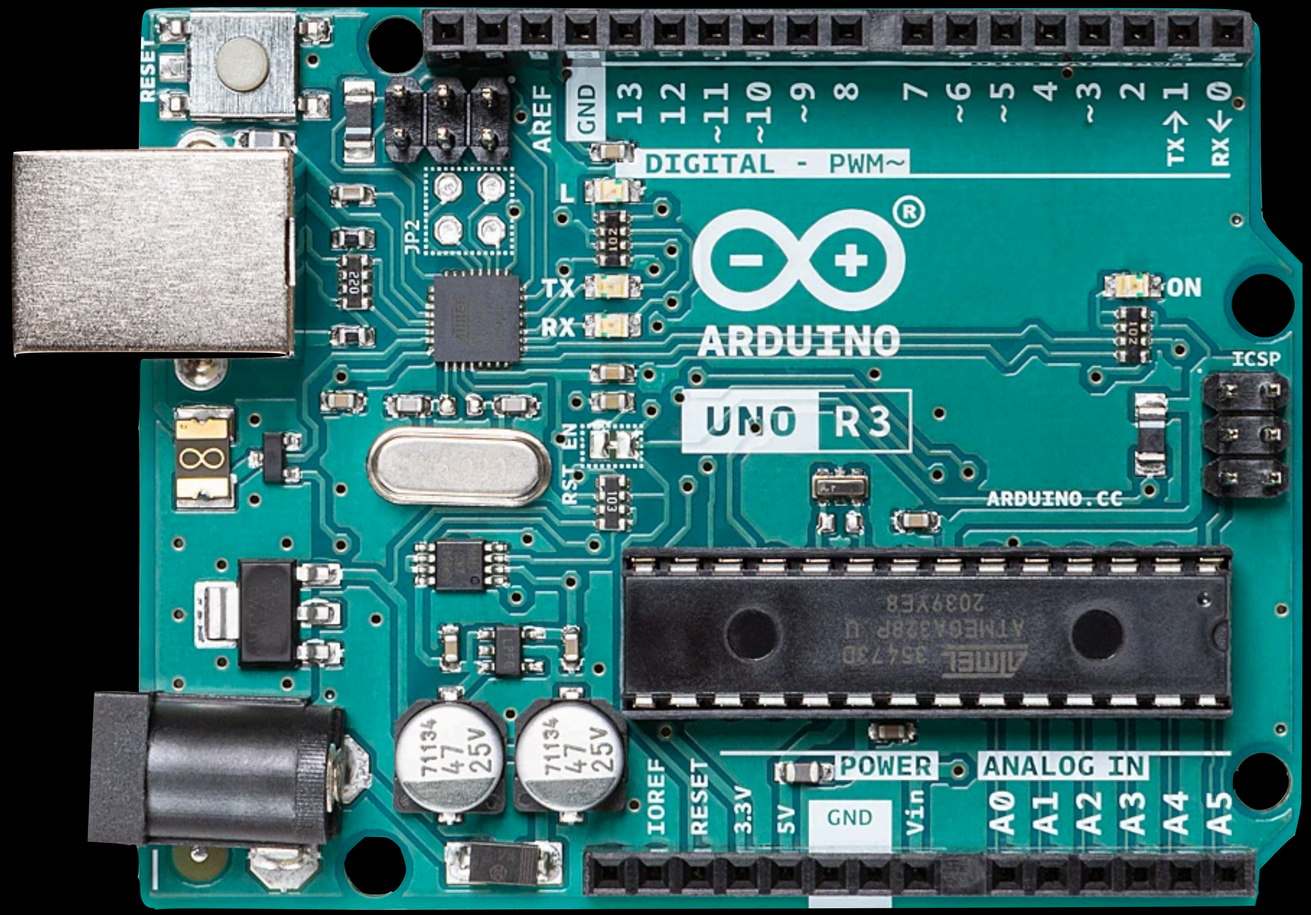
https://github.com/JeffGregorio/LibAG

Arduino Synthesis



Arduino Synthesis

Sample Rate	Sample Period
8kHz	125 µS
16kHz	62.5 µS
22.05kHz	45.4 µS
44.1kHz	22.7 µS
48kHz	20.8 µS
96kHz	10.4 µS
192kHz	5.2 µS

Arduino Synthesis

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Naive Arduino Synthesizer

Sine Wave Synthesis

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

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

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

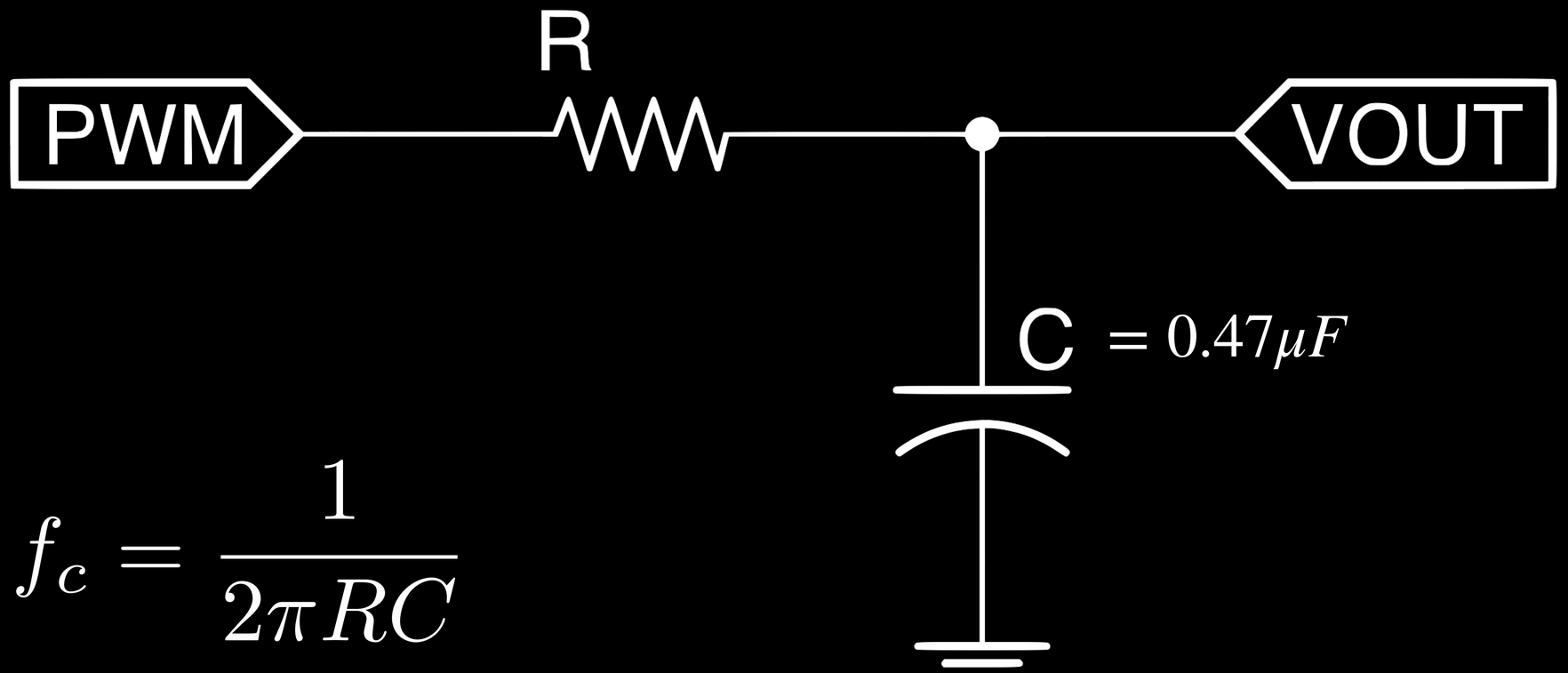
Precompute

```
const float fs = 8e3;
float phase = 0;
float freq = 220.0/fs * TWO_PI;
```

Render

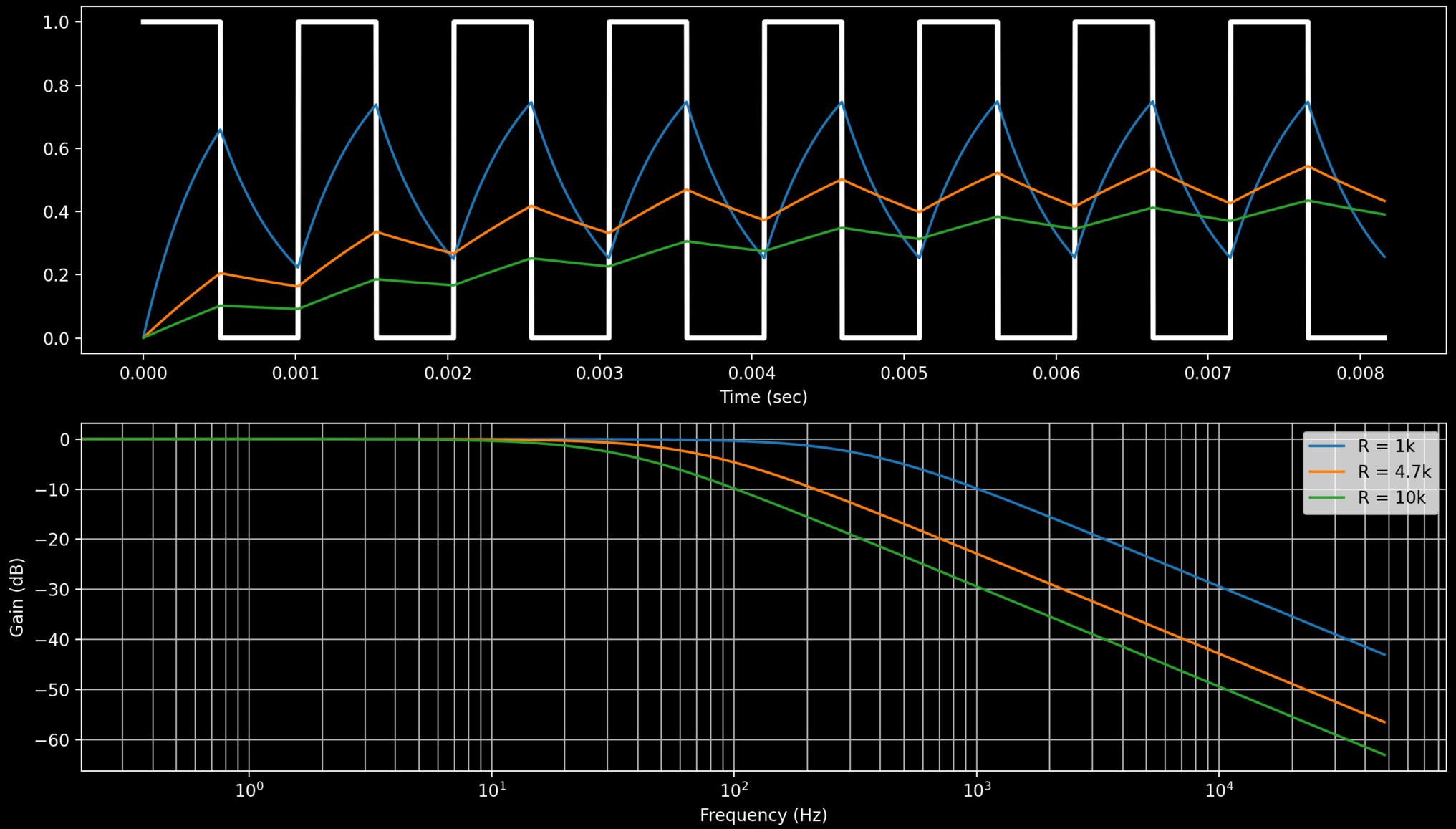
```
float sample = sinf(phase);
phase += freq;
if (phase > TWO_PI)
  phase -= TWO_PI;
```

Low Pass Filter

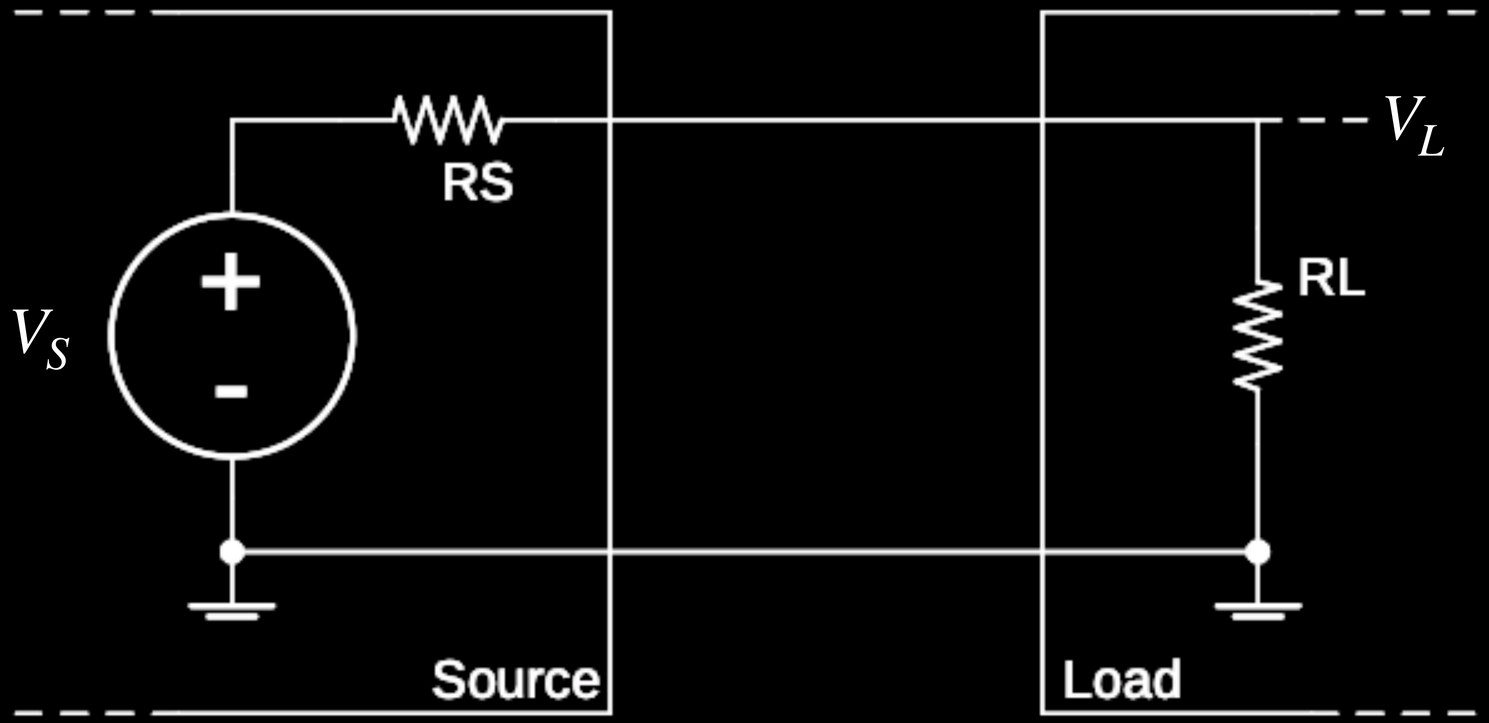


$$f_c = \frac{1}{2\pi RC}$$

Low Pass Filter

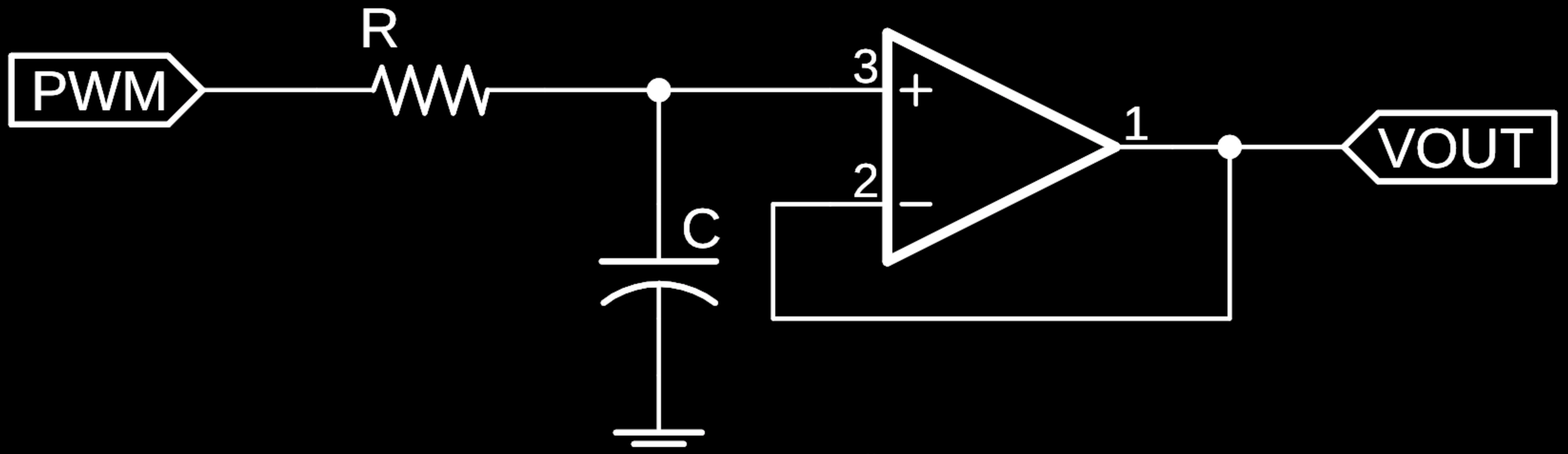


Impedance Bridging



$$V_L = V_S \frac{R_L}{R_S + R_L}$$

Low Pass Filter, Buffered



Caveat:
Use a ‘Rail-to-Rail’ Op Amp!

TLV2372

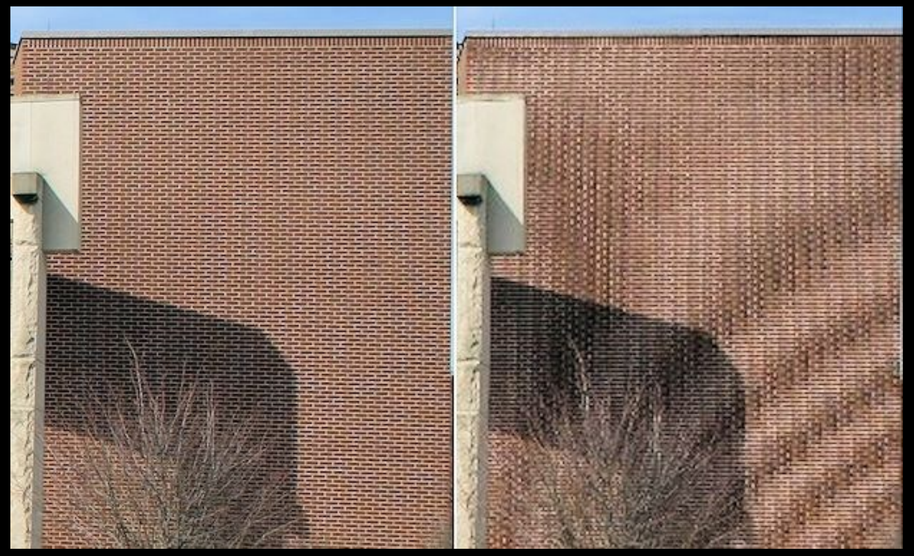
Exponential Frequency Control

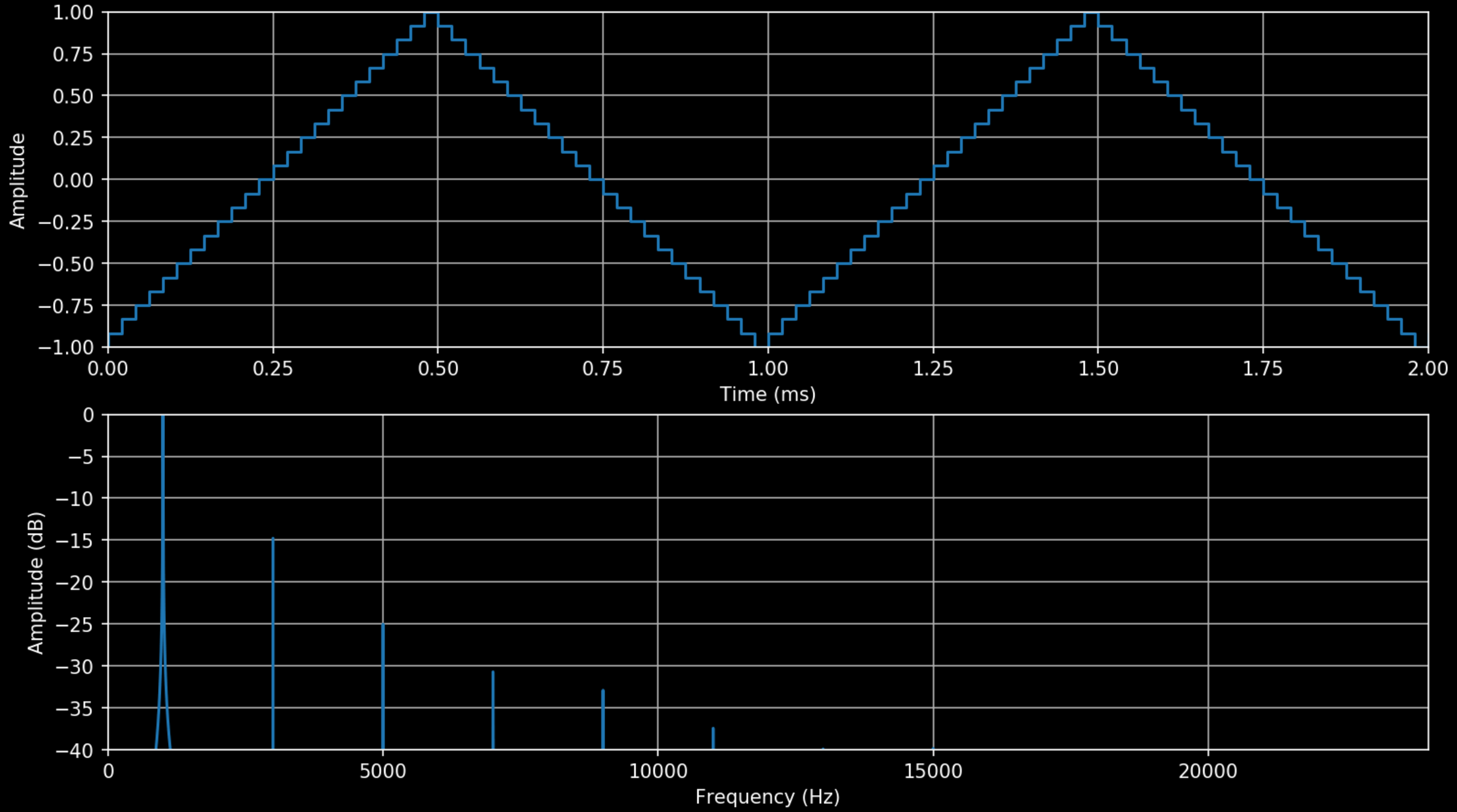
Example:
MIDI Note to Frequency

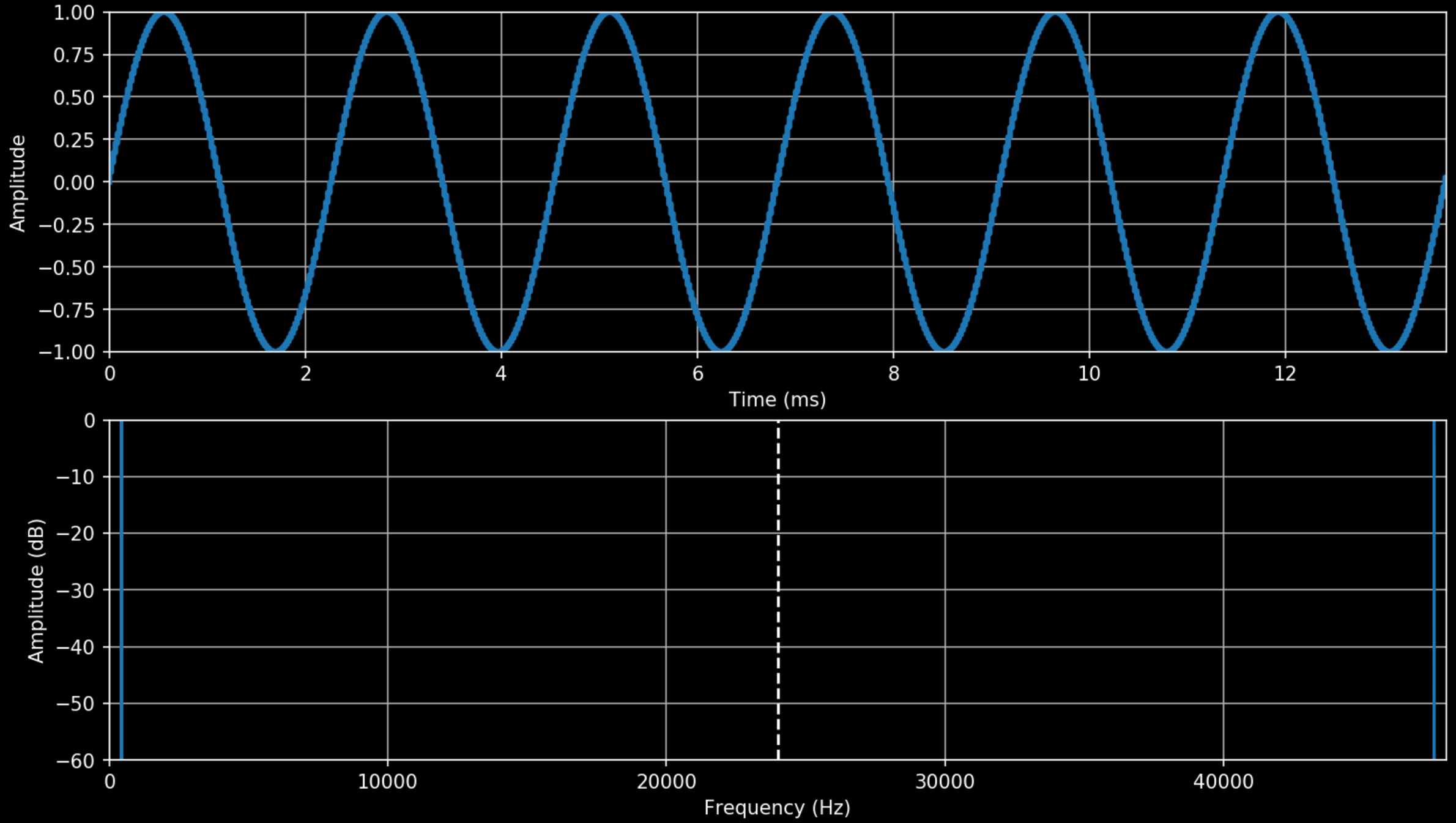
$$f_0 = 440 \cdot 2^{\frac{m - 69}{12}}$$

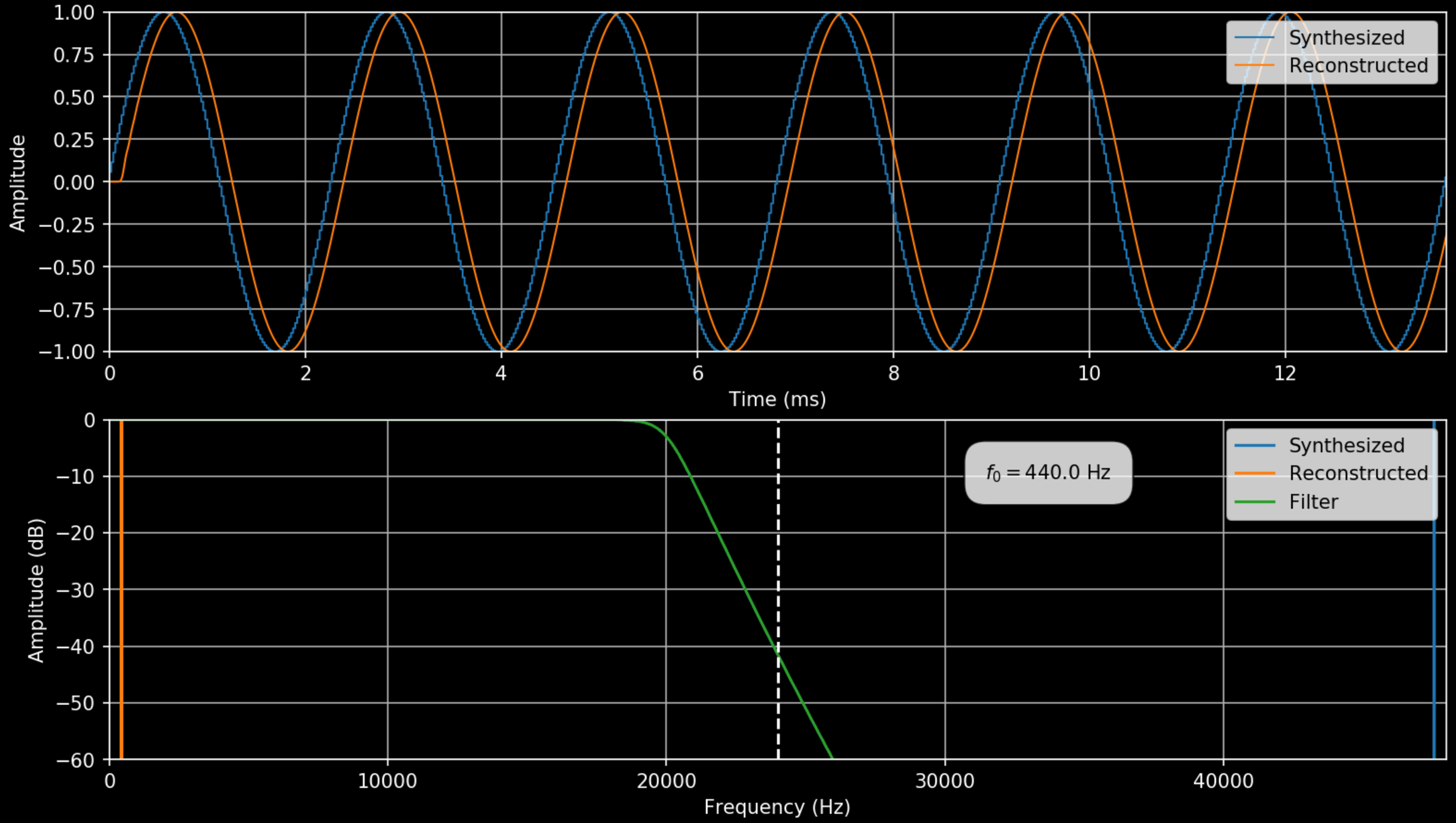
Digital to Analog Conversion

Aliasing:



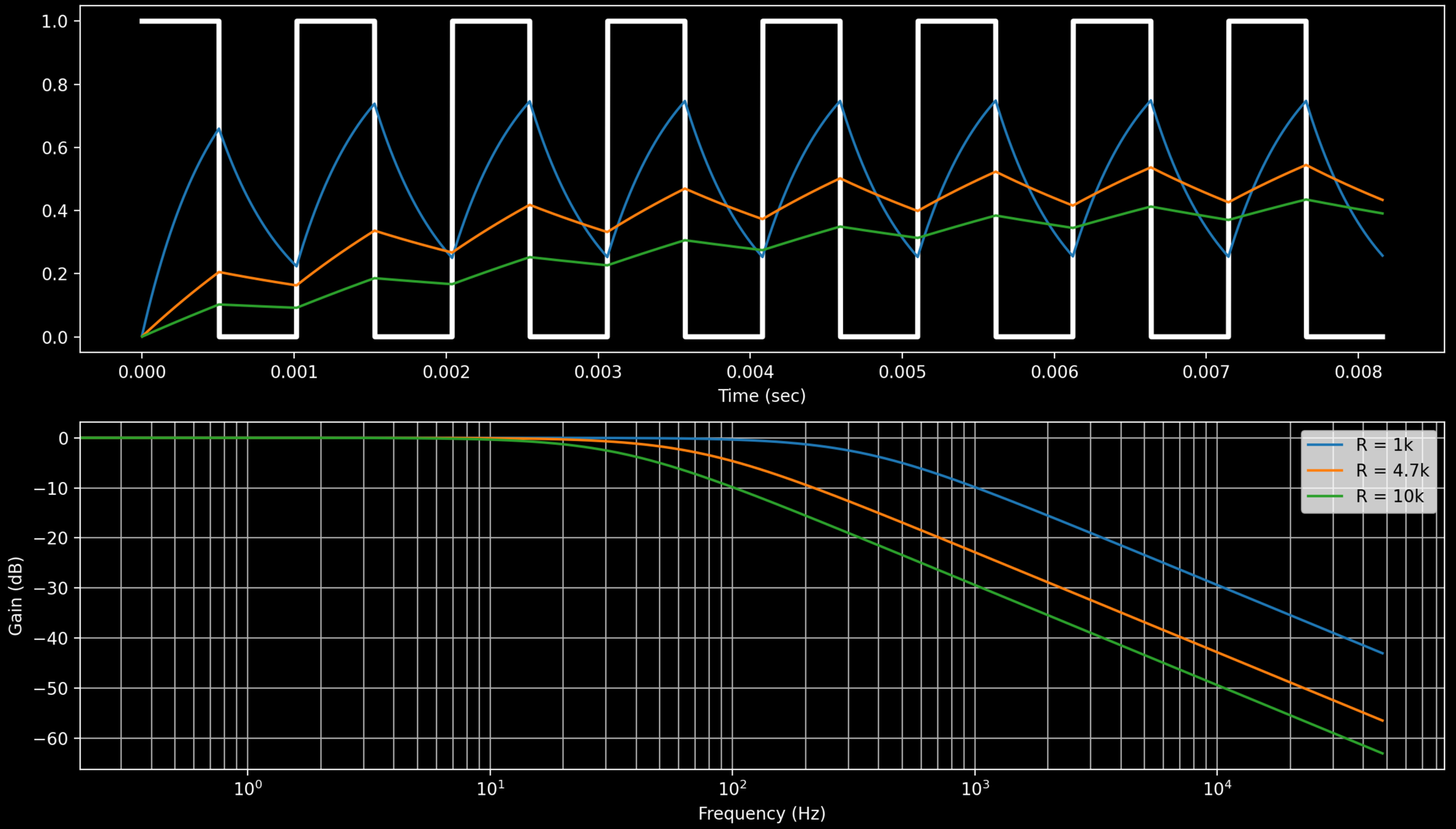

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Time (ms)
0
−5
−10
−15
−20
−25
−30
−35
−40
Amplitude (dB)
0
5000
10000
15000
20000
Frequency (Hz)

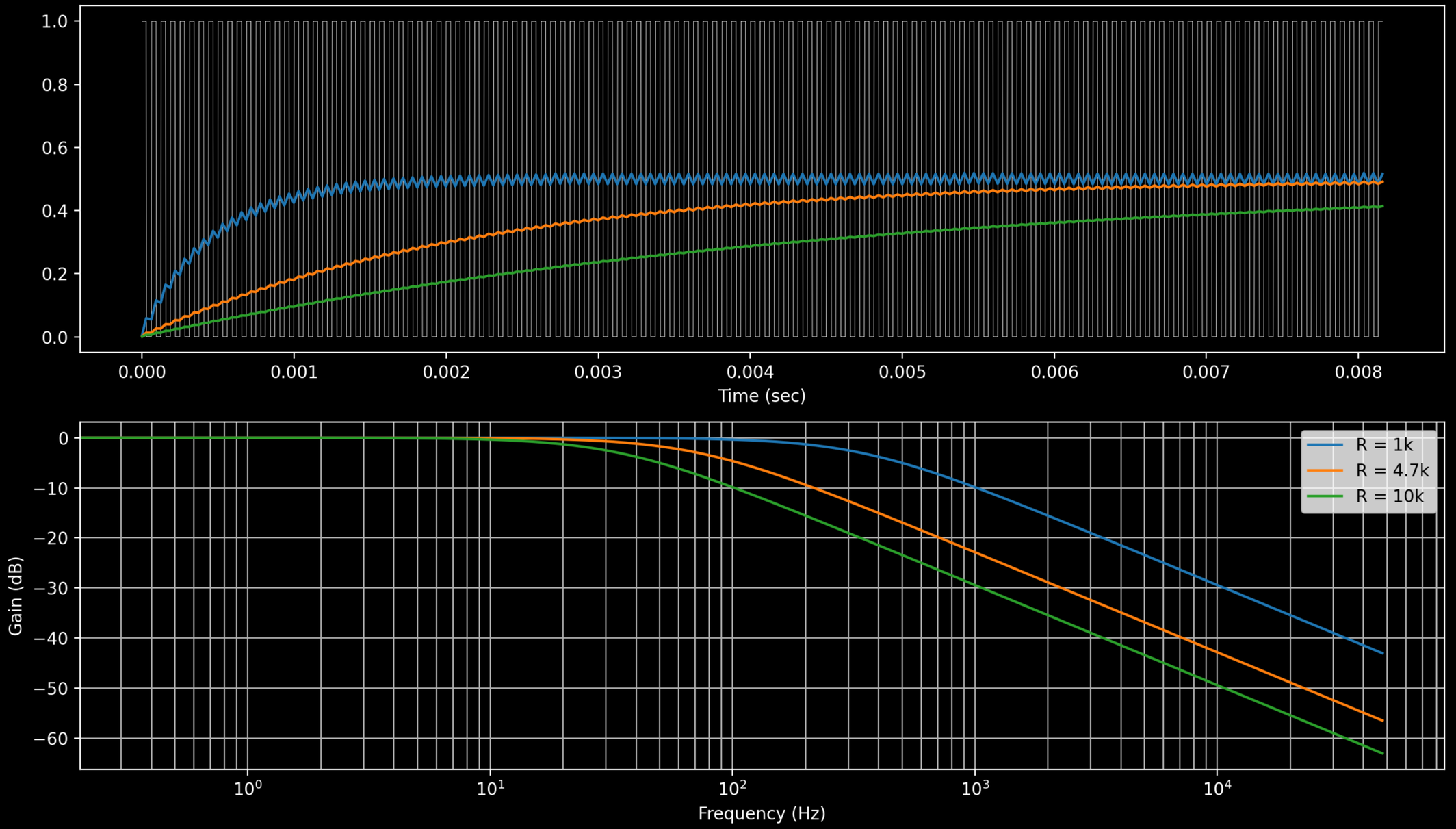

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0
2
4
6
8
10
12
Time (ms)
0
−10
−20
−30
−40
−50
−60
Amplitude (dB)
0
10000
20000
30000
40000
Frequency (Hz)


Amplitude
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
Time (ms)
Synthesized
Reconstructed
Amplitude (dB)
0
−10
−20
−30
−40
−50
−60
0
10000
20000
30000
40000
Frequency (Hz)
$f_0 = 440.0$ Hz
Synthesized
Reconstructed
Filter

Naive Synth Problems

- `analogWrite()` uses 980Hz PWM


1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
Time (sec)
0
−10
−20
−30
−40
−50
−60
Gain (dB)
10^0
10^1
10^2
10^3
10^4
Frequency (Hz)
R = 1k
R = 4.7k
R = 10k


1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
Time (sec)
0
−10
−20
−30
−40
−50
−60
Gain (dB)
10^0
10^1
10^2
10^3
10^4
Frequency (Hz)
R = 1k
R = 4.7k
R = 10k

Naive Synth Problems

- `analogWrite()` uses 980Hz PWM
- `delayMicroseconds()` wastes processor time; doesn't guarantee timing
- `analogRead()` requests samples one at a time instead of continuously
- AVRs lack floating point hardware, so `float` operations are slow
- `powf()` and `sinf()` are slooooooow

	Arduino	AVR/LibAG
1. Digital I/O	digitalWrite(2, HIGH)	PORTD |= (1 << PD2)
	2.4 µS	0.3 µS
2. Analog Output	analogWrite(6, val)	timer.pwm_write_a(val)
	4.4 µS	0.3 µS
3. Timing Control	delayMicroseconds(dt_us)	timer.init_ctc(ocr_val)
	dt_us µS	0 µS
4. Analog Input	val = analogRead(A0)	adc.update() val = adc.result[0]
	112 µS	2.4 µS

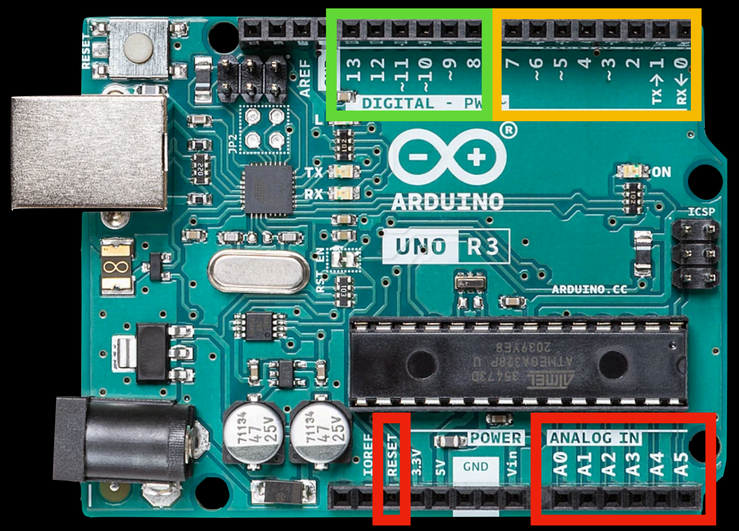
Topics

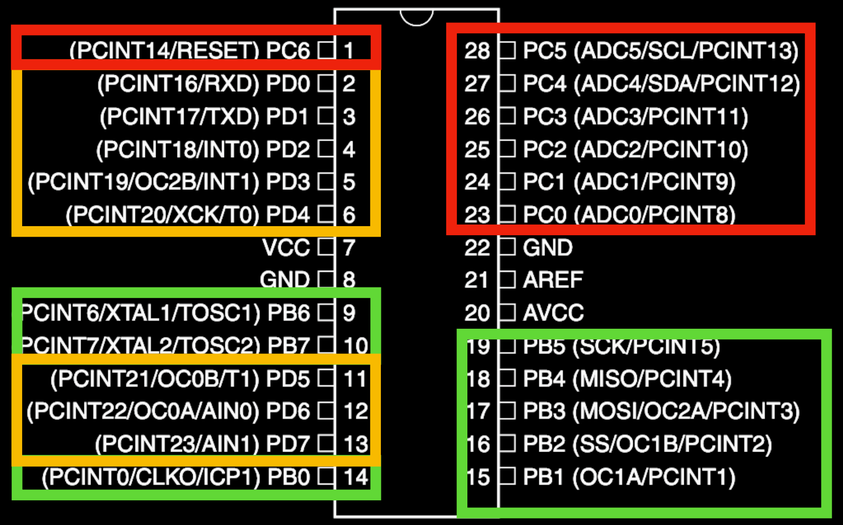
1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

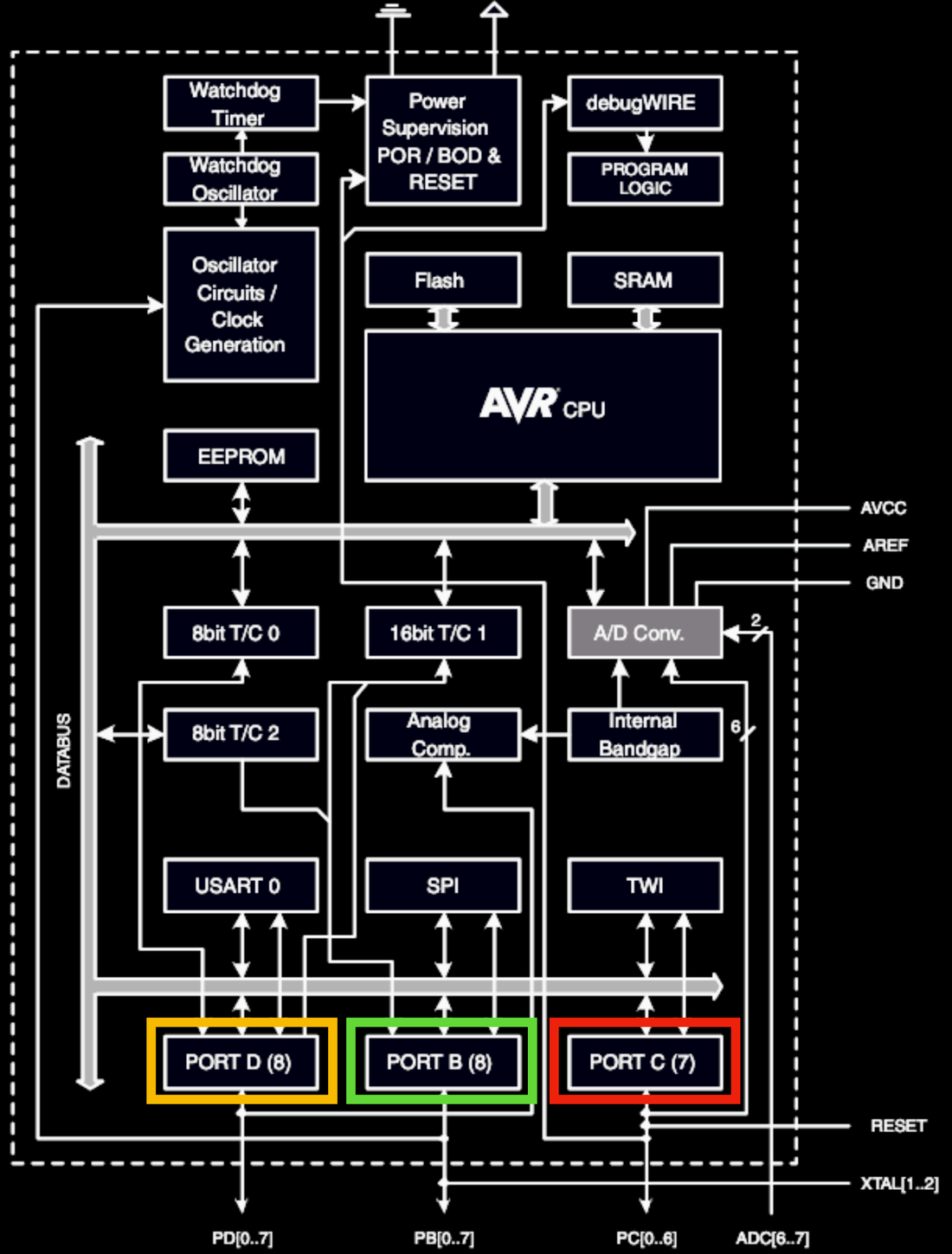
Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

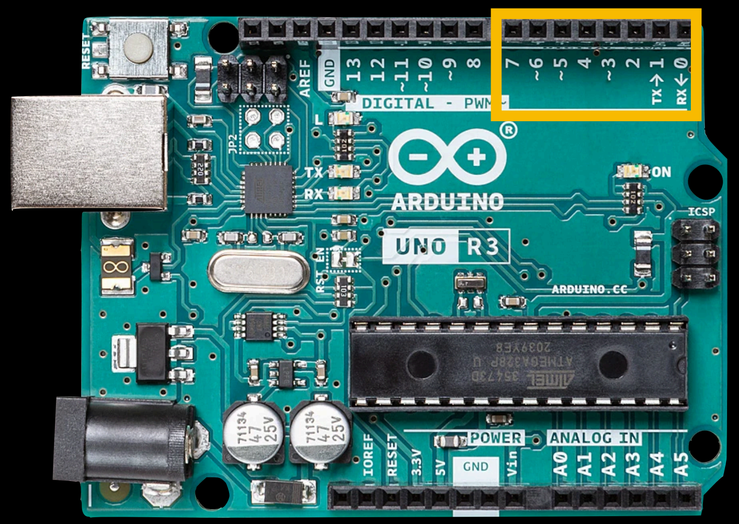
I/O Ports: Atmega328p

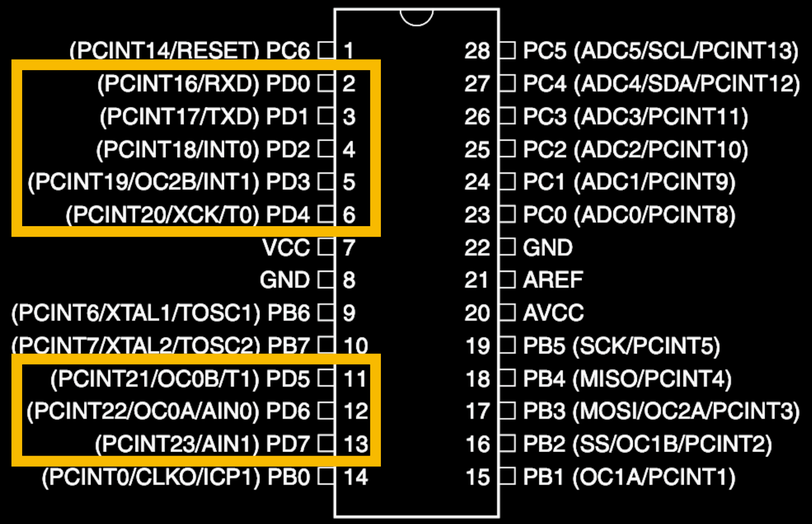


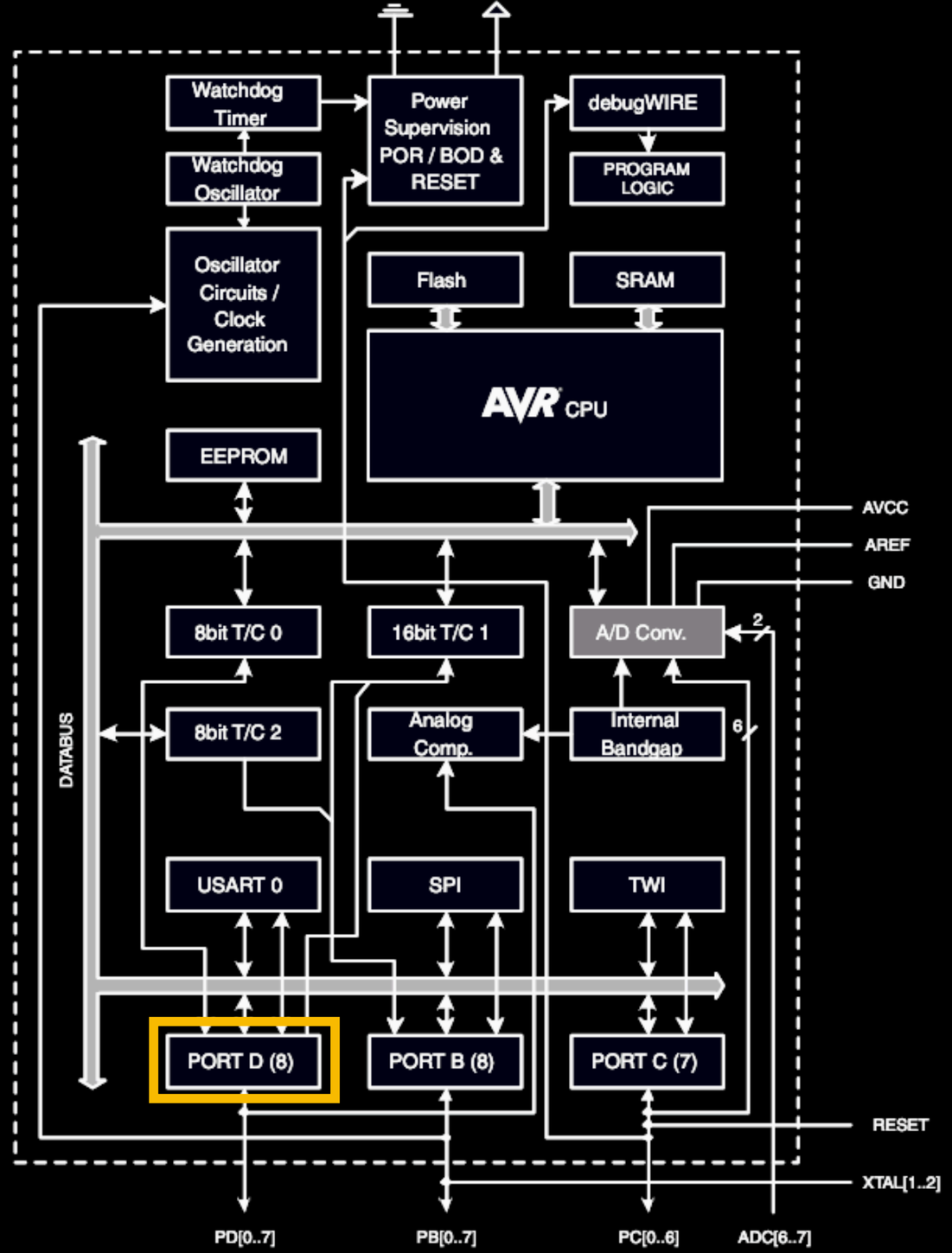




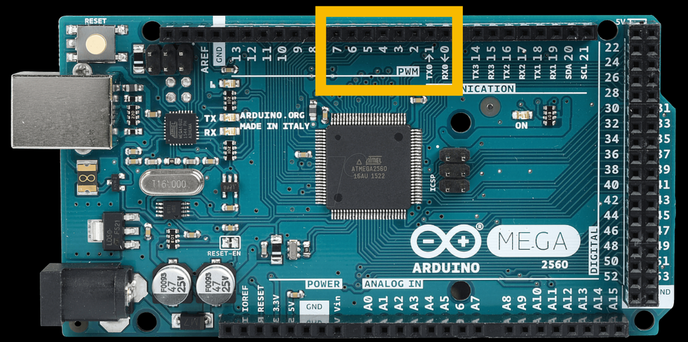
Port D: Atmega328p

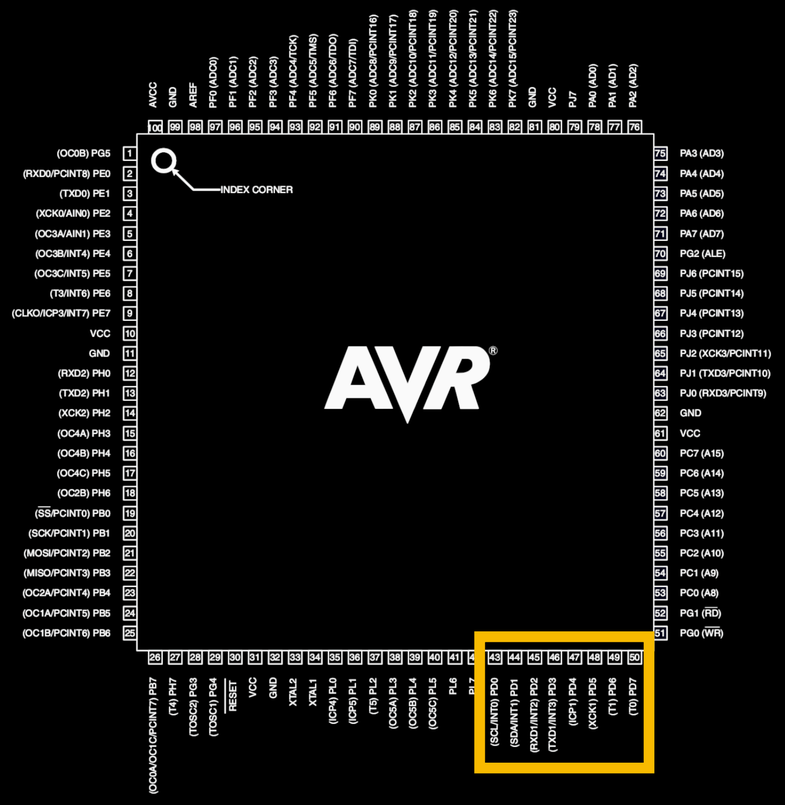


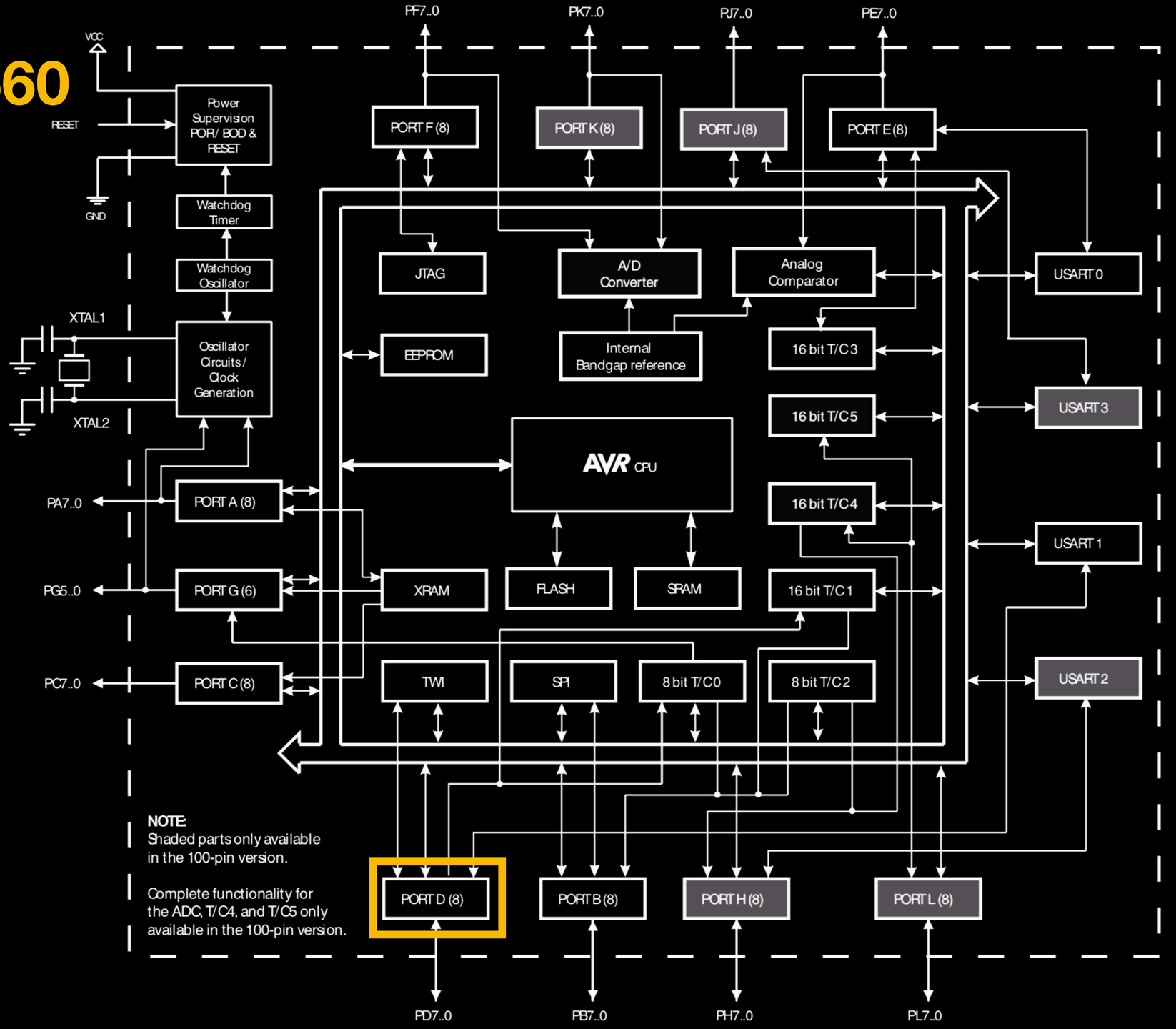




Port D: Atmega2560







13.4: I/O Ports Register Description

Write:

13.4.8: PORTD

7	6	5	4	3	2	1	0
PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0

- Control the states of digital pins 0-7

Read:

13.4.10: PIND

7	6	5	4	3	2	1	0
PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0

- Access the state of digital pins 0-7 (does not need to be configured as input)

Config:

13.4.9 DDRD

7	6	5	4	3	2	1	0
DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0

- Configure digital pins 0-7 as inputs (LOW) or outputs (HIGH)
  - If a pin is configured as input, setting the associated PORTD bit HIGH activates a pullup resistor

Setting Bits

	7	6	5	4	3	2	1	0
PORTD	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
Mask	0	0	1	0	0	0	0	0
PORTD | Mask	X_7	X_6	1	X_4	X_3	X_2	X_1	X_0

```
// Equivalent ways of setting bit 5
digitalWrite(5, HIGH);  // Arduino
PORTD |= 0b00100000;    // Binary
PORTD |= 32;            // Decimal
PORTD |= 0x20;          // Hex
PORTD |= (1 << 5);      // Bit shifting (2 to the 5th power)
PORTD |= (1 << PD5);    // Bit shifting (using AVR pin names)
```

Setting Bits (Multiple)

	7	6	5	4	3	2	1	0
PORTD	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
Mask	0	0	1	0	0	1	0	0
PORTD | Mask	X_7	X_6	1	X_4	X_3	1	X_1	X_0

```
// Equivalent ways of setting bit 5
digitalWrite(5, HIGH);  // Arduino
digitalWrite(2, HIGH);
PORTD |= 0b00100100;    // Binary
PORTD |= 36;            // Decimal
PORTD |= 0x24;          // Hex
PORTD |= (1 << 5) | (1 << 2);      // Bit shifting
PORTD |= (1 << PD5) | (1 << PD2);  // Bit shifting (using AVR pin names)
```

Clearing Bits

	7	6	5	4	3	2	1	0
PORTD	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
Mask	1	1	0	1	1	1	1	1
PORTD & Mask	X_7	X_6	0	X_4	X_3	X_2	X_1	X_0

```
// Equivalent ways of clearing bit 5
digitalWrite(5, LOW);   // Arduino
PORTD &= 0b11011111;    // Binary
PORTD &= 223;           // Decimal
PORTD &= 0xDF;          // Hex
PORTD &= ~(1 << 5);     // Bit shifting (2 to the 5th power) and inverting
PORTD &= ~(1 << PD5);   // Bit shifting (using AVR pin names) and inverting
```

Clearing Bits (Multiple)

	7	6	5	4	3	2	1	0
PORTD	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
Mask	1	1	0	1	1	0	1	1
PORTD & Mask	X_7	X_6	0	X_4	X_3	0	X_1	X_0

```
// Equivalent ways of clearing bit 5
digitalWrite(5, LOW);   // Arduino
digitalWrite(2, LOW);
PORTD &= 0b11011011;    // Binary
PORTD &= 219;           // Decimal
PORTD &= 0xDB;          // Hex
PORTD &= ~((1 << 5) | (1 << 2));       // Bit shifting and inverting
PORTD &= ~((1 << PD5) | (1 << PD2));   // Bit shifting and inverting
```

Toggling Bits

	7	6	5	4	3	2	1	0
PORTD	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
Mask	0	0	1	0	0	0	0	0
PORTD ^ Mask	X_7	X_6	$\overline{X_5}$	X_4	X_3	X_2	X_1	X_0

```
// Equivalent ways of toggling bit 5
digitalWrite(5, !digitalRead(5));   // Arduino
PORTD ^= 0b00100000;     // Binary
PORTD ^= 32;             // Decimal
PORTD ^= 0x20;           // Hex
PORTD ^= (1 << 5);       // Bit shifting (2 to the 5th power)
PORTD ^= (1 << PD2);     // Bit shifting (using AVR pin names)
```

Toggling Bits (Multiple)

	7	6	5	4	3	2	1	0
PORTD	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
Mask	0	0	1	0	0	1	0	0
PORTD ^ Mask	X_7	X_6	$\overline{X_5}$	X_4	X_3	$\overline{X_2}$	X_1	X_0

```
// Equivalent ways of toggling bit 5
digitalWrite(5, !digitalRead(5));   // Arduino
digitalWrite(2, !digitalRead(2));
PORTD ^= 0b00100100;     // Binary
PORTD ^= 36;             // Decimal
PORTD ^= 0x24;           // Hex
PORTD ^= (1 << 5) | (1 << 2);       // Bit shifting
PORTD ^= (1 << PD5) | (1 << PD2);   // Bit shifting (using AVR pin names)
```

Digital Input

```
bool state = false;

void setup() {
  pinMode(4, INPUT);
}

void loop() {
  bool s = digitalRead(4, HIGH);
  if (s != state) {
    handle_change();
    state = s;
  }
  delay(dt);
}
```

```
bool state = false;

void setup() {
  DDRD &= ~(1 << PD4);
}

void loop() {
  bool s = PIND & (1 << PD4);
  if (s != state) {
    handle_change();
    state = s;
  }
  delay(dt);
}
```

Digital Input

```
bool state = false;

void setup() {
  pinMode(4, INPUT_PULLUP);
}

void loop() {
  bool s = digitalRead(4, HIGH);
  if (s != state) {
    handle_change();
    state = s;
  }
  delay(dt);
}
```

```
bool state = false;

void setup() {
  DDRD &= ~(1 << PD4);   // Input
  PORTD |= (1 << PD4);   // Pull-up
}

void loop() {
  bool s = PIND & (1 << PD4);
  if (s != state) {
    handle_change();
    state = s;
  }
  delay(dt);
}
```

Digital Output

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

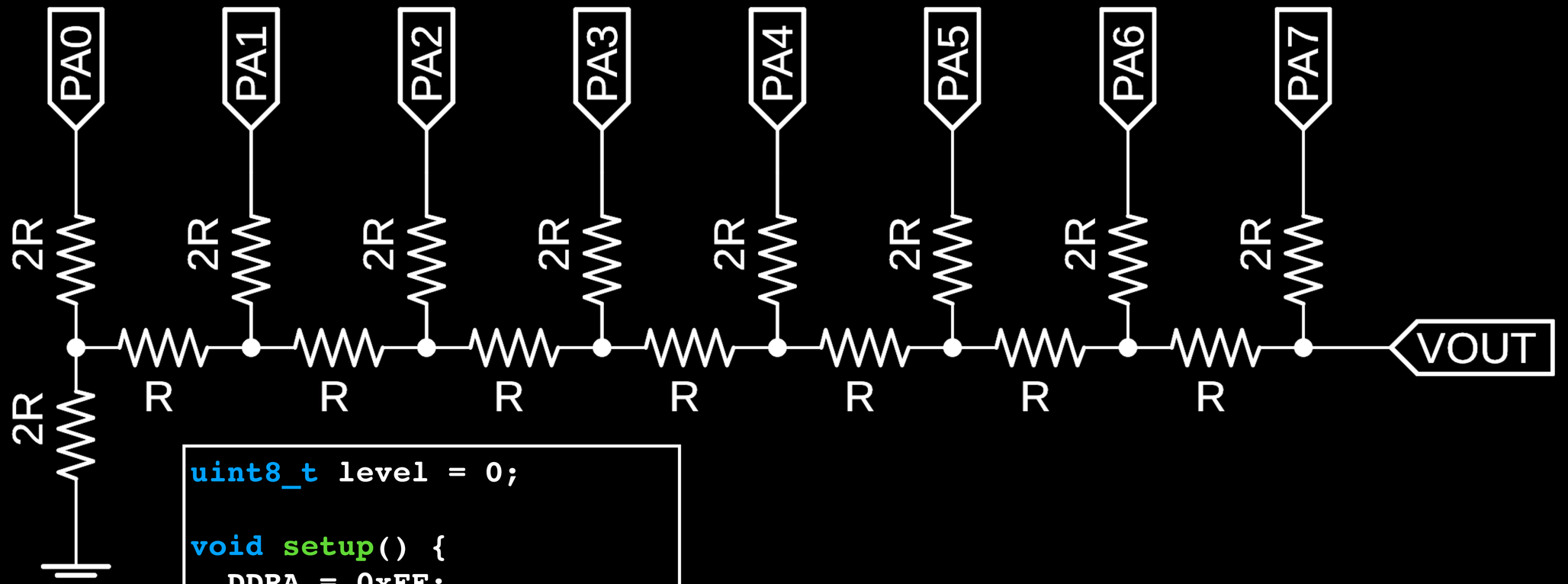
```
void setup() {
  DDRB |= (1 << PB5);
}

void loop() {
  PORTB |= (1 << PB5);
  delay(1000);
  PORTB &= ~(1 << PB5);
  delay(1000);
}
```

Digital Output

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

```
void setup() {
  DDRB |= (1 << PB5);
}

void loop() {
  PORTB ^= (1 << PB5);
  delay(1000);
}
```

R-2R Ladder DAC




```
uint8_t level = 0;

void setup() {
  DDRA = 0xFF;
}

void loop() {
  PORTA = level++;
  delayMicroseconds(125);
}
```

Sawtooth

$\approx 31.25\ Hz$

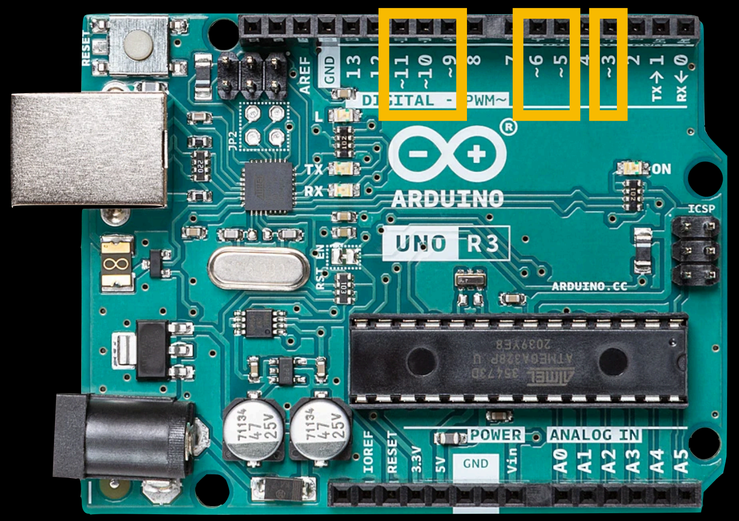
Topics

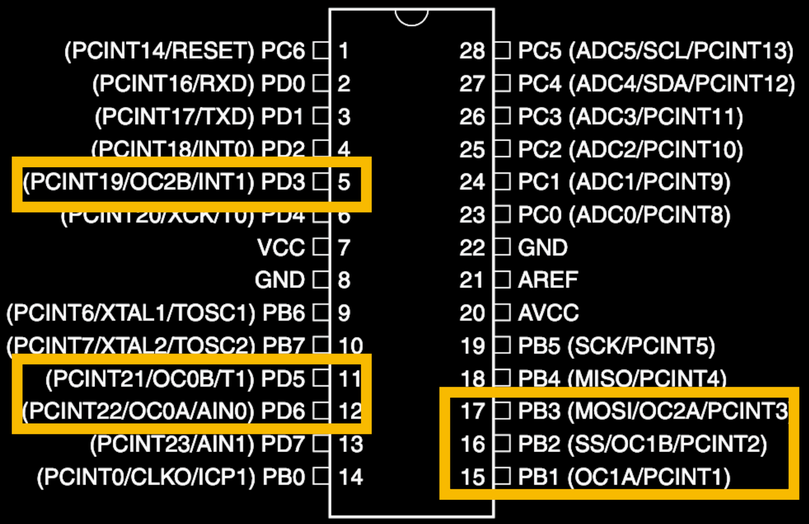
1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

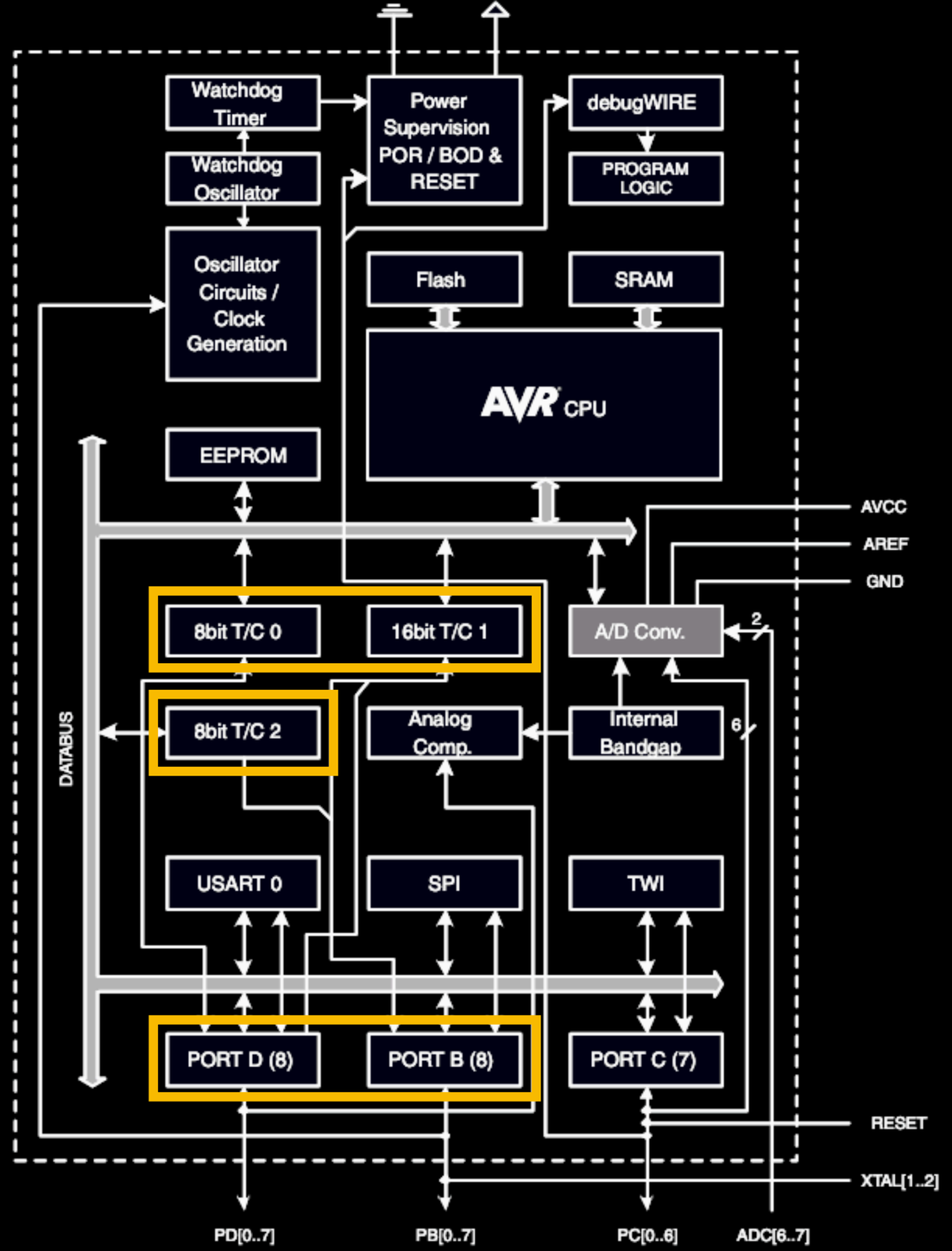
Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

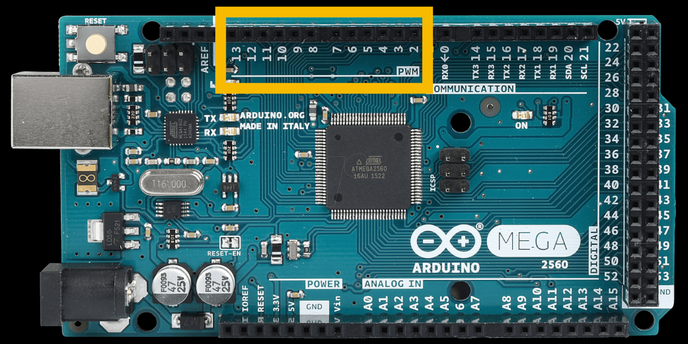
PWM: Atmega328p

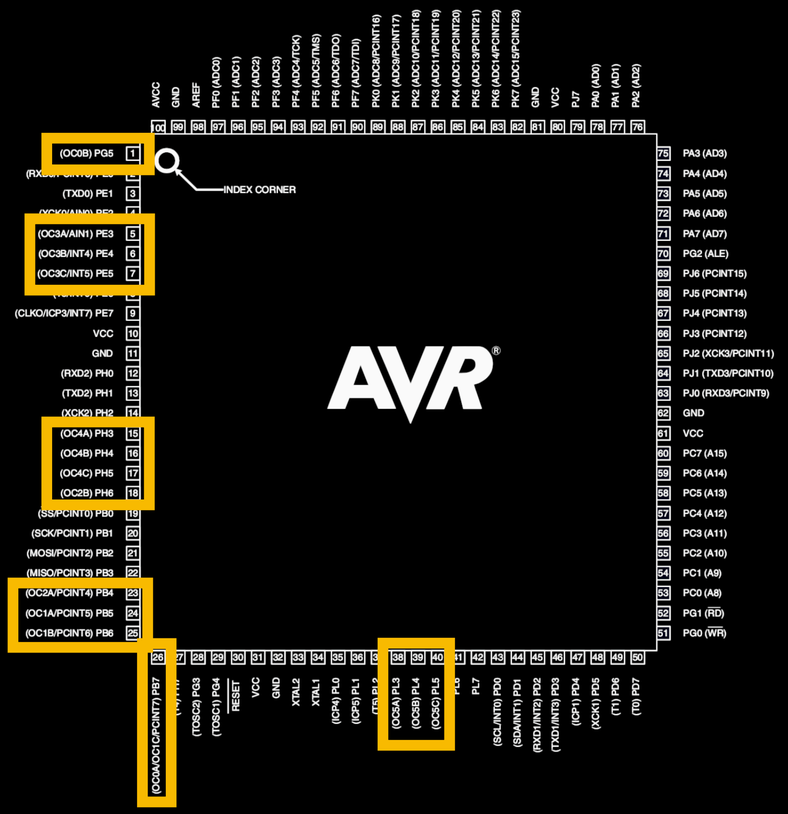


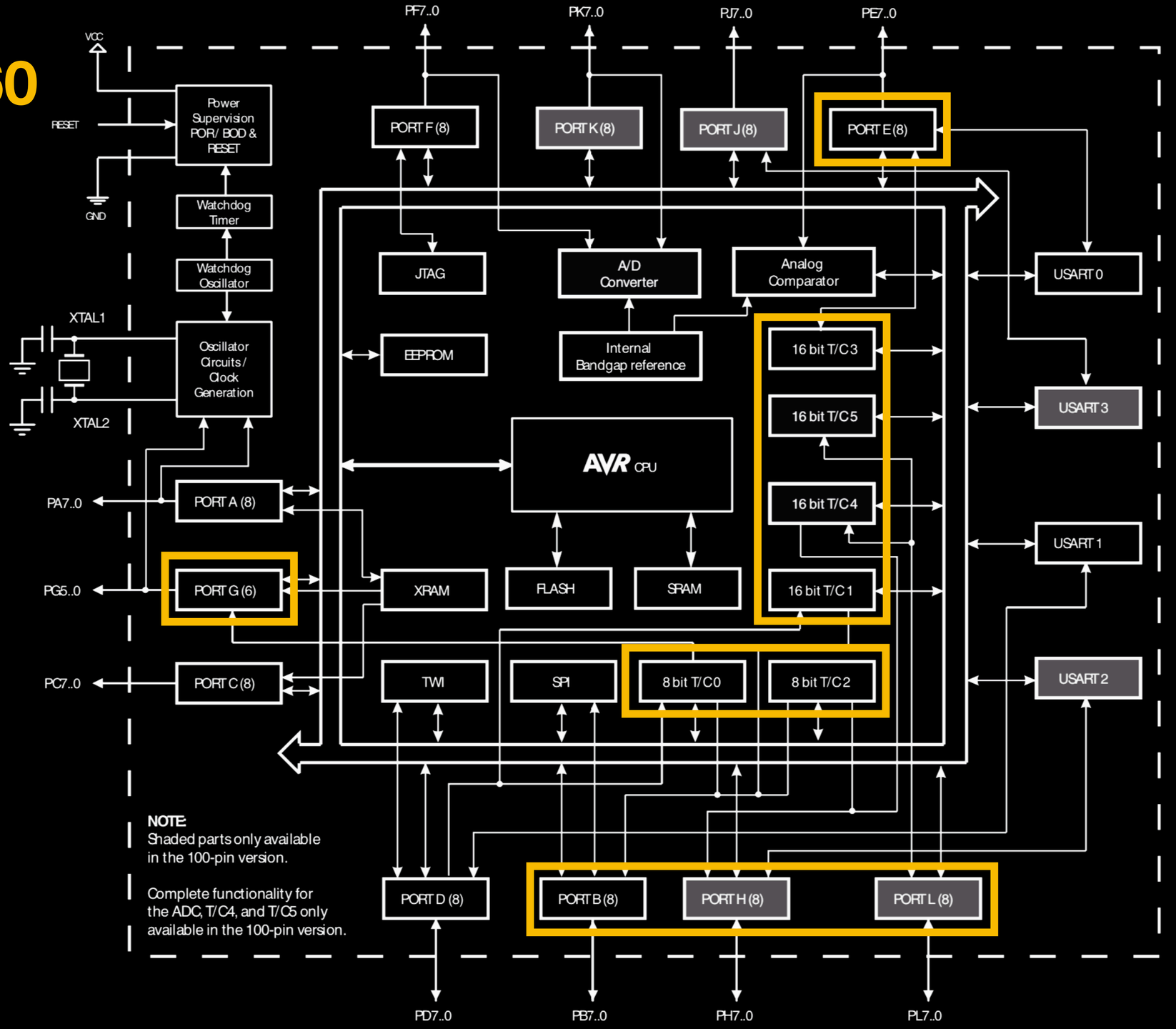




PWM: Atmega2560







14.9: Timer 0 (8-bit) Register Description

State:

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

- 8-bit register holding the current timer count value, in range [0, 255]

Channels:

	7	6	5	4	3	2	1	0
14.9.4: OCR0A	OCR0A[7:0]							
14.9.5: OCR0B	OCR0B[7:0]							

- 8-bit registers holding values in range [0, 255] that can map to [0, 100]% duty cycle on associated OCR pins

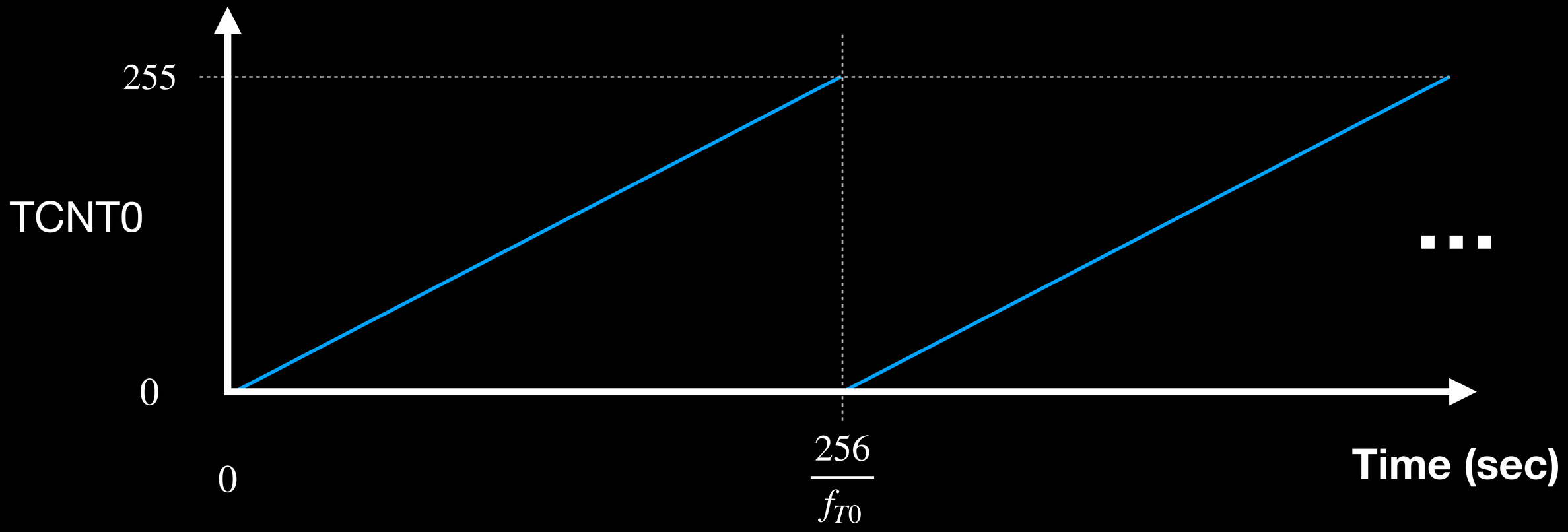
Config:

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

- These control the timer’s various modes and possible rates

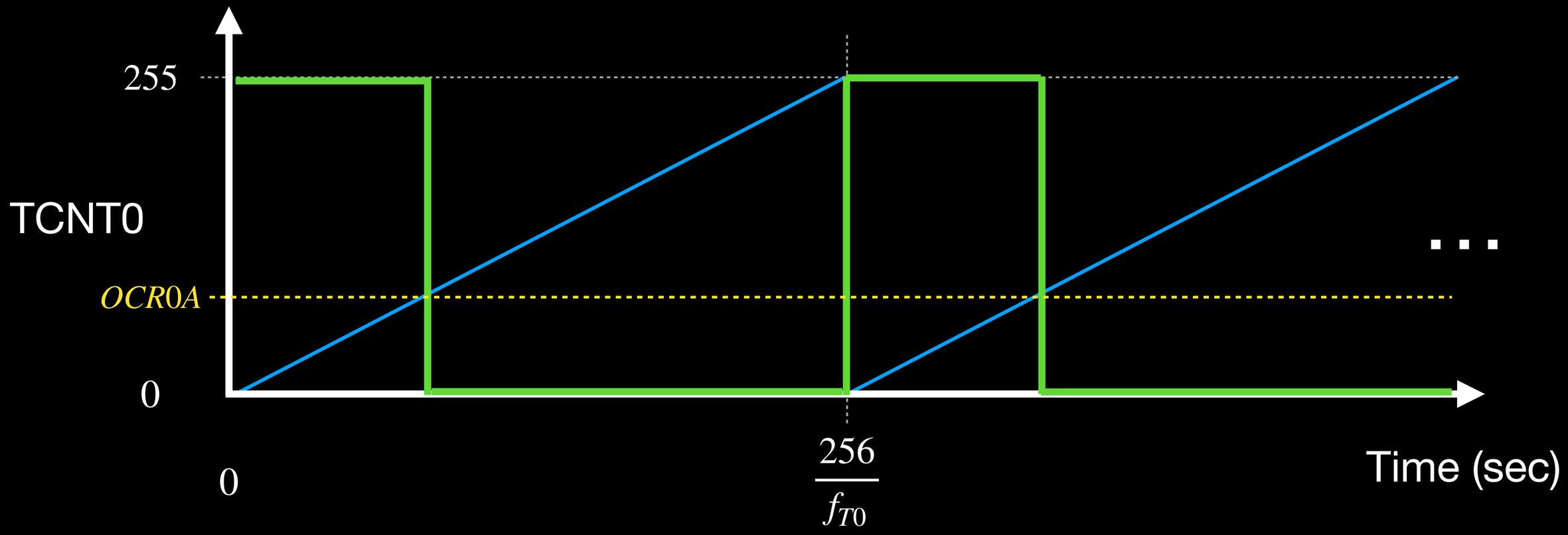
14.9.3: Timer 0 Count Value Register

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



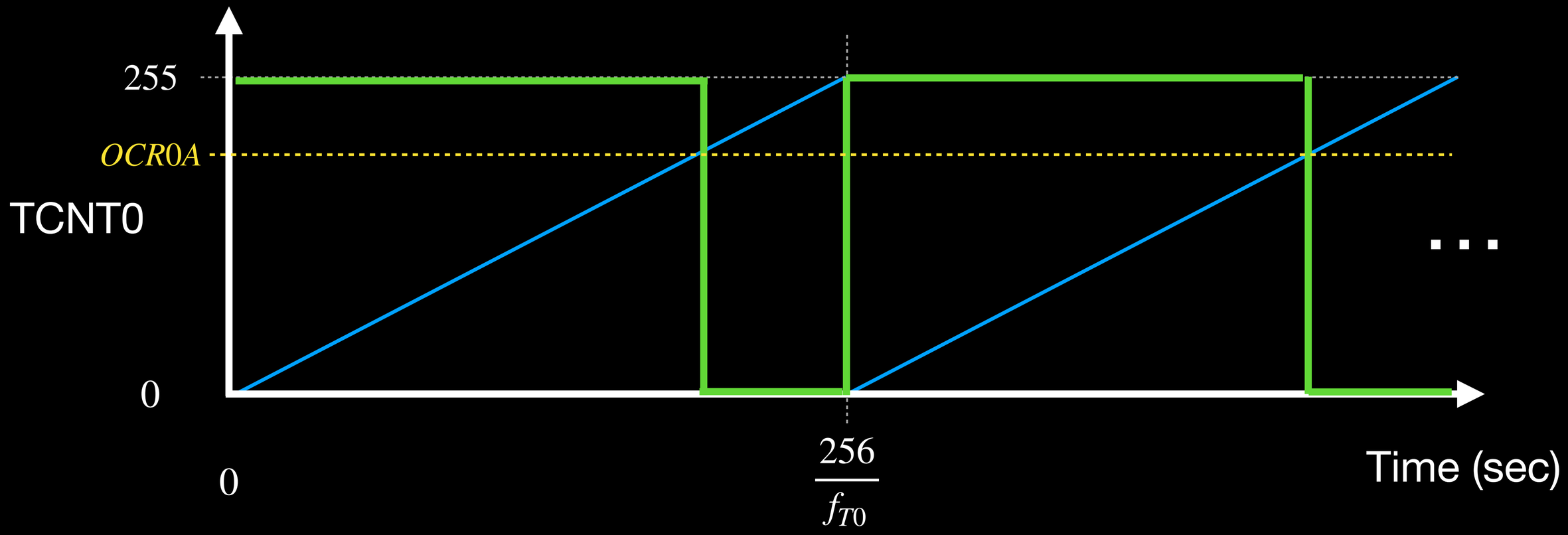
14.9.4-5: Timer 0 Output Compare Registers

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

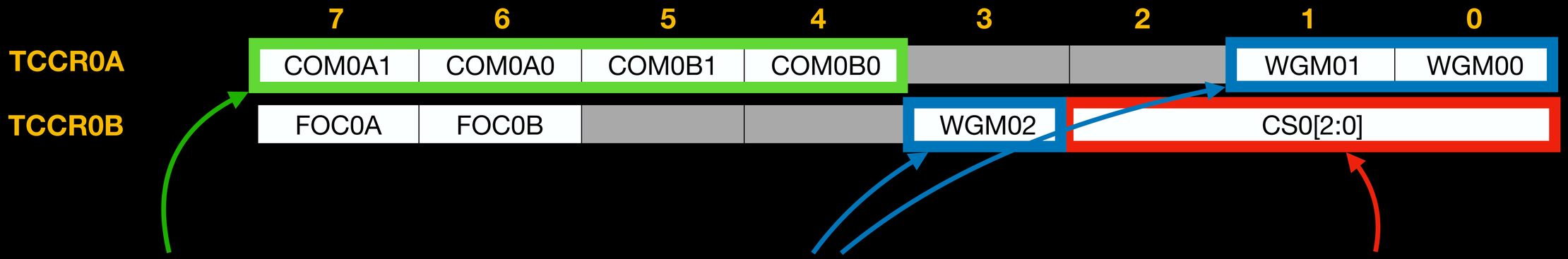


14.9.4-5: Timer 0 Output Compare Registers

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



14.9.1-2: Timer 0 Control Registers



“Compare Output Mode”

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

“Waveform Generator Mode”

- Configure for PWM or other modes

“Clock Select”

- Prescaler dividing system clock or external pin

Timer 0 Control Registers: Clock Select

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

$$f_{PWM} = \frac{\frac{16MHz}{1}}{256}$$

$$f_{PWM} = 62.5kHz$$

Table 14-9. Clock Select Bit Description

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)	16MHz
0	1	0	$clk_{I/O}$/8 (From prescaler)	2MHz
0	1	1	$clk_{I/O}$/64 (From prescaler)	250kHz
1	0	0	$clk_{I/O}$/256 (From prescaler)	62.5kHz
1	0	1	$clk_{I/O}$/1024 (From prescaler)	15.625kHz

Timer 0 Control Registers: Waveform Generator Mode

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

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM0	WGM0	WGM0	Timer/Counter Mode of	TOP	Update OCR0x	TOV Flag Set
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

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

Table 14-3. Compare Output Mode, Fast PWM

Channel A

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
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)

Table 14-6. Compare Output Mode, Fast PWM

Channel B

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on Compare Match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on Compare Match, clear OC0B at BOTTOM, (inverting mode)

When CV Resolution Matters

$$\frac{256 \text{ steps}}{5 \cancel{\text{ volts}}} \cdot \frac{1 \cancel{\text{ volt}}}{1 \cancel{\text{ oct}}} \cdot \frac{1 \cancel{\text{ oct}}}{12 \text{ semitones}} = 4.27 \frac{\text{steps}}{\text{semitone}}$$

Bit Resolution	steps/semitone
8-bit	4.27
9-bit	8.53
10-bit	17.07
11-bit	34.13
12-bit	68.27
13-bit	136.53
14-bit	273.07
15-bit	546.13
16-bit	1092.27

15.11: Timer 1 (16-bit) Register Description

State:

	7	6	5	4	3	2	1	0
15.11.4: TCNT1L	TCNT1L[7:0]							
TCNT1H	TCNT1H[7:0]							

- Pair of 8-bit registers holding the current timer count value, in range [0, 65535]

Channels:

	7	6	5	4	3	2	1	0
15.11.5: OCR1AL	OCR1AL[7:0]							
OCR1AH	OCR1AH[7:0]							

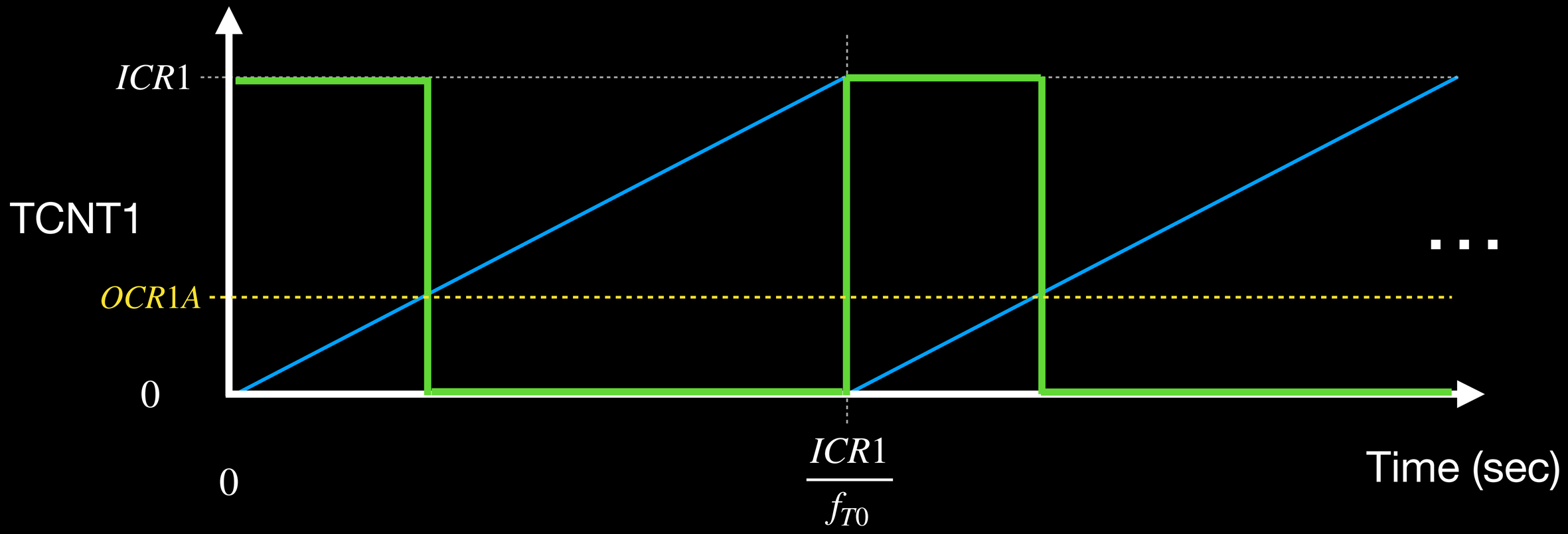
	7	6	5	4	3	2	1	0
15.11.6: OCR1BL	OCR1BL[7:0]							
OCR1BH	OCR1BH[7:0]							

- Pairs of 8-bit registers holding values in range [0, 65535] that can map to [0, 100]% duty cycle on associated OCR pins

15.11.7: Timer 1 (16-bit) Input Capture Registers

	7	6	5	4	3	2	1	0
ICR1L	ICR1L[7:0]							
ICR1H	ICR1H[7:0]							

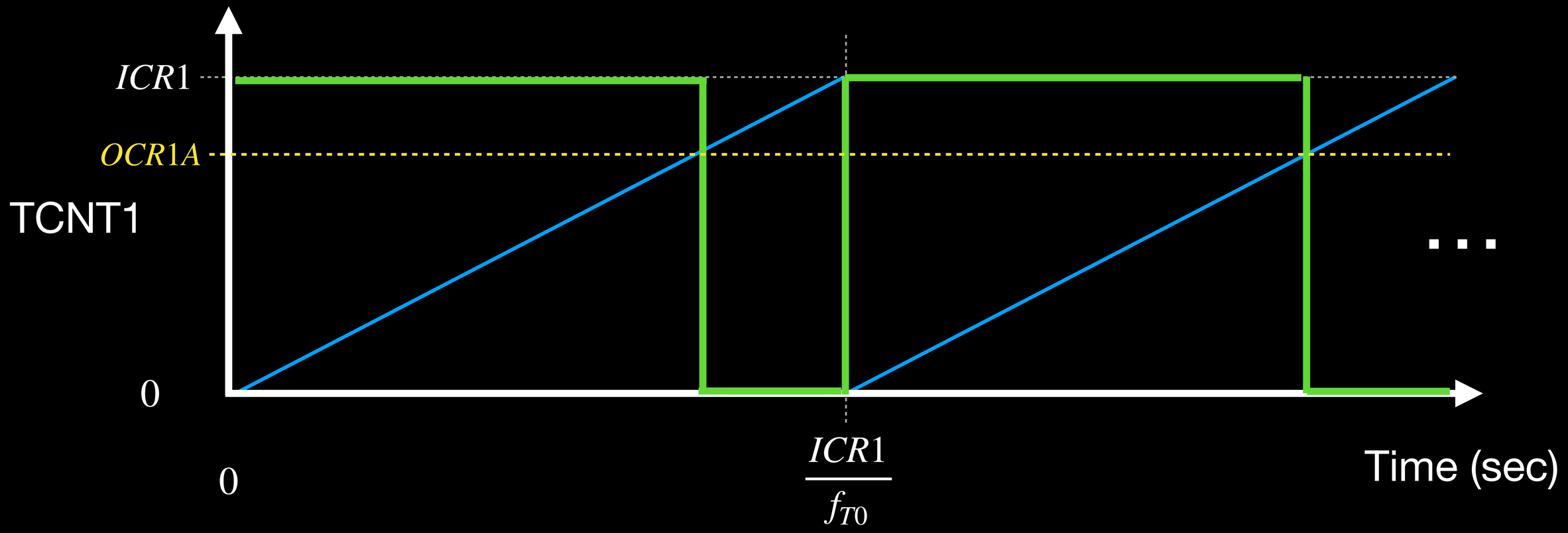
- Pair of 8-bit registers holding a value in range [0, 65535] that determine the TOP value at which the timer resets



15.11.7: Timer 1 (16-bit) Input Capture Registers

	7	6	5	4	3	2	1	0
ICR1L	ICR1L[7:0]							
ICR1H	ICR1H[7:0]							

- Pair of 8-bit registers holding a value in range [0, 65535] that determine the TOP value at which the timer resets



15.11.7: Timer 1 (16-bit) Input Capture Registers

- Pair of 8-bit registers holding a value in range [0, 65535] that determine the TOP value at which the timer resets

Rate

$$f_{PWM} = \frac{\frac{16MHz}{prescaler}}{ICR1 + 1}$$

Resolution

$$OCR1 \in [0, ICR1]$$

15.11.7: Timer 1 (16-bit) Input Capture Registers

- **Pair of 8-bit registers holding a value in range [0, 65535] that determine the TOP value at which the timer resets**

ICR1 (Dec)	ICR1 (Hex)	Rate (f_{T0} = 16MHz/1)	Resolution
255	0x00FF	62500	8-bit
511	0x01FF	31250	9-bit
1023	0x03FF	15625	10-bit
2047	0x07FF	7812.5	11-bit
4095	0x0FFF	3906.25	12-bit
8191	0x1FFF	1953.125	13-bit
16383	0x3FFF	976.5625	14-bit
32767	0x7FFF	488.28125	15-bit
65535	0xFFFF	244.140625	16-bit

15.11.1-3: Timer 1 Control Registers, WGM

	7	6	5	4	3	2	1	0
TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0			WGM11	WGM10
TCCR1B	ICNC1	ICES1		WGM13	WGM12	CS1[2:0]		

Table 20-6. Waveform Generation Mode Bit Description

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	TOP	Update OCR0x at	TOV Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0X03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Topics

1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

Timer 0 Control Registers: Waveform Generator Mode

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

Table 14-8. Waveform Generation Mode Bit Description

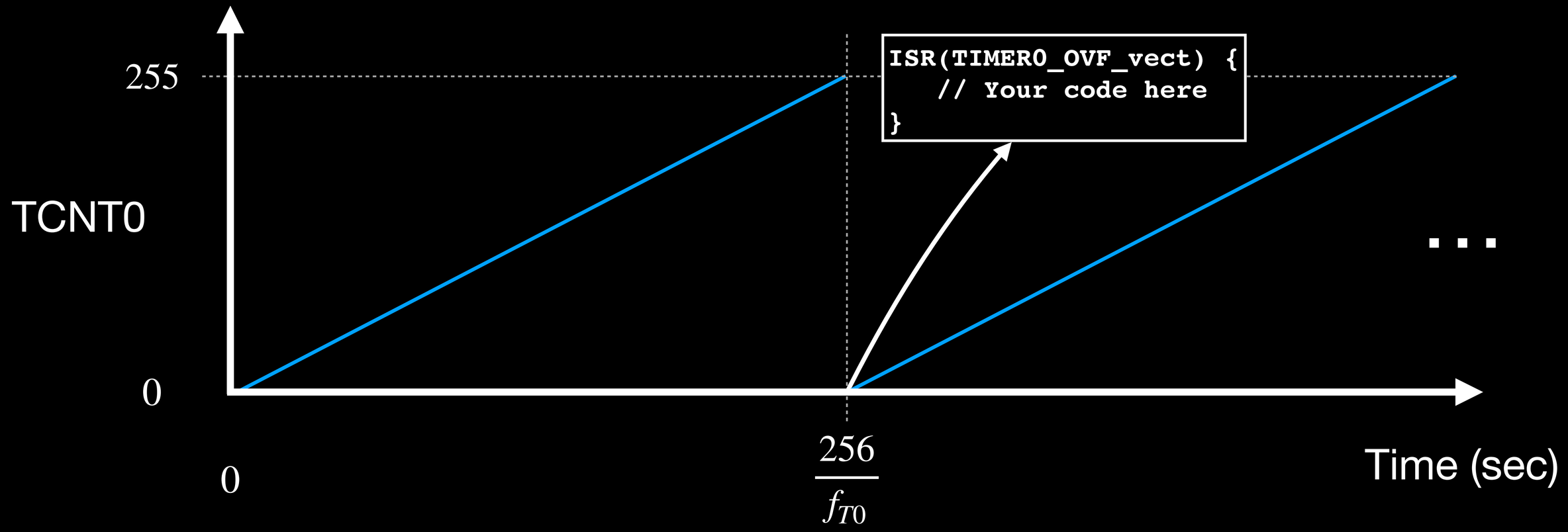
Mode	WGM0	WGM0	WGM0	Timer/Counter Mode of	TOP	Update OCR0x	TOV Flag Set
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Timer 0 Interrupt Mask Register

"Timer Overflow Interrupt Enable"

14.9.6: TIMSK0

7	6	5	4	3	2	1	0
					OCIEB	OCIEA	TOIE

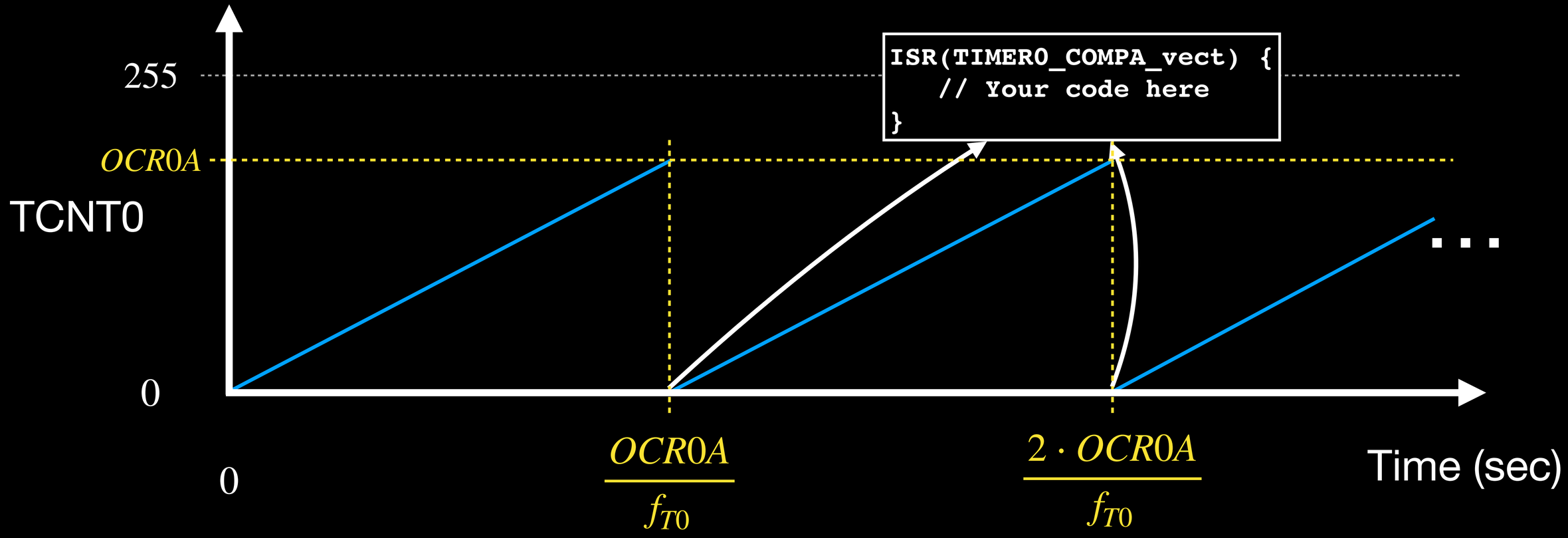


Timer 0 Interrupt Mask Register

"Output Compare Interrupt Enable"

14.9.6: TIMSK0

7	6	5	4	3	2	1	0
					OCIEB	OCIEA	TOIE



Timer 0 Interrupt Mask Register

"Output Compare Interrupt Enable"

14.9.6: TIMSK0

Sample Rate

$$f_s = \frac{\frac{16MHz}{prescaler}}{OCR0A + 1}$$

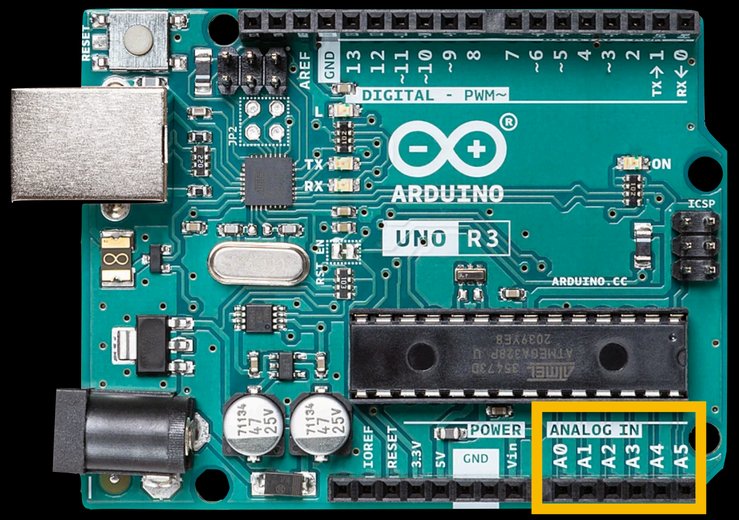
Topics

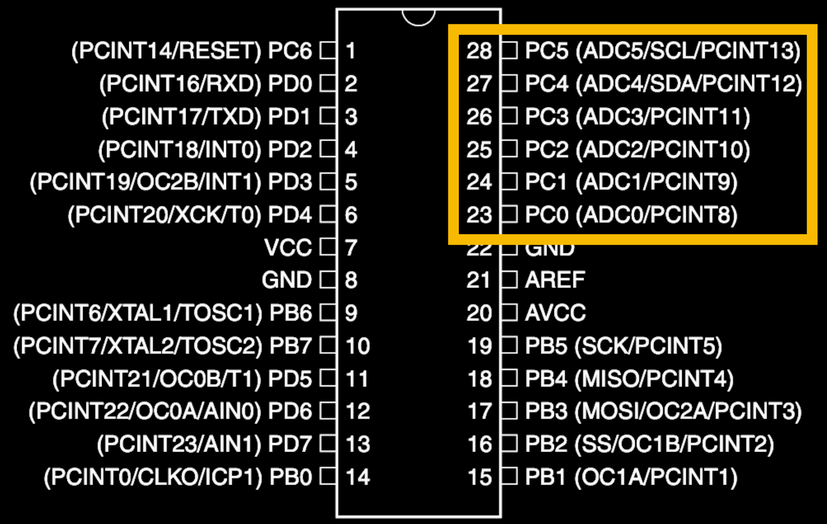
1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

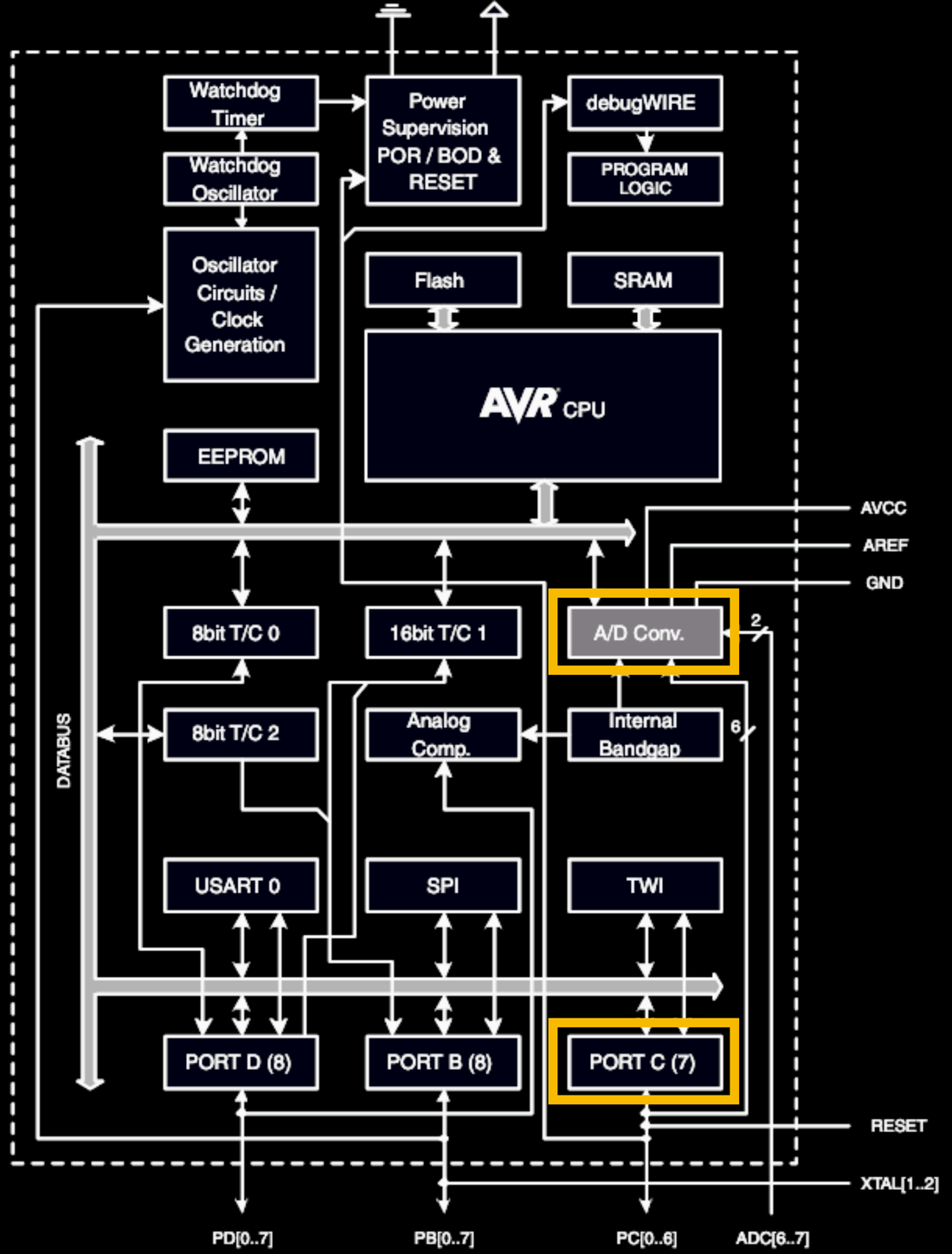
Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

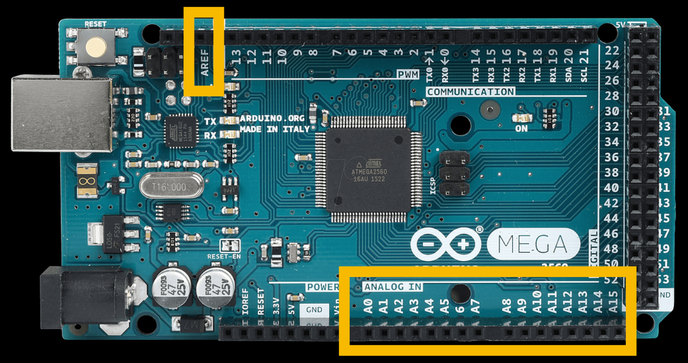
ADC: Atmega328p

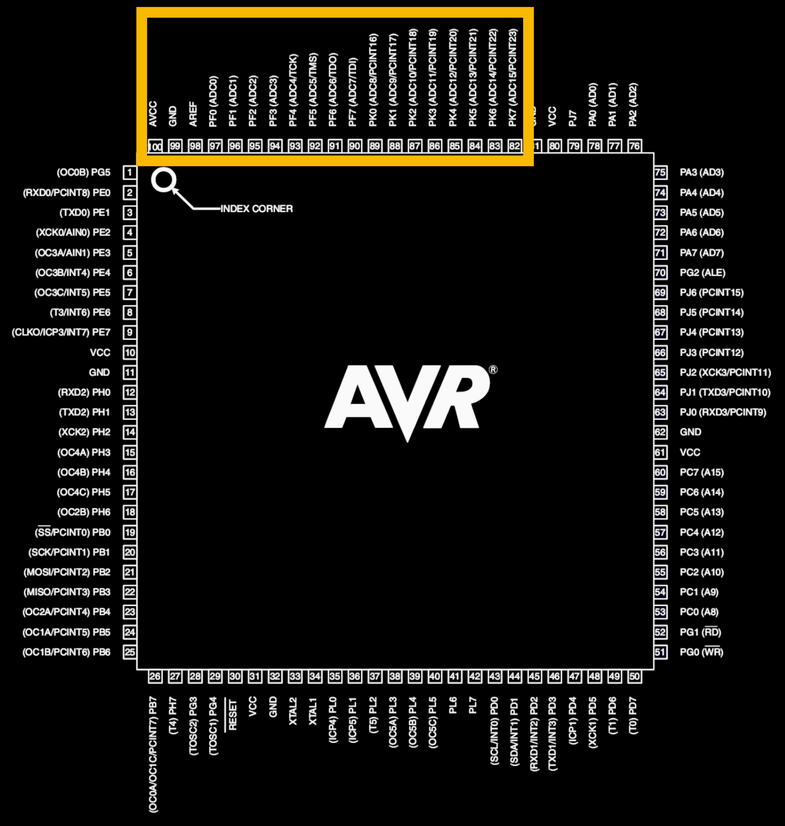


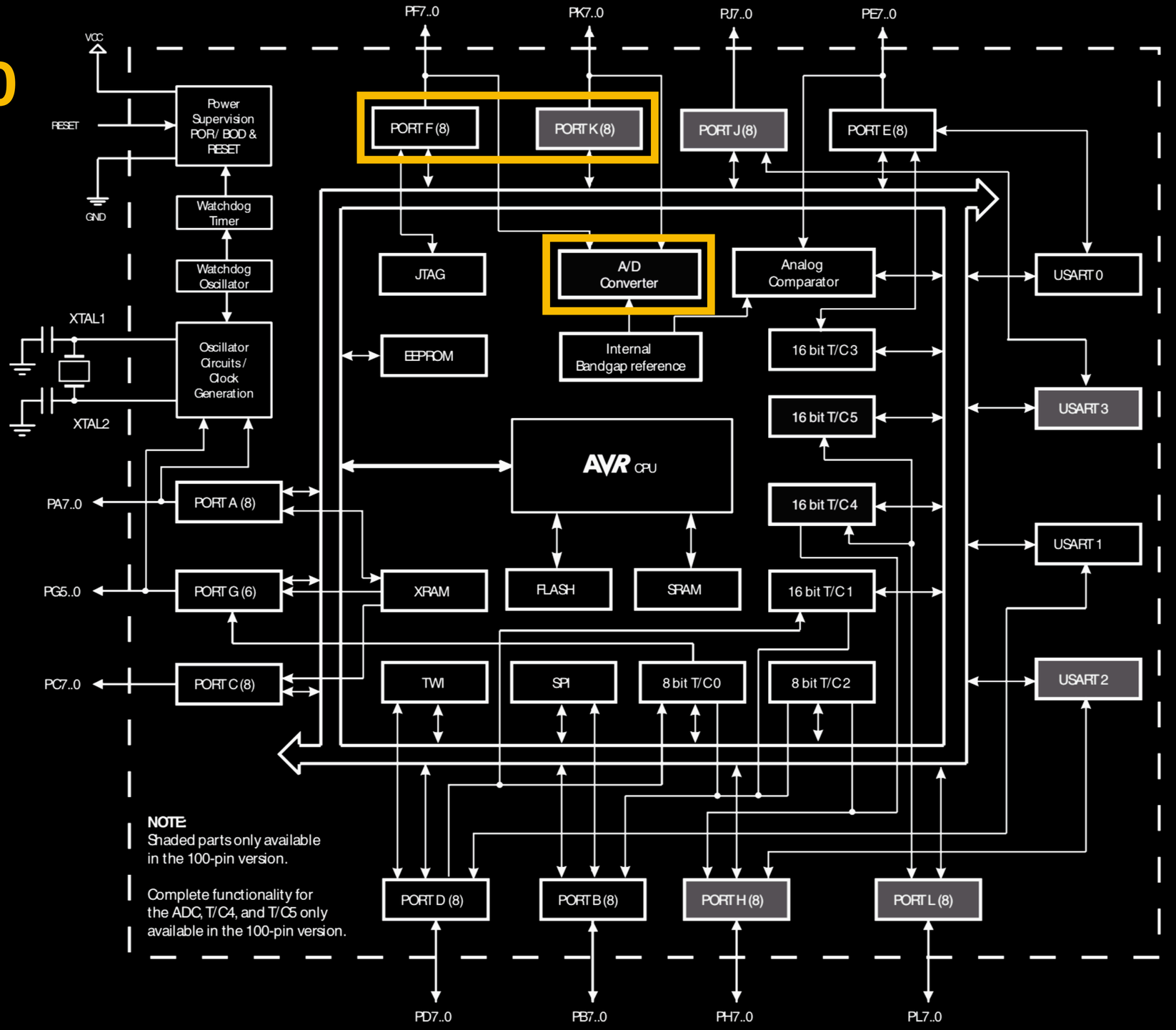




ADC: Atmega2560







26.8 ADC Register Description

Channels:

	7	6	5	4	3	2	1	0
23.9.1: ADMUX	REFS1	REFS2	ADLAR	-	MUX3	MUX2	MUX1	MUX0

- Select ADC channels, configure voltage reference, configure results format

Results:

	7	6	5	4	3	2	1	0
23.9.3: ADCL	ADCL[7:0]							
ADCH	ADCH[7:0]							

- Retrieve the ADC conversion results (most and least significant bytes)

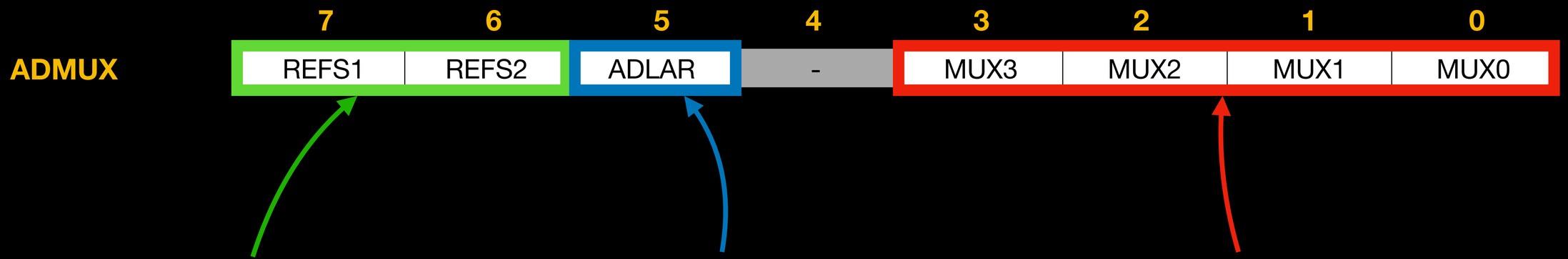
Config:

	7	6	5	4	3	2	1	0
23.9.2: ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
23.9.4 ADCSRB	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0

- Configure timing; initiate conversions manually, or trigger conversions automatically

23.9.1 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]”

- Atmega328p has 6-8 channels
- Atmega2560 has additional channels, differential measurement, and gain options

ADMUX: Voltage Reference

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS2	ADLAR	-	MUX3	MUX2	MUX1	MUX0

Table 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

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

ADMUX: Right Adjust ADC Results

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS2	ADLAR	-	MUX3	MUX2	MUX1	MUX0

ADLAR = 0
(Right Adjusted)

	7	6	5	4	3	2	1	0
ADCH	-	-	-	-	-	-	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

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

***Note: the ADC updates after ADCH is read, so read from ADCL first!**

ADMUX: Left Adjust ADC Results

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS2	ADLAR	-	MUX3	MUX2	MUX1	MUX0

ADLAR = 1
(Left Adjusted)

	7	6	5	4	3	2	1	0
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	-	-	-	-	-	-

```
// Get 10-bit conversion result
uint16_t result = (ADCL >> 6) | (ADCH << 2);

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

*Note: the ADC updates after ADCH is read, so read from ADCL first!

ADMUX: Channel Selection

	7	6	5	4	3	2	1	0
ADMUX	REFS1	REFS2	ADLAR	-	MUX3	MUX2	MUX1	MUX0

Table 23-4. Input Channel Selection

MUX3	MUX2	MUX1	MUX0	Single Ended Input
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7
1	0	0	0	ADC8
1	0	0	1	(reserved)
1	0	1	0	(reserved)
1	0	1	1	(reserved)
1	1	0	0	(reserved)
1	1	0	1	(reserved)
1	1	1	0	1.1V (V_{BG})
1	1	1	1	0V (GND)

ADCSR: 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
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCSRB	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0

“Enable”

- Set bit to use ADC
- Clear to disable

ADCSR: ADC Control and Status Registers

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

	7	6	5	4	3	2	1	0
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCSRB	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0

“Start Conversion”

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

“Interrupt Flag”

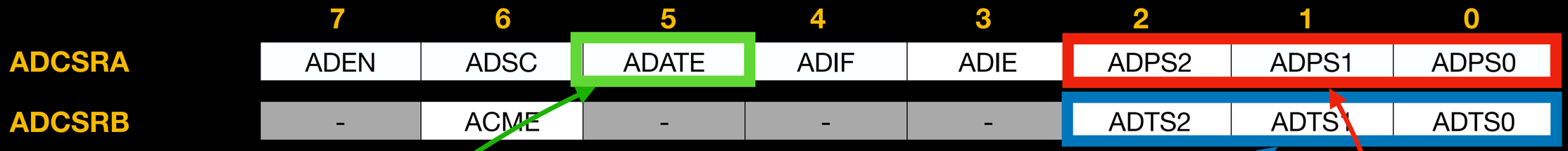
- ADC sets bit HIGH when conversion is complete
- Check the bit before retrieving the result

“Interrupt Enable”

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

ADCSR: 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
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCSRB	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0

Table 23-6. ADC Auto Trigger Source Selection

ADTS2	ADTS1	ADTS2	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

Free Running Mode

- Trigger conversions at rate

$$f_s = \frac{\frac{16\ MHz}{prescaler}}{13}$$

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
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCSRB	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0

Table 23-6. ADC Auto Trigger Source Selection

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

Timer OCR Trigger

- Trigger conversions at rate

$$f_s = \frac{\frac{16\ MHz}{prescaler_{Timer0}}}{OCR0A + 1}$$

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
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCSRB	-	ACME	-	-	-	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

Table 23-6. ADC Auto Trigger Source Selection

ADPS2	ADPS1	ADPS0	Division Factor	ADC Clock Rate (f_{clk}=16MHz)	Free Running f_s
0	0	0	2	8MHz	615.38kHz
0	0	1	2	8MHz	615.38kHz
0	1	0	4	4MHz	307.69kHz
0	1	1	8	2MHz	153.85kHz
1	0	0	16	1MHz	76.921kHz
1	0	1	32	500kHz	38.462kHz
1	1	0	64	250kHz	19.231kHz
1	1	1	128	125kHz	9.615kHz

Topics

1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

	`float f1, f2, f3;` [0, 1.0]	`uint16_t u1, u2, u3;` [0, 0xFFFF]
Addition	`f3 = f1 + f2;`	`u3 = u1 + u2;`
	8.8 µS	1 µS
Subtraction	`f3 = f1 - f2;`	`u3 = u1 - u2;`
	7.8 µS	1 µS
Multiplication	`f3 = f1 * f2;`	`u3 = ((uint32_t)u1 * u2) >> 16;`
	10.5 µS	2.6 µS
Division	`f3 = f1 / f2;`	`u3 = u1 / u2;`
	31 µS	14 µS

Unsigned and Signed Integers

Bits	Byte	`uint8_t`	`int8_t`
00000000	0x0	0	0
00100000	0x20	32	32
01000000	0x40	64	64
01100000	0x60	96	96
01111111	0x7F	127	127
10000000	0x80	128	-128
10100000	0xA0	160	-96
11000000	0xC0	192	-64
11100000	0xE0	224	-32
11111111	0xFF	255	-1
		Range [0, 255]	Range [-128, 127]

Unsigned to Signed Conversion

Just subtract (the compiler handles all the two's complement business)

```
uint8_t unsigned = … ;
int8_t signed = unsigned - 0x80;    // - (1 << 7)
```

```
uint16_t unsigned = … ;
int16_t signed = unsigned - 0x8000;    // - (1 << 15)
```

```
uint32_t unsigned = … ;
int32_t signed = unsigned - 0x80000000;    // - (1 << 31)
```

UQ8 Fixed Point

Bits	Byte	`uint8_t`	UQ8
00000000	0x0	0	0
00100000	0x20	32	0.125
01000000	0x40	64	0.25
01100000	0x60	96	0.375
01111111	0x7F	127	0.49609375
10000000	0x80	128	0.5
10100000	0xA0	160	0.625
11000000	0xC0	192	0.75
11100000	0xE0	224	0.875
11111111	0xFF	255	0.99609375

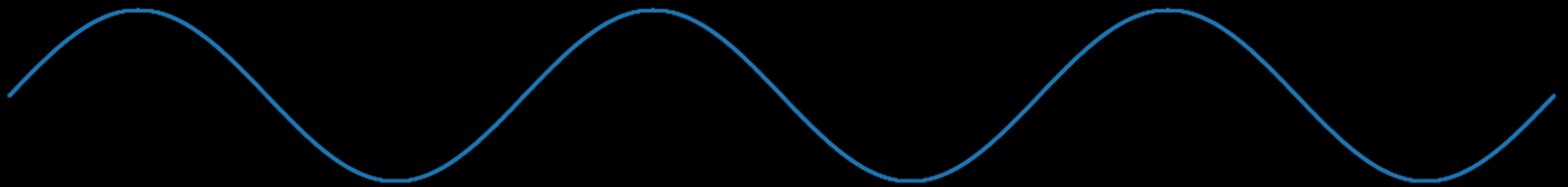
Q8 Fixed Point

Bits	Byte	`int8_t`	Q8
00000000	0x0	0	0
00100000	0x20	32	0.25
01000000	0x40	64	0.5
01100000	0x60	96	0.75
01111111	0x7F	127	0.9921875
10000000	0x80	-128	-1
10100000	0xA0	-96	-0.75
11000000	0xC0	-64	-0.5
11100000	0xE0	-32	-0.25
11111111	0xFF	-1	-0.0078125

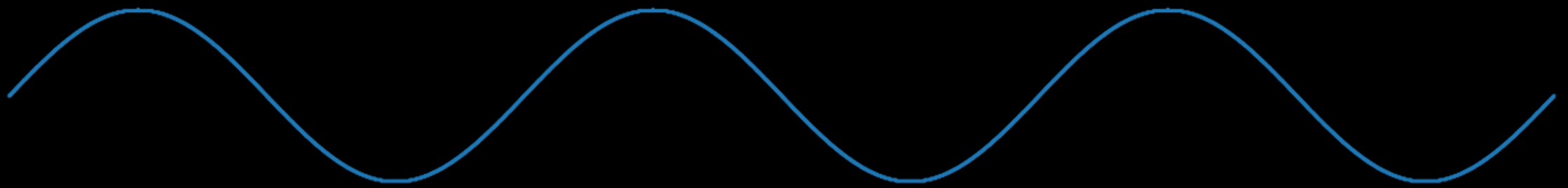
Q8 Fixed Point

	`uint8_t u1, u2, u3;`
Addition	`u3 = u1 + u2;`
Subtraction	`u3 = u1 - u2;`
Multiplication	`u3 = ((uint16_t)u1 * u2) >> 8;`
Division	`u3 = u1 / u2;`

Modular



Saturating



Q8 Fixed Point

```
#include “FixedPoint.h”
```

	`uint8_t u1, u2, u3;`
Addition	`u3 = addsat8(u1, u2);`
Subtraction	`u3 = subsat8(u1, u2);`
Multiplication	`u3 = qmul8(u1, u2);`
Division	`u3 = u1 / u2;`

Addition

	Modular	Saturating
`uint8_t`	0.56 µS	0.8 µS
`uint16_t`	1 µS	1.3 µS
`uint32_t`	1.9 µS	2.4 µS

Subtraction

	Modular	Saturating
`uint8_t`	0.56 µS	0.9 µS
`uint16_t`	1 µS	1.4 µS
`uint32_t`	1.9 µS	2.5 µS

Topics

1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

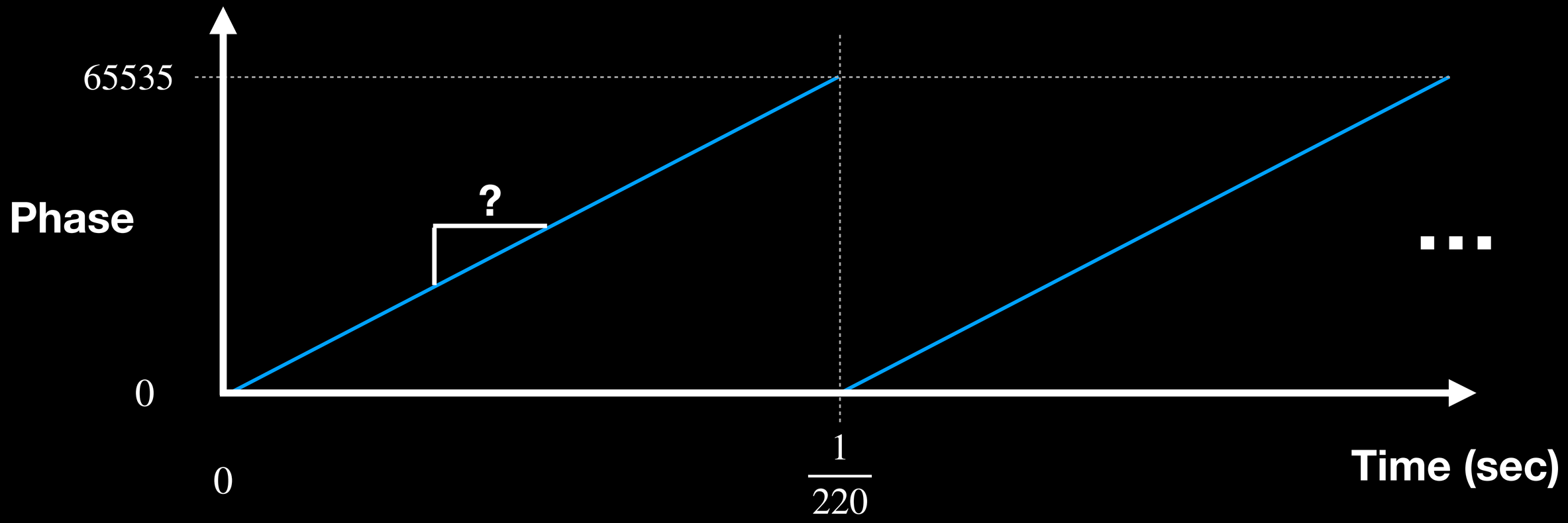
Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

Digital Oscillators: 16-bit Phasor

Using sample rate f_s = 16kHz, and a 16-bit phase accumulator

Synthesize f_0 = 220Hz



Digital Oscillators: Normalized Frequency

Using sample rate f_s = 16kHz, and a 16-bit phase accumulator

Synthesize f_0 = 220Hz

$$\frac{220 \text{ cycles}}{1 \text{ second}} \times \frac{1 \text{ second}}{16000 \text{ samples}} \times \frac{2^{16} \text{ steps}}{1 \text{ cycle}} = \frac{901.12 \text{ steps}}{1 \text{ sample}}$$

Range [0, 2^{16}-1]

Generalized: $$f_0 \times \frac{1}{f_s} \times 2^R$$

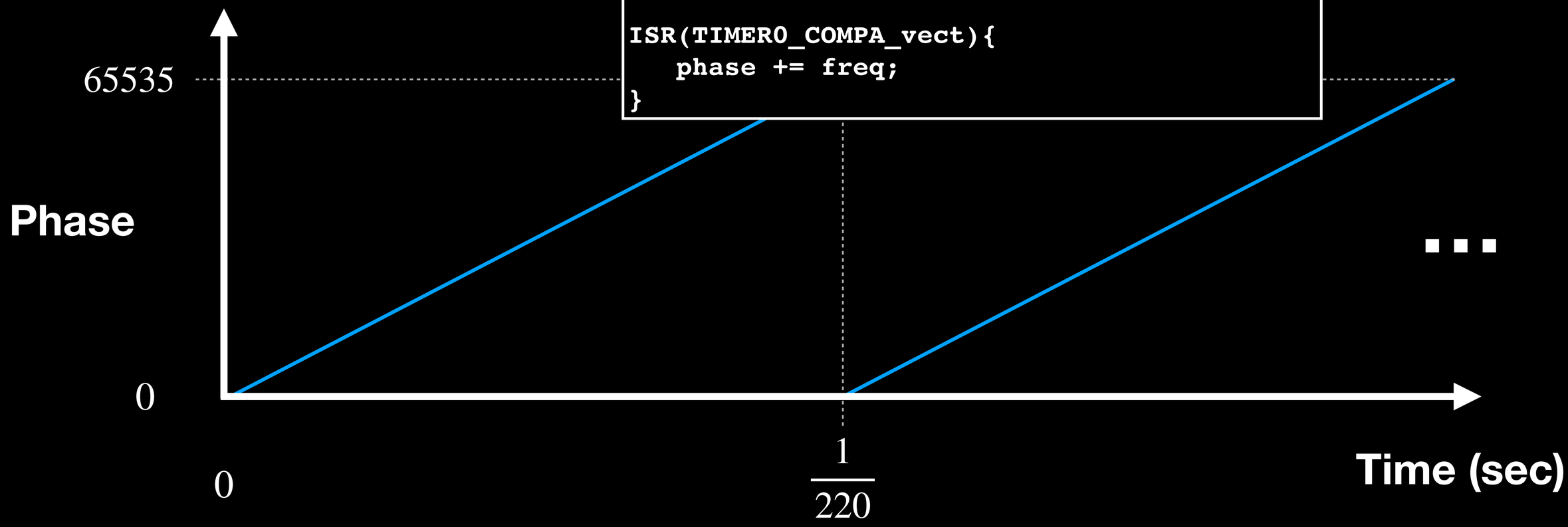
Range [0, 2^R-1]

Digital Oscillators: 16-bit Phasor

Using sample rate f_s = 16kHz, and a 16-bit phase accumulator

Synthesize f_0 = 220Hz

```
float fs = 16e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 0xFFFF;

ISR(TIMER0_COMPA_vect){
   phase += freq;
}
```

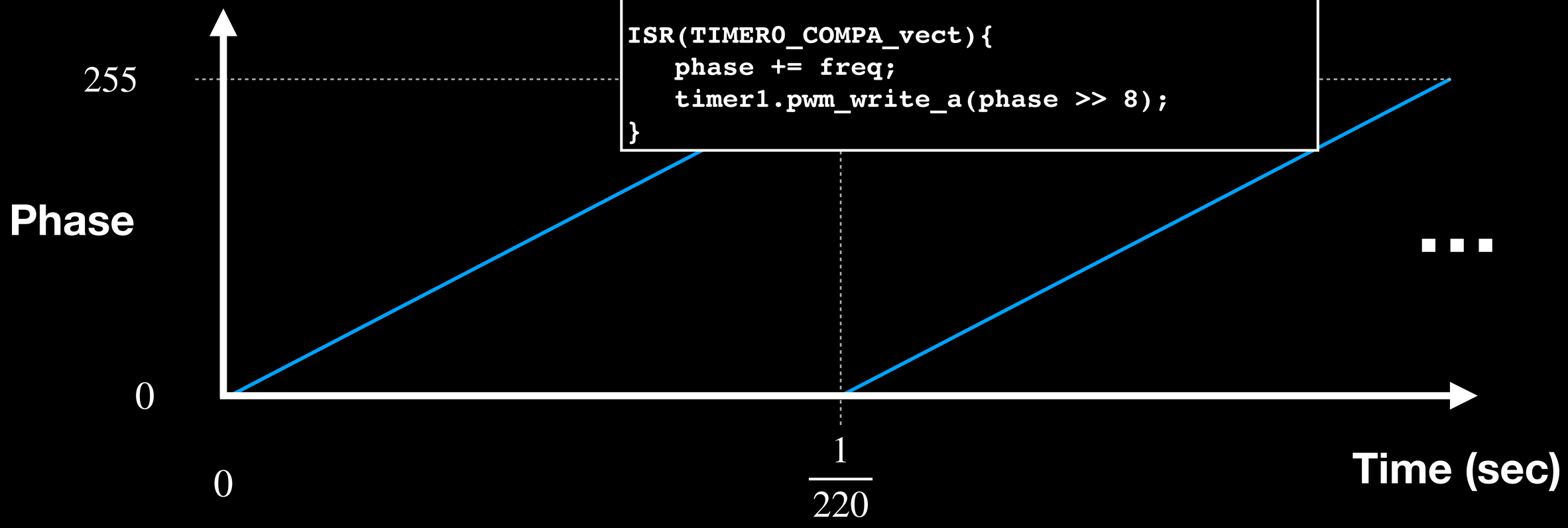


Digital Oscillators: 16-bit Phasor

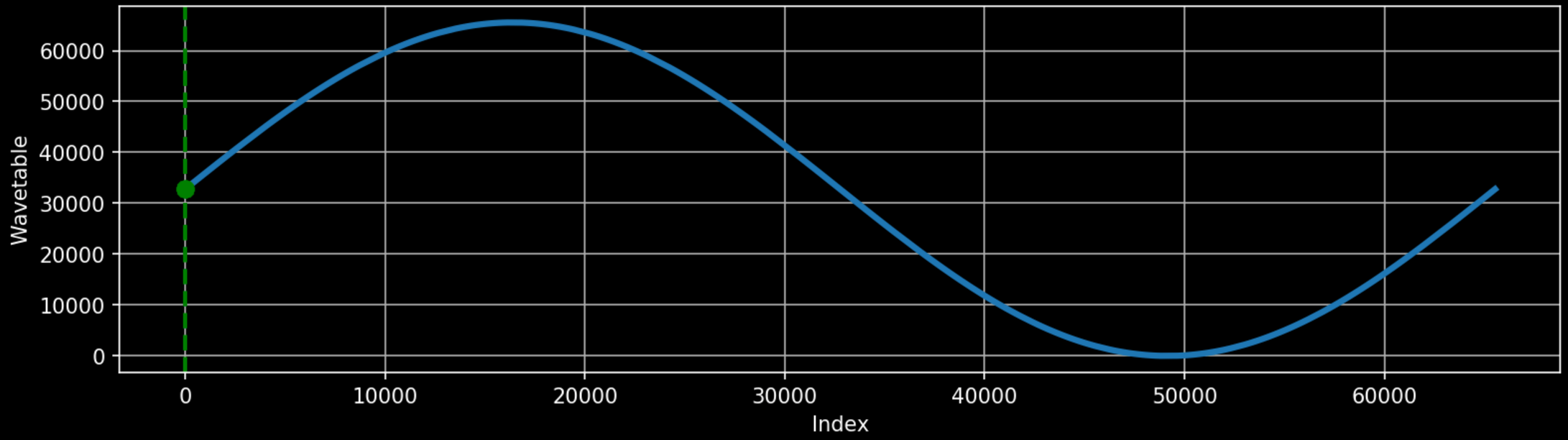
8-bit DAC and 16-bit Signal?

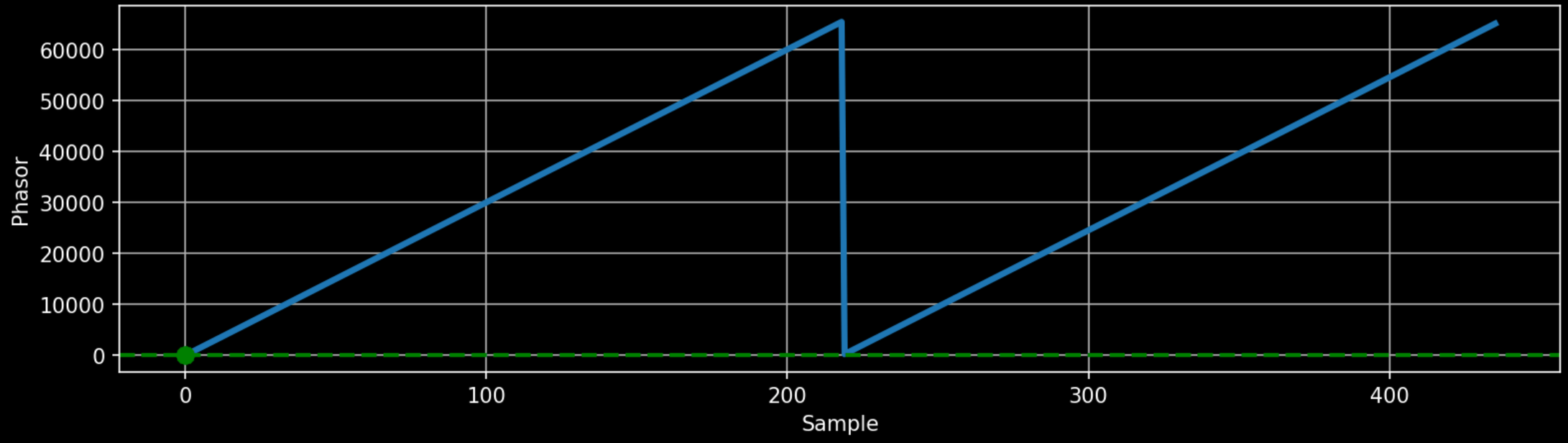
Don’t divide; bit shift!

```
float fs = 16e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 0xFFFF;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   timer1.pwm_write_a(phase >> 8);
}
```



Digital Oscillators: Wavetable





Digital Oscillators: Wavetable

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

$$x[n] = table(n \cdot (2^{16} - 1) \cdot \frac{f_0}{f_s})$$

Floating Point

```
float fs = 16e3;
float phase = 0;
float freq = 220.0/fs * 2*M_PI;
float sine_sample;

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

Fixed Point

```
float fs = 16e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 65535;
uint16_t wavetable[65536];
uint16_t sine_sample;

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

$132kB$ 👎

$\gg 2kB$

(ATmega328 Memory)

Digital Oscillators: Wavetable

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

$$x[n] = table(n \cdot (2^9 - 1) \cdot \frac{f_0}{f_s})$$

Floating Point

```
float fs = 16e3;
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 = 16e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 511;
uint16_t wavetable[512];
uint16_t sine_sample;

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

$1kB$

$< 2kB$

(ATmega328 Memory)

Digital Oscillators: Wavetable

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

$$x[n] = table(n \cdot (2^9 - 1) \cdot \frac{f_0}{f_s})$$

Floating Point

```
float fs = 16e3;
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 = 16e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 511;
uint16_t wavetable[512];
uint16_t sine_sample;

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

9-bit Freq Resolution

Digital Oscillators: Wavetable

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

$$x[n] = table(n \cdot \frac{2^{16} - 1}{2^7} \cdot \frac{f_0}{f_s})$$

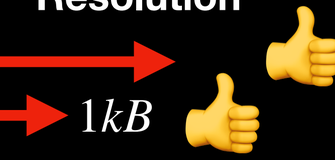
Floating Point

```
float fs = 16e3;
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 = 16e3;
uint16_t phase = 0;
int16_t freq = 220.0/fs * 65535;
uint16_t wavetable[512];
uint16_t sine_sample;

ISR(TIMER0_COMPA_vect){
   phase += freq;
   sine_sample = wavetable[phase >> 7];
}
```

16-bit Freq Resolution



$1kB$

$< 2kB$

(ATmega328 Memory)

Digital Oscillators: Sine Wave Table

Initialization in SRAM (2kB Max)

```
#define TAB_LEN 512
uint16_t wavetable[TAB_LEN];

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

Hard-coding in PROGMEM (32kB Max)

```
#include <arv/pgmspace.h>

const uint16_t wavetable[] PROGMEM = {
    0x8000, 0x8032, 0x8064, 0x8096, 0x80c9, 0x80fb, 0x812d, 0x815f,
    0x8192, 0x81c4, 0x81f6, 0x8229, 0x825b, 0x828d, 0x82bf, 0x82f2,

                                    ⋮

    0x7d0d, 0x7d40, 0x7d72, 0x7da4, 0x7dd6, 0x7e09, 0x7e3b, 0x7e6d,
    0x7ea0, 0x7ed2, 0x7f04, 0x7f36, 0x7f69, 0x7f9b, 0x7fcd, 0x7fff
};
```

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

$$x[n] = table(n \cdot (2^{16} - 1) \cdot \frac{f_0}{f_s})$$

Normalize!
(or don't)

Initialization in SRAM

```
#define TAB_LEN 512
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;
    }
}
```

Normalized Exponential Frequency Table

Table range for fixed max/min ratios

$$[0.001,\ 1.0]\frac{cyc}{samp} \times \frac{0xFFFF\ steps}{1\ cyc}$$

Set maximum frequency

$$\frac{200\ cyc}{1\ sec} \times \frac{1\ sec}{fs\ samp} \times \frac{0xFFFF\ steps}{1\ cyc}$$

Scaling the result using qmul16()

```
#include “FixedPoint.h”
#include “tables/exp16x10x1000.h”

uint16_t scale = 200.0f/fs * 0xFFFF;                          // Max frequency
uint16_t freq = qmul16(scale * exp16x10_1000[analogRead(A0)]); // Lookup and scale
```

Normalized Exponential Frequency Table

Hard-coding in PROGMEM (32kB Max)

```
#include <arv/pgmspace.h>

const uint16_t exp16x10_1000[] PROGMEM = {
    0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0044, 0x0044, 0x0045,
    0x0045, 0x0046, 0x0046, 0x0047, 0x0047, 0x0048, 0x0048, 0x0049,

                                    ⋮

    0xe756, 0xe8e8, 0xea7b, 0xec12, 0xedac, 0xef48, 0xf0e7, 0xf289,
    0xf42d, 0xf5d5, 0xf77f, 0xf92d, 0xfadd, 0xfc90, 0xfe46, 0xffff
};
```

Scaling the result (using ParamTable16)

```
#include “ParamTable.h”
#include “tables/exp16x10x1000.h”

ParamTable16 freq_table(exp16x10_1000, 200.0f/fs * 0xFFFF);
uint16_t freq = freq_table.lookup(analogRead(A0));
```

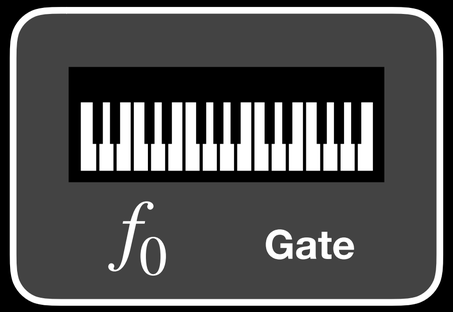
Topics

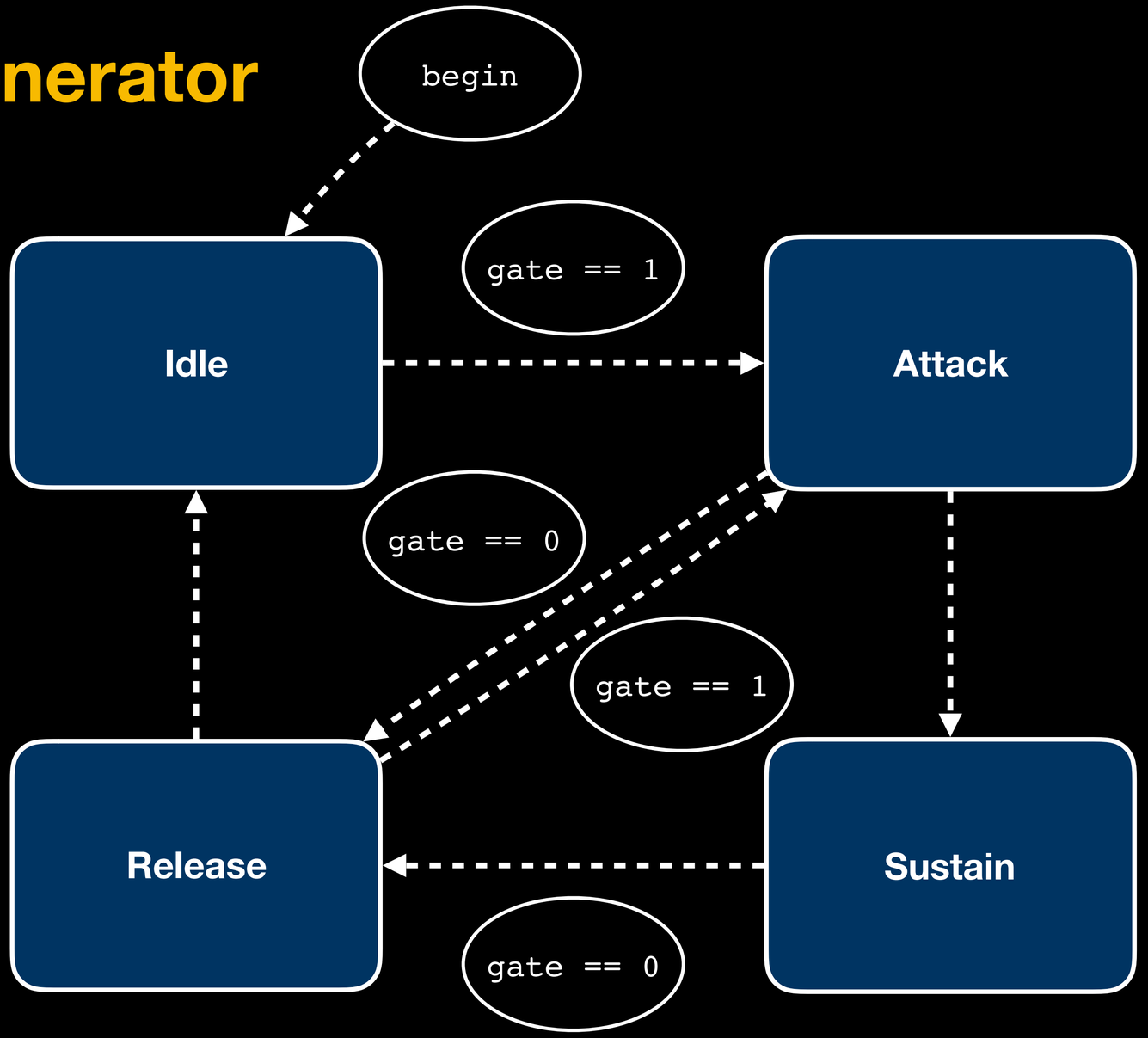
1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

Examples

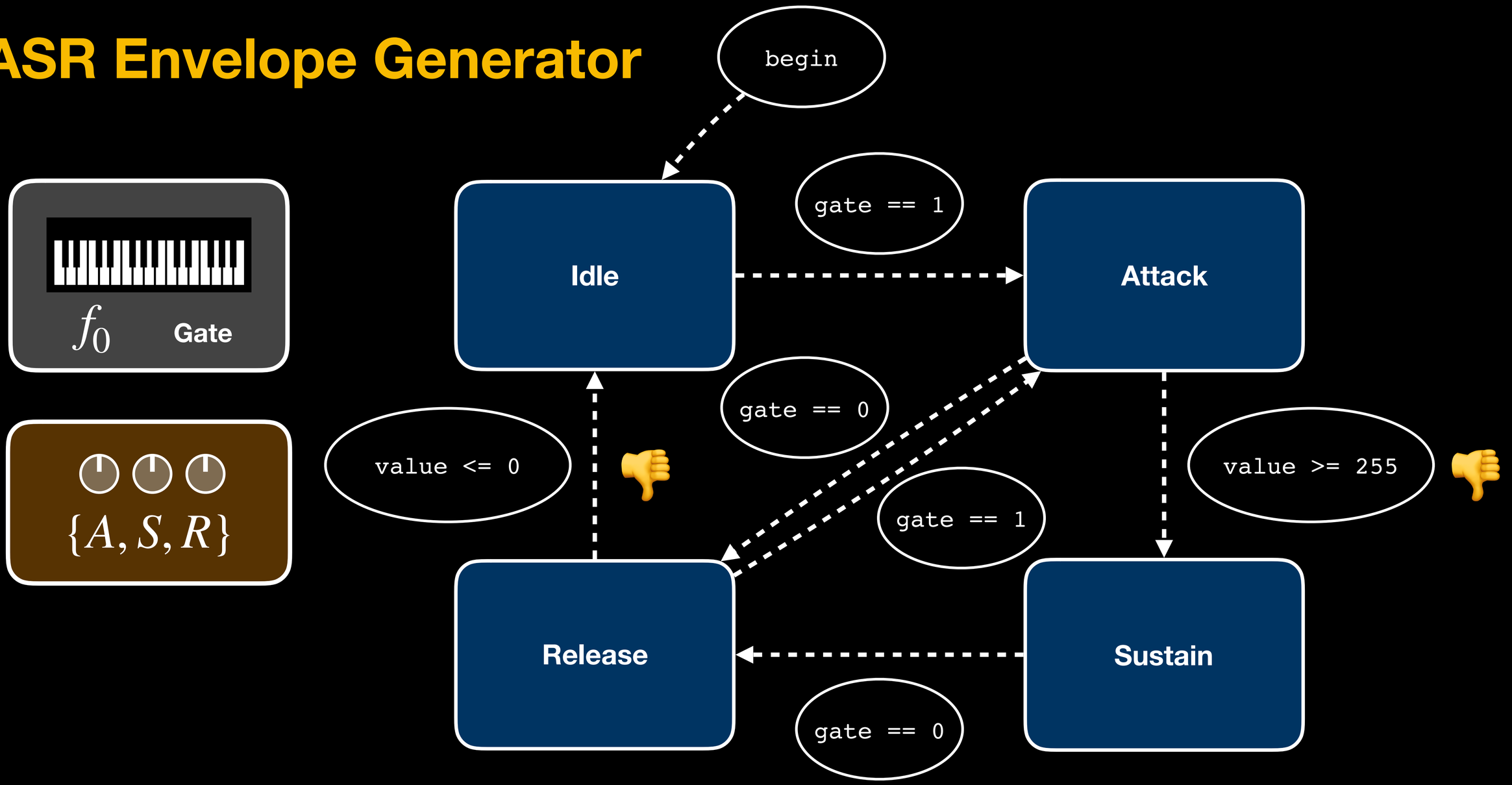
1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

ASR Envelope Generator

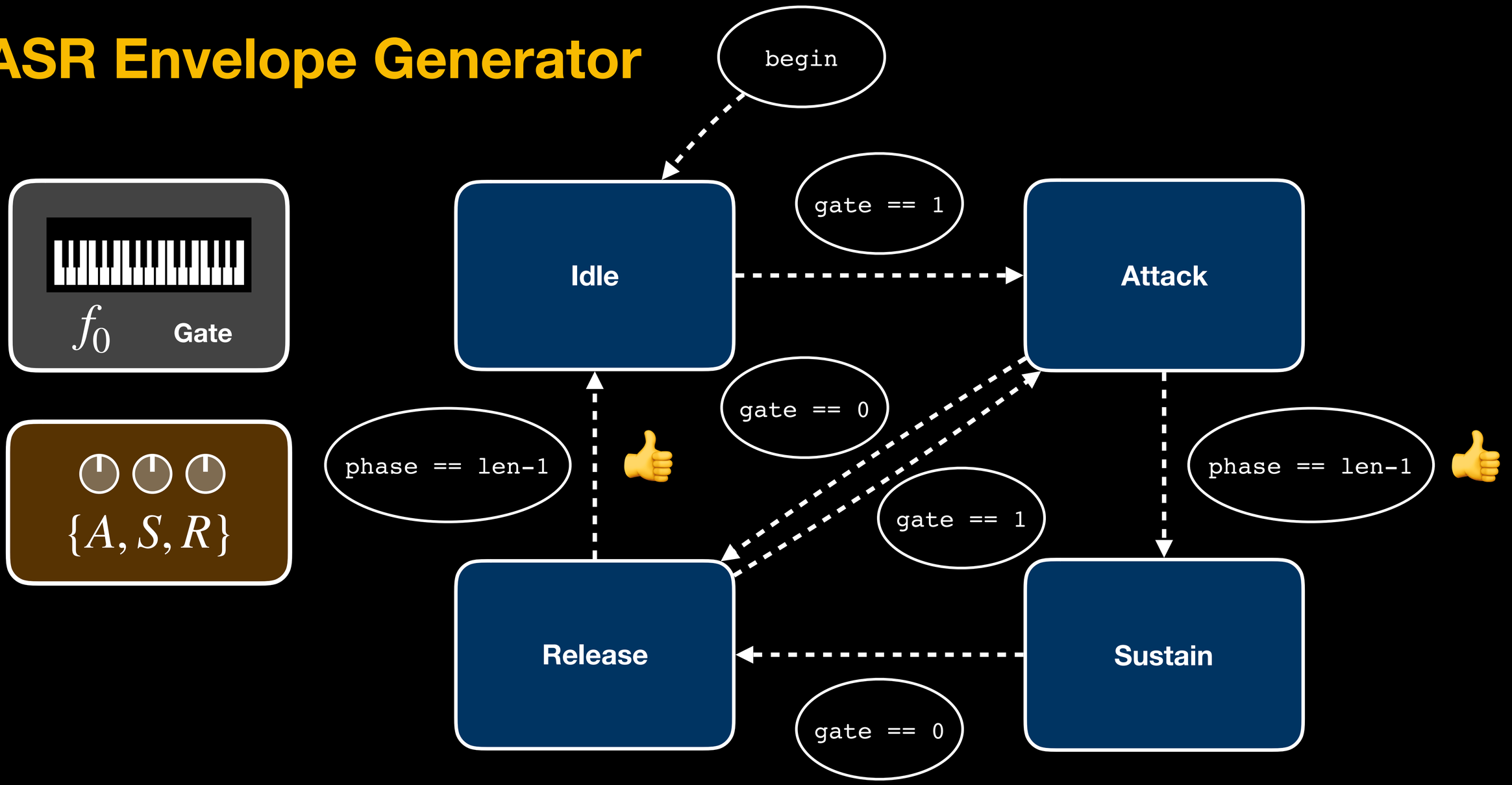




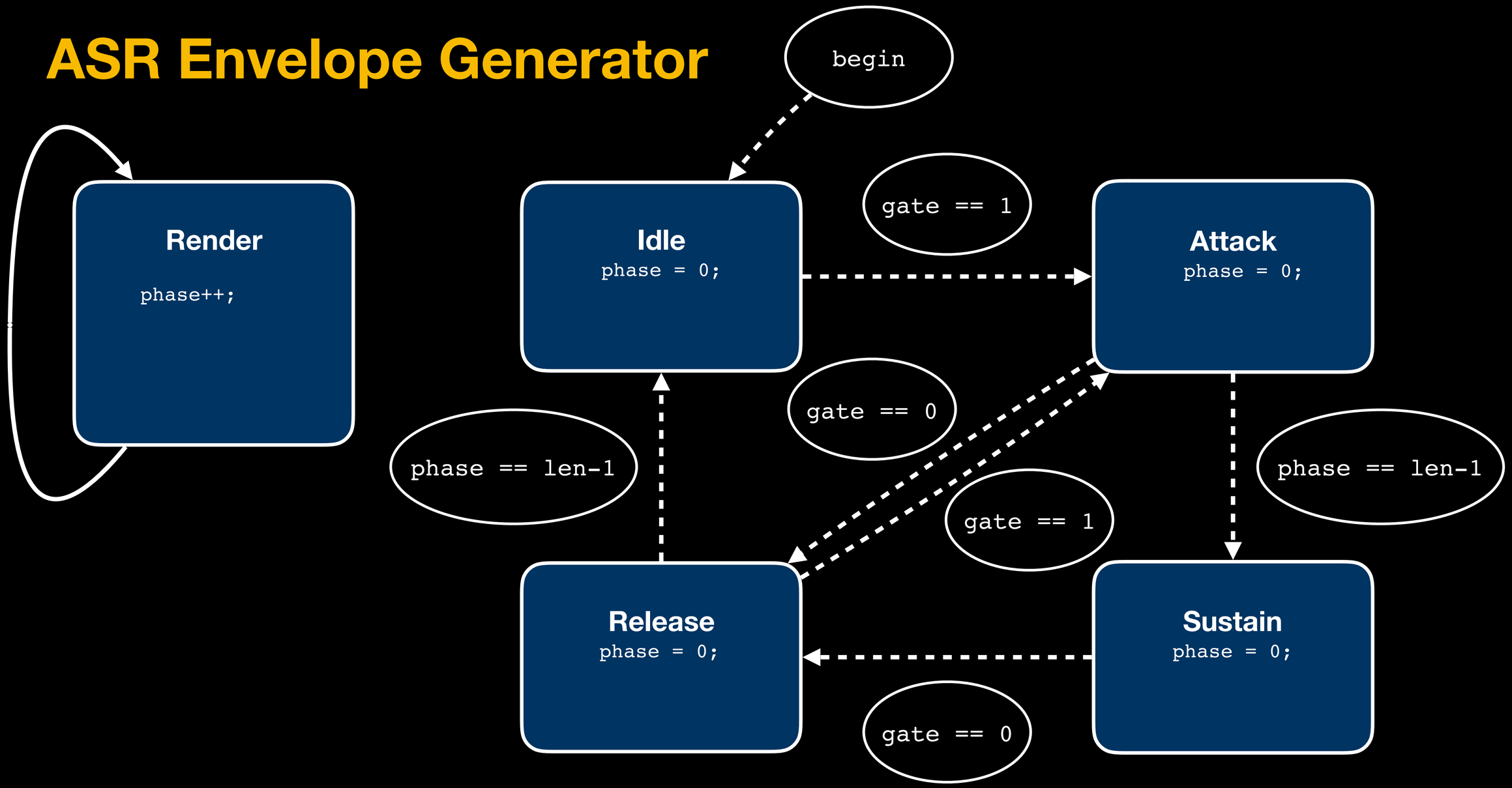
ASR Envelope Generator



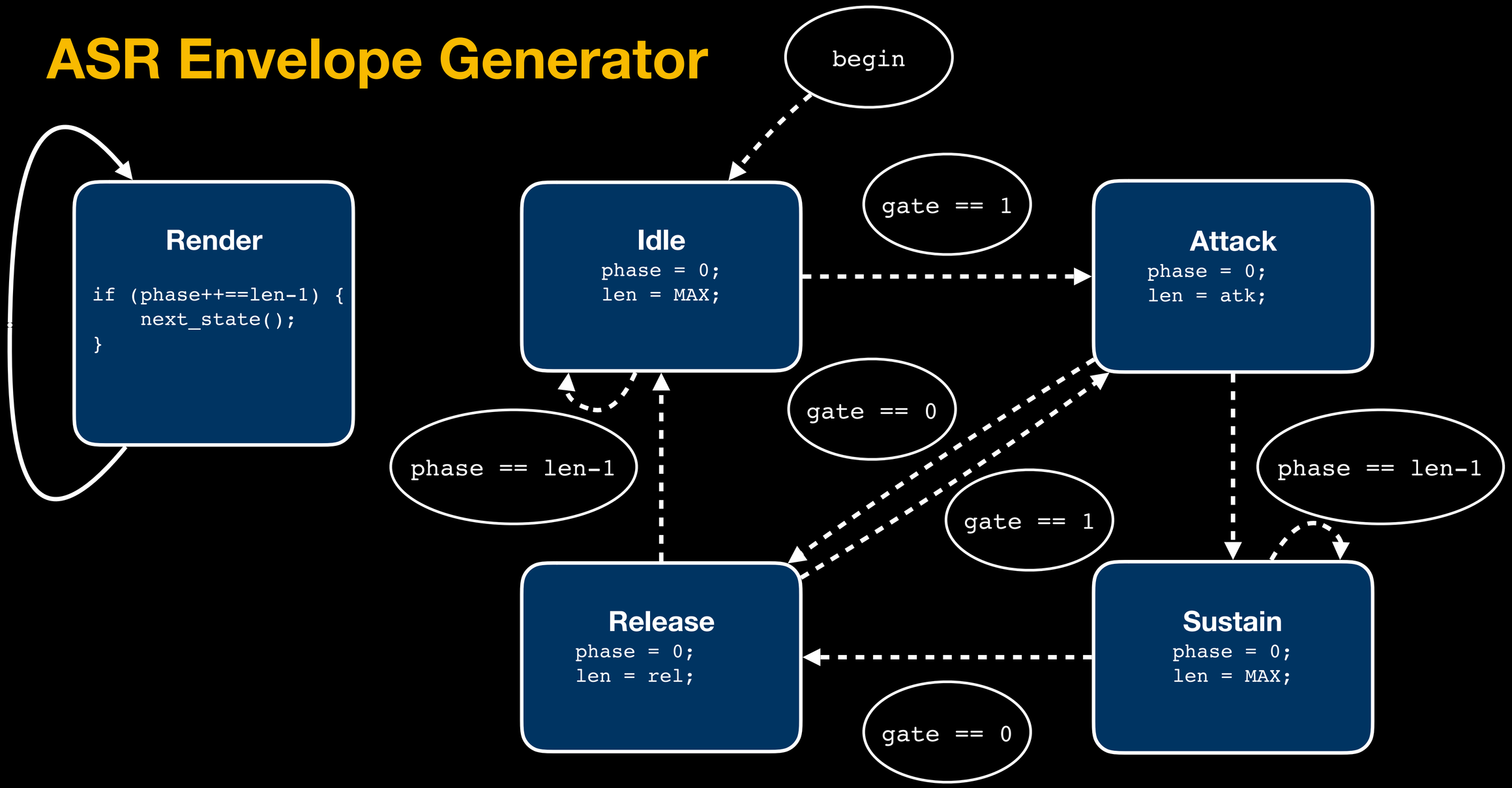
ASR Envelope Generator



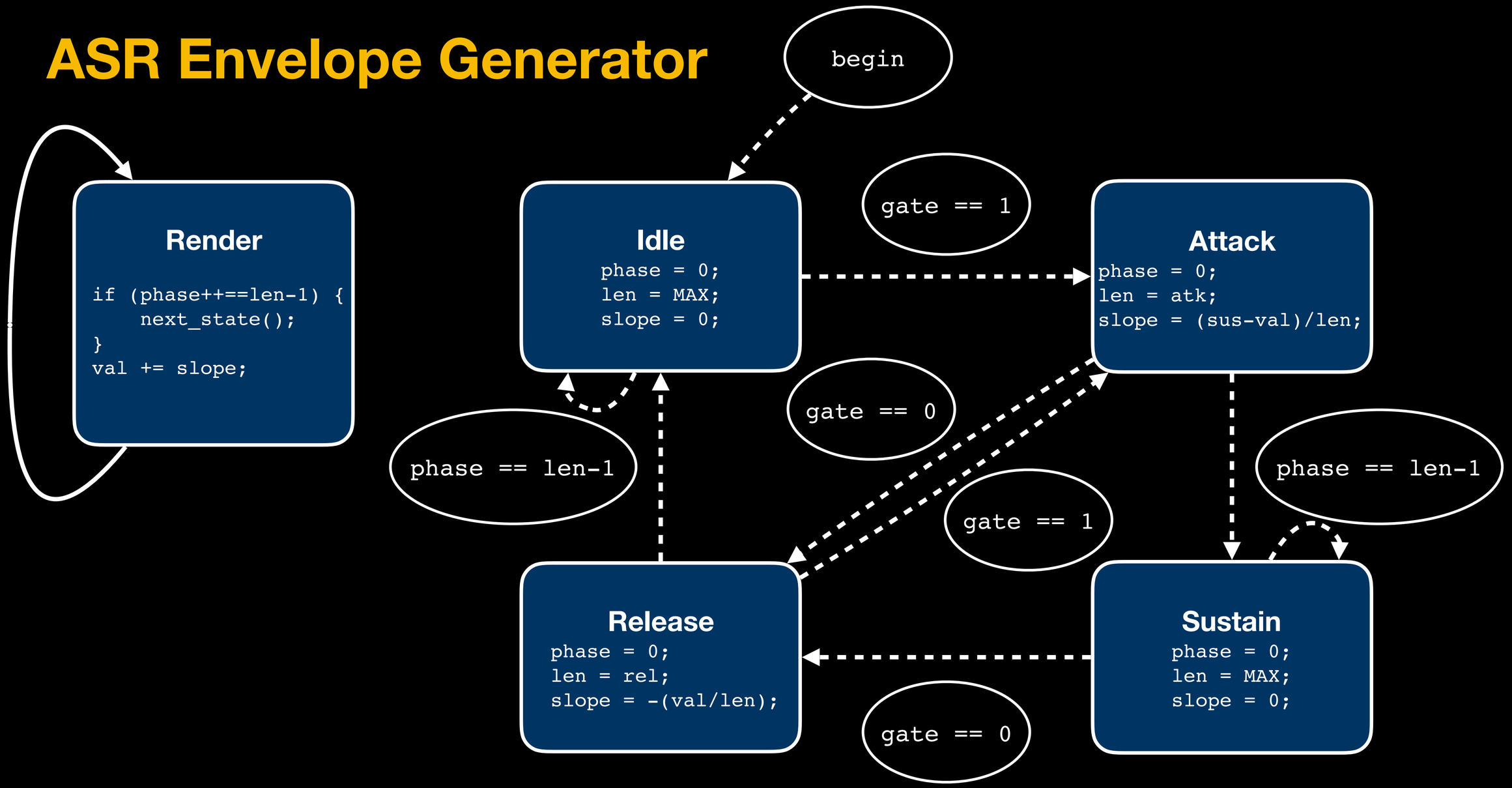
ASR Envelope Generator



ASR Envelope Generator



ASR Envelope Generator



Topics

1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

Moving Average Filter

$$y[n] = \frac{x[n] + x[n-1] + x[n-2] + x[n-3]}{4}$$

n	x[n]	y[n]
0	0	0
1	0	0
2	0	0
3	128	32
4	128	64
5	128	96
6	128	128
7	128	128
8	128	128

Moving Average Filter

$$y[n] = \frac{x[n] + x[n-1] + x[n-2] + x[n-3]}{4}$$

```
#define BUF_LEN 4

struct CircularBuffer {

  uint8_t idx = 0;
  uint16_t buf[BUF_LEN];

  void process(uint16_t sample) {
    buf[idx++] = sample;
    if (idx == BUF_LEN)
      idx = 0;
  }
}
```

```
#define BUF_LEN 4

struct MovingMean {

  uint8_t idx = 0;
  uint16_t buf[BUF_LEN];
  uint16_t val = 0;

  uint16_t process(uint16_t sample) {
    val -= buf[idx] / BUF_LEN;
    buf[idx++] = sample;
    val += sample / BUF_LEN;
    if (idx == BUF_LEN)
      idx = 0;
  }
}
```

Moving Average Filter

$$y[n] = \frac{x[n] + x[n-1] + x[n-2] + x[n-3]}{4}$$

```
#define BUF_LEN 4

struct MovingMean {

  uint8_t idx = 0;
  uint16_t buf[BUF_LEN];
  uint16_t val = 0;

  uint16_t process(uint16_t sample) {
    val -= buf[idx] / BUF_LEN;
    buf[idx++] = sample;
    val += sample / BUF_LEN;
    if (idx == BUF_LEN)
      idx = 0;
  }
}
```

```
#define BUF_B 2

struct MovingMean {

  uint8_t idx = 0;
  uint16_t buf[(1 << BUF_B)];
  uint16_t val = 0;

  uint16_t process(uint16_t sample) {
    val -= buf[idx] >> BUF_B;
    buf[idx++] = sample;
    val += sample >> BUF_B;
    if (idx == (1 << BUF_B))
      idx = 0;
  }
}
```

Power of two length replaces division with bit shifting

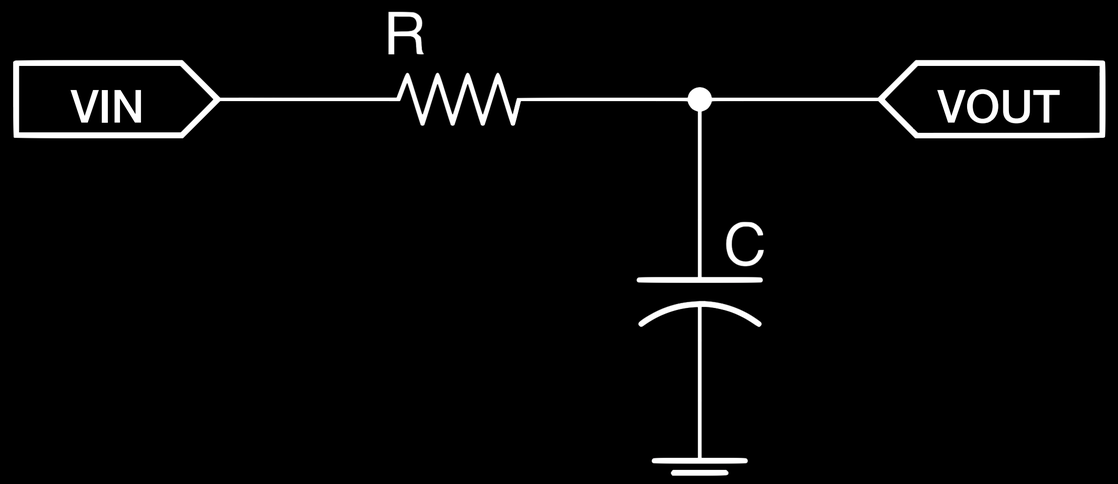
Exponential Moving Average Filter

$$y[n] = \alpha x[n] + (1 - \alpha) y[n - 1]$$

New Input Weight

Previous Output "Inertia"

$$\alpha \in (0, 1)$$



$$\alpha = \frac{2\pi \frac{f_c}{f_s}}{2\pi \frac{f_c}{f_s} + 1}$$

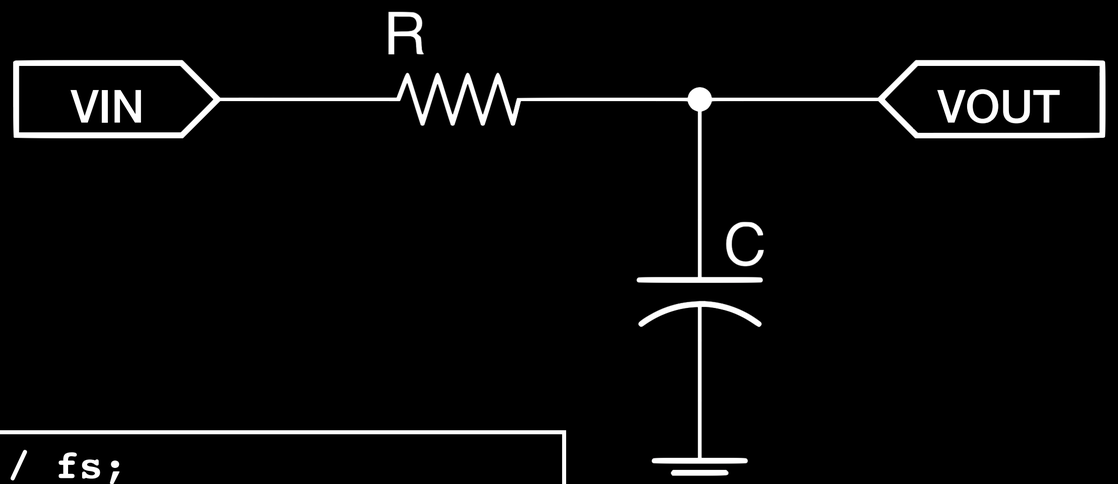
https://en.wikipedia.org/wiki/Low-pass_filter#Discrete-time_realization

Exponential Moving Average Filter

$$y[n] = \alpha x[n] + (1 - \alpha) y[n-1]$$

$$y[n] = y[n-1] - \alpha(x[n] - y[n-1])$$

$$\alpha = \frac{2\pi\frac{f_c}{f_s}}{2\pi\frac{f_c}{f_s} + 1}$$




```
float freq = TWO_PI * 200 / fs;
uint32_t coeff = 0xFFFF * freq / (freq + 1);
uint16_t val = 0;

uint16_t process(uint16_t sample) {
    if (sample > val)
      val += coeff * (sample - val) >> 16;
    else
      val -= coeff * (val - sample) >> 16;
    return val;
}
```

https://en.wikipedia.org/wiki/Low-pass_filter#Discrete-time_realization

Normalized Filter Coefficient Tables

Hard-coding in PROGMEM (32kB Max)

```
#include <arv/pgmspace.h>

const uint16_t coeff16x10_1000[] PROGMEM = {
    0x0199, 0x019c, 0x019f, 0x01a2, 0x01a4, 0x01a7, 0x01aa, 0x01ad,
    0x01b0, 0x01b3, 0x01b6, 0x01b9, 0x01bc, 0x01be, 0x01c1, 0x01c5,

                                    ⋮

    0xd9a9, 0xd9e2, 0xda1a, 0xda51, 0xda89, 0xdac0, 0xdaf7, 0xdb2d,
    0xdb64, 0xdb9a, 0xdbd0, 0xdc05, 0xdc3a, 0xdc70, 0xdca4, 0xdcd9
};
```

Scaling the result (using ParamTable16)

```
#include “ParamTable.h”
#include “tables/coeff16x10x1000.h”

float max_freq = TWO_PI * 200 / fs;
ParamTable16 coeff_table(coeff16x10_1000, 0xFFFF * max_freq / (max_freq + 1));
uint16_t coeff = coeff_table.lookup(analogRead(A0));
```

Topics

1. Digital I/O
2. Analog Output
3. Timing Control
4. Analog Input
5. Fixed Point

Examples

1. Sinusoidal LFO
2. Envelope Generator
3. Low Pass Filter
4. MIDI to CV/Gate

MIDI DIN-5 Input Circuit

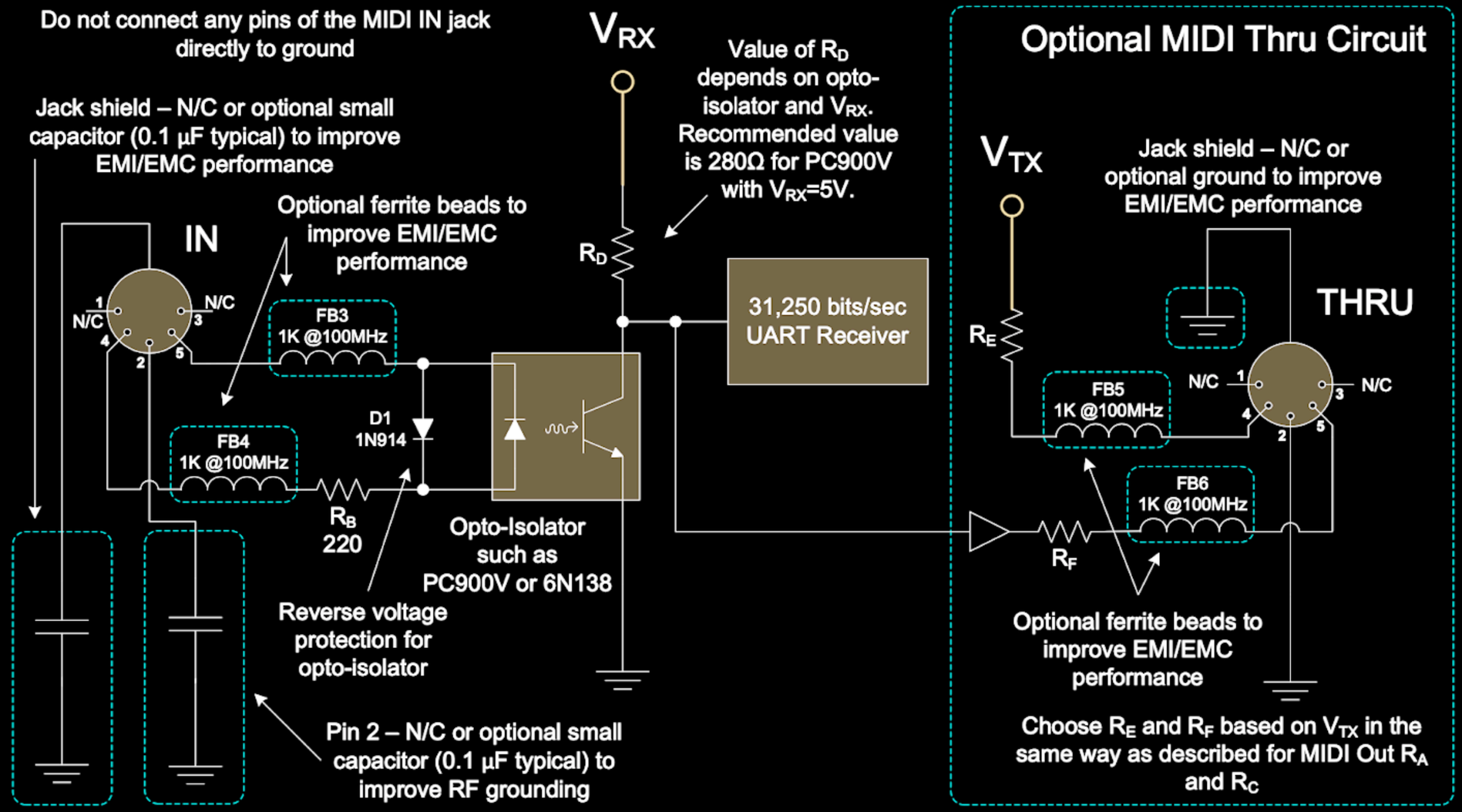
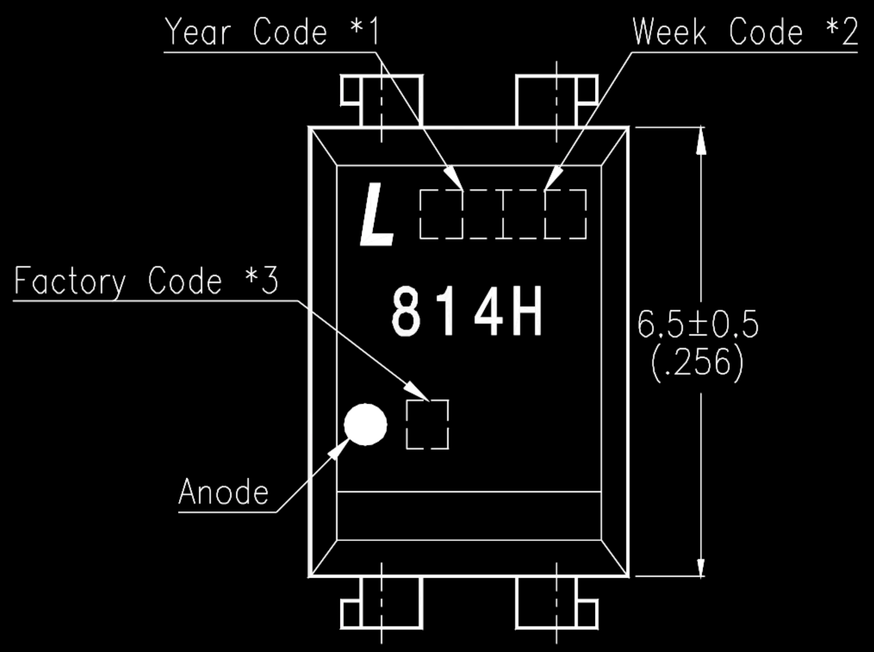


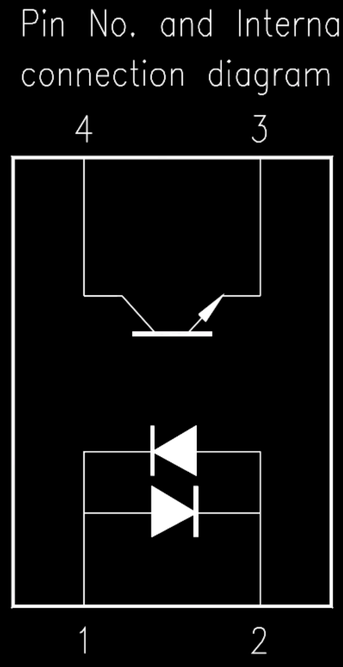
Figure 2 – MIDI IN and THRU circuit

https://www.midi.org/specifications/midi-transports-specifications/5-pin-din-electrical-specs

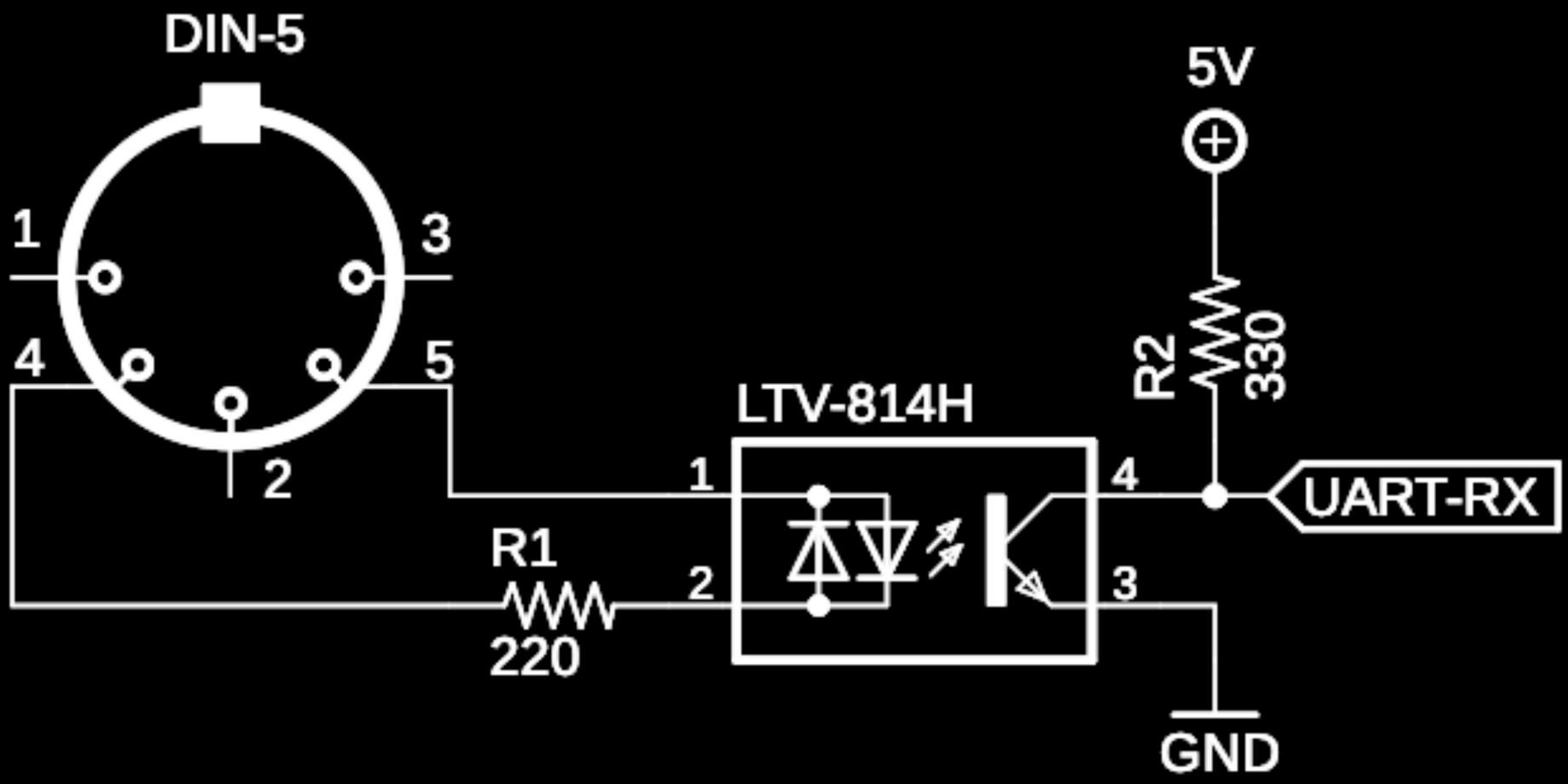
MIDI DIN-5 Input Circuit

LTV-814H Transistor Output Optocoupler





MIDI DIN-5 Input Circuit



MIDI to CV Conversion

$$\frac{256 \text{ steps}}{5 \cancel{\text{ volts}}} \cdot \frac{1 \cancel{\text{ volt}}}{1 \cancel{\text{ oct}}} \cdot \frac{1 \cancel{\text{ oct}}}{12 \text{ semitones}} = 4.27 \frac{\text{steps}}{\text{semitone}}$$

Bit Resolution	steps/semitone
8-bit	4.27
9-bit	8.53
10-bit	17.07
11-bit	34.13
12-bit	68.27
13-bit	136.53
14-bit	273.07
15-bit	546.13
16-bit	1092.27

MIDI to CV Conversion, 12-bit Resolution

$$\frac{4096 \text{ steps}}{5 \text{ volts}} \cdot \frac{1 \text{ volt}}{1 \text{ oct}} \cdot \frac{1 \text{ oct}}{12 \text{ semitones}} = 68.27 \frac{\text{steps}}{\text{semitone}}$$

Floating Point

```
uint8_t note = 48;
float scale = 68.27;

void note_on() {
   uint16_t out;
   out = roundf(note * scale);
   timer1.pwm_write_a(out);
}
```

Fixed Point

```
uint8_t note = 48;
uint32_t scale = 6827;

void note_on() {
   uint16_t out;
   out = (note * scale) / 100;
   timer1.pwm_write_a(out);
}
```

https://github.com/sparkfun/MIDI_Shield/blob/master/Firmware/MIDI-CV/MIDI-CV.ino

MIDI to CV Conversion, 12-bit Resolution

$$\frac{4096 \text{ steps}}{5 \text{ volts}} \cdot \frac{1 \text{ volt}}{1 \text{ oct}} \cdot \frac{1 \text{ oct}}{12 \text{ semitones}} = 68.27 \frac{\text{steps}}{\text{semitone}}$$

Fixed Point w/ Division

```
uint8_t note = 48;
uint32_t scale = 6827;

void note_on() {
   uint16_t out;
   out = (note * scale) / 100;
   timer1.pwm_write_a(out);
}
```

https://github.com/sparkfun/MIDI_Shield/blob/master/Firmware/MIDI-CV/MIDI-CV.ino

Fixed Point w/o Division

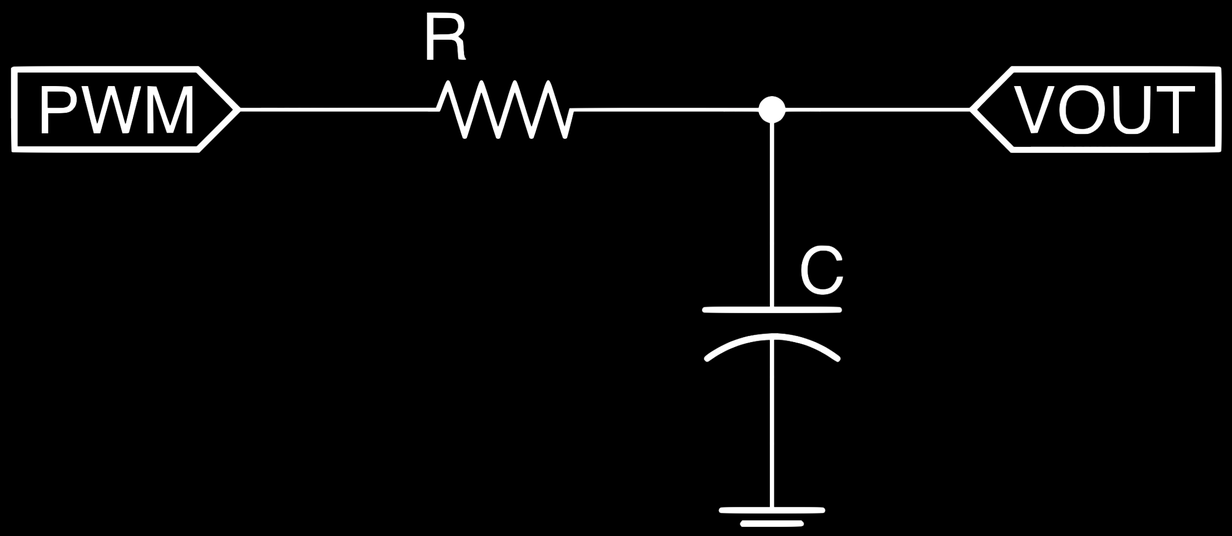
```
uint8_t note = 48;
uint32_t scale = 68.27 * 128;

void note_on() {
   uint16_t out;
   out = (note * scale) >> 7;
   timer1.pwm_write_a(out);
}
```

Appendix

1. Reconstruction Filters

Reconstruction Filters

First Order Low-Pass, Passive:

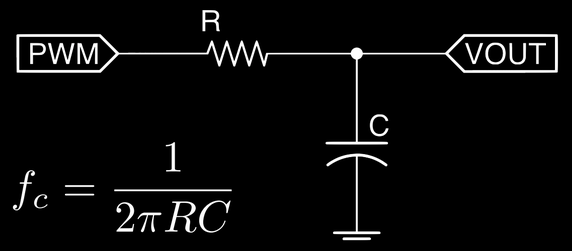


$$f_c = \frac{1}{2\pi RC}$$

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:

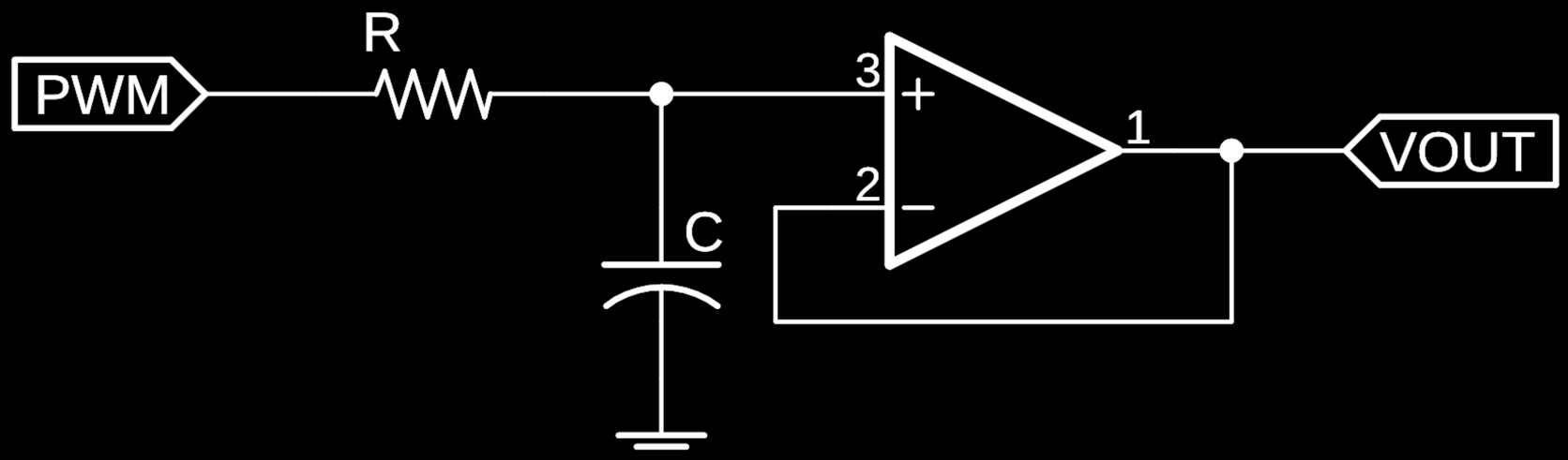
$$250Hz = \frac{1}{2\pi R \cdot 0.1\mu F}$$

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

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

Reconstruction Filters

1st Order Low-Pass, Active:

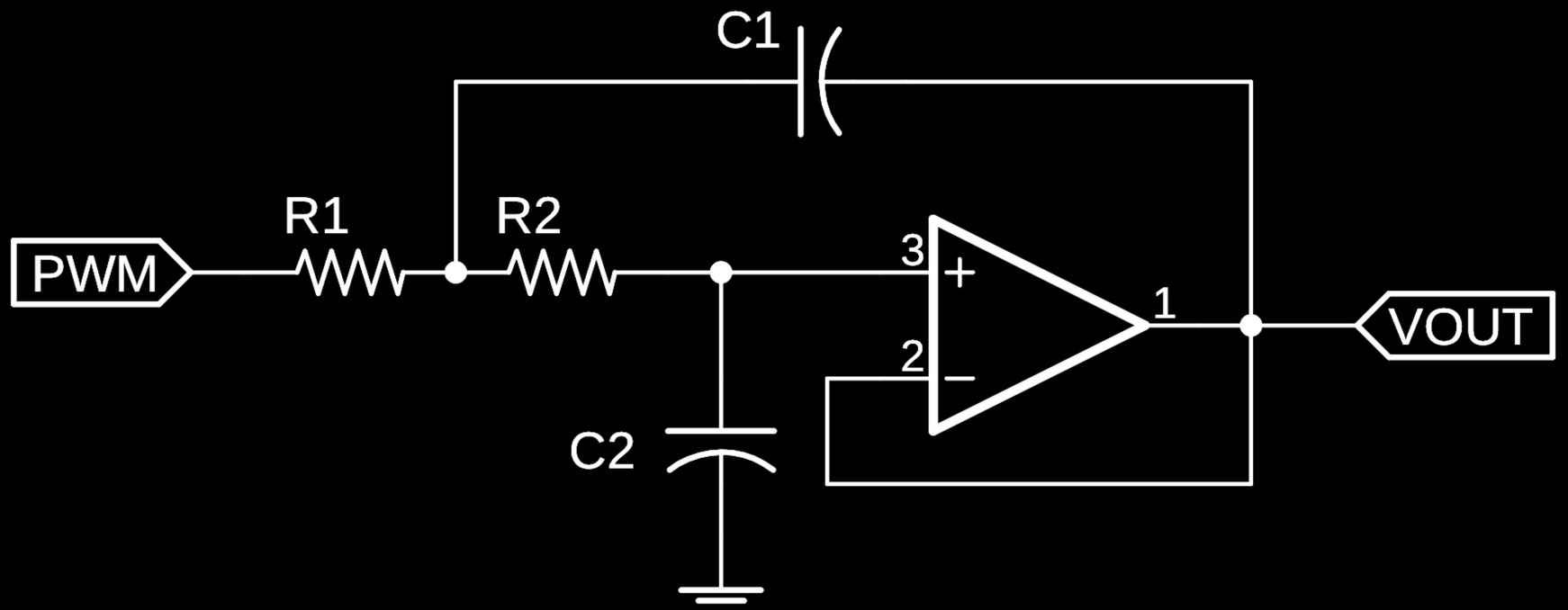


Caveat:
Use a ‘Rail-to-Rail’ Op Amp!
TLV2372

Slope = -6dB/Octave

Reconstruction Filters

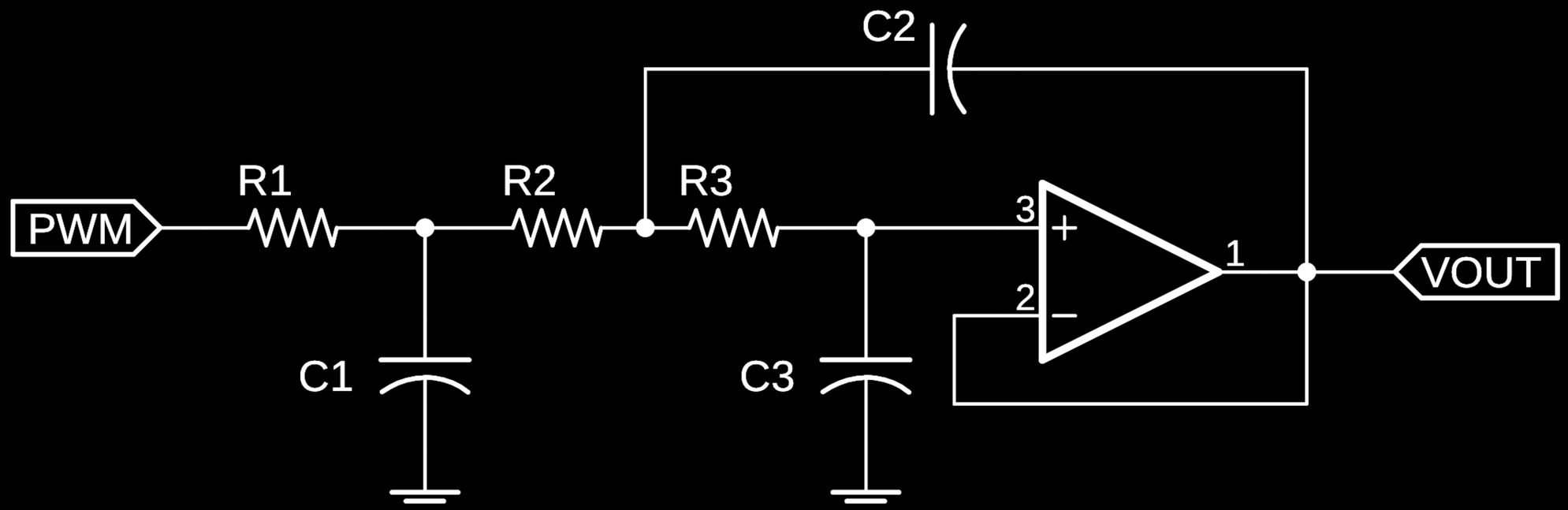
2nd Order Low-Pass, Active (Sallen-Key):



Slope = -12dB/Octave

Reconstruction Filters

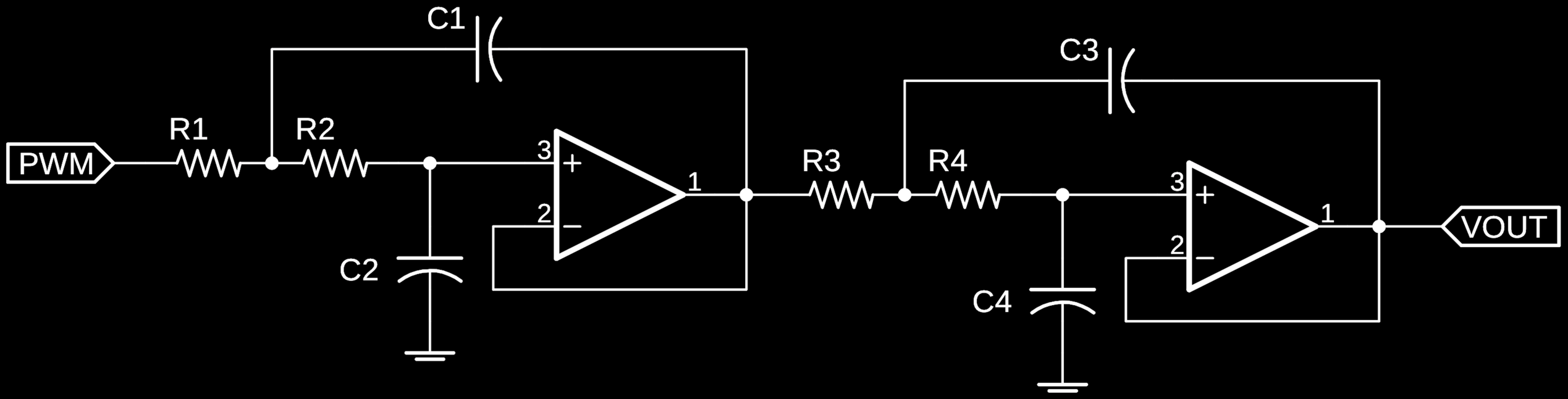
3rd Order Low-Pass, Active (Sallen-Key):



Slope = -18dB/Octave

Reconstruction Filters

4th Order Low-Pass, Active (Sallen-Key):



Slope = -24dB/Octave

Reconstruction Filters

- Determine desired attenuation at Nyquist
- Select filter cutoff f_c frequency and order

Frequency	Attenuation (dB)				
	1st Order (-6dB/oct)	2nd Order (-12dB/oct)	3rd Order (-18dB/oct)	4th Order (-24dB/oct)	5th Order (-30dB/oct)
fc	-3	-3	-3	-3	-3
2fc	-9	-15	-21	-27	-33
4fc	-15	-27	-39	-51	-63
8fc	-21	-39	-57	-75	-93
16fc	-27	-51	-75	-99	-123
32fc	-33	-63	-93	-123	-153
64fc	-39	-75	-111	-147	-183

Reconstruction Filters

- 2nd order design tool
  - http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm
  - Specify f_c, and use Q = 0.707 for a flat passband (Butterworth filter)
- 3rd order design tool
  - http://sim.okawa-denshi.jp/en/Sallenkey3Lowkeisan.htm
- 4th order design
  - Cascade two 2nd order sections
  - Both with the same f_c
  - First with Q = 0.54119610, second with Q = 1.3065630

Reconstruction Filters

- Nth order design:
  - N even?
    - Cascade $\frac{N}{2}$ 2nd order sections
  - N odd?
    - Cascade 1st order passive with $\frac{N}{2} - 1$ 2nd order sections
  - Same f_c for each stage
  - Determine Q values: Ear Level Engineering - Cascading Filters