Fully-Synchronized Synthesizer (FSS)

A prototype demonstrating the future of synthesizer interfacing.

Jacob Peterson

Brady Hartog

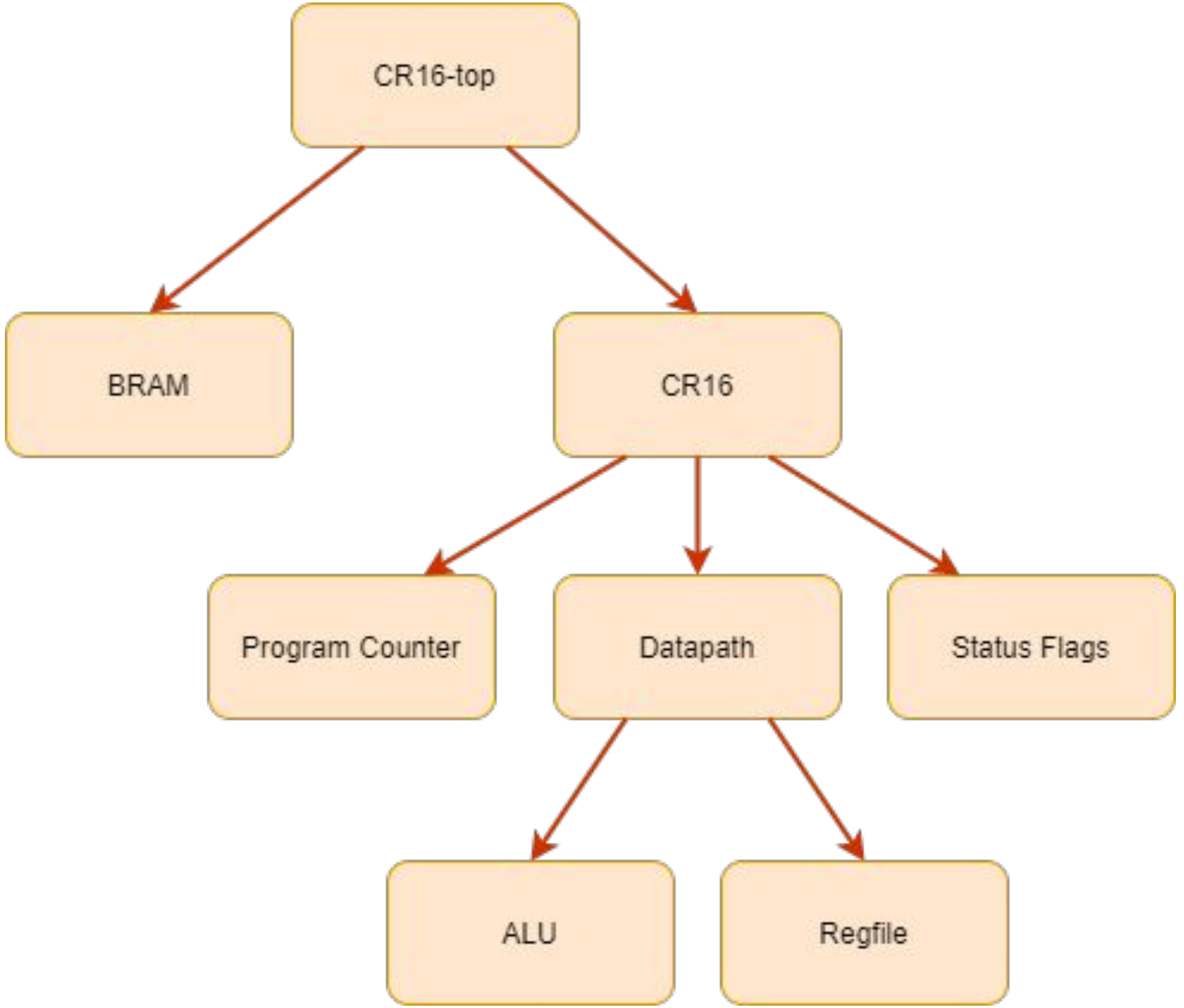
Isabella Gilman

Nate Hansen

Our Progress

Isabella Gilman

Structure


CR16-top
BRAM
CR16
Program Counter
Datapath
Status Flags
ALU
Regfile

Module Overview

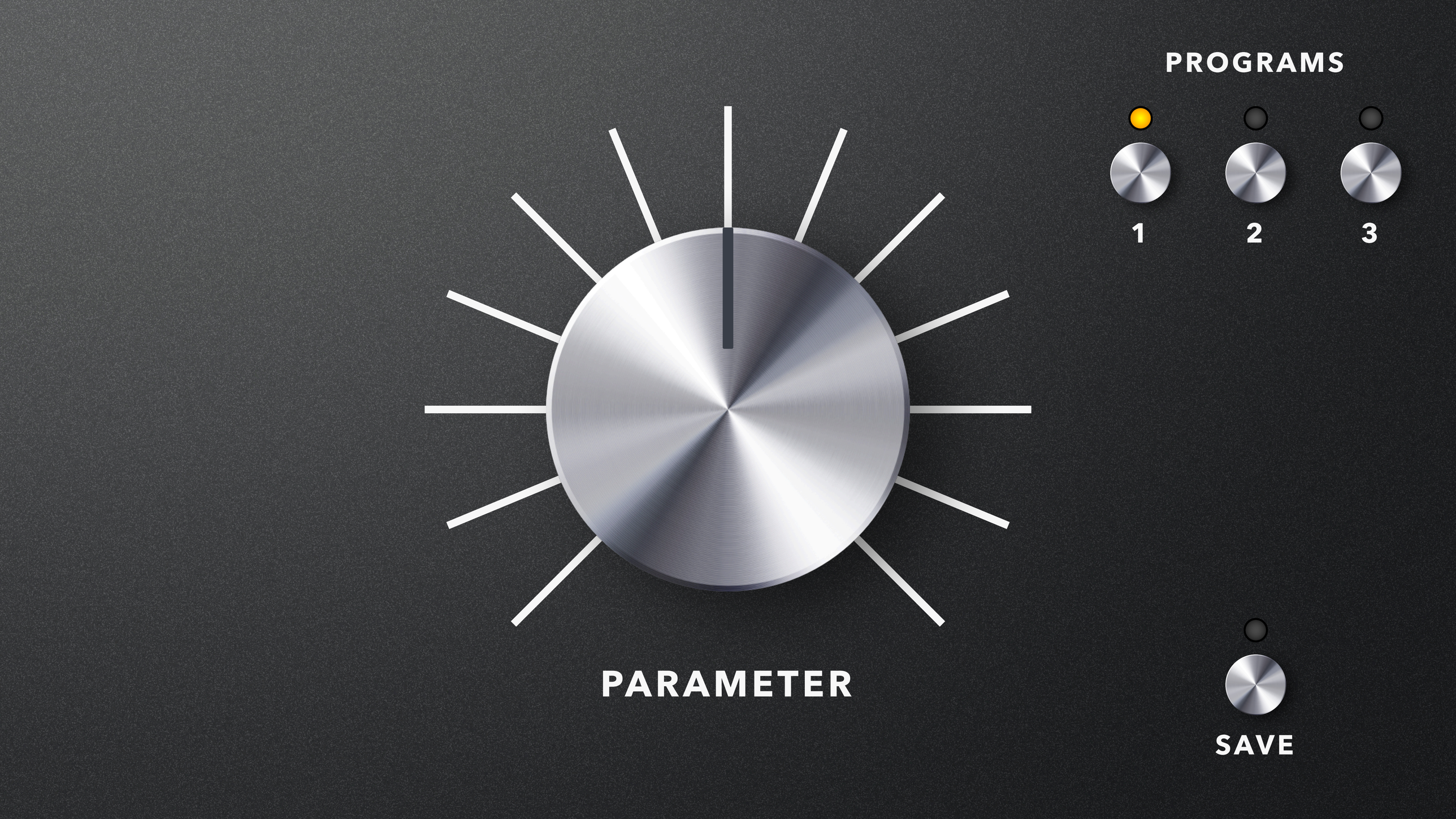
- CR16-top
  - Connects the BRAM and CR16 modules so that the CR16 can process instructions loaded from memory.
- BRAM
  - Instantiates memory on the FPGA.
- CR16
  - Connects the datapath to the program counter. Essentially tells CR16 which instruction to perform based on the program counter.
  - Implements Fetch, Decode, and Execute.
- Datapath
  - Contains ALU and Regfile.
  - Able to load outputs of the ALU back into the Regfile.

Our Progress

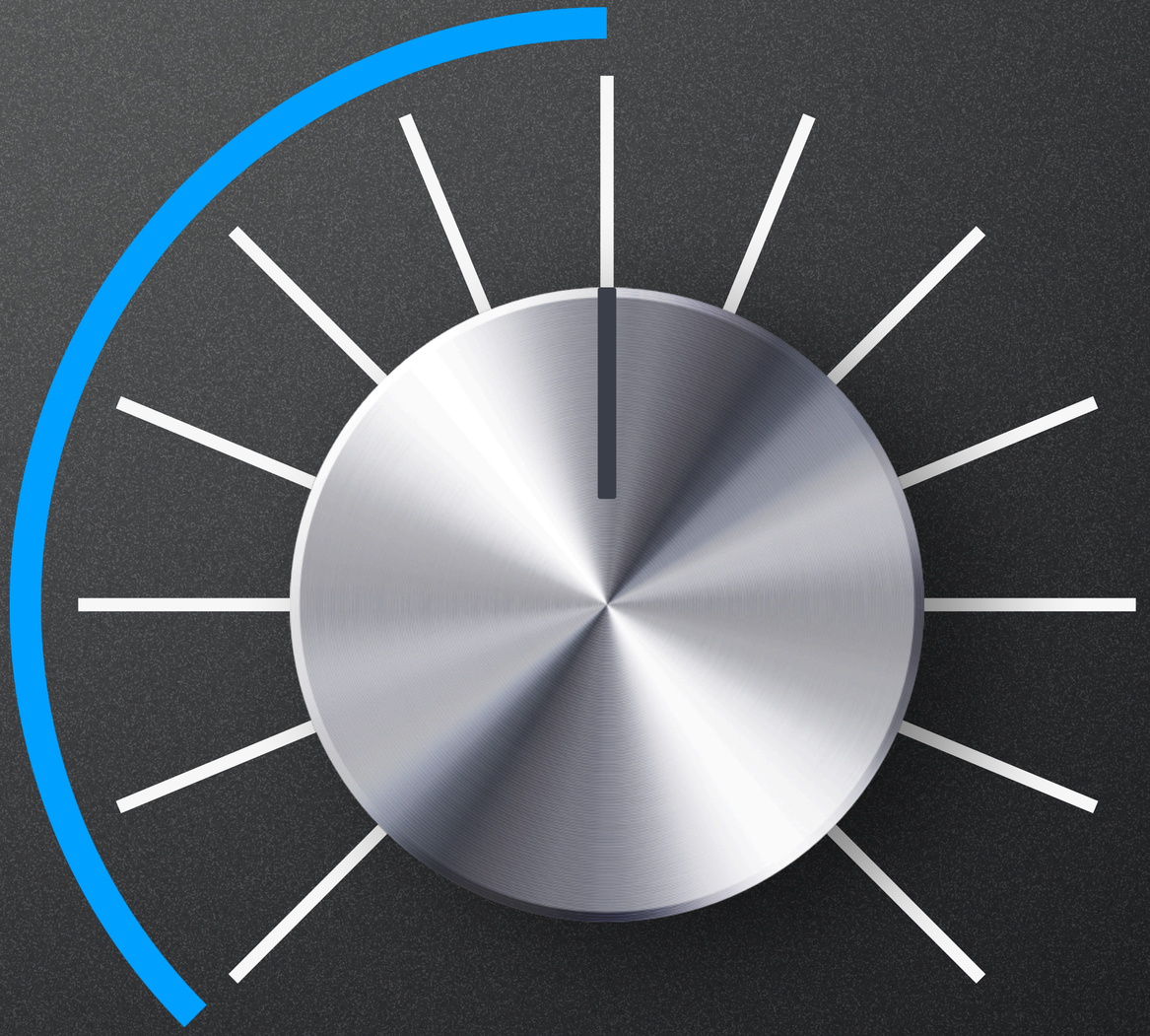
Currently, the CR16 FSM is able to perform R-type instructions. Load/Store type instructions as well as jump and branching will be implemented in the coming week.

Application Overview

Brady Hartog


PROGRAMS
1
2
3
PARAMETER
SAVE

physical value

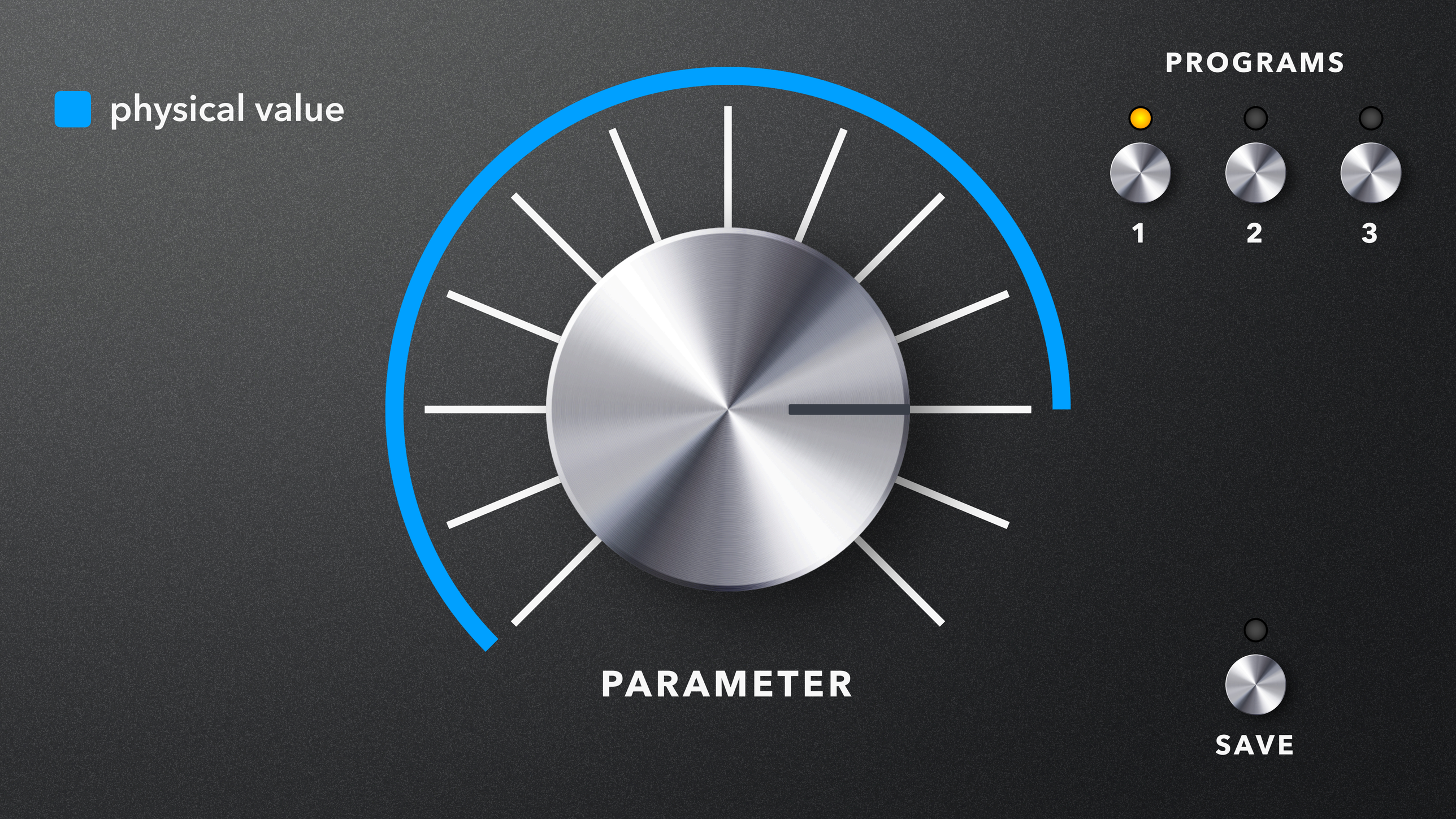


PARAMETER

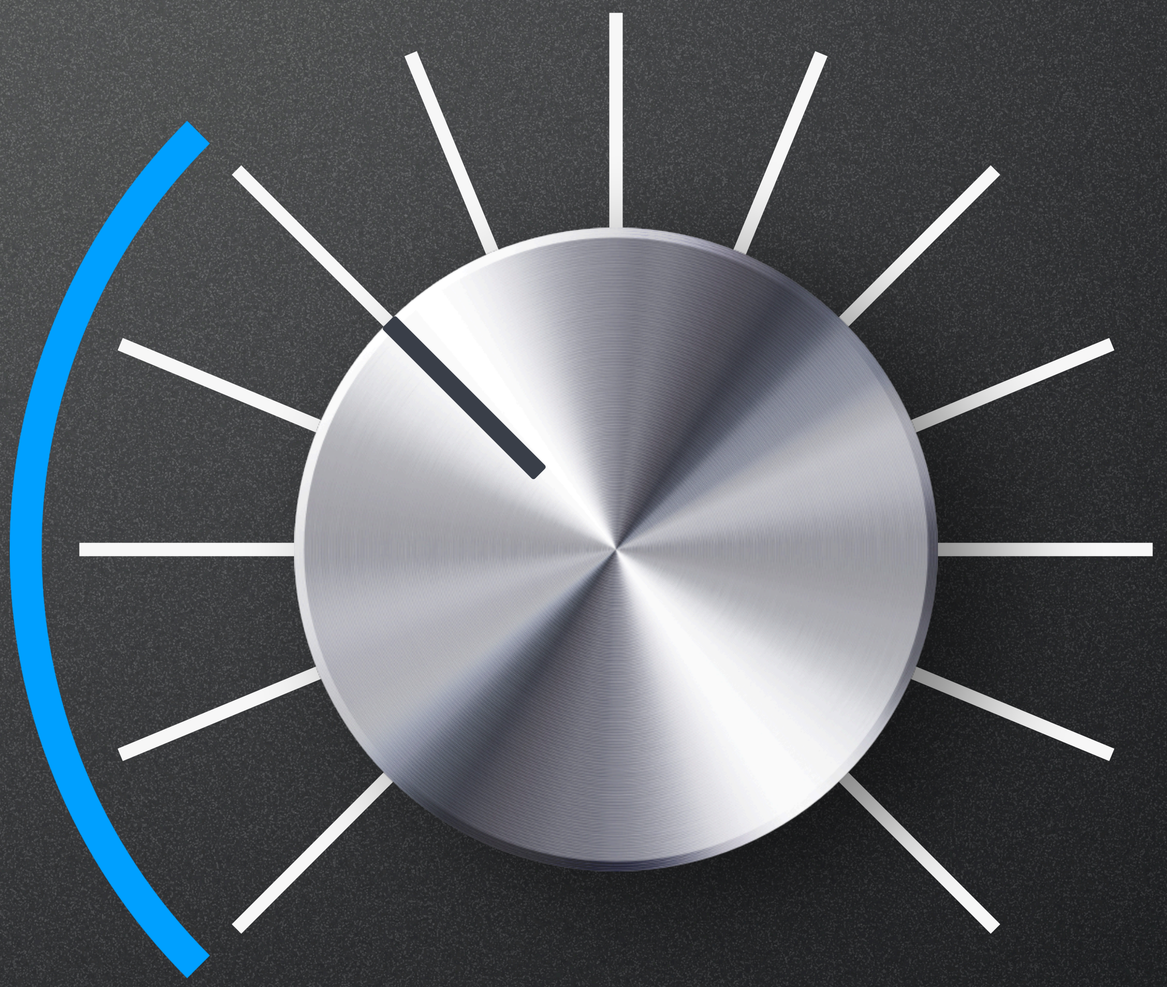
PROGRAMS

1 2 3

SAVE


physical value
PROGRAMS
1
2
3
PARAMETER
SAVE

physical value

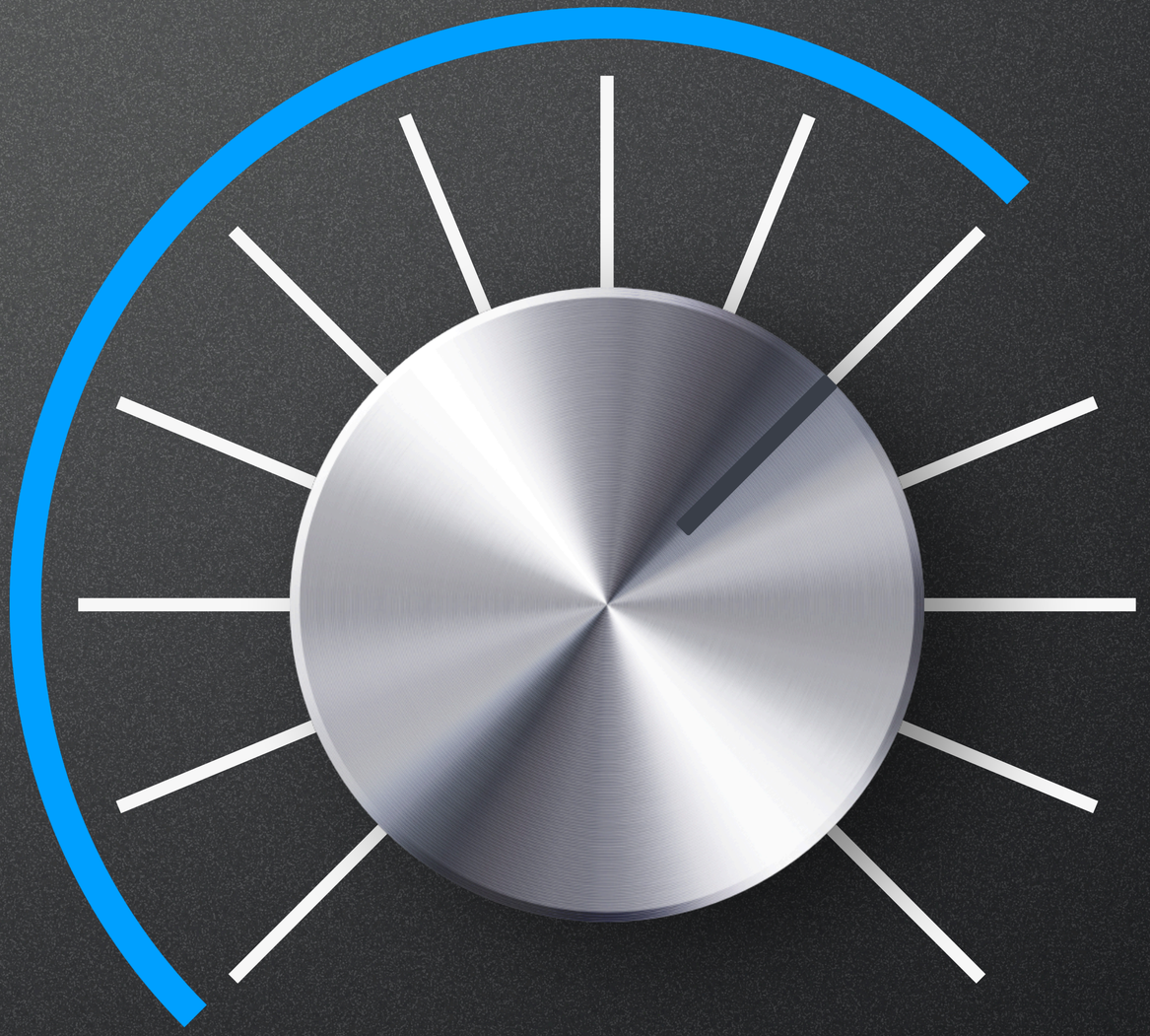


PROGRAMS

1 2 3

SAVE

physical value

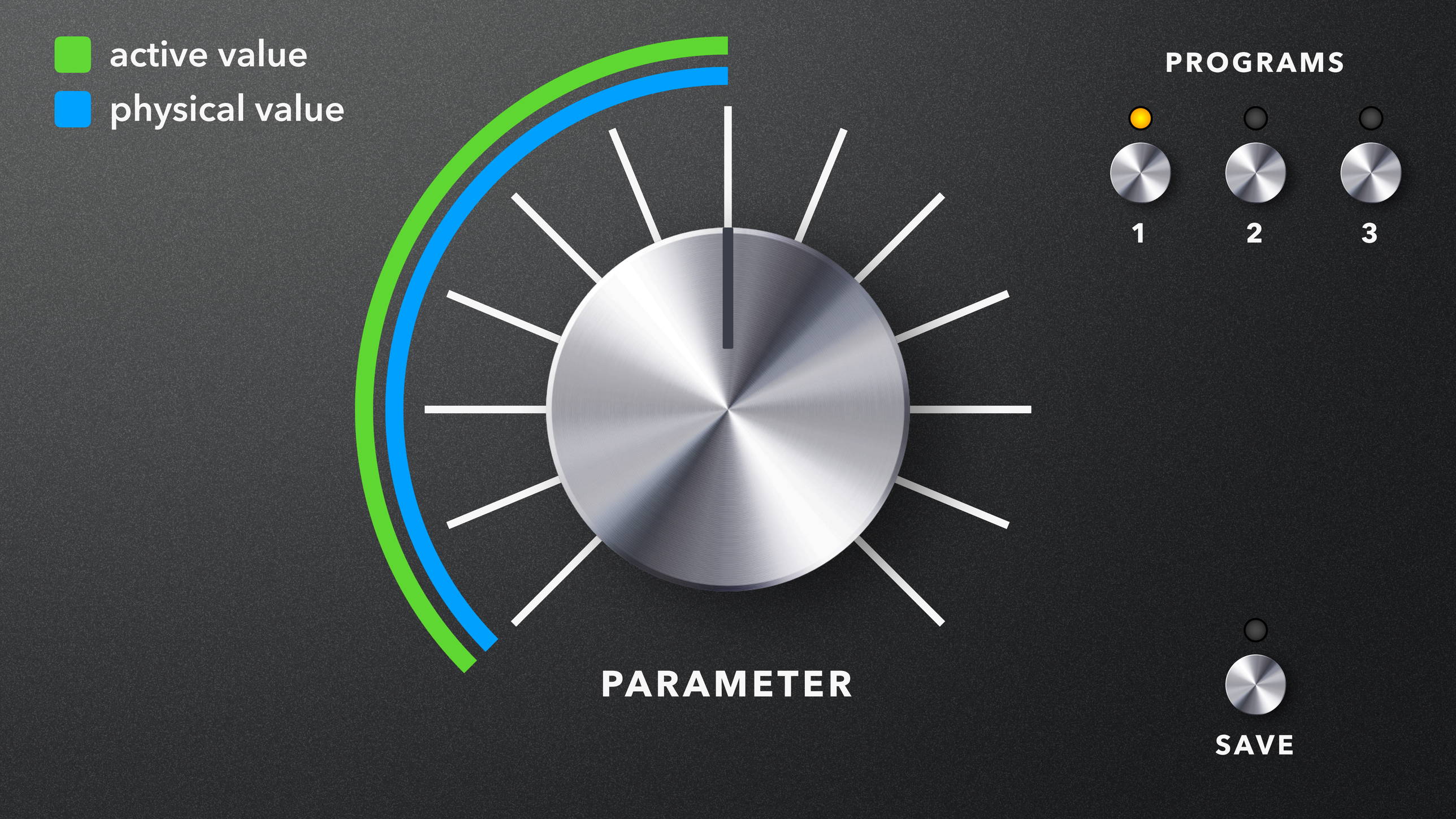


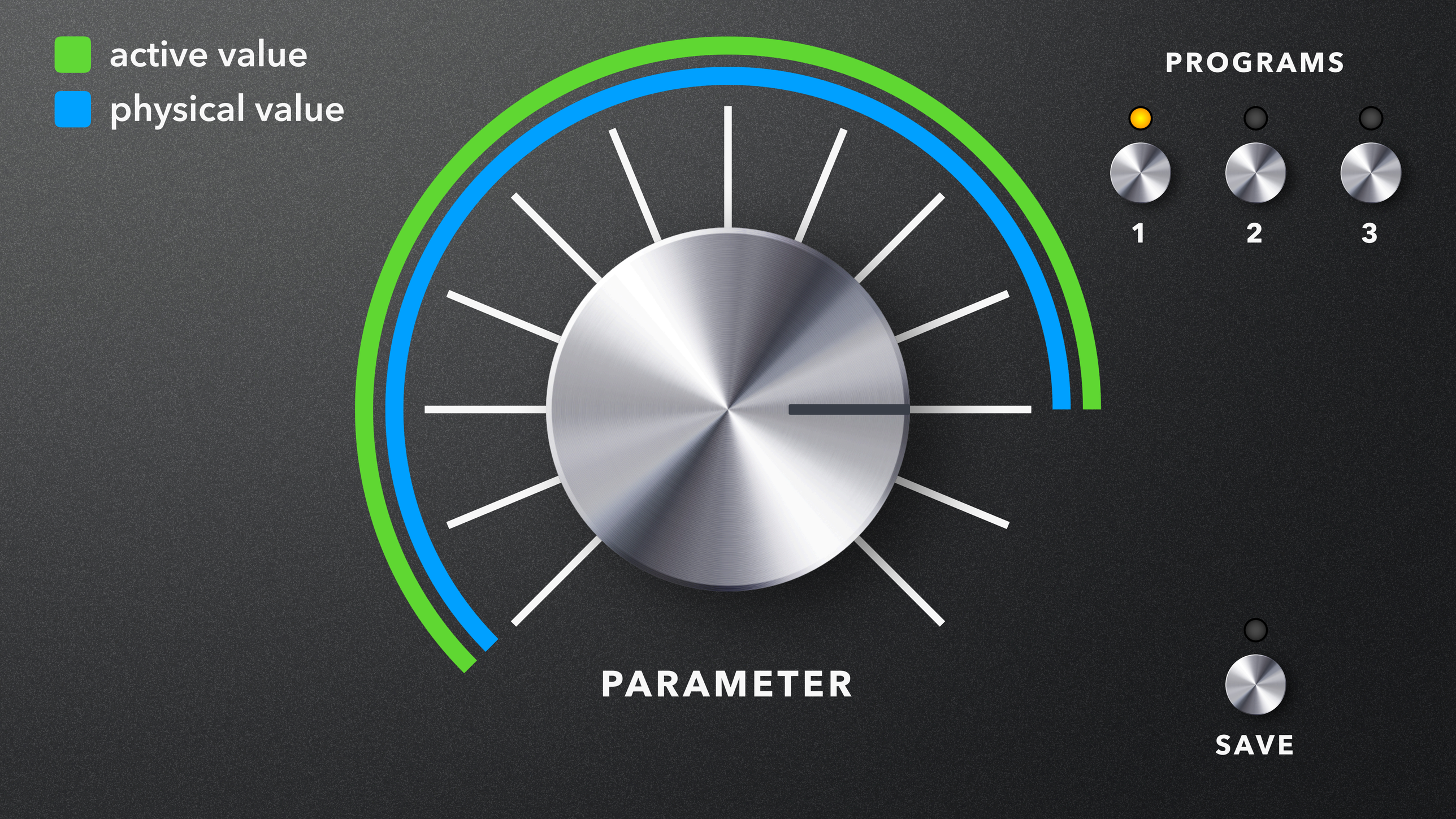
PARAMETER

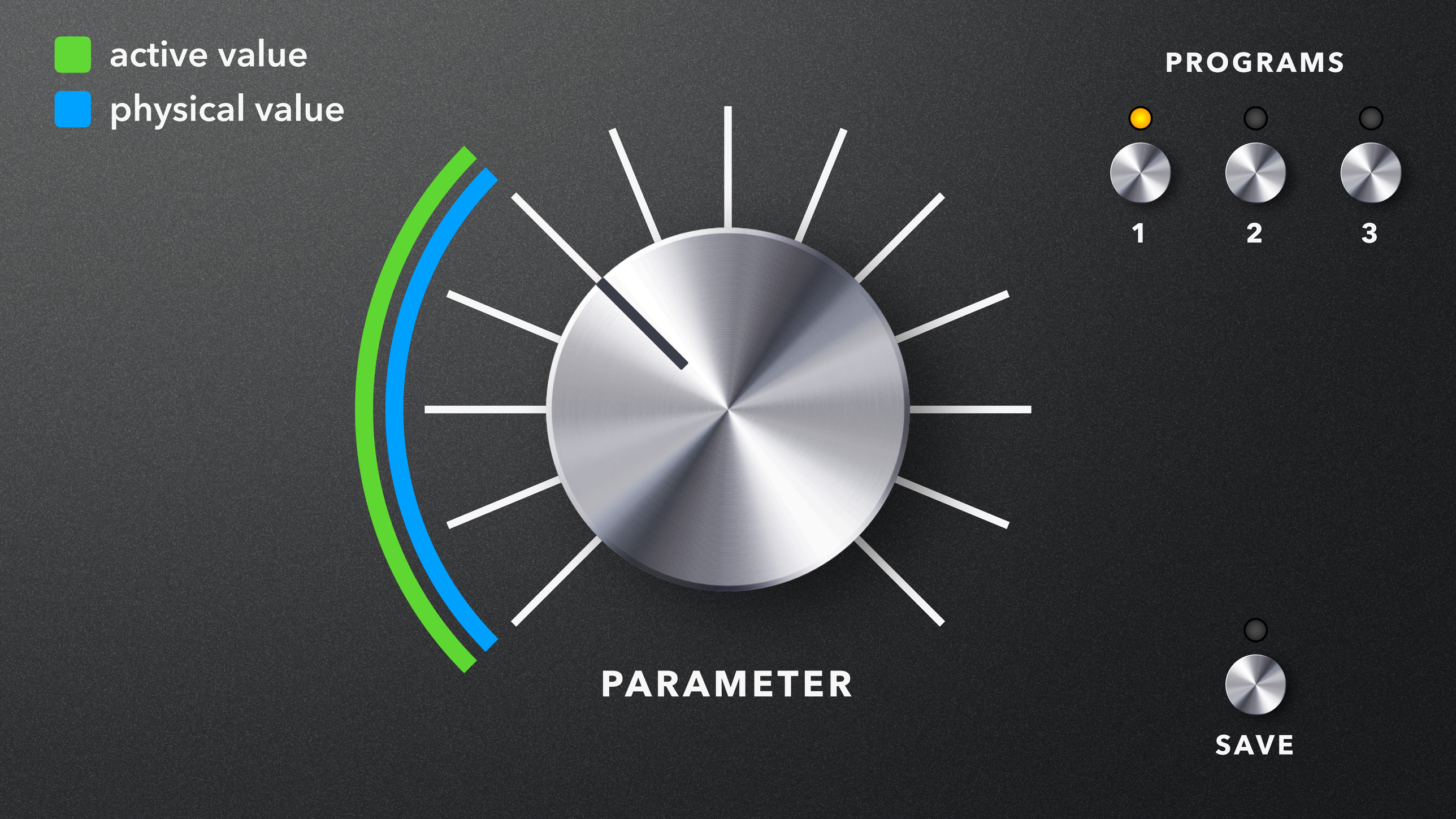
PROGRAMS

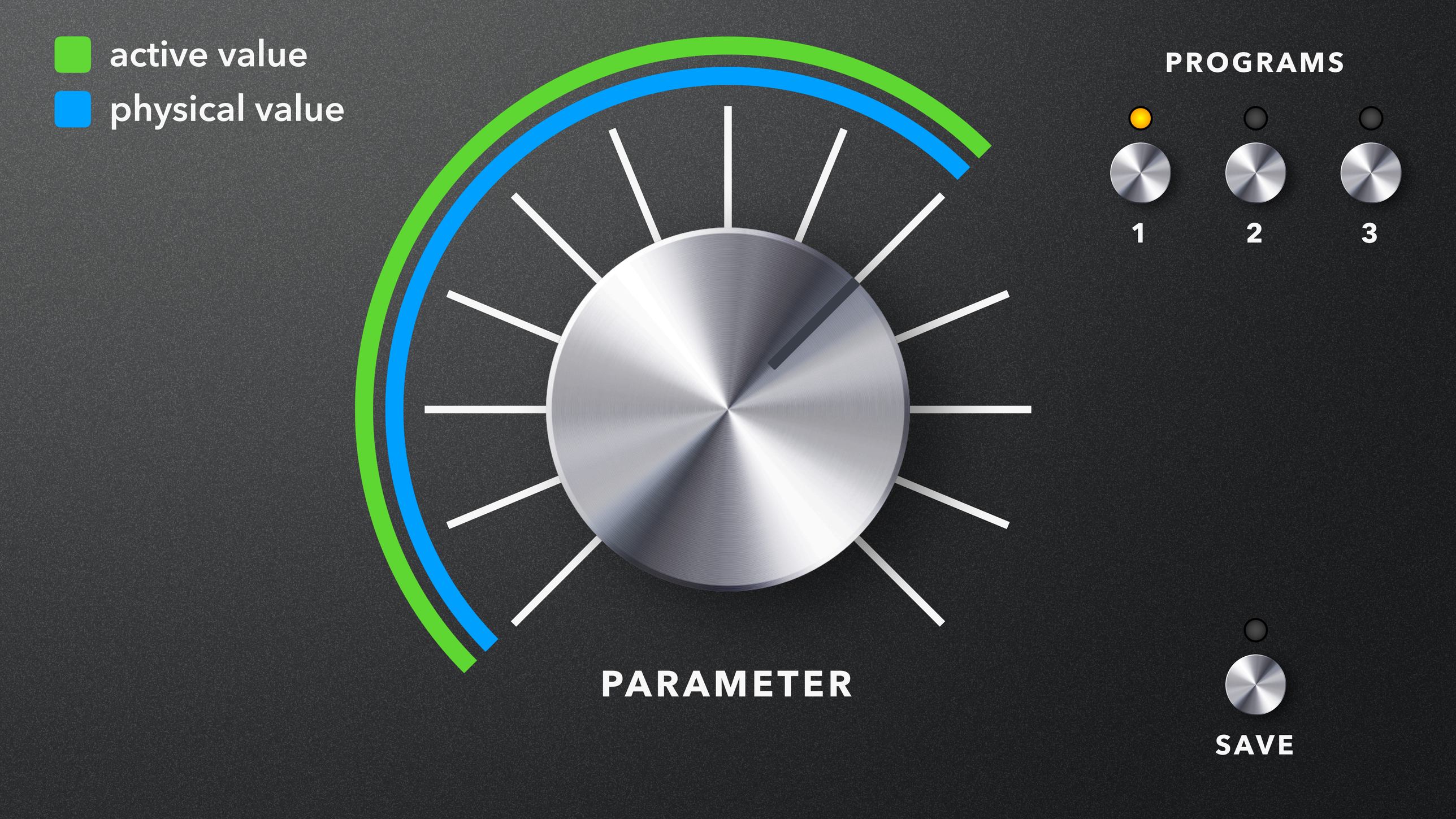
1 2 3

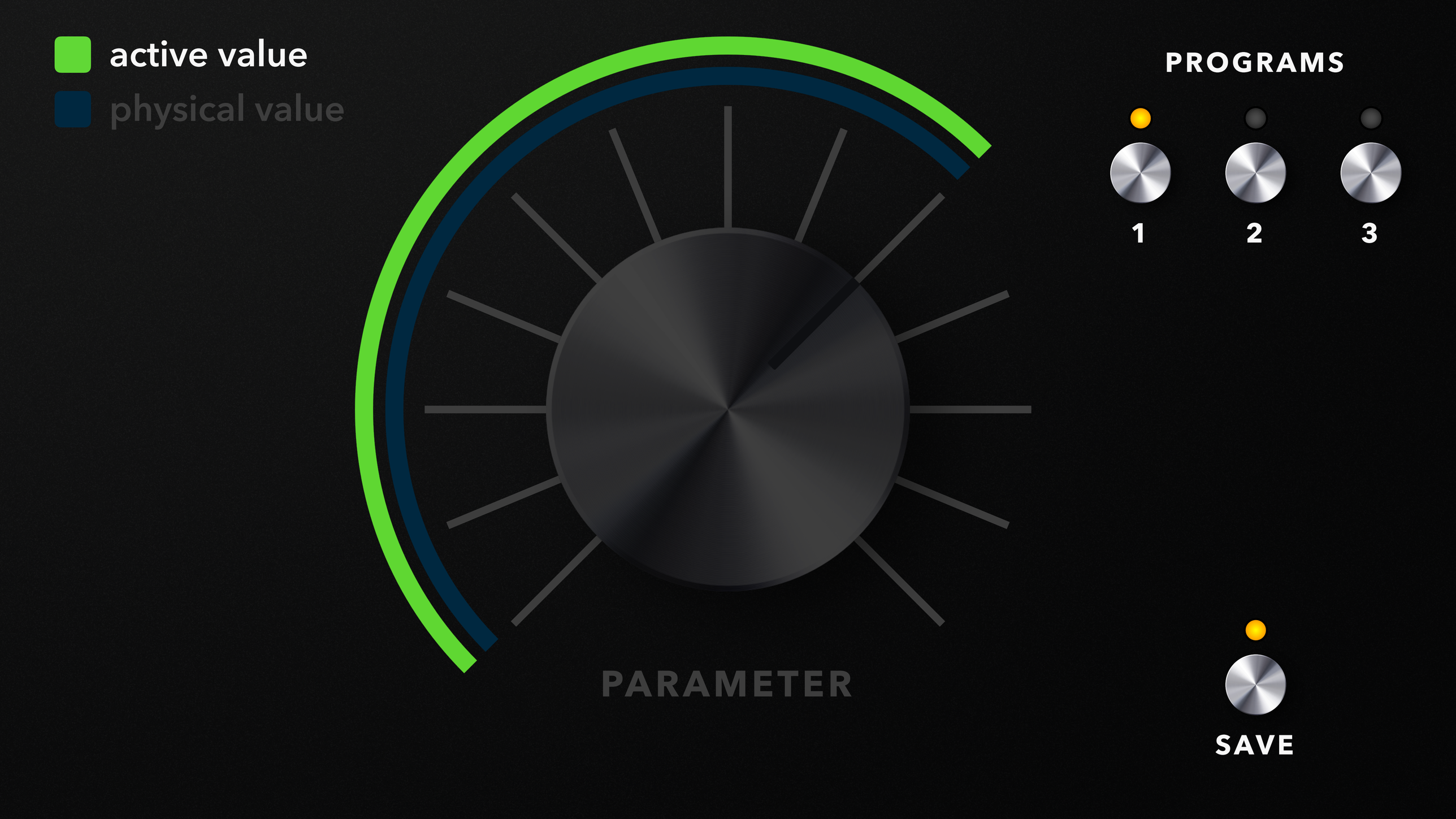
SAVE

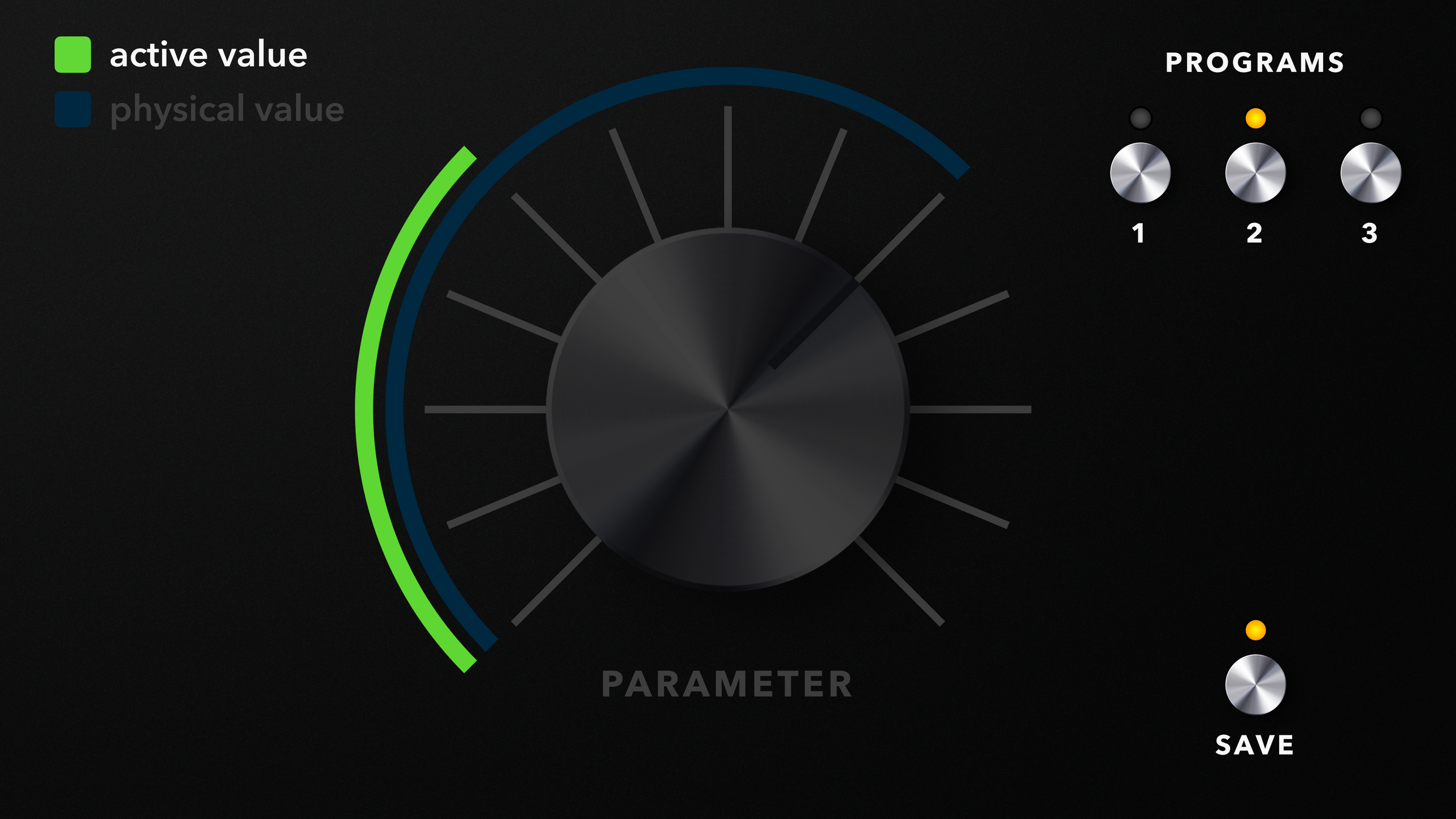

active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE

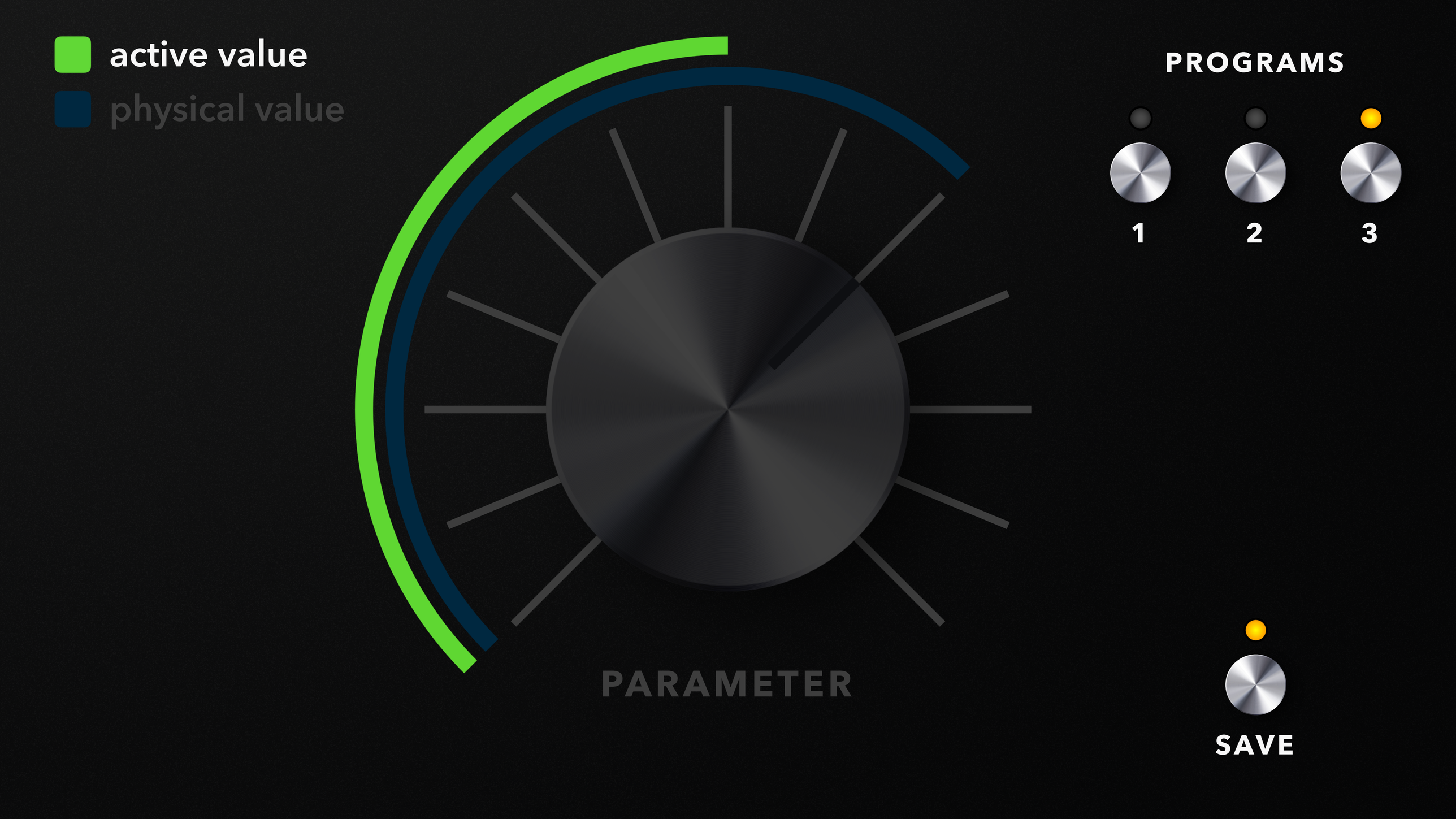

active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE

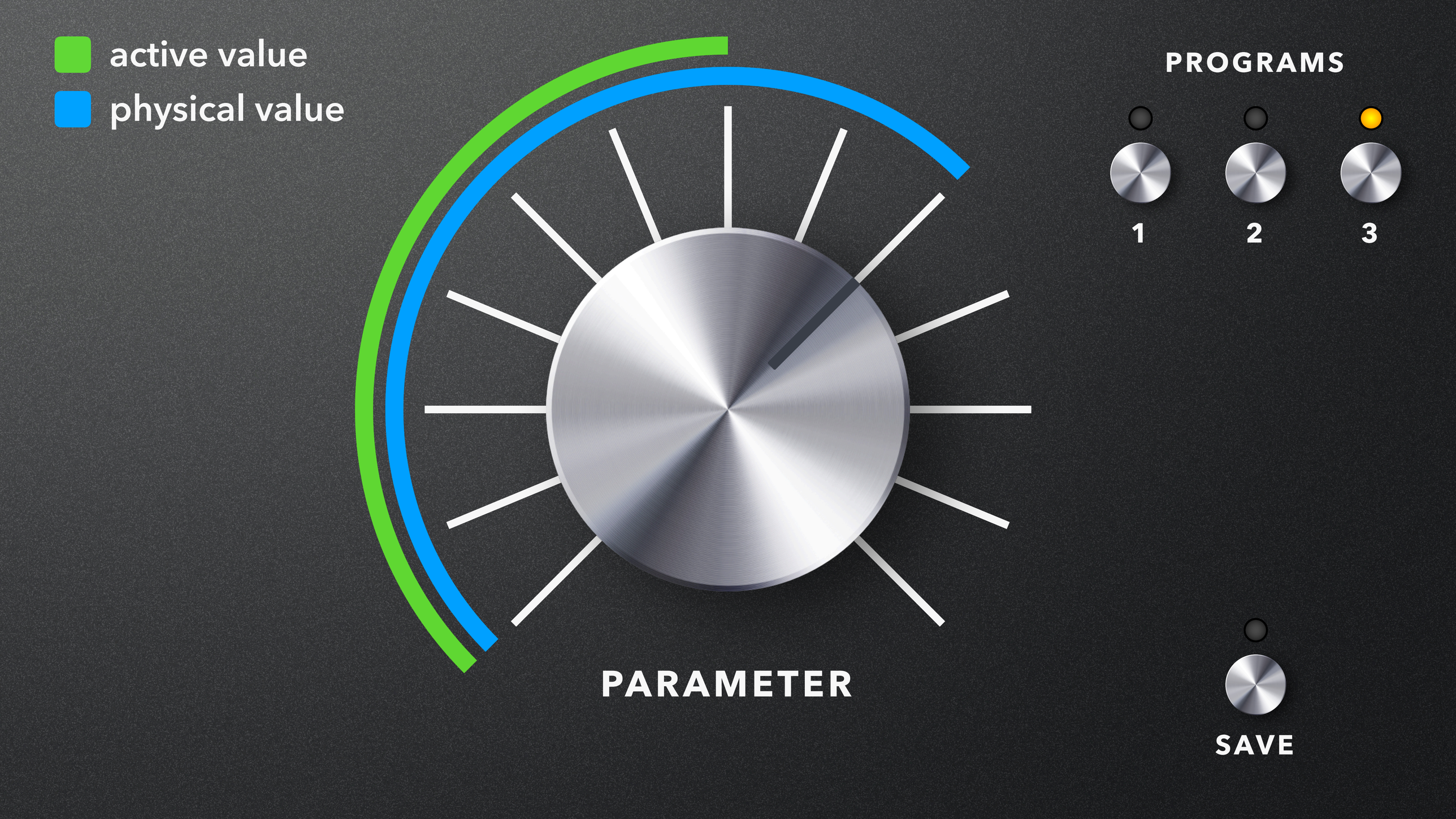

active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE


active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE

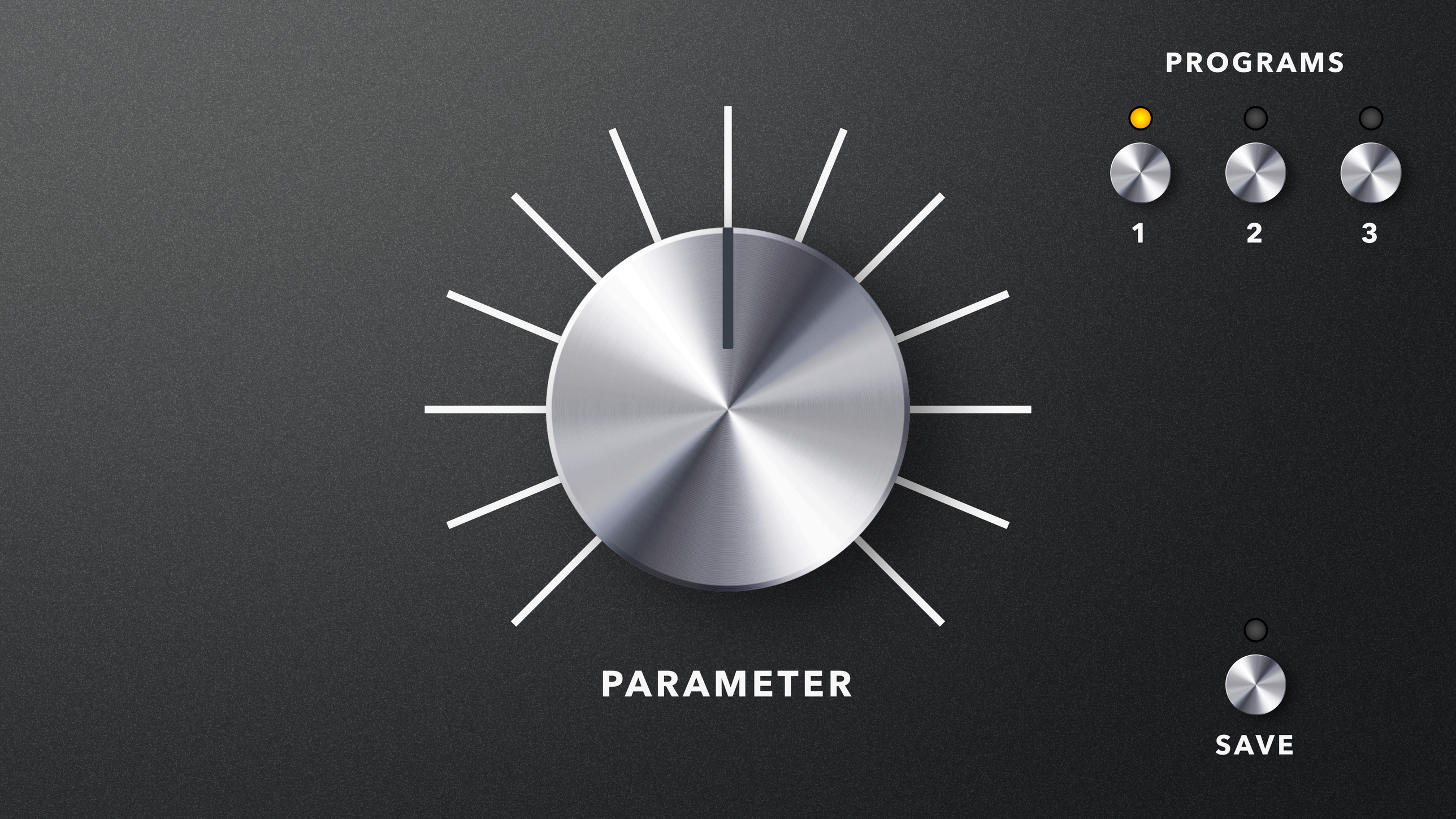

active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE


active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE


active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE


active value
physical value
PROGRAMS
1
2
3
PARAMETER
SAVE


active value
physical value
PROGRAMS
1
2
3
Values are out of sync!
PARAMETER
SAVE


PROGRAMS
1
2
3
PARAMETER
SAVE


PROGRAMS
1
2
3
PARAMETER
SAVE

physical + active value

PARAMETER

PROGRAMS

1 2 3

SAVE

physical + active value

PARAMETER

PROGRAMS

1 2 3

SAVE

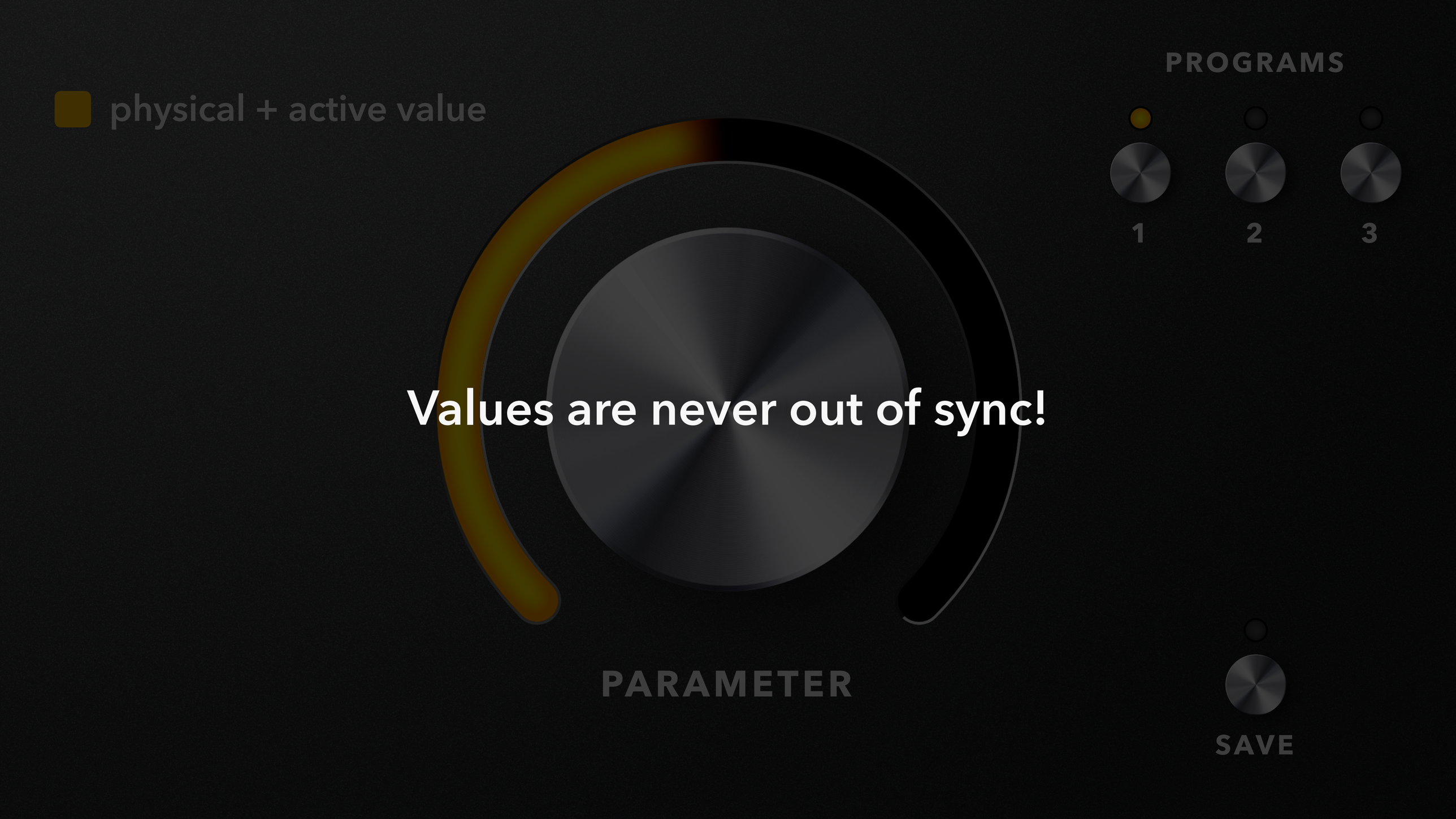
physical + active value

PARAMETER

PROGRAMS

1 2 3

SAVE


physical + active value
PROGRAMS
1
2
3
Values are never out of sync!
PARAMETER
SAVE


PROGRAMS
1
2
3
PARAMETER
SAVE


PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER
PARAMETER

The Future of the Synth Interface

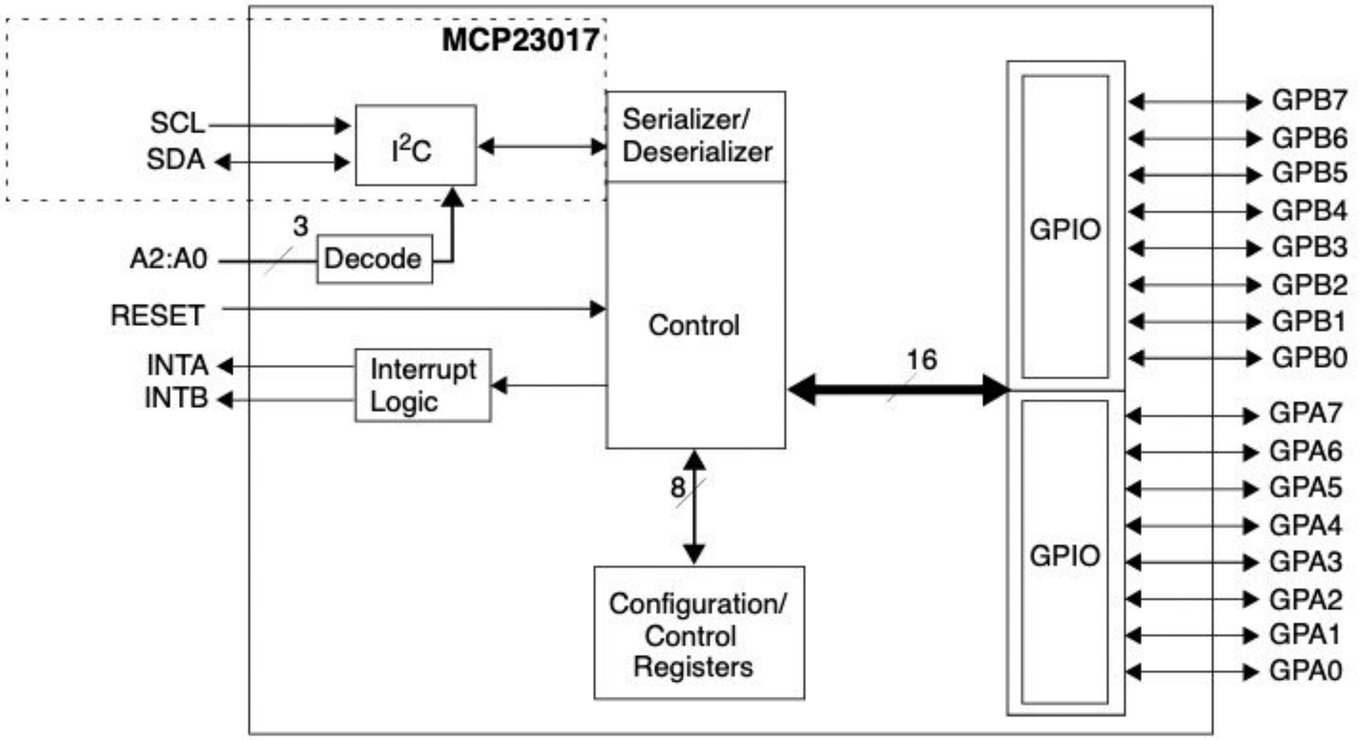
Fully-Synchronized Synthesizer (FSS) Prototype

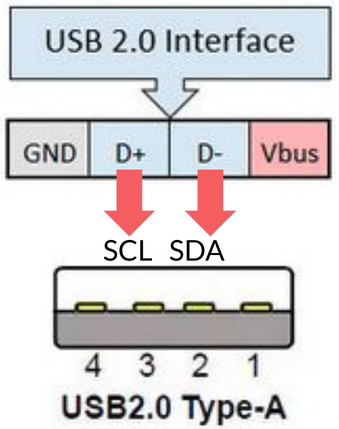
Hardware

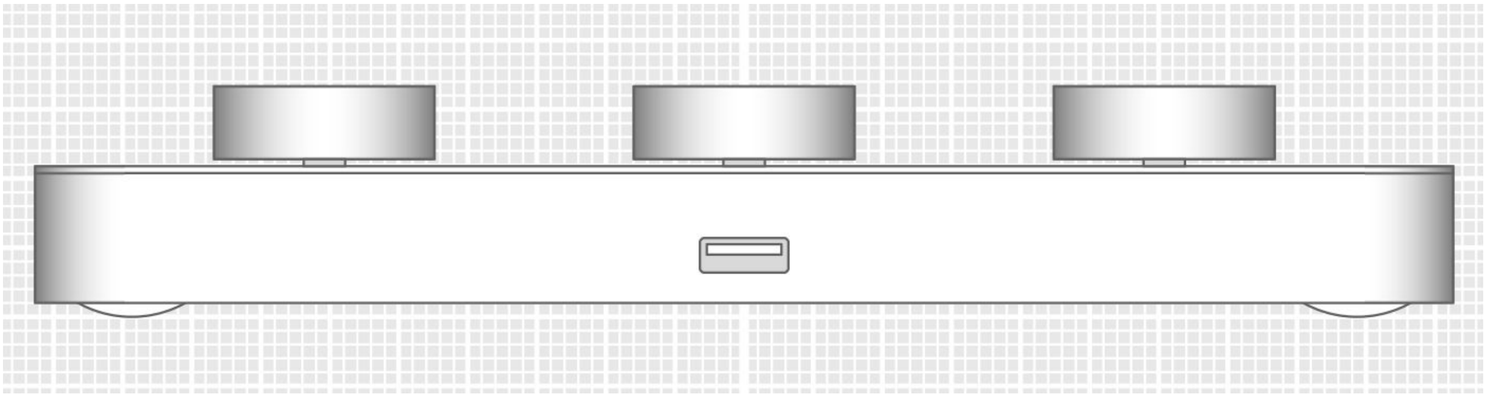
Jacob Peterson

FSS Interfacing

- A single cable interface

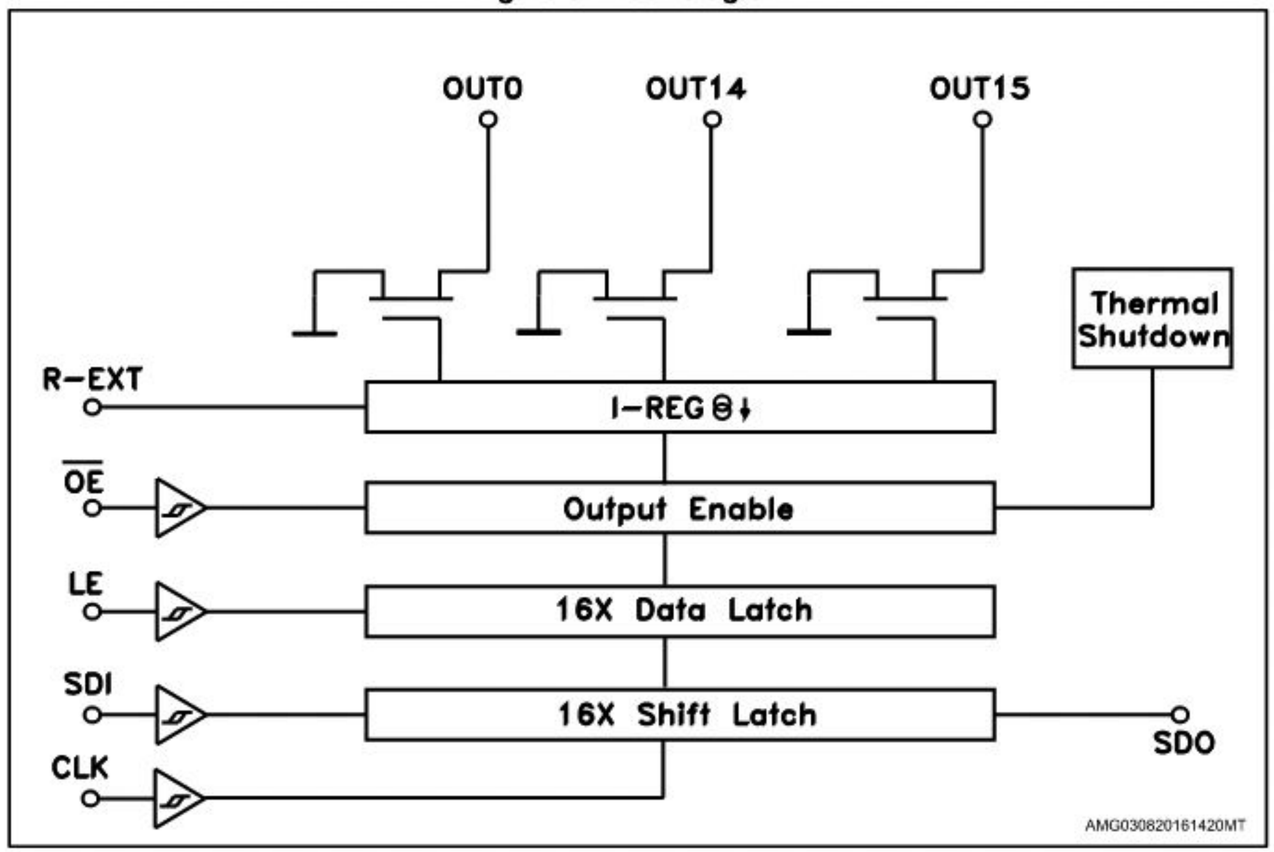






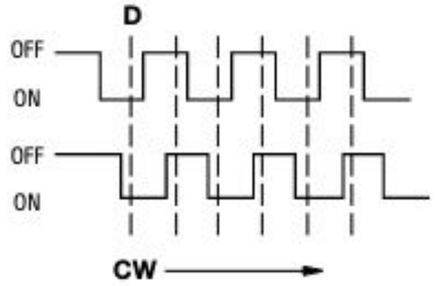
LED Drivers

Figure 6: Block diagram

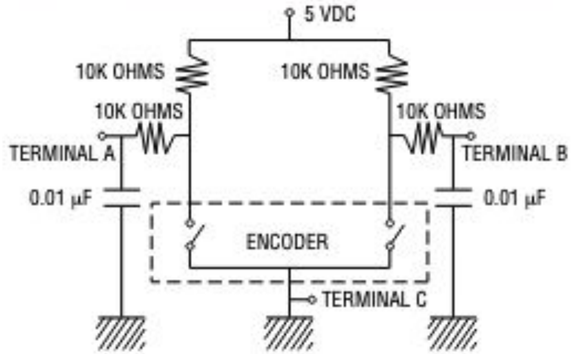


Rotary Encoders

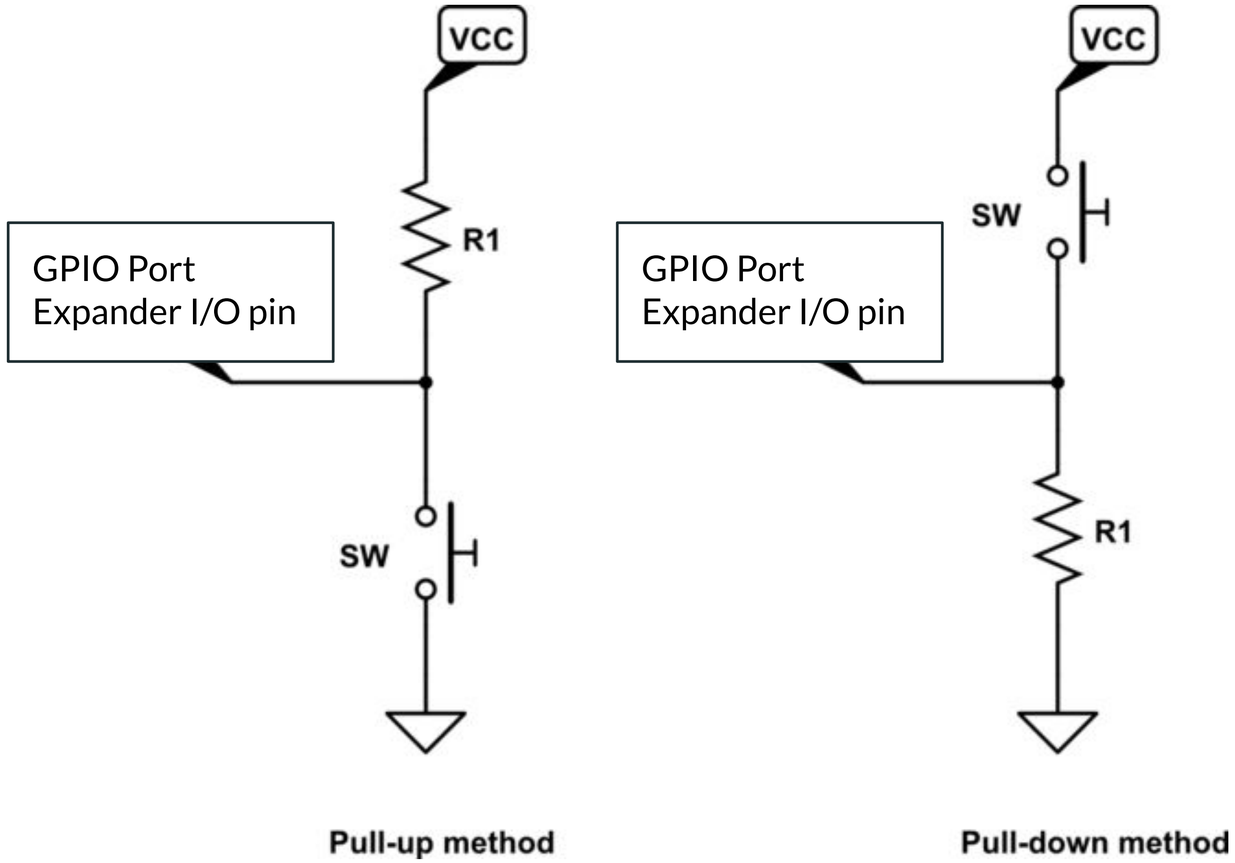
Quadrature Output Table



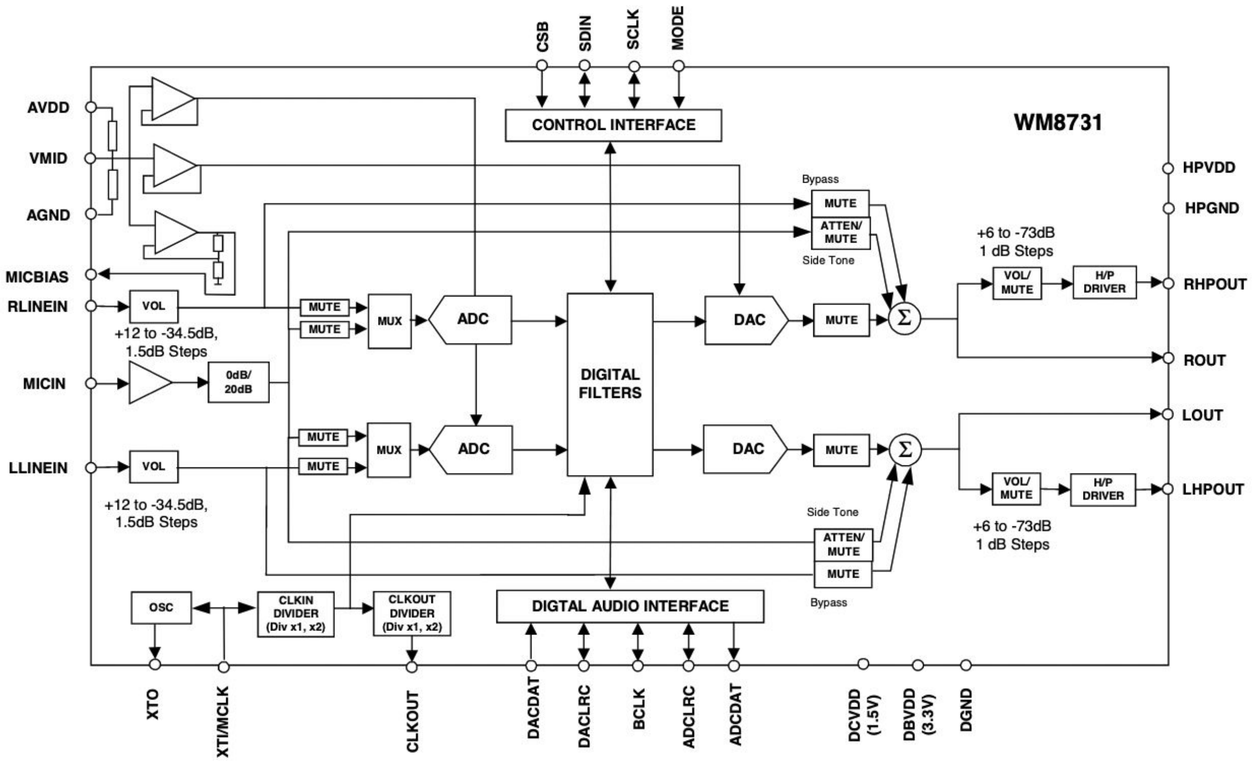
Suggested Filter Circuit



Push Buttons



WM8731 Audio Codec on FPGA Board



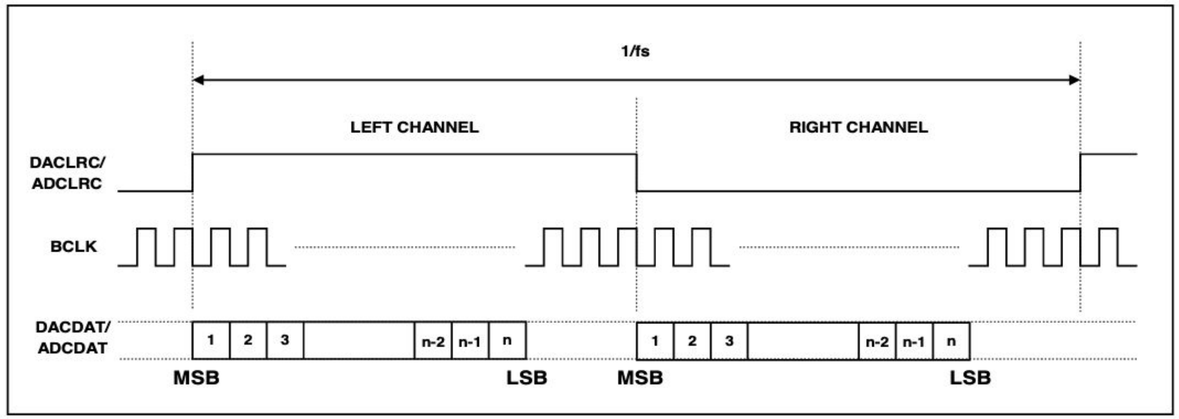
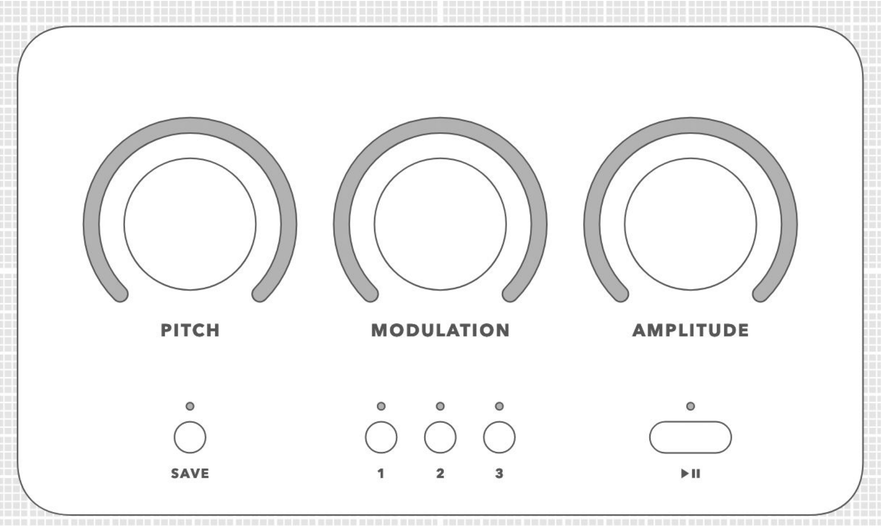


Figure 25 Left Justified Mode

Tools & Services



- VCS: Git/Github, istyle-verilog-formatter
- MCAD software of choice: SolidWorks
- ECAD software of choice: EasyEDA (easyeda.com)
  - Schematic design, PCB layout, Spice simulations
  - Cloud-based (collaborative)
  - Huge schematic/footprint library
  - Not as popular as KiCad, EAGLE, or Altium, but growing
- PCB manufacturers:
  - JLCPCB (easy export from EasyEDA, can assemble/reflow components for us)
  - OSHPark (use reflow oven)
- UofU Machine Shop
  - Water Jet or CNC anodized aluminum
  - CNC aluminum knobs and buttons
  - CNC wood blank for wood chassis
  - Stencil or Silk Screen to paint onto aluminum surface

Something pretty cool...

- Two instances of CR16 for “dual-core” processor
  - One core for audio codec interfacing via I2C and generating sound waveform to send off to audio codec DAC, one core for polling GPIO expander chip via I2C.
  - Core 1 connected to Port A of BRAM, Core 2 connected to Port B of BRAM, or we can partition BRAM for each core and have cores interface with each other using the LOADX and STOREX instructions.

Software

Nate Hansen

LSH	Rdest, Ramount	`Rdest = Rdest << Ramount`	0000	Rdest	1001	Ramount	$0 \leq$ Ramount ≤ 15 since registers are only 16-bits
LSHI	Rdest, ImmLo	`Rdest = Rdest << Imm`	0000	Rdest	1010	ImmLo	$0 \leq$ ImmLo ≤ 15
RSH	Rdest, Ramount	`Rdest = Rdest >> Ramount`	0000	Rdest	1011	Ramount	$0 \leq$ Ramount ≤ 15
RSHI	Rdest, ImmLo	`Rdest = Rdest >> Imm`	0000	Rdest	1100	ImmLo	$0 \leq$ ImmLo ≤ 15
ALSH	Rdest, Ramount	`Rdest = Rdest <<< Ramount`	0000	Rdest	1101	Ramount	$0 \leq$ Ramount ≤ 15
ALSHI	Rdest, ImmLo	`Rdest = Rdest <<< Imm`	0000	Rdest	1110	ImmLo	$0 \leq$ ImmLo ≤ 15
ARSH	Rdest, Ramount	`Rdest = Rdest >>> Ramount`	0000	Rdest	1111	Ramount	$0 \leq$ Ramount ≤ 15
ARSHI	Rdest, Imm	`Rdest = Rdest >>> Imm`	1111	Rdest	0000	ImmLo	$0 \leq$ ImmLo ≤ 15
MOV	Rdest, Rsrc	`Rdest = Rsrc`	1111	Rdest	0001	Rsrc	Copies Rsrc into Rdest
MOVIL	Rdest, Lower Imm	`Rdest[7:0] = Imm`	1010	Rdest	ImmHi	ImmLo	Zero extended Imm, moves immediate value into lower bits of Rdest
MOVIU	Rdest, Upper Imm	`Rdest[15:8] = Imm`	1011	Rdest	ImmHi	ImmLo	Zero padded Imm, moves immediate value into upper bits of Rdest
J[condition]	Rtarget	`if [condition]:` `PC = Rtarget`	1111	condition	0010	Rtarget	[condition] bit patterns are in Table 2.
B[condition]	Displacement Imm	`if [condition]:` `PC += Imm`	1100	condition	ImmHi	ImmLo	[condition] bit patterns are in Table 2. Immediate is sign extended 2's complement for program counter/address displacement.
CALL	Rtarget	`Pushes PC onto stack,` `PC = Rtarget`	1111	xxxx	0011	Rtarget	Used for nested subroutines

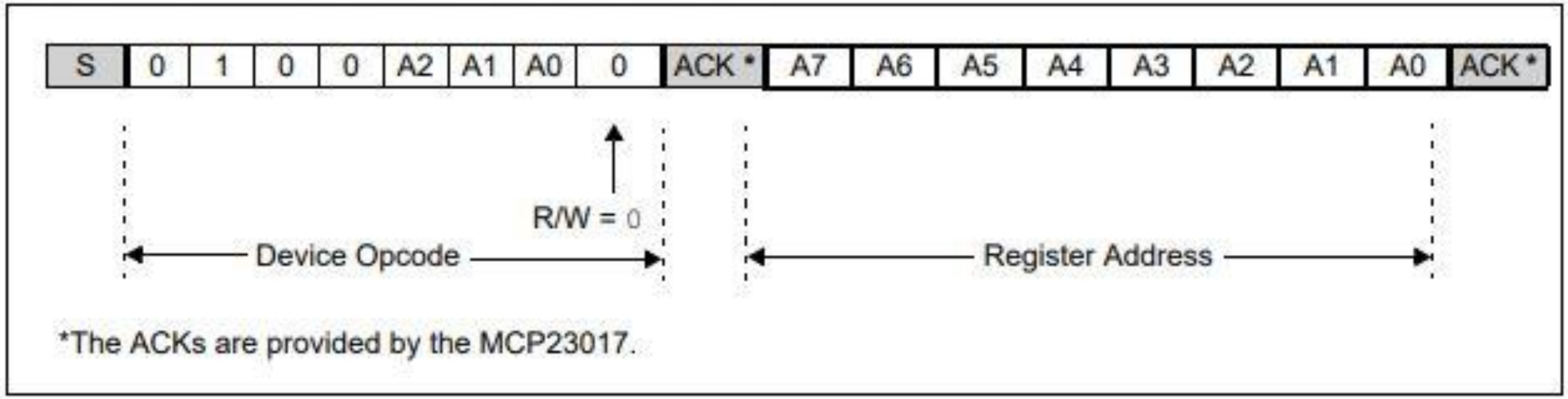
RET		`Pops top of stack into PC, PC++`	1111	xxxx	0100	xxxx	Used to return from nested subroutine
LPC	Rdest	`Rdest = PC`	1111	Rdest	0101	xxxx	Sets Rdest to the current instruction address/PC
LSF	Rdest	`Rdest[4:0] = status flags`	1111	Rdest	0110	xxxx	Sets 5 least significant bits of Rdest to the current status flags
SSF	Rsrc	`Status flags = Rsrc[4:0]`	1111	xxxx	0111	Rsrc	Sets current status flags to 5 least significant bits of Rsrc
PUSH	Rsrc	`rsp--, Main memory value at rsp = Rsrc`	1111	xxxx	1000	Rsrc	Pushes Rsrc onto top of stack
POP	Rdest	`Rdest = Main memory value at rsp, rsp++`	1111	Rdest	1001	xxxx	Pops top of stack into Rdest
LOAD	Rdest, Raddr	`Rdest = Main memory value at Raddr`	1111	Rdest	1010	Raddr	Used to load data at Raddr into Rdest from main memory
STORE	Raddr, Rsrc	`Main memory value at Raddr = Rsrc`	1111	Raddr	1011	Rsrc	Used to store data at Raddr from Rsrc to main memory
LOADX	Rdest, Raddr	`Rdest = External memory at Raddr`	1111	Rdest	1100	Raddr	Used to load data at Raddr into Rdest from external/peripheral memory/registers
STOREX	Raddr, Rsrc	`External memory value at Raddr = Rsrc`	1111	Raddr	1101	Rsrc	Used to store data at Raddr from Rsrc to external/peripheral memory/registers

Table 3: Register Naming and Conventions

Register Index	Register Name	Meaning
4'd15	rsp	Stack pointer with an address starting at 0xFFFF (2^{16}) and grows downward towards dynamically allocated memory
4'd14	r14	4th subroutine argument
4'd13	r13	3rd subroutine argument
4'd12	r12	2nd subroutine argument
4'd11	r11	1st subroutine argument
4'd10	r10	Return value of subroutine
4'd9	r9	Caller-owned
4'd8	r8	Caller-owned
4'd7	r7	Caller-owned
4'd6	r6	Caller-owned
4'd5	r5	Callee-owned
4'd4	r4	Callee-owned
4'd3	r3	Callee-owned
4'd2	r2	Callee-owned
4'd1	r1	Callee-owned
4'd0	r0	Callee-owned

Software for I2C

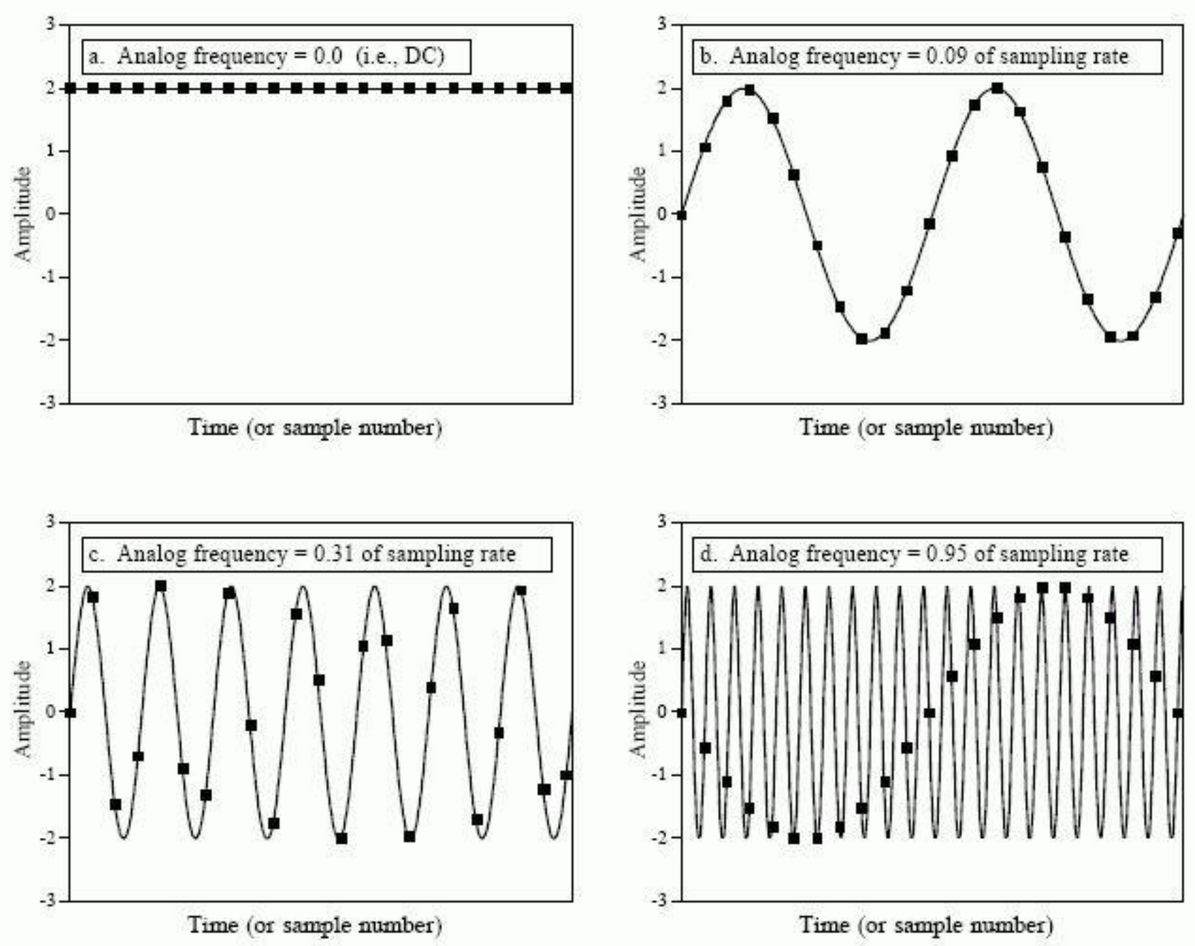
FIGURE 3-6: I²C ADDRESSING REGISTERS



- I2C Protocol:
  - Two options: Implement a Verilog module to handle I2C protocol, or implement protocol control in software.
- The main function will poll the GPIO expander chip for the I/O state of the rotary encoders and buttons via I2C.

Sound Generation/Manipulation

- Parametrize the frequency and amplitude of a simple sine wave.
- .text assembly section, lookup table for sine, triangle, square, sawtooth waves.
- Values will be hex with bit depth of ~10.



The Sampling Theorem (dspguide.com)

Timeline

Isabella Gilman

Timeline

November 11th - finish CR16

November 18th - CAD drawings, purchase parts, start writing assembly

November 25th - machine/print parts, stain wood, reflow PCB components, assemble FSS

December 2nd - finish code and testing

December 9th - super cool, revolutionary, jaw-dropping, unbelievable, unimaginable, amazing demo video