Materiały do laboratorium *programowania interfejsów szeregowych*

Spis treści

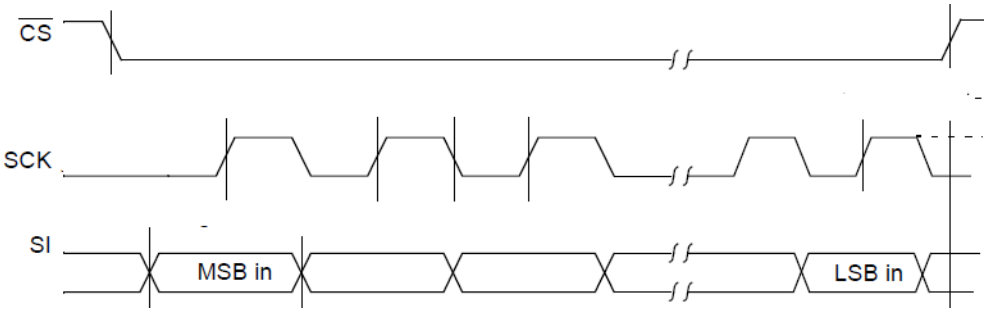
1. SPI Basic

Przetwornik MCP4921 jest 12-bitowym przetwornikiem cyfra/analog sterowanym za pośrednictwem interfejsu SPI.

Układ:

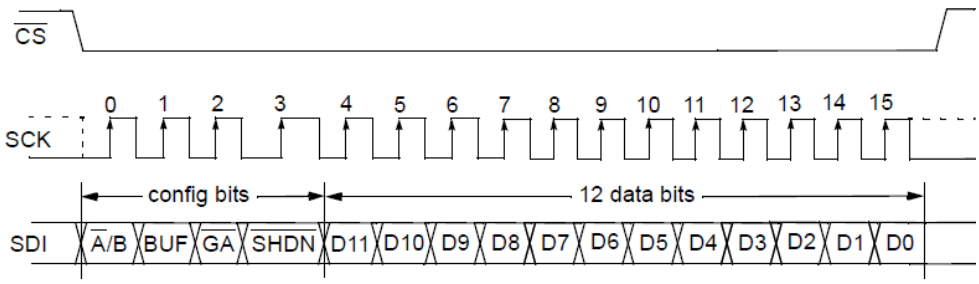
- wymaga podania zewnętrznego napięcia referencyjnego (podano 5V)
- posiada wzmacniacz wyjściowy o:
  - konfigurowalnym wzmocnieniu x1 lub x2.
  - Wyjściu, które może być przełączone w stan wysokiej impedancji
- zależności od obudowy zawiera dwa kanały (A i B) albo jeden kanał (A). W naszym przypadku używamy wersji z jednym kanałem.

Na poniższym rysunku pokazano schemat transmisji pojedynczego słowa (2 bajty) sterującego przetwornikiem.



Poniżej zamieszczono kolejność i znaczenie poszczególnych bitów w słowie sterującym.

bit 15 **A/B:** DAC_A or DAC_B Select bit
1 = Write to DAC_B
0 = Write to DAC_A
bit 14 **BUF:** V_{REF} Input Buffer Control bit
1 = Buffered
0 = Unbuffered
bit 13 **GA:** Output Gain Select bit
1 = 1x ($V_{OUT} = V_{REF} * D/4096$)
0 = 2x ($V_{OUT} = 2 * V_{REF} * D/4096$)
bit 12 **SHDN:** Output Power Down Control bit
1 = Output buffer enabled
0 = Output buffer disabled
bit 11-0 **D11:D0:** DAC Data bits
12 bit number “D” which sets the output value.
Contains a value between 0 and 4095.



Napisać **funkcję** void **DAC_MCP4921_Set**(unsigned int uiVoltage).
Zadaniem funkcji jest ustawienie na wyjściu przetwornika cyfra/analog (MCP4921) napięcia podanego w argumencie uiVoltage (wartość wyrażona w bitach nie w woltach).

W pierwszej kolejności należy zapoznać się z opisem modułu SPI.
Należy zwrócić szczególną uwagę na:

- rys. 26,
- bity CPHA,CPOL,MSTR,LSBF (SPI Control Register),
- SPI Data Register,
- bit SPIF (SPI Status Register),
- SPI Clock Counter Register.

Funkcja DAC_MCP4921_Set powinna w pierwszej kolejności dokonać **inicjalizacji** pinów i modułu SPI czyli ustawić:

- funkcję odpowiednich pinów na SPI (patrz Pin Connect Block),
- pin P0.10 (używany dalej jako Chip Select, CS) na wyjściowy,
- ustawić tryb pracy SPI0 na „master”
- ustawić bity *cpha* i *cpol* odpowiednio do współpracy z przetwornikiem
- ustawić częstotliwość zegara SPI0 na 1/8 częstotliwości taktowania peryferiów (*pclk*)

Następnie funkcja powinna dokonać **transmisji** za pomocą SPI dwóch bajtów sterujących o odpowiedniej zawartości, czyli:

- ustawić odpowiednią wartość na linii CS,
- zainicjować wysłanie starszego bajtu sterowania,
- poczekać na zakończanie transmisji,
- zainicjować wysłanie młodszego bajtu sterowania,
- poczekać na zakończanie transmisji,
- ustawić odpowiednią wartość na linii CS.

Działanie funkcji przetestować wpisując co sekundę do przetwornika wartość minimalną, maksymalną i „środkową”. Działanie sprawdzić multimetrem lub oscyloskopem.
UWAGA: Przed uruchomieniem programu podać 3.3V na linię SSL (poprosić prowadzącego o pomoc)

2. SPI Advanced

W „SPI Basic" funkcje pracowały bezpośrednio na rejestrach SPI. Nie jest to złe rozwiązanie jeżeli nie mamy zamiaru dokładać więcej obsługiwanych układów (nazywanych dalej Slave-ami) oraz jeżeli nie zamierzamy zmieniać mikrokontrolera. Wcześniej czy później jednak jeden z dwóch wymienionych przypadków zwykle ma miejsce.
Dodanie nowego Slave-a wymaga napisania nowej funkcji komunikującej się bezpośrednio z modułem SPI.
Ponieważ każda rodzina mikrokontrolerów ma inaczej rozwiązane moduły SPI dlatego zmiana mikrokontrolera wymaga ponownego napisania funkcji sterujących Slave-ami.
Wynikiem takiego „prostego" podejścia do jest wydłużenie czasu implementacji wnikające z wprowadzania nowego kodu (nowych błędów). Zmiana mikrokontrolera wiąże się z koniecznością przepisania wszystkich funkcji do wszystkich układów.

Podejście zastosowane w „SPI Advanced" polega na zastosowaniu dwóch funkcji pośredniczących między funkcjami sterującymi poszczególnymi urządzeniami podłączonymi do SPI-a („Slave-ami") a modułem SPI.

Zadaniem funkcji **`SPI_ConfigMaster(SPI_FrameParams)`** jest konfiguracja modułu SPI (zerowego) czyli:

- konfiguracja odpowiednich pinów do pracy z SPI
  UWAGA: funkcja nie odpowiada za sygnał CS
- ustawienie parametrów ramki transmisyjnej (CPHA,CPOL, LSBF)
- Ustawienie częstotliwości transmisji (podzielnik zegara peryferiów)
- Ustawienie modułu do pracy w trybie „Master"

Funkcja przyjmuje jako argument strukturę:

```
struct SPI_FrameParams{
   unsigned char  ucCpha;
   unsigned char  ucCpol;
   unsigned char  ucClsbf;
   unsigned char  ClkDivider;
}
```

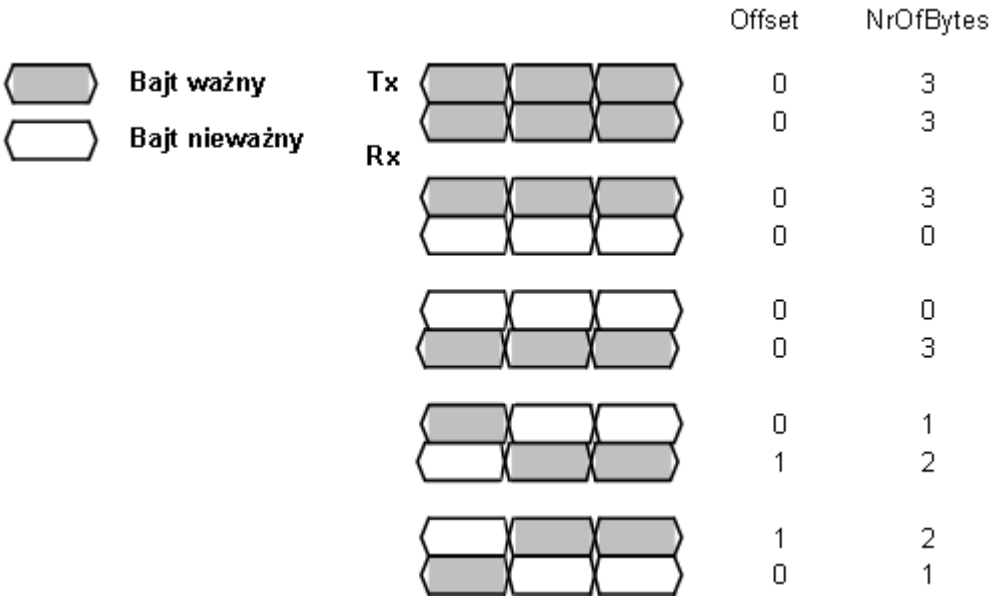
Zadaniem funkcji **`SPI_ExecuteTransaction(struct SPI_TransactionParams)`** jest transmisja/odbiór pewnej ilości bajtów. Ilość oraz zawartość bajtów wynikają ze specyfiki urządzenia oraz ewentualnie funkcjonalności, którą chcemy w danym momencie urzyć (np. w przypadku pamięci może to być zapis lub odczyt).
Jako argument funkcja przyjmuje zmienną typu `TransactionParams`. zmienna definiuje parametry transmisji ale nie są to jak w przypadku **`SPI_Config`** parametry transmisji pojedynczego bajtu ale informacja mówiąca ile bajtów ma być wysłanych/odebranych i jaką maja mieć zawartość (w przypadku wysyłanych). Krótki opis parametrów umieszczono w definicji struktury po komentarzu.

```
struct SPI_TransactionParams{

      unsigned char *pucBytesForTx;     // wskaźnik na tablicę z bajtami do wysłania
```

```
    unsigned char  ucNrOfBytesForTx; // ilość bajtów do wysłania
    unsigned char  ucTxBytesOffset;  // offset bajtów do wysłania

    unsigned char *pucBytesForRx;    // wskaźnik na tablicę na odebrane bajty
    unsigned char  ucNrOfBytesForRx; // ilość bajtów do odebrania
    unsigned char  ucRxBytesOffset;  // offset bajtów do odebrania
};
```

Znaczenie pól `BytesOffset i ucNrOfBytes` pokazano na poniższym rysunku.

Rysunek pokazuje wartości parametrów Offset i NrOfBytes dla bajtów nadawanych i odbieranych dla pięciu różnych transakcji. Przykładowo w ostatniej transakcji najpierw odbieramy 1 bajt a potem wysyłamy 2 bajty (mowa tu o ważnych bajtach)



Definicje funkcji `void SPI_ExecuteTransaction(struct SPI_FrameParams,struct SPI_TransactionParams);`

UWAGA: funkcja nie odpowiada za sygnał CS

Testy przetwornika DAC MCP4921

Zadaniem funkcji **`DAC_MCP4921_InitCSPin`** jest pinu kontrolującego wejście CS przetwornika na wyjściowy i ustawienie go w stan nieaktywny (‘1’).

Zadaniem funkcji **`DAC_MCP4921_Set_Adv`**`(unsigned int uiData)` jest ustawienie napięcia na wyjściu przetwornika MCP4921 proporcjonalnego do argumentu wywołania (`uiData`).
Test funkcji należy przeprowadzić na definiując komendę "dacset" (keyword - DACSET z jednym argumentem liczbowym. Przykład komendy „dacset 0x0100 ”.
UWAGA: funkcja odpowiada za sygnał CS

Testy 8-bitowej bramy MCP23S09

Jest to ośmiobitowa bramą wejścia/wyjścia sterowana za pośrednictwem interfejsu SPI. Brama pozwala sterować kierunkiem oraz stanem ośmiu linii cyfrowych. W przypadku ustawienia linii jako wejściowych układ pozwala odczytywać ich stan. Do sterowania układem służy osiem rejestrów. Każdy rejestr posiada swój adres. Do implementacji funkcji testowych opisanych w dalszej części rozdziału konieczne jest użycie dwóch rejestrów: *Direction Register* (adres: 0*)* i *Port Register* (Adres: 9). Opis rejestrów poniżej.

I/O DIRECTION REGISTER

Controls the direction of the data I/O.

When a bit is set, the corresponding pin becomes an input. When a bit is clear, the corresponding pin becomes an output.

REGISTER 1-2: IODIR – I/O DIRECTION REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **IO7:IO0:** Controls the direction of data I/O <7:0>

1 = Pin is configured as an input.
0 = Pin is configured as an output.

PORT REGISTER

The GPIO register reflects the value on the port. Reading from this register reads the port. Writing to this register modifies the Output Latch (OLAT) register.

REGISTER 1-11: GPIO – GENERAL PURPOSE I/O PORT REGISTER

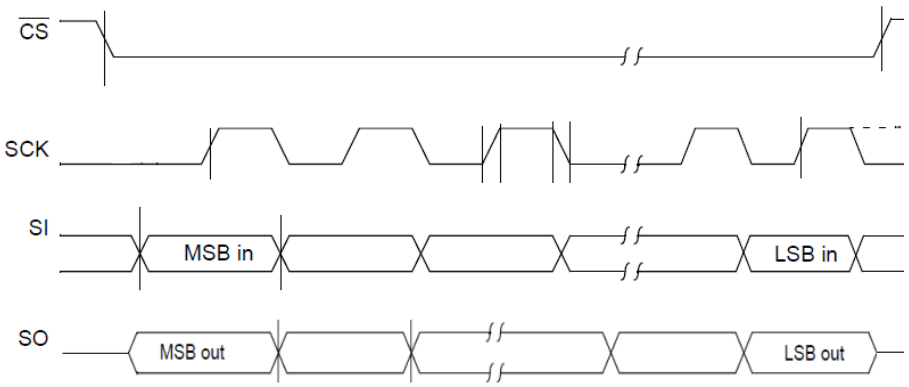
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

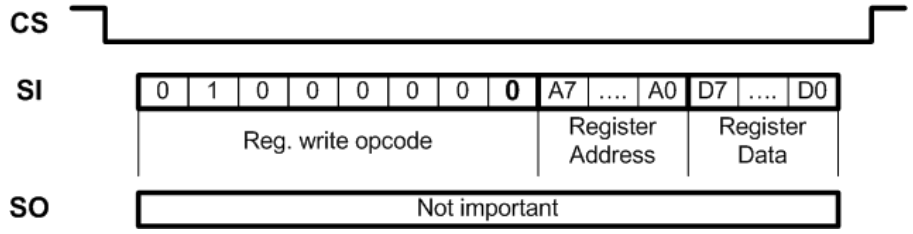
bit 7-0 **GP7:GP0:** Reflects the logic level on the pins <7:0>.

1 = Logic-high.
0 = Logic-low.

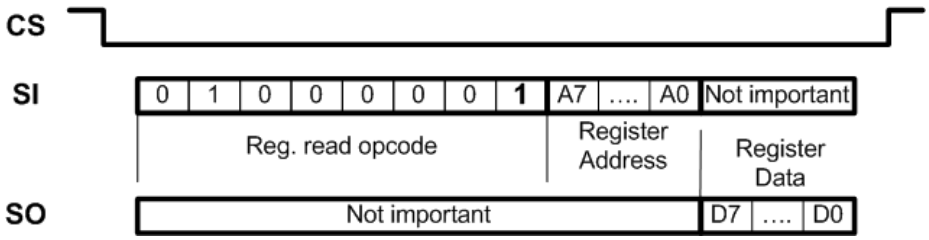
Na poniższym rysunku pokazano schemat transmisji pojedynczego słowa.



Następnie zamieszczono kolejność i znaczenie poszczególnych bitów w słowie sterującym podczas **zapisu rejestru** o danym adresie



i **odczytu rejestru** o danym adresie



Zadaniem funkcji **Port_MCP23S09_InitCSPin** jest pinu kontrolującego wejście CS przetwornika na wyjściowy i ustawienie go w stan nieaktywny (‘1’).

Zadaniem, funkcji **Port_MCP23S09_Set**(unsigned char ucData)jest wpisanie na wyjścia portu MCP23S09 wartości podanej w argumencie wywołania.
Należy pamiętać o ustawieniu pinów portu MCP23S09 jako wyjściowe.
Test funkcji należy przeprowadzić na definiując komendę "portset" (keyword - PORTSET) z jednym argumentem liczbowym. Przykładowo komenda „portset 0x0055” powinna zapalić co 2 bit.
UWAGA: funkcja odpowiada za sygnał CS

Zadaniem, funkcji unsigned char **Port_MCP23S09_Get**(void) jest odczyt wartości z wejścia portu MCP23S09.
Należy pamiętać o ustawieniu pinów jako wejściowe.
UWAGA: funkcja odpowiada za sygnał CS
Test funkcji należy przeprowadzić na definiując komendę "portget" (keyword - MCP_PORT_RD) bez żadnego argumentu. Przykładowo wysłanie do uC komendy „portget” powinno powodować odesłanie do PC komendy „portget 0x00ZZ” gdzie ZZ powinno odzwierciedlać stan pinów na wejściu portu.

3. I²C Basic

Przed rozpoczęciem implementacji podanych niżej funkcji sugeruje się zapoznanie z poniższymi dokumentami:

- „I2C_wprowadzenie_pl.pdf" – krótkie wprowadzenie do I2C (po polsku)
- „lpc2119_2129_2194_2292_2294_user_manual.pdf" – opis i2c w manualu uC
- "Uart_SPI_I2C.pdf" – przykład użycia I2C podany przez NXP
- "hitex_lpc-arm-book_rev10-screen.pdf" str. 90-94, przykład użycia I2C podany przez firmę Hitex
- „8XC552_I2C.pdf" opis modułu I2C użytego w mikrokontrolerze

Napisać funkcję **`I2C_Init`**`(void)`, której zadaniem jest przeprowadzenie inicjalizacji potrzebnych do pracy z I2C.
Funkcja powinna:

- Ustawić odpowiednie piny do pracy z I2C
- Zerować flagi w rejestrze kontrolnym
- Ustawiać częstotliwość zegara (oba rejestry na 0x80)
- Ustawić odpowiednio kontroler przerwań (UWAGA: należy stworzyć wcześniej funkcję obsługi przerwania od I2C - `I2C_Interrupt`)

Test

Napisać funkcję **`PCF8574_Write`**`(unsigned char ucData)`, która zainicjuje wpisanie na wyjście portu PCF8574 wartość podaną w argumencie.

W odróżnieniu od SPI w przypadku I2C transmisja powinna działać „na" przerwaniach dlatego należy stworzyć funkcję obsługi przerwania - **`I2C_Interrupt`**

UWAGA: Należy pamiętać, że funkcja `PCF8574_Write` tylko inicjalizuje transmisję ale nie czeka na jej zakończenie, czyli że wszystkie zmienne lokalne znikną zanim transmisja dobiegnie końca.

4. I²C Advanced

W „I2C Basic” funkcja **`PCF8574_Write`** pracowała bezpośrednio na rejestrach I2C. Nie jest to złe rozwiązanie jeżeli nie mamy zamiast dokładać więcej obsługiwanych układów oraz jeżeli nie zamierzamy zmieniać mikrokontrolera. Wcześniej czy później jednak jeden z dwóch wymienionych przypadków zwykle ma miejsce.
Ponowne zastosowane podejścia „SPI Basic” wiąże się z przypominaniem sobie od nowa sterownika I2C na danym procesorze oraz kopiowaniu/modyfikowaniu kodu z istniejących funkcji. Wynikiem jest wydłużenie czasu implementacji wnikające z wprowadzania nowego kodu (nowych błędów). Zmiana mikrokontrolera wiąże się z koniecznością przepisania wszystkich funkcji do wszystkich układów.

Podejście zastosowane w „I2C Advanced” polega na zastosowaniu funkcji („I2C_ExecuteTransaction(sI2C)”) pośredniczącej między I2C a funkcja obsługującymi poszczególne „Slavy”.

Wywołując funkcje przekazujemy jej wszystkie parametry wynikające ze specyfiki danego „Sleva” i konkretnej funkcjonalności, którą chcemy wykorzystać.

Definicja struktury definiującej parametry pojedynczej transakcji I2C w trybie master wygląda następująco:

```
enum I2CTransmisionMode {TX,RX,RX_AFTER_TX,TX_AFTER_RX } ;

struct I2C_Params{
    enum I2CTransmisionMode eI2CTransmisionMode;

    unsigned char ucSlaveAddress;

    unsigned char *pucBytesForTx;
    unsigned char ucNrOfBytesForTx;

    unsigned char *pucBytesForRx;
    unsigned char ucNrOfBytesForRx;

    unsigned char ucDone;
};
```

Pole **`eI2CTransmisionMode`** oznacza tryb transmisji. Docelowo należy zaimplementować Tx,Rx,RxAfterTx.

Pole **`ucSlaveAddress`** oznacza sygnaturę układu scalonego, z którym zamierzamy się komunikować.

Pole **`ucDone`** informuje jedynką o zakończeniu transmisji.

Znaczenie reszty pól jest analogiczne jak w przypadku „SPI Advanced”

Testy

Napisać funkcję **`PCF8574_Write`**`(unsigned char ucData)`, która zainicjuje wpisanie na wyjście portu PCF8574 wartość podaną w argumencie.

Napisać funkcję **`PCF8574_Read`**`(void)`, która zainicjuje odczyt stanu wejść portu PCF8574 . Odczytany bajt powinien znaleźć się w zmiennej globalnej `ucPCF8574_Input`.

Napisać funkcję **`void MC24LC64_ByteWrite( unsigned int WordAddress, unsigned char Data)`** realizującą zapis bajta do pamięci MC24LC64.

Napisać funkcję **`void MC24LC64_RandomRead (unsigned int WordAddress)`** realizującą odczyt bajta z pamięci MC24LC64.

5. CAN Basic

Przed rozpoczęciem implementacji podanych niżej funkcji sugeruje się zapoznanie z poniższymi dokumentami:

- „CAN_wprowadzenie_pl.pdf" – krótkie wprowadzenie do CAN(po polsku)
- "hitex_lpc-arm-book_rev10-screen.pdf" rozdział o CAN – pominąć rozdział o błędach
- „lpc2119_2129_2194_2292_2294_user_manual.pdf" – rozdział o CAN

Mikrokontroler LPC2129 posiada 2 kontrolery CAN – CAN1, CAN2. Wyjścia obu z nich zostały wyprowadzone na złącza DB9 EVM-a. Zadaniem programu jest wysyłanie co 1 sekundę pojedynczego komunikatu z CAN1 do CAN2.

Należy pamiętać że nadajnik CAN sygnalizuje wysłanie komunikatu tylko wtedy kiedy któryś z odbiorników potwierdził odbiór. W praktyce oznacza to, że aby zrealizować zdefiniowane powyżej zadanie należy również odpowiednio ustawić odbiornik.

Zadanie

W pierwszej kolejności należy napisać funkcję **`CanInit`**`(void)`. Funkcja powinna składać się z dwóch zaznaczony wyraźnie sekcji realizujących inicjalizację nadajnika (CAN1) i odbiornika (CAN2).

W sekcji inicjalizacji nadajnika powinny znajdować się :

- konfiguracja pinów,
- ustawienie „bit timing-u" jak w przykładzie z Hitexa,
- ustawienie ilości bajtów danych na 4

W sekcji inicjalizacji odbiornika powinny znajdować się:

- konfiguracja pinów,
- ustawienie „bit timing-u" jak w nadajniku,
- wyłączenie filtra komunikatów („Acceptance filter" ma przepuszczać wszystkie komunkiaty)
- zwolnienie bufora odbiornika

W trakcie ustawiania rejestrów konfiguracyjnych kontrolery powinny znajdować się w stanie resetu.

W funkcji należy wykorzystać rejestry PINSEL, MOD, BTR, TFI1, AFMR, CMR.

W funkcji „**`main`**`()`" należy:

1. Wstawić fragment kodu, który będzie co 10 razy na sekundę wysyłał komunikat (4 bajty)z informacją o stanie przycisków (użyć „ReadButtons()"). Inicjacja wysłania powinna odbywać się tylko pod warunkiem zakończenia wysyłania poprzedniego komunikatu.

2. Wstawić fragment kodu który w przypadku odebrania komunikatu (CAN2) będzie zapalał diodę o numerze odpowiadającym numerowi naciśniętego przycisku oraz zwalniał odbiornik (umożliwiał odebranie następno komunikatu)

W funkcji należy wykorzystać rejestry : SR, TID1, TDA1, CMR, RDA.

6. CAN Advanced

W poprzednim punkcie odbiornik odbierał wszystkie wysyłane przez nadajnik komunikaty, bez względu na ich identyfikator. Programowe filtrowanie komunikatów, odbieranie i sprawdzanie identyfikatora każdego komunikatu który pojawi się na magistrali jest nieefektywne. Z tego powodu większość kontrolerów CAN posiada moduł, który w sposób nie absorbujący (lub absorbujący w małym stopniu) jednostkę główną mikrokontrolera pozwala na filtrowanie przychodzących komunikatów ze względu na identyfikator.

„Acceptance Filter" został opisany w „hitex_lpc-arm-book_rev10-screen.pdf" i „lpc2119_2129_2194_2292_2294_user_manual.pdf".

Zadanie

Należy zmodyfikować program z poprzedniego zadania tak aby odbiornik akceptował tylko komunikaty o identyfikatorze „17"

Oprócz podanych w poprzednich punktach rejestrów należy użyć: SFF_sa, SFF_GRP_sa, EFF_sa, EFF_GRP_sa, ENDofTable,