BEÁGYAZOTT RENDSZEREK (C)

7. forduló

BOSCH
Invented for life

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladatlaphoz

Az utolsó fordulókhoz érkeztünk, így megosztunk 1-2 fontos információt a továbbiakról:

- a versennyel kapcsolatos észrevételeket december 5-ig tudjátok velünk megosztani a szokásos helyen
- az utolsó fordulóhoz kapcsolódó megoldások november 30-án érhetők el
- a végeredményről tájékoztatás decemberben, részletek hamarosan

Sok sikert az utolsó fordulóhoz!

Mekk Mester szeretné a szélmérőjét akkumulátorról üzemeltetni, és úgy döntött, hogy csinál hozzá egy napelemes töltőt. Mivel az ESP32 ADC-je pontatlan, ezért az MCP3428 ADC-t szeretné használni.

Az MCP3428 ADC 4 csatornája elegendő az akkumulátor és a napelem feszültség- és árammméréseit elvégezni, amivel megvalósítható egy MPPT algoritmus.

Adatlapok:

MCP3428: https://ww1.microchip.com/downloads/en/DeviceDoc/22226a.pdf

A projekt az Arduino keretrendszeren alapul.

1. feladat 3 pont

Milyen célból alkalmazzák az MPPT (Maximum Power Point Tracking) algoritmust a napelemes rendszerekben?

Válaszok

- [ ] Az MPPT algoritmust arra használják, hogy optimalizálják a napelem hatékonyságát azzal, hogy folyamatosan követi és módosítja az elektromos terhelést annak érdekében, hogy a cella a maximális teljesítményen működjön, és a lehető legtöbb energiát vonja ki a napfényből.
- [ ] MPPT algoritmusokat azért alkalmaznak a napenergia rendszerekben, hogy alkalmazkodjanak a környezeti feltételek változásaihoz, például a napfény intenzitásának és hőmérsékletének változásaihoz, és ezzel maximalizálják az energia kibocsátást.
- [ ] Az MPPT algoritmusok fő célja, hogy állandó feszültséget és áramot tartsanak fenn a napcellákból, ami hatékony energia kinyerést eredményez.

2. feladat 4 pont

Mekk Mester szeretne írni egy saját Arduino drivert az MCP3428-hoz. Az I2C-hez az Arduino Wire library-t fogja használni. Melyik megoldás helyes a konfigurációs regiszter megfelelő kezeléséhez?

Válasz

- [ ]

```
#ifndef MCP3428_H__
#define MCP3428_H__

#include <stdint.h>
#include <string.h>

#include "Arduino.h"
#include "Wire.h"

class MCP3428 {
  public:
    enum ConversionType : uint8_t { ONE_SHOT = 0, CONTINUOUS };

    enum Channel : uint8_t { CH1 = 0, CH2, CH3, CH4 };

    enum Resolution : uint8_t { R12B = 0, R14B, R16B };

    enum Gain : uint8_t { GAINx1 = 0, GAINx2, GAINx4, GAINx8 };

    union ConfigRegister {
        uint8_t value;
        struct __attribute__((__packed__)) {
            uint8_t gain : 2;
            uint8_t resolution : 2;
            uint8_t conversionType : 1;
```

```
            uint8_t channel : 2;
            uint8_t ready : 1;
        };
    } __attribute__((__packed__));

public:
    MCP3428(TwoWire& wire, uint8_t address)
        : mWire(wire)
        , mAddress(address) {
        memset(&mChannelConfig, 0, sizeof(ConfigRegister));
        mChannelConfig[0].channel = CH1;
        mChannelConfig[1].channel = CH2;
        mChannelConfig[2].channel = CH3;
        mChannelConfig[3].channel = CH4;
    }

    float read(Channel channel);

    void setMode(Channel channel, ConversionType mode) {
        mChannelConfig[channel].conversionType = mode;
    }
    void setResolution(Channel channel, Resolution resolution) {
        mChannelConfig[channel].resolution = resolution;
    }
    void setGain(Channel channel, Gain gain) {
        mChannelConfig[channel].gain = gain;
    }

private:
    void writeConfigReg(Channel channel) {
        mWire.beginTransmission(mAddress);
        mWire.write(mChannelConfig[channel].value);
        mWire.endTransmission();
    }
    void startNewConversion(Channel channel) {
        mChannelConfig[channel].ready = 1;
        writeConfigReg(channel);
    }
    void read(Channel channel, int16_t* adc, ConfigRegister* configReg) {
        uint8_t hValue, lValue = 0;

        mWire.beginTransmission(mAddress);
        mWire.write(mChannelConfig[channel].value);
        mWire.endTransmission();
        mWire.requestFrom(mAddress, 3);
        hValue = mWire.read();
        lValue = mWire.read();
        configReg->value = mWire.read();
```

```
        *adc = ((uint16_t)(hValue) << 8) | lValue;
    }

  private:
    TwoWire& mWire;
    uint8_t mAddress;
    ConfigRegister mChannelConfig[4];
};

#endif // MCP3428_H__
```



```
#ifndef MCP3428_H__
#define MCP3428_H__

#include <stdint.h>
#include <string.h>

#include "Arduino.h"
#include "Wire.h"

class MCP3428 {
  public:
    enum ConversionType : uint8_t { ONE_SHOT = 0, CONTINUOUS };

    enum Channel : uint8_t { CH1 = 0, CH2, CH3, CH4 };

    enum Resolution : uint8_t { R12B = 0, R14B, R16B };

    enum Gain : uint8_t { GAINx1 = 0, GAINx2, GAINx4, GAINx8 };

    union ConfigRegister {
        uint8_t value;
        struct __attribute__((__packed__)) {
            uint8_t ready : 1;
            uint8_t channel : 2;
            uint8_t conversionType : 1;
            uint8_t resolution : 2;
            uint8_t gain : 2;
        };
    } __attribute__((__packed__));

  public:
    MCP3428(TwoWire& wire, uint8_t address)
        : mWire(wire)
        , mAddress(address) {
        memset(&mChannelConfig, 0, sizeof(ConfigRegister));
```

```
        mChannelConfig[0].channel = CH1;
        mChannelConfig[1].channel = CH2;
        mChannelConfig[2].channel = CH3;
        mChannelConfig[3].channel = CH4;
    }

    float read(Channel channel);

    void setMode(Channel channel, ConversionType mode) {
        mChannelConfig[channel].conversionType = mode;
    }
    void setResolution(Channel channel, Resolution resolution) {
        mChannelConfig[channel].resolution = resolution;
    }
    void setGain(Channel channel, Gain gain) {
        mChannelConfig[channel].gain = gain;
    }

  private:
    void writeConfigReg(Channel channel) {
        mWire.beginTransmission(mAddress);
        mWire.write(mChannelConfig[channel].value);
        mWire.endTransmission();
    }
    void startNewConversion(Channel channel) {
        mChannelConfig[channel].ready = 1;
        writeConfigReg(channel);
    }
    void read(Channel channel, int16_t* adc, ConfigRegister* configReg) {
        uint8_t hValue, lValue = 0;

        mWire.beginTransmission(mAddress);
        mWire.write(mChannelConfig[channel].value);
        mWire.endTransmission();
        mWire.requestFrom(mAddress, 3);
        hValue = mWire.read();
        lValue = mWire.read();
        configReg->value = mWire.read();
        *adc = ((uint16_t)(hValue) << 8) | lValue;
    }

  private:
    TwoWire& mWire;
    uint8_t mAddress;
    ConfigRegister mChannelConfig[4];
};
```

```
#endif // MCP3428_H__
```

3. feladat 6 pont

Ebben a feladatban az MCP3428 driver mért értékének a kiolvasását szeretnénk implementálni. Válaszd ki a helyes megoldást!

Válasz

```
#include "mcp3428_driver.h"

float MCP3428::read(Channel channel) {
    int16_t adcValue;
    ConfigRegister readedConfig;

    if (mChannelConfig[channel].conversionType == ONE_SHOT) {
        mChannelConfig[channel].ready = 1;
    } else {
        mChannelConfig[channel].ready = 0;
    }
    writeConfigReg(channel);

    do {
        read(channel, &adcValue, &readedConfig);
        if (readedConfig.ready == 1) {
            delay(1);
        }

    } while (readedConfig.ready == 1);

    if (mChannelConfig[channel].resolution == R12B) {
        adcValue = adcValue & 0x87FF;
    } else if (mChannelConfig[channel].resolution == R14B) {
        adcValue = adcValue & 0x9FFF;
    }

    return 0.001f * (float)(adcValue) / (1 << (mChannelConfig[channel].resolut
}
```

```
#include "mcp3428_driver.h"

float MCP3428::read(Channel channel) {
    int16_t adcValue;
    ConfigRegister readedConfig;

    if (mChannelConfig[channel].conversionType == ONE_SHOT) {
        mChannelConfig[channel].ready = 1;
    } else {
        mChannelConfig[channel].ready = 0;
    }
    writeConfigReg(channel);

    do {
        read(channel, &adcValue, &readedConfig);
        if (readedConfig.ready == 1) {
            delay(1);
        }

    } while (readedConfig.ready == 1);

    return 0.001f * (float)(adcValue) / (1 << (mChannelConfig[channel].resolut
}
```

```
#include "mcp3428_driver.h"

float MCP3428::read(Channel channel) {
    int16_t adcValue;
    ConfigRegister readedConfig;

    if (mChannelConfig[channel].conversionType == ONE_SHOT) {
        mChannelConfig[channel].ready = 1;
    } else {
        mChannelConfig[channel].ready = 0;
    }
    writeConfigReg(channel);

    do {
        read(channel, &adcValue, &readedConfig);
        if (readedConfig.ready == 1) {
            delay(1);
        }

    } while (readedConfig.ready == 1);
```

```
    if (mChannelConfig[channel].resolution == R12B) {
        adcValue = adcValue & 0x0FFF;
    } else if (mChannelConfig[channel].resolution == R14B) {
        adcValue = adcValue & 0x3FFF;
    }

    return 0.001f * (float)(adcValue) / (1 << (mChannelConfig[channel].resolut
}
```

4. feladat 10 pont

Mekk Mester szeretne írni a napelemes töltőjéhez egy MPPT algoritmust. Ehhez a P&O (perturb and observe) módszert választotta, azonban az algoritmusba valahol egy hiba csúszott be. Melyik sorban van a hiba?

```
void mpptTask() {
    State state = State::Off;
    float solarVoltage = 0.0f;
    float batteryVoltage = 0.0f;
    float solarCurrent = 0.0f;
    float currentSolarPower = 0.0f;
    float prevSolarPower = 0.0f;
    float dutyCycle = 0.0f;
    bool increase = false;

    while (true) {
        solarVoltage = getSolarVoltage();
        solarCurrent = getSolarCurrent();
        batteryVoltage = getBatteryVoltage();
        switch (state) {
            case State::Start:
                if (batteryVoltage > BATTERY_VOLTAGE_MAXIMUM || solarVoltage < SOLAR_PANEL_MIN_VOLTAGE) {
                    break;
                }
                dutyCycle = (batteryVoltage / solarVoltage * 100);
                mPwm.setDutyCycle(dutyCycle);

                solarVoltage = getSolarVoltage();
                solarCurrent = getSolarCurrent();
                prevSolarPower = solarVoltage * solarCurrent;
                increase = true;
                state = State::Operate;
                break;
            case State::Operate:
                if ((solarCurrent < INPUT_CURRENT_MINIMUM) || (solarVoltage < batteryVoltage)) {
                    dutyCycle = 0.0f;
                    mPwm.setDutyCycle(dutyCycle);
                    state = State::Off;
                } else {
                    currentSolarPower = solarVoltage * solarCurrent;

                    if (currentSolarPower > prevSolarPower) {
                        increase = !increase;
                    }
                    if (increase) {
                        dutyCycle += 0.5;
                    } else {
                        dutyCycle -= 0.5;
                    }
                    mPwm.setDutyCycle(dutyCycle);

                    prevSolarPower = currentSolarPower;
                }
                break;
            case State::Off:
                if (solarVoltage > batteryVoltage) {
                    state = State::Start;
                }
                break;

            default:
                break;
        }
        delay(50);
    };
}
```

Válasz

Megoldások beküldése