Kasper's Generic ATmega328P API (KGAA)

Still work in progress!

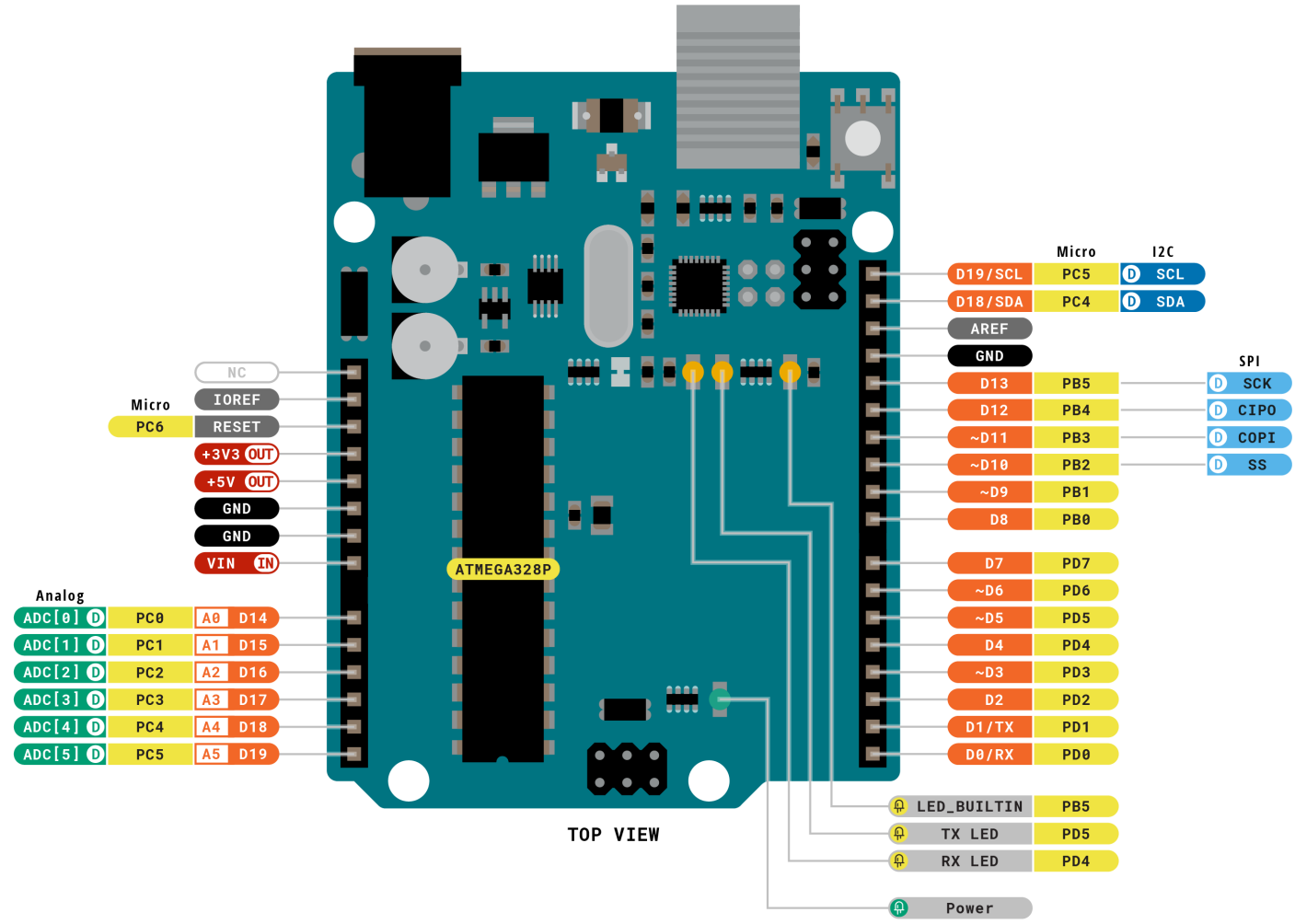
Download

Introduction

This is a generic API for the ATmega328P microcontroller. It is written in C and is intended to be used with the Arduino Uno. The API is written to be as simple as possible, while still being powerful enough to be used in most projects.

Table of Contents

Arduino Uno pinout



Legend:
Digital
I2C
Power
Analog
SPI
Ground
Main Part
Analog

ARDUINO

ARDUINO UNO REV3
SKU code: A000066
Pinout
Last update: 6 Oct, 2022

Source

Getting Started

Basic blink example

This example will blink the LED on pin 13 (PB5) on the Arduino Uno every second.

Code

```
void setup() // This function is called once when the program starts
{
    setPinMode(LED_PORT, LED_PIN, OUTPUT); // Set the LED pin mode to OUTPUT

    setPinState(LED_PORT, LED_PIN, LOW); // Set the LED pin state to LOW
}

bool loop() // This function is called every time the program loops
{
    togglePinState(LED_PORT, LED_PIN); // Toggle the LED pin state

    delayMs(1000); // Delay for 1000 milliseconds

    return true; // Return true to keep looping
}
```

Explanation

In this example, we are first setting the LED pin mode to OUTPUT. Then we are setting the LED pin state to LOW. Then we are toggling the LED pin state. Then we are delaying for 1000 milliseconds and returning true to keep looping.

Basic analog read example

This example will read the analog value of pin A0 (PC0) on the Arduino Uno every second. Then it prints the value to the serial monitor.

Code

```
void setup() // This function is called once when the program starts
{
    initADC(ADC_PRESCALER_128); // Initialize the ADC with a prescaler of 128

    initSerial(9600); // Initialize the serial with a baud rate of 9600
}

bool loop() // This function is called every time the program loops
{
    uint16_t adcValue = getAnalogValue(0); // Read the ADC value of pin A0

    char buffer[SERIAL_BUFFER_SIZE]; // Create a buffer to store the string

    sprintf(buffer, "ADC value: %d\n", adcValue); // Format the string

    writeToSerial(buffer); // Write the string to the serial

    delayMs(1000); // Delay for 1000 milliseconds

    return true; // Return true to keep looping
}
```

Explanation

In this example, we enable the ADC and the Serial. Then we read the analog value of pin A0 and store it in a variable. Then we create a buffer to store the string we want to print to the serial. Then we format the string and write it to the serial. Then we delay for 1000 milliseconds and return true to keep looping.

Basic semi-dynamic array example

This example shows how to use the semi-dynamic array functions. In this case a semi-dynamic array of `char` is used to generate a string. The string is then printed to the serial monitor.

Code

```
void setup() // This function is called once when the program starts
{
    initSerial(9600); // Initialize the serial with a baud rate of 9600

    charArray_t charArray; // Create a charArray_t variable

    if (initCharArray(&charArray, "Hello, world!\0", 14)) // Initialize the charArray_t variable
    {
        writeToSerial(charArray.data); // Write the string to the serial
    }
    else
    {
        writeToSerial("Error initializing charArray!\n"); // Write an error message to the serial
    }

    freeCharArray(&charArray); // Free the charArray_t variable
}
```

Explanation

In this example, we initialize the serial with a baud rate of 9600. Then we create a `charArray_t` variable. Then we initialize the `charArray_t` variable with the string `"Hello, world!\0"` and a length of 14. Then we check if the initialization was successful. If it was, we write the string to the serial. If it wasn't, we write an error message to the serial. Then we free the `charArray_t` variable.

Analog

Initialization of the ADC (Analog to Digital Converter)

To initialize the ADC, call the `initADC(ADC_Prescaler_t)` function. The function has one parameter, which is the ADC Prescaler.

ADC prescaler

You can set the prescaler when initializing the ADC. The prescaler can be set to 2, 4, 8, 16, 32, 64 or 128. The default setting is 128, which will give a 125 kHz ADC clock frequency when using the 16 MHz system clock.

A faster ADC clock frequency will result in a higher resolution, but will also increase the power consumption. A slower ADC clock frequency will result in a lower resolution, but will also decrease the power consumption.

Disabling the ADC

To disable the ADC, call the `disableADC()` function.

Getting an analog value

To get the ADC, call the `getAnalogValue(uint8_t)` function. The function has one parameter, which is the (channel) pin number of the ADC pin you want to read. The function will return the ADC value as an `uint16_t` value.

Example

```
void setup()
{
    initADC(ADC_PRESCALER_128); // Initialize the ADC with a prescaler of 128
    // Some code...
}

bool loop()
{
    uint16_t adcValue = getAnalogValue(0); // Read the ADC value of pin A0
    // Some code...
}
```

Macros explained

Macro	Value	Description
ADC	0x78 or 01111000	The ADC Data Register (ADC) stores the result of the most recent ADC conversion. Once the conversion is complete, the digital value representing the analog input signal is placed in this register, making it available for further processing or retrieval.
ADCSRA	0x7A or 01111010	The ADC Control and Status Register A (ADCSRA) is a critical register that governs the operation of the ADC. It enables or disables the ADC, triggers conversions, sets the prescaler for ADC clock division, and enables ADC interrupt. Additionally, it is used to check if the ADC has finished converting.
ADEN	0x80 or 10000000	ADC Enable Bit (ADEN) is a bit in the ADCSRA register. Setting this bit enables the ADC. When ADEN is set, the ADC is enabled. When ADEN is cleared, the ADC is disabled.
ADSC	0x40 or 01000000	ADC Start Conversion Bit (ADSC) is a bit in the ADCSRA register. Setting this bit initiates an ADC conversion. When ADSC is set, the ADC begins sampling and converting the analog input signal to a digital value.
ADMUX	0x7C or 01111100	The ADC Multiplexer Selection Register (ADMUX) is responsible for selecting the input channel for the Analog-to-Digital Converter (ADC). A multiplexer, in this context, is a device that allows you to choose one of several analog signals and route it to the ADC for conversion.
REFS0	0x40 or 01000000	Reference Selection Bit 0 (REFS0) is a bit in the ADMUX register. It is used to specify the reference voltage source for the ADC. In this case, when REFS0 is set, the reference voltage is derived from the AVCC pin, which is often connected to the supply voltage.
RXEN0	0x04 or 00000100	Receiver Enable Bit (RXEN0) is a bit in the UCSR0B register. Setting this bit enables the USART receiver. When RXEN0 is set, the USART receiver is enabled. When RXEN0 is cleared, the USART receiver is disabled.
TXEN0	0x03 or	Transmitter Enable Bit (TXEN0) is a bit in the UCSR0B register. Setting this bit enables the USART transmitter. When TXEN0 is set, the

Macro	Value	Description
	00000011	USART transmitter is enabled. When TXEN0 is cleared, the USART transmitter is disabled.
UCSZ00	0x01 or 00000001	USART Character Size Bit 0 (UCSZ00) is a bit in the UCSR0C register. It is used to specify the number of data bits to be transmitted or received. In this case, when UCSZ00 is set, the USART will transmit or receive 8-bit characters.
UCSZ01	0x02 or 00000010	USART Character Size Bit 1 (UCSZ01) is a bit in the UCSR0C register. It is used to specify the number of data bits to be transmitted or received. In this case, when UCSZ01 is set, the USART will transmit or receive 8-bit characters.
UCSR0B	0x1C or 00011100	The USART Control and Status Register B (UCSR0B) is a critical register that governs the operation of the USART. It enables or disables the USART, enables the receiver and transmitter, and enables USART interrupt. Additionally, it is used to check if the USART has finished transmitting or receiving.
UCSR0C	0x2C or 00101100	The USART Control and Status Register C (UCSR0C) is a critical register that governs the operation of the USART. It sets the USART mode of operation, sets the number of data bits, sets the parity mode, and sets the number of stop bits.

Delay

For now, the only delay function is `delayMs(uint16_t)`, which delays the program for the specified amount of milliseconds. The function has one parameter, which is the amount of milliseconds to delay.

This is a blocking function, which means that the program will not do anything else while the function is running.

Example

```
void setup()
{
    // Some code...
}

bool loop()
{
    // Some code...
    delayMs(1000); // Delay for 1000 milliseconds
}
```

Digital

Main

Serial