Introduction to Microcontrollers

Sensors & Registers – Alternate Session

Goals

- Learn more about sensors
- Do measurements from a simple temperature sensor
- Recap of registers

Sensors

- Converts/Samples data from physical world
- Useful for most systems
- Typical sensor measurements
  - Temperature
  - Pressure
  - Acceleration
  - ++ (Anything really)
- Typical output formats
  - Analog signal (Voltage or current)
  - Data bus
  - PWM
- Sensors are not perfect
  - Resolution
  - Tolerance
  - Transfer function
    - Usually not linear!
- Calibration
  - No sensor is identical
  - Accurate measurements require calibration for each sensor

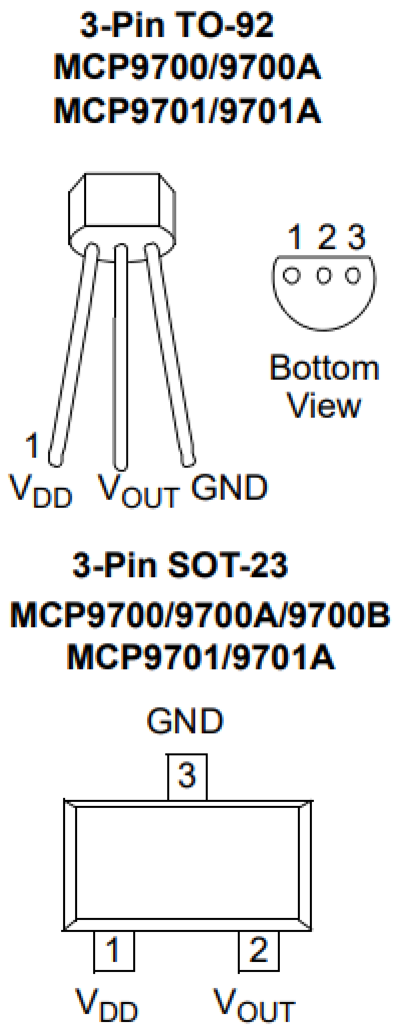
Sensors

- AVRs are well suited to make sensors!
  - Cheap
  - Power efficient
  - Decent set of peripherals
  - Can offload main processor
  - Simplify sensor interface
  - Autocalibration
- AVRs are decent at mathematics
  - For 8-bit micros
  - Smart sensors
- Don’t be afraid to make modules with AVRs
  - Simplify interfaces
  - Streamline your prototyping

- A few examples of sensors I’ve made using AVRs
  - Capacitance meter
  - Position decoder
  - Photoelectric presence detector
  - Tank level meter
  - eFuse
  - Thermocouple front end
  - Sound spectrum analyzer
  - ++

MCP9700

- Temperature sensor
  - Analog voltage output
  - Microcontroller interface = ADC
- Supply voltage range
  - 2.3 – 5.5 Volts
- Output voltage range
  - 0.1 – 1.75 Volts
  - Dependent on temperature



Task: MCP9700 (15-20 min)

- Go to github
  - Part 5
  - Link to datasheet for sensor
  - Guide on how to print floating points
- Create a new project
  - Copy UART + ADC project from last time
- Measure temperature and print to TeraTerm
  - Read the datasheet to find the voltage to temperature transfer function
    - Make some assumptions
  - Round temperature to nearest degree Celsius
    - printf()

Task: MCP9700 Solution

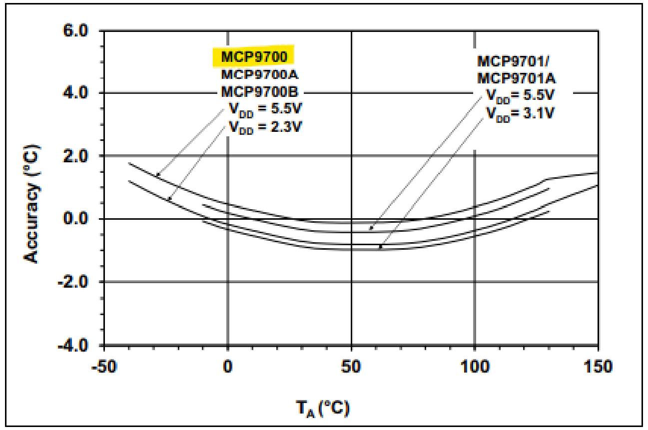


FIGURE 2-5: *Accuracy vs. Ambient Temperature, with V_{DD}.*

EQUATION 4-1: SENSOR TRANSFER FUNCTION

$$V_{OUT} = T_C \times T_A + V_{0°C}$$

Where:

T_A = Ambient Temperature

V_{OUT} = Sensor Output Voltage

$V_{0°C}$ = Sensor Output Voltage at 0°C (see **DC Electrical Characteristics** table)

T_C = Temperature Coefficient (see **DC Electrical Characteristics** table)

Assumptions:

V_OUT is linear

- OK, at the cost of non-linear accuracy
- What the datasheet recommends

V_0C = 500 mV

- OK, since accuracy is not good anyways
- For good sensors, this should be measured

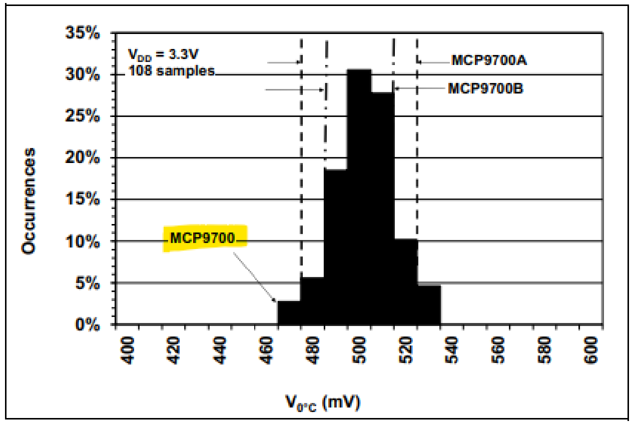


FIGURE 2-8: *Output Voltage at 0°C (MCP9700/9700A/9700B).*

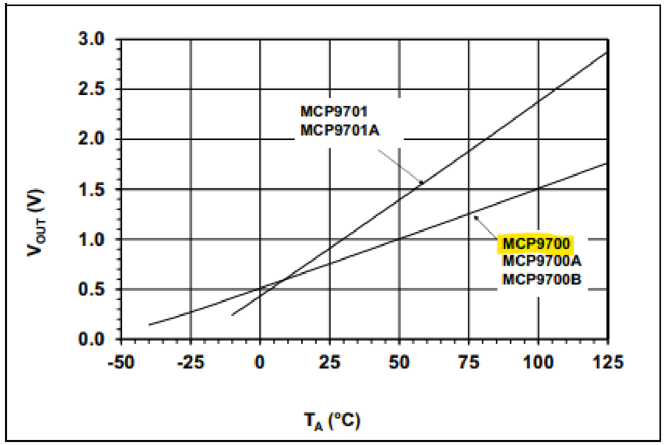


FIGURE 2-17: *Output Voltage vs. Ambient Temperature.*

Task: MCP9700 Solution

EQUATION 4-1: SENSOR TRANSFER FUNCTION

$$V_{OUT} = T_C \times T_A + V_{0°C}$$

Where:

T_A	=	Ambient Temperature
V_{OUT}	=	Sensor Output Voltage
$V_{0°C}$	=	Sensor Output Voltage at 0°C (see **DC Electrical Characteristics** table)
T_C	=	Temperature Coefficient (see **DC Electrical Characteristics** table)

$$V_{OUT} = T_C T_A + V_{0°C}$$

$$V_{OUT} = V_{REF} \frac{\mathrm{RES}}{2^B - 1}$$

$$T_C = 0.01 \;\; [\mathrm{V}\;\;/\mathrm{C}]$$

$$V_{0°C} = 0.5 \; [\mathrm{V}]$$

$$T_A = \frac{\mathrm{RES} \cdot V_{REF} - V_{0°C}(2^B - 1)}{T_C(2^B - 1)}$$

Sensor Output						
Output Voltage, T_A = 0°C	$V_{0°C}$	—	500	—	mV	**MCP9700/9700A/9700B**
Temperature Coefficient	T_C	—	10.0	—	mV/°C	**MCP9700/9700A/9700B**

$$T_A = \frac{\mathrm{RES} V_{REF} - V_{0°C}(2^B - 1)}{T_C(2^B - 1)}$$

$$B = 12$$

$$V_{REF} = \mathrm{VDD} = 3.3$$

$$T_A = \frac{3.3 \cdot \mathrm{RES} - 0.5 \cdot 4095}{0.01 \cdot 4095}$$

$$T_A = \frac{3.3\mathrm{RES} \text{ - } 2047.5}{40.95}$$

Floats are not optimal for most microcontrollers, but sometimes they are ok to use. Two alternatives are using integers with scaling, or fixed point notation

Task: MCP9700 Solution

- Solution on github
  - As well as how to get printf to print floats

Recap: Registers

- What are these C/C++ operators?
  - Boolean
    - &&
    - ||
    - !
    - AND
    - OR
    - NOT
  - Bitwise
    - &
    - |
    - ^
    - ~
    - >>
    - <<
    - and
    - or
    - xor
    - not
    - right shift
    - left shift

AND

A	B	=
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	0

```
  1 0 0 1
& 1 0 1 0
---------
= 1 0 0 0
```

```
  1 0 0 1
| 1 0 1 0
---------
= 1 0 1 1
```

```
  1 0 0 1
^ 1 0 1 0
---------
= 0 0 1 1
```

```
Variable
  1 0 0 1 1 1 0 1   <<   3
= 1 1 1 0 1 0 0 0
```

```
Variable
  1 0 0 1 1 1 0 1   >>   3
= 0 0 0 1 0 0 1 1
```

```
Variable
~ 1 0 0 1 1 1 0 1
= 0 1 1 0 0 0 1 0
```

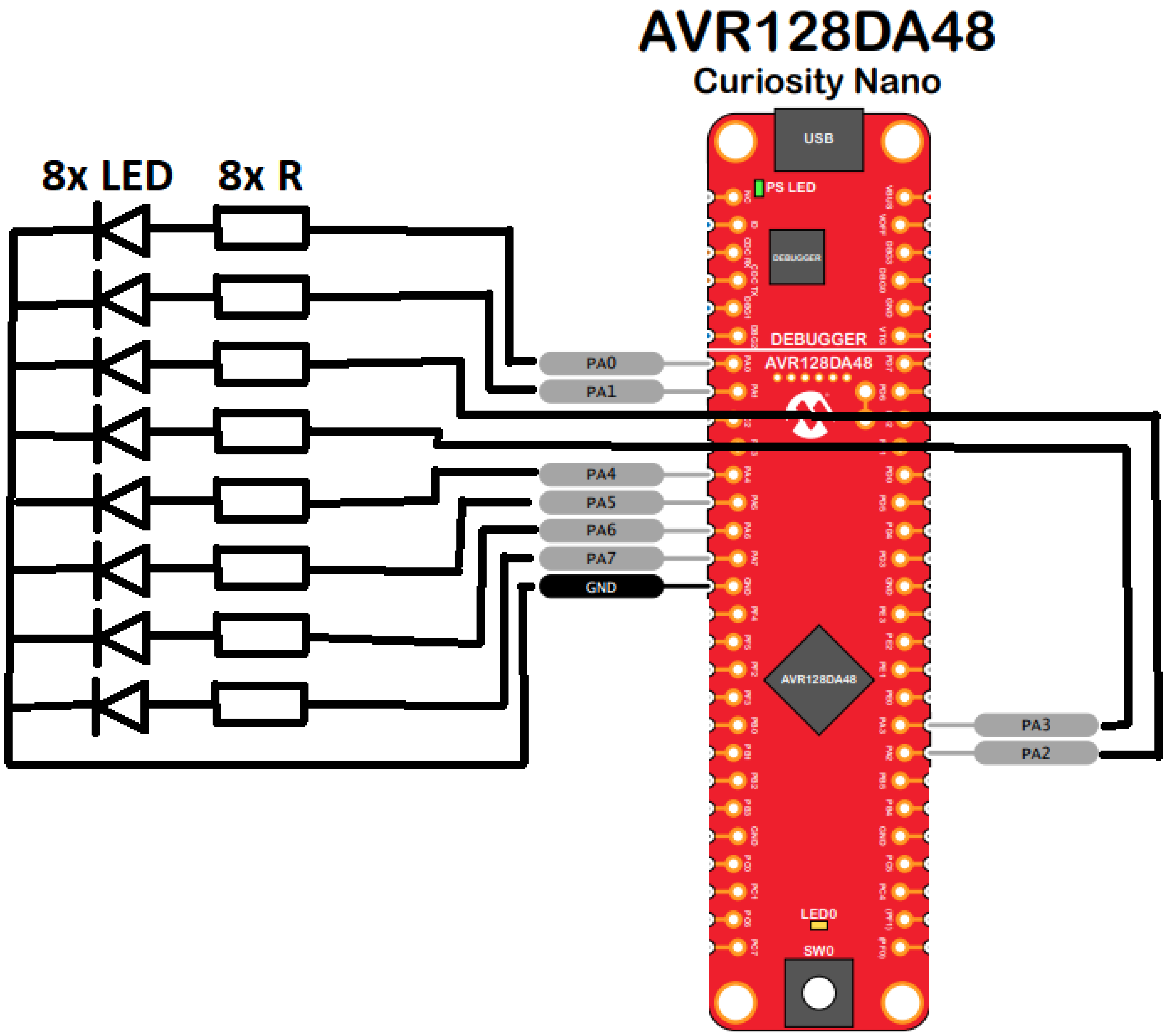
Recap: Registers

- These are equivalent

• X |= Y	X = X | Y	xxxSET = Y
• X &= ~Y	X = X & ~Y	xxxCLR = Y
• X ^= Y	X = X ^ Y	xxxTGL = Y

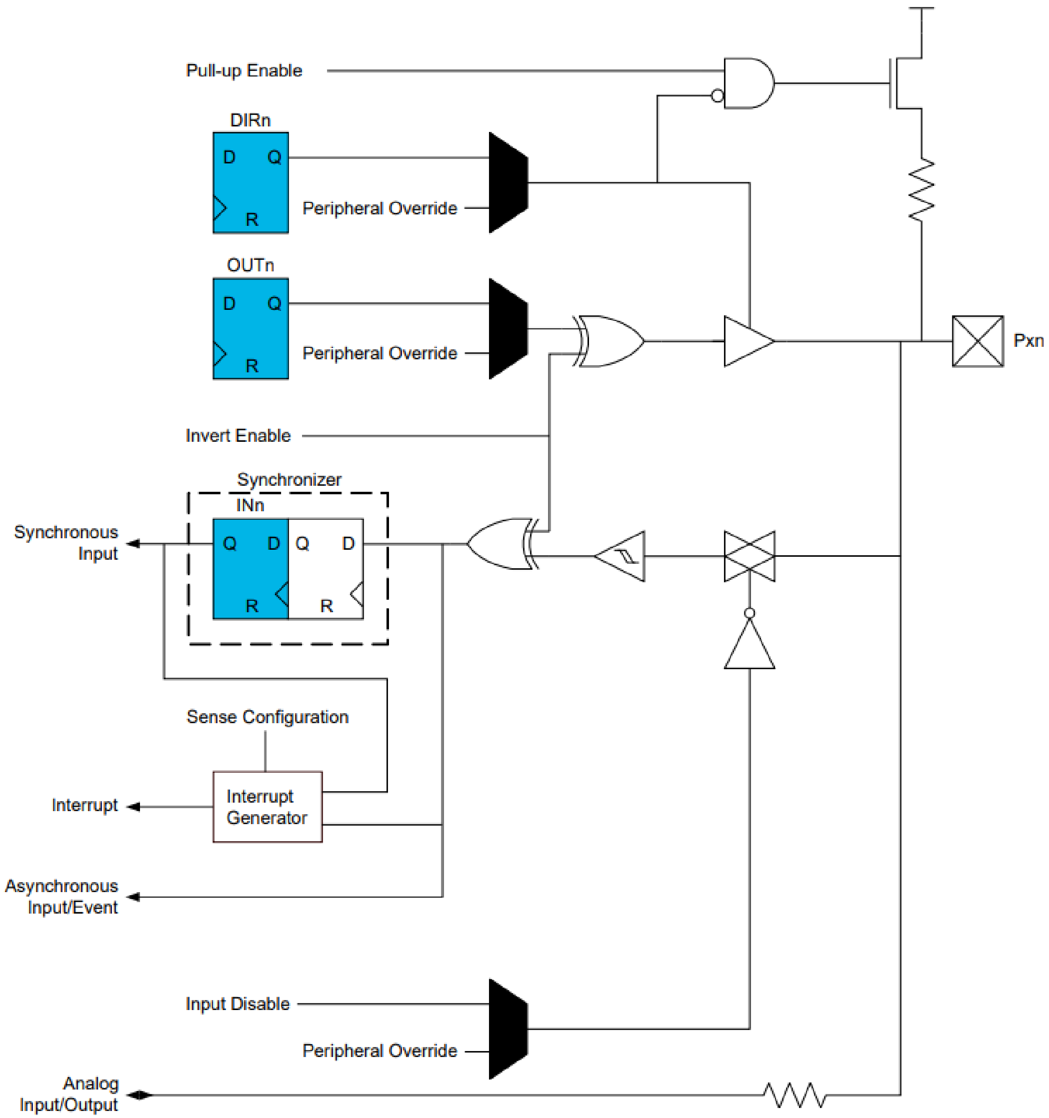
Recap: Registers

- Hardware setup for tasks
  - 8x Resistors
  - 8x LEDs
  - 2x wires for PA2 and PA3
    - Make sure PORTC is input
  - Press into breadboard
- LED
  - Long leg – to GPIO via resistor
    - One resistor per LED
  - Short leg – Connect to GND
    - Every short leg connected to GND



Recap: Registers

- PORT



Recap: Registers – Test code!

- We will learn about registers with PORTA
- Start by making all pins connected to PORTA outputs
  - PORTA.DIR = 0xFF;
- Increment PORTA.OUT register value
  - PORTA.OUT++;
  - Add delay!
- What happens?

```
// Define clock frequency for util/delay.h
#define F_CPU 4000000

// Include memory mapped peripheral addresses
#include <avr/io.h>

// Include util/delay.h for _delay_ms()
#include <util/delay.h>


int main(void)
{
    // Make every pin on PORTA outputs (Connected to LEDs)
    PORTA.DIR = 0xFF; // = 0b11111111 = 255 = PIN7_bm | PIN6_bm | PIN5_bm | ... | PIN0_bm;

    while (1)
    {
        // Increment OUT value
        PORTA.OUT++;
        _delay_ms(500);
    }
}
```

Recap: Registers – Test code!

- What obout OUTSET?
- What do you think will happen?

```c
// Define clock frequency for util/delay.h
#define F_CPU 4000000

// Include memory mapped peripheral addresses
#include <avr/io.h>

// Include util/delay.h for _delay_ms()
#include <util/delay.h>


int main(void)
{
    // Make every pin on PORTA outputs (Connected to LEDs)
    PORTA.DIR = 0xFF; // = 0b11111111 = 255 = PIN7_bm | PIN6_bm | PIN5_bm | ... | PIN0_bm;

    uint8_t outval = 0;
    while (1)
    {
        // Increment OUT value
        PORTA.OUTSET = outval++;
        _delay_ms(500);
    }
}
```

Recap: Registers – Test code!

- What about incrementing DIR?
- Does it matter that the OUT register is written before DIR?

```
int main(void)
{
    // Store '1's in the OUT registers for PORTA
    PORTA.OUT = 0xFF;

    while (1)
    {
        // Increment direction register
        PORTA.DIR++;
        _delay_ms(500);
    }
}
```

Recap: Registers

1 0 0 1 1 0 1 0

2^7 2^6 2^5 . . . . 2^0

Dec	Bin	Hex
3	0b00000011	0x03
127	0b01111111	0x7F
170	0b10101010	0xAA

- What LED is lit up after
  - T0?
  - T1?
  - T2?
  - T3?
  - T4?
  - T5?

```
while (1)
{
    PORTA.OUT = 1;                      // T0
    _delay_ms(500);

    PORTA.OUT = 0x01;                   // T1
    _delay_ms(500);

    PORTA.OUT = 0b00000001;             // T2
    _delay_ms(500);

    PORTA.OUT = PIN1_bm;                // T3
    _delay_ms(500);

    PORTA.OUT = PIN2_bm | PIN3_bm;      // T4
    _delay_ms(500);

    PORTA.OUT &= ~PIN2_bm;              // T5
    _delay_ms(500);
}
```

Recap: Registers – Test code!

- How do the LEDs behave with this code?

```
int main(void)
{
    PORTA.DIR = 0xFF;

    while (1)
    {
        PORTA.OUT = PIN3_bm | PIN4_bm;
        _delay_ms(500);
        PORTA.OUT = 0;
        _delay_ms(500);
    }
}
```

- What about this one?

```
int main(void)
{
    PORTA.DIR = 0xFF;

    while (1)
    {
        PORTA.OUT = 0xAA;
        _delay_ms(500);
        PORTA.OUT = 0x55;
        _delay_ms(500);
    }
}
```

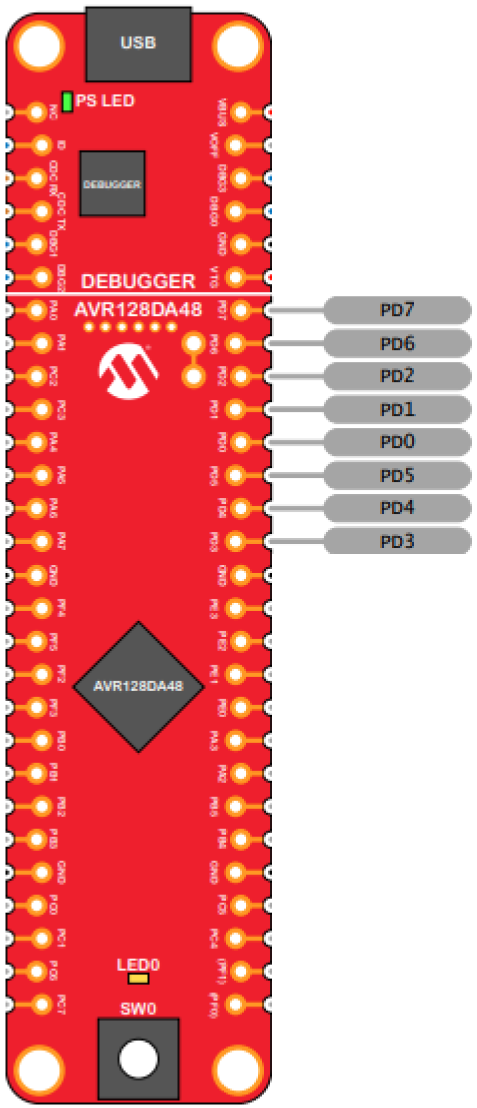
Recap: Registers – Test code!

- What does this code do?

```c
int main(void)
{
    PORTA.DIR = 0xFF;
    PORTD.DIR = 0x00;

    while (1)
    {
        PORTA.OUT = PORTD.IN;
    }
}
```

- What happens if you apply voltages to PORTD?



Recap: Registers

- What happens in this scenario?
- Is this *illegal*?!

```c
int main(void)
{
    PORTA.DIR   = 0xFF;
    PORTA.OUT   = TCB_RUNSTDBY_bm
                | TCB_CLKSEL_DIV2_gc
                | TCB_ENABLE_bm;

    while (1)
    {
    }
}
```

- Try it!

22.5.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY	CASCADE	SYNCUPD	CLKSEL[2:0]			ENABLE
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 6 – RUNSTDBY Run Standby
Writing a '1' to this bit will enable the peripheral to run in Standby sleep mode.

Bit 5 – CASCADE Cascade Two Timer/Counters
Writing this bit to '1' enables cascading of two 16-bit Timer/Counters type B (TCBn) for 32-bit operation using the Event System. This bit must be '1' for the timer/counter used for the two Most Significant Bytes (MSBs). When this bit is '1', the selected event source for capture (CAPT) is delayed by one peripheral clock cycle. This compensates the carry propagation delay when cascading two counters via the Event System.

Bit 4 – SYNCUPD Synchronize Update
When this bit is written to '1', the TCB will restart whenever TCAn is restarted or overflows. This can be used to synchronize capture with the PWM period. If TCAn is selected as the clock source, the TCB will restart when that TCAn is restarted. For other clock selections, it will restart together with TCA0.

Bits 3:1 – CLKSEL[2:0] Clock Select
Writing these bits selects the clock source for this peripheral.

Value	Name	Description
0x0	DIV1	CLK_PER
0x1	DIV2	CLK_PER / 2
0x2	TCA0	CLK_TCA from TCA0
0x3	TCA1	CLK_TCA from TCA1
0x4	-	Reserved
0x5	-	Reserved
0x6	-	Reserved
0x07	EVENT	Positive edge on event input

Bit 0 – ENABLE Enable
Writing this bit to '1' enables the Timer/Counter type B peripheral.

Recap: Registers

- What is the difference?

```
int main(void)
{
    PORTA.DIR   = 0xFF;

    while (1)
    {
        PORTA.OUT |= TCB_CLKSEL_DIV2_gc;
        _delay_ms(500);

        PORTA.OUT |= TCB_CLKSEL_DIV1_gc;
        _delay_ms(500);
    }
}

int main(void)
{
    PORTA.DIR   = 0xFF;

    while (1)
    {
        PORTA.OUT = TCB_CLKSEL_DIV2_gc;
        _delay_ms(500);

        PORTA.OUT = TCB_CLKSEL_DIV1_gc;
        _delay_ms(500);
    }
}
```

22.5.1 Control A

Name: CTRLA
Offset: 0x00
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY	CASCADE	SYNCUPD		CLKSEL[2:0]		ENABLE
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bit 6 – RUNSTDBY Run Standby
Writing a '1' to this bit will enable the peripheral to run in Standby sleep mode.

Bit 5 – CASCADE Cascade Two Timer/Counters
Writing this bit to '1' enables cascading of two 16-bit Timer/Counters type B (TCBn) for 32-bit operation using the Event System. This bit must be '1' for the timer/counter used for the two Most Significant Bytes (MSBs). When this bit is '1', the selected event source for capture (CAPT) is delayed by one peripheral clock cycle. This compensates the carry propagation delay when cascading two counters via the Event System.

Bit 4 – SYNCUPD Synchronize Update
When this bit is written to '1', the TCB will restart whenever TCAn is restarted or overflows. This can be used to synchronize capture with the PWM period. If TCAn is selected as the clock source, the TCB will restart when that TCAn is restarted. For other clock selections, it will restart together with TCA0.

Bits 3:1 – CLKSEL[2:0] Clock Select
Writing these bits selects the clock source for this peripheral.

Value	Name	Description
0x0	DIV1	CLK_PER
0x1	DIV2	CLK_PER / 2
0x2	TCA0	CLK_TCA from TCA0
0x3	TCA1	CLK_TCA from TCA1
0x4	-	Reserved
0x5	-	Reserved
0x6	-	Reserved
0x07	EVENT	Positive edge on event input

Bit 0 – ENABLE Enable
Writing this bit to '1' enables the Timer/Counter type B peripheral.

Recap: Registers

- What happens?

```c
int main(void)
{
    PORTA.DIR = 0xFF;                           // T0
    PORTA.OUT = 0xFF;                           // T1
    PORTA.OUTCLR = PIN1_bm;                     // T2
    PORTA.OUTSET = PIN7_bm;                     // T3
    PORTA.OUTTGL = PIN7_bm;                     // T4
    PORTA.OUTTGL = (1<<7);                      // T5
    PORTA.OUTCLR = 0b10000000;                  // T6
    PORTA.OUT &= ~PIN2_bm;                      // T7
    PORTA.OUT &= ~(PIN3_bm | PIN4_bm);  // T8

    while (1)
    {

    }
}
```

Recap: Registers

- What is the difference?

```c
int main(void)
{
    PORTA.DIRSET = PIN0_bm | PIN4_bm;

    while (1)
    {
        if (PORTA.IN & PIN4_bm)
        {
            PORTA.OUTTGL = PIN0_bm;
        }
    }
}
```

```c
int main(void)
{
    PORTA.DIRSET = PIN0_bm | PIN4_bm;
    PORTA.OUTSET = PIN4_bm;

    while (1)
    {
        if (PORTA.IN & PIN4_bm)
        {
            PORTA.OUTTGL = PIN0_bm;
        }
    }
}
```

Recap: Registers

- What is the C/C++ equivalent for these
  - A = Variable (in a struct)
  - B = Value

A B

```
int main(void)
{
    PORTA.OUTSET = PIN6_bm;

    while (1)
    {

    }
}
```

Recap: Registers

- Finish the code
- Use the datasheet to create the correct logic
- Hint:
  - Mechanical buttons bounce when pressed
    - Micros can see this and multiple presses are registered
  - Debouncing is a glitch filter
  - There are better solutions for this, but delay is simple

```
int main(void)
{
    PORTA.DIR = 0xFF; // PA0, PA1, ..., PA7 = OUTPUT
    PORTC.PIN7CTRL = PORT_PULLUPEN_bm;  // Button pull-up enable

    while (1)
    {
        if ( <BUTTON PRESSED> )
        {
            // Button debounce
            _delay_ms(10);

            // Wait for button to be released
            while ( <BUTTON PRESSED> );

            <INCREMENT PORT A OUT REGISTER>
        }
    }
}
```

Recap: Registers

- Other tasks you can try on your own
  - Get familiar with OUT, OUTSET, OUTCLR, OUTTGL by playing with these registers
  - Get familiar with DIR
  - Make some LED(s) light on a particular IN register value
    - Make sure you use pull-up on input pins and short to GND to change value (like button)
    - For example, make PC6 LED light up on PORTA.IN = 123, otherwise off
      - Remember that PC6 is active LOW
  - Make a periodic interrupt with TCB that toggles a LED every
    - 500 ms
    - 250 ms
    - 100 ms
  - Make a periodic interrupt with RTC
    - RTC peripheral are two peripherals in one
      - Use PITCTRLA, PITINTCTRL, PITINTFLAGS
      - Leave the other registers alone
        - TOP value is by default 0xFFFF

Next time

- Learn how to make circuits without the development board
  - We will make our «own development board»
- Modify the development board such that it can program other AVR devices
  - Learn how to use ATMEL ICE programming tool
  - And how to debug your program
- Use what we have learnt to control a DC motor
  - Speed control with PWM and ADC
  - Direction control with PORT
- Unleash the beast
  - Modify the CPU clock frequency to the maximum (24 MHz)
- Last session!
  - Challenge: Think of a project you could do with an AVR (optional)
  - Feedback: Anonymous form (optional, but greatly appreciated)