Лабораторна робота №8

Протоколи підключення мікроконтролера ATmega128

Мета: Дослідити основні протоколи підключення мікроконтролера **ATmega128**

8.1 Теоретичні відомості

UART означає універсальний асинхронний приймач/передавач і визначає протокол або набір правил для обміну послідовними даними між двома пристроями. UART дуже простий і використовує лише два дроти між передавачем і приймачем для передачі та прийому в обох напрямках. Обидва кінці також мають заземлення. Комунікація в UART може бути симплексною (дані надсилаються лише в одному напрямку), напівдуплексною (кожна сторона розмовляє, але лише по одній) або повнодуплексною (обидві сторони можуть передавати одночасно). Дані в UART передаються у вигляді кадрів. Коротко описано та пояснено формат і зміст цих кадрів.

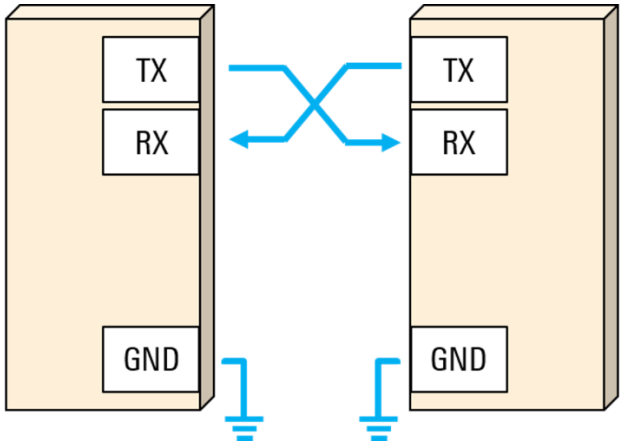


Рис.8.1 – UART схема підключення.

UART був одним із найперших послідовних протоколів. Колись повсюдні послідовні порти майже завжди базуються на UART, а пристрої, що використовують інтерфейси RS-232, зовнішні модеми тощо, є типовими прикладами використання UART.
В останні роки популярність UART зменшилася: такі протоколи, як SPI та I2C, замінюють UART між мікросхемами та компонентами. Замість зв'язку через послідовний порт більшість сучасних комп'ютерів і периферійних пристроїв тепер використовують такі технології, як Ethernet і USB. Однак UART все ще використовується для додатків з меншою швидкістю та меншою пропускною здатністю, оскільки він дуже простий, недорогий і легкий у реалізації.

Таймінг і синхронізація протоколів UART

Одна з великих переваг UART полягає в тому, що він асинхронний – передавач і приймач не мають спільного тактового сигналу. Хоча це значно спрощує протокол, воно висуває певні вимоги до передавача та приймача. Оскільки вони не мають спільного тактового сигналу, обидва кінці повинні передавати з однаковою, заздалегідь обумовленою швидкістю, щоб мати однакову синхронізацію бітів. Найпоширеніші швидкості передачі даних UART, які використовуються сьогодні, — 4800, 9600, 19,2 КБ, 57,6 КБ і 115,2 КБ. Окрім однакової швидкості передачі даних, обидві сторони з'єднання UART також повинні використовувати однакову структуру кадру та параметри. Найкращий спосіб зрозуміти це — подивитися на кадр UART.

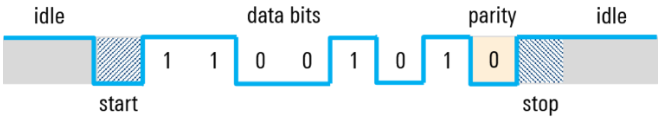


Рис.8.2 – Форма кадру UART

Як і в більшості цифрових систем, «високий» рівень напруги використовується для позначення логічної «1», а «низький» рівень напруги використовується для позначення логічного «0». Оскільки протокол UART не визначає конкретних напруг або діапазонів напруг для цих рівнів, інколи високий рівень також називають «міткою», а низький — «пробілом». Зауважте, що в стані очікування (коли дані не передаються) лінія утримується на високому рівні. Це дозволяє легко виявити пошкоджену лінію або передавач.

Біти запуску та зупинки

Оскільки UART є асинхронним, передавач повинен сигналізувати про надходження бітів даних. Це досягається за допомогою стартового біта. Початковий біт — це перехід від високого стану очікування до низького стану, за яким одразу слідують біти даних користувача. Після завершення обробки бітів даних стоп-біт вказує на кінець даних користувача. Стоп-біт - це або перехід назад у високий стан або стан очікування, або перебування у високому стані на додатковий бітовий час. Другий (необов'язковий) стоп-біт може бути налаштований, як правило, щоб дати приймачу час підготуватися до наступного кадру, але це рідко зустрічається на практиці.

Біти даних

Біти даних є даними користувача або «корисними» бітами і йдуть відразу після початкового біта. Може бути від 5 до 9 біт даних користувача, хоча найчастіше використовуються 7 або 8 біт. Ці біти даних зазвичай передаються з молодшим бітом першим.

Приклад: якщо ми хочемо надіслати велику літеру «S» у 7-бітному ASCII, послідовність бітів буде 1 0 1 0 0 1 1. Спочатку ми змінюємо порядок бітів, щоб розмістити їх у порядку молодших бітів, тобто 1 1 0 0 1 0 1, перш ніж їх розсилати. Після надсилання останнього біта даних стоп-біт використовується для завершення кадру, і лінія повертається в стан очікування.

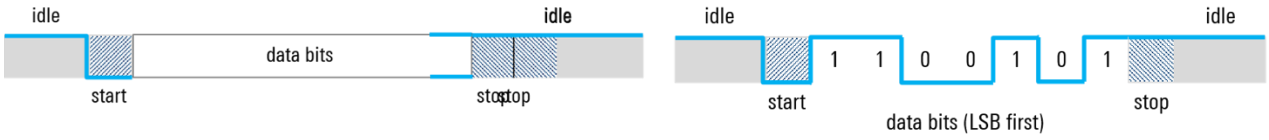


Рис.8.3 – Біти запуску та зупинки

Кадр UART також може містити додатковий біт парності, який можна використовувати для виявлення помилок. Цей біт вставляється між кінцем бітів даних і стоп-бітом. Значення біта парності залежить від типу паритету, який використовується (парний чи непарний):

У парній парності цей біт встановлюється так, що загальна кількість одиниць у кадрі буде парною.

У непарності цей біт встановлюється так, що загальна кількість одиниць у кадрі буде непарною.

Приклад: велика буква «S» (1 0 1 0 0 1 1) містить три нулі та 4 одиниці. Якщо використовується парна парність, біт парності дорівнює нулю, оскільки вже є парна кількість одиниць. Якщо використовується непарна парність, то біт парності має бути одиницею, щоб кадр мав непарну кількість 1с.

Біт парності може виявити лише один перевернутий біт. Якщо перевернуто більше ніж один біт, неможливо надійно виявити їх за допомогою одного біта парності.

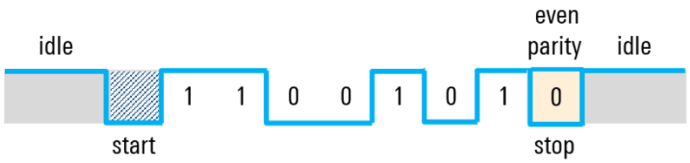


Рис.8.3 – Приклад біта парності.

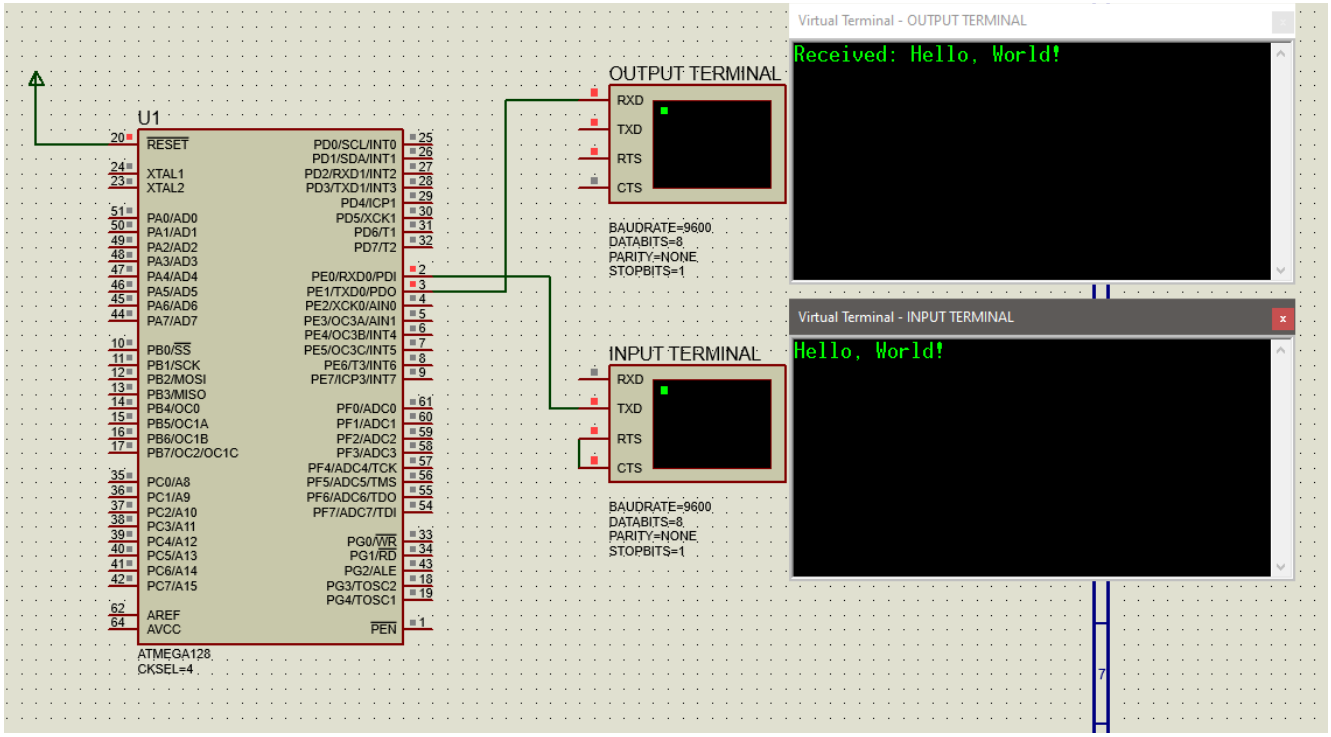


Рис.8.4 – Передача повідомлення за допомогою UART протоколу між двома терміналами.

BAUD rate (частота передачі) — це швидкість, з якою дані передаються або приймаються через UART (Universal Asynchronous Receiver/Transmitter). Вона визначається кількістю **бітів на секунду (bps)**. Наприклад, **9600 baud** означає, що 9600 бітів передаються за одну секунду.

Синхронізація: Передача даних через UART є асинхронною, тобто немає спільного тактового сигналу між пристроями (наприклад, між мікроконтролером і ПК). Замість цього, передавач і приймач повинні працювати з однаковою BAUD rate, щоб коректно обробляти дані.

На мікроконтролерах AVR, таких як ATmega128, налаштування BAUD rate здійснюється через регістр **UBRR** (USART Baud Rate Register).

$$UBRR = \frac{F_{\text{osc}}}{16 \cdot \text{BAUD}} - 1$$

де Fosc — частота тактового сигналу мікроконтролера (наприклад, 8 МГц або 16 МГц), BAUD — бажана частота передачі даних.

SPI (англ. Serial Peripheral Interface) - це послідовний синхронний інтерфейс, який є чотирьох-провідним, та призначений для повнодуплексного обміну даними між ведучим та підлеглими пристроями. Інтерфейс був розроблений компанією Motorola в середині 1980-х років.

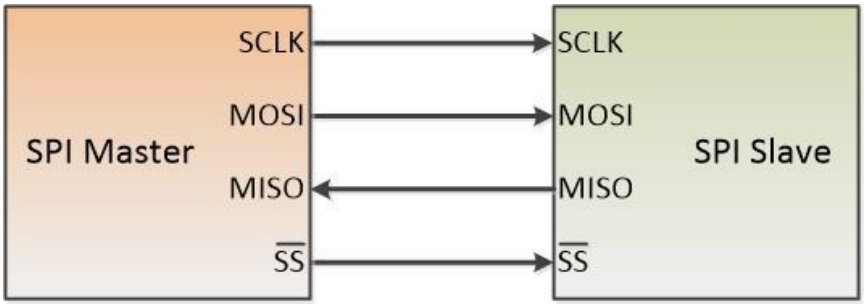
Інтерфейсна шина SPI використовується для передачі даних між мікроконтролерами та периферійними пристроями, такими як регістри зсуву, датчики, оперативна та flash пам'ять, АЦП, ЦАП, SD-карти та ін. Шина SPI використовує окрему лінію синхронізації та лінії даних.

У протоколі SPI є два типи пристроїв: Master (ведучий) та Slave (підлеглий, відомий). Ведучий пристрій є головним на шині (зазвичай це мікроконтролер), а всі інші пристрої (периферійні мікросхеми чи навіть інші мікроконтролери) вважаються підлеглими (веденими).

Ведучий пристрій генерує тактовий сигнал синхронізації (**SCLK**), по якому синхронізується передача вхідних та вихідних даних. На схемі вхідний та вихідний сигнали позначаються MOSI та MISO:

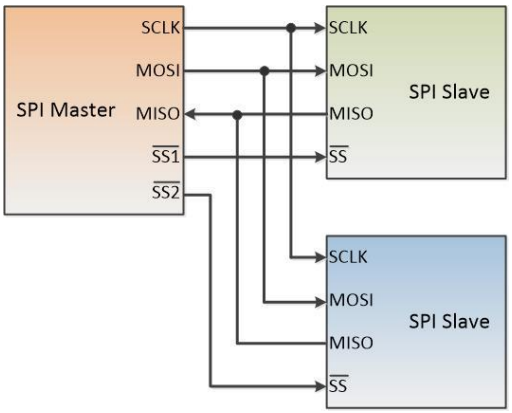
MOSI (англ. Master Out Slave In) — вихід ведучого, вхід підлеглого (веденого). Дані передаються від ведучого пристрою до підлеглого.

MISO (англ. Master In Slave Out) — вхід ведучого, вихід підлеглого (веденого). Дані передаються від підлеглого пристрою до ведучого.



Також на шині практично завжди присутній сигнал **SS** або **CS** (англ. Chip Select або Slave Select), який призначений для вибору підлеглого (веденого) пристрою. Зазвичай це активний низький сигнал, який стає високим при завершенні сеансу зв'язку. Коли є кілька підлеглих

пристроїв, то ведучий пристрій має окремий вихід цього сигналу для кожного підлеглого пристрою. Наприклад, нижче на рисунку ми маємо два сигнали вибору веденого пристрою SS1 та SS2 для двох периферійних пристроїв:



Також багато виробників називають ведучий пристрій **контролером**, а підлеглий - **периферійним** пристроєм. При цьому MISO буде називатись **POCI**, а MOSI - **PICO**.

Оскільки сигнал синхронізації генерується головним, то потік даних контролюється головним. Протягом одного тактового сигналу одночасно передається один біт даних від головного до підлеглого пристрою (по MOSI) та один біт від підлеглого до головного (по MISO). Таким чином в обох напрямках передається один байт даних за один сеанс передачі. Отже, SPI є повнодуплексним зв’язком.

Якщо що кілька підлеглих під'єднані до ведучого в шині SPI, тільки один підлеглий буде активним. Головний пристрій використовує ніжку SS для вибору відповідного підлеглого пристрою.

На апаратному рівні передача зазвичай включає два регістри зсуву по вісім бітів на головному і підлеглому пристроях. Обидва регістри зсуву з'єднані, щоб утворити цикл. Головний пристрій має також генератор тактового сигналу синхронізації. Дані, як правило, зсуваються зі першим старшим бітом (MSB).

Передача здійснюється пакетами. Довжина пакета, як правило, становить 1 байт (8 біт). Перед початком передачі, дані поміщають у зсувні регістри. Після цього провідний пристрій починає генерувати імпульси синхронізації лінії SCLK, що призводить до взаємного обміну даними. Передача даних здійснюється біт за бітом від ведучого по лінії MOSI та від веденого по лінії MISO доти, поки не буде передано усі вісім біт. Якщо потрібно обмінятися додатковими даними, регістри зсуву перезавантажуються, і процес повторюється.

Проте у більшості випадків SPI працює шляхом передачі ведучим пристроєм команд, а периферійним пристроєм - відповідей. Нижче показана діаграма сигналів, де контролер висилає команду 0x53, а периферійний пристрій відсилає число 0x46 - відповідь на команду. У SPI контролер може виставляти рівень (високий або низький) тактового сигналу синхронізації SCLK та фазу синхронізації даних. Рівень тактового сигналу в стані очікування виставляється параметром CPOL. Параметр CPHA виставляє вибірку даних по наростаючому або спадному фронту синхронізації.

Коли полярність тактового сигналу встановлена на низьку (0), а фаза тактового сигналу встановлена на «перший фронт», тоді маємо режим 0. У цьому випадку дані зміщуються на лінію MOSI, коли SCLK піднімається з низького (0) до високого (1). У режимі 3 фаза синхронізації

також налаштована на зміщення даних на лінію MOSI на «першому фронті», але в цьому випадку полярність SCLK висока (1), що означає, що дані повинні бути зміщені, коли SCLK падає з високого (1) на низький (0).

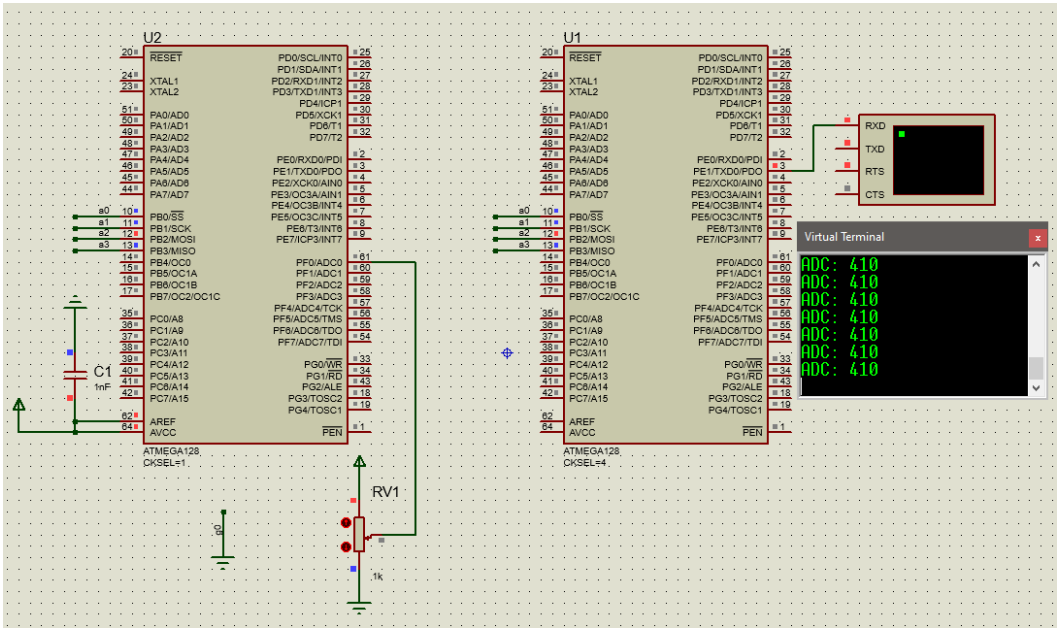


Рис.8.5 – Схема підключення за допомогою SPI протоколу.

8.2 Порядок виконання лабораторної роботи

1. На основі UART протоколу реалізувати підключення 2х терміналів (рис8.4)
2. Здійснити передачу повідомлення у вигляді: «Прізвище І.П., Група-Номер групи»(додаток 1) за допомогою UART протоколу, де 1 термінал виконує роль введення, а інший виведення інформації.
3. Використати 2 мікрочіпа ATmega128 та підключити їх за допомогою SPI протоколу(рис.8.5)
4. Налаштувати на одному мікрочіпі(master) алгоритм відправлення інформації.(Додаток 2)
5. По комунікаційним каналам передати данні з зміного резистору.
6. Налаштувати на іншому мікрочіпі (slave) алгоритм отримання повідомлення. (Додаток 3)
7. Вивести за допомогою UART протоколу на термінал отриману інформацію у вигляді: «Прізвище І.П., Група-Номер групи, ADC: значення з АЦП». Наприклад: (Petrenko I.O. IVTK-014, ADC: 576). Значення АЦП на власний розсуд.

8.3 Зміст звіту

1. Найменування і мета роботи.
2. Схема підключення через UART протокол та симуляція роботи.
3. Схема підключення через SPI протокол та симуляція роботи.
4. Висновки по роботі.

8.4 Контрольні запитання

1. Що таке UART і як працює цей протокол?
2. Які параметри необхідно налаштувати для коректної роботи UART?
3. Що відбувається, якщо передавач і приймач мають різні BAUD rate?
4. Що таке SPI і в яких випадках його застосовують?

5. Які ролі мають пристрої Master і Slave в SPI?
6. Як організувати обмін даними між мікроконтролером і периферійним пристроєм через SPI?

```
#define F_CPU 8000000UL // Частота мікроконтролера
#include <avr/io.h>
#include <util/delay.h>

// Ініціалізація UART
void UART_init(unsigned int baud) {
	unsigned int ubrr = F_CPU / 16 / baud - 1; // Розрахунок UBRR
	UBRR0H = (unsigned char)(ubrr >> 8);
	UBRR0L = (unsigned char)ubrr;
	UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Увімкнення приймача і передавача
	UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // Формат кадру: 8 біт даних, 1 стоп-біт
}

// Передача символу через UART
void UART_transmit(unsigned char data) {
	while (!(UCSR0A & (1 << UDRE0))); // Очікування готовності до передачі
	UDR0 = data; // Передача даних
}

// Передача рядка через UART
void UART_sendString(const char *str) {
	while (*str) {
		UART_transmit(*str++); // Надсилаємо символи рядка
	}
}

// Прийом символу через UART
unsigned char UART_receive(void) {
	while (!(UCSR0A & (1 << RXC0))); // Очікування отримання даних
	return UDR0; // Повертаємо отриманий символ
}

// Прийом рядка через UART
void UART_receiveString(char *buffer, unsigned int bufferSize) {
	unsigned int i = 0;
	while (i < bufferSize - 1) { // Залишаємо місце для нуль-термінатора
		char receivedChar = UART_receive();
		if (receivedChar == '\r' || receivedChar == '\n') { // Завершення рядка
			break;
		}
		buffer[i++] = receivedChar;
	}
	buffer[i] = '\0'; // Додаємо нуль-термінатор
}

int main(void) {
	UART_init(9600); // Ініціалізація UART з швидкістю 9600 бод

	char buffer[128]; // Буфер для отримання рядків

	while (1) {
		// Отримуємо рядок
		UART_receiveString(buffer, sizeof(buffer));

		// Відправляємо отриманий рядок назад
		UART_sendString("Received: ");
		UART_sendString(buffer);
		UART_sendString("\r\n");

		_delay_ms(1000); // Затримка для зручності
	}

	return 0;
}
```

```
#include <avr/io.h>
#include <util/delay.h>

#define SS_PIN PB0

void SPI_MasterInit(void) {
	// Встановлюємо MOSI, SCK, SS як виходи
	DDRB = (1 << PB2) | (1 << PB1) | (1 << SS_PIN);
	// Встановлюємо MISO як вхід
	DDRB &= ~(1 << PB3);
	// Увімкнення SPI, Master, SCK = fosc/16
	SPCR = (1 << SPE) | (1 << MSTR);
	// Активуємо Slave
	PORTB &= ~(1 << SS_PIN);
}

void SPI_MasterTransmit(uint16_t data) {
	// Передаємо старший байт
	SPDR = (data >> 8);  // Відокремлюємо старший байт
	while (!(SPSR & (1 << SPIF)));  // Очікуємо завершення передачі

	// Передаємо молодший байт
	SPDR = (data & 0xFF);  // Відокремлюємо молодший байт
	while (!(SPSR & (1 << SPIF)));  // Очікуємо завершення передачі
}

uint16_t ADC_Read(uint8_t channel) {
	// Вибір каналу
	ADMUX = (ADMUX & 0xF8) | channel;
	// Запуск перетворення
	ADCSRA |= (1 << ADSC);
	// Очікуємо завершення
	while (ADCSRA & (1 << ADSC));
	// Повертаємо результат
	return ADC;
}

void ADC_Init(void) {
	ADMUX = (1 << REFS0);  // Опорна напруга AVCC
	ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);  // Включення АЦП, дільник 64
}

int main(void) {
	SPI_MasterInit();
	ADC_Init();

	while (1) {
		uint16_t adc_value = ADC_Read(0);  // Зчитуємо аналоговий сигнал
		SPI_MasterTransmit(adc_value);  // Передаємо 16-бітне значення
		_delay_ms(100);
	}
}
```

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

// Ініціалізація SPI (Slave)
void SPI_SlaveInit(void) {
	// Встановлюємо MISO як вихід
	DDRB = (1 << PB3);
	// Увімкнення SPI
	SPCR = (1 << SPE);
}

// Отримання даних через SPI
uint16_t SPI_SlaveReceive(void) {
	uint16_t data = 0;

	// Отримуємо старший байт
	while (!(SPSR & (1 << SPIF)));  // Очікуємо завершення прийому
	data = SPDR << 8;  // Зберігаємо старший байт

	// Отримуємо молодший байт
	while (!(SPSR & (1 << SPIF)));  // Очікуємо завершення прийому
	data |= SPDR;  // Додаємо молодший байт

	return data;
}

// Ініціалізація UART
void UART_Init(uint16_t ubrr) {
	// Встановлення швидкості передачі
	UBRR0H = (uint8_t)(ubrr >> 8);
	UBRR0L = (uint8_t)ubrr;

	// Увімкнення передавача
	UCSR0B = (1 << TXEN0);

	// Налаштування формату даних: 8 біт, 1 стоп-біт
	UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

// Передача символу через UART
void UART_Transmit(char data) {
	while (!(UCSR0A & (1 << UDRE0)));  // Очікуємо готовності до передачі
	UDR0 = data;  // Передаємо дані
}

// Передача рядка через UART
void UART_TransmitString(char *str) {
	while (*str) {
		UART_Transmit(*str++);
	}
}

int main(void) {
	SPI_SlaveInit();
	UART_Init(51);

	uint16_t adc_value = 0;
	char buffer[16];

	while (1) {
		adc_value = SPI_SlaveReceive();  // Отримуємо 16-бітне значення

		// Формуємо текст для виведення
		sprintf(buffer, "ADC: %d\r\n", adc_value);
		UART_TransmitString(buffer);  // Виводимо у віртуальний термінал

		_delay_ms(500);  // Затримка для зручності перегляду
```

```
        }
}
```