UnoLib documentation

digital.pas

version 11/10/2025

The digital.pas module contains low-level constants for digital pins. Some routines were originally preprocessor functions (macros) in the Arduino core library, but because FPC doesn't support such routines, they were translated as regular procedures or functions. Consequently, the binary code produced by FPC is slightly larger than the Arduino code.

Routines
`function` **`DigitalPinToBitmask`**`(const aPin: UInt8): UInt8;` Returns the bitmask corresponding to the specified pin, which maps the Arduino pin number to a bit on its respective port (B, C, or D). *Parameters* *aPin*: number of pin according to Arduino numeration. Valid range is 0 to 19. Note: as macro in Arduino sources.
`function` **`DigitalPinToPort`**`(const aPin: UInt8): UInt8;` Returns the port of specified pin according to Arduino numeration. *Parameters* *aPin*: number of pin according to Arduino numeration. Valid range is 0 to 19. Note: as macro in Arduino sources.
`function` **`DigitalPinToTimer`**`(const aPin: UInt8): UInt8;` Returns the timer of specified pin according to Arduino numeration. *Parameters* *aPin*: number of pin according to Arduino numeration. Valid range is 0 to 19. Note: as macro in Arduino sources.
`function` **`DigitalPinToPCICR`**`(const aPin: UInt8): PUInt8;` Returns the PCICR (Pin Change Interrupt Control Register) of specified pin according to Arduino numeration. *Parameters* *aPin*: number of pin according to Arduino numeration. Valid range is 0 to 19. Note: as macro in Arduino sources.

```
function DigitalPinToPCICRbit(const aPin: UInt8): UInt8;
```

Returns the bit index of PCICR (Pin Change Interrupt Control Register) of specified pin according to Arduino numeration.

Parameters
aPin: number of pin according to Arduino numeration. Valid range is 0 to 19.

Note: as macro in Arduino sources.

```
function DigitalPinToPCMSK(const aPin: UInt8): PUInt8;
```

Returns the PCMSK (Pin Change Mask Register) of specified pin according to Arduino numeration.

Parameters
aPin: number of pin according to Arduino numeration. Valid range is 0 to 19.

Note: as macro in Arduino sources.

```
function DigitalPinToPCMSKbit(const aPin: UInt8): UInt8;
```

Returns the bit index of PCMSK (Pin Change Mask Register) of specified pin according to Arduino numeration.

Parameters
aPin: number of pin according to Arduino numeration. Valid range is 0 to 19.

Note: as macro in Arduino sources.

```
procedure PinMode(const aPin, aMode: UInt8);
```

Configures the specified port I/O as an input or an output.

Parameters
aPin: number of pin according to Arduino numeration. Valid range is 0 to 19.
aMode: valid values are INPUT or OUTPUT.

```
procedure DigitalWrite(const aPin, aVal: UInt8);
```

Sets the specified digital pin to HIGH or LOW state.

Parameters
aPin: number of pin according to Arduino numeration. Valid range is 0 to 19.
aVal: valid values are HIGH or LOW.

```
function DigitalRead(const aPin: UInt8): integer;
```

Returns the state of specified digital pin. Valid values are HIGH or LOW.

Parameters
aPin: number of pin according to Arduino numeration. Valid range is 0 to 19.

Note: Change return type to UInt8.

`function` **`ShiftIn`**`(dataPin, clockPin, bitOrder: UInt8): UInt8;` Gets 8 bit of data and assembles them into 1 byte of data. It can be used as a software implementation of serial interface. The main application is reading the states of multiple switches or buttons using parallel-serial shift registers (e.g. 74HC165), which saves microcontroller pins. *Parameters* *dataPin*: data pin number according to Arduino numeration. Valid range is 0 to 19. *clockPin*: clock pin number according to Arduino numeration. Valid range is 0 to 19. *bitOrder*: Bit order of result byte. Valid values are LSBFIRST or MSBFIRST.
`procedure` **`ShiftOut`**`(dataPin, clockPin, bitOrder, val: UInt8);` Sends a byte of data bit by bit. It is used to synchronously send (transmit) a byte (8 bits) of data serially from a microcontroller to a peripheral device (e.g., a shift register, like the 74HC595). *Parameters* *dataPin*: data pin number according to Arduino numeration. Valid range is 0 to 19. *clockPin*: clock pin number according to Arduino numeration. Valid range is 0 to 19. *bitOrder*: Bit order of result byte. Valid values are LSBFIRST or MSBFIRST. *val*: byte to send.
`procedure` **`DigitalDebugStart;`** Implementation of simple debugging using internal LED, turning on the LED.
`procedure` **`DigitalDebugBreak;`** Implementation of simple debugging using internal LED, turning off the LED and then entering the infinite loop.

Example

This example is the simplest, most classic microcontroller demonstration—the "Hello World" of embedded programming. Its purpose is to make the built-in LED (Pin 13) on the Arduino Uno blink on and off continuously at a fixed interval of one second, using a blocking delay method.

At the very beginning, it sets Pin 13 (the built-in LED) as an output pin calling *PinMode*(*LedPin*, OUTPUT), and ensures the LED is off using *DigitalWrite*(*LedPin*, LOW). Next, the infinite loop (*while true do*) repeatedly executes two pairs of instructions:

1) Sets the pin voltage high, turning the LED on using *DigitalWrite*(*LedPin*, HIGH) and then pauses the entire program execution for 1000 milliseconds via *Delay*(1000).

2) Sets the pin voltage low, turning the LED off using *DigitalWrite*(*LedPin*, LOW) and then pauses the entire program execution for 1000 milliseconds via *Delay*(1000).

```
program TestBlink;

{$IF NOT (DEFINED(atmega328p) or DEFINED(arduinouno) or
DEFINED(arduinonano) or DEFINED(fpc_mcu_atmega328p) or
DEFINED(fpc_mcu_arduinouno) or DEFINED(fpc_mcu_arduinonano))}
 {$Fatal Invalid controller type, expected: atmega328p, arduinouno, or
arduinonano}
{$ENDIF}

{$mode objfpc}

uses
  defs, timer, digital;

const
  LedPin = 13; //internal LED

begin
  PinMode(LedPin, OUTPUT);
  DigitalWrite(ledPin, LOW);

  while True do
  begin
    DigitalWrite(ledPin, HIGH);
    Delay(1000);
    DigitalWrite(ledPin, LOW);
    Delay(1000);
  end;
end.
```