BEÁGYAZOTT RENDSZEREK (C)

6. forduló

BOSCH

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladathoz

Most, hogy a perifériákat már jól tudja kezelni az STM32 vezérlővel, Gipsz Szabolcs nekikezd a fő számítógépre háruló magasabb szintű vezérlés kidolgozásának. Bár sokat vacillált a Raspberry Pi és az Nvidia Jetson Nano között, végül úgy találta, hogy a Raspberry Pi 4 teljesítménye már bőven elegendő lesz a célra, az ára pedig jóval kedvezőbb. Először a kapcsolatot szeretné megteremteni a két vezérlő között.

Segíts Szabolcsnak az Rpi és az STM közötti kapcsolat kialakításában!

A megoldásban a következő adatlapok lesznek a segítségedre:

STM32F103RB adatlap: https://www.st.com/resource/en/datasheet/stm32f103rb.pdf

NUCLEO-F103RB adatlap: https://www.st.com/resource/en/data_brief/nucleo-f103rb.pdf

Raspberry Pi 4 adatlap: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

BCM2711 adatlap: https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf

HAL API dokumentáció: https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf

Felhasznált idő: 20:00/20:00 **Elért pontszám:** 0/22

1. feladat 0/5 pont

Milyen kommunikációs kapcsolatok támogatottak plusz hardverelemek (a kábelt nem számítva 😉) használata nélkül a Raspberry Pi és az STM32 Nucleo között? **Válaszd ki az összes lehetségeset!**

Válaszok

- [ ] RS-485
- [x] GPIO
  Ez a válasz helyes, de nem jelölted meg.
- [x] UART
  Ez a válasz helyes, és meg is jelölted.
- [x] SPI
  Ez a válasz helyes, de nem jelölted meg.
- [x] I2C

Ez a válasz helyes, de nem jelölted meg.

- [x] CAN
  Ez a válasz helytelen, de megjelölted.
- [x] USB Serial
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Profibus
- [x] Ethernet
  Ez a válasz helytelen, de megjelölted.

Magyarázat

A GPIO, UART, SPI és I2C interfészek mindkét eszközön megtalálhatók és az egyező jelszintnek köszönhetően közvetlenül össze is köthetők. Az I2C becsapós, mert mind az SDA mind az SCL vonalon felhúzóellenállásra is szükség van, az STM belső felhúzóellenállásai viszont túl nagyok (40k körül). Amiért mégis a helyes válaszok közé tartozik, az a Raspberry Pi-ken gyárilag megtalálható 1k8 értékű felhúzóellenállások ezen a két vonalon. A Nucleo kártya USB csatlakozón keresztül is közvetlenül csatlakoztatható a Raspberryhez, így az STLink Virtual COM Porton keresztül is megvalósítható a kommunikáció a két kártya között.

Az RS-485, az Ethernet és a CAN buszos kapcsolathoz transciever komponensre van szükség, a Profibus pedig RS-485-öt használ fizikai rétegként.

2. feladat 0/2 pont

Mivel pont-pont összeköttetésre van szükség és nem tervez más vezérlőt a Raspberryhez kapcsolni, Szabolcs az SPI interfész mellett dönt.

Mik az előnyei az SPI-nak a többi interfészhez képest? Jelöld meg az IGAZ állításokat!

Válaszok

- [x] Az SPI nagyobb átviteli sebességet tesz lehetővé, mint az I2C és az UART
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Az SPI interfészen többféle protokoll implementálható, mint GPIO-n
- [x] Az SPI kapcsolathoz kevesebb vezetékre van szükség, mint UART esetén
  Ez a válasz helytelen, de megjelölted.
- [ ] Egy SPI buszra több eszközt lehet felfűzni, mint I2C-re
- [x] Az SPI kapcsolathoz kevesebb külső hardverelemre van szükség, mint CAN busz esetén
  Ez a válasz helyes, és meg is jelölted.

Magyarázat

I2C interfészen jellemzően párszáz kilobites, de maximum pár megabites sebességet érhetünk el. UART esetén szintén néhány megabites maximum sebességet érhetünk el. SPI esetén ez több tíz megabit is lehet.

GPIO-n „bit bang-olással” bármilyen vezetékes protokollt implementálhatunk, így ez a válasz hamis.

SPI esetén minimum GND, CLK, MISO/MOSI vonalakra szükség van, ahogy UART-nál is GND, RX/TX-re. Tehát egy- és kétirányú kapcsolatnál is az UART-hoz van szükség kevesebb vonalra.

SPI buszra praktikusan néhány eszközt szokás kötni a slave select (SS) vonalak miatt, de nincs kőbe vésett határ. I2C buszra 7 bites címzés mellett maximum 127, 10 bites címzés mellett 1023 eszközt köthetünk, de a vonalkapacitás is korlátozza az eszközök számát, igaz, ezt extenderekkel/repeaterekkel, multiplexerekkel át lehet lépni, tehát szintén nincs elvi maximum. Elméletben tehát egyik busznál sincsen határ, gyakorlatban egy SPI buszon belül az eszközök száma jóval kisebb.

SPI kapcsolathoz nincs szükség külső hardverelemekre, míg CAN buszra csatlakozáshoz igen.

3. feladat 0/15 pont

Szabolcs tervei szerint SPI kapcsolaton keresztül fog kommunikálni a Raspberry az STM-mel. Erre a célra az STM-en az SPI2 perifériát, a Raspberryn pedig a CE0 slave select lábhoz társított /dev/spidev0.0 eszközt fogja használni.

Megjegyzés: a Raspberry normál működéshez az AUX_SPIx_CNTL0_REG 6. bitjét reset után 1-be állítja.

Válaszd ki a helyesen együttműködő kódpárt!

Válasz

- 1.válasz:

```
//
// Source code for Raspberry
//
#include <iostream>
#include <wiringPiSPI.h>
#define SPI_CHANNEL 0
#define SPI_CLOCK_SPEED 1000000
#define SPI_MODE 0

void fetchCommandFromUpstream(unsigned char *buf);
void processResponseFromDownstream(unsigned char *buf);

int main(int argc, char **argv)
{
    int fd = wiringPiSPISetupMode(SPI_CHANNEL, SPI_CLOCK_SPEED, SPI_MODE);
    if (fd == -1) {
        std::cout << "Failed to init SPI communication.\n";
        return -1;
    }
    std::cout << "SPI communication successfully setup.\n";

    unsigned char buf[255] = { 0 };
    while(1) {
        fetchCommandFromUpstream(buf);
        std::cout << "Rpi -> STM: " << buf << "\n";
        wiringPiSPIDataRW(SPI_CHANNEL, buf, sizeof(buf));
        std::cout << "Rpi <- STM: " << buf << "\n";
        processResponseFromDownstream(buf);
    }
    return 0;
}

//
// Source code for STM32"
//
#include "main.h"

#define SPI_TIMEOUT_VALUE 5000

SPI_HandleTypeDef hspi2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_SPI2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI2_Init();

  uint8_t rx_data[255] = {0};
  uint8_t tx_data[255] = {0};

  while (1) {
      HAL_StatusTypeDef hal_status;
      hal_status = HAL_SPI_Receive(&hspi2, rx_data, sizeof(rx_data), SPI_TIMEOUT_VALUE);
      if(hal_status == HAL_OK) {
          processRequest(rx_data, tx_data);
          HAL_SPI_Transmit(&hspi2, tx_data, sizeof(tx_data), SPI_TIMEOUT_VALUE);
      }
  }
}

static void MX_SPI2_Init(void) {
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
```

```
  {
    Error_Handler();
  }
}
```

2.válasz:

```cpp
//
// Source code for Raspberry
//
#include <iostream>
#include <wiringPiSPI.h>
#define SPI_CHANNEL 0
#define SPI_CLOCK_SPEED 500000
#define SPI_MODE 1

void fetchCommandFromUpstream(unsigned char *buf);
void processResponseFromDownstream(unsigned char *buf);

int main(int argc, char **argv)
{
    int fd = wiringPiSPISetupMode(SPI_CHANNEL, SPI_CLOCK_SPEED, SPI_MODE);
    if (fd == -1) {
        std::cout << "Failed to init SPI communication.\n";
        return -1;
    }
    std::cout << "SPI communication successfully setup.\n";

    unsigned char buf[255] = { 0 };
    while(1) {
        fetchCommandFromUpstream(buf);
        std::cout << "Rpi -> STM: " << buf << "\n";
        wiringPiSPIDataRW(SPI_CHANNEL, buf, sizeof(buf));
        std::cout << "Rpi <- STM: " << buf << "\n";
        processResponseFromDownstream(buf);
    }
    return 0;
}
```

```cpp
//
// Source code for STM32"
//
#include "main.h"

#define SPI_TIMEOUT_VALUE 5000

SPI_HandleTypeDef hspi2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_SPI2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI2_Init();

  uint8_t rx_data[255] = {0};
  uint8_t tx_data[255] = {0};

  while (1) {
      HAL_StatusTypeDef hal_status;
      hal_status = HAL_SPI_Receive(&hspi2, rx_data, sizeof(rx_data), SPI_TIMEOUT_VALUE);
      if(hal_status == HAL_OK) {
          processRequest(rx_data, tx_data);
          HAL_SPI_Transmit(&hspi2, tx_data, sizeof(tx_data), SPI_TIMEOUT_VALUE);
      }
  }
}

static void MX_SPI2_Init(void) {
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_SLAVE;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
    Error_Handler();
  }
}
```

Ez a válasz helytelen, de megjelölted.

3.válasz:

```cpp
//
// Source code for Raspberry
//
#include <iostream>
#include <wiringPiSPI.h>
#define SPI_CHANNEL 0
#define SPI_CLOCK_SPEED 1000000
#define SPI_MODE 0

void fetchCommandFromUpstream(unsigned char *buf);
void processResponseFromDownstream(unsigned char *buf);

int main(int argc, char **argv)
{
    int fd = wiringPiSPISetupMode(SPI_CHANNEL, SPI_CLOCK_SPEED, SPI_MODE);
    if (fd == -1) {
        std::cout << "Failed to init SPI communication.\n";
        return -1;
    }
    std::cout << "SPI communication successfully setup.\n";

    unsigned char buf[255] = { 0 };
    while(1) {
        fetchCommandFromUpstream(buf);
        std::cout << "Rpi -> STM: " << buf << "\n";
        wiringPiSPIDataRW(SPI_CHANNEL, buf, sizeof(buf));
        std::cout << "Rpi <- STM: " << buf << "\n";
        processResponseFromDownstream(buf);
    }
    return 0;
}

//
// Source code for STM32"
//
#include "main.h"

#define SPI_TIMEOUT_VALUE 5000

SPI_HandleTypeDef hspi2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_SPI2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI2_Init();

  uint8_t rx_data[255] = {0};
  uint8_t tx_data[255] = {0};

  while (1) {
      HAL_StatusTypeDef hal_status;
      hal_status = HAL_SPI_Receive(&hspi2, rx_data, sizeof(rx_data), SPI_TIMEOUT_VALUE);
      if(hal_status == HAL_OK) {
          processRequest(rx_data, tx_data);
          HAL_SPI_Transmit(&hspi2, tx_data, sizeof(tx_data), SPI_TIMEOUT_VALUE);
      }
  }
}

static void MX_SPI2_Init(void) {
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_SLAVE;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Ez a válasz helyes, de nem jelölted meg.

4.válasz:

```
//
// Source code for Raspberry
//
#include <iostream>
#include <wiringPiSPI.h>
#define SPI_CHANNEL 0
#define SPI_CLOCK_SPEED 1000000
#define SPI_MODE 1

void fetchCommandFromUpstream(unsigned char *buf);
void processResponseFromDownstream(unsigned char *buf);

int main(int argc, char **argv)
{
    int fd = wiringPiSPISetupMode(SPI_CHANNEL, SPI_CLOCK_SPEED, SPI_MODE);
    if (fd == -1) {
        std::cout << "Failed to init SPI communication.\n";
        return -1;
    }
    std::cout << "SPI communication successfully setup.\n";

    unsigned char buf[255] = { 0 };
    while(1) {
        fetchCommandFromUpstream(buf);
        std::cout << "Rpi -> STM: " << buf << "\n";
        wiringPiSPIDataRW(SPI_CHANNEL, buf, sizeof(buf));
        std::cout << "Rpi <- STM: " << buf << "\n";
        processResponseFromDownstream(buf);
    }
    return 0;
}

//
// Source code for STM32"
//
#include "main.h"

#define SPI_TIMEOUT_VALUE 5000

SPI_HandleTypeDef hspi2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_SPI2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI2_Init();

  uint8_t rx_data[255] = {0};
  uint8_t tx_data[255] = {0};

  while (1) {
      HAL_StatusTypeDef hal_status;
      hal_status = HAL_SPI_Receive(&hspi2, &rx_data, sizeof(rx_data), SPI_TIMEOUT_VALUE);
      if(hal_status == HAL_OK) {
          processRequest(rx_data, &tx_data);
          HAL_SPI_Transmit(&hspi2, tx_data, sizeof(tx_data), SPI_TIMEOUT_VALUE);
      }
  }
}

static void MX_SPI2_Init(void) {
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_SLAVE;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_LSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  }
}
```

Magyarázat

Az első megoldásban a Raspberryn és az STM-en is SPI master perifériát inicializáltunk, ami így nem fog működni.

A második megoldásban az STM-en a Clock Polarity (CPOL) Low (0), a Clock Phase (CPHA) pedig az első élre van beállítva, ami SPI mode 0-nak felel meg. A Raspberryn futó kódban viszont SPI mode 1-et állítottunk be.

A harmadik a jó megoldás.

A negyedik megoldásban az STM-en az SPI Clock Polarity Low, a Clock Phase a 2. (lefutó) élre van beállítva, ami SPI mode 1-nek felel meg. A megoldáshoz tartozó Raspberrys kódban is SPI mode 1 van kiválasztva, így ez jó, viszont a bitsorrend az STM-en LSB first, míg a Raspberryn a default (és egyetlen opció) MSB first, ami miatt ez a megoldás hibás. A bufferek explicit cím szerinti átadása csak optikai tuning, hiszen a tömbök átadása implicit is cím szerint történik.