Protocoale sincrone de comunicatie seriala

Teorie

Pentru a reduce numarul de sarme prin care se comunica intre diferitele componente ale unui circuit electric, se poate apela la comunicatia seriala. Practic, se reduce complexitatea montajului, pretul platit fiind viteza ceva mai scazuta de transfer de date.

In laboratoarele anterioare am folosit comunicatia seriala de tip RS-232. Acest protocol este unul de tip asincron – ceasul de sincronizare date este regenerat la receptor. Avantajul major este ca obligatorie e o singura sarma – cea de date (a nu se uita si sarma de GND, punctul de referinta in orice montaj electric). Dezavantajul este ca atat emitatorul cat si receptorul trebuie sa aiba un oscilator foarte stabil. Pretul platit pe oscilator se poate justifica doar daca acest protocol este utilizat intre echipamente aflate de o distanta medie (metru de cablu de cupru s-ar putea sa fie mai scump decat cristalul de quartz)

Pentru comunicatia intre componentele aflate pe aceasi placa sau chiar in acelasi echipament, se poate utiliza un protocol sincron. Acest protocol are nevoie de o sarma auxiliara: semnalul de ceas. Un avantaj major este ca aceste protocoale se pot utiliza intr-o configuratie one_master - multi_slave sau chiar multi_master – multi_slave. Practic, comunicatia are loc pe o magistrala, magistrala care uneste circuitul master de circuitele slave, toata activitatea desfasurandu-se sub controlul masterului.

Mai multe firme au incercat sa isi impuna standardul pe piata. In acest moment exista 4 standarde, fiecare cu avantaje si dezavantaje.

I²C – Inovatia PHILIPS – **I**C **2** IC **C**omunication - sau IIC
Acest protocol inventat si dezvoltat de Philips a aparul prima oara pe placile televizoarelor. Este nevoie doar de doua sarme: SCL (ceasul) si SDA(data). Configuratiile posibile sunt multi_master – multi slave, existand mecanism de arbitrare intre masteri.
Avantaje:

- configuratie multi master – posibilitatea arbitrarii masterilor
- clock-stretching – perifericele lente pot “cere” un delay de la master
- utilizarea intensiva a bit-ului ACK (acknowledge)
    - detectia foarte simpla a perifericelor active (in functie de configuratia placii, avand un soft unic, se poate detecta la start-up care sunt perifericele prezente pe magistrala)
    - validarea rapida a datelor
- doar doua sarme pentru toata magistrala

Dezavantaje:

- viteza mica de transfer
- selectia perifericuli (prin adresa de slave) adauga overhead la comunicatie

O varianta mai flexibila (la nivelul legatura de date, daca privim IIC prin prizma stivei OSI) a acestui protocol este **T**wo**W**ire**I**nterface. Microcontrolerul folosit la laborator este capabil sa utilizeze acest protocol fie din postura de master fie din cea de slave.

SPI – Inovatie Motorola – **S**erial **P**eriferical **I**nterface
Motorola a dezvoltat aceast protocol si l-a impus prin intermediu microcontrolerelor HC11. Este un protocol foarte rapid, care insa nu permite direct prezenta pe magistrala a doi sau mai multi masteri. Schema de arbitrare trebuie efectuata extern, folosind un circuit auxiliar. In plus, perifericele mai lente nu pot semnaliza o cerere de amanare/intarziere a comunicatie. Fata de IIC, mai apare o sarma. Daca la IIC, SDA este sarma de date si este bidirectionala, la SPI apar doua sarme, **M**aster**O**utput**S**lave**I**nput si **M**aster**I**nput**S**lave**O**utput. MOSI si MISO sunt unidirectionale. Microcontrolerul folosit la laborator este capabil sa utilizeze acest protocol (implementare hardware).
Avantaje:

- viteza foarte mare
- selectia directa a slave-lui (dar, pentru fiecare slave, este nevoie de o noua sarma)
- simplitate in utilizare (la nivelul legatura de date OSI)

Dezavantaje:

- chiar cu un singur slave, este nevoie de 4 sarme iar cu 2 slave, este nevoie de 5 sarme.
- lipsa arbitrarii intre masteri
- lipsa clock-stretching-ului – perifericele lente nu au ce cauta pe aceasta magistrala
- lipsa unui bit de ACK – validarea pachetelor trebuie facuta la un nivel superior

Ca un contracandidat, National Semiconductor a lansat **uWire**, un protocol foarte asemanator cu SPI-ul, dar care vine cu o imbunatatire. Pe o magistrala uWire, un periferic lent poate semnaliza cererea pentru delay – echivalentul lui clock-stretching de la IIC.

1Wire – Inovatia Maxim – sau Button Wire interface
Maxim a vrut sa simplifice si mai mult viata designerilor de PCB-uri (Printed Circuit Board), si a pus la dispozitie acest protocol. Avantajul este unul singur: este nevoie de o singura sarma intre slave si master. Este posibila configuratia multi_slave, dar multi_master se poate realiza numai cu o schema de arbitrare externa. Este un protocol greu de implementat. (se poate considera ca fiind un protocol asincron)
Avantaje:

- un singur fir ! (slave-ul se poate chiar alimenta din acest fir !)

Dezavantaje:

- viteza mica de transfer
- greu de implementat
- patent Maxim – putine periferice suporta acest protocol

I²C – Detaliere

Pentru simplificare, vom considera cea mai simpla arhitectura: un singur master si un singur slave.

Din punctul de vedere OSI, nivelul fizic se defineste prin:

- doar doua sarme:
  - **SCL** – ceasul sincron generat de master
  - **SDA** – linia de date – bidirectionala
- nivelele sunt TTL, respectiv
  - VIL ('0' Logic) avem intre 0 si 0.8V
  - VIH ('1' Logic) avem intre 2V si 5V (5V este chiar VCC-ul
- driverele pinilor sunt:
  - SDA = open-drain/input si SCL = open-drain (la master = microcontroler)
  - SDA = open-drain/input si SCL = input (la slave)
- liniile din magistrala (SDA si SCL) sunt prevazute cu cate o rezistenta de pull-up

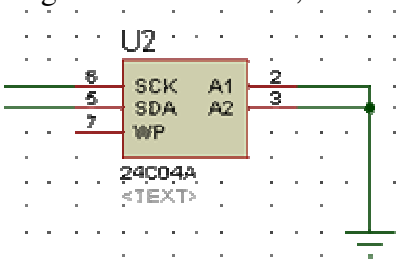
Din punct de vedere al legaturii de date, primitivele care trebuie implementate la master (microcontroler) sunt (combinatii intre SCL si SDA):

- START – orice transfer incepe cu aceasta primitiva generata de master
- STOP – transferurile se termina odata cu aceasta primitiva
- SendByte – masterul trimite un byte si asteapta un bit de ACK
- GetByte_ACK – masterul primeste un byte si raspunde cu bit de ACK
- GetByte_NoACK - masterul primeste un byte si raspunde cu bit de NoACK

Practic, exceptand primitivele START si STOP, comunicatia se desfasoara intotdeauna pe 8 biti plus un bit de acknowledge. Daca masterul trimite un byte, slave-ul raspunde imediat cu un bit care poate fi ACK sau NoACK. Cand slave-ul trimite un byte, masterul trebuie sa ii raspunda imediat cu un bit care poate fi ACK sau NoACK (atentie, masterul/microcontrolerul trebuie sa aiba pregatit (setat intern) bit-ul de ACK sau NoACK inainte sa preia byte-ul de la slave).

Comunicatia se desfasoara bidirectional intre master si slave. Initial, master-ul trimite date, urmand ca slave-ul sa trimita ulterior date. Practic, slave-ul nu trimite niciodata date fara sa i se ceara.
Semnalul de SCL este generat de master. Pentru slave, acest semnal este necesar pentru sincronizare, bitii care intra si ies prin SDA sunt sincronizati cu SCL. Viteza cu care se misca SCL-ul este determinata de modul in care este setat microcontrolerul.

Totul incepe cand masterul trimite un START. Dupa aceasta primitiva, masterul trimite un byte. Acest byte contine headerul tipic IIC (1010xxxx) cat si adresa slave-ului cu care vrea sa comunice. Adresa unui slave pe o magistrala este unica (un fel de MAC address – dar foarte simpla de pana la 3 biti) si este setata legand la masa sau la VCC pinii dedicati ai slave-ului (vezi schema laboratorului 5 PM, circuitul U2 are pinii A2 si A1 legati la masa – astfel, adresa slave-ului U2 este 00)



Acest prim byte trimis de master contine pe ultima pozitie (LSB) bitul de R/W. Daca bitul este 0, atunci urmatorii bytes vor fi trimisi de master (sensul este de la master la slave). Daca R/W este 1, atunci, urmatorii bytes (toti pana la aparitia unui STOP), vor veni de la slave. Odata setata directia, ea nu se mai schimba pana la aparitia unui STOP.

Scenariu 1: Masterul trimite date slave-ului
START
SendByte(**Header Write** + Slave address) + GetACK
SendByte(info for slave) + GetACK
……..
SendByte(info for slave) + GetACK
STOP

Scenariu 2: Masterul citeste date de la slave
START
SendByte(**Header Read** + Slave address) + GetACK
GetByte(info from slave) + SendACK
……
GetByte(info from slave) + Send**No**ACK
STOP

Detaliere byte Header (din pagina de catalog 24c04)

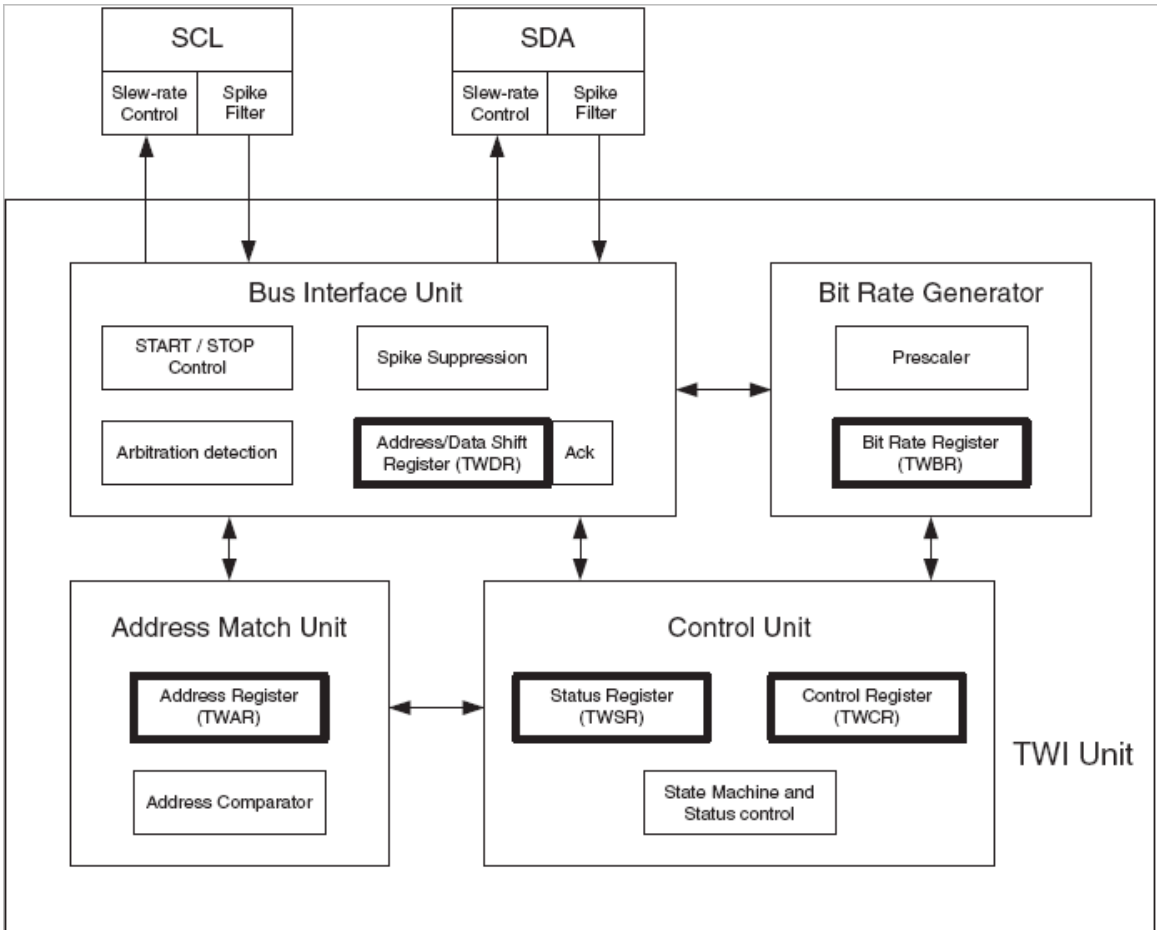
1	0	1	0	A_2	A_1	a_8	$R/\overline{W}$

CAT24C04

Pe primele 4 poziti se gasesc headerul tipic pentru IIC Philips. Urmeaza A2 si A1 care trebuie sa “match” configuratia pinilor de adresa a slave-ului (vezi mai sus). In cadrul laboratorului 5 PM, pinii A2 si A1 sunt pusi la masa, in header trebuie trimis 0 pe aceste pozitii. Bitul **a8** are o semnificatie speciala, el reprezentand bit-ul 8 din adresa memoriei.

I²C – Implementare ATMEGA16

Microcontrolerul ATMega16 pune la dispozitia utilizatorului un automat pentru realizarea comunicatiei TWI (care este un super-set al I²C). Cei doi pini SDA si SCL sunt sub controlul unitatii TWI imediat ce bit-ul TWEN din registrul TWCR este setat. In cadrul laboraturului 5 PM, arhitectura studiata este one_master – one_slave. Microcontrolerul fiind setat ca master, determina neutilizarea registrului TWAR, registru folosit numai cand controlerul TWI este setat ca slave.



Practic, registrii care trebuie setati sunt:

- **TWBR** – **T**wo**W**ire**B**audRate
- **TWCR** – **T**wo**W**ire**C**ontrol**R**egister

Pentru trimiterea si receptionarea datelor, se foloseste:

- **TWDR** – **T**wo**W**ire**D**ata**R**egister

Starea curenta a interfetei se poate afla citind registrul:

- **TWSR** – **T**wo**W**ire**S**tatus**R**egister

Setarea ratei de transfer se face o singura data la initializare. Unitatea TWI merge pe baza divizarii ceasului intern (Q: Cum merge unitatea USART ?)

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

TWBR este valoarea continuta in registru cu acelasi nume
TWPS sunt doi biti continuti in registrul TWSR – sunt bitii de prescaler
O valoare tipica pentru frecventa IIC este de 100KHz

Primitivele detaliate anterior se realizeaza prin setarea bitilor din registrul TWCR.

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **TWEA** TWI va genera bitul de ACK (dupa ce receptioneaza un byte)
- **TWSTA** TWI va genera primitiva START
- **TWSTO** TWI va genera primitiva STOP
- **TWEN** TWI este ON
- **TWINT** TWI Interrupt Flag – se seteaza cand TWI-ul a terminat job-ul

Practic:

- se trimite START daca TWCR este setat cu bitii TWINT, TWSTA si TWEN
- se trimite STOP daca TWCR este setat cu bitii TWINT, TWSTO si TWEN
- se trimite un byte daca:
    1. se scrie valoare in TWDR
    2. se seteza TWCR cu bitii TWINT si TWEN
    3. se asteapta pana cand bitul TWINT din registrul TWCR devine 0
- se receptioneaza un byte
    1. se seteaza TWCR cu bitii TWINT, TWEN si TWEA (daca este cu ACK)
    2. se asteapta pana cand bitul TWINT din registrul TWCR devine 0
    3. se citeste byte-ul din TWDR

24c04 – Detalii

24c04 este de tip EEPROM (**E**rasable**E**lectrical**P**rogrammable**R**ead**O**nly**M**emory)

24Cxx este un nume generic pentru memoriile de tip IIC.
25Cxx este un nume generic pentru memoriile de tip SPI

24C**04** – 04 denota capacitatea memoriei in kilobits – 4 Kbits – adica 512 bytes

Pentru a accesa aceasta memorie avem nevoie de 9 biti de adresa. In continuare ne vom referii la aceasta adresa ca fiind adresa de memorie – a nu se face confuzie cu adresa de slave.

Pentru a putea scrie/citi in/din aceasta memorie, trebuie sa furnizam:

- adresa de slave a device-ului (se poate observa ca se poate pune in circuit pana la 4 astfel de memorii). In cadrul lab 5 pm, adresa de slave este 0
- adresa locatiei de memorie unde vrem sa scriem/citim data
- data care trebuie scrisa

Scenariu 1: Masterul scrie 0x55 in memorie la locatia 0
(Nota: 1010 este standard IIC
00 este adresa de slave
0 este adresa a8 (MSB) a memorie
0 este bitul de R/W – in acest caz este Write

START
SendByte(10100000) GetAck() - HEADER
SendByte(00000000) GetAck() - ADRESA MEMORIE
SendByte(01010101) GetAck() - DATA
STOP
Wait pana cand se scrie informatia in memorie – in jur de 10 msec

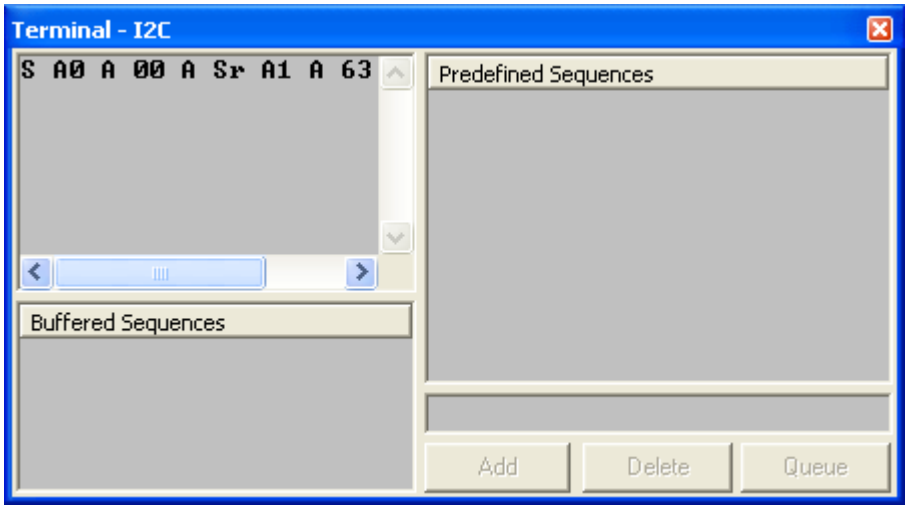
Scenariu 2: Masterul citeste o data din memorie de la locatia 0
(Nota: 1010 este standard IIC
00 este adresa de slave
0 este adresa a8 (MSB) a memorie
0 este bitul de R/W – in acest caz prima ora este Write, apoi Read

START
SendByte(10100000) GetAck() - HEADER de WRITE
SendByte(00000000) GetAck() - ADRESA MEMORIE
START
SendByte(10100001) GetAck() - HEADER de READ
GetByte( ) SendNoAck() - DATA
STOP

Rezolvare laborator

Daca se utilizeaza simulatorul Proteus, un tool foarte bun pentru depanarea aplicatiilor cu IIC este **IIC Debugger** din meniul Virtual instruments 

In timpul rularii/simularii aplicatiei, se poate deschide Terminal – I2C si se poate urmarii activitatea pe bus-ul IIC.

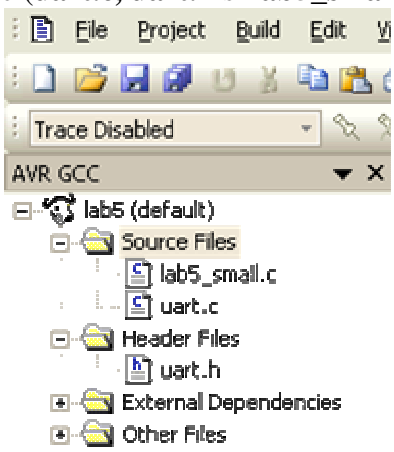


Exemplu: Daca se ruleaza lab5_small.hex, efectul se poate vede in prima linie din terminal: **S A0 A 00 A Sr A1 A 63 A 72 N**
Traducerea poate fi de genul:
S = Start
Send **A0** = 10100000 = Header write pentru slave cu adresa 00 si adresa de memorie 0
A = Ack (care vine de la slave)
Send **00** = restul de adresa de memorie (a7…a0)
A = Ack (care vine de la slave)
Sr = Start repetitiv - teoretic un Start nu poate aparea decat dupa un Stop
Send **A1** = 10100001 = Header read pentru slave cu adresa 00 si adresa de memorie 0
A = Ack (care vine de la slave)
Get **63**
A = Ack (care vine de la master de data asta)
Get **72**
N = NoAck (care vine de la master) – echivalent cu Stop

Cerinte laborator 5 PM

1) Parcurgerea tutorialului
- datasheet-ului ATMEGA16
- datasheet-ului 24c04

2) Creearea unui proiect (blank) AvtStudio cu numele **lab5**
- adaugarea fisierelor de pe site (**uart.c, uart.h** si **lab5_small.c**)



- completarea TODO-urilor conform cu tutorialul
- compilarea
- rularea in Proteus a **lab5_small.hex** folosind schema **lab5.dsn** de pe site
- adaugati functia eeprom_write_byte(memory_address, byte)
- verficati ce raporteaza eeprom_read()
- adaugati functia eeprom_write(memory_address, data_length, buffer)
- verficati ce raporteaza eeprom_read()

3) Inlocuiti in proiect fisierul **lab5_small.c** cu **lab5.c** (de pe site)
- completarea TODO-urilor
- compilarea
- rularea in Proteus
- programarea placutei

Proiectul **Lab5** asteapta o comanda de la VTerm sau HiperTerminal. Aceasta comanda poate fi **read** sau **write**. Programul intelege comanda, si o executa, afisand rezultatul in fereastra VTerm-ului. Scrierea se face de la o adresa specificata, iar citirea se face pe intreaga memorie de la adresa 0.