Introduction to Microcontrollers

What is a microcontroller

The Course

- Main objective: Understand how microcontrollers can be used to make electric circuits

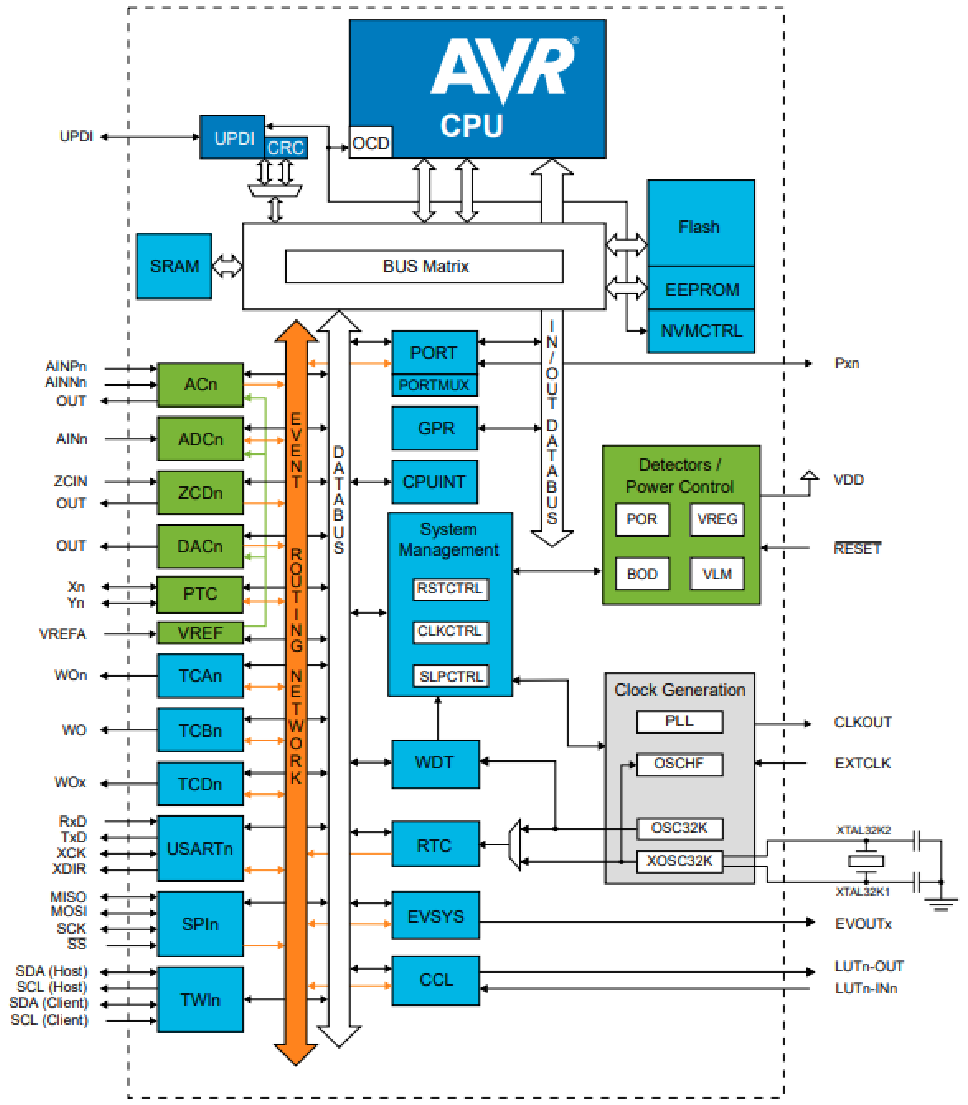
- Course structure
  - 6 parts
  - Gradual movement towards independent work
- Assumptions
  - Know basic coding
  - Know basic electronics
- The course is an introduction
  - Still a lot of information

Goals Today

- Understand what a microcontroller is
- Know where microcontrollers are used
- Know why microcontrollers are used
- Know what an AVR is, and why it is used in this course
- Understand a project design flow
- Blink a LED

What is a microcontroller

- Small, integrated computer
  - Internal memory and peripherals
- Compared to a regular computer
  - Everything is integrated
  - Limited memory
  - Limited processing power
  - Low power consumption
  - Low cost
  - Good real-time performance



What is a microcontroller

- Microcontrollers are found everywhere
    - Replaces old circuits
- Cheap
    - Mass produced
    - Few external components
    - Easy circuit board layout
- Microcontrollers are software controlled
    - Faster time to market
    - Easy to add functionality
    - Software can compensate for some hardware mistakes

What is a microcontroller

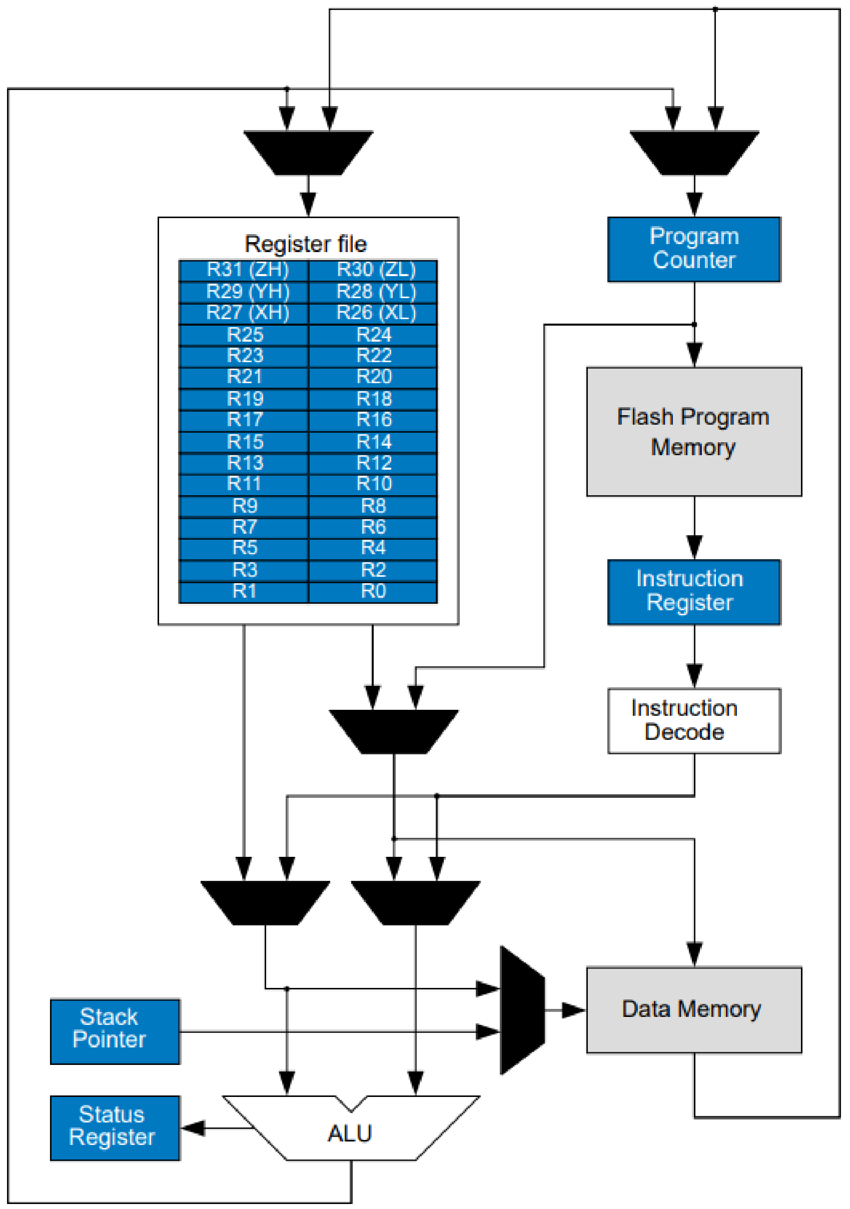
- Pros
  - Cheap
    - Though high end models may become expensive
  - Easy to use
  - Fast time to market
  - Easy to add functionality
  - Easy to make circuits

- Cons
  - Intrinsically complex
    - Simple electric circuits are replaced with complex ones
  - Software bugs
    - Adds an additional layer of debugging
  - Memory dependent
    - Memory has a limited lifetime
  - Use low voltages
    - Regulators may be needed
  - Limited output current
  - Poor at parallel tasks
    - Though some microcontrollers have multiple cores

What is an AVR

- 8-bit microcontroller
  - 135 16- or 32-bit wide instructions
  - ALU connected to 32 8-bit registers
  - Hardware multiplier with fraction support
- RISC-IV
  - Unofficially AVR is Alf and Vegard’s RISC processor
  - Made in Norway
- Different families with different capabilities
  - In this course we will focus on the DA family

Figure 6-1. AVR® CPU Architecture



Why AVR

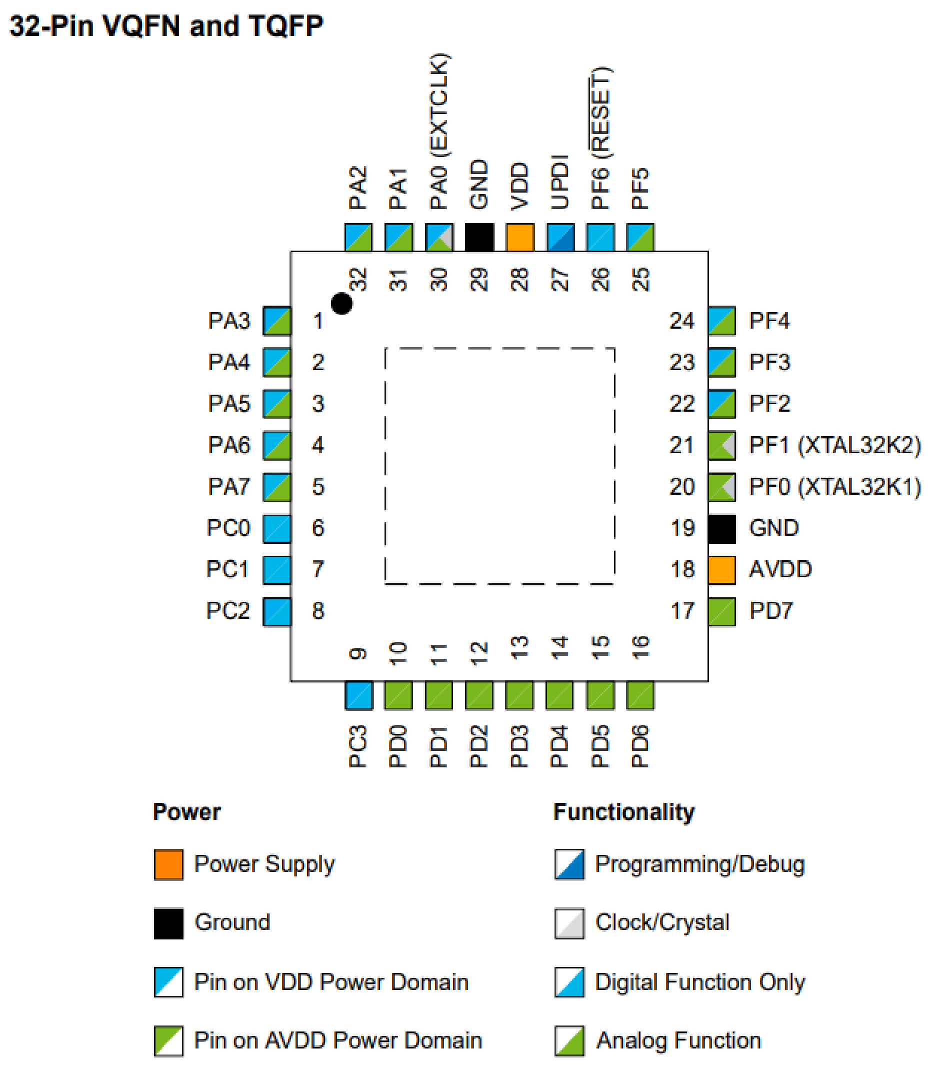
- Simple microcontrollers
  - Easier to understand peripherals and architecture
  - Most projects are simple and don’t need a high-end microcontroller
- Scalable
  - Higher end microcontrollers such as ARM based ones are similar
    - Though peripherals are usually more complex for ARM
  - Datasheets are identical
    - Within the same manufacturer*
- Powerful and robust
  - Very reliable due to their simplicity
- 5V compatible
  - Which makes the transition from Arduino easier

Microcontroller design flow

- Plan
  - High Level Design
  - Specification for hardware and software
- Software design
  - Starting software development early prevents mistakes
  - It is usually better to understand what microcontroller is needed once software development has started
- Create a schematic of your circuit
  - It is especially important that inputs/outputs are connected to the correct pins
- Circuit board layout
- Manufacturing
- Testing of hardware and software
- Verification

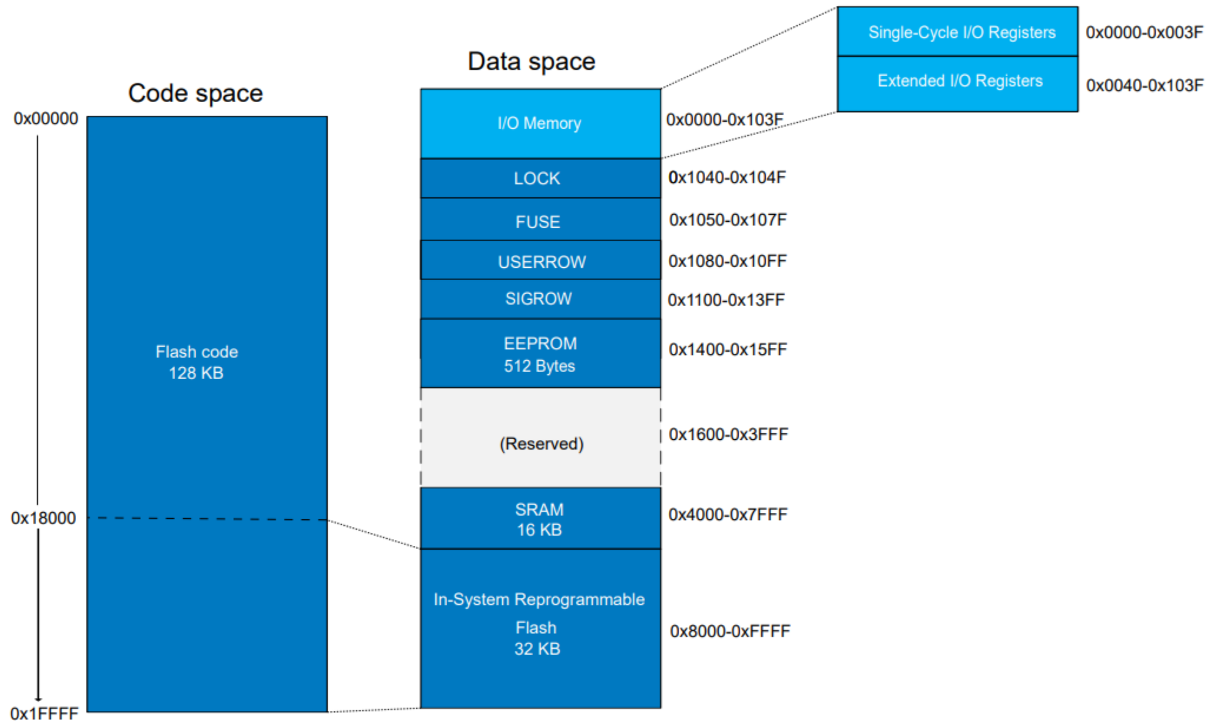
Software to hardware

- A microcontroller connects to other hardware through “pins”
  - Physical pins on the integrated circuit package are connected to traces on a printed circuit board
- Functionality may vary on each pin
- Usually referred to as General Purpose Input/Output (GPIO)



Memory & Data Space

- Memory
  - Instruction memory (Flash)
  - Data memory (RAM + Flash)
- Data space
  - Everything connected to the internal data bus
  - Includes special registers and hardware
    - Memory mapped hardware

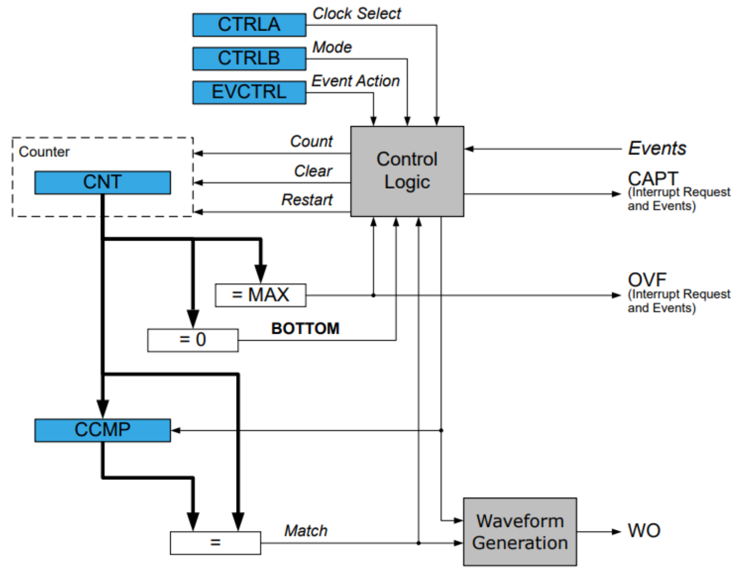


Peripherals

- Peripheral is a circuit that does a dedicated task
- Typical tasks
  - Data conversion
  - Timing
  - Waveform generation
  - Data transfer
  - Control
- A peripheral must be understood before it can be used
  - The datasheet provides clear instructions and descriptions of all peripherals
- AVR Datasheets:
  - Blue boxes are registers we can manipulate

Block Diagram

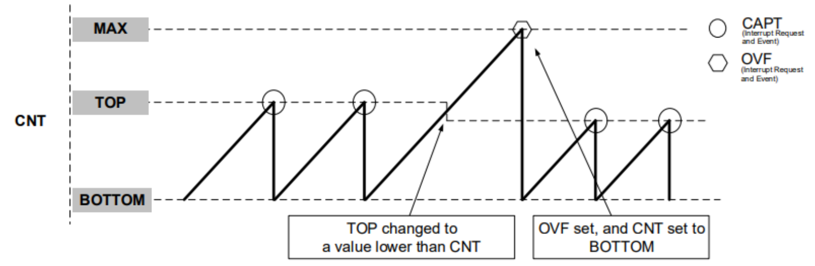
Figure 22-1. Timer/Counter Type B Block Diagram



22.3.3.1.1 Periodic Interrupt Mode

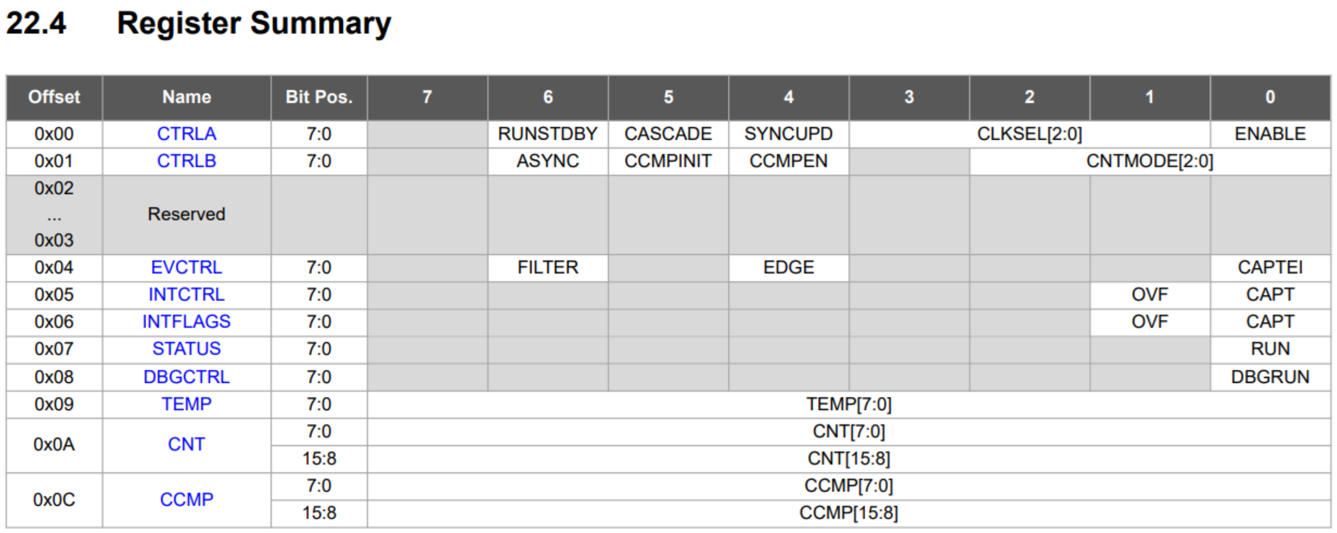
In the Periodic Interrupt mode, the counter counts to the capture value and restarts from BOTTOM. A CAPT interrupt and event is generated when the CNT is equal to TOP. If TOP is updated to a value lower than CNT, upon reaching MAX, an OVF interrupt and event is generated, and the counter restarts from BOTTOM.

Figure 22-3. Periodic Interrupt Mode



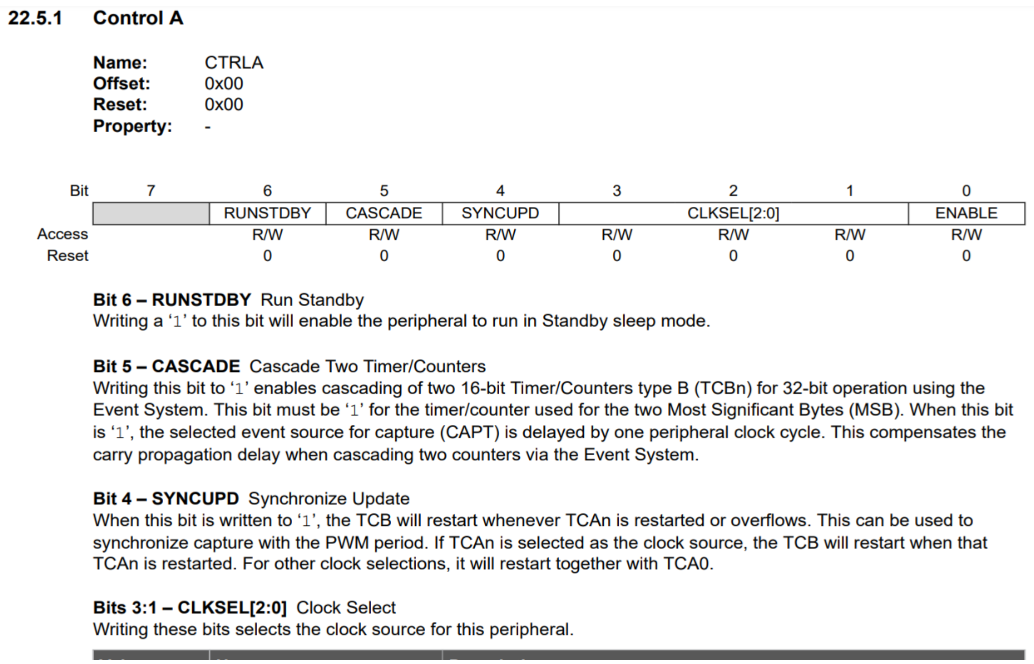
Peripheral Registers

- Inside data space
- Special “variables” connected to peripherals
  - Registers may control behavior of peripheral circuits
    - Example: Pulse-Width Modulation
  - Registers may also provide information from a peripheral
    - Example: Analog to Digital Converter



22.4 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0		RUNSTDBY	CASCADE	SYNCUPD		CLKSEL[2:0]		ENABLE
0x01	CTRLB	7:0		ASYNC	CCMPINIT	CCMPEN			CNTMODE[2:0]	
0x02 ... 0x03	Reserved									
0x04	EVCTRL	7:0		FILTER		EDGE				CAPTEI
0x05	INTCTRL	7:0							OVF	CAPT
0x06	INTFLAGS	7:0							OVF	CAPT
0x07	STATUS	7:0								RUN
0x08	DBGCTRL	7:0								DBGRUN
0x09	TEMP	7:0				TEMP[7:0]				
0x0A	CNT	7:0				CNT[7:0]				
		15:8				CNT[15:8]				
0x0C	CCMP	7:0				CCMP[7:0]				
		15:8				CCMP[15:8]				



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 (MSB). 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.

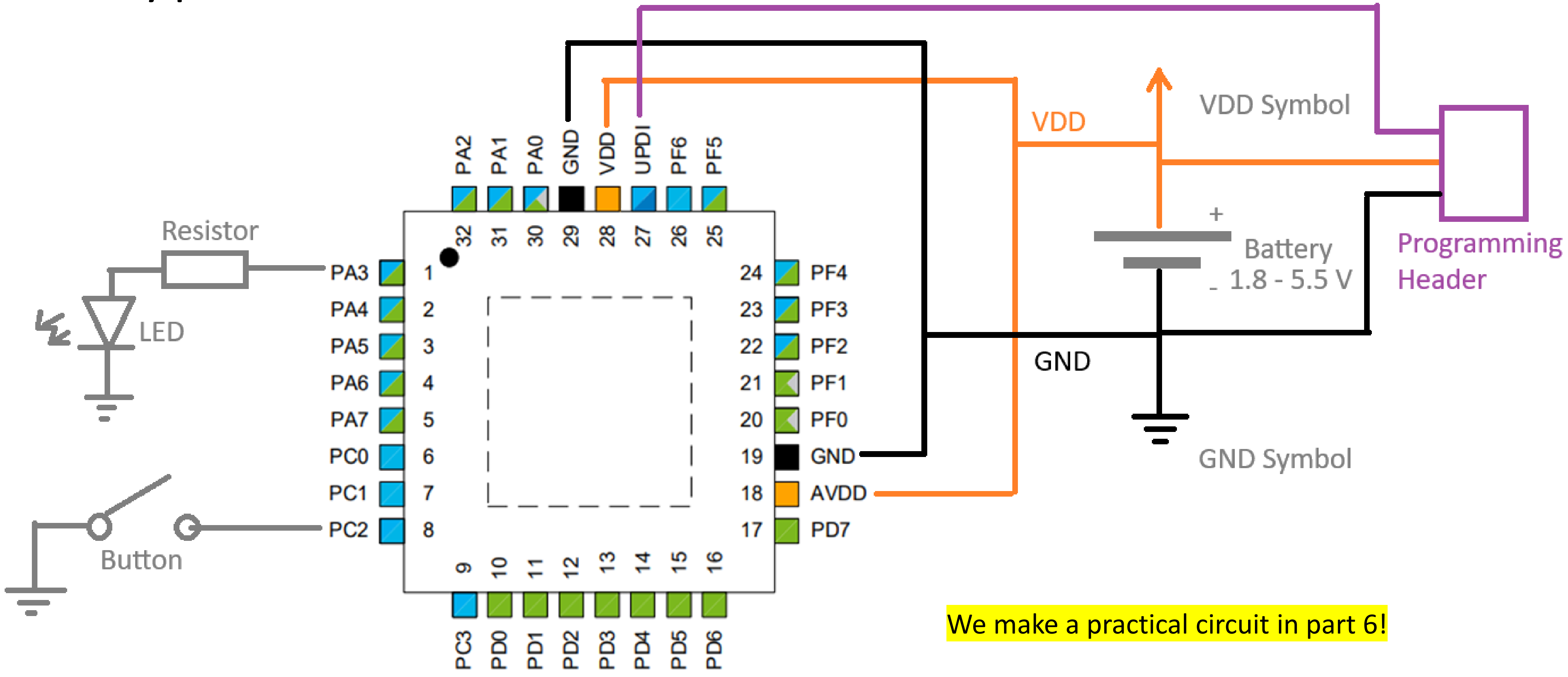
Bit 4 – SYNCUPD Synchronize Update
When this bit is written to ‘1’, the TCB will restart whenever TCAn is restarted or overflows. This can be used to synchronize capture with the PWM period. If TCAn is selected as the clock source, the TCB will restart when that TCAn is restarted. For other clock selections,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

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

Pop quiz

- What is a microcontroller
- What is a peripheral
- Where are microcontrollers used?
- What are some pros/cons of using a microcontroller

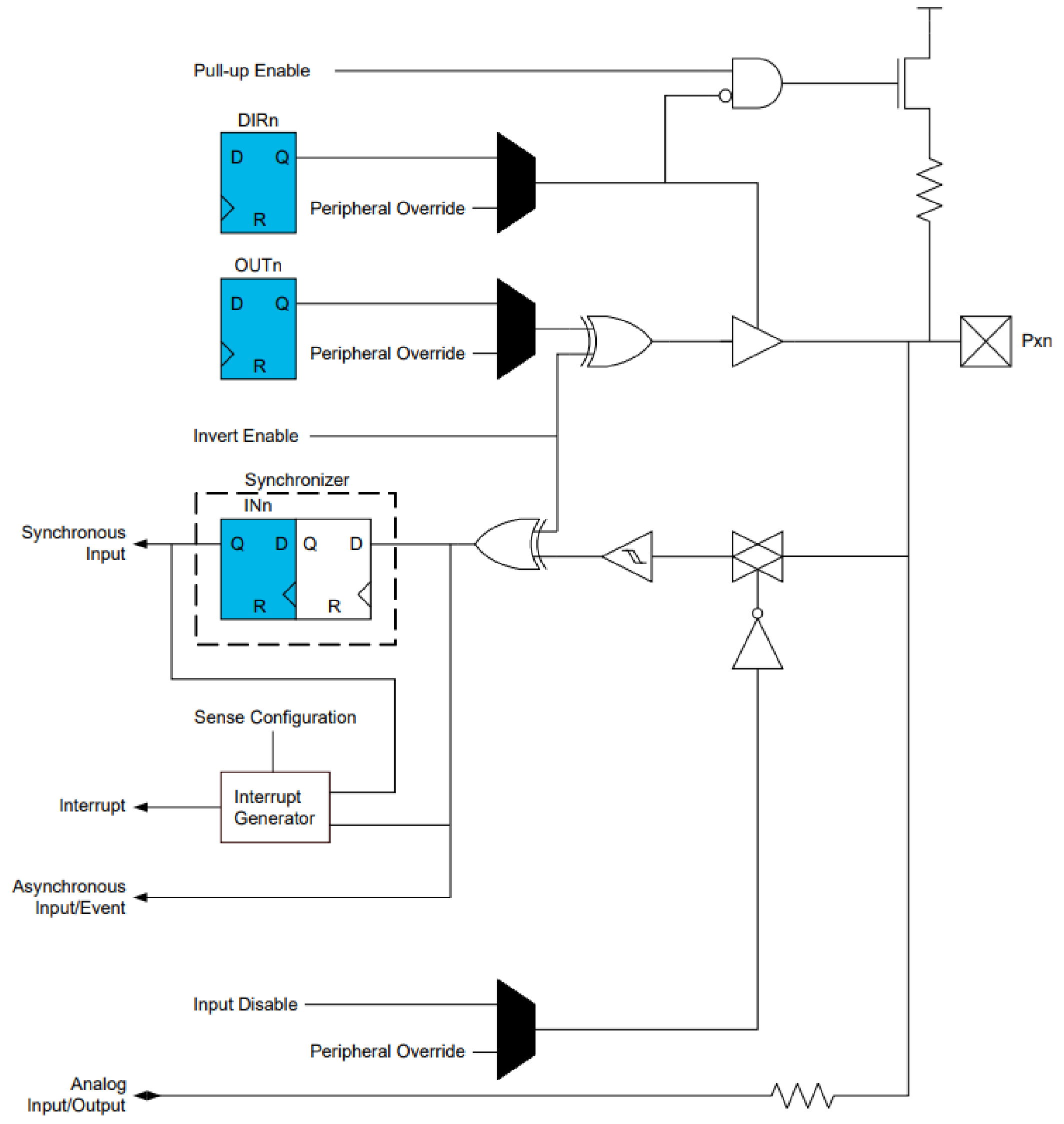
Typical setup



We make a practical circuit in part 6!

General Purpose Input/Output

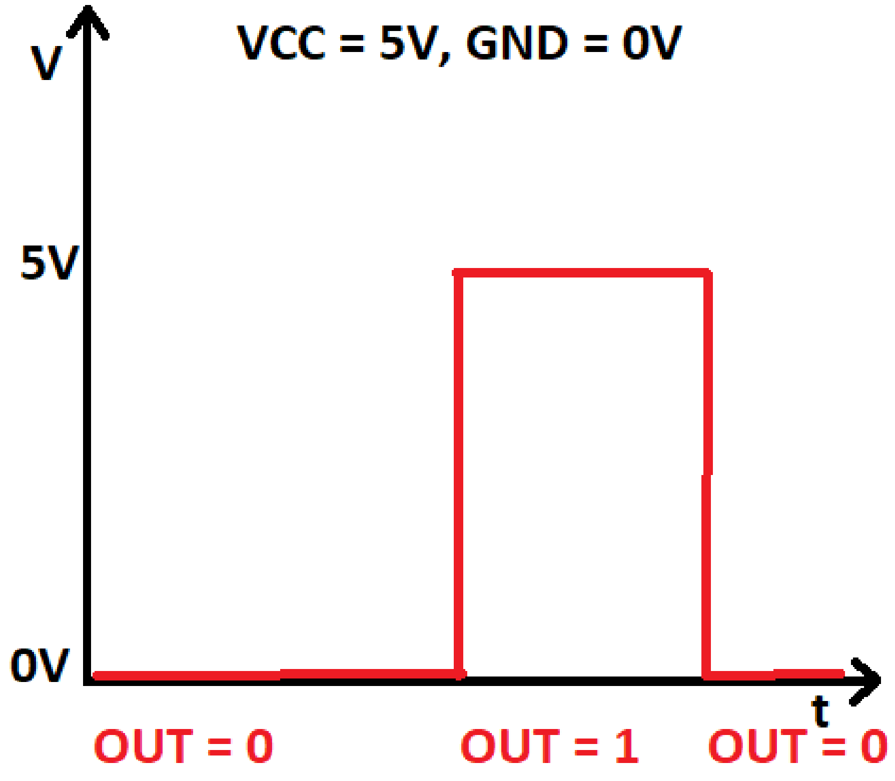
- GPIO for short
- Output
  - Can change the voltage applied to a pin
  - Logical ‘0’ is GND
  - Logical ‘1’ is VDD
- Input
  - Can determine the logical state of an applied voltage
    - Note that a voltage can only be considered as a one or a zero!
- Other peripherals may override the GPIOs



GPIO as Output

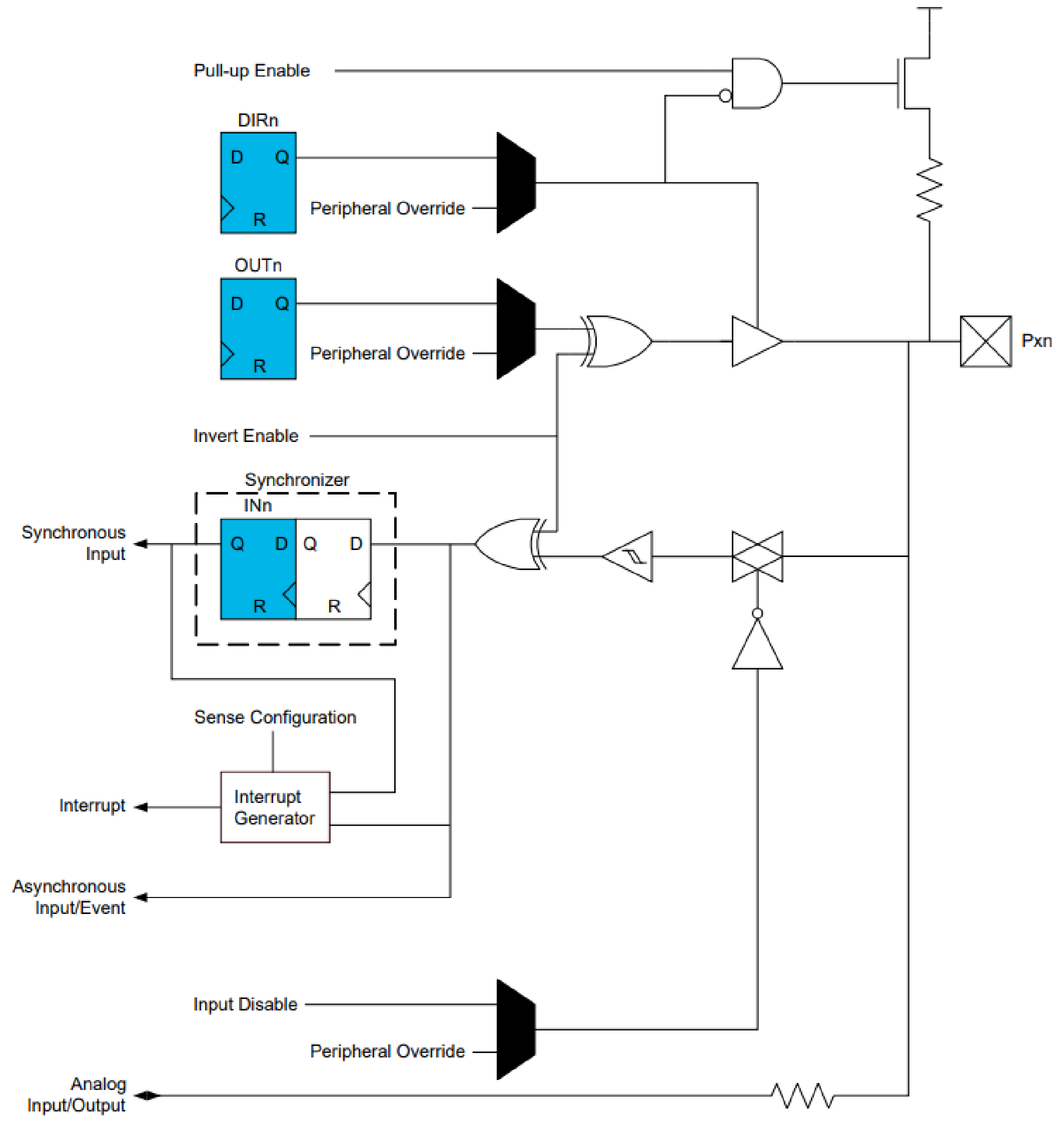
- Direction register must be set to ‘1’
- Output register determines output voltage
    - Logical ‘0’ : 0 Volts
    - Logical ‘1’ : VDD Volts
        - Where VDD is the power supply pin of the microcontroller
- AVR specifics
    - VDD range : 1.8 – 5.5 Volts
    - IO Source/Sink capability : 40 mA (50 mA absolute max)
    - IC Source/Sink capability : 200 mA per VDD or GND

VCC = VDD



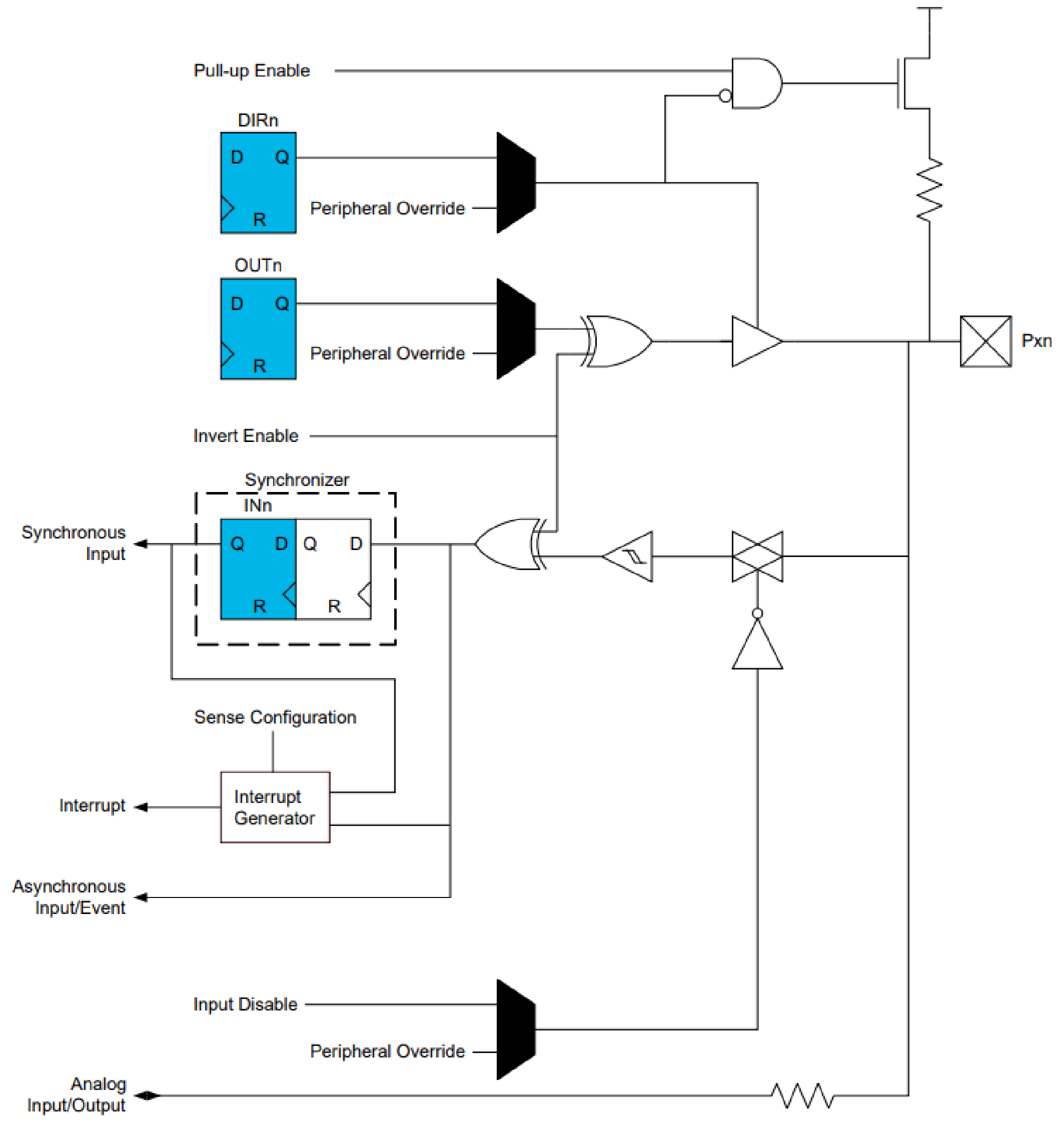
GPIO as Input

- May have internal pull-up resistor
  - Pulls logic to a default state
  - Handy for push buttons!
- Input buffer has hysteresis!
  - Logic ‘0’ : 0.2 * VDD
  - Logic ‘1’ : 0.8 * VDD
- Input buffer may be disabled
  - To save power or to avoid interference when used as an analog input
- Input buffer has a synchronizer
  - To avoid metastability issues
  - An asynchronous input is also available



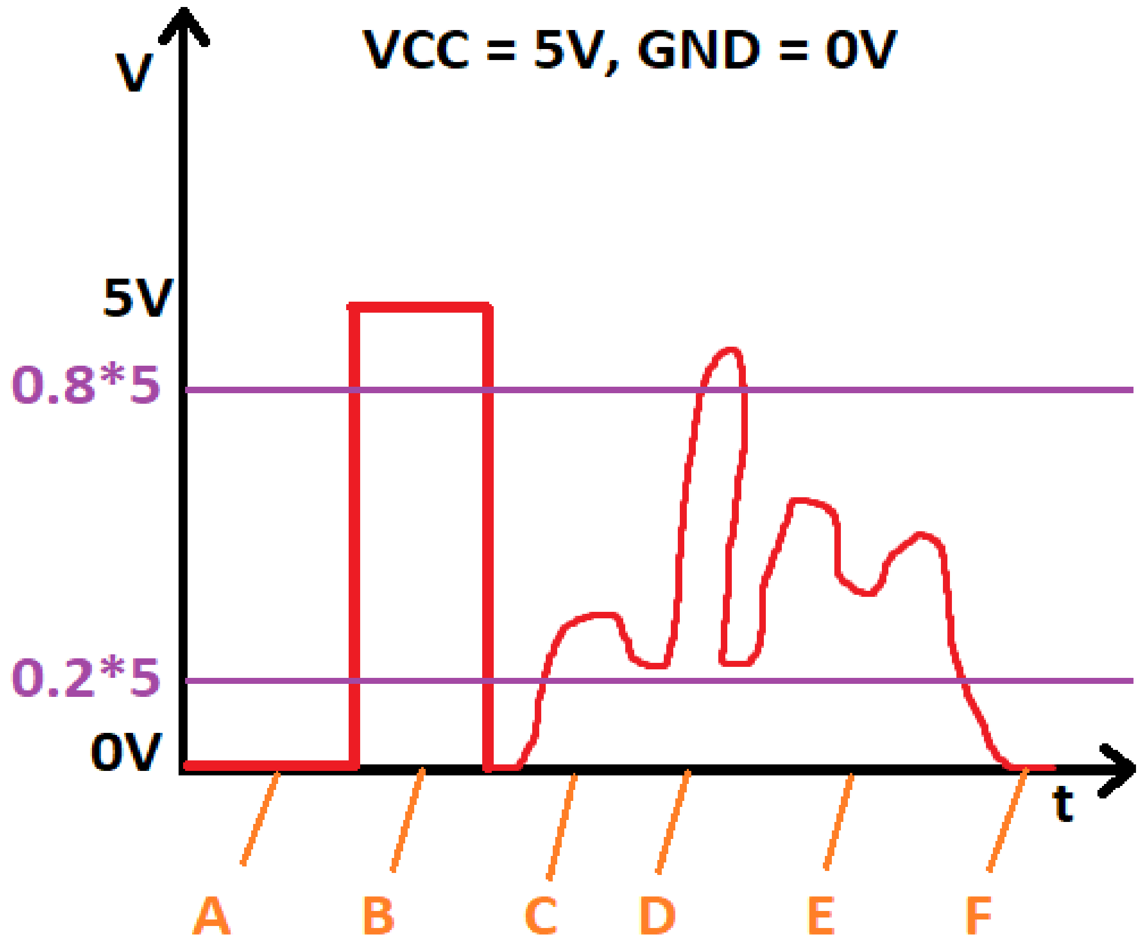
GPIO as Input

- May be connected to internal peripherals
  - Analog
  - Digital
- Can be used to generate interrupts
- Connected to the event system



Pop-Quiz

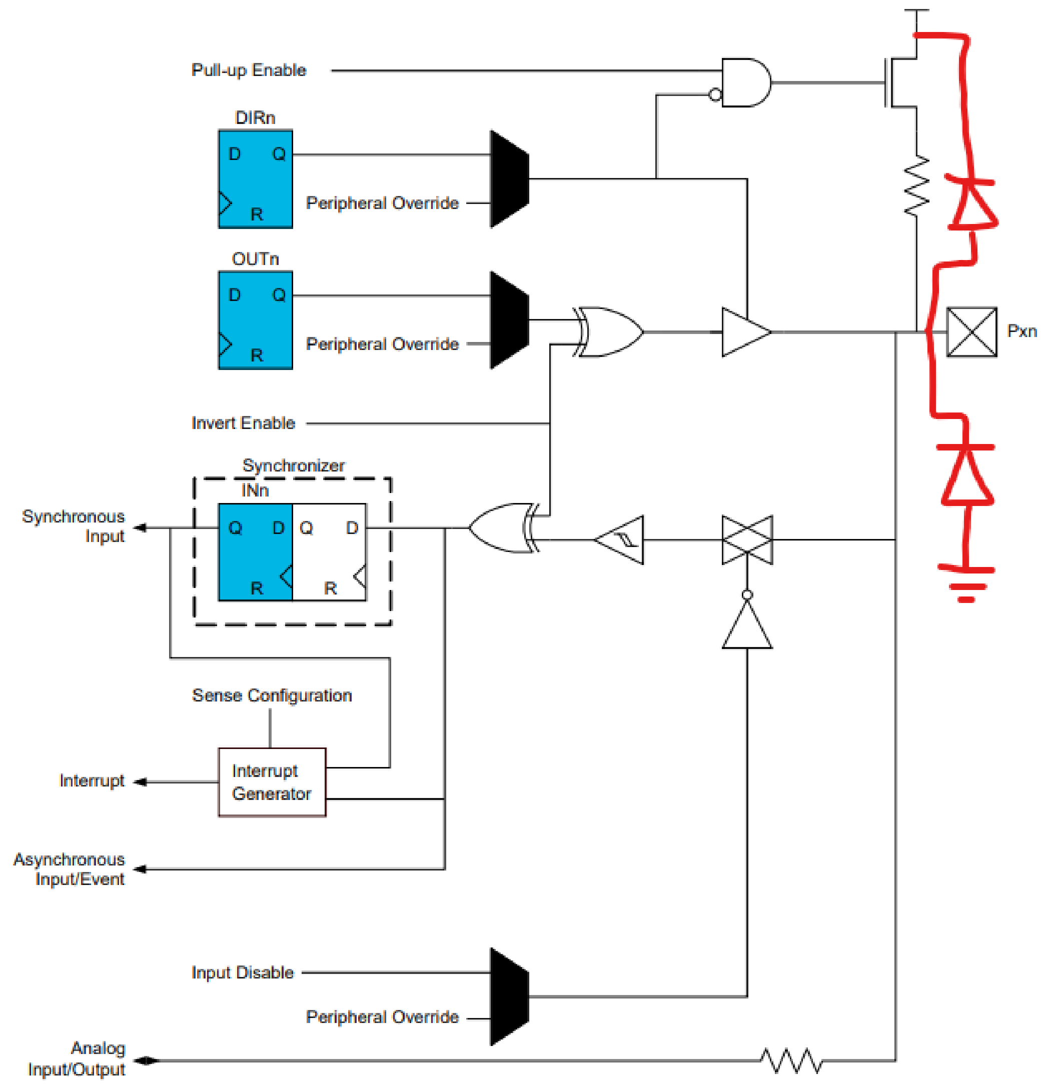
- What is the logic input?
  - A
  - B
  - C
  - D
  - E
  - F
- What is going on in the region C – F?
  - How can we avoid this behavior?



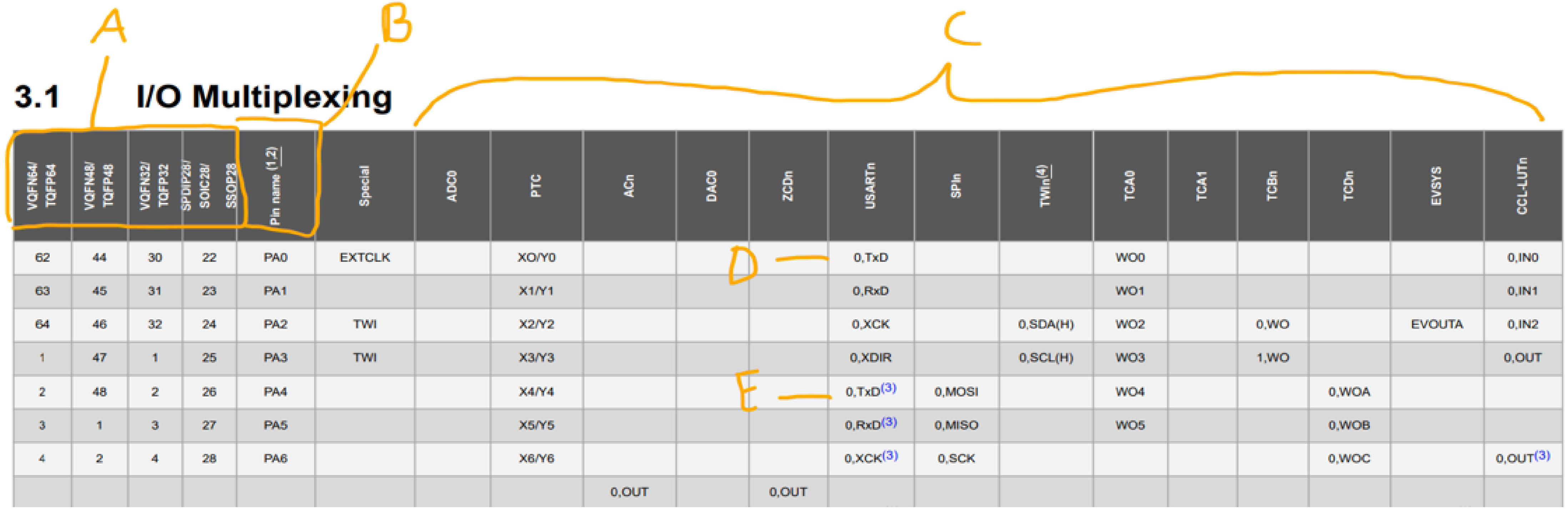
GPIO Hidden Features!

- Not everything is apparent from the GPIO block diagram
  - Information is distributed throughout the datasheet
  - Some information is only available in the Electrical Characteristics section
- Typical hidden features
  - Clamping diodes
  - Maximum IO voltage and current
  - Input impedance
  - Pull-up resistor value

Figure 17-1. PORT Block Diagram



GPIO Multiplexing



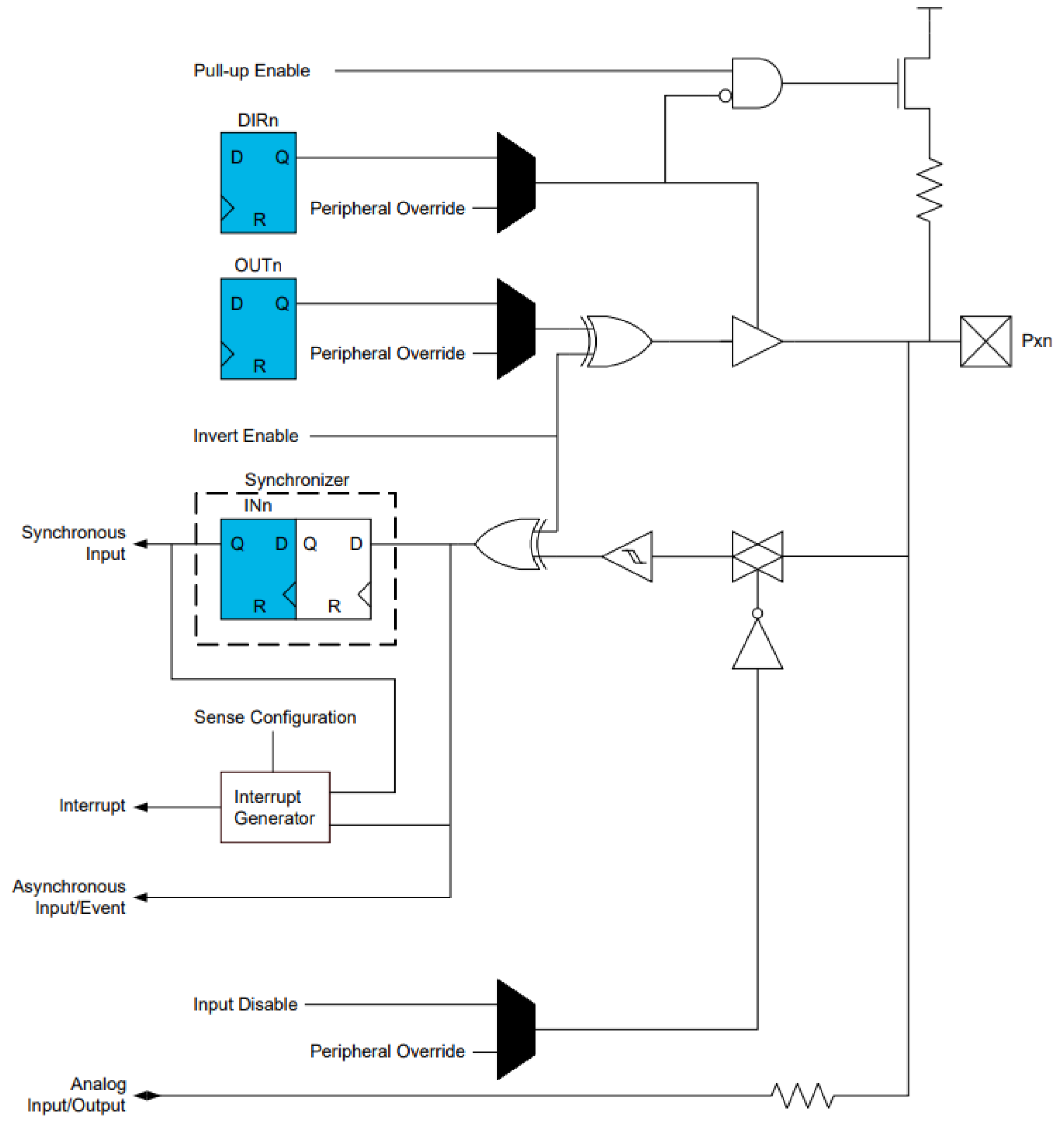
3.1 I/O Multiplexing

VQFN64/ TQFP64	VQFN48/ TQFP48	VQFN32/ TQFP32	SPDIP28/ SOIC28/ SSOP28	Pin name (1,2)	Special	ADC0	PTC	ACn	DAC0	ZCDn	USARTn	SPIn	TWIn(4)	TCA0	TCA1	TCBn	TCDn	EVSYS	CCL-LUTn
62	44	30	22	PA0	EXTCLK		XO/Y0				0,TxD			WO0					0,IN0
63	45	31	23	PA1			X1/Y1				0,RxD			WO1					0,IN1
64	46	32	24	PA2	TWI		X2/Y2				0,XCK		0,SDA(H)	WO2		0,WO		EVOUTA	0,IN2
1	47	1	25	PA3	TWI		X3/Y3				0,XDIR		0,SCL(H)	WO3		1,WO			0,OUT
2	48	2	26	PA4			X4/Y4				0,TxD(3)	0,MOSI		WO4			0,WOA		
3	1	3	27	PA5			X5/Y5				0,RxD(3)	0,MISO		WO5			0,WOB		
4	2	4	28	PA6			X6/Y6				0,XCK(3)	0,SCK					0,WOC		0,OUT(3)
								0,OUT		0,OUT									

- A: Physical pins on the package
- B: Pin name
  - Shared across packages
- C: Peripherals connected to pin
- D: Instance, Function
  - USART0 TxD
- E: Alternative peripheral placement

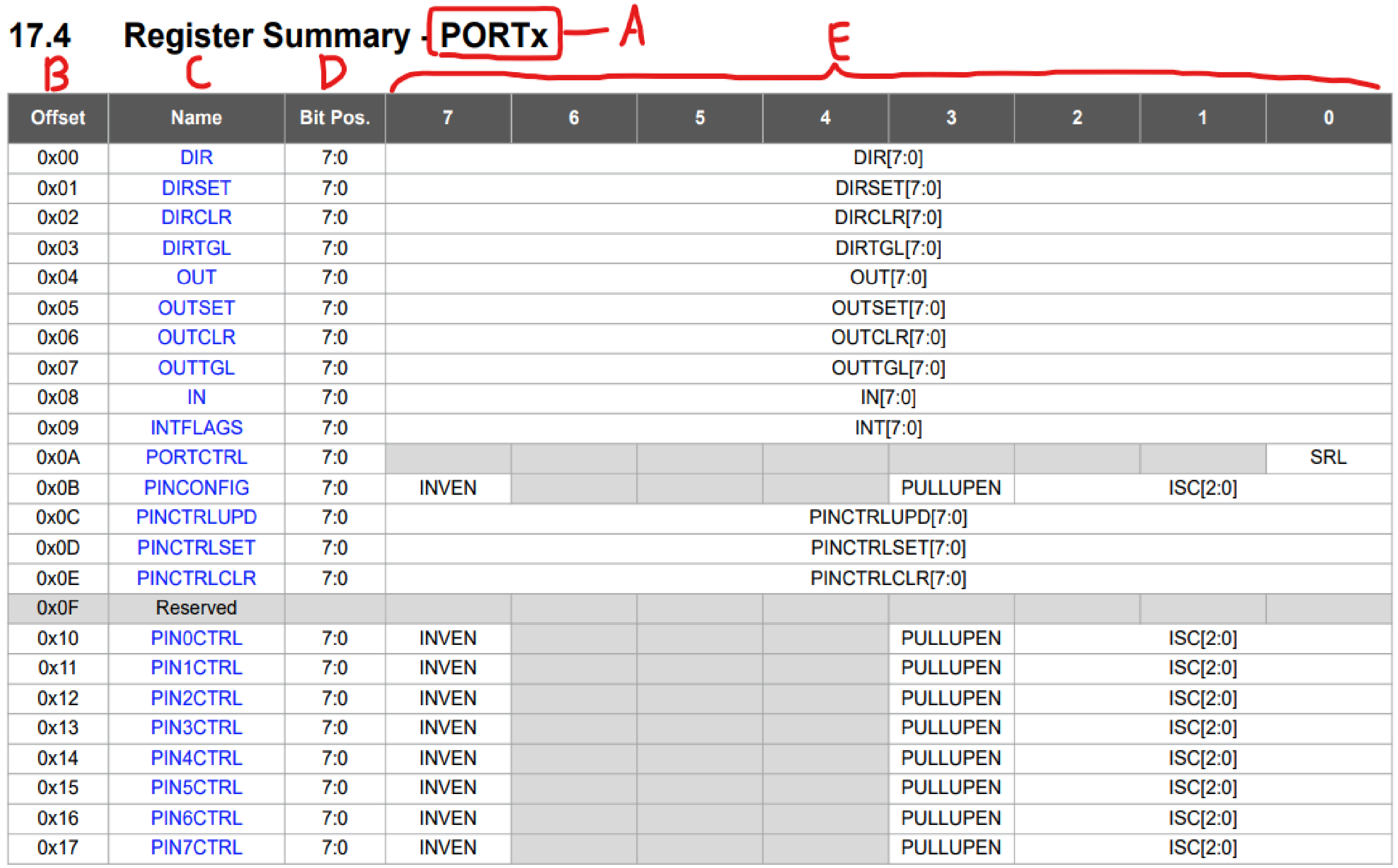
PORT

- A group of up to eight GPIO
- PORT is independent of peripherals connected to GPIO
- Used to control a group of GPIO as input/output
- Registers in blue are actual registers we can read from and write to in software



PORT Register Summary

- A: Peripheral name
  - ‘x’ is instance
  - Pin PA3 (x=A) PORTA pin 3
  - PB7 is PORTB pin 7 etc
- B: Offset from PORTx in memory
- C: Register name
- D: Register size [7:0] = 8 bit = 1 Byte
- E: What the different bits in a register represent

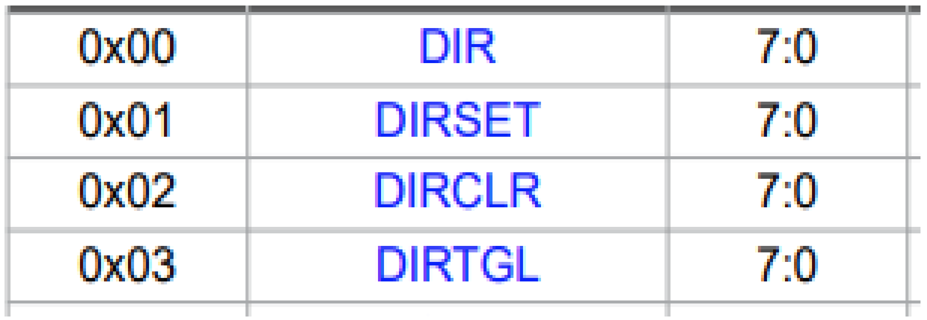


17.4 Register Summary - $\boxed{\text{PORTx}}$

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	DIR	7:0	DIR[7:0]							
0x01	DIRSET	7:0	DIRSET[7:0]							
0x02	DIRCLR	7:0	DIRCLR[7:0]							
0x03	DIRTGL	7:0	DIRTGL[7:0]							
0x04	OUT	7:0	OUT[7:0]							
0x05	OUTSET	7:0	OUTSET[7:0]							
0x06	OUTCLR	7:0	OUTCLR[7:0]							
0x07	OUTTGL	7:0	OUTTGL[7:0]							
0x08	IN	7:0	IN[7:0]							
0x09	INTFLAGS	7:0	INT[7:0]							
0x0A	PORTCTRL	7:0								SRL
0x0B	PINCONFIG	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x0C	PINCTRLUPD	7:0	PINCTRLUPD[7:0]							
0x0D	PINCTRLSET	7:0	PINCTRLSET[7:0]							
0x0E	PINCTRLCLR	7:0	PINCTRLCLR[7:0]							
0x0F	Reserved									
0x10	PIN0CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x11	PIN1CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x12	PIN2CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x13	PIN3CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x14	PIN4CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x15	PIN5CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x16	PIN6CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		
0x17	PIN7CTRL	7:0	INVEN				PULLUPEN	ISC[2:0]		

PORT Direction

- DIR register determines the direction for each GPIO in a PORT group
- DIRSET, DIRCLR, DIRTGL are additional registers that can be used to override the value in DIR
  - DIR can also be read/written directly
  - These additional registers apply bitwise logic to their input and place the result in the DIR register
- We will come back to this next time

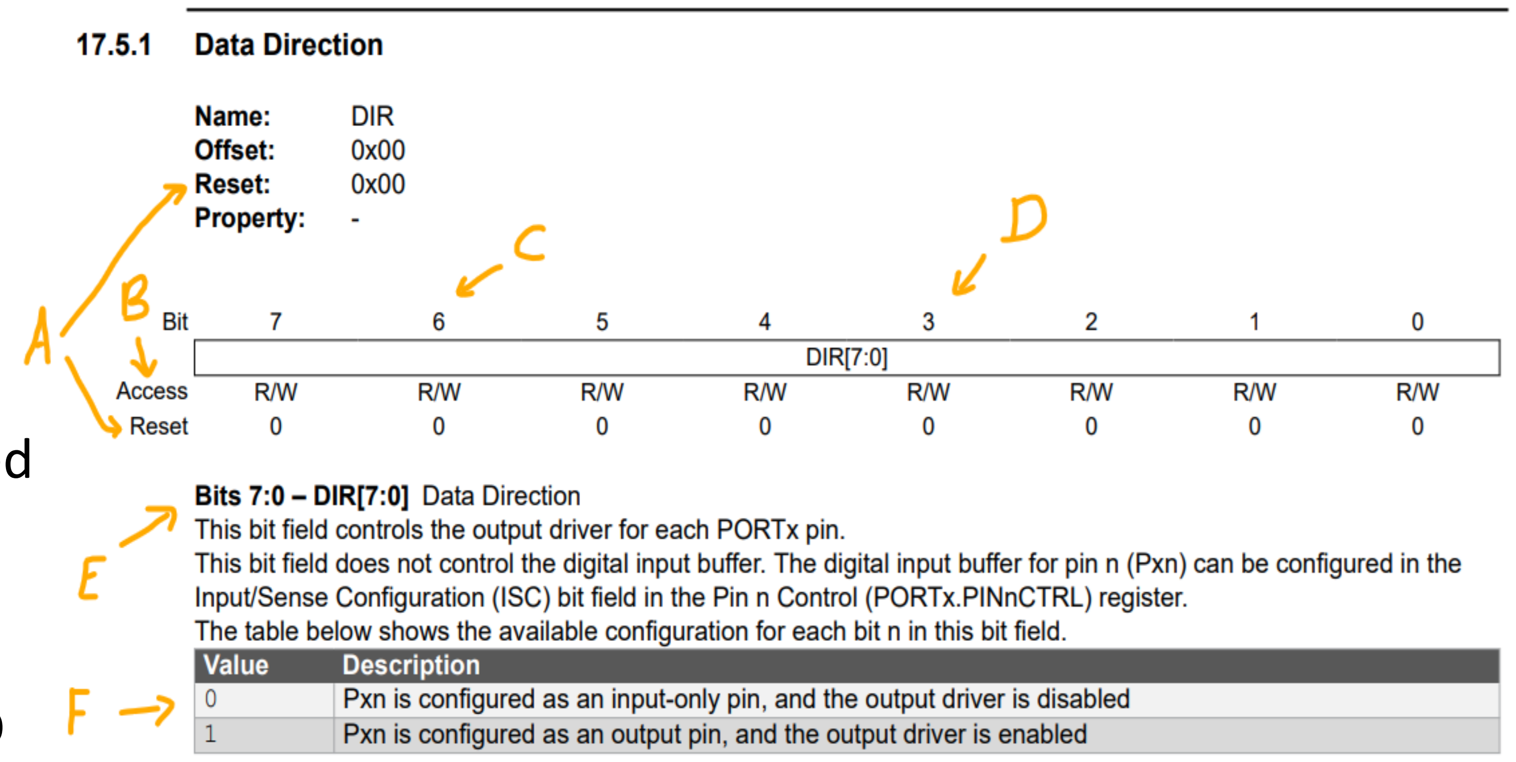


0x00	DIR	7:0
0x01	DIRSET	7:0
0x02	DIRCLR	7:0
0x03	DIRTGL	7:0

- DIRSET
  - DIR |= <value>
- DIRCLR
  - DIR &= ~<value>
- DIRTGL
  - DIR ^= <value>

DIR Register

- A: Register value at reset
- B: Access
  - Read, Write, Read/Write
- C: bit 6 is pin 6 for a given port
  - PA6 direction is represented by this bit if the PORTx peripheral is PORTA
- D: What about this?
- E: Description of bit group
- D: What different values represent
  - Here values are bitwise



17.5.1 **Data Direction**

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

Bit	7	6	5	4	3	2	1	0
	DIR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – DIR[7:0] Data Direction
This bit field controls the output driver for each PORTx pin.
This bit field does not control the digital input buffer. The digital input buffer for pin n (Pxn) can be configured in the Input/Sense Configuration (ISC) bit field in the Pin n Control (PORTx.PINnCTRL) register.
The table below shows the available configuration for each bit n in this bit field.

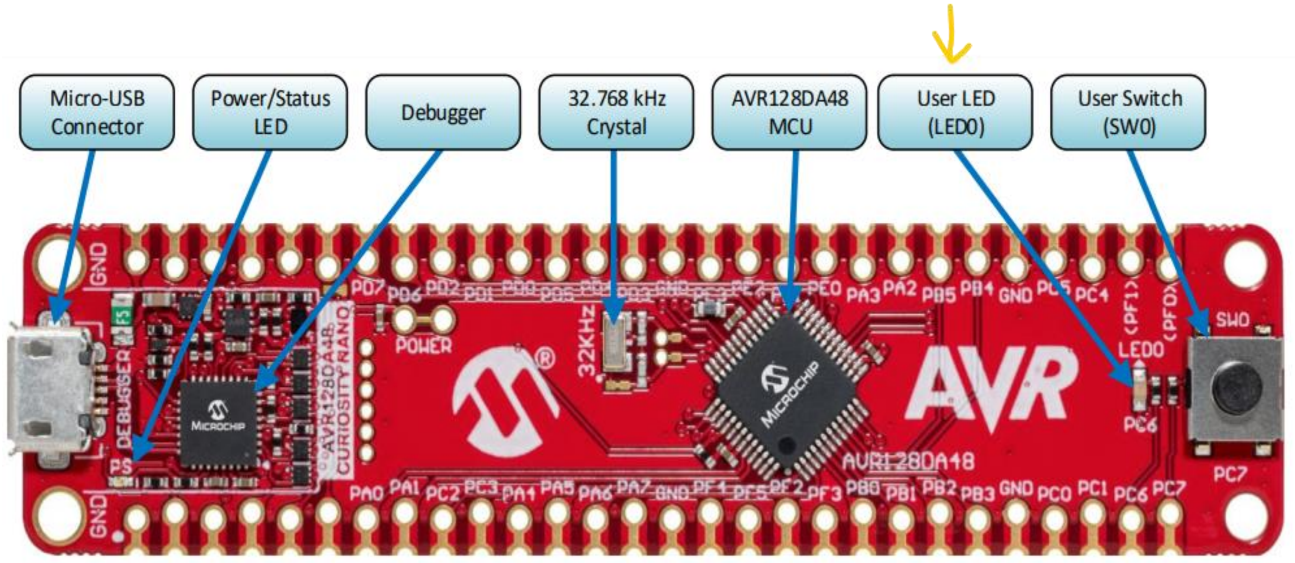
Value	Description
0	Pxn is configured as an input-only pin, and the output driver is disabled
1	Pxn is configured as an output pin, and the output driver is enabled

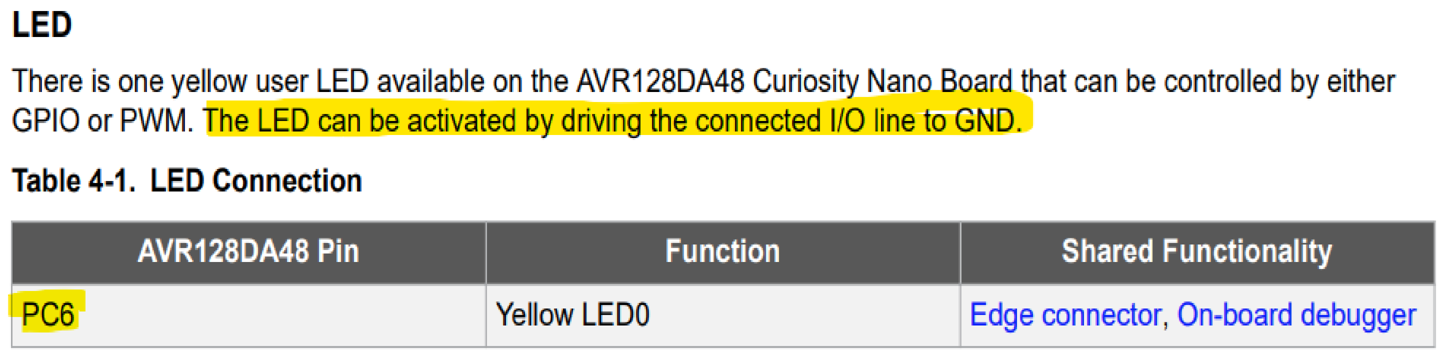
PORT Registers

- OUT Register
  - Determines the output value of a pin
  - But only when direction is ‘1’ – Output
  - Can be manipulated directly or through OUTSET, OUTCLR, OUTTGL
- IN Register
  - Read only register
  - What logic level is at a pin
  - Always available, even if DIR = ‘1’
    - Must commonly used when DIR = ‘0’
- INxCTRL
  - Additional settings per pin in a PORT group
  - Pull-up resistor enable
  - Interrupt control
  - Invert logic

Task: LED Blink

- Task: Toggle LED





LED

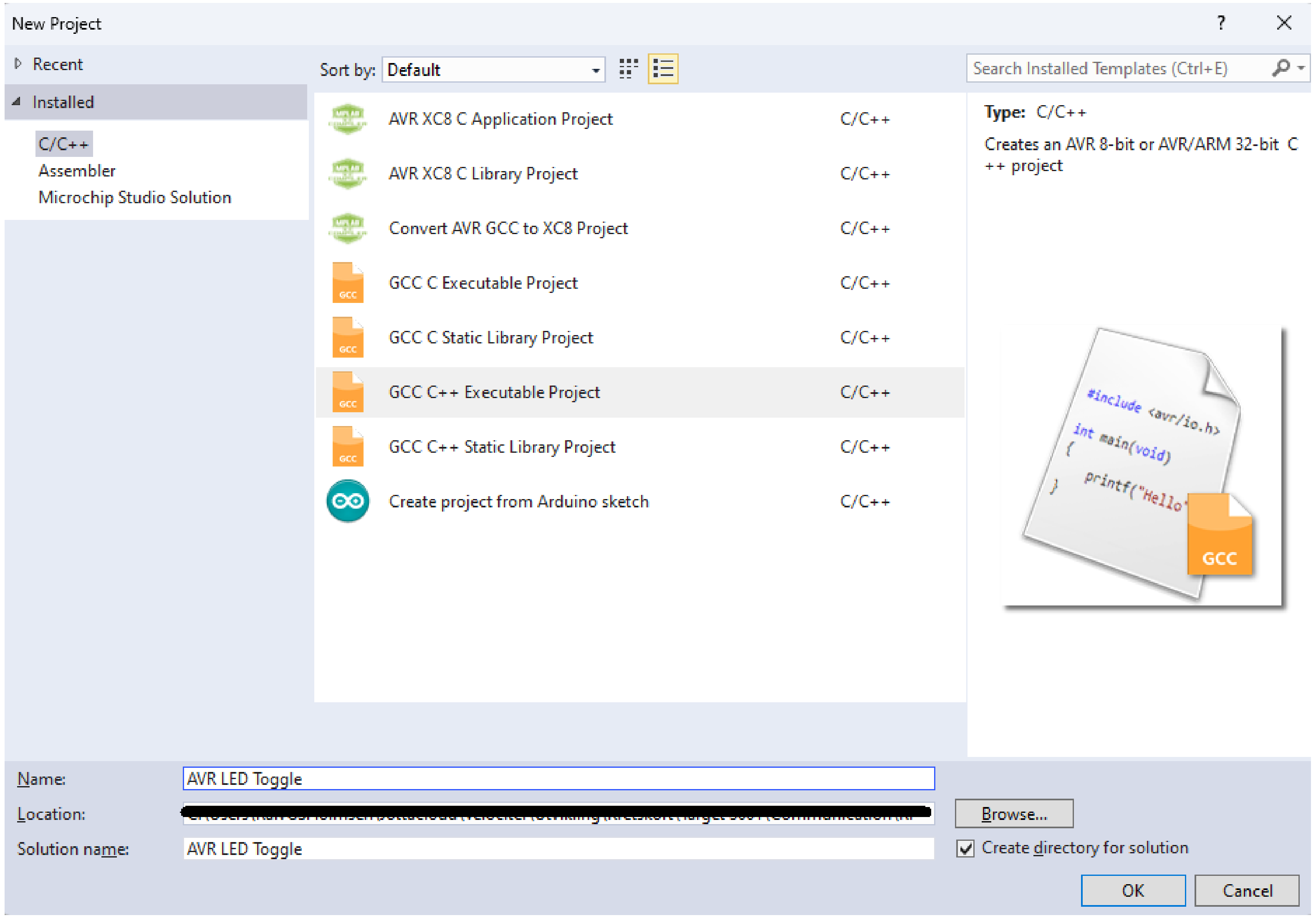
There is one yellow user LED available on the AVR128DA48 Curiosity Nano Board that can be controlled by either GPIO or PWM. The LED can be activated by driving the connected I/O line to GND.

Table 4-1. LED Connection

AVR128DA48 Pin	Function	Shared Functionality
PC6	Yellow LED0	Edge connector, On-board debugger

Task: LED Blink

- Create a new project
  - File / New Project
- Select GCC C++ Executable Project
  - Name the project "AVR LED Toggle"



New Project

Recent
Installed
C/C++
Assembler
Microchip Studio Solution

Sort by: Default

Search Installed Templates (Ctrl+E)

AVR XC8 C Application Project	C/C++
AVR XC8 C Library Project	C/C++
Convert AVR GCC to XC8 Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++

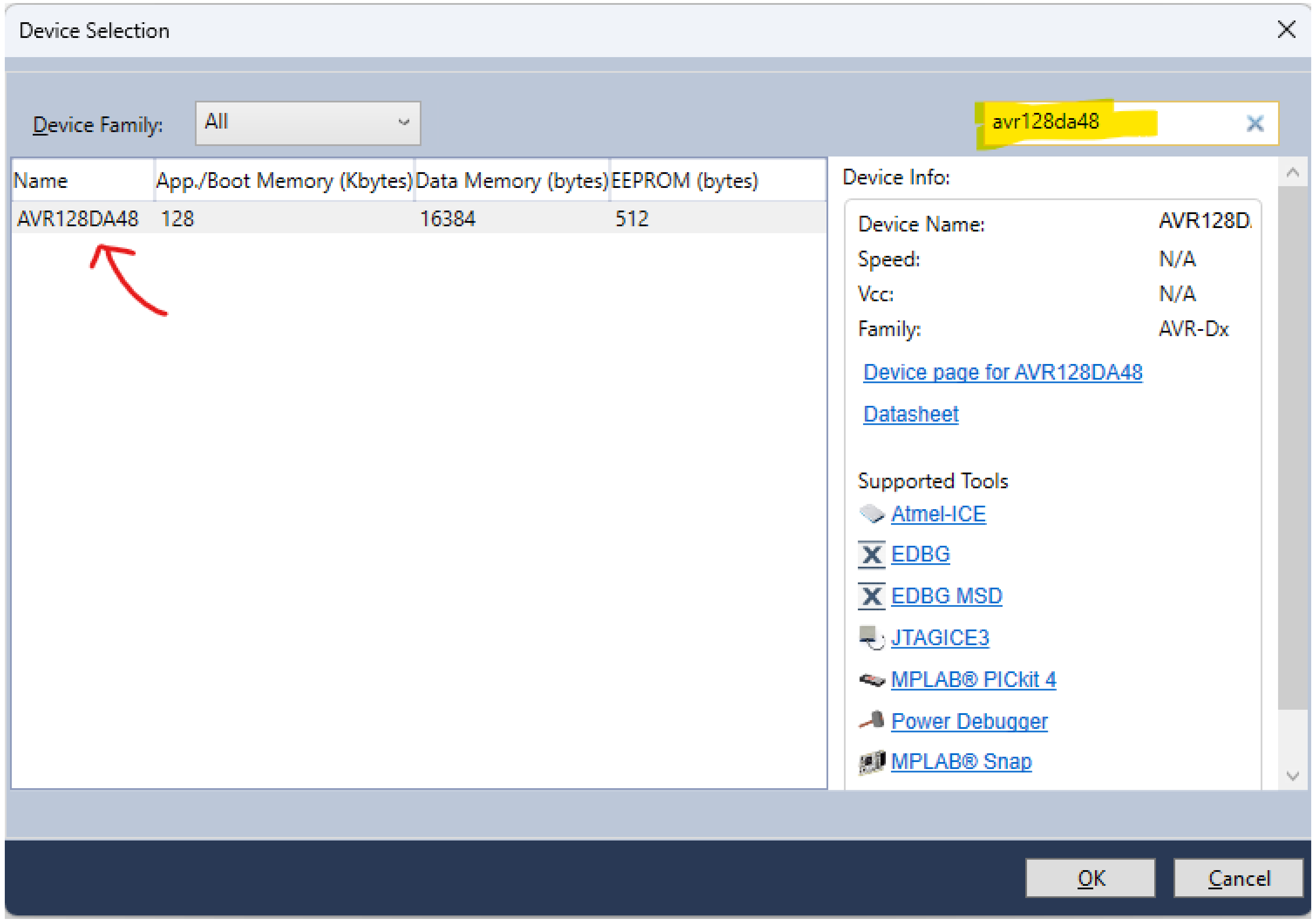
Creates an AVR 8-bit or AVR/ARM 32-bit C ++ project

Name: AVR LED Toggle
Location: [redacted]
Solution name: AVR LED Toggle

Browse...
☑ Create directory for solution
OK
Cancel

Task: LED Blink

- Search for and select AVR128DA48


Device Selection

Device Family: All

avr128da48

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes)
AVR128DA48	128	16384	512

Device Info:

Device Name:	AVR128D
Speed:	N/A
Vcc:	N/A
Family:	AVR-Dx

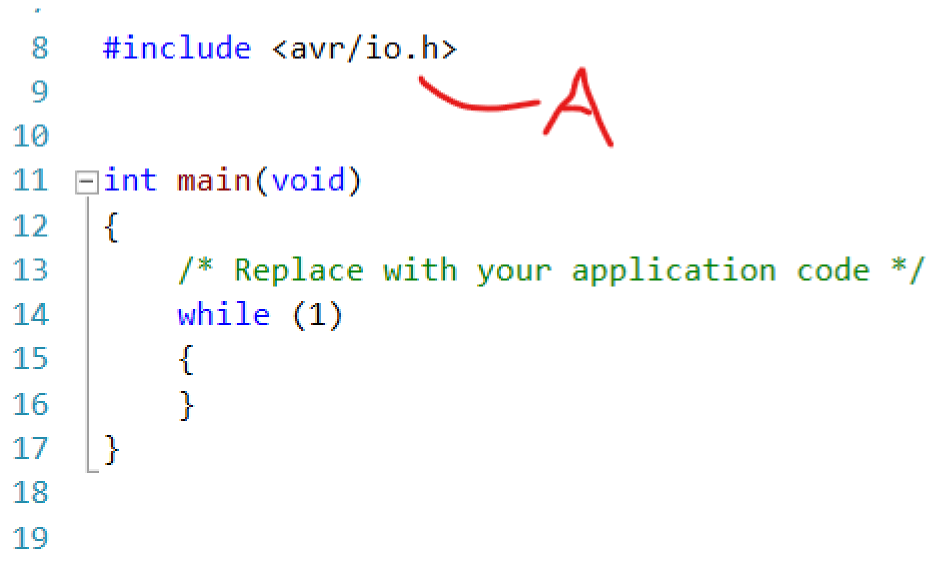
Device page for AVR128DA48

Datasheet

Supported Tools
- Atmel-ICE
- EDBG
- EDBG MSD
- JTAGICE3
- MPLAB® PICkit 4
- Power Debugger
- MPLAB® Snap

Task: LED Blink

- A: Make sure avr/io.h is included
  - This library defines the addresses for the data space
- The main function is run when the microcontroller boots
- Everything within the while loop runs forever
- Peripherals are typically configured before the while loop
  - This is typically referred to as the setup phase



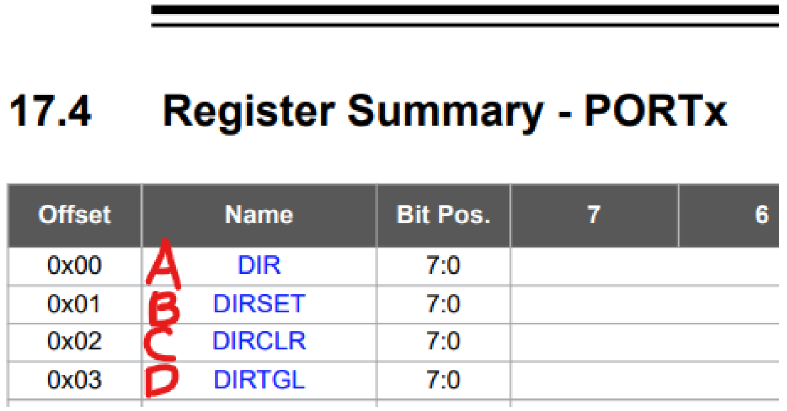
```
#include <avr/io.h>


int main(void)
{
    /* Replace with your application code */
    while (1)
    {
    }
}
```

Task: LED Blink

- avr/io.h uses a struct system for addressing different registers in data space
- Typical structure convention
  - PERIPHERAL.REGISTER_NAME
  - Example: PORTC.DIR = 64
    - Sets PC6 as output
- The io library also includes easy to read bitmasks
  - 64 = 0b01000000 = 0x40 = (1 << 6) = PIN6_bm
  - Always use bitmasks as this makes your code more readable!

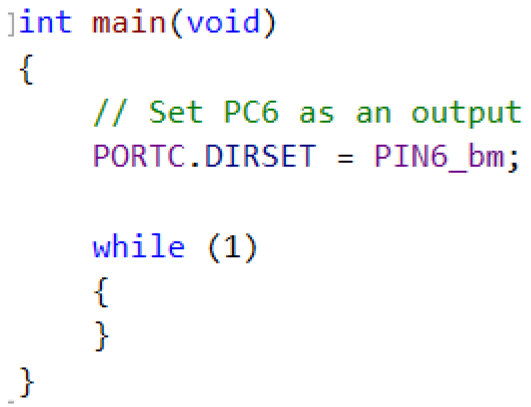
Task: LED Blink

- Set the direction of pin PC6 as an output
  - The 6th bit in the DIR register for PORTC must be set to ‘1’
- A: DIR
  - The DIRECTION register
  - Holds all the direction bits
- B: DIRSET
  - Writing a ‘1’ to any bit here sets the corresponding bit in the DIR register tp ‘1’ regardless of its previous value
  - In other words “SET as output”
- C: DIRCLR
  - Same as DIRSET, but “writes” a ‘0’ (clear)
- A ‘0’ in the dir register makes the pin an input
- D: DIRTGL
  - Inverts whatever the previous value was
- A ‘0’ becomes ‘1’, and a ‘1’ becomes a ‘0’



17.4 Register Summary - PORTx

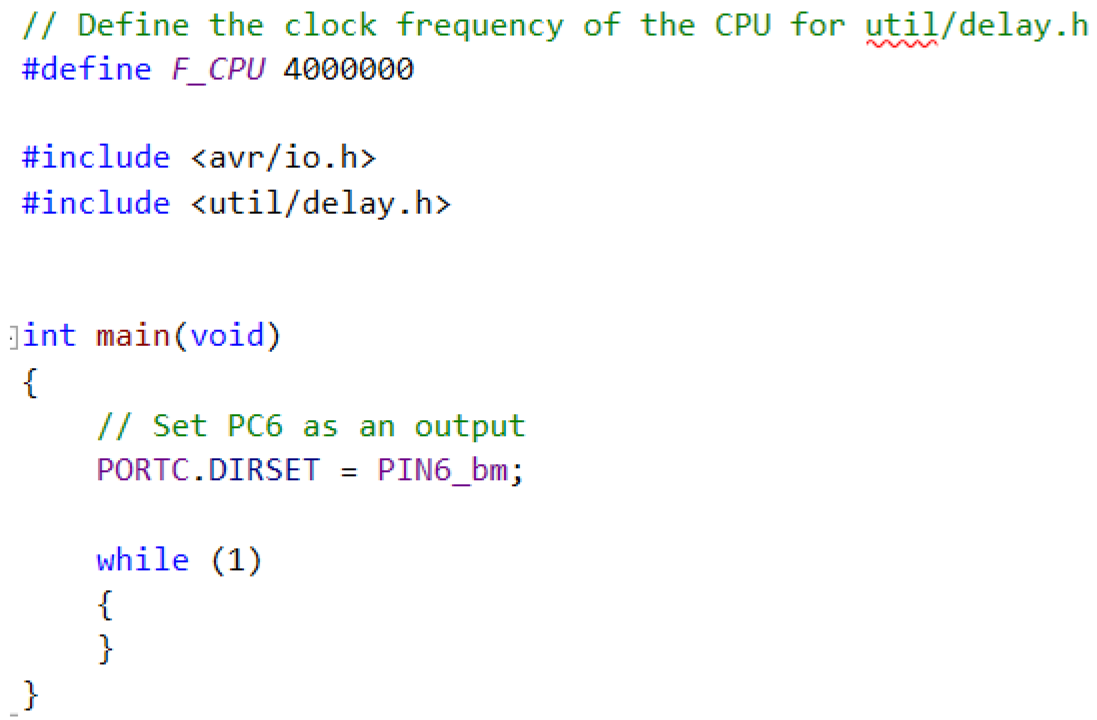
Offset	Name	Bit Pos.	7	6
0x00	A DIR	7:0		
0x01	B DIRSET	7:0		
0x02	C DIRCLR	7:0		
0x03	D DIRTGL	7:0		



```
int main(void)
{
    // Set PC6 as an output
    PORTC.DIRSET = PIN6_bm;

    while (1)
    {
    }
}
```

Task: LED Blink

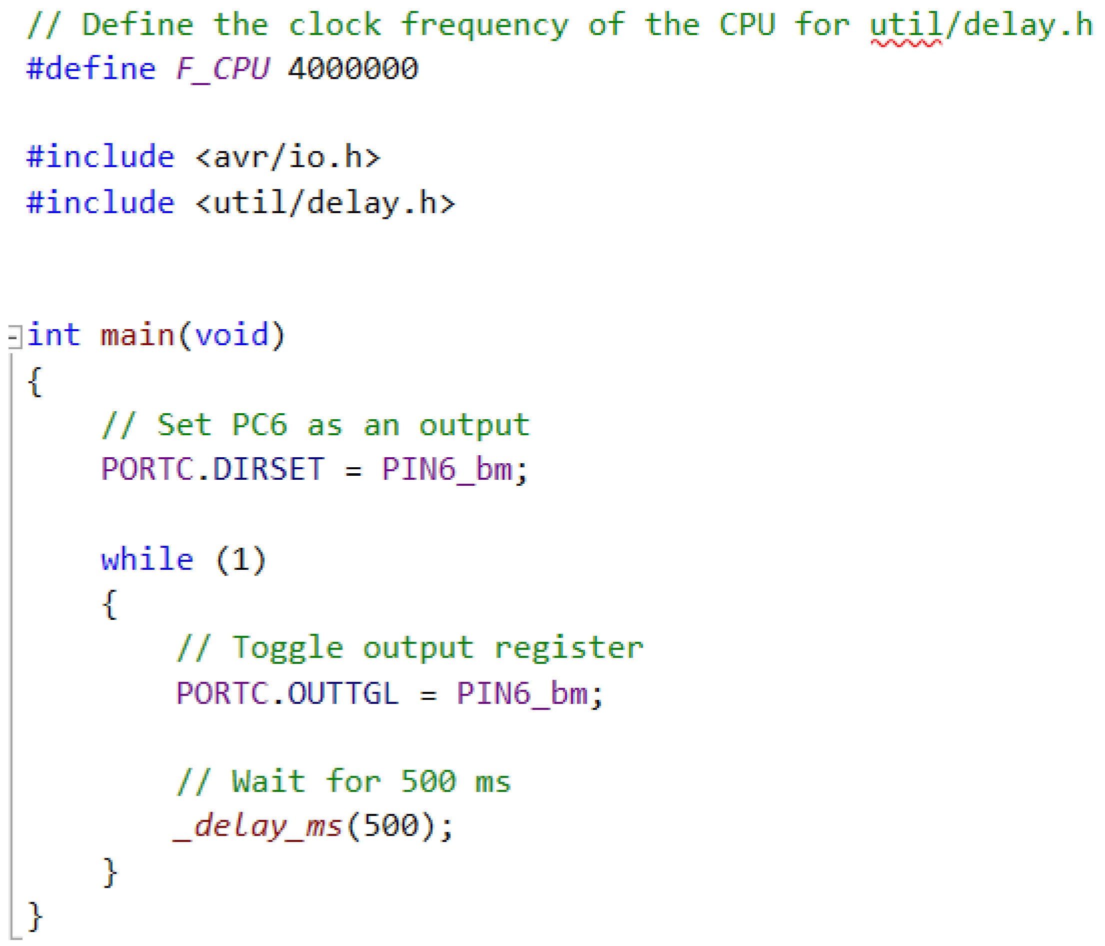
- Define F_CPU as 4 MHz
  - #define F_CPU 4000000
  - Used by delay.h
- Include delay.h
  - #include <util/delay.h>
  - Grants delay functions
    - _delay_ms()
    - _delay_us()



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

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


int main(void)
{
    // Set PC6 as an output
    PORTC.DIRSET = PIN6_bm;

    while (1)
    {
    }
}
```

Task: LED Blink

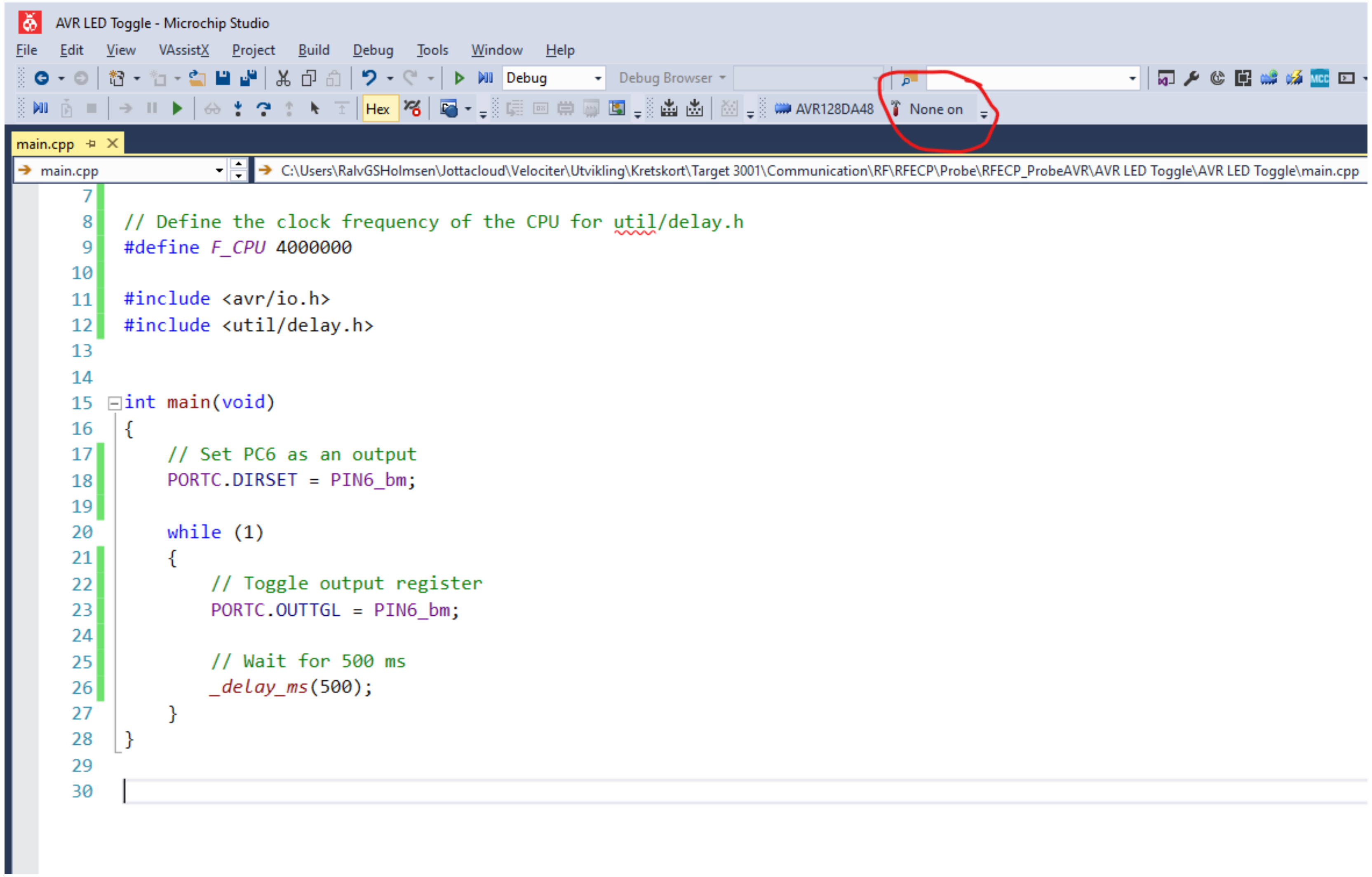
- Toggle the value in the OUT register every 500 ms
- The OUTTGL register does the arithmetic for us
  - We only need to provide a bitmask
- A blocking delay function is used



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

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


int main(void)
{
    // Set PC6 as an output
    PORTC.DIRSET = PIN6_bm;

    while (1)
    {
        // Toggle output register
        PORTC.OUTTGL = PIN6_bm;

        // Wait for 500 ms
        _delay_ms(500);
    }
}
```

Task: LED Blink

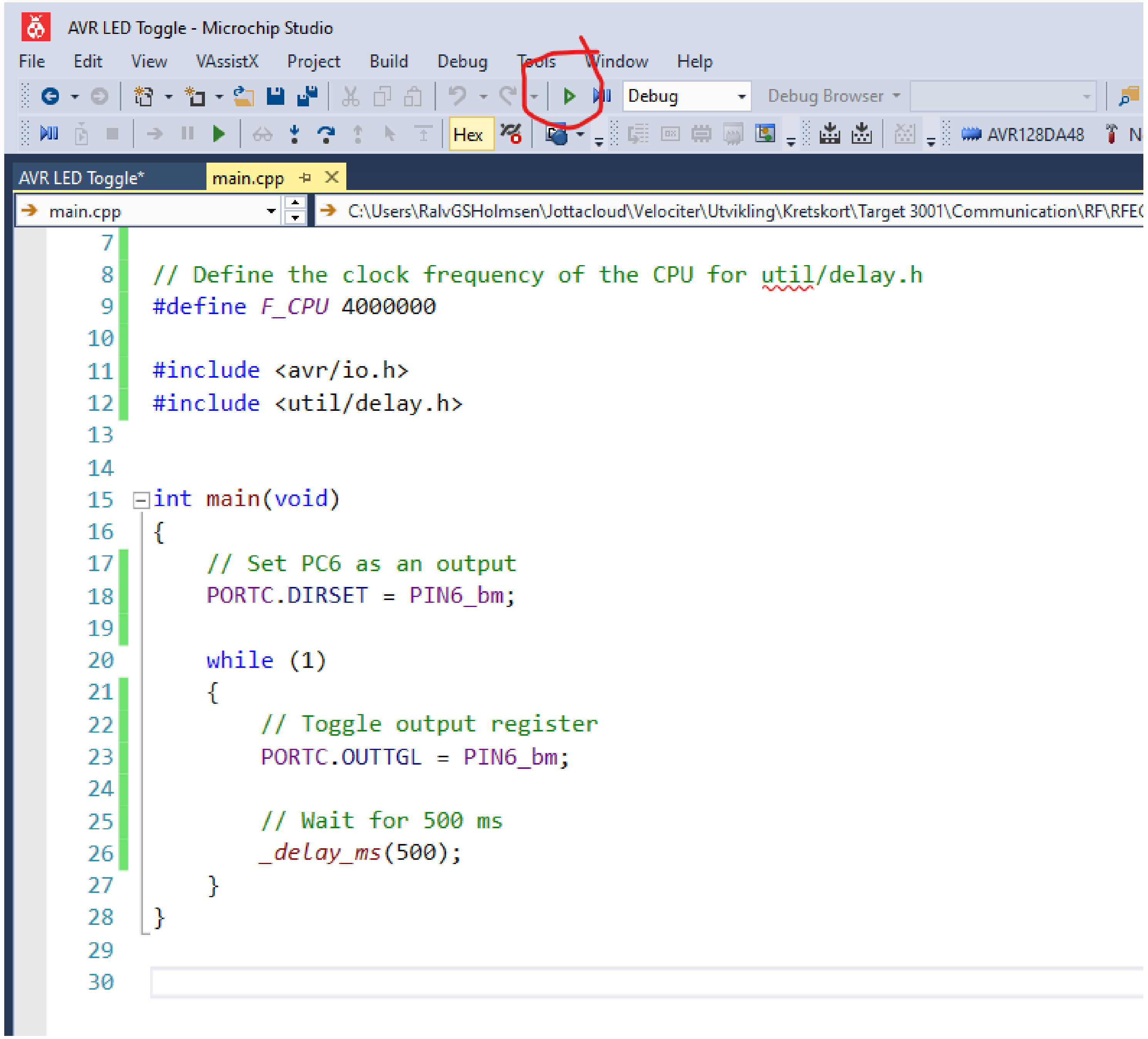
- Select the programming tool
- From the drop-down menu select AVR128DA48 Curiosity


AVR LED Toggle - Microchip Studio
File Edit View VAssistX Project Build Debug Tools Window Help
Debug
Debug Browser
Hex
AVR128DA48
None on
main.cpp
C:\Users\RalvGSHolmsen\Jottacloud\Velociter\Utvikling\Kretskort\Target 3001\Communication\RF\RFECP\Probe\RFECP_ProbeAVR\AVR LED Toggle\AVR LED Toggle\main.cpp

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

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


int main(void)
{
    // Set PC6 as an output
    PORTC.DIRSET = PIN6_bm;

    while (1)
    {
        // Toggle output register
        PORTC.OUTTGL = PIN6_bm;

        // Wait for 500 ms
        _delay_ms(500);
    }
}

```

Task: LED Blink

- Program the device!
- Watch the LED toggle


AVR LED Toggle - Microchip Studio
File
Edit
View
VAssistX
Project
Build
Debug
Tools
Window
Help
Debug
Debug Browser
Hex
AVR128DA48
AVR LED Toggle*
main.cpp
main.cpp
C:\Users\RalvGSHolmsen\Jottacloud\Velociter\Utvikling\Kretskort\Target 3001\Communication\RF\RFE

```

// Define the clock frequency of the CPU for util/delay.h
#define F_CPU 4000000

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


int main(void)
{
    // Set PC6 as an output
    PORTC.DIRSET = PIN6_bm;

    while (1)
    {
        // Toggle output register
        PORTC.OUTTGL = PIN6_bm;

        // Wait for 500 ms
        _delay_ms(500);
    }
}


```

Done for today

- Next time
  - Boolean and bitwise logic
  - Number representations
  - How to manipulate registers
  - Learn more about the GPIO and how to use PORT
- Questions?