MTM

Standardy komunikacji międzyukładowej w systemach wbudowanych

Instrukcja do ćwiczeń laboratoryjnych

Mirosław Żołądź

2018

Spis treści

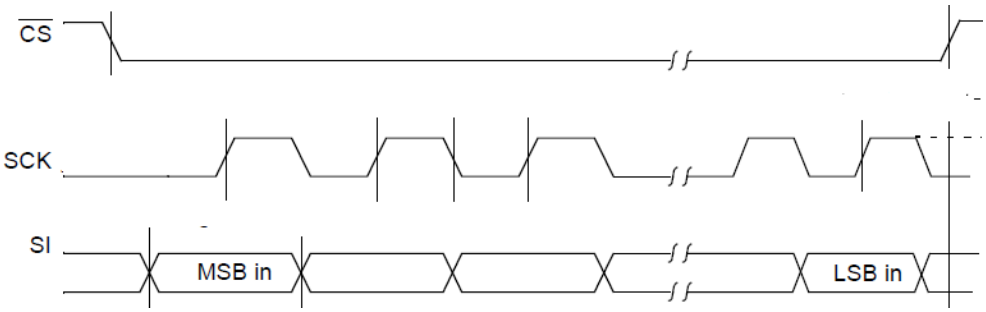
1. SPI Basic

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

Układ:

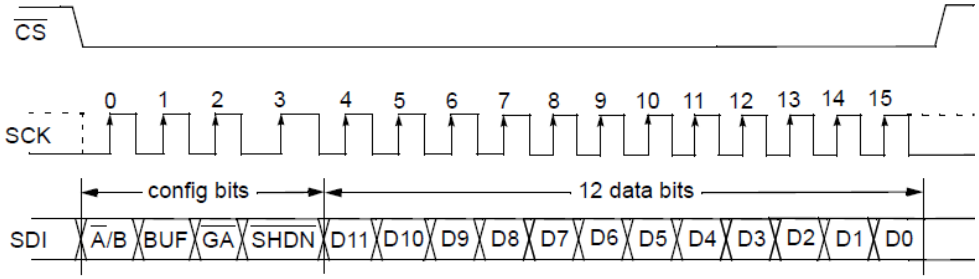
- wymaga podania zewnętrznego napięcia referencyjnego (podano 5V)
- posiada wzmacniacz wyjściowy o:
  - konfigurowalnym wzmocnieniu x1 lub x2.
  - Wyjściu, które może być przełączone w stan wysokiej impedancji
- zależności od obudowy zawiera dwa kanały (A i B) albo jeden kanał (A). W naszym przypadku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



Zadanie 1.

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 cyklicznie do przetwornika wartość minimalną, maksymalną i „środkową” (sprawdzić oscyloskopem).

UWAGA: Przed uruchomieniem programu podać 3.3V na linię SSL (poprosić prowadzącego o pomoc)

Zadanie 2.

Napisać funkcję **DAC_MCP4921_Set_mV(**unsigned int uiVoltage**),** która będzie przyjmować argument wyrażony w miliwoltach. Funkcja powinna używać funkcji DAC_MCP4921_Set.

Działanie funkcji przetestować ustawiając cyklicznie na wyjściu przetwornika napięcia 0 V, 0.5 V i 1 V (sprawdzić oscyloskopem).

Zadanie 3.

a) Napisać program, który będzie generował sinusoidę o wartości średniej 1V i amplitudzie p-p 2V. Na pojedynczy okres sinusoidy powinien składać się z 360 ustawień przetwornika. Zmierzyć czas trwania jednego okresu.

b) Obliczyć czas potrzebny na wygenerowanie jednego okresu biorąc pod uwagę tylko transmisję po magistrali SPI i porównać z rzeczywistym czasem trwania jednego okresu.
Wyjaśnić różnicę. Zmierzyć przy pomocy oscyloskopu czas pomiędzy poszczególnych transakcjami na szynie SPI.

c) Zmodyfikować program tak aby znacząco skrócić czas trwania jednego okresu sinusoidy.

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.

Bardziej uniwersalne podejście polega na zastosowaniu dwóch funkcji pośredniczących między funkcjami sterującymi poszczególnymi urządzeniami podłączonymi do SPI-a („Slave-ami”) a modułem SPI.

Zadanie 1

Napisać funkcję **`SPI_ConfigMaster(SPI_FrameParams)`**, która skonfiguruje moduł SPI0 (zerowego) ,tj.:

- ustawi odpowiednie piny do pracy z SPI
  UWAGA: funkcja nie odpowiada za sygnał CS
- ustawi parametry ramki transmisyjnej (CPHA,CPOL, LSBF)
- ustawi częstotliwość transmisji (podzielnik zegara peryferiów)
- ustawi 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;
}
```

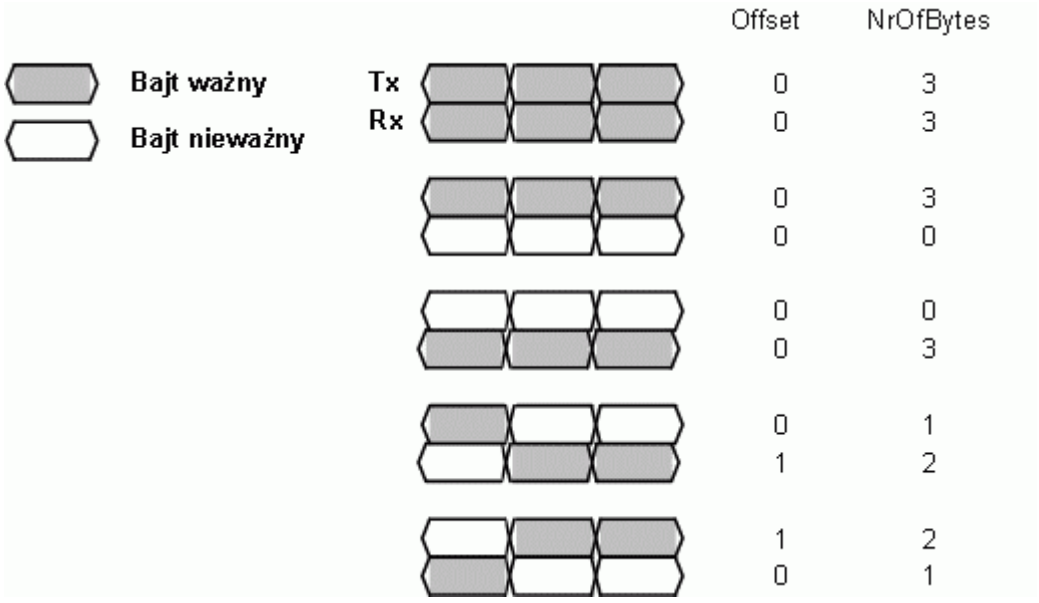
Napisać funkcję **`SPI_ExecuteTransaction(struct SPI_TransactionParams)`**, której zadaniem jest transmisja/odbiór pewnej ilości bajtów. Ilość oraz zawartość bajtów wynikają ze specyfiki urządzenia oraz ewentualnie funkcjonalności, którą chcemy w danym momencie użyć (np. w przypadku pamięci może to być 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 bajta 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)



UWAGA: funkcja nie odpowiada za sygnał CS

Test. Zrealizować funkcjonalność z ćwiczenia 3c z rozdziału SPI Basic z użyciem funkcji **`SPI_ConfigMaster i ExecuteTransaction.`**

Brama 8-bitowa 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

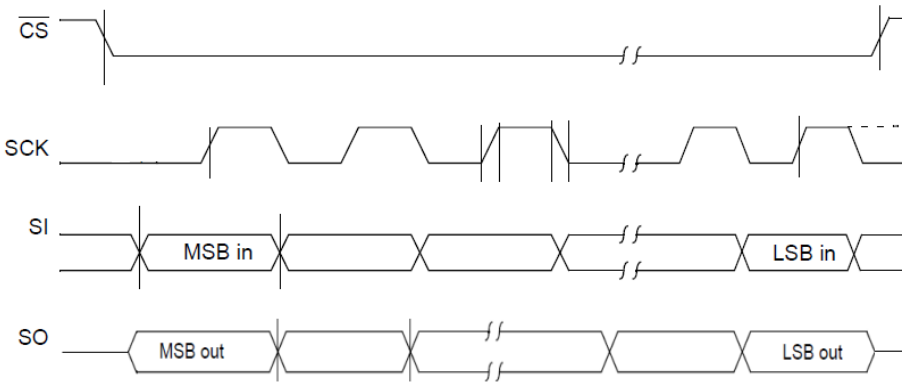
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
bit 7							bit 0

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

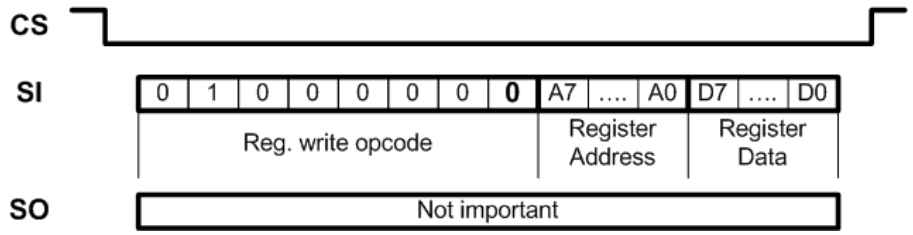
bit 7-0 **GP7:GP0:** Reflects the logic level on the pins <7:0>.

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

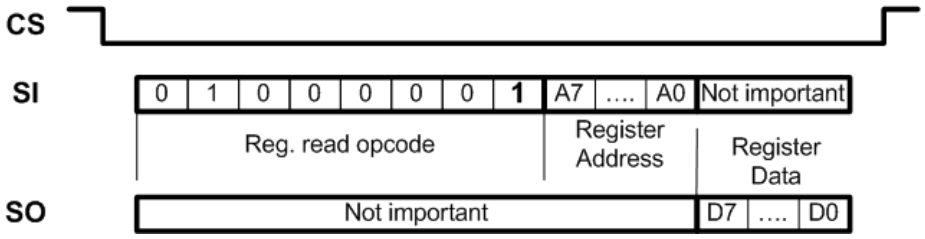
Na poniższym rysunku pokazano schemat transmisji pojedynczego słowa.



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



i **odczytu rejestru** o danym adresie



Zadanie 2. Zaimplementować i przetestować poniższe funkcje

`Port_MCP23S09_InitCSPin:` zadaniem funkcji jest inicjalizacja pinu kontrolującego wejście CS przetwornika na wyjściowy i ustawienie go w stan nieaktywny (‘1’).

`Port_MCP23S09_Set(unsigned char ucData):` zadaniem funkcji jest wpisanie na wyjścia portu `MCP23S09` wartości podanej w argumencie wywołania. Należy pamiętać o ustawieniu pinów portu `MCP23S09` jako wyjściowe.

Testy:

a) wyświetlanie na LED-ach licznika binarnego
b) implementacja komendy "spi_port_set" z jednym argumentem liczbowym. Przykładowo wysłanie do uC komendy „spi_port_set 0x55” powinna zapalić co drugą diodę LED.

`unsigned char Port_MCP23S09_Get(void):` zadaniem funkcji jest jest odczyt wartości z wejścia portu `MCP23S09`. Należy pamiętać o ustawieniu pinów jako wejściowe.

Test: zaimplementować komendę " spi_port_get „. Przykładowo wysłanie do uC komendy „spi_port_get” powinno powodować odesłanie do PC łańcucha „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 mikrokontrolera
- "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

Zadanie 1

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)

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.

Test funkcji `PCF8574_Write` przeprowadzić pisząc program wyświetlający licznik binarny na diodach LED

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(…)`, pełniącej rolę uniwersalnego drivera I2C pośredniczącego między modułem I2C a funkcjami specyficznymi poszczególnych „slaw-ów".

Zadanie

Napisać funkcję `ExecuteTransaction(struct I2C_Params)` realizującą transakcję I2C w sposób określony przez pola poniższej struktury

```
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 tryby: `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

1) Napisać funkcję **PCF8574_Write**(unsigned char ucData), która zainicjuje wpisanie na wyjście portu PCF8574 wartość podaną w argumencie. Test przeprowadzić z użyciem terminala i komendy „portwr value

2) Napisać funkcję **PCF8574_Read**(void), która zainicjuje odczyt stanu wejść portu PCF8574 . Odczytany bajt powinien znaleźć się w zmiennej globalnej ucPCF8574_Input. Test przeprowadzić z użyciem terminala i komendy „portrd”

3) Napisać funkcję **void MC24LC64_ByteWrite(unsigned int WordAddress, unsigned char Data)** realizującą zapis bajta do pamięci MC24LC64. Test przeprowadzić z użyciem terminala i komendy „memwr addr value”

4) Napisać funkcję **void MC24LC64_RandomRead (unsigned int WordAddress)** realizującą odczyt bajta z pamięci MC24LC64. Test przeprowadzić z użyciem terminala i komendy „memrd addr”

5. CAN

1. Przed wykonaniem zadań sugeruje się zapoznanie z poniższymi dokumentami:
   - Rozdział „Instrukcja obsługi zestawów uruchomieniowych**Zestaw do ćwiczeń z CAN-em**”
   - „Strona SKMSW\Załączniki\CAN**CAN_wprowadzenie_pl.pdf**” – krótkie wprowadzenie do CAN (po polsku)
   - “Strona SKMSW\Załączniki\SPI_I2C**hitex_lpc-arm-book_rev10-screen.pdf**” rozdział o CAN (można pominąć rozdział o błędach)
   - „Strona MITP\Załączniki**lpc2119_2129_2194_2292_2294_user_manual.pdf**” – rozdział o CAN

2. W załącznikach znajdują się 4 katalogi zawierające materiały do ćwiczeń z CAN.

Katalog „CAN”, oprócz krótkiego wprowadzenia do interfejsu CAN, zawiera programy do testów zestawu uruchomieniowego.

Pozostałe katalogi zawierają szkielety kodów źródłowych programów do poszczególnych zadań (pliki *.c) oraz wersje wykonywalne poszczególnych zadań (pliki *.hex).

Treść zadań znajduje się w komentarzu na początku plików z kodami źródłowymi.

6. Obsługa zestawów uruchomieniowych

SPI i I^{2}C

Zestaw wymaga zasilania 12 V. Należy je podać z dogniazdkowego zasilacza sieciowego.

Wszystkie układy scalone znajdujące się na zestawie zasilane są z napięcia 3.3V.

Zestaw składa się części SPI i części I2C.

W obu częściach znajdują się dwa zestawy złącz typu goldpin.

Zestaw złącz znajdujący się przy krawędzi płytki służy do połączenia linii sterujących z mikrokontrolerem.

Zestaw wewnątrz płytki służy do obserwacji stanu linii sterujących przy pomocy oscyloskopu.

Do podłączania masy oscyloskopu służą piny znajdujące się w rogach płytki.

Do magistrali SPI podłączone są przetwornik C/A (MCP4921) oraz 8-bitowa brama wejścia/wyjścia (MCP23S09T).

Napiecie referencyjne przetwornika jest równe 3.3V
Napięcie na wyjściu przetwornika można obserwować za pomocą multimetru z 2-milimetrowymi końcówkami (dwumilimetrowe otwory po prawej i lewej stronie przetwornika) oraz za pomocą oscyloskopu goldpin z podpisem V_OUT.

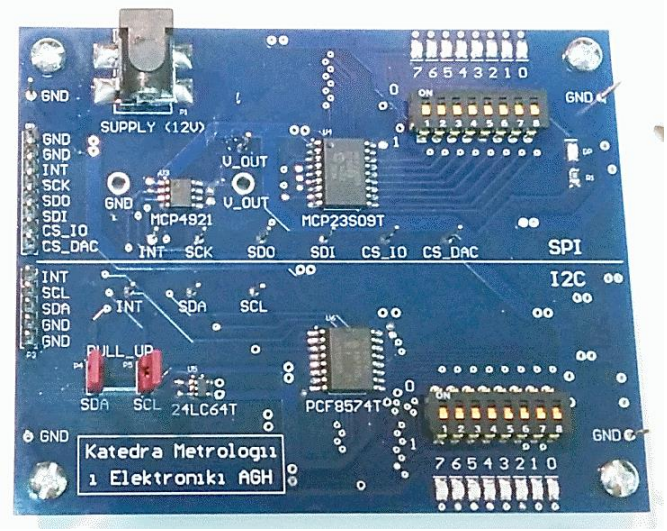
Stany na wejściach bramy można ustalać za pomocą przełączników typu *dipswith* (żółte przełączniki).
Stany na wyjściach bramy można obserwować za pomocą paska LED-ów przy dipswitchach.

Do magistrali I2C podłączone są 8-bitowa brama wejścia/wyjścia (PCF8574) oraz pamięć EEPROM (24LC64T).

Stany na wejściach bramy można ustalać za pomocą przełączników typu *dipswith* (żółte przełączniki).
Stany na wyjściach bramy można obserwować za pomocą paska LED-ów przy dipswitchach.

Linie adresowe bramy podłączone są do potencjału GND.

Obie linie magistrali są „pociągane” do napięcia zasilającego za pomocą rezystorów poprzez zworki.



CAN

1. Zestaw jest oparty na mikrokontrolerze LPC2129 będącym wersją mikrokontrolera używanego na MiTP oraz do SPI/I2C ale zawierającego dwa moduły CAN.

2. Przełączanie mikrokontrolera między trybami programowania i komunikacji odbywa się za pomocą zworek ISP/IAP i RST_EN, tj:

- Programowanie: obie w lewo patrząc od strony przycisku resetu
- Komunikacja: obie w prawo

4. Do testów sprawności EVM-ów należy użyć programów : Can1_to_Can2.hex, Can2_to_Can1.hex (załączniki na stronie przedmiotu)

Przed uruchomieniem programów połączyć gniazda CAN1 i CAN2 za pomocą kabla z trzeba wtyczkami żeńskimi.

Po wgraniu każdego z programów powinien przesuwać się punkt świetlny.