MCT 4334

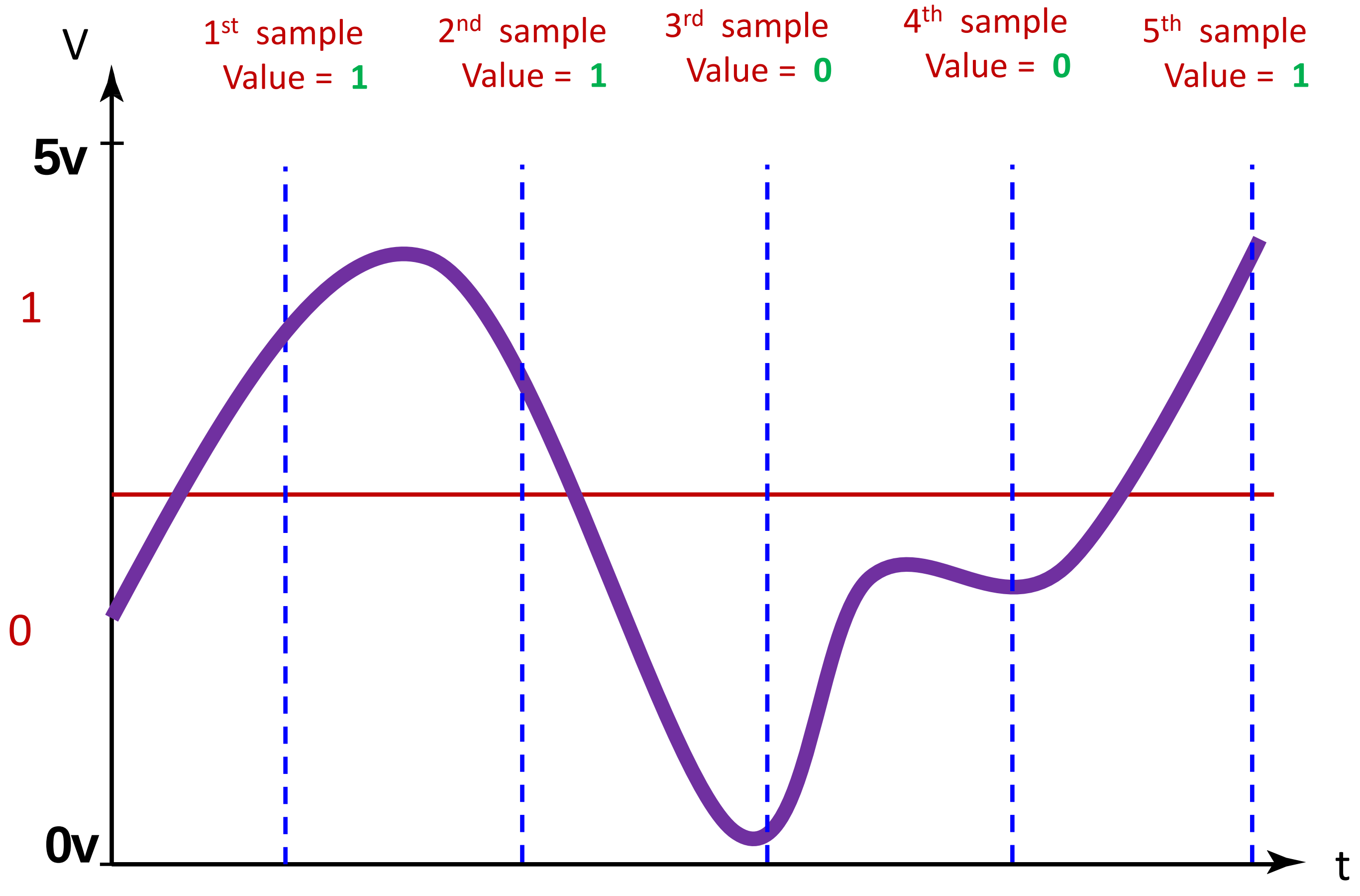
Embedded System Design

Week 05 Analog Input

Outline

- ADC architecture based on combination logic vs sequential logic
- ADC in ATmega328p
- Programming examples

- An analog voltage (from 0 to 5V) has infinite number possible values.
- An ADC converts a analog **value** (continuous) to a discrete number of levels so that it can be stored/processed



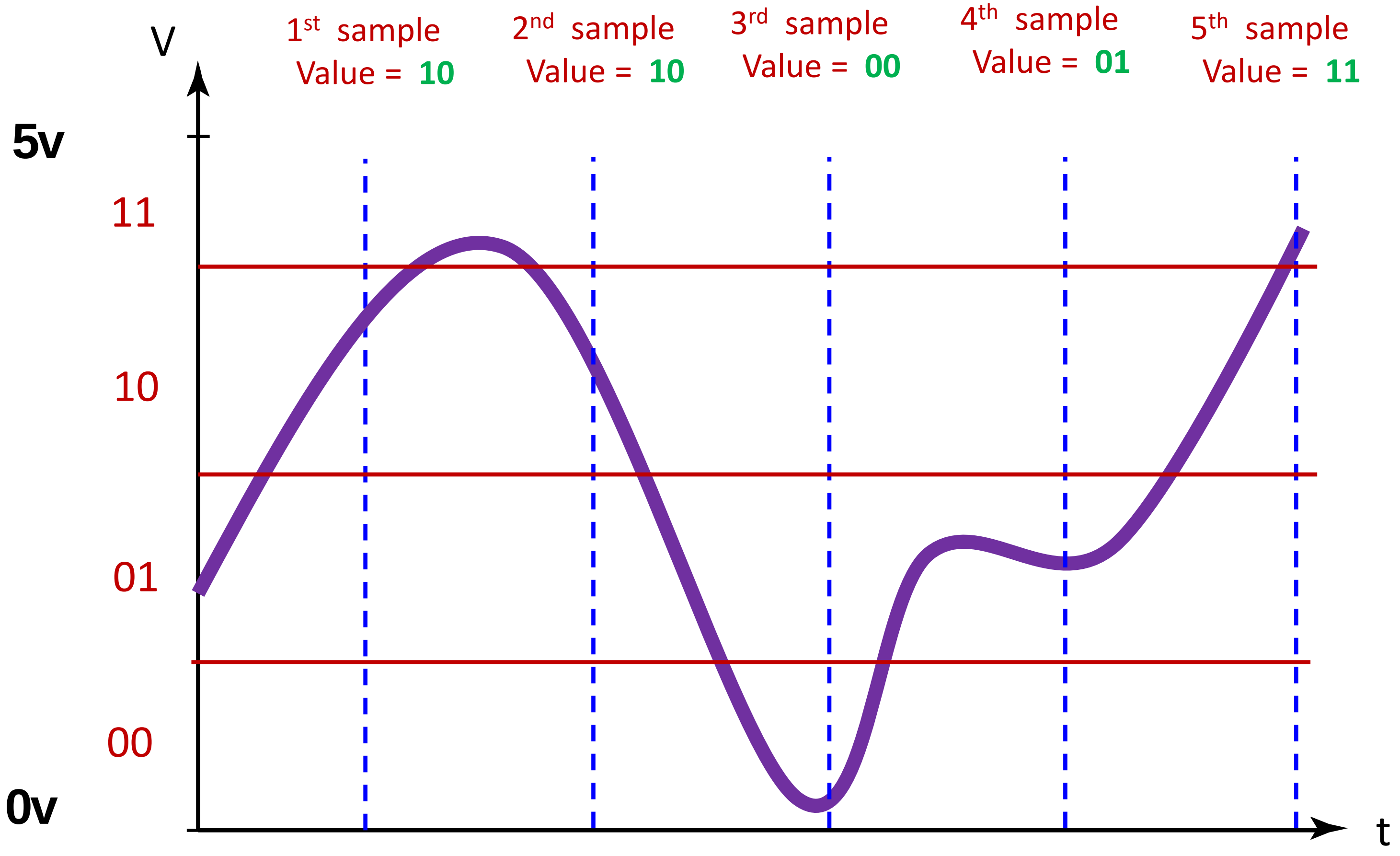
1-bit ADC

Note:

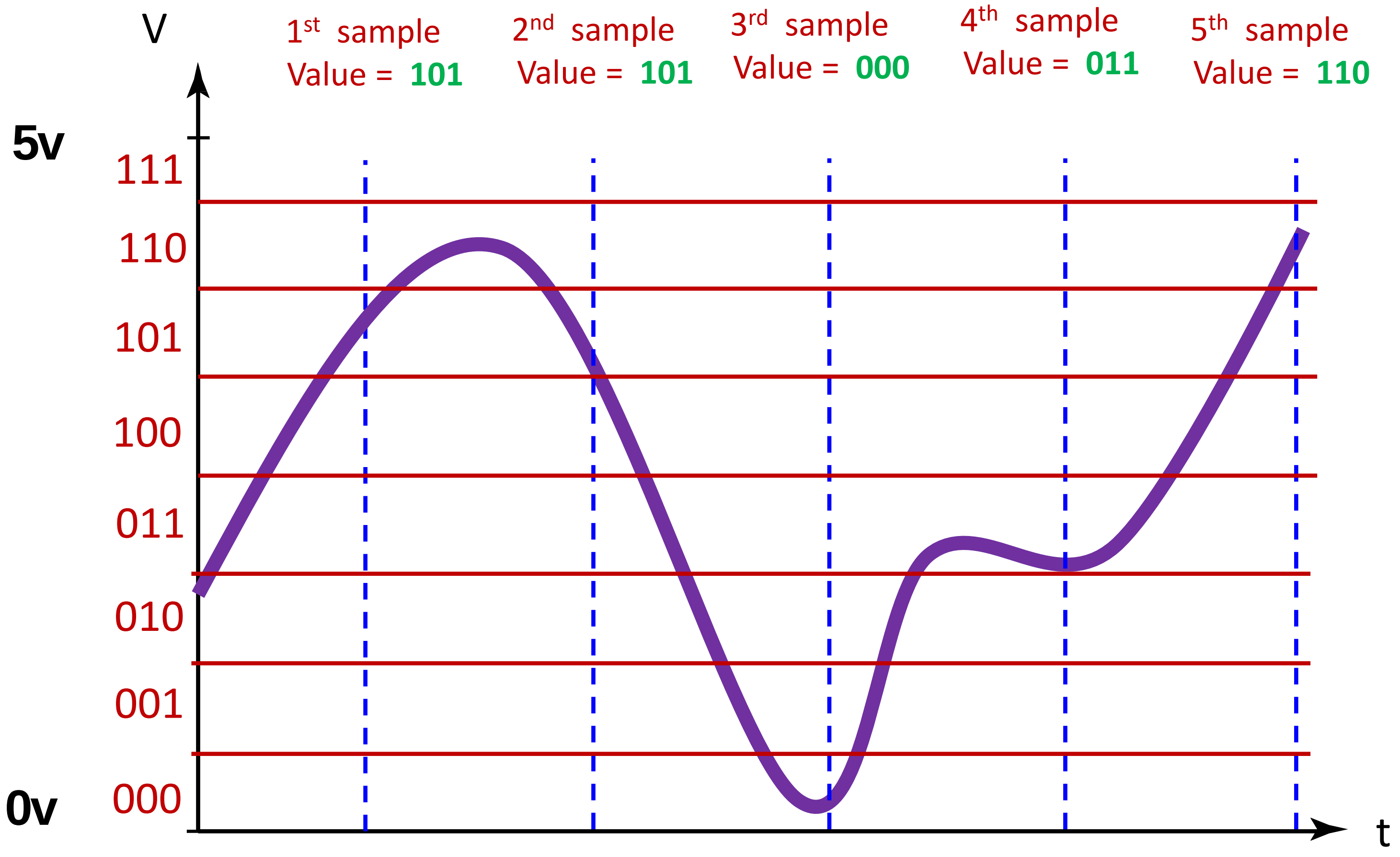
denotes taking a sample

We can take samples any time we want

2-bit ADC

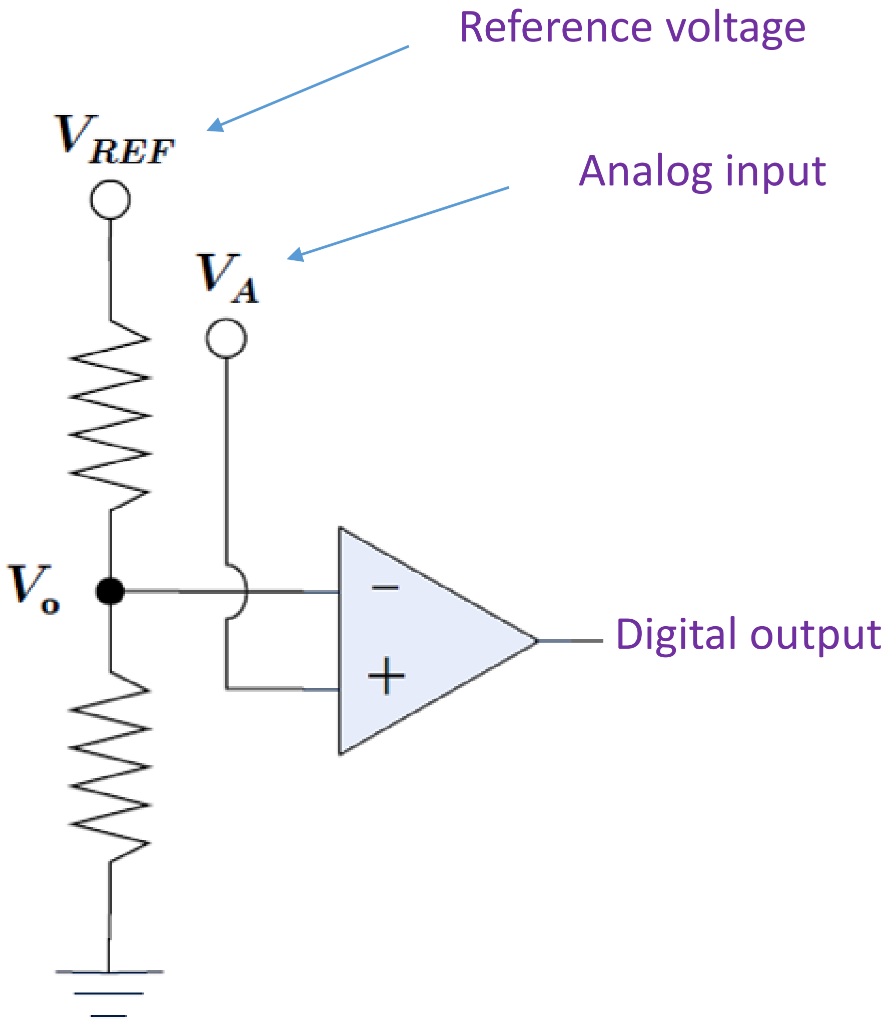

V
1st sample
Value = 10
2nd sample
Value = 10
3rd sample
Value = 00
4th sample
Value = 01
5th sample
Value = 11
5v
11
10
01
00
0v
t

3-bit ADC



V
1st sample
Value = 101
2nd sample
Value = 101
3rd sample
Value = 000
4th sample
Value = 011
5th sample
Value = 110
5v
111
110
101
100
011
010
001
0v
000
t

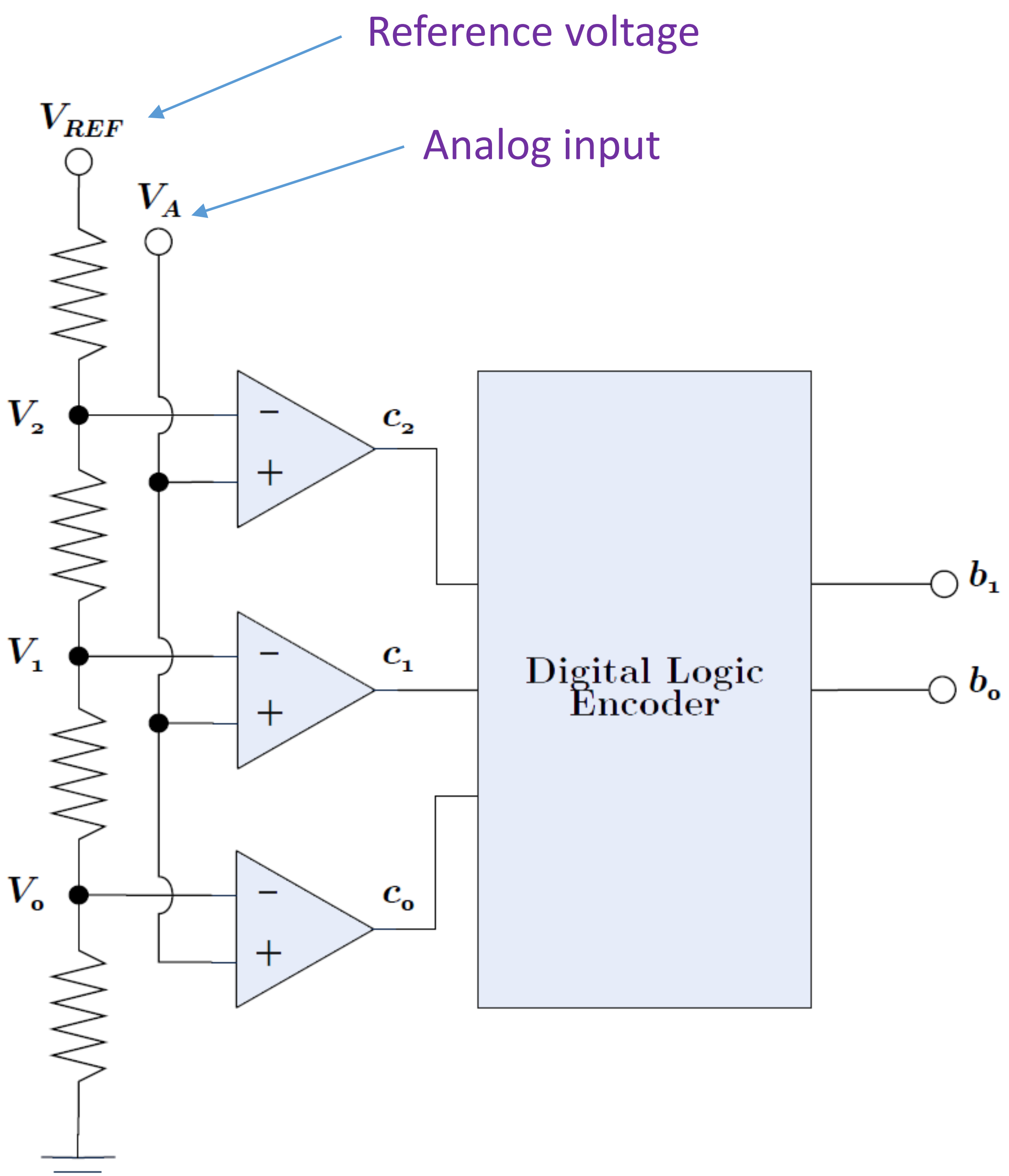
1-bit ADC



- V_0 = half of V_{REF} (if the two resistors have the same resistance)

- Digital output will be
  HIGH if the analog input voltage is higher than the reference voltage
  LOW if the analog input voltage is lower than the reference voltage

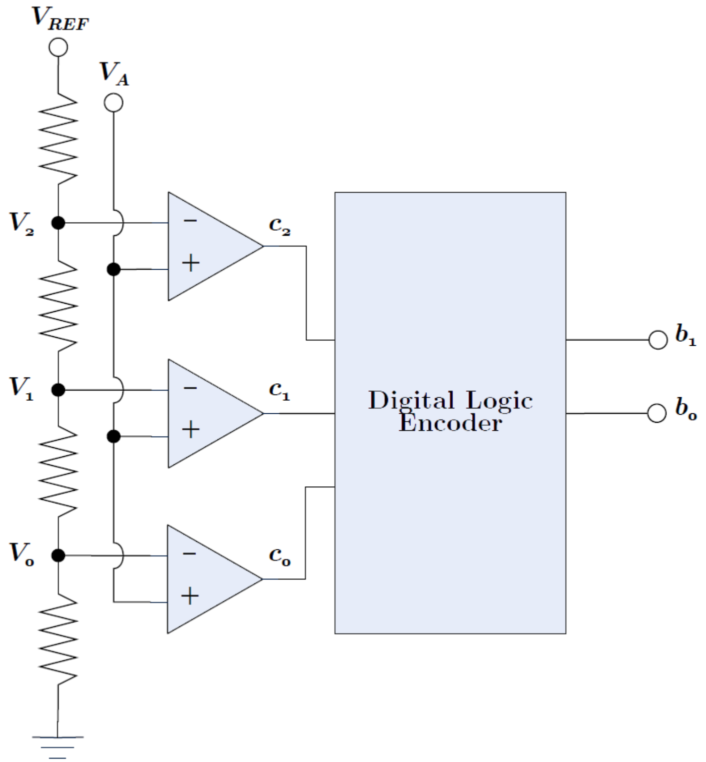
2-bit ADC

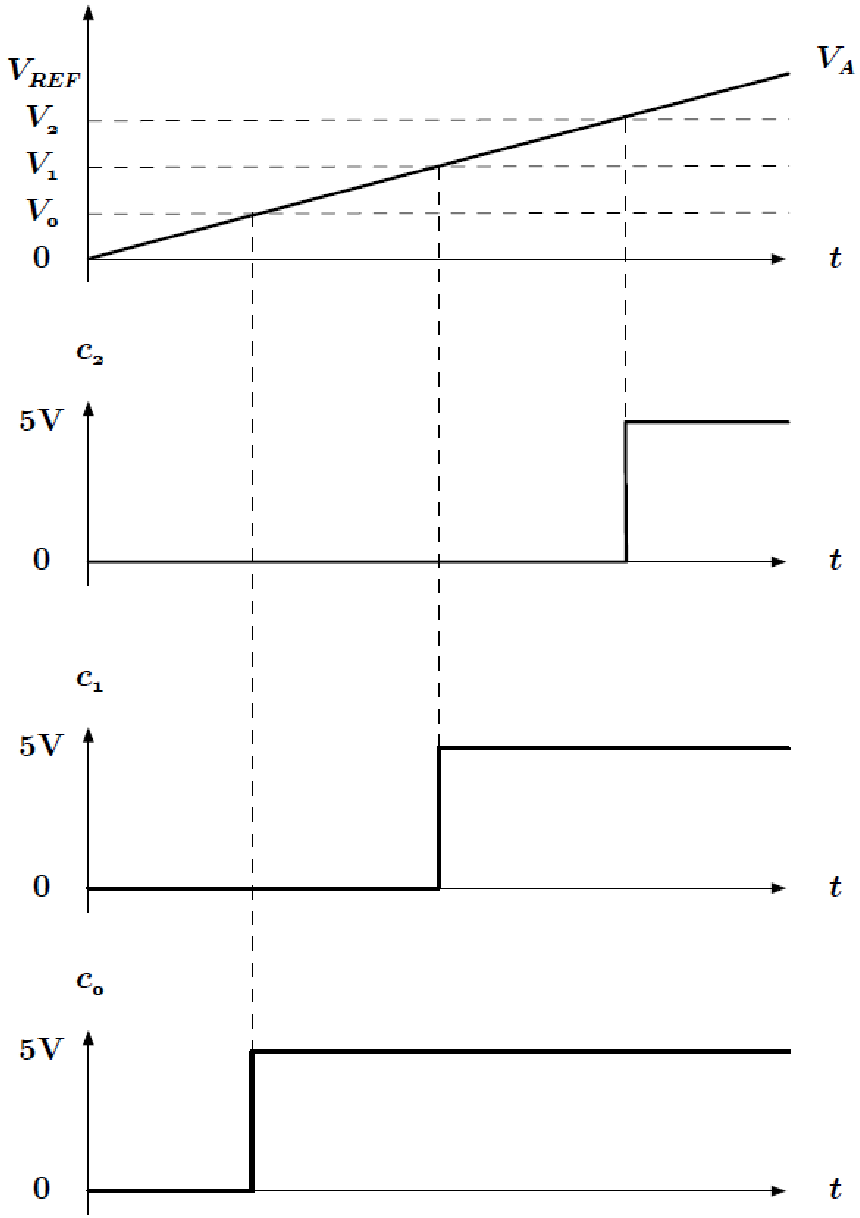


V_0 is 1/4 of V_{REF}.
V_1 is 2/4 of V_{REF}.
V_2 is 3/4 of V_{REF}.

If $V_A < V_0$,	$C_0=0$	$C_1=0$	$C_2=0$
If $V_0 < V_A < V_1$,	$C_0=1$	$C_1=0$	$C_2=0$
If $V_1 < V_A < V_2$,	$C_0=1$	$C_1=1$	$C_2=0$
$V_A > V_2$,	$C_0=1$	$C_1=1$	$C_2=1$

The effect of changing V_A linearly over time





- There are only four possible outputs

  C_0=0 C_1=0 C_2=0
  C_0=1 C_1=0 C_2=0
  C_0=1 C_1=1 C_2=0
  C_0=1 C_1=1 C_2=1

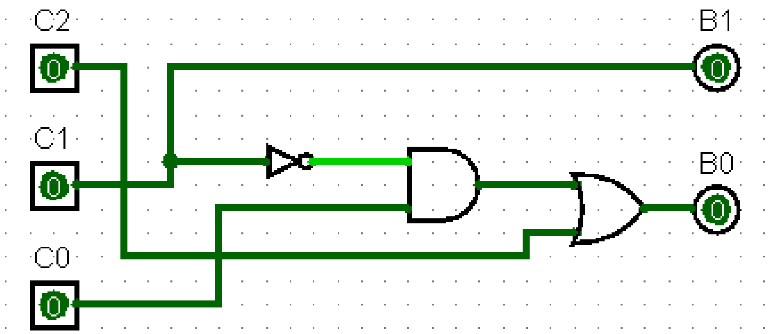
- Two bits are sufficient to represent them

- A combination circuit with 3 inputs and 2 outputs (encoder) can be designed

Truth table

Input			Output	
C2	C1	C0	B1	B0
0	0	0	0	0
0	0	1	0	1
0	1	0	x	x
0	1	1	1	0
1	0	0	x	x
1	0	1	x	x
1	1	0	x	x
1	1	1	1	1

Circuit



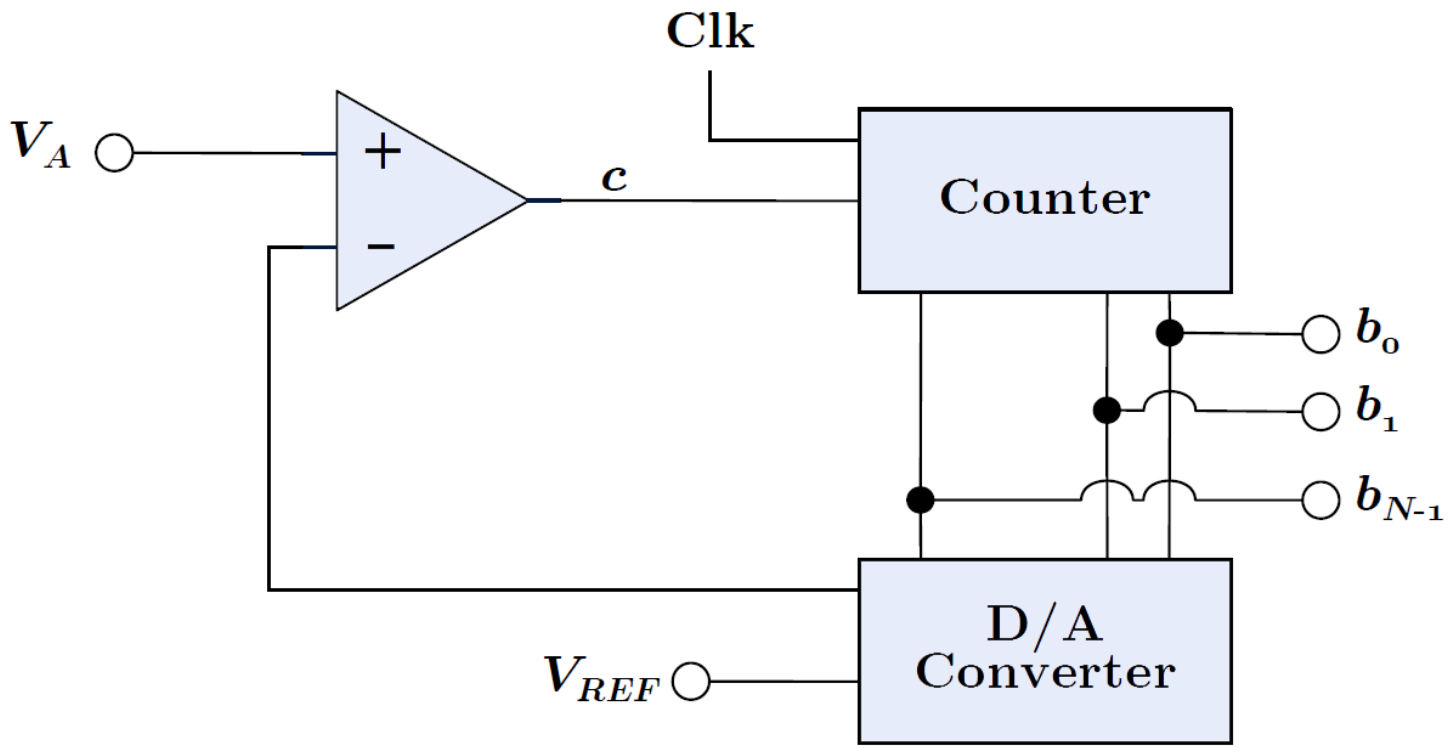
Number of bits	Number of different voltage levels	Voltage resolution (for $V_{REF} = 5V$)
1	2	2.5V
2	4	1.25V
3	8	625mV
4	16	312.5mV
5	32	156.25mV
6	64	78.125mV
7	128	39.0625mV
8	256	19.53125mV
9	512	9.765625mV
10	1024	4.882813mV

Pros and cons of ADCs based on combination logic

- Very fast (almost instant)
- N-bit ADC requires 2^N -1 comparators, 2^N resistors and certain number of logic gates

ADC-based on sequential logic

- Successive-approximation ADC uses a binary search technique.
- N-bit successive approximation ADC takes N clock cycles to complete
- In each iteration, the uncertainty interval gets halved.



Example of 4-bit successive approximation ADC

Determine the digital output value of a 4-bit successive approximation ADC if analog input is 4.1343134 and V_{REF} = 5V.

Answer:

4-bit resolution allows 16 different levels
Voltage difference per level = 5V / 16 = 0.3125

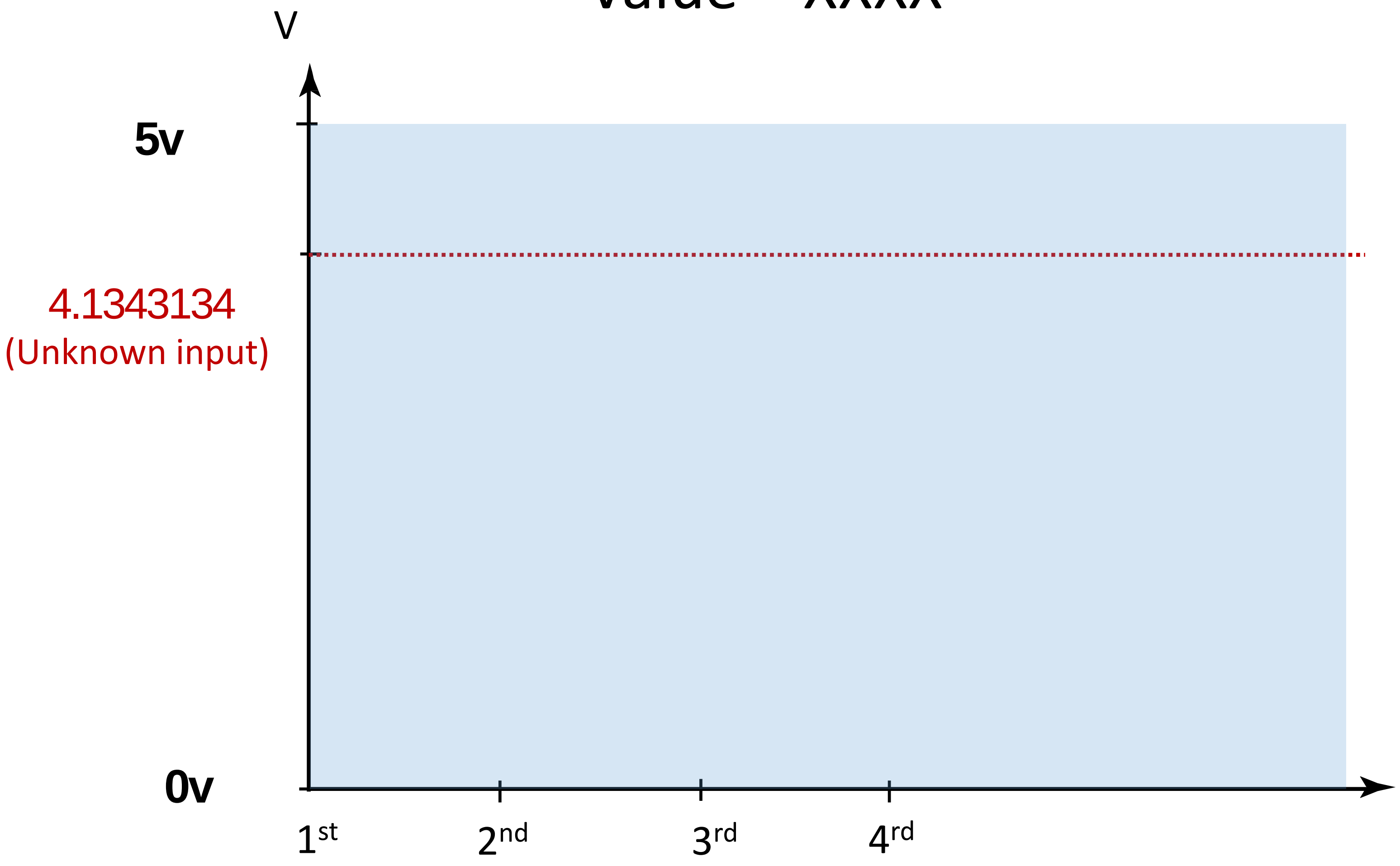
Digital	Analog (V)
0000	0.3125
0001	0.625
0010	0.9375
0011	1.25
0100	1.5625
0101	1.875
0110	2.1875
0111	2.5
1000	2.8125
1001	3.125
1010	3.4375
1011	3.75
1100	4.0625
1101	4.375
1110	4.6875
1111	5

Answer: → **1101**

Successive approximation ADC

Initially

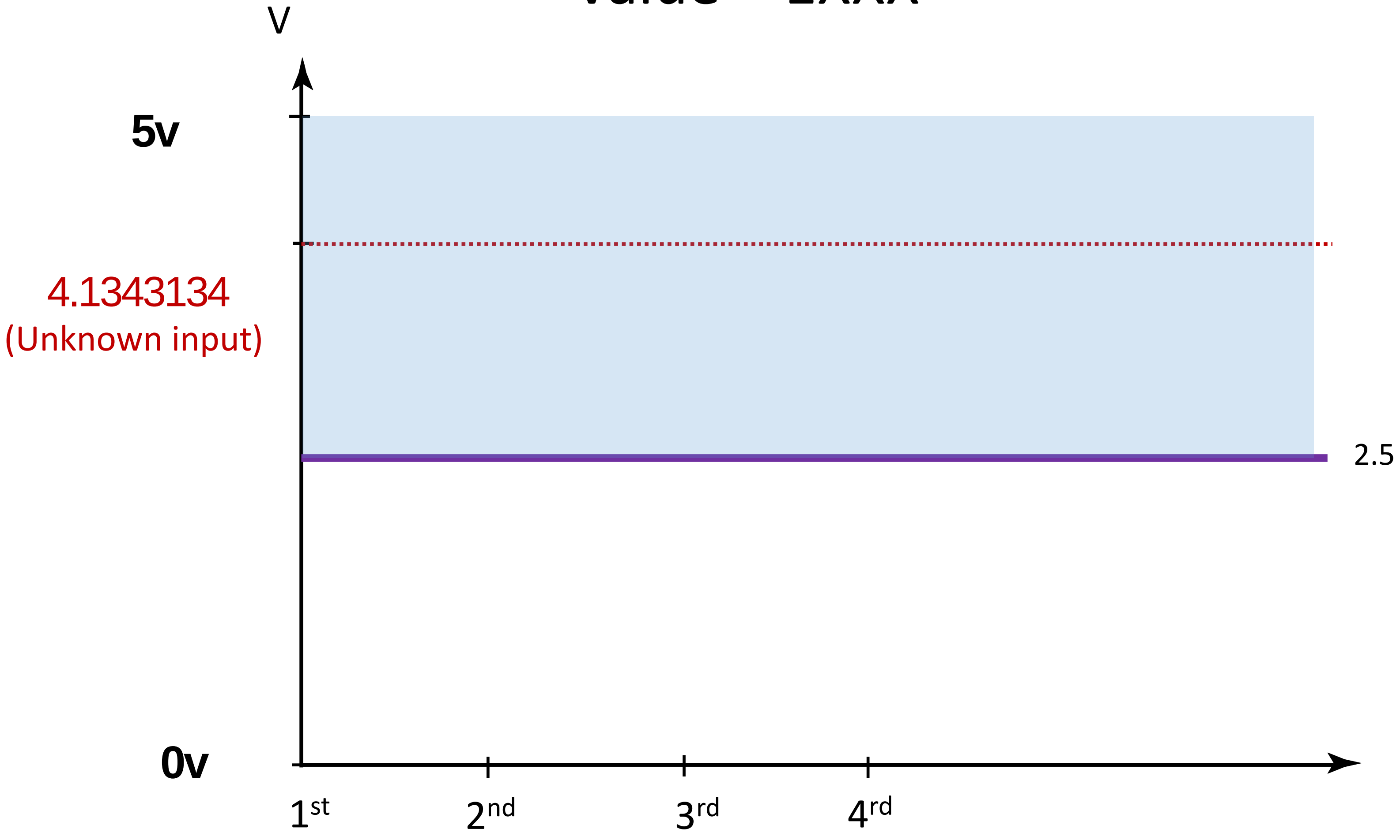
Value = XXXX



Successive approximation ADC

After 1st iteration

Value = 1XXX

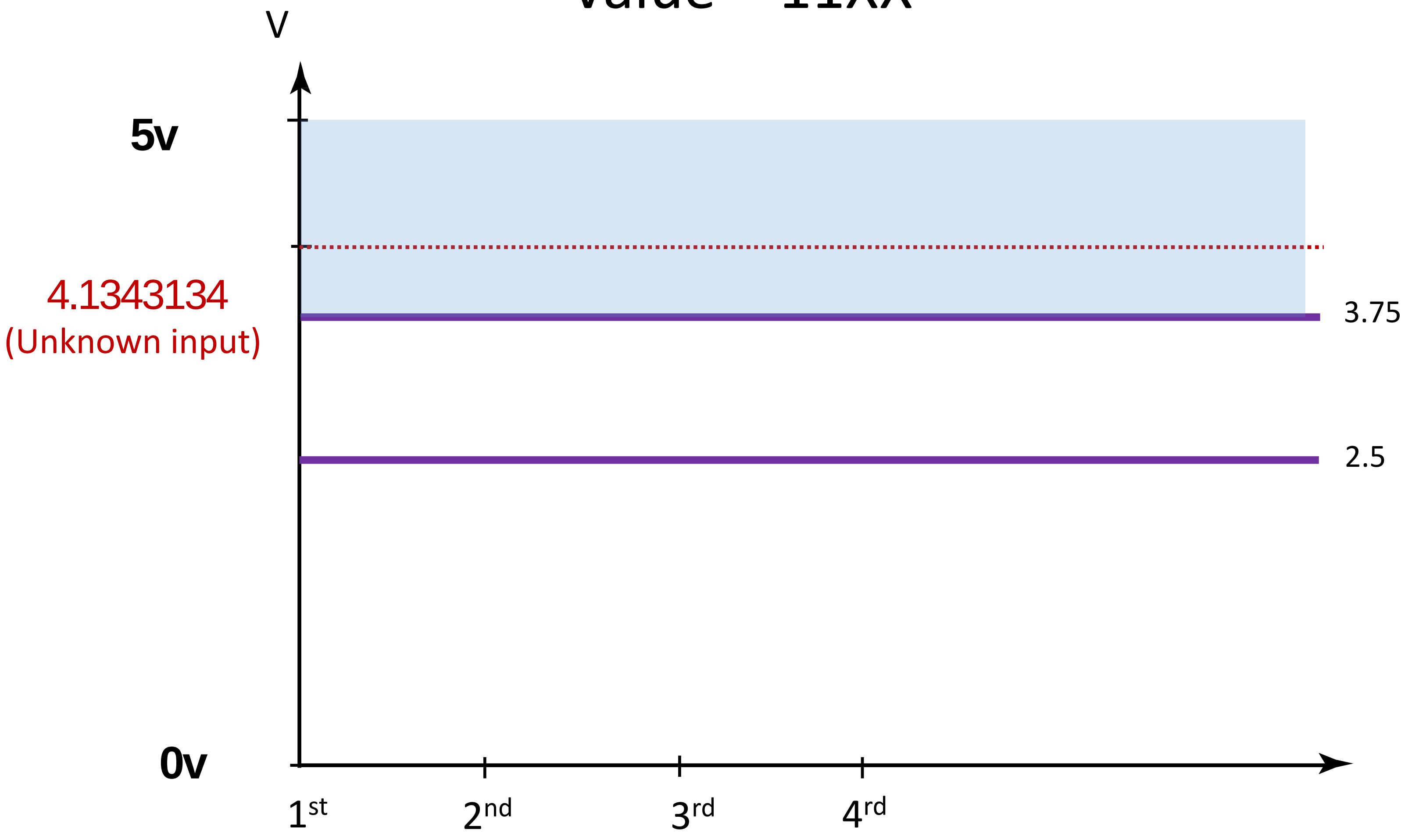


- The number input could be anywhere between 0 and 5V
- So the best guess is 2.5V
- The value is higher than the guess, so the MSB is 1

Successive approximation ADC

After 2nd iteration

Value = 11XX

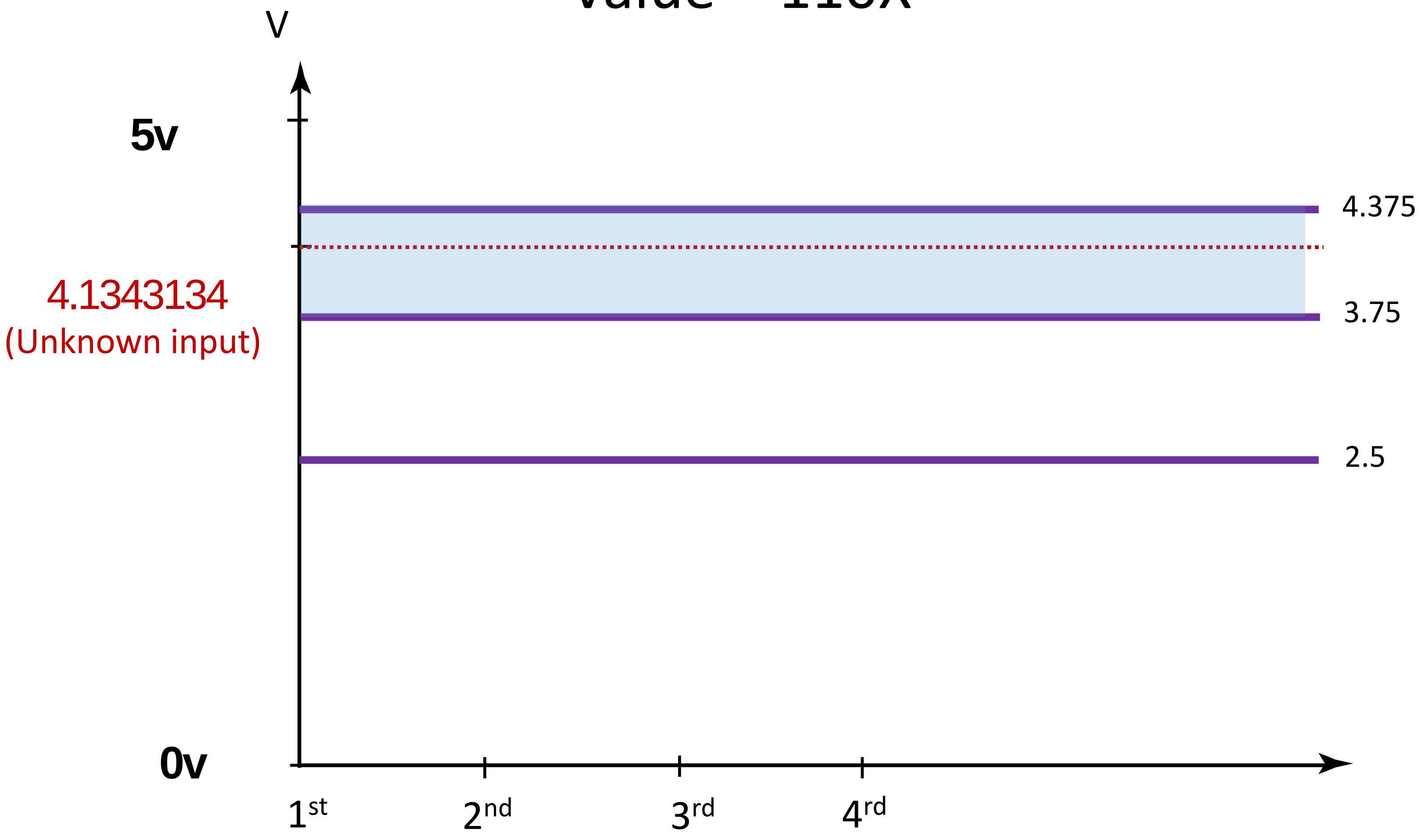


- Now the number is anywhere between 2.5 and 5V.
- So the best guess now is 3.75V
- The value is higher than the guess, so the next bit is 1

Successive approximation ADC

After 3^{rd} iteration

Value = 110X

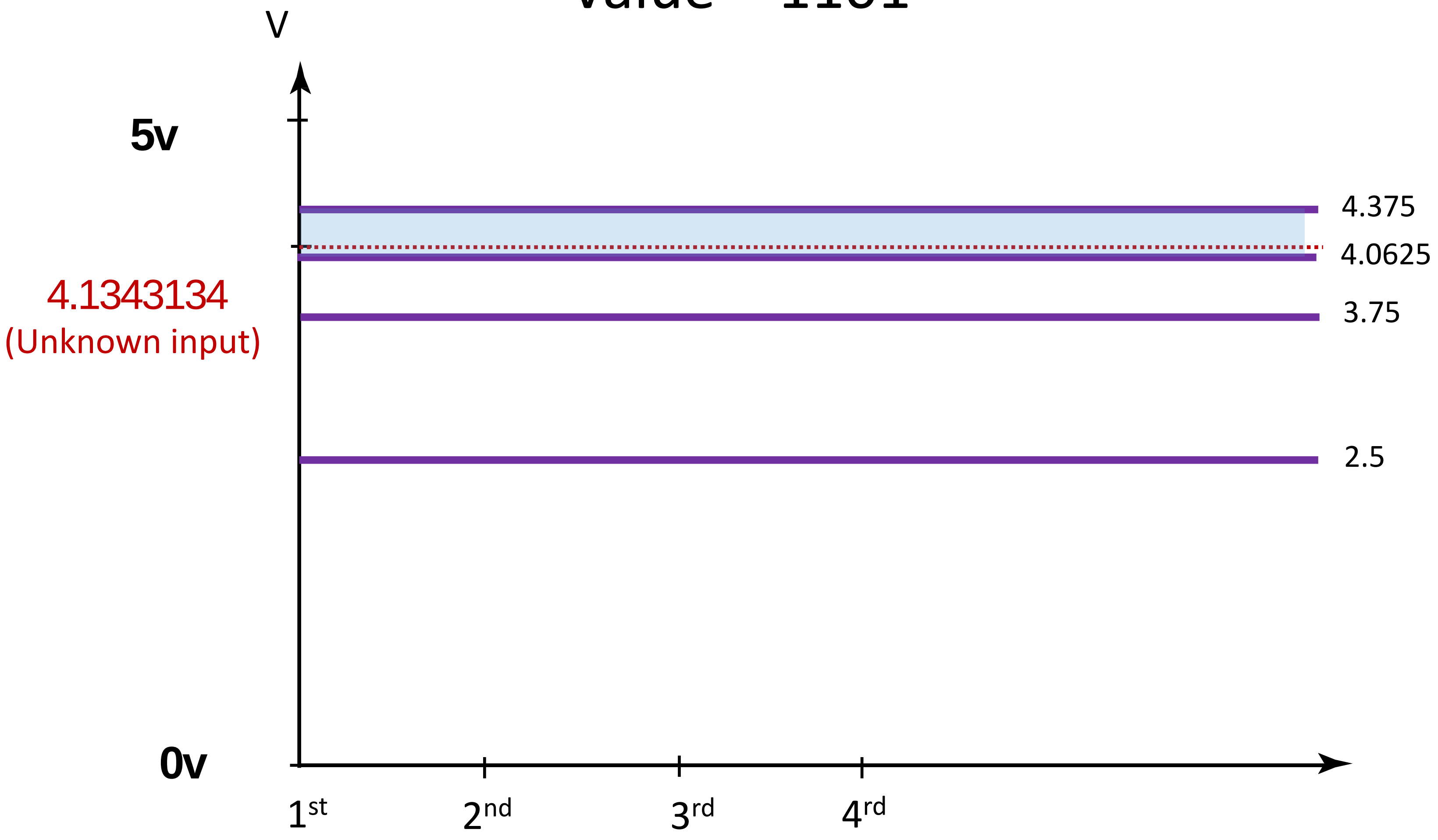


- Now the number is anywhere between 3.75 and 5V.
- So the best guess now is 4.375V.
- The value is lower than the guess, so the next bit is 0

Successive approximation ADC

After 4th iteration

Value = 1101

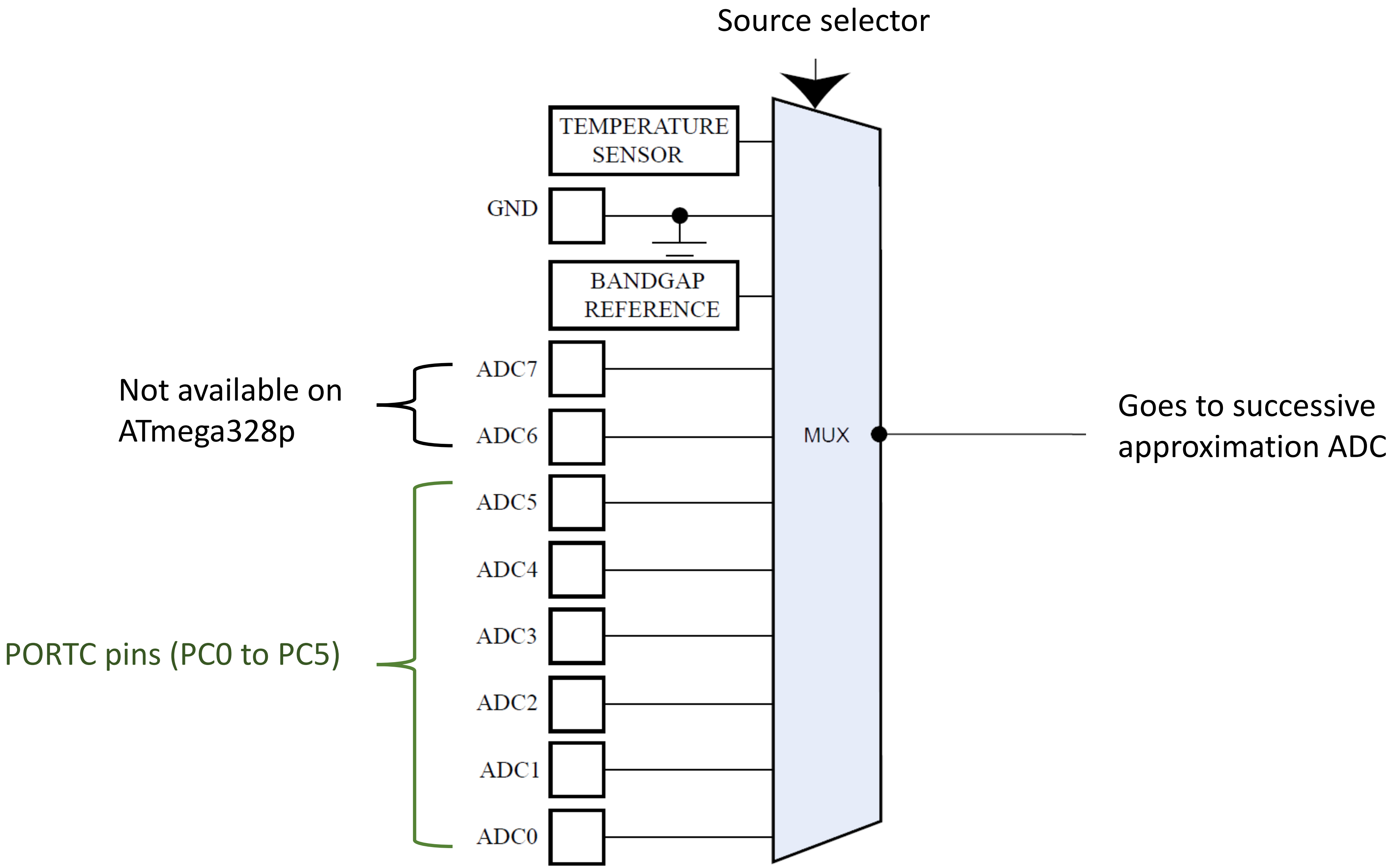


- Now the number is anywhere between 3.75 and 4.375.
- So the best guess now is 4.0625.
- The value is higher than the guess, so the next bit is 1

ADCs on ATmega328p

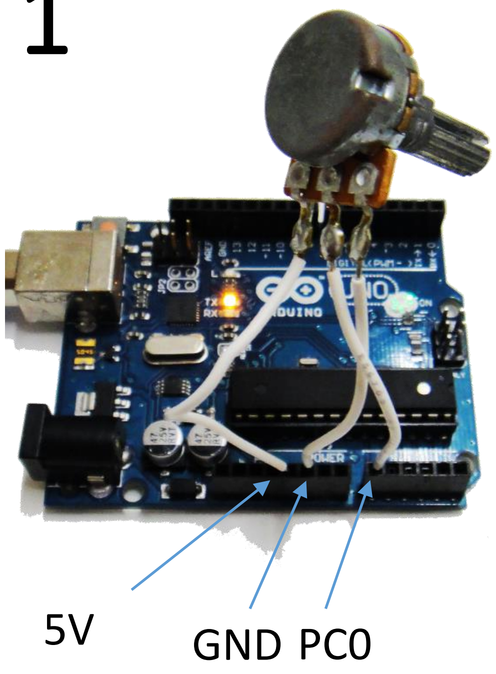
- ATmega328p has **six** 10-bit ADCs on Port C (PC0 to PC5).
- The ADCs use successive approximation technique.
- ADC functionality gets turned on by the init() function of the Arduino library
- Recommended clock rate for ADC is 50kHz to 200kHz
- Upper limit is 1 MHz.
- Each conversion takes 13 ADC clock cycles. (The first one takes 25 ADC clock cycles)
- Only one channel can be converted at a time.

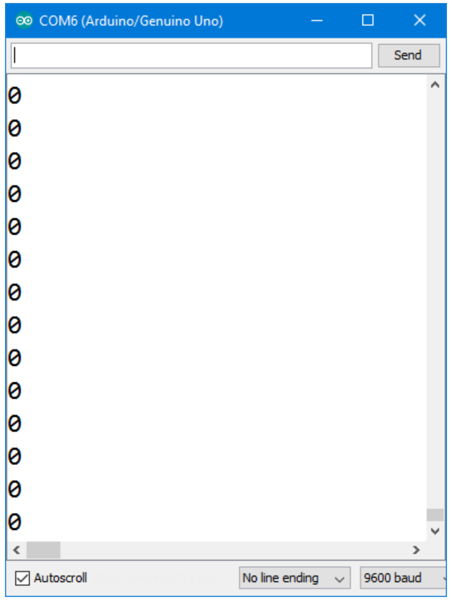
```
#if defined(ADCSRA)
    #if F_CPU >= 16000000
        sbi(ADCSRA, ADPS2);
        sbi(ADCSRA, ADPS1);
        sbi(ADCSRA, ADPS0);
    #elif F_CPU >= 8000000
        sbi(ADCSRA, ADPS2);
        sbi(ADCSRA, ADPS1);
        cbi(ADCSRA, ADPS0);
    #elif F_CPU >= 4000000
        sbi(ADCSRA, ADPS2);
        cbi(ADCSRA, ADPS1);
        sbi(ADCSRA, ADPS0);
    #elif F_CPU >= 2000000
        sbi(ADCSRA, ADPS2);
        cbi(ADCSRA, ADPS1);
        cbi(ADCSRA, ADPS0);
    #elif F_CPU >= 1000000
        cbi(ADCSRA, ADPS2);
        sbi(ADCSRA, ADPS1);
        sbi(ADCSRA, ADPS0);
    #else
        cbi(ADCSRA, ADPS2);
        cbi(ADCSRA, ADPS1);
        sbi(ADCSRA, ADPS0);
    #endif
        sbi(ADCSRA, ADEN);
#endif
```

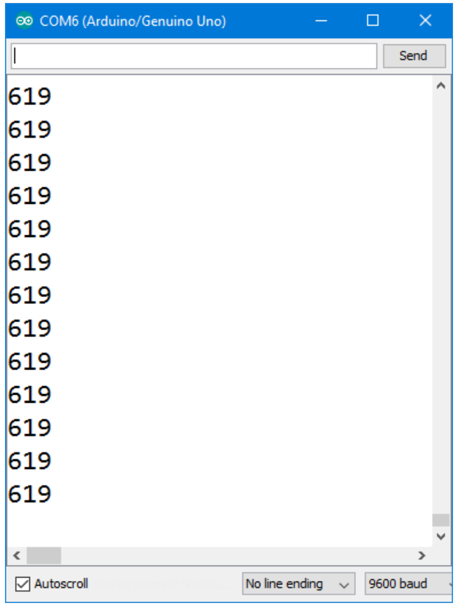

Source selector
TEMPERATURE SENSOR
GND
BANDGAP REFERENCE
Not available on ATmega328p
ADC7
ADC6
MUX
Goes to successive approximation ADC
PORTC pins (PC0 to PC5)
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0

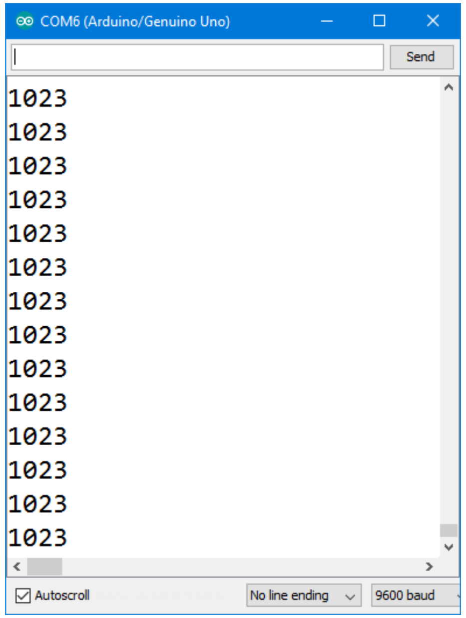
Example 1

The program prints out values from 0 to 1023 (according to the position of the potentiometer)










```
int main()              //To run on Arduino, just change this function to void setup()
{
    Serial.begin(9600);   //For now, we are still using the Serial library

    for (;;)
    {
        int value = analogRead(A0); //int or unsigned int does not matter
        Serial.println(value);
    }
}
```

Bad practice example 1

- If your program involves thresholding the value acquired by the ADC, you do not really need to use the ADC.

- A simple comparator can do the job

```
int main()
{

    for (;;)
    {
        int value = analogRead(A0);
        if (value < 448)
        {
                //Do some stuff
        }
        else
        {
                //Do other stuff
        }
    }
}
```

Bad practice example 2

- Another example of thresholding the value acquired by the ADC into 4 categories.
- 4 comparators can do the job. Two GPIO lines are needed as digital input.

```
int main()
{

    for (;;)
    {
        int value = analogRead(A0);
        if (value < 448)
        {
                //Do some stuff
        }
        else if (value < 741)
        {
                //Do some stuff
        }
        else if (value < 983)
        {
                //Do some stuff
        }
        else
        {
                //Do some stuff
        }
    }
}
```

Bad practice example 3

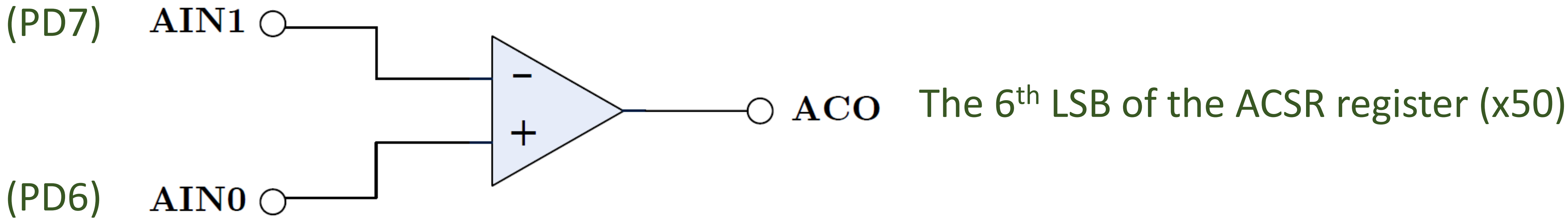
- Example of point-less conversion to double.

```
int main()
{

    for (;;)
    {
        int value = analogRead(A0);
        double voltage = value * (5.0 / 1023.0);
        if (voltage < 2.323)
        {
            //Do some stuff
        }
    }
}
```

Analog comparator on ATmega328p

- ATmega328p has one analog comparator.
- The output of the comparator can trigger an interrupt.
- The output of the comparator can also be read from the ACSR register.



ACSR Register

Analog Comparator Control and Status Register (x50)

Bit	7	6	5	4	3	2	1	0
0x50	ACD	ACBG	ACO	ACIF	ACIE	ACIC	ACIS1	ACIS0
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Default	0	0	-	0	0	0	0	0

Bit	Long form	Function
ACD	Analog Comparator Disable	1 to turn on analog comparator
ACBG	Analog Comparator Bandgap Select	If it is 1, the fixed 1.1V replaces the + input to the comparator.
ACO	Analog Comparator Output	The output of the analog comparator
Remaining		Interrupt related bits. Refer to the datasheet.

ADCSRA

ADC Control and Status Register A (x7A)

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Bit	Long form	Function
ADEN	ADC Enable	1 to turn on ADC circuitry
ADSC	ADC Start Conversion	Set this bit to start ADC conversion. After conversion, it is auto-cleared.
ADATE	ADC Auto Trigger Enable	Set this bit to enable auto trigger (the ADC will start conversion at the PGT of a trigger signal selected in the ADCSRB register)
ADIF,ADIE		Interrupt related bits. Refer to the datasheet.
ADPS	Pre-scaler selector	Clock division factor = 2^{ADPS}

Example

If the clock speed of the system is 16 MHz and ADPS is set to 111, determine the clock speed of ADC.

ADPS	Division Factor (2^{ADPS})
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

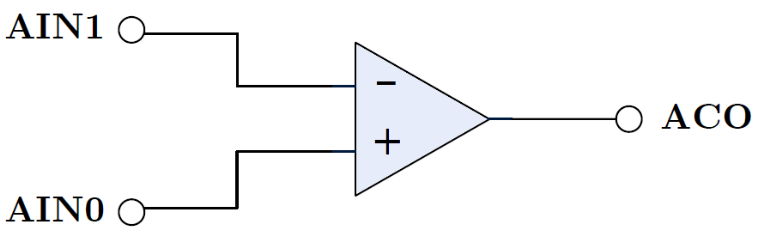
ADC's clock speed = 16Mhz / 128 = 125kHz

This is the speed of ADC set by Arduino in the init() function

This translates to 125kHz/13 ≈ 9615 samples per second

ADCSRB

ADC Control and Status Register B (x7B)



Bit	7	6	5	4	3	2	1	0
0x7B	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Bit	Long form	Function
ACME	Analog Comparator Multiplexer Enable	Normally the negative input to the analog comparator is connected to AIN1 pin. In reality, any input signal from the ADC MUX can act as negative input to the analog comparator (if this bit is set and ADC is disabled)
ADTS	Auto-trigger source	If ADATE (auto-trigger enable) is set in the ADCSRA register These bits select the auto-trigger source If ADATE is cleared These bits have no effect

ADTS	Trigger Source
000	Continuous running mode
001	Triggered by output of analog comparator
010	Triggered by external interrupt 0 (INT0)
011	Triggered by Timer/Counter0 Compare Match A
100	Triggered by Timer/Counter0 Overflow
101	Triggered by Timer/Counter1 Compare Match B
110	Triggered by Timer/Counter1 Overflow
111	Triggered by Timer/Counter1 Capture Event

ADMUX

ADC Multiplexer Selection Register (x7C)

Bit	7	6	5	4	3	2	1	0
0x7C	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Bit	Long form	Function
REFS	Reference Source	These bits select the reference voltage for the ADC.
ADLAR	ADC Left Adjust Result	This bit affects the presentation of the ADC conversion result. Set this bit to left adjust the result.
MUX	Multiplexer	These bits select the input source of the ADC.

MUX	Source
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6 [not available on ATmega328p]
0111	ADC7 [not available on ATmega328p]
1000	Temperature sensor
1001-1101	Reserved
1110	1.1V (V_{BG})
1111	GND

REFS	Reference source
00	AREF (external reference on pin 21)
01	AVCC (power supply for ADC)
10	Reserved
11	Internal 1.1v (band-gap reference)

Note:

01 is the default mode on Arduino Uno.

For mode 01 and 11, it is recommended to place a bypass capacitor at AREF pin. Refer to the datasheet.

ADCH & ADCL

ADC Data Register High byte (x79) and Low byte (x78)

- The result of the analog to digital conversion process (10-bit number) is placed in two registers.
- The placement depends on ADLAR (left adjust flag defined in ADMUX)

Bit	7	6	5	4	3	2	1	0
0x79	-	-	-	-	-	-	ADC9	ADC8
0x78	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Read/Write	R	R	R	R	R	R	R	R
Default	0	0	0	0	0	0	0	0

If ADLAR = 0

Bit	7	6	5	4	3	2	1	0
0x79	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
0x78	ADC1	ADC0	-	-	-	-	-	-
Read/Write	R	R	R	R	R	R	R	R
Default	0	0	0	0	0	0	0	0

If ADLAR = 1

DIR0

Digital Input Disable Register 0 (x7E)

Bit	7	6	5	4	3	2	1	0
0x7E	-	-	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- When the respective bits are set, the digital input buffer on the corresponding ADC pin gets disabled.
- The corresponding PIN Register bit will always read as 0 when this bit is set.
- When an analog signal is applied to the ADC7...0 pin and the digital input from this pin is not needed, this bit should be set to reduce power consumption in the digital input buffer.

DIR1

Digital Input Disable Register 1 (x7F)

Bit	7	6	5	4	3	2	1	0
0x7F	-	-	-	-	-	-	AIN1D	AIN0D
Read/Write	R	R	R	R	R	R	R/W	R/W
Default	0	0	0	0	0	0	0	0

- When this bit is set, the digital input buffer on the AIN1/AIN0 pin gets disabled.
- The corresponding PIN Register bit will always read as zero when this bit is set.
- When an analog signal is applied to the AIN1/AIN0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

Two modes of ADC

- **On-demand mode** (default and most widely used)

  Auto-trigged disabled (ADATE=0)

  On conversion at a time

  Arduino uses this mode

- **Auto-trigger mode**

  Auto-triggered enabled (ADATE=1)

  Trigger source can be selected.

  Trigger source (ADTS) = 000 (default) (the most common trigger source)

  When ADTS = 000, As soon as one conversion gets completed, another conversion automatically gets carried out

On-demand mode

Procedure

- Set the desired channel and analog reference voltage in the ADMUX register.
- Set ADCSRA = 1100 0XXX to start the conversion process. (XXX = desired clock rate)
- Wait until the ADSC bit is back to low (which denotes that the conversion process is complete) Or use *ADC-complete interrupt*.
- Read the low byte of the ADC data register, followed by the high byte
- Merge high byte & low byte

Note:

After turning on ADC circuitry for the first time, it is good to wait a while before ordering to perform conversion

Set this bit to start conversion

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Example 2

Write a program that continuously prints out the analog value of PC3 through the serial port. Assume the followings:

- The clock speed of the microcontroller is 16MHz
- The desired clock speed of ADC is 125 kHz.
- Use AVCC as analog reference.
- On-demand mode

Desired pre-scaler = 16 MHz / 125kHz = 128	ADPS = 111
Analog reference = AVCC	REFS = 01
Analog source = PC1	MUX = 0011

```
char *admux = (char*) 0x7C;      //Signed/unsigned does not matter if you don't do decimal work
volatile unsigned char *adcsra = (unsigned char*) 0x7A;
volatile unsigned char *adch = (unsigned char*) 0x79;
volatile unsigned char *adcl = (unsigned char*) 0x78;
```

The volatile keyword makes sure that the CPU supplies the direct values from the registers instead of copies from other parts of the main memory

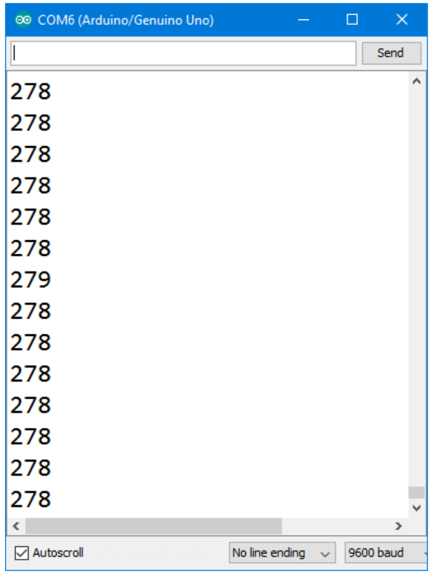
```
int main()               //To run on Arduino, just change this function to void setup()
{
    *admux = 0b01000011;    //Set analog reference to AVCC and analog source to PC3.
    *adcsra = 0b11000111;   //Enable ADC, start conversion and set pre-scaler to 128
    Serial.begin(9600);

    while(1)
    {
        *adcsra |= 0b01000000;    //Start conversion (Set ADSC bit)

        while (((*adcsra) & 0b01000000)) //As long as ADSC bit is HIGH
        {
                //Do nothing
        }

        int lowbyte  = (*adcl);  //Capture low byte
        int highbyte = (*adch);  //Capture high byte
        int value = (highbyte<<8) | lowbyte;   //Merge high byte and low byte
        Serial.println(value);
    }
}
```

Note, we let conversion start early because the first conversion is slow



Auto-trigger mode

Procedure

- Set the desired channel and analog reference voltage in the ADMUX register.
- Set ADCSRA = 1110 0YYY to start the conversion process. (XXX = desired clock rate)
- Read the low byte of the ADC data register, followed by the high byte
- Merge high byte & low byte

Set this bit to start conversion

Note:

After turning on ADC circuitry for the first time, it is good to wait a while before ordering to perform conversion

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Example 3

Modify example 2 to put the ADC in continuously running mode (auto-trigger)

```
char *admux = (char*) 0x7C;
char *adcsra = (char*) 0x7A;
volatile unsigned char *adch = (unsigned char*) 0x79;
volatile unsigned char *adcl = (unsigned char*) 0x78;


int main()           //To run on Arduino, just change this function to void setup()
{
    *admux = 0b01000011; //Set analog reference to AVCC and analog source to PC3.
    *adcsra = 0b11100111; //Enable ADC, start conversion, enable auto-triggering
                          // and set pre-scaler to 128
    Serial.begin(9600);

    while(1)
    {
        int lowbyte  = (*adcl); //Capture low byte
        int highbyte = (*adch); //Capture high byte

        int value = (highbyte << 8) | lowbyte; //Merge high byte and low byte
        Serial.println(value);
    }
}
```

- Auto-triggering works best if there is just one analog channel to be measured.
- Capturing different MUX channels with auto-triggering is a bit tricky.