Controlul pinilor GPIO ai unei plăci de dezvoltare Raspberry Pi dintr-o interfaţă web

Autori: Laurenţiu-Adrian MOLNAR
Robert MISARĂŞ

Coordonator: asist. dr. ing. Sergiu NIMARĂ

2017-2018

Caracteristici:

- Interfața web va conține o opțiune de selecție a ieșirilor / intrărilor (GPIO) active de pe placa de dezvoltare. Pinii GPIO nefolosiți vor fi powered-down.
- Interfața web va permite selecția modului de utilizare a fiecărui pin activ al plăcii de dezvoltare: on / off (0 sau 1 logic) sau posibilitate de generare semnal PWM;
- Se va implementa o metodă securizată pentru logarea în interfața web, de exemplu cu Google VPN Authenticator;
- Starea setărilor curente dorite de utilizator va fi salvată fie în memoria EEPROM a microcontrolerului, fie pe un card SD, astfel încât, la întreruperea alimentării, sistemul să repornească din ultima stare stabilă cunoscută.

Descrierea plăcii - Raspberry Pi Zero W

Raspberry Pi Zero W extinde familia Pi Zero. Câteva caracteristici:

- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Precum Pi Zero, caracteristicile hardware sunt:

- SoC BCM2835
- 1GHz, single-core ARM11 CPU
- 512MB RAM
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector

BCM2835 este SoC care prin intermediul căruia sunt realizate funcţionalităţile plăcii de dezvoltare. Acesta conţine următoarele echipamente ce pot fi accesate de microprocesorul ARM:

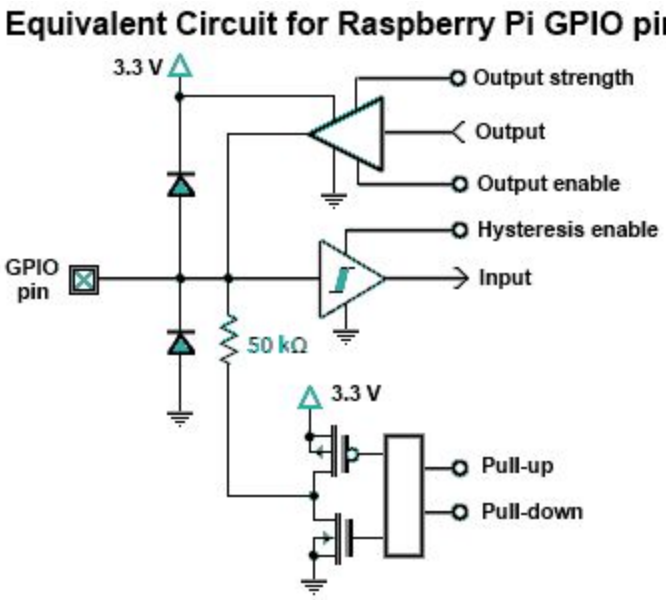
- Timere
- Interrupt controller
- GPIO - utilizați de aplicația prezentată
- USB

- PCM / I2S
- DMA controller
- I2C master
- I2C / SPI slave
- SPI0, SPI1, SPI2
- PWM
- UART0, UART1

TODO: scris despre GPIO şi modulul Wi-Fi, adăugat link la datasheet pentru BCM???, bibliografie

Descrierea modulelor utilizate:

Raspberry Pi Zero W pune la dispoziţie 40 de pini GPIO (General Purpose Input Output), distribuiţi în două rânduri a câte 20. Dintre aceştia, 2 pini sunt de alimentare cu 5V, unul este de alimentare cu 3.3V, 8 sunt pini de GND, iar restul sunt pini configurabili. Majoritatea pinilor configurabili au şi roluri secundare (de exemplu, pini de SPI sau I2C).



Schema bloc a unui pin GPIO pe Raspberry Pi

Mai multe detalii legate de BCM2835 se pot găsi în datasheet-ul circuitului. De asemenea, în datasheet se poate găsi funcţionarea regiştrilor care stau la baza modulului GPIO.

Codul folosit:

index.js

```
const express = require('express');
const app = express();

const rpio = require('rpio');

// An array containing pin descriptions. It is used on the frontend
for styling purposes

const pinout = ["", "3v3", "5v", "gpio2", "5v",
"gpio3","gnd","gpio4","gpio14","gnd","gpio15","gpio17",
"gpio18","gpio27","gnd","gpio22","gpio23","3v3","gpio24","gpio10","gn
d","gpio9","gpio25","gpio11","gpio8","gnd","gpio7","gpio0","gpio1","g
pio5","gnd",
"gpio6","gpio12","gpio13","gnd","gpio19","gpio16","gpio26","gpio20","
gnd","gpio21"]
  .map(el => el.toUpperCase());

  // console.log(pinout.length);

app.use(require("nocache")());

// The app uses Pug as its view engine. Pug code is compiled
internally into HTML code before being served over HTTP
app.set('view engine', 'pug');

// GET endpoint on /; It renders the Raspberry PI Zero W pinout along
with the GPIO configuration controls

app.get("/", (req, res) => {

  res.render("index", {title: "Remote GPIO control", pinout: pinout})
});

// GET endpoint on /gpio; The client sends a GET request whenever
```

```
they click a button in the app. The URL contains information
// about the pin to be configured

app.get("/gpio", (req, res) => {
  console.log(req.query);

  let error;

  // Configure the required pin accordingly

  switch(req.query.dir) {
    case 'in':
      rpio.open(req.query.pin, rpio.INPUT);
      error = false;
      break;
    case 'out':
      rpio.open(req.query.pin, rpio.OUTPUT, (req.query.value ==
'high') ? rpio.HIGH : rpio.LOW);
      error = false;
      break;
    default:
      console.log('There was an error. Try again!');
      error = true;
      break;
  }

  // Send a JSON response about the pin for debugging purposes on the
client side

  res.json({
    pin: req.query.pin,
    direction: req.query.dir,
    value: (req.query.dir == 'out') ? req.query.value : 'null',
    error: error
  });

});
```

```
// Serve static files from /static endpoint
app.use("/static", express.static("static"));

// The app will be hosted on port 8000 of localhost and exposed via
localtunnel
app.listen(8000, () => console.log(`Server running on port 8000`));
```

/static/js/app.js

```
document.querySelectorAll('input[type="submit"]')
  .forEach((el) => el.addEventListener('click', (evt) => {
    evt.preventDefault();

    /*

      Once a button was clicked, we gather the information about the pin
configuration the user is about to make.
      The information includes the direction of the pin (input/output), the
pin number and the value if the pin is configured
      as an output (High/Low)

    */

    let pin = evt.target.id.split('_')[1]; // Get the pin number from the
button id

    console.log(`Pin ${pin} was changed`);

    let url = "/gpio?";

    let dir = document.querySelector(`input[id$=_${pin}]:checked`);

    let value;

    if(dir == null) {
      alert("Choose in or out");
      return;
    } else if (dir.value == 'out') value =
document.querySelector('[name=voltage]:checked').value;

    // Build the final url according to the configuration
```

```
    url += `pin=${pin}&dir=${dir.value}`;

    if(dir.value == 'out') url += `&value=${value}`;

    console.log(url);

    if(dir) {

      // Send a GET request to the URL we made earlier. The URL tells the
backend how the pin should be configured

      fetch(url)
      .then((response) => response.json().then(data => console.log(data)))
      .catch((err) => console.error(err))
      ;
    }

  }));

// The voltage controller is initially hidden. Clicking an out label will
make it visible to choose a value for the pin. Default is LOW

document.querySelectorAll('label[for^=out]').
  forEach((el) => el.addEventListener('click', (evt) => {
    document.querySelector('.voltage-controller').style.display = "block";
  }));

// Configuring a pin as an input will hide the voltage controller because
you cannot set the value of an input
document.querySelectorAll('label[for^=in]').
  forEach((el) => el.addEventListener('click', (evt) => {
    document.querySelector('.voltage-controller').style.display = "none";
  }));


// Clicking the RESET button on the voltage controller will reset the
voltage option to LOW
document.querySelector('.voltage-controller > a').addEventListener('click',
(evt) => {
  evt.preventDefault();
  document.querySelector('[value=low]').checked = true;
```

```
})
```

Restul codului reprezintă logica de prezentare, adică fişierul **`index.pug`**, care conţine layout-ul paginii şi **`/static/css/app.css`**, care este utilizat pentru a stiliza pagina aplicaţiei.

Fişierul index.js conţine logica de rutare conform URL-ului solicitat de către client şi logica de configurare a pinilor plăcii de dezvoltare.

Fişierul app.js conţine logica pentru client, adică funcţiile care se execută atunci când utilizatorul apasă butoanele de configurare a pinilor. La o apăsare de buton, aplicaţia trimite o cerere de tip `GET` în fundal către server pentru a comanda un anumit pin. URL-ul este de forma `/gpio?pin=[1...40]&dir=[in/out][&value=some_voltage]`
. Astfel, server-ul va interpreta parametrii de căutare (query string) de după „?” şi va configura pinul în modul dorit de utilizator.

Bibliografie:

https://www.npmjs.com/package/rpio
https://pythonhosted.org/RPIO/
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf
https://expressjs.com/
https://localtunnel.github.io/www/
https://www.ostechnix.com/localtunnel-make-local-server-accessible-online/
https://www.raspberrypi.org/products/raspberry-pi-zero-w/
https://www.youtube.com/