The background is a close-up, slightly blurred image of a green printed circuit board (PCB). A large, square, gold-colored microchip is the central focus, with its pins visible. Other components like capacitors and smaller chips are scattered around it. The overall color palette is dominated by the green of the PCB and the gold of the chip, with a semi-transparent blue overlay across the entire image.

**MCT 4334**

# **Embedded System Design**

**Week 09 Power Management**

# Outline

- Current consumption and batteries
- Reducing power consumption in hardware
- Reducing power consumption in software

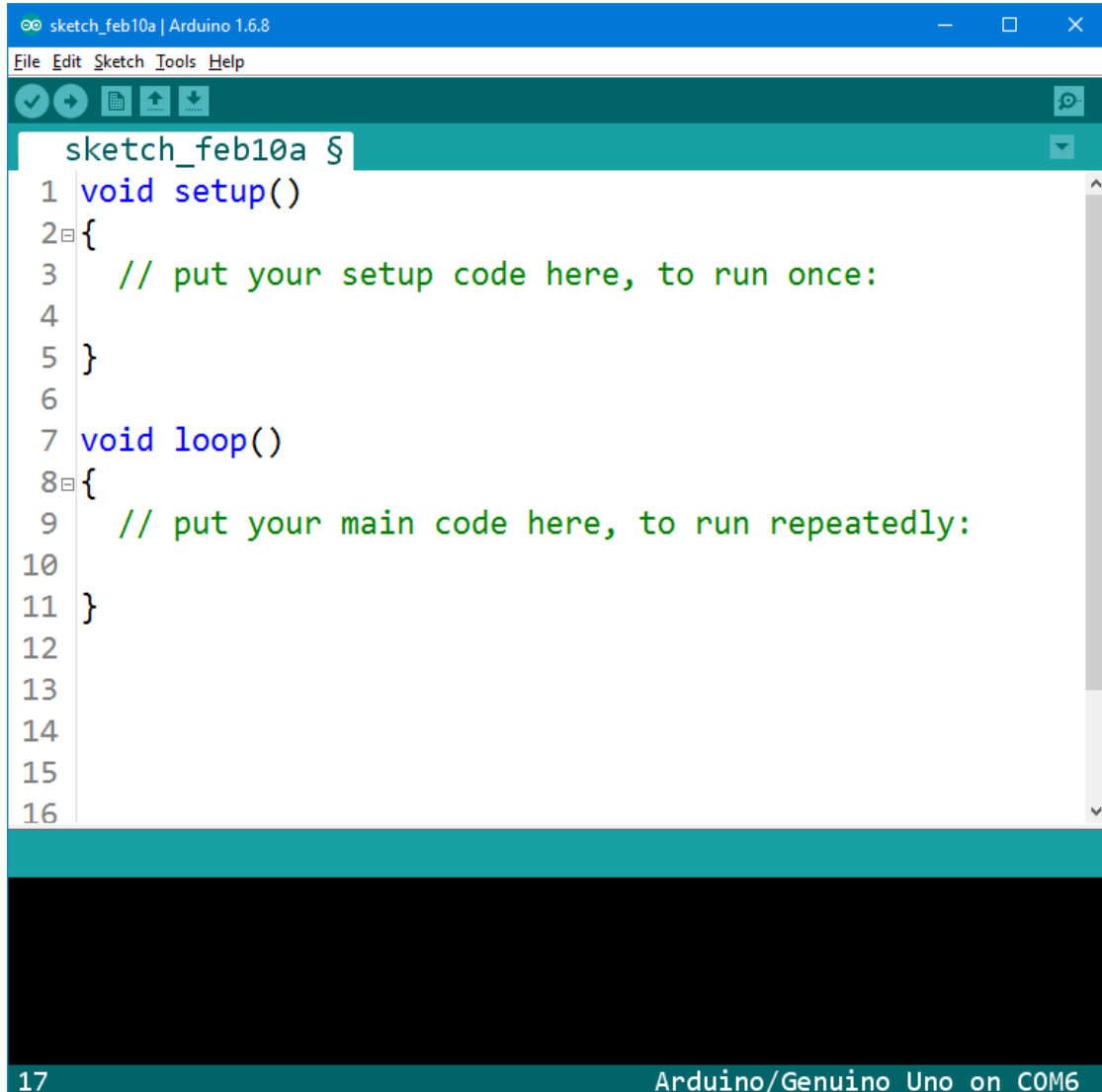
# Current Consumption

- The SI unit of current is C/s.
- Batteries generate flow of charge. The capacity of a battery is often expressed in either mAh or Ah.

## Example

If a circuit draws 1A of current from a battery rated 1 Ah, the battery will last a maximum of 1 hour.

# Current Consumption of Arduino



The screenshot shows the Arduino IDE interface. The title bar reads "sketch\_feb10a | Arduino 1.6.8". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, running, and other functions. The main text area contains the following code:

```
1 void setup()
2 {
3   // put your setup code here, to run once:
4
5 }
6
7 void loop()
8 {
9   // put your main code here, to run repeatedly:
10
11 }
12
13
14
15
16
```

At the bottom of the IDE, there is a status bar that reads "17" on the left and "Arduino/Genuino Uno on COM6" on the right.

- The simplest Arduino program
- It does nothing (trapped in an endless loop)

**How much current does it draw?**

When external 5V supply  
connected to the 5V pin of  
Arduino



When external 9V supply  
connected to the Vin pin of  
Arduino



Note: the amount of current drawn will differ from  
board to board and environment to environment.

# Example

How long will a blank Arduino last on a 9V battery with capacity 600mAh?

Maximum 11.52 hours

In reality, much less than that.



# Reducing power consumption

There are two ways to reduce power consumption

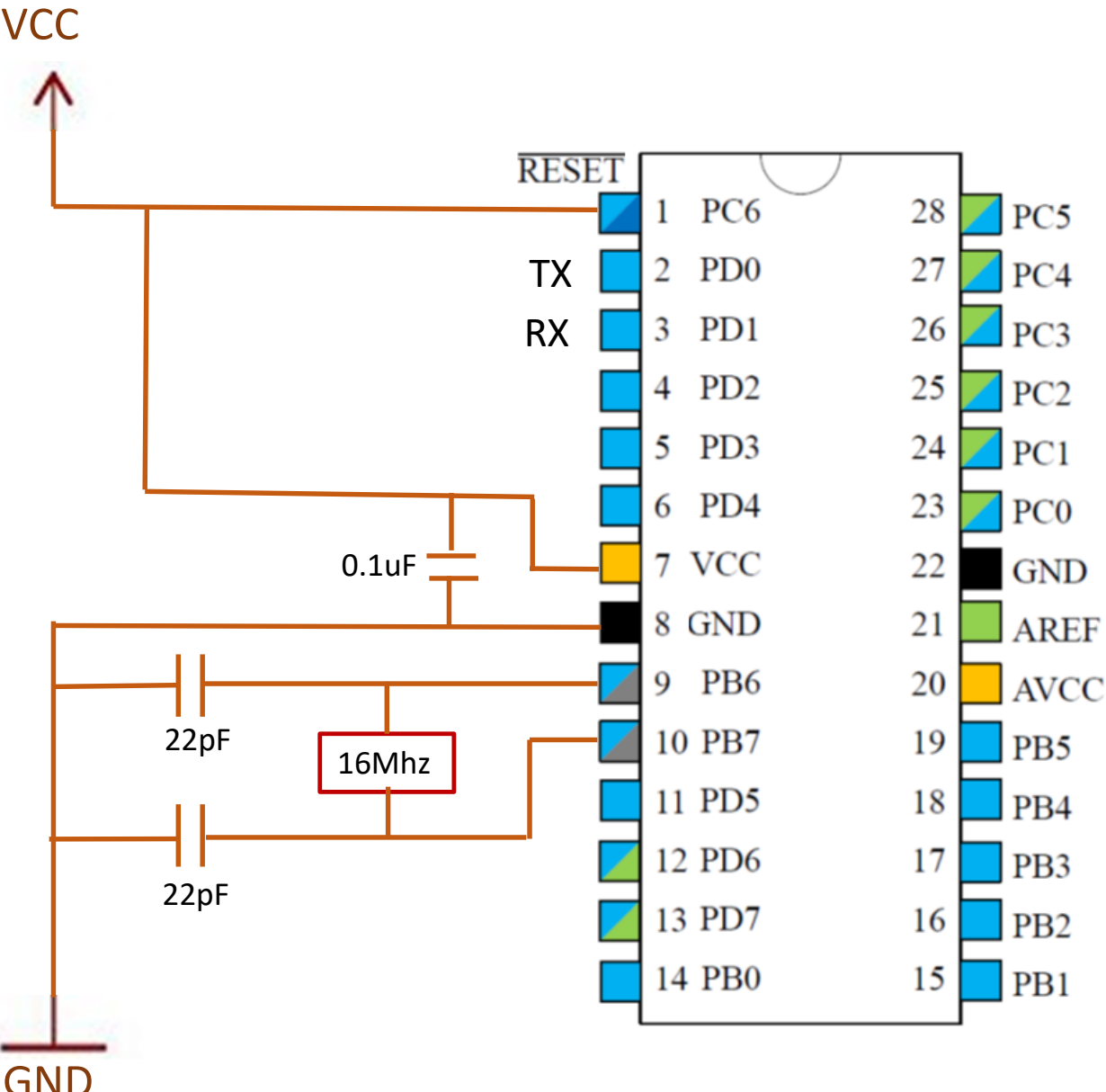
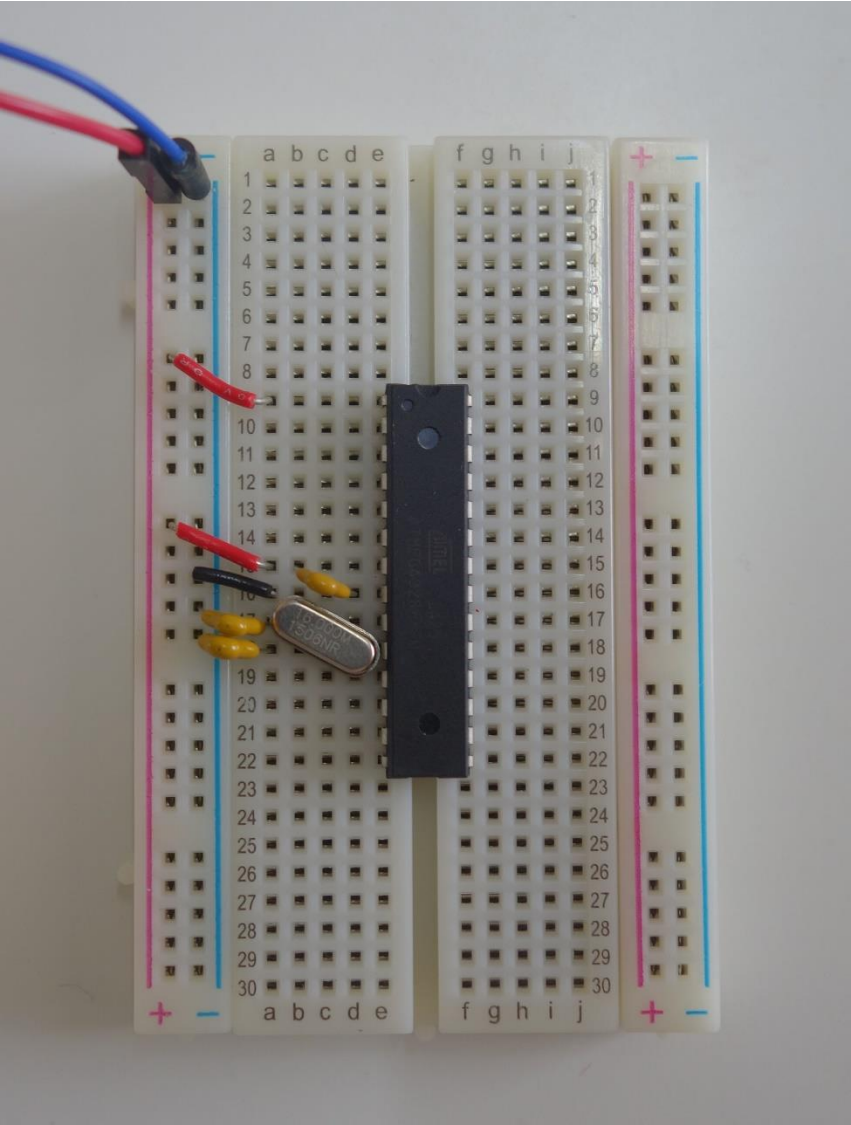
- Hardware-based
- Software-based

# Hardware-based techniques

- Take the ATmega328p out of Arduino (Easiest of all)
- Use efficient voltage regulators (or run directly from batteries if possible)
- Lower the CPU clock speed
- Lower the operating voltage



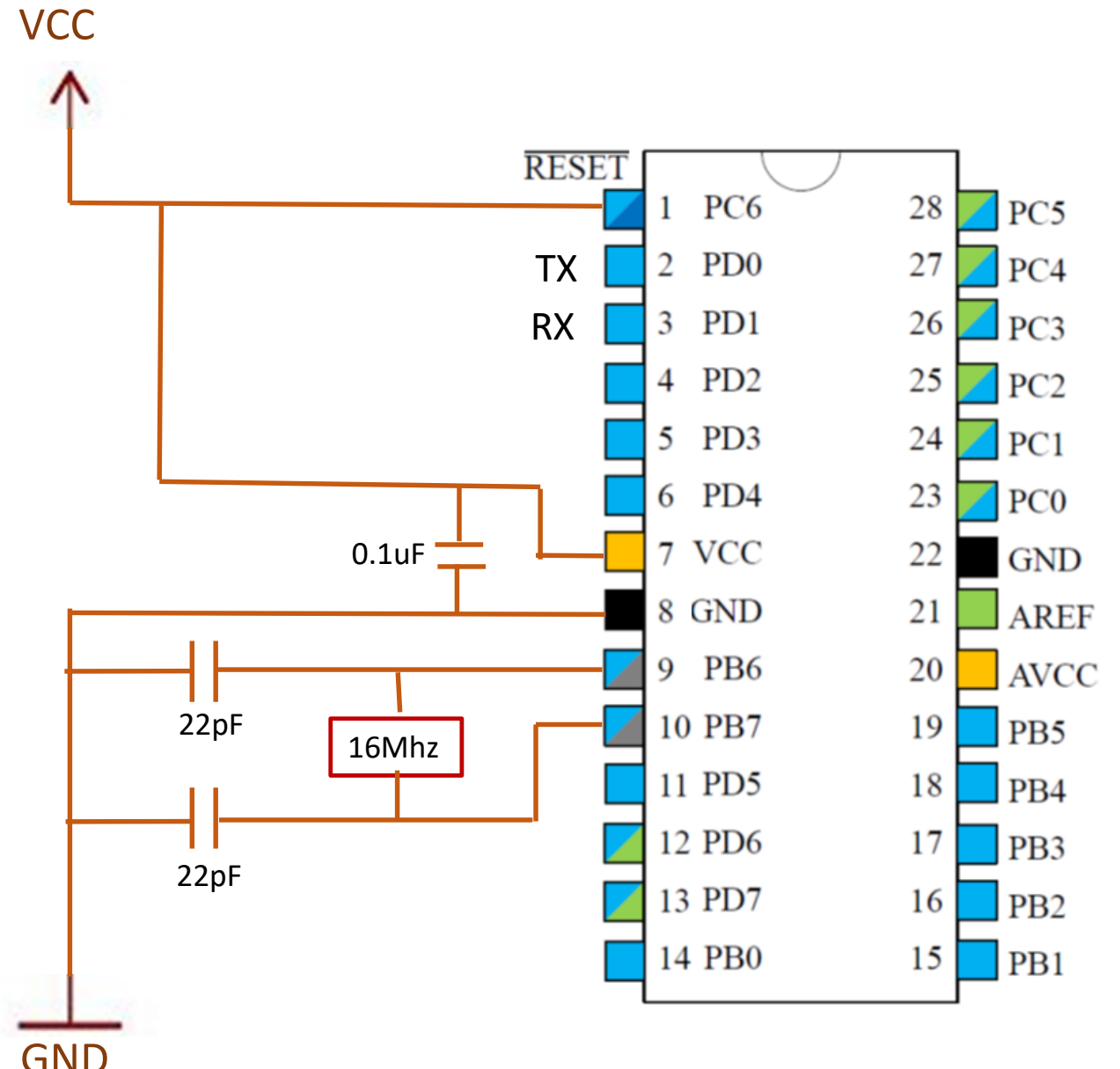
# ATmega328p out of Arduino (Standalone)



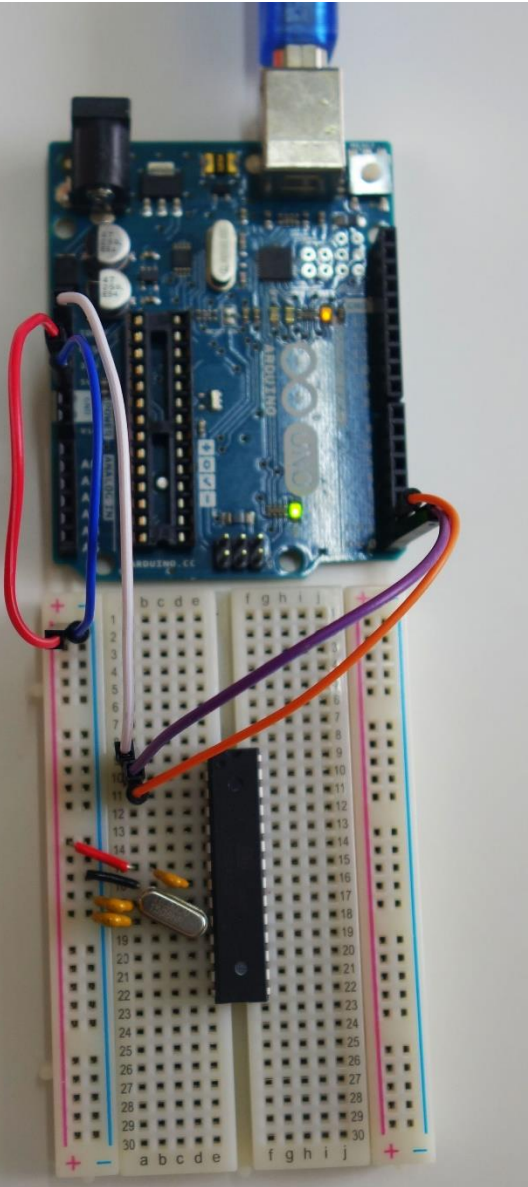
# ATmega328p out of Arduino (Standalone)



- PC6 is the  $\overline{\text{RESET}}$  PIN. It is active low. It must be tied HIGH to prevent resetting.
- A bypass capacitor (0.1uF) should be placed across VCC and GND.
- ATmega328p has 8MHz and 128kHz internal oscillators. External clock crystal is not necessary. However, the built-in boot-loader only works with 16MHz frequency.
- Connect the ADC power pins if you need to use ADC.



# Programming Standalone ATmega328p



- Stand-alone ATmega328p can be programmed using USB to serial converter or FTDI breakouts.
- Arduino can still be used to program standalone ATmega328p.

How?

- Connect the **RX** pin of ATmega328p to the **RX** pin of Arduino.
- Connect the **TX** pin of ATmega328p to the **TX** pin of Arduino
- Connect the **RESET** pin of ATmega328p to the **RESET** pin of Arduino.  
(Previously **RESET** was tied HIGH)

# Power consumption of Standalone ATmega328p

5V @ 16MHz

Power consumption dropped to around 11.57mA (without using a voltage regulator).  
(4 times less than Arduino)



## Reasons:

- Arduino uses a linear voltage regulator. Linear voltage regulators are known to be very inefficient (Around 70% efficiency)
- Arduino has onboard LEDs that consume power.
- Arduino has ATmega16u that handles USB-Serial conversion. It consumes power.



# Power consumption of Standalone ATmega328p

5V @ 16MHz

4.5V @ 16MHz

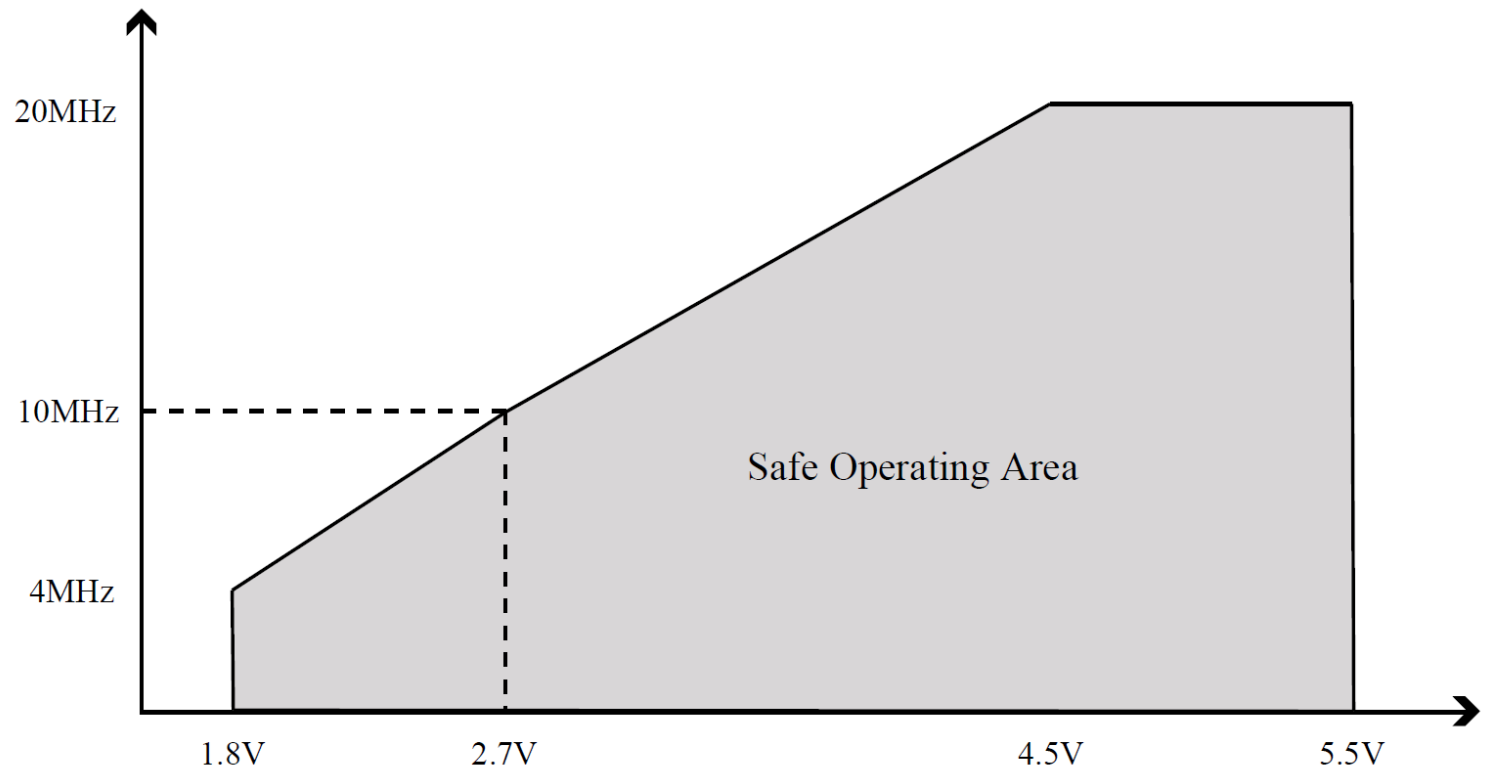
4.0V @ 16MHz

3.5V @ 16MHz



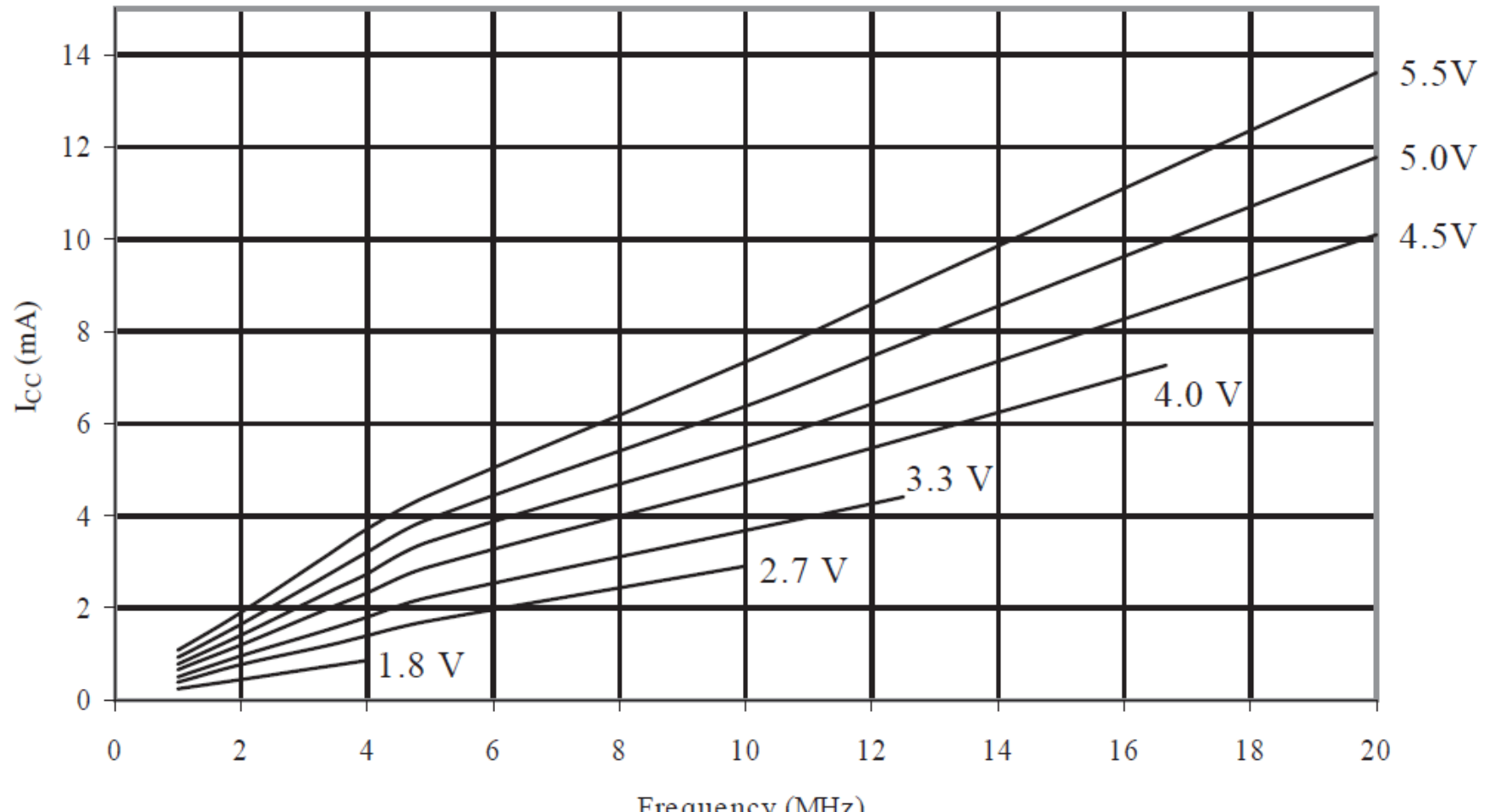
# Lowering frequency and voltage

- Reducing the operating voltage without reducing clock speed appropriately could make the CPU behave erratically.
- Reduce both clock speed and operating voltage to lower current consumption.



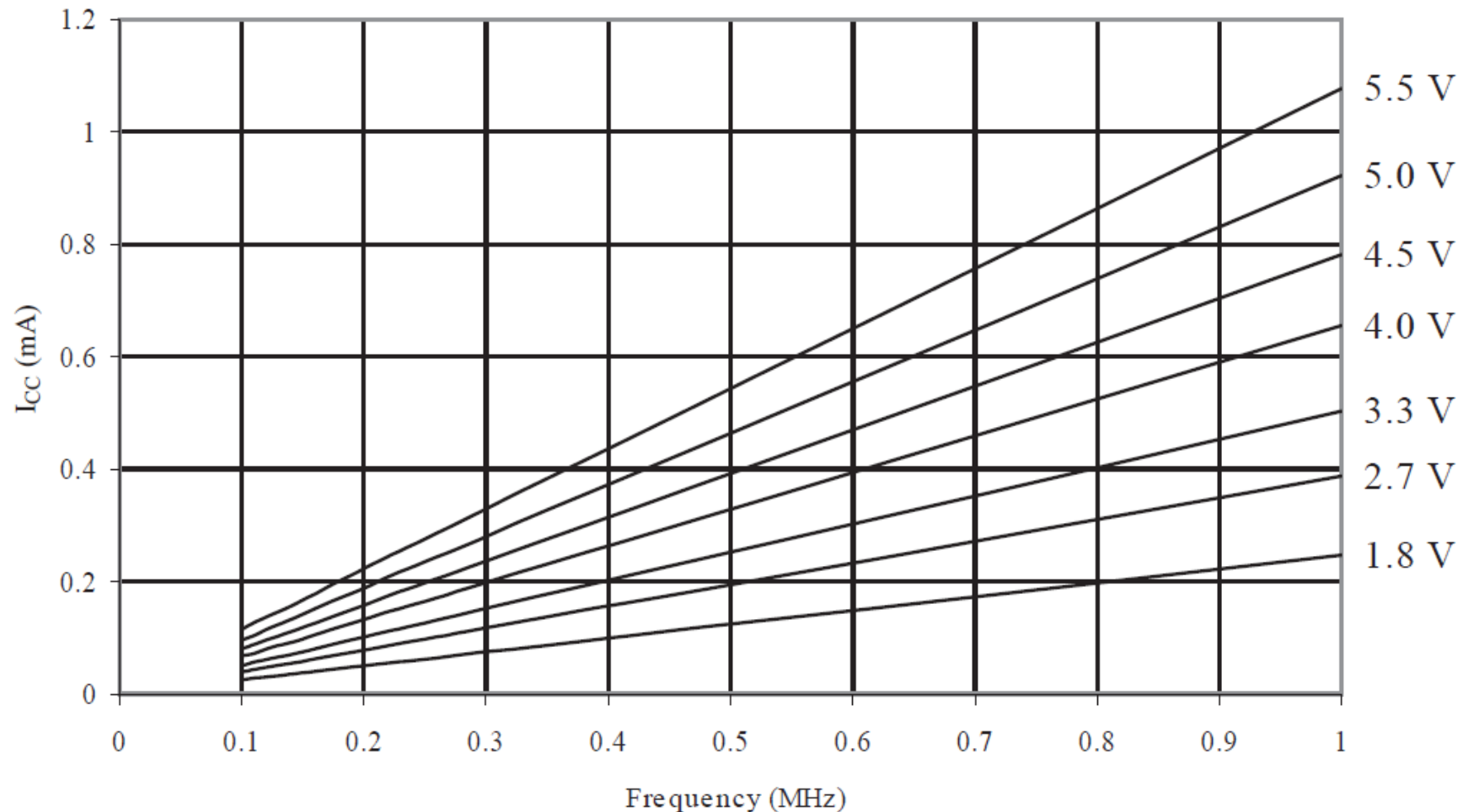
# Current Consumption (Active CPU)

Current consumption of ATmega328p for varying clock speeds (from 1 MHz to 20 MHz)



# Current Consumption (Active CPU)

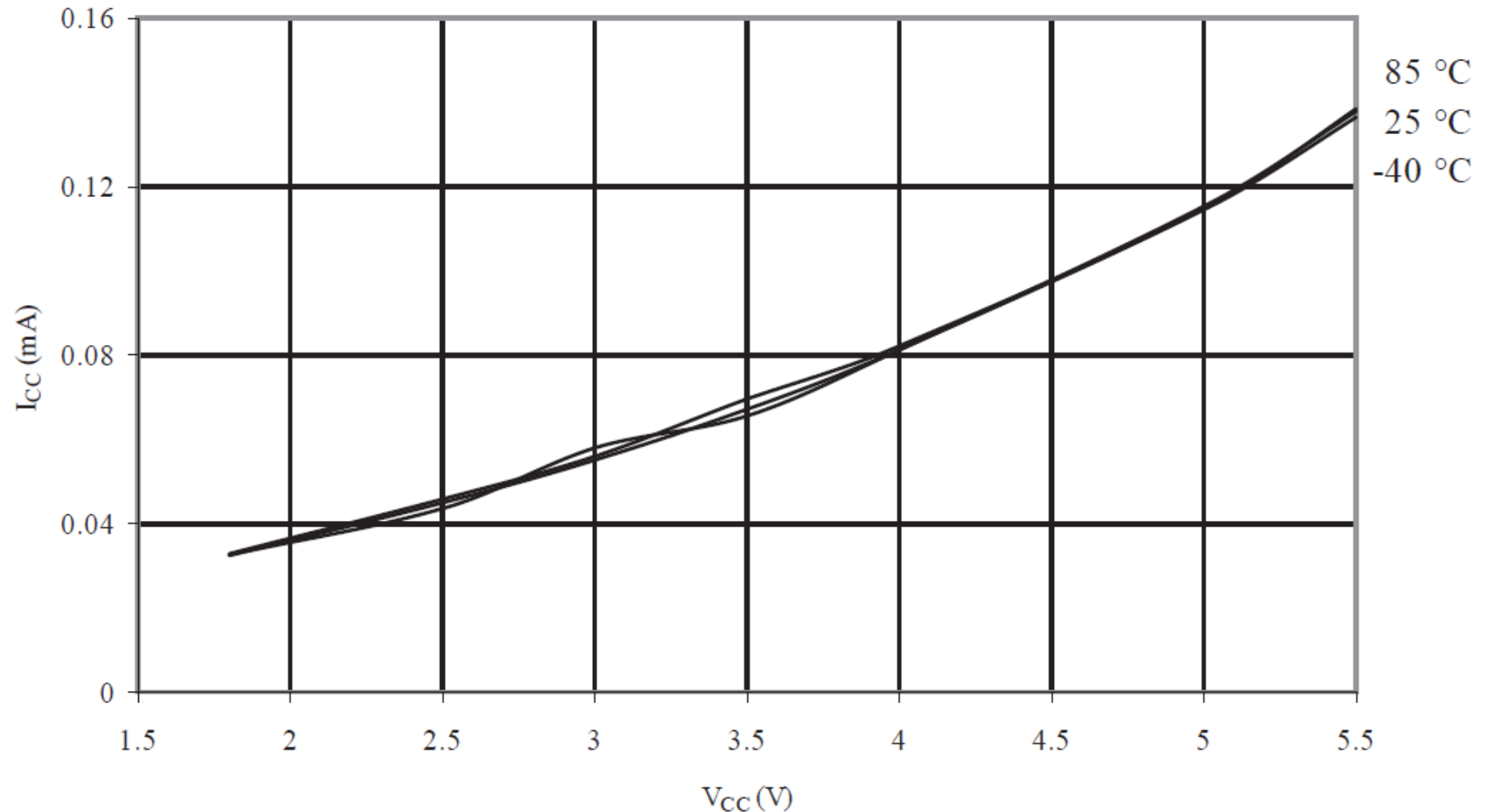
Current consumption of ATmega328p for clock speeds less than 1 MHz





# Current Consumption (Active CPU)

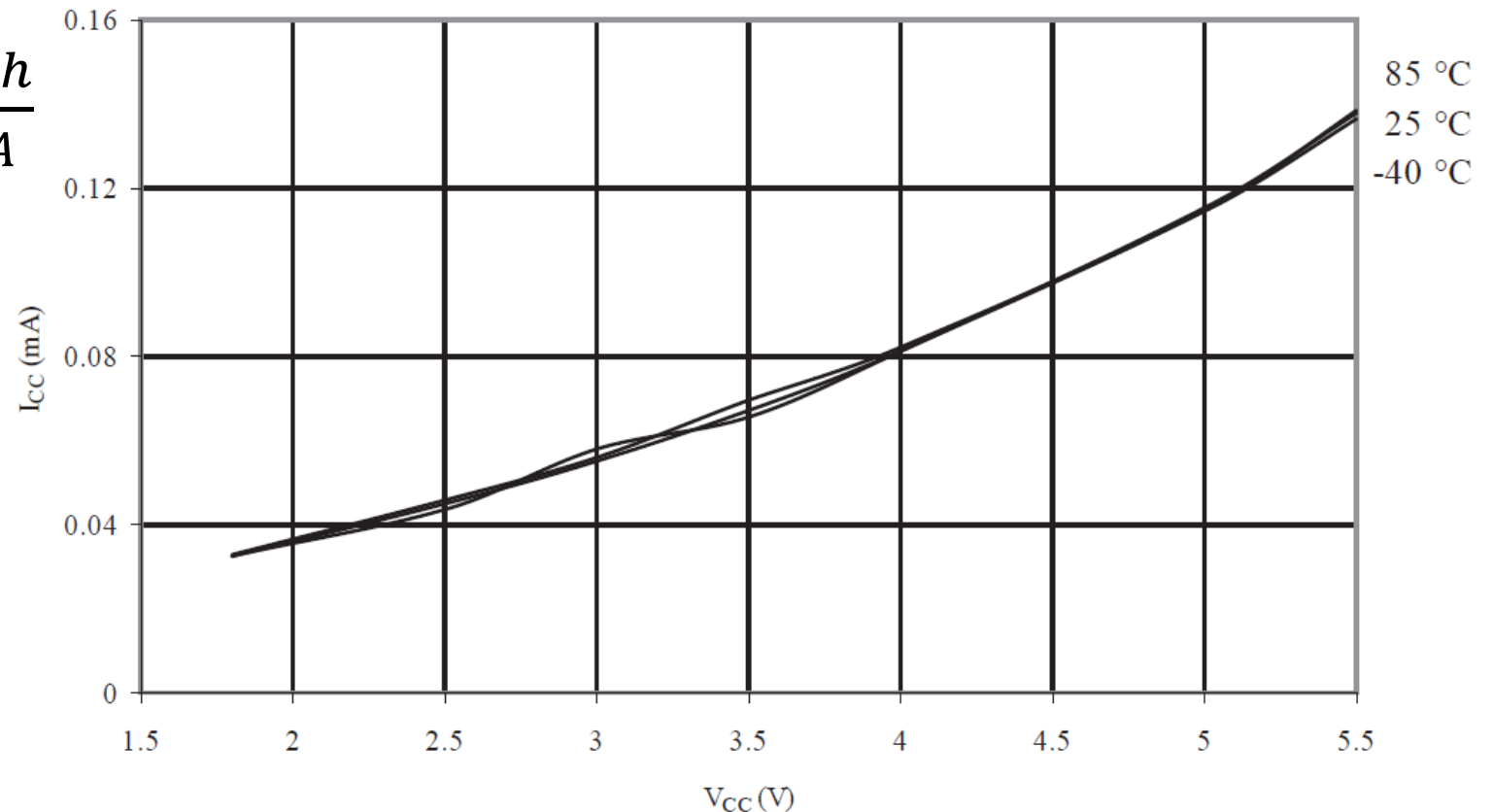
Current consumption of ATmega328p using built-in 128KHz low-power oscillator



# Example

Suppose that the operating voltage of an ATmega328p is 4V. It is clocked by the internal 128KHz low-power oscillator. Determine the maximum number of days the microcontroller would last if it is powered by a battery with a capacity of 600mAh. Assume the CPU is always active and no external peripherals are attached to the microcontroller.

$$\begin{aligned}\text{Maximum number of hours} &= \frac{600 \text{ mAh}}{0.08 \text{ mA}} \\ &= 7,500 \text{ hours or 312 days}\end{aligned}$$



# Software-based Techniques

- Turning off external devices while they are not being used
- Setting unused GPIO pins as output
- Turning off ADC
- Turning off analog comparator
- Sleeping
- Turning off brown-out detector
- Turning off timers

# Setting unused pins as output

```
int main()  
{  
    char *ddrb = (char *)0x24;  
    char *ddrc = (char *)0x27;  
    char *ddrd = (char *)0x2A;  
  
    *ddrb = 0b11111111;  
    *ddrc = 0b11111111;  
    *ddrd = 0b11111111;  
  
    for(;;)  
    {  
  
    }  
}
```

Current consumption: 10.60mA  
(5V @ 16MHz)

A slight improvement.



# Turning off ADC

```
int main()  
{  
    char *ddrb = (char *)0x24;  
    char *ddrc = (char *)0x27;  
    char *ddrd = (char *)0x2A;  
    char *adcsra = (char *)0x7A;  
  
    *ddrb = 0b11111111;  
    *ddrc = 0b11111111;  
    *ddrd = 0b11111111;  
    *adcsra = 0;  
  
    for(;;)  
    {  
  
    }  
}
```

Current consumption: 10.48mA  
(5V @ 16MHz)



# Turning off Analog Comparator

```
int main()  
{  
    char *ddrb = (char *)0x24;  
    char *ddrc = (char *)0x27;  
    char *ddrd = (char *)0x2A;  
    char *adcsra = (char *)0x7A;  
    char *acsr = (char *)0x50;  
  
    *ddrb = 0b11111111;  
    *ddrc = 0b11111111;  
    *ddrd = 0b11111111;  
    *adcsra = 0;  
    *acsr = 1 << 7;  
  
    for(;;)  
    {  
  
    }  
}
```

Current consumption: 10.14mA  
(5V @ 16MHz)



# SMCR

## Sleep Mode Control Register (x53)

Bit	7	6	5	4	3	2	1	0
0x53	-	-	-	-	SM2	SM1	SM0	SE
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

SM	Sleep mode
000	Idle
001	ADC noise reduction
010	Power-down
011	Power-save
100	-
101	-
110	Standby
111	Extended standby

SE = Sleep Enable

Set SE to enable sleeping

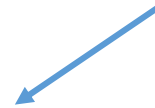
The **power-down** mode is the most power-efficient mode.

## Two-step Procedure for sleeping (power-down mode)

1. Set SMCR = 0101 (Mode = power-down and sleep enabled = true)
2. Call the “**sleep**” instruction. The **sleep** instruction is one of the 131 instructions in the RISC instruction set of ATmega328p. Assembly commands can be invoked through inline assembler in C.

```
asm( "sleep" );
```

Name of assembly instruction to be executed.





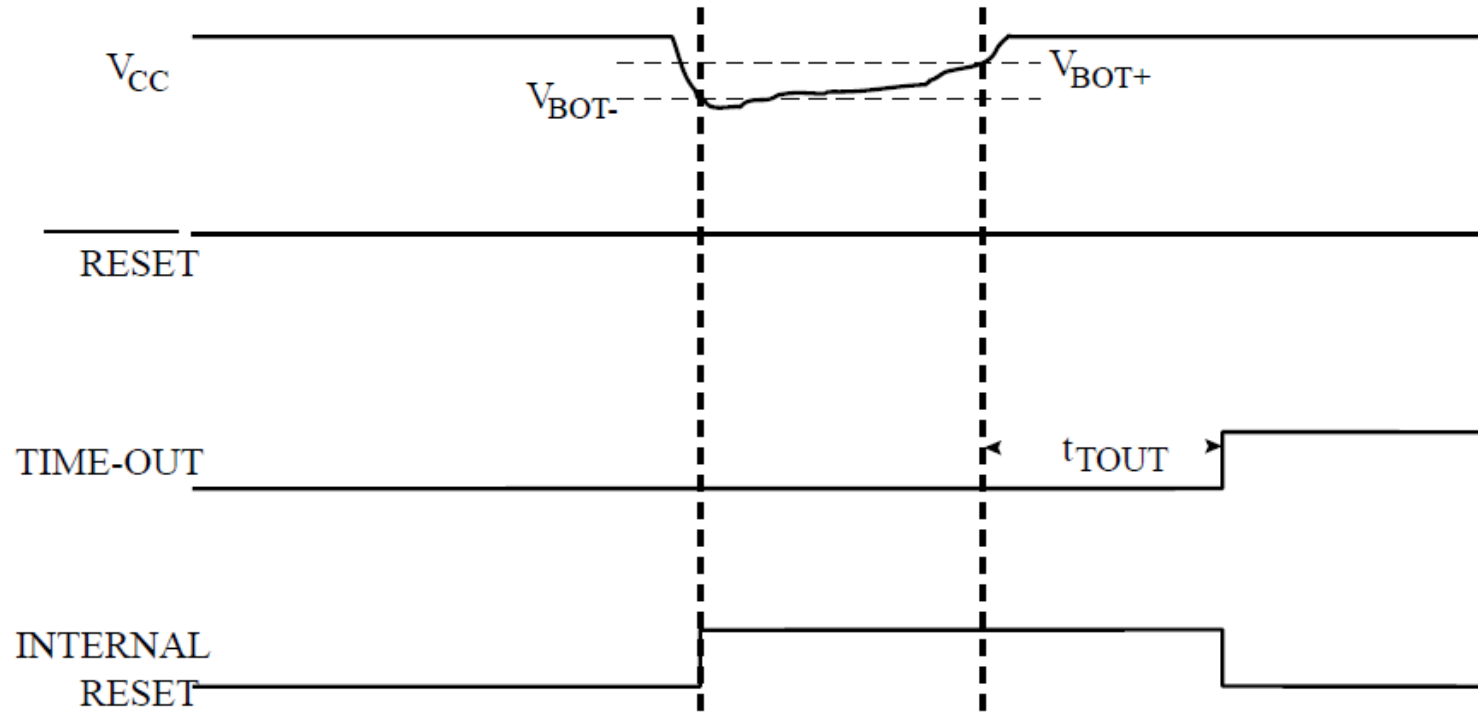
# Sleeping

```
int main()  
{  
    char *adcsra = (char *)0x7A;  
    char *acsr = (char *)0x50;  
  
    *adcsra = 0;  
    *acsr = 1 << 7;  
  
    Sleep();  
  
    for(;;)  
    {  
  
    }  
}  
  
void Sleep()  
{  
    char *smcr = (char *)0x53;  
  
    *smcr = 5;    //Sleep mode = POWER DOWN, Sleep Enable = True  
    asm("sleep"); //Invoke in line assembler to sleep  
}
```

Current consumption: 59.0  $\mu\text{A}$



# Brown-out Detection



A brown out is a short dip in power supply.

ATmega328p has an on-chip Brown-out detection circuit that resets the microcontroller when the power supply falls below a specific level.

Brown-out detector can be disabled during sleep to conserve power.

# MCUCR

MCU Control Register (x55)

Bit	7	6	5	4	3	2	1	0
0x55	-	BODS	BODSE	PUD	-	-	IVSEL	IVCE
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W
Default	0	0	0	0	0	0	0	0

**BODS:** BOD Sleep

**BODSE:** BOD Sleep Enable

Other bits: Out of the scope of this course. Refer to the datasheet.

To disable BOD during sleep,

- BODS and BODSE must be simultaneously set within a single instruction. (11)
- BODS must be set again and BODSE must be cleared simultaneously in the next instruction. (10)

# Sleeping with BOD disabled

```
int main()  
{  
    char *adcsra = (char *)0x7A;  
    char *acsr = (char *)0x50;  
  
    *adcsra = 0;  
    *acsr = 1 << 7;  
  
    DeepSleep();  
  
    for(;;)  
    {  
    }  
}  
  
void DeepSleep()  
{  
    char *smcr = (char *)0x53;  
    volatile char *mcucr = (char *) 0x55;  
    *smcr = 5;    //Sleep mode = POWER DOWN, Sleep Enable = True  
    *mcucr = 0b01100000;    //Set Both BODS and BODSE  
    *mcucr = 0b01000000;    //Set BODS and clear BODSE  
    asm("sleep");    //Invoke in line assembler to sleep  
}
```

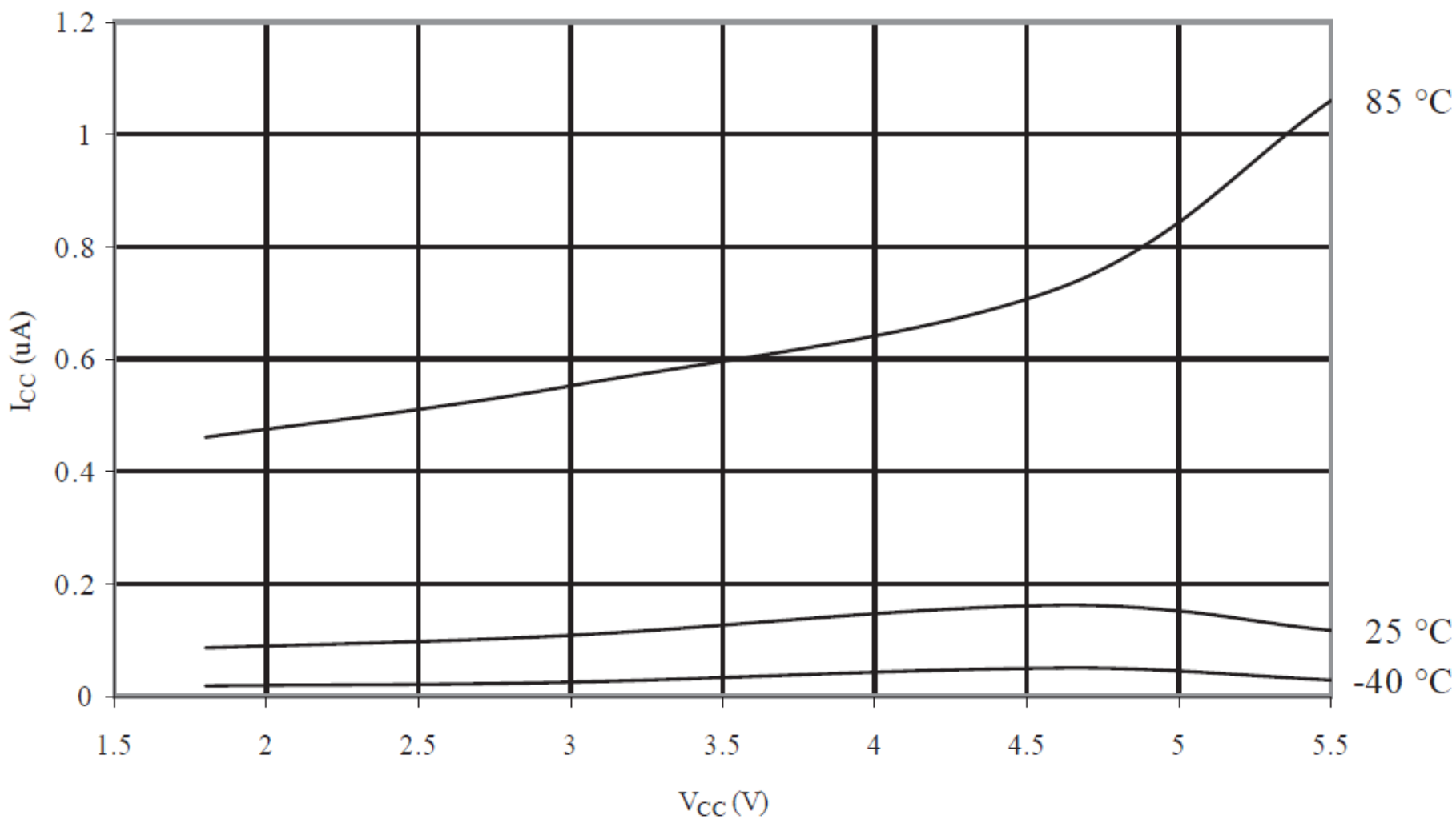
Current consumption: 400nA

(Near the limit of the multi-meter)



# Power-down Current Consumption

Since the CPU is sleeping, the current consumption does not depend on clock frequency.



Current consumption depends on voltage level and the environment in general.

# Example

If the power-down current consumption of ATmega328p is 400nA, how long will it last on a 5V supply with a capacity of 600mAH (in the power-down mode)?

Maximum 1,500,000 hours or 171 years!

The current consumption is much less than the self-discharge rate of typical batteries.

In reality, the MCU cannot survive longer than the shelf lives of batteries (for example: around 5 years for alkaline batteries)

# Example

The ATmega328p is part of an embedded system. The microcontroller endlessly wakes up from sleep every **8 seconds** using a watch-dog timer, performs some operations for **1 second** and then goes back to sleep.

The current consumption of the system while the CPU is sleeping is **50uA**.

The current consumption of the system while the system is in operation is **10mA**.

How long will the system last on a power supply with a capacity of 600mAh?

$$\text{Average current consumption} = \frac{50u \times 8 + 10m \times 1}{9} = 1.1\bar{5} \text{ mA.}$$

$$\text{Maximum duration} = \frac{600}{1.1\bar{5}} = 159.23 \text{ hours}$$

# More extreme way of power saving

- Cut-off  $V_{cc}$  to the microcontroller when the microcontroller does not have anything to do.
- Give power back either periodically using a timer IC or in response to a change in sensor value (using a latching circuit).
- This eliminates the need to configure interrupts.



# Case Study 1

Design a power-efficient embedded system for two-channel RF key fob.

When the **red** button is pressed, call a function **TransmitChannel1()**

When the **green** button is pressed, call a function **TransmitChannel2()**

Assume that these functions have already been written.



Tip:

Sleep to conserve power!

You can use external interrupts int0 (**red button**) and int1 (**green button**) to wake up the CPU.

```
int main() //Just change this to void setup() for Arduino
```

```
{  
    DisableADCandComparator();  
    EnableExternalInterrupts();  
  
    for(;;)  
    {  
        DeepSleep();  
  
        char* pind = (char*) 0x29;  
        bool red_pressed = ((*pind) & 2); //PD2  
        bool green_pressed = ((*pind) & 4); //PD3  
        if (red_pressed)  
        {  
            TransmitChannel1();  
        }  
        if (green_pressed)  
        {  
            TransmitChannel2();  
        }  
    }  
}
```

```
ISR(INT0_vect)
```

```
{  
  
}  
ISR(IN1_vect)
```

```
{  
  
}
```

```
void DisableADCandComparator()
```

```
{  
    char* adcsra = (char *)0x7A;  
    char* acsr = (char *) 0x50;  
    *adcsra = 0; //Disable ADC  
    *acsr = 1 << 7; //Disable comparator  
}
```

```
void EnableExternalInterrupts()
```

```
{  
    char* sreg = (char*) 0x5F;  
    char* eimsk = (char*) 0x3D;  
    char* eicra = (char*) 0x69;  
    *sreg |= (1 << 7); //Enable interrupts  
    *eimsk = 3; //Enable INT0 and INT1  
    *eicra = 15; //Set trigger mode to LOW TO HIGH  
}
```

```
void TransmitChannel1()
```

```
{  
    //Transmit code for channel 1  
}
```

```
void TransmitChannel2()
```

```
{  
    //Transmit code for channel 2  
}
```

Disclaimer: untested code

# Case Study 2



Design a power-efficient embedded system for a 20-key IR remote control.  
When any button has been pressed, call a function `Transmit()`.  
Assume that this function has already been written.  
Buttons are connected to Port B (0 to 5), Port C (0 to 5), Port D (0 to 7)

Tip:  
Sleep to conserve power!  
You can use pin change interrupts to wake up the CPU.

```

int main() //void Setup()
{
    DisableADCandComparator();
    EnablePCInterrupts();

    while(1)
    {
        DeepSleep();
        Transmit();

    }

ISR(PCINT0_vect)
{

}

ISR(PCINT1_vect)
{

}

ISR(PCINT2_vect)
{

}

```

Disclaimer: untested code

```

void DisableADCandComparator()
{
    char* adcsra = (char *)0x7A;
    char* acsr = (char *) 0x50;
    *adcsra = 0; //Disable ADC
    *acsr = 1 << 7; //Disable comparator
}

void EnablePCInterrupts()
{
    char* sreg = (char*) 0x5F;
    char* pcicr = (char*) 0x68;
    char* pcmsk0 = (char*) 0x6B;
    char* pcmsk1 = (char*) 0x6C;
    char* pcmsk2 = (char*) 0x6D;

    *sreg |= (1 << 7); //Enable interrupts
    *pcicr = 7; //Enable PC interrupts
    *pcmsk0 = 31; //Enable PC PORT B (0 to 5)
    *pcmsk1 = 31; //Enable PC PORT C (0 to 5)
    *pcmsk2 = 255; //Enable PC PORT D (0 to 7)
}

void Transmit()
{
    //Find out which pin is HIGH and
    //Transmit respective code at 38kHz
}

```

Happy Sleeping!