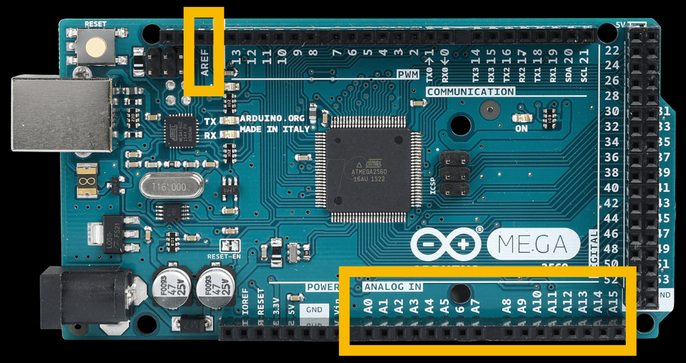
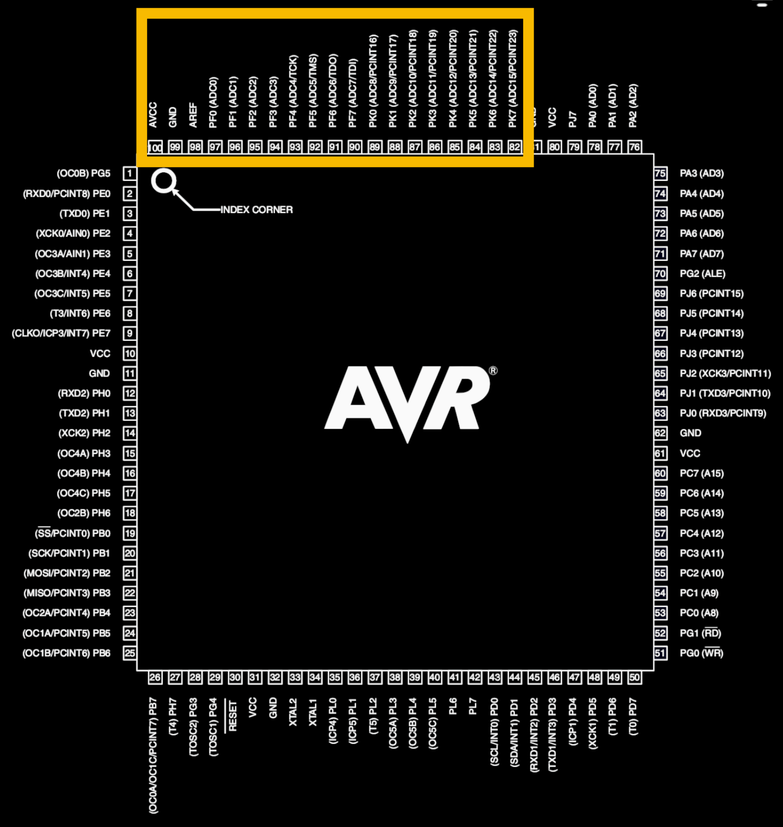
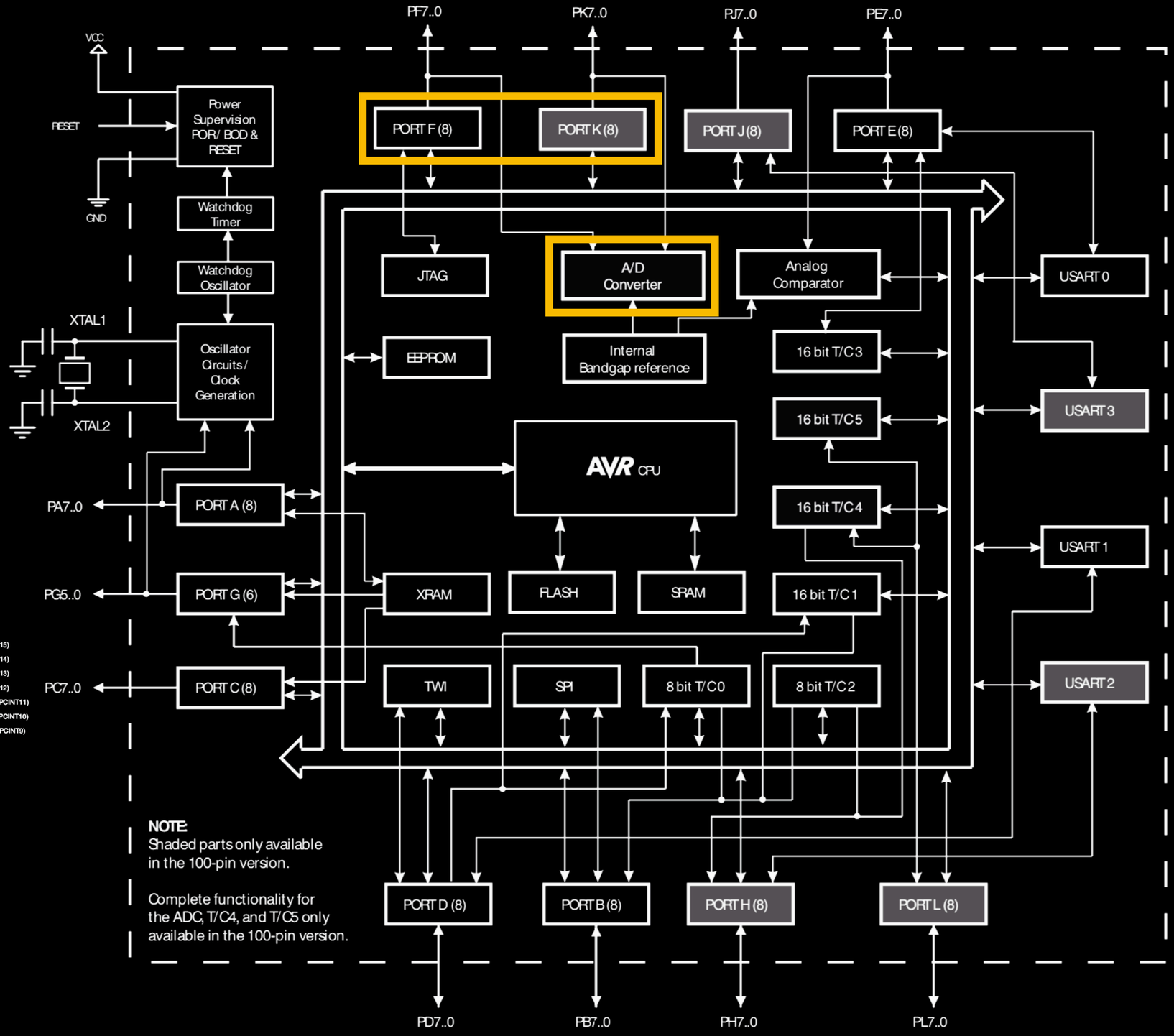
ADC


AREF
PWM
COMMUNICATION
ARDUINO.ORG
MADE IN ITALY
MEGA
POWER
ANALOG IN


AVCC
GND
AREF
PF0 (ADC0)
PF1 (ADC1)
PF2 (ADC2)
PF3 (ADC3)
PF4 (ADC4/TCK)
PF5 (ADC5/TMS)
PF6 (ADC6/TDO)
PF7 (ADC7/TDI)
PK0 (ADC8/PCINT16)
PK1 (ADC9/PCINT17)
PK2 (ADC10/PCINT18)
PK3 (ADC11/PCINT19)
PK4 (ADC12/PCINT20)
PK5 (ADC13/PCINT21)
PK6 (ADC14/PCINT22)
PK7 (ADC15/PCINT23)
INDEX CORNER
AVR


VCC
RESET
GND
Power Supervision POR/ BOD & RESET
Watchdog Timer
Watchdog Oscillator
XTAL1
XTAL2
Oscillator Circuits / Clock Generation
PA7..0
PORT A (8)
PG5..0
PORT G (6)
PC7..0
PORT C (8)
PF7..0
PK7..0
PJ7..0
PE7..0
PORT F (8)
PORT K (8)
PORT J (8)
PORT E (8)
JTAG
A/D Converter
Analog Comparator
USART 0
EEPROM
Internal Bandgap reference
16 bit T/C 3
16 bit T/C 5
AVR CPU
USART 3
16 bit T/C 4
USART 1
XRAM
FLASH
SRAM
16 bit T/C 1
TWI
SPI
8 bit T/C 0
8 bit T/C 2
USART 2
NOTE:
Shaded parts only available in the 100-pin version.
Complete functionality for the ADC, T/C4, and T/C5 only available in the 100-pin version.
PORT D (8)
PORT B (8)
PORT H (8)
PORT L (8)
PD7..0
PB7..0
PH7..0
PL7..0

26.8 ADC Register Description

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
ADSCRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADSCRB	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0

	15	14	13	12	11	10	9	8
ADCH	–	–	–	–	–	–	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
	7	6	5	4	3	2	1	0

Improving on the R-Meter

- Differential measurement
- Programmable Gain

ADMUX - “ADC Multiplexer Selection Register”

Vertical scaling, formatting of conversion results, and channel selection

“[Voltage] Reference Selection”

$$ADC = \frac{1024 \cdot V_{IN}}{V_{REF}}$$

“Left Adjust Result”

- 10-bit result stored in two 8-bit registers
  - (1) left adjusted
  - (0) right adjusted

“Analog Channel and Gain Selection [Multiplexer]”

- Choose single pins for single-ended measurement
  - or use pin pairs for differential measurement
- Some channels have multiple gain options

ADMUX: Voltage Reference

$$ADC = \frac{1024 \cdot V_{IN}}{V_{REF}}$$

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1	REFS0	Voltage Reference Selection[1]
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

ADMUX: Voltage Reference

$$ADC = \frac{1024 \cdot V_{IN}}{V_{REF}}$$

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1	REFS0	Voltage Reference Selection[1]
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- External AREF (0) is useful for conditioned or variable references
- AVCC (1) gives maximum voltage input range
  - Arduino Mega defaults to this and includes the external capacitor
- Internal references (2, 3) are useful for increasing resolution for small signals

ADMUX: Left Adjust ADC Results

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

ADLAR = 0
(Right Adjusted)

	15	14	13	12	11	10	9	8
ADCH	–	–	–	–	–	–	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
	7	6	5	4	3	2	1	0

```
// Get 10-bit conversion result
uint8_t lsb = ADCL;
uint8_t msb = ADCH;
uint16_t result = (msb << 8) | lsb;
```

*Note: the ADC updates after ADCH is read, so if you need ADCL, read it first!

ADMUX: Left Adjust ADC Results

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

ADLAR = 1
(Left Adjusted)

	15	14	13	12	11	10	9	8
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	–	–	–	–	–	–
	7	6	5	4	3	2	1	0

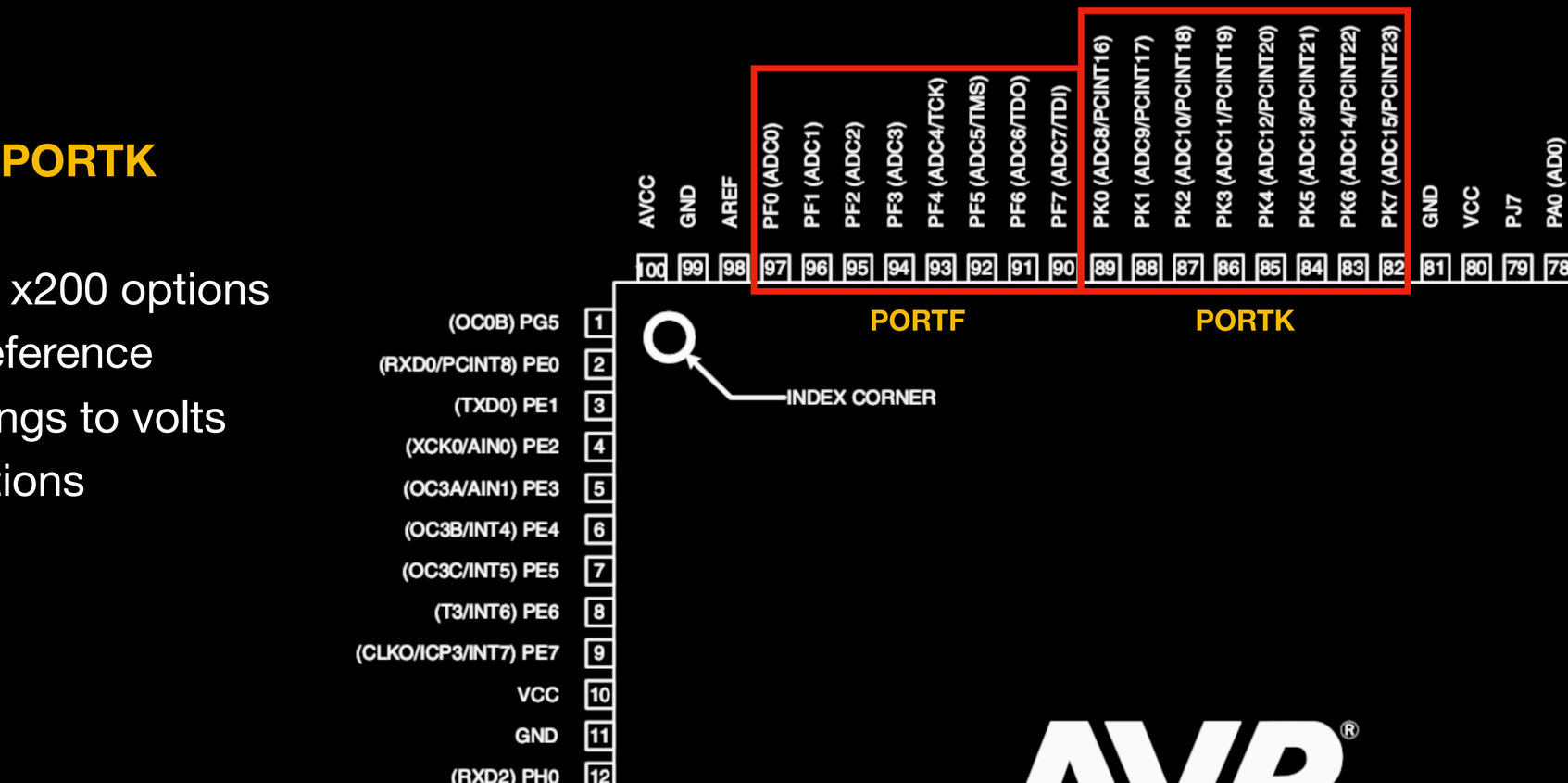
```
// Get 8-bit conversion result
uint8_t result = ADCH;
```

ADMUX: Analog Channel Selection

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Note: There is also a MUX5 bit in register **ADSCRB**

- Read from pins on **PORTF** and **PORTK**
- Single ended or differential
- Channels 3:0 have x1, x10, and x200 options
- Also read from bandgap 1.1V reference
  - Useful for scaling ADC readings to volts
- See sec 26.8.2 for full list of options



ADSCR: ADC Control and Status Registers

Configure to trigger conversions manually or automatically from a number of sources

	7	6	5	4	3	2	1	0
ADSCRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADSCRB	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0

“Enable”

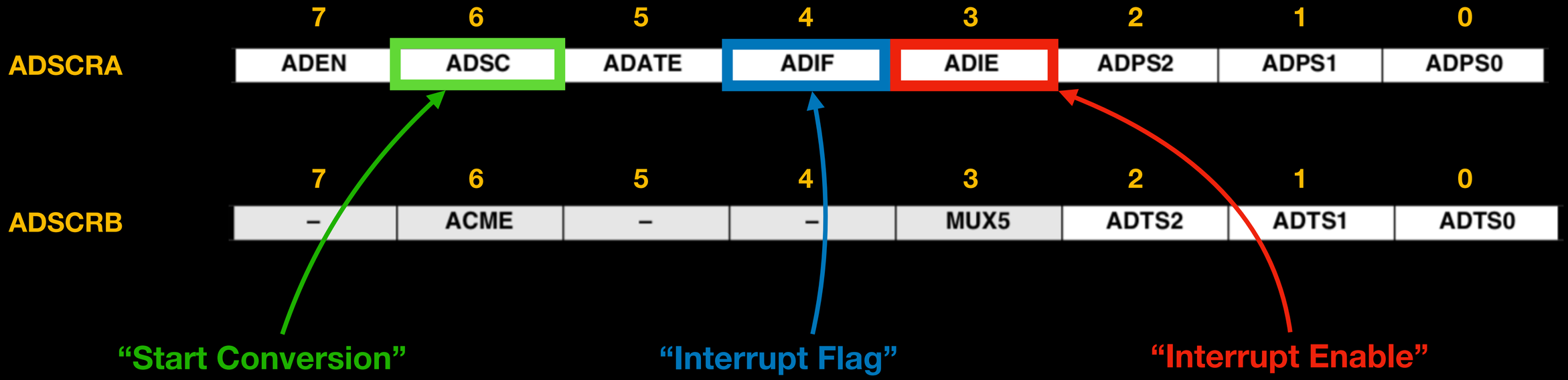
- Set bit to use ADC
- Clear to disable

```
// Enable the ADC
ADSCRA |= 0b10000000;
ADSCRA |= 0x80;
ADSCRA |= 128;
ADSCRA |= 1 << 7;
ADSCRA |= 1 << ADEN;
ADSCRA |= _BV(ADEN);
```

```
// Disable the ADC
ADSCRA &= 0b01111111;
ADSCRA &= ~0b10000000;
ADSCRA &= ~0x80;
ADSCRA &= ~128;
ADSCRA &= ~(1 << 7);
ADSCRA &= ~(1 << ADEN);
ADSCRA &= ~_BV(ADEN);
```

ADSCR: ADC Control and Status Registers

Useful bits for manually initiating single conversions à la `analogRead()`



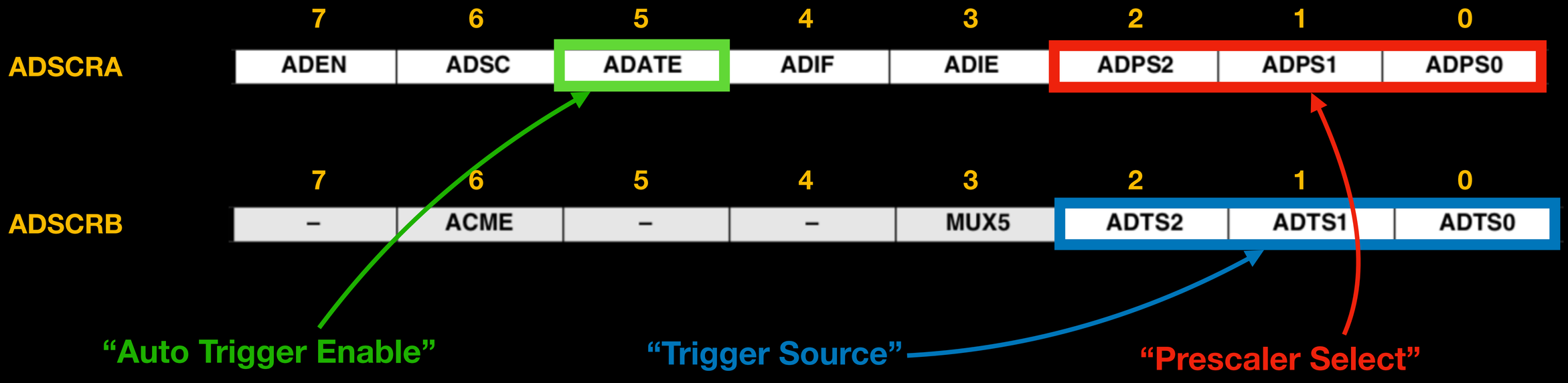
- Set bit to start an ADC conversion
- expect a result in **13 clock cycles**

- ADC sets bit HIGH when conversion is complete
- Check the bit before retrieving the result

- Set to have ADC call **`ISR(ADC_vect)`** after it completes a conversion

ADSCR: ADC Control and Status Registers

Useful bits for triggering conversions automatically



“Auto Trigger Enable”

- Set to enable automatic conversions

“Trigger Source”

- Conversions triggered externally or by peripherals

“Prescaler Select”

- ADC runs on a divided system clock

ADSCR: ADC Control and Status Registers

ADC conversions can be triggered externally, by other peripherals, or by the ADC's previous conversion

	7	6	5	4	3	2	1	0
ADSCRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

	7	6	5	4	3	2	1	0
ADSCRB	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0

Free Running Mode

- Trigger conversions at f_s = 16MHz/prescaler/13
- Must initiate the first conversion manually by setting ADSC

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADSCR: ADC Control and Status Registers

ADC conversions can be triggered externally, by other peripherals, or by the ADC's previous conversion

	7	6	5	4	3	2	1	0
ADSCRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

	7	6	5	4	3	2	1	0
ADSCRB	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0

ADC Rate / Resolution Trade-off

- ADC clock (16MHz/prescaler) must be > 50kHz
- ADC clock < 200kHz for full 10-bit resolution
- ADC clock < 1MHz for higher rate and reduced resolution

$$16\ MHz/128/13 \approx 9.6kHz$$

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADSCR: ADC Control and Status Registers

The remaining bits

	7	6	5	4	3	2	1	0
ADSCRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

	7	6	5	4	3	2	1	0
ADSCRB	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0

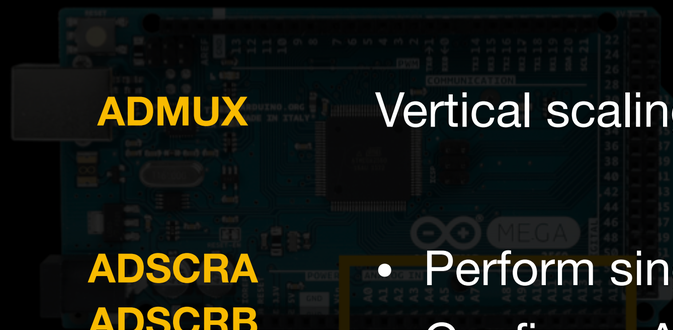
“Analog Comparator Multiplexer Enable”

- Use with Analog Comparator peripheral compare voltages to ADC pins (see sec 25.1)

“[Multiplexer] bit 6”

- Leftover MSB to **ADMUX** bits 4:0

ADC Summary



ADMUX

Vertical scaling, formatting of conversion results, and channel selection

ADSCRA
ADSCRB

- Perform single conversions manually like `analogRead()`
- Configure ADC to convert on timer overflow or output compare
- Configure ADC in free running mode

ADCH
ADCL

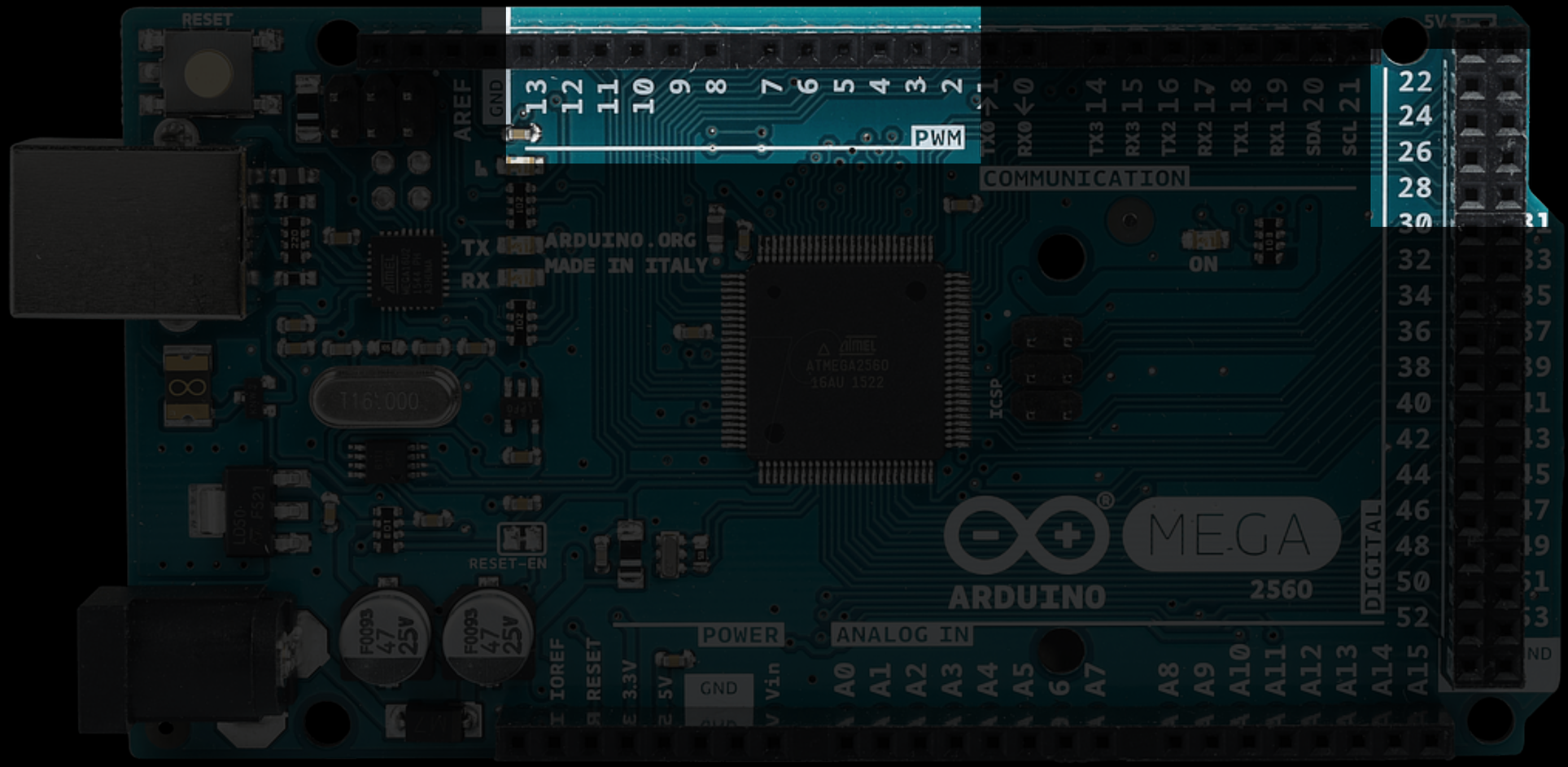
10-bit result value

- Right-adjusted for efficient 10-bit retrieval
- Left adjusted for efficient 8-bit retrieval

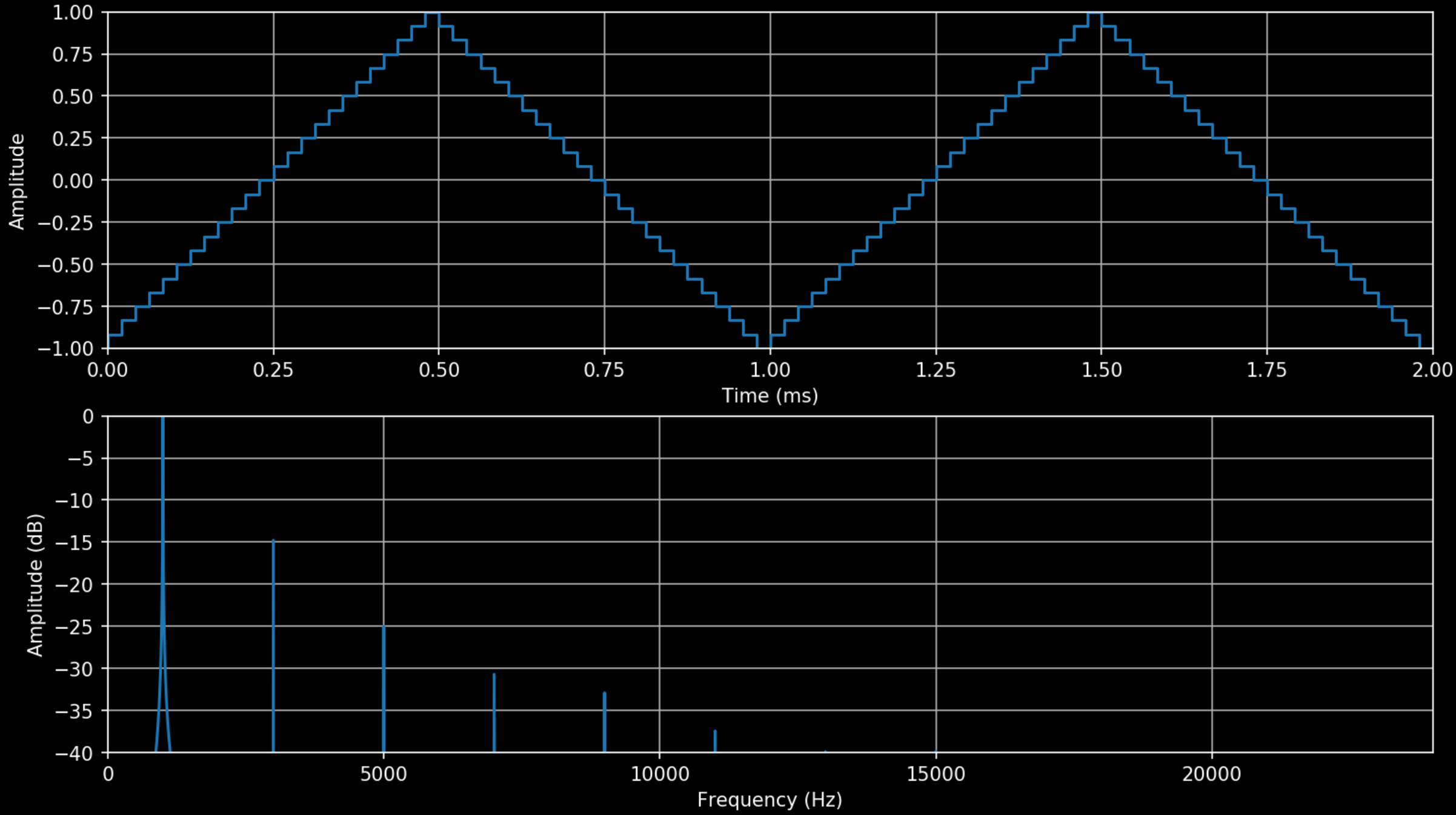
Improving on the R-Meter

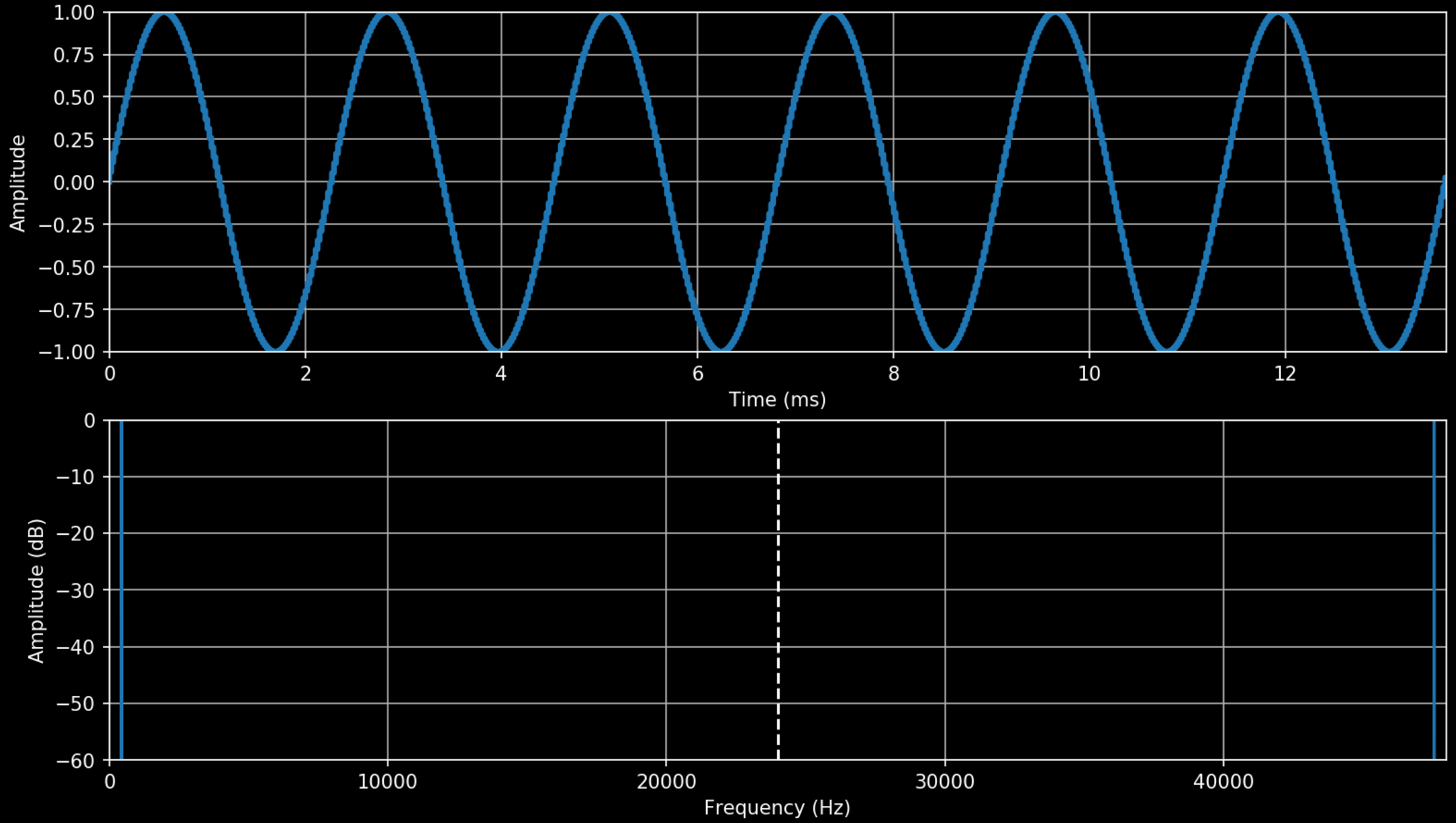
- Use external voltage reference < VCC or internal 1.1V or 2.56V sources
- Use differential pins with x1, x10, and x200 gain options
- Analog precision requires low noise, separate digital and analog grounds

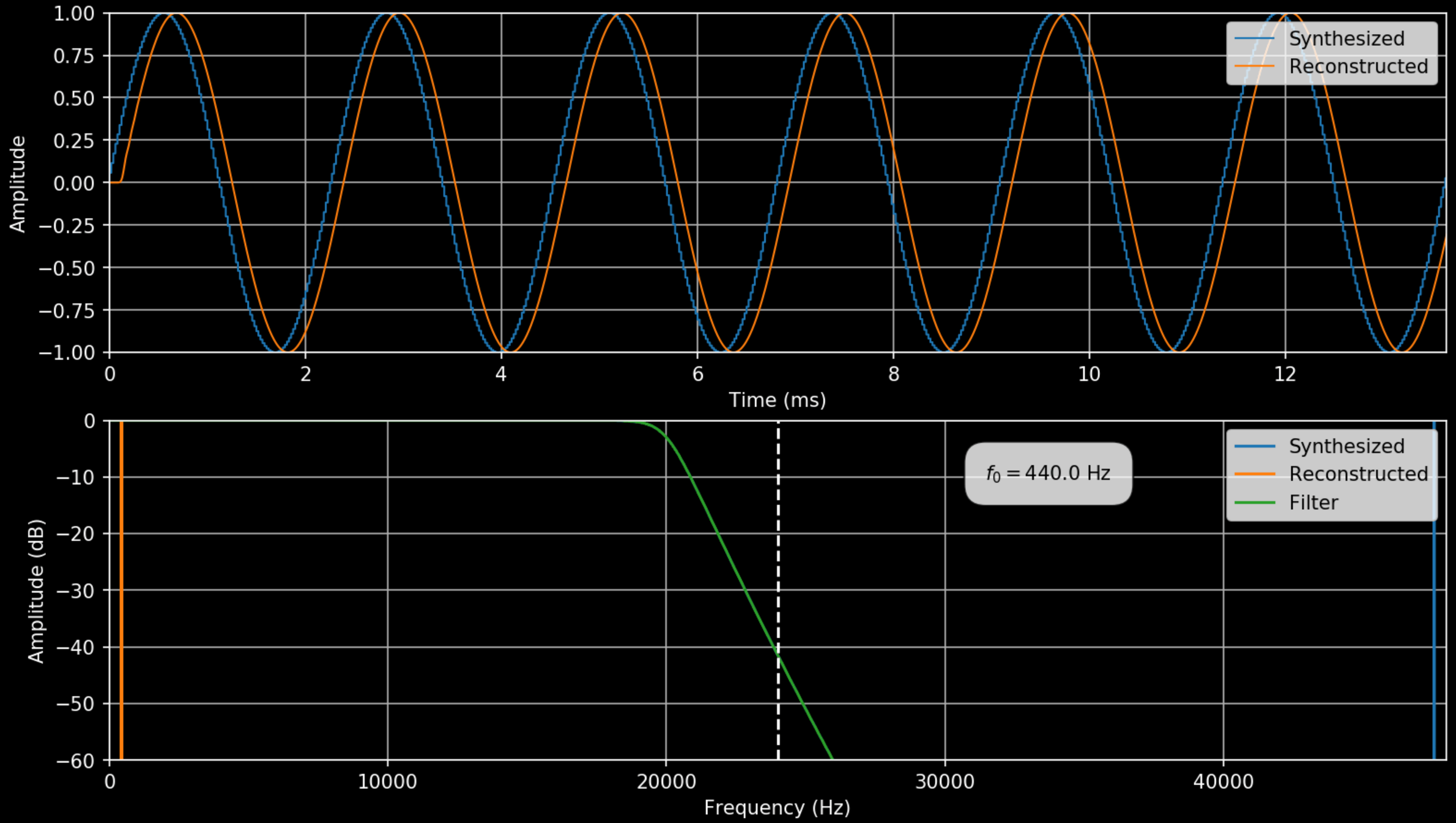
Mixed Signal Systems: Analog Output

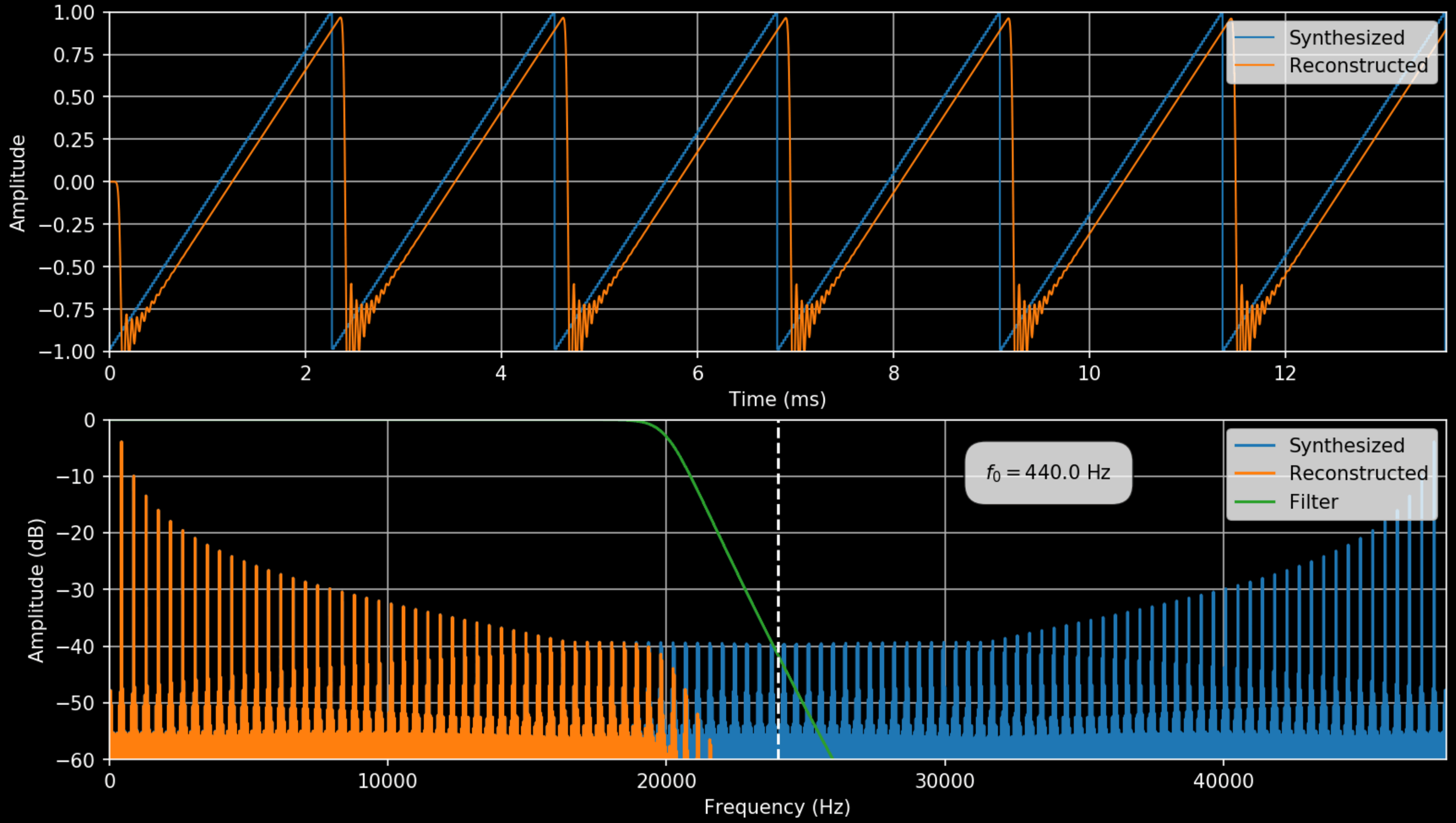


Caveats for Periodic Signals: Aliasing


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Time (ms)
0
−5
−10
−15
−20
−25
−30
−35
−40
Amplitude (dB)
0
5000
10000
15000
20000
Frequency (Hz)

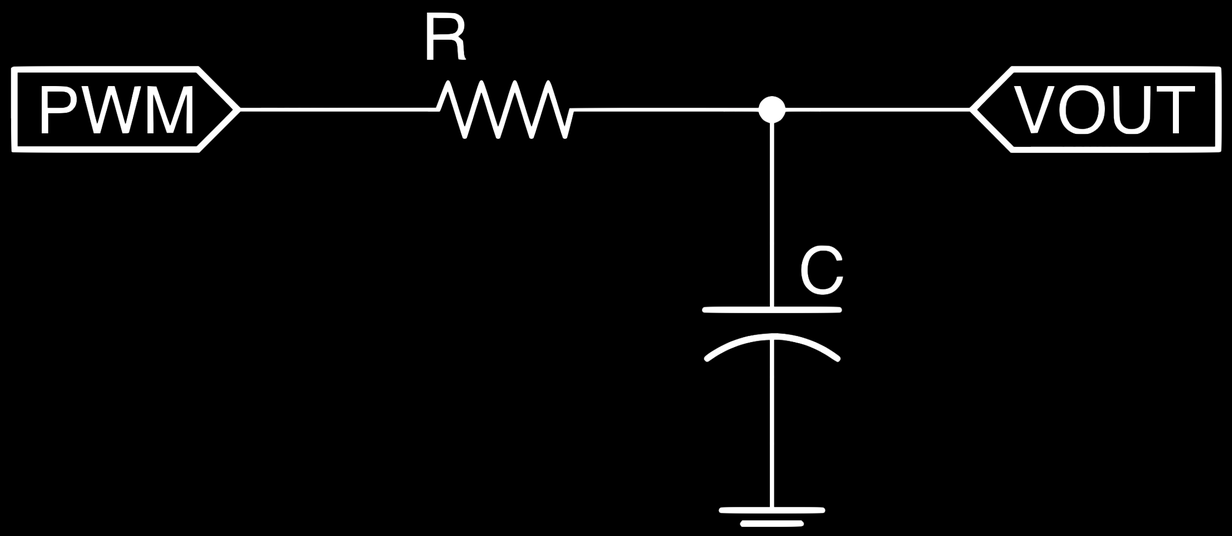

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0
2
4
6
8
10
12
Time (ms)
0
−10
−20
−30
−40
−50
−60
Amplitude (dB)
0
10000
20000
30000
40000
Frequency (Hz)


Amplitude
Time (ms)
Synthesized
Reconstructed
Amplitude (dB)
Frequency (Hz)
$f_0 = 440.0$ Hz
Synthesized
Reconstructed
Filter


Amplitude
Time (ms)
Synthesized
Reconstructed
Amplitude (dB)
Frequency (Hz)
$f_0 = 440.0$ Hz
Synthesized
Reconstructed
Filter

Reconstruction Filters

First Order Low-Pass, Passive:

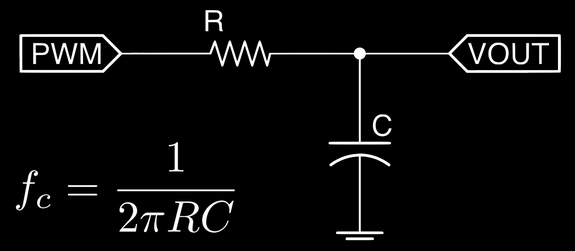


$$f_c = \frac{1}{2\pi RC}$$

Slope = -6dB/Octave

Frequency	Attenuation
fc	-3dB
2fc	-9dB
4fc	-15dB
8fc	-21dB
16fc	-27dB
32fc	-33dB
64fc	-39dB

Reconstruction Filters



$$f_c = \frac{1}{2\pi RC}$$

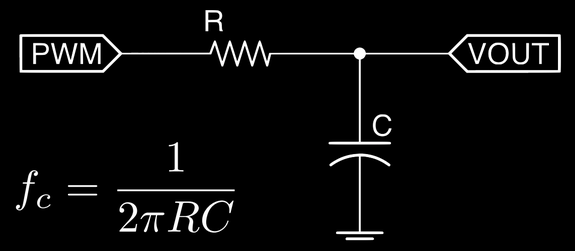
Design for sample rate of 16kHz and -33dB attenuation at 8kHz:

$$f_c = \frac{8000Hz}{32} = 250Hz$$ **Acceptable?**

Pick common R or C value, calculate the other:

Frequency	Attenuation
fc	-3dB
2fc	-9dB
4fc	-15dB
8fc	-21dB
16fc	-27dB
32fc	-33dB
64fc	-39dB

Reconstruction Filters



$$f_c = \frac{1}{2\pi RC}$$

Design for sample rate of 16kHz and -33dB attenuation at 8kHz:

$$f_c = \frac{8000Hz}{32} = 250Hz$$ **Acceptable?**

Pick common R or C value, calculate the other:

$$250Hz = \frac{1}{2\pi R \cdot 0.1\mu F}$$

$$R = 6336\Omega \approx 6.8k\Omega$$ **Close enough!**

Frequency	Attenuation
fc	-3dB
2fc	-9dB
4fc	-15dB
8fc	-21dB
16fc	-27dB
32fc	-33dB
64fc	-39dB

Digital Oscillators

Using sample rate f_s = 8kHz

Synthesize f_0 = 220Hz

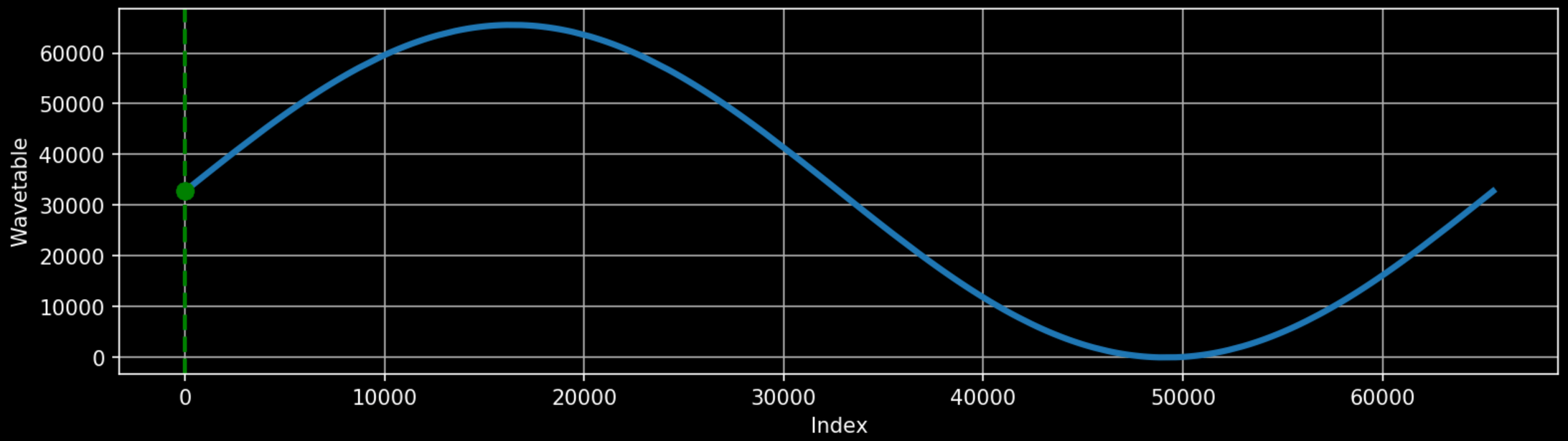
Continuous Time $x(t) = sin(2\pi f_0 t)$

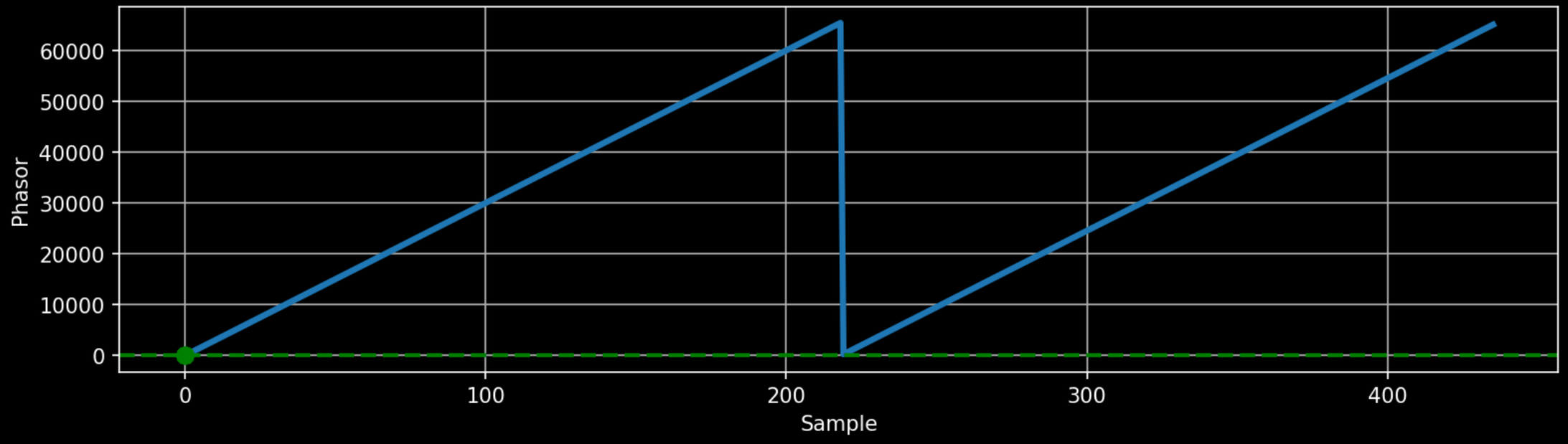
Discrete Time $x[n] = sin(2\pi f_0 \frac{n}{f_s})$

```
float fs = 8e3;
float phase = 0;
float freq = 220.0/fs * 2*M_PI;
float sine_sample;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   if (phase > 2*M_PI)
       phase -= 2*M_PI;
   sine_sample = sinf(phase);
}
```

Digital Oscillators: Wavetable





Digital Oscillators: 16-bit Phasor

Using sample rate f_s = 8kHz

Synthesize f_0 = 220Hz

Discrete Time $x[n] = sin(2\pi f_0 \frac{n}{f_s})$

Floating Point

```
float fs = 8e3;
float phase = 0;
float freq = 220.0/fs * 2*M_PI;
float sine_sample;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   if (phase > 2*M_PI)
       phase -= 2*M_PI;
   sine_sample = sinf(phase);
}
```

Fixed Point

```
float fs = 8e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 65535;
uint16_t wavetable[65536];
uint16_t sine_sample;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   sine_sample = wavetable[phase];
}
```

$132kB$ 👎

$\gg 8kB$

(ATmega2560 Memory)

Digital Oscillators: 16-bit Phasor

Using sample rate f_s = 8kHz

Synthesize f_0 = 220Hz

Discrete Time $x[n] = sin(2\pi f_0 \frac{n}{f_s})$

Floating Point

```
float fs = 8e3;
float phase = 0;
float freq = 220.0/fs * 2*M_PI;
float sine_sample;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   if (phase > 2*M_PI)
       phase -= 2*M_PI;
   sine_sample = sinf(phase);
}
```

Fixed Point

```
float fs = 8e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 65535;
uint16_t wavetable[1024];
uint16_t sine_sample;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   sine_sample = wavetable[phase >> 6];
}
```

$2kB$ 

$< 8kB$

(ATmega2560 Memory)

Digital Oscillators: Sine Wave Table

Discrete Time $x[n] = sin(2\pi f_0 \frac{n}{f_s})$

Initialization

```
#define TAB_LEN 1024
uint16_t wavetable[TAB_LEN];

void wavetable_init() {
    for (int i = 0; i < TAB_LEN; i++) {
        wavetable[i] = (1 + sinf(2*M_PI*i / (TAB_LEN-1))) * 32767.5;
    }
}
```

Digital Oscillators: Frequency Control

Map a 7-bit control signal n to frequency range $f_0 \in [0.2, 200]\ Hz$

Linear

$$m = \frac{200 - 0.2}{127}$$

$$f[n] = 0.2 + m \cdot n$$

with $n \in [0, 127]$

Exponential

$$c = \sqrt[127]{\frac{200}{0.2}}$$

$$f[n] = 0.2 \cdot c^n$$

with $n \in [0, 127]$

Digital Oscillators: Exponential Frequency Control

Map a R-bit control signal n to frequency range $f_0 \in [f_{min}, f_{max}]\ Hz$

$$c = \sqrt[2^R - 1]{\frac{f_{max}}{f_{min}}}$$

$$f[n] = f_{min} \cdot c^n$$

Precompute and use ADC reading for table lookup!

with $n \in [0,\ 2^R - 1]$

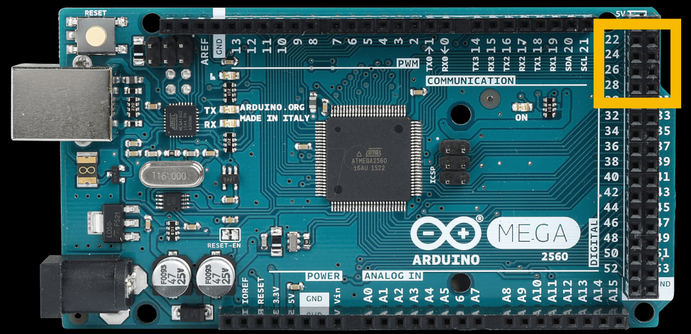
Digital Oscillators: Exponential Frequency Table

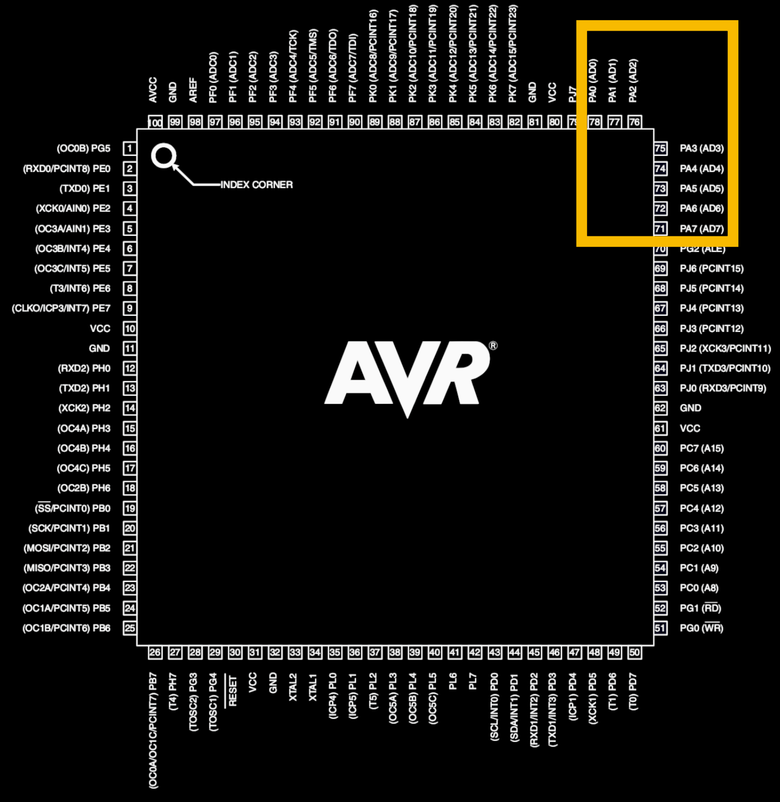
Initialization

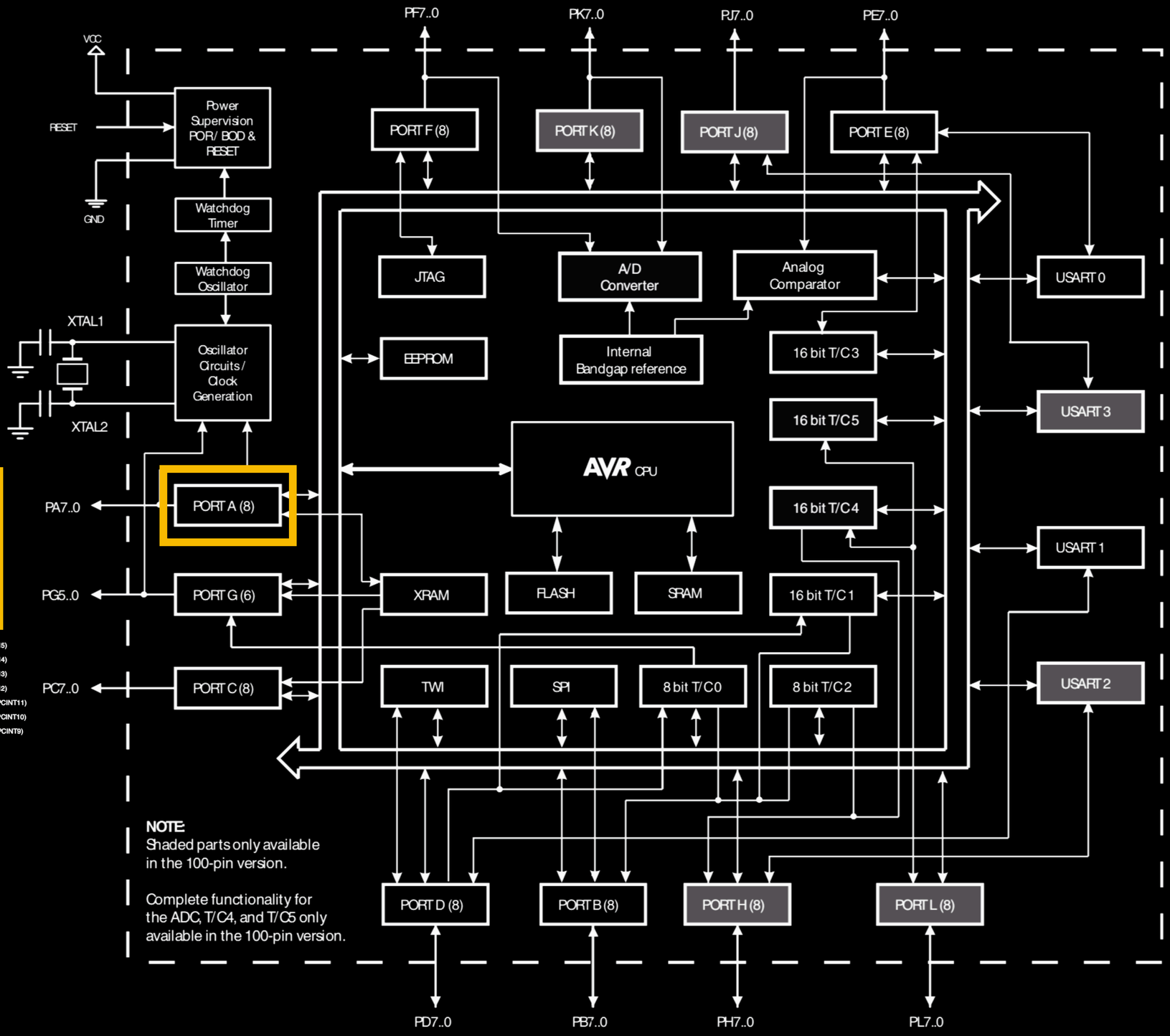
```
#define TAB_LEN 1024
uint16_t freqtable[TAB_LEN];

void freqtable_init(float f_min, float f_max) {
    float coeff = powf(f_max/f_min, 1.0/(TAB_LEN-1));
    float val = f_min;
    for (int i = 0; i < TAB_LEN; i++) {
        freqtable[i] = roundf(val * 65535.5);
        val *= coeff;
    }
}
```

Port A



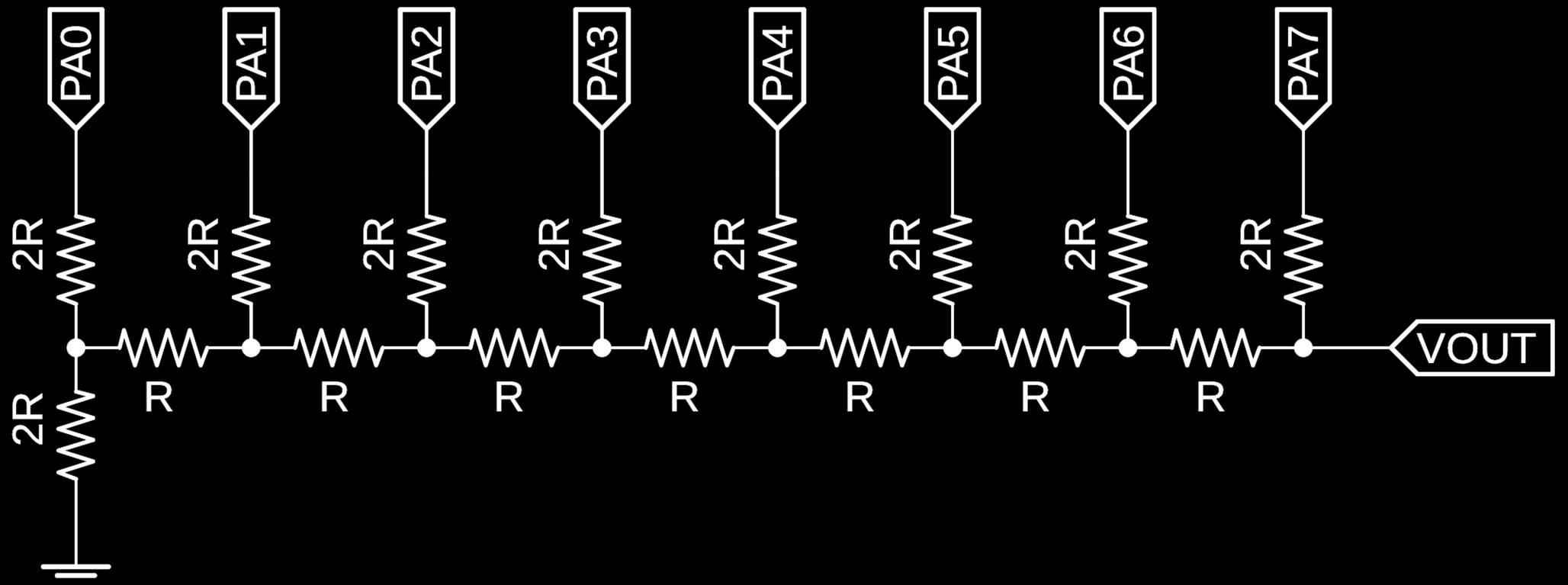




NOTE:
Shaded parts only available in the 100-pin version.

Complete functionality for the ADC, T/C4, and T/C5 only available in the 100-pin version.

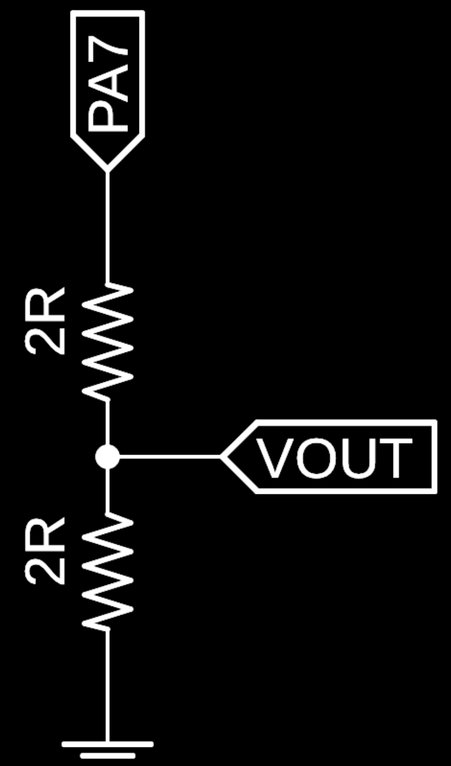
R-2R Ladder DAC



E6 Values: 1, 1.5, 2.2, 3.3, 4.7, 6.8

E12 Values: 1, 1.2, 1.5, 1.8, 2.2, 2.7, 3.3, 3.9, 4.7, 5.6, 6.8, 8.2

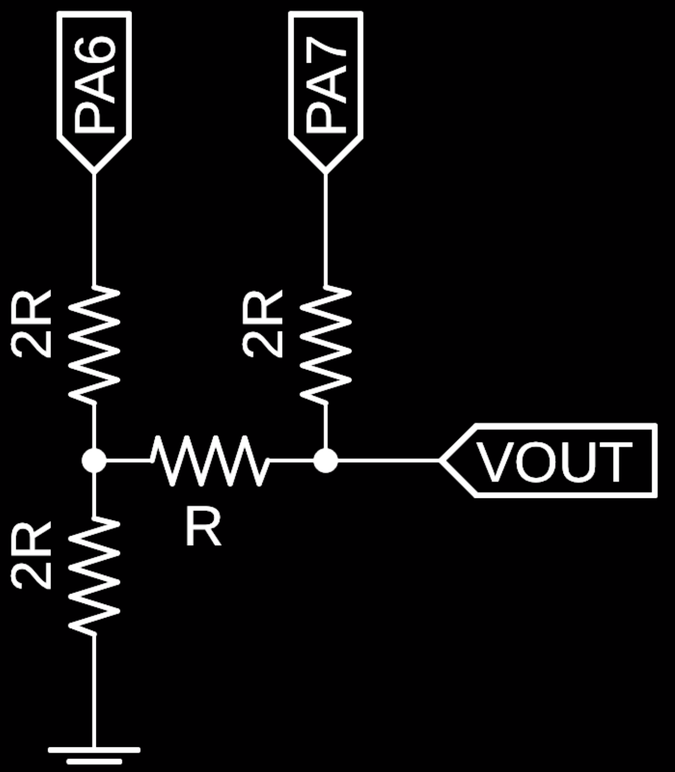
R-2R Ladder DAC



$$V_{out} = \frac{PA7}{2}$$

R-2R Ladder DAC

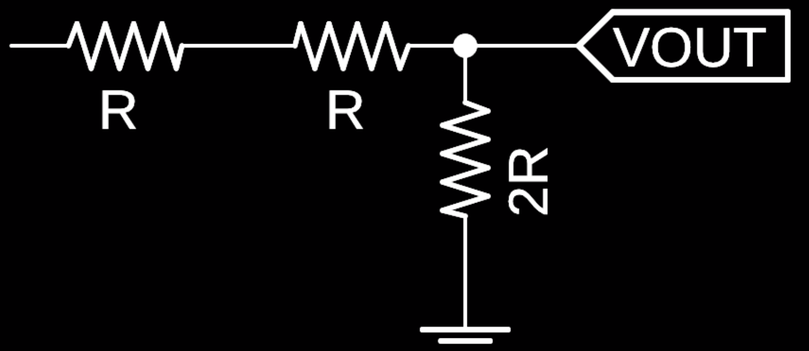
$$V_{TH} = \frac{PA6}{2}$$



$$V_{out} = \frac{PA7}{2} + ?$$

R-2R Ladder DAC

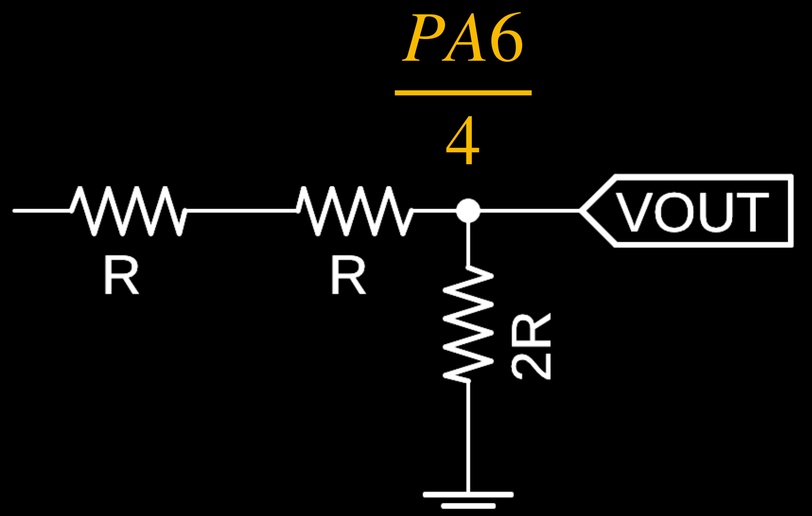
$$V_{TH} = \frac{PA6}{2}$$



$$V_{out} = \frac{PA7}{2} + ?$$

R-2R Ladder DAC

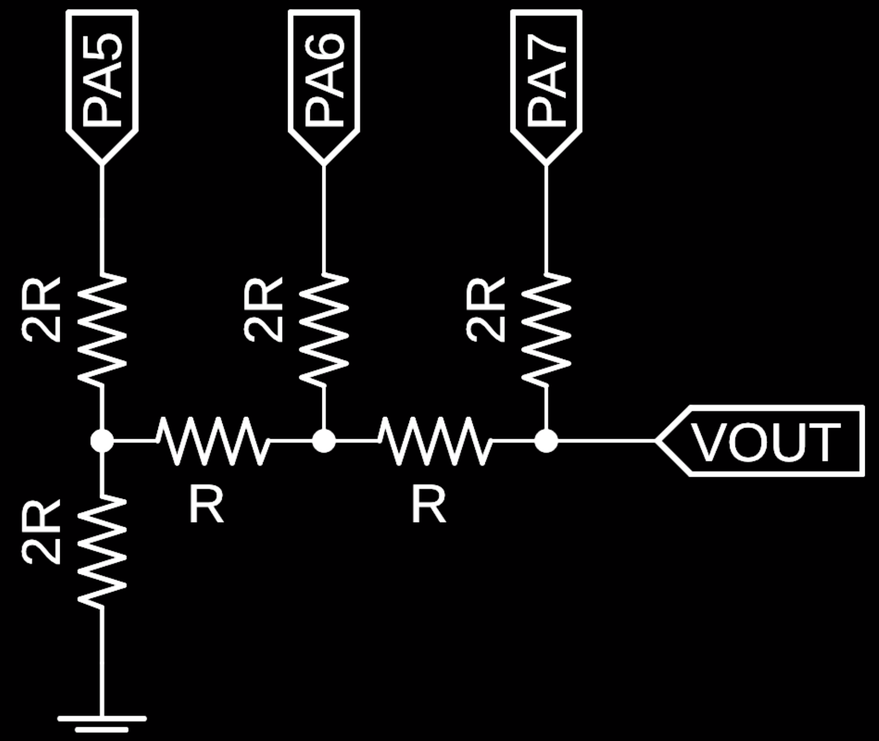
$$V_{TH} = \frac{PA6}{2}$$



$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4}$$

R-2R Ladder DAC

$$V_{TH} = \frac{PA5}{2}$$

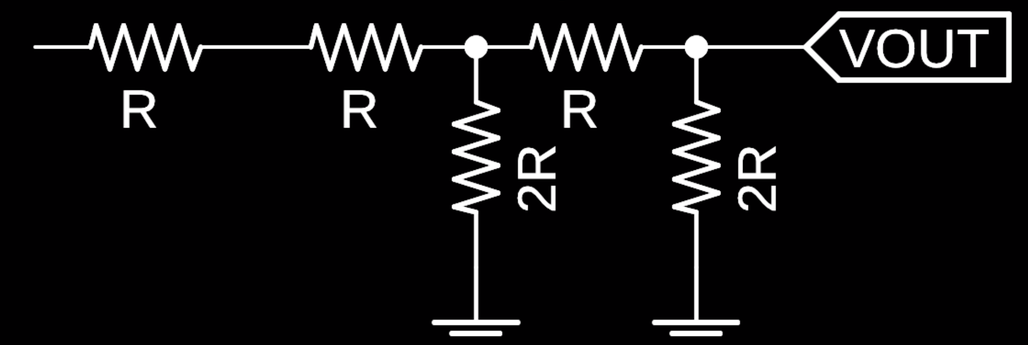


$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + ?$$

R-2R Ladder DAC

$$V_{TH} = \frac{PA5}{2}$$

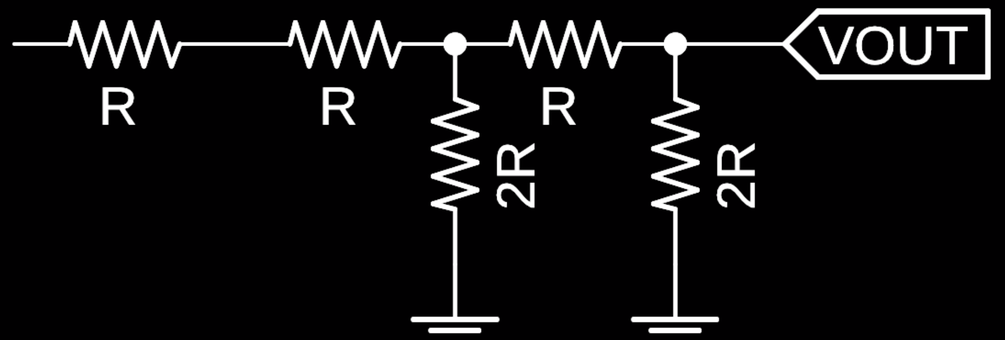
$$V_{TH} = \frac{PA5}{4}$$



$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + ?$$

R-2R Ladder DAC

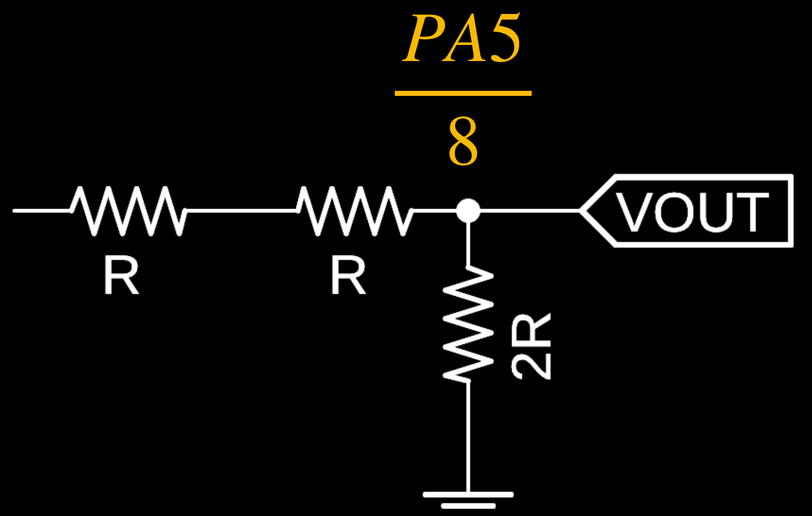
$$V_{TH} = \frac{PA5}{4}$$



$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + ?$$

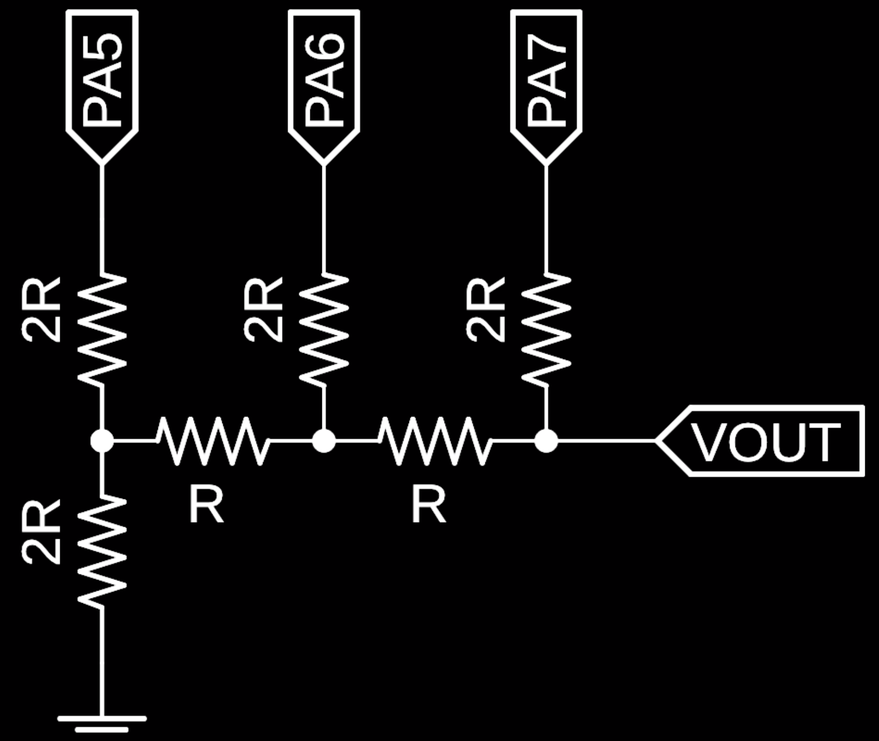
R-2R Ladder DAC

$$V_{TH} = \frac{PA5}{4}$$



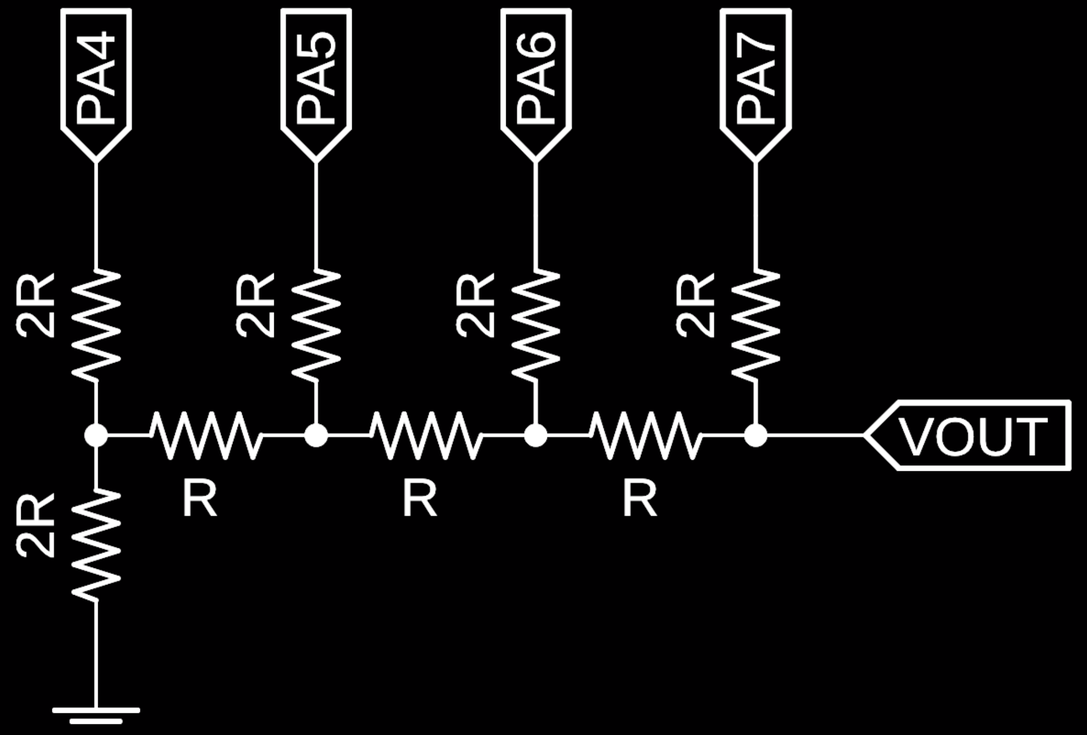
$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8}$$

R-2R Ladder DAC



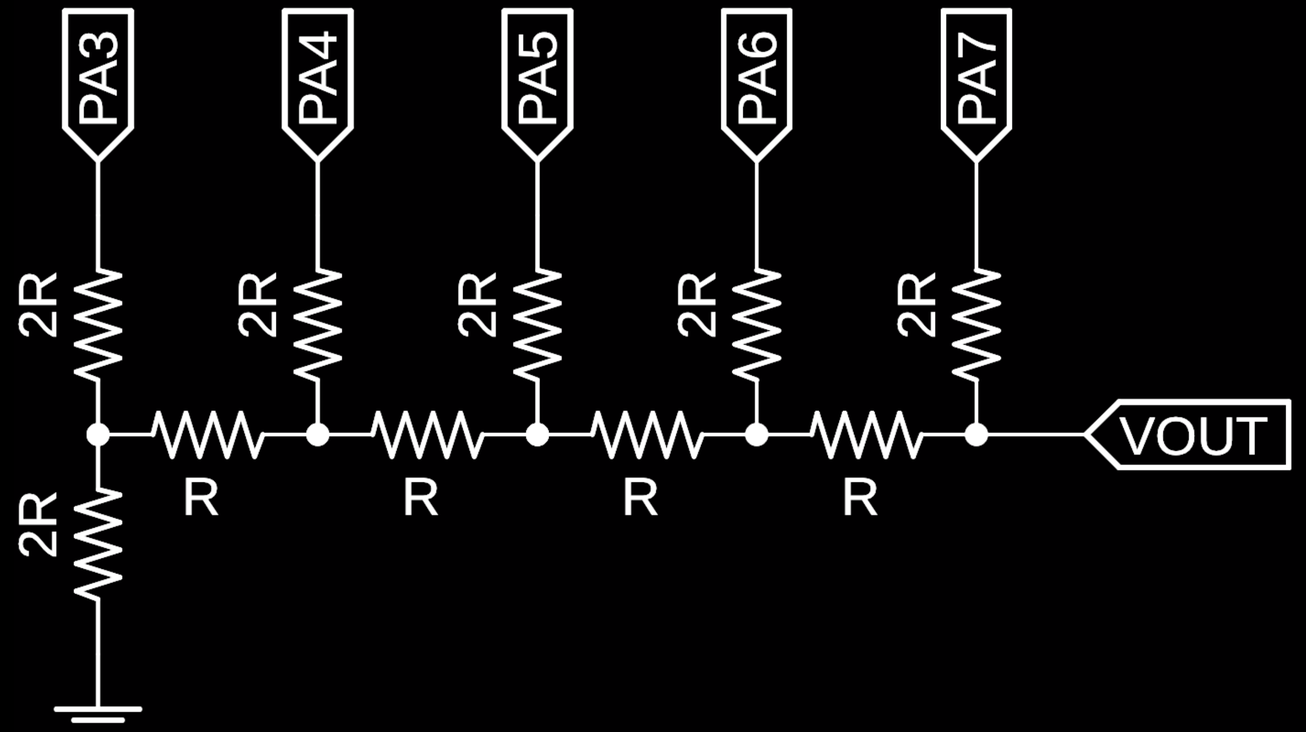
$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8}$$

R-2R Ladder DAC



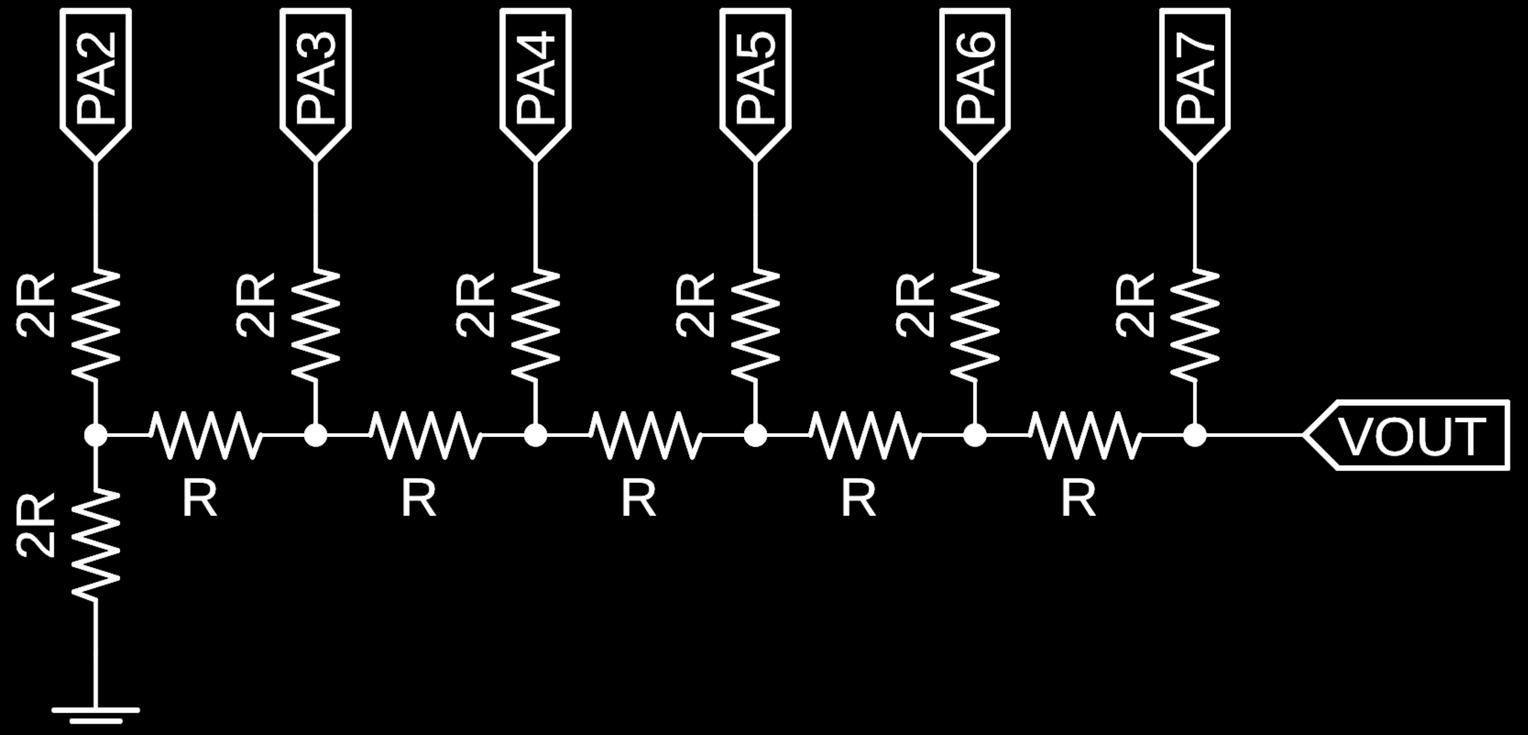
$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8} + \frac{PA4}{16}$$

R-2R Ladder DAC



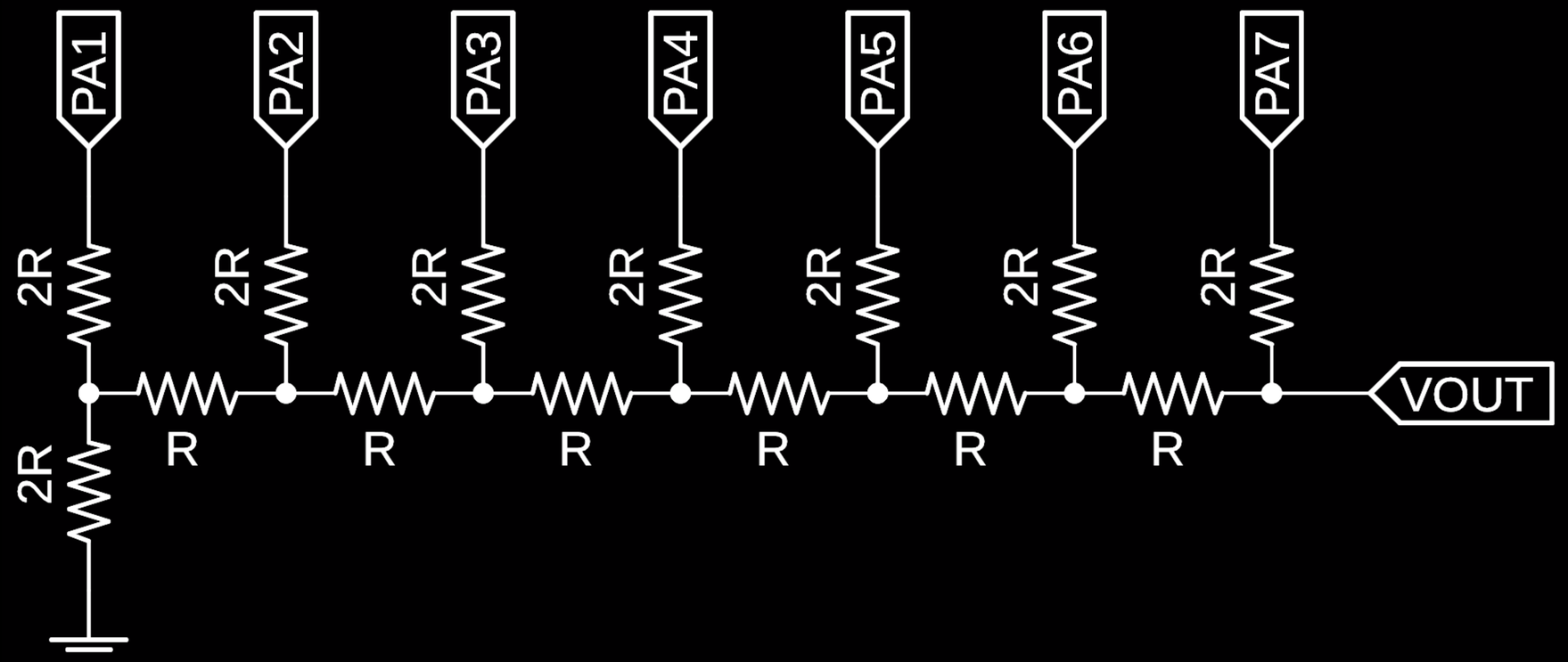
$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8} + \frac{PA4}{16} + \frac{PA3}{32}$$

R-2R Ladder DAC



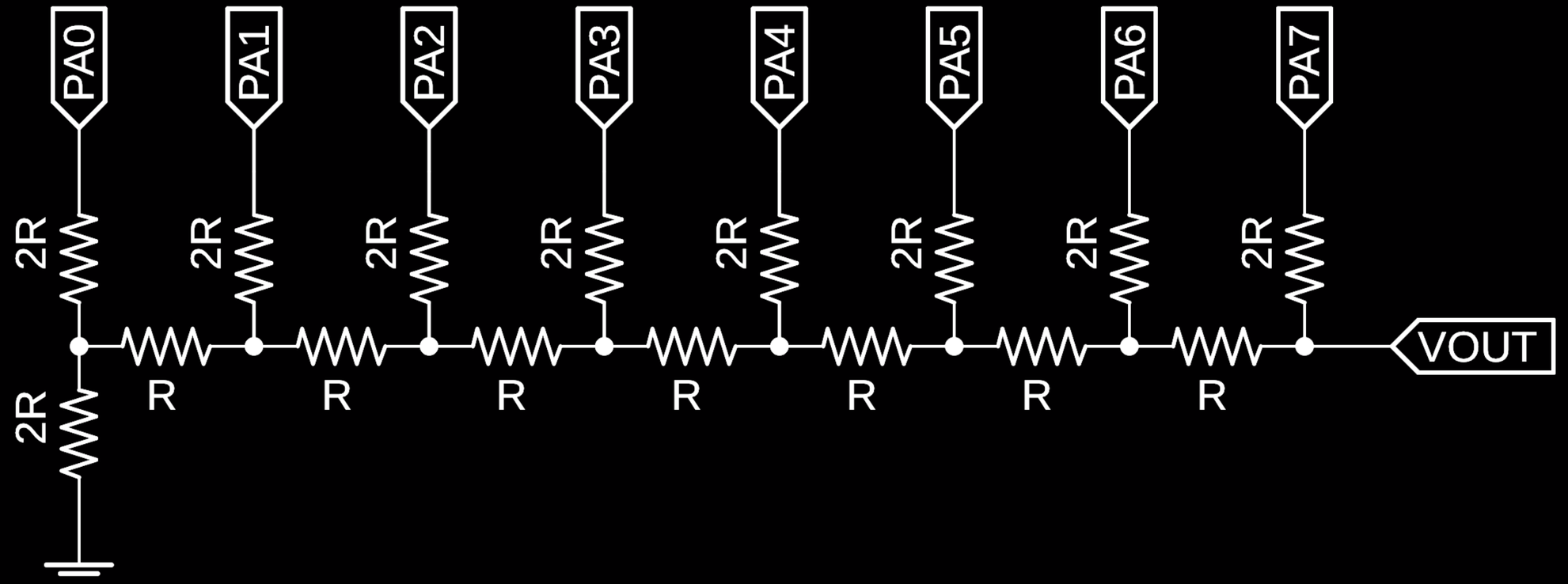
$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8} + \frac{PA4}{16} + \frac{PA3}{32} + \frac{PA2}{64}$$

R-2R Ladder DAC



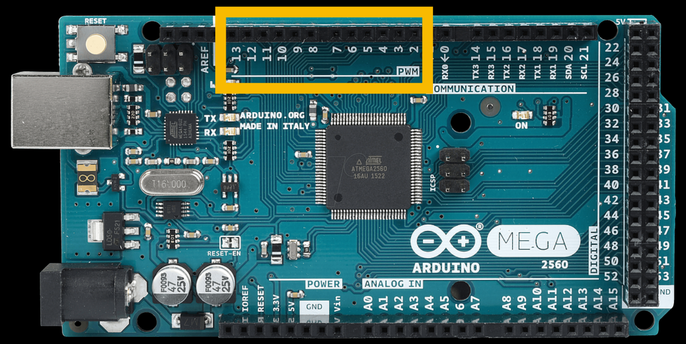
$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8} + \frac{PA4}{16} + \frac{PA3}{32} + \frac{PA2}{64} + \frac{PA1}{128}$$

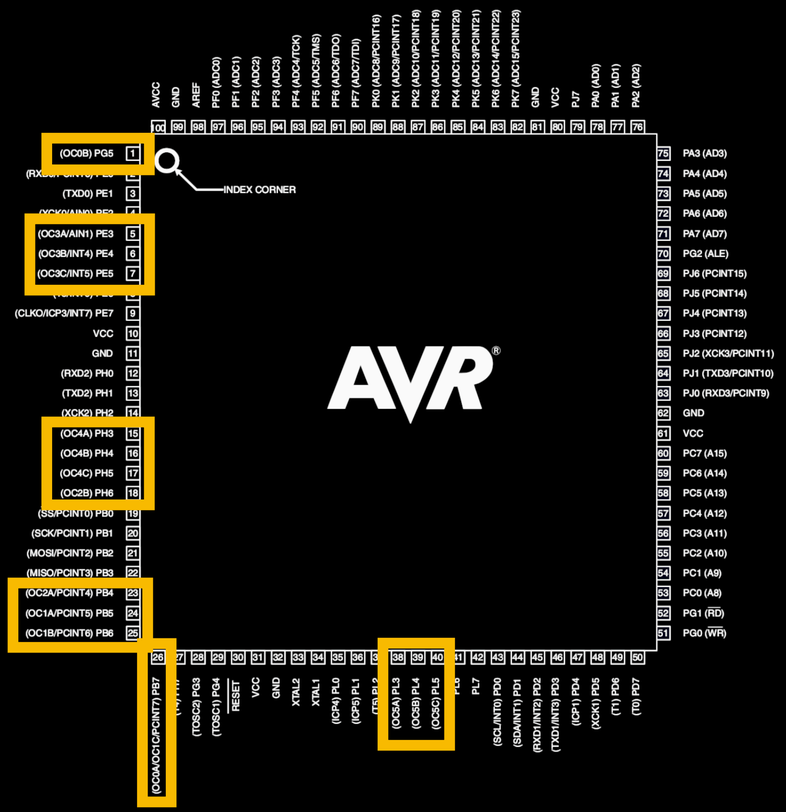
R-2R Ladder DAC

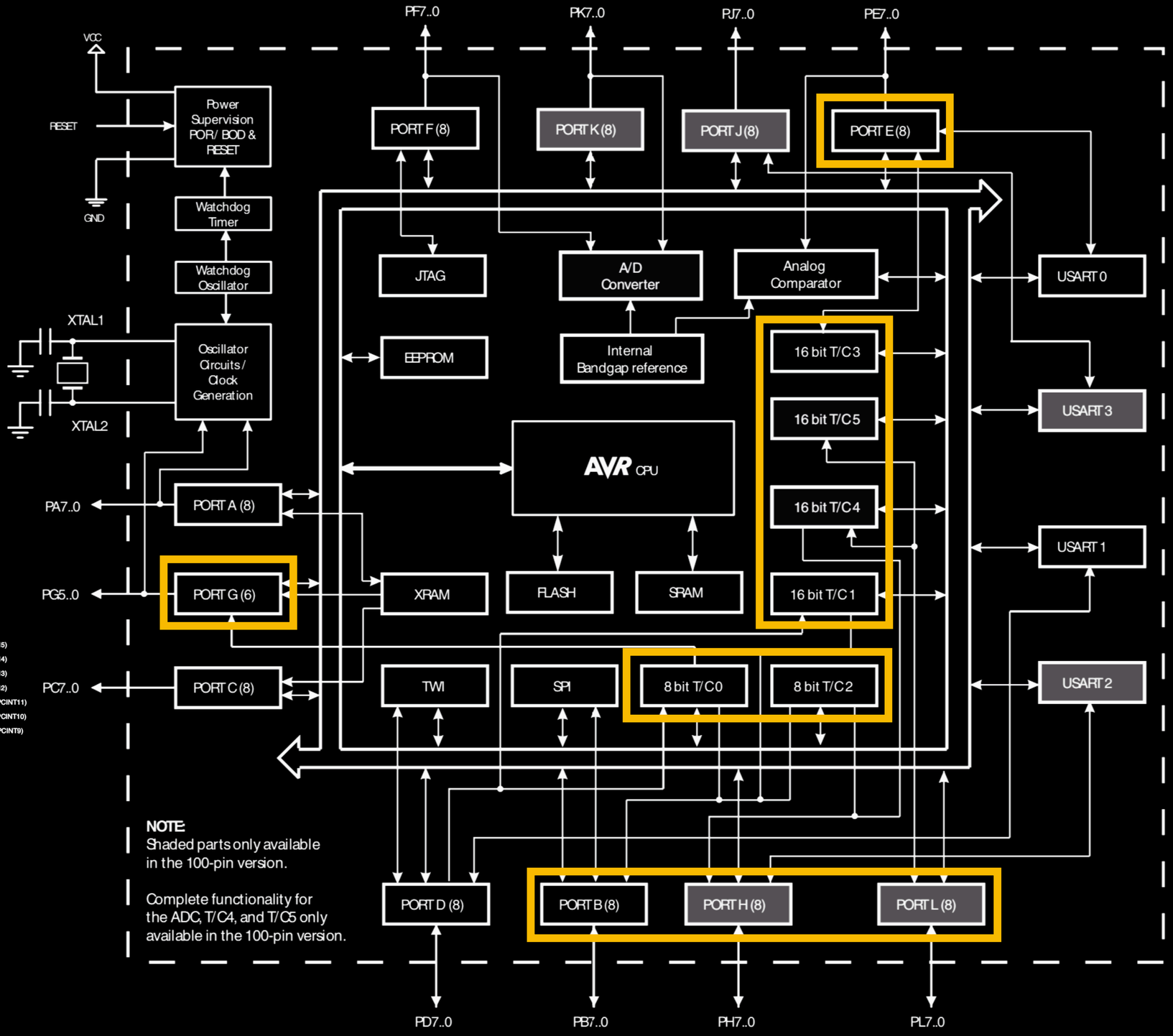


$$V_{out} = \frac{PA7}{2} + \frac{PA6}{4} + \frac{PA5}{8} + \frac{PA4}{16} + \frac{PA3}{32} + \frac{PA2}{64} + \frac{PA1}{128} + \frac{PA0}{256}$$

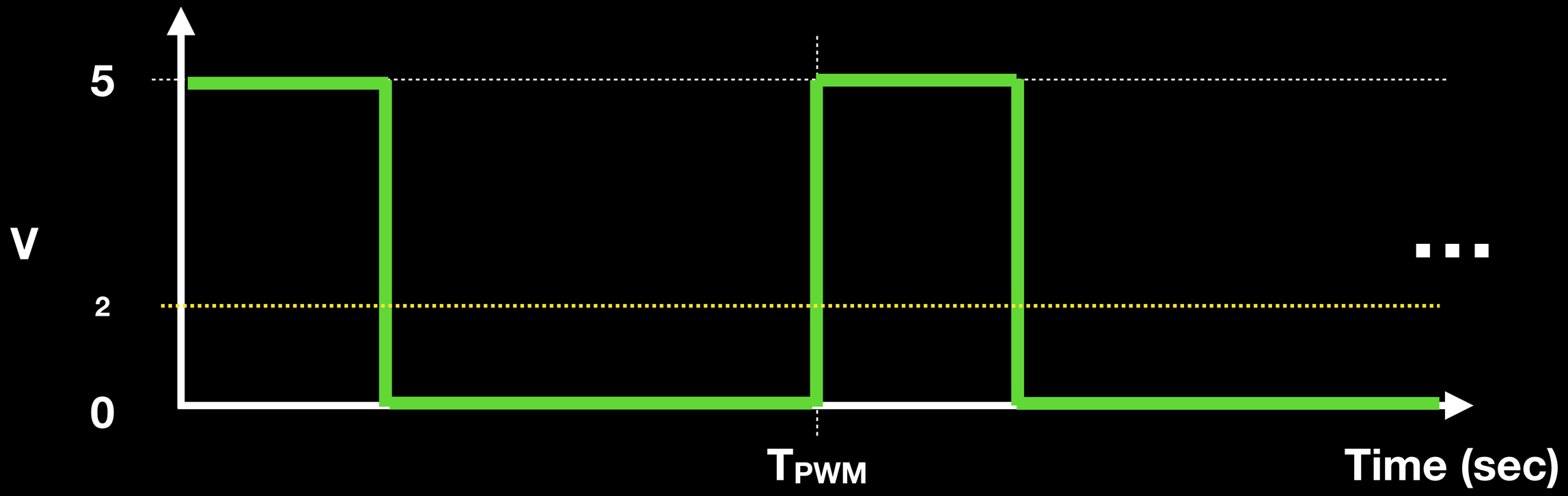
PWM


PWM
COMMUNICATION
DIGITAL
POWER
ANALOG IN
ARDUINO.ORG
MADE IN ITALY
MEGA
ARDUINO
2560


INDEX CORNER
AVR


VCC
RESET
GND
XTAL1
XTAL2
PA7..0
PG5..0
PC7..0
PF7..0
PK7..0
PJ7..0
PE7..0
PD7..0
PB7..0
PH7..0
PL7..0
Power Supervision POR/ BOD & RESET
Watchdog Timer
Watchdog Oscillator
Oscillator Circuits / Clock Generation
PORT A (8)
PORT G (6)
PORT C (8)
PORT F (8)
PORT K (8)
PORT J (8)
PORT E (8)
PORT D (8)
PORT B (8)
PORT H (8)
PORT L (8)
JTAG
A/D Converter
Analog Comparator
EEPROM
Internal Bandgap reference
AVR CPU
XRAM
FLASH
SRAM
TWI
SPI
8 bit T/C0
8 bit T/C2
16 bit T/C3
16 bit T/C5
16 bit T/C4
16 bit T/C1
USART 0
USART 3
USART 1
USART 2
NOTE:
Shaded parts only available in the 100-pin version.
Complete functionality for the ADC, T/C4, and T/C5 only available in the 100-pin version.

Pulse Width Modulation



16.9.3: Timer 0 Count Value Register

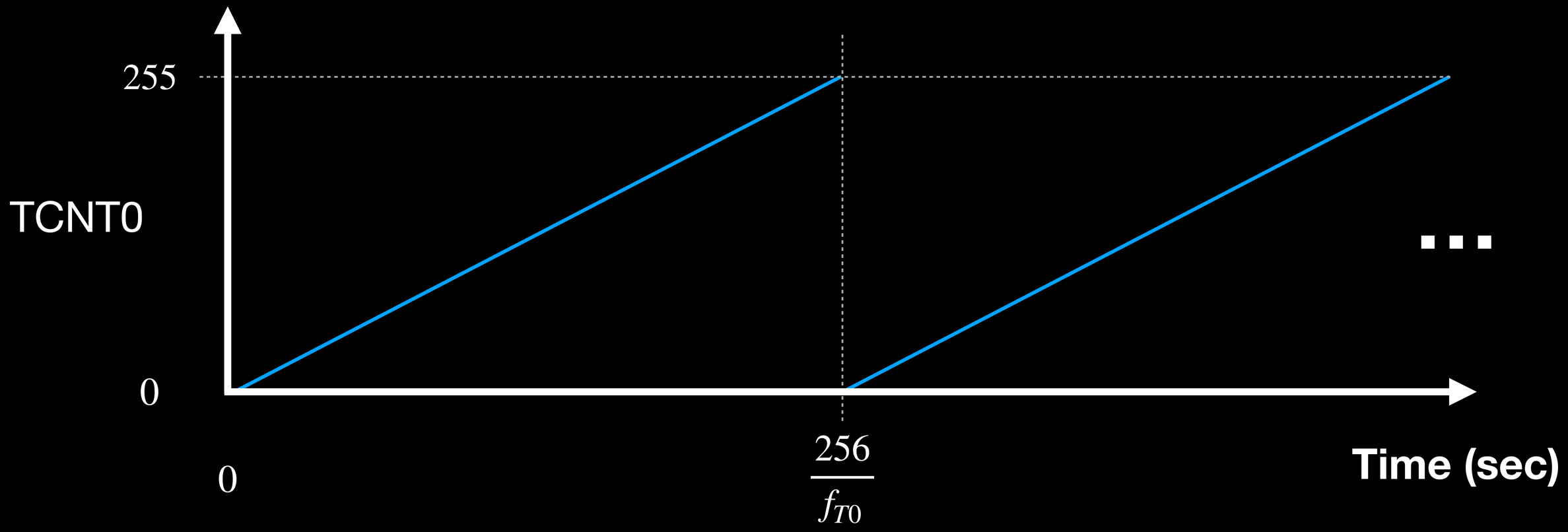
	7	6	5	4	3	2	1	0
TCNT0	TCNT0[7:0]							
OCR0A	OCR0A[7:0]							
TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
TCCR0B	FOC0A	FOC0B			WGM02	CS0[2:0]		

Fast PWM Mode

- Nyquist criterion also applies to PWM, so we need to use 8-bit timers (0, 2) to get PWM rates well above our sample rate
- 16-bit timers (1, 3-5) can also use this mode, but resulting rates are too slow for audio synthesis

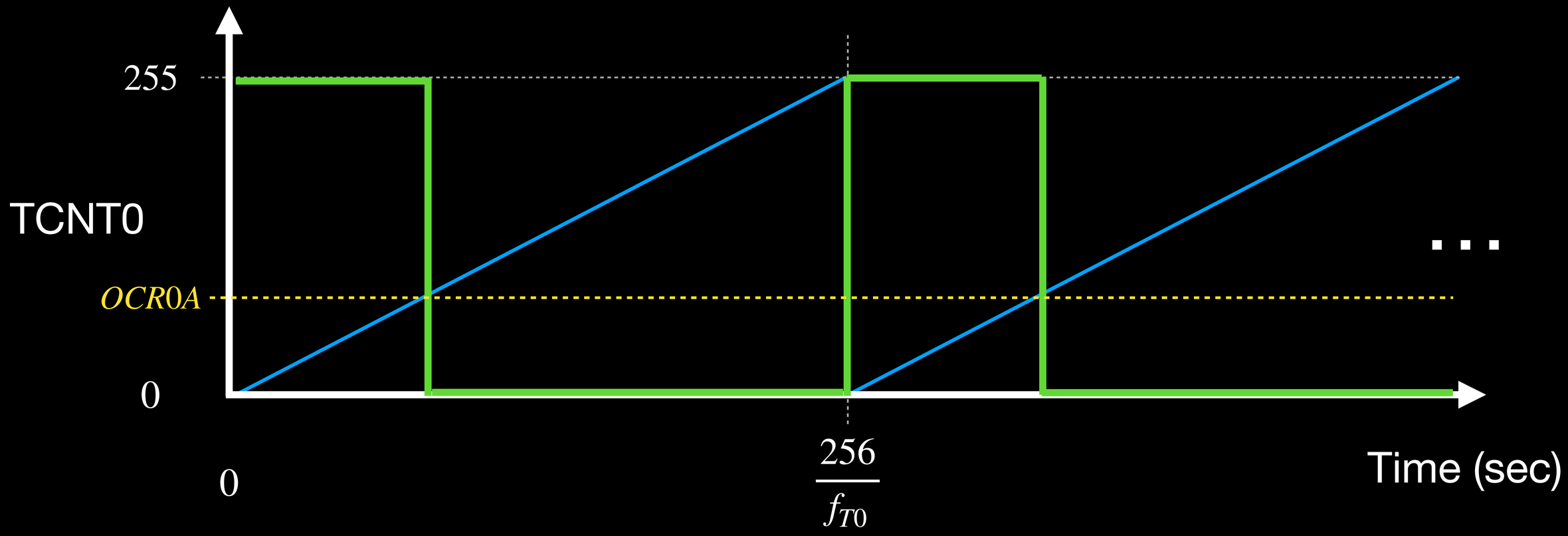
16.9.3: Timer 0 Count Value Register

	7	6	5	4	3	2	1	0
TCNT0	TCNT0[7:0]							



16.9.4-5: Timer 0 Output Compare Registers

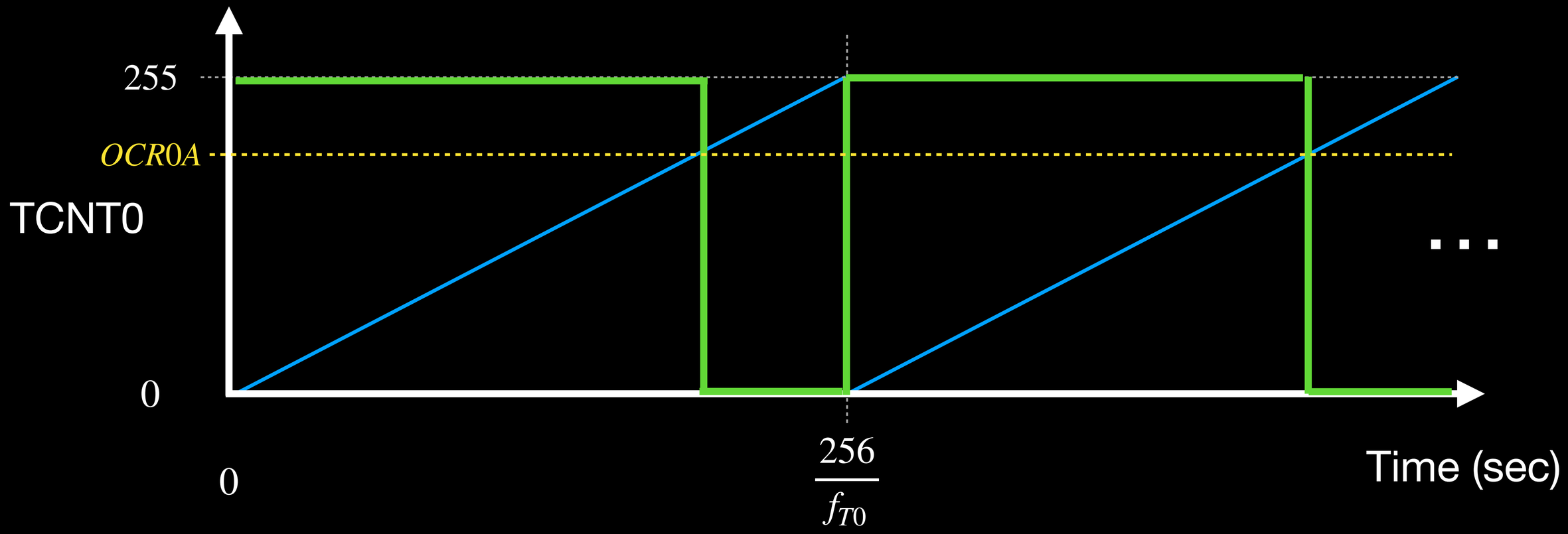
	7	6	5	4	3	2	1	0
TCNT0	TCNT0[7:0]							
	7	6	5	4	3	2	1	0
OCR0A	OCR0A[7:0]							



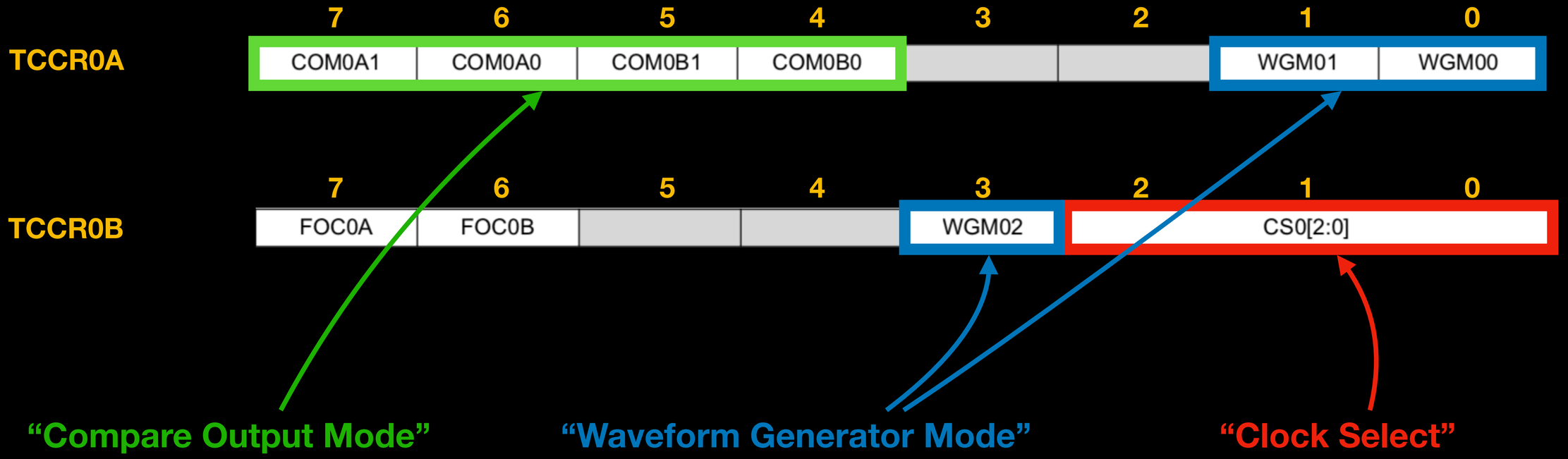
16.9.4-5: Timer 0 Output Compare Registers

	7	6	5	4	3	2	1	0
TCNT0	TCNT0[7:0]							

	7	6	5	4	3	2	1	0
OCR0A	OCR0A[7:0]							



16.9.1-2: Timer 0 Control Registers



“Compare Output Mode”

- Control the behavior of the Timer’s output pins (OCR0A, OCR0B)
- Different behavior for PWM than for non-PWM modes

“Waveform Generator Mode”

- Normal, PWM, CTC, etc.

“Clock Select”

- Prescaler dividing system clock or external pin

Timer 0 Control Registers: Waveform Generator Mode

TCCR0A

7	6	5	4	3	2	1	0
COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00

TCCR0B

7	6	5	4	3	2	1	0
FOC0A	FOC0B			WGM02	CS0[2:0]		

Use Fast PWM for signal synthesis

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Timer 0 Control Registers: Compare Output Mode

TCCR0A

7	6	5	4	3	2	1	0
COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00

TCCR0B

7	6	5	4	3	2	1	0
FOC0A	FOC0B			WGM02	CS0[2:0]		

Use Non-Inverting Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected WGM02 = 1: Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM (non-inverting mode)
1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM (inverting mode)

Timer 0 Control Registers: Clock Select

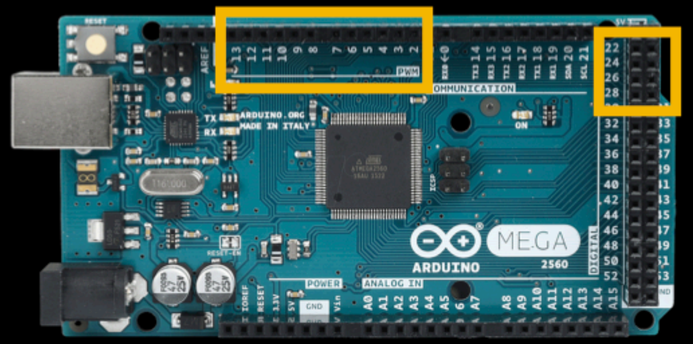
	7	6	5	4	3	2	1	0
TCCR0B	FOC0A	FOC0B			WGM02	CS0[2:0]		

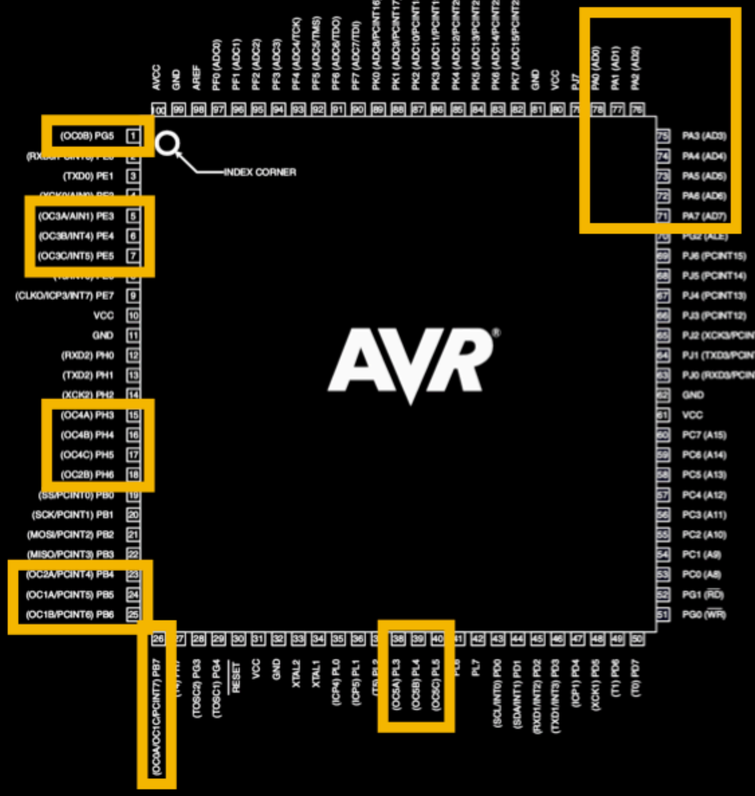
$$f_{T0} = \frac{16MHz}{prescaler}$$

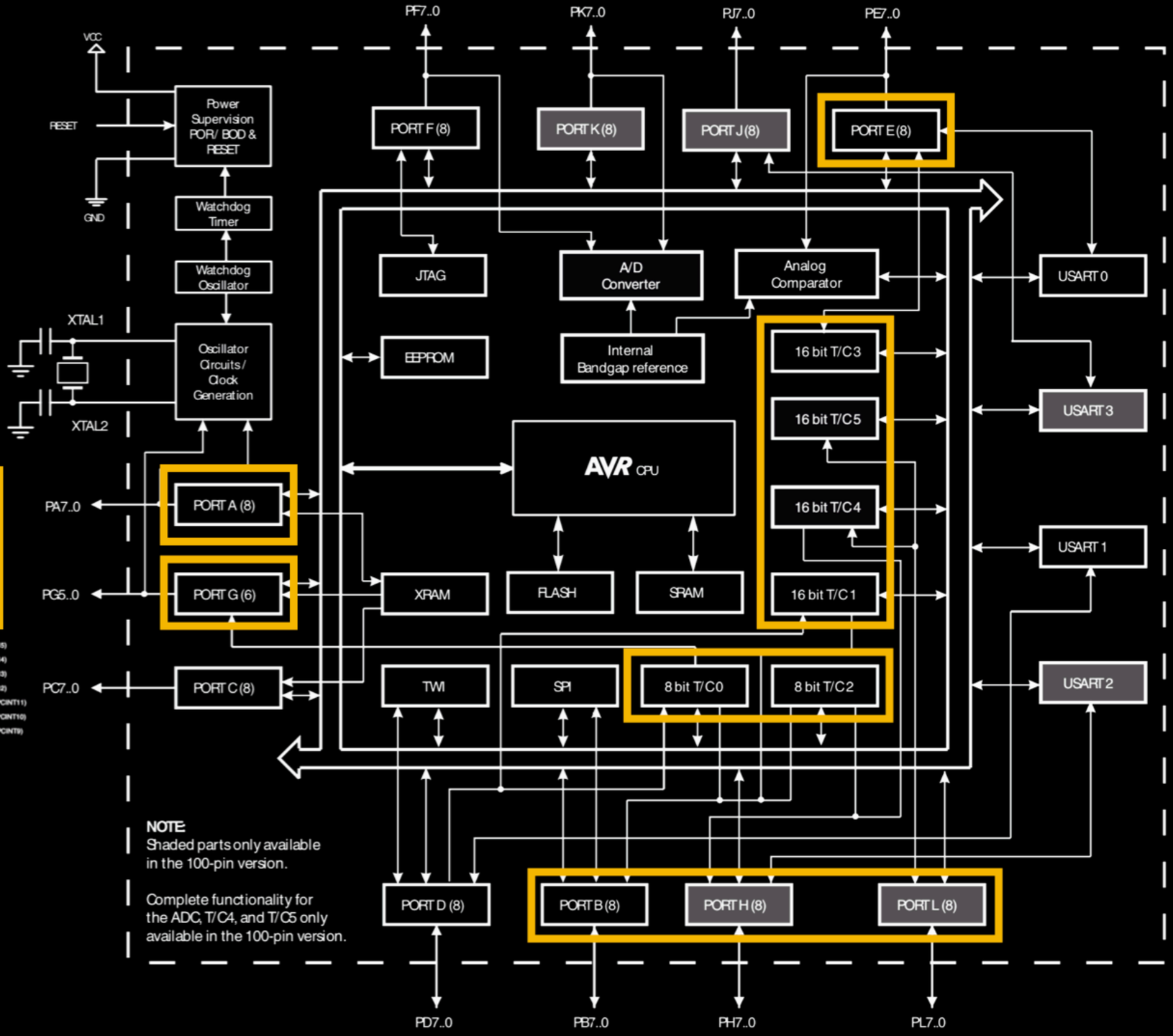
CA02	CA01	CS00	Description	f_{T0} (f_{clk}=16MHz)
0	0	0	No clock source (Timer/Counter stopped).	
0	0	1	$clk_{I/O}$/1 (No prescaling)	16 MHz
0	1	0	$clk_{I/O}$/8 (From prescaler)	2 MHz
0	1	1	$clk_{I/O}$/64 (From prescaler)	250 kHz
1	0	0	clkI/O/256 (From prescaler)	62.5 kHz
1	0	1	$clk_{I/O}$/1024 (From prescaler)	15.625 kHz

PWM Frequency = 16MHz / prescaler / 256

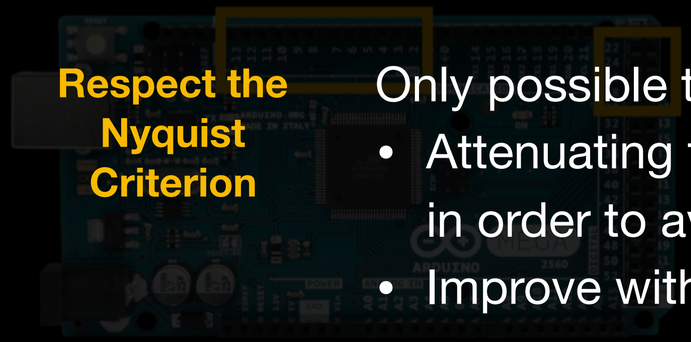
DAC Summary


ARDUINO
MEGA
2560


AVR
INDEX CORNER


VCC
RESET
GND
XTAL1
XTAL2
PF7..0
PK7..0
PJ7..0
PE7..0
Power Supervision POR/ BOD & RESET
Watchdog Timer
Watchdog Oscillator
Oscillator Circuits / Clock Generation
PORT F (8)
PORT K (8)
PORT J (8)
PORT E (8)
JTAG
A/D Converter
Analog Comparator
USART 0
EEPROM
Internal Bandgap reference
16 bit T/C 3
USART 3
16 bit T/C 5
AVR CPU
16 bit T/C 4
USART 1
PA7..0
PORT A (8)
PG5..0
PORT G (6)
XRAM
FLASH
SRAM
16 bit T/C 1
PC7..0
PORT C (8)
TWI
SPI
8 bit T/C 0
8 bit T/C 2
USART 2
NOTE:
Shaded parts only available in the 100-pin version.
Complete functionality for the ADC, T/C4, and T/C5 only available in the 100-pin version.
PORT D (8)
PORT B (8)
PORT H (8)
PORT L (8)
PD7..0
PB7..0
PH7..0
PL7..0

DAC Summary



Respect the Nyquist Criterion

Only possible to synthesize or process frequencies below half the sample rate

- Attenuating fs/2 with low order filters means we attenuate much of the usable band in order to avoid aliasing
- Improve with higher order filters -> sharper rolloff -> wider passband

Use Table Lookup and Integer Math

The ATmega2560 does not have floating point hardware

- Synthesis on this processor can cover most of the audio band if it's done efficiently
- Table lookups for sine wave and pre-computed frequency control save rendering cycles

Fast PWM: Timer 0, Timer 2

8-bit amplitude resolution at high PWM rates 16MHz/prescaler/256

Timer 1, Timer 3, Timer 4 Timer 5

16-bit amplitude resolution at much lower PWM rates 16MHz/prescaler/65536

Week 3 Deliverables

1. A function generator with the following features:
   A. Sine wave output signal via fast PWM and reconstruction filter
      • *Use this signal to drive an LED + current limiting resistor*
   B. Variable frequency from 0 to 30Hz using a potentiometer
      • *Use frequency table lookup*
   C. A SCPI serial interface that sets the frequency with the command `:SOURCE:FREQUENCY [Value in Hz]`
      • *The knob should control the frequency until a SCPI command is received, which disables the knob until the Arduino is restarted*
2. A video of your system demonstrating features 1A, 1B, and 1C
3. A report detailing your system and justifying the following design choices:
   A. The system's sample rate
   B. The PWM frequency
   C. The reconstruction filter's cutoff frequency

Hint 1

Get deliverables 1A and 1B working first

- Use pieces of `dac_pwm.ino` and/or `adc_free_running.ino`, along with the code snippets for digital oscillators (Analog Output Video)

Sample rate = control rate?

- The output signal is periodic, but the input isn't
- Oversampling can be useful, but control signals can often be measured at much lower rates

Nyquist criterion applies to both sample rate and PWM rate

- DAC should be done with a timer in fast PWM mode
- `analogWrite()` default rate (470Hz) is too low
    - and single conversions limit

Hint 2

In a separate Arduino sketch, adapt `mega_dmm.ino` (from Week 1) to respond to :SOURCE:FREQUENCY [value in Hz], then remove any unneeded code

Hint 3

Integrate the two after testing them separately

ARDUINO MEGA

PINOUT DIAGRAM