Серийни синхронни интерфейси

Автор: гл. ас. д-р инж. Любомир Богданов

Европейски съюз

ПРОЕКТ BG051PO001--4.3.04-0042
„Организационна и технологична инфраструктура за учене през целия живот и развитие на компетенции”
Проектът се осъществява с финансовата подкрепа на
Оперативна програма „Развитие на човешките ресурси”,
съфинансирана от Европейския социален фонд на Европейския съюз
Инвестира във вашето бъдеще!

ЕСФ България
Европейски социален фонд

Съдържание

Интерфейс SPI

SPI (**S**erial **P**eripheral **I**nterface) – интерфейс за комуникация между две и повече ИС в рамките на една вградена система. Това е сериен, синхронен, напрежителен, несиметричен интерфейс, използващ до 4 проводника за обмен на данни. Изходите и входовете са несиметрични. Интерфейсът е измислен от Motorola.

MOSI (**M**aster **O**utput / **S**lave **I**nput) – изход на главното устройство / вход на подчиненото.

MISO (**M**aster **I**nput / **S**lave **O**utput) – вход на главното устройство / изход на подчиненото.

SCK (**S**lave **C**loc**k**) – синхронизиращ тактов сигнал, изработван от главното и подаван към подчиненото устройство.

SS (**S**lave **S**elect) – сигнал за избор на подчинено устройство.

Интерфейс SPI

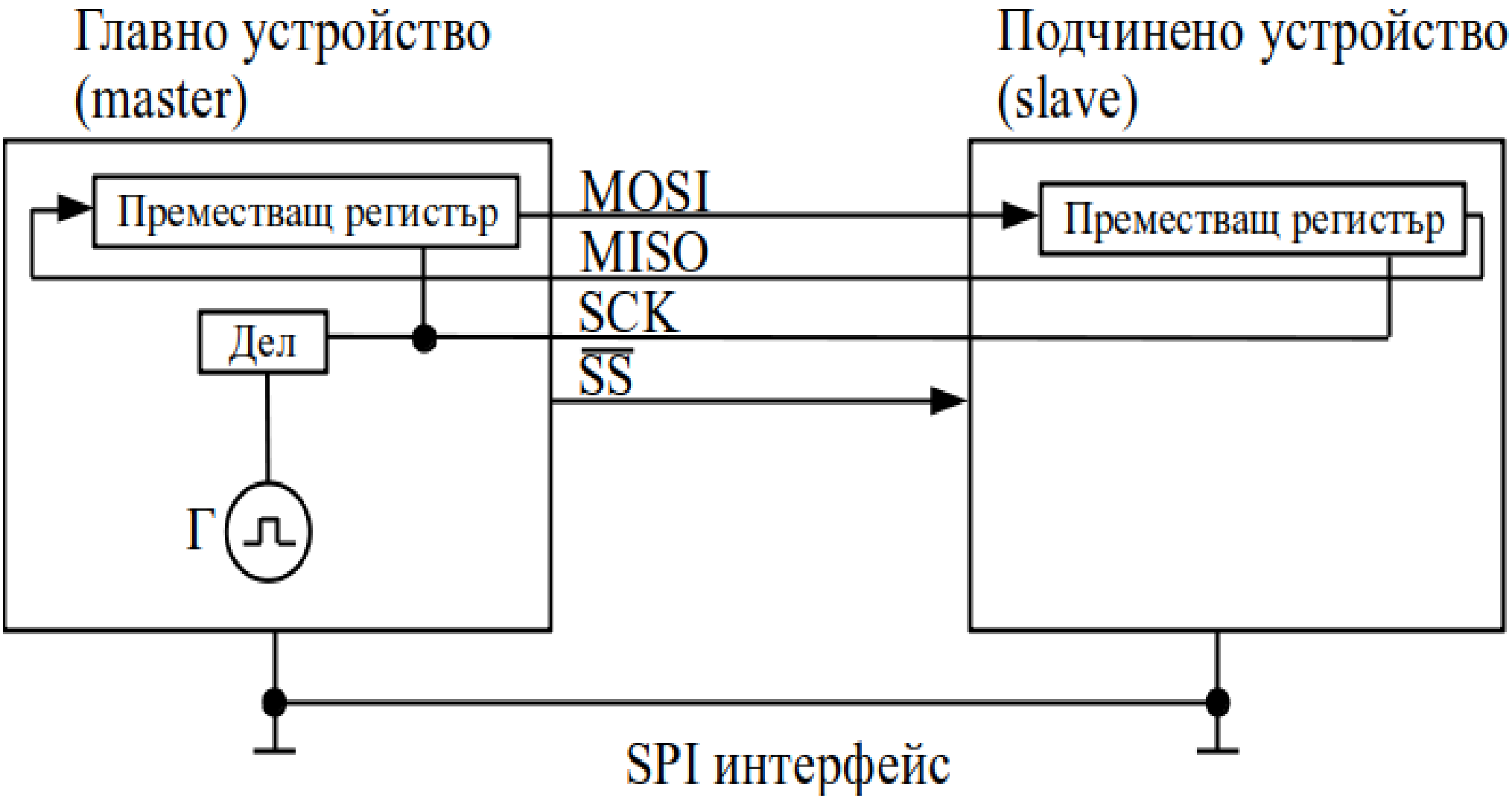
При SPI главното (или още – master) устройство задава синхронизиращия тактов сигнал, спрямо който се предават данните.

Подчиненото устройство (или още – slave) приема правилно данните благодарение на този сигнал.

Генератор на такт има само в главното устройство.

На следващия слайд е дадено едно типично свързване на две ИС по SPI интерфейс.

Интерфейс SPI



Интерфейс SPI

От блоковата схема е видно, че работните регистри на този интерфейс може да се свържат в т.нар. **кръгов буфер**, при който едновременно с изпращане на данни от главното устройство се приемат данни от подчиненото.

Ако например искаме само да четем от подчиненото устройство, може да запишем нули в него, които ще “избутат” (shift) данните от slave-а и ще влязат в master устройството.

Интерфейс SPI

Протоколът е сравнително прост – реализира 4 варианта (SPI modes) на обмен на данни.

На следващия слайд е дадена таблица с тези режими в зависимост от полярността (CPOL) и фронта на тактовия сигнал (или още - фазата на данните спрямо тактовия сигнал, CPHA).

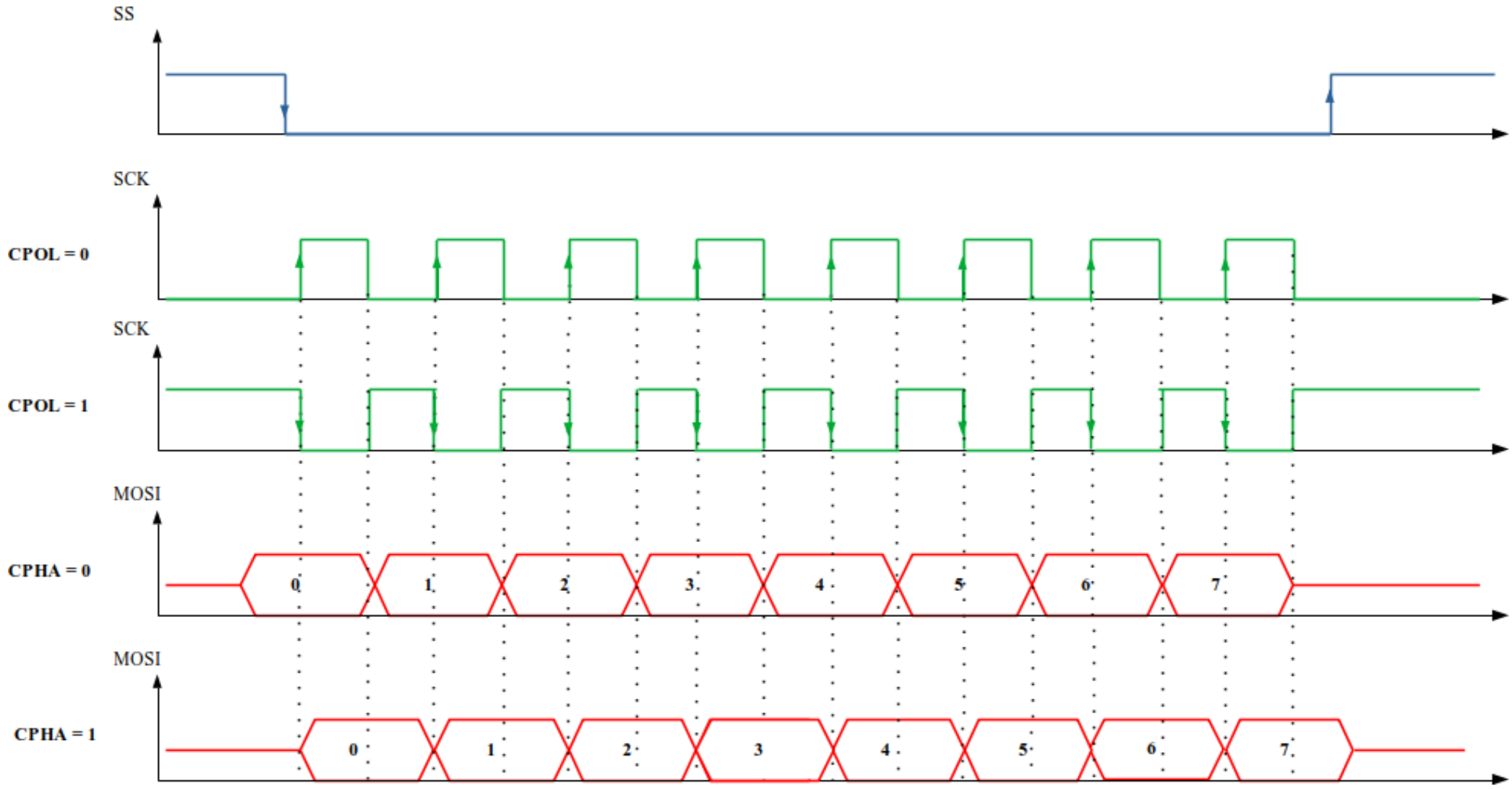
Интерфейс SPI

CPOL	CPHA	Режим
0	0	0
0	1	1
1	0	2
1	1	3

CPOL – определя логическото състояние на проводника, осигуряващ тактов сигнал (SCK), когато по интерфейса няма обмен на данни.

CPHA – определя фронта на тактовия сигнал, по който ще се предават данните.

Интерфейс SPI



Интерфейс SPI

Фазата в SPI протокола може да бъде запомнена и по следния начин – ако данните се четат от приемащото устройство по **първия фронт**, то **CPHA = 0**. Ако се четат по **втория фронт**, то **CPHA = 1**. Дали този фронт ще е нарастващ или спадащ зависи от CPOL.

Думата, изпращана по SPI интерфейс се нарича **SPI фрейм**. Нейната разредност може да се задава програмно при повечето SPI модули.

Чиповете, които използват SPI, обикновено поддържат 8-, 16-, 24- и 32-битови фреймове. Всъщност няма ограничение в дължината и могат да са по-големи.

Интерфейс SPI

Свързването на периферни ИС към SPI интерфейса може да стане по два начина:

***паралелно** – MOSI, MISO и SCK на периферните ИС и микроконтролера са свързани едни към други, а за всяка периферна ИС е осигурен отделен !SS извод.
MISO задължително трябва да поддържа **високоимпедансно състояние**, когато чипът не е избран.

Предимство:
→ бърз обмен на данни.

Недостатъци:
→ използването на повече изводи (повече от един !SS)
→ по-големия товарен капацитет, свързан към изхода MOSI на микроконтролера

Интерфейс SPI

***последователно** (**daisy-chain**) – MOSI извода се свързва към SI (Slave Input) извода на първата ИС. Извода SO на първата ИС се свързва към извода SI на втората и т.н. Последният SO се свързва към MISO извода на микроконтролера. Образува се кръгов буфер от регистрите на всички ИС.

Предимство:

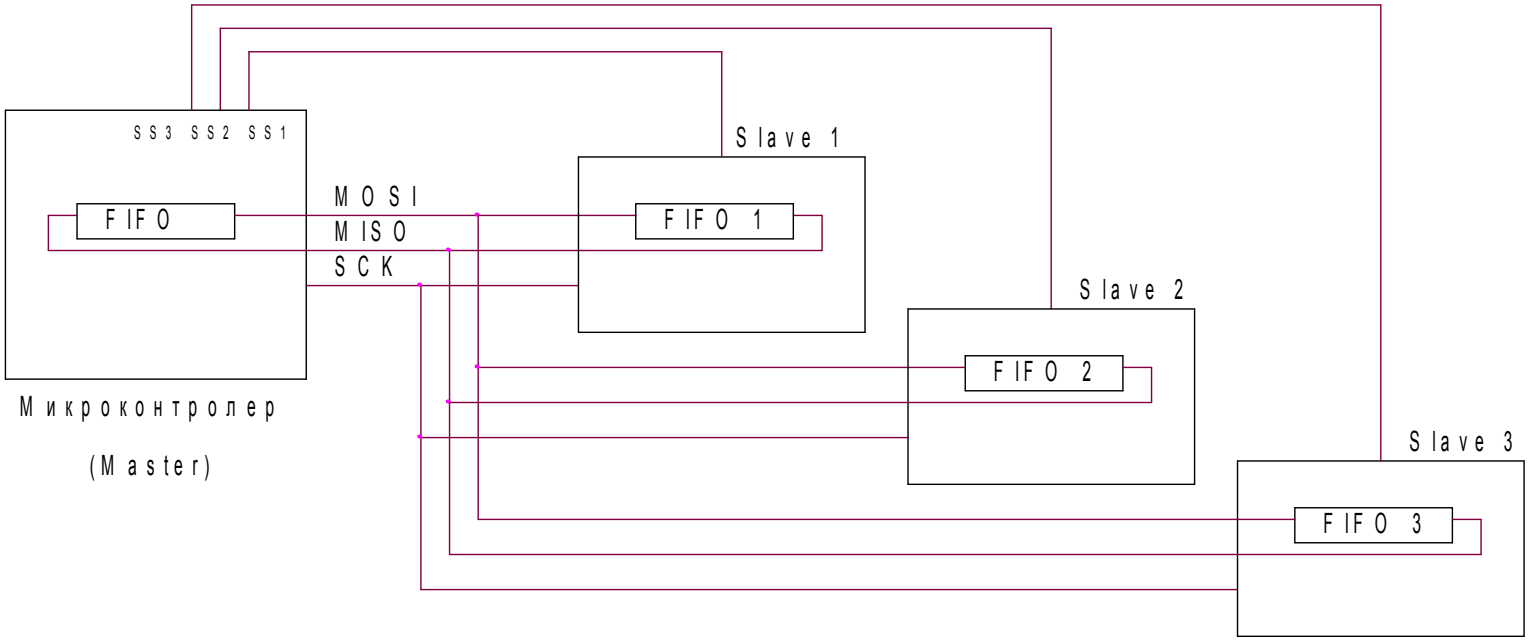
→ минимален брой изводи – 4 (или 3, ако MISO или MOSI не се използват).

Недостатък:

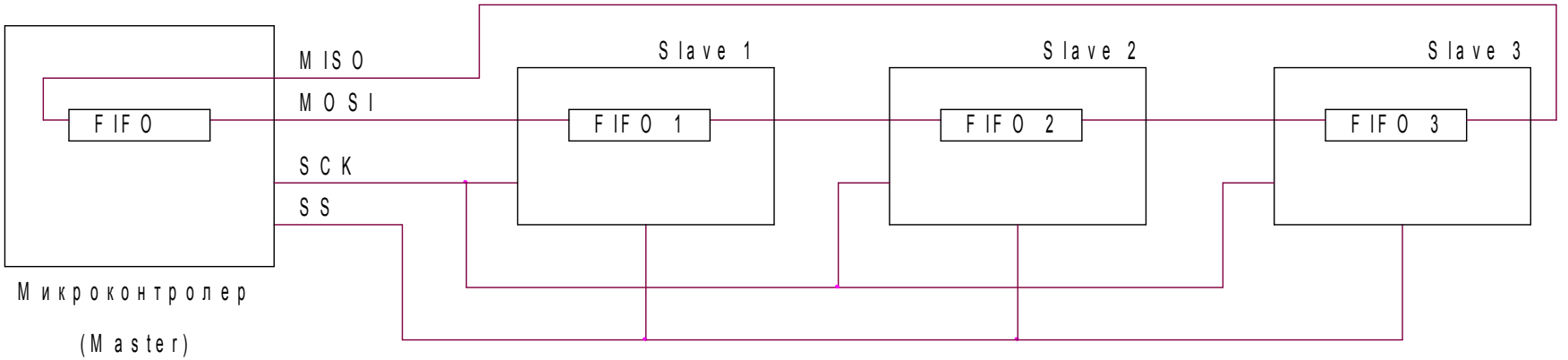
→ по-бавен трансфер спрямо паралелното свързване

Интерфейс SPI

Паралелно свързване



Свързване тип "Daisy-chain"



Интерфейс SPI

Ако предаването на данни по SPI се осъществява само в една посока, то единият от изводите MISO или MOSI е излишен. Ако микроконтролерът само ще чете от подчиненото устройство, **MOSI извода е излишен**. Ако само ще записва в подчиненото устройство, **MISO извода е излишен**.

Невинаги се използва названието SPI за този интерфейс. Някои фирми използват **Microwire** (National Semiconductor), **SSI** (Synchronous Serial Interface на Texas Instruments) и др. Принципно не се различават от оригиналния SPI.

Към SPI може да се свързват най-различни външни периферни ИС като АЦП, ЦАП, температурен датчик, акселерометър, външна Flash памет и др.

SD картите използват SPI за трансфер на данните.

Интерфейс SPI

Някои SPI модули позволяват да се настройват времената (в брой тактове) на трансфера. Четири от най-важните параметри са:

***пред-закъснение** (pre-delay) – времето от падането на !SS в логическа 0 до появяването на данните по MOSI линията.

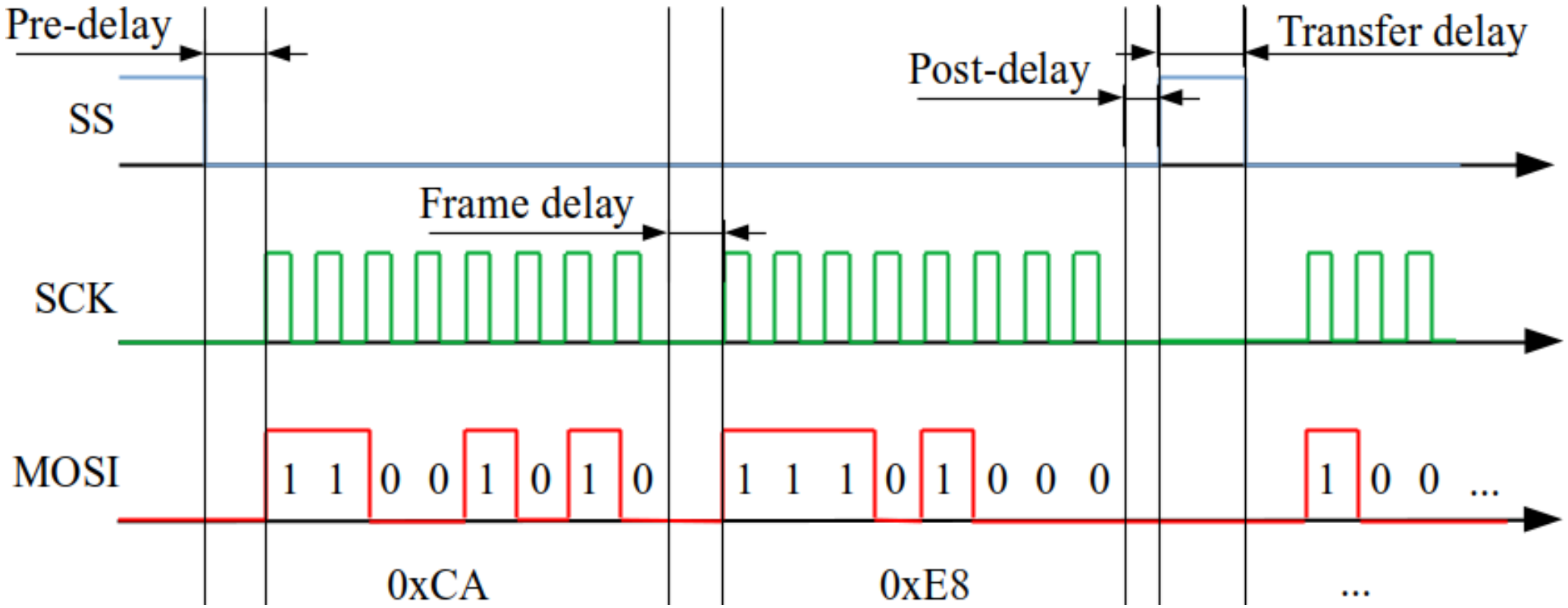
***фреймово закъснение** (frame delay) – закъснението между две съседни думи.

Интерфейс SPI

***след-закъснение** (post-delay) – времето от изпращането на последния бит от последния фрейм по MOSI линията до вдигането на !SS в логическа 1.

***трансферно закъснение** (transfer delay) – минималното време, през което !SS трябва да седи в логическа 1 преди отново да падне в логическа 0.

Интерфейс SPI

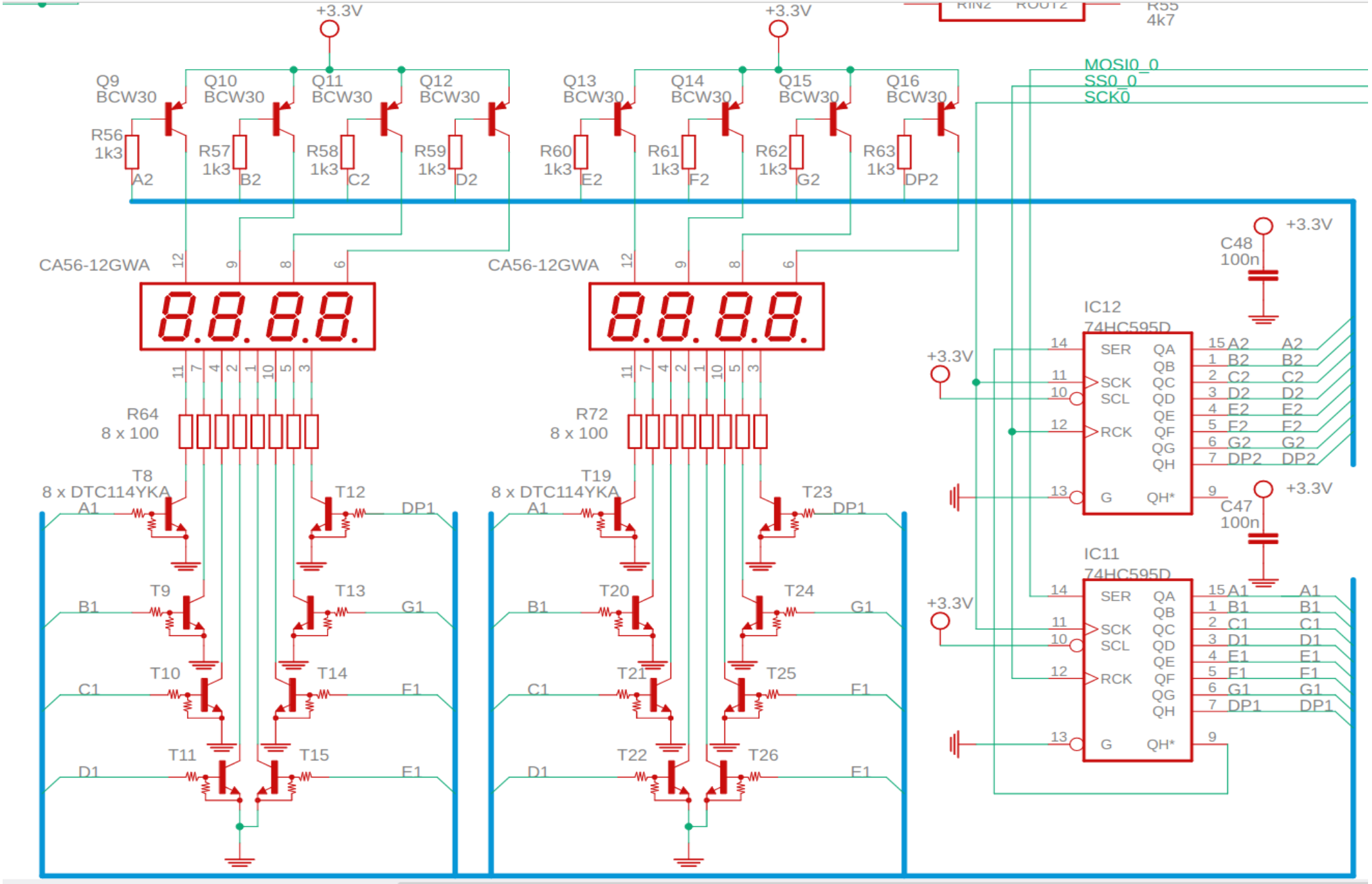


Интерфейс SPI

SPI може да се използва с преместващи регистри от серията 7400 и 4000 като “разширител на порт” [1]. Понеже SPI използва до 4 извода, добавянето на произволен брой входно/изходни сигнали не е проблем.

На следващият слайд е показана динамична индикация с 8 x 7-сегментни индикатори. За управление на светодиодите са необходими 16 сигнала. На теория всичките изводи може да се свържат към μCU, който в случая е LPC845. На практика, за да не се хабят изводите му, се добавят два преместващи регистъра 74HC595, които са свързани към SPI. Така на μCU са му необходими **само 3** – MOSI, SS, SCK.

Интерфейс SPI



Интерфейс SPI

За да има динамично обхождане на сегментите се пуска таймер, който периодично генерира прекъсване (2 ms). Във функцията на прекъсването се изпращат байтове по SPI интерфейса. Съдържанието на байтовете е направено така:

*битовете, които избират индикатор съдържат една логическа нула (активното ниво за избор на индикатор е 0, заради PNP транзисторите Q9 ÷ Q16), която се премества последователно от най-левия към най-десния индикатор.
*битовете, които избират сегмент следват специална комбинация от нули и единици, които съответстват на число, буква или специален символ. Не всички букви могат да бъдат изобразени на такъв вид индикатор.

Повече за индикацията в лекцията за въвеждане и извеждане на информация в μCU.

Интерфейс SPI

Записът в преместващите регистри става посредсвтом кода, даден по-долу. За !SS се използва GPIO извод, понеже към интерфейса има свързани други схеми, а контролера не поддържа паралелно свързване.

```
void segm_led_show_digit(uint8_t line_number, uint8_t digit_position, uint8_t segments, uint8_t dot){
    uint8_t tx_buff[2] = { 0x00, 0xFF };

    ... попълване на tx_buff с два байта

    while(pps_flags.spi_busy){ }
    pps_flags.spi_busy = 1;
    GPIO_PinWrite(GPIO, 1, 7, 0);
    spi_write_half_word(tx_buff);
    GPIO_PinWrite(GPIO, 1, 7, 1);
    pps_flags.spi_busy = 0;
}
```

Интерфейс SPI

Функцията spi_write_half_word( ) се използва за запис на байтовете в 74HC595:

```
void spi_write_half_word(uint8_t *half_word){
    spi_transfer_t xfer = {0};

    xfer.txData = half_word;
    xfer.rxData = NULL;
    xfer.dataSize = 2;
    xfer.configFlags = kSPI_EndOfFrame | kSPI_ReceiveIgnore;

    SPI_MasterTransferBlocking(SPI0, &xfer);
}
```

Интерфейс SPI

Инициализацията на SPI модула е показана по-долу.

```
void spi_init(void){
        gpio_pin_config_t gpio_init_struct;

        IOCON_PinMuxSet(IOCON, IOCON_INDEX_PIO0_0, IOCON_MODE_INACT); //SCK0
        IOCON_PinMuxSet(IOCON, IOCON_INDEX_PIO1_7, IOCON_MODE_INACT); //SSEL0.0
        IOCON_PinMuxSet(IOCON, IOCON_INDEX_PIO1_8, IOCON_MODE_INACT); //SSEL0.1
        IOCON_PinMuxSet(IOCON, IOCON_INDEX_PIO1_9, IOCON_MODE_INACT); //MOSI0 - spi flash
        IOCON_PinMuxSet(IOCON,  IOCON_INDEX_PIO0_6,  IOCON_MODE_INACT);  //MOSI0  -  7segm
led
        IOCON_PinMuxSet(IOCON, IOCON_INDEX_PIO0_13, IOCON_MODE_PULLUP); //MISO0

        SWM_SetMovablePinSelect(SWM0, kSWM_SPI0_SCK, kSWM_PortPin_P0_0);
        SWM_SetMovablePinSelect(SWM0, kSWM_SPI0_MOSI, kSWM_PortPin_P0_6);
        SWM_SetMovablePinSelect(SWM0, kSWM_SPI0_MISO, kSWM_PortPin_P0_13);

        gpio_init_struct.pinDirection = kGPIO_DigitalOutput;
        gpio_init_struct.outputLogic = 1;
        GPIO_PinInit(GPIO, 1, 7, &gpio_init_struct); //SSEL0.0
        GPIO_PinInit(GPIO, 1, 8, &gpio_init_struct); //SSEL0.1

        CLOCK_Select(kSPI0_Clk_From_MainClk);

        spi_clock_source_hz = CLOCK_GetFreq(kCLOCK_MainClk);
```

Интерфейс SPI

```
spi_init_struct.baudRate_Bps = SPI0_SEGM_LED_BAUD_RATE;
spi_init_struct.clockPolarity = kSPI_ClockPolarityActiveLow;
spi_init_struct.clockPhase = kSPI_ClockPhaseSecondEdge;
spi_init_struct.dataWidth = 15;
spi_init_struct.delayConfig.frameDelay = 5;
spi_init_struct.delayConfig.postDelay = 5;
spi_init_struct.delayConfig.preDelay = 5;
spi_init_struct.delayConfig.transferDelay = 5;
spi_init_struct.direction = kSPI_MsbFirst;
spi_init_struct.enableLoopback = 0;
spi_init_struct.enableMaster = 1;
spi_init_struct.sselNumber = kSPI_SselDeAssertAll;
spi_init_struct.sselPolarity = kSPI_SpolActiveAllLow;

SPI_MasterInit(SPI0, &spi_init_struct, spi_clock_source_hz);
}
```

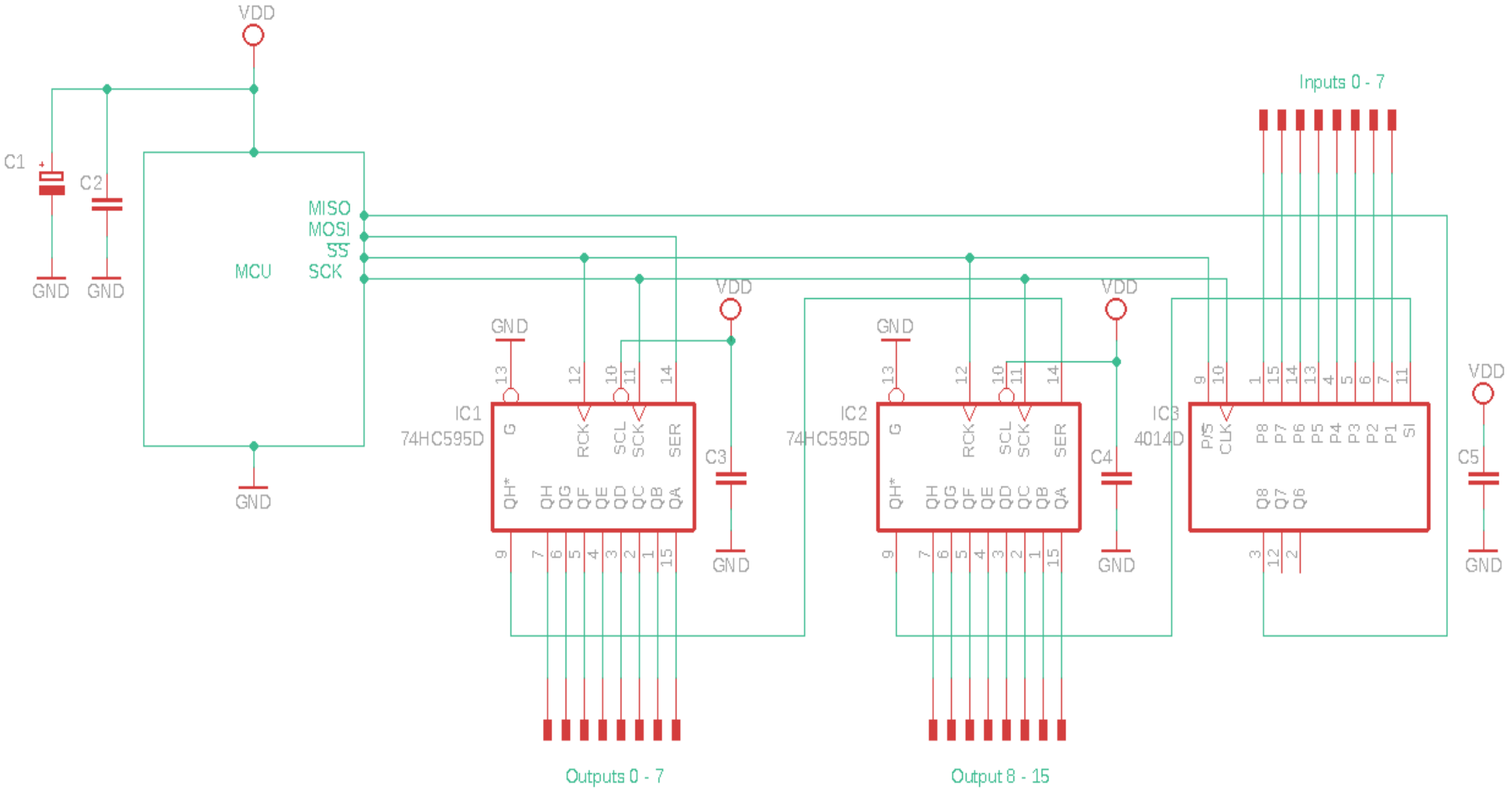
Интерфейс SPI

На следващият слайд е показан пример, в който 74HC595D се използва за изход, а 4014D – за вход. Обърнете внимание как регистрите са свързани последователно и образуват кръгов буфер. Един трансфер може да се осъществи с 24-битов фрейм.

*При запис - старшите 8 бита са без значение и отговарят на регистъра на 4014D, а **младшите 16 бита** ще установят нивата на **изходите на 74HC595D**.

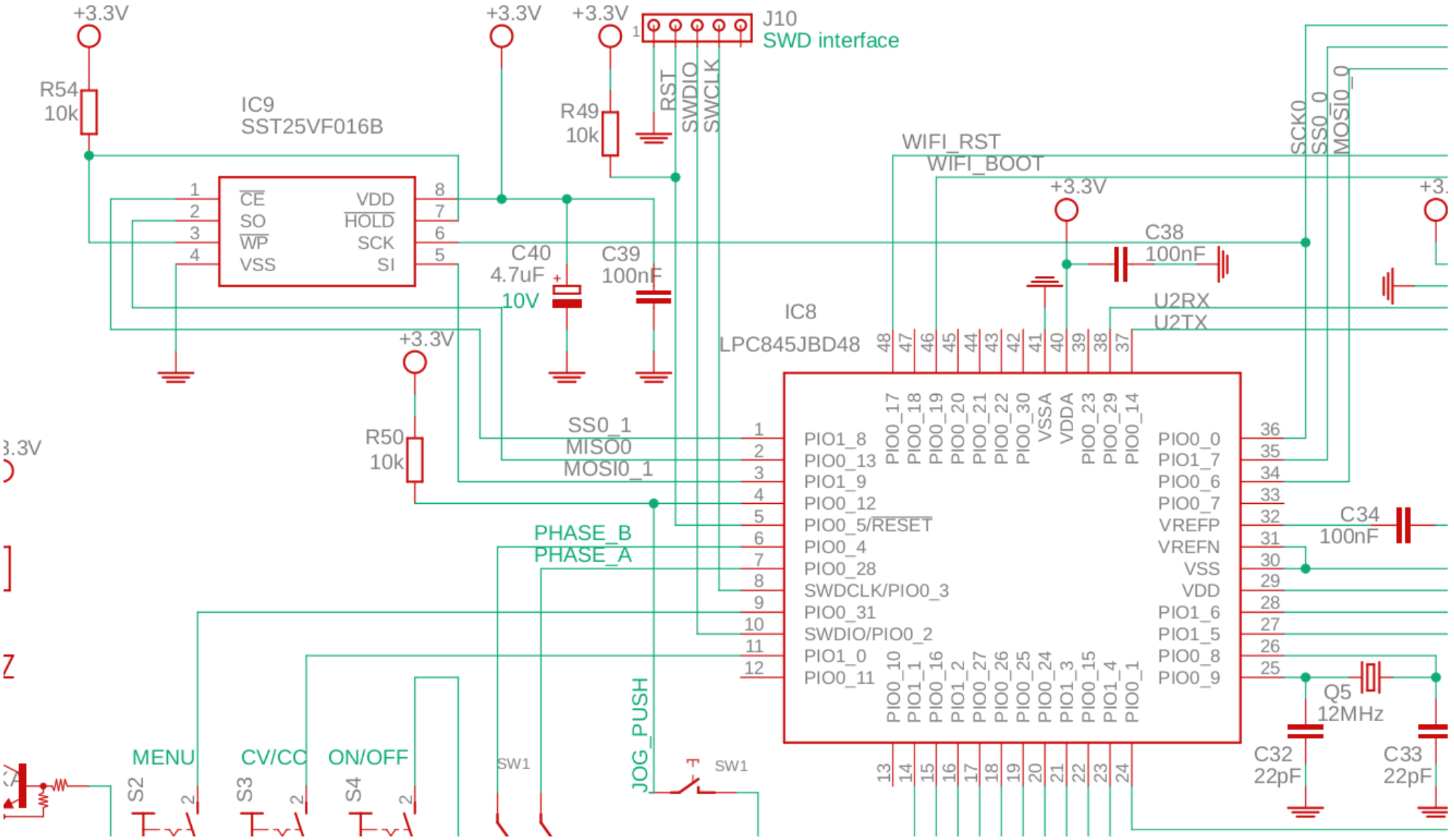
*При четене - полученият по MISO обратно фрейм ще съдържа входната информация. **Старшите 8** бита ще отразяват състоянието на **входовете на 4014D**, а младшите 16 бита са без значение (те ще съдържат числата, записани в 74HC595D от предишния трансфер).

Интерфейс SPI



Интерфейс SPI

Пример с външна SPI флаш памет SST25VF016B.



Интерфейс SPI

Външната SPI флаш памет SST25VF016B на Microchip е 16 Mbit (2 MB).

Скоростта на SPI интерфейса ѝ е до 50 MHz.

Свързването е 4 проводно.

По интерфейса се различават три вида числа – инструкции, параметри на инструкциите и данни. Всеки трансфер започва **първо с инструкция**, след това с **параметри** на инструкцията (ако има) и завършва с запис/четене на **данни**.

Интерфейс SPI

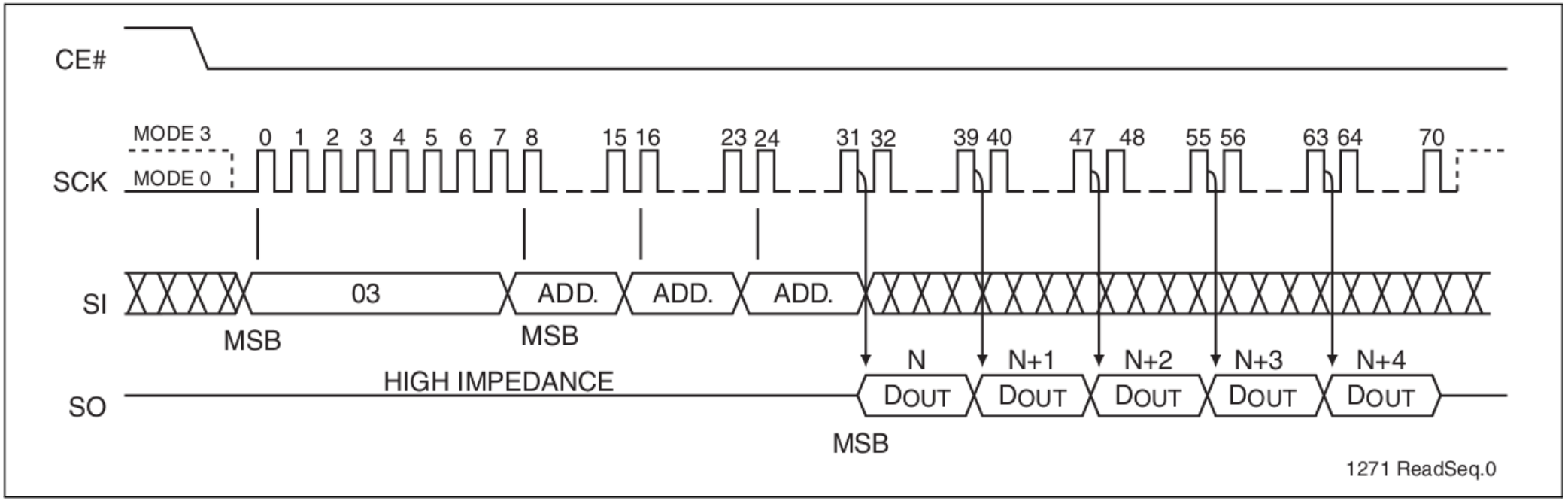
Поддържани инструкции:

Instruction	Description	Op Code Cycle[1]	Address Cycle(s)[2]	Dummy Cycle(s)	Data Cycle(s)	Maximum Frequency
Read	Read Memory at 25 MHz	0000 0011b (03H)	3	0	1 to ∞	25 MHz
High-Speed Read	Read Memory at 50 MHz	0000 1011b (0BH)	3	1	1 to ∞	50 MHz
4 KByte Sector-Erase[3]	Erase 4 KByte of memory array	0010 0000b (20H)	3	0	0	50 MHz
32 KByte Block-Erase[4]	Erase 32 KByte block of memory array	0101 0010b (52H)	3	0	0	50 MHz
64 KByte Block-Erase[5]	Erase 64 KByte block of memory array	1101 1000b (D8H)	3	0	0	50 MHz
Chip-Erase	Erase Full Memory Array	0110 0000b (60H) or 1100 0111b (C7H)	0	0	0	50 MHz
Byte-Program	To Program One Data Byte	0000 0010b (02H)	3	0	1	50 MHz
AAI-Word-Program[6]	Auto Address Increment Programming	1010 1101b (ADH)	3	0	2 to ∞	50 MHz
RDSR[7]	Read-Status-Register	0000 0101b (05H)	0	0	1 to ∞	50 MHz
EWSR	Enable-Write-Status-Register	0101b 0000b (50H)	0	0	0	50 MHz
WRSR	Write-Status-Register	0000 0001b (01H)	0	0	1	50 MHz
WREN	Write-Enable	0000 0110b (06H)	0	0	0	50 MHz
WRDI	Write-Disable	0000 0100b (04H)	0	0	0	50 MHz
RDID[8]	Read-ID	1001 0000b (90H) or 1010 1011b (ABH)	3	0	1 to ∞	50 MHz
JEDEC-ID	JEDEC ID read	1001 1111b (9FH)	0	0	3 to ∞	50 MHz
EBSY	Enable SO to output RY/BY# status during AAI programming	0111 0000b (70H)	0	0	0	50 MHz
DBSY	Disable SO as RY/BY# status during AAI programming	1000 0000b (80H)	0	0	0	50 MHz

Интерфейс SPI

Пример с четене на няколко байта, разположени на последователни адреси.

FIGURE 4-3: READ SEQUENCE

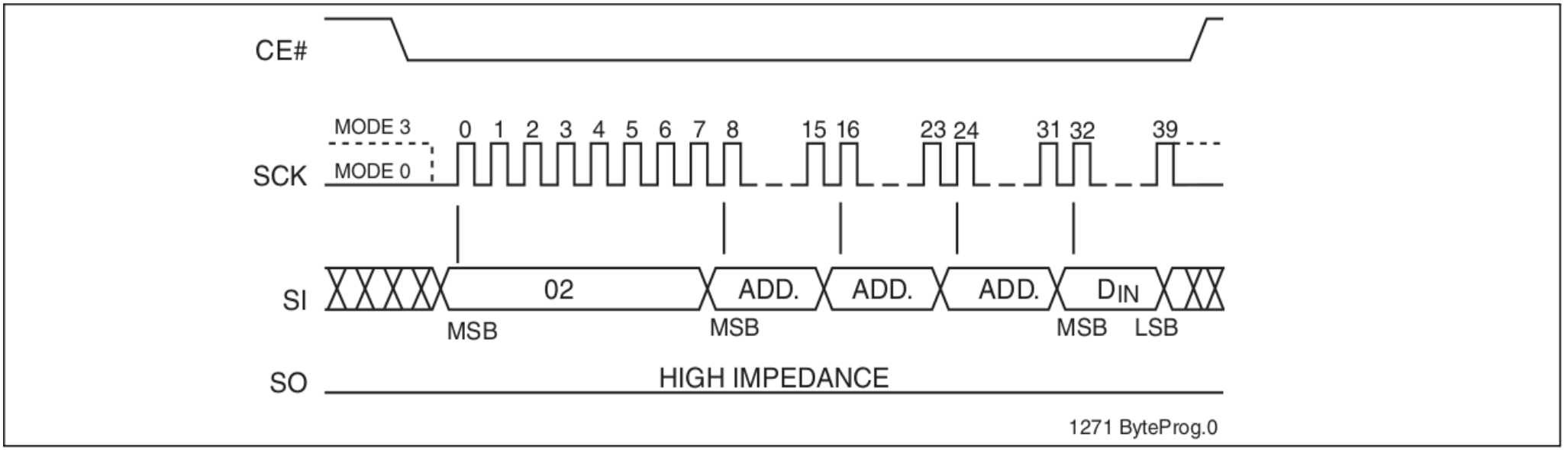


Интерфейс SPI

Пример със запис на един байт. Преди да бъде записан байта, съответния сектор/страница трябва да бъде **изтрит**.

Повече за флаш паметите в лекцията за настройка и диагностика на микропроцесорни системи.

FIGURE 4-5: BYTE-PROGRAM SEQUENCE



Интерфейс SPI

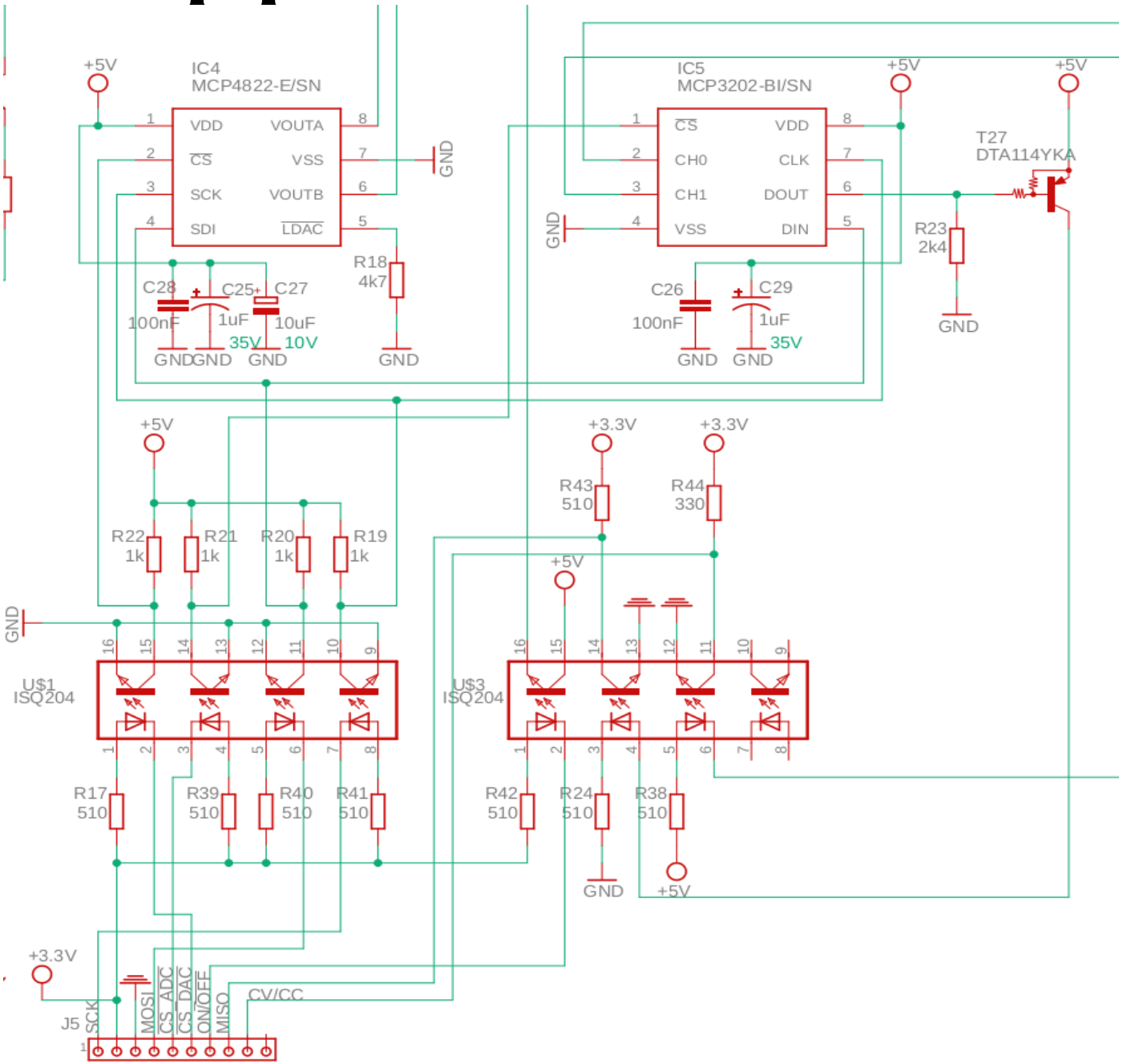
Галванично разделяне и транслиране на нива. Пример с ЦАП, АЦП.

Сигналите са неинвертирани. Ако се инвертирани, трябва:

uint8_t byte_inv;

byte_inv = ~byte_data;

spi_write(byte_inv);



Интерфейс SPI

При ток през фотодиода = 10 mA, времената за отпушване/запушване на фото-транзистора са съответно 3 и 2.5 μs. Това означава, че всеки бит ще отнеме поне 10 μs, което отговаря на 100 kHz, или 100 kbit/s.

Авторката Jan Axelson препоръчва 5 ÷ 10 пъти по-ниска скорост от теоретично максималната за по-сигурно предаване. Такива **скорости са бавни** за SPI интерфейс.

Има още по-бавни оптрони с on/off времена = 20 μs ...

Input	Forward Voltage (V_F)		1.2	1.65	V	I_F = 50mA
	Reverse Current (I_R)			10	μA	V_R=4V
Output	Collector-emitter Breakdown (BV_{CEO}) (Note 2)	70			V	I_C = 1mA
	Emitter-collector Breakdown (BV_{ECO})	6			V	I_E = 100μA
	Collector-emitter Dark Current (I_{CEO})			50	nA	V_{CE} = 10V
Coupled	Current Transfer Ratio (CTR) (Note 2)					
	IS201, ISD201, ISQ201	75			%	10mA I_F, 10V V_{CE}
	IS201, ISD201, ISQ201	10			%	1mA I_F, 10V V_{CE}
	IS202, ISD202, ISQ202	125		250	%	10mA I_F, 10V V_{CE}
	IS202, ISD202, ISQ202	30			%	1mA I_F, 10V V_{CE}
	IS203, ISD203, ISQ203	225		450	%	10mA I_F, 10V V_{CE}
	IS203, ISD203, ISQ203	50			%	1mA I_F, 10V V_{CE}
	IS204, ISD204, ISQ204	200		400	%	10mA I_F, 10V V_{CE}
	IS204, ISD204, ISQ204	100			%	1mA I_F, 10V V_{CE}
	Collector-emitter Saturation Voltage $V_{CE(SAT)}$		0.2	0.4	V	10mA I_F , 2mA I_C
	Input to Output Isolation Voltage V_{ISO}	5300			V_{RMS}	See note 1
		7500			V_{PK}	See note 1
	Input-output Isolation Resistance R_{ISO}	$5x10^{10}$			Ω	V_{IO} = 500V (note 1)
	Output Turn on Time t_{ON}		3.0		μs	I_F = 10mA
	Output Turn off Time t_{OFF}		2.5		μs	V_{CE} = 5V, R_L = 75Ω

SD карти

SD картите (**S**ecure **D**igital) са памети, проектирани за първи път през 1999 от фирмите Toshiba, SanDisk, Matsushita [1], [2], [3], [4].

Произвеждат се в **три различни корпуса**:

*оригинален (9 извода / 25 MHz)
*мини (11 извода / 50 MHz)
*микро (8 / 50 MHz)

В зависимост от **капацитета** на използваната флаш памет биват 4 вида:

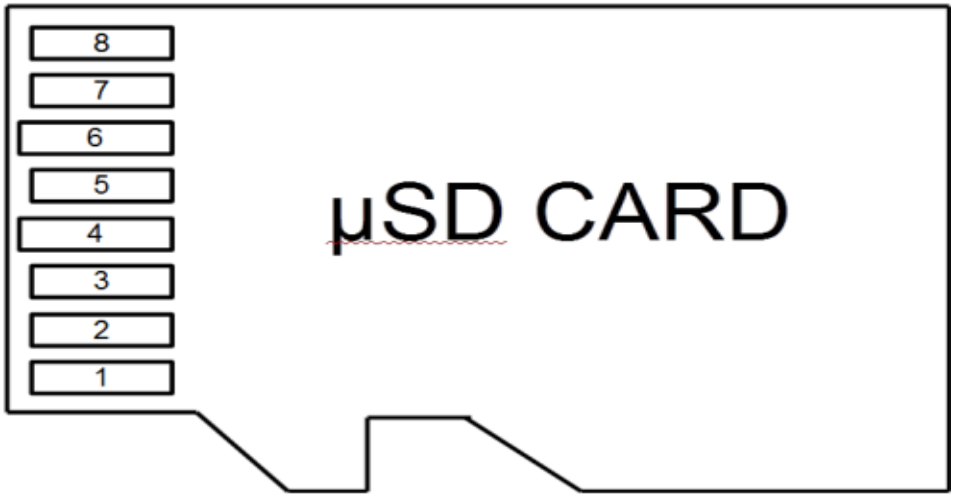
*стандартен (SD Standard Capacity) до 2 GB,
*висок (SD High Capacity) до 32 GB,
*разширен (SD eXtended Capacity) до 2 TB,
*входно/изходен (SD Input/Output) – вместо флаш памет, в слота се включват GPS, модеми, FM радио, RFID четец, Wi-Fi модули и други

SD карти

µSD картата може да използва **2 интерфейса** –
*4-битов паралелен
*SPI (режим 0)

Издърпващи резистори към сигналите DAT0/DAT1/DAT2/DAT3/CMD са желателни. Тяхната стойност е от 10 ÷ 100 kΩ

Извод	Паралелен интерфейс	SPI интерфейс
8	DAT1	
7	DAT0	DO (MISO)
6	VSS	
5	CLK	SCLK (SCK)
4	VDD	
3	CMD	DI (MOSI)
2	DAT3	CS (SS)
1	DAT2	



Фиг. 1 – Разположение на изводите на микро SD карта.

SD карти

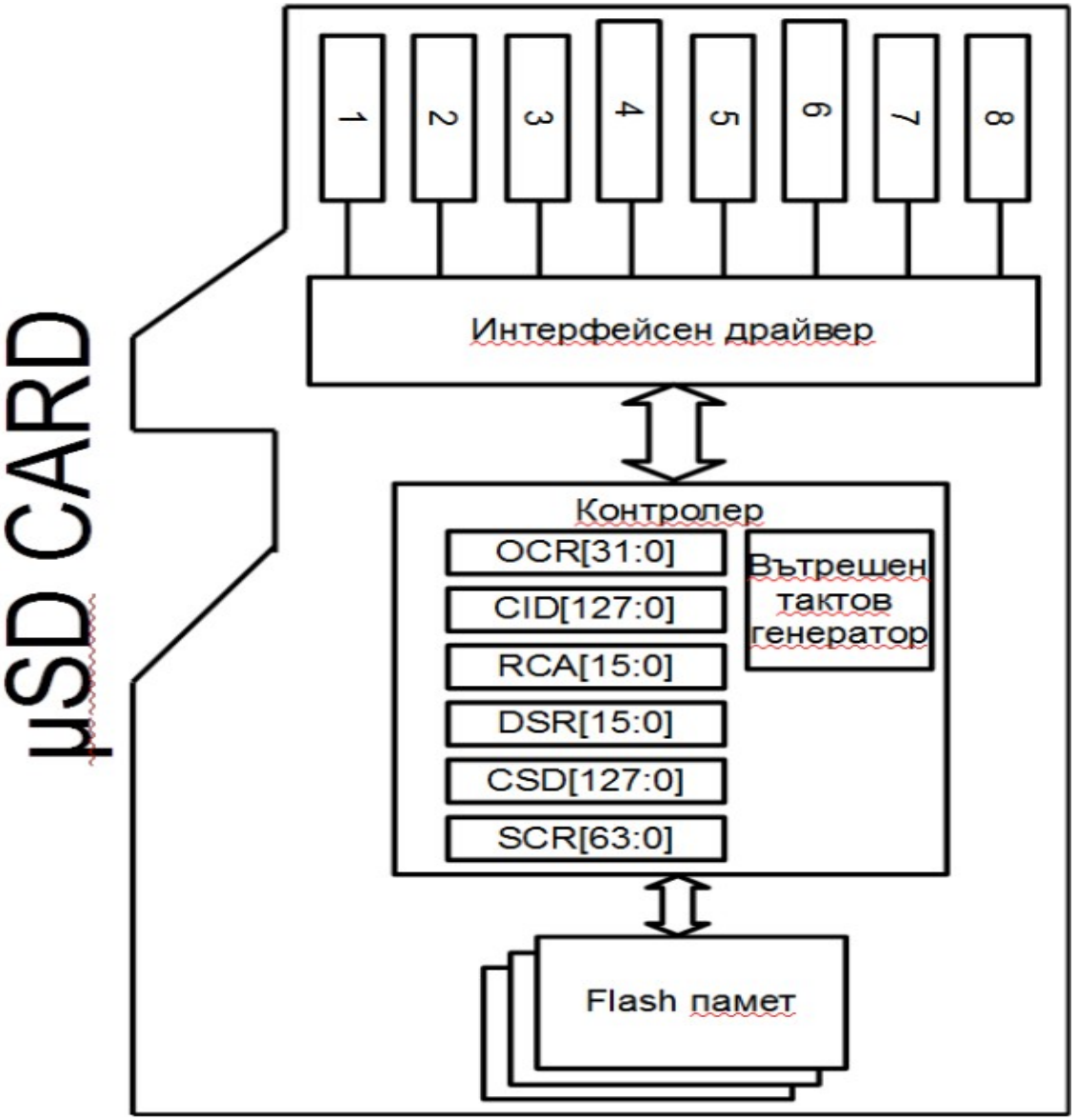
На фиг. 2 е показана блоковата схема на μSD. От нея се вижда, че има **контролер**, който приема **команди** по интерфейса и осъществява записа и четенето на данни от флаш паметта [5], [6].

Логиката се захранва с 2.7 ÷ 3.6 V, а консумацията при запис може да достигне **десетки милиампери**.

Достъпът до паметта става с **групи от байтове**, наречени **блок**.

Размерът на блока по **подразбиране е 512 байта**. Контролерът поддържа команда, с която може да се зададе друг размер на блока (чрез запис в CSD регистъра).
Някой карти поддържат блокове от 1 ÷ 2048 байта, а други само 512/1024/2048.

SD карти



Фиг. 2 – Блокова схема на микро SD карта.

SD карти

При подаване на захранване към картата (чрез вкарването ѝ в SD цокъла) **по подразбиране** тя е конфигурирана за работа с **паралелен интерфейс**.

Инициализационна процедура за работа с SPI.

==========

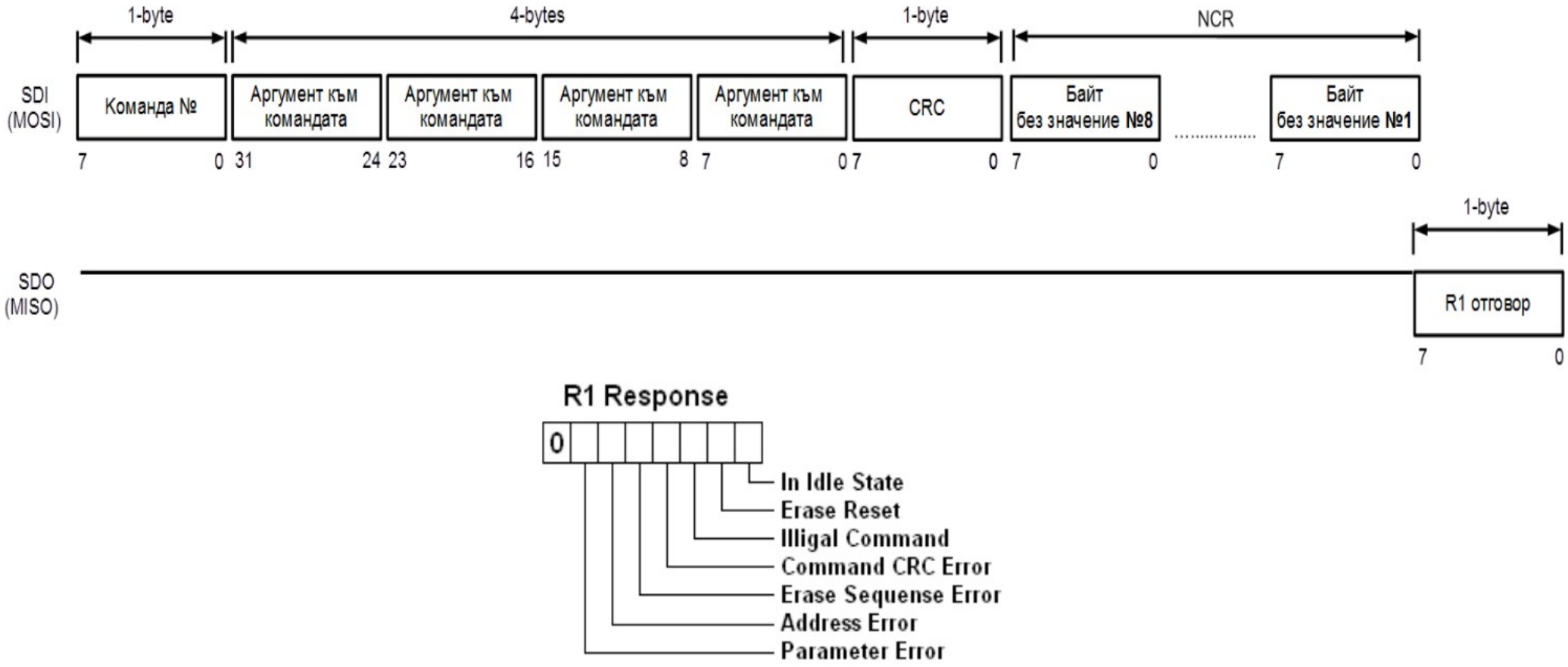
За да се **избере SPI** трябва DI (MOSI) и CS (SS) да се задържат в логическа единица, а SCLK трябва да генерира 74 или повече импулса с честота 100 kHz или 400 kHz. След това контролерът вече ще приема команди по SPI, като първата от тях трябва да е CMD0 (рестарт).

SD карти

Подаването на команди към картата има форматът, показан на фиг. 3. Първо се изпраща **номер на командата**, след това **аргумент**, **CRC байт** за проверка и накрая от **0 ÷ 8 байта** време (наречено NCR), в което контролерът на картата трябва да изпрати **отговор**. В зависимост от изпратеният номер на командата, контролерът връща като отговор различен брой байтове, чието съдържание обикновено са статус битове. Примерни видове отговори са R1(фиг. 3), R2, R3 и R7.

Някои команди отнемат повече време от NCR. Тогава се изпраща отговор R1b (R1 + busy flag), който представлява R1 отговор, последван от флаг за заето устройство, който в случая се реализира с SDO (MISO) = логическа 0 (т.е. предават се постоянно данни 0x00). Микроконтролерът трябва да изчака, докато не получи 0xFF от контролера на SD картата, което означава, че командата е била обработена. Списък с командите е даден на фиг. 4 [7], [8].

SD карти



Фиг. 3 – Комуникация по SPI с SD карта.

SD карти

Command Index	Argument	Response	Data	Abbreviation	Description
CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
ACMD41(*1)	*2	R1	No	APP_SEND_OP_COND	For only SDC. Initiate initialization process.
CMD8	*3	R7	No	SEND_IF_COND	For only SDC V2. Check voltage range.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD16	Block length[31:0]	R1	No	SET_BLOCKLEN	Change R/W block size.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Leading command of ACMD<n> command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.

*1:ACMD<n> means a command sequense of CMD55-CMD<n>.

*2: Rsv(0)[31], HCS[30], Rsv(0)[29:0]

*3: Rsv(0)[31:12], Supply Voltage(1)[11:8], Check Pattern(0xAA)[7:0]

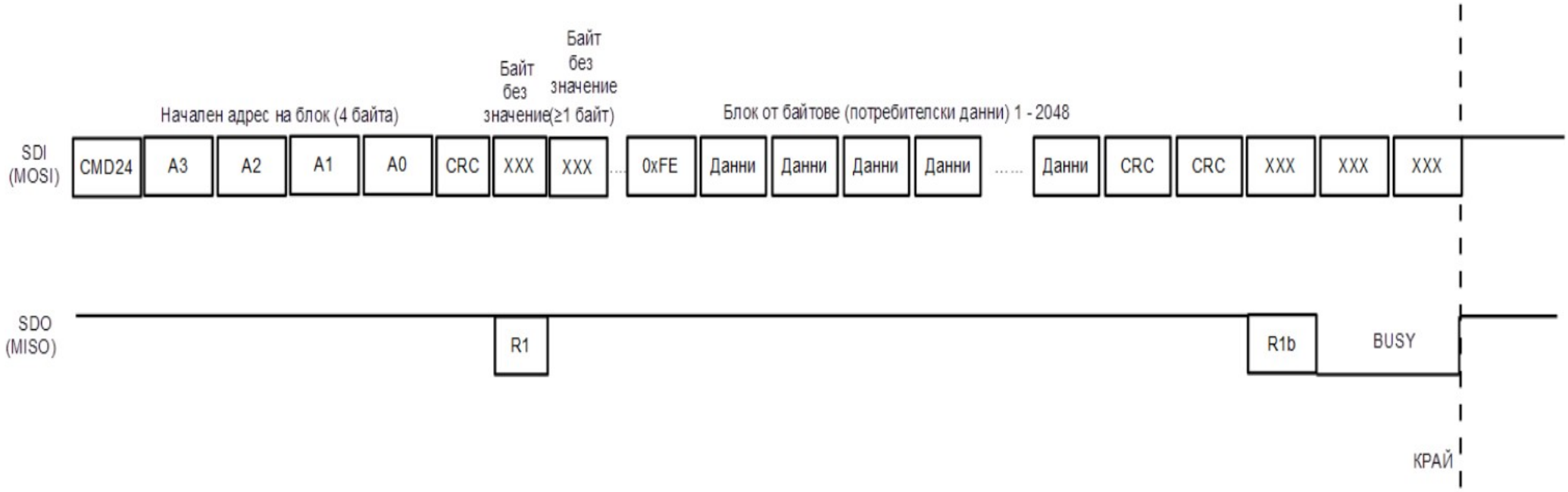
Фиг. 4 – Видове команди, приемани от SD карта.

SD карти

Запис в картата е показан на фиг. 5. Започва се с изпращане на команда CMD24 с аргумент число, указващо адреса на първия байт от блока, в който ще се записва. Изпраща се един байт с CRC. Получава се R1 отговор, след което се изчаква пауза от ≥ 1 байт. След това се изпраща т.нар. **даннов символ (data token),** представляващ **байт 0xFE**. След data token-а се изпращат байтовете на блока, чийто брой варира в зависимост от типа на картата (1 ÷ 2048 байта).

Накрая се изпращат и 2 байта CRC към блока (в SPI режим CRC байтовете не се проверяват от SD по подразбиране, но тази функция може да се включи с CMD59; изключение правят команди CMD0 и CMD8, които винаги трябва да са последвани от валиден CRC байт). След двата CRC байта SD картата отговаря с R1b байт, като busy флагът е държан в нула, докато всички байтове от блока са записани. Последното може да отнеме десетки милисекунди, затова е предвидена команда за запис на няколко блока наведнъж (CMD25).

SD карти

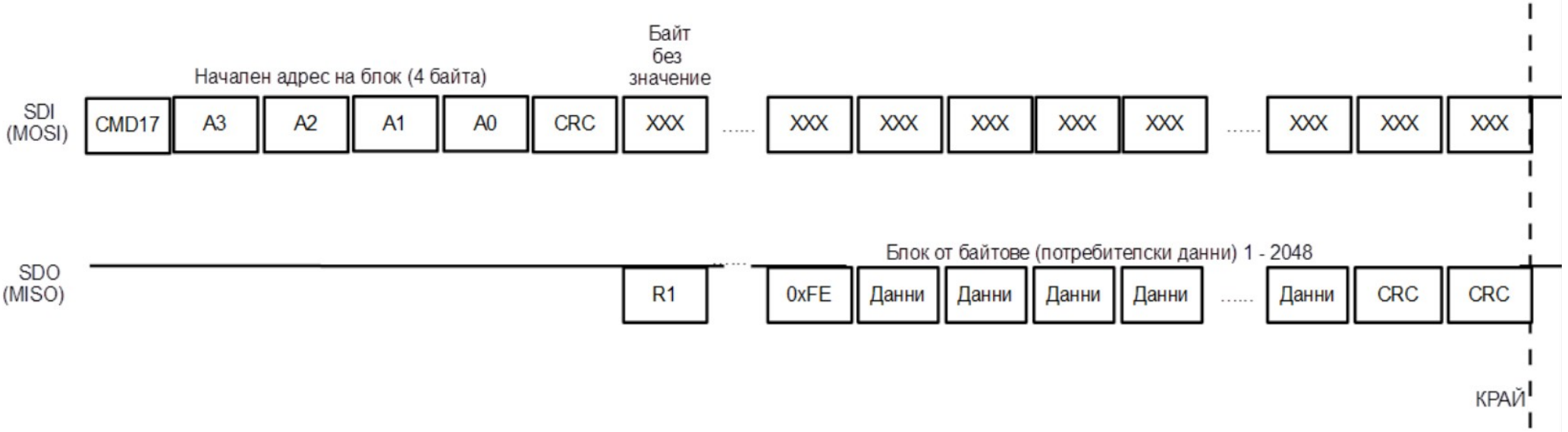


Фиг. 5 – Запис на блок в SD карта.

SD карти

Четене от картата е показано на фиг. 6. Започва се с изпращане на командата CMD17 с аргумент число, указващо адреса на първия байт от блока, от който ще четем. Изпраща се един байт с CRC. Приема се R1 отговор. Изчакват се няколко байта (0xFF), докато пристигне данновия символ (0xFE). Когато той пристигне, следват прочетените данни от блока. В края на прочетените данни има два байта CRC, които могат да бъдат игнорирани в SPI режим. С тях приключва трансферът. Четенето може да стане на няколко блока с командата CMD18.

SD карти



Фиг. 6 – Четене на блок в SD карта.

SD карти

Файловата система контролира съхранението и извличането на данните от паметта. Файлова система се използва с различни видове памети. С помощта на файловата система данните се разделят и идентифицират лесно, което позволява по-добро съхранение на данни от различен тип. Без файлова система не бихме могли да кажем докъде се съхраняват байтовете на едни данни и къде започват байтовете на други. При използване на файлова система данните (байтовете) се групират във логически единици, наречени файлове. Файловете се разполагат на различни адреси в паметта. Възможно е един файл да е разположен в няколко блока на различни адреси. Когато изтрием този файл, освободените блокове могат да се използват за други файлове. След дълга употреба на паметта е възможно голямо „накъсване“ на файловете по различни адреси. Тогава се казва, че

SD карти

паметта е фрагментирана. Достъпът до файл във фрагментирана памет е по-бавен от достъп до файл, разположен на последователни адреси. Затова паметта трябва да се дефрагментира периодично. Използването на файлова система води до намаляване на полезния размер на паметта. Във всеки един файл и в началните адреси от паметта има мета-данни, използвани само от файловата система. Тези данни не са потребителски. Въпреки това използването на файлова система е желателно, тъй като логическото разделяне на файлове води до улеснен достъп до информацията. В резултат програмата, използваща тази информация, не трябва да имплементира алгоритъм за достъп до паметта (това вече е направено от файловата система с функции като **fopen**, **fclose**, **fseek** и т.н.). Допълнително файловата система имплементира проверка за грешки (error correction), back-up на данните и контрол на достъпа до файловете.

SD карти

Съществуват различни видове файлови системи. MMC/SD спецификациите препоръчват следните файлови системи:

*FAT12 за карти с обем ≤64MB;
*FAT16 при 128 MB ÷ 2 GB;
*FAT32 при 4 GB ÷ 32 GB;
*exFAT при 64 GB ÷ 2TB.

SD карти

На следващите няколко слайда е дадена примерна програма, която записва файл с име myfile.txt и съдържание “Hello, world!” в SD карта от микроконтролера LM4F232 и FAT32 файлова система.

Файловата система изисква периодично извикване (10 ms) на функцията disk_timerproc( ) за правилната работа на вътрешната ѝ логика.

```
void SysTickHandler(void){
   // Call the FatFs tick timer.
   disk_timerproc();
}
```

SD карти

```
int main(void){
    WORD bytes_written;
    FATFS fatfs;
    FIL myfile;

    char *msg = "Hello, world!"; //13+1 chars

    … Инициализация на системен такт, таймер и SPI интерфейс …

    //API функции за достъп до SD картата и работа с файлове
    f_mount(0, &fatfs);   //Mount the file system, using logical disk 0.
    f_open(&myfile, "myfile.txt", FA_WRITE | FA_CREATE_NEW);
    f_write(&myfile, msg, 14, &bytes_written);
    f_close(&myfile);
    f_mount(0, NULL);   //Unmount the file system

    while(1){ }
}
```

Интерфейс QSPI

QSPI интерфейсът (Quad SPI) е паралелен, синхронен, напрежителен, несиметричен интерфейс, базиран на SPI.

Предаването е от вида полу-дуплекс.

Към сигналите **!SS** и **SCK** се добавят 4 нови IO0 ÷ IO4.

Сигнали IO0 ÷ IO4 – използват се за предаване на 4-бита данни в паралел. Така за 2 такта на SCK може да се предадат 8 бита. При обикновения SPI тази операция ще отнеме 8 такта => **QSPI е 4 пъти по-бърз от SPI**.

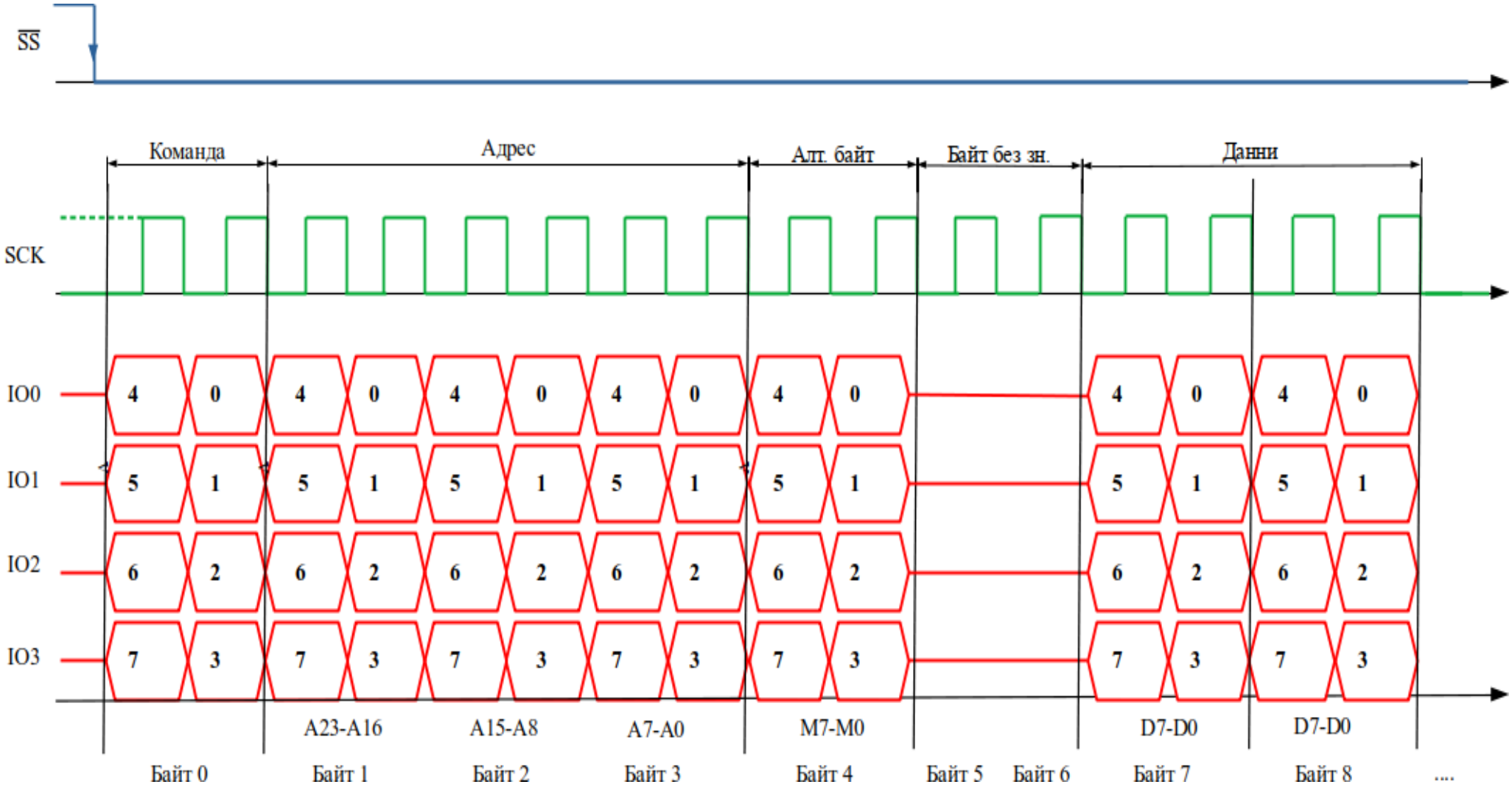
Интерфейс QSPI

DSPI интерфейсът (Dual SPI) е еквивалентен на QSPI, но използва само два проводника за данни – IO0 и IO1.

QSPI се използва за добавяне на външна флаш памет, която е **внедрена в адресното поле на μCU** (memory mapped). Така μPU може да извлича инструкции и данни от външния флаш чип все едно е вътрешна ROM памет (виж XIP). Все пак достъпът е по-бавен спрямо вътрешната памет, но по-бърз спрямо ако интерфейсът е оригиналния SPI.

Протоколът за обмен на данни се осъществява от QSPI модул, подобно на контролера на DRAM [10].

Интерфейс QSPI



Интерфейс QSPI

Команда – операцията, която контролерът на FLASH паметта трябва да изпълни.

Адрес – начален адрес, от който ще се осъществява операцията.

Алтернативен байт (alternate byte, mode byte) – допълнителен байт за XIP-свързани команди (виж сл. слайд). Може да се пропусне, или не. Ако е включен, може да не е само 1. Тогава се изпращат 2, 3 или 4 алтернативни байта [12].

Байтове без значение – използват се за забавяне на комуникацията (при високи скорости). Например при превключване на IO0 ÷ IO4 от изходи на входове.

Даннови байтове – байтовете, които ще се пишат/четат от FLASH. Може да са данни, както и микропроцесорни инструкции.

Интерфейс QSPI

XIP (E**x**ecute **I**n **P**lace) – метод, чрез който външна флаш памет се внедрява в адресното поле на μPU. Тогава към паметта се изпращат XIP команди, които са по-бързи от обикновените [12].

XIP влизане – подава се команда за запис/четене с включен алтернативен байт. След деактивиране на !SS, флаш чипът остава в XIP режим. При следващото активиране на !SS, фазата за “команда” се прескача и се отива директно към фаза “адресиране”. Сега операцията, която се извършва, е същата като предходната.

Интерфейс QSPI

XIP изпълнение – алтернативният байт винаги се предава 0xAX (X – без значение) или 0xA5 по време на изпълнение на XIP команда.

XIP излизане – алтернативният байт се предава различен от 0xAX или 0xA5 по време на изпълнение на XIP команда. Следващото активиране на !SS ще започне обикновен трансфер (с фаза, в която се предава команда).

Интерфейс QSPI

QSPI може да се използва с предаване на данните и по двата фронта на тактовия сигнал.

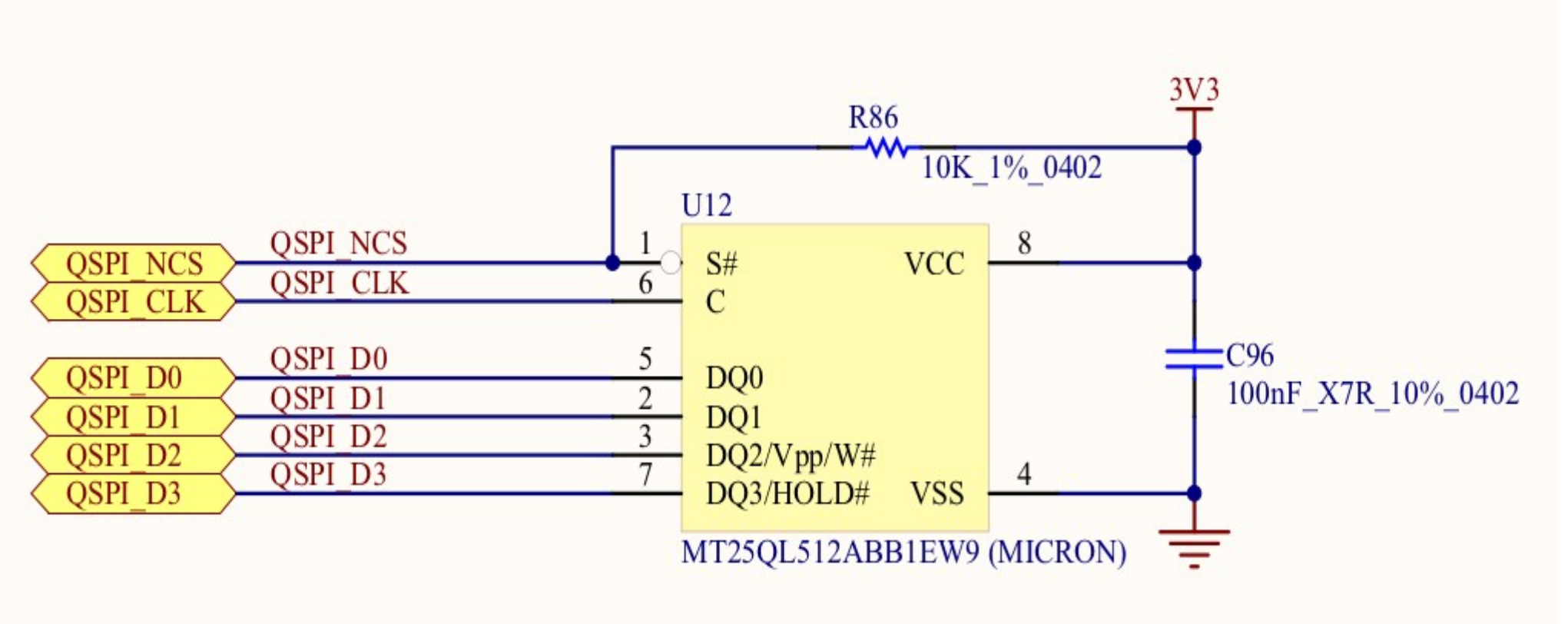
DDR (**D**ouble **D**ata **R**ate) – точно както при DRAM.

Тогава скоростта се увеличава:

*2 пъти спрямо оригиналния QSPI
*8 пъти спрямо оригиналния SPI

Интерфейс QSPI

Пример – връзка на STM32F769 и MT25QL512. Забележете, че паметта може да работи с DSPI, тогава другите два сигнала стават !W (write protect) и !HOLD.



Интерфейс QSPI

Издършващият резистор е, за да се подсигури едновременното покачване на нивото на !CS пина със захранващото напрежение при пускане на захранването.

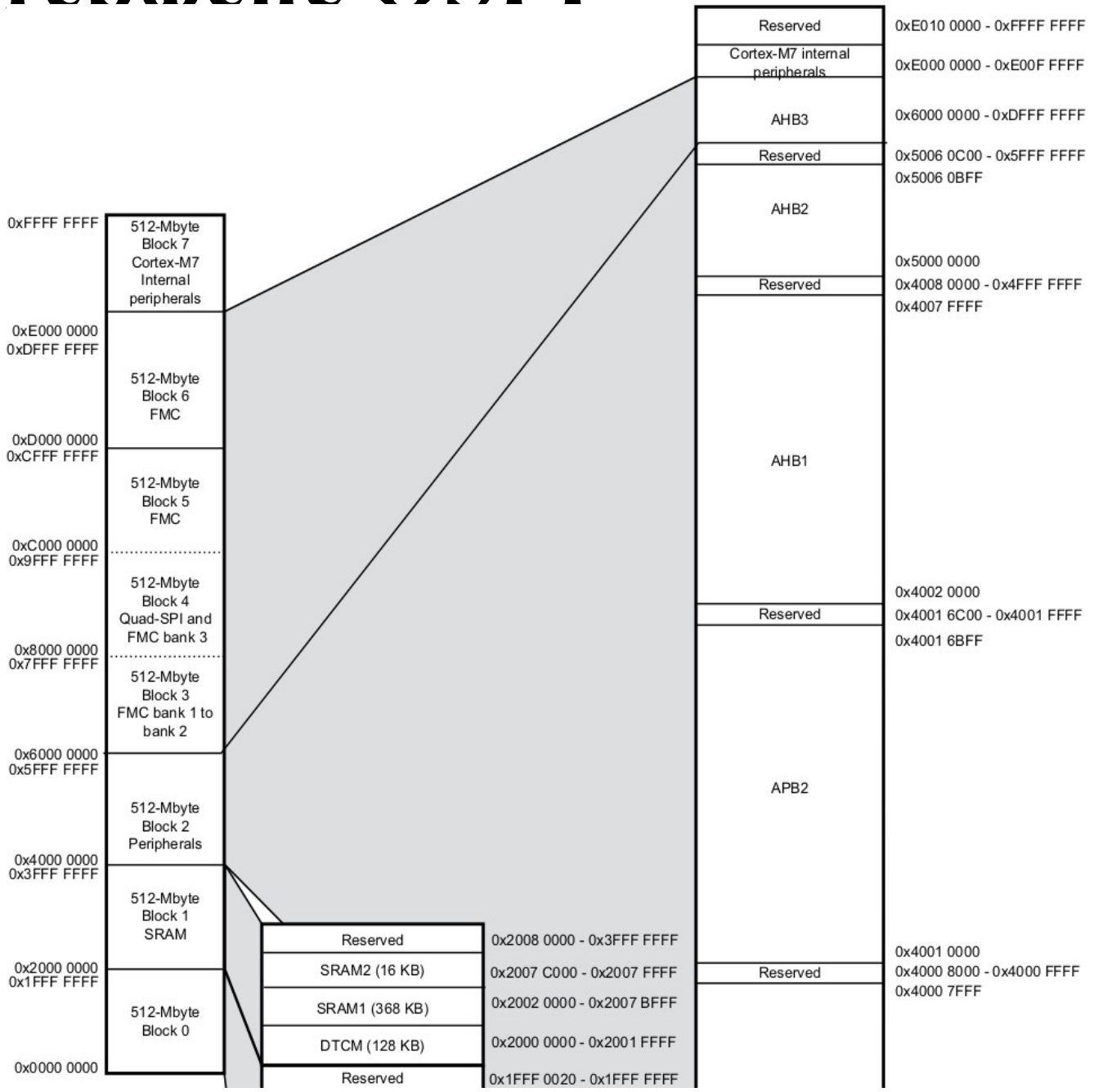
Целта е да се избегне приемането на грешни команди при стартиране на системата[11].

Интерфейс OSPI

Карта на паметта на STM32F769I. Регионът с адреси

8000.0000 ÷ 9FFF.FFFF

е запазен за външни QSPI флаш памети.



Интерфейс I2C

I^2C интерфейсът (IIC – **I**nter **I**ntegrated **C**ircuit) е създаден от фирмата Philips и се използва, за комуникация между ИС в рамките на една вградена система. Това е сериен, синхронен, напрежителен, несиметричен, еднополярен интерфейс.

Предаването на информацията се осъществява посредством сигналите:

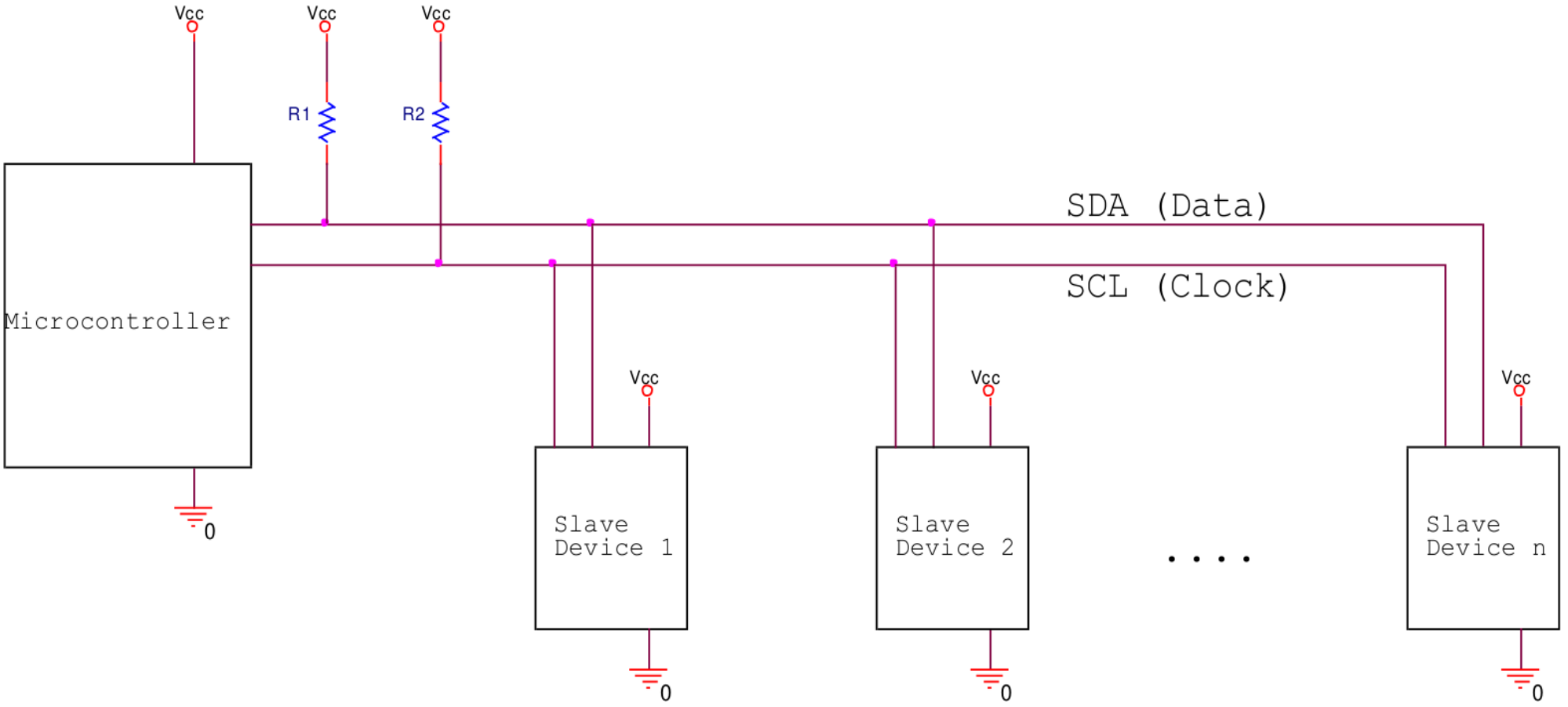
***SDA** – данните, предавани серийно по този проводник.

***SCL** – тактовият сигнал, по който е синхронизирано изпращането, бит по бит, на данните.

Интерфейс I2C

Към двата проводника са свързани издърпващи (pull-up) резистори към захранването на системата. Те са необходими, защото предаването на информация е двупосочно от вида **полу-дуплекс**. Това означава, че микроконтролерът ще използва изводът си, свързан към SDA, веднъж като вход и веднъж като изход. Аналогично – подчиненото устройство ще използва SDA като вход, когато иска да приеме информация и след това като изход, когато иска да предаде информация.

Интерфейс I2C



Интерфейс I2C

Към двата проводника са свързани издърпващи (pull-up) резистори към захранването на системата. Те са необходими, защото предаването на информация е двупосочно от вида **полу-дуплекс**. Това означава, че микроконтролерът ще използва изводът си, свързан към SDA, веднъж като вход и веднъж като изход. Аналогично – подчиненото устройство ще използва SDA като вход, когато иска да приеме информация и след това като изход, когато иска да предаде информация.

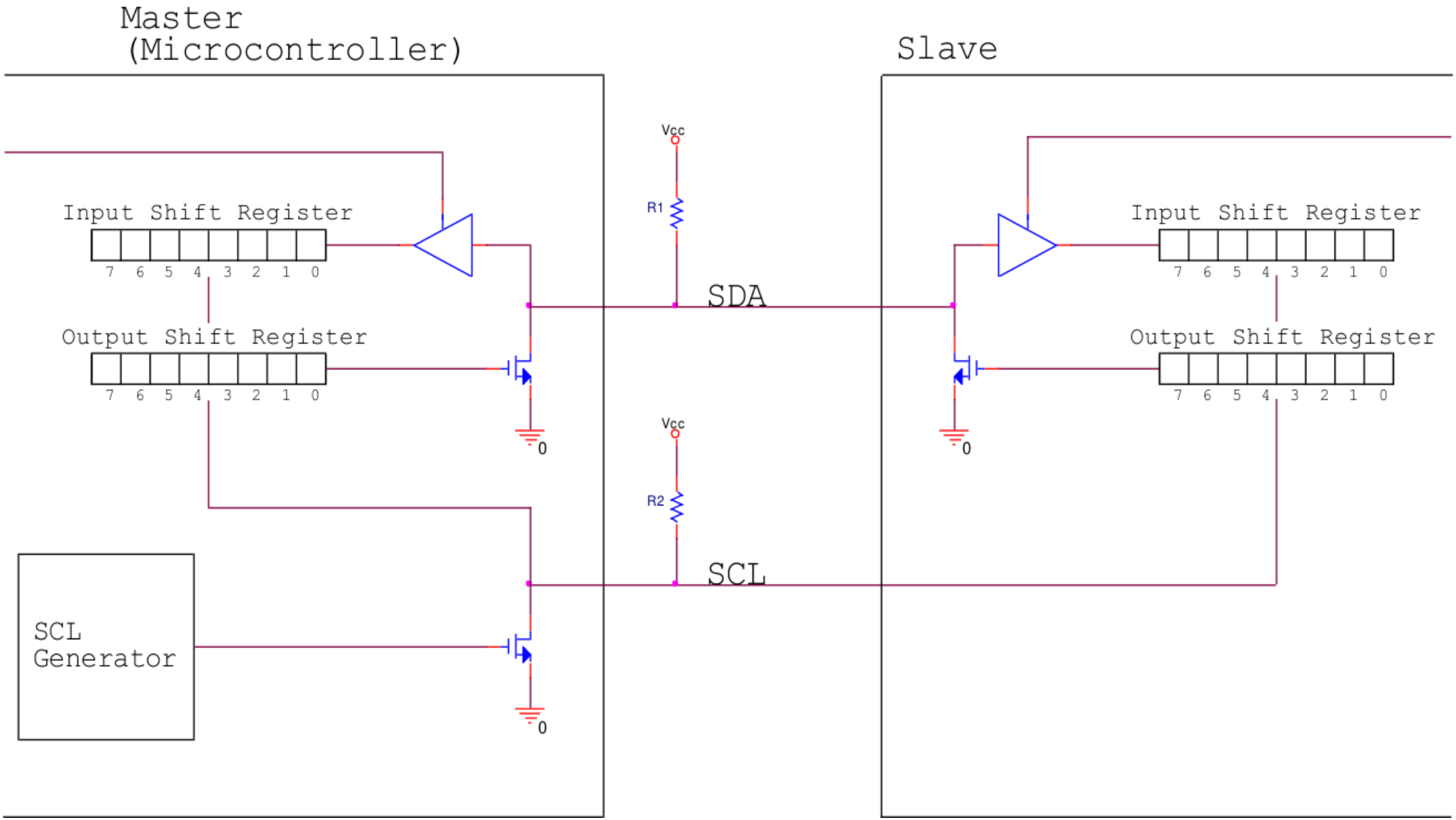
Интерфейс I2C

Тази конфигурация води до една теоретична конфликтна ситуация, в която ако и двата чипа са установили изводите си SDA като изходи и единият е в логическа 1, а другият – в логическа 0, то ще се получи късо съединение между захранващия извод Vcc и масата GND.

За да се избегне тази възможност **за SDA се използват изходи с отворен дрейн** (или колектор), които изискват режимен издърпващ резистор.

SCL също включва такъв резистор, защото протокола I^2C дефинира ситуация, в която някой от чиповете задържа тактовия сигнал в логическа 0.

Интерфейс I2C



Интерфейс I2C

Аналогични на I^2C са интерфейсите:

***SMBus** – създаден от Intel през 1995. Дефинира по-строго изискванията за ниво и времеви параметри (timings) на сигнала.

***I^2S** – използва се за предаване на цифров звуков сигнал. Съдържа един допълнителен проводник (Word Select), указващ изпращаният байт на кой канал принадлежи (при стерео предаване – ляв или десен).

I^2C интерфейса се характеризира със следните скорости на предаване:

***Оригинална версия** от 1982 – **100 kbit/s**.

***Версия 1** (1992) – **400 kbit/s** (Fast mode).

*Версия 2 (1998) – 3.4 Mbit/s (High-speed mode).

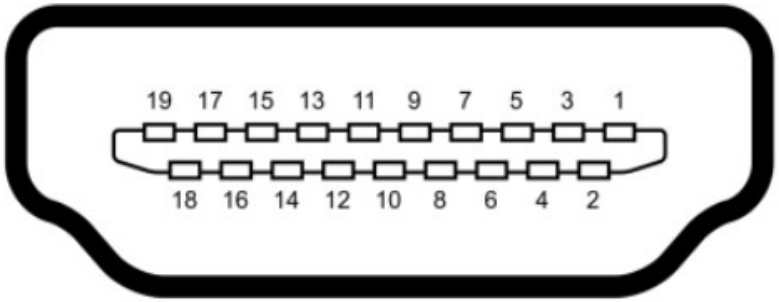
*Версия 3 (2007) – 1 Mbit/s (Fast mode plus).

*Версия 4 (2012) – 5 Mbit/s (Ultra Fast mode).

Интерфейс I2C

Въпреки че този интерфейс се използва главно за комуникация между ИС в рамките на една вградена система, то не липсват примери за приложения и в комуникацията между две отделни устройства.

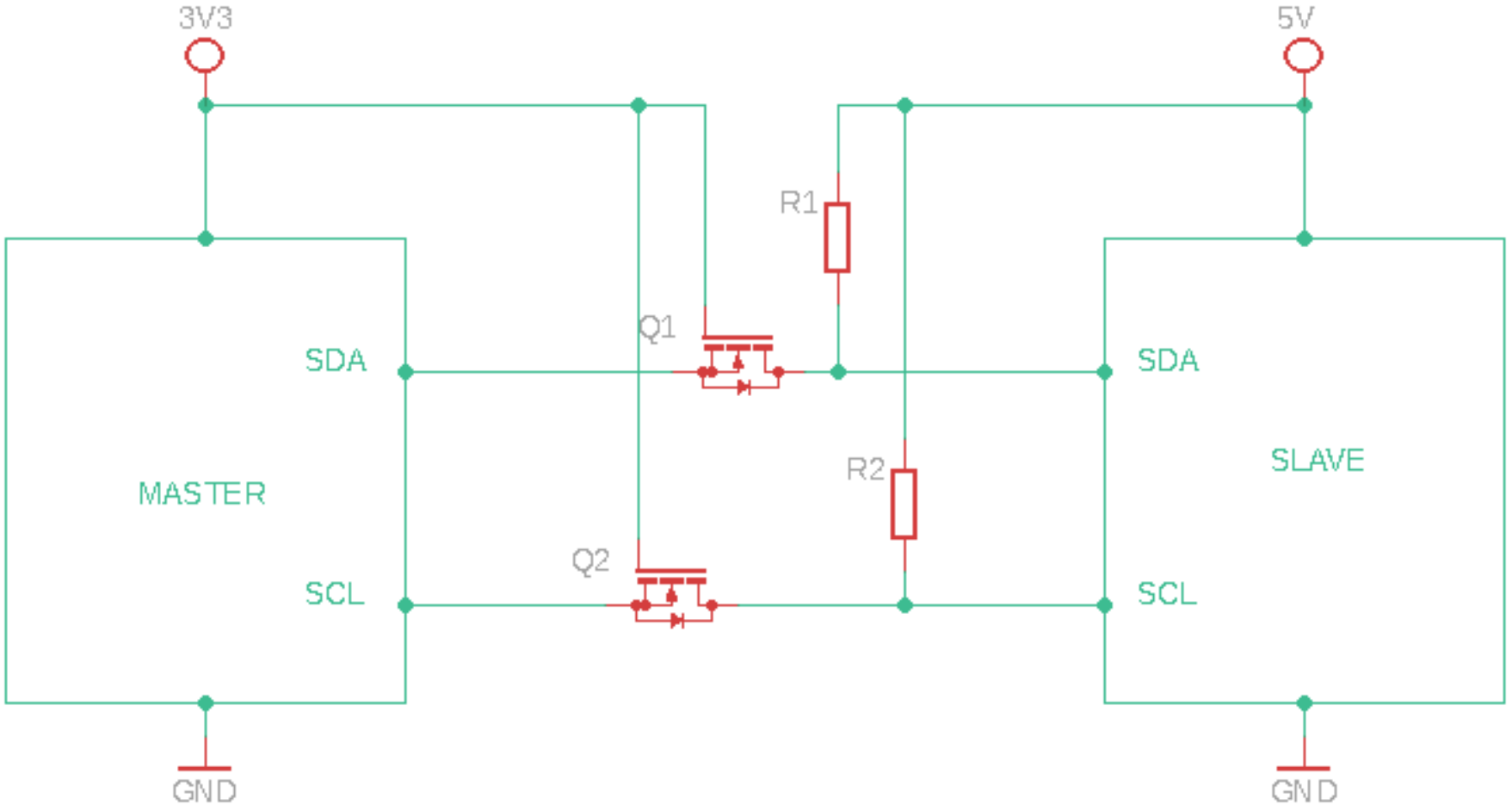
Пример - всеки персонален компютър използва I 2 C като част от HDMI интерфейса фиг. 6.3. С негова помощ се „опознават" поддържаните видео формати от изобразяващото устройство (монитор, телевизор, прожектор и други).



Фиг. 6.3-HDMI куплунг. I^2C на извод 15 – SCL и извод 16 – SDA.

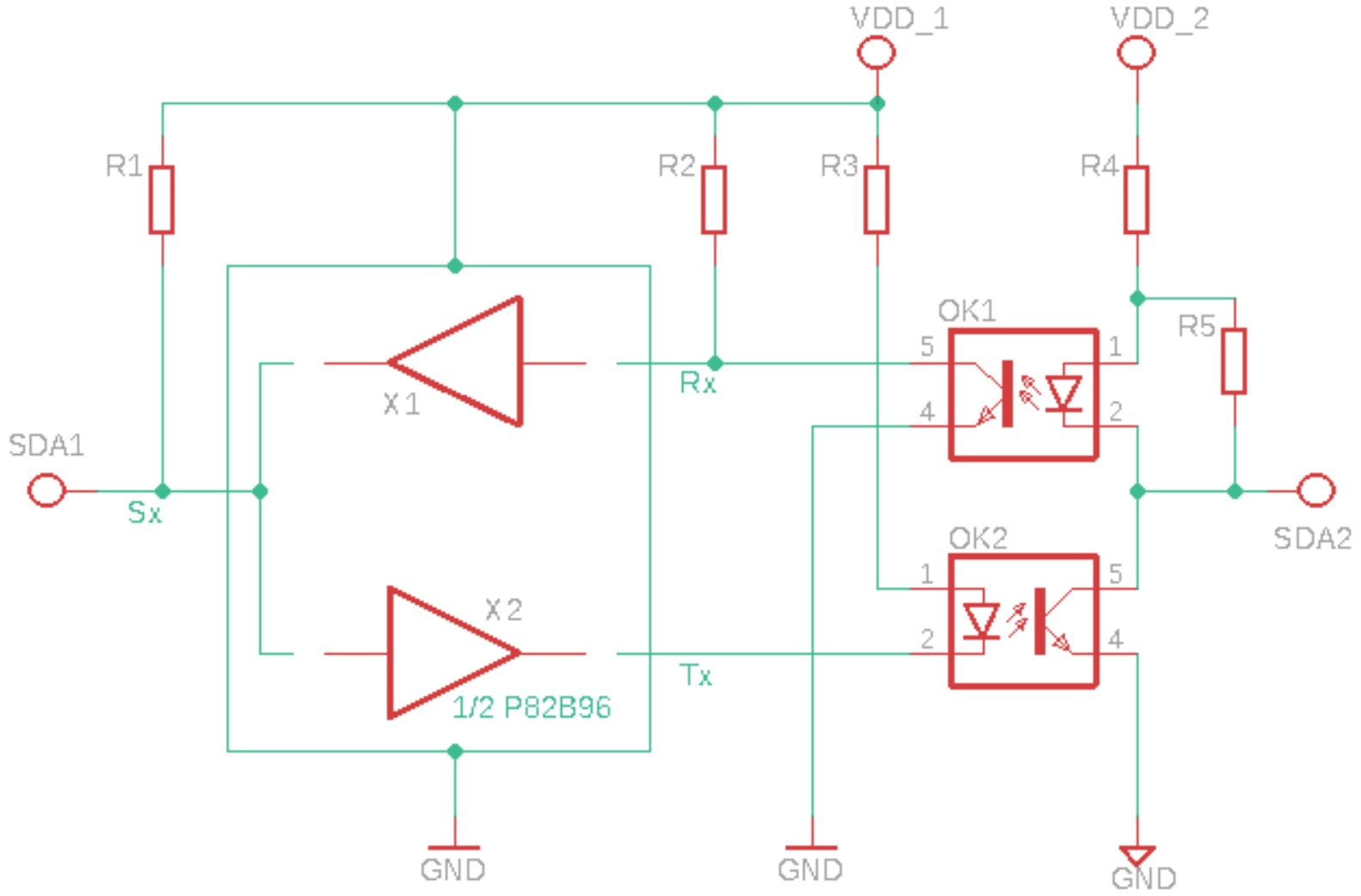
Интерфейс I2C

Транслирането на нивата между две схеми с различни захранващи напрежения трябва да е **двупосочно** и може да стане по начина, показан в лекцията “Паралелни интерфейси”.



Интерфейс I2C

Галванично разделяне също е възможно. Показаната схема позволява работа до около 5 kHz с 4N36 [9]. (R5 е bootstrap резистор, за да се отпуши транзисторът на ОК2, когато диодът на ОК



Интерфейс I2C

Как схемата от миналия слайд не “увисва” (latch up) само в едно състояние, подобно на bus keeper-а?

Интерфейс I2C

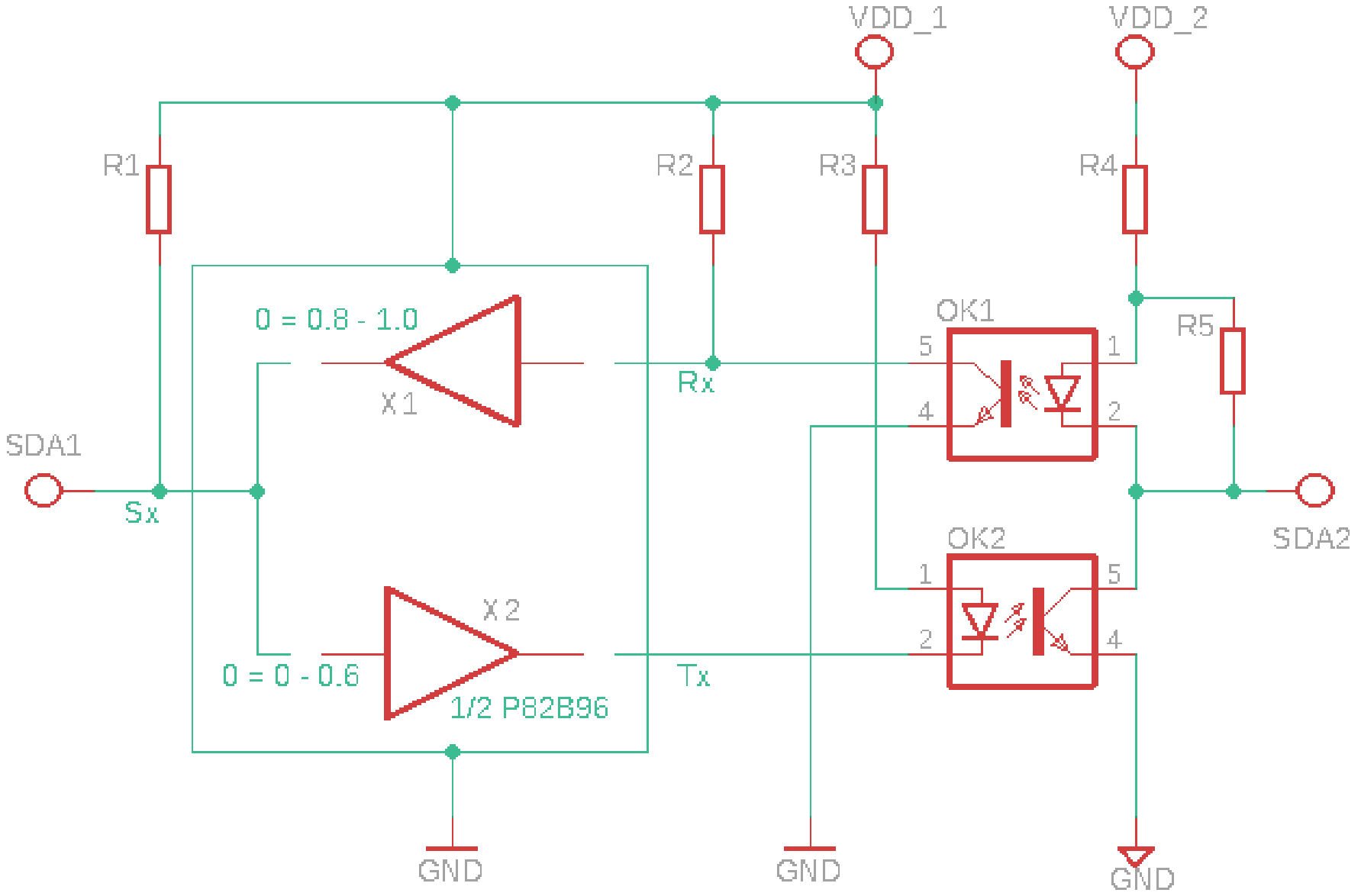
Буферите са специални – изхода на единия от тях е проектиран да имат **нестандартна логическа нула**, която да може да се приеме от I2C устройството, но да не може да се приеме от входа на другия буфер. Така буферите се “разминават” и никога не се задейства положителна обратна връзка.

При VDD=5V

==========

Sx_out(0)=0.8 ÷ 1.0V (по I^2C 0 ÷ 1.5V се счита за нула)
Sx_in(0) = 0 ÷ 0.6 V (Tx = 0V при Sx ≤ 0.6V)
Sx_in(1) = 0.7 ÷ 5 V (Tx = 5V при Sx ≥ 0.7V)
=> При Rx=0, Tx=1

Интерфейс I2C



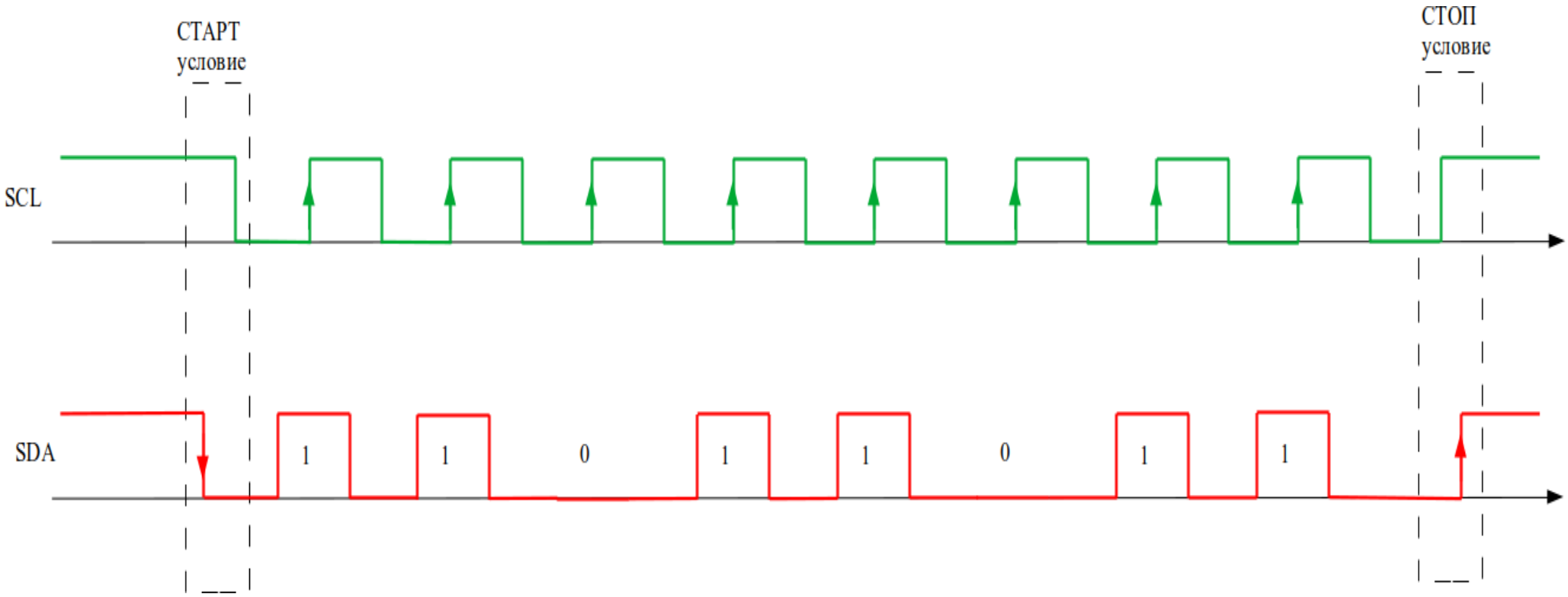
Интерфейс I2C

I^2C комуникацията се осъществява по стандартизиран **протокол**.

Обменът на данни започва с **условие СТАРТ,** продължава с трансфер на данни и завършва с **условие СТОП**.

Условията старт и стоп са специални събития, реализирани чрез комбинация на логически нива и фронтове на SCL и DATA линиите, които са уникални и се различават от данните. На следващия са показани едно старт и стоп условие.

Интерфейс I2C



Интерфейс I2C

Старт условие се генерира при спадащ фронт на SDA линията, докато SCL линията е във високо ниво.

Стоп условие се генерира при нарастващ фронт на SDA линията, докато SCL линията е във високо ниво.

Между старт и стоп условията се изпращат **потребителските данни**. Всеки бит от тях трябва да запази своето състояние, докато SCL е във високо ниво (в противен случай ще се генерира лъжливо старт или стоп условие и трансферът ще се наруши), т.е. трансферът на данни се осъществява по нарастващ фронт.

Интерфейс I2C

Форматът на данните при **запис** от микроконтролер в подчинена ИС е показан на следващата фигура.

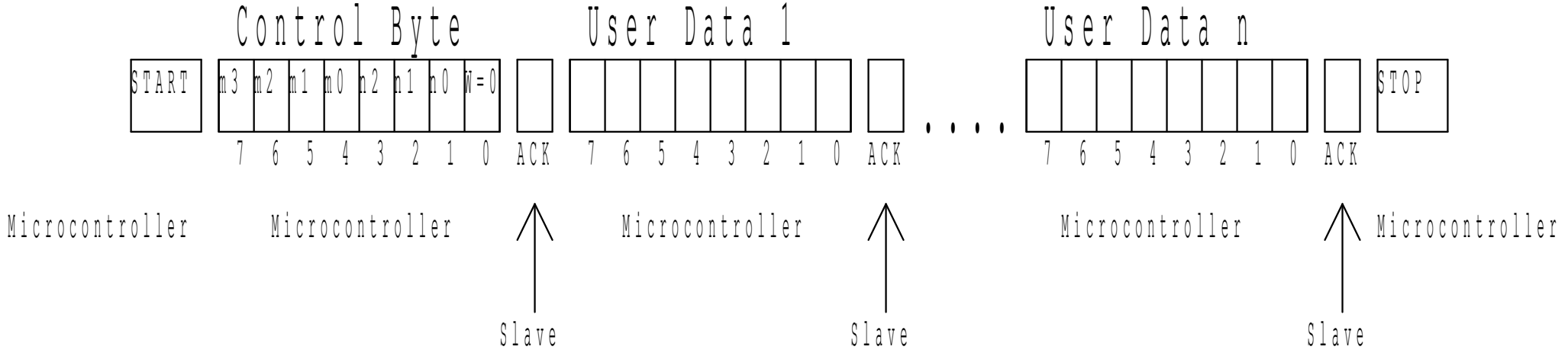
Микроконтролерът генерира СТАРТ условие, след което изпраща **контролен байт**. Този байт достига до всички подчинени устройства. Старшата тетрада (битове m0 ÷ m3) съдържа контролен код, който е различен и фиксиран за различните подчинени чипове – трябва да се провери datasheet-а им.

Интерфейс I2C

Следват три бита (n0 ÷ n2), които указват младшата част на **7-битовия адрес** на подчиненото устройство и понякога могат да бъдат променяни. Благодарение на тях към една I²C шина могат да се свържат повече от един чип от един и същи вид. Последният бит от контролния байт е четене/запис (R/!W) и при **запис трябва да е 0**. Ако на I²C шината има устройство със зададения адрес (битове m0 ÷ m3 + n0 ÷ n2), то трябва да отговори с изпращане на един бит, наречен **ACK** (acknowledge) или още - потвърждение. Неговото ниво е **логическа 0**. След това се изпращат потребителските данни User Data 1 ... User Data n и най-накрая се генерира условие стоп, с което трансферът приключва.

Интерфейс I2C

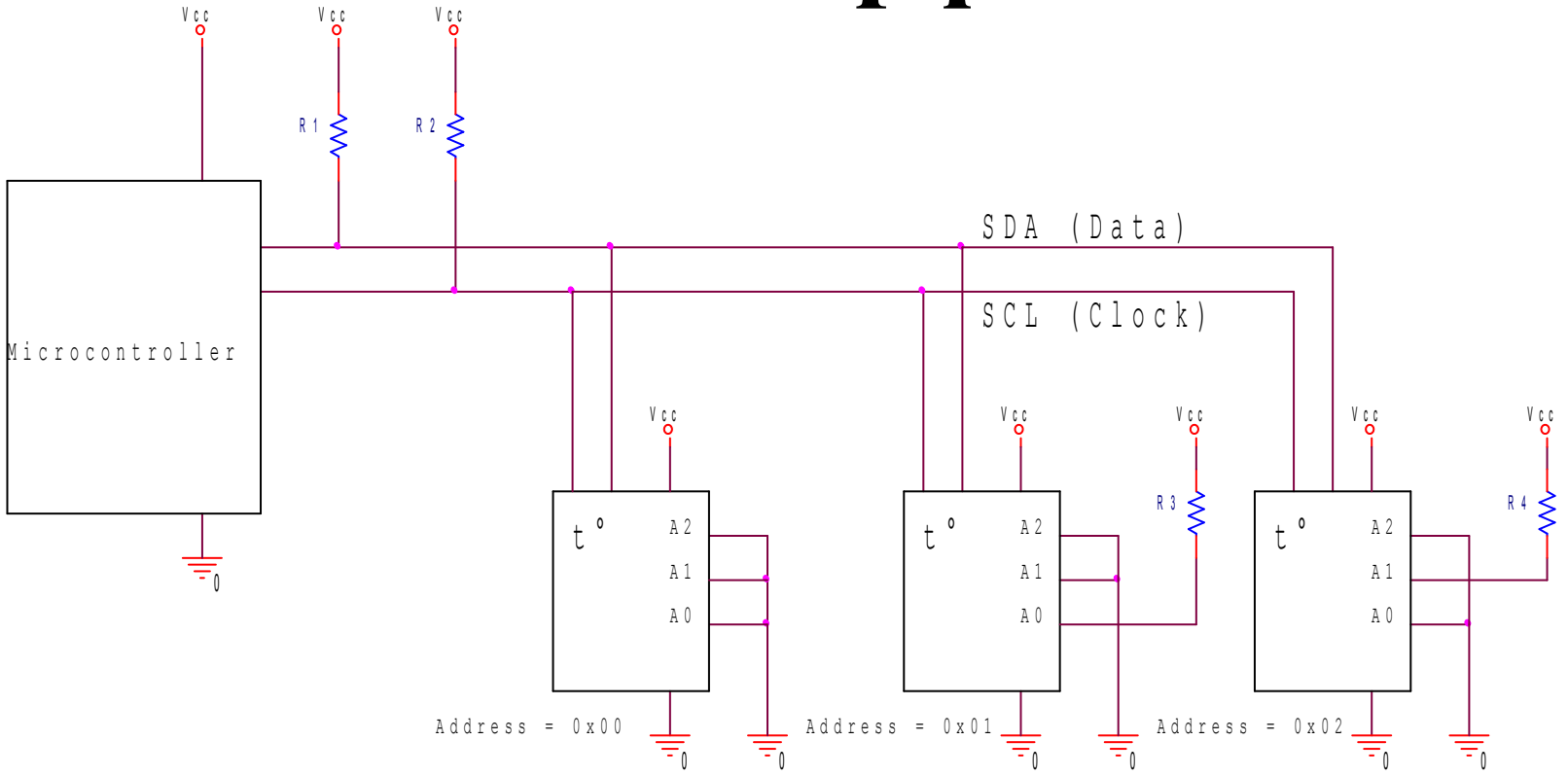
Запис на данни от главно в подчинено устройство.

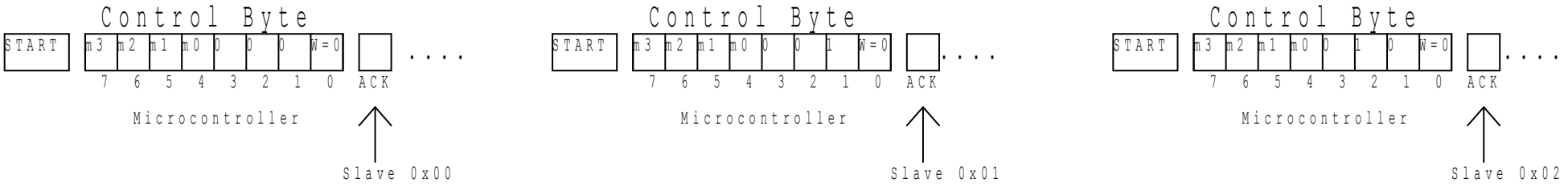


Интерфейс I2C

На фигурата на следващия слайд е показано свързване на три еднакви температурни датчика, чиито адреси са зададени хардуерно с изводи 1, 2 и 3 (рефлектират върху битове n0 ÷ n2).

Интерфейс I2C





Интерфейс I2C

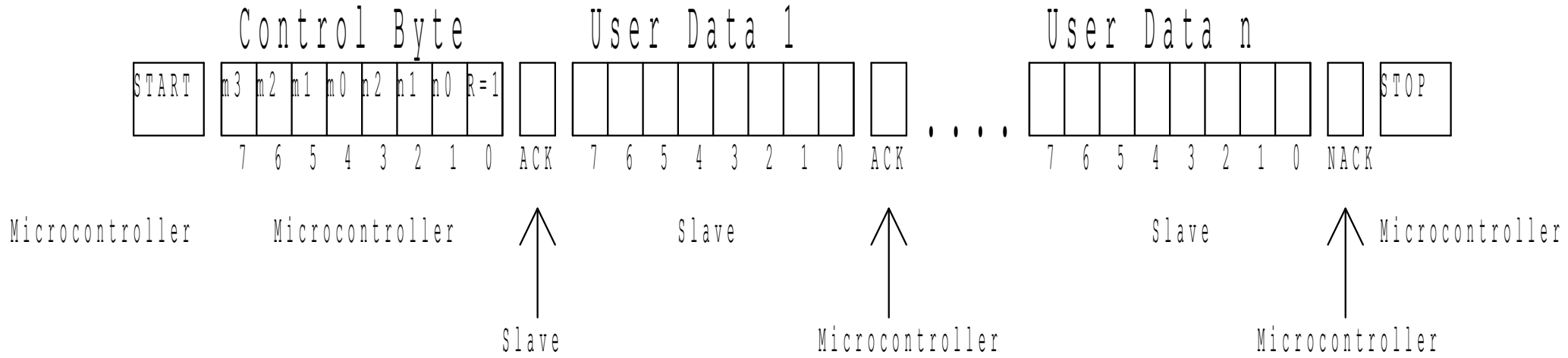
Трансферът на данните от подчинена ИС към микроконтролера се определя като **четене** и форматът е показан на следващата фигура.

Да се обърне внимание на предавателя и приемника под всеки байт!

Ако микроконтролерът иска да спре приемането на данни, той трябва да генерира **NACK** бит (No Acknowledge, **логическа 1**) и след това условие СТОП.

Интерфейс I2C

Четене на данни от подчинено устройство.



Интерфейс I2C

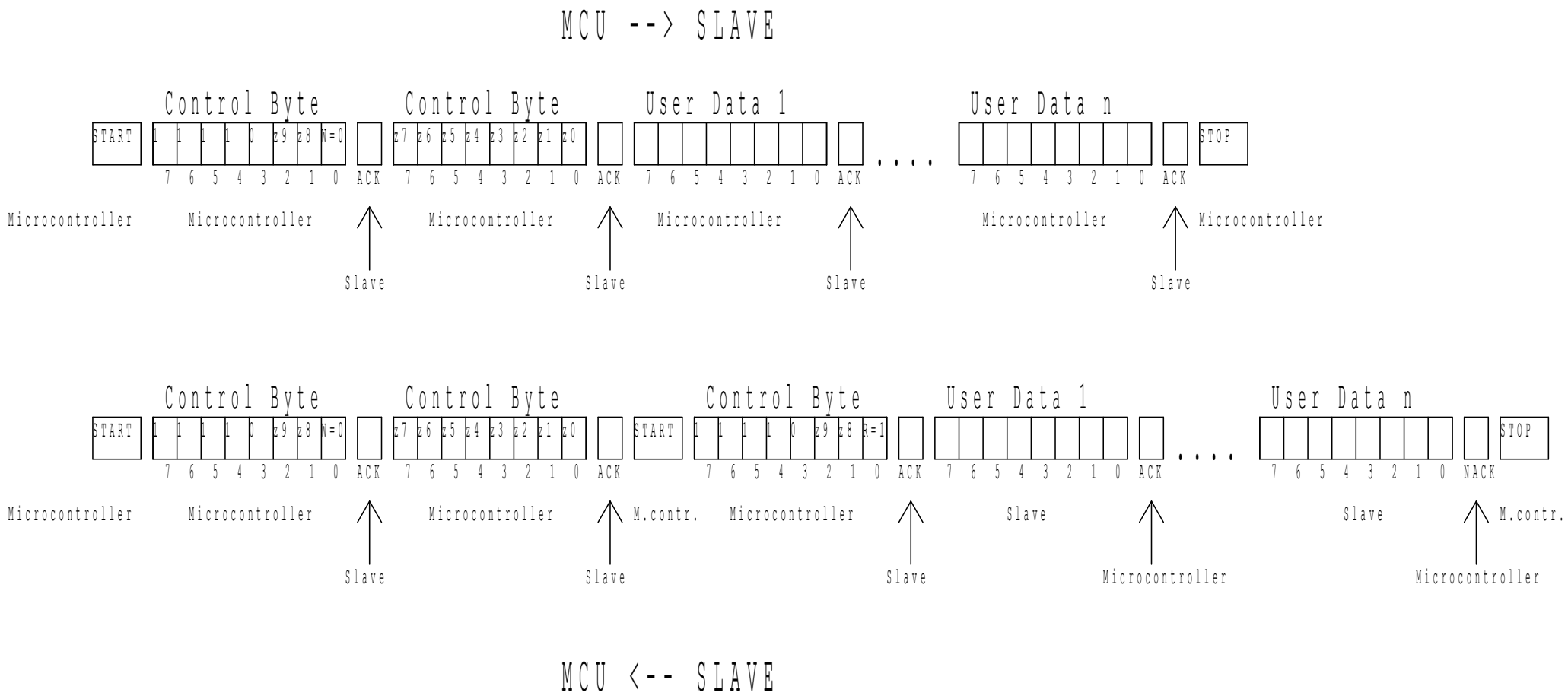
Отделените седем бита m0 ÷ m3 и n0 ÷ n2 позволяват на една I^2C шина да се свържат до $2^7 = 128$ чипа (минус един броудкаст адрес и няколко за бъдещо ползване).

Версия 1 на протокола дефинира някои леки промени в протокола, които позволяват адресиране на $2^{10} = 1024$ (т.е. 10-битов адрес) чипа.

То е показано на следващия слайд. Новото тук е фиксираното число **11110** в контролния байт. То указва, че ще се използва **10-битово адресиране**. Следват битове z9 ÷ z0, които са 10-битовия адрес на подчиненото устройство. Да се обърне внимание на двойното изпращане на условие СТАРТ и редуването на W=0 и R=1 при четене (MCU <--> SLAVE)!

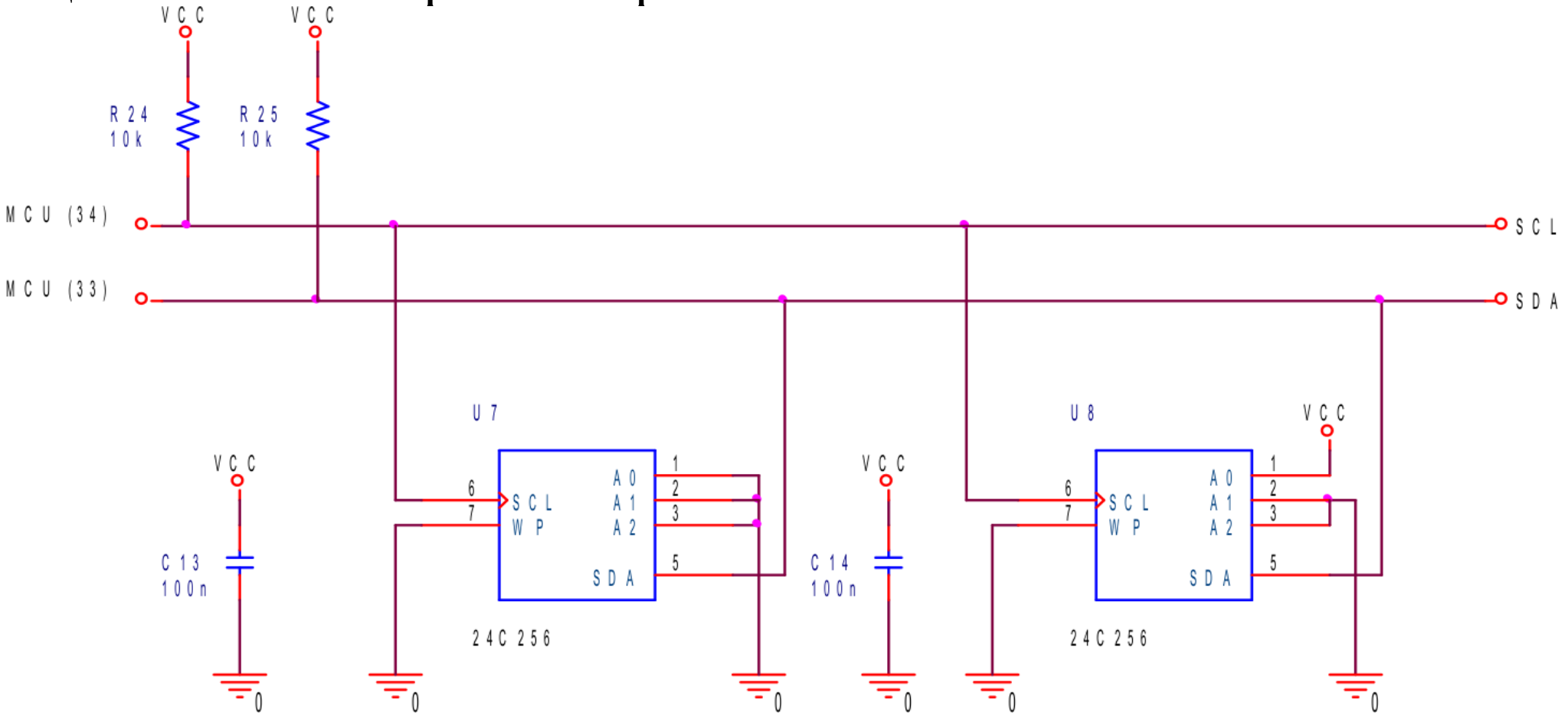
Интерфейс I2C

Четене и запис при 10-битово адресиране.



Интерфейс I2C

Пример – на схемата по-долу е показано свързване на 2 чипа EEPROM памет от един и същи вид – AT24C256 (капацитет: 32768 8-битови думи). Микроконтролерът е PIC18F4550 и цялата система работи при VCC=5V.



Интерфейс I2C

Запис на байт в AT24C256. Забележете вътрешния адрес (first + second word address), който за главното устройство са просто два даннови байта, но за паметта са вътрешен 16-битов адрес.

Figure 7. Device Address

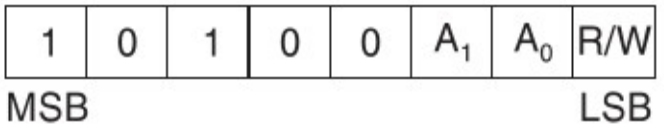
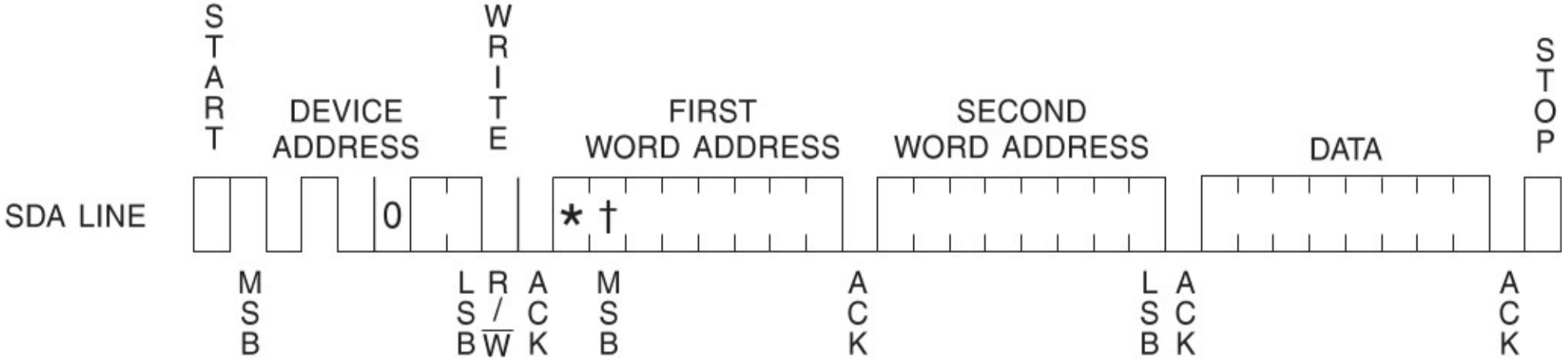


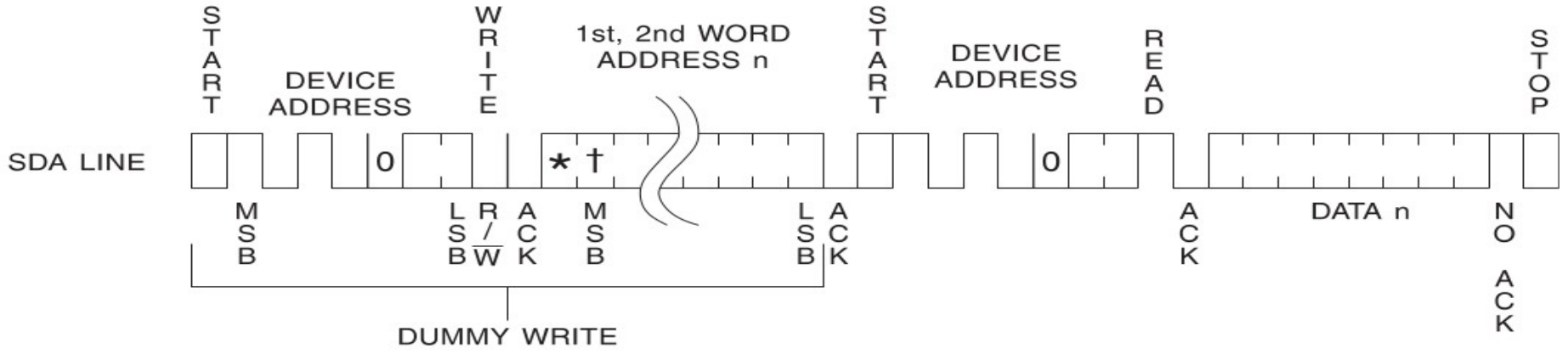
Figure 8. Byte Write



Интерфейс I2C

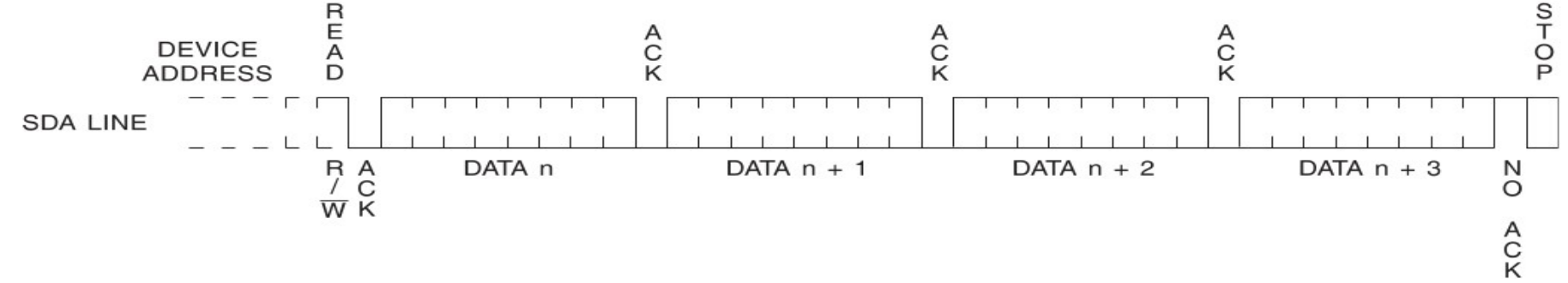
Четене на байт/ове от AT24C256. Забележете как се задава адрес (вътрешен адресен брояч), който бива четен след второто START условие.

Figure 11. Random Read



Notes: (* = DON'T CARE bit)
(† = DON'T CARE bit for the 128K)

Figure 12. Sequential Read



Интерфейс Wi-Fi

Wi-Fi интерфейсът (**Wi**reless **Fi**delity, IEEE 802.11) е асинхронен радио интерфейс, осигуряващ връзка на устройства в мрежа.

Този интерфейс се използва от OSI модела и дефинира последните му два слоя – MAC и PHY (както е при Ethernet).

Използват се 14 носещи честоти (=канала), всяка от които е отместена от другата с 5 MHz. Всеки канал използва 22-мегахерцова честотна лента.

Интерфейс Wi-Fi

IEEE 802.11b е първият вариант на безжичния интерфейс и осигурява скорост на обмен до 11 Mbps в радиус до 30 метра. Честотният диапазон е 2.4 ÷ 2.5 GHz (съвпада с Bluetooth, но чрез софтуерни алгоритми се избират различни канали и не би трябвало да се затрудни предаването на данни във Wi-Fi мрежата).

IEEE 802.11a работи в честотния обхват 5.15 – 5.35 GHz и е по-незасегнат от смущения спрямо 802.11b. Скоростта на обмен е завишена на 54 Mbps.

IEEE 802.11g работи в диапазона 2.4 ÷ 2.5 GHz, но скоростта на обмен е по-висока спрямо 802.11b – 54 Mbps.

IEEE 802.11n може да работи в честотните диапазони около 2.4 и 5 GHz, а скоростта е 54 ÷ 600 Mbps.

Интерфейс Wi-Fi

LAN (**L**ocal **A**rea **N**etwork) - мрежа, обхващаща различен брой компютри, разположени в област не по-голяма от 10 km. Броят на компютрите варира много и зависи от мрежовата структура и използваните кабели. LAN може да се нарече мрежа от три компютъра в три съседни блока, както и 1000 компютъра в един квартал [13].

WLAN (**W**ireless **L**ocal **A**rea **N**etwork) - LAN мрежа, използваща ефира за преносна среда. В употреба влизат стандартите 802.11.

WAN (**W**ide **A**rea **N**etwork) – международни мрежи, обхващащи голяма географска област. Наи-известният пример е **Интернет**. Друг пример са корпоративните мрежи (Интранет) на фирми, имащи клонове по цял свят и които са си изградили WAN за спомагане дейността на фирмата. Такива мрежи осигуряват обмен от около 1 ÷ 6Mbit/s, което е значително по-малко от скоростите, използвани в LAN.

Интерфейс Wi-Fi

Вградени системи, свързани в Интернет може да се нарекат IoT устройства.

IoT (**I**nternet **o**f **T**hings) – съвременна тенденция за свързване на различни електронни уреди с микропроцесори в глобалната мрежа Интернет. Това са перални, хладилници, кафеварки, ключове за осветление, метеорологични станции, уреди работещи в полето – вградени системи за напояване в земеделието, електронно отчитане на парно и водомери, и т.н.

Интерфейс Wi-Fi

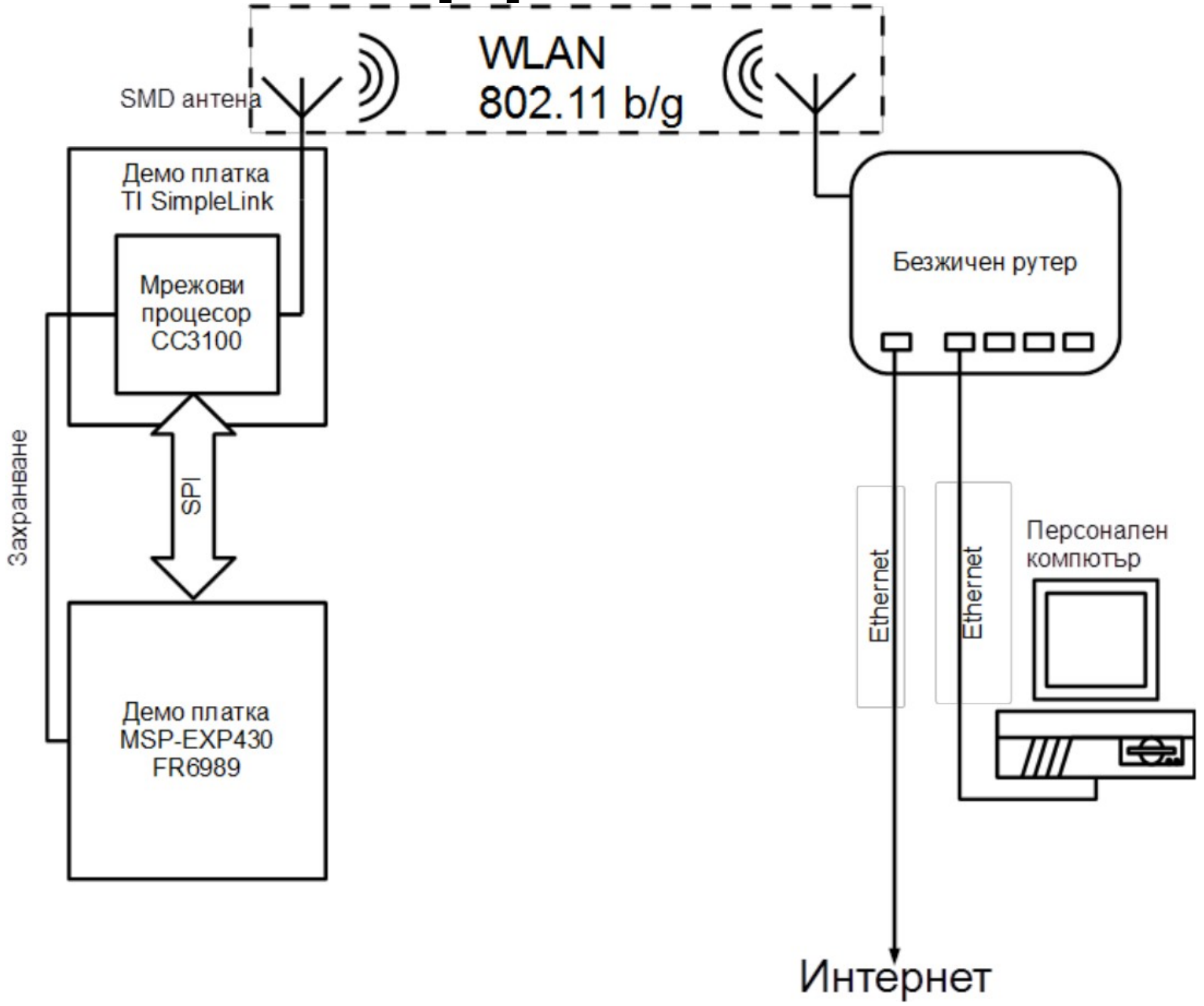
За да се свърже една вградена система в Интернет, може да се подходи по два начина:

*към съществуващ главен μCU да се добави подчинен **мрежови процесор**, който да направи връзката. Данните се предават по UART, SPI, I2C и други към главния μCU.

*главният μCU да има вграден Wi-Fi модул (включително и радиото).

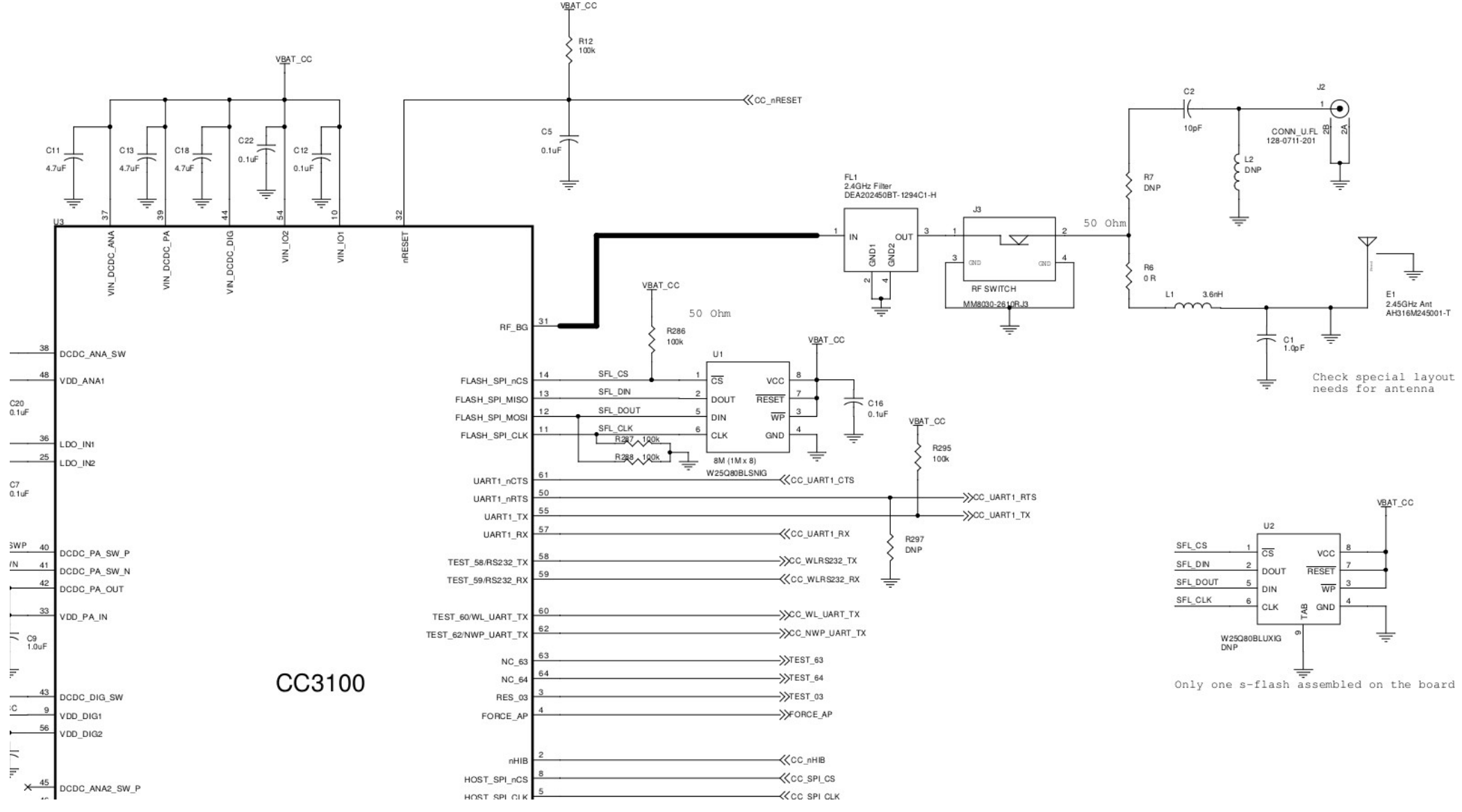
Мрежови процесор – микроконтролер с вградени RAM, ROM и Wi-Fi радио, който има една основна функция – да поддържа комуникация в Интернет.
Пример – мрежови процесори са CC3100 на Texas Instruments и ESP8266 на Espressif.

Интерфейс Wi-Fi



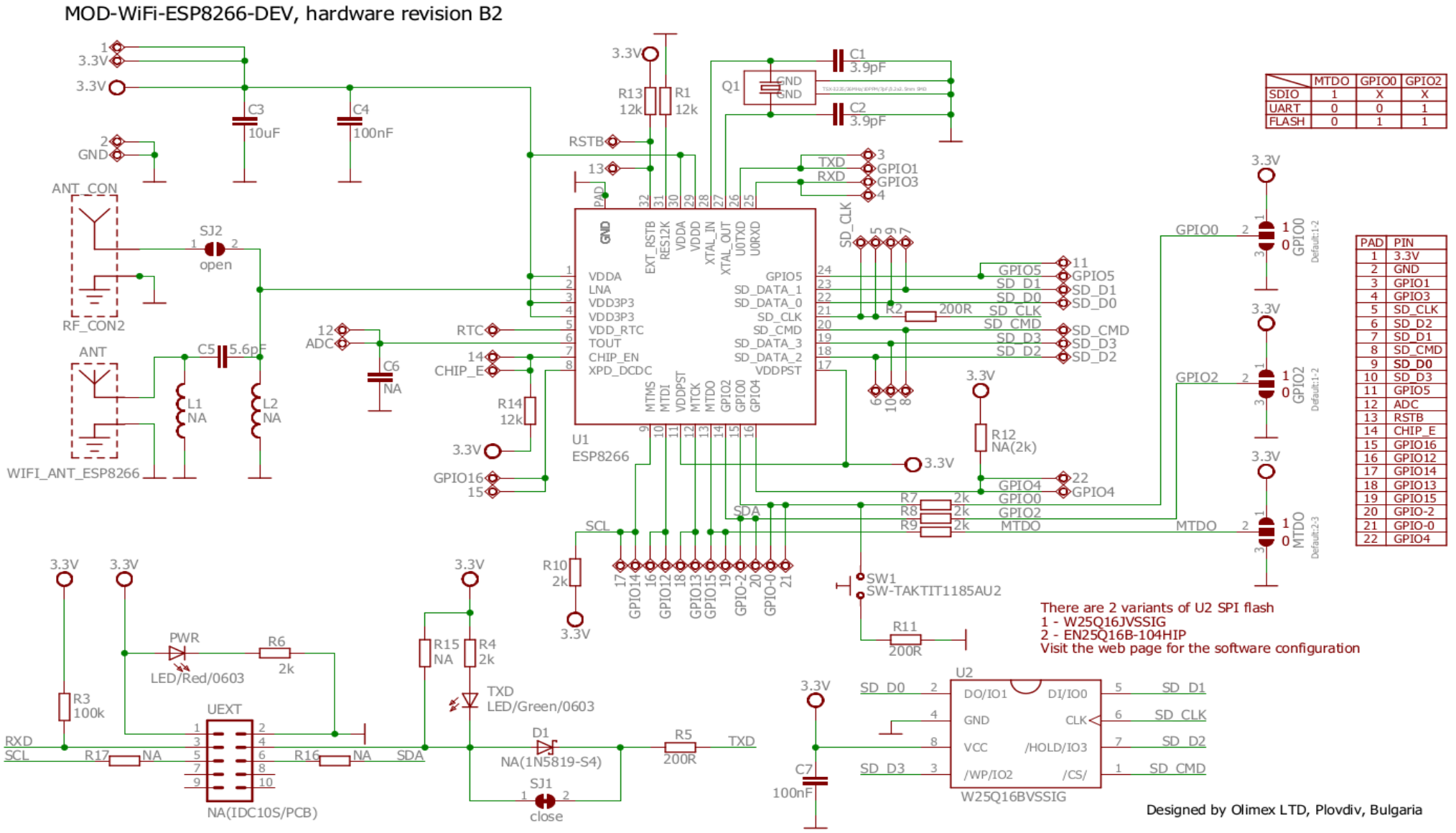
Интерфейс Wi-Fi

Част от схемата на свързване на CC3100. Забележете външната SPI флаш памет и Wi-Fi антена.



Интерфейс Wi-Fi

Български Wi-Fi модул (от Olimex), базиран на ESP8266. Забележете външната SPI флаш памет и Wi-Fi антена.



	MTDO	GPIO0	GPIO2
SDIO	1	X	X
UART	0	0	1
FLASH	0	1	1

PAD	PIN
1	3.3V
2	GND
3	GPIO1
4	GPIO3
5	SD_CLK
6	SD_D2
7	SD_D1
8	SD_CMD
9	SD_D0
10	SD_D3
11	GPIO5
12	ADC
13	RSTB
14	CHIP_E
15	GPIO16
16	GPIO12
17	GPIO14
18	GPIO13
19	GPIO15
20	GPIO-2
21	GPIO-0
22	GPIO4

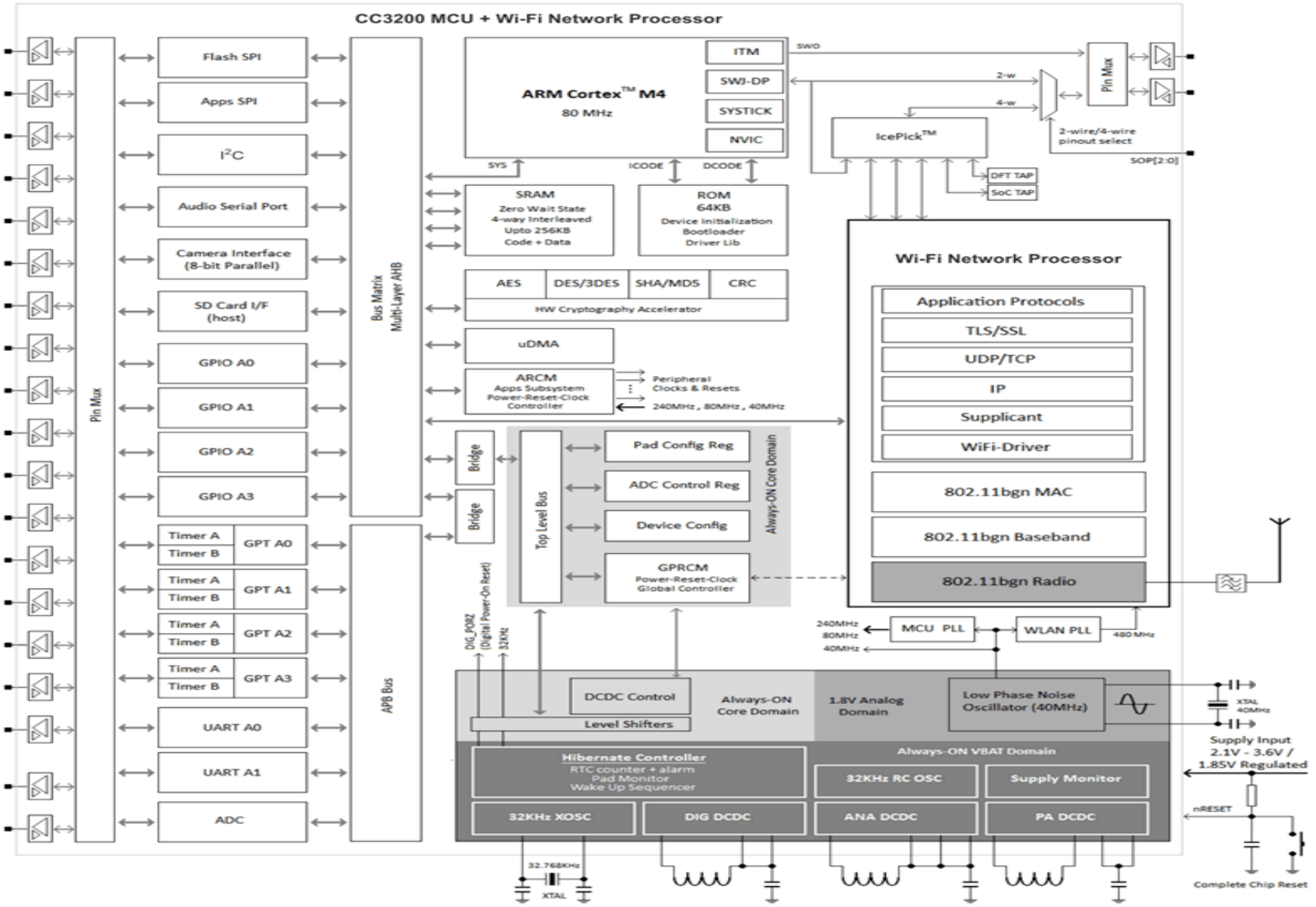
Интерфейс Wi-Fi

ESP8266 използва операционна система за реално време.

```
void wifi_init_sta(void){
        wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();

        wifi_deinit_sta();

        tcpip_adapter_init();
        esp_event_loop_init(event_handler, NULL);
        esp_wifi_init(&cfg);
        esp_wifi_set_mode(WIFI_MODE_STA);
        esp_wifi_set_config(ESP_IF_WIFI_STA, &wub_conf.wifi_config);
        esp_wifi_start();

        xTaskCreate(wub_wifi_server_task, "wub_server_task", 8192, NULL,
WUB_WIFI_TASK_PRIORITY, &wub_server_task_h);

        DEBUGOUT("ssid: %s\n\r", wub_conf.wifi_config.sta.ssid);
        DEBUGOUT("pass: %s\n\r", wub_conf.wifi_config.sta.password);
}
```

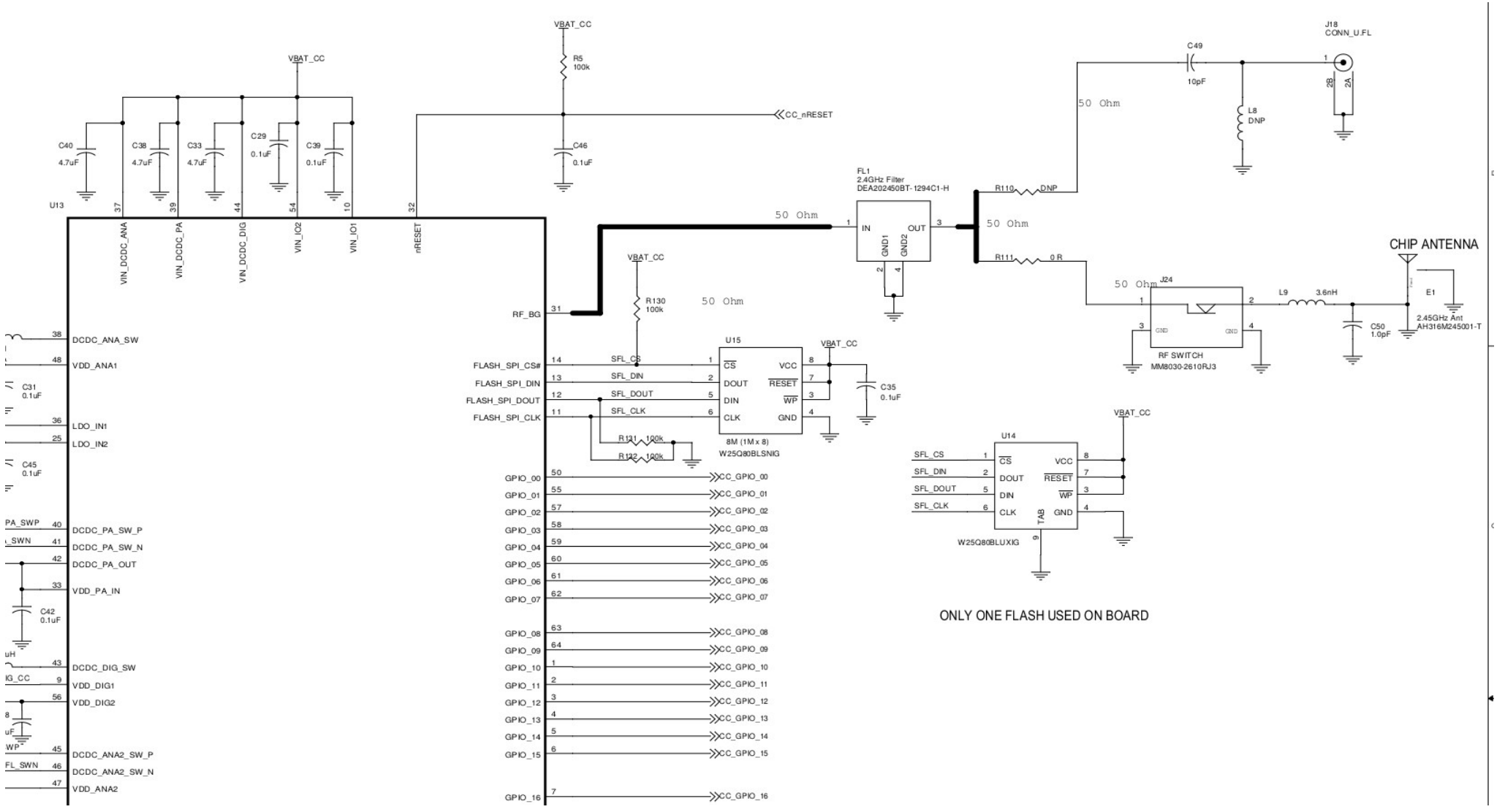
Интерфейс Wi-Fi

CC3200 на Texas Instruments е микроконтролер + мрежови процесор (поради сложната вътрешна архитектура се наричат още SoC).



Интерфейс Wi-Fi

Част от схемата на свързване на CC3200. Забележете външната SPI флаш памет и Wi-Fi антена.



Интерфейс Wi-Fi

Широко разпространен протокол за комуникация в Интернет е TCP/IP.

IP адрес – уникално 32-битово цяло число, присвоено на дадено устройство в Интернет. Такъв адрес е необходим при предаването на информация между две и повече устройства. Изобразява се в десетичен вид, като всеки байт е разделен с точки (например 192.168.1.10, 10.1.1.23, 81.150.230.33 и други).

32-битови адреси се използват във версия номер 4 (**IPv4**) на IP протокола . С 32-битово число може да се адресират до 2^{32} = 4 294 967 296 устройства.
Поради широката употреба на световната мрежа Интернет този брой се оказва недостатъчен, затова в следващата версия адресите са **128-битови** (**IPv6**).

Интерфейс Wi-Fi

Интерфейс Wi-Fi

TCP протокол (**T**ransmission **C**ontrol **P**rotocol) – протоколът осигурява пренасянето на непрекъсната поредица от данни, като за разлика от повечето Интернет протоколи, работи с изграждане на логическа връзка и **гарантиране на доставката на данните**. Софтуерът, реализиращ този протокол е на най-високото ниво на абстракция в TCP/IP комуникацията. Гарантиране предаването на данните означава, че ако някой пакет се изгуби от IP протокола, TCP слоя ще детектира това и ще го изпрати наново.

Интерфейс Wi-Fi

IP протокол (**I**nternet **P**rotocol) – стандартизиран механизъм за предаване на информация в световната мрежа Интернет. Той дефинира комуникация, която се осъществява чрез разделяне на данните на пакети в подателя и събирането им в получателя. Всеки пакет, освен полезните данни, съдържа и IP адреса на подателя/получателя, заедно с различни служебни данни. При този протокол **доставката на информация не се гарантира**, т.е. някои от пакетите могат да бъдат изгубени.

Интерфейс Wi-Fi

DHCP протокол (**D**ynamic **H**ost **C**onfiguration **P**rotocol) – аналогичен на IP протокола, но разликата е, че при първоначалното включване на устройството параметрите на IP протокола (IP адрес, Mask, Gateway и др.) се получават автоматично от DHCP сървър. Това означава, че потребителят ще получи автоматично, без негова намеса, всички необходими настройки, за да се свърже към Интернет. Характерното за този протокол е, че при всяко стартиране на компютъра си, потребителят получава IP адрес, който може да не съвпада с IP адреса от предишна сесия (предишното включване) на компютъра. Нещо повече – този IP адрес може да се промени по време на настоящата сесия.

Интерфейс Wi-Fi

Връзката между две устройства в Интернет се основава на принципа **клиент-сървър**, при която:

***клиент** се нарича компютърна програма, изискваща създаването на връзка и използване на някакъв компютърен ресурс;

***сървърно приложение** е компютърна програма, която предоставя даден ресурс на една или повече клиентски програми. Сървърното приложение „слуша“ за идващи заявки от клиенти, обслужва ги и продължава да „слуша“ за следващи клиенти. Типичен пример за комуникация клиент-сървър е качването на файл в Google Drive. Тук Интернет браузърът (Mozilla, Opera, Google Chrome) се нарича клиент, а Google Drive – сървърно приложение. Предоставеният ресурс е дисково пространство.

Интерфейс Wi-Fi

За да може клиент да се свърже към сървър е необходимо да се знаят поне **два параметъра** на сървъра:

***IP адрес**

***порт**, на който сървърното приложение очаква заявки от клиентите.

Под порт се разбира софтуерен механизъм за връзка, а не физически порт (куплунг). Софтуерните портове са необходими, за да се реализира достъп до сървъра от много потребители към много сървърни програми, а **физически конекторът за връзка може да е само един**.

Валидни числа за портове са 0 ÷ 65535.

Интерфейс Wi-Fi

Благодарение на портовете, **един сървър** може да предоставя по **няколко ресурса** едновременно. За целта, едно такова приложение трябва да е **многозадачно** (multi-threaded).

Пример - по подразбиране уеб страниците се обслужват на порт 80, т.е. когато напишем в браузъра www.mysite.com, всъщност ще се преведе като www.mysite.com:80 (с двуеточие се указва номера на порта)

Пример - клиентската програма, осигуряваща достъп до файловата система на сървър – SSH. Връзката на SSH се осъществява винаги на порт 22.

Пример - клиентската програма за контрол на версията Git. Връзката на Git се осъществява винаги на порт 9418.

Интерфейс Wi-Fi

От гледна точка на програмиста, **връзката клиент-сървър** се осъществява посредством т.нар. **сокети** (socket), които наподобяват файлове.

Сокетът е **крайна точка за комуникация** (endpoint) в една компютърна мрежа.

За да започне комуникацията, потребителската програма трябва да отвори сокет, след което да извърши всички операции по комуникацията с помощта на указател към този сокет (точно както се използват файлови указатели) и накрая да затвори сокета.

Този вид достъп наподобява файловия достъп (отваряне, четене/запис, затваряне).

Интерфейс Wi-Fi

Пример с CC3100 и използване на сокети.

Име на функцията	Кратко описание
sl_Socket( )	Създава крайна точка за комуникация, сокет, и връща указател към нея.
sl_Bind( )	Присвоява на сокета статичния IP адрес на мрежовата карта. Ако тя използва DHCP, като адрес на функцията трябва да се подаде нула. Присвоява на сокета порт, на който ще се осъществяват връзките.
sl_Listen( )	Използва се от сървърни програми. Отваря се присвоеният порт на присвоения IP адрес. От този момент нататък системата очаква („слуша“) за пристигащи връзки от клиенти.
sl_Accept( )	Приема връзка с клиент. Тази функция връща указател към сокета на клиента (с негова помощ може да разберем например IP адреса на клиента). Блокираща функция.
sl_Connect( )	Използва се от клиентски програми. На тази функция се предава IP адреса на сървъра и неговия порт, към който ще се свързваме.

Интерфейс Wi-Fi

Пример с CC3100 и използване на сокети.

sl_Recv( )	Връща приетите байтове от сървъра/клиента. Блокираща функция.
sl_Send( )	Изпраща определен брой байтове към сървъра/клиента.

Интерфейс Bluetooth

Bluetooth е безжичен, асинхронен, радио интерфейс. Работи в честотния диапазон 2400–2483.5 MHz. Скоростта на обмен достига до 24 Mbit/s (версия 3 на протокола).

Използва се комуникация между централно (central) и периферно устройство (peripheral).

Едно централно устройство може да си комуникира с до 7 периферни (до версия 3) или с произволен брой периферни (версия 4 & 5, BLE).

Устройствата се разделят на класове в зависимост от радиуса на действие:
*клас 1 – до 100 метра
*клас 2 – до 10 метра
*клас 3 – до 1 метър

Интерфейс Bluetooth

Версия 4 и 5 на протокола включва допълнителен протокол за енергийно ефективни приложения, който се нарича BLE (**B**luetooth **L**ow **E**nergy).

Скоростта на предаване е ограничена до 1 Mbit/s (v4) и 2 Mbit/s (v5).

Използват се 40 честотни канала през 2 MHz.

Три от каналите (37, 38, 39) се използват за т.нар. рекламиране.

При осъществяване на връзка, **устройствата превключват честотните канали периодично** (frequency hopping). Целта е да се повиши шумоустойчивостта и възможността за работа на много устройства едновременно.

Интерфейс Bluetooth

Рекламиране (advertising) – процес на периодично изпращане на опознавателни пакети от периферно (peripheral) устройство. Периодът е в границите 20 ÷ 10240 ms. Чрез тези пакети Bluetooth устройството може да бъде открито и централно устройство може да се свърже с него.

Сканиране (scanning) – процес на откриване на всички периферни устройства в близост. Сканирането се извършва от централно устройство.

Свързване (connecting) – процес на изграждане на връзка между централно и периферно устройство [14]. Централното устройство избира към кое периферно да се свърже на базата на рекламиращите пакети (които съдържат име на устройство, уникален номер на предлаганите ресурси – UUID, MAC адрес и др.).

Интерфейс Bluetooth

В процеса на свързване централното устройство изпраща пакет, наречен CONNECT_REQ, който трябва да извлече 4 параметъра от периферното устройство, за да може да се свърже с него:

*схема на честотно превключване (**frequency hopping sequence**)

*интервал на връзката (**connection interval**) – времето, за което връзката ще използва един честотен канал, след което ще превключи на друг. (7.5 ÷ 4000 ms)

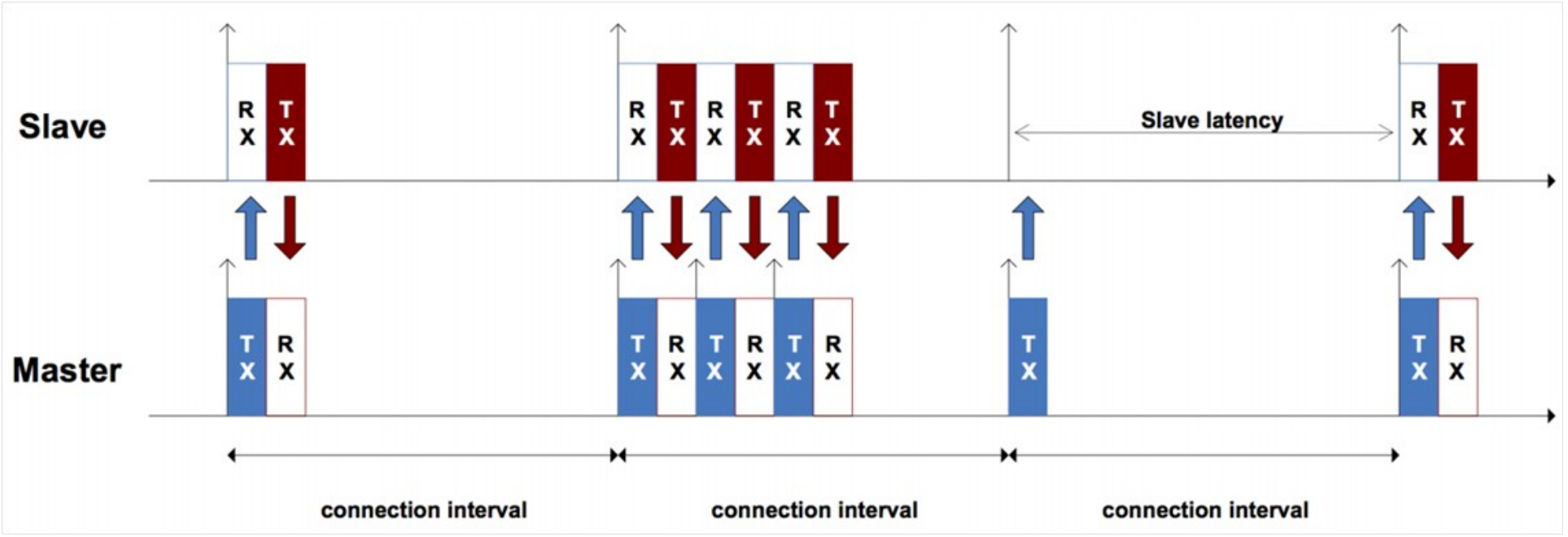
Интерфейс Bluetooth

*времезакъснение на подчиненото устройство (**slave latency**) – ако периферното устройство няма данни, които да предаде, може да не отговаря на централното устройство в рамките на един slave latency период.
[0 ÷ ((supervision timeout/connection interval)-1) ms]

*таймаут на връзка (**supervision timeout**) – максималното време между два правилно приети пакета, при което връзката се счита за осъществена. Ако вторият пакет пристигне по-късно от това време, връзката ще се счита за прекъсната. (100 ÷ 32000 ms)

Интерфейс Bluetooth

Когато се осъществи връзка, централното и периферното устройство си обменят информация на равни интервали от време [14].



Интерфейс Bluetooth

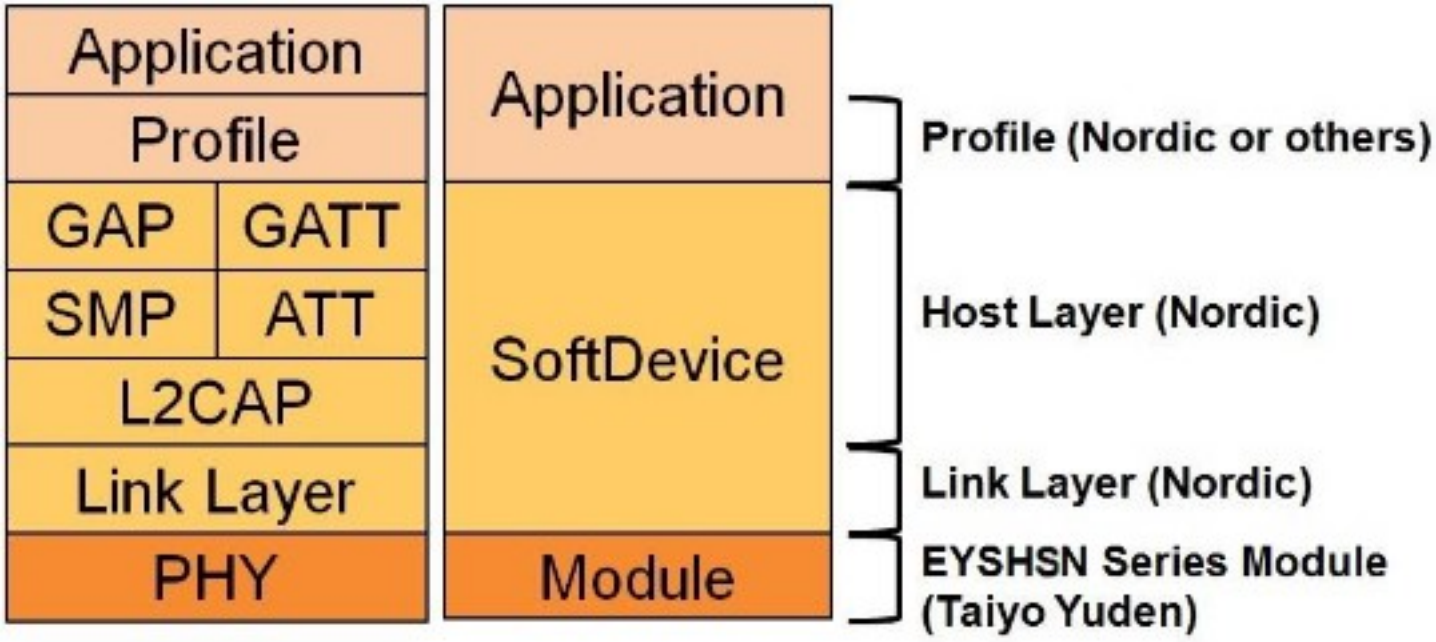
Съгласуване (pairing) – процес на предаване на служебна информация (ключове) между две устройства с цел криптиране на връзката между тях.

Запаметяване (bonding) – процес на записване на ключове за криптиране на връзка във всяко от съгласуваните устройства. Това позволява съгласуването да се извърши само първия път при свързването на двете устройства. Следващите свързвания не се нуждаят от фазата съгласуване и връзката е криптирана с ключ от преди.

Интерфейс Bluetooth

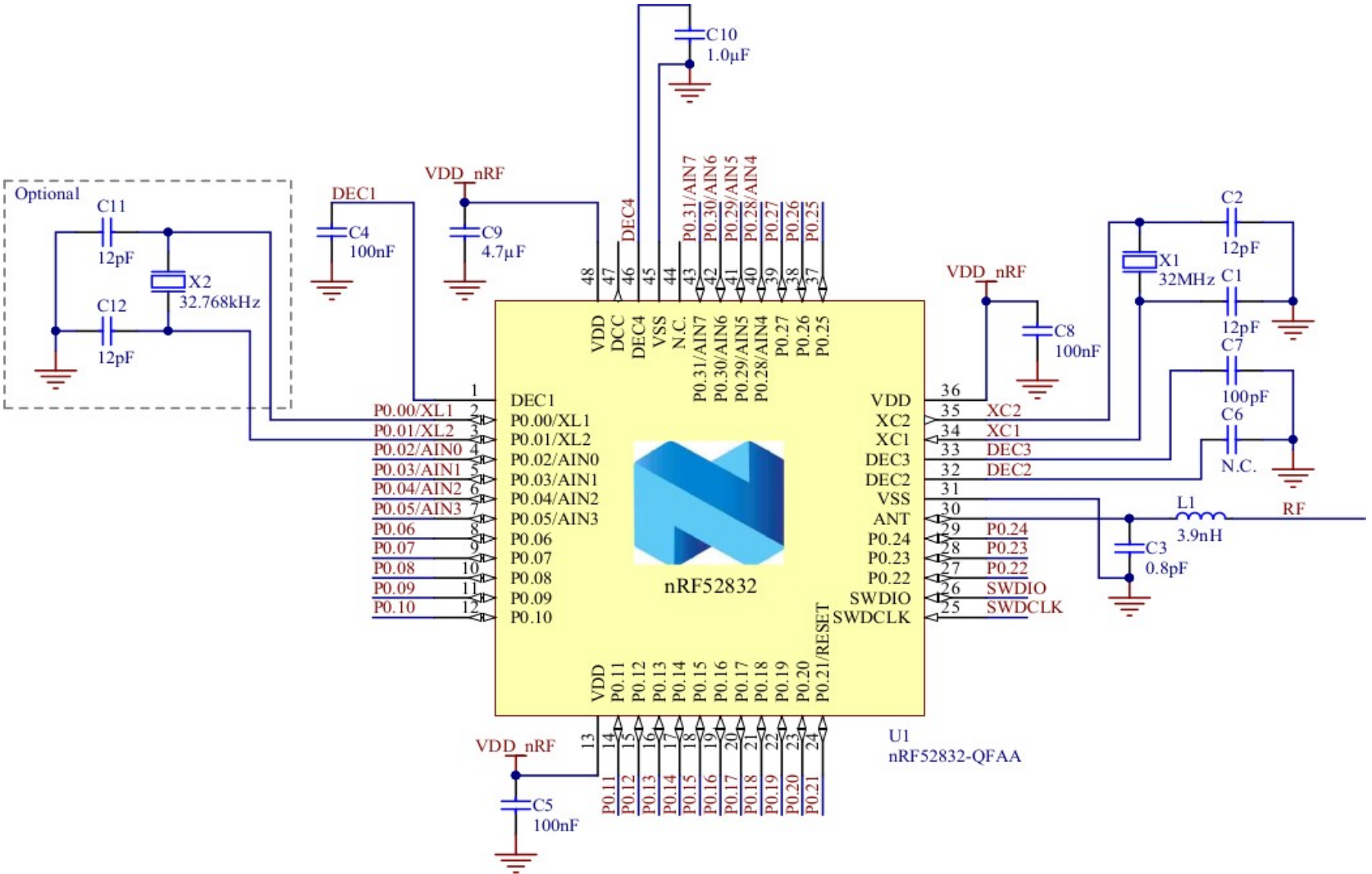
Пример – Nordic Semiconductor NRF52832 е микроконтролер с Bluetooth свързаност (v.4 и v.5, BLE). Фирмата Taiyo Yuden предлага Bluetooth сертифицирани модули, базирани на NRF51 & NRF52 фамилиите.

Bluetooth библиотеката е наречена SoftDevice. Приложната програма използва функции от нея и няма директен достъп до Bluetooth модула.



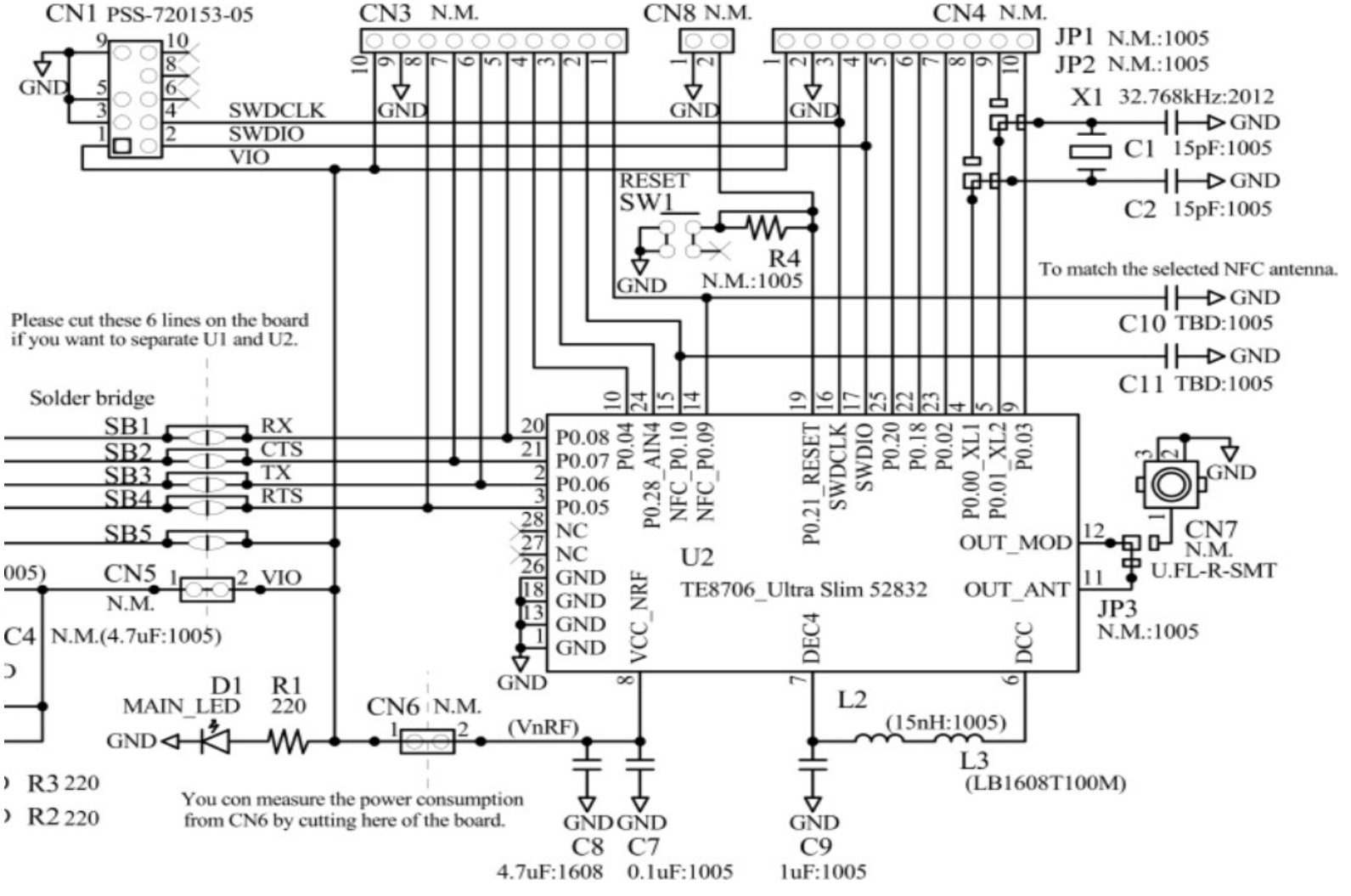
Интерфейс Bluetooth

Схема на свързване с минимум компоненти, дадена от Nordic.



Интерфейс Bluetooth

Схема на свързване на Bluetooth модул, дадена от Taiyo Yuden. Забележете L2, L3 и C9 – това са елементи за вътрешния импулсен постояннотоков преобразувател, който се използва в ултра-маломощни приложения за спестяване на енергия.



Интерфейс Bluetooth

Примерен код за инициализация на NRF52832.

```
  void user_bluetooth_init(void){
  uint32_t err_code;
  const nrf_clock_lf_cfg_t sd_lf_config =      {
            .source       = NRF_SDH_CLOCK_LF_SRC,
            .rc_ctiv      = NRF_SDH_CLOCK_LF_RC_CTIV,
            .rc_temp_ctiv = NRF_SDH_CLOCK_LF_RC_TEMP_CTIV,
            .accuracy     = NRF_SDH_CLOCK_LF_ACCURACY
  };
       timers_init();
       ble_stack_init();
       gap_params_init();
       gatt_init();
       services_init();
       advertising_init();
       conn_params_init();
      err_code = ble_advertising_start(&m_advertising, BLE_ADV_MODE_FAST);
     APP_ERROR_CHECK(err_code);
  }
```

Интерфейс Bluetooth

Приемането става чрез прекъсване.

```
volatile uint8_t my_array[32];

static void nus_data_handler(ble_nus_evt_t * p_evt){
    for (uint32_t i = 0; i < p_evt->params.rx_data.length; i++){
            my_array[i] = p_evt->params.rx_data.p_data[i];
        }
}
```

Предаването става чрез извикване на API функция.

```
uint32_t my_bluetooth_send(uint16_t buffer_len, uint8_t *buffer){
uint32_t err_code;

err_code = ble_nus_data_send(&m_nus, buffer, &buffer_len, m_conn_handle);

return err_code;
}
```

Литература

[1] http://elm-chan.org/docs/mmc/mmc_e.html
[2] http://www.dejazzer.com/ee379/lecture_notes/lec12_sd_card.pdf
[3] SD Specifications, Part 1, Physical Layer, Simplified Specification, Version 2.00
[4] SD Specifications, Part E1, SDIO Simplified Specification, Version 2.00, February 8, 2007
[5] Samsung SD & MicroSD Card product family SDA 3.0 specification compliant-Up to High Speed mode
[6] TOSHIBA SD Card Specification
[7] https://www.sdcard.org
[8] AN10911 SD(HC)-memory card and MMC interface conditioning, NXP, 2013.
[9] AN460, “Using the P82B96 for bus interface”, Philips Semiconductor, 2001.
[10] AN4760, “Quad-SPI interface on STM32 microcontrollers and microprocessors”, ST Microelectronics, 2020.
[11] D. McKenna, “Using the QuadSPI Module on MPC56xxS”, AN4186, NXP Semiconductor, 2010.
[12] S. Singh, “Designing with Cypress Quad SPI (QSPI) F-RAM”, AN218375, Cypress Semiconductor, 2017.
[13] Г. Илиев, Д. Атамян, “Мрежи за данни и интернет комуникации”, Нови знания, 2009.
[14] https://microchipdeveloper.com/wireless:ble-link-layer-connections