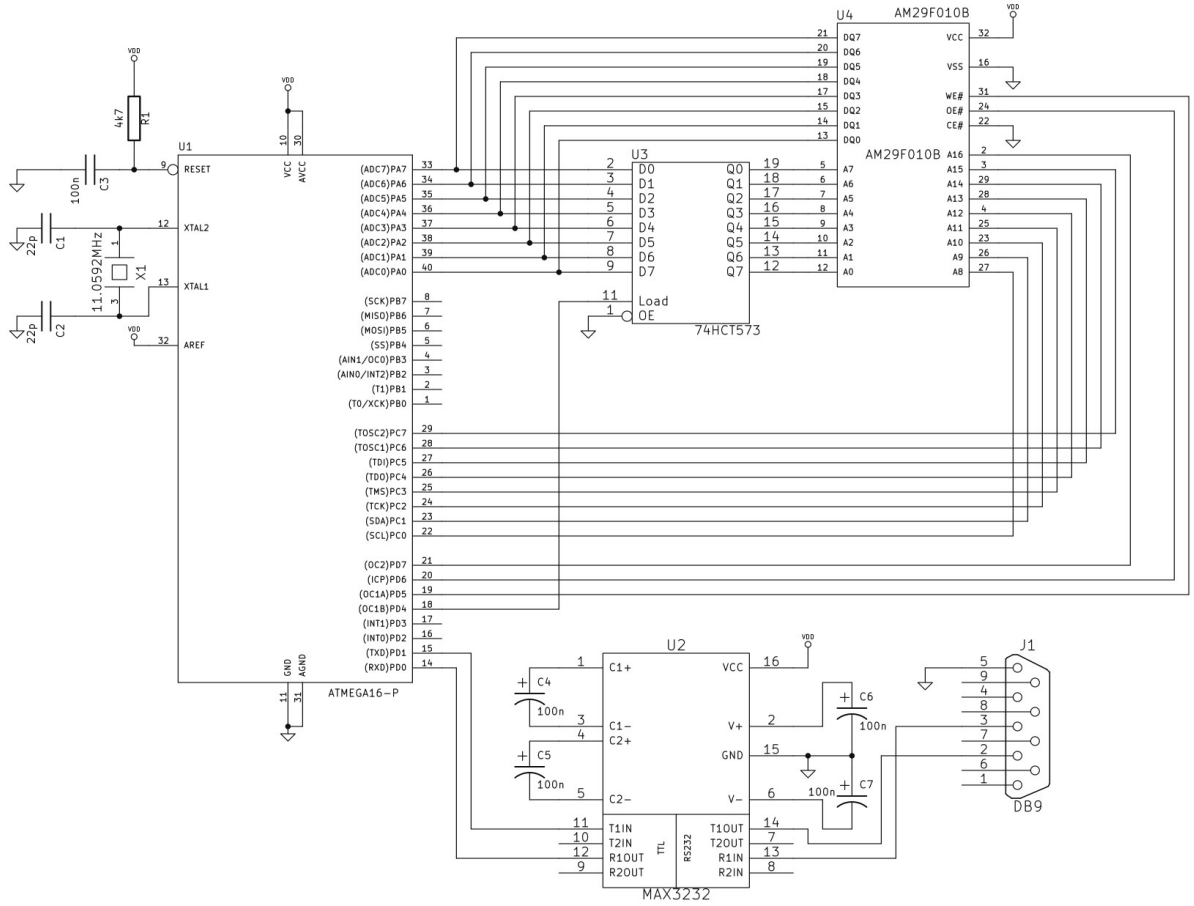
Laboratorium 11.

Równoległa pamięć FLASH AM29F010B

Ćwiczenie ma na celu zapoznanie użytkownika ze sposobem multipleksowania szyny adresowej i szyny danych w celu wykorzystania równoległych pamięci w systemach mikroprocesorowych.

Część 1:

Celem minimalizacji wykorzystania linii w mikrokontrolerze przy wykorzystaniu pamięci równoległych stosuje się metodę multipleksowania szyny danych i adresowej z wykorzystaniem zatrzasku typu D. W naszym przypadku rolę tą spełnia układ 74HCT573.



Schemat 1: Podłączenie pamięci równoległej FLASH AM29F010B do mikrokontrolera

W momencie gdy na linii LE - Latch Enable (*Load* na schemacie) wysterowanej linią PD4 mikrokontrolera, pojawi się stan niski, stan na liniach Q0-Q7 zostanie zatrzaśnięty (zapamiętany). Gdy na linii LE panuje stan wysoki, na liniach Q0-Q7 występują stany linii D0-D7.

Zatrzaśnięcie młodszej części adresu w zatrzasku pozwala wykorzystać 8 linii portu PB do transmisji danych z albo do pamięci. Starsza część adresu wystawiona jest w tym przypadku na porcie PC, a najstarszy bit adresu ustawiany jest z linii PD7 mikrokontrolera (cały adres jest 17-bitowy). Do sterowania transmisją danych do i z pamięci wykorzystywane są linie *WE#* - stan niski załącza zapis do pamięci, *OE#* - stan niski załącza odczyt z pamięci, które sterowane są liniami PD5 i PD6 mikrokontrolera.

Połączyć układ wg schematu 1. Do układu U3 (zatrzask) podłączyć zasilanie +5V na nóżkę nr 20 (nie zaznaczone na schemacie) – patrz nota katalogowa 74HC573.

Część 2:

Zapoznać się z notą katalogową układu AM29F010B.
Przygotować procedury odczytu, zapisu, kasowania sektora oraz całej pamięci układu AM29F010B, oraz odczytu sygnatury. Wykorzystać magistralę szeregową RS232 do testowania przesyłania danych z komputera do pamięci AM29F010B oraz do odczytu z pamięci AM29F010B do komputera. Korzystając z Terminal'a przetestować działanie obsługi pamięci.

UWAGA: Ze względu na swój reżim pracy pamięci FLASH wymagane jest zastosowanie niewielkiego opóźnienia w procedurze odczytu. Po ustawieniu w stan niskii linii OE# wystarczające jest odczekanie dwóch taktów zegara mikrokontrolera ATmega16 z kwarcem 11.0592MHz. Można to zrealizować stosując dwie assemblerowe wstawki w programie z jedno cyklowymi instrukcjami, która niczego nie zmieniają: NO OPERATION. Wstawka assemblerowa ma postać:

```
asm("NOP");
```

Komendy wysyłane przez RS232 powinny zapewnić conajmniej:
`id` – zwrot informacji o pamięci (Manufacturer ID, Chip ID)
`erase` – kasowanie całej pamięci flash
`serase` – kasowanie sektora SA0
`program` <dane> – zaprogramowanie pamięci flash ciągiem znaków <dane> - z bufora odczytu RS232. Zainicjować bufor o wielkości 200 znaków.
`read` – odczyt 200 bajtów danych z pamięci

Przykładowy plik nagłówkowy `amflash.h`

```
#ifndef AMFLASH_H_
#define AMFLASH_H_

#include <avr/io.h>
#include <inttypes.h>
#include <util/delay.h>
#include <stdio.h>

#define AM_DATA_P       PORTA
#define AM_DATA_IN      PINA
#define AM_DATA_P_DIR   DDRA

#define AM_A0_P         AM_DATA_P
#define AM_A0_P_DIR     AM_DATA_P_DIR

#define READ_DIR        AM_A0_P_DIR = 0x00
#define WRITE_DIR       AM_A0_P_DIR = 0xff

#define AM_A8_P         PORTC    //Address high (A8-A15)
#define AM_A8_P_DIR     DDRC

#define AM_A16_P        PORTD    //Address most significant bit port & dir
#define AM_A16_P_DIR    DDRD
#define AM_A16          PD7      //Most significant bit (A16)

#define AM_LATCH_P      PORTD
#define AM_L            PD4
```

```
#define ADR_LATCH            AM_LATCH_P &= ~(1 << AM_L)
#define ADR_UNLATCH          AM_LATCH_P |= (1<< AM_L)

#define AM_WE                PD5
#define AM_OE                PD6
#define AM_WE_HI             AM_LATCH_P |= (1 << AM_WE)
#define AM_WE_LO             AM_LATCH_P &= ~(1 << AM_WE)
#define AM_OE_HI             AM_LATCH_P |= (1 << AM_OE)
#define AM_OE_LO             AM_LATCH_P &= ~(1 << AM_OE)

// AM29F010B
// 8 blocks x 16kB sectors
#define SA0       0x00000
#define SA1       0x04000
#define SA2       0x08000
#define SA3       0x0C000
#define SA4       0x10000
#define SA5       0x14000
#define SA6       0x18000
#define SA7       0x1C000

// autoselect codes
#define CHIP_MANUF_ADDR      0x00
#define CHIP_ID_ADDR         0x01
#define CHIP_PROT_ADDR       0x02  // | SAx

#define CHIP_AMD             0x01
#define     CHIP_ID          0x20
#define SECTOR_PROT          0x01
#define SECTOR_UNPROT        0x00

// status results
#define FLASH_PASS      0
#define FLASH_FAIL      1

//-----------------------------------------------------------------------------
//            AMD AM29F010B
//-----------------------------------------------------------------------------

// status
#define DQ7_POLLING     0x80
#define DQ6_TOGGLE      0x40
#define DQ5_TIMEOUT     0x20
#define DQ3_ETIMER      0x08

// unlock
#define CMD1_ADDR       0x555
#define CMD1_UNLOCK     0xAA
#define CMD2_ADDR       0x2AA
#define CMD2_UNLOCK     0x55
// special addresses
#define CMD3_ADDR       CMD1_ADDR
#define CMD4_ADDR       CMD1_ADDR
#define CMD5_ADDR       CMD2_ADDR
#define CMD6_ADDR       CMD1_ADDR

// commands
#define CMD3_RESET           0xF0
#define CMD3_AUTOSELECT      0x90
#define CMD3_PROGRAM         0xA0
```

```
#define CMD3_ERASE           0x80
#define CMD4_ERASE           CMD1_UNLOCK
#define CMD5_ERASE           CMD2_UNLOCK
#define CMD6_ERASE_CHIP      0x10
#define CMD6_ERASE_SECT      0x30
#define CMD1_ERASE_SUSP      0xB0
#define CMD1_ERASE_RESU      0x30

void AM_init(void);
void AM_write(uint32_t addr, uint8_t data);
uint8_t AM_read(uint32_t addr);
void AM_set_addr(uint32_t addr);
void AM_cmd_unlock(void);
void AM_reset(void);
uint8_t AM_autoselect(uint32_t addr);
void AM_program(uint32_t addr, uint8_t data);
void AM_chip_erase(void);
void AM_sector_erase(uint32_t addr);
uint8_t AM_data_polling(uint32_t addr, uint8_t data);

#endif /* AMFLASH_H_ */
```