

Lecture 10

Analog Output

CE346 – Microprocessor System Design
Branden Ghena – Fall 2025

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

Administrivia

- Design presentations all next week!
 - Happy to discuss things before then via Piazza
 - I am providing groups with feedback as I have time to
 - Sent a couple of emails already
(most are just thoughts, no major concerns)
 - Let me know if there's a particular day you can't present on
- 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

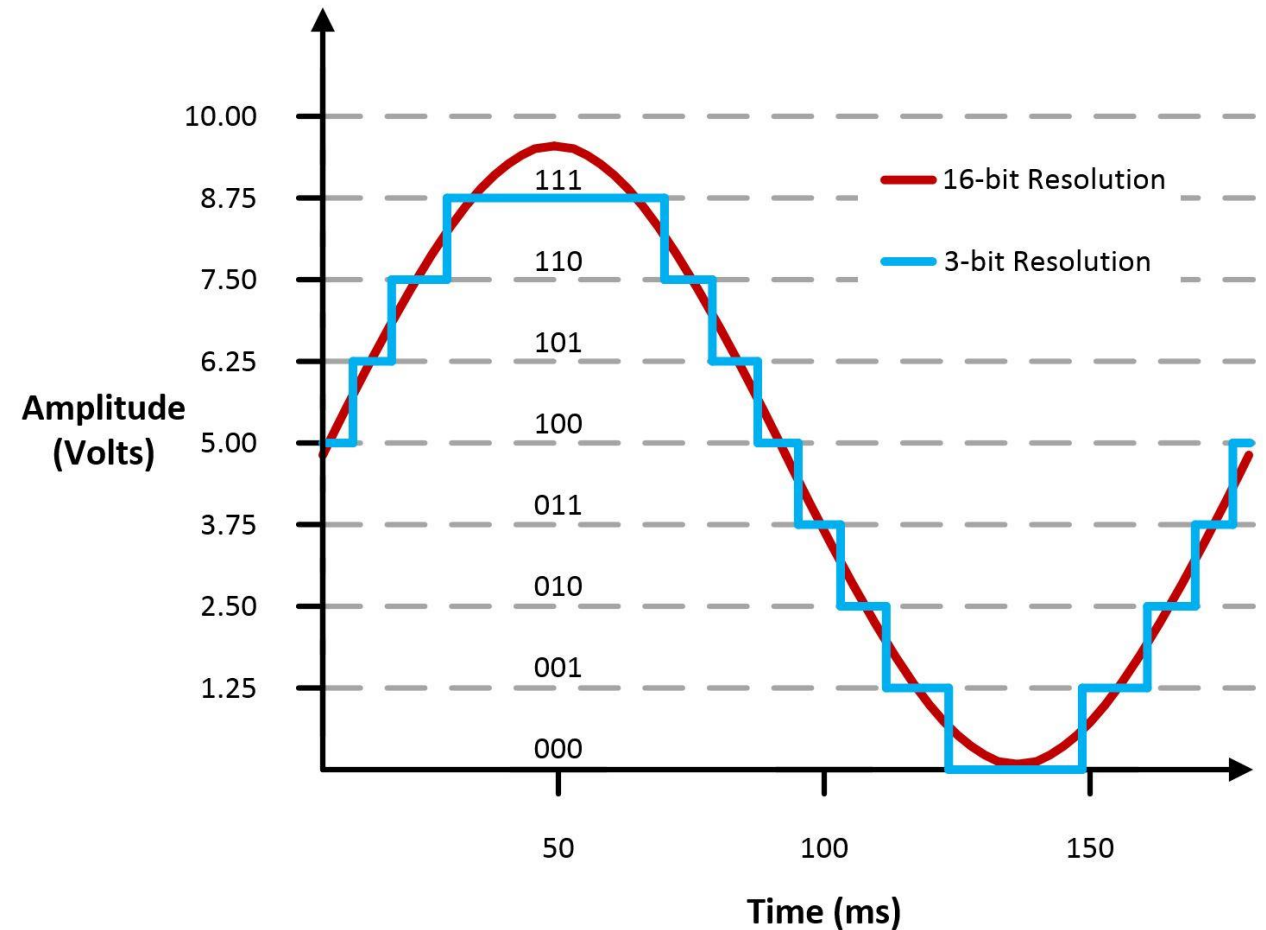
- Explore common methods for generating analog signals
- Understand the role of Digital-to-Analog converters
- Discuss the concepts of Pulse-Width Modulation
 - And the nRF52 implementation of it
- Work through examples of how to use PWM

Outline

- **Digital-to-Analog Converters**
- Pulse-Width Modulation
- nRF52 PWM
- Using 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

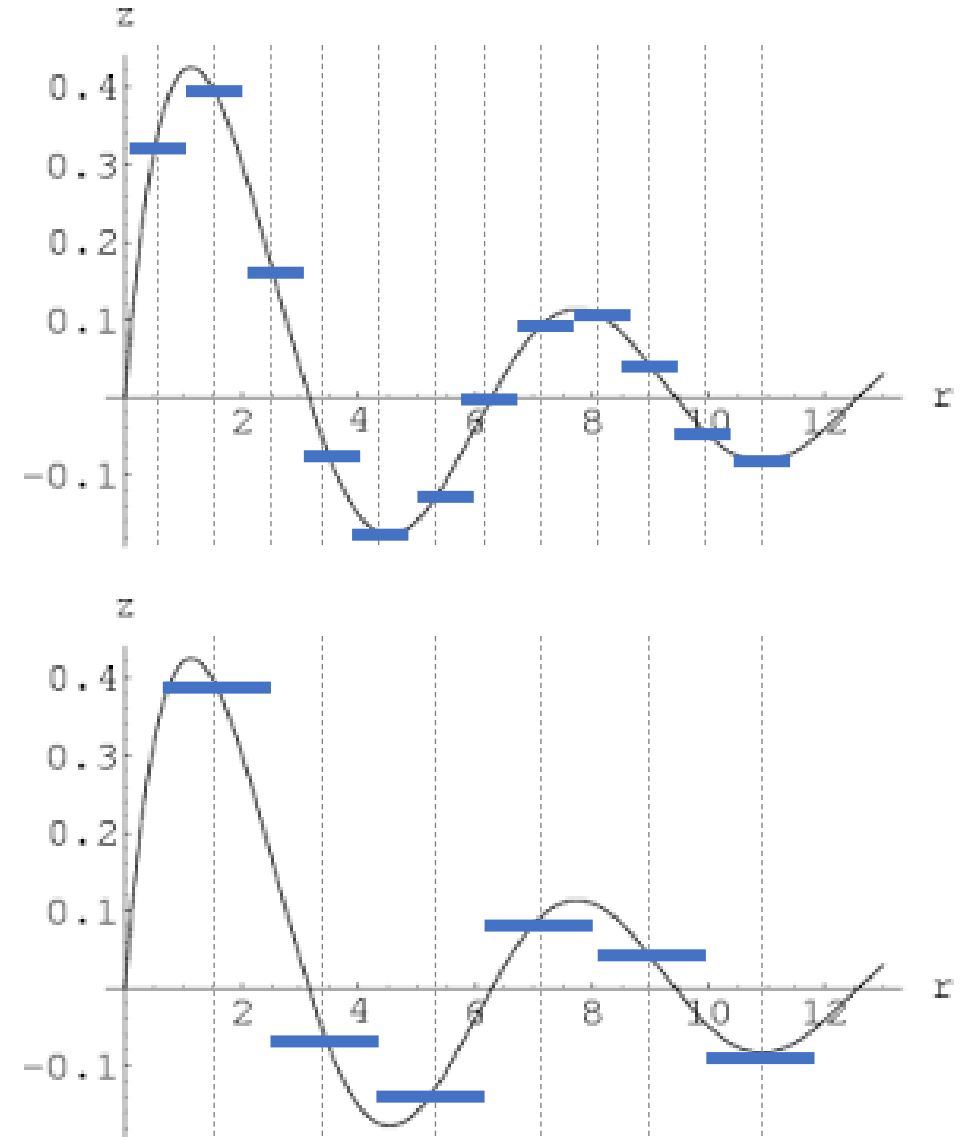


High resolution versus high frequency

- 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

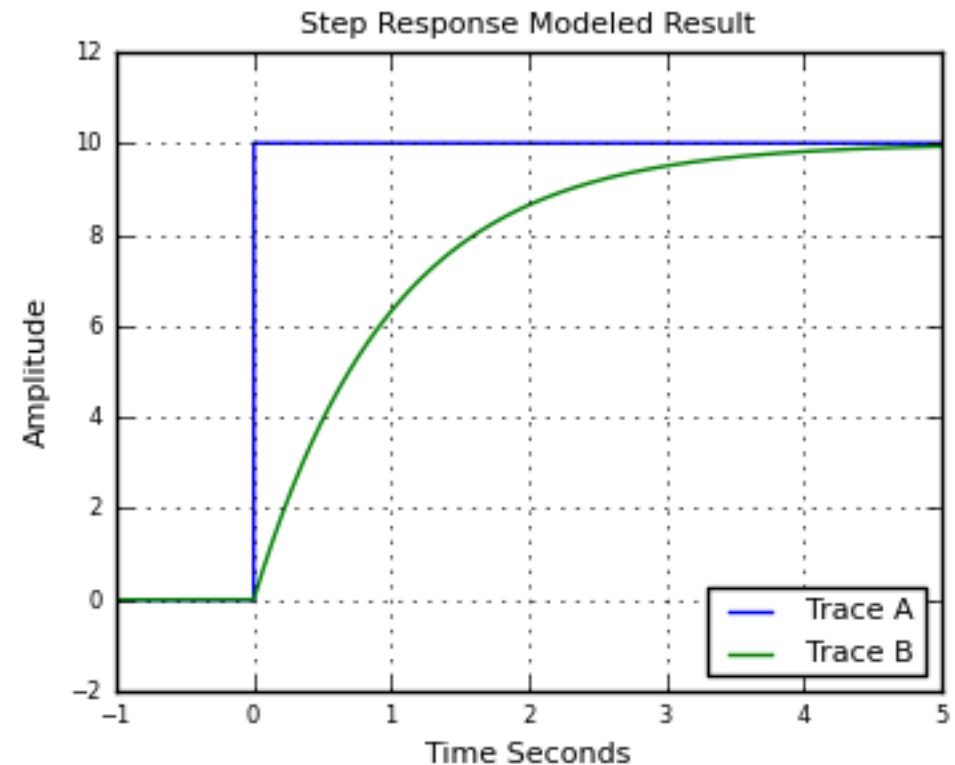
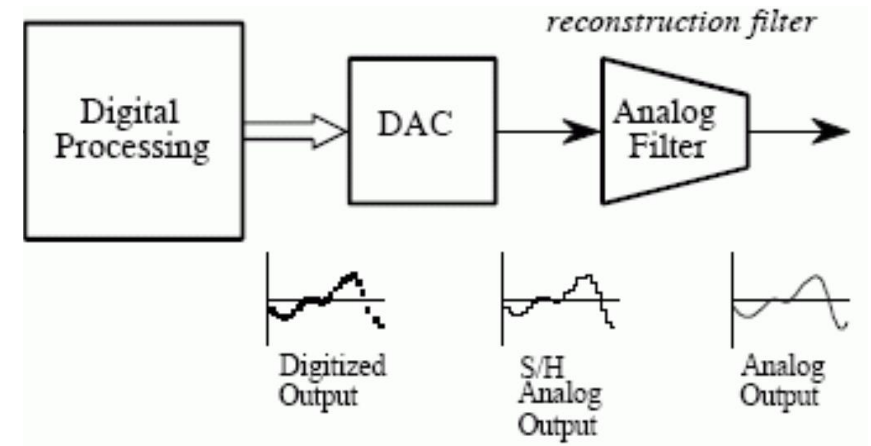
Infinite resolution is not sufficient

- 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



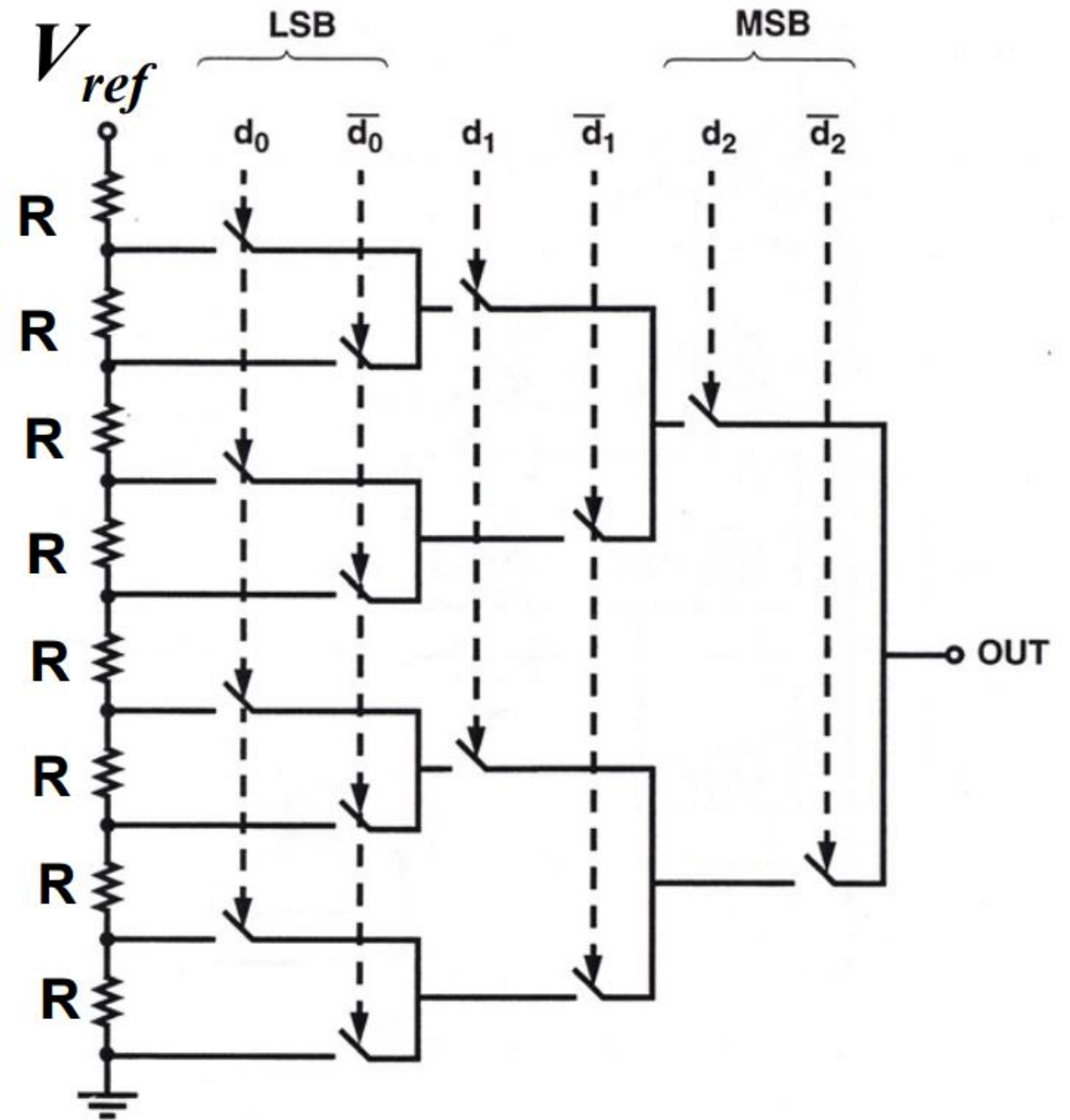
Low-pass filter smooths output

- 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



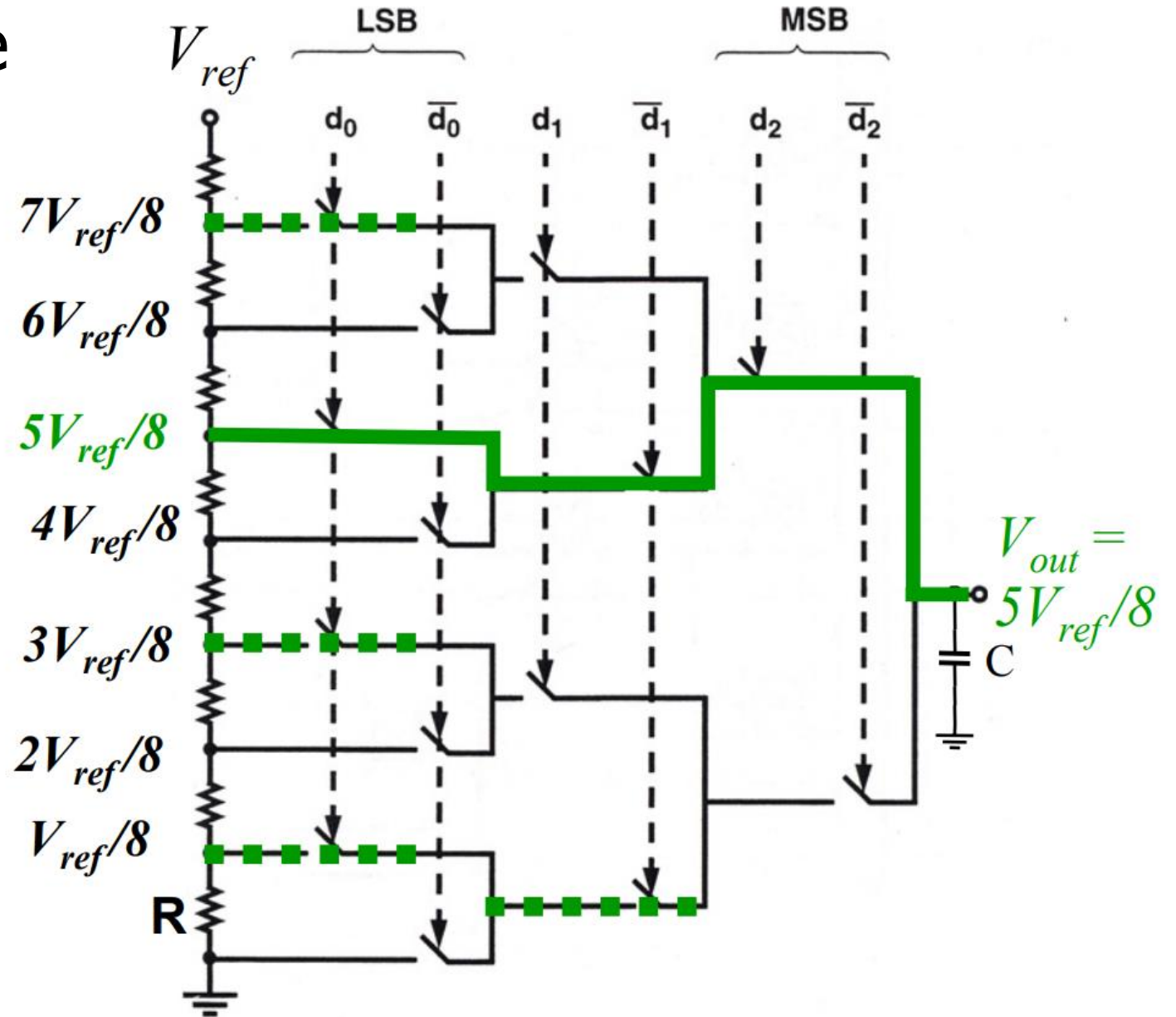
Resistor string DAC

- 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

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

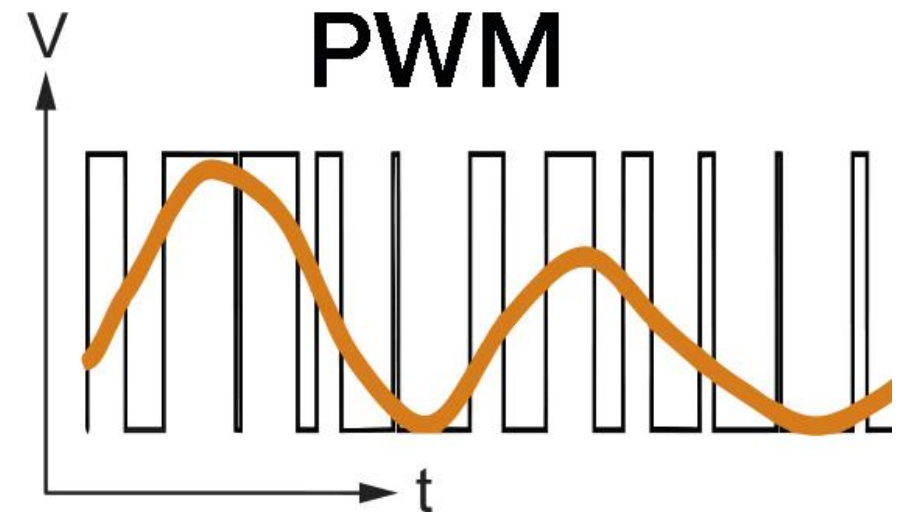
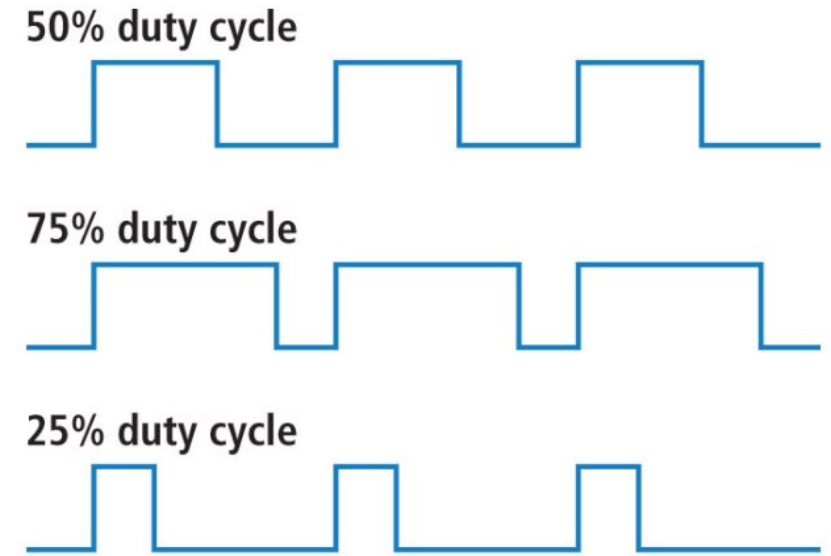
- Not rare, but not ubiquitous either
 - Every microcontroller has GPIO
 - Just about every microcontroller has an ADC
 - *Some* microcontrollers have DACs (the nRF52833 does not!)
- Reasons
 - Hardware is complicated (but we could fit it if we wanted)
 - Use cases are uncommon (and might need very high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

Outline

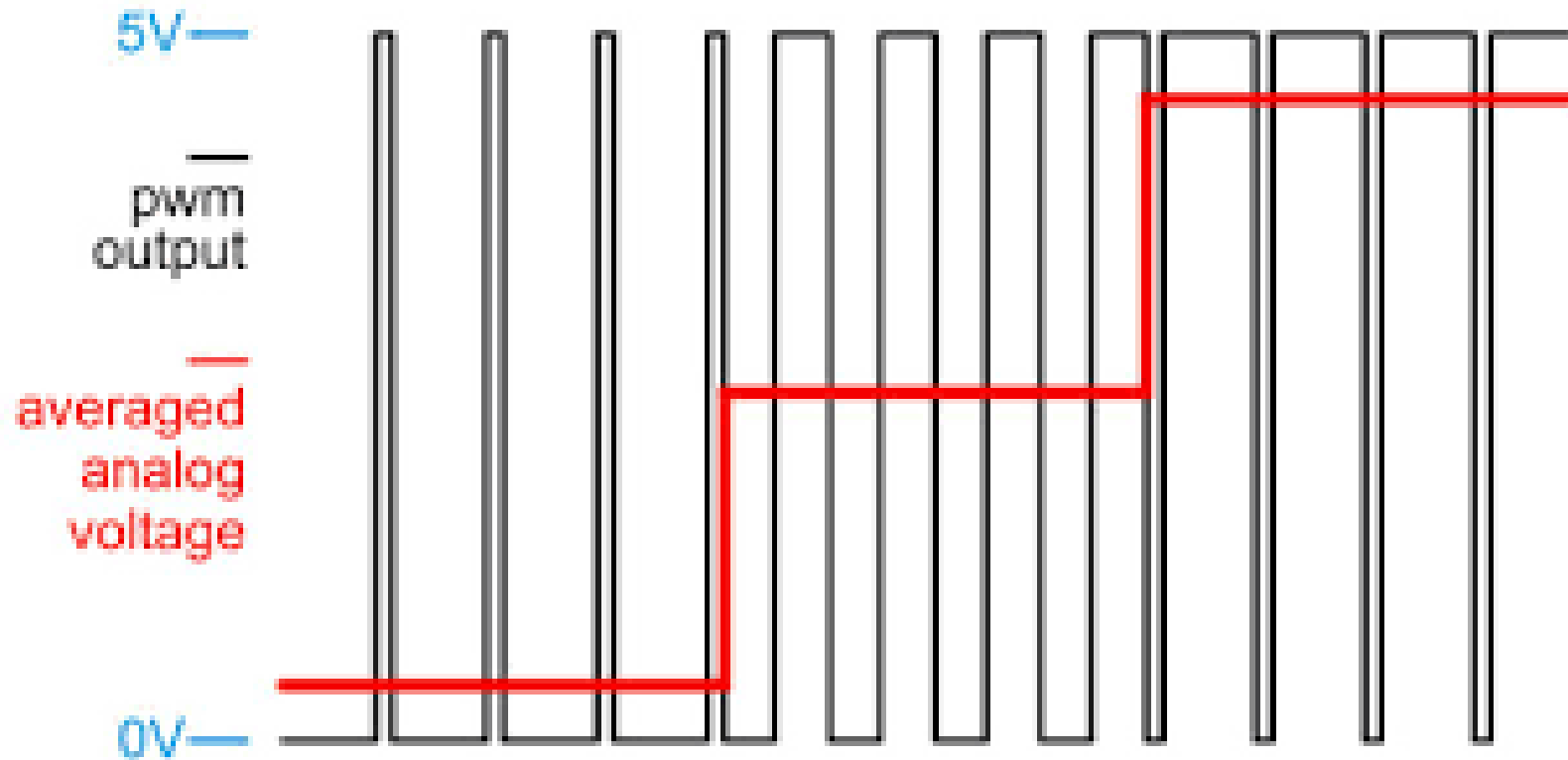
- Digital-to-Analog Converters
- **Pulse-Width Modulation**
- nRF52 PWM
- Using PWM

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
- Controls duty cycle, not frequency!



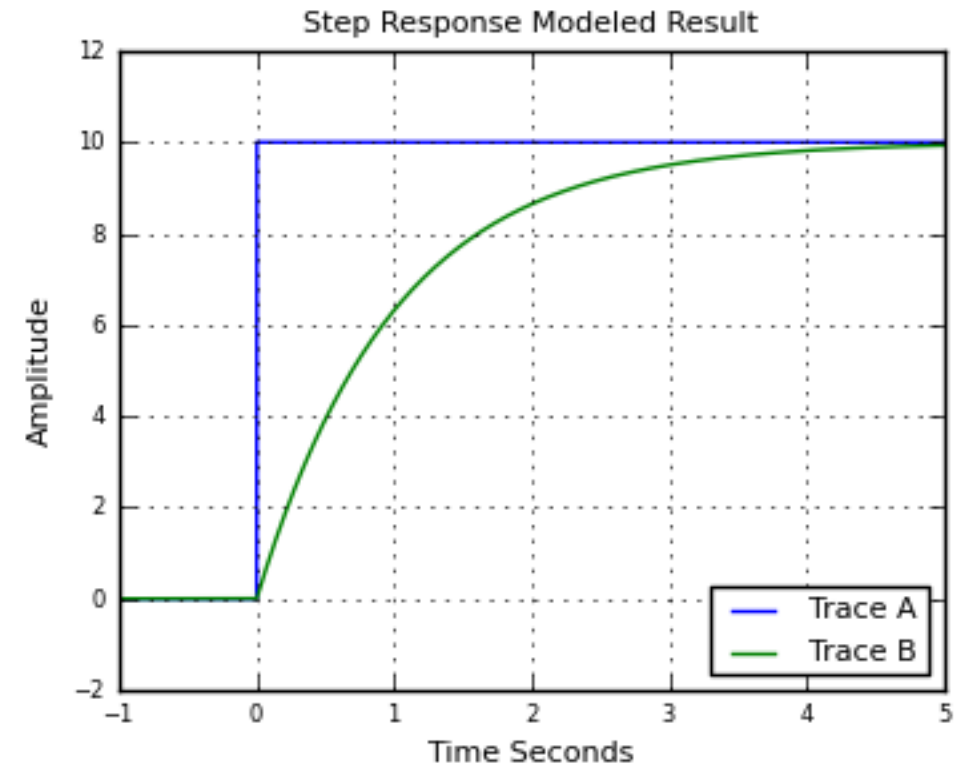
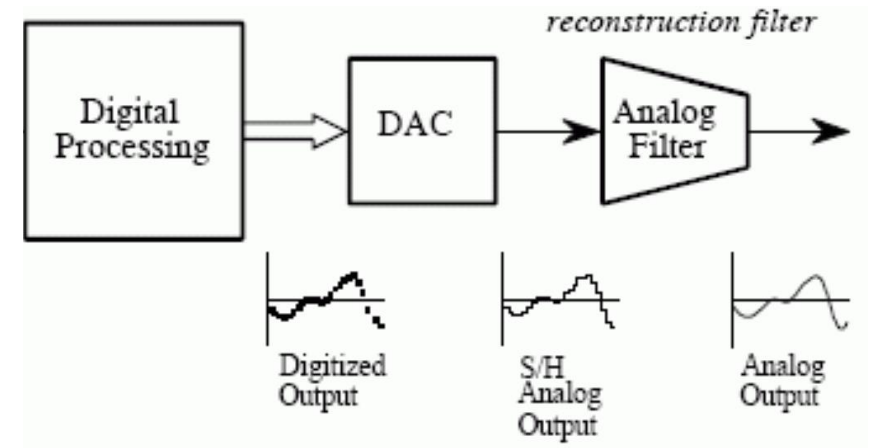
PWM to Analog Signal example



- 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

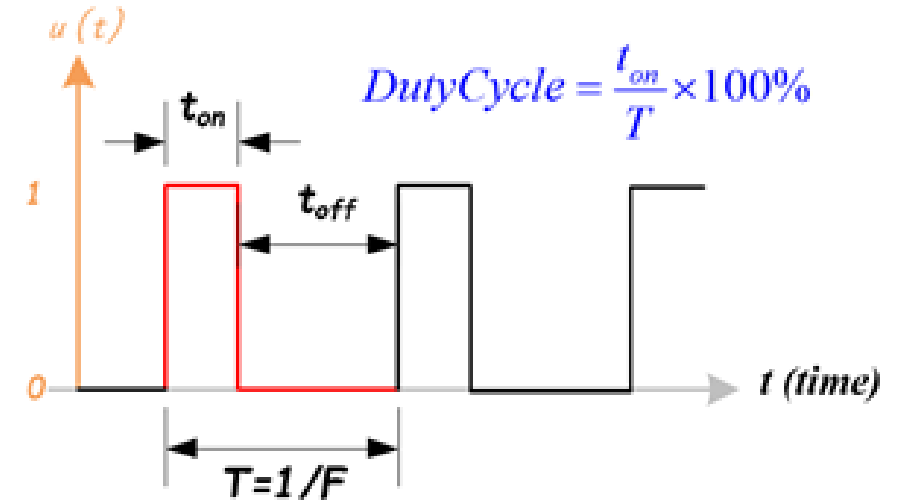
Low-pass approach works here too

- 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!



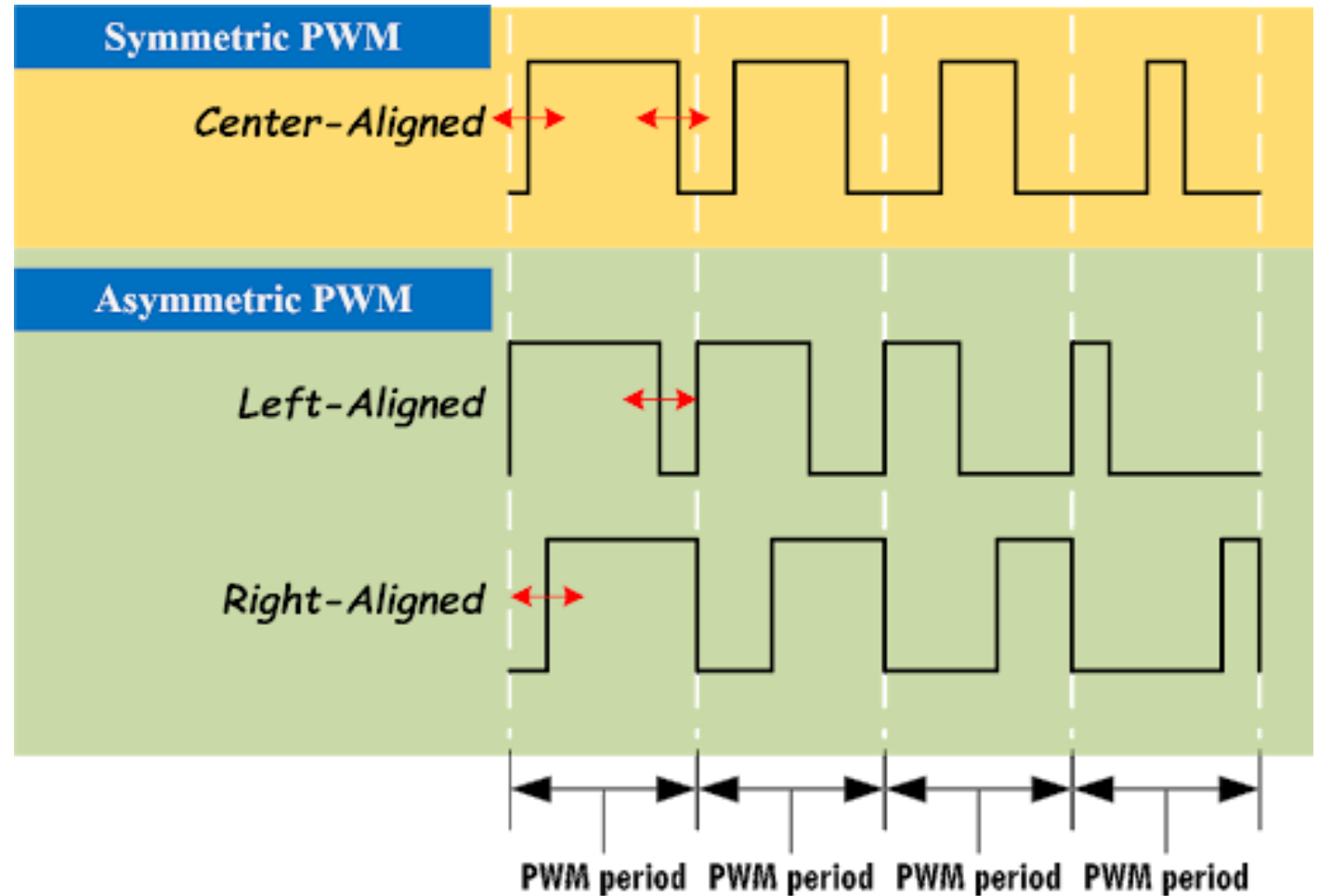
Controlling PWM

- 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 using an interrupt
 - Connect timer expiration to GPIO toggling

PWM is an example of encoding data on a signal

- 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

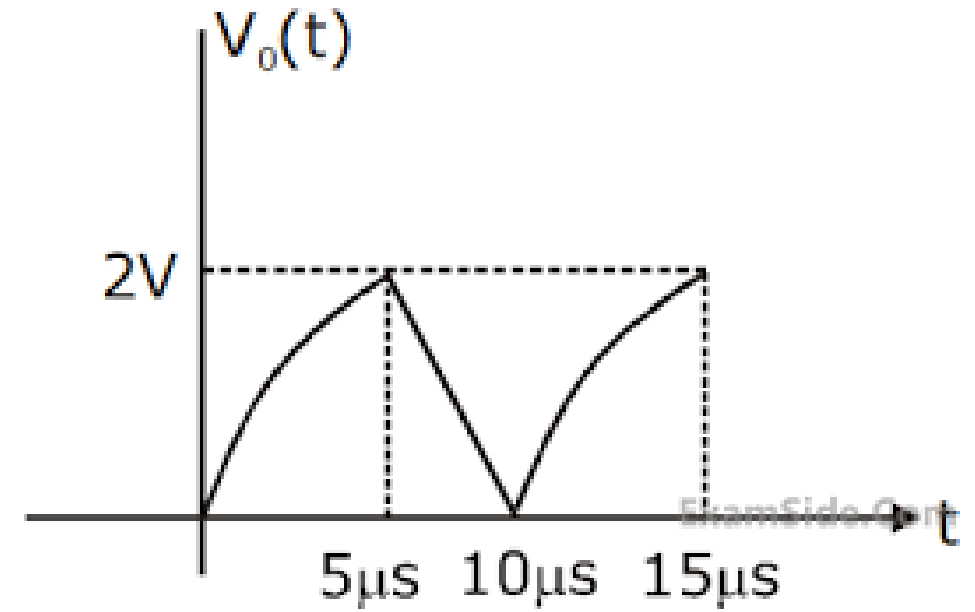
- Servos
 - Duty cycle chooses angle or rotation speed
- Motor controllers
 - Duty cycle chooses current and therefore speed
- LED brightness
 - And “breathing” effect
- Audio
 - Can sound okay if frequency is high enough

Controlling LED Matrix brightness

- Option 1: PWM peripheral
 - Use a single PWM output to control the column
 - When timer fires, change which column pin is used for PWM
 - PWM frequency should be much faster than row change frequency
 - Alternatively, use five PWM outputs to control rows for individual control
- Option 2: do it manually (instead of PWM peripheral)
 - Need to apply duty cycle on top of the existing stepping through rows
 - Add 5 new one-shot app timers, one for each column
 - Fire some percentage into the time the LED is active (within the 2 ms)
 - Use to toggle individual column LED back to off
 - Result: the LED is on for some portion of the time it should be active

Break + Open Question

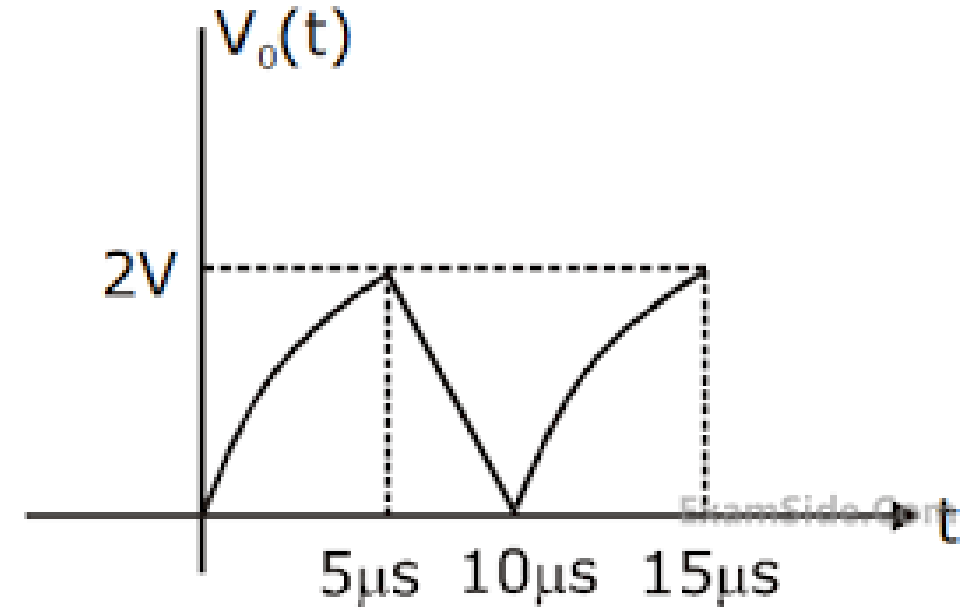
- Imagine you want to represent the following signal with PWM
 - **What should the PWM period be?**



- **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

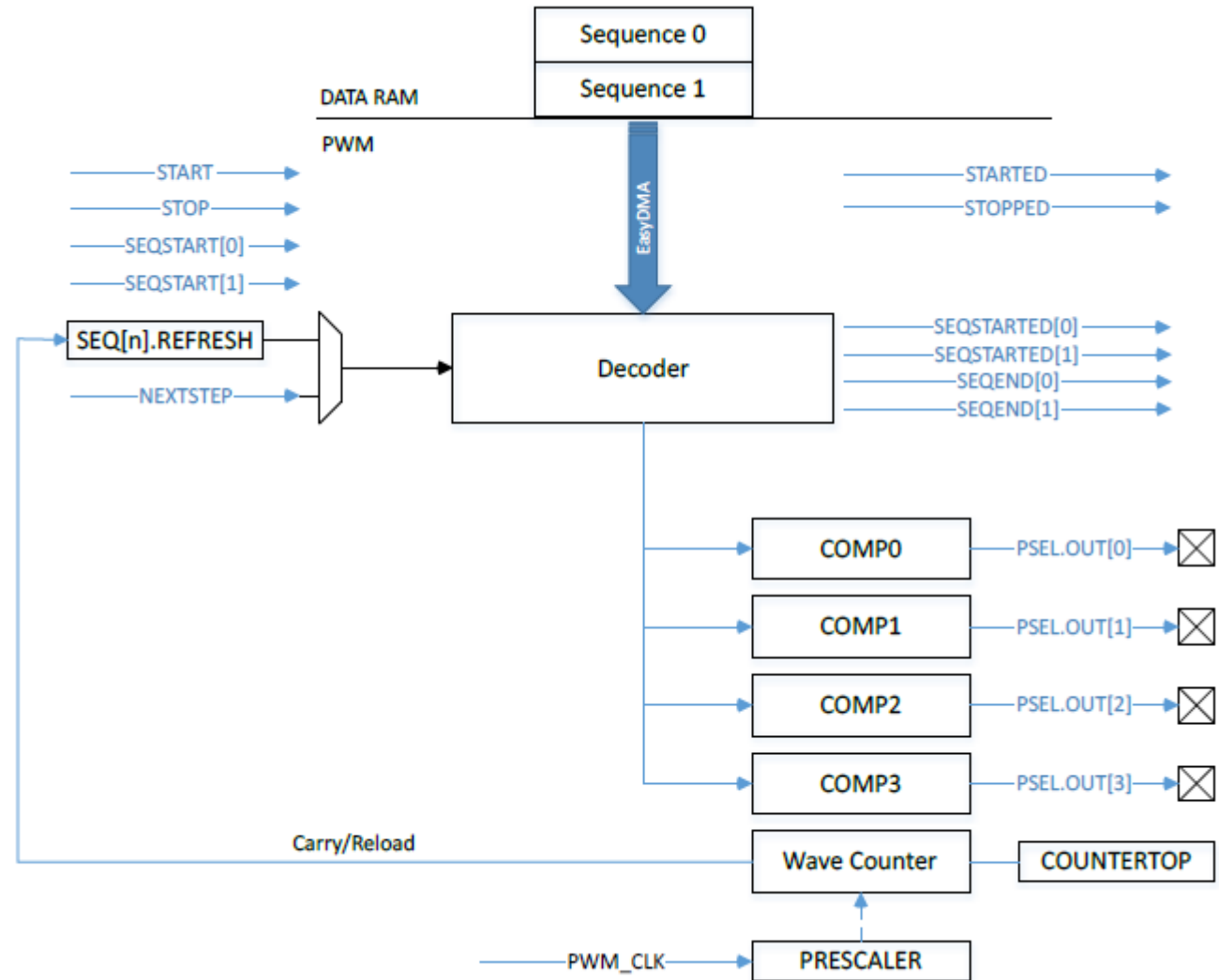
- Digital-to-Analog Converters
- Pulse-Width Modulation
- **nRF52 PWM**
- Using PWM

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

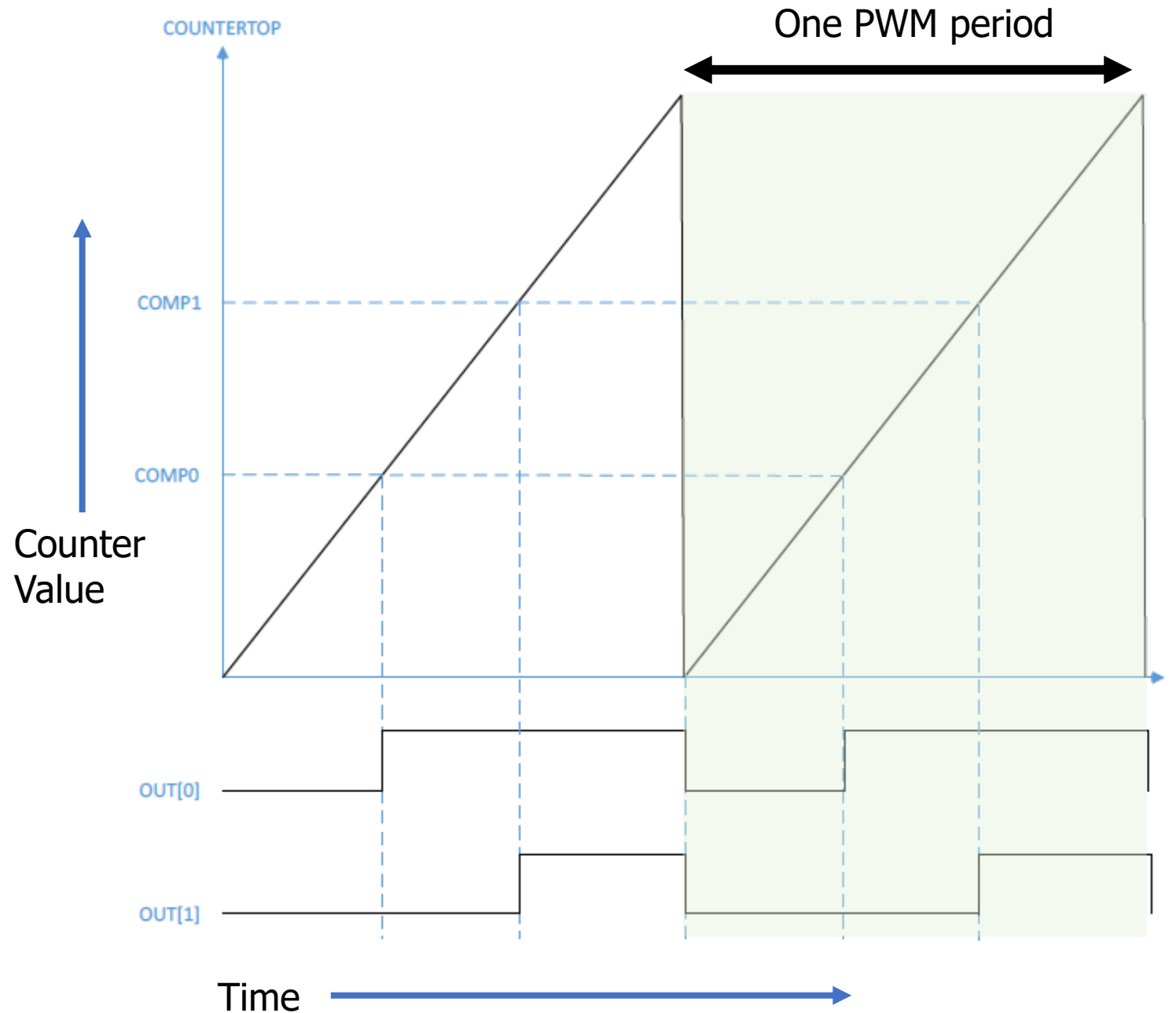
nRF52 PWM peripheral

- Uses internal timer to create PWM output on up to 4 pins
 - 4 peripheral instances, 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

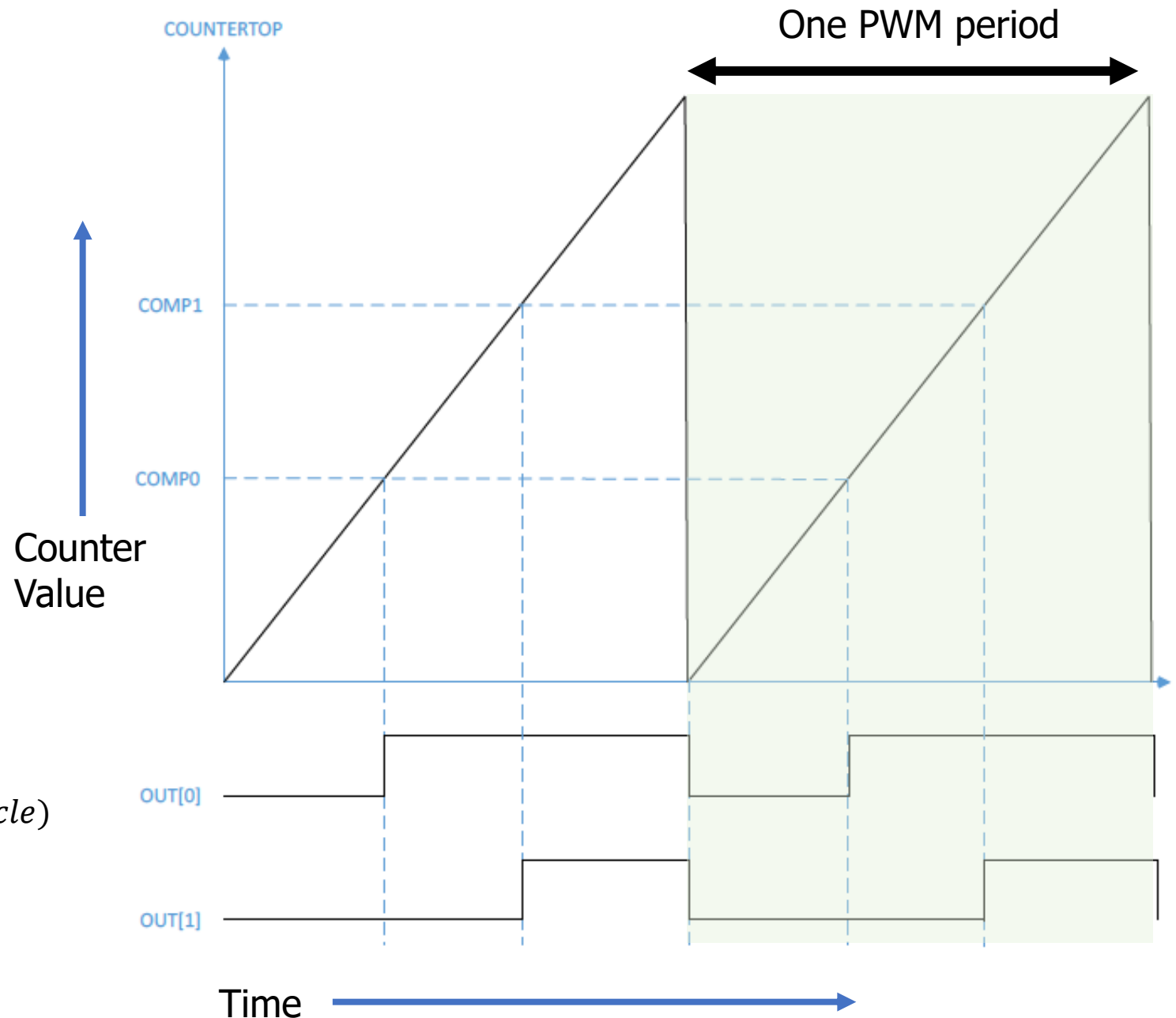


PWM example

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

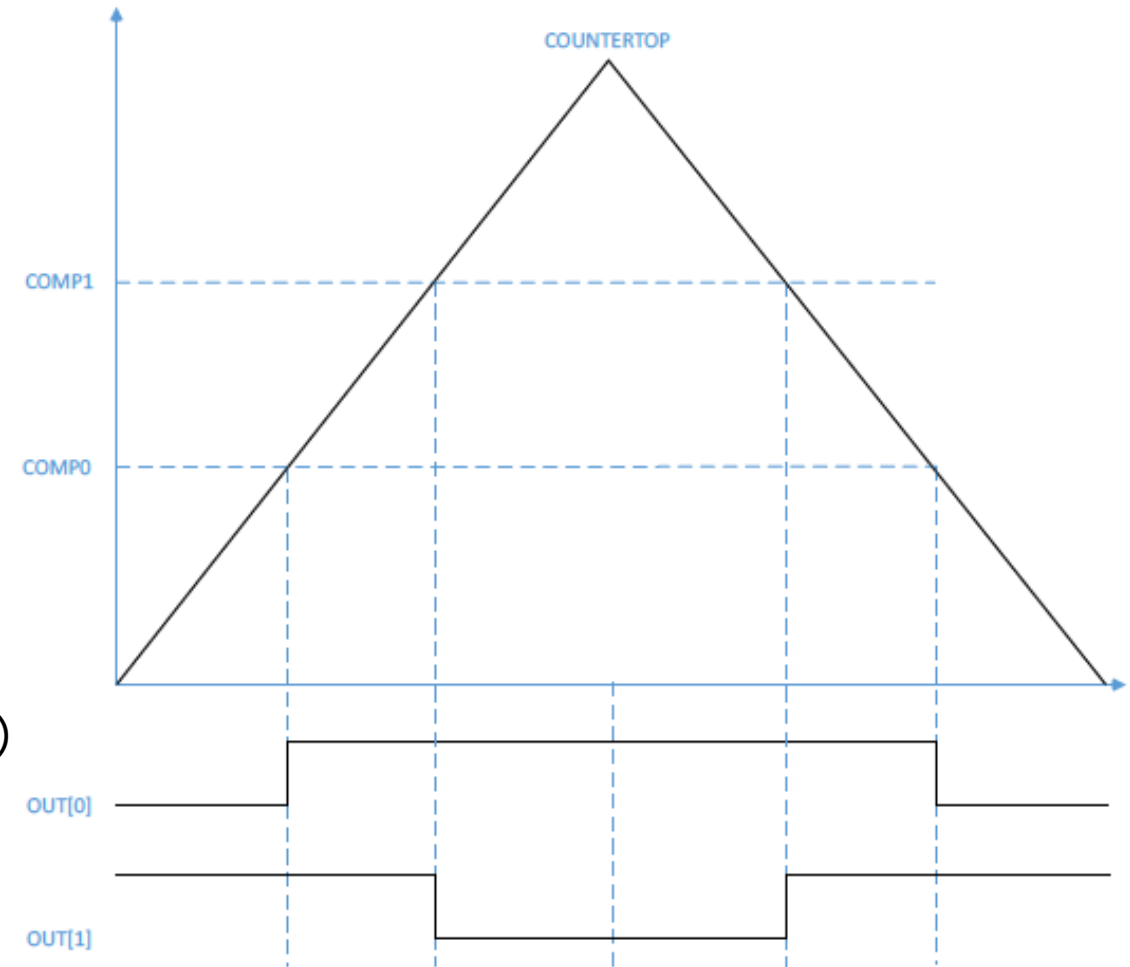
(left-aligned) $COMP = COUNTERTOP * DutyCycle$



Center-aligned PWM

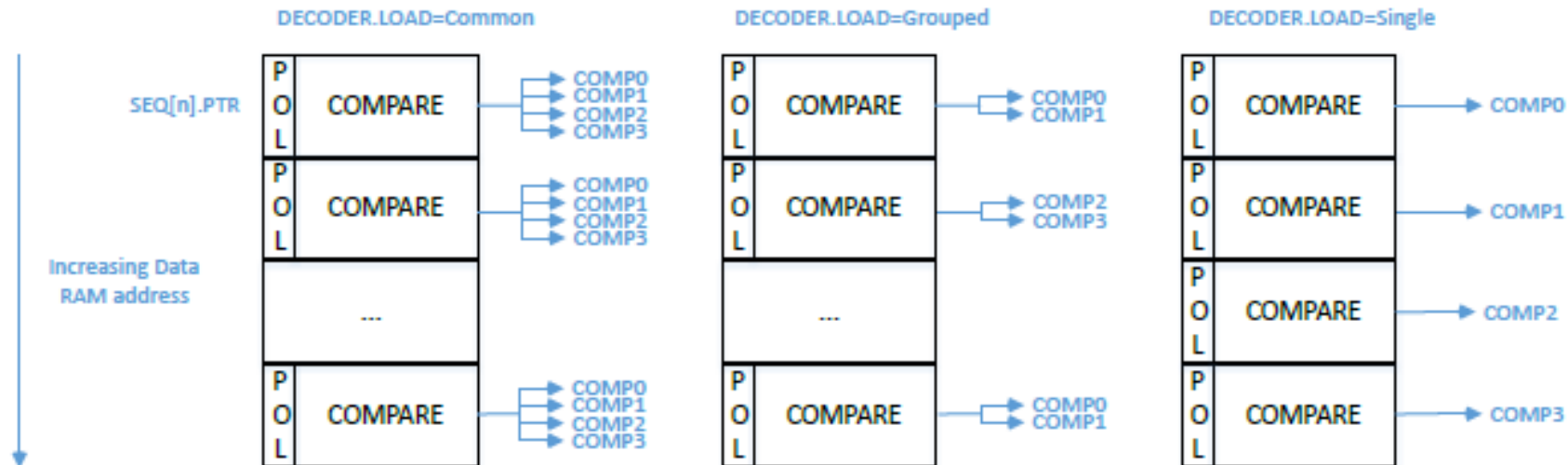
- Up-and-down mode enables center-aligned PWM
- Duty Cycle
 - Comp triggers toggle on rise
 - Comp triggers toggle again on fall

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



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

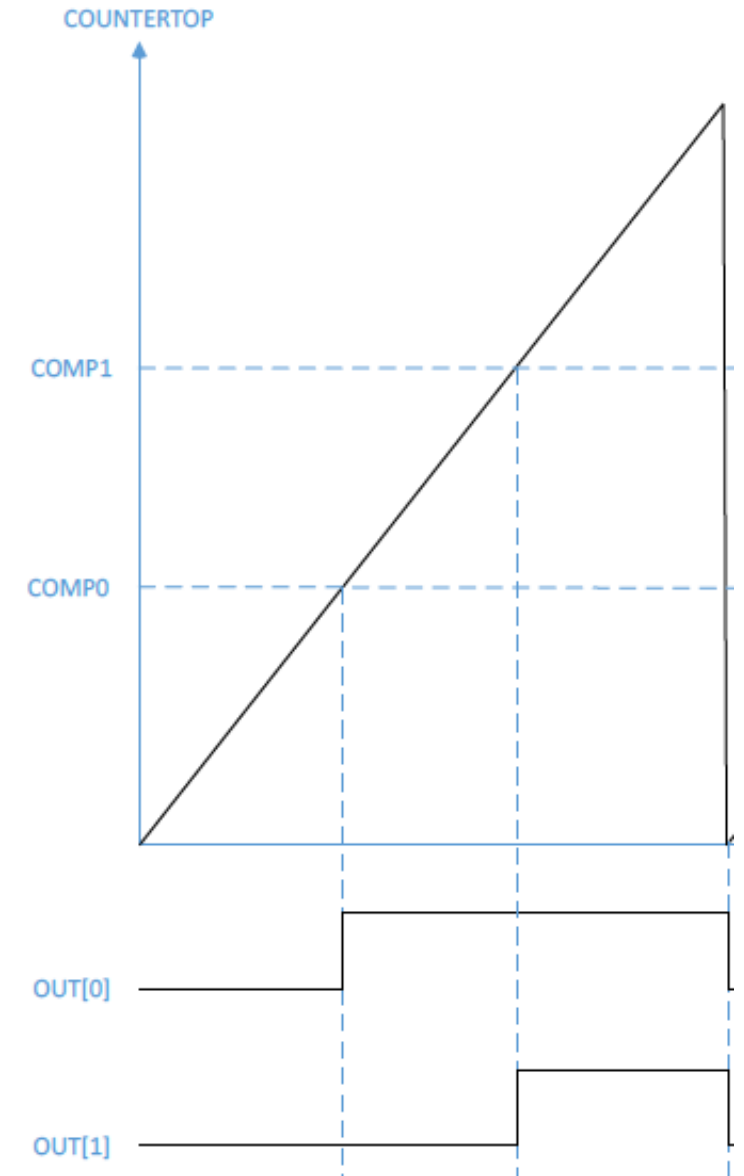


Other configurations

- 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 - “Double Buffering”
 - Allows continuous signal (for example, music)

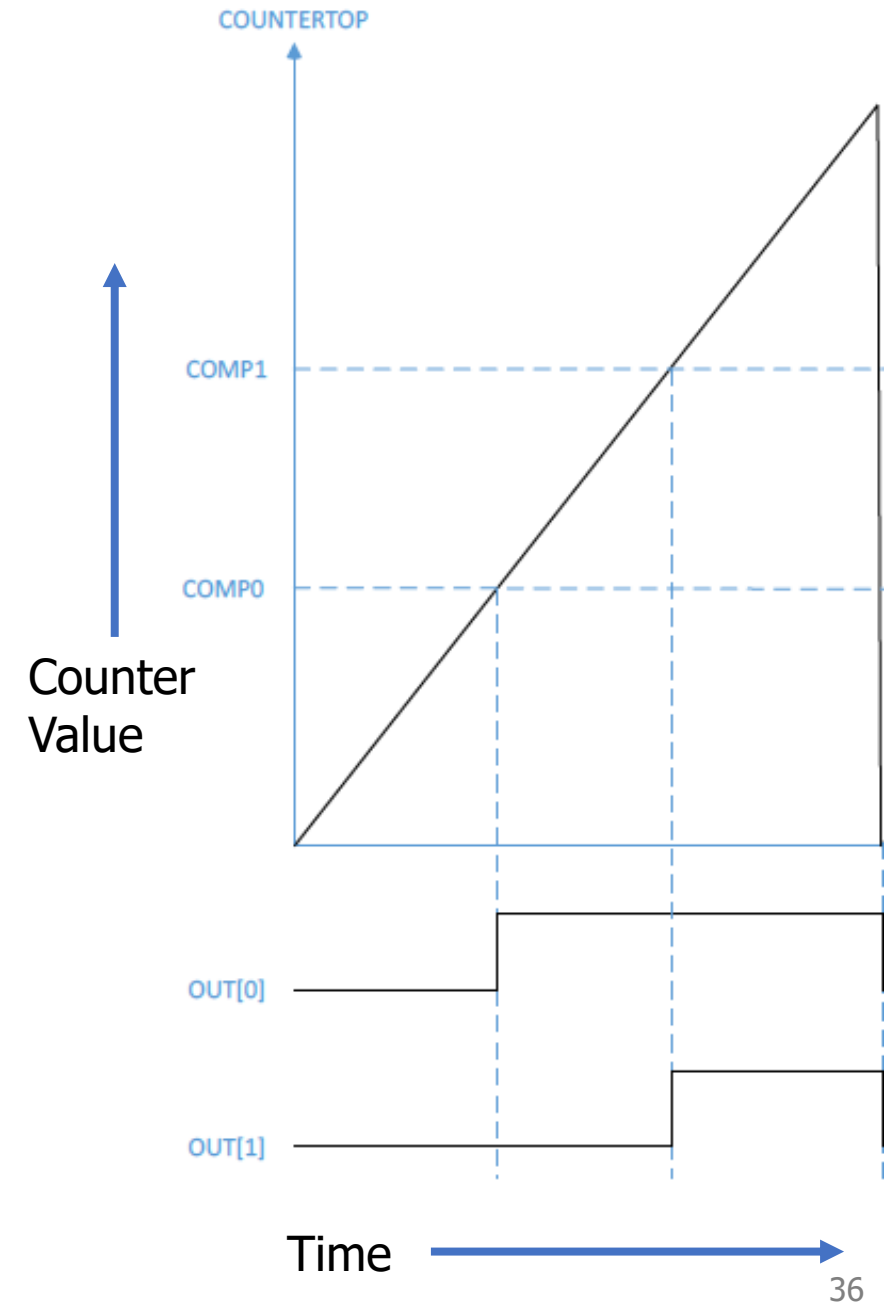
Trading speed and accuracy

- 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
 - Compare to audio output at 44.1 kHz



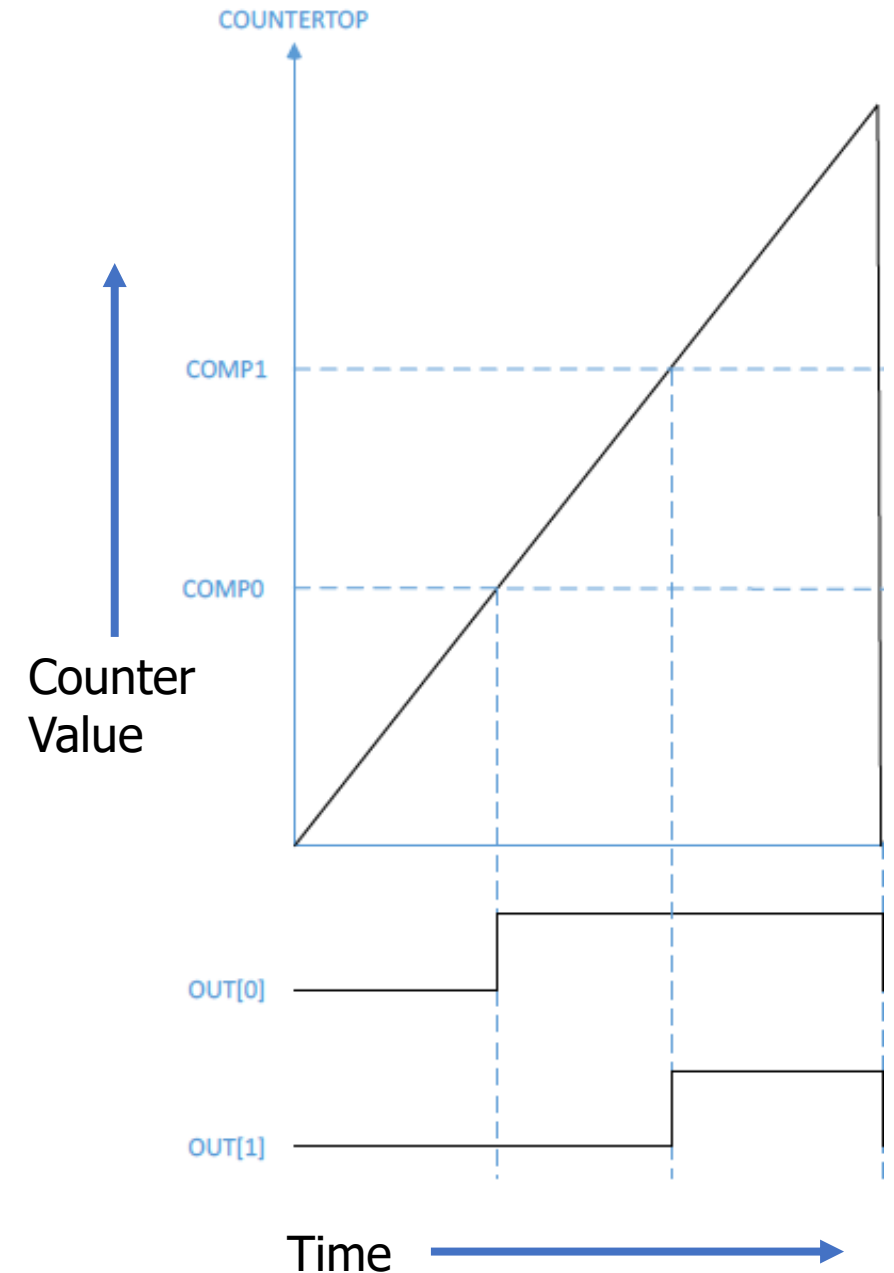
Break + Question

- Fastest PWM clock speed is 16 MHz (16000000 ticks per second)
- What COUNTERTOP value gets you 44.1 kHz? (44100 PWM cycles per second)
 - Hint: COUNTERTOP unit – Ticks
- How many bits in resolution would the corresponding DAC have?



Break + Question

- Fastest PWM clock speed is 16 MHz (16000000 ticks per second)
- What COUNTERTOP value gets you 44.1 kHz? (44100 PWM cycles per second)
 - Hint: COUNTERTOP unit – Ticks
 - $16 \text{ MHz} / 44.1 \text{ kHz} = 362.8$
- How many bits in resolution would the corresponding DAC have?
 - $\log_2(362) = 8.5$



Outline

- Digital-to-Analog Converters
- Pulse-Width Modulation
- nRF52 PWM
- **Using PWM**

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](#)

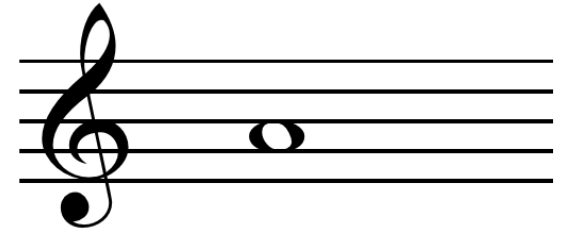
Number of times that each duty cycle is to be repeated (after being played once). Ignored in [NRF_PWM_STEP_TRIGGERED](#) mode.

[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

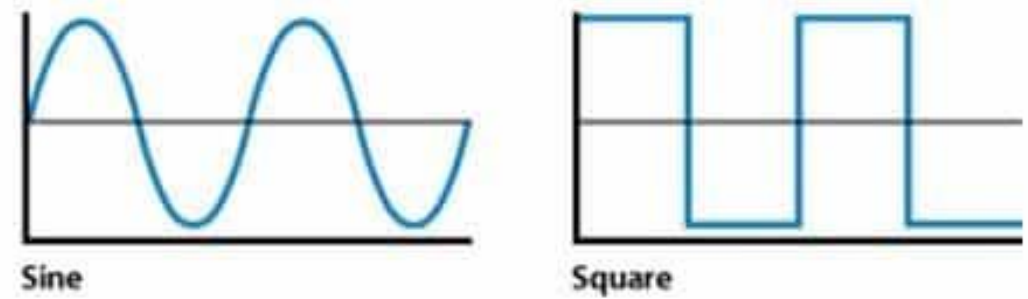
Example, playing a note with a square wave



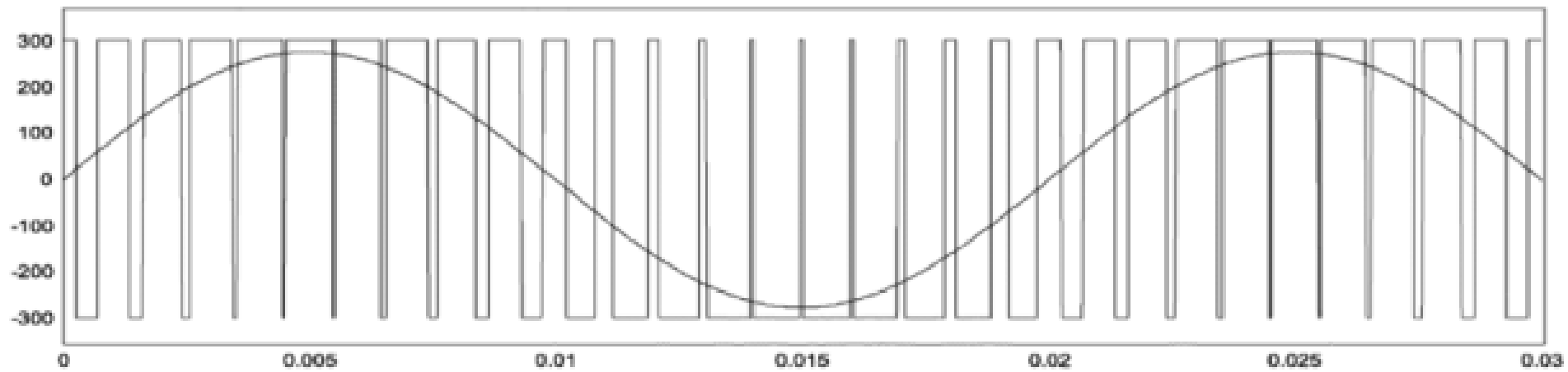
- Pick PWM frequency to match note frequency
 - Combination of PRESCALER, COUNTERTOP, and repeats
 - 440 Hz for the note A
 - PRESCALER 1 MHz, COUNTERTOP 2273 -> 440 Hz
- Set duty cycle of PWM to control volume
 - 50% duty cycle -> COMP value of 1137
- Set sequence with an array of length 1, content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

Playing sine waves instead of square waves

- Sine waves sound different than square waves
 - Sine is more smooth
 - Square is more harsh and buzzy



- For better sound quality, we can play sine waves via PWM

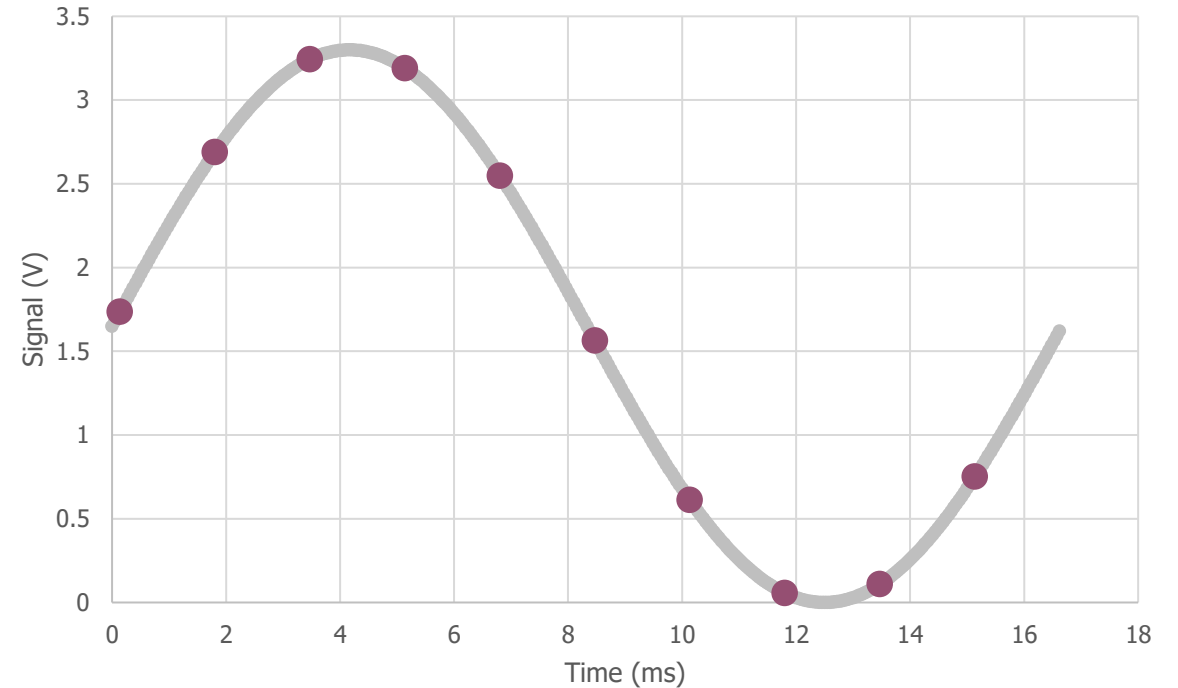


Steps to playing a note with a sine wave

1. Determine sampling rate and set PRESCALAR and COUNTERTOP
2. Calculate sine wave values for some amount of duration
 - Reasonable to calculate one second of data at a time
 - `sin()` function might be too slow here
 - Instead calculate sine values in advance and interpolate into an array based on frequency
3. Translate sine wave values into duty cycle values
 - Fill an array with these duty cycle values
4. Play the array of duty cycle values
 - That gets one second of audio
 - For music, might change schedule a timer for when to stop current playback and play the next note instead

Example: playing an arbitrary signal

- Goal: recreate this analog signal
- Have to pick a “sampling rate” for representing the signal
- The purple dots are sampled at 600 Hz (10 samples per wave)
 - We’ll use the PWM to play those values

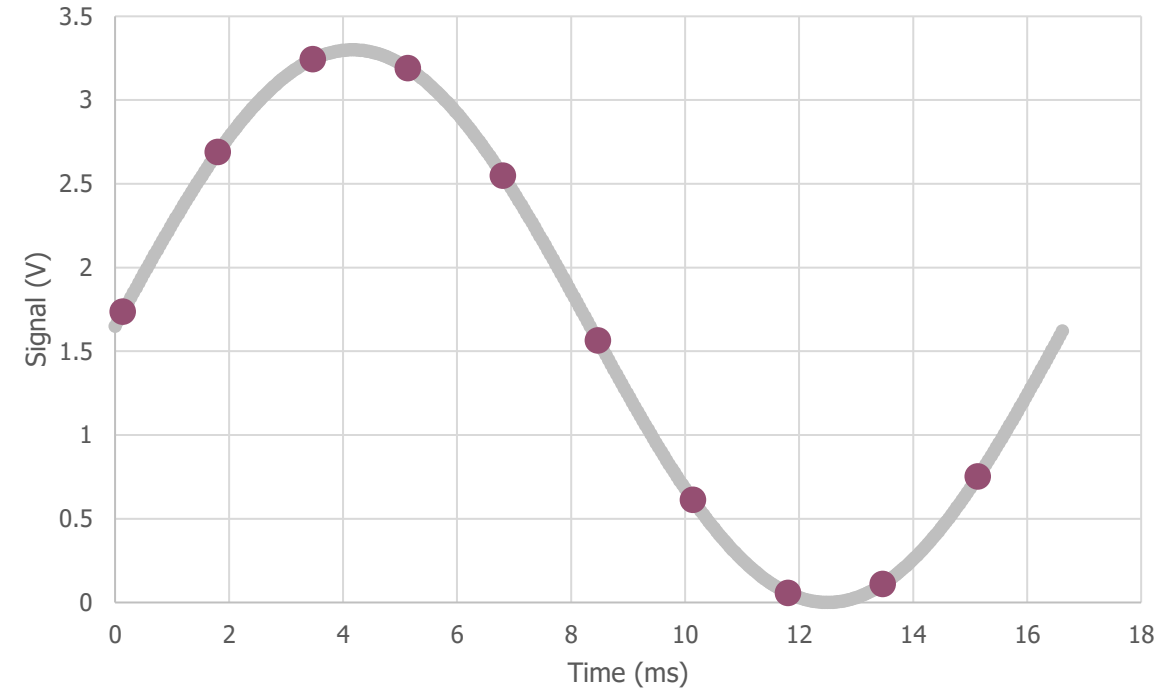


Step 1: determine PRESCALAR and COUNTERTOP

- PRESCALAR
 - Let's just set the PWM to maximum rate: 16 MHz
- COUNTERTOP (**you determine this**)
 - $16,000,000 \text{ ticks per second} / 600 \text{ cycles per second} = 26,667 \text{ ticks per cycle}$
 - Set COUNTERTOP to 26667
 - If we wanted repeats, we'd further decrease this
 - 10 repeats -> COUNTERTOP = 2667

Step 2: calculate signal values

Sample Time (ms) Value (Volts)		
1	0.139	1.736
2	1.806	2.688
3	3.472	3.244
4	5.139	3.190
5	6.806	2.549
6	8.472	1.564
7	10.139	0.612
8	11.806	0.056
9	13.472	0.110
10	15.139	0.751



- Voltage values calculated from the signal
 - Need to convert to values relative to COUNTERTOP

Converting voltage to COMP values

- Sample 0: **(you determine this)**

- Duty cycle:

- = 1.736 volts / 3.3 volts

- = 53%

- COMP = Duty cycle * COUNTERTOP:

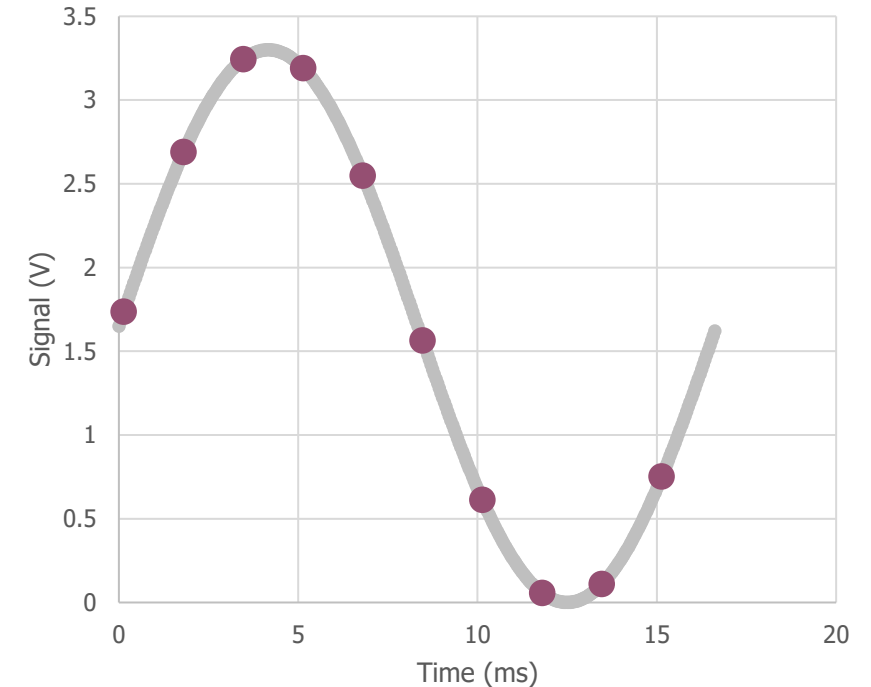
- = 1.736 / 3.3 * 26667

- = 14031

Sample Time (ms)		Value (Volts)
1	0.139	1.736
2	1.806	2.688
3	3.472	3.244
4	5.139	3.190
5	6.806	2.549
6	8.472	1.564
7	10.139	0.612
8	11.806	0.056
9	13.472	0.110
10	15.139	0.751

Resulting COMP values

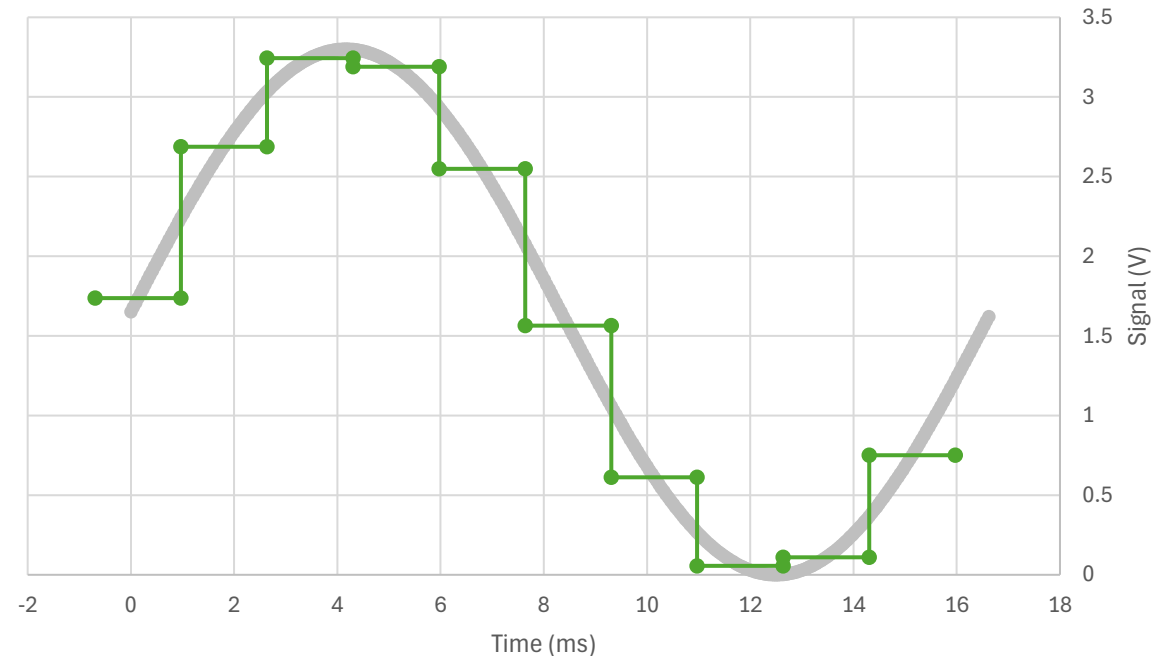
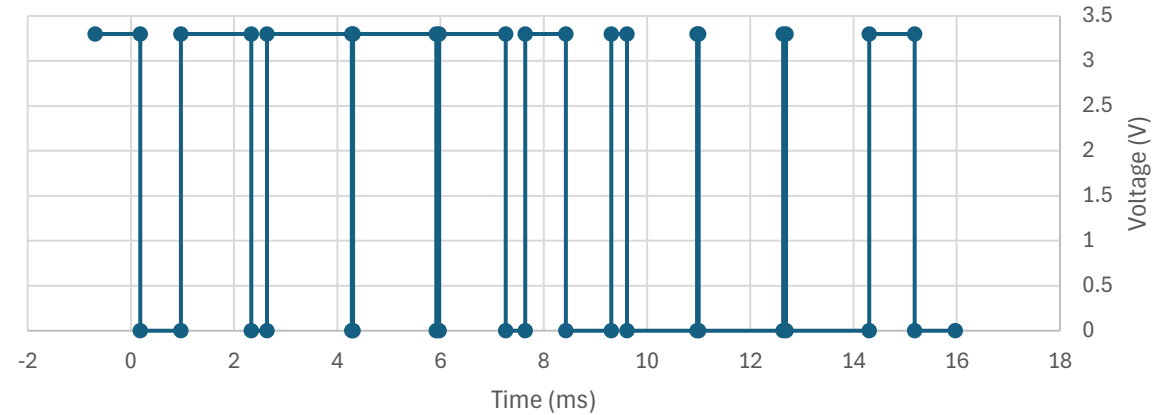
Sample	Time (ms)	Value (Volts)	Value (Duty Cycle)	Value (COMP)
1	0.139	1.736	53%	14031
2	1.806	2.688	81%	21725
3	3.472	3.244	98%	26213
4	5.139	3.190	97%	25781
5	6.806	2.549	77%	20595
6	8.472	1.564	47%	12636
7	10.139	0.612	19%	4942
8	11.806	0.056	2%	454
9	13.472	0.110	3%	886
10	15.139	0.751	23%	6072



- So sequence_data values will be:
{14031, 21725, 26213, 25781, 20595, 12636, 4942, 454, 886, 6072}

Resulting signal

- Blue lines are the actual PWM signal we generate
- Green lines are the averaged voltage output of the PWM
 - Faster sampling rate would better represent the signal
- Note: if exact start time of wave matters, need to start PWM a little early (half a sample)



Outline

- Digital-to-Analog Converters
- Pulse-Width Modulation
- nRF52 PWM
- Using PWM