

ASSEMBLY & C

WEEK 3-1

AGENDA

week	onderwerp	week	week
1	de structuur van AVR-assembly AVR instructies AVR registers en I/O ATmega memory map Atmel Studio AVR expressies en directives AVR addressing modes	3	de structuur van C-programma's ATMEL studio en AVR libc typen, constanten en operatoren AVR register access in C control statements functies & stackframe visibility scope arrays & strings struct & enum
2	flow of control spring instructies, control structuren Arduino UNO AVR studio stack & subroutines interrupts timer/counters switch bounce	4	interrupts in C TM1638 led&key UART PWM & ADC using a TTC-scheduler state diagram

AGENDA

- **timer/counter**
- switch bounce
- intro C
- de structuur van een C-programma
- ATMEL studio
- AVR libc
- register access
- typen en constanten

WAT IS EEN (UP/DOWN) COUNTER ?



Figure 27. 8-bit Timer/Counter Block Diagram

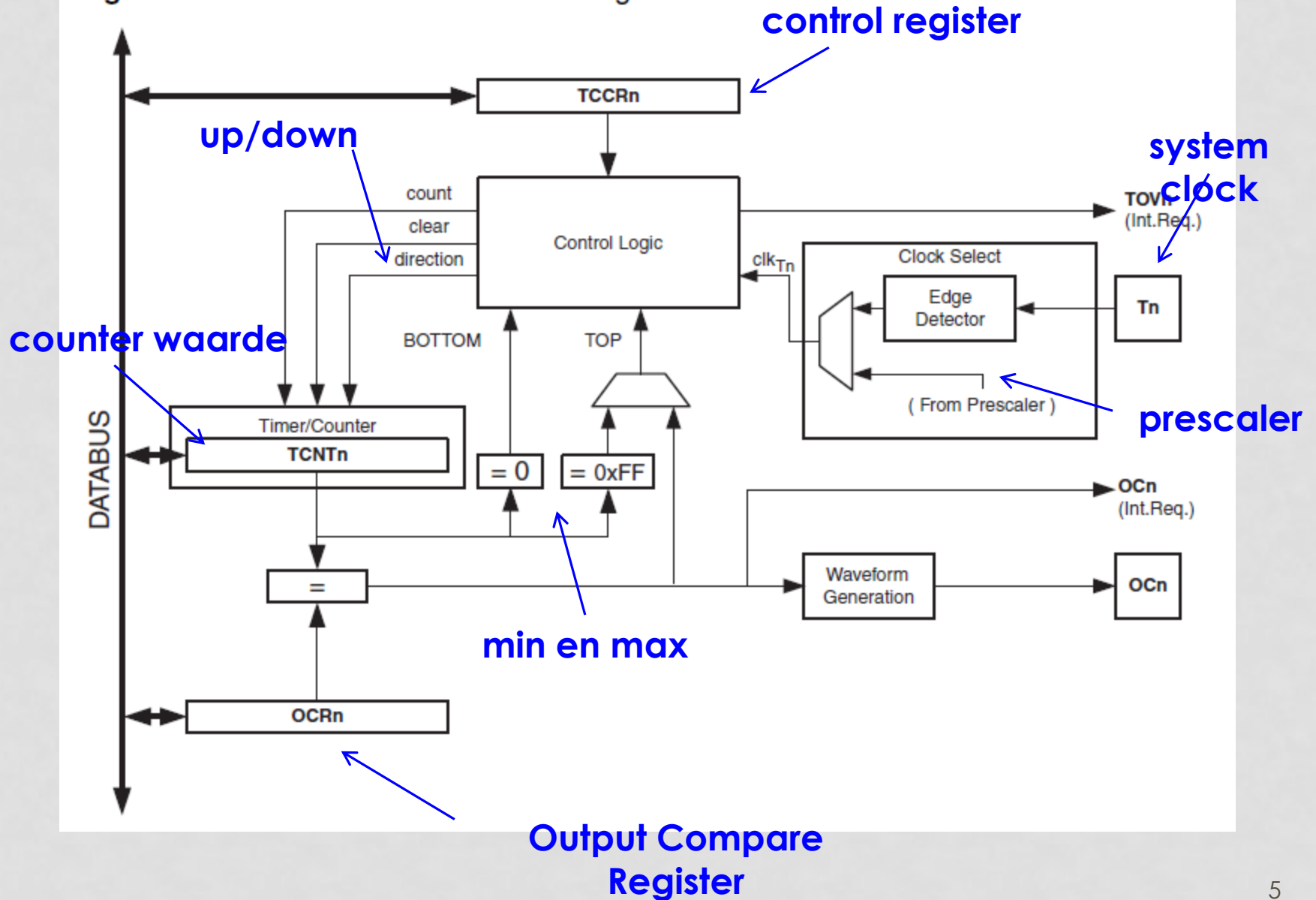
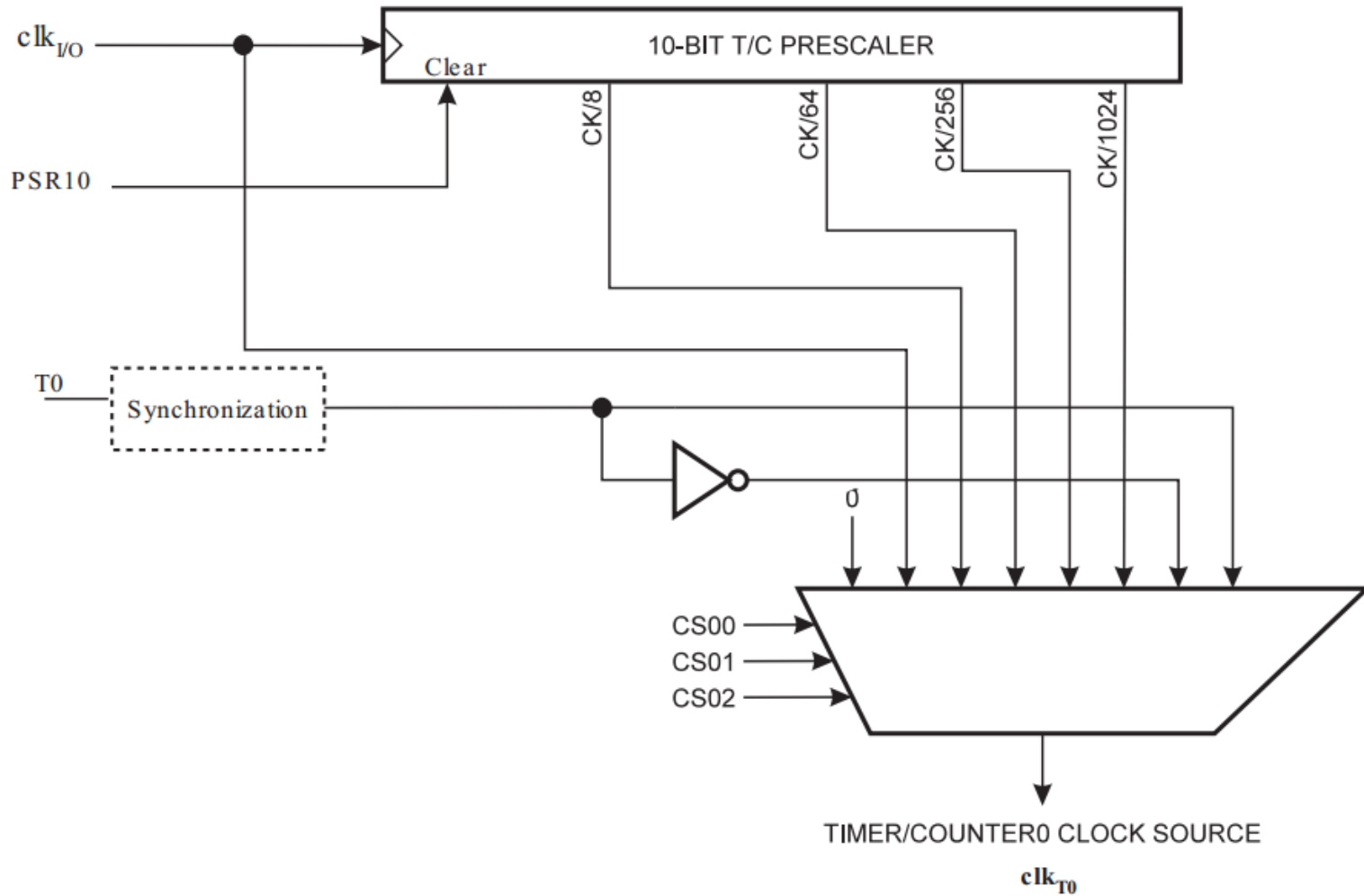


Figure 3-1. Prescaler for Timer0/1



TIMERS/COUNTERS

- er zijn 3 timer/counters on-chip (328P)
 - timer 0 & timer 2 : 8 bit
 - timer 1 : 16-bit
- bij elke clock-cycle wordt teller "automatisch" verhoogd of verlaagd
- de systeem klok kan worden gedeeld : **prescaling** met 8, 64, 256 of 1024
- De timers hebben 2 onafhankelijke Output Compare Units (A & B)

TIMER EVENTS

- interrupts kunnen worden ge-triggerd door timer events :
 - **overflow** ('over de kop'), bijvoorbeeld van \$FF naar \$00
 - **compare match** : counter waarde = output compare register

TWEE MODES

- Normal Mode
 - counter telt op
 - overflow als waarde van \$ff naar \$00
- Clear Timer on Compare Mode (CTC Mode)
 - counter telt totdat waarde in Output Compare Register bereikt wordt ("compare match")
 - daarna wordt teller op 0 gezet

ENABLE TIMER INTERRUPTS

- enable gewenste interrupts in the Timer Interrupt Mask Register (TIMSK)
- elke event heeft een eigen interrupt enable bit in TIMSK
 - TOIE_n and OCIE_n ($n = 0, 1, 2$)
- interrupt vectoren : TIMER_n_OVF_vect en TIMER_n_COMP_vect

OPDRACHT

- stel :
 - $f_{\text{cpu}} = 100 \text{ Hz}$
 - prescale = 2
 - elke 2 seconde een interrupt
- CTC mode : wat is de waarde van OCR ?

$f_{\text{klok}} = 100/2 = 50 \text{ Hz}$ dus periode = 0.02 sec
dus tot 100 tellen om 2 seconde te bereiken

```

; verbind LEDs met port B !!!
.include "m328Pdef.inc"
.def saveSR = r17
.def temp = r18
.org 0x0000
rjmp init

.org OC1Aaddr
rjmp TIMER1_COMP_ISR    ; adres ISR (Timer1 Output Compare Match)

init:
; init stack pointer
ldi R16, high(RAMEND)
out SPH, R16
ldi R16, low(RAMEND)
out SPL, R16

; init Output Compare Register
; 1 sec = (256/16.000.000) * 62500
; write high byte before the low byte !
ldi temp, high(62500)
sts OCR1AH, temp
ldi temp, low(62500)
sts OCR1AL, temp

```

```

; zet prescaler op 256 & zet timer in CTC-mode
ldi temp, (1 << CS12) | (1 << WGM12)
sts TCCR1B, temp
; enable interrupt
ldi temp, (1 << OCIE1A)
sts TIMSK1, temp
; init port
ser temp ; tmp = $FF
out DDRB, temp ; Port B is output port (via LEDs)
out PORTB, temp ; LEDs uitzetten

sei ; enable alle interrupts

; wacht in lus op interrupt
loop:
    rjmp loop

TIMER1_COMP_ISR:
    ; ISR wordt elke seconde aangeroepen
    in saveSR, SREG ; save SREG
    in r16, PORTB ; schrijf de inhoud van PORTB naar r16
    com r16 ; inverteer alle bits $00 <-> $FF
    out PORTB, r16 ; schrijf waarde r16 naar PORTB
    out SREG, saveSR ; restore SREG
    reti ; return from interrupt

```

SECONDE TELLER IN C

output compare register

```
void init_timer (void)
{
    OCR1A = (uint16_t) 62500;
    // 1 sec = (256/16.000.000) * 62500
    TCCR1B = (1 << CS12) | (1 << WGM12);
    // prescale op 256 & CTC-mode; top = OCR1A
    TIMSK |= 1 << OCIE1A;
    // Timer1 Output Compare Match Interrupt Enable
}

//
// iedere seconde wordt deze functie uitgevoerd
//
ISR (TIMER1_COMPA_vect)
{
}
```

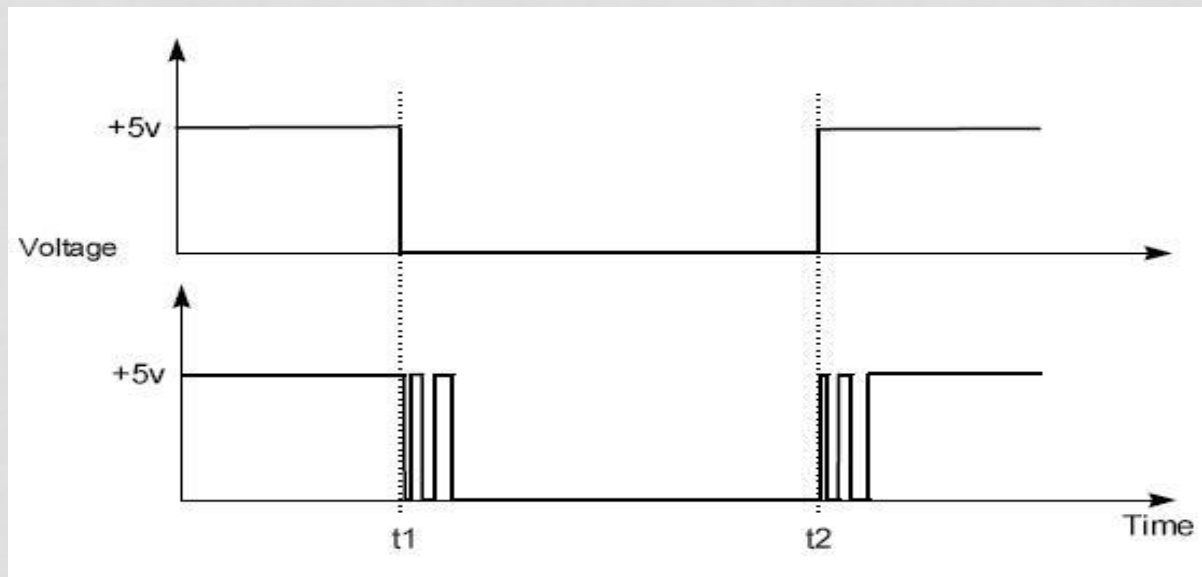
kristal frequentie

AGENDA

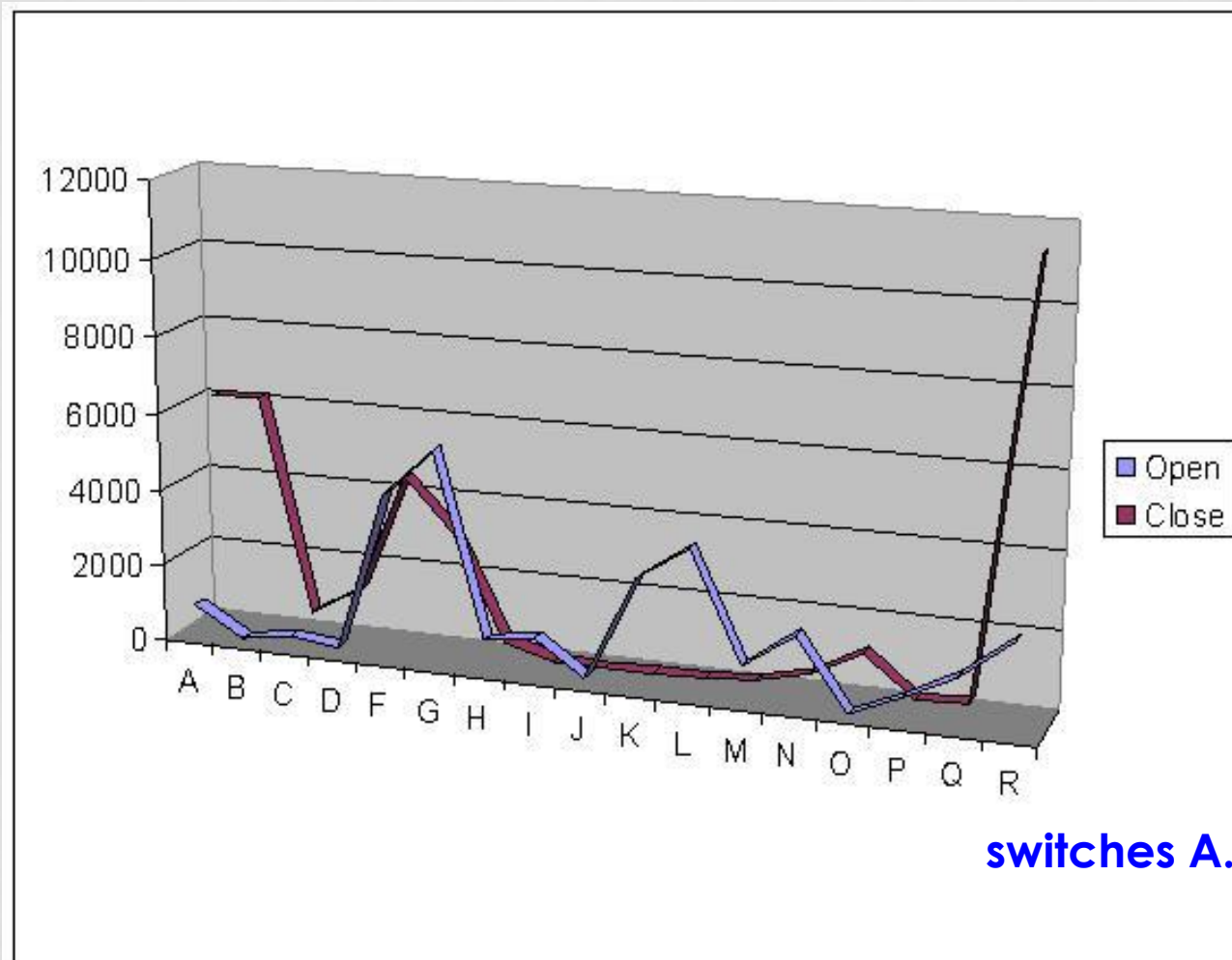
- timer/counter
- **switch bounce**
- intro C
- de structuur van een C-programma
- ATMEL studio
- AVR libc
- register access
- typen en constanten

SWITCH BOUNCE (DENDER)

- indrukken of loslaten van een schakelaar kan veel 0-1 overgangen genereren
- bouncing is meestal over binnen 20 ms



μs



switches A..R

*Bounce times in microseconds, for opening and closing each switch (number A to R).
Switch E was left out, as its 157 msec bounces would horribly skew the graph.*

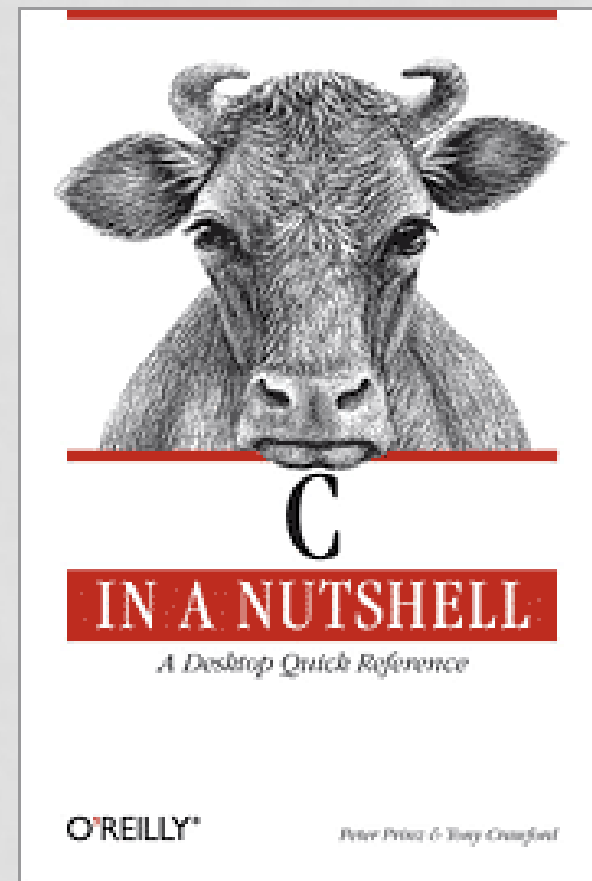
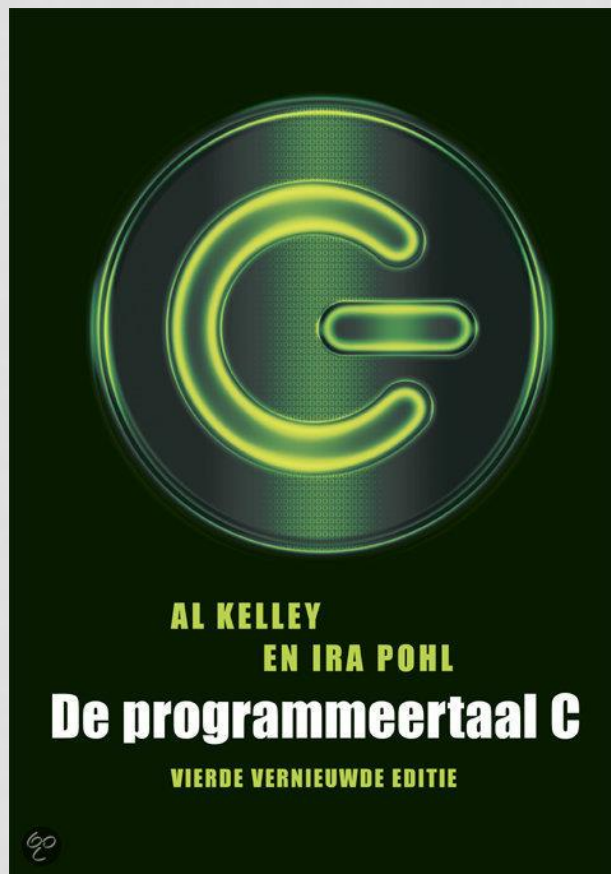
SWITCH BOUNCE (DENDER)

- HW oplossing: gebruik een debouncer (IC)
- SW oplossing 1 : wait-and-see
 - wacht 20 ms (delay) en kijk dan opnieuw of waarde hetzelfde is
- SW oplossing 2 : tellen
 - kijk elke 2 ms (polling); heb je 5x zelfde waarde gezien, dan weet je dat schakelaar 10ms ingedrukt is
- gebruik hiervoor **timer interrupt** om de schakelaar uit te lezen
 - vanuit ISR toestand van de schakelaar opslaan in globale variabele die in "main loop" wordt uitgelezen

AGENDA

- timer/counter
- switch bounce
- **intro C**
- de structuur van een C-programma
- ATMEL studio
- AVR libc
- register access
- typen en constanten

BOEKEN



INTRO C



- ontwikkeld in 1969 ... 1973 door Dennis Ritchie bij Bell Labs
 - ontwikkeling van UNIX
- Kernighan + Ritchie "The C Programming Language" in 1987 (The K&R)
- invloed op : awk, csh, C++, C#, Java, JavaScript, Perl, PHP, python, ..
- ANSI C : standaard in 1989
 - American National Standards Institute
 - werd bekend als C89, later C99

INTRO C

- huidige standaard is ISO C11
 - new interfaces for threads and atomic operations
- GNU C = standaard C + extensies (-std=c11 optie)
- compilers voor elk platform (OS en/of CPU)
- C++ is extensie van C
- C is niet een strikte subset van C++
 - maar het is mogelijk om C te schrijven die voldoet aan zowel C als de C++ standaard

WAAROM C ?

- efficiënt : embedded systeem beperkt geheugen en beperkte performance (ander voorbeeld : gaming)
- 'low level' toegang tot h/w
- porteren : compiler moet veel CPU's ondersteunen
 - core is goed te porteren (i/o etc. zit in standard library)
- beschikbaarheid programmeurs : populaire taal

AVR-GCC

Architectures [\[edit \]](#)

GCC target processor families as of version 4.3 include:

- Alpha
- ARM
- AVR
- Blackfin
- Epiphany (GCC 4.8)
- H8/300
- HC12
- IA-32 (x86)
- IA-64 (Intel Itanium)
- MIPS
- Motorola 68000
- PA-RISC
- PDP-11
- PowerPC
- R8C / M16C / M32C
- SPARC
- SPU
- SuperH
- System/390 / zSeries
- VAX
- x86-64

- C standard library is platform specific
- avrlibc supports avr architecture

AGENDA

- timer/counter
- switch bounce
- intro C
- **de structuur van een C-programma**
- ATMEL studio
- AVR libc
- register access
- typen en constanten

```

// Arduino UNO test programma
// verbind PORT B5 (pin 13) met een LED

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

#define BLINK_DELAY_MS 1000

int main (void)
{
    // set pin 5 of PORTB for output
    DDRB |= _BV(DDB5);

    while(1) {
        // set pin 5 high to turn led on
        PORTB |= _BV(PORTB5);
        _delay_ms(BLINK_DELAY_MS);

        // set pin 5 low to turn led off */
        PORTB &= ~_BV(PORTB5);
        _delay_ms(BLINK_DELAY_MS);
    }
}

```

← include files
 # : preprocessor directives

← main() wordt altijd
 eerst aangeroepen

```

// Arduino UNO test programma
// verbind PORT B0-B5 (pin 8-13) met de LEDs

#include <avr/io.h>           ← include files
#include <avr/interrupt.h>    ← # : preprocessor directives

volatile int gv_b = 0x00;    ← globale variabele

// initialiseren van de timer
// iedere seconde wordt er een interrupt aangeroepen
void init_timer(void)
{
    OCR1A = (uint16_t)62500;    // 1 sec = (256/16.000.000) * 62500
    TCCR1B = (1 << CS12) | (1 << WGM12); // prescale op 256,
                                        // top counter = value OCR1A (CTC mode)
    TIMSK1 = 1 << OCIE1A;      // Timer 1 Output Compare A Match Interrupt Enable
}

void init_port(void)          ← functie
                               definitie
{
    DDRB = 0xff;              // set port B as output
    PORTB = gv_b;             ← register een
                               waarde geven
}

```

```
// interrupt service routine
```

```
ISR(TIMER1_COMPA_vect)
```

```
{
```

```
    // LED's inverteren
```

```
    gv_b = ~gv_b;
```

```
    PORTB = gv_b;
```

```
} ← { ... } block
```

```
int main(void) ← main() wordt altijd  
                eerst aangeroepen
```

```
{
```

```
    init_port(); ← een functie
```

```
    init_timer();    aanroepen
```

```
    sei();
```

```
    while(1){}; // loop
```

```
    return 0;
```

```
}
```

SOURCE FILES

- structuur :
 - pre-processor directives
 - globale declaraties
 - functie definities
- compileren
 - Atmel studio : "Build Solution F7"
 - command line : gcc my_prog.c
- C source file .c + header files .h = translation unit
 - compiler genereert object myprog.o
 - linker genereert executable my_prog.hex

DE C PREPROCESSOR

- importeren files

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16E6
#include <util/delay.h>
#include "distance.h"
```

- <> library header file
 - " " eigen header file
- symbolic constant : `#define PI 3.14`
 - betekent : vervang identifier "PI" door string "3.14"

CONST

- gebruik Const i.p.v. pre-processor symbolic

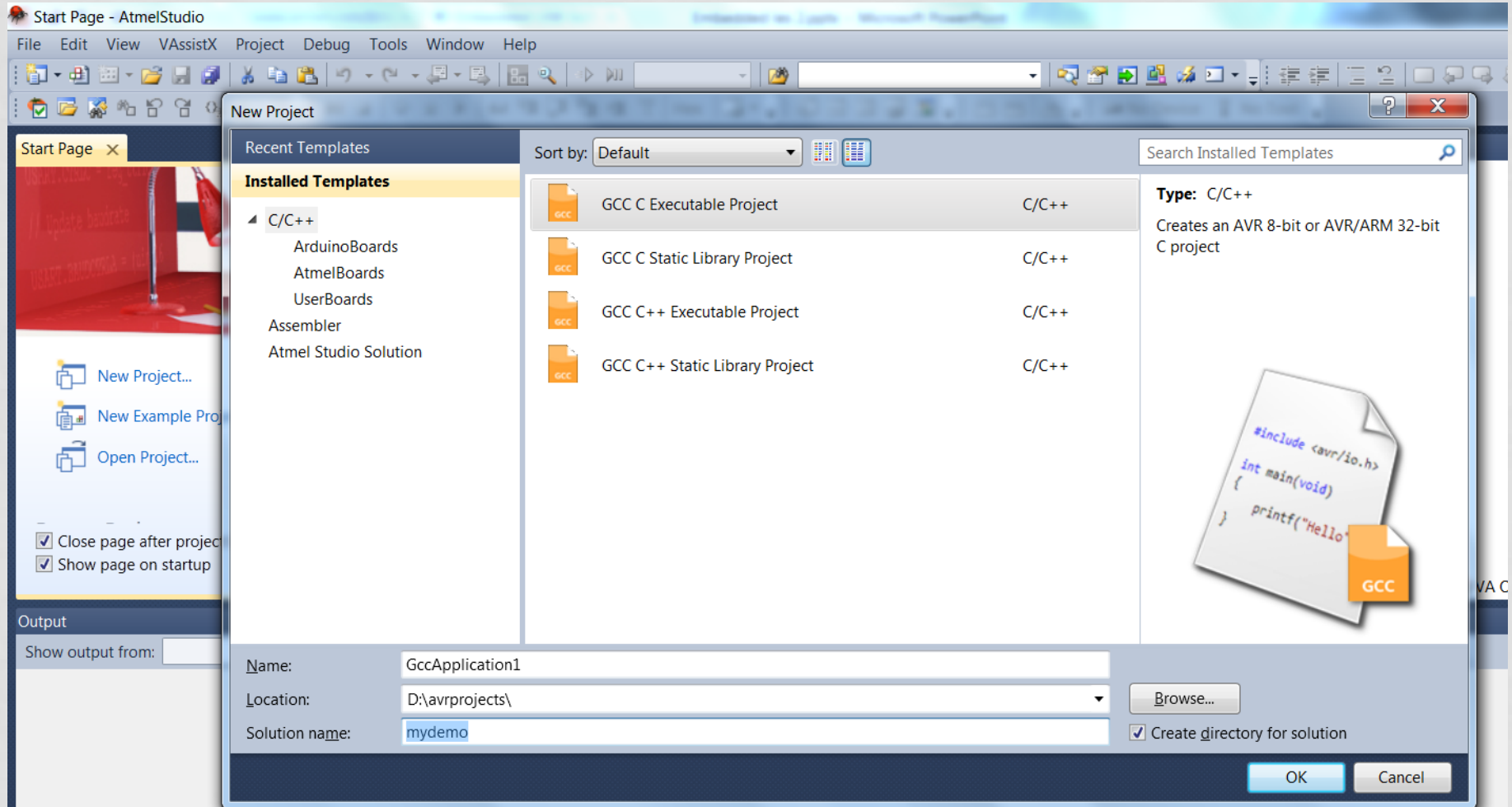
```
const int DELAY_VALUE = 1000;
```

- met #define :
 - niet zichtbaar in debugger, niet in symbol table
 - heeft geen scope
 - heeft geen type

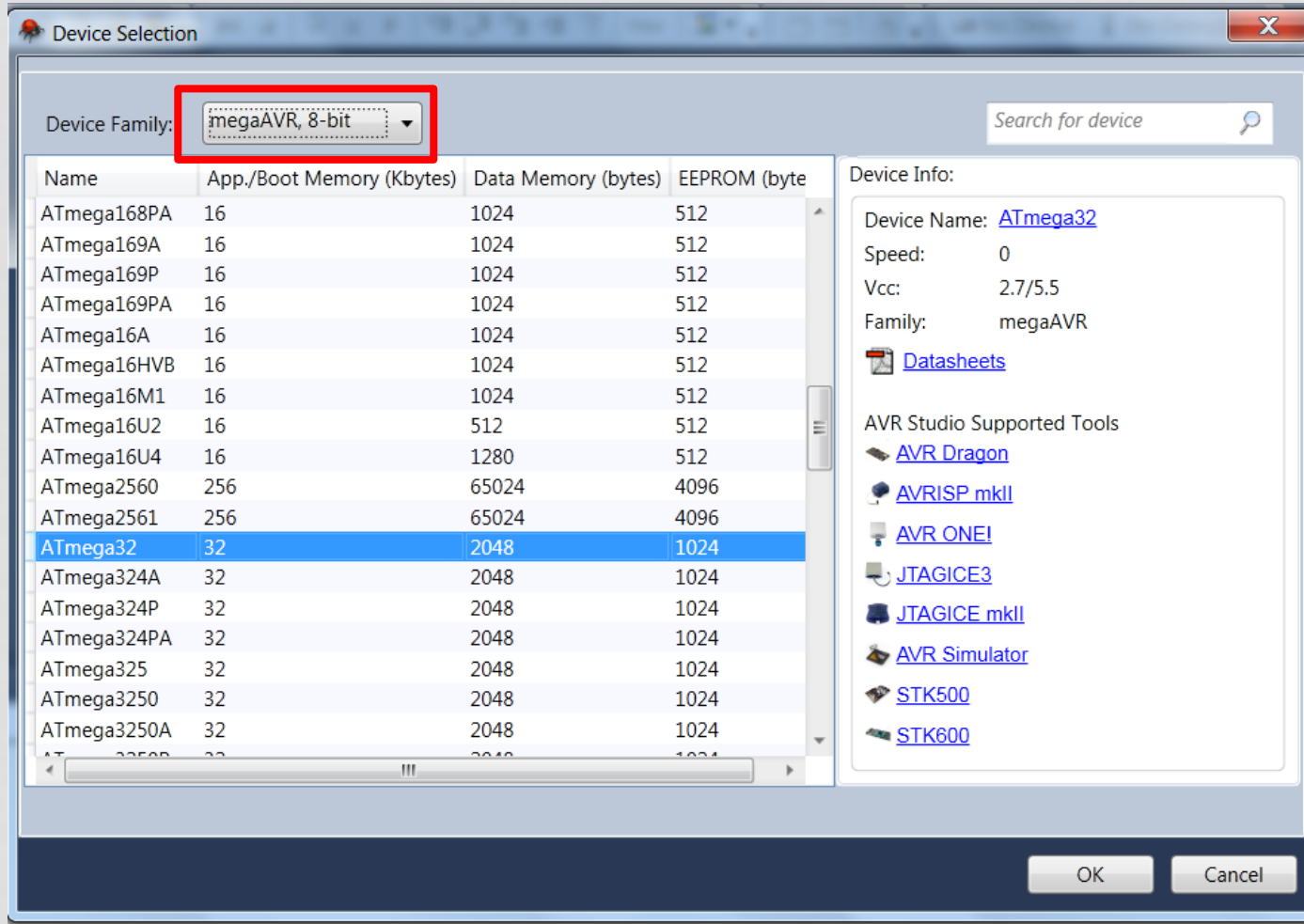
AGENDA

- timer/counter
- switch bounce
- intro C
- de structuur van een C-programma
- **ATMEL studio**
- AVR libc
- register access
- typen en constanten

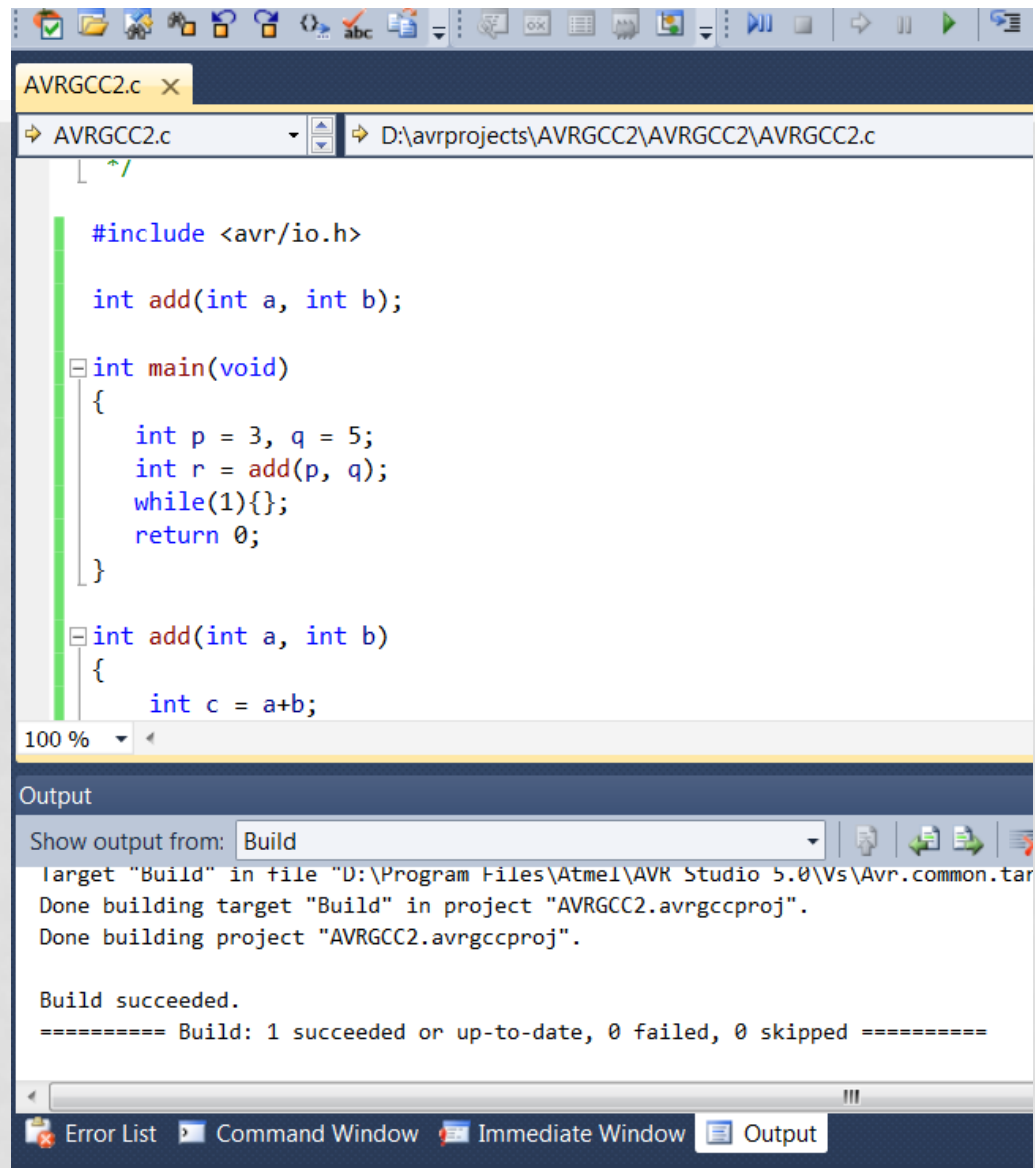
CREATE A PROJECT



CREATE A PROJECT



BUILD A PROJECT



The screenshot displays the AVR Studio 5.0 IDE interface. The main window shows a C source file named `AVRGCC2.c` located at `D:\avrprojects\AVRGCC2\AVRGCC2\AVRGCC2.c`. The code defines a function `add` and a `main` function that calls it. The `main` function initializes `p=3` and `q=5`, calls `add(p, q)` to get `r`, enters an infinite loop `while(1){};`, and returns 0. The `add` function simply returns the sum of its arguments.

Below the code editor is the 'Output' window, which shows the results of a build. The 'Show output from:' dropdown is set to 'Build'. The output text indicates that the target 'Build' was successfully built in the project 'AVRGCC2.avrgccproj', and the overall build succeeded.

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

int add(int a, int b);

int main(void)
{
    int p = 3, q = 5;
    int r = add(p, q);
    while(1){};
    return 0;
}

int add(int a, int b)
{
    int c = a+b;
}
```

Output

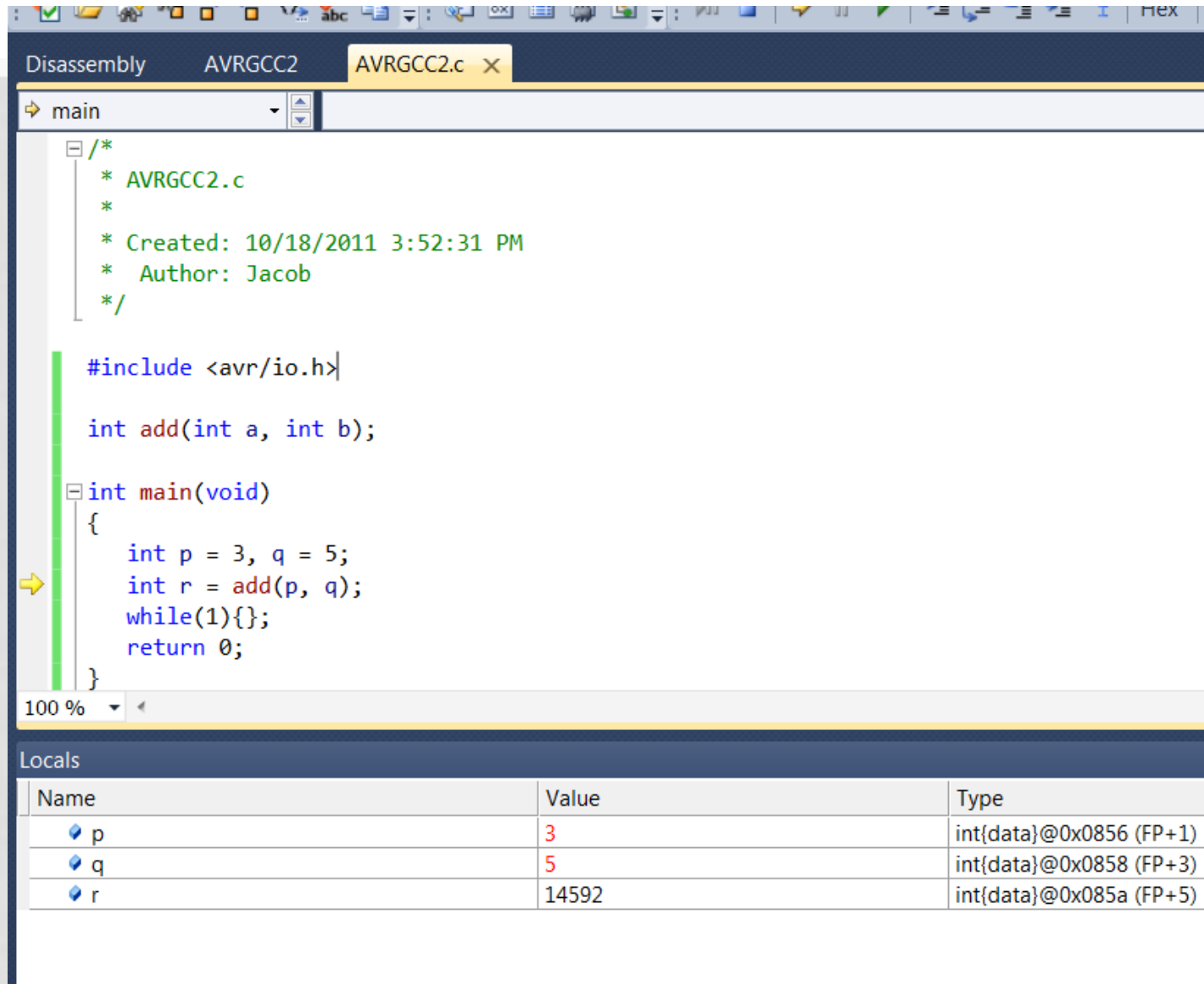
Show output from: Build

target "Build" in file "D:\Program Files\Atmel\AVR Studio 5.0\Vs\Avr.common.targets" was built successfully.
Done building target "Build" in project "AVRGCC2.avrgccproj".
Done building project "AVRGCC2.avrgccproj".

Build succeeded.
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====

At the bottom of the IDE, there are tabs for 'Error List', 'Command Window', 'Immediate Window', and 'Output'.

DEBUGGER



The screenshot shows a debugger window with the following components:

- Disassembly:** AVRGCC2
- AVRGCC2.c** (Active File)
- main** (Current Function)
- Code View:** Shows the source code of the program. A yellow arrow points to the line `int r = add(p, q);` in the `main` function.
- Locals:** A table showing the current values of local variables.

```
/*
 * AVRGCC2.c
 *
 * Created: 10/18/2011 3:52:31 PM
 * Author: Jacob
 */

#include <avr/io.h>

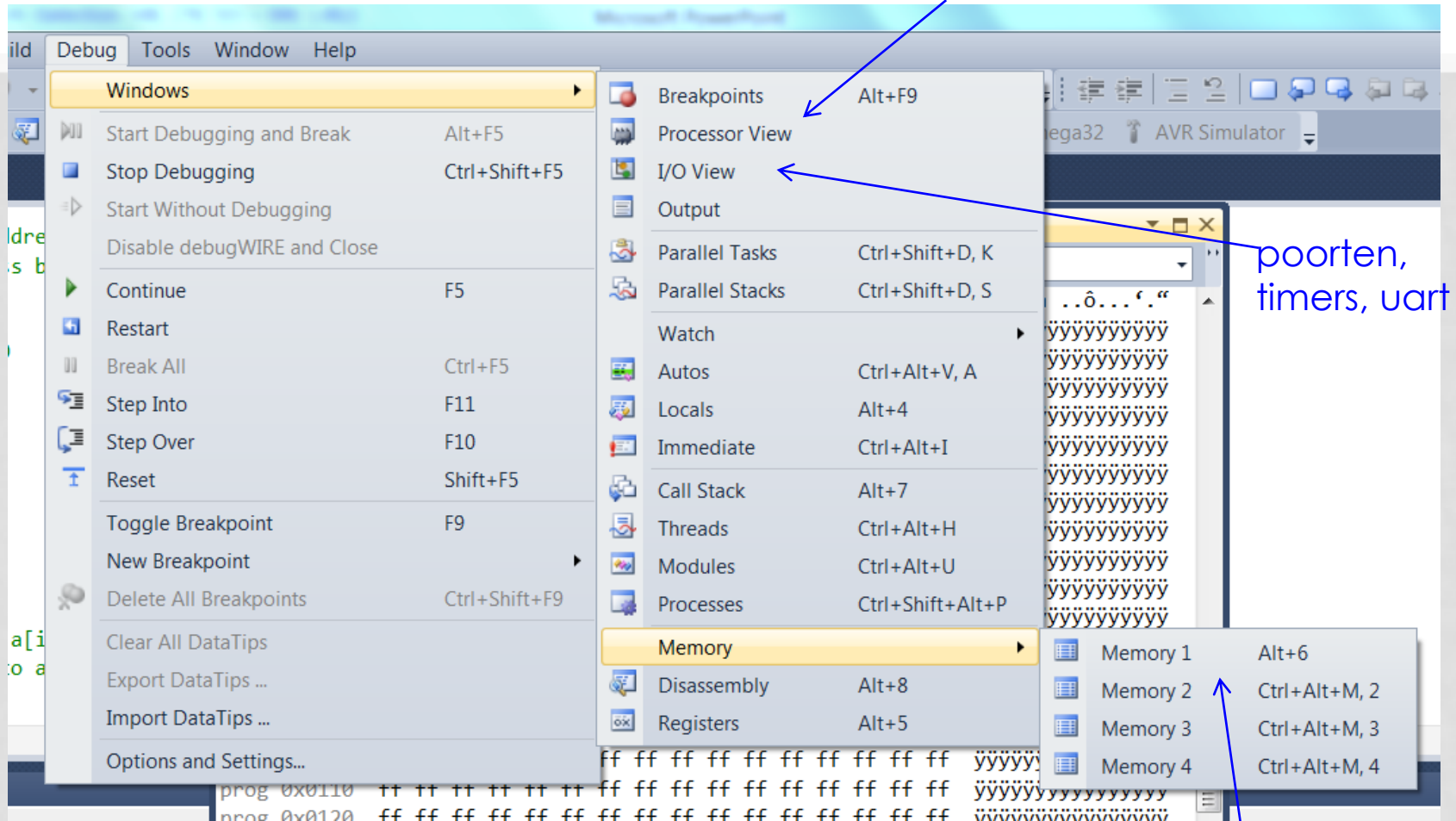
int add(int a, int b);

int main(void)
{
    int p = 3, q = 5;
    int r = add(p, q);
    while(1){};
    return 0;
}
```

Name	Value	Type
p	3	int(data)@0x0856 (FP+1)
q	5	int(data)@0x0858 (FP+3)
r	14592	int(data)@0x085a (FP+5)

DEBUGGER

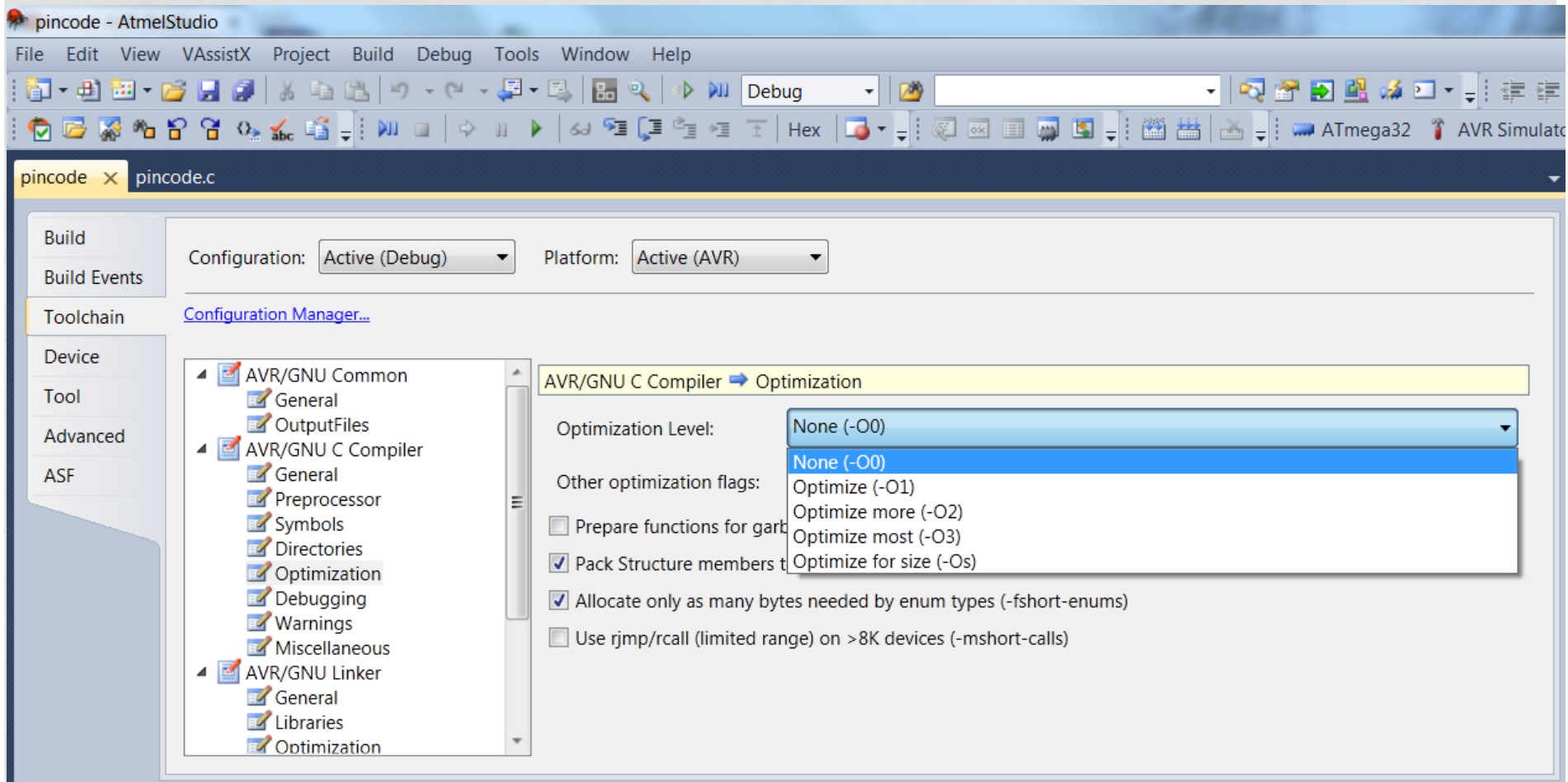
CPU registers



tip : in processor, I/O en memory window kun je ook waarden aanpassen

code en data geheugen

DEBUGGER : NO OPTIMIZATION !



AVR-GCC, AVR-OBJCOPY EN AVRDUDE

- `avr-gcc` : cross compiler/linker
 - ELF formaat : Executable and Linking Format
 - = standaard binary formaat voor unix-varianten op x86
- `avr-objcopy`
 - extract `.elf` naar `.hex`
 - Intel HEX veel gebruikt voor MCU's
 - is een tekst file
- `avrdude`
 - download `.hex` naar MCU flash memory

AVR-GCC, AVR-OBJCOPY EN AVRDUDE

```
avr-gcc -W -mmcu=atmega32 -Os myStk500test.c -o  
myStk500test.elf
```

```
avr-objcopy -j .text -j .data -O ihex  
myStk500test.elf myStk500test.hex
```

```
avrdude -v -p m32 -c STK500 -e -P /dev/ttyUSB0 -U  
flash:w:myStk500test.hex
```


AGENDA

- timer/counter
- switch bounce
- intro C
- de structuur van een C-programma
- ATMEL studio
- **AVR libc**
- register access
- typen en constanten

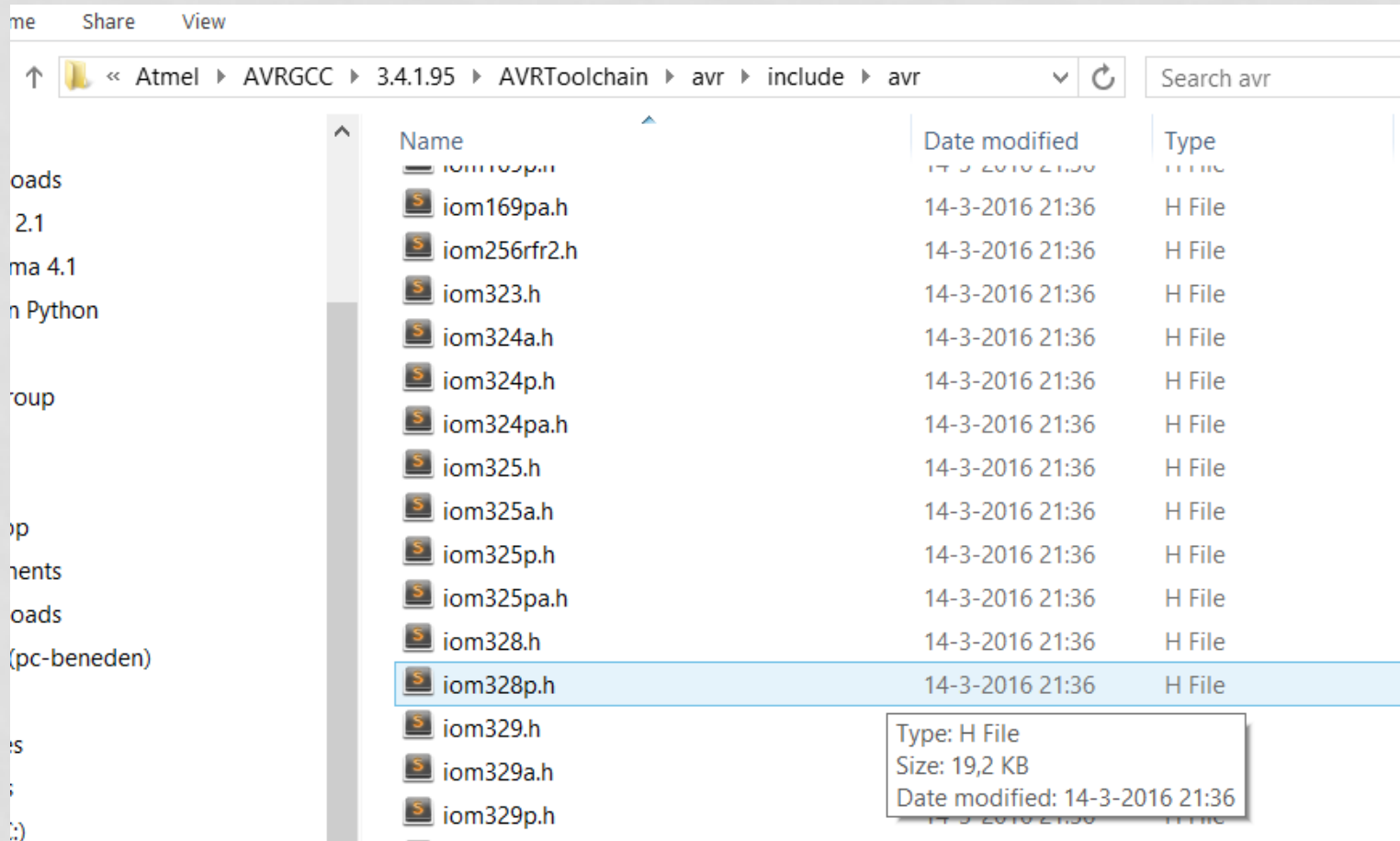
AVR LIBC

- subset van de standard C library
 - aangepast voor AVR MCU's
- startup code voor applicatie
 - immers, er is geen OS
- library header files in
 - `c:\Program Files\Atmel\Atmel Studio 6.0\extensions\Atmel\AVRGCC\3.4.1.95\AVRToolchain\avr\include`
- manual
 - <http://www.nongnu.org/avr-libc/user-manual/index.html>

ENKELE AVR LIBC MODULES

- `<avr/io.h>`: device-specific I/O
 - definities voor de Atmega328P
- `<avr/interrupt.h>`: interrupts
- `<avr/delay.h>` : delays
 - compiler optimizations must be enabled
 - delay time must be a constant
- `<stdint.h>`: standard integer types
 - e.g. `uint8_t`
- `<avr/sfr_defs.h>` : some handy macro's
 - e.g. `_BV(x)`, `loop_until_bit_is_set()`

MAPPING IO.H



Name	Date modified	Type
iom169pa.h	14-3-2016 21:36	H File
iom256rfr2.h	14-3-2016 21:36	H File
iom323.h	14-3-2016 21:36	H File
iom324a.h	14-3-2016 21:36	H File
iom324p.h	14-3-2016 21:36	H File
iom324pa.h	14-3-2016 21:36	H File
iom325.h	14-3-2016 21:36	H File
iom325a.h	14-3-2016 21:36	H File
iom325p.h	14-3-2016 21:36	H File
iom325pa.h	14-3-2016 21:36	H File
iom328.h	14-3-2016 21:36	H File
iom328p.h	14-3-2016 21:36	H File
iom329.h	14-3-2016 21:36	H File
iom329a.h	14-3-2016 21:36	H File
iom329p.h	14-3-2016 21:36	H File

MAPPING IO.H

- in project file :

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <SchemaVersion>2.0</SchemaVersion>
    ...
    <avrdevice>ATmega328P</avrdevice>
```

- #include <avr/io.h> is mapped to :
C:\Program Files (x86)\Atmel\Atmel Studio 6.0\extensions\Atmel\AVRGCC\3.4.1.95\AVRToolchain\avr\include\avr\iom328p.h

AGENDA

- timer/counter
- switch bounce
- intro C
- de structuur van een C-programma
- ATMEL studio
- AVR libc
- **register access**
- typen en constanten

OPERATOREN

- rekenkundige operatoren: +, -, *, /, %, ++, --
- relationele operatoren: >, >=, <, <=, ==, !=
- logische operatoren: &&, ||, !
- bitwise operatoren: >>, <<, &, |, ~, ^
- toekennings operatoren: =, +=, etc.

BITWISE OPERATOREN

Symbol	Operator
&	bitwise AND
	bitwise inclusive OR
^	bitwise XOR (eXclusive OR)
<<	left shift
>>	right shift
~	bitwise NOT (one's complement) (unary)

STDINT.H

- integers met vaste storage size
- `uint8_t` : 8 bit unsigned 0..255
`uint8_t pins = PIND;`
- `uint16_t` : 16 bit unsigned 0..65535
`OCR1A = (uint16_t)43200;`
- in C *geen* type byte (er is wel een unsigned char)
- `uchar = uint8_t`
- see http://www.nongnu.org/avr-libc/user-manual/group__avr__stdint.html

REGISTER ACCESS IN C

```
// _BV(b) is een macro voor (1<<b)

if (PIND & _BV(4)){}; // if bit 4 is set

PORTB = 0xff;

PORTB = (1<<5)|(1<<7); // PORTB=0b1010 0000

GICR |= (1<<INT1); // enable INT1

MCUCR |= (1<<ISC11)|(1<<ISC10);

PORTB |= (1<<PB0); // set PB0

PORTB &= ~(1<<PB0); // clear PB0

PORTB ^= (1<<PB0); // toggle PB0 (1 => 0 and 0 => 1)
```

**XOR Truth
Table**

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

REGISTER ACCESS IN C

```
DDRB = 0x0F;           // lower nibble = output, higher nibble = input
```

```
DDRA = 0x00;           // set port A as input  
x = PINA;               // read contents of port A
```

```
DDRB = 0xFF;           // set all pins of port B as output  
PORTB = 0xFF;           // write data to port B
```

```
DDRC |= _BV(4);         // set only pin 4 of port C as output  
PORTC |= _BV(4);         // write 1 to pin 4 of port C
```

```
DDRC &= ~(1<<5);        // set only pin 5 of port C as input  
x = PINC & (1<<5)       // read pin 5 of port C
```

AGENDA

- timer/counter
- switch bounce
- intro C
- de structuur van een C-programma
- ATMEL studio
- AVR libc
- register access
- **typen en constanten**

TYPEN IN C

- basis typen
 - integer
 - float
 - enumerated typen
 - 0, 1, 2, ...
 - char
 - type void
 - pointer typen
- samengestelde typen
 - arrays
 - structures

LITERALS EN CONSTANTEN

- voorbeelden literal constants :
 - 1234 is een int
 - 2221UL is unsigned long
 - 33.4 is een double
 - 665.77f is een float
- 0x01AB : 0x voor een geheel getal is hexadecimaal
- character constanten: 'a', 'b', '\n', '\t', etc.
- string literals: "Hello world!\n"
 - een string is een array van chars eindigend op '\0'
- 'A' is één karakter en "A" zijn er twee !