# IO pins microcontrollers

## AVR / ARduino – ATMEGA32U4 (ARduino Leonardo – Brainbox AVR)



DDR registers bepalen input of output (0 = input, 1 is output)

PORT registers bepalen de toestand van een output pin,

PIN registers geven de toestand weer van een input pin



# ARDUINO BRAINBOX AVR/ARDUINO DEMO PROGRAMS

# BBA: O-20

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

04/2016 - ATMEL STUDIO 7

This program demonstrates how ARDUINO IDE can be used to program the Brainbox Arduino - based on the ARduino Leonardo

For this program I have used real embedded C code - combined with Arduino library instructions to demonstrate that they can be used together.

This program blinks led 13 (blue led at pin C7) and led 17 (red led at RXled pin B0)

The pinout diagram of the BBA (www.E2CRE8.be) visualizes the relation between the ARduino pin names and the real pin names

This program can be used as inspiration to control any output pin of the Brainbox Arduino

These programs are used in school environments and the C code is kept low level for that reason

\*/

void setup**()** **{**

 // put your setup code here, to run once:

 DDRB **=** 0b00000001**;** // RB0 (pin17) = output pin - using real embedded C code

 pinMode**(**13**,** OUTPUT**);** // pin 13 (pinC7) = output pin - using ARduino library instruction & pin name

**}**

void loop**()** **{**

 // put your main code here, to run repeatedly:

 PORTB **=** 0b00000000**;** // turn led at pin B0 off

 digitalWrite**(**13**,** HIGH**);** // turn the LED 13 on (HIGH is the voltage level)

 \_delay\_ms**(**250**);** //wait for 250msec

 PORTB **=** 0b00000001**;** // turn led at pin B0 on

 digitalWrite**(**13**,** LOW**);** // turn the LED 13 on (HIGH is the voltage level)

 \_delay\_ms**(**250**);** //wait for 250msec

**}**

# BBA: O-BUZZER

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

04/2016 - ARDUINO IDE C

This program demonstrates how Arduino IDE can be used to program the Brainbox Arduino - based on the Arduino Leonardo

This program makes minimal use of the Arduino libraries. This brings you closer to real embedded programming.

The Pinout diagram of the BBA (www.e2cre8.be) visualizes the relation between the Arduino pin names and the real pin names

THis program generates a siren sound at pin E6 (pin 7) - this pin is connected to a buzzer on the BBA

These programs are used in school environments and the C code is kept low level for that reason

 \*/

void setup**()** **{**

 // put your setup code here, to run once:

DDRE **=** 0b01000000**;** // pin E6 (ARduino pin 7) connected to buzzer = output pin

**}**

void loop**()** **{**

 // put your main code here, to run repeatedly:

 **for** **(**int x **=** 250**;** x**>**0**;** x**=**x**-**1**)** // repeat the next 4 instructions 250 times (250 x 2msec = 0.5sec)

 **{**

 PORTE **=** 0b01000000**;** // make pin E6 high

 \_delay\_ms**(**1**);** //wait

 PORTE **=** 0b00000000**;** // make pin E6 low

 \_delay\_ms**(**1**);** //wait

 **}**

 **for** **(**int x **=** 500**;** x**>**0**;** x**=**x**-**1**)** // repeat the next 4 instructions 500 times (2500 x 1msec = 0.5sec)

 **{**

 PORTE **=** 0b01000000**;** // make pin E6 high

 \_delay\_us**(**500**);** //wait

 PORTE **=** 0b00000000**;** // make pin E6 low

 \_delay\_us**(**500**);** //wait

 **}**

**}**

# BBA: I-DIG

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

04/2016 - ARDUINO IDE C

This program demonstrates how Arduino IDE can be used to program the Brainbox Arduino - based on the Arduino Leonardo

This program makes minimal use of the Arduino libraries. This brings you closer to real embedded programming.

The Pinout diagram of the BBA (www.e2cre8.be) visualizes the relation between the Arduino pin names and the real pin names

This program 'reads' the level of the input pin F0 (Arduino pin 23). When this pin is high, the siren will sound with a 500Hz signal -

when it is low - the siren will sound with a 1KHz signal

The siren is a buzzer that is connected to pin E6

These programs are used in school environments and the C code is kept low level for that reason

 \*/

#define LEDS DDRF

void setup**()** **{**

 // put your setup code here, to run once:

 LEDS **=** 0b00000000**;** // pin F0 (ARduino pin 23) is digital input (all pins of portF are digital inputs...)

 DDRE **=** 0b01000000**;** // pin E6 (ARduino pin 7) connected to buzzer = output pin

**}**

void loop**()** **{**

 // put your main code here, to run repeatedly:

 **if** **(**PINF **&** 0b00000001**)** // when the input of F0 is high (we used a mask to only look at pin 0 of port F)

 **{**

 PORTE **=** 0b01000000**;** // make pin E6 high

 \_delay\_ms**(**1**);** //wait

 PORTE **=** 0b00000000**;** // make pin E6 low

 \_delay\_ms**(**1**);** //wait

 **}**

 **else** // when the input of F0 is low

 **{**

 PORTE **=** 0b01000000**;** // make pin E6 high

 \_delay\_us**(**500**);** //wait

 PORTE **=** 0b00000000**;** // make pin E6 low

 \_delay\_us**(**500**);** //wait

 **}**

**}**

# AD conversie

## ATMEGA32U4 (Brainbox AVR / Arduino)



# BBA: IAN

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

04/2016 - ARDUINO IDE C

This program demonstrates how the Analog inputs channels can be programmed in "C"

Use the datasheet of the ATMEGA32U4 from page 298 for more info

We have written 2 functions

READ\_ADC\_BYTE\_CHANNEL: converts the Analog input into a 8 bit digital value - tested for channel 0-7

READ\_ADC\_INT\_CHANNEL: converts the Analog input into a 10 bit digital value - tested for channel 0-7

Connect a potmeter to ADC7 (AN0 ARduino)

to demonstrate the result of the AD conversion is used to change the frequency at the buzzer - connected to E6 on the BBA

\*/

char ADRESULT **=** 0**;**

void setup**()** **{**

 // put your setup code here, to run once:

 unsigned int READ\_ADC\_INT\_CHANNEL**(** unsigned char channel **);** // function prototype - see below

 unsigned char READ\_ADC\_BYTE\_CHANNEL**(** unsigned char channel **);** // function prototype - see below

 DDRF **=** 0b00000000**;** // pin F0 (ARduino pin 23) is digital input (all pins of portF are digital inputs...)

 DDRE **=** 0b01000000**;** // pin E6 (ARduino pin 7) connected to buzzer = output pin

**}**

void loop**()** **{**

 // put your main code here, to run repeatedly:

 ADRESULT **=** READ\_ADC\_BYTE\_CHANNEL**(**7**);** // convert the analog signal of channel 7 (ARduino A0) into a 8 bit digital signal and store it into "ADRESULT"

 PORTE **=** 0b01000000**;** // make pin E6 high

 **for** **(**char x **=** ADRESULT**;** x**>**0**;** x**--)** // make pin E6 high for as many usec as ADRESULT

 **{**

 \_delay\_us**(**1**);**

 **}**

 //wait

 PORTE **=** 0b00000000**;** // make pin E6 low

 **for** **(**char x **=** ADRESULT**;** x**>**0**;** x**--)** // // make pin E6 hlow for as many usec as ADRESULT

 **{**

 \_delay\_us**(**1**);**

 **}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ADC FUNCTIONS - ADC FUNCTIONS - ADC FUNCTIONS - ADC FUNCTIONS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// returns a 10 bit result from an AD conversion from a certain ADC channel

//Channels 0-7

// A 10 bit result needs to be loaded in an integer variable

unsigned int READ\_ADC\_INT\_CHANNEL**(** unsigned char channel **){**

 DDRF **&=** **~(**1**<<**channel**);** // config selected channel as input

 // Right adjust + choice of channel

 ADMUX **=** 0b01000000 **+** channel**;**

 // Activate ADC - Stop conversion - prescaler 128 (for ADC 50K<..<200k) (16Mhz / 128 = 125000)

 ADCSRA **=** 0b10000111**;**

 // start conversion and wait for completion

 ADCSRA **|=** **(**1**<<**ADSC**);**

 **while** **(**ADCSRA **&** **(**1**<<**ADSC**));**

 // return A/D conversion result

 unsigned char ADCLBuff **=** ADCL**;** // first read ADCL!!

 **return** **(** **((**unsigned int**)** ADCH **)** **<<** 8 **)** **+** ADCLBuff**;**

**}**

// returns a 8 bit result from an AD conversion from a certain ADC channel

//Channels 0-7

// A 8 bit result needs to be loaded in a BYTE variable

unsigned char READ\_ADC\_BYTE\_CHANNEL**(** unsigned char channel **){**

 DDRF **&=** **~(**1**<<**channel**);** // config selected channel as input

 // Left adjust + choice of channel

 ADMUX **=** 0b01100000 **+** channel **;**

 // Activate ADC - Stop conversion - prescaler 128 (for ADC 50K<..<200k) (16Mhz / 128 = 125000)

 ADCSRA **=** 0b10000111**;**

 // start conversion and wait for completion

 ADCSRA **|=(**1**<<**ADSC**);**

 **while** **(**ADCSRA **&** **(**1**<<**ADSC**));**

 // return A/D conversion result

 **return** ADCH**;**

**}**

# BBA: PWM

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

04/2016 - ARDUINO IDE C

This program demonstrates how Arduino IDE can be used to program the Brainbox Arduino - based on the Arduino Leonardo

This program makes minimal use of the Arduino libraries. This brings you closer to real embedded programming.

The Pinout diagram of the BBA (www.e2cre8.be) visualizes the relation between the Arduino pin names and the real pin names

This program generates PWM signals at 6 different output pins - it is necessary to look at the datasheet for further explanation

 // OC1A = PWM at B5

 // OC1B = PWM at B6

 // OC1C = PWM at B7

 // OC3A = PWM at C6

 // OC4A = PWM at C7 - BLUE LED

 // OC4D = PWM at D7

The PWM frequency is +/- 60Hz, but this can be changed by altering the prescaler bits (see datasheet)

 The PWM duty cycle (0-255) is set by the position of a potmeter connected to to ADC7 (AN0 ARduino)

\*/

char ADRESULT **=** 0**;**

void setup**()** **{**

 // put your setup code here, to run once:

unsigned char READ\_ADC\_BYTE\_CHANNEL**(** unsigned char channel **);** // function prototype

 DDRC **=** 0b11000000**;** // pinC7 & C6 output

 DDRB **=** 0b11100000**;** // pinB7, B6, B5 output

 DDRD **=** 0b10000000**;** // pinC7 = blue led = output

 //Timer 1 PWM configuration - datasheet page 130

 // Set for Fast PWM \_ 8 BIT - CLK/1024

 // OC1A = PWM at B5

 // OC1B = PWM at B6

 // OC1C = PWM at B7

 TCCR1A **=** 0b10101001**;** // COM1A1 COM1A0 COM1B1 COM1B0 COM1C1 COM1C0 WGM11 WGM10

 TCCR1B **=** 0b00001101**;** // ICNC1 ICES1 WGM13 WGM12 CS12 CS11 CS10

 //Timer 3 PWM configuration - datasheet page 130

 // Set for Fast PWM \_ 8 BIT - CLK/1024

 // OC3A = PWM at C6

 TCCR3A **=** 0b10000001**;** // COM1A1 COM1A0 COM1B1 COM1B0 COM1C1 COM1C0 WGM11 WGM10

 TCCR3B **=** 0b00001101**;** // ICNC1 ICES1 WGM13 WGM12 CS12 CS11 CS10

 // Timer 4 PWM configuration - datasheet page 160

 // Set for Fast PWM - CLK/1024

 // OC4A = PWM at pin C7 - BLUE LED

 // OC4B = PWM at pin B6 - not used - double with OC1B

 // OC4D = PWM at pin D7

 TCCR4A **=** 0b10100011**;** // COM4A1 COM4A0 COM4B1 COM4B0 FOC4A FOC4B PWM4A PWM4B

 TCCR4B **=** 0b00001011**;** // PWM4X PSR4 DTPS41 DTPS40 CS43 CS42 CS41 CS40

 TCCR4C **=** 0b10101001**;** // COM4A1S COM4A0S COM4B1S COMAB0S COM4D1 COM4D0 FOC4D PWM4D (! be aware that 7:4 are shadow registers of TCCR4A and need to be programmed the same)

 TCCR4D **=** 0b00000000**;** // FPIE4 FPEN4 FPNC4 FPES4 FPAC4 FPF4 WGM41 WGM40 (WGM = 00 = fast PWM)

**}**

void loop**()** **{**

 // put your main code here, to run repeatedly:

 ADRESULT **=** READ\_ADC\_BYTE\_CHANNEL**(**7**);** // convert the analog signal of channel 7 (ARduino A0) into a 8 bit digital signal and store it into "ADRESULT"

 OCR1AL **=** ADRESULT**;** //PWM value at pin B5 (OC1A)

 OCR1BL **=** ADRESULT**;** //PWM value at pin B6 (OC1B)

 OCR1CL **=** ADRESULT**;** //PWM value at pin B7 (OC1C)

 OCR3AL **=** ADRESULT**;** //PWM value at pin C6 (OC3A)

 OCR4A **=** ADRESULT**;** //PWM value at pin C7 (OC4A) - BLUE LED

 OCR4D **=** ADRESULT**;** //PWM value at pin D7 (OC4D)

**}**

// AD FUNCTION

// returns a 8 bit result from an AD conversion from a certain ADC channel

//Channels 0-17

// A 8 bit result needs to be loaded in a BYTE variable

unsigned char READ\_ADC\_BYTE\_CHANNEL**(** unsigned char channel **){**

 DDRF **&=** **~(**1**<<**channel**);** // config selected channel as input

 // Left adjust + choice of channel

 ADMUX **=** 0b01100000 **+** channel **;**

 // Activate ADC - Stop conversion - prescaler 128 (for ADC 50K<..<200k) (16Mhz / 128 = 125000)

 ADCSRA **=** 0b10000111**;**

 // start conversion and wait for completion

 ADCSRA **|=(**1**<<**ADSC**);**

 **while** **(**ADCSRA **&** **(**1**<<**ADSC**));**

 // return A/D conversion result

 **return** ADCH**;**

**}**

## Arduino Interrupt vector benamingen

1 Reset

 2 External Interrupt Request 0 (pin D2) (INT0\_vect)

 3 External Interrupt Request 1 (pin D3) (INT1\_vect)

 4 Pin Change Interrupt Request 0 (pins D8 to D13) (PCINT0\_vect)

 5 Pin Change Interrupt Request 1 (pins A0 to A5) (PCINT1\_vect)

 6 Pin Change Interrupt Request 2 (pins D0 to D7) (PCINT2\_vect)

 7 Watchdog Time-out Interrupt (WDT\_vect)

 8 Timer/Counter2 Compare Match A (TIMER2\_COMPA\_vect)

 9 Timer/Counter2 Compare Match B (TIMER2\_COMPB\_vect)

10 Timer/Counter2 Overflow (TIMER2\_OVF\_vect)

11 Timer/Counter1 Capture Event (TIMER1\_CAPT\_vect)

12 Timer/Counter1 Compare Match A (TIMER1\_COMPA\_vect)

13 Timer/Counter1 Compare Match B (TIMER1\_COMPB\_vect)

14 Timer/Counter1 Overflow (TIMER1\_OVF\_vect)

15 Timer/Counter0 Compare Match A (TIMER0\_COMPA\_vect)

16 Timer/Counter0 Compare Match B (TIMER0\_COMPB\_vect)

17 Timer/Counter0 Overflow (TIMER0\_OVF\_vect)

18 SPI Serial Transfer Complete (SPI\_STC\_vect)

19 USART Rx Complete (USART\_RX\_vect)

20 USART, Data Register Empty (USART\_UDRE\_vect)

21 USART, Tx Complete (USART\_TX\_vect)

22 ADC Conversion Complete (ADC\_vect)

23 EEPROM Ready (EE\_READY\_vect)

24 Analog Comparator (ANALOG\_COMP\_vect)

25 2-wire Serial Interface (I2C) (TWI\_vect)

26 Store Program Memory Ready (SPM\_READY\_vect)

# BBA External interrupt INT0

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

11/01/2017

Demo of external interrupt at INT0 pin PD0 (pin 3 on ARduino)

\*/

// deze includes en defines zijn niet nodig in IDE

# include <avr/io.h>

# include <avr/interrupt.h>

# define F\_CPU 16000000 // Xtal op 16MHz

volatile byte state **=** LOW**;**

void setup**()**

**{**

 DDRC **=** DDRC **|** 0b10000000**;** //pin RC7 (Pin13) output

 EICRA **=** **(**1 **<<** ISC00 **)** **|** **(**1 **<<** ISC01 **);** // 1 1 The rising edge of INT0 generates an interrupt request

 EIMSK **=** **(**1 **<<** INT0**);** // External interrupt request mask enable

 sei**();** //Enable global interrupts

**}**

void loop**()**

**{**

 PORTC **=** **(**state **<<** 7**);** // state determines the state of the led at RC7 - pin 13

**}**

ISR **(**INT0\_vect**)** // interrupt routine that executes every time a rising edge occurs on INT0 pin

**{**

 state **=** **!**state**;** // inverse the variable "state"

**}**

# TIMER0 interrupt

!! Deze code zou moeten werken, maar werkt niet omdat Arduino IDE Timer0 gebruikt voor de delay functies.

TIMER1 is een 16 bit teller

TIMER 4 is een 10 bit asynchrone teller

/\*

www.E2CRE8.be - Brainbox Arduino - by Bart Huyskens

11/01/2017

Demo of timer int every 1sec - using TMR0 - no prescaler

incrementing TMR0 at 16Mhz and

generating 62500 overflow interrupts/sec: 16000000/256 = 62500/sec

\*/

// deze includes en defines zijn niet nodig in IDE

# include <avr/io.h>

# include <avr/interrupt.h>

# define F\_CPU 16000000 // Xtal op 16MHz

volatile byte state **=** LOW**;**

volatile unsigned long int secteller **=** 0**;**

void setup**()**

**{**

 DDRC **=** DDRC **|** 0b10000000**;** //pin RC7 (Pin13) output

 TCNT0 **=** 0x00**;** // clear TMR0

 TCCR0B **|=** **(**1 **<<** CS00 **);** // 001 - no prescaling - TMRO runs at 16Mhz

 TIMSK0 **|=** **(**1 **<<** TOIE0**);** // enable Timer0 interrupt at TMR0 overflow

 sei**();** //Enable global interrupts

**}**

void loop**()**

**{**

 PORTC **=** **(**state **<<** 7**);** // state determines the state of the led at RC7 - pin 13

**}**

ISR**(**TIMER0\_OVF\_vect**)** // interrupt routine that executes every time TMR0 overflows 255 >> 0

**{**

 **if** **(**secteller **>=** 6250**)** // this is true - every second

 **{** // after 62500 ints - 1 sec is passed

 state **=** **!**state**;** // flip pinC7

 secteller **=** 0**;** // reset secteller

 **}**

 **else**

 **{**

 secteller **++;** // increment secteller

 **}**

**}**