Introduction to Microcontrollers

UART and ADC

Goals

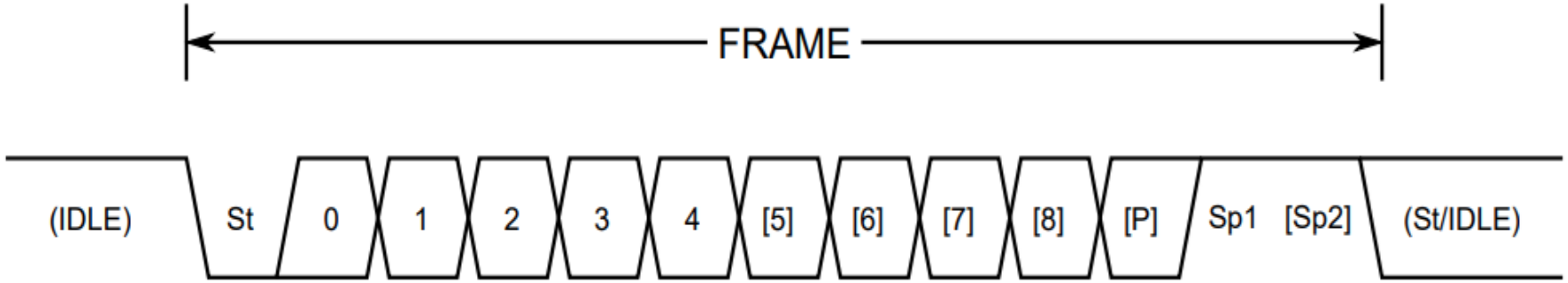
- Know what an ADC is, and some basic topologies
- Know what UART is and what it is used for
- Learn to use ADC peripherals with AVRs
- Learn to use UART peripherals with AVRs

UART

- In modern AVRs referred to as USART
  - Universal Synchronous/Asynchronous Receiver/Transmitter
    - Asynchronous mode is the most common implementation method
  - We will still refer to it as UART, as this is the common name
  - RS232
- Fully duplex
  - One transmit line
  - One receive line
- Can be half duplex when used
  - In one wire mode
    - TX and RX on one physical pin
  - RS485 half duplex mode
    - Can be fully duplex
    - Noise immunity means long wires can be used (up to 1km)
    - Requires external circuitry

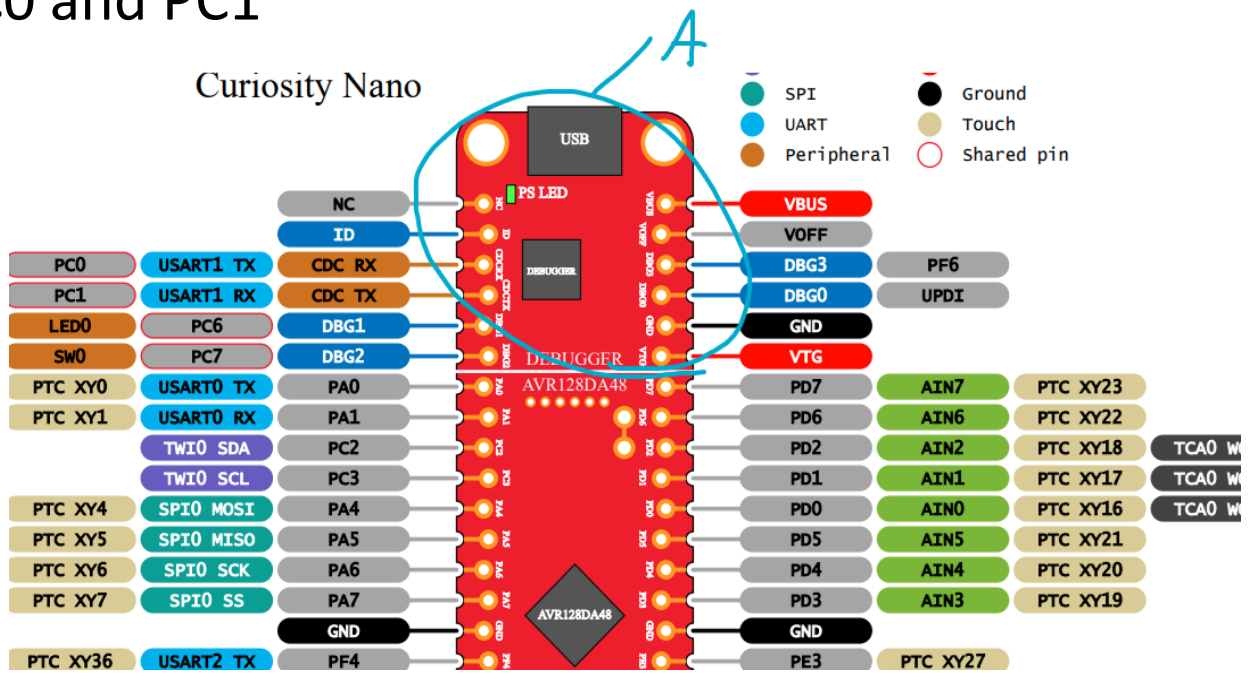
UART

- Common interface
  - Between microcontrollers
  - Debug interface for connecting to computers
    - Usually through a RS232 to USB converter
- Frames
  - 1 start bit
  - 5-9 data bits (8 data bits is most common)
  - None, Even, or Odd parity (None common)
  - 1-2 stop bit(s) (1 is common)



UART on AVR128DA48 Curiosity Nano

- Has on-board debugger and programmer
- Has on-board UART to USB converter
  - Connected to USART1 through PC0 and PC1
- A: On-board debugger



USART Peripheral: Basic Implementation

24.4 Register Summary - USART

Offset	Name	Bit Pos.								
0x00	RXDATAL	7:0	DATA[7:0]							
0x01	RXDATAH	7:0	RXCIF	BUFOVF				FERR	PERR	DATA[8]
0x02	TXDATAL	7:0	DATA[7:0]							
0x03	TXDATAH	7:0								DATA[8]
0x04	STATUS	7:0	RXCIF	TXCIF	DREIF	RXSIF	ISFIF		BDF	WFB
0x05	CTRLA	7:0	RXCIE	TXCIE	DREIE	RXSIE	LBME	ABEIE	RS-485[1:0]	
0x06	CTRLB	7:0	RXEN	TXEN		SFDEN	ODME	RXMODE[1:0]		MPCM
0x07	CTRLC	7:0	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]		
0x07	CTRLC	7:0	CMODE[1:0]					UDORD	UCPHA	
0x08	BAUD	7:0	BAUD[7:0]							
		15:8	BAUD[15:8]							
0x0A	Reserved									
0x0B	DBGCTRL	7:0								DBGRUN
0x0C	EVCTRL	7:0								IREI
0x0D	TXPLCTRL	7:0	TXPL[7:0]							
0x0E	RXPLCTRL	7:0		RXPL[6:0]						

USART Peripheral: Basic Implementation

- Only used in transmit mode
  - Receiving is a bit more complicated
    - An example will be provided on github after part 4 – Timers and Interrupts
- Powerful!
  - We can send data from our microcontroller to our computer
  - Very useful for debugging!
- Easy
  - We only need to consider four registers as the other registers are correct by default, or because they are not needed

USART Peripheral: Setup

- Only CTRLB and BAUD registers
  - CTRLB is a control register which is only used to enable the transmitter hardware
    - USART_TXEN_bm must be set to ‘1’
  - BAUD registers are two registers that are joined together
    - Controls the baud rate of the transmitter and receiver hardware
    - Dependent on the CPU clock frequency!

Bit	7	6	5	4	3	2	1	0
	RXEN	TXEN		SFDEN	ODME	RXMODE[1:0]		MPCM
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

Bit 7 – RXEN Receiver Enable
Writing this bit to ‘1’ enables the USART receiver. The receiver will override normal port operation for the RxD pin when enabled. Disabling the receiver will flush the receive buffer invalidating the FERR, BUFOVF, and PERR flags. In GENAUTO and LINAUTO mode, disabling the receiver will reset the auto-baud detection logic.

Bit 6 – TXEN Transmitter Enable
Writing this bit to ‘1’ enables the USART transmitter. The transmitter will override normal port operation for the TxD pin when enabled. Disabling the transmitter (writing TXEN to '0') will not become effective until ongoing and pending transmissions are completed (i.e. when the Transmit Shift register and Transmit Buffer register does not contain data to be transmitted). When the transmitter is disabled, it will no longer override the TxDn pin, and the pin direction is set as input automatically by hardware, even if it was configured as output by the user.

USART Peripheral: Setup

Table 25-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Conditions	Baud Rate (Bits Per Seconds)	USART.BAUD Register Value Calculation
Asynchronous	$f_{BAUD} \leq \frac{f_{CLK\_PER}}{S}$ $USART.BAUD \geq 64$	$f_{BAUD} = \frac{64 \times f_{CLK\_PER}}{S \times BAUD}$	$BAUD = \frac{64 \times f_{CLK\_PER}}{S \times f_{BAUD}}$
Synchronous Host	$f_{BAUD} \leq \frac{f_{CLK\_PER}}{S}$ $USART.BAUD \geq 64$	$f_{BAUD} = \frac{f_{CLK\_PER}}{S \times BAUD[15:6]}$	$BAUD[15:6] = \frac{f_{CLK\_PER}}{S \times f_{BAUD}}$

BAUD register calculation found on page 370 in the datasheet

```
#define F_CPU 4000000
#define BAUD(BAUD_RATE) ( (uint16_t)((64.0 * (float)F_CPU) / (16.0 * (float)BAUD_RATE) + 0.5) )

#include <avr/io.h>


int main(void)
{
    USART1.CTRLB    = USART_TXEN_bm;
    USART1.BAUD     = BAUD(9600);

    while (1)
    {
    }
}
```

S is the number of samples per bit

- Asynchronous Normal mode: S = 16
- Asynchronous Double-Speed mode: S = 8
- Synchronous mode: S = 2

USART Peripheral: Setup

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)
5	3	5	1	PA7	CLKOUT		X7/Y7	0,OUT 1,OUT 2,OUT		0,OUT 1,OUT 2,OUT	0,XDIR(3)	0,$\overline{SS}$					0,WOD	EVOUTA (3)	
6				VDD															
7				GND															
8	4			PB0			X8/Y8				3,TxD			WO0(3)	WO0				4,IN0
9	5			PB1			X9/Y9				3,RxD			WO1(3)	WO1				4,IN1
10	6			PB2			X10/Y10				3,XCK		1,SDA(H)(3)	WO2(3)	WO2			EVOUTB	4,IN2
11	7			PB3			X11/Y11				3,XDIR		1,SCL(H)(3)	WO3(3)	WO3				4,OUT
12	8			PB4			X12/Y12				3,TxD(3)	1,MOSI(3)		WO4(3)	WO4	2,WO(3)	0,WOA(3)		
13	9			PB5			X13/Y13				3,RxD(3)	1,MISO(3)		WO5(3)	WO5	3,WO	0,WOB(3)		
14				PB6			X14/Y14				3,XCK(3)	1,SCK(3)	1,SDA(C)(3)				0,WOC(3)		4,OUT(3)
15				PB7			X15/Y15				3,XDIR(3)	1,$\overline{SS}$(3)	1,SCL(C)(3)				0,WOD(3)	EVOUTB (3)	
16	10	6	2	PC0							1,TxD	1,MOSI		WO0(3)		2,WO			1,IN0
17	11	7	3	PC1							1,RxD	1,MISO		WO1(3)		3,WO(3)			1,IN1
18	12	8	4	PC2	TWI						1,XCK	1,SCK	0,SDA(H)(3)	WO2(3)				EVOUTC	1,IN2
19	13	9	5	PC3	TWI						1,XDIR	1,$\overline{SS}$	0,SCL(H)(3)	WO3(3)					1,OUT

While the USART1 peripheral is setup correctly for transmission, we need to configure the GPIO as well

```c
int main(void)
{
    // Enable transmitter for USART1
    USART1.CTRLB    = USART_TXEN_bm;

    // Calculate the baud rate for USART1
    USART1.BAUD     = BAUD(9600);

    // Set PC0 as output
    PORTC.DIRSET    = PIN0_bm;

    // Set PC1 as input
    PORTC.DIRCLR    = PIN1_bm;

    while (1)
    {
    }
}
```

While clearing PC1 is technically not needed, it is a good habit

USART Peripheral: Transmit

- Transmitting a byte is as simple as inserting the value into the TXDATAL register
  - But we have to see if the USART peripheral is ready for new data first!

25.5.5 USART Status Register

Name: STATUS
Offset: 0x04
Reset: 0x20
Property: -

Bit	7	6	5	4	3	2	1	0
	RXCIF	TXCIF	DREIF	RXSIF	ISFIF		BDF	WFB
Access	R	R/W	R	R/W	R/W		R/W	W
Reset	0	0	1	0	0		0	0

Bit 7 – RXCIF USART Receive Complete Interrupt Flag
This flag is set when there are unread data in the receive buffer and cleared when the receive buffer is empty.

Bit 6 – TXCIF USART Transmit Complete Interrupt Flag
This flag is set when the entire frame in the transmit shift register has been shifted out, and there are no new data in the transmit buffer (TXDATAL and TXDATAH) registers. It is cleared by writing a '1' to it.

Bit 5 – DREIF USART Data Register Empty Interrupt Flag
This flag is set when the Transmit Data (USARTn.TXDATAL and USARTn.TXDATAH) registers are empty and cleared when they contain data not yet moved into the transmit shift register.

```
void cdc_send_char(char c)
{
    while (!(USART1.STATUS & USART_DREIF_bm));

    USART1.TXDATAL = c;
}
```

USART Peripheral: Transmit

- To send strings we simply need to transmit every char in the string buffer!
- For simplicity we include string.h for the strlen() function
  - But a while loop looking for the NULL character is just as valid
    - IF you have a timeout function

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

void cdc_send_char(char c)
{
    while (!(USART1.STATUS & USART_DREIF_bm));

    USART1.TXDATAL = c;
}
void cdc_send_string(char* str)
{
    for (uint8_t i=0; i < strlen(str); i++)
    {
        cdc_send_char(str[i]);
    }
}
```

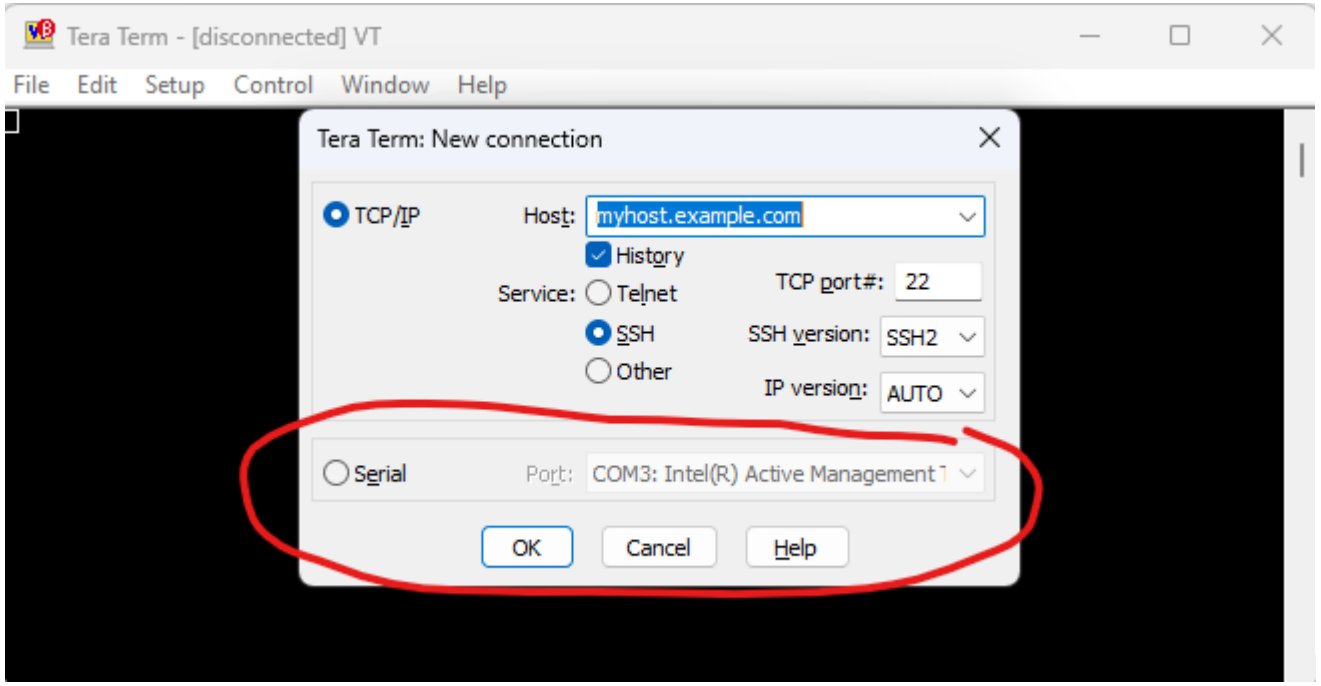
USART Peripheral: printf

- Learning how printf is created is beyond the scope of this course
  - It is instead given
  - This printf can not print floats as is, but this can be added at the cost of additional memory consumption
    - This will be added to github later
  - The library stdio.h is also required
    - Variadic functions

```
#include <string.h>
#include <stdio.h>

void cdc_send_char(char c)
{
    while (!(USART1.STATUS & USART_DREIF_bm));

    USART1.TXDATAL = c;
}
void cdc_send_string(char* str)
{
    for (uint8_t i=0; i < strlen(str); i++)
    {
        cdc_send_char(str[i]);
    }
}
void cdc_print(const char* text, ...)
{
    va_list argptr;
    char buffer[100];

    va_start(argptr, text);
    vsprintf(buffer, text, argptr);

    cdc_send_string(buffer);
    va_end(argptr);
}
```

USART Collective Task

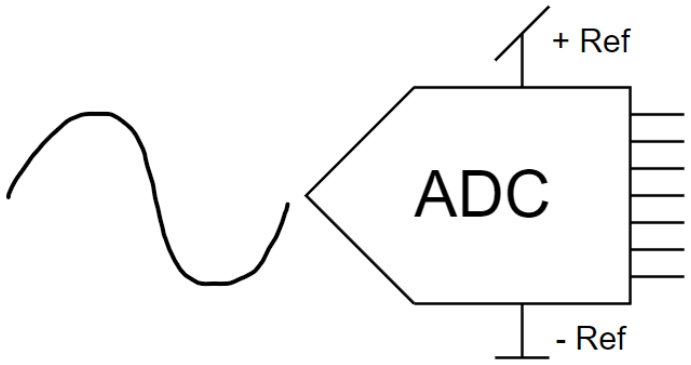
- Open teraterm
  - Select serial
  - Select AVR Curiosity board
- Include util/delay.h
- Add the following to your program
  - Upload and see what happens!




```c
int main(void)
{
    // Enable transmitter for USART1
    USART1.CTRLB    = USART_TXEN_bm;

    // Calculate the baud rate for USART1
    USART1.BAUD     = BAUD(9600);

    // Set PC0 as output
    PORTC.DIRSET    = PIN0_bm;

    // Set PC1 as input
    PORTC.DIRCLR    = PIN1_bm;

    uint8_t counter = 0;

    while (1)
    {
        cdc_print("Counting %d\r\n", counter++);
        _delay_ms(500);
    }
}
```

ADC

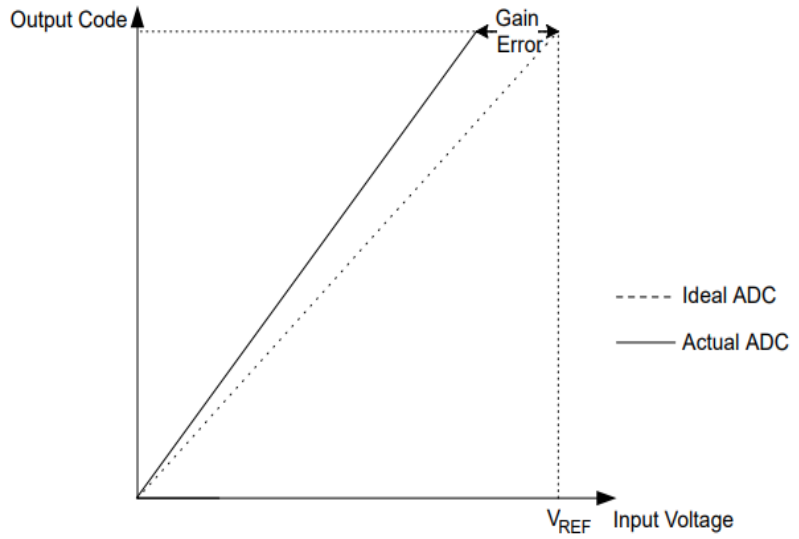
- Analog to Digital Converter
  - Signal conversion; Continuous to Discrete
- Measurement voltage
  - Between negative and positive reference
    - For AVRs, negative reference is always GND
    - Positive reference can be VDD or an internal or external reference
- Resolution
  - How many discrete steps there are between positive and negative references
  - Resolution can be increased with signal processing techniques at the cost of sample rate



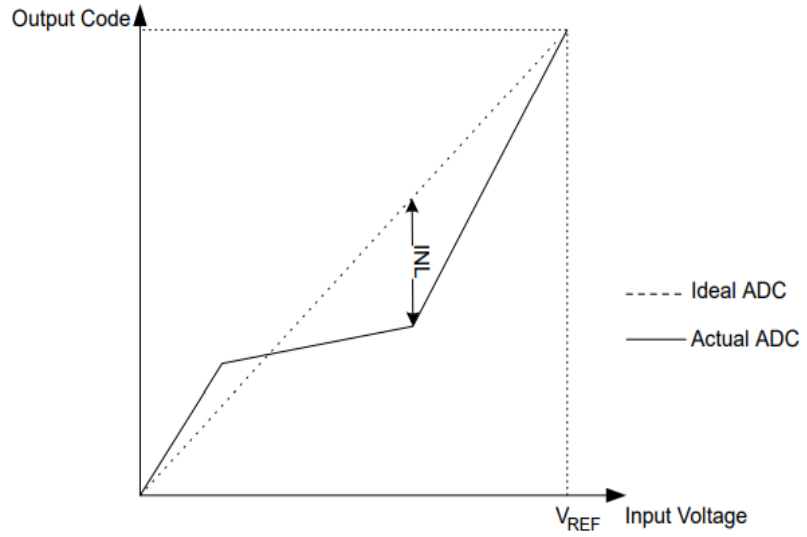
ADC

- Resolution
  - 2^n where n is number of bits
  - 8-bit ADC has 2^8 = 256 steps
    - Since 0 is included, the highest value is 255
- Accuracy & Precision
  - There is always a quantitation error as the least significant bit is always rounded
  - Offset error
  - Gain error
  - Integral and differential error
  - ++++

Gain error



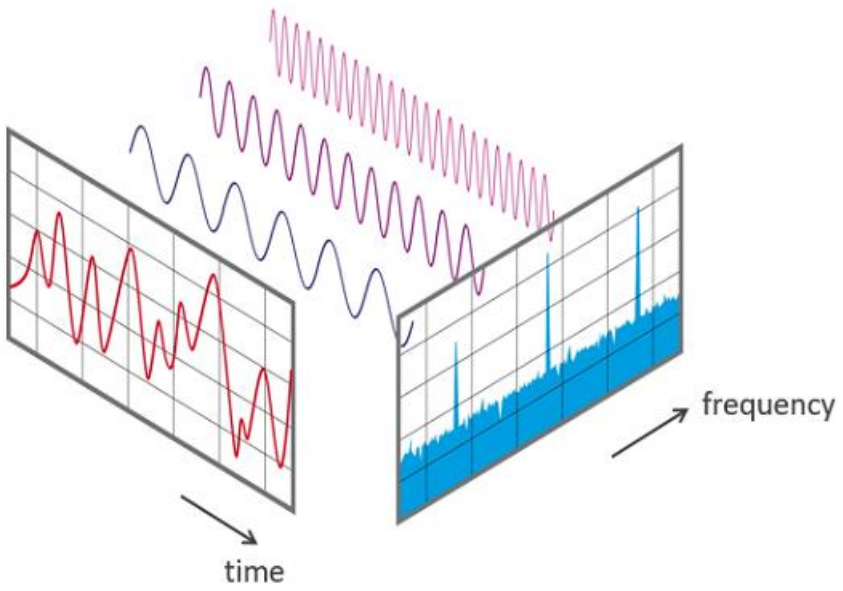
Integral error

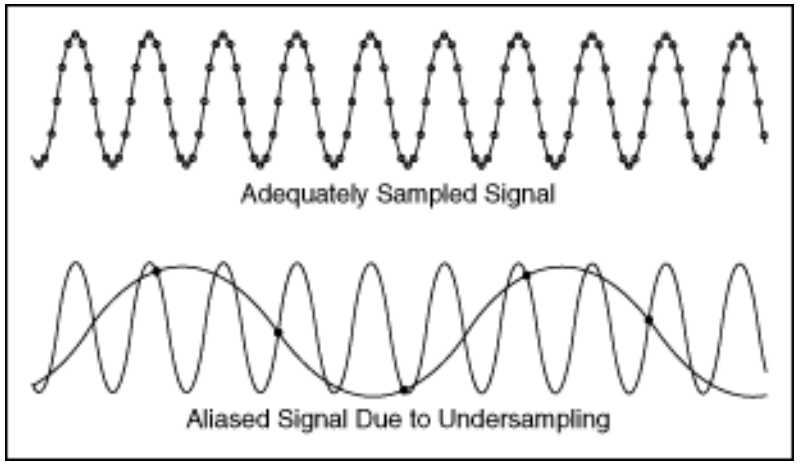


A quick note on sampling

- Sampling frequency is how many samples an ADC can do per second
- All signals can be reconstructed/deconstructed into a sum of sines
  - This is the basis of the Fourier transform
- Shannon-Nyquist theorem
  - Must sample at least twice as fast as the bandwidth
  - Avoid aliasing
- Make sure your ADC can sample the signal you are looking for
  - Similarly make sure you are not sampling something else (filtering)
- Aliasing can also be useful
  - Requires planning

Stolen from the internet, but I cannot remember from where





ADC

- Many different types
  - Flash ADC
  - Delta-sigma ADC
  - Integrating ADC
  - RC network ADC
  - SAR ADC
  - ++++
- Most microcontrollers use SAR ADC
  - Successive AppRoximation ADC
- Result
  - Normalize then scale
  - Single ended – unsigned value
    - Only MUXPOS
  - Differential – signed value
    - MUXPOS and MUXNEG
  - Where
    - V = Measured voltage in volts
    - Vref is reference voltage in volts
    - RES Is ADC result (unitless)
    - B is ADC resolution (bit width)

$$V_{single} = V_{Ref} \frac{\mathrm{RES}}{2^B - 1} \quad [0,\ 2^B - 1]$$

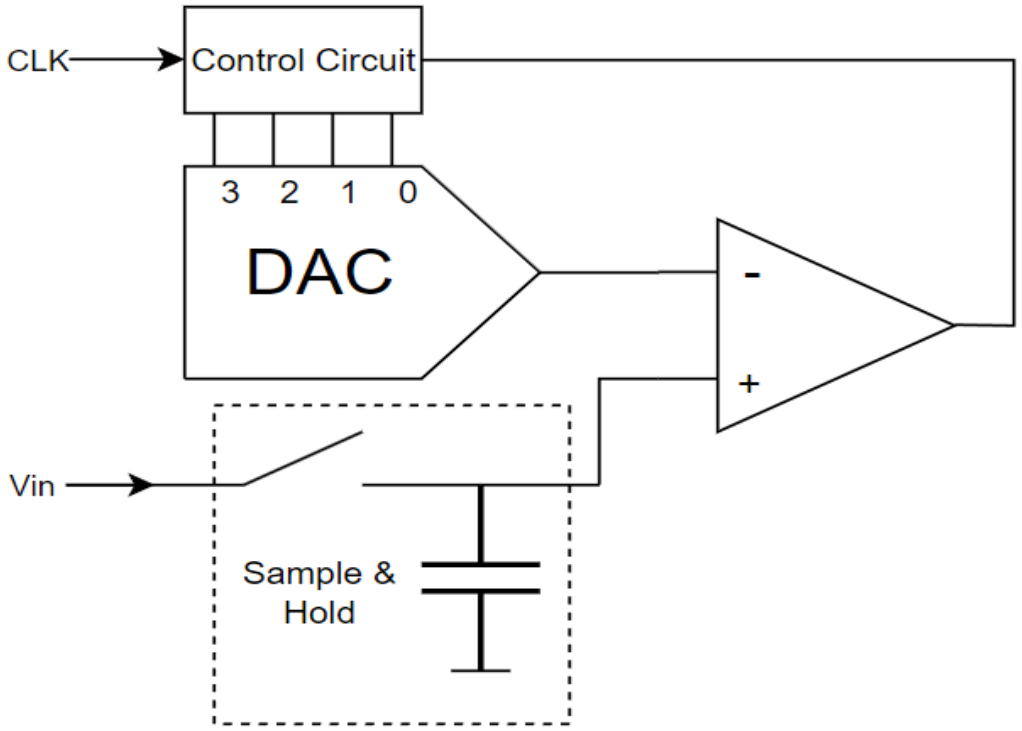
$$V_{diff} = V_{Ref} \frac{\mathrm{RES}}{2^{B-1}} \quad [-2^{B-1},\ 2^{B-1} - 1]$$

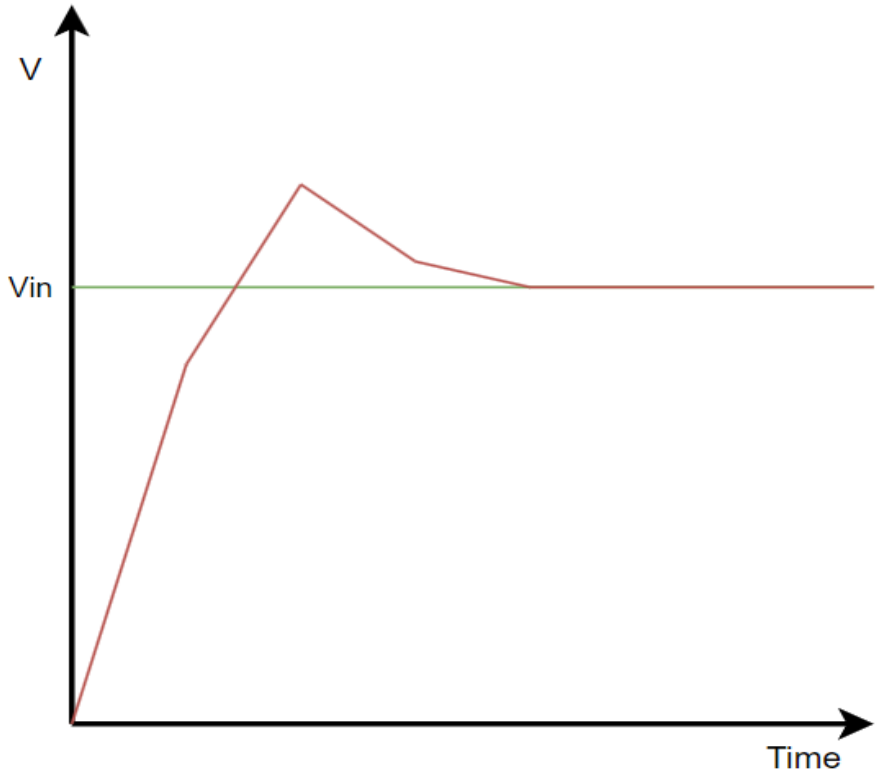
Example:
ADC resolution = 12 bits
Single ended measurement: [0, 4095]
Differential measurement: [-2048, 2047]

Typical implementation

```
#define ADC_VREF    1.024   // Define reference voltage for the ADC (optional)
#define ADC_RES     4095    // Define the ADC resolution (optional)
float adc_single_get()
{
    float v_measure;
    uint16_t result;

    // Get unsigned result from ADC
    result = ADC0.RES;

    // Convert 12 bit result to voltage
    v_measure = (ADC_VREF * result) / ADC_RES; // Scale before normalizing as integers discard number after decimal point

    return v_measure;
}
```

ADC

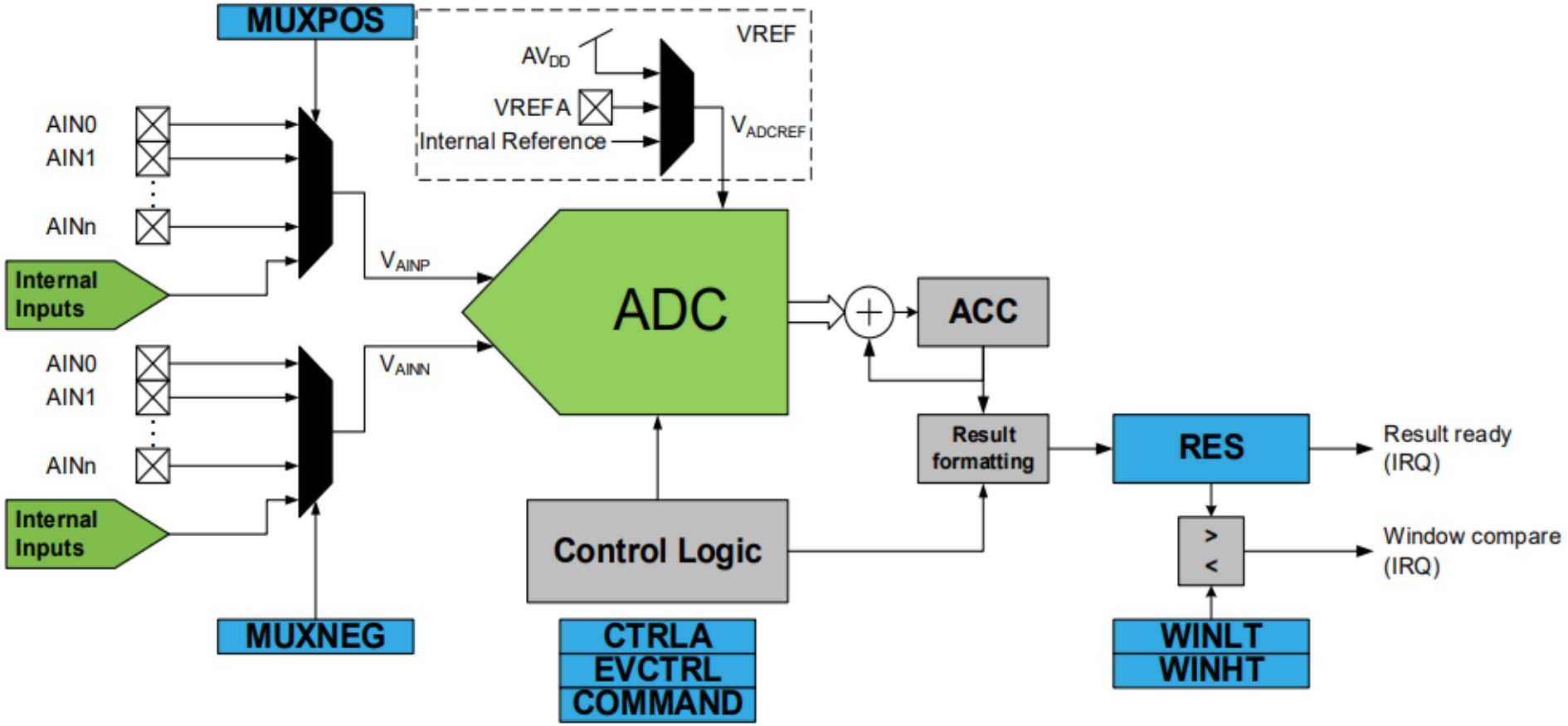




ADC in AVRs

31.2.1 Block Diagram

Figure 31-1. Block Diagram



Task: ADC

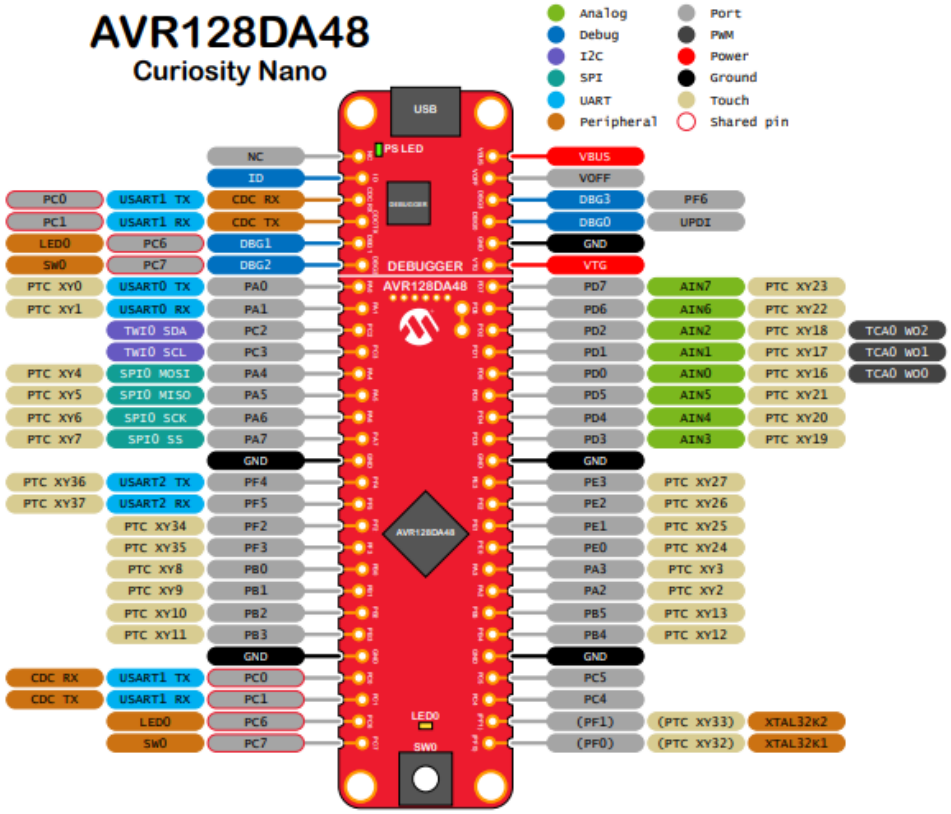
- Update the UART code to transmit ADC results every 500 ms instead

- Most basic ADC implementation
  - Blocking
  - Only one ADC channel

- First part together
- Last part on your own

Task: ADC Part 1

- Find a suitable pin
  - Must be internally connected to the ADC
- Connect a potentiometer to the pin
  - Potentiometers are three terminal devices
  - Middle pin is connected to AVR pin
  - Pins on left and right are interchangeable
    - Connect one to VDD
    - Connect the other to GND



Task: ADC Part 2

- Two peripherals are used for analog to digital measurements
  - VREF
  - ADC
- VREF
  - Only one instance for entire AVR – VREF
  - Select reference voltage for analog parts
    - We want the reference for the ADC to be VDD in this case
    - Ratiometric measurement
- ADC
  - Only one instance for entire AVR – ADC0
  - Select what pin is connected to ADC
  - Select a prescaler
  - Enable ADC

Task: ADC Part 2

- Set VREF to VDD

```
// Set the ADC reference to VDD
VREF.ADC0REF     = VREF_REFSEL_VDD_gc;
```


19.5.1 ADC0 Reference

Name: ADC0REF
Offset: 0x00
Reset: 0x00
Property: -

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

Bit 7 – ALWAYSON Reference Always On
This bit controls whether the ADC0 reference is always on or not.

Value	Description
0	The reference is automatically enabled when needed
1	The reference is always on

Bits 2:0 – REFSEL[2:0] Reference Select
This bit field controls the reference voltage level for ADC0.
Note:

1. The values given for internal references are only typical. Refer to the *Electrical Characteristics* section for further details.

Value	Name	Description
0x0	1V024	Internal 1.024V reference(1)
0x1	2V048	Internal 2.048V reference(1)
0x2	4V096	Internal 4.096V reference(1)
0x3	2V500	Internal 2.500V reference(1)
0x4	-	Reserved
0x5	VDD	VDD as reference
0x6	VREFA	External reference from the VREFA pin
0x7	-	Reserved

Task: ADC Part 2

ADC - Analog-to-Digital Converter

- Connect ADC to correct pin
  - I chose PD7

```
// Make PD7 input (optional as it is an input by default)
PORTD.DIRCLR    = PIN7_bm;

// Connect ADC to PD7 (AIN7)
ADC0.MUXPOS     = ADC_MUXPOS_AIN7_gc;
```

31.5.7 MUX Selection for Positive ADC Input

Name: MUXPOS
Offset: 0x08
Reset: 0x00
Property: -

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

Bits 6:0 – MUXPOS[6:0] MUX Selection for Positive ADC Input
This bit field selects which analog input is connected to the positive input of the ADC. If this bit field is changed during a conversion, the change will not take effect until the conversion is complete.

Value	Name	Description
0x00-0x0F	AIN0-AIN15	ADC input pin 0-15
0x10-0x15	AIN16-AIN21	ADC input pin 16-21
0x16-0x3F	-	Reserved
0x40	GND	Ground
0x41	-	Reserved
0x42	TEMPSENSE	Temperature sensor
0x43-0x47	-	Reserved
0x48	DAC0	DAC0
0x49	DACREF0	AC0 DAC Voltage Reference
0x4A	DACREF1	AC1 DAC Voltage Reference
0x4B	DACREF2	AC2 DAC Voltage Reference
Other	-	Reserved

It is best to disable the input buffer for the pin we connect to our analog signals. This is because the input buffer may add switching noise to our signal. We will not do that today

Task: ADC Part 2

- Set the ADC Prescaler
  - Max clock frequency for ADC is 2 MHz
- Prescaler divides system clock by set values for the ADC
- When the CPU runs at 4 MHz we can use DIV2
  - 4 MHz / 2 = 2 MHz
  - Since this is the default value, it can technically be ignored
    - It is a good habit to set it regardless
- For the maximum CPU frequency of 24 MHz
  - DIV12 shall be used

31.5.3 Control C

Name: CTRLC
Offset: 0x02
Reset: 0x00
Property: -

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

Bits 3:0 – PRESC[3:0] Prescaler
This bit field defines the division factor from the peripheral clock (CLK_PER) to the ADC clock (CLK_ADC).

Value	Name	Description
0x0	DIV2	CLK_PER divided by 2
0x1	DIV4	CLK_PER divided by 4

```
// Set the ADC clock divider (Max frequency for ADC is 2 MHz [F_CPU / DIV])
ADC0.CTRLC      = ADC_PRESC_DIV2_gc;
```

37.17 ADC

Table 37-23. ADC Accuracy Specifications

Operating Conditions: V_{DD} = 3.0V, T_A = 25°C						
Symbol	**Description**	**Min.**	**Typ. †**	**Max.**	**Unit**	**Conditions**
N_R	Resolution	—	—	12	bit	
E_{INL}	Integral nonlinearity error	-1.5	0.1	1.5	LSb	V_{DD}=V_{REF}=3.0V
E_{DNL}	Differential nonlinearity error(1)	-1	0.1	1	LSb	V_{DD}=V_{REF}=3.0V
E_{OFF}	Offset error	-5	2.5	5	LSb	V_{DD}=V_{REF}=3.0V
E_{GAIN}	Gain error	-5	1.5	5	LSb	V_{DD}=V_{REF}=3.0V
E_{ABS}	Absolute error	—	—	—	LSb	V_{DD}=V_{REF}=3.0V
V_{ADCREF}	ADC reference voltage	1.8	—	V_{DD}	V	
V_{AIN}	Full-scale range	GND	—	V_{ADCREF}	V	
Z_{AIN}	Recommended impedance of analog voltage source	—	1	—	kΩ	
R_{VREFA}	ADC voltage reference ladder impedance(2)	—	50	—	kΩ	

† Data in the "Typ." column is at T_A = 25°C and V_{DD} = 3.0V unless otherwise specified. These parameters are for design guidance only and are not tested.

Notes:
1. The ADC conversion result never decreases with an increase in the input and has no missing codes.
2. This is the impedance seen by the VREFA pin when the external reference is selected.

Table 37-24. ADC Conversion Timing Specifications

Symbol	Description	Min.	Typ. †	Max.	Unit	Conditions
T_{CLK_ADC} *	ADC clock period	0.5	—	8	µs	
t_{CNV}	Conversion time	—	$13.5T_{CLK\_ADC} + 2T_{CLK\_PER}$	—		
t_{ACQ}	Acquisition time	—	$2T_{CLK\_ADC}$	—	µs	
f_{ADC} *	Sample rate	8	—	130	ksps	
t_S	Sampling time	—	$2T_{CLK\_ADC}$	—		
t_{SENSE} *	Delay for changing MUXPOS to TEMP	—	40	—	µs	
t_{ADC_INIT} *	Initialization time	—	6	—	µs	

Task: ADC Part 2

- Enable ADC

```
// Enable ADC, default config = single ended, 12b mode
ADC0.CTRLA      = ADC_ENABLE_bm;
```

31.5.1 Control A

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

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY		CONVMODE	LEFTADJ	RESSEL[1:0]		FREERUN	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 7 – RUNSTDBY Run in Standby
This bit determines whether the ADC still runs during Standby.

Value	Description
0	ADC will not run in Standby sleep mode. An ongoing conversion will finish before the ADC enters sleep mode.
1	ADC will run in Standby sleep mode

Bit 5 – CONVMODE Conversion Mode
This bit defines if the ADC is working in Single-Ended or Differential mode.

Value	Name	Description
0x0	SINGLEENDED	The ADC is operating in Single-Ended mode where only the positive input is used. The ADC result is presented as an unsigned value.
0x1	DIFF	The ADC is operating in Differential mode where both positive and negative inputs are used. The ADC result is presented as a signed value.

Bit 4 – LEFTADJ Left Adjust Result
Writing a '1' to this bit will enable left adjustment of the ADC result.

Bits 3:2 – RESSEL[1:0] Resolution Selection
This bit field selects the ADC resolution. When changing the resolution from 12-bit to 10-bit, the conversion time is reduced from 13.5 CLK_ADC cycles to 11.5 CLK_ADC cycles.

Value	Description
0x00	12-bit resolution
0x01	10-bit resolution
Other	Reserved

Bit 1 – FREERUN Free-Running
Writing a '1' to this bit will enable the Free-Running mode for the ADC. The first conversion is started by writing a '1' to the Start Conversion (STCONV) bit in the Command (ADCn.COMMAND) register.

Bit 0 – ENABLE ADC Enable

Value	Description
0	ADC is disabled
1	ADC is enabled

Task: ADC Part 2

```
// Set the ADC reference to VDD
VREF.ADC0REF    = VREF_REFSEL_VDD_gc;

// Make PC0 an output
PORTC.DIRSET    = PIN0_bm;

// Make PC1 an input (optional as it is an input by default)
PORTC.DIRCLR    = PIN1_bm;

// Make PD7 input (optional as it is an input by default)
PORTD.DIRCLR    = PIN7_bm;

// Connect ADC to PD7 (AIN7)
ADC0.MUXPOS     = ADC_MUXPOS_AIN7_gc;

// Set the ADC clock divider (Max frequency for ADC is 2 MHz [F_CPU / DIV])
ADC0.CTRLC      = ADC_PRESC_DIV2_gc;

// Enable ADC, default config = single ended, 12b mode
ADC0.CTRLA      = ADC_ENABLE_bm;
```

Task: ADC Part 3

- Do measurements in the while(1) loop
- Store result in a variable
- Print variable via UART
- Play with the potentiometer to make sure you get some readings
  - Values should be in the range 0-4095
  - Optional: Convert result to voltage!
- Solution on next slide
  - Try first
  - Make sure to use the datasheet!

```
uint16_t adc_result;
while (1)
{
    // Start ADC measurement
    <COMMAND> register

    // Wait for ADC measurement to complete (blocking)
    <INTFLAGS> register

    // Store result in a adc_result
    <RES> register

    // Print the ADC result
    cdc_print("ADC Result = %d\r\n", adc_result);
    _delay_ms(500);
}
```

Task: ADC Part 3 Solution

```
while (1)
{
    // Start ADC measurement
    ADC0.COMMAND = ADC_STCONV_bm;

    // Wait for ADC measurement to be complete (blocking)
    while (!(ADC0.INTFLAGS & ADC_RESRDY_bm));

    // Store ADC result in a variable
    adc_result = ADC0.RES;

    // Print the ADC result
    cdc_print("ADC Result = %d\r\n", adc_result);
    _delay_ms(500);
}
```

Next time

- Timers/Counters
  - How useful it is to count
- Interrupts
  - How special events can be used to execute code outside main loop

- Questions?