR® (CodeVisionAVR®).
Tersedia juga **AN SD FAT.PDF** untuk pengenalan awal terhadap protokol SD Card dan format FAT16.

Trademark & Copyright

MCS-51 is a registered trademark of Intel Corporation.
AVR is a registered trademark of Atmel Corporation.
BASCOM-8051 is copyright by MCS Electronics.
CodeVisionAVR is copyright by Pavel Haiduc, HP InfoTech s.r.l.
DT-51 is a trademark of Innovative Electronics.

- *Terima Kasih atas kepercayaan Anda menggunakan produk kami, bila ada kesulitan, pertanyaan atau saran mengenai produk ini silahkan menghubungi technical support kami :*

support@innovativeelectronics.com



EMS SD/MMC/FRAM *Application Note*

Akses SD Card & FRAM Menggunakan AVR

Oleh: Tim IE

Secure Digital (SD) atau MultiMedia Card (MMC) seringkali digunakan sebagai sarana penyimpan data pada Personal Digital Assistant (PDA), kamera digital, dan telepon seluler (ponsel). AN ini akan menggunakan EMS SD/MMC/FRAM sebagai sarana penyimpanan data dengan menggunakan SD card dan FRAM yang tersedia.
Aplikasi ini terdiri dari 4 program:

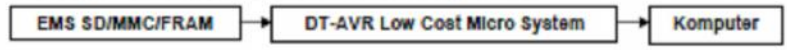
1. DT-AVR Low Cost Micro System membaca parameter format data SD Card lalu mengirimkannya secara UART ke komputer.
   a. Menggunakan EEPROM ATmega8535 sebagai buffer data.
   b. Menggunakan FRAM sebagai buffer data.
2. Berdasar parameter yang sudah dibaca dan sudah dimasukkan ke program, DT-AVR Low Cost Micro System akan menulis sebuah file beserta isinya ke dalam SD Card.
   a. Menggunakan EEPROM ATmega8535 sebagai buffer data.
   b. Menggunakan FRAM sebagai buffer data.

Bahasa pemrograman yang digunakan adalah bahasa C (CodeVisionAVR©).
Modul yang diperlukan:

- 1 DT-AVR Low Cost Micro System,
- 1 EMS SD/MMC/FRAM,
- 1 SD Card dengan format FAT16 (umumnya berkapasitas antara 32 MB hingga 2 GB)

Pastikan SD card yang digunakan tidak menyimpan data penting atau terproteksi karena program pada AN ini akan menghapus/merusak isi sebelumnya.

Adapun blok diagram sistem secara keseluruhan adalah sebagai berikut:



Gambar 1
Blok Diagram

Hubungan antara modul-modul tersebut terdapat pada tabel berikut:

DT-AVR Low Cost Micro System	EMS SD/MMC/FRAM
GND (PORTC pin 1)	GND (J2 pin 1)
VCC (PORTC pin 2)	VCC (J2 pin 2)
PC.0 (PORTC pin 3)**	SCL (J2 pin 3)
PC.1 (PORTC pin 4)**	SDA (J2 pin 4)
PB.2 (PORTB pin 5)*	CD (J2 pin 5)
PB.3 (PORTB pin 6)*	WP (J2 pin 6)
PB.4 (PORTB pin 7)*	CSSD (J2 pin 7)
PB.5 (PORTB pin 8)**	MOSI (J2 pin 8)
PB.6 (PORTB pin 9)**	MISO (J2 pin 9)
PB.7 (PORTB pin 10)**	SCK (J2 pin 10)

* Pin ini tidak mutlak dan dapat diganti pin lain tetapi harus mengubah program
**Pin ini mutlak digunakan jika menggunakan komunikasi TWI dan SPI hardware

Tabel 1
Hubungan DT-AVR Low Cost Micro System dengan EMS SD/MMC/FRAM

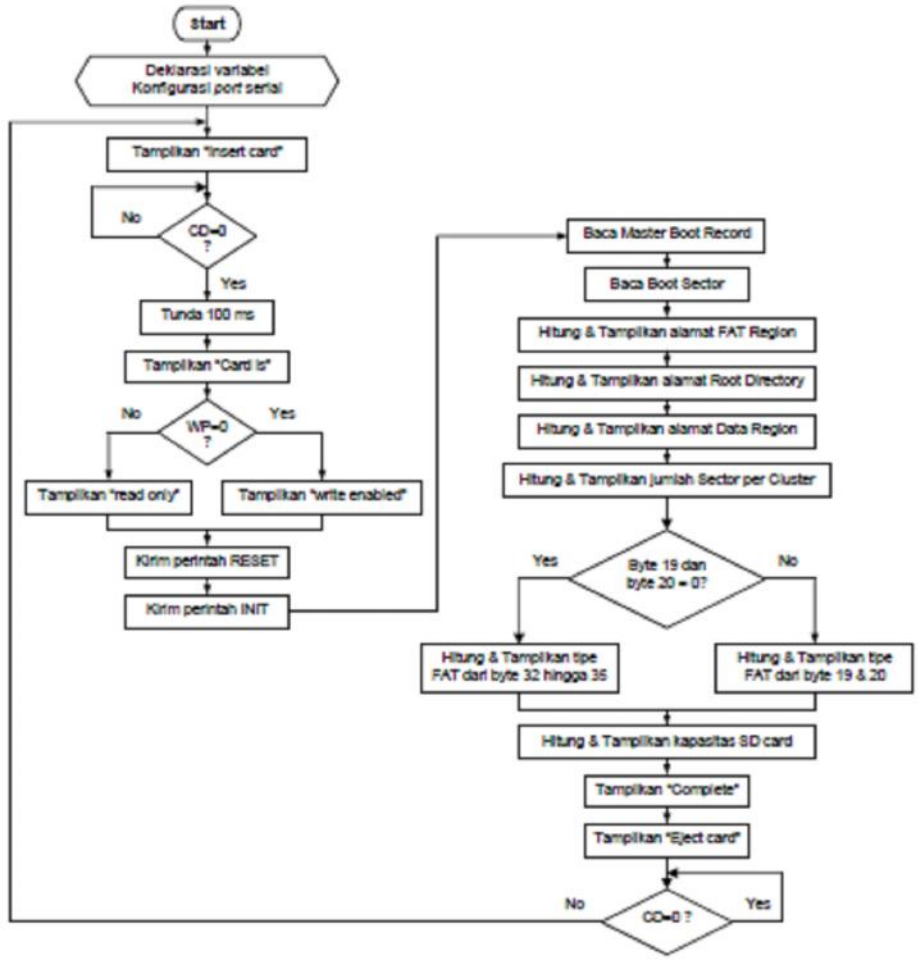
Jumper J4 & J5 (DT-AVR Low Cost Micro System) perlu diatur agar PD.0 dan PD.1 terhubung ke rangkaian UART RS-232 (posisi 1-2). Hubungkan DT-AVR Low Cost Micro System ke komputer melalui kabel serial yang tersedia.

Pada komputer, gunakan program Terminal pada CodeVisionAVR©, HyperTerminal©, Terminal©, atau program terminal lainnya dengan *baud rate* 9600, 8 bit data, 1 bit stop, tanpa bit *parity*, dan tanpa *flow control*.

Setelah semua rangkaian dan catu daya terhubung dengan benar, programlah **Read w EEPROM – cvavr.hex** atau **Read w RAM – cvavr.hex** DT-AVR Low Cost Micro System menggunakan **DT-HIQ AVR In System Programmer** atau divais *programmer* lain yang mendukung.

Jika pemrograman ISP terganggu, coba lepas SD Card atau lepas modul EMS SD/MMC/FRAM atau ubah program dan pindahkan konekasi ke port lain.

F*lowchart* dari program **Read w EEPROM – cvavr.prj** dan **Read w RAM – cvavr.prj** adalah sebagai berikut:



Gambar 2
***Flowchart* Program Read w EEPROM – cvavr.prj dan Read w RAM – cvavr.prj**

Cara kerja program secara garis besar adalah sebagai berikut:

1. Pertama program melakukan deklarasi variabel yang akan digunakan untuk menampung data dan parameter yang berhubungan dengan SD card. Program juga melakukan konfigurasi port serial pada *baud rate* 9600, 8 bit data, 1 bit stop, tanpa bit *parity*, dan tanpa *flow control*.
2. Program mengirimkan "Insert card" ke komputer lalu menunggu adanya kartu yang dimasukkan ke slot.
3. Setelah ada kartu yang dimasukkan ke slot (CD=0), program akan menunda 100 milidetik sebelum mengakses kartu. Hal ini bertujuan untuk memastikan bahwa kartu sudah tepat berada di tempatnya saat akan diakses. Nilai waktu tunda ini dapat diubah atau dihilangkan jika dirasa tidak diperlukan.
4. Program juga akan memeriksa apakah tuas pada SD card dalam posisi terkunci (WP=1) atau tidak (WP=0), dan menampilkan kondisinya pada layar terminal komputer.
5. Langkah awal sebelum mengakses SD card untuk pertama kalinya adalah mengirimkan perintah Reset dan Init ke SD card.
6. Karena SD card yang digunakan memiliki format FAT16, maka parameter yang harus dibaca disesuaikan dengan format FAT16. Yang pertama harus dilakukan adalah membaca Master Boot Record (berada di sektor 0) untuk mengetahui lokasi Boot Sector.
7. Lalu Boot Sector dibaca secara keseluruhan. Nilai pada alamat-alamat tertentu diambil dan dihitung sehingga didapat parameter antara lain: alamat FAT Region, alamat Root Directory, alamat Data Region, jumlah sector per cluster, tipe FAT, dan kapasitas SD card.
8. Program menampilkan pesan "Complete" lalu "Eject Card" dan menunggu kartu dikeluarkan.
9. Berikut ini tampilan yang muncul pada program terminal:

```
Insert card
Card is read only      atau    Card is write enabled
Reset - OK
Init - OK
FATregion: 59
RootDir: 81
DataRegion: 113
SectorCluster: 8
Type: FAT16
Size (MB): 30
Complete
Eject card
```

10. Setelah kartu dikeluarkan (CD=1), program kembali ke langkah nomor 2.

Pada program **Read w EEPROM – cvavr.prj**, data yang dibaca disimpan dalam variabel yang diletakkan pada EEPROM. Hal ini dikarenakan internal RAM Atmega8535 tidak mencukupi. Besarnya EEPROM sesuai dengan *buffer* yang diperlukan untuk akses SD card yaitu 512 byte sehingga proses menjadi mudah. Namun hal ini akan mengakibatkan usia pemakaian EEPROM menjadi cepat habis. **Oleh karena itu, penggunaan EEPROM sebagai *buffer* tidak dianjurkan dan program ini hanya digunakan sebagai contoh saja.**

Sedangkan pada program **Read w RAM – cvavr.prj**, data yang dibaca akan disimpan ke dalam variabel sementara dan akan langsung disimpan ke dalam FRAM. Internal RAM ATmega8535 yang digunakan hanya 256 byte yang disesuaikan dengan kapasitas 1 page FRAM. Oleh karena itu program membutuhkan 2 page FRAM saat mengakses SD card dimana 256 byte pertama diletakkan pada page 0 dan 256 byte kedua diletakkan pada page 1.

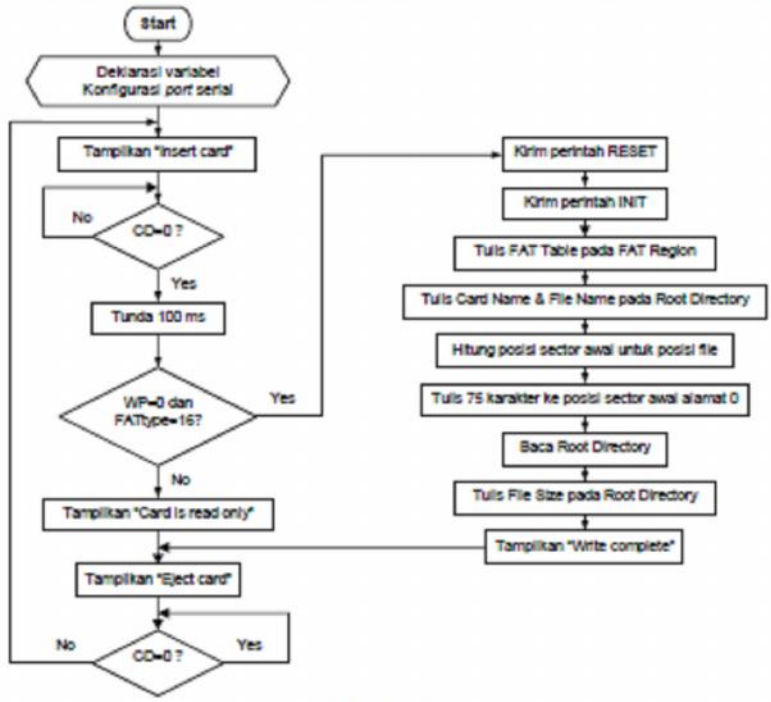
Setelah program **Read w EEPROM – cvavr.prj** dan **Read w RAM – cvavr.prj** berjalan normal dan menampilkan parameter seperti pada langkah 9, maka sesuaikan program **Write w EEPROM – cvavr.prj** dan **Write w RAM – cvavr.prj** dengan parameter tersebut. Contoh setelah baris program untuk deklarasi konstanta pada **Write w EEPROM – cvavr.c** dan **Write w RAM – cvavr.c** diganti:

```
unsigned long int FATregion=59;
unsigned long int RootDir=81;
unsigned long int DataRegion=113;
unsigned char SectorCluster=8;
unsigned char FATtype=16;
```

Periksa dan pastikan bahwa nilai yang dituliskan sama dengan nilai yang telah dibaca oleh program Read w EEPROM – cvavr.prj dan Read w RAM – cvavr.prj.
Kesalahan atau perbedaan nilai dapat mengakibatkan format FAT16 pada SD card kacau dan tidak dapat diakses oleh card reader, kamera digital, PDA, dsb.

Setelah itu lakukan *compile* terhadap **Write w EEPROM - cvavr.prj** dan **Write w RAM – cvavr.prj** dan program ke DT-AVR Low Cost Micro System.

F*lowchart* dari program **Write w EEPROM – cvavr.prj** dan **Write w RAM – cvavr.prj** adalah sebagai berikut:



Gambar 3
***Flowchart* Program Write w EEPROM – cvavr.prj dan Write w RAM – cvavr.prj**

Cara kerja program secara garis besar adalah sebagai berikut:

1. Pertama program melakukan deklarasi variabel yang akan digunakan untuk menampung data dan parameter yang berhubungan dengan SD card. Program juga melakukan konfigurasi port serial pada *baud rate* 9600, 8 bit data, 1 bit stop, tanpa bit *parity*, dan tanpa *flow control*.
2. Program mengirimkan "Insert card" ke komputer lalu menunggu adanya kartu yang dimasukkan ke slot (CD=0).
3. Setelah ada kartu yang dimasukkan ke slot (CD=0), program akan menunda 100 milidetik sebelum mengakses kartu. Hal ini bertujuan untuk memastikan bahwa kartu sudah tepat berada di tempatnya saat akan diakses. Nilai waktu tunda ini dapat diubah atau dihilangkan jika dirasa tidak diperlukan.
4. Jika kartu yang dimasukkan ke slot dalam posisi terkunci (WP=1), maka program akan menampilkan "Card is read only" dan "Eject card" lalu menunggu SD card dikeluarkan. Setelah SD card dikeluarkan (CD=1), program kembali ke langkah nomor 2.
5. Jika kartu yang dimasukkan tidak terkunci dan tipe FAT yang dimasukkan adalah FAT16, maka program akan melanjutkan akses ke SD card.

6. Langkah awal sebelum mengakses SD card untuk pertama kalinya adalah mengirimkan perintah Reset dan Init ke SD card.
7. Karena SD card yang digunakan memiliki format FAT16, maka proses menulis file harus disesuaikan dengan format FAT16. Program akan menulis tabel FAT pada FAT Region.
8. Program akan menulis Nama Kartu dan Nama File pada Root Directory.
9. Nilai Root Directory (yang terletak di FRAM page 2) akan dibaca untuk mengetahui posisi sector awal untuk file.
10. Isi file sebanyak 75 karakter dituliskan ke posisi sector awal mulai alamat 0.
11. Program membaca Root Directory.
12. Parameter ukuran file diubah lalu dituliskan kembali ke Root Directory.
13. Program menampilkan pesan "Write Complete" lalu "Eject Card" dan menunggu kartu dikeluarkan.
14. Berikut ini tampilan yang muncul pada program terminal jika SD card tidak terkunci dan proses penulisan berhasil:

    ```
    Insert card
    Init - OK
    Writing:
    FAT Table...
    Card Name...
    File Name...
    File Contents...
    File Size...
    Write complete
    Eject card
    ```

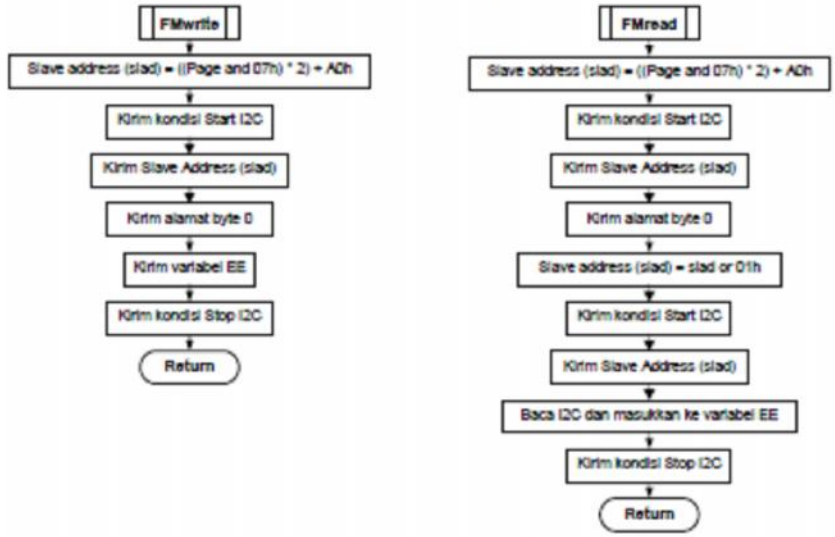
15. Setelah kartu dikeluarkan (CD=1), program kembali ke langkah nomor 2.

Setelah proses tulis selesai, maka SD card dapat dilepas lalu dibaca pada komputer menggunakan card reader atau pada PDA. Di dalamnya akan terdapat 1 file bernama TESTINGS.TXT sebesar 77 byte yang berisi: "0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz"

Pada program **Write w EEPROM – cvavr.prj**, data yang dibaca disimpan dalam variabel yang diletakkan pada EEPROM. Hal ini dikarenakan internal RAM Atmega8535 tidak mencukupi. Besarnya EEPROM sesuai dengan *buffer* yang diperlukan untuk akses SD card yaitu 512 byte sehingga proses menjadi mudah. Namun hal ini akan mengakibatkan usia pemakaian EEPROM menjadi cepat habis. **Oleh karena itu, penggunaan EEPROM sebagai *buffer* tidak dianjurkan dan program ini hanya digunakan sebagai contoh saja.**

Sedangkan pada program **Write w RAM – cvavr.prj**, data yang dibaca akan disimpan ke dalam variabel sementara dan akan langsung disimpan ke dalam FRAM. Internal RAM ATmega8535 yang digunakan hanya 256 byte yang disesuaikan dengan kapasitas 1 page FRAM. Oleh karena itu program membutuhkan 2 page FRAM saat mengakses SD card. Pada program ini, page FRAM yang digunakan untuk proses baca dan tulis dibedakan sehingga jumlah page yang digunakan adalah 4 page. Pada proses baca, 256 byte pertama diletakkan pada page 0 dan 256 byte kedua diletakkan pada page 1. Pada proses tulis, 256 byte pertama diletakkan pada page 2 dan 256 byte kedua diletakkan pada page 3.

Flowchart dari rutin program FMread dan FMwrite adalah sebagai berikut:



Gambar 5
***Flowchart* Rutin FMread dan FMwrite**

Cara kerja rutin FMwrite adalah sebagai berikut:

1. Pertama rutin akan menghitung alamat slave sesuai dengan parameter page (pager) yang diberikan saat memanggil rutin.
2. Lalu rutin akan mengirim kondisi Start I2C yang diikuti dengan alamat slave yang telah dihitung.
3. Rutin selalu menulis mulai dari alamat 0 untuk tiap page sehingga data yang dikirim berikutnya adalah 0.
4. Semua isi variabel EE sebanyak 256 byte dituliskan ke FRAM.
5. Setelah semua data terkirim, rutin mengakhiri dengan mengirim kondisi Stop I2C.

Cara kerja rutin FMread adalah sebagai berikut:

1. Pertama rutin akan menghitung alamat slave sesuai dengan parameter page (pager) yang diberikan saat memanggil rutin.
2. Lalu rutin akan mengirim kondisi Start I2C yang diikuti dengan alamat slave yang telah dihitung.
3. Rutin selalu membaca mulai dari alamat 0 untuk tiap page sehingga data yang dikirim berikutnya adalah 0.
4. Lalu kondisi Start I2C akan dikirim yang diikuti dengan alamat slave untuk proses baca (bit 0 = 1).
5. Data sebanyak 256 byte pada page FRAM tersebut akan dibaca dan disimpan ke dalam variabel EE.
6. Setelah semua data dibaca, rutin mengakhiri dengan mengirim kondisi Stop I2C.

Listing program terdapat pada *folder* CVAVR.

Selamat berinovasi!