**Estufa automatizada**

Victor Hugo Bezerra Tavares

FGA

Universidade de Brasília

Gama, Brasil

victorhugo.tavares@hotmail.com

Lucas Rocha F. S. e Barros

FGA

Universidade de Brasília

Gama, Brasil

lucas.oct8@gmaill.com

***Palavras-chave: MSP430, Hardware, Software.***

1. Introdução

É notória a importância da utilização das plantas pelo ser humano. Um exemplo disso é quando seu uso medicinal pode ser mais eficaz que de um antibiótico para o tratamento de doenças ou condições. Uma planta crescendo em ambiente controlado seria sujeita todo o tempo a temperatura, umidade e luminosidade perfeitos para seu desenvolvimento, e isso implicaria em uma planta mais robusta, sem contar o rápido amadurecimento. A ideia do projeto seria, então, criar uma estufa automatizada para poder controlar os fatores citados acima.

Para fazer o controle da estufa tem-se como ideia usar o microcontrolador msp430 que irá receber dados do sensores de temperatura e umidade e baseado nesses sinais será feito o controle de forma devida da estufa.

1. Desenvolvimento

A. Descrição de Hardware

Para a realização deste projeto foi utilizada a seguinte lista de materiais:

| ITEM | QUANTIDADE |
| --- | --- |
| MSP430G2553  launchpad | 1 |
| Sensor de Temperatura DS18B20 | 1 |
| Módulo Sensor Umidade do Solo  (LM393) | 1 |
| Display (LCD 16x02 - AM/PT) | 1 |
| Jumpers | - |
| Válvula Solenóide (Modelo à Designar) | 1 |
| Protoboard | 1 |
| Lâmpada incandescente | 1 |

O hardware consiste em uma estufa que contém um sensor de umidade, um sensor de temperatura, uma válvula solenóide e uma lâmpada incandescente.

O sensor de umidade do solo consiste em duas partes, uma sonda que entra em contato com o solo, e um pequeno módulo contendo um chip comparador LM393, que vai ler os dados que vêm do sensor e enviá-los para a MSP430. Como saída, temos um pino D0, que fica em nível 0 ou 1 dependendo da umidade, e um pino de saída analógica (A0), que possibilita monitorar com maior precisão usando uma porta analógica do microcontrolador. Dependendo do valor de umidade apresentando será acionada a válvula solenóide que irá irrigar a planta. O solenóide necessita de alimentação externa, pois a MSP430 não possui corrente necessária para o funcionamento do mesmo.

O sensor de temperatura usado é uma versão à prova de água do sensor DS18B20 que opera entre -55°C até +125°C e com precisão de ±0,5°C se estiver trabalhando dentro da faixa de -10°C até +85°C. O sensor irá monitorar a temperatura ambiente e de acordo com a faixa estabelecida irá controlar a intensidade luminosa incidente na planta com a lâmpada incandescente, que também irá necessitar de alimentação externa devido a limitação de corrente do microcontrolador.

Internamente o sensor DS18B20 possui um tipo de memória chamada scratchpad e uma mini EEPROM. O scratchpad possui 8 bytes, sendo que os bytes 0 e 1 armazenam a temperatura lida. Os bytes 2 e 3 são utilizado para definirmos um alarme quando determinada temperatura for atingida, máxima e mínima respectivamente. Já o byte 4 é registro de configuração(ou configuration register). É aqui que se define a resolução que queremos de nosso sensor. Os bytes 5, 6 e 7 são reservados, sem uso no momento. Ainda podemos considerar um nono byte (ou byte 8), que armazena o CRC.

Já a mini-EEPROM está diretamente relacionada aos bytes 2,3 e 4, armazenando o mesmo tipo de informação. A EEPROM é uma memória não volátil, ou seja, os valores nela armazenados não são apagados quando “desligamos” nosso sensor, e também já sabem que a EEPROM normalmente é uma memória lenta. O motivo da EEPROM espelhando os dados é muito simples: no scratchpad o sensor lê rapidamente os valores, como os limites do alarme e a resolução a utilizar, porém estas informações só valem enquanto o sensor está energizado. Sendo assim, o sensor lê a EEPROM na inicialização e copia os dados para o scratchpad.

As conexões da parte eletrônica do projeto foram feitas no fritzing e podem ser visualizadas no anexo 1.

A. Descrição de Software

Nesse ponto de controle desenvolvemos apenas os códigos que

possibilitam a leitura dos sensores de temperatura (DS18B20) e de

umidade (MDL + comparador LM393). No próximo ponto de controle

serão desenvolvidos os códigos para os acionadores (lâmpada

incandescente caso a temperatura esteja muito baixa e uma válvula

solenóide que irá irrigar a planta caso a umidade do solo esteja muito baixa).

Para o sensor de umidade não utilizamos nenhuma biblioteca específica,

os comandos do Energia já foram suficientes. Neste código (anexo x) é lido o valor de umidade *valor\_analogico=analogRead(pino\_sinal\_analogico);*

O software recebe os dados do sensor em valores de tensão e transforma em valores que variam de 0 a 1023, com 0 sendo 0 volts e 1023 sendo 5 volts. E em seguida é comparado esse valor ao número que pode ser usado como comparação, um parâmetro. Entre 0 e 400: solo úmido, entre 400 e 800: solo relativamente úmido e entre 800 e 1023: solo seco.

Se é possível definir a umidade relativa posteriormente não será

complicado tomar uma decisão a partir disso, como por exemplo acionar

a válvula.

Agora considerando o sensor de temperatura e tendo como base

seu datasheet é tido que a temperatura medida analogicamente pelo sensor

é convertida já por ele em um sinal digital de 9 a 12 bits. Isso já

teoricamente facilita um pouco pelo fato de não ser necessário

trabalhar com o conversor analógico-digital nessa parte. Porém,

para poder decodificar esse sinal é necessária a inclusão de uma

biblioteca própria para esse sensor (OneWire.h)que inclusive já

vem no Energia. Além do mais, com essa biblioteca fica muito menos

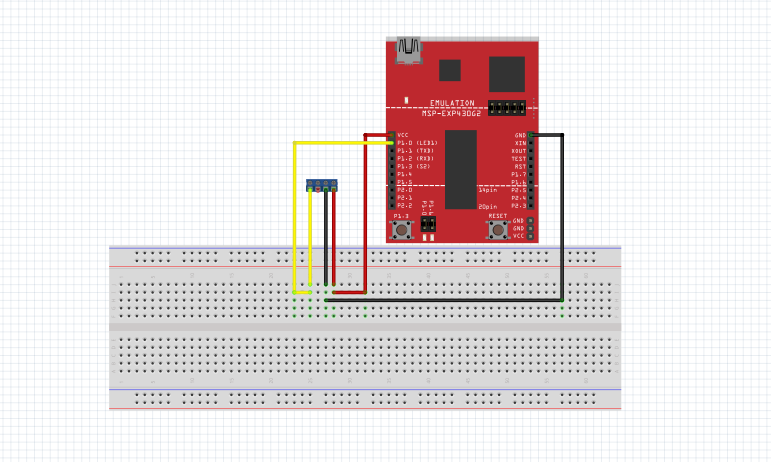
complexo o desenvolvimento do código ou alteração.

Posteriormente tentamos comunicar os sensores com o Msp430 por meio da IDE IAR WorkBench.

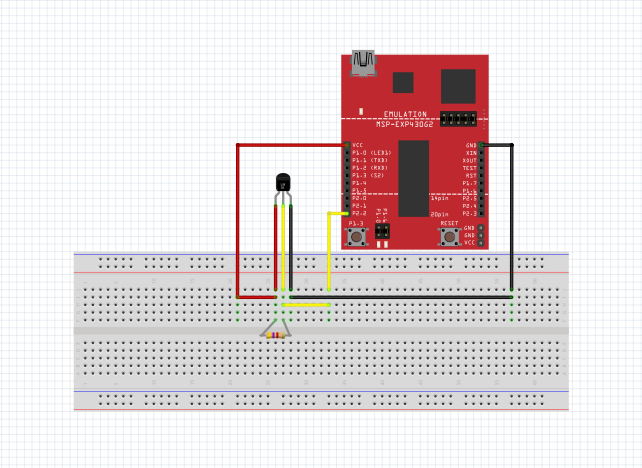
Inicialmente procuramos apenas a capacidade de poder sinalizar quando o solo estivesse úmido, o que foi conseguido com êxito (anexo 4).

Porém não conseguimos ler a temperatura com o sensor DS18B20 por não termos encontrado um protocolo para tal comunicação, apenas utilizando a biblioteca OneWire.h. Já iniciamos um código, que obviamente contém alguns erros mas que pretendemos desenvolver para o próximo Ponto de Controle (anexo 5).

Anexo 1:



*Imagem 1(Esquema de pinagem sensor de umidade)*



*Imagem 2(Esquema de pinagem sensor de temperatura)*

Anexo 2

int pino\_sinal\_analogico = A0;

int valor\_analogico = 0;

void setup()

{

Serial.begin(9600);

pinMode(pino\_sinal\_analogico, INPUT);

}

void loop()

{

//Le o valor do pino A0 do sensor

valor\_analogico = analogRead(pino\_sinal\_analogico);

//Mostra o valor da porta analogica no serial monitor

Serial.print("Porta analogica: ");

Serial.print(valor\_analogico);

//Solo umido

if (valor\_analogico > 0 && valor\_analogico < 400)

{

Serial.println(" Status: Solo umido");

}

//Solo com umidade moderada

if (valor\_analogico > 400 && valor\_analogico < 800)

{

Serial.println(" Status: Umidade moderada");

}

//Solo seco

if (valor\_analogico > 800 && valor\_analogico < 1023)

{

Serial.println(" Status: Solo seco");

}

delay(100);

}

Anexo 3

#include <OneWire.h>  
  
  
OneWire ds(10);   
  
void setup(void) {  
 Serial.begin(9600);  
}  
  
void loop(void) {  
 byte i;  
 byte present = 0;  
 byte type\_s;  
 byte data[12];  
 byte addr[8];  
 float celsius, fahrenheit;  
   
 if ( !ds.search(addr)) {  
 Serial.println("No more addresses.");  
 Serial.println();  
 ds.reset\_search();  
 delay(250);  
 return;  
 }  
   
 Serial.print("ROM =");  
 for( i = 0; i < 8; i++) {  
 Serial.write(' ');  
 Serial.print(addr[i], HEX);  
 }  
  
 if (OneWire::crc8(addr, 7) != addr[7]) {  
 Serial.println("CRC is not valid!");  
 return;  
 }  
 Serial.println();

switch (addr[0]) {  
 case 0x10:  
 Serial.println(" Chip = DS18S20");   
 type\_s = 1;  
 break;  
 case 0x28:  
 Serial.println(" Chip = DS18B20");  
 type\_s = 0;  
 break;  
 case 0x22:  
 Serial.println(" Chip = DS1822");  
 type\_s = 0;  
 break;  
 default:  
 Serial.println("Device is not a DS18x20 family device.");  
 return;  
 }   
  
 ds.reset();  
 ds.select(addr);  
 ds.write(0x44, 1);   
   
 delay(1000);   
   
 present = ds.reset();  
 ds.select(addr);   
 ds.write(0xBE);   
  
 Serial.print(" Data = ");  
 Serial.print(present, HEX);  
 Serial.print(" ");  
 for ( i = 0; i < 9; i++) {   
 data[i] = ds.read();  
 Serial.print(data[i], HEX);  
 Serial.print(" ");  
 }  
 Serial.print(" CRC=");  
 Serial.print(OneWire::crc8(data, 8), HEX);  
 Serial.println();  
  
 int16\_t raw = (data[1] << 8) | data[0];  
 if (type\_s) {  
 raw = raw << 3;   
 if (data[7] == 0x10) {  
   
 raw = (raw & 0xFFF0) + 12 - data[6];  
 }  
 } else {  
 byte cfg = (data[4] & 0x60);  
   
 if (cfg == 0x00) raw = raw & ~7;   
 else if (cfg == 0x20) raw = raw & ~3;  
 else if (cfg == 0x40) raw = raw & ~1; //  
   
 }  
 celsius = (float)raw / 16.0;  
 fahrenheit = celsius \* 1.8 + 32.0;  
 Serial.print(" Temperature = ");  
 Serial.print(celsius);  
 Serial.print(" Celsius, ");  
 Serial.print(fahrenheit);  
 Serial.println(" Fahrenheit");  
}

Anexo 4

void main(void)

{

{

int ValorRecebido;

WDTCTL = WDTPW + WDTHOLD;

ADC10CTL1 = INCH\_5 + ADC10DIV\_3 ;

ADC10CTL0 = SREF\_0 + ADC10SHT\_3 + ADC10ON;

ADC10AE0 |= BIT5;

P1SEL |= BIT5;

BCSCTL1 = CALBC1\_1MHZ;

DCOCTL = CALDCO\_1MHZ;

P1OUT = 0;

P1DIR |= BIT0;

while(1)

{

\_\_delay\_cycles(1000);

ADC10CTL0 |= ENC + ADC10SC;

ValorRecebido = ADC10MEM;

if(ValorRecebido <= 500){

P1OUT = BIT0;

}

else

P1OUT = 0;

}

}

}

Anexo 5

#include "msp430g2553.h"

#include "OneWire.h"

#define OW\_IN P1IN

#define OW\_OUT P1OUT

#define OW\_DIR P1DIR

#define OW\_PIN BIT0

unsigned char scratchpad[10],i;

//------------------------------------------------------------------------------

// delay em ms

//

void usdelay(int interval){

TACCTL0 = CCIE;

TACCR0 = TAR + interval\*8;

TA0CTL = TASSEL\_2 + MC\_2;

LPM0;

}

/\*

\* This function generates the reset pulse and check for 1-wire device presence. If found, it retuns 1, else return 0

\*/

int owReset(void){

int lineState;

OW\_OUT &= ~OW\_PIN; // p1.0 = 0

OW\_DIR |= OW\_PIN; // p1.0 como saida

usdelay(480);

OW\_DIR &= ~OW\_PIN;

usdelay(70);

lineState = OW\_IN & OW\_PIN;

usdelay(300);

if (lineState == 1){

return (1);

} else {

return (0);

}

}

/\*

\* READ\_BIT - reads a bit from the one-wire bus.

\*/

unsigned char owReadBit(void){

unsigned char lineState;

OW\_OUT &= ~OW\_PIN;

OW\_DIR |= OW\_PIN;

usdelay(6);

OW\_DIR &= ~OW\_PIN;

usdelay(10);

lineState = OW\_IN & OW\_PIN;

usdelay(55);

return (lineState);

}

/\*

\* WRITE\_BIT - writes a bit to the one-wire bus, passed in bitval.

\*/

void owWriteBit(char bitval)

{

OW\_OUT &= ~OW\_PIN;

OW\_DIR |= OW\_PIN;

usdelay(6);

if(bitval==1){

OW\_DIR &= ~OW\_PIN;

usdelay(64);

} else {

usdelay(50);

OW\_DIR &= ~OW\_PIN;

usdelay(10);

}

}

/\*

\* READ\_BYTE reads a byte from the one-wire bus.

\*/

unsigned char owReadByte(void){

unsigned char i;

unsigned char value = 0;

for (i=0;i<8;i++) {

if(owReadBit()){

value|=0x01<<i; //lê byte in, um bit de cada vez e depois desloca-o para a esquerda

}

}

return(value);

}

/\*

\* WRITE\_BYTE - writes a byte to the one-wire bus.

\*/

void owWriteByte(char val){

unsigned char i;

unsigned char temp;

for (i=0; i<8; i++) { // escrever 1 bit por vez

temp = val>>i;

temp &= 0x01;

owWriteBit(temp);

}

}

void main(void)

{

WDTCTL = WDTPW + WDTTMSEL;

BCSCTL1 = CALBC1\_8MHZ;

DCOCTL = CALDCO\_8MHZ;

\_\_enable\_interrupt();

owReset(); // Check device presence

owWriteByte(0xcc); // Skip ROM

owWriteByte(0x44); // Convert Temperature

usdelay(1000);

owReset(); // Check device presence

owWriteByte(0xcc); // Skip ROM

owWriteByte(0xbe); // ler scratchpad

for (i=0; i<8; i++) { // escrever um byte por vez

scratchpad[i]= owReadByte();

}

owReset(); // para de ler

LPM0;

}

#pragma vector=TIMER0\_A0\_VECTOR

\_\_interrupt void Timer\_A0 (void)

{

TA0CTL = 0; // parar TA

LPM0\_EXIT; // retornar

}

1. Referências

Hackster Io, Medidor de temperatura e pressão. Disponível em: <<https://www.hackster.io/55877/temperature-and-humidity-meter-iot-887cba>>

Instruction Tables, Estufa Automatizada. Disponível em: <<http://www.instructables.com/id/Automated-Greenhouse/>>

<<https://hackaday.io/project/2375-gardenautomationandsensornetwork#menu-description>>

<<http://ieeexplore.ieee.org/document/7967388/>>

<<https://github.com/fisherinnovation/FI-Automated-Greenhouse>>

<<https://www.linkedin.com/pulse/agriculture-projects-irrigation-based-8051-avr-msp430-prakash/>>

<<https://www.ripublication.com/ijtam17/ijtamv12n4_11.pdf>>

<<https://link.springer.com/article/10.1007/s11277-009-9881-2>>

<https://github.com/odd13/greenHouse>

<<https://www.pjrc.com/teensy/td_libs_OneWire.html>>

<<http://www.smallbulb.net/2012/238-1-wire-and-msp430>>

<<https://www.filipeflop.com/blog/monitore-sua-planta-usando-arduino/>>

<https://e2e.ti.com/support/microcontrollers/msp430/f/166/t/490177>