

Lecture 10

Analog Output

CE346 – Microprocessor System Design
Branden Ghena – Fall 2022

Some slides borrowed from:
Josiah Hester (Northwestern), Prabal Dutta (UC Berkeley)

Administrivia

- Everyone should have gotten feedback by now
 - If you haven't, either check spam folder or I somehow forgot about you
- Design presentations all next week!
 - Happy to discuss things before then either after class or on Campuswire
- Drop deadline is Friday next week
 - I'm not worried about anyone in CE346
 - But if you're worried, I'm happy to talk about it.

Today's Goals

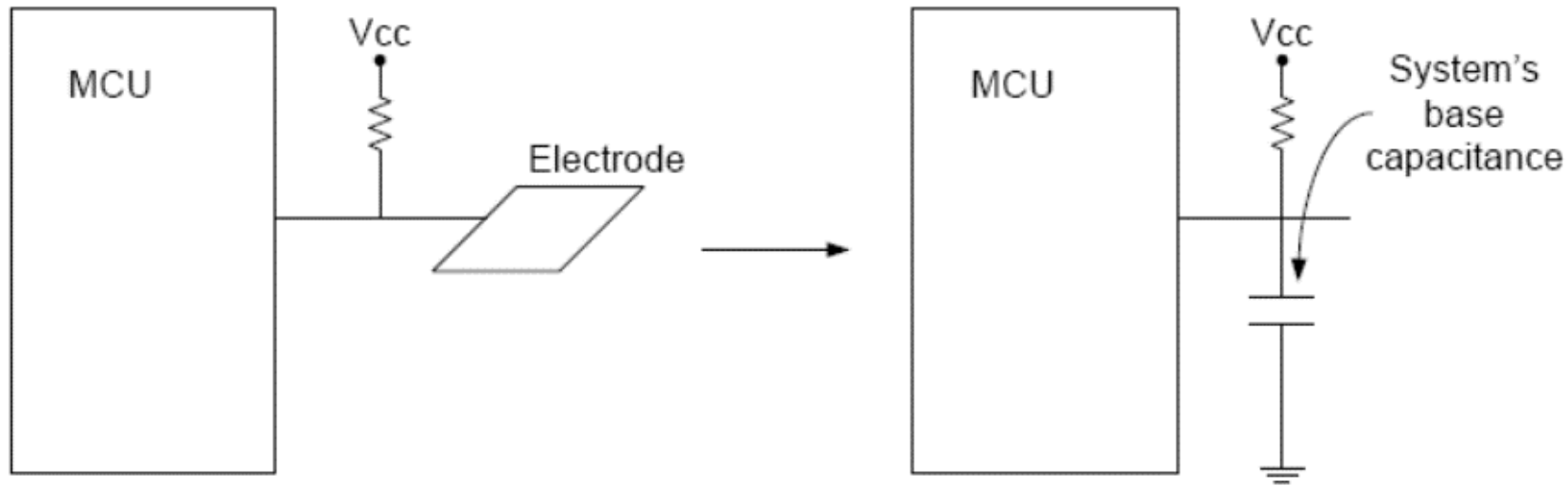
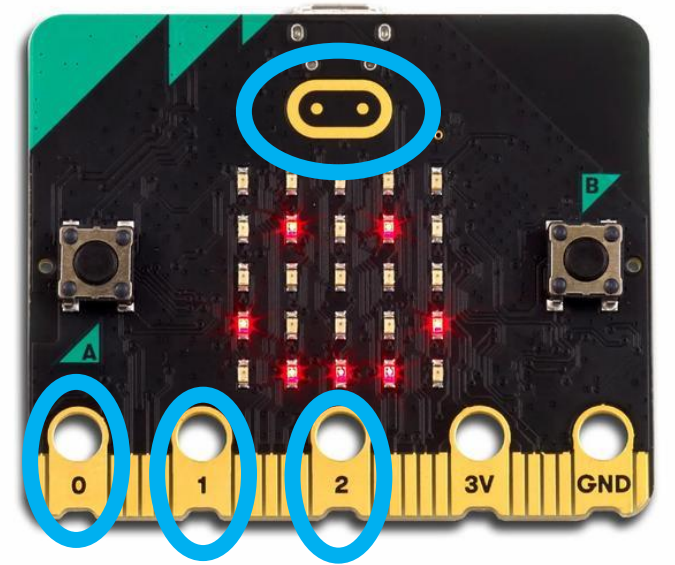
- Discuss one last sensor: capacitive touch
- Explore common methods for generating analog signals
- Understand the role of Digital-to-Analog converters
- Discuss the concepts of Pulse-Width Modulation
 - And the nRF52 implementation of it

Outline

- **Capacitive Touch Sensing**
- Digital-to-Analog Converters
- Pulse-Width Modulation
- nRF52 PWM

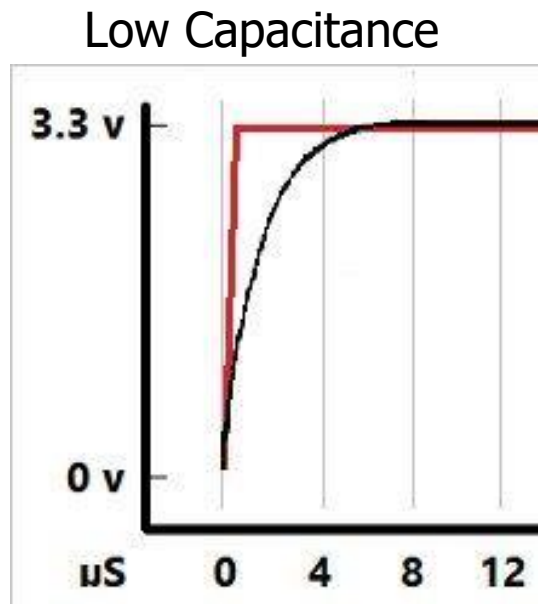
Capacitive Touch Sensor

- Pull-up resistors connected to metal pads
 - Also connected to GPIO pin
- Acts as a capacitor connected to ground

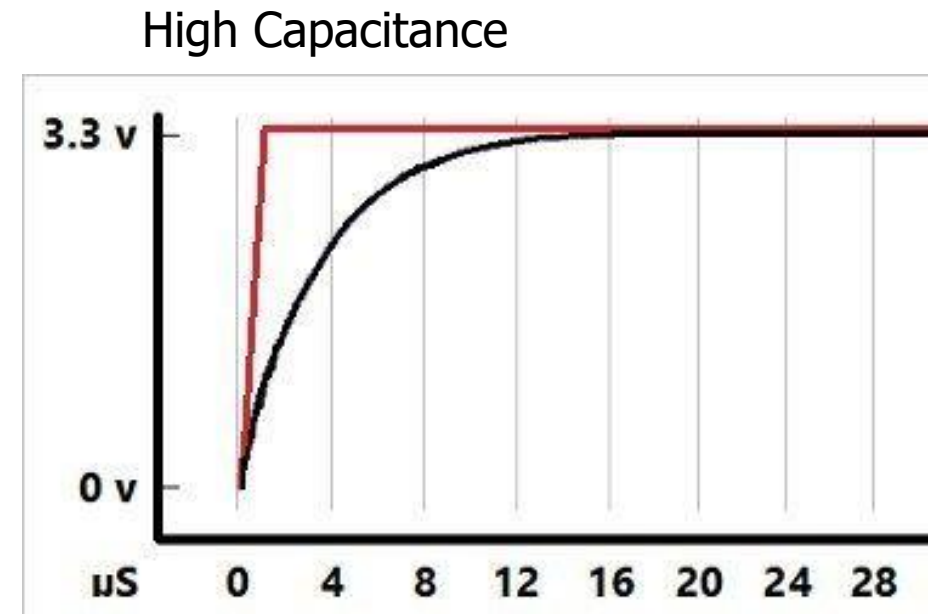


The touch pad recharges on its own

- If you drive the GPIO pin attached to it low, touchpad clears low
- If you make the GPIO pin an input (high impedance)
 - The touchpad gets pulled high, which takes some amount of time



The more
capacitance the
longer this takes!



Capacitive touch sensing method

1. Drive GPIO pin low
 - Connects the pad to ground
2. Set GPIO pin as input and enable low-to-high interrupt
 - Gets an interrupt when the pad finally becomes high on its own
 - Use a timer to determine time until interrupt
 - $\sim 70 \mu\text{s}$ with no finger, \leq milliseconds with finger
 - Needs to timeout after a few milliseconds and declare "touched"
3. Repeat periodically (a few times a second is probably good enough)

Sudden large increase in rise time \Rightarrow someone is touching!

- Finger acts as a large capacitor

Capacitive touch works on any metal surface

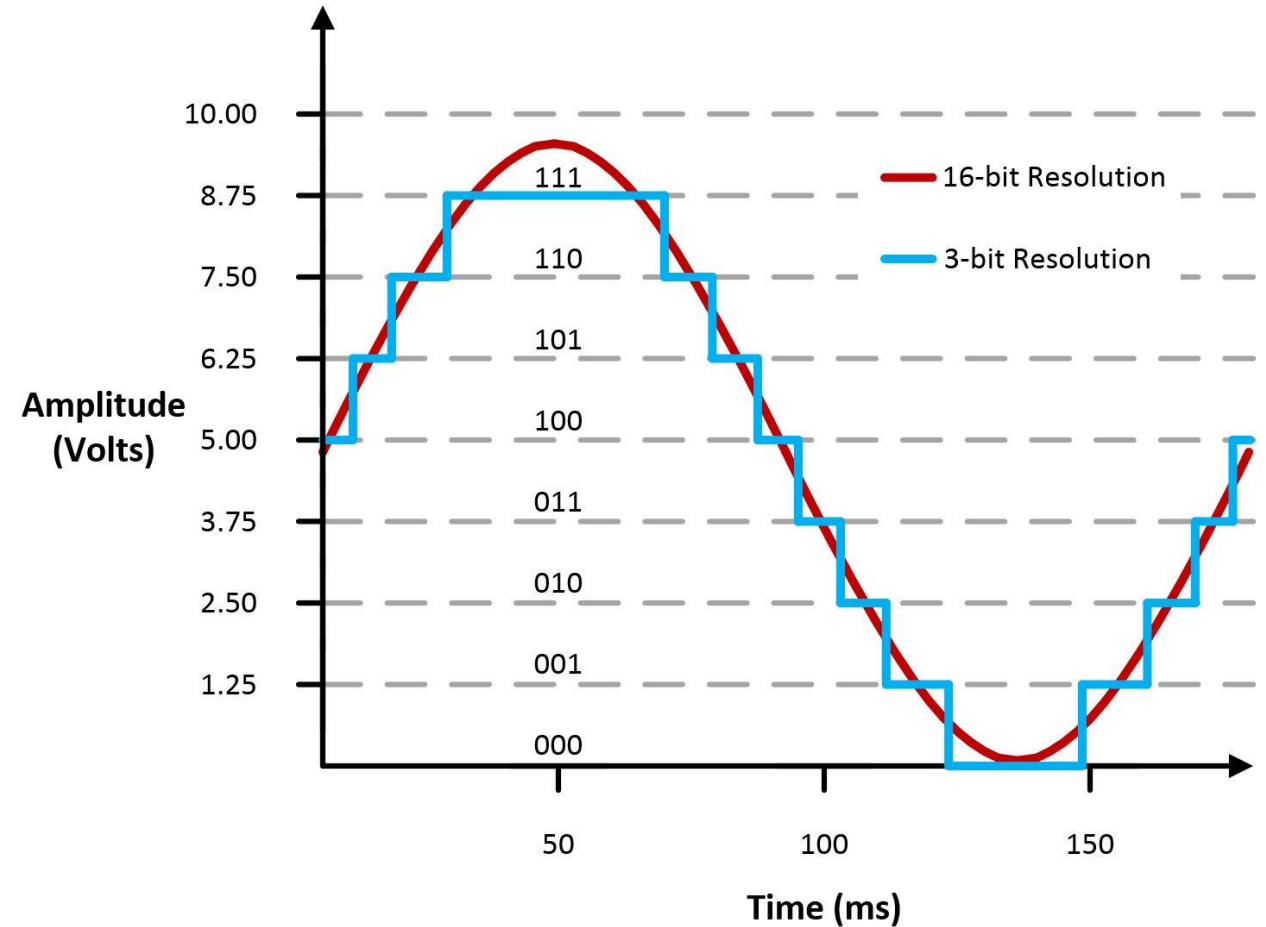
- Idea: Microbit door handle sensor
- Connect a wire and a pull-up resistor to a metal door handle to sense when someone is touching it!
 - Timing will be very different from capacitive pad, but should be repeatable and distinguishable from human touch

Outline

- Capacitive Touch Sensing
- **Digital-to-Analog Converters**
- Pulse-Width Modulation
- nRF52 PWM

Digital-to-Analog Converters

- Generates an analog voltage
- DACs are conceptually the inverse of ADCs
 - Number of bits of resolution choose analog step size
 - Frequency determines step duration

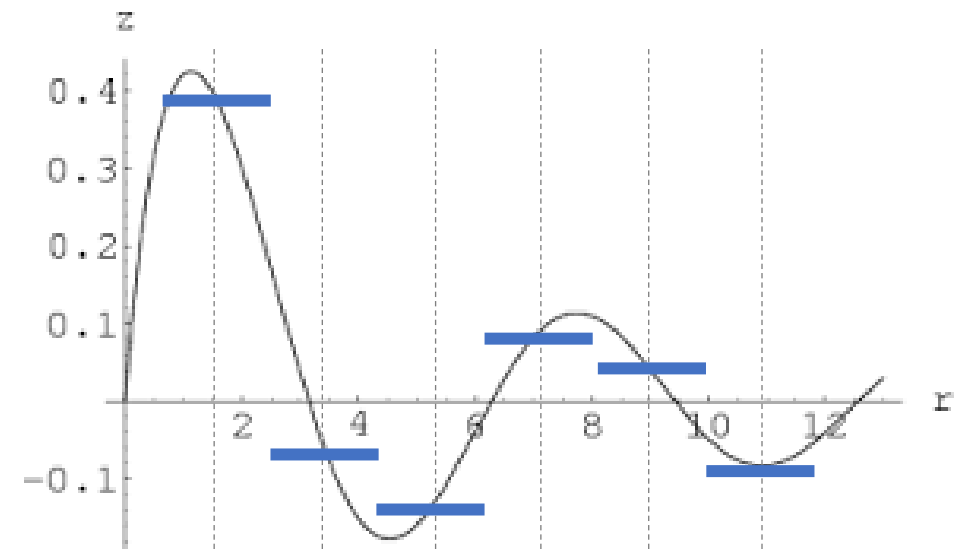
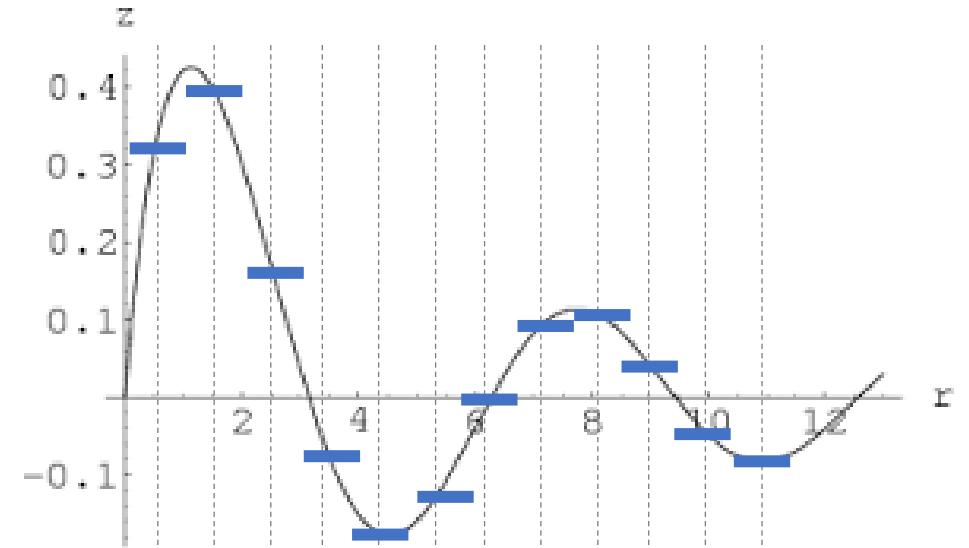


High resolution versus high frequency

- What role does each play in a DAC?
Which is more important?
- High resolution can accurately represent a voltage
- High frequency can accurately represent a changing voltage
- In practice:
 - Need high *enough* resolution, then as high of frequency as possible

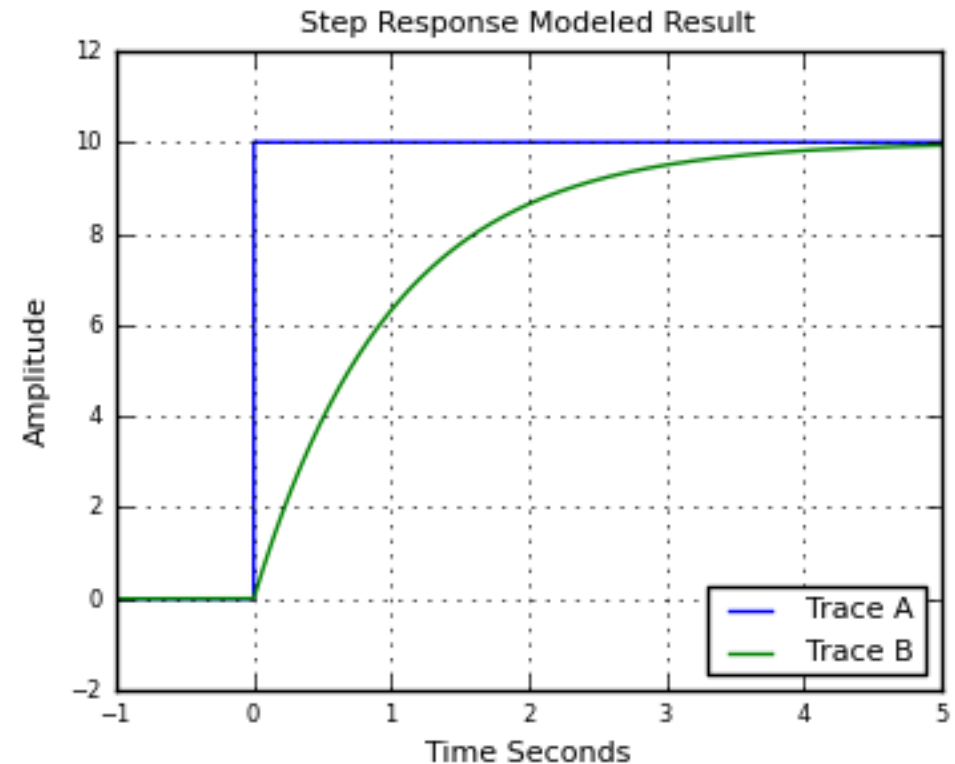
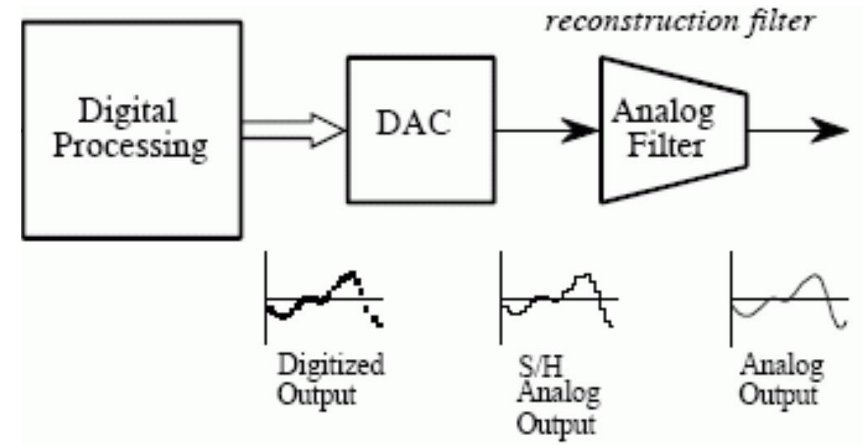
Infinite resolution is not sufficient

- DAC frequency corresponds to representable signal changes
 - Rise and fall times
- Even an infinite resolution DAC cannot represent a signal if it is not fast enough



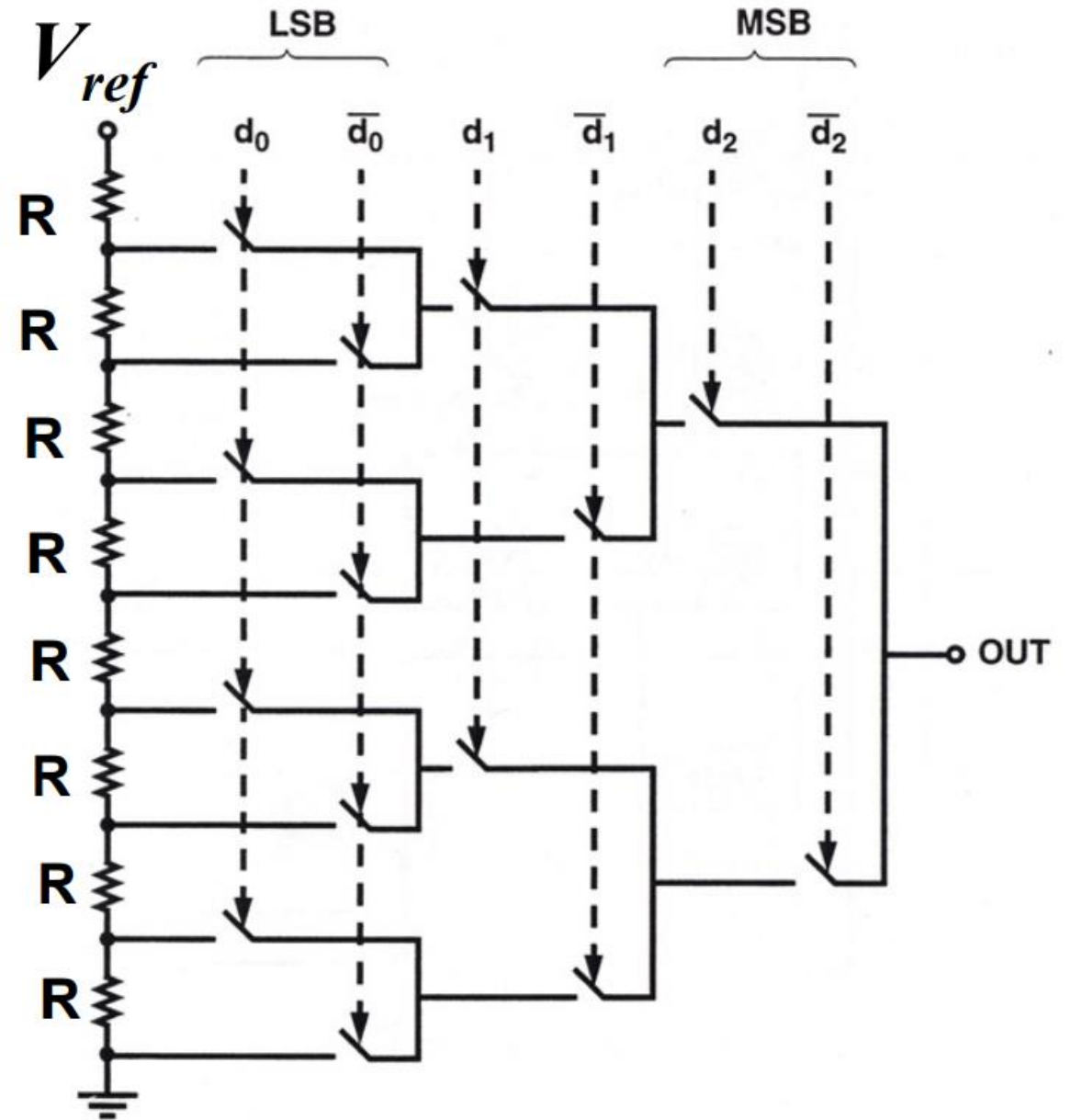
Low-pass filter smooths output

- Low-pass filter delays changes in voltage and smoothly transitions between them
 - Low-frequency signals stay
 - High-frequency are smoothed
- Greatly improves quality of output but must be tuned to the desired signal frequency
 - Usually not included in microcontroller



Resistor string DAC

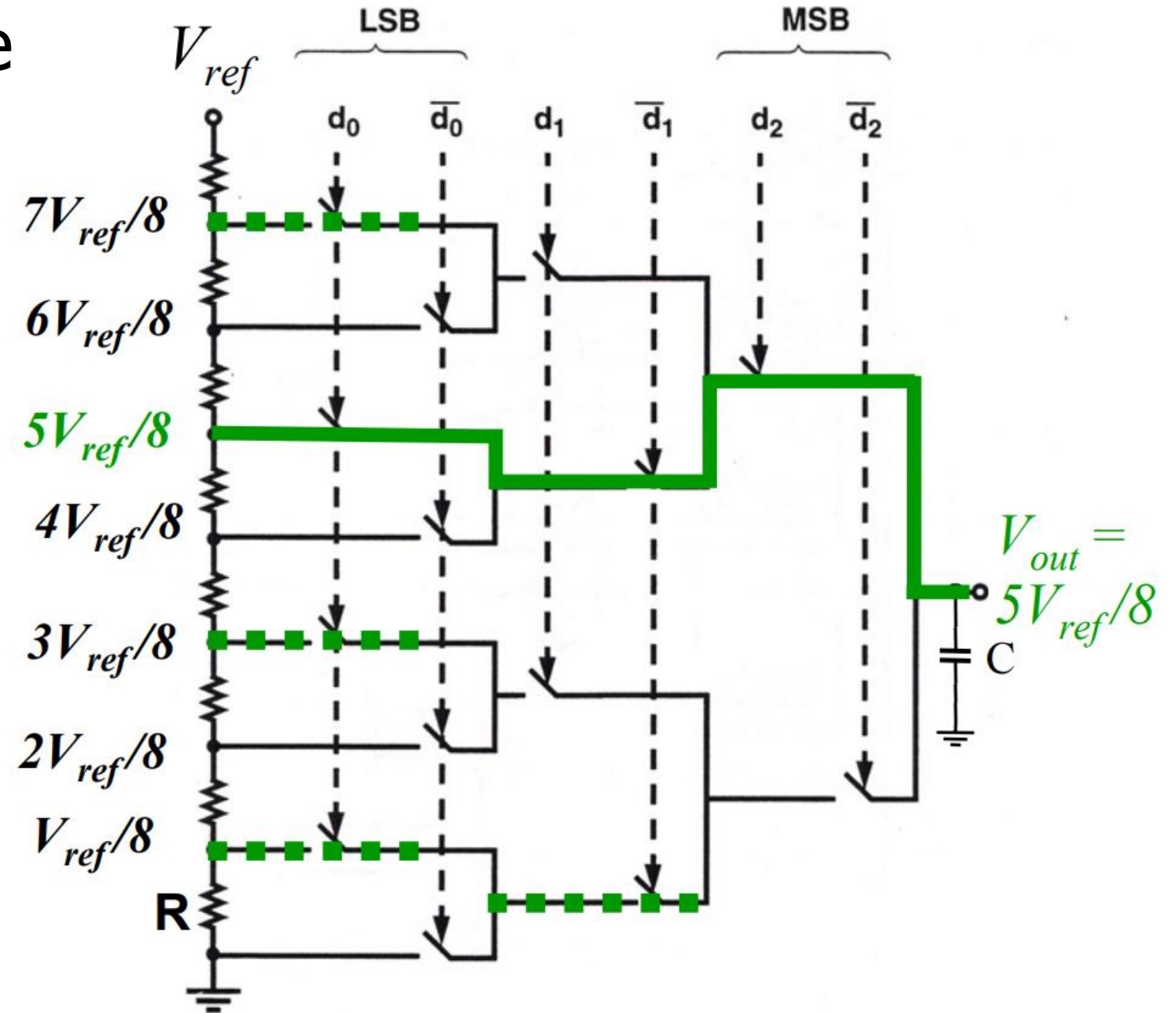
- Use series of voltage dividers and switches to set output voltage
 - Generates equally spaced voltages that can be selected between
- Needs output buffer to provide stable current
- Takes a lot of resistors
 - And resistors take a lot of silicon



Resistor string example

- $V_{out} = code * \frac{V_{ref}}{2^{resolution}}$

- Input code is **101**
 - Selects switches such that $5/8 * V_{ref}$ is connected to output



Break + DAC applications

- What do you use an analog output for?

Break + DAC applications

- What do you use an analog output for?
 - Audio output
 - But it needs to be high quality (resolution and speed)
 - Motors
 - But only with a controller that actually drives them with enough current
 - LED brightness
 - Not Much
 - And these last two can be done more easily with PWM

DACs are not in all microcontrollers

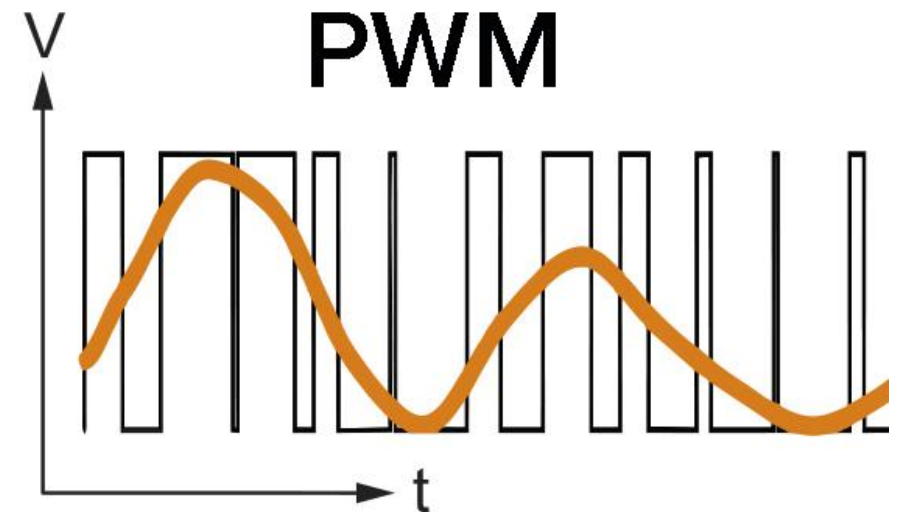
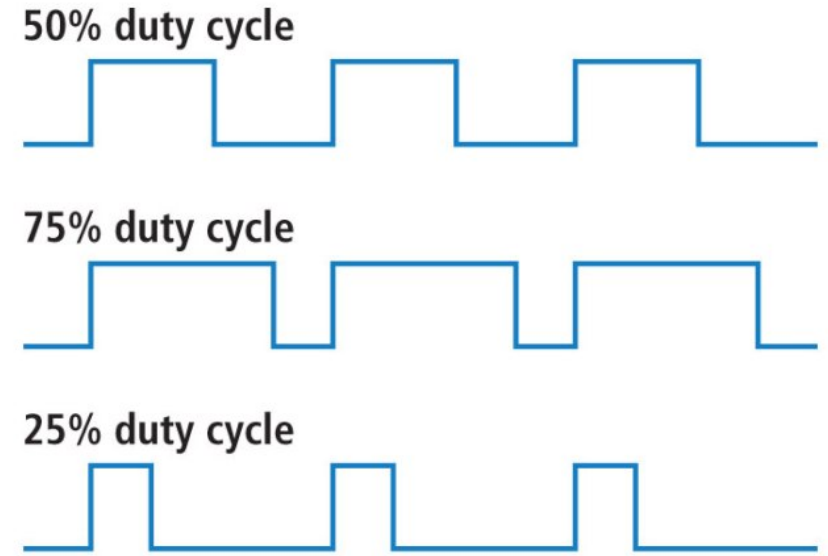
- Not rare, but not ubiquitous either
 - Every microcontroller has GPIO
 - Just about every microcontroller has an ADC
 - Some microcontrollers have DACs (the nRF52833 does not!)
- Reasons
 - Hardware is complicated (but we could fit it if we wanted)
 - Use cases are uncommon (and might need very high quality)
 - Many devices can be controller digitally
 - Pulse-Width Modulation (PWM) can emulate usably analog signals

Outline

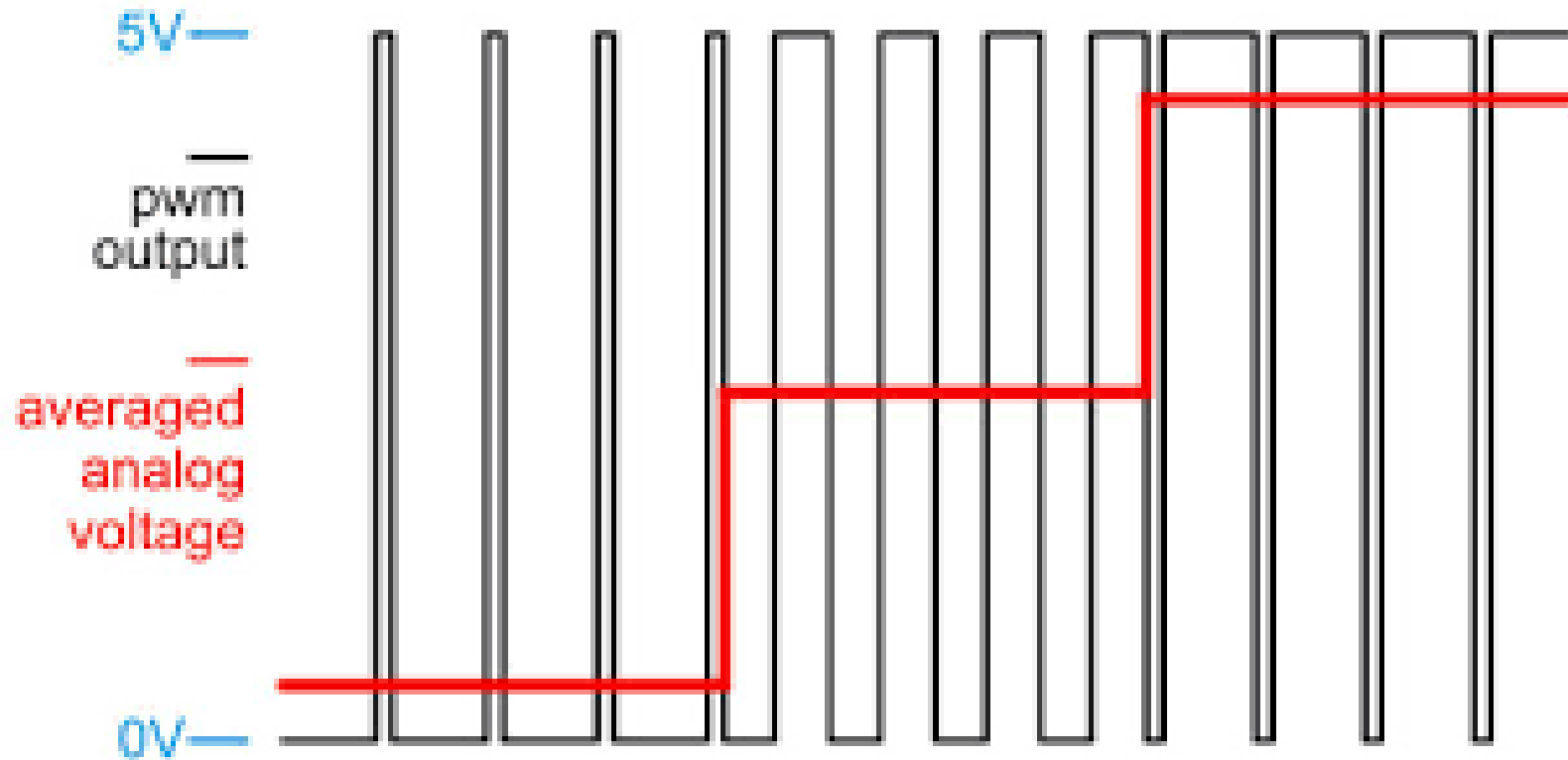
- Capacitive Touch Sensing
- Digital-to-Analog Converters
- **Pulse-Width Modulation**
- nRF52 PWM

Pulse-Width Modulation

- Much easier to control high or low than an analog output
- Idea: modify how long a signal is high within some switching frequency, a.k.a duty cycle
 - On 50% of the time for half voltage
 - On 10% of the time for tenth voltage
- Duty cycle, not frequency!



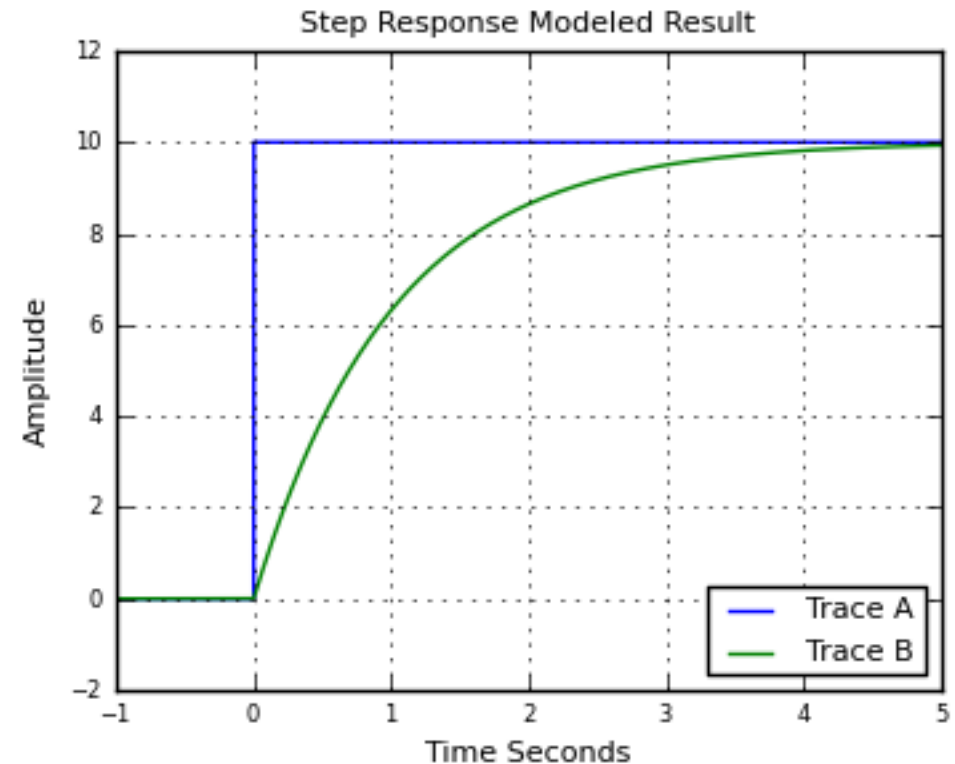
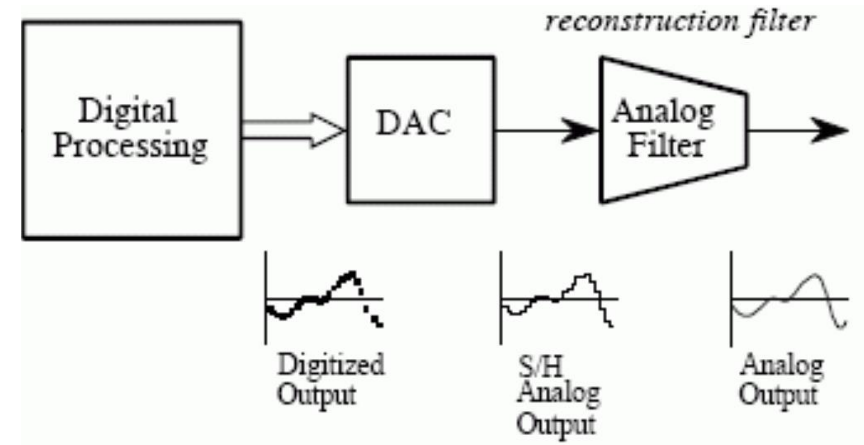
PWM to Analog Signal example



- PWM period should be much faster than the desired analog signal
 - PWM duty cycle represents the voltage along the way
 - Multiple duty cycles per output point makes it more accurate

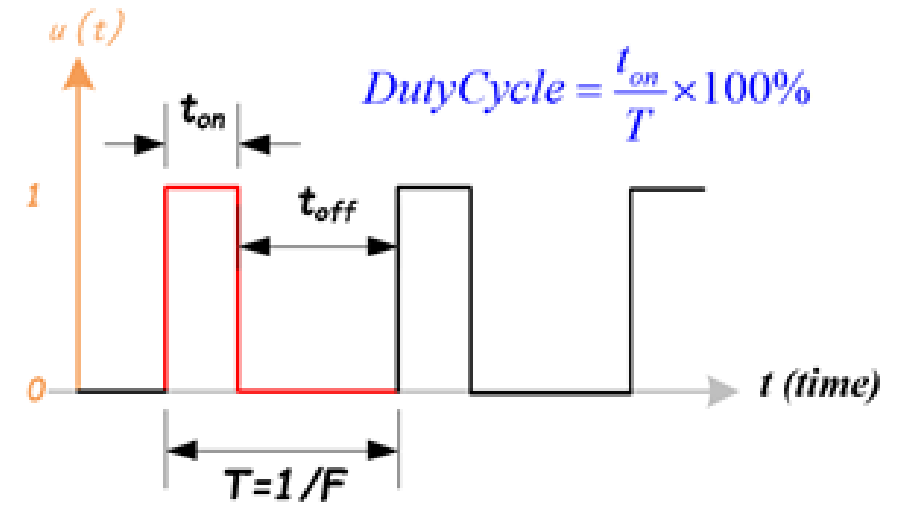
Low-pass approach works here too

- Importantly, many devices are inherent low-pass filters
- Heaters, Motors
 - Low-pass by physical design
 - I.e., they can't start/stop quickly
- LEDs are not
 - But our eyes are!



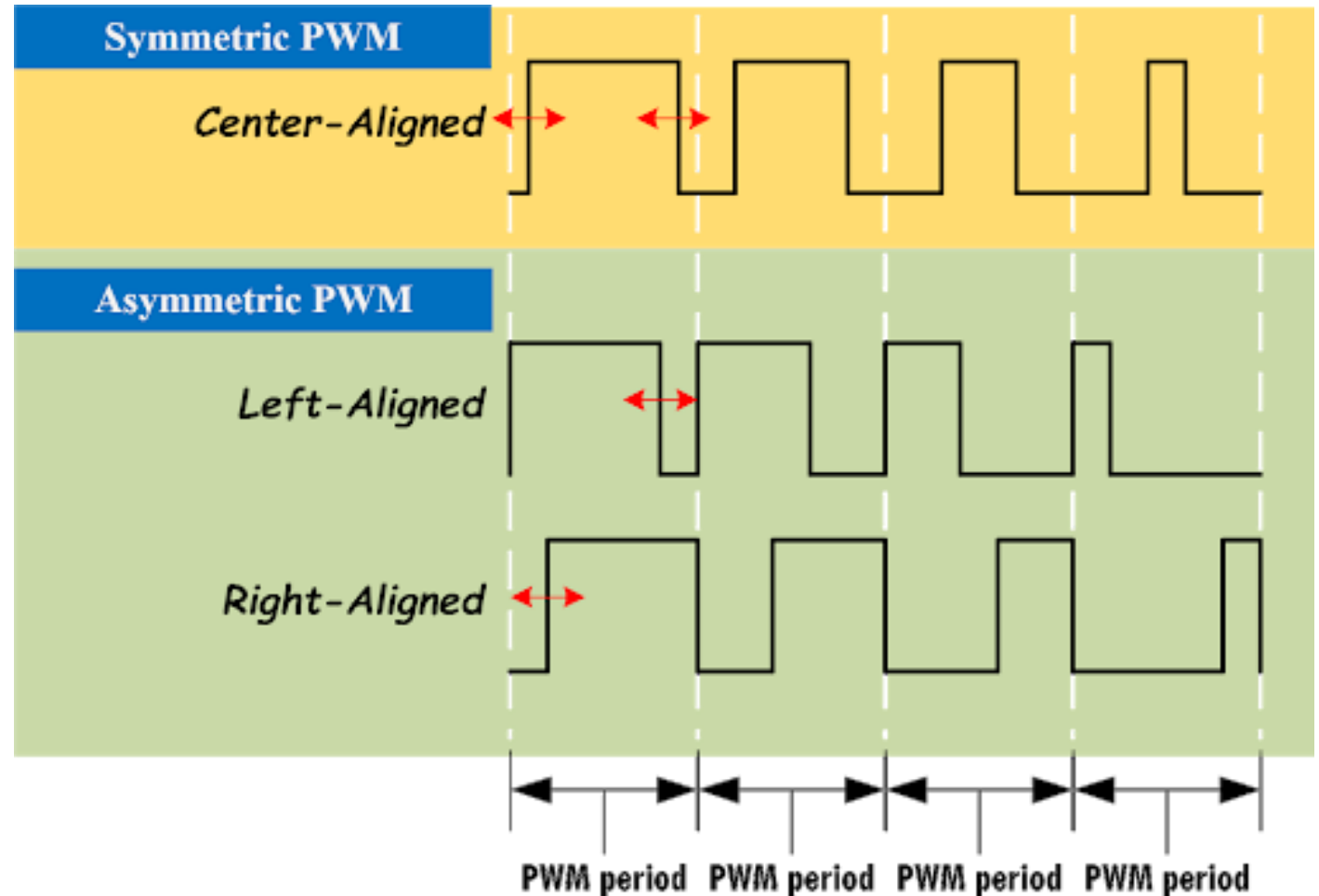
Controlling PWM

- Vary duty cycle by selecting transition points
 - Time when set
 - Time when unset
- Repeat every cycle
 - Period much faster than signal if possible
 - Makes analog approximation more accurate
 - The faster you run it, the less likely it matters that it is not actually analog
 - Example: LED switching frequency
- Duty cycle could vary cycle-by-cycle if it must



PWM alignment

- Can select alignment as well
 - Equivalent to a phase delay
- Centering produces cleaner analog output
 - Less harmonics
- Not relevant for most devices



Every microcontroller can do PWM

- Not every microcontroller has a PWM peripheral
- But every microcontroller has timers and digital outputs

- All that is needed is a GPIO and a Timer (or two)
 - Timer determines when to turn GPIO on and off
 - Often can be automated in hardware rather than use interrupt handler

PWM is a method of encoding data

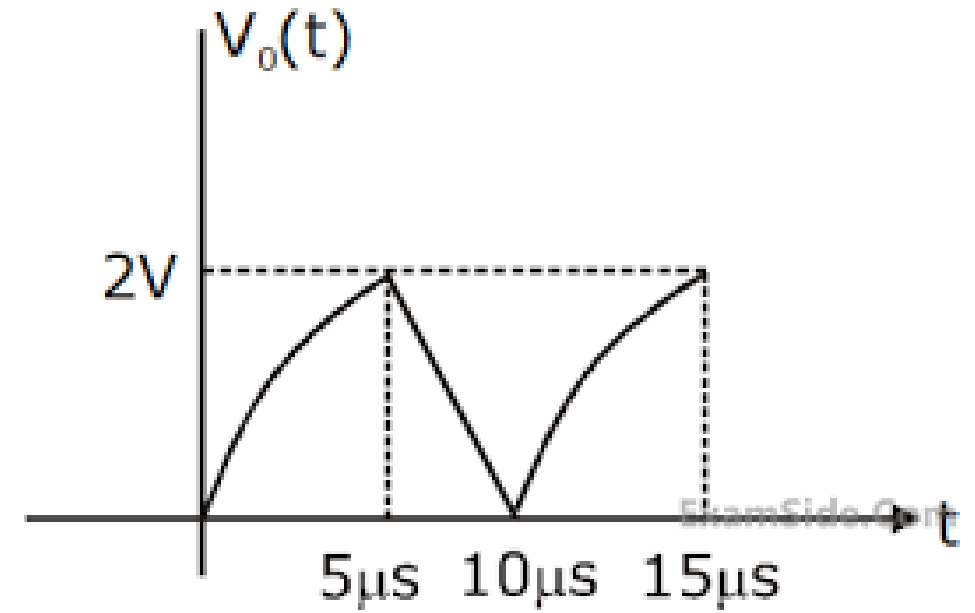
- PWM is a pulse-width modulated signal
- There are many other ways to “modulate” a signal to transmit data
 - Amplitude, Frequency, and Phase are common
 - Layers data on top of an existing “carrier signal”
- Used especially for high-speed communication
 - Wired (cable lines) or Wireless (basically everything)

PWM applications

- Servos
 - Duty cycle chooses angle or rotation speed
- Motor controllers
 - Duty cycle chooses current and therefore speed
- LED brightness
 - And “breathing” effect
- Audio
 - Can sound okay if frequency is high enough

Break + Open Question

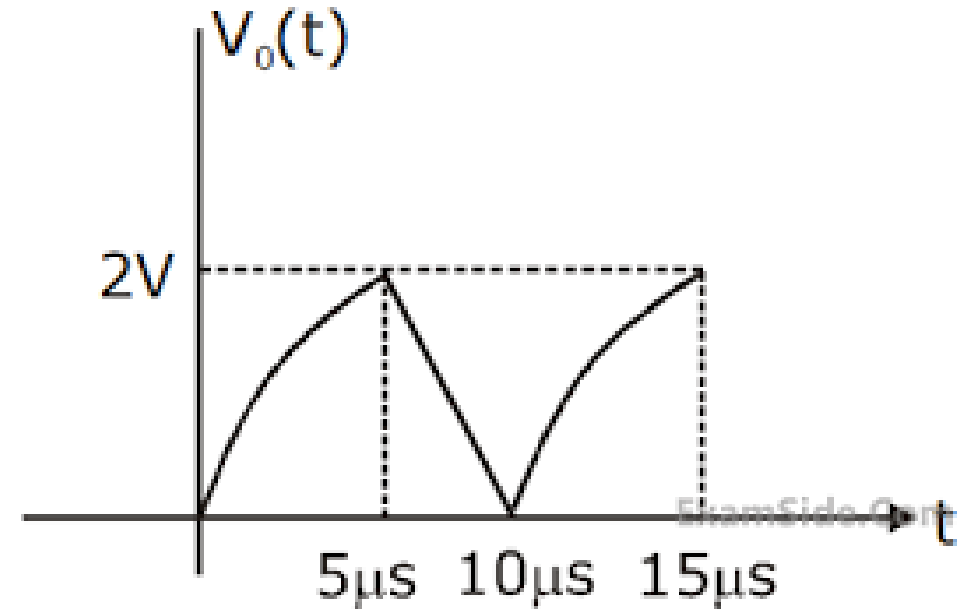
- Imagine you want to represent the following signal with PWM
 - **What should the PWM period be?**



- **What kinds of duty cycle values would you use? (3.3v is 100%)**

Break + Open Question

- Imagine you want to represent the following signal with PWM
 - **What should the PWM period be?**
 - Signal period is $\sim 10 \mu\text{s}$
 - PWM period should be at least 2x that
 - 10x faster seems like a good start
 - Then if we want multiple PWM outputs per sample, that's $\sim 20\text{-}40\text{x}$ faster



- **What kinds of duty cycle values would you use?** (3.3v is 100%)
 - $2/3.3 = 61\%$ duty cycle max
 - $0/3.3 = 0\%$ duty cycle min

Outline

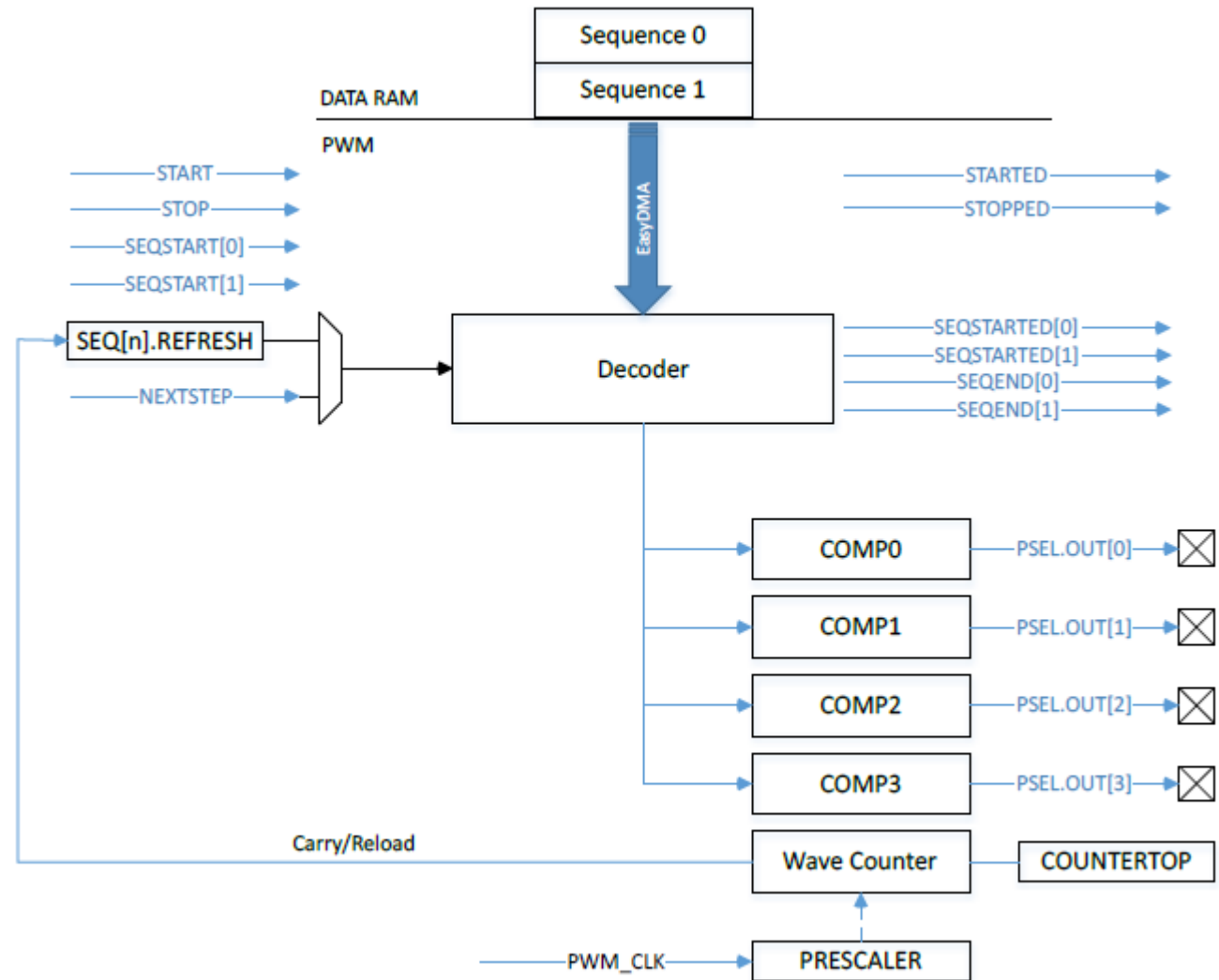
- Capacitive Touch Sensing
- Digital-to-Analog Converters
- Pulse-Width Modulation
- **nRF52 PWM**

nRF52 PWM – theory of operation

- A clock continuously adds to a counter value
 - (just like the Timer peripheral does)
- When the counter value reaches COMP[n], the GPIO value on channel **n** changes from high to low (or vice-versa)
- When the counter value reaches COUNTERTOP, the GPIO value on channel **n** changes from low to high (or vice-versa)
 - AND the counter value resets to zero

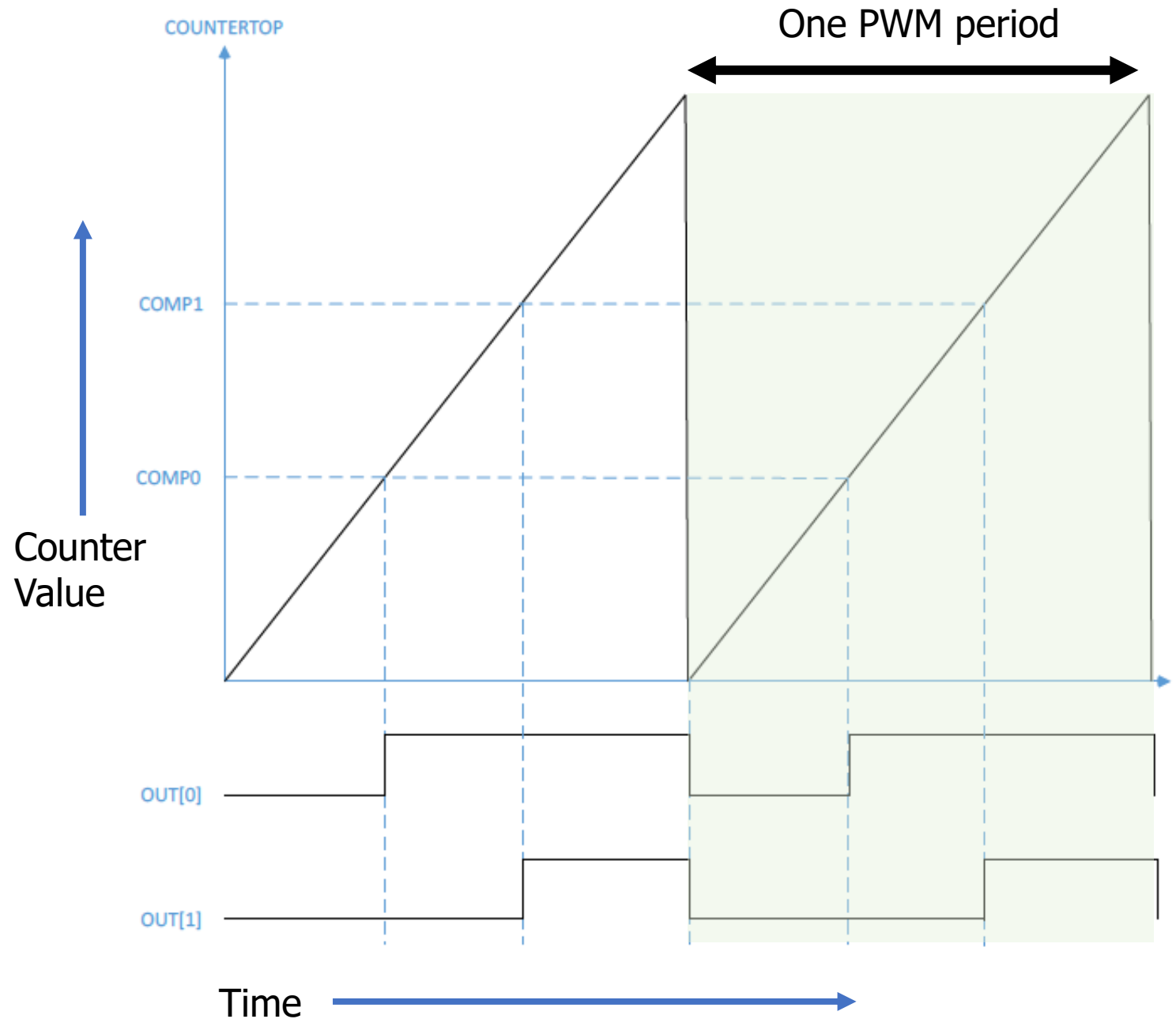
nRF52 PWM peripheral

- Uses internal timer to create PWM output on up to 4 pins
 - 4 peripherals, so up to 16 pins total
- Loads compare values via DMA to rapidly vary "analog" signal



PWM example

- Counter increments up to COUNTERTOP, resets and continues
- Period/Frequency
 - Chosen by COUNTERTOP and timer PRESCALER

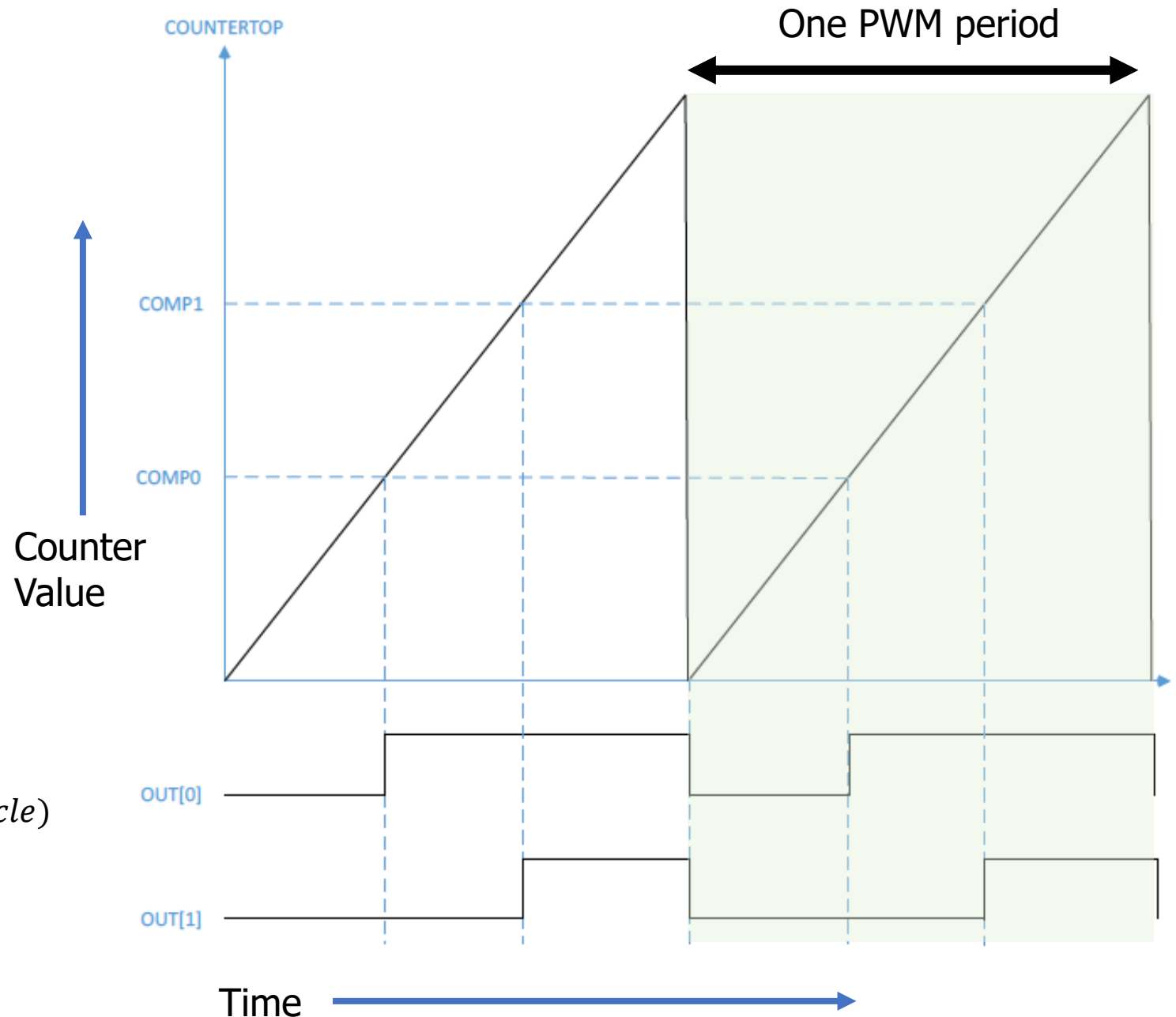


PWM example

- Counter increments up to COUNTERTOP, resets and continues
- Duty Cycle
 - COMP0 chooses first toggle point for OUT[0]
 - Second toggle point is when the timer resets

(right-aligned) $COMP = COUNTERTOP - (COUNTERTOP * DutyCycle)$

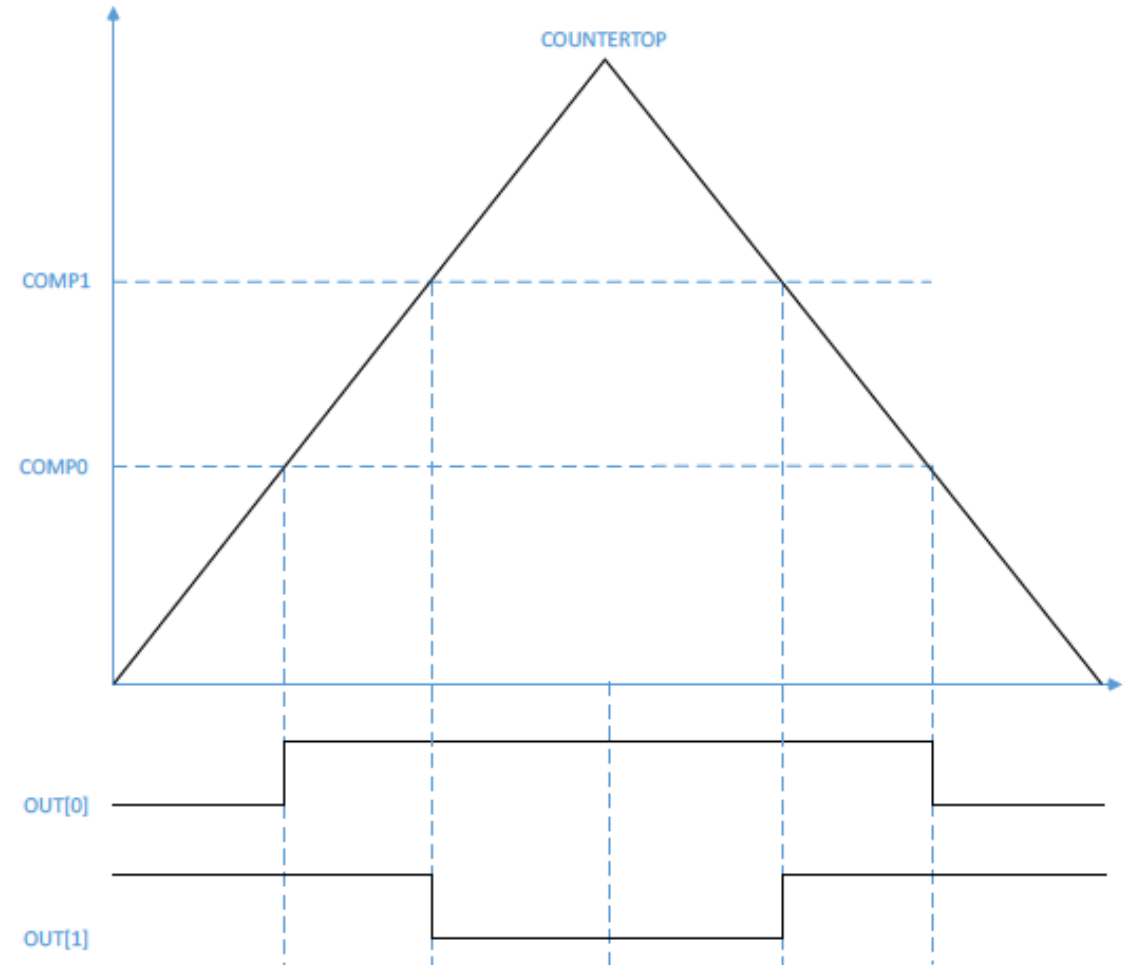
(left-aligned) $COMP = COUNTERTOP * DutyCycle$



Center-aligned PWM

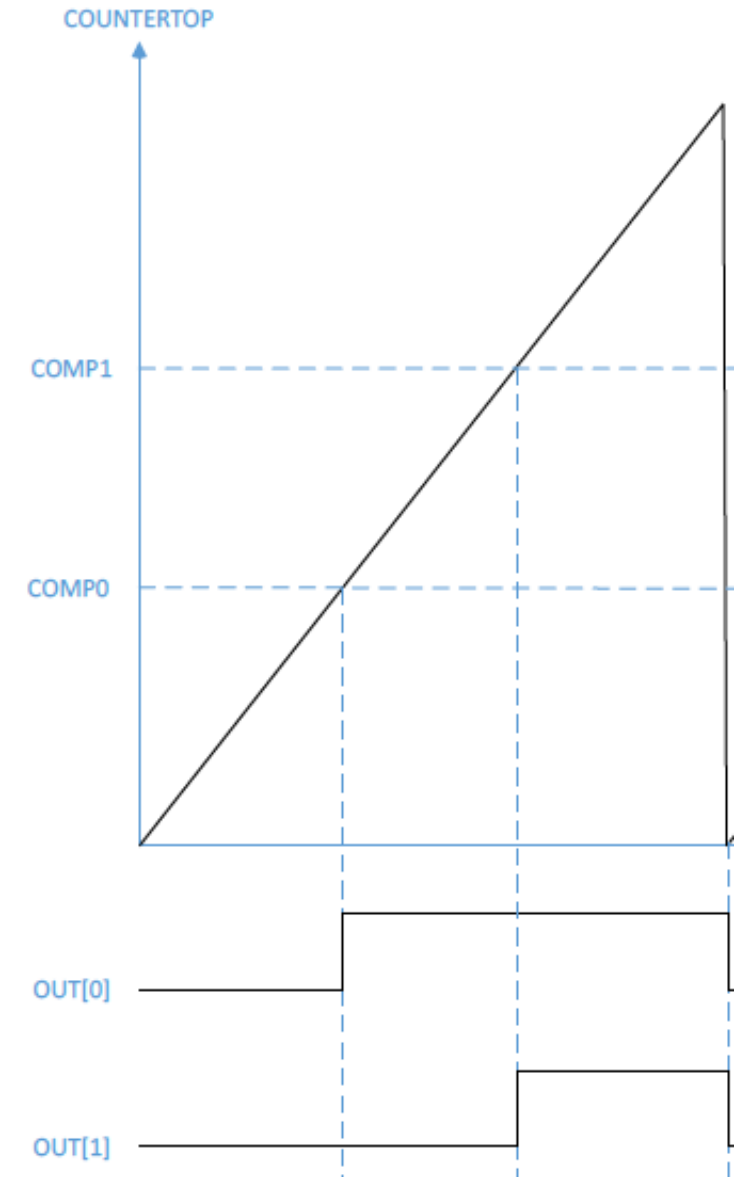
- Up-and-down mode enables center-aligned PWM
- Duty Cycle
 - Comp triggers toggle on rise
 - Comp triggers toggle again on fall

$$COMP = COUNTERTOP - (COUNTERTOP * 0.5 * DutyCycle)$$



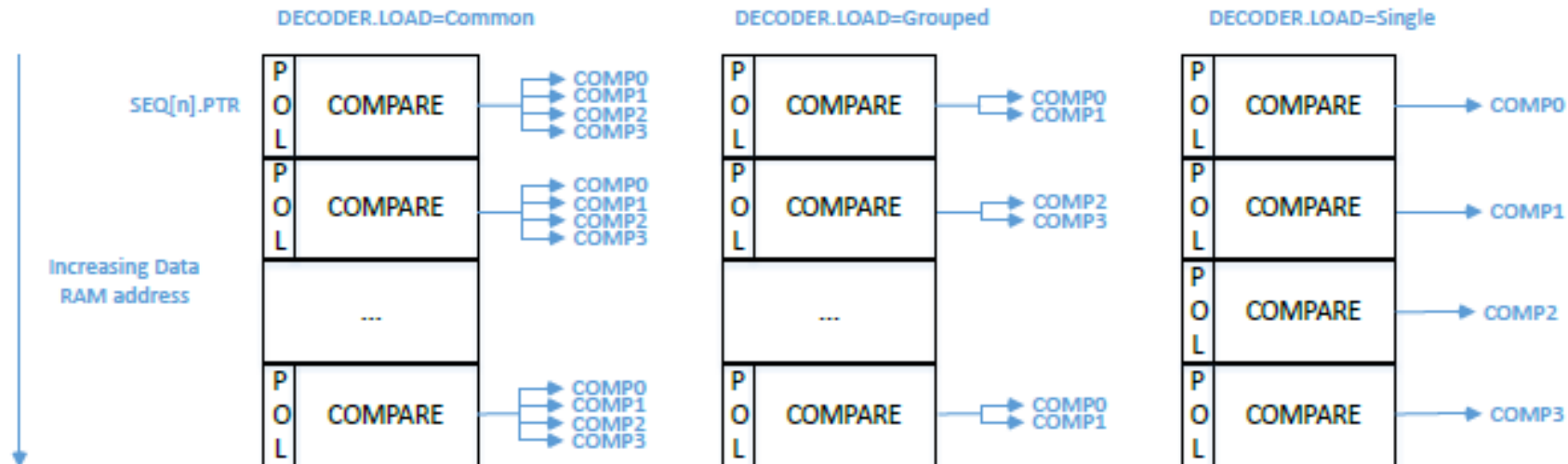
Trading speed and accuracy

- How do you get the most accurate PWM values?
 - Select the largest COUNTERTOP possible
 - Most possible COMP values
 - Up to 15-bit resolution (32767 max)
- How do you get the fastest PWM frequency?
 - Select the smallest COUNTERTOP possible
 - PRESCALER also affects this
 - 16 MHz – 128 kHz (8 possible values)
- Fastest PRESCALER + largest COUNTERTOP equals 488 Hz
 - Likely need to sacrifice resolution for speed



DMA with PWM

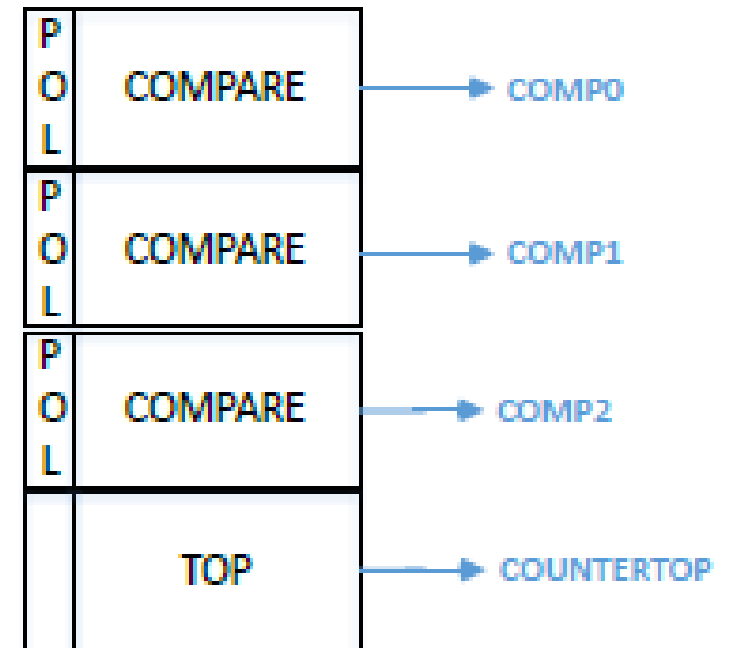
- Every N periods it loads a new configuration from RAM
 - N combined with PRESCALER and COUNTERTOP chooses “analog signal” period
- Configuration sets COMP values for each output channel
 - Also sets polarity (starting value: low or high)
- Application of memory loads to channels is configurable



Waveform mode

- Also has the option to change COUNTERTOP every N PWM periods
- Allows arbitrary waveforms to be created
 - Frequency changes every period
 - Duty cycle can also change each period
- We don't normally need this, as a constant frequency with changing duty cycle should be fine

DECODER.LOAD=WaveForm



Other configurations

- How many times the entire DMA sequence repeats
 - 0 to large number, infinite with a configuration in SHORTS
- How long to delay between repeating sequence cycles
 - Repeats last PWM configuration
- Two DMA sequence configurations (0 and 1)
 - Can modify one while the other is playing
 - Allows continuous signal (for example, music)

nRF SDK PWM driver

https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsdk_nrf5_v16.0.0%2Fgroup_nrfx_pwm.html

- Initialize PWM with base configuration
 - Output pins, Clock frequency, COUNTERTOP, DMA grouping mode
 - Handler for events from peripheral
- `nrfx_pwm_simple_playback(instance, sequence, count, flags)`
 - Instance: pointer to global variable with registers
 - Sequence: struct containing sequence to be played (see next slide)
 - Count: number of times (1 or more) to repeat sequence
 - Flags: stop peripheral when done, loop forever, various events

Sequence struct

Data Fields

[nrf_pwm_values_t values](#)

Pointer to an array with duty cycle values. This array must be in Data RAM. [More...](#)

[uint16_t length](#)

Number of 16-bit values in the array pointed by `values`.

[uint32_t repeats](#)

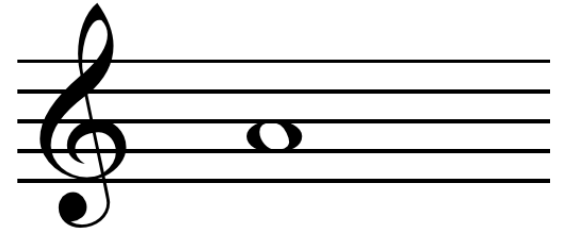
Number of times that each duty cycle is to be repeated (after being played once). Ignored in [NRF_PWM_STEP_TRIGGERED](#) mode.

[uint32_t end_delay](#)

Additional time (in PWM periods) that the last duty cycle is to be kept after the sequence is played. Ignored in [NRF_PWM_STEP_TRIGGERED](#) mode.

- `values`: pointer to array of `uint16_t` values (union of types)
- `length`: length of array
- `repeats`: number of times to repeat each individual value
 - Sets period for "analog value" changing

Example, playing a note



- Pick PWM frequency to match note frequency
 - Combination of PRESCALER, COUNTERTOP, and repeats
 - 440 Hz for the note A
 - PRESCALER 1 MHz, COUNTERTOP 2273 -> 440 Hz
- Set duty cycle of PWM to control volume
 - 50% duty cycle -> COMP value of 1137
- Set sequence with an array of length 1, content is {1137} (polarity 0)
 - Repeats 0, end_delay 0
- Set playback_count to 1 and flags to NRFX_PWM_FLAG_LOOP

Controlling LED Matrix brightness

- Option 1: PWM peripheral
 - Need to use multiple PWM peripherals to get 5 pins
 - Could only allow brightness to be controlled for the entire matrix
 - Then use a single PWM output to control the row
 - When timer fires, change which row pin is used for PWM
- Option 2: do it manually (for individual control)
 - Can't determine duty cycle when the row is turned on
 - Each row already at 100 Hz, duty cycling would be slower and visible
 - Instead add 5 new one-shot app timers, one for each column
 - Fire at some time while the row is active (within that 2 ms)
 - Use to toggle column LED back to off

Outline

- Capacitive Touch Sensing
- Digital-to-Analog Converters
- Pulse-Width Modulation
- nRF52 PWM