Tiny Tapeout SKY 25b Datasheet

github.com/TinyTapeout/tinytapeout-sky-25b

February 14, 2026

Table of Contents

Chip Renders **1**
Shuttle Map 2
GDS 3
Logic Density 4
Projects **5**
0000 Chip ROM 6
0001 Tiny Tapeout Factory Test 8
0002 Tiny Tapeout Hackathon UofT Day 2 Template 10
0003 Test 11
0004 Fawwaz's Project 12
0005 DAMTinyTape 13
0006 Bob design 14
0007 doritos 15
0008 Tiny_Tapeout_Test_Gates 17
0009 Leena and Gabby's coding masterpiece 18
0010 EJ Wokwi proj. 19
0011 Minhala's tinytapeout 20
0012 AnalogClockTest1 21
0013 Simon Memory Game 22
0014 Tiny Tapeout Test Gates 23
0015 tiny tapeout 24
0032 WOKWI Design - Joelle 25
0033 WokwiDesign 26
0034 Semi-Conductor Design 27
0035 gds 28
0036 chipchip 29
0037 Hardware Hackday Test Gates 30

0038 Tiny Tapeout Gates 31
0039 Tinytapeoutchipnikki 32
0040 Overly Complicated Light Switch 33
0041 Inverting Numbers 34
0042 Single Digit Sums by Mark Angelov 35
0043 Tiny Tapeout Test Gates 36
0044 Tiny tapeout test gates 38
0045 Arjun's 1st Wokwi design 39
0046 flipper 1 to 5 40
0047 Johans Wokwi design 41
0065 Chip Design (Working title) 43
0067 FlippyFlop 44
0069 Chip-Test 45
Artwork MPW-6 "Hack SoC" 46
0070 Tiniest GPU V2 47
0071 4 switch, seven segment display 49
0073 T Flip Flop 50
0075 Tiny Tapeout Test Gates 51
0077 tiny tapout test gates 52
0078 Blake2s 53
0079 Nicky - Tiny Tapeout 61
0128 Function Generator 62
0129 EmilsChip 66
0130 Simon Says memory game 67
0131 first-chip-project 70
0132 Lauflicht_Erweiteung_HTL_LEONDING 71
0133 EnDecoder 72

0134 Tiny Tapeout Template 73
0135 Austrochip 2025 Tiny Tapeout Project 74
0136 Tiny Tapeout Projekt 75
0137 transformer-attention-engine 76
0138 Tiney Tapeout Test Gates 78
0139 tt_primitiv_rotating_7segment 79
0140 Tiny tapeout test gates 80
0141 VGA Pattern Generator 82
0142 INTERCAL ALU 84
0143 tiny tapeout test gates 89
0160 Wowki 90
0161 Arrow Board 91
0162 Tiney Tapeout Semiconduxtor 98
0163 Ring Counter 99
0164 Svens Woki design 100
0165 Tiny Tapeout Test Gates 101
0166 Tiny Tapeout Project - Binary to Decimal Display 102
0167 FullAdder 103
0168 Tiny Tapeout Template Copy 104
0169 Project Title 105
0170 tinytapeout 106
0171 Jarle Chip 108
Artwork MPW-2 poly layers 109
0172 Tiny Tapeout Test Gates 110
0173 Wokwicreation 111
0174 sigma uncount 112
0175 Tiny Tapeout Test Games 113

192 VGA clock 114
193 Austrochip 2025 Workshop Design 115
194 RGB Mixer demo 116
195 Full Adder 117
196 spi_slave_ctrl 118
197 First Wokwi design 119
198 Jsilicon 120
199 Digital Filter 125
200 SimplePattern 127
201 4-Bit ALU 129
202 Crazy logic gates 131
203 Lauflichterweitung 132
204 TinyTapeoutProject 133
205 Universal Binary to Segment Decoder 134
206 base-4-circuit-addition 143
207 Hardware UTF Encoder/Decoder 144
227 Asicle v2 149
229 Ring Oscillators 152
231 Nauta OTA with digital trimming 155
233 Analog VGA CSDAC experiments (TTSKY25b) 157
238 UNAL's RISCV 161
239 tnt's variant of SKY130 mini-MOSbius 162
256 test-project. 164
257 TTSKY25A Register File Test 165
258 Can you count binary? 166
259 Relaxation oscillator 168
260 Tiny_Tapeout_Scoreboard 169

261 Relax_Oscillator 171
262 7SegDec 172
263 Oscillating Bones 173
264 7segCounter 175
Artwork Detail - Fibonacci design (MPW-2) 176
265 ttsky25_se_opamps 177
266 Shift_Register 179
268 2 Bit Counter 180
269 Tiny Flash ADC 181
270 Davids first Wokwi design 183
271 Onchip - Ring VCO 11 stages x2 184
288 4-bit Johnson Counter 186
289 HAD TT 187
290 test 188
291 Tiny Tapeout Binary to Decimal 189
292 miloh-tinytapeout-supercon2025 191
293 6502 Chip Selector 192
294 JBs first chip 194
295 Digit Fidgit 195
296 TinyTapeout 196
297 mysteryform 197
298 sparkhom SR flip flop 198
299 LeviathonGds 199
300 DUMB-Turing 200
301 AndNand 203
302 my first wokwi design 204
303 Tiny Tapeout 20251031 205

320 TinyTapeout Wokwi 206
322 8-Bit-CPU 207
324 Hamming (7,4) Encoder and Decoder 208
326 Tiny Tapeout 210
328 Test 211
329 Patrick first TT chip 212
330 Polyphonic MIDI-Synth 213
331 Simple Logic-Gate Controlled LEDs 217
332 counter7seg 218
333 jku-sky-LEDSpinner 219
334 tt-test 221
335 4 Bit Counter 222
416 Arko 223
Artwork 555 render (TT06) 224
417 4Bit_CPU 225
418 DETDFF (Tiny Tapeout) 226
419 j-initial-cycler 228
420 wokwi-adder 229
421 TinyTapeout2 230
422 OctoPWM 231
423 Jon_And_Angie_08281999 233
424 Sleepy Chip 234
425 Seven Segment Display Driver 242
426 Simple Tapped Delay Line 243
427 First WOWKI Project 244
428 Lights-Out 245
429 My Tiny Tapeout 250

430 Tiny Tapeout Workshop Supercon 25 251
431 Fastest Finger First 252
448 Brainf*ck ASIC 254
450 VGA Screensaver with embedded bitmap ROM 257
452 My (S)VGA Playground 259
454 ASAP CPU v1 260
456 First Wokwi design 262
458 Tiny Tapeout Wokwi Bsp 263
460 PWM_LED_dimmer 264
462 tt_classic_vga_clock_by_TT 265
481 kitt_scanner 267
482 SIC-1 8-bit SUBLEQ Single Instruction Computer 269
483 Classic 8-bit era Programmable Sound Generator AY-3-8913 273
484 8 Bit SAR ADC 281
485 Audio Transient Shaper 283
486 OnChip - Bandgap Reference 286
487 Baud-Rate PAM4 Clock and Data Recovery (CDR) 288
489 DreiMann 289
490 Time Domain ADC 291
491 JK flip-flop 294
493 60/50Hz AC clock 295
494 Tiny Whisper 298
Artwork SkullFET render (MPW-4) 300
495 Noise shaping delta-sigma DAC 301
514 FemtoRV register file test 305
516 Analog 8 bit 3.3v R2R DAC 309
518 TT25 3B DAC and RingO 311

520 3V3 Opamp and DRAM cell 312
526 mini mosbius 314
544 myTTproject 316
545 7-Segment-Animations 317
546 Tiny Calculator 318
547 instrumented_ring_oscillator_two 320
548 bytebeat 323
549 Flip the Flop 324
550 6bit ALU 325
551 4-bit CLA 328
552 FSM 329
553 Tiny Pong 332
554 PWM Generator 333
555 Negative Nature #4 336
556 lime_cic_filter 337
557 Negative Nature #3 341
558 tinytapeout test gates 342
559 Negative Nature #2 343
576 TTSKY25B Register File Validation 344
578 rc_servo_motor_xy 352
580 LEDs Racer 354
582 VGA Screensaver with Tiny Tapeout Logo 355
584 Tiny Shader v2 357
586 TTSKY25b-RISCV-Core 364
588 VGA Screensaver with Zero to ASIC Logo 367
590 Verilog Meetup Template Project Example 370
608 Electronic Voting Machine 371

609 tt-wokwi 372
610 AXIS_UART_TTO 373
611 Numbers 375
612 Numerically Controlled Oscillator 376
Artwork VGA clock render 378
613 Siebensegment_Chip 379
614 Morse Code Detector 380
615 Kleine Tapeout Vorlagen-Kopie 382
616 Tiny Hardware Authentication Engine 383
617 Tiny Tapeout meli 387
618 Binary Counter 388
619 Template 389
620 Manchester Encode/Decode with built-in 5X PLL 390
621 Tiny tapeout test 392
622 Negative Nature #1 393
623 Tiny Tapeout Template Hammer 394
642 SimProc (Simple Processor) 395
648 megabytebeat 396
654 CEJMU WSPR Encoder 398
672 JKU Matt Venn workshop simple project 400
673 Secret Password 401
674 Test 402
675 Four-bit adder tiny tapeout 403
676 4 bit adder 404
677 Tiny Takeout Test Gates 405
678 TinyTapeoutTest 406
679 not a dumpster fire 407

680 Digital_gate 408
681 fdfs 409
682 in-81004-1 silicon 410
683 Tiny Tapeout Test Gates 411
684 Arjun's First Template Copy 412
685 QEI (Quadrature Encoder Interface) 413
686 TestGDS 415
687 Tiny Turing Machine 416
708 TinyQV Risc-V SoC 421
712 2048 sliding tile puzzle game (VGA) 425
716 TV-B-Gone-EU (ROM Macro variant) 427
718 Cynergy4MIE SPI Slave Component 429
736 Tiny Tapeout Template Copy 435
Artwork VGA clock render 436
737 Tiny Tapeout Template Project 437
738 TinyBF 438
739 Tiny Tapeout Binary to 7SD 443
740 WokWi Test 444
741 Anushka and Tina Wokwi 445
742 Wokwi Template 446
743 Tahmid and Sunny's First Wokwi Design 447
744 ShilpaTinytapeout 448
745 RHYTHM LOOPER and RANDOMIZER 9000 449
746 PWM_selector 452
747 SKY130 technology characterization 454
748 WokwiPWM 455
749 Bouncing squares 458

750 TicTacToe 459
751 MORSEASCII 460
774 YM2413 FM synthesis audio chip 461
782 PSM_asic 462
800 tt-tinytone 463
801 Tiny RAM DFF 2r1w 464
802 FIR Filter 467
803 NCO 468
804 hardcoded sequencer 470
805 Dual-Channel PWM with SPI Control + Extra Test Logic 471
806 Mastermind 473
807 test 474
808 React Chip 475
809 cirucuit 476
810 RagnarMegaRaknare 477
811 Tiny Tapeout Test Gates 478
812 Uart and Temperature sensor 480
813 Tiny Tapeouts Chip Design 482
814 vga_projekt_ed_nkpng 483
815 xorshift 484
838 KianV RISC-V RV32E Baremetal SoC 485
844 SPELL 487
Artwork TT06 IC - decapped 492
846 VGA Pride 493
864 JulkeGussinatorn 2.0 497
865 I2C Slave 498
866 7-Segment Counter 500

867 EZ Reservoir 502
868 Andrew chip design 505
869 Morse Code Detector (With Serial RX) 506
870 Counter 508
871 dyno-tt 509
872 Anton's-Hardware-Hack 510
873 4 hour CPU 511
874 SignalSorter 512
875 Flappy VGA 513
876 The Adder 515
877 LFSR Driven Cryptography 516
878 Tiny Tapeout Hradware Workshop Vaibhav 519
879 Verilog Multistage Oscillator with Enable and Counter 521
897 Template copy 523
899 Auberger tiny tapeout 524
901 7-segment-games 525
903 Paircheck 526
905 Chip 527
907 Lauflicht_HTL_Leonding 528
909 My first Wokwi project 529
910 KianV uLinux SoC 531
911 TimosChip 534
961 Tiny Tapeout 535
963 Tiny Tapeout Test 536
965 Traffic Light Controller 537
967 Piffl first Wokwi project 540
969 8-Bit ALU 541

970 KianV uLinux RISC-V regfile edition 542
971 simple adder and subtractor 545
973 The Ultimate ASIC 546
974 TinyTapeStation 547
Artwork Oscillating Bones 549
975 Test project 550

Pinout 551

The Tiny Tapeout Multiplexer 552
Overview 552
Operation 552
Pinout 555

Special Thanks 557

Team 558

Using This Datasheet 559
Structure 559
Badges 559
Callouts 560
Figures & Footnotes 560
Updates 560

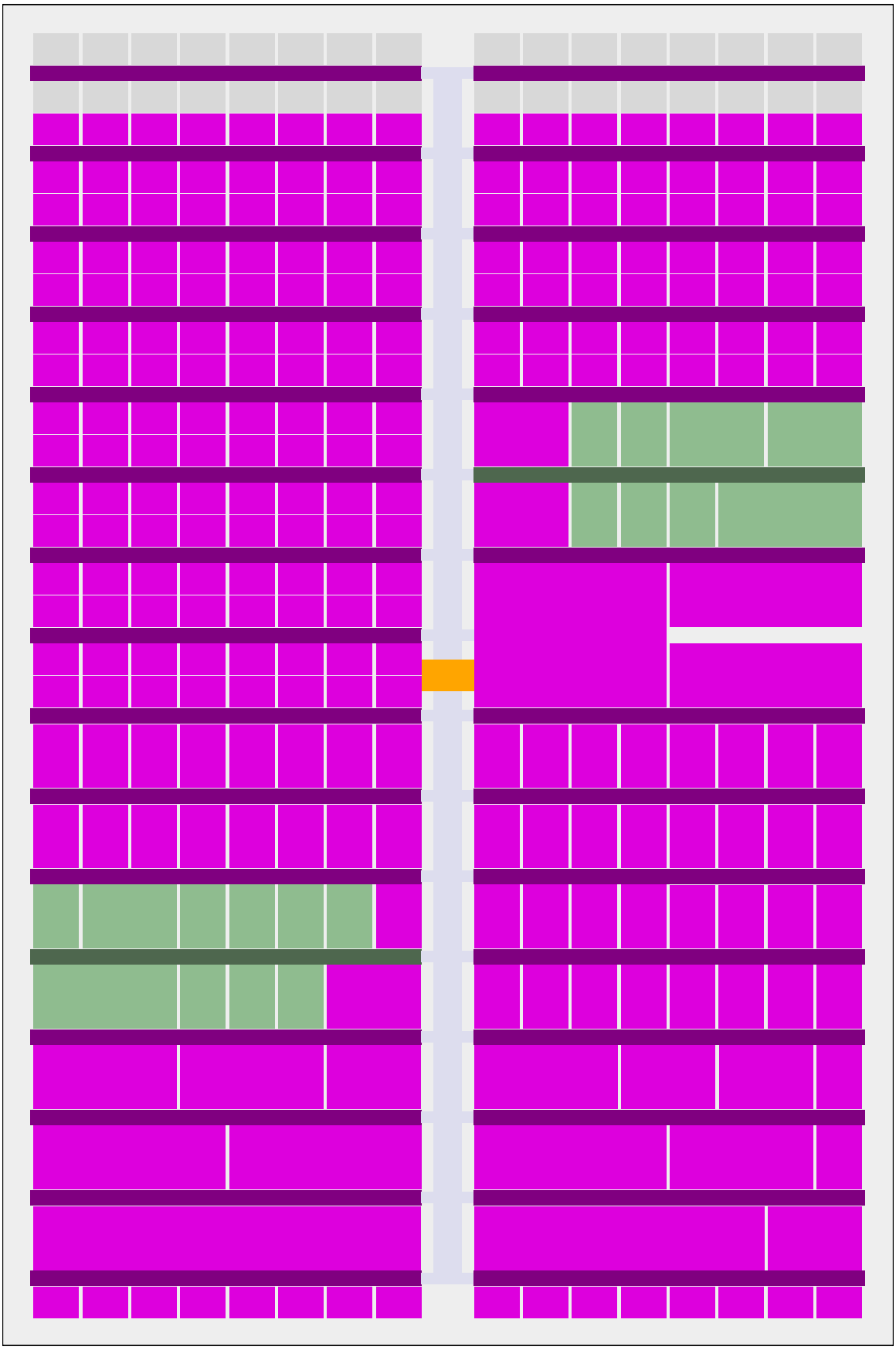
Where is *your* design? 561

Chip Renders

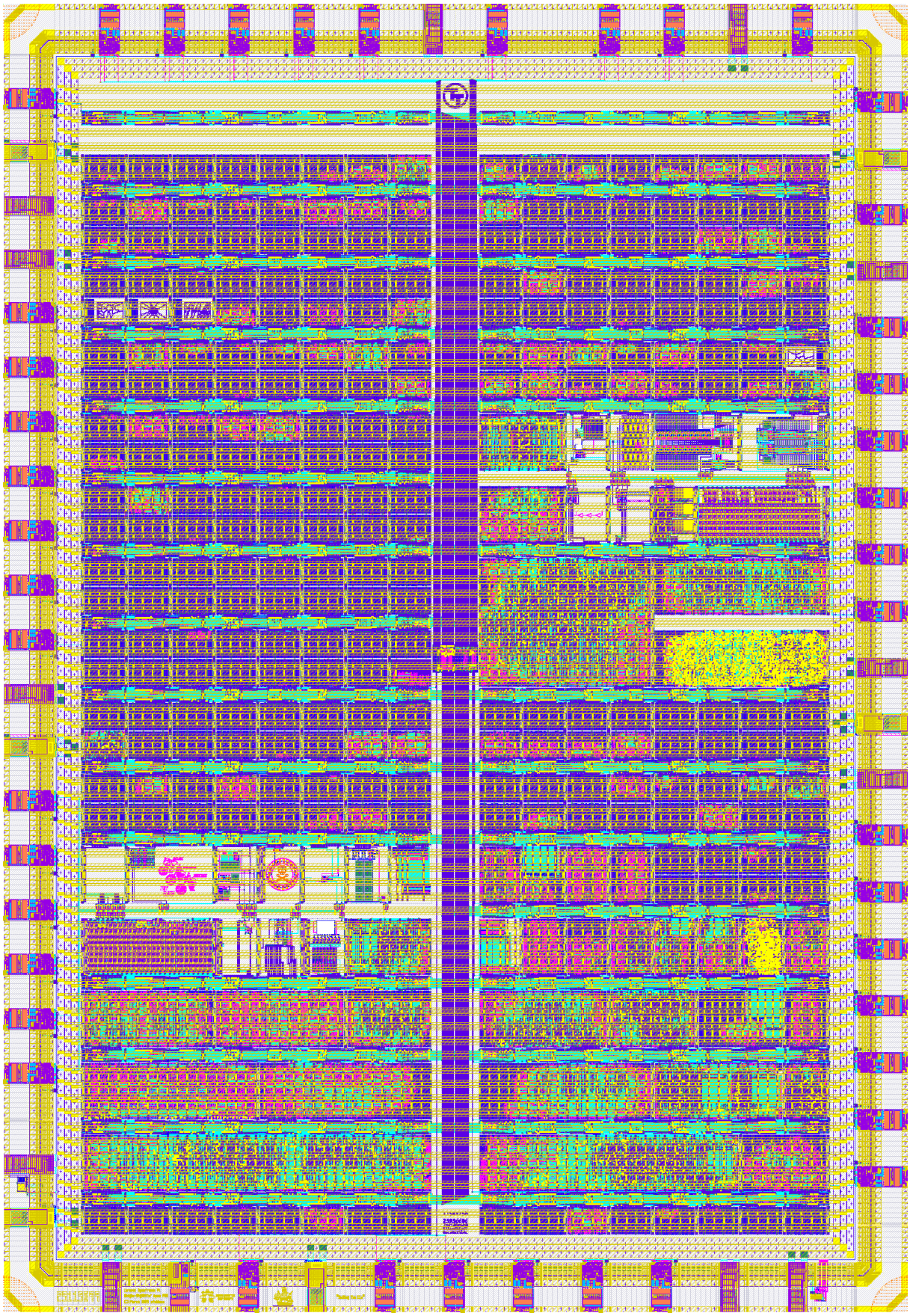
Online chip viewer

tinytapeout.com/
decap/ttsky25b

Shuttle Map

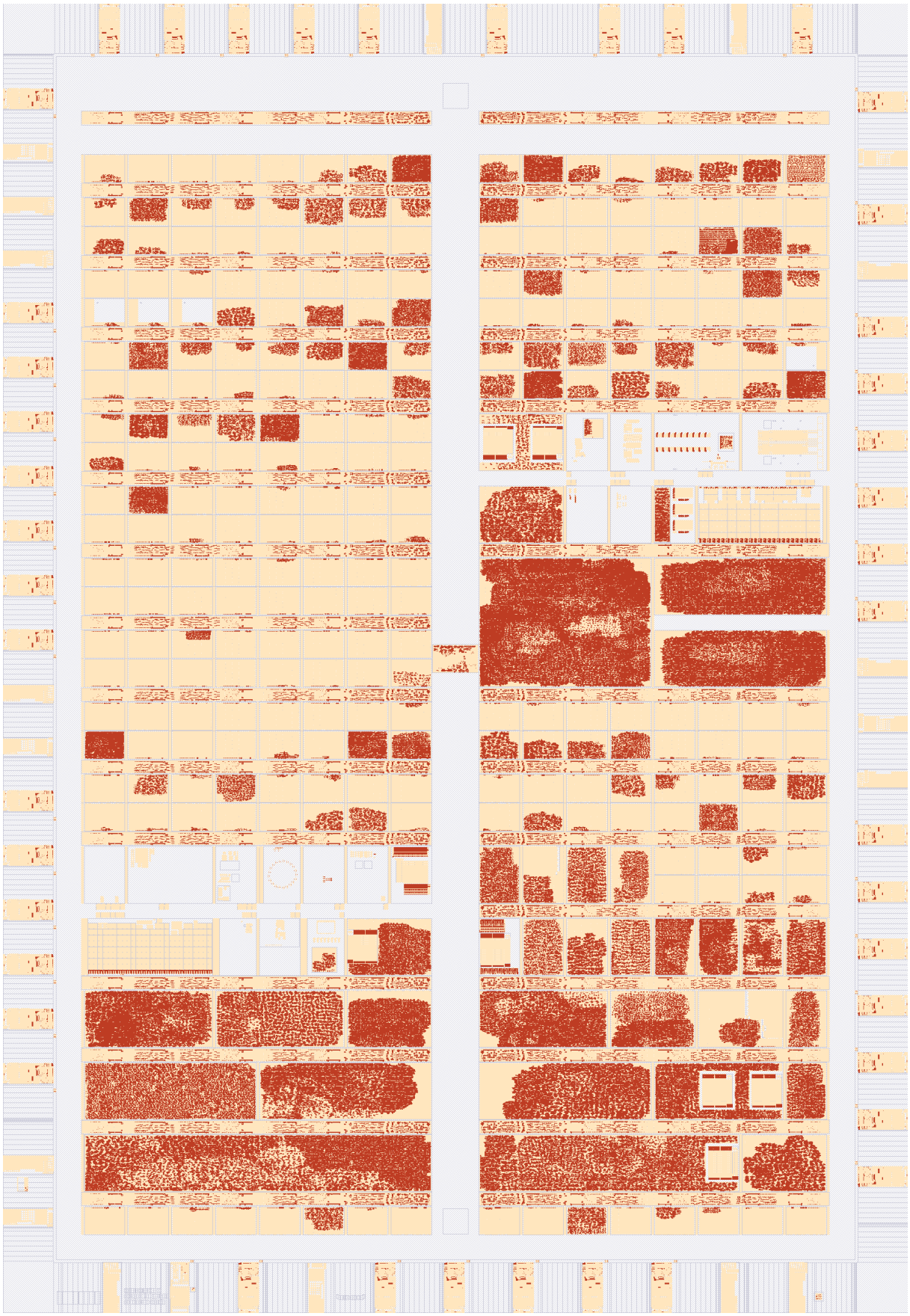


GDS



Logic Density

Local Interconnect Layer



Projects

Chip ROM

by **Uri Shaked**

0000 HDL Project

`github.com/TinyTapeout/tt-chip-rom`

"ROM with information about the chip"

How it works

ROM memory that contains information about the Tiny Tapeout chip. The ROM is 8-bit wide and 256 bytes long.

The ROM layout

The ROM layout is as follows:

Address	Length	Encoding	Description
0	8	7-segment	Shuttle name (e.g. "tt07"), null-padded
8	8	7-segment	Git commit hash
32	96	ASCII	Chip descriptor (see below)
248	4	binary	Magic value: `"TT\xFA\xBB"`
252	4	binary	CRC32 of the ROM contents, little-endian

The chip descriptor

The chip descriptor is a simple null-terminated string that describes the chip. Each line is a key-value pair, separated by an equals sign. It contains the following keys:

Key	Description	Example value
shuttle	The identifier of the shuttle	tt07
repo	The name of the repository	TinyTapeout/tinytapeout-07
commit	The commit hash *	a1b2c3d4

* The commit hash is only included for Tiny Tapeout 5 and later.

Here is a complete example of a chip descriptor:

```
shuttle=tt07
repo=TinyTapeout/tinytapeout-07
commit=a1b2c3d4
```

How the ROM is generated

The ROM is automatically generated by tt-support-tools while building the final GDS file of the chip. Look at the `rom.py` file in the repository for more details.

Reading the ROM

There are two ways to address ROM, depending on the value of the `rst_n` pin:

1. When `rst_n` is high: Set the `ui_in` pins to the desired address.
2. When `rst_n` is low: Toggle the `clk` pin to read the ROM contents sequentially, starting from address 0.

In both cases, the ROM data for the selected address will be available on the `uo_out` pins, one byte at a time.

How to test

The first 16 bytes of the ROM are 7-segment encoded and contain the shuttle name and commit hash. You can dump them by holding `rst_n` low and toggling the `clk` pin, and observing the on-board 7-segment display.

Alternatively, you can keep `rst_n` high and set the `ui_in` pins to the desired address using the first four on-board DIP switches, while observing the on-board 7-segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	addr[0]	data[0]	—
1	addr[1]	data[1]	—
2	addr[2]	data[2]	—
3	addr[3]	data[3]	—
4	addr[4]	data[4]	—
5	addr[5]	data[5]	—
6	addr[6]	data[6]	—
7	addr[7]	data[7]	—

Tiny Tapeout Factory Test

by **Tiny Tapeout**

0001 HDL Project

`github.com/TinyTapeout/ttsky25b-factory-test`

"Factory test module"

How it works

The factory test module is a simple module that can be used to test all the I/O pins of the ASIC.

It has three modes of operation:

1. Mirroring the input pins to the output pins (when `rst_n` is low).
2. Mirroring the bidirectional pins to the output pins (when `rst_n` is high `sel` is low).
3. Outputing a counter on the output pins and the bidirectional pins (when `rst_n` is high and `sel` is high).

The following table summarizes the modes:

`rst_n`	`sel`	Mode	uo_out value	uio pins
0	X	Input mirror	ui_in	High-Z
1	0	Bidirectional mirror	uio_in	High-Z
1	1	Counter	counter	counter

The counter is an 8-bit counter that increments on every clock cycle, and resets when `rst_n` is low.

How to test

1. Set `rst_n` low and observe that the input pins (`ui_in`) are output on the output pins (`uo_out`).
2. Set `rst_n` high and `sel` low and observe that the bidirectional pins (`uio_in`) are output on the output pins (`uo_out`).
3. Set `sel` high and observe that the counter is output on both the output pins (`uo_out`) and the bidirectional pins (`uio`).

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	sel / in_a[0]	output[0] / counter[0]	in_b[0] / counter[0]
1	in_a[1]	output[1] / counter[1]	in_b[1] / counter[1]
2	in_a[2]	output[2] / counter[2]	in_b[2] / counter[2]
3	in_a[3]	output[3] / counter[3]	in_b[3] / counter[3]
4	in_a[4]	output[4] / counter[4]	in_b[4] / counter[4]
5	in_a[5]	output[5] / counter[5]	in_b[5] / counter[5]
6	in_a[6]	output[6] / counter[6]	in_b[6] / counter[6]
7	in_a[7]	output[7] / counter[7]	in_b[7] / counter[7]

Tiny Tapeout Hackathon UofT Day 2 Template

by **Joshua Shao**

0002 Wokwi Project

github.com/joshuashao8/TinyTapeout
wokwi.com/projects/445256090314494977

"A really basic chip that was made in a Hackathon event!"

How it works

Has only or & and gates. Meant to be a demo test for simple logic gates.

How to test

Turn A & B & C on & off.

External hardware

Seven Segment Display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output or	—
2	input c	—	—
3	input d	—	—
4	input e	—	—
5	input f	—	—
6	input g	—	—
7	input h	—	—

Test

by **Zain khalaf**

0003 10 kHz Wokwi Project

`github.com/zain935/zazain`

`wokwi.com/projects/445163656445179905`

"an end gate with two inputs and one output"

How it works

My project does AND on inputs 1 and 1.

How to test

If output is one only when inputs 1 and 2 are both one, we're good.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in1	out1	—
1	in2	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Fawwaz's Project

by **Fawwaz Mir**

0004 | 10 kHz | Wokwi Project

`github.com/notclxpz/Project-1`

`wokwi.com/projects/445171420903985153`

"Test some logic gates with clock divider"

How it works

output 0 uses OR logic via input 1 and input 2 output 1 uses AND logic via input 2 and input 3 output 2 uses XOR logic via input 4 and input 5 output 3 uses NAND logic via input 5 and input 6

How to test

Start the simulation and press the on and off switches to play around with the logics.

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output OR	—
1	input b	output AND	—
2	input c	output XOR	—
3	input d	output NAND	—
4	input e	—	—
5	input f	—	—
6	input g	—	—
7	input h	—	—

DAMTinyTape

by **DAM**

0005 Wokwi Project

`github.com/shakurrffffff/workshoptinytapeoutuw`

`wokwi.com/projects/445164514940747777`

"Chip Design Workshop"

How it works

Adjust the clock for it to say a letter Explain how your project works

How to test

Test it Explain how to use your project

External hardware

None List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	1	1	—
1	1	1	—
2	1	1	—
3	1	1	—
4	1	1	—
5	1	1	—
6	1	1	—
7	1	1	—

Bob design

by **Bobby Allen**

0006 Wokwi Project

`github.com/Hyperbalistic/GDS`

`wokwi.com/projects/445163677488010241`

"bin to dec converter"

How it works

Uses And Gates to map out decimal values with complimentry inputs.

How to test

Input Binary values, will output values 0 - 7 converting from base 2 to base 10 values

External hardware

7 Segment display, and a set of 4 switches

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	OUT0	—
1	—	OUT1	—
2	IN0	OUT2	—
3	IN1	OUT3	—
4	IN3	OUT4	—
5	IN4	OUT5	—
6	—	OUT6	—
7	—	OUT7	—

doritos

by **Aiden Lim**

0007 Wokwi Project

`github.com/Aiden-1108/Tiny-tapeout`

`wokwi.com/projects/445163940815949825`

"Testing Logic Gates"

How it works

This project demonstrates a simple 4-bit binary counter system using digital logic components, a 7-segment display, and a manual clock input. The core functionality revolves around counting in binary and displaying the decimal equivalent on a 7-segment display.

How to test

Power the Circuit: Ensure all components are connected to VCC and GND appropriately.

Reset the Counter:

Press the RESET button to initialize the counter to 0.

Step Through the Count:

Press the Step button to increment the counter.

The 7-segment display will show numbers from 0 to 9 as you continue pressing.

After 9, the counter wraps back to 0 (assuming it's a BCD counter). ## External hardware

Push Buttons (x2):

One for manual clock ("Step"), one for reset.

10kΩ Resistor:

Used as a pull-down resistor on the Reset button.

4-bit Binary Counter (possibly a 74LS90 or similar):

For generating the binary count.

AND Gate:

Used to control the clock signal.

BCD to 7-Segment Decoder (e.g., 74LS47 or similar):

Converts binary to segment control signals.

7-Segment Display (Common Cathode):

Displays decimal digits (0–9).

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	nand1:A	nand1:OUT	—
1	nand1:B	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny_Tapeout_Test_Gates

by **Prakhar_Anant**

0008 10 kHz Wokwi Project

`github.com/s-anant/tinytapout-`
`wokwi.com/projects/445172860504617985`

"Tiny Tapeout Design"

How it works

First we take input from the switches, than it goes to the input, throught the and logic gate, finally going through one end of the LED. ## How to test Switch on pins 1 and 3. ## External hardware

External hardware used in our project is LED.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	input pin for and logic gate	output from and logic gate to LED	—
2	—	—	—
3	input pin for and logic gate	output from and logic gate to LED	—
4	just passing through	passing through in order to light the number eight	—
5	just passing through	passing through in order to light the number eight	—
6	just passing through	passing through in order to light the number eight	—
7	just passing through	passing through in order to light the number eight	—

Leena and Gabby's coding masterpiece

by **Leena** & **Gabby**

0009 | 10 kHz | Wokwi Project

`github.com/GabriellaS11/Tiny-Takeout`

`wokwi.com/projects/445163607540610049`

"Test some logic gates and a clock divider"

How it works

sends electic pulses to flash light from input a to input b ## How to test Test the connections with output AND ## External hardware

N/A

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output nand	—
2	—	output or	—
3	—	output clock div 16	—
4	—	output clock div 12	—
5	—	—	—
6	—	—	—
7	—	—	—

EJ Wokwi proj.

by **Jimin** & **Esther**

0010 | 1 kHz | Wokwi Project

`github.com/SweetDynamite/Esther-x-Jimin-PROJ.`

`wokwi.com/projects/445173979376212993`

"Shows 7 on the LED display no matter which switch is on. (I'mma kms)"

How it works

All of the outputs are connected to the leds that composes number 7 shape.

How to test

Press "start", and turn on any switch-any swith-just at least one. Then the number 7 will always appear on the LED display.

External hardware

List external hardware used in your project

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	i0	o0	io0
1	i1	o1	io1
2	i2	o2	io2
3	i3	o3	io3
4	i4	o4	io4
5	i5	o5	io5
6	i6	o6	io6
7	i7	o7	io7

Minhala's tinytapeout

by **Minha** & **Hala**

0011 Wokwi Project

`github.com/Minha-1/Minhala`

`wokwi.com/projects/445163652263454721`

"Display's one letter of the alphabet"

How it works

2 AND gates are attached to input 1 and input 0 The AND gates were attached to output 1 and 2

How to test

Turn all the switches on and play, the 7-segment display will flash and "H"

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output and	—
2	—	–	—
3	—	–	—
4	—	OUT4	—
5	—	OUT5	—
6	—	OUT6	—
7	—	OUT7	—

AnalogClockTest1

by **Ahaan** & **Bruce**

0012 Wokwi Project

github.com/ahaansethi12/Ahaan---Bruce-s-GDS

wokwi.com/projects/445163790648286209

"Analog Clock with switches... I think"

How it works

"Flipping the switches turns the lights on the analog clock off or on." ## How to test

"Flip the switch to turn on the lights." ## External hardware

"Dip-switch-8, seven-segment display, and NAND gate"

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN1 for OUT0	OUT0 for SEVSEG1:A	—
1	IN0 for OUT0	OUT1 for SEVSEG1:B	—
2	IN2 for OUT2	OUT2 for SEVSEG1:C	—
3	IN3 for OUT3	OUT3 for SEVSEG1:D	—
4	IN4 for OUT4	OUT4 for SEVSEG1:E	—
5	IN5 for OUT5	OUT5 for SEVSEG1:F	—
6	IN6 for OUT6	OUT6 for SEVSEG1:G	—
7	IN7 for OUT7	OUT7 for SEVSEG1:DP	—

Simon Memory Game

by **Erfan Nezhad Ahmadi** & **Joshua Pretti**

0013 Wokwi Project

github.com/erfannezhad08/UW-Chip-Design

wokwi.com/projects/445163742018450433

"Memorize and repeat the order of colours"

How it works

Placeholder

How to test

Placeholder

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a AND	AND	1
1	input b AND	—	2
2	—	—	3
3	—	—	4
4	input c	a	5
5	input d	b	6
6	input e	c	7
7	input f	d	8

Tiny Tapeout Test Gates

by **Khalid Elagha**

0014 Wokwi Project

github.com/KhalidElagha/Khalid

wokwi.com/projects/445163874814362625

"A clock that displays the time"

How it works

Our clock will be digital and we will use gates to allow the clock to change from right to left with binary ## How to testt

just run the program and the clock will start from 12:00 am ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any led display to show the digital time

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	and1a	0	—
1	and1b	1	—
2	—	2	—
3	out7	3	—
4	out4	4	—
5	—	5	—
6	out1	6	—
7	out3	7	—

tiny tapeout

by **alex** & **sofia**

0015 Wokwi Project

`github.com/zhana8592/tinytapeout`
`wokwi.com/projects/445163803460843521`

"simple logic gate test"

How it works

switch 1,2,3,4 to test out combinations, lights up clock stuff yay

How to test

use switches!

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	0	0	—
1	1	1	—
2	2	2	—
3	3	3	—
4	—	4	—
5	—	5	—
6	—	6	—
7	—	7	—

WOKWI Design - Joelle

by **Joelle Kassir**

0032 Wokwi Project

`github.com/joellehk/WOKWI-Workshop-Design---Joelle`
`wokwi.com/projects/445254767711795201`

"Displays the letter 'J'."

How it works

The NOT, NAND, and OR gates display is the letter 'J'.

How to test

Switch 1, 4, 5, 6, and 8 on (have 2, 3, and 7 off).

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	on	output to 7-segment display	—
1	off	no output to 7-segment display	—
2	off	no output to 7-segment display	—
3	on	output to 7-segment display	—
4	on	output to 7-segment display	—
5	on	output to 7-segment display	—
6	off	no output to 7-segment display	—
7	on	output to 7-segment display	—

WokwiDesign

by **Sasha.S.**

0033 Wokwi Project

`github.com/sallythesalmon/design`

`wokwi.com/projects/445265377104476161`

"Chip Design Using Logic Gates"

How it works

1 AND connected to input 0&1, OR connected to input 2&3, 2 AND connected to input 4&5

How to test

Flip on the off switches rsponsible for each gate

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a of 1 AND gate	output a of 1 AND gate	—
1	input b of 1 AND gate	output b of 1 AND gate	—
2	input a of 1 OR gate	output a of 1 OR gate	—
3	input b of 1 OR gate	output b of 1 OR gate	—
4	input a of 2 AND gate	output a and b of 2 AND gate	—
5	input b of 2 AND gate	output a and b of 2 AND gate	—
6	input 6	no output	—
7	input 7 wire	output 7 wire	—

Semi-Conductor Design

by **Abdulbasit Farhan**

0034 Wokwi Project

`github.com/PEAWEUDS/TinyTapeoutWorkshop`

`wokwi.com/projects/445255067501843457`

"8 segment display that will light up certain areas depending on the switch turned on"

How it works

A 8 segment display that is controlled by 8 total switches that go through 4 AND gates. Out of those 4 AND gates, The top 2 and bottom 2 AND gates are attached to a OR gate that each controls 4 segments of the display.

How to test

Its an AND and OR basic system.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input 0	input or	—
1	input 1	input or	—
2	input 2	input or	—
3	input 3	input or	—
4	input 4	input or	—
5	input 5	input or	—
6	input 6	input or	—
7	input 7	input or	—

gds

by **Victoria**

0035 Wokwi Project

`github.com/victoriaponcas/tinytapeout`
`wokwi.com/projects/445255496359022593`

"gds creation"

How it works

Figure it out

How to test

I dont know yet

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	a	1
1	a	b	2
2	a	c	3
3	a	d	4
4	a	e	5
5	a	f	6
6	a	g	7
7	a	h	8

chipchip

by **Yahya Wazeer**

0036 Wokwi Project

`github.com/yaya311/555-semiconductor-`
`wokwi.com/projects/445255110472500225`

"lights up the light based on switches you flip?"

How it works

i dunno ## How to test

i dunno

External hardware

i dunno

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN1	OUT1	OUT1
1	IN2	OUT2	OUT2
2	IN3	OUT3	OUT3
3	IN4	OUT4	OUT4
4	IN5	OUT5	OUT5
5	IN6	OUT6	OUT6
6	IN7	OUT7	OUT7
7	IN8	OUT8	OUT8

Hardware Hackday Test Gates

by **Aaron Tran**

0037 Wokwi Project

`github.com/aarontran321/Hardware-Hackdays-Chip-Contruction-`
`wokwi.com/projects/445254867374774273`

"Test some logic gates and a clock divider"

How it works

I used a NOT Gate on input 1 to output 1 and an AND Gate on input 5&6 to Output 5

How to test

To test it, run the simulation and have the switches turned in this sequential order. On, Off, On, Off, Off, On, Off, Off.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	Not gate	output NOT	—
2	—	—	—
3	—	—	—
4	—	—	—
5	and1: A	and1: output	—
6	and1: b	—	—
7	—	—	—

Tiny Tapeout Gates

by **Rohan Chopra**

0038 | 10 kHz | Wokwi Project

`github.com/rohannn-09/tiny-tapeout`

`wokwi.com/projects/445254813154997249`

"Uses AND, OR, NAND and NOT gates to output when conditions are met."

How it works

Input a and b are connect to the AND gate, input c and d are connected to the NAND gate, input e and f are connected to the OR gate and input g is connect to the NOT gate.

How to test

INPUT a AND b | output | INPUT e OR f | output | 0 0 | 0 | 0 0 | 0 | 1 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1 1 | 1 | 1 1 | 1 |

INPUT c NAND d | output | INPUT g | ouput | 0 0 | 1 | 0 | 0 | 1 0 | 1 | 1 | 0 | 0 1 | 1 | 1 1 | 0 |

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output and	—
2	input c	output nand	—
3	input d	output nand	—
4	input e	output or	—
5	input f	output or	—
6	input g	output not	—
7	input h	output h	—

Tinytapeoutchipnikki

by **Niharika Agarwal**

0039 Wokwi Project

`github.com/nikkiled/Tinytapeoutchipnikki`

`wokwi.com/projects/445254908700218369`

"Test some logic gates and a clock divider"

How it works

OR, AND and NAND gates were used to connect to input 0, input 1, input 2 and input 3

How to test

Set the inputs and check the outputs with the On and Off buttons

External hardware

N/A

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	input b	output b	—
2	input c	output c	—
3	—	output and	—
4	—	output nand	—
5	—	—	—
6	—	—	—
7	—	—	—

Overly Complicated Light Switch

by **Mateo Pena**

0040 Wokwi Project

`github.com/ilpomato/WorkshopComputerChip`

`wokwi.com/projects/445255383410583553`

"Sends out signals to all output ports if the proper inputs are added."

How it works

Sends out a signal to all output ports when:

- The four pairs of inputs each pass an XOR gate,
- The outputs of those XOR gates pass two AND gates,
- The outputs of those AND gates pass an XNOR gate

How to test

Set all inputs to 0, or set one input in each pair as a 1 and the other as a 0, then check if the outputs are powered in both scenarios.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output xor and xnor	—
1	input b	output xor and xnor	—
2	input c	output xor and xnor	—
3	input d	output xor and xnor	—
4	input e	output xor and xnor	—
5	input f	output xor and xnor	—
6	input g	output xor and xnor	—
7	input h	output xor and xnor	—

Inverting Numbers

by **Wajeeh Abdulwahab**

0041 Wokwi Project

github.com/Eggman-Wajeeh/Wajeehs-Tiny-Tapeout

wokwi.com/projects/445254976953093121

"Outputs 1 ligh segment for each switch"

How it works

inputs 1-8 is connected to input 1-7. Each inverter inverts the input and activates the corresponding light on the clock ## How to test

set the inputs and check if they turn the corresponding light ## External hardware

Clock

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input_0	output_inverter1	—
1	input_1	output_inverter2	—
2	input_2	output_inverter3	—
3	input_3	output_inverter4	—
4	input_4	output_inverter5	—
5	input_5	output_inverter6	—
6	input_6	output_inverter7	—
7	input_7	output_inverter8	—

Single Digit Sums by Mark Angelov

by **Mark Angelov**

0042 Wokwi Project

`github.com/Mark-Angelov/Single-Digit-Sums`

`wokwi.com/projects/445254731311517697`

"This project displays the one-digit sums created by the possible number inputs."

How it works

Turn a switch on to add to the total, which would then be displayed on the 7-segment display. Sums can only have a total of one digit; for example, switching 0 and 7 on will output 9, while switching 1 and 7 on will output nothing because it makes 10, which is not a single-digit sum.

How to test

Set the inputs to add to a number from 0-9 and check if the output shows the correct number.

External hardware

Requires a single 7-segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	adds 1 to the sum	top segment of the display	—
1	adds 2 to the sum	right top segment of the display	—
2	adds 3 to the sum	right bottom segment of the display	—
3	adds 4 to the sum	bottom segment of the display	—
4	adds 5 to the sum	left bottom segment of the display	—
5	adds 6 to the sum	left top segment of the display	—
6	adds 7 to the sum	middle segment of the display	—
7	adds 8 to the sum	dot segment of the display	—

Tiny Tapeout Test Gates

by **Shivam Beepath**

0043 | 10 kHz | Wokwi Project

github.com/shivambeepath021/wokwi-chip

wokwi.com/projects/445254980988014593

"Test some logic gates and a clock driver"

How it works

The circuit uses logic gates (AND, NAND, and OR) to process two binary inputs. The inputs are set using the DIP switch, and the outputs are displayed through a 7-segment display.

A clock (triggered by the Step button) sends pulses to the system. Each press advances the logic state. A Reset button clears the circuit to its initial state.

The Input block receives 8 input bits (from the DIP switch). The Output block shows the results of logic operations on the display. The 7-segment display represents the logic result in visual numeric form.

How to test

Press RESET to clear the circuit.

Set the DIP switch inputs (e.g., IN0 and IN1 for a and b).

Press STEP to clock the circuit and update the display.

Observe the 7-segment output or output pins.

Compare the results with the expected logic table below:

Input A	Input B	Output AND	Output NAND	Output OR
0	0	0	1	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

Optionally, connect a clock source (e.g., 10 kHz) to the CLK pin to automate stepping. You should see the outputs (and segments of the display) flashing at different rates depending on the logic combination.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Clock (Step button)	Output 0 (to 7-segment display segment a)	Not connected
1	Reset (active LOW)	Output 1 (to 7-segment display segment b)	Not connected
2	IN0 from DIP switch	Output 2 (to 7-segment display segment c)	Not connected
3	IN1 from DIP switch	Output 3 (to 7-segment display segment d)	Not connected
4	IN2 from DIP switch	Output 4 (to 7-segment display segment e)	Not connected
5	IN3 from DIP switch	Output 5 (to 7-segment display segment f)	Not connected
6	IN4 from DIP switch	Output 6 (to 7-segment display segment g)	Not connected
7	IN5–IN7 from DIP switch (unused in logic)	Output 7 (extra/unused output)	Not connected

Tiny tapeout test gates

by **Aaron Hui**

0044 | 10 kHz | Wokwi Project

github.com/itsaaronhui/tiny-tapeout

wokwi.com/projects/445256131623154689

"Prints out the letter A"

How it works

This design implements basic digital logic gates — NOT, NOR, and NAND — to produce an output pattern forming the ASCII representation of the letter A. When an input signal is applied to ui[0], the logic chain processes the signal through the configured NOT, NOR, and NAND gates. The resulting high/low output levels on uo[0] and uo[1] correspond to the predefined bit pattern that, when observed on a connected LED array or logic analyzer, visually represents the letter A.

How to test

Connect the Tiny Tapeout board to power and clock (10 kHz). Apply a logic high (1) or low (0) to the input pin ui[0]. Observe the output signals on uo[0] (NOR) and uo[1] (NAND). If connected to an LED matrix or viewer, the pattern displayed should correspond to the letter A. Optionally, use a logic analyzer to verify the expected high/low logic states on each output.

External hardware

None needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input not	output nor	—
1	—	output nand	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Arjun's 1st Wokwi design

by **Arjun Wadhawan**

0045 Wokwi Project

github.com/wwhhyy100/1st-Wokwi-design

wokwi.com/projects/445254714007910401

"Turn on or off the switches to view different parts of the clock light up"

How it works

Explain how your project works

This project has an analog clock which each individual light will light up depending on which number is switched and in what order

How to test

Explain how to use your project

Switch the numbered switchs on or off and the lights on the analog clock will light up depending on what numbers you switched on

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Seven segment LED display (analog clock)

Set of 8 electrical switches in a single package

12mm Tactile Switch Button (momentary push button). x2

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	IN7	OUT7	—

flipper 1 to 5

by **Alexander Suk**

0046 Wokwi Project

`github.com/Asook15/Asook-GDS`

`wokwi.com/projects/445254767859645441`

"flipping the second switch changes the number from 1 to 5 and vice versa"

How it works

Flipping the second switch on the dip switch 8 changes the output of the number from 1 to 5 and vice versa

How to test

ensure the chip is powered, then flip the second switch on the dip switch 8
External hardware

seven-segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	AND gate, takes from in 1 and in 2	AND gate, goes to out 1	—
1	NAND gate, takes form in 1 and in 3	NAND gate, goes to out 0,3,5,6	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Johans Wokwi design

by **Johan Praveen**

0047 Wokwi Project

github.com/johantpraveen-prog/johanwokwibuild

wokwi.com/projects/445254845860088833

"my project connects the chip to the lghts so i can create whatever i want with the right switches."

How it works

it works with the not gates, electricity, the switches, the and gates, and lots of other sciences

How to test

So you turn on and turn off the switches to turn on and off various lines. then to turn on or turn off the button you can turn on and turn off the big switch with the blue lines. the first switch on the row of switches on the side turn on and turns of the top one, the second one does the top right one, the third one does the bottom right one, the fourth one does the bottom one, the fifth and sixth one together does the bottom left line, the seventh one does the top left one and the eight and last one does the middle line.

External hardware

resistor, pushbottons, a seven segment display, and a slide switch.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	0 it oes to a not gate	1 it goes to the top most line in the eight	—
1	0 it oes to a not gate	1 it goes to the top right line	—
2	0 it oes to a not gate	1 it goes to the the bottom right line	—
3	0 it oes to a not gate	1 it goes to the bottom line	—
4	1 goes to an and gate	1 goes to the bottom left line	—
5	1 goes to an and gate	1 goes to the top left line	—
6	1 its just a straight wire	1 goes to the middle line	—

#	Input	Output	Bidirectional
7	0 it goes to a not gate	1 goes to the dot	—

Chip Design (Working title)

by **Akshay** & **Osman**

0065 Wokwi Project

github.com/ItchyMiner/Chip-Design
wokwi.com/projects/445163922612663297

"TBD"

How it works

TBD

How to test

TBD

External hardware

TBD

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output nand	—
2	input c	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

FlippyFlop

by **Joshua Ensley** & **Maanvik Arora**

0067 Wokwi Project

github.com/MaanvikArora/TinyTapeout-Michael-Jackson

wokwi.com/projects/445171727534897153

"Flipping the flops using flipflops"

How it works

Player 1, 2, 3 and 4 are connected to NOR gates, which are connected to DSR Flip Flops, the flip flops outputs go through OR gates and back to the NOR gates such that if any of DSR Flip Flops are High the NORs output Low.

How to test

Not applicable at this point

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Button 1	Light 1	—
1	Player 2	Light 2	—
2	Player 3	Light 3	—
3	Player 4	Light 4	—
4	Reset	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Chip-Test

by **Ibrahim-L.**

0069 | 10 kHz | Wokwi Project

`github.com/lam-1tech/TInyTapreout-Chip2`

`wokwi.com/projects/445172053537213441`

"chip that does something"

How it works

and gates are used to turn on number segment thing ## How to test use multimeter on beep setting and check for connections

Explain how to use your project in 4-7 go to out 4-7 in 2-3 go to out 3 and 0 in 0-1 go to out 1 and 2 ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

number segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	EXTOUT0
1	IN1	OUT1	EXTOUT1
2	IN2	OUT2	EXTOUT2
3	IN3	OUT3	EXTOUT3
4	IN4	OUT4	EXTOUT4
5	IN5	OUT5	EXTOUT5
6	IN6	OUT6	EXTOUT6
7	IN7	OUT7	EXTOUT7

MPW-6 "Hack SoC" – Illustrated by Máximo Balestrini.

Tiniest GPU V2

by **Matt Pongsagon**

0070 | 25.175 MHz | HDL Project

`github.com/pongsagon/tt_um_pongsagon_tinygpu_v2`

"A GPU with transformation & lighting, rasterization, flat shading, double buffer, z-buffer"

How it works

TinyGPU v2.0. A standalone GPU that can display a model file from FLASH. Render 1K tri at 6.5fps in 320x240, 4-bit color.

```
Spec:
- GPU can performs transformation & lighting, rasterization
- 4-bit double buffer, 8-bit depth buffer store on QSPI RAM
- max tri 1K
- backface culling
- 1 dynamic directional light, flat shading
- use Gamepad to transform the model and rotate light
- run at 25Mhz. When fab, it will use around 200k transistor
```

How to test

Updating document, Please come back again

External hardware

- QSPI PMOD
- TinyVGA PMOD
- GamePad PMOD
- SNES controller

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	SPI latency[0]	R1	CS
1	SPI latency[1]	G1	SD0
2	SPI latency[2]	B1	SD1
3	—	VSync	SCK
4	gamepad_latch	R0	SD2
5	gamepad_clk	G0	SD3

#	Input	Output	Bidirectional
6	gamepad_data	B0	CS1
7	—	HSync	CS2

4 switch, seven segment display

by **Beckett** & **Nam**

`0071` `10 kHz` `Wokwi Project`

`github.com/AverageFemtFan/Project-1`

`wokwi.com/projects/445163628668378113`

"Uses four switches to control seven outputs, in this case to display a number"

How it works

Four switches are connected to a decoder filled with AND and OR gates that are able to manipulate 7 ouuputs

How to test

Set the inputs and check the outputs match with the expected results | A | B | C | D | Display | a | b | c | d | e | f | g | |—|—|—|—|———|—|—|—|—|—|—|—| | 0 | 0 | ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input A	output a	—
1	input B	output c	—
2	input C	—	—
3	input D	—	—
4	input H	—	—
5	—	—	—
6	—	—	—
7	—	—	—

T Flip Flop

by **Victor** & **Thomas**

0073 Wokwi Project

`github.com/BigBrain244466666/T-Flipflop`

`wokwi.com/projects/445174163497792513`

"Uses toggle to create a output"

How it works

A sequential logic circuit that toggles its output state (flips from 0 to 1 or 1 to 0) when the "T" input is high and a clock pulse is received.

How to test

T Q -Q 0 0 0 1 0 1 0 1 1 1 1 0

Clock going to high

External hardware

LED to test

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	Q	—
1	—	Bar Q	—
2	Input	—	—
3	Clock	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny Tapeout Test Gates

by **Sammy** & **Samyak**

0075 | 10 kHz | Wokwi Project

`github.com/MindlessTruffle/test-wokowi`
`wokwi.com/projects/445164196289996801`

"Test sum logic gates and a clock divider"

How it works

does cool stuff

How to test

run it

External hardware

computer

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output nand	—
1	input b	output nand	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

tiny tapout test gates

by **alexandre ney guimaraes**

0077 | 10 kHz | Wokwi Project

`github.com/jwsong2040/junwoo`

`wokwi.com/projects/445165735970991105`

"test some logic gates and a clock divider"

How it works

It is a combination circut

How to test

provide sginal to in0,in1,in2,in3

External hardware

led, light,buttons on input

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	and1	output and	—
1	and2	output or	—
2	or1	—	—
3	or2	—	—
4	in4	—	—
5	in5	—	—
6	in6	—	—
7	in7	—	—

Blake2s

by **Julia Desmazes**

0078 | 66.6667 MHz | HDL Project

`github.com/Essenceia/blake2_asic`

"Blake2s hashing algoryithm, supporting secret keys and block streaming."

This ASIC is a hashing accelerator for the Blake2 cryptographic hash function (RFC 7693).

It is a fully featured Blake2s implementation supporting both block streaming and using a secret key, with a maximum hash rate of 41.42 MB/s and a target operating frequency of 66 MHz.

Blake2s Algorithm

Blake2 is a cryptographic hash function used for applications such as digital signatures, integrity protection and message authentication. It comes in 2 variants, Blake2b and the less memory intensive Blake2s.

—	BLAKE2s
Block bytes	bb = 64
Hash bytes	1 <= nn <= 32
Key bytes	0 <= kk <= 32
Input bytes	0 <= ll < 2**64

In Blake2s, data is processed in blocks of $bb = 64$ bytes, where each block is run through a compression function for multiple rounds of mixing operations.

Each Blake2s run can be configured with specific values for kk, nn, and ll.

The parameter kk represents the optional key length in bytes ($[0; 32]$), where $kk = 0$ means keyless hashing and $kk > 0$ enables keyed mode.

nn specifies the output hash size in bytes ($[1; 32]$), while ll denotes the total input message length in bytes (up to $2^{64} - 1$).

After processing all input data blocks, the final state is truncated to nn bytes to produce the hash output.

Usage

The typical sequence to offload the hashing operation to the accelerator would go as follows:

1. Reset the accelerator (necessary on init)
2. Configure the hash parameters kk, nn, ll (can be reused once configured)
3. Stream the input data by blocks of 64 bytes
4. Read the hash result

All data exchanges with the accelerator are in little endian, and when sending multiple-byte-long arrays, the lower indexes are sent first.

Notes:

- Empty data transfer cycles, as in one or more clock cycles where `valid_i` would go low in the middle of the transfer of both the input data and the configuration, are supported.

Reset

In order to reset this accelerator to its default uninitialized state, deassert the `rst_n` signal for at least 5 clock cycles. During normal operations, `rst_n` should be set to `1`.

During at least 5 clock cycles:

- `rst_n` is set to `0`

Example

Typical reset sequence:

Figure 78.1: rst waves

Sending the Configuration

The configuration packet is 10 bytes long and has the following format:

Byte:	0	1	2	3	4	5	6	7	8	9
	kk	nn	ll							

Figure 78.2: Configuration packet

kk and nn are both 8 bits wide, and ll is 64 bits wide, and all use little endian.

Sending the configuration takes 10 data transfer cycles, during which:

- `valid_i` is set to `1`
- `cmd_i[1:0]` is set to `0`, indicating we are sending the configuration packet
- `data_i[7:0]` sends the next byte of the configuration packet

Example

In this example we are sending the following configuration:

- $kk = 1$ (1 Byte)
- $nn = 32$ (1 Byte)

- $ll = 67$ (8 Bytes)

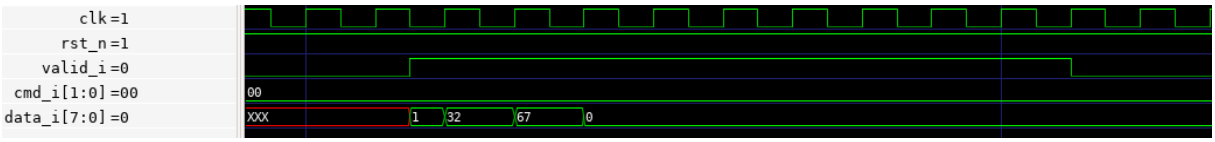


Figure 78.3: config waves

Software

In the firmware, the `send_config` function defined in `data_wr_utils.h` is used to send a configuration to the accelerator.

```
void send_config(uint8_t kk, uint8_t nn, uint64_t ll, uint
dma_chan, pinout_t *p, size_t pl, PIO pio, uint sm);
```

Parameters :

- `kk` configuration value, key length
- `nn` configuration value, final hash length in bytes
- `ll` configuration value, raw data length
- `dma_chan` is the DMA channel used to offload copying data between the memory and the RP2040's PIO
- `p` is a pointer to the shared pre-allocated pool of memory we can temporarily use to allocate the necessary `pinout_t`
- `pio` is the base address of the PIO where the data write program is running
- `sm` is the index of the PIO state machine where the data write program is running

Sending Data

Just like in the original hashing algorithm, the stream of data to be hashed must first be padded with `0x00` to a multiple of 64 bytes, then starting from the lowest indexes first, blocks are sent one by one, one byte at a time.

The sequence to send a block is as follows:

- Wait for ASIC to set `ready_v_o` to 1

Then, start the 64 data transfer cycles during which:

- `valid_i` is set to 1
- `cmd_i[1:0]` is set to :
  - 1 if this is the first data transfer cycle of the first block
  - 3 if this is the last data transfer cycle of the last block
  - 2 by default
- `data_i[7:0]` contains the current data byte

The `ready_v_o` signal indicates the accelerator is ready to receive data. In order to improve performance, users can skip waiting for this signal to be re-

asserted between each byte transfer and can safely proceed with sending the entire block as soon as the `ready_v_o` signal is observed at `1`.

⚠️ Critical Timing Requirement: When using the optimization, since it takes 2 clock cycles for the new value of `ready_v_o` to be written on the output pin, users must guarantee at least a 30ns gap (for 66MHz) between the end of the previous block write and the evaluation of the next `ready_v_o` signal. The current firmware guarantees such a gap.

Single Block Example

This is an example of a simple data transfer sequence where the entirety of the data fits within a single block.

Given there is a single block, meaning it is both the first and last block, the `mode_i` control bits are set to `1` on the first cycle and `3` on the last cycle.

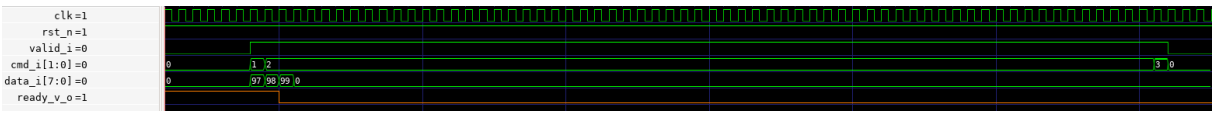


Figure 78.4: single block data transfer example

Multi-Block Example

In this example we are sending two blocks of data.

This example shows the data transfer associated with the configuration waves used as an example above ($kk = 1, nn = 32, ll = 67$).

The first block contains the key of size $kk = 1$ byte, the key's contents are 'a' and padded with `0x00` up to 64 bytes.

After the first block has finished sending, we wait until the accelerator asserts the `ready_v_o` signal before starting the second transfer.

The second block contains the $ll - (kk > 0? 64 : 0)$, here 3, bytes of data "abc", again padded with `0x00` until 64 bytes.

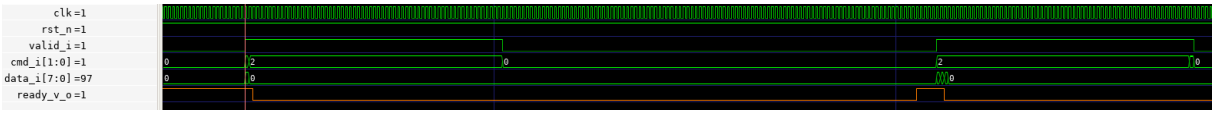


Figure 78.5: multi block data transfer example

Software

In the firmware, for sending data to the accelerator we use the `send_data` function defined in the `data_wr_utils.h` header.

```
void send_data(uint8_t *data, size_t dl, pinout_t *p, size_t pl,
uint dma_chan, PIO pio, uint sm);
```

Parameters :

- `data` is a pointer to the raw data (not extended to a multiple of 64 bytes) to be hashed
- `dl` is the `data` length in bytes
- `p` is a pointer to the shared pre-allocated pool of memory we can temporarily use to allocate the necessary `pinout_t`
- `dma_chan` is the DMA channel used to offload copying data between the memory and the RP2040's PIO
- `pio` is the base address of the PIO where the data write program is running
- `sm` is the index of the PIO state machine where the data write program is running

Slow Output Mode

For the `sky130b` shuttle, although the maximum stable GPIO input switching frequency is 66 MHz, due to a weak driver on the output buffer path resulting in much higher slew rate, the current maximum output stable supported transitioning frequency is 33 MHz.

In order to allow more room for experimenting with the limits of the maximum stable output switching rate while supporting a more stable operating mode, the "slow output" mode was added to this design.

⚠️ Users simply looking to reliably use the accelerator should always have the slow output mode set.

This mode can be enabled by setting `output_mode_i[1:0]` at any time while the accelerator is hashing or receiving data, but for more reliability, we recommend the user simply clamp these pins using the GPIO.

Setting the slow output mode:

- `output_mode_i[1:0]` is set to `3`

Setting the default fast output mode:

- `output_mode_i[1:0]` is set to `0`

Reading the Hash

After the accelerator finishes hashing the last block, it will begin streaming out the final hash result.

In Blake2, the nn configuration parameter specifies how many bytes long the resulting hash should be. This accelerator follows this convention and will only return nn bytes as a result.

Since this accelerator was designed to interface with an embedded MCU and not another accelerator or an FPGA, the accelerator asserts the `hash_v_o` signal ahead of starting to stream out the result. This is done so that we can allow the RP2040 PIO to detect the start of the result sequence and initiate

capturing the data. Because of this, this accelerator is tightly co-designed with the RP2040 in mind and cannot be ported to other MCU families, as it is reliant on a 15ns/30ns (if slow mode is set) reaction time, followed by very timing-accurate capture of the GPIO values. See `firmware/data_rd.pio` for this PIO assembly program.

If slow output mode is set (see the previous section about it), all data steps in the data output sequence take 2 clock cycles; otherwise, each step takes 1 cycle.

The hash read sequence has 2 parts:

1. `h_v_o` (`hash_v_o`) is set to `1` for 1 step (1/2 clock cycles) in order to let the PIO initiate data capture
2. The hash result is streamed over nn steps:
   - `h_v_o` (`hash_valid_o`) is set to `1`
   - `h_o` (`hash_o[7:0]`) contains the hash result

Example Hash result in slow output mode

In this example, slow output mode is set, and the accelerator is returning a hash result of $nn = 32$ bytes long.

Figure 78.6: Slow read waves output

Example Hash result in fast output mode

In this example, the defatul fast output mode is used, and the accelerator is returning a hash result of $nn = 32$ bytes long.

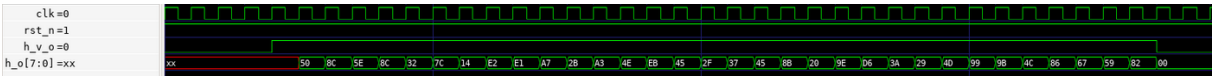


Figure 78.7: Fast read waves output

Software

In order to capture this hash result, given the high-speed nature of the transfer and the precision needed in the capture, the entire capture sequence is offloaded to the PIO. Additionally, since, depending on the configured nn, the PIO RX FIFO (internal rx buffer) might be too small to store all of the hash result, a DMA stream is set up to automatically transfer RX FIFO entries to memory.

Given there is only on the order of 1.5µs between the last data transfer cycle of the last block and the start of the hash result, and given this is not remotely enough time for our MCU to reliably set up a new data capture, the hash result read must be set up before the last data transfer cycle. In practice, in the firmware we use `setup_rd_dma_hash_stream` to set up this DMA stream before the start of any input data streaming, see `firmware/main.c`.

Unlike the input data streaming from the MCU to the accelerator, the hash result is a gapless stream where data is transferred at every step. As such, it is very important that the MCU be able to stream uninterrupted without dropping any of the result bytes. Because of this, the DMA stream between the PIO SM used for reading and the memory is set up to be of the highest priority.

Set up DMA stream to capture the hash result, part of `data_rd_utils.h`:

```
void setup_rd_dma_hash_stream(uint dma_chan, uint nn, uint8_t*
buffer, size_t bl, PIO pio, uint sm);
```

Parameters:

- `dma_chan` - high priority DMA channel used to read from the PIO SM to memory
- `nn` - hash result length in bytes
- `buffer` - pointer to the target location in memory the DMA should write to
- `bl` - buffer length in bytes
- `pio` - base address of the PIO performing the hash read operation
- `sm` - index of the PIO state machine currently running with the hash read program

Given all the reading and transfer operations were already handled under the hood by the PIO and DMA, the `read_hash` function only needs to wait for the end of the DMA transfer operations and copy the hash result written by the DMA into the `hash` buffer.

Read the hash result from the DMA target memory location, part of `data_rd_utils.h`:

```
void read_hash(uint8_t* hash, uint8_t nn, uint8_t* buffer, size_t
bl, uint dma_chan);
```

Parameters:

- `hash` - pointer to the buffer to which the hash result will be copied
- `nn` - hash result length in bytes
- `buffer` - pointer to the target location in memory the DMA should write to
- `bl` - buffer length in bytes
- `dma_chan` - high priority DMA channel used to read from the PIO SM to memory

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data_i[0]	hash_o[0]	valid_i[0]

#	Input	Output	Bidirectional
1	data_i[1]	hash_o[1]	cmd_i[0]
2	data_i[2]	hash_o[2]	cmd_i[1]
3	data_i[3]	hash_o[3]	ready_v_o
4	data_i[4]	hash_o[4]	output_mode_i[0]
5	data_i[5]	hash_o[5]	output_mode_i[1]
6	data_i[6]	hash_o[6]	—
7	data_i[7]	hash_o[7]	hash_valid_o[7]

Nicky - Tiny Tapeout

by **Nicky**

0079 Wokwi Project

github.com/panda-blip/Tiny-Tapeout2

wokwi.com/projects/445163669892121601

"A Simple 8 bit binary counter"

How it works

8 T flip-flops are chained together to output an 8 bit integer, which will tick up by 1 when input 1 is pressed.

How to test

Press the button several times and check if the outputs match the expected values ## External hardware

A single push button connected to Input 1. 8 LED's arranged in a line connected to each output.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Count Up	Bit 1	—
1	—	Bit 2	—
2	—	Bit 3	—
3	—	Bit 4	—
4	—	Bit 5	—
5	—	Bit 6	—
6	—	Bit 7	—
7	—	Bit 8	—

Function Generator

by **Dominik Brandstetter**

0128 | 1 MHz | HDL Project

`github.com/DomnikBrandstetter/jku-sky-FunctionGenerator`

> *"An 8-bit TinyTapeout function generator targeting the AD5330 parallel DAC. A write-only register interface selects a constant signal, a sine wave, or programmable trapezoid/pulse waveforms with adjustable amplitude, offset, phase, and timing."*

Module: `tt_um_FG_TOP_Dominik_Brandstetter`

A compact, **programmable 8-bit function generator** for TinyTapeout. It outputs either a **constant DC level**, a **CORDIC-based sine**, or a **programmable trapezoid/pulse** (rise–hold–fall). A tiny on-chip timer with prescaler drives sample timing. Configuration uses TinyTapeout GPIOs via a simple write-only register interface. The 8-bit output feeds an external DAC or resistor ladder.

Target DAC: Analog Devices **AD5330** (parallel, 8-bit).

🚀 Quick Start

1. **Wire the DAC** (see *AD5330 Hookup* below).
2. Power up with **`ENABLE=0`**.
3. (Optional) Program **`CR0..CR6`** while halted.
4. Drive **`ENABLE=1`** → it runs.
   - Default registers give a clean **2 kHz sine** out of the box.

🧰 Top-Level I/O (TinyTapeout)

Signal	Dir	W	Purpose
`ui_in[7:0]`	in	8	Register **data bus** (write-only)
`uo_out[7:0]`	out	8	**DAC data** (parallel 8-bit)
`uio_in[7]`	in	1	**ENABLE**: Low = halt, High = run
`uio_in[6]`	in	1	**WR_n** (active-low) – write selected register
`uio_in[5:3]`	in	3	**ADDR[2:0]** – select `CR0..CR6` (0..6)
`uio_out[0]`	out	1	**dac_clr_n** → AD5330 `/CLR`
`uio_out[1]`	out	1	**dac_pd_n** → AD5330 power-down (**high = enabled**)

`uio_out[2]`	out	1	**dac_wr_n** → AD5330 `/WR` pulse on new sample
`clk`	in	1	System clock
`rst_n`	in	1	Async reset (active-low)
`ena`	in	1	Always 1 on TinyTapeout

Write protocol: Assert **`WR_n=0` ≥ 3 clocks** with `ADDR` & `DATA` stable to update the target register.

🗺️ Register Cheat-Sheet (7 × 8-bit → 56-bit bus `{CR0..CR6}`)

Bits	From	What it does
55	CR0	**Constant mode** (1 = force DC = AMPLITUDE)
54	CR0	**Mod select** (0 = Waveform, 1 = Sine)
53..48	CR0	**Prescaler** (timer divider)
47..40	CR1	**Counter** (sine: phase increment · waveform: period)
39..32	CR2	**Initial phase** (sine) · **ON duration** (waveform)
31..24	CR3	**Rise step** per tick (waveform)
23..16	CR4	**Fall step** per tick (waveform)
15..8	CR5	**Amplitude** (sine & waveform; DC level in Constant mode)
7..0	CR6	**Offset** (added after selection)

Reset defaults:
`CR0=0x49`, `CR1=0x05`, `CR2=0x00`, `CR3=0x00`, `CR4=0x00`, `CR5=0x32`, `CR6=0x00`.

CR0 — Mode & Prescaler (addr 0, reset `0x49`)

```
[55] Constant mode    1 → output = AMPLITUDE (modulated path
ignored)
[54] Mod select       0 → Waveform, 1 → CORDIC sine
[53:48] Prescaler     6-bit timer divider (timebase=100kHz @1MHz
clock_i)
```

CR1 — Timer Counter (addr 1, reset `0x05`)

```
[47:40] Counter
        • Sine: phase increment per sample (sets frequency)
        • Waveform: period value for RISE/ON/FALL FSM
```

CR2 — Phase / ON Duration (addr 2, reset `0x00`)

```
[39:32] Initial phase / ON time
        • Sine: initial accumulator load
        • Waveform: ON ticks (RISE+ON before FALL)
```

CR3 — RISE Step (addr 3, reset `0x00`)

```
[31:24] k_rise   step/tick during RISE (0 → AMPLITUDE slope)
```

CR4 — FALL Step (addr 4, reset `0x00`)

```
[23:16] k_fall   step/tick during FALL (AMPLITUDE → 0 slope)
```

CR5 — Amplitude (addr 5, reset `0x32`)

```
[15:8]  AMPLITUDE   peak for sine & waveform; constant level in DC
mode
```

CR6 — Offset (addr 6, reset `0x00`)

```
[7:0]   OFFSET      signed (two's complement) post-add
        • Overflow wraps (no saturation)
```

🎛️ Mode Recipes

Sine (default)

- Ensure `CR0[55]=0`, `CR0[54]=1`.
- Set **frequency**: with `Prescaler` (`CR0[53:48]`), and with `Counter` (`CR1`).
- Set **level**: `AMPLITUDE` (`CR5`) + `OFFSET` (`CR6`).

Constant (DC)

- Set `CR0[55]=1`.
- Set **level**: `AMPLITUDE` (`CR5`) + `OFFSET` (`CR6`).

Trapezoid / Pulse

- Set `CR0[55]=0`, `CR0[54]=0`.
- Shape with `k_rise` (`CR3`), `ON duration` (`CR2`), `k_fall` (`CR4`).
- Timing base: `Prescaler` (`CR0`) + `Counter` (`CR1`).
- Set **level**: `AMPLITUDE` (`CR5`) + `OFFSET` (`CR6`).

🔌 AD5330 Hookup (Minimal)

- **Data:** `uo_out[7:0]` → AD5330 `DB[7:0]`
- **Control:**
  - `uio_out[2]` → `/WR`
  - `uio_out[1]` → `PD_n` (**high = enabled**)
  - `uio_out[0]` → `/CLR` (**high** in normal operation)

📝 Notes & Best Practices

- All DAC control lines are **active-low**.
- Leave **headroom**: `AMPLITUDE + OFFSET` must fit 8-bit (wraps otherwise).

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	DATA_IN_0	DAC_OUT_0	dac_clr_OUT (active low)
1	DATA_IN_1	DAC_OUT_1	dac_pd_OUT (active low)
2	DATA_IN_2	DAC_OUT_2	dac_wr_OUT (active low)
3	DATA_IN_3	DAC_OUT_3	CONFIG ADDR_0_IN
4	DATA_IN_4	DAC_OUT_4	CONFIG ADDR_1_IN
5	DATA_IN_5	DAC_OUT_5	CONFIG ADDR_2_IN
6	DATA_IN_6	DAC_OUT_6	WR_enable_IN (active low)
7	DATA_IN_7	DAC_OUT_7	Enable_IN (active low)

EmilsChip

by **Emil**

0129 Wokwi Project

`github.com/RubaEm220048/emilschip`

`wokwi.com/projects/442983487202965505`

"nixfunktionierendiggalata"

How it works

Pfusch PfuschPfuschPfuschP

Simon Says memory game

by **Uri Shaked**

0130 | 50 kHz | HDL Project

github.com/urish/tt-simon-game

"Repeat the sequence of colors and sounds to win the game"

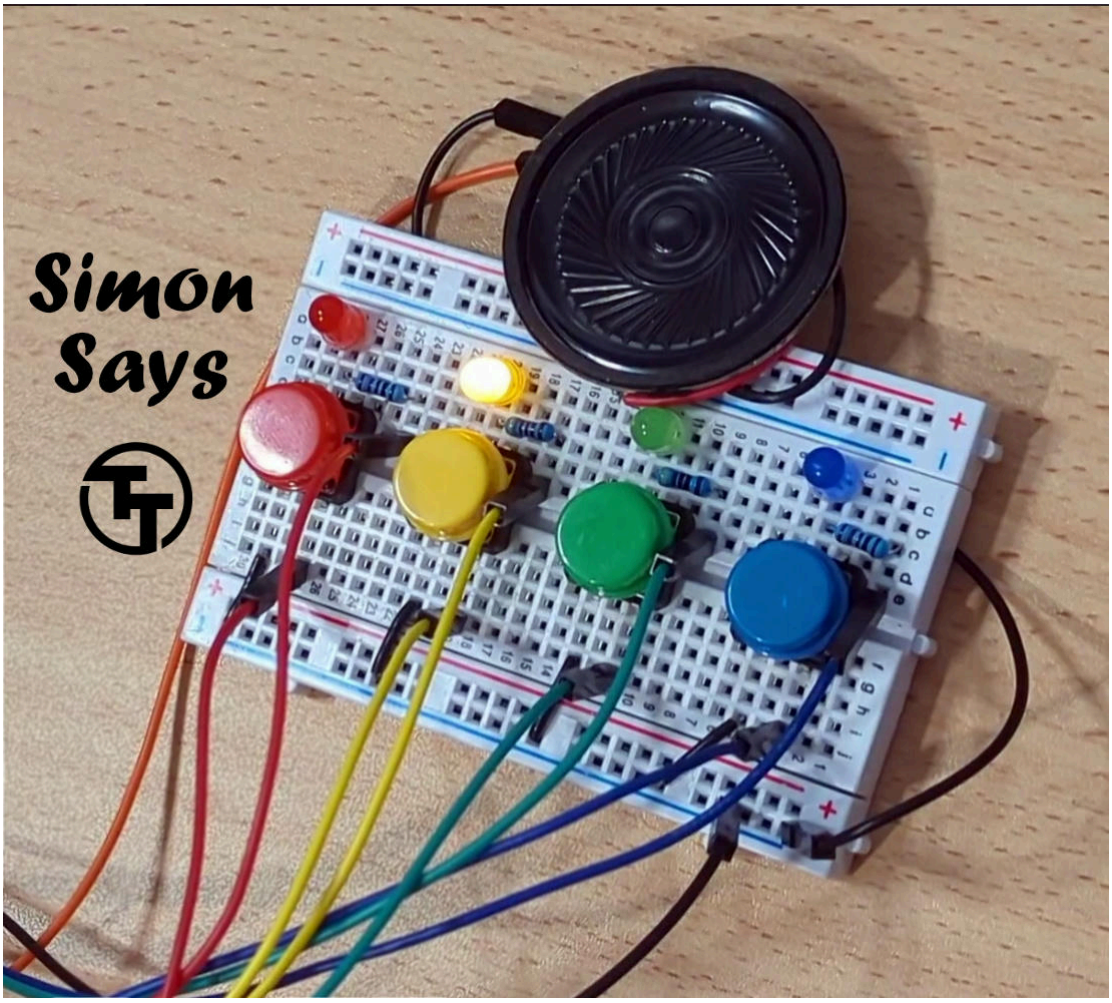


Figure 130.1: Simon Says Game

How it works

Simon says is a simple electronic memory game: the user has to repeat a growing sequence of colors. The sequence is displayed by lighting up the LEDs. Each color also has a corresponding tone.

In each turn, the game will play the sequence, and then wait for the user to repeat the sequence by pressing the buttons according to the color sequence. If the user repeated the sequence correctly, the game will play a

"leveling-up" sound, add a new color at the end of the sequence, and move to the next turn.

The game continues until the user has made a mistake. Then a game over sound is played, and the game restarts.

Check out the online simulation at https://wokwi.com/projects/408757730664700929 (including wiring diagram).

Clock settings

The `clk_sel` input selects the clock source:

- `0`: external 50 KHz clock, provided through the `clk` input.
- `1`: internal clock, generated by the `ring_osc` module, with a frequency of 62.5 KHz.

The internal clock is generated by a 13-stage ring oscillator, divided by 8192 to get the desired frequency. The divider value was determined by measuring the output frequency of tt_um_urish_ringosc_cnt from Tiny Tapeout 5.

When using the internal clock, its signal is also output on the `uo_out[7]` pin for debugging purposes.

How to test

Use a Simon Says Pmod to test the game.

Provide a 50 KHz clock input, reset the game, and enjoy!

If you don't have the Pmod, you can still connect the hardware manually as follows:

1. Connect the four push buttons to pins `btn1`, `btn2`, `btn3`, and `btn4`. Also connect each button to a pull down resistor.
2. Connect the LEDs to pins `led1`, `led2`, `led3`, and `led4`, matching the colors of the buttons (so `led1` and `btn1` have the same color, etc.). Don't forget current-limiting resistors!
3. Connect the speaker to the `speaker` pin (optional).
4. Connect the seven segment display as follows: `seg_a` through `sev_g` to individual segments, `dig1` to the common pin of the tens digit, `dig2` to the common pin of the ones digit. Set `seginv` according to the type of 7 segment display you have: high for common anode, low for common cathode.
5. Reset the game, and then press any button to start it. Enjoy!

External Hardware

Simon Says Pmod or four push buttons (with pull-down resistors), four LEDs, and optionally a speaker/buzzer and two digit 7-segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	btn1	led1	seg_a
1	btn2	led2	seg_b
2	btn3	led3	seg_c
3	btn4	led4	seg_d
4	seginv	speaker	seg_e
5	—	dig1	seg_f
6	—	dig2	seg_g
7	clk_sel	clk_internal	—

first-chip-project

by **moritz puehringer**

0131 Wokwi Project

github.com/Moritzpuehringer/new-project-chip
wokwi.com/projects/442977774857423873

"test some logical gates"

How it works

when pressing clock and 1 it hsould set 1 on the display ## How to test

when pressing clock and 2 it hsould set 2 on the display ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Anzeige auf 1 schalten	—	—
1	Anzeige auf 2 schalten	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Lauflicht_Erweiteung_HTL_LEONDING

by **Florian Ackerbauer**

0132 | 10 kHz | Wokwi Project

`github.com/florian-c-a/Lauflicht_Erweiterung_HTL_LEONGING`
`wokwi.com/projects/442987960046504961`

"Lauflicht"

How it works

Lauflicht erweitung

How to test

turn is on an wath the led

External hardware

led

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	LED1	—
1	—	LED2	—
2	—	LED3	—
3	—	LED4	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

EnDecoder

by **Felix Franz**

0133 | 50 kHz | HDL Project

github.com/workfefra/FeFr

"Encrypts/Decrypts a 4 bit signal with a 4 bit key"

How it works

This project Is Encrypter/Decoder. The user can choose wether he wants to encrypt a 4 bit code or decrypt it with a 4 bit key. The key determines internal changes to the 4 bit code. the output creates a en/de-cryptet code. ## How to test

Choose a 4 bit code; choose a 4 bit key and put it in the EnDecoder with '0' on the mode_i input; when getting a high pulse on the done_o output there should be a 4 bit code on the code_o output. If this output is put in the input with '1' at mode_i the output should be the original input try with different keys and input codes.

External hardware

No external hardware needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	code_i[0]	output[0]	mode_i
1	code_i[1]	output[1]	done_i
2	code_i[2]	output[2]	—
3	code_i[3]	output[3]	—
4	key_i[3]	done_o	—
5	key_i[2]	—	—
6	key_i[1]	—	—
7	key_i[0]	—	—

Tiny Tapeout Template

by **Valentin Loibingdorfer**

0134 Wokwi Project

github.com/Hinuitsu/4-Bit-Counter

wokwi.com/projects/442987575358494721

"4-Bit Downcounter"

How it works

A 4-bit down counter is a digital circuit that counts backward in binary from 1111 (decimal 15) down to 0000 (decimal 0). It consists of four flip-flops, each holding one bit of the binary number: Q3 is the most significant bit and Q0 is the least significant bit. Every time a clock pulse arrives, the counter subtracts one from its current value. The outputs change in a fixed sequence: 1111 → 1110 → 1101 → ... → 0001 → 0000, and then it returns to 1111 again.

How to test

I dunno, if it counts downwards from F-0 it works

External hardware

7 segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	I0	O0	—
1	I1	O1	—
2	I2	O2	—
3	I3	O3	—
4	I4	O4	—
5	I5	O5	—
6	I6	O6	—
7	I7	O7	—

Austrochip 2025 Tiny Tapeout Project

by **Dylan Baumann**

0135 | 50 MHz | Wokwi Project

github.com/Bawady/austrochip_tt_workshop

wokwi.com/projects/442977515486800897

"MUTEX, inefficient C-gate and LFSR"

How it works

Hooks up the clock to some LFSR that drives some outputs, other outputs show the MUTEX grants and the C gate output.

How to test

Check if MUTEX grants are mutually exclusive, if C gate behaves like a C gate and if the other outputs show a 4 bit PRNG sequence.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	mutex r1	mutex g1	—
1	mutex r2	mutex g2	—
2	cgate input 1	cgate out	—
3	cgate input 2	lfsr 0	—
4	—	lfsr 1	—
5	—	lfsr 2	—
6	—	lfsr 3	—
7	—	—	—

Tiny Tapeout Projekt

by **Tobias Pretl**

0136 | 10 kHz | Wokwi Project

`github.com/KitKatHTL/TinyTapeout-Tobias`
`wokwi.com/projects/442977490552687617`

"Simple 2 input complexer"

How it works

1 and 2 are the inputs and 1 2 3 are the outputs

How to test

input 1 and 2 are controlling it

External hardware

nothing

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output 1 complexer	—
1	input b	output 2 complexer	—
2	—	output 3 complexer	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

transformer-attention-engine

by **Tarik Ibrahimovic**

0137 | 60 MHz | HDL Project

`github.com/tarik-ibrahimovic/transformer_attention_engine`

"Transformer attention engine with a credit based pipeline"

Here you go—drop-in text for your datasheet. It's short, objective, and matches the I/O in your module.

How it works

- **Purpose.** Computes one dot-product over 4 terms from sequential 8-bit Q0.7 samples, then outputs `exp(mac/2)` in **UQ3.6** (9-bit) format.
- **Interface.**
  - Inputs:
    - `ui_in[7:0]` — data (Q0.7 signed).
    - `uio_in[0]` — `vld_slv_in` (input valid).
    - `uio_in[3]` — `rdy_mst_in` (output handshake from master).
  - Outputs:
    - `uo_out[7:0]` — result LSBs (UQ3.6).
    - `uio_out[4]` — result MSB (bit 8).
    - `uio_out[2]` — `vld_mst_out` (output valid).
    - `uio_out[1]` — `rdy_slv_out` (always 1, input always ready).
- **Operation.**
  1. The core ingests **pairs** of consecutive samples. On the first valid, it latches the value; on the second valid, it multiplies the new sample by the latched one and **accumulates** into a 17-bit MAC.
  2. After **4 such products** (i.e., **8 input samples total, provided as 4 back-to-back pairs**), it computes `exp(mac/2)` via the `ex` block and exposes the 9-bit UQ3.6 result.
  3. `vld_mst_out` goes high to signal a ready result; it de-asserts once the master asserts `rdy_mst_in`.

How to test

1. **Reset & enable.** Drive clock, hold `rst_n=0` for ≥1 cycle, then set `rst_n=1`. Keep `ena=1`.

2. **Feed inputs (4 pairs = 8 samples).** For each of **4 pairs**:
   - **First of pair:** place sample A on `ui_in`, set `uio_in[0]=1` for one (or more) cycle(s).
   - **Second of pair:** place sample B on `ui_in`, set `uio_in[0]=1` again → the product `A*B` is accumulated.
   - You may keep `uio_in[0]=0` when idle. Input is always ready (`uio_out[1]=1`).
3. **Read result.**
   - After the 4th pair, `uio_out[2]=1` (output valid).
   - Read 9-bit UQ3.6 result: `{uio_out[4], uo_out[7:0]}`.
   - Acknowledge by setting `uio_in[3]=1` for ≥1 cycle; `uio_out[2]` will drop.
4. **Repeat.** Provide the next 4 pairs to get the next result.

Notes.

- Input format: Q0.7 (signed).
- Output format: UQ3.6 (unsigned 9-bit).
- Input back-pressure is not asserted (module is always ready).

External hardware

i will

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	ui_in[0]	uo_out[0]	uio_out[0]
1	ui_in[1]	uo_out[1]	uio_out[1]
2	ui_in[2]	uo_out[2]	uio_out[2]
3	ui_in[3]	uo_out[3]	uio_out[3]
4	ui_in[4]	uo_out[4]	uio_out[0]
5	ui_in[5]	uo_out[5]	uio_out[1]
6	ui_in[6]	uo_out[6]	uio_out[2]
7	ui_in[7]	uo_out[7]	uio_out[3]

Tiney Tapeout Test Gates

by **Theo Kreisel**

0138 | 1 kHz | Wokwi Project

github.com/TheoKreisel/-bung1_Theo

wokwi.com/projects/442977606409919489

"Test some logic gates and a clock divider."

How it works

123

How to test

123

External hardware

123

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	not uesd	not uesd	not uesd
1	not uesd	not uesd	not uesd
2	not uesd	not uesd	not uesd
3	not uesd	not uesd	not uesd
4	not uesd	not uesd	not uesd
5	not uesd	not uesd	not uesed
6	not uesd	not uesd	not uesd
7	not uesd	not uesd	not uesd

tt_primitiv_rotating_7segment

by **fl00l**

0139 | 10 kHz | Wokwi Project

`github.com/floll/tt_primitiv_rotating_7segment`

`wokwi.com/projects/442983161563395073`

"primitiv_rotating_7segment"

How it works

It only uses the clock and than loops around may need a few secend to reset from undefined flip flop states

How to test

Not tests jet jsut run it

External hardware

a pmod 7 segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	IN7	OUT7	—

Tiny tapeout test gates

by **Abdulrahman Alsabagh**

`0140` `10 kHz` `Wokwi Project`

`github.com/Abdulrahman-AL-Sabagh/tiny-tapeout-workshop`

`wokwi.com/projects/442987201599784961`

"Just a simple test"

How it works

AND, NAND and OR gates are connected to input a and b 16 flip flops are used to divide the clock spped with 2 ouptus, one at the 12th flop and one at the 16th.

How to test

Set the inputs and check the outputs match with the expected results:

```
| input a and b | output AND | output NAND | output OR |
|---------------|------------|-------------|-----------|
| 0 0           | 0          | 1           | 0         |
| 0 1           | 0          | 1           | 1         |
| 1 0           | 0          | 1           | 1         |
| 1 1           | 1          | 0           | 1         |
```

```
Set the clock to 10KHz and you should see the 5th output flashing
quickly and the 4th output flashing slowly
```

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	—	output and
1	input b	—	output nand
2	—	—	output or
3	—	—	output clock div 16
4	—	—	—
5	—	—	—
6	—	—	—

#	Input	Output	Bidirectional
7	—	—	—

VGA Pattern Generator

by **Lukas Dragoste**

0141 | 25 MHz | HDL Project

`github.com/LukasDragoste/TinyTapeout`

"Generating patterns and displaying them on a monitor by using the VGA protocol"

How it works

This project implements a simple VGA pattern generator that produces colorful geometric visuals on a monitor.
The logic generates standard **640×480 @ 60 Hz** VGA timing signals (HSYNC, VSYNC, front/back porch, and visible area)
using horizontal and vertical counters implemented in Verilog.

The color output is created procedurally from the pixel coordinates.
Each pixel's RGB values are computed based on bitwise operations such as XOR and pattern masks, resulting in
different shapes and wave-like effects.
Six output bits (`R1, G1, B1, R0, G0, B0`) form a 6-bit color depth (2 bits per channel), providing 64 distinct colors.

The three least significant input bits (`ui_in[2:0]`) select between different visual modes such as
checkerboard, moiré lines, plasma gradients, and pseudo-random noise.
The remaining upper bits (`ui_in[7:3]`) modify palette or variation parameters.
Each frame is generated in real time using a **25 MHz pixel clock**.

How to test

Simulation (recommended)

You can simulate the design using the included cocotb testbench:

```
cd test
make -B
```

This produces a preview file called frame_200x150.ppm, which contains a portion of the generated VGA image. Open the PPM file with any image viewer to see the visual pattern. Changing the value of dut.ui_in.value inside test/test.py allows you to test other pattern modes.

Hardware testing After fabrication, connect the chip or TinyTapeout demo board to a TinyVGA PMOD adapter:

- Connect the VGA output pins (uo_out[7:0]) to the TinyVGA adapter.
- Attach the adapter to a VGA monitor configured for 640×480 @ 60 Hz.
- Use the input switches (ui_in[7:0]) to select between pattern and color variations.
- HSYNC and VSYNC are active-low and follow standard VGA timing.

Expected behavior: The monitor displays colorful geometric or dynamic-looking patterns, changing instantly when input switches are toggled.

External hardware

- Display: Any VGA-compatible monitor (640×480 @ 60 Hz recommended)
- Adapter: TinyVGA PMOD or similar 8-bit DAC adapter for connecting uo_out pins to VGA lines
- Inputs: 8-bit DIP switches or GPIO inputs for ui_in[7:0]
- Power: Supplied by the TinyTapeout evaluation board (no external supply needed)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Pattern select bit 0	Red MSB	—
1	Pattern select bit 1	Green MSB	—
2	Pattern select bit 2	Blue MSB	—
3	Palette / variation 0	VSync (active low)	—
4	Palette / variation 1	Red LSB	—
5	Palette / variation 2	Green LSB	—
6	Palette / variation 3	Blue LSB	—
7	Palette / variation 4	HSync (active low)	—

INTERCAL ALU

by **Rebecca G. Bettencourt**

0142 HDL Project

github.com/RebeccaRGB/intercal-alu

"An ALU for the five operators of the INTERCAL programming language."

How it works

As an educational project, it is inevitable that Tiny Tapeout would attract various pedagogical examples of common logic circuits, such as ALUs. While ALUs for common operations such as addition, subtraction, and binary bit-wise logic are surprisingly common, it is much rarer to encounter one that can calculate the five operations of the INTERCAL programming language. Due to either the cost-prohibitive nature of Warmenhovian logic gates or general lack of interest, such a feat has never been performed until now. With chip production finally within reach of the average person, all it takes is one person who has more dollars than sense to design the fabled INTERCAL ALU (Arrhythmic Logic Unit).

The pin assignments for this design are roughly as follows. The /OE (output enable) and /WE (write enable) signals are active low, so should be set HIGH by default.

#	Dedicated Input	Dedicated Output	Bidirectional I/O
0	A0 (address)	D0 (output only)	D0 (input and output only)
1	A1 (address)	D1 (output only)	D1 (input and output only)
2	S0 (selector)	D2 (output only)	D2 (input and output only)
3	S1 (selector)	D3 (output only)	D3 (input and output only)
4	S2 (selector)	D4 (output only)	D4 (input and output only)
5	S3 (selector)	D5 (output only)	D5 (input and output only)
6	/OE (output enable)	D6 (output only)	D6 (input and output only)
7	/WE (write enable)	D7 (output only)	D7 (input and output only)

This ALU has two 32-bit registers, B and A (in no particular order). (These may also be thought of as four 16-bit registers, AL, AH, BL, and BH.) To write a byte to a register, set A0 and A1 to the byte address, set S0 LOW for the A register or HIGH for the B register, set S1 through S3 LOW, set the bidirectional I/O pins to the byte value, set /WE LOW, then set /WE HIGH again. (Do not set S1 through S3 HIGH when writing, or else something unpredictable will happen, most likely nothing.)

To read a register or result, set A0 and A1 to the byte address, set S0 through S3 to the desired operation, set /OE LOW, read the byte value from the bidirectional I/O pins, then set /OE HIGH. Results can also be read from the dedicated outputs; the dedicated outputs are not affected by the /OE signal, as they do not need to care about your feelings.

The operations supported are listed below. An attempt was made to make it understandable.

						Address				
						A	3	2	1	0
Selector						A1	1	1	0	0
S	S3	S2	S1	S0	Operation	A0	1	0	1	0
0	0	0	0	0	A		AH		AL	
1	0	0	0	1	B		BH		BL	
2	0	0	1	0	AND16		& AH		& AL	
3	0	0	1	1	AND32		& A			
4	0	1	0	0	OR16		V AH		V AL	
5	0	1	0	1	OR32		V A			
6	0	1	1	0	XOR16		? AH		? AL	
7	0	1	1	1	XOR32		? A			
8	1	0	0	0	MINGLE16L		AL $ BL			
9	1	0	0	1	MINGLE16H		AH $ BH			
10	1	0	1	0	SELECT16		AH~ BH		AL~ BL	
11	1	0	1	1	SELECT32		A ~ B			

Operations 0 and 1 simply return the current value of the A or B register, respectively. This corresponds with the values of S0 through S3 used in write mode. This is not unintentional. This might also explain why S1 through S3 must be LOW in write mode.

Operations 2 through 7 correspond to INTERCAL's unary AND, unary OR, and unary XOR operators, represented by ampersand (&), book (V), and what (?), respectively. From the INTERCAL manual:

These operators perform their respective logical operations on all pairs of adjacent bits, the result from the first and last bits going into the first bit of the result. The effect is that of rotating the operand one place to the right and ANDing, ORing, or XORing with its initial value. Thus, #&77 (binary = 1001101) is binary 0000000000000100 = 4, #V77 is binary 1000000001101111 = 32879, and #?77 is binary 1000000001101011 = 32875. Operations 2, 4, and 6 work on the 16-bit halves of the A register independently, while operations 3, 5, and 7 work on the 32-bit whole of the A register.

Operations 8 and 9 correspond to INTERCAL's *interleave* (also called *mingle*) operator, represented by big money ($). From the INTERCAL manual:

The interleave operator takes two 16-bit values and produces a 32-bit result by alternating the bits of the operands. Thus, #65535$#0 has the 32-bit binary form 101010....10 or 2863311530 decimal, while #0$#65535 = 0101....01 binary = 1431655765 decimal, and #255$#255 is equivalent to #65535. Operation 8 returns the interleave of the lower halves of A and B, while operation 9 returns the interleave of the upper halves of A and B. (Should the chip fabrication process allow for it, operation 8½ will, of course, return the interleave of the middle halves of A and B.)

Operations 10 and 11 correspond to INTERCAL's *select* operator, represented by sqiggle ( ). From the INTERCAL manual:

The select operator takes from the first operand whichever bits correspond to 1's in the second operand, and packs these bits to the right in the result. Both operands are automatically padded on the left with zeros. [...] For example, #179 #201 (binary value 10110011 11001001) selects from the first argument the 8th, 7th, 4th, and 1st from last bits, namely, 1001, which = 9. But #201 #179 selects from binary 11001001 the 8th, 6th, 5th, 2nd, and 1st from last bits, giving 10001 = 17. #179 #179 has the value 31, while #201 #201 has the value 15. To help understand the select operator, the INTERCAL manual also provides a helpful circuitous diagram.

Use of operations 12 and above is not recommended, unless undefined behavior is required.

How to test

The following example calculations found in the INTERCAL manual should be particularly illuminating.

S	A	B	F
`MINGLE16L` (8)	0	256	65536
`MINGLE16L` (8)	65535	0	2863311530
`MINGLE16L` (8)	0	65535	1431655765

`MINGLE16L` (8)	255	255	65535
`SELECT16` (10)	51	21	5 *
`SELECT16` (10)	179	201	9
`SELECT16` (10)	201	179	17
`SELECT16` (10)	179	179	31
`SELECT16` (10)	201	201	15
`AND16` (2)	77	—	4
`OR16` (4)	77	—	32879
`XOR16` (6)	77	—	32875

These test cases are included in the (unfortunately Python and not INTERCAL) `test.py` file. As these are likely more INTERCAL operations than any sensible person will ever perform, they should be sufficient for testing purposes. However, for curiosity's sake, an extensive set of additional test cases have also been included.

* Not found in the INTERCAL manual.

External hardware

The ALU may be used without external hardware, although seeing the output values may present a challenge. Instead, it is recommended to use a microcontroller of some sort to drive the inputs and read the outputs, as microcontrollers are designed to do. The implementation of the rest of the INTERCAL language is left as an exercise for the reader.

Further reading

The INTERCAL Programming Language Revised Reference Manual by Donald R. Woods and James M. Lyon with revisions by Louis Howell and Eric S. Raymond (can recommend highly enough)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
`0`	A0 (address)	D0	D0
1	A1 (address)	D1	D1
2	S0 (selector)	D2	D2
3	S1 (selector)	D3	D3
4	S2 (selector)	D4	D4
5	S3 (selector)	D5	D5

#	Input	Output	Bidirectional
6	/OE (output enable)	D6	D6
7	/WE (write enable)	D7	D7

tiny tapeout test gates

by **grabner moritz**

0143 Wokwi Project

`github.com/mograbner/mo`

`wokwi.com/projects/442977792040445953`

"test"

How it works

two NANDs are connected to work like a flipflop ## How to test

switch on one of the first two switches (Reset and Set) and the LED should do different things ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any 7 segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	R	1	—
1	S	2	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Wowki

by **Arwin Ghahremanian**

0160 | 1 Hz | Wokwi Project

`github.com/Arwin2012/Tinytapeout`

`wokwi.com/projects/445266058506969089`

"Wokwi"

How it works

it makes 678 by pressing either on the first four switches ## How to test flick the butttons and play around ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	If it is on and ui[1] is off and either ui[2] is off or ui[3] is off it displays a 6 but if both ui[2] or ui[3] are on then nothing pops up but if this and ui[1] are off while both ui[2] and ui[3] are on then 7 pops up but if this and ui[1] are off and only one of ui[2] and ui[3] are on then it displays an 8	—	—
1	If it is off and ui[0] is on and ui[2] is off and ui[3] is off it displays a 6 but if both ui[2] or ui[3] are on then nothing pops up but if this and ui[0] are off while both ui[2] and ui[3] are on then 7 pops up but if this and ui[0] are off and only one of ui[2] and ui[3] are on then it displays an 8	—	—
2	what it said above	—	—
3	what it said above	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Arrow Board

by **Rebecca G. Bettencourt**

0161 | 1 Hz | HDL Project

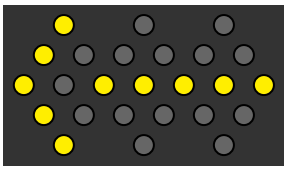
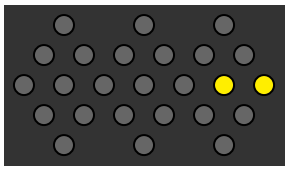
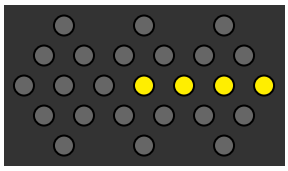
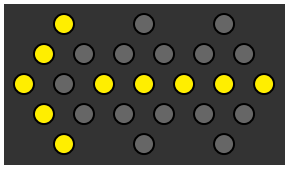
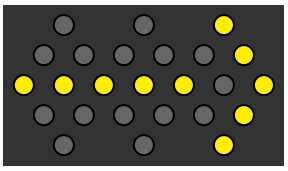
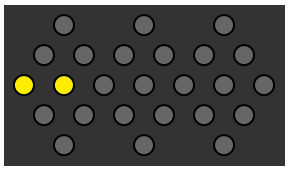
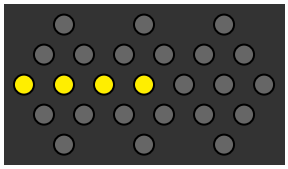
`github.com/RebeccaRGB/tt-arrowboard`

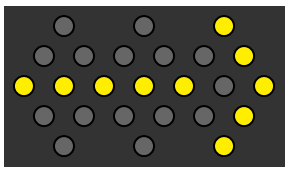
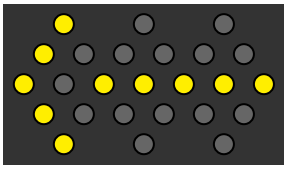
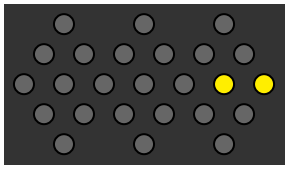
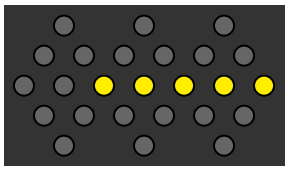
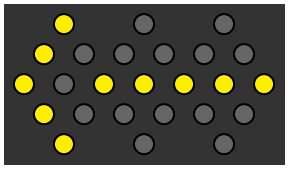
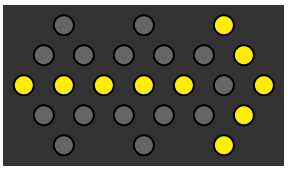
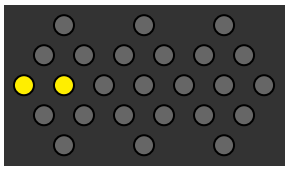
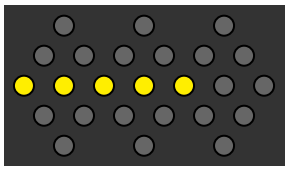
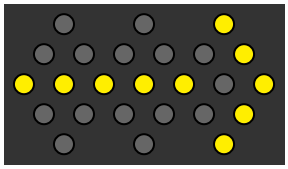
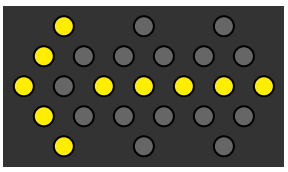
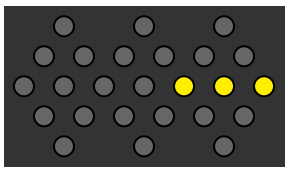
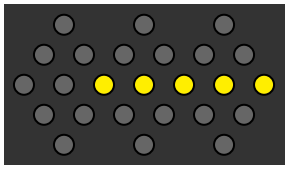
"Displays animated arrow signs."

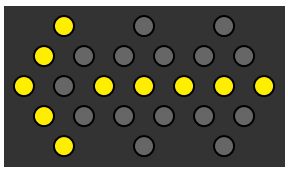
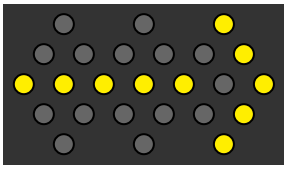
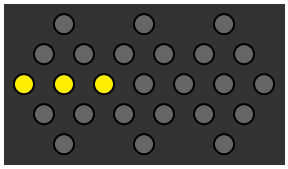
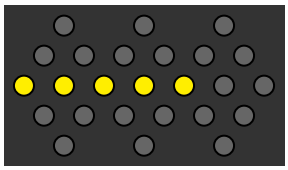
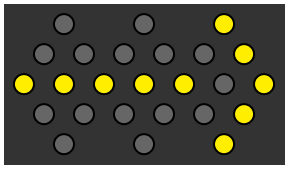
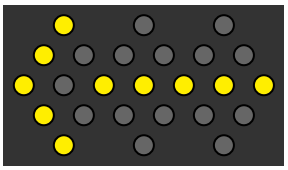
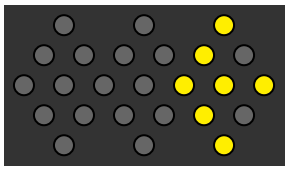
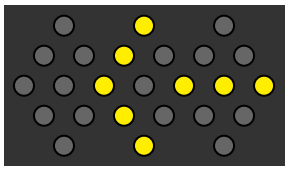
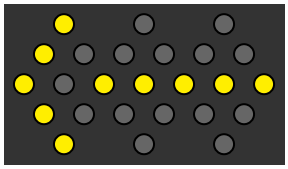
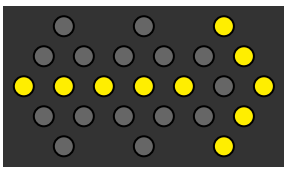
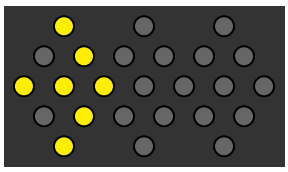
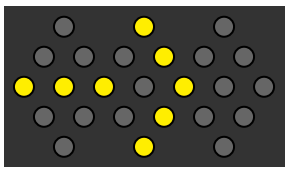
How it works

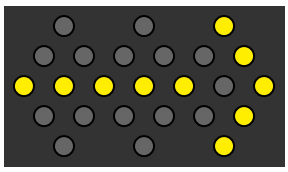
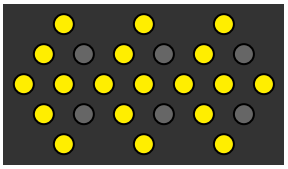
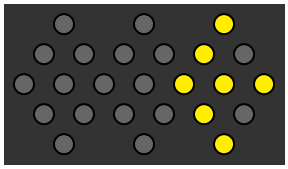
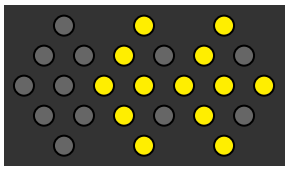
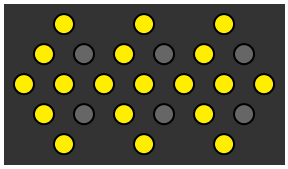
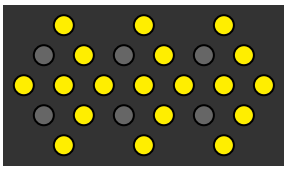
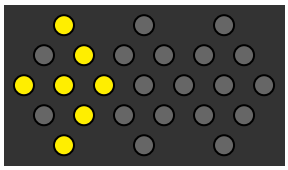
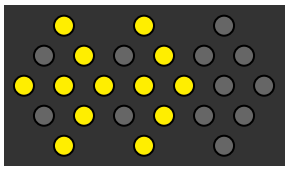
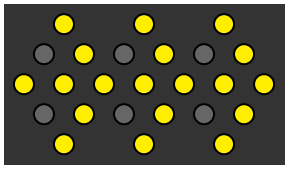
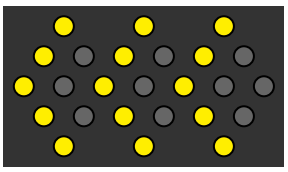
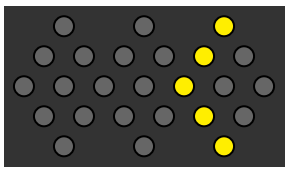
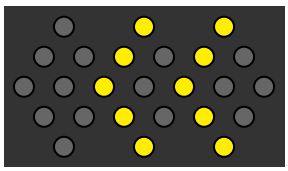
This project displays various flashing and animated arrow signs used for controlling traffic.

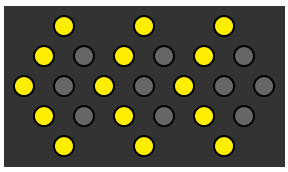
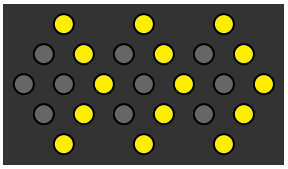
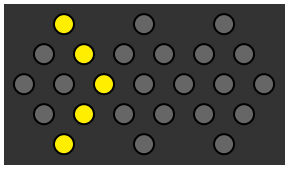
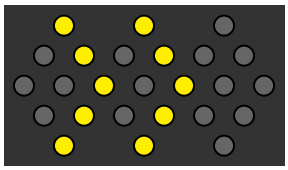
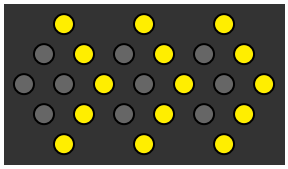
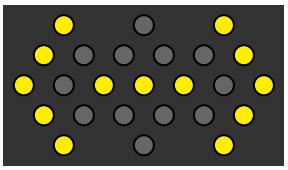
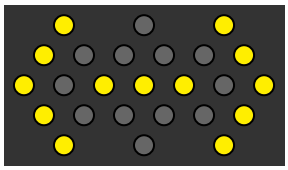
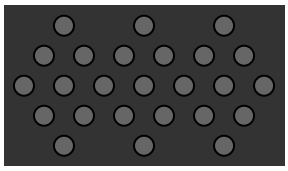
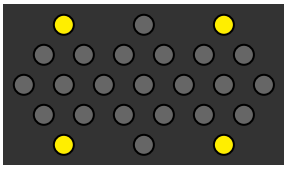
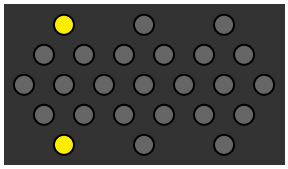
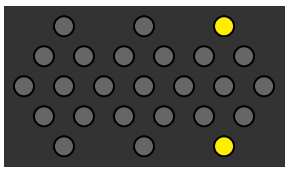
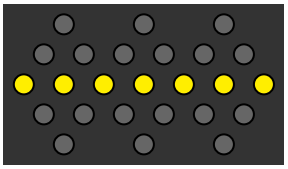
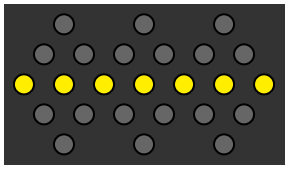
There are sixteen patterns selectable with inputs 0-3. Input 4 enables flashing and input 5 enables an animated sequence.

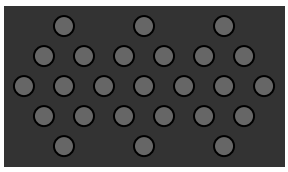
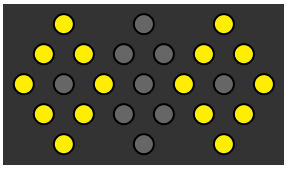
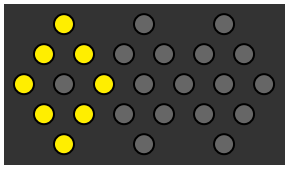
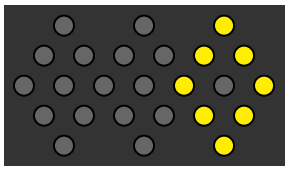
Pattern	Steady/Flashing	Animated Sequence
0		  
1		 

		
2		  
3		  
4		 

		
5		  
6		  
7		 

		
8		  
9		  
10		 

		
11		  
12		 
13		 
14		

		
15		 

How to test

Set `ui_in[7]` low and `ui_in[6]` low. All outputs should be low.

Set `ui_in[7]` high and `ui_in[6]` low. All outputs should be high.

Set `ui_in[3:0]` to one of the sixteen patterns. Set `ui_in[4]` low and `ui_in[5]` low. Set `ui_in[6]` high. Pulse `clk` at a desired rate. Outputs should reflect a steady pattern.

Set `ui_in[3:0]` to one of the sixteen patterns. Set `ui_in[4]` high and `ui_in[5]` low. Set `ui_in[6]` high. Pulse `clk` at a desired rate. Outputs should alternate between a steady pattern and a blank pattern.

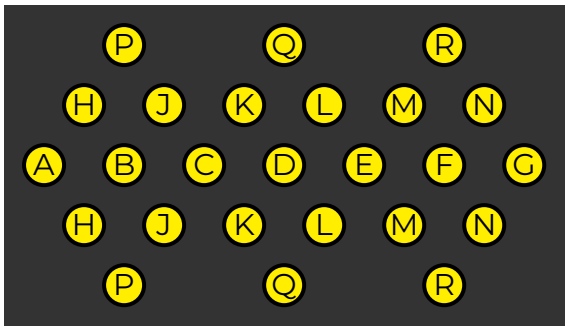
Set `ui_in[3:0]` to one of the sixteen patterns. Set `ui_in[4]` low and `ui_in[5]` high. Set `ui_in[6]` high. Pulse `clk` at a desired rate. Outputs should reflect a sequential pattern with a repeated last frame.

Set `ui_in[3:0]` to one of the sixteen patterns. Set `ui_in[4]` high and `ui_in[5]` high. Set `ui_in[6]` high. Pulse `clk` at a desired rate. Outputs should reflect a sequential pattern with a blank last frame.

Special cases: The animated versions of patterns 12 and 14 behave the same as the flashing versions and are unaffected by input 4. The animated versions of patterns 13 and 15 always alternate between the same two frames without repeated or blank frames.

External hardware

An array of 25 LEDs in the following configuration:



Lamp	Output	Lamp	Output
A	`uo_out[0]`	J	`uio_out[0]`
B	`uo_out[1]`	K	`uio_out[1]`
C	`uo_out[2]`	L	`uio_out[2]`
D	`uo_out[3]`	M	`uio_out[3]`
E	`uo_out[4]`	N	`uio_out[4]`
F	`uo_out[5]`	P	`uio_out[5]`
G	`uo_out[6]`	Q	`uio_out[6]`
H	`uo_out[7]`	R	`uio_out[7]`

Set `ui_in[7]` high if a high level lights the LED or low if a low level lights the LED.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	pattern[0]	lamp A	lamp J
1	pattern[1]	lamp B	lamp K
2	pattern[2]	lamp C	lamp L
3	pattern[3]	lamp D	lamp M
4	flashing	lamp E	lamp N
5	sequential	lamp F	lamp P
6	/LT (lamp test)	lamp G	lamp Q
7	/AL (active low)	lamp H	lamp R

Tiney Tapeout Semiconduxtor

by **Vikrant Yadav**

0162 Wokwi Project

github.com/7788ybfqbz-cmyk/Tiny-Tapeout-Semiconductor

wokwi.com/projects/445255081544375297

"Each input turns on its cooresponding output"

How it works

Inverter 1-8 is connected to input 0-7, each inverter invertes the input and activates the cooresponding light on the clock.

How to test

input 0 | inverter | output 0 |
0 | 1 | 1|

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input_0	output_inverter1	—
1	input_1	output_inverter2	—
2	input_2	output_inverter3	—
3	input_3	output_inverter4	—
4	input_4	output_inverter5	—
5	input_5	output_inverter6	—
6	input_6	output_inverter7	—
7	input_7	output_inverter8	—

Ring Counter

by **Zachary Teper**

0163 | 1 Hz | Wokwi Project

`github.com/ZTeper/ieee_hardware_hack_days`

`wokwi.com/projects/444269677791635457`

"Ring Counter with 8 inputs and 8 outputs + clock and reset"

How it works

This project is a ring counter that shifts binary data to the right and rotates around. Each input pin can be used to set each of the bits HIGH. The CLK input is used to shift the register. The RST_N input is asserted HIGH to rotate the counter, or asserted LOW to clear the data.

How to test

Set RST_N to HIGH, set IN0 to HIGH, then toggle the CLK a few times to inject 1s into the counter. Set IN0 to LOW, then toggle the CLK a few times to inject 0s into the counter. The counter will rotate the data around on each rising clock edge.

External hardware

Connect a push-button to the CLK. Connect a slide-switch to RST_N. Connect a set of 8 DIP switches to the data inputs. Connect a set of 8 LEDs (with current limiting resistors) to the data outputs.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	IN7	OUT7	—

Svens Woki design

by **Sven**

0164 Wokwi Project

`github.com/sven24git/cooypy`

`wokwi.com/projects/445255326319805441`

“ My project connects the switch to the light”

How it works

My project connects the siwtch to the light.

How to test

By using the switch

External hardware

N/A

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	AND	AND	—
1	AND	AND	—
2	inverter	inverter	—
3	inverter	inverter	—
4	inverter	inverter	—
5	inverter	inverter	—
6	wire	wire	—
7	wire	wire	—

Tiny Tapeout Test Gates

by **Aman Barolia**

0165 Wokwi Project

github.com/Aman-Barolia/TinyTapeout

wokwi.com/projects/445254953748100097

"This projects tests logic gates and a clock divider. It has been designed to show a resembilng output of the greek alphabet lambda."

How it works

This project uses XOR and NOT gates to connect to inputs a, b, c, d, e, f, g, and h. hhhhh

How to test

Set the inputs and check if they match with the outputs. hhhh

External Hardware

List external hardware used in your project (e.g. PMOD, LED Display, etc), if anyjjj

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a, XOR gate	output a, XOR gate	input a, XOR gate
1	input b, XOR gate	output b, NOT gate	input b, XOR gate
2	input c, NOT gate	output c, NOT gate	input c, NOT gate
3	input d, NOT gate	output d, NOT gate	input d, NOT gate
4	input e, NOT gate	output e, XOR gate	input e, NOT gate
5	input f, XOR gate	o5	input f, XOR gate
6	input g, XOR gate	output g, NOT gate	input g, XOR gate
7	input h, NOT gate	o7	input h, NOT gate

Tiny Tapeout Project - Binary to Decimal Display

by **Ovi M**

0166 Wokwi Project

github.com/ovidevan/ovi_tinytapeout

wokwi.com/projects/446565876953355265

"Testing Logic Gate - Binary to decimal converter (3bits)"

How it works

A 3 bit Binary to decimal segment display converter.

How to test

Toggle the input swtiches (B2, B1, B0) to convert binary input to decimal numbers displayed on the output segment (0 to 7 can be displayed).

Used Truth table to design the binary to decimal converter using AND/OR gates.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in1 B0	connected to display segment	—
1	in2 B1	connected to display segment	—
2	in3 B2	connected to display segment	—
3	in4 Not Connected	connected to display segment	—
4	Not Connected	connected to display segment	—
5	Not Connected	connected to display segment	—
6	in7 Not Connected	connected to display segment	—
7	in8 Not Connected	Not connected	—

FullAdder

by **InkSpireTo**

0167 Wokwi Project

`github.com/InkSpiredTO/Celine_GDS`

`wokwi.com/projects/445263189281882113`

"This is an adder!"

How it works

This is an adder. (https://en.wikipedia.org/wiki/Adder_(electronics)) It uses AND, OR, XOR.

How to test

Turn on the switches. Yay.

External hardware

7 segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output xor	—
1	input b	output or	—
2	input c	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny Tapeout Template Copy

by **Andrej Nedeljkovic**

0168 | 10 kHz | Wokwi Project

github.com/AndrejKai/Tiny-Tapeout-Template-Copy
wokwi.com/projects/445260761196909569

"Test of logic gates"

How it works

input a,b,c,d,e,f,g,h

How to test

change the inputs: a,b,c,d,e,f,g,h from 0 to 1 or vice versa

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output not	—
2	input c	output xor	—
3	input d	—	—
4	input e	output or	—
5	input f	output not	—
6	input g	output or or	—
7	input h	output or	—

Project Title

by **My name**

0169 | 1 kHz | Wokwi Project

`github.com/acidicsolution/tinytapeouttraining`
`wokwi.com/projects/444910895295771649`

"This is a description"

How it works

This would be a description of how this circuit works.

How to test

If you are reading this, I am doing this for training purposes.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input1	Output1	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

tinytapeout

by **Clayton Mo**

0170 Wokwi Project

github.com/froogsarecool/tinytapeout

wokwi.com/projects/445265925317879809

"project"

How it works

NOT gates are connected to input a and b respectively. OR gate is connected to inputs c and d. XNOR gate is connected to inputs e and f. XOR gate is connected to inputs g and h.

How to test

Set the inputs and check the outputs match with the expected sequence results.

Test case 1: Set all inputs to 0 except for a and b. Output XNOR = 1. All other outputs are 0.

Test case 2: Set inputs c and e = 0. Set all other inputs to 1. Output XNOR = 1. Output OR = 1.

Test case 3: Set inputs a, c, e, and g = 0. Set all other inputs to 1. First output NOT = 1. Second output NOT = 0. Output XOR = 1.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output not	—
1	input b	output not	—
2	input c	—	—
3	input d	—	—
4	input e	—	—
5	input f	output or	—
6	input g	output xnor	—

#	Input	Output	Bidirectional
7	input h	output xor	—

Jarle Chip

by **Cole** & **Jarod**

0171 Wokwi Project

`github.com/crrthompson2014-star/jarle-Tiny-Tapeout`

`wokwi.com/projects/445276353959823361`

"Logic Conversion Circuit"

How it works

This project counts in binary from 0 - 7 and converts it to the outputs for a 7-segment display with the push of a button.

How to test

Reset the counter by pulling pin 0 high, and add one to the counter by pulling the clock pin high. You should see the 7-segment display increment by one on each press of the button.

External hardware

Two buttons and a single 7-segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IncreaseCount	G	—
1	—	F	—
2	—	E	—
3	—	D	—
4	—	C	—
5	—	B	—
6	—	A	—
7	—	—	—

MPW-2 poly layers – Designed by Matt Venn. Illustrated by Máximo Balestrini.

Tiny Tapeout Test Gates

by **Aandre Jahangir**

0172 Wokwi Project

`github.com/D1x1eFlat1ne/Practice_Digital_Logic`
`wokwi.com/projects/445255578007454721`

"Test some logic gates"

How it works

XOR gate are connected to input 1 and 2, connected to output 0

How to test

A | B | Out 0 | 0 | 0 1 | 0 | 1 0 | 1 | 1 1 | 1 | 0

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input A XOR gate	out xor gate	—
1	input B XOR gate	—	—
2	—	out not	—
3	—	out not	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Wokwicreation

by **Anu M**

0173 Wokwi Project

github.com/Anudevan77/Anutiny

wokwi.com/projects/445264784091688961

"test logic gate"

How it works

AND, NAND, and OR gates are connected to input a and b. w3 logic gates connected to the output

How to test

use the stiff.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in 1 Nand	out Nand	—
1	in2 NAND	out2	—
2	in2	direct out from switch[2]	—
3	in1 OR	out OR	—
4	in2 OR	out4	—
5	in5	direct out from switch[5]	—
6	in1 OR	out6	—
7	in2 OR	out OR	—

sigma uncount

by **Tyler**

0174 Wokwi Project

`github.com/migraine7341/wokwi`

`wokwi.com/projects/445256528755654657`

"xor spam"

How it works

7 consecutive xnor gates linking with the previous one + number input

How to test

read above and figure it out

previous gate xnor current number	output xnor
0 0	1
1 0	0
0 1	0
1 1	1

multiple outputs will produced varying displays

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in 1	out 1	—
1	in 2	out 2	—
2	in 3	out 2	—
3	in 4	out 3	—
4	in 5	out 4	—
5	in 6	out 5	—
6	in 7	out 6	—
7	in 8	out 7	—

Tiny Tapeout Test Games

by **Athirai M**

0175 Wokwi Project

github.com/athirai65/Tiny-Tapeout---1
wokwi.com/projects/445262455513487361

"Test some logic gates."

How it works

AND and OR gates are connected to input a and b.

How to test

Set input a and b high. Look for out 0 to go high.

External hardware

Default

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input a AND Gate	Output a of AND Gate	—
1	Input b AND Gate	Output b of AND Gate	—
2	Input a of OR Gate 1	—	—
3	Input b of OR Gate 1	Output a of OR Gate 1	—
4	Input a of OR Gate 2	Output b of OR Gate 1	—
5	Input b of OR Gate 2	Outout a of OR Gate 2	—
6	—	—	—
7	—	—	—

VGA clock

by **Matt Venn**

0192 | 31.5 MHz | HDL Project

`github.com/mattvenn/tt08-vga-clock`

"Shows the time on a VGA screen"

How it works

Races the beam! Font is pre generated and loaded into registers. 6 bit colour keeps register count low.

Every minute the colours cycle.

How to test

Hook up a VGA monitor to the outputs and provide a clock at 31.5 MHz.

Adjust time with the inputs[2:0], and choose the type of VGA PMOD with the input[3].

External hardware

VGA PMOD - you can use one of these VGA PMODs:

- https://github.com/mole99/tiny-vga
- https://github.com/TinyTapeout/tt-vga-clock-pmod

Set input[3] low to use tiny-vga and high to use vga-clock

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	adjust hours	hsync / R1	—
1	adjust minutes	vsync / G1	—
2	adjust seconds	B0 / B1	—
3	PMOD type select	B1 / VS	—
4	—	G0 / R0	—
5	—	G1 / G0	—
6	—	R0 / B0	—
7	—	R1 / HS	—

Austrochip 2025 Workshop Design

by **Florian Huemer**

0193 | 50 MHz | Wokwi Project

`github.com/fhuemer/austrochip_tinytapeout_workshop`

`wokwi.com/projects/442984104166367233`

"a simple LFSR"

How it works

The design contains a 3 bit LFSR and NAND gate.

How to test

The result of the NAND operation between the inputs a and b is output on output 0. The inverted state of the LFSR is output on 5-7.

External hardware

Switches for the inputs, LEDs for the outputs.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	nand_out	—
1	b	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	lfsr0_n	—
6	—	lfsr1_n	—
7	—	lfsr2_n	—

RGB Mixer demo

by **Matt Venn**

0194 | 10 MHz | HDL Project

`github.com/mattvenn/ttsky25a-rgb-mixer`

"Zero to ASIC demo project"

How it works

Debounce the inputs, drive an encoder module, and output a PWM signal for each encoder.

How to test

Twist each encoder and the LEDs attached to the outputs should change in brightness.

By setting the debug port to 0, 1 or 2, the internal value of each encoder is output on the bidirectional outputs.

External hardware

Use 3 digital encoders attached to the first 6 inputs.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	enc0 a	pwm0	encoder bit 0
1	enc0 b	pwm1	encoder bit 1
2	enc1 a	pwm2	encoder bit 2
3	enc1 b	—	encoder bit 3
4	enc2 a	—	encoder bit 4
5	enc2 b	—	encoder bit 5
6	debug bit 0	—	encoder bit 6
7	debug bit 1	—	encoder bit 7

Full Adder

by **Philipp**

0195 Wokwi Project

`github.com/psommeregger/ttsky-wokwi-template`
`wokwi.com/projects/442978797853619201`

"Required"

How it works

A simple full adder

How to test

Don't

External hardware

Not sponsored...

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in0	out0	—
1	in1	out1	—
2	in2	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

spi_slave_ctrl

by **felixzhou**

0196 | 50 MHz | HDL Project

`github.com/zhoufelix0558/spi_slave`

"spi slave for register r&w"

How it works

This Module simulate a simple spi slave controller for internal registers read&write. sys_clk should 4x faster than spi_clk at least.

How to test

External hardware

a external controller connect spi_ncs/spi_clk/spi_mosi/spi_miso

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	SPI_NCS	SPI_MISO	—
1	SPI_CLK	FIFO_OUT_SERIAL	—
2	SPI_MOSI	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

First Wokwi design

by **Tobias Matzinger**

0197 Wokwi Project

github.com/tjmatzinger-sudo/First-Github

wokwi.com/projects/442983541057844225

"Test some logic gates"

How it works

two nands connected in a way to build a simple flip-flop

How to test

switch the 1 switch on and only then you can turn the light on and off with the second switch

External hardware

7-segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	S1	L1	not used14
1	S2	not used7	not used15
2	not used1	not used8	not used16
3	not used2	not used9	not used17
4	not used3	not used10	not used18
5	not used4	not used11	not used19
6	not used5	not used12	not used20
7	not used6	not used13	not used21

Jsilicon

by **JunHyeok Seo (mirseo)**

0198 | 12 MHz | Wokwi Project

`github.com/mirseo/JSilicon`

`wokwi.com/projects/0`

"A dual-mode 8-bits CPU/ALU core with Manual and Auto (ROM) modes, built in Verilog."

JSilicon v0.2 – A Dual-Mode 8-bit CPU/ALU Core

A JavaScript-Inspired Prototype Built Under Constraint

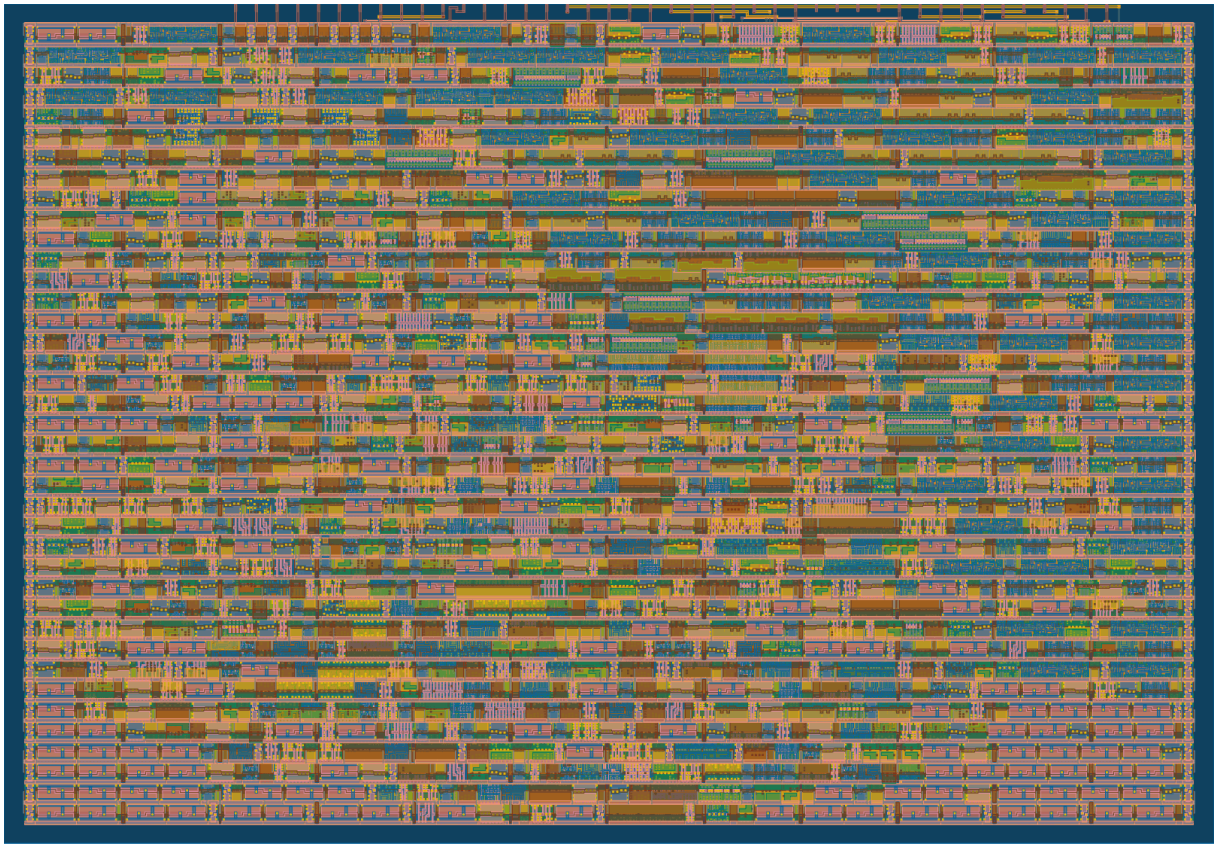


Figure 198.1: JSilicon Render Image

JSilicon is an **8-bit CPU/ALU core** designed and implemented from scratch during my mandatory military service in South Korea (2025). This project serves as a proof-of-concept, showing that a complete silicon design is achievable even in highly constrained environments.

Version 0.2 expands on the original manual ALU functionality by adding a CPU mode that automatically executes pre-programmed instructions. To enable this, key CPU components such as a Program Counter (PC), an instruction decoder, and a register file have been integrated.

Inspired by JavaScript's simplicity and the philosophy of accessible silicon design, the JSilicon series aims to develop an ASIC that can natively power a JS runtime.

Overview

- **PC (Program Counter & ROM)** - Stores a 16x8-bit instruction set in its internal ROM and sequentially fetches them in CPU mode.
- **Decoder** - Parses instructions from PC and generates control signals for other components.
- **REG (Register File)** - Contains two 8-bit general-purpose registers (R0, R1) that serve as the CPU's workspace.
- **ALU (Arithmetic Logic Unit)** - Performs eight fundamental arithmetic and logic operations, including addition, subtraction, and multiplication.
- **SWITCH** - A multiplexer that selects the data path based on the Mode pin, switching between external manual inputs and the internal CPU core.
- **FSM (Finite State Machine)** - Acts as the central controller, managing the timing for ALU execution and UART transmission.
- **UART_TX** - Serializes the computation result and transmits it to an external device like a PC or MCU.

Pinout

Pin	Direction	Description
`clk`	Input	System clock (12 MHz)
`rst_n`	Input	Active-low reset
`ena`	Input	Chip enable (Active-High)
`ui_in[7:4]`	Input	**(Manual Mode)** Operand A (4-bit)
`ui_in[3:0]`	Input	**(Manual Mode)** Operand B (4-bit)
`uio_in[7:5]`	Input	**(Manual Mode)** Opcode (3-bit)
`uio_in[4]`	Input	**Mode Select** (0: Manual, 1: CPU)
`uo_out[7]`	Output	UART Busy Status (1: Busy)
`uo_out[6:0]`	Output	ALU Result, bit [6:0] (Lower 7 bits) (`alu_result[6:0]`)
`uio_out[7:1]`	Output	ALU Result, bits [15:8] (Upper 8 bits) (`alu_result[15:8]`)
`uio_out[0]`	Output	UART TX serial data output (9600 bps)

How to test

1. **Manual Mode(Mode = 0)**
   Set the `uio_in[4]` pin to '0' to use the core as a manual ALU.
   A. Provide operands: Provide **Operation A** on `ui_in[7:4]` and **Operation B** on `ui_in[3:0]` (4 bits each).

   B. Choose operation: Set the desired operation using the Opcode on `uio_in[7:5]`.
   - `000` : A + B
   - `001` : A - B
   - `010` : A * B
   - `011` : A / B
   - `100` : A % B
   - `101` : A == B
   - `110` : A > B
   - `111` : A < B

   C. Read the result: The result will instantly appear on the `uo_out` and `uio_out` pins.

   D. Serial output (optional): The same result is sent via UART on the `uio_out[0]` pin.
   - Connect a USB-to-serial adapter (9600 bps, 8N1) to read it on a PC or MCU.
   - `uio_out[0]` reflects the UART TX line state for monitoring.

   E. Reset the design: Drive `rst_n` low to reset the FSM and ALU state, then bring it high again to start a new computation.

2. **CPU Mode (Mode = 1)**
   Set the `uio_in[4]` pin to '1' to run the core in automatic CPU mode.

   1. Once the chip is enabled (`ena` = 1), it will automatically execute the program stored in its internal ROM.
   2. External inputs `ui_in` and `uio_in[7:5]` are ignored in this mode.
   3. The default built-in program is as follows:
      - `ADD 3`
      - `SUB 2`
      - `MUL 5`
      - `NOP` (then, repeats)
   4. The result of each instruction is streamed to the `uo_out` and `uio_out` pins and also transmitted via UART.

Notes

- **Clock** : Design expects a 12Mhz input clock. (TinyTapeout standard)
- **Logic Levels** : All I/O pins use 3.3 V CMOS Logic
- **Bidirectional Pins** : The uio pins are used for both input and output. As inputs, uio[7:5] select the manual opcode and uio[4] selects the operation mode. As an output, only uio[0] is actively driven for UART TX. The remaining pins (uio[3:1]) are unused.

Vision

JSilicon is not just a chip - it's a story of building silicon under constraints.

This first version was created entirely during mandatory military service in South Korea, demonstrating that hardware innovation is possible even in the most limited environments. Future versions will expand JSilicon into a more capable CPU core RISC-like capabilities.

Milestone - JSilicon v0.2 GDS Layout

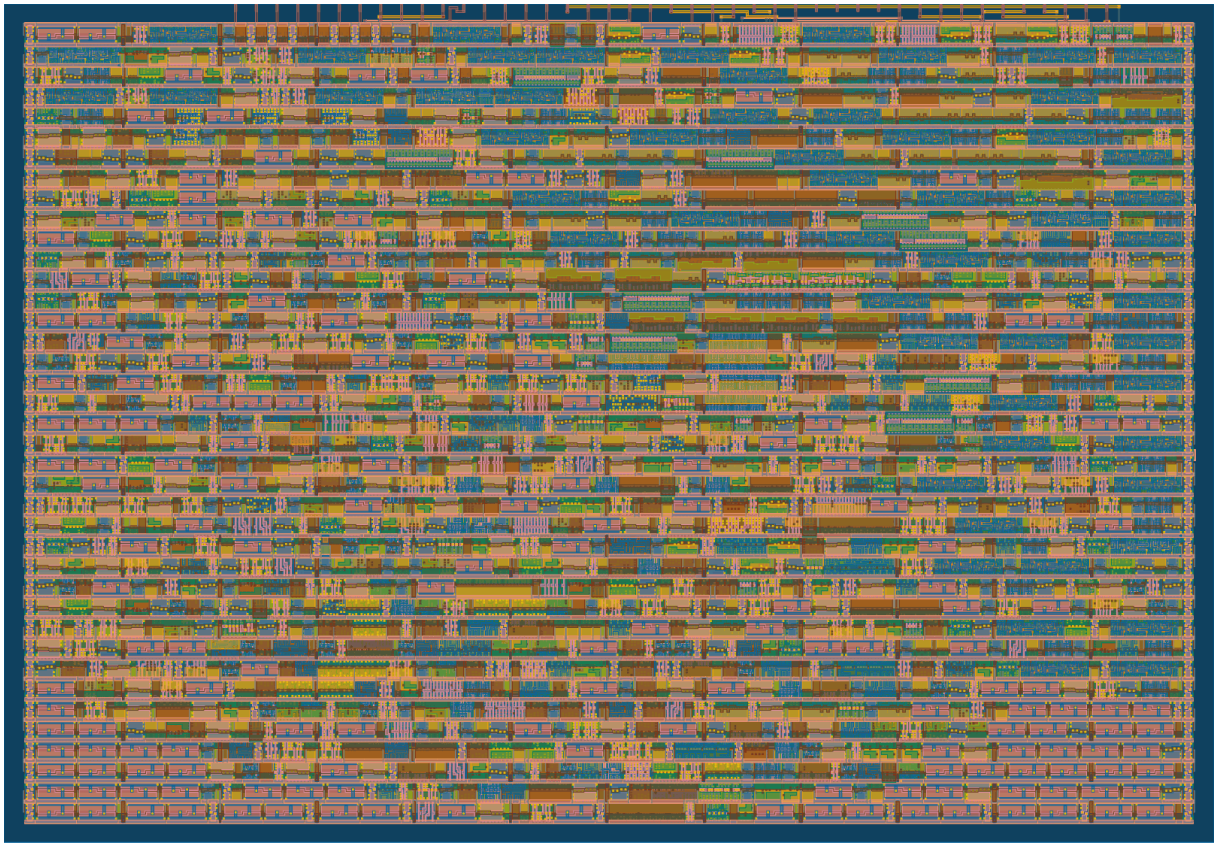


Figure 198.2: JSilicon GDS Layout

In October 2025, JSilicon v0.2 reached a major milestone:
the successful generation of a complete **GDSII layout**, marking the transition from logic design to physical silicon.

Explore the physical layout of **JSilicon v0.2** in full 3D.
This interactive viewer lets you navigate through the final GDSII structure of the chip — from standard cells to routing layers — exactly as it will appear on silicon.

View JSilicon v0.2 GDS Layout in 3D

License

This project is licensed under the MIT License.

Author Message

Hello, I'm JunHyeok Seo, currently serving my mandatory military service here in the Republic of Korea.

The reason I created the JSilicon project was to prove that even time spent in the military can be meaningful. And I will see it through. Even when there seems to be no path forward, I will find a way, just as I always have.

You can create something like this even with a low-spec computer or on a device that lags and crashes when you open just one or two browser tabs. Do not give up. Just as I managed to build this chip, you can achieve it too.

Copyright 2025. JunHyeok Seo (mirseo). All rights reserved.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Operand B bit[0]	ALU result[0]	UART TX Output
1	Operand B bit[1]	ALU result[1]	—
2	Operand B bit[2]	ALU result[2]	—
3	Operand B bit[3]	ALU result[3]	—
4	Operand A bit[0]	ALU result[4]	Mode Select (0:Manual, 1:CPU)
5	Operand A bit[1]	ALU result[5]	Manual Opcode[0] (LSB)
6	Operand A bit[2]	ALU result[6]	Manual Opcode[1]
7	Operand A bit[3]	UART Busy Status	Manual Opcode[2] (MSB)

Digital Filter

by **Gregor Flachs**

0199 HDL Project

`github.com/FlachsGregor/7_Segment_Display_Controller`

"A third order digital FIR filter with 8-bit architecture, configurable cut-off frequency and selectable low-pass or high-pass behaviour."

TinyTapeout: Digital FIR Filter

Module: `tt_um_df_top`

A **third order digital FIR filter** for TinyTapeout. It has a **configurable cut-off frequency** and **two selectable behaviours**. The sampling frequency is equal to the clock frequency. Results are available after **one clock-cycle**. Data is **unsigned** and the output is bounded to fit 8 bits.

Features

- Four cut-off frequencies: **0.125, 0.375, 0.625 and 0.875**
- Two filter behaviours: **lowpass and highpass**
- **Single-cycle execution**
- **No overflows**

Top-Level I/O (TinyTapeout)

Signal	Dir	W	Description
`clk`	in	1	System clock
`rst_n`	in	1	Async reset (active-low)
`ena`	in	1	Always '1' on TinyTapeout
`ui_in[7:0]`	in	8	*Unsigned* **input data**
`uo_out[7:0]`	out	8	*Unsigned* **output data**
`uio_in[7:4]`	in	4	*Unused* - should be set to '0'
`uio_in[3]`	in	1	**Config enable** (active-high) - change current config
`uio_in[2]`	in	1	Config **filter behaviour** ('0' = lowpass, '1' = highpass)
`uio_in[1:0]`	in	2	Config **cut-off frequency** (wg = b0.xx1)
`uio_out[7:0]`	out	8	*Unused* - set to '0'
`uio_oe[7:0]`	out	8	*Unused* - set to '0'

Reset defaults:

- Filter behaviour: **lowpass**
- Cut-off frequency: **0.125**

Testing on hardware

1. Connect **DAC + oscilloscope** to output.
2. Connect **ADC + signal generator** to input.
3. **Reset** filter.
4. (Optional) **Configure** filter.
5. Test filter with **different input signals**.

Notes

1. Reset before changing config.
2. Output signal may be slightly cut off.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data_in[0]	data_out[0]	config_wg[0]
1	data_in[1]	data_out[1]	config_wg[1]
2	data_in[2]	data_out[2]	config_hp
3	data_in[3]	data_out[3]	config_en
4	data_in[4]	data_out[4]	—
5	data_in[5]	data_out[5]	—
6	data_in[6]	data_out[6]	—
7	data_in[7]	data_out[7]	—

SimplePattern

by **Poorn**

0200 Wokwi Project

`github.com/Poorn-Mehta/TinyTapeoutWorkshop`

`wokwi.com/projects/413387009513254913`

"Display P on 7-seg display when 0x46 is inputted"

How it works

This is a very simple project that displays 'P' on the 7-segment display, when the 8-bit input is 0x46. This is achieved through simple logic design & a few number of basic logic gates.

How to test

By default, when all inputs are 0 (0x00), the 7-segment output should have inversed segment states required for displying P (which basically means that if you turn OFF all segments required for P and turn ON the rest of the 'digit' making segments then that's what it would be showing). It wasn't intentional but I think it's pretty cool that it turned out this way.

As different patterns are inputted, different segments of the display will turn on/off. When the right combination (0x46 –> IN1, IN2, IN6 are logic 1 and every other INs are 0) is provided in the input, the 7-segment display should light up segments to display P (OUT2, OUT3, OUT7 are OFF, every other is lit up or ON).

External hardware

A breadboard, 1.8V power supply, some connector wires, 8 position DIP switch, and a 7 segment display is required.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	—	—
1	—	OUT1	—
2	IN2	—	—
3	IN3	—	—
4	—	OUT4	—

#	Input	Output	Bidirectional
5	IN5	OUT5	—
6	IN6	—	—
7	IN7	—	—

4-Bit ALU

by **Ilir Shala**

0201 | 50 MHz | Wokwi Project

github.com/EngGhaith/TEXT2XY-SCOPE

wokwi.com/projects/0

"Mini 4-Bit ALU"

How it works

This design implements a 4-bit ALU with an integrated 8×4-bit register file.

Inputs are mapped as follows:

signal function ui_in[3:0] operand A (4-bit) ui_in[7:4] operand B (4-bit) / register address uio[3:0] opcode (4-bit) clk TinyTapeout clock rst_n active-low reset

The ALU supports basic arithmetic and logical operations as well as register-file access:

opcode function 0000 A + B 0001 A − B 0010 A AND B 0011 A OR B 0100 A XOR B 0101 A << B[1:0] 0110 A >> B[1:0] 0111 pass B 1000 write A → reg[B] 1001 read reg[B] 1010 A + reg[B] 1011 A − reg[B]

The output uo_out[7:0] contains the result and flags:

bit meaning 7 ZERO flag (result=0) 6 SIGN flag (MSB of result) 5 OVERFLOW flag (signed add/sub) 4 CARRY flag (unsigned add/sub) 3:0 result (4-bit)

How to test

Use DIP switches or microcontroller GPIO lines to drive ui_in and uio pins.

Examples:

write value 7 into register 3 ui_in = { B=3 , A=7 }, opcode=1000

read register 3 ui_in = { B=3 , A=don't care }, opcode=1001

add register 3 + A (A on ui_in lower nibble) opcode=1010

You can observe the result and flags on the 8 output pins.

External hardware

None required.

For demo you can use:

8 DIP switches for ui_in 4 DIP switches for opcode 8 LEDs for uo_out

A simple MCU board (Arduino / Pico) makes more comfortable testing, but is optional.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	ui_in[0]	uo_out[0]	uio_in[0]
1	ui_in[1]	uo_out[1]	uio_in[1]
2	ui_in[2]	uo_out[2]	uio_in[2]
3	ui_in[3]	uo_out[3]	uio_in[3]
4	ui_in[4]	uo_out[4]	—
5	ui_in[5]	uo_out[5]	—
6	ui_in[6]	uo_out[6]	—
7	ui_in[7]	uo_out[7]	—

Crazy logic gates

by **Fatemeh** & **Yaning**

0202 | 10 kHz | Wokwi Project

github.com/Atlantis961/YES

wokwi.com/projects/445165988923203585

"The craziest logic gates ever made"

How it works

It is super complicated, even I don't know. It's just a bunch of gates that are thrown together to see how they will react with each other. A fun experiment.

How to test

Just hook it up to a seven-segment or an LED bar so see if it works or not. When everything is off the seven-segment should show 4.

External hardware

DIP switch, resistors, and LED bar graph/seven-segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a AND b	—
1	input b	output NOT	—
2	input c	output XNOR	—
3	input d	output MUX	—
4	input e	output OR	—
5	input f	output NAND	—
6	input g	output NOR	—
7	input h	output XOR	—

Lauflichterweitung

by **Andlinger Alexander**

0203 | 10 kHz | Wokwi Project

github.com/HE210317/Lauflichterweiterung

wokwi.com/projects/442987944543304705

"abc"

How it works

i di it later

How to test

later

External hardware

later

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	t1	a1	—
1	t2	b1	—
2	t3	c1	—
3	t4	d1	—
4	t5	e1	—
5	t6	f1	—
6	t7	g1	—
7	t8	h1	—

TinyTapeoutProject

by **Ayan** & **Artur**

0204 | 10 kHz | Wokwi Project

`github.com/Ayan-Nalawade/TinyTapeoutTemplate`
`wokwi.com/projects/445172771884208129`

"Something"

How it works

Magic

How to test

Dark magic

External hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	OUT0	—
1	—	OUT1	—
2	IN0	OUT2	—
3	IN1	OUT3	—
4	IN3	OUT4	—
5	IN4	OUT5	—
6	—	OUT6	—
7	—	OUT7	—

Universal Binary to Segment Decoder

by **Rebecca G. Bettencourt**

0205 HDL Project

github.com/RebeccaRGB/ubcd

"Decodes various binary codes to various segmented displays."

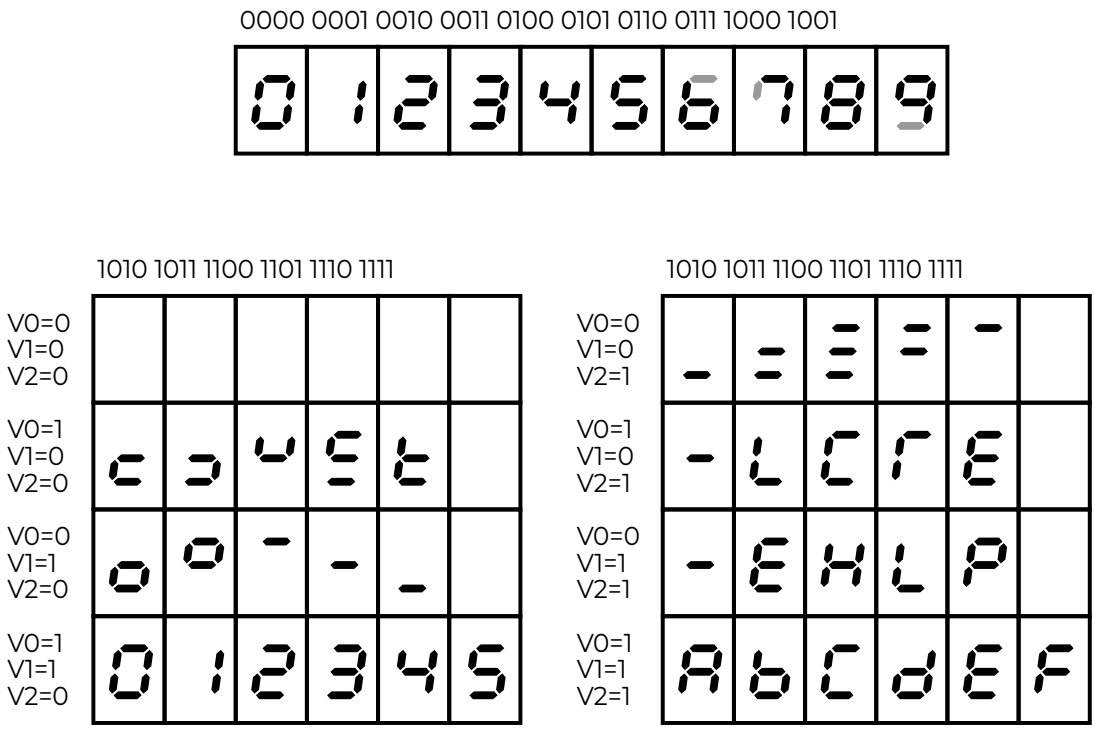
How it works

This project is composed of four modules:

- A BCD to seven segment decoder with a wide variety of options for customizing the appearance of digits
- An ASCII to seven segment decoder with two different "fonts"
- A dual BCD to Cistercian numeral decoder
- A BCV (binary-coded *vigesimal*) to Kaktovik numeral decoder

BCD to seven segment decoder

This mode converts a decimal digit in BCD to its representation on a standard seven segment display. There are inputs that affect the display of the digits 6, 7, and 9, and eight different options for handling out-of-range values. These inputs allow this decoder to match the behavior of just about any other BCD to seven segment decoder, making it *universal*.



The signals used in this mode are:

- **/AL** - Active low. If HIGH, outputs will be HIGH when lit. If LOW, outputs will be LOW when lit.
- **/BI** - Blanking input. If LOW, all segments will be blank regardless of other inputs, including **/LT**.
- **/LT** - Lamp test. When **/BI** is HIGH and **/LT** is LOW, all segments will be lit.
- **/RBI** - Ripple blanking input. If the BCD value is zero and **/RBI** is LOW, all segments will be blank.
- **V0**, **V1**, **V2** - Selects the output when the BCD value is out of range.
- **X6** - When HIGH, the extra segment **a** will be lit on the digit 6.
- **X7** - When HIGH, the extra segment **f** will be lit on the digit 7.
- **X9** - When HIGH, the extra segment **d** will be lit on the digit 9.
- **A**, **B**, **C**, **D** - BCD input from least significant bit **A** to most significant bit **D**.
- **a**, **b**, **c**, **d**, **e**, **f**, **g** - Outputs for a seven segment display.
- **/RBO** - Ripple blanking output. HIGH when BCD value is nonzero or **/RBI** is HIGH.

The pin assignments in this mode are:

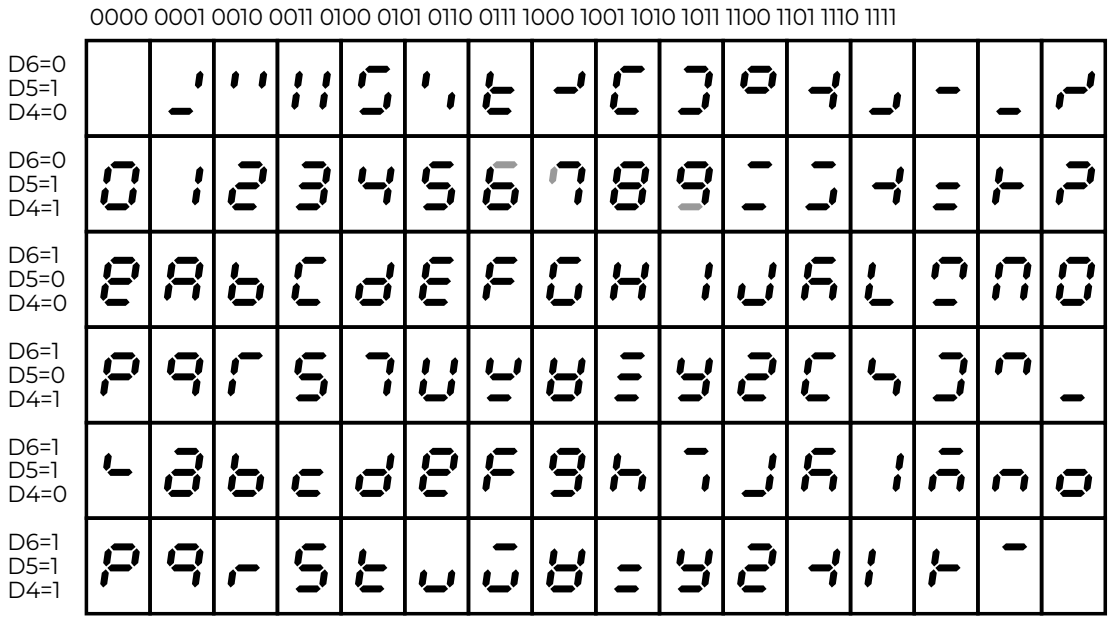
—	Dedicated Input	Dedicated Output	Bidirectional
0	A	Segment a	Input - X6
1	B	Segment b	Input - X7

2	C	Segment c	Input - X9
3	D	Segment d	Input - /LT
4	V0	Segment e	Input - /BI
5	V1	Segment f	Input - /AL
6	V2	Segment g	Input - LOW
7	/RBI	/RBO	Input - LOW

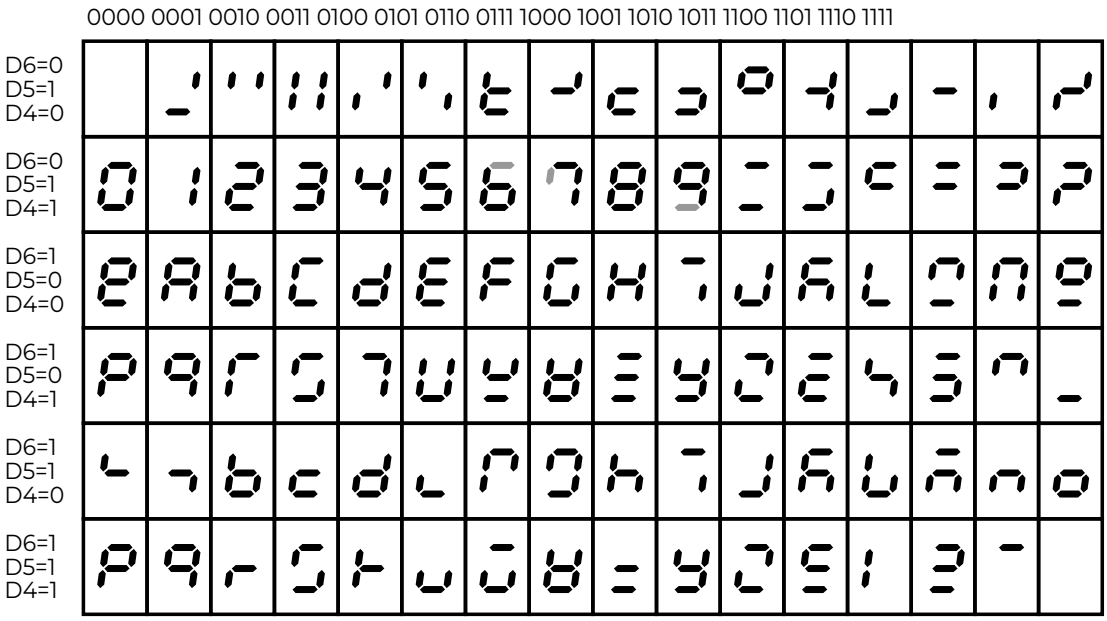
ASCII to seven segment decoder

This mode converts an ASCII character to a representation on a standard seven segment display. Like with the BCD decoder, there are inputs that affect the display of the digits 6, 7, and 9. There are also two choices of “font” and the option to display lowercase letters as uppercase or as lowercase.

FS=0:



FS=1:



The signals used in this mode are:

- **/AL** - Active low. If HIGH, outputs will be HIGH when lit. If LOW, outputs will be LOW when lit.
- **/BI** - Blanking input. If LOW, all segments will be blank regardless of other inputs.
- **FS** - Font select. Selects one of two “fonts.”
- **LC** - Lower case. If LOW, lowercase letters will appear as uppercase.
- **X6** - When HIGH, the extra segment **a** will be lit on the digit 6.
- **X7** - When HIGH, the extra segment **f** will be lit on the digit 7.
- **X9** - When HIGH, the extra segment **d** will be lit on the digit 9.
- **D0**...**D6** - ASCII input from least significant bit **D0** to most significant bit **D6**.
- **a**, **b**, **c**, **d**, **e**, **f**, **g** - Outputs for a seven segment display.
- **/LTR** - Letter. LOW when the input is a letter (A...Z or a...z).

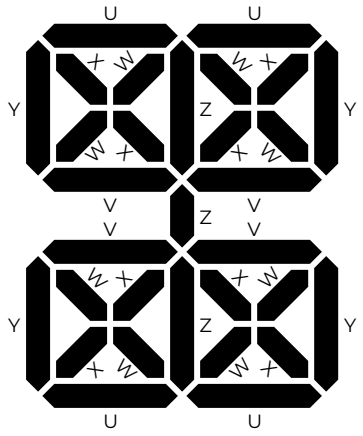
The pin assignments in this mode are:

—	Dedicated Input	Dedicated Output	Bidirectional
0	D0	Segment a	Input - X6
1	D1	Segment b	Input - X7
2	D2	Segment c	Input - X9
3	D3	Segment d	Input - FS
4	D4	Segment e	Input - /BI

5	D5	Segment f	Input - /AL
6	D6	Segment g	Input - HIGH
7	LC	/LTR	Input - LOW

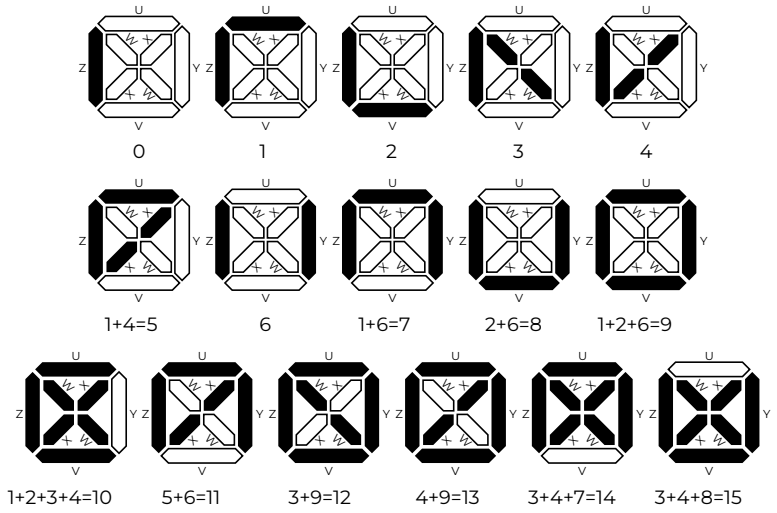
Dual BCD to Cistercian numeral decoder

This mode converts two decimal digits in BCD to their representations on the segmented display for Cistercian numerals shown below.



The patterns produced for each input value are shown below.

Patterns as seen in top right (units) position:



The signals used in this mode are:

- **/AL** - Active low. If HIGH, outputs will be HIGH when lit. If LOW, outputs will be LOW when lit.

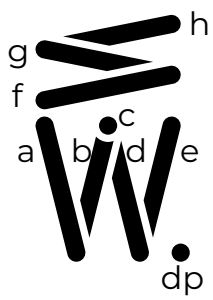
- **/BI** - Blanking input. If LOW, all segments will be blank regardless of other inputs, including **/LT1** and **/LT2**.
- **/LT1** - Lamp test for digit 1. When **/BI** is HIGH and **/LT1** is LOW, all segments for digit 1 will be lit.
- **/LT2** - Lamp test for digit 2. When **/BI** is HIGH and **/LT2** is LOW, all segments for digit 2 will be lit.
- **A1**, **B1**, **C1**, **D1** - BCD input for digit 1 from least significant bit **A1** to most significant bit **D1**.
- **A2**, **B2**, **C2**, **D2** - BCD input for digit 2 from least significant bit **A2** to most significant bit **D2**.
- **U1**, **V1**, **W1**, **X1**, **Y1** - Outputs for digit 1 on a Cistercian segmented display.
- **U2**, **V2**, **W2**, **X2**, **Y2** - Outputs for digit 2 on a Cistercian segmented display.

The pin assignments in this mode are:

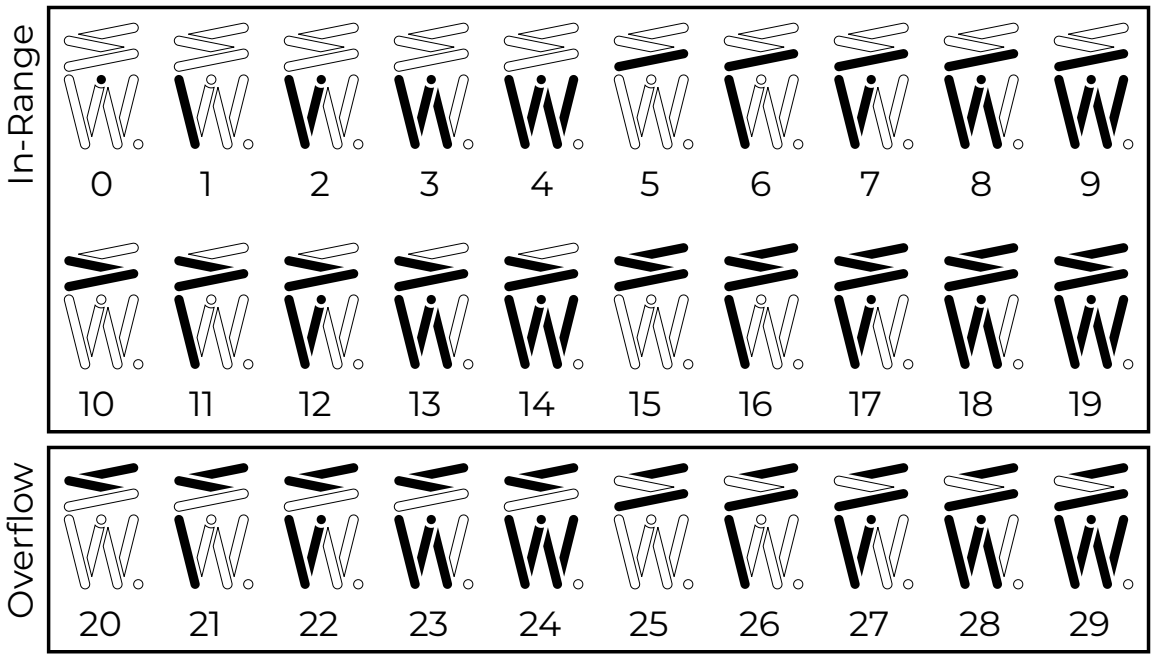
—	Dedicated Input	Dedicated Output	Bidirectional
0	A1	Segment U1	Output - Y1
1	B1	Segment U2	Output - Y2
2	C1	Segment V1	Input - /LT1
3	D1	Segment V2	Input - /LT2
4	A2	Segment W1	Input - /BI
5	B2	Segment W2	Input - /AL
6	C2	Segment X1	Input - LOW
7	D2	Segment X2	Input - HIGH

BCV to Kaktovik numeral decoder

This mode converts a *vigesimal* (base 20) digit in BCV (binary-coded vigesimal) to its representation on the segmented display for Kaktovik numerals shown below.



The patterns produced for each input value are shown below.



The signals used in this mode are:

- **/AL** - Active low. If HIGH, outputs will be HIGH when lit. If LOW, outputs will be LOW when lit.
- **/BI** - Blanking input. If LOW, all segments will be blank regardless of other inputs, including **/LT**.
- **/LT** - Lamp test. When **/BI** is HIGH and **/LT** is LOW, all segments will be lit.
- **/RBI** - Ripple blanking input. If the BCV value is zero and **/RBI** is LOW, all segments will be blank.
- **/VBI** - Overflow blanking input. If the BCV value is out of range and **/VBI** is LOW, all segments will be blank.
- **A**, **B**, **C**, **D**, **E** - BCV input from least significant bit **A** to most significant bit **E**.
- **a**, **b**, **c**, **d**, **e**, **f**, **g**, **h** - Outputs for a Kaktovik segmented display.
- **/RBO** - Ripple blanking output. HIGH when BCV value is nonzero or **/RBI** is HIGH.
- **V** - Overflow. HIGH when BCV value is out of range (greater than or equal to 20).

The pin assignments in this mode are:

—	Dedicated Input	Dedicated Output	Bidirectional
0	A	Segment a	Output - h
1	B	Segment b	Output - V
2	C	Segment c	—
3	D	Segment d	Input - /LT

4	E	Segment e	Input - /BI
5	—	Segment f	Input - /AL
6	/VBI	Segment g	Input - HIGH
7	/RBI	/RBO	Input - HIGH

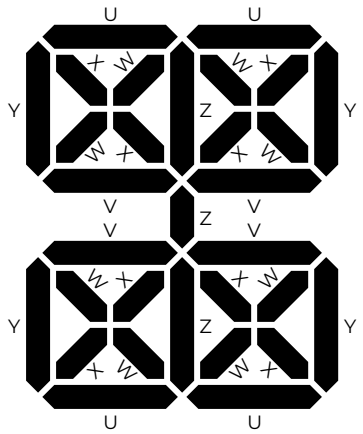
How to test

The `test` directory includes extensive tests for each of the four modules.

External hardware

For the BCD and ASCII modes, a standard seven-segment display is used.

For the Cistercian mode, a segmented display like the one below is used. There are design files for such a display here.



For the Kaktovik mode, a segmented display like the one below is used. There are design files for such a display here.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	A; D0; A1; A	Segment a; U1; a	X6; X6; Y1; h
1	B; D1; B1; B	Segment b; U2; b	X7; X7; Y2; V
2	C; D2; C1; C	Segment c; V1; c	X9; X9; /LT1; -
3	D; D3; D1; D	Segment d; V2; d	/LT; FS; /LT2; /LT
4	V0; D4; A2; E	Segment e; W1; e	/BI (blanking input)
5	V1; D5; B2; -	Segment f; W2; f	/AL (active low)
6	V2; D6; C2; /VBI	Segment g; X1; g	M0 (mode select)
7	/RBI; LC; D2; /RBI	/RBO; /LTR; X2; /RBO	M1 (mode select)

base-4-circuit-addition

by **AQUAWarrior** & **DigitalDemon**

0206 Wokwi Project

`github.com/AQUA-Warrior/base-4-addition-circuit`

`wokwi.com/projects/445163720250008577`

"Basic base 4 addition using 8 switch design"

How it works

- Uses 8 pins split into 2 different groups (A and B)
- Does binary addition between the two sets of 4 switches.

How to test

If programmed properly: Switch 0, switch 3 should output 2 only

External hardware

Buzzer 5x - LED

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Adder Pin 1 (A)	Output Pin 1	—
1	Adder Pin 2 (A)	Output Pin 2	—
2	Adder Pin 3 (A)	Output Pin 3	—
3	Adder Pin 4 (A)	Output Pin 4	—
4	Adder Pin 1 (B)	Output Pin 5	—
5	Adder Pin 2 (B)	—	—
6	Adder Pin 3 (B)	—	—
7	Adder Pin 4 (B)	—	—

Hardware UTF Encoder/Decoder

by **Rebecca G. Bettencourt**

0207 HDL Project

github.com/RebeccaRGB/hardware-utf8

"Converts Unicode code points between UTF-8, UTF-16, and UTF-32."

How it works

This project contains hardware logic to convert between the UTF-8, UTF-16, and UTF-32 encodings for Unicode text.

It will detect and raise an error signal on overlong encodings, out of range code point values, and invalid byte sequences.

(You can optionally disable range checking if you wish to use the original UTF-8 spec that supports values up to 0x7FFFFFFF.)

Basic operation

- In the initial state, all dedicated inputs should be set HIGH.
- At any time, set /RESET (rst_n) LOW and pulse CLK to reset all inputs and outputs to initial state.
- At any time, set /ROUT (input 0) LOW and pulse CLK to seek to the beginning of the output.
- You can set ERRS or /PROPS (input 1) HIGH to get an error status on the dedicated outputs.
- You can set ERRS or /PROPS (input 1) LOW to get character properties on the dedicated outputs.
- You can set CHK (input 2) HIGH to raise an error signal when the code point value is out of range (≥0x110000).
- You can set CHK (input 2) LOW to ignore out of range code point values and encode/decode values up to 0x7FFFFFFF.
- You can set CBE (input 3) HIGH to specify big endian order for UTF-32 and UTF-16 input and output.
- You can set CBE (input 3) LOW to specify little endian order for UTF-32 and UTF-16 input and output.

Inputting UTF-32

1. Set READ or /WRITE (input 4) LOW.
2. Set /CIO (input 5, character I/O) LOW.
3. Set bidirectional I/O to the first byte of the UTF-32 word and pulse CLK.

4. Set bidirectional I/O to the second byte of the UTF-32 word and pulse CLK.
5. Set bidirectional I/O to the third byte of the UTF-32 word and pulse CLK.
6. Set bidirectional I/O to the fourth byte of the UTF-32 word and pulse CLK.
7. Set /CIO (input 5, character I/O) HIGH.
8. Set READ or /WRITE (input 4) HIGH.
9. If READY (output 0) is HIGH and ERROR (output 5) is LOW, the input and output are both valid.
10. If READY (output 0) is LOW or ERROR (output 5) is HIGH, the input was out of range (≥0x110000 or, if CHK is LOW, ≥0x80000000).

Inputting UTF-16

1. Set ERRS or /PROPS (input 1) LOW.
2. Set READ or /WRITE (input 4) LOW.
3. Set /UIO (input 6, UTF-16 I/O) LOW.
4. Set bidirectional I/O to the first byte of the first UTF-16 word and pulse CLK.
5. Set bidirectional I/O to the second byte of the first UTF-16 word and pulse CLK.
6. If HIGHCHAR (output 3) is LOW, skip to step 9.
7. Set bidirectional I/O to the first byte of the second UTF-16 word and pulse CLK.
8. Set bidirectional I/O to the second byte of the second UTF-16 word and pulse CLK.
9. Set /UIO (input 6, UTF-16 I/O) HIGH.
10. Set READ or /WRITE (input 4) HIGH.
11. Set ERRS or /PROPS (input 1) HIGH.
12. If READY (output 0) is HIGH and ERROR (output 5) is LOW, the input and output are both valid.
13. If RETRY (output 1) is HIGH, the first word was a high surrogate but the second word was not a low surrogate. The output will be the high surrogate only; the last word will need to be processed again.

Inputting UTF-8

1. Set READ or /WRITE (input 4) LOW.
2. Set /BIO (input 7, byte I/O) LOW.
3. Set bidirectional I/O to the current byte of the UTF-8 sequence and pulse CLK.
4. Repeat step 3 until READY (output 0) or ERROR (output 5) is HIGH.
5. If READY (output 0) is HIGH and ERROR (output 5) is LOW, the input and output are both valid.

6. If RETRY (output 1) is HIGH, the UTF-8 sequence was truncated (not enough continuation bytes). The output will be the truncated sequence only; the last byte will need to be processed again.
7. If INVALID (output 2) is HIGH, the UTF-8 sequence was a single continuation byte or invalid byte (0xFE or 0xFF).
8. If OVERLONG (output 3) is HIGH, the UTF-8 sequence was an overlong encoding.
9. If NONUNI (output 4) is HIGH, the UTF-8 sequence was out of range (≥0x110000).

Outputting UTF-32

1. Set READ or /WRITE (input 4) HIGH.
2. Set /CIO (input 5, character I/O) LOW.
3. Pulse CLK and read the first byte of the UTF-32 word from the bidirectional I/O.
4. Pulse CLK and read the second byte of the UTF-32 word from the bidirectional I/O.
5. Pulse CLK and read the third byte of the UTF-32 word from the bidirectional I/O.
6. Pulse CLK and read the fourth byte of the UTF-32 word from the bidirectional I/O.
7. Set /CIO (input 5, character I/O) HIGH.
8. If the UTF-32 word is within range, the input and output are both valid.
9. If the UTF-32 word is not within range, then the input was either incomplete or invalid.

Outputting UTF-16

1. Set READ or /WRITE (input 4) HIGH.
2. If UEOF (output 6) is HIGH, then the input was either incomplete or invalid.
3. Set /UIO (input 6, UTF-16 I/O) LOW.
4. Pulse CLK and read the next byte of the UTF-16 sequence from the bidirectional I/O.
5. Repeat step 4 until UEOF (output 6) is HIGH.
6. Set /UIO (input 6, UTF-16 I/O) HIGH.

Outputting UTF-8

1. Set READ or /WRITE (input 4) HIGH.
2. If BEOF (output 7) is HIGH, then the input was either incomplete or invalid.
3. Set /BIO (input 7, byte I/O) LOW.

4. Pulse CLK and read the next byte of the UTF-8 sequence from the bidirectional I/O.
5. Repeat step 4 until BEOF (output 7) is HIGH.
6. Set /BIO (input 7, byte I/O) HIGH.

Error status

When ERRS or /PROPS (input 1) is HIGH, the dedicated outputs will be:

#	Name	Meaning
0	READY	The input and output are complete sequences.
1	RETRY	The previous input was invalid or the start of another sequence and was ignored. Process the output, reset, and try the previous input again.
2	INVALID	The input and output are invalid.
3	OVERLONG	The UTF-8 input was an overlong sequence.
4	NONUNI	The code point value is out of range (≥0x110000). (This is set independently of the CHK input; the CHK input only changes whether this counts as an error.)
5	ERROR	Equivalent to (RETRY or INVALID or OVERLONG or (NONUNI and CHK)).

If all of these outputs are LOW, the accumulated input is incomplete and more input is required (underflow).

Character properties

When ERRS or /PROPS (input 1) is LOW, the dedicated outputs will be:

#	Name	Meaning
0	NORMAL	The code point value is valid and not a C0 or C1 control character, surrogate, private use character, or noncharacter.
1	CONTROL	The code point value is valid and a C0 or C1 control character (0x00-0x1F or 0x7F-0x9F).
2	SURROGATE	The code point value is valid and a UTF-16 surrogate (0xD800-0xDFFF).
3	HIGHCHAR	The code point value is valid and either a high surrogate (0xD800-0xDBFF) or a non-BMP character (≥0x10000).

4	PRIVATE	The code point value is valid and either a private use character (0xE000-0xF8FF, ≥0xF0000) or the high surrogate of a private use character (0xDB80-0xDBFF).
5	NONCHAR	The code point value is valid and a noncharacter (0xFDD0-0xFDEF or the last two code points of any plane).

If all of these outputs are LOW, there is no valid code point in the output.

How to test

The `test.py` file covers a comprehensive set of test cases which are listed in a separate file to avoid bloating the TT09 manual.

External hardware

Any device that needs to process Unicode text.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	/ROUT	READY; NORMAL	I/O LSB
1	ERRS, /PROPS	RETRY; CONTROL	I/O
2	CHK	INVALID; SURROGATE	I/O
3	CBE, /CLE	OVERLONG; HIGHCHAR	I/O
4	READ, /WRITE	NONUNI; PRIVATE	I/O
5	/CIO	ERROR; NONCHAR	I/O
6	/UIO	UEOF	I/O
7	/BIO	BEOF	I/O MSB

Asicle v2

by **htfab**

0227 | 25.175 MHz | HDL Project

github.com/htfab/asicle2

"Wordle clone in raw silicon"

How it works

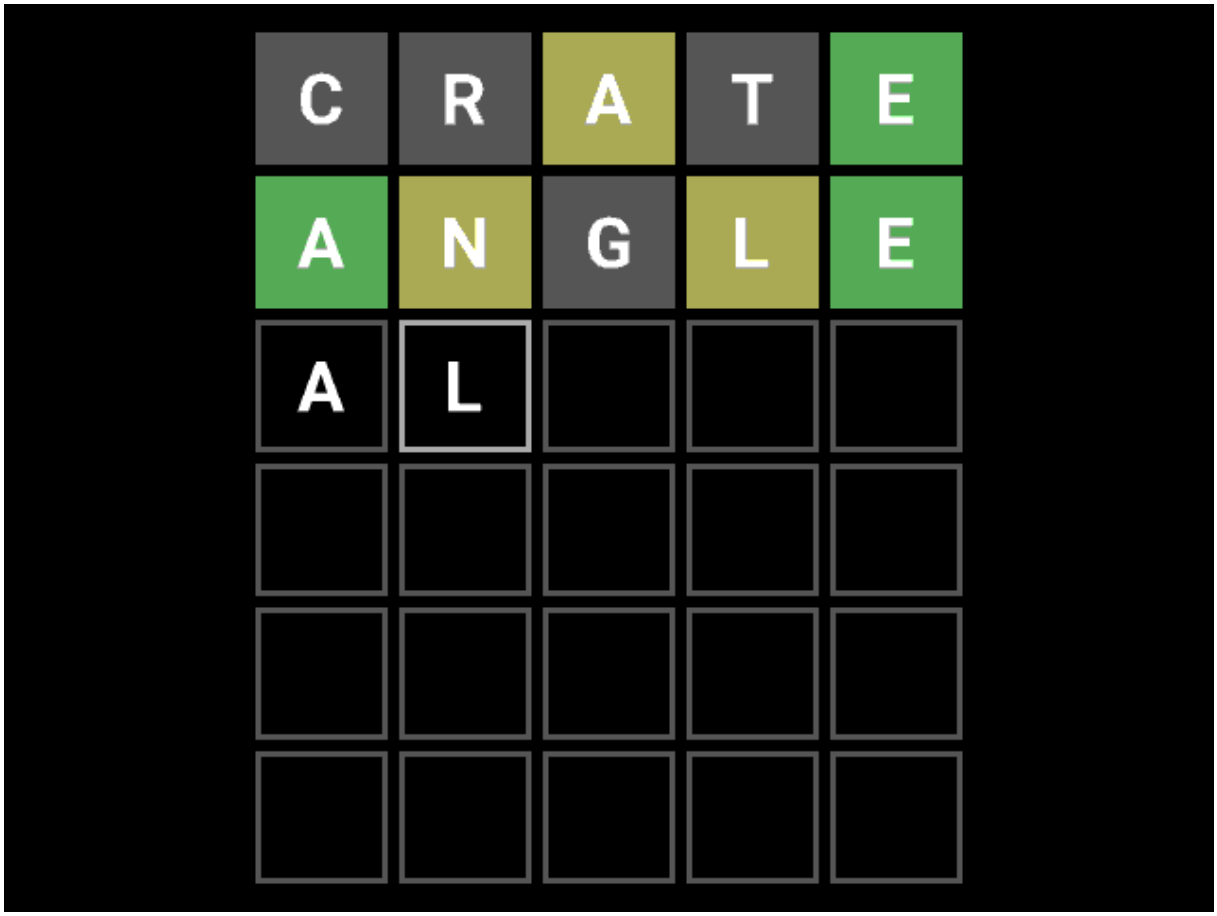


Figure 227.1: VGA screenshot with questions CRATE and ANGLE

Asicle is a Wordle clone implemented directly in hardware. A first version of it was taped out on the Google-Skywater MPW6 shuttle. This second version is a rewrite for Tiny Tapeout.

The 25-fold decrease in area mostly comes from moving the word list and font bitmaps to external flash on the QSPI Pmod, with some architectural changes to compensate for slower memory access. The design was also adapted to the Tiny Tapeout ecosystem by using the Gamepad Pmod for input and the Tiny VGA Pmod for output.

How to test

- Connect the Pmods:
  - Gamepad to input port (optional, you can also drive the input pins using the commander app or momentary push buttons)
  - QSPI to bidirectional port
  - Tiny VGA to output port
- Flash the data file to the QSPI Pmod using the Tiny Tapeout flasher
- Select the design
- Set the clock to 25.175 MHz and reset the design
- Play the game

If you haven't played Wordle before, the aim is to guess a five-letter English word in six attempts. Each time you get feedback: a green square indicates that the letter is correct, a yellow square indicates that it appears in the hidden word but at a different position, and a grey square means that the letter doesn't appear in the solution at all.

Gamepad controls:

- ↑ ↓: change the letter in the selected position
- ← →: move the selection
- A: make a guess
- START: start a new game (if the current one is finished)
- SELECT+START: start a new game (any time)
- SELECT+X: show the solution*
- SELECT+Y: re-roll the solution*

(* only in debug mode)

Direct input using `ui_in`:

- 0 1: change the letter in the selected position
- 2 3: move the selection
- 4: make a guess
- 5: start a new game
- 6: show the solution
- 7: re-roll the solution

External hardware

- Tiny VGA Pmod
- QSPI Pmod
- Gamepad Pmod (optional)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	up	tinyvga: r1	qspi: cs0 (flash)
1	down	tinyvga: g1	qspi: sd0/mosi
2	left	tinyvga: b1	qspi: sd1/miso
3	right	tinyvga: vsync	qspi: sck
4	guess / gamepad: latch	tinyvga: r0	qspi: sd2
5	new game / gamepad: clock	tinyvga: g0	qspi: sd3
6	peek (debug) / gamepad: data	tinyvga: b0	qspi: cs1 (unused)
7	roll (debug)	tinyvga: hsync	qspi: cs2 (unused)

Ring Oscillators

by **Matt Venn**

229 Analog Project

`github.com/mattvenn/tt08-analog-ring-osc`

"Ring Oscillators using analog output pins"

How it works

Aiming to create 2 ring oscillators at around 600MHz and 300MHz. The output will be quite attenuated due to the pad.

- Ring oscillator 1 is made of 18 inverters and a NAND gate for enable.
- Ring oscillator 2 is made of 36 inverters and a NAND gate for enable.

To get a good output current, a 2 stage inverter is used with large drive transistors.

- Ring oscillator 1
- Ring oscillator 2
- Driver

The output waveform of the 600MHz is expected to be as shown in the cyan trace (out_parax). The ring_out_parax and pre_drive_parax are internal signals. See the xschem test bench for more details.

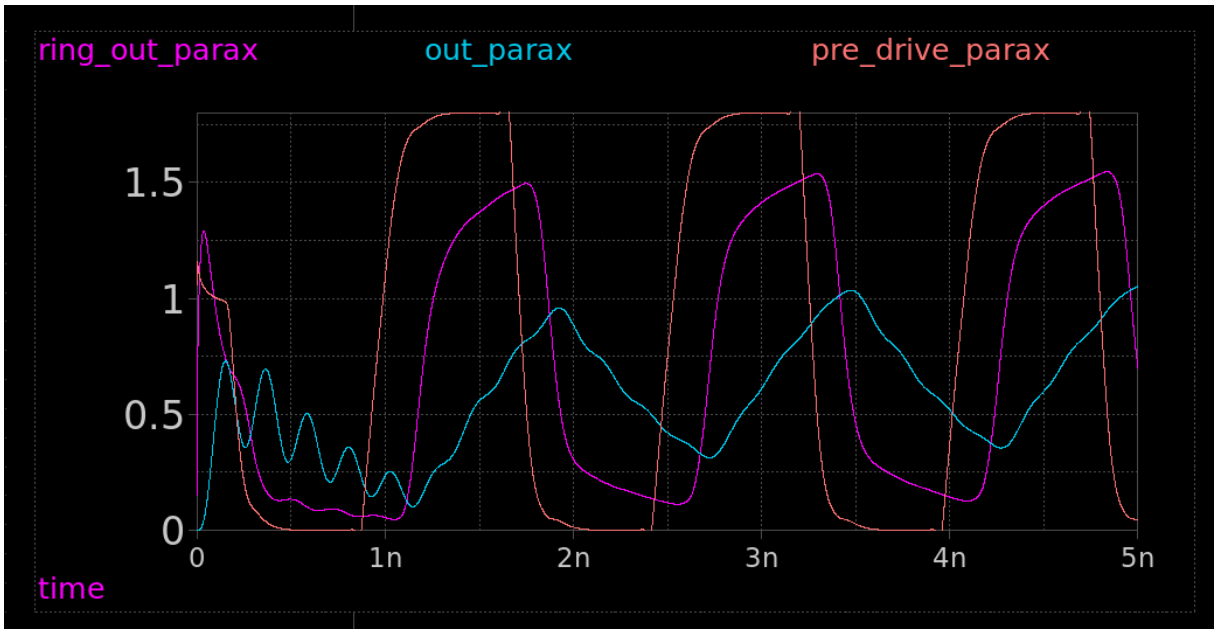


Figure 229.1: output waveform

How to test

- Enable 600 MHz oscillator 1 by setting user input pin 0 high and measure the signal at analog output 0.
- Enable 300 MHz oscillator 2 by setting user input pin 1 high and measure the signal at analog output 1.

The 300 MHz ring oscillator had a problem with a missing contact in TT08, and is not expected to work. It has been fixed for TT09.

External hardware

Oscilloscope.

Tested

Oscillator 1 is working, but 2 is broken due to a missing via (fixed in updated version). This is oscillator 1. The amplitude is lower because the pull down resistor has been left in place on the demoboard.

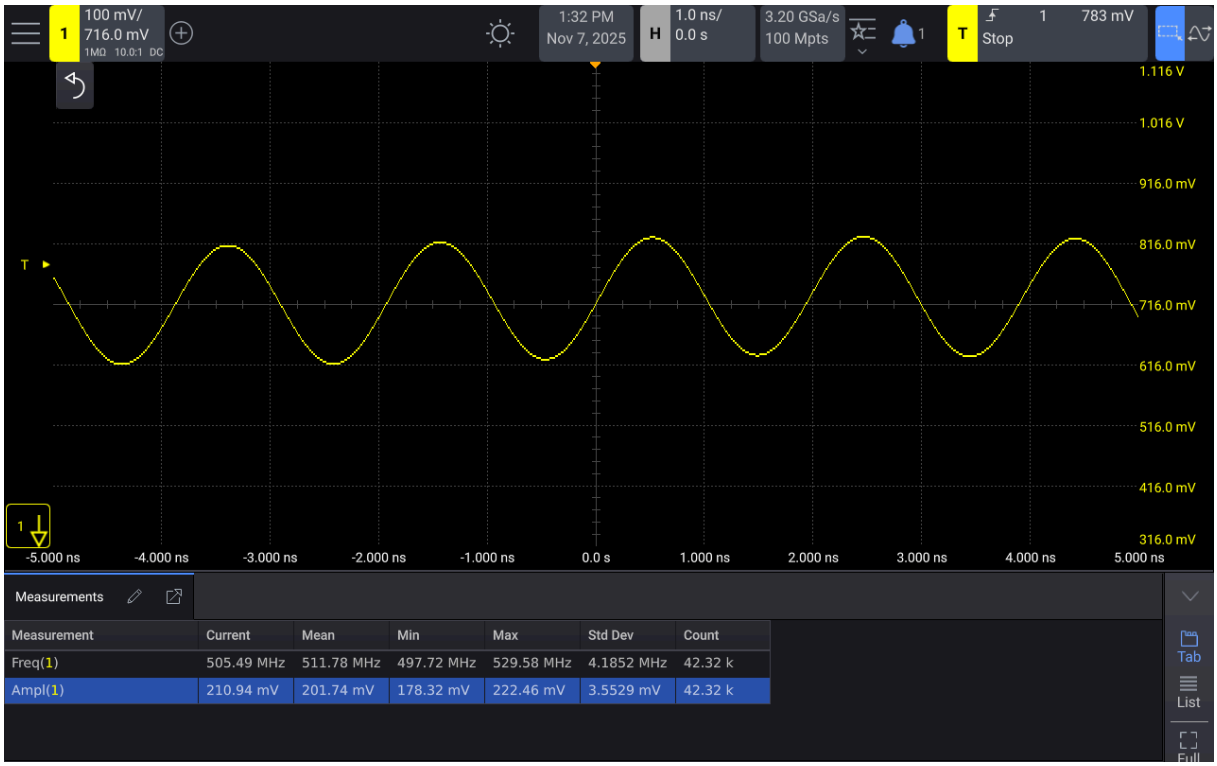


Figure 229.2: test

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Enable ring 1	ring_oscillator1	—
1	Enable ring 2	ring_oscillator2	—

#	Input	Output	Bidirectional
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	5	ring0
1	0	ring1

Nauta OTA with digital trimming

by **Astria Nur Irfansyah**

0231 Analog Project

github.com/nurirfansyah/ttsky25a-irfansyah-nauta

"Simple Nauta OTA with 2-bit digital trimming"

How it works

Nauta OTA with digital trim is based on the OTA circuit proposed by Bram Nauta. The digital trim is a subset of the digitally-tunable Nauta OTA described in [3]. One tuning implementation is described in [4].

[1] Nauta, B. (1993). Analog CMOS filters for very high frequencies. Springer US. https://doi.org/10.1007/978-1-4615-3580-5 [2] Nauta, B. (1992). A CMOS transconductance-C filter technique for very high frequencies. Solid-State Circuits, IEEE Journal Of, 27(2), 142–153. https://doi.org/10.1109/4.127337 [3] Nicholson, A. P., Iberzanov, A., Jenkins, J., Hamilton, T. J., & Lehmann, T. (2016). A Statistical Design Approach for a Digitally Programmable Mismatch-Tolerant High-Speed Nauta Structure Differential OTA in 65-nm CMOS. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, PP(99), 1–12. https://doi.org/10.1109/TVLSI.2016.2526048 [4] Irfansyah, A. N., Nicholson, A. P., Iberzanov, A., Jenkins, J., Lehmann, T., & Hamilton, T. J. (2016). Automatic tuning of digitally-controllable positive-feedback OTAs in continuous-time sigma–delta modulators. Analog Integrated Circuits and Signal Processing, 89(2), 469–483. https://doi.org/10.1007/s10470-016-0820-3

How to test

Provide differential input ac signals, and observe output. Digital trimming is done through digital pins.

External hardware

External hardware not specifically required, other than normal test equipment.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	sw_p_A1	—	—
1	sw_p_B1	—	—

#	Input	Output	Bidirectional
2	sw_p_A05	—	—
3	sw_p_B05	—	—
4	sw_n_A1	—	—
5	sw_n_B1	—	—
6	sw_n_A05	—	—
7	sw_n_B05	—	—

Analog Pins

ua#	analog#	Description
0	5	in_p
1	0	in_n
2	4	out_p
3	1	out_n

Analog VGA CSDAC experiments (TTSKY25b)

by **algofoogle (Anton Maurovic)**

0233 | 25.175 MHz | Analog Project

github.com/algofoogle/ttsky25b-analog-vga-dacs

"Current switching matrix DAC implementations for analog VGA displays"

Overview

This is a revision of the same design as was submitted to TTSKY25a. Both projects continue from my previous VGA DAC experiments (tt06-grab-bag, tt08-vga-fun, and ttihp0p3-antonalog), to produce 24-bit analog outputs (red, green, and blue channels of 8 bits each) and drive a VGA monitor.

The TTSKY25a version is expected to have some bad output glitches near the points where it would switch between rows. This was mostly found (in simulation, at least) to be caused by capacitive coupling between the column select lines and the Vbias straps – thus briefly pulling the Vbias line as multiple column select lines all switch at the same time (which would have an increasingly severe effect when more icells are turned on). This TTSKY25b version attempts to improve on that with much better shielding of the Vbias line, as well as a lower-impedance Vbias generator circuit.

How it works

Like the TTSKY25a version, this project implements three current-switching DACs, intended for generating analog VGA display signals, using a 16x16 matrix of unit current sources, converting 8-bit digital inputs to inverted analog current sink "outputs" in the range of 3.3V down to 1.3V (depending on external pull-up resistance).

First, a "unit" current cell ("icell") layout is created, which includes a 3.3V-capable current sink NFET with a tiny bit of digital logic that allows the cell to be activated by row/column intersection, or by the whole row being turned on as one. Above that cell is higher-level routing that will make it cooperate and mesh with its neighbours:

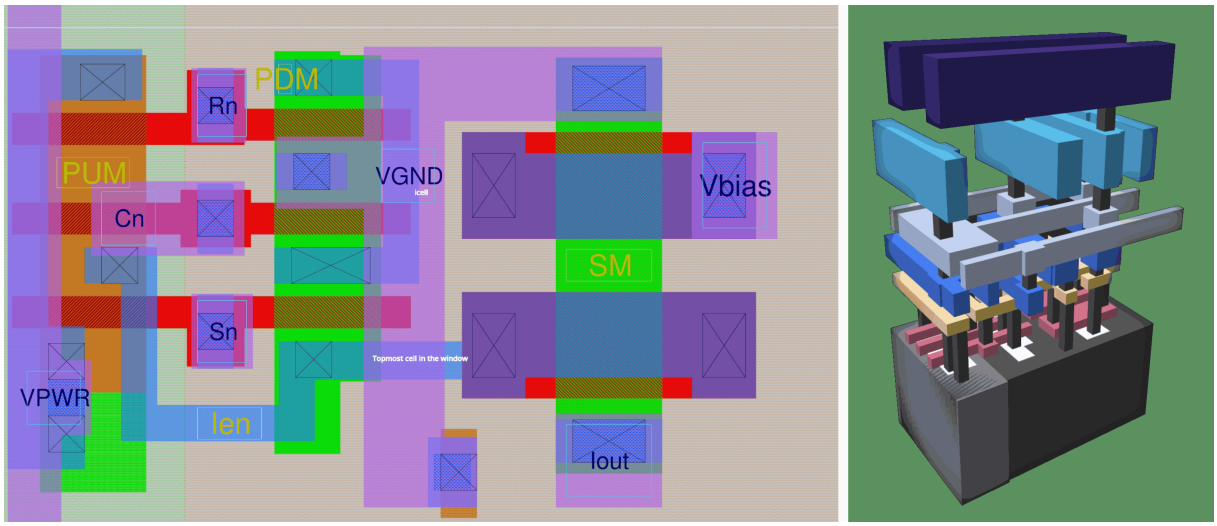


Figure 233.1: Icell layout and 3D rendering

These are arranged into a row of 16 cells, plus a dummy cell either side to help improve matching:

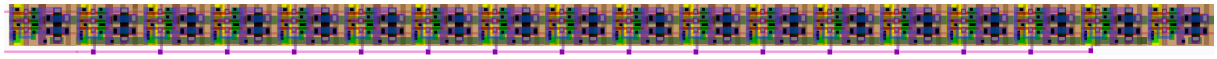


Figure 233.2: Row of Icell instances

16 instances of the row are stacked, and combined with two 4-to-15 thermometer encoder blocks (i.e. 4 MSBs of the input code becomes the "row select" while the 4 LSBs become the "column select") plus the "Vbias" programmable reference voltage circuit for controlling the amount of current each "icell" will sink:

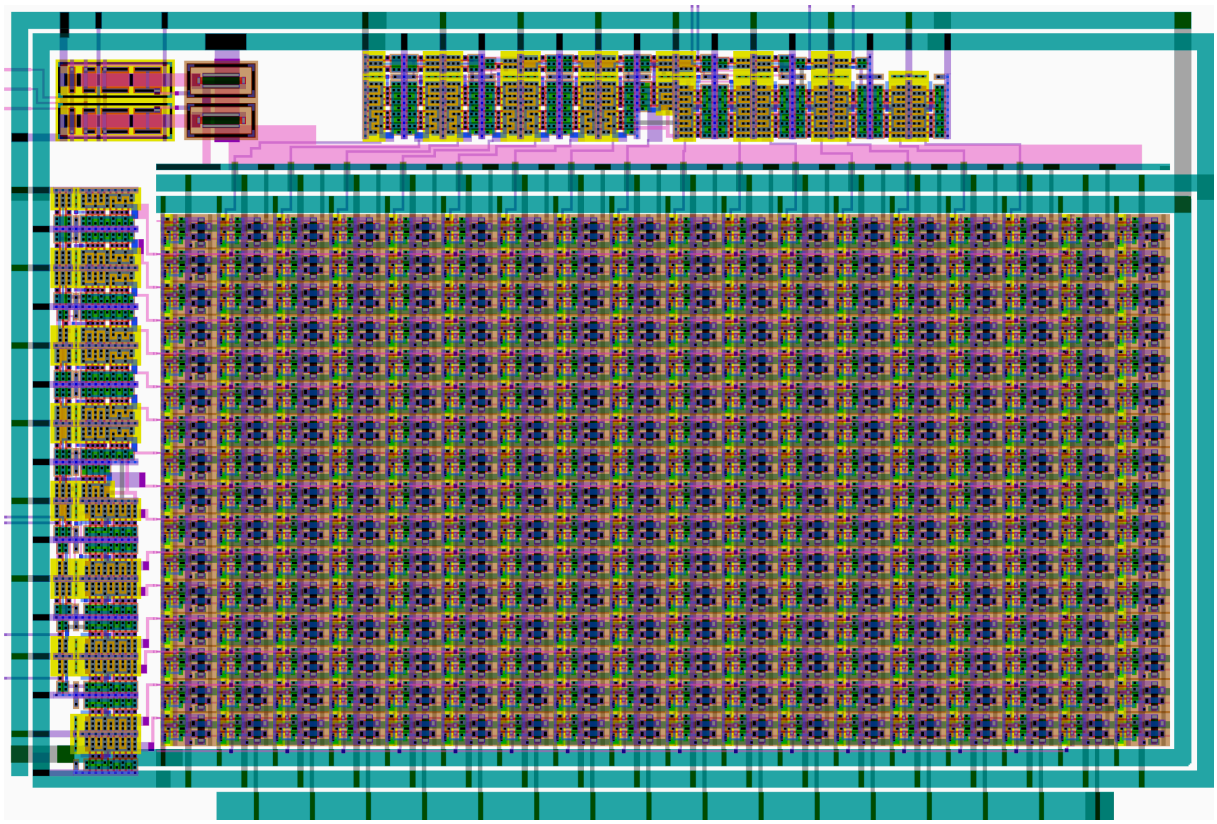


Figure 233.3: Main DAC matrix layout

Three instances of this are combined with a digital controller block that is hardened with OpenLane 2, and then manually placed as a macro using the Magic VLSI layout tool:

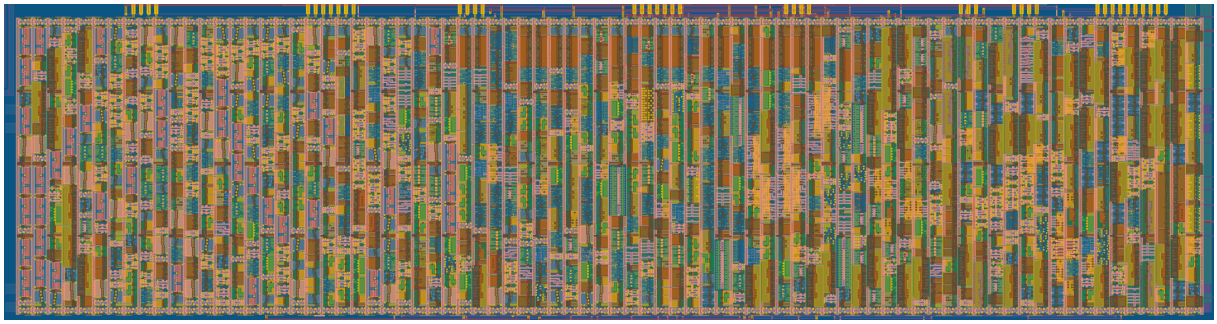


Figure 233.4: Digital controller block

The controller interfaces with the outside world using the standard Tiny Tapeout pinout, and besides generating the VGA control signals, it also generates various test patterns and pretty displays which it sends to the 3 DACs.

TBC!

How to test

TBC!

The TinyVGA PMOD can be used with this design, for basic digital RGB222 (6-bit colour) output.

To test the analog outputs, though: Use a pull-up resistor on each of the red, green, and blue analog output pins – pull up to 3.3V using (say) a 500-ohm or 1k resistor, then shift this to 0-0.7V using an inverting opamp configuration. Alternatively, use a TIA (Transimpedance Amplifier) externally.

Set Vbias internally using digital inputs (through 8 possible levels).

For more information on the digital control block's test modes, you can mostly use the TT08 version's documentation as your reference: https://tinytapeout.com/chips/tt08/tt_um_algofoogle_tt08_vga_fun

External hardware

TBC!

- VGA display, optionally with the TinyVGA PMOD.
- 3 opamp circuits for converting each colour channel's current sink to a 0-0.7V range.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	mode[0] / dac_in[0]	r7	vblank_out
1	mode[1] / dac_in[1]	g7	hblank_out
2	mode[2] / dac_in[2]	b7	bias_in[0]
3	mode[3] / dac_in[3]	vsync	bias_in[1]
4	mode[4] / dac_in[4]	r6	bias_in[2]
5	mode[5] / dac_in[5]	g6	—
6	mode[6] / dac_in[6]	b6	—
7	mode[7] / dac_in[7]	hsync	—

Analog Pins

ua#	analog#	Description
0	5	ra
1	4	ga
2	3	ba
3	2	vbias_red

UNAL's RISCV

by **Carlos Camargo Bareño**

0238 HDL Project

github.com/cicamargoba/femto_UN

"Bruno Levi's femtoRV imnplementation"

How it works

This is a femtoRV implememtation using SPI flasg and spiRAM ## How to test

You must program an external flash memory wioth desired application, all communication and debug messages are displayed in UART

External hardware

one LED one USB-SERIAL dongle

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	spi_miso	spi_mosi	—
1	spi_miso_ram	spi_mosi_ram	—
2	RXD	spi_cs_n	—
3	—	spi_cs_n_ram	—
4	—	spi_clk_ram	—
5	—	spi_clk	—
6	—	LEDS	—
7	—	TXD	—

tnt's variant of SKY130 mini-MOSbius

by **Sylvain Munaut**

0239 Analog Project

`github.com/smunaut/ttsky-mini-mosbius`

"Alternative layout for Andrew Kang's mini-MOSbius for SKY130 Tiny-Tapeout"

How it works

The original MOSbius project by Peter Kinget et al. is a custom chip with basic MOS devices and building blocks interconnected by a reconfigurable array of analog switches to be able to build circuits and experiment with analog circuit design.

Matthew Venn commissioned an adaptation of the concept to fit on Tiny Tapeout Skywater shuttles and a proposal by Andrew Kang was accepted. See the attached document.

Andrew completed the schematic design along with simulation end of August 2025 but before he completed layout, I decided to use this as a base to build my own layout variant.

The design of the analog part is pretty much unchanged, only minor adaptations guided by layout were made. The digital control part was adapted quite a bit.

How to test

A configuration bitstream needs to be loaded serially to control all the analog switches on-board. The software suite to generate this is yet to be written.

External hardware

Depends on what circuit you want to build and how you want to test it ...

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data_in	data_out	—
1	enable	—	—
2	—	—	—
3	—	—	—

#	Input	Output	Bidirectional
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	5	Reference Bias
1	0	Bus 1A
2	4	Bus 2A
3	1	Bus 3A
4	3	Bus 4A
5	2	Bus 5A

test-project.

by **Lukas Rabeder**

0256 | 10 kHz | Wokwi Project

`github.com/LukasRabeder/watchdog`
`wokwi.com/projects/442977704392041473`

"uses flipflop to change segments on clk"

How it works

This is a testproject.

How to test

Test.

External hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in_0	out_0	—
1	in_1	out_1	—
2	in_2	out_2	—
3	in_3	out_3	—
4	in_4	out_4	—
5	in_5	out_5	—
6	in_6	out_6	—
7	in_7	out_7	—

TTSKY25A Register File Test

by **Sylvain Munaut**

0257 HDL Project

`github.com/smunaut/ttsky25a-rf-test`

"Test structures for a future register file macro"

How it works

It's basically a SRAM with 1 write port and 2 read ports. See design data for information

How to test

No test procedure has been written yet and it's not meant to be tested by the "demo board" but instead on a dedicated test platform.

External hardware

To do any meaningful timing testing you'll need some FPGA hardware to drive the various control signal in sequence with precise timings.

The exact testing platform is still TBD.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	tbd	tbd	tbd
1	tbd	tbd	tbd
2	tbd	tbd	tbd
3	tbd	tbd	tbd
4	tbd	tbd	tbd
5	tbd	tbd	tbd
6	tbd	tbd	tbd
7	tbd	tbd	tbd

Can you count binary?

by **Tobias Pfaffeneder**

0258 | 1 kHz | HDL Project

`github.com/TobiasPfaffeneder/tt-sky25b-can-you-count-binary-game`

> *"A logic game designed for the Tiny Tapeout with a simple goal: Convert decimal numbers displayed on the seven-segment display into their binary equivalents using the onboard DIP switches before the timer ends."*

Module: `tt_um_dip_switch_game_TobiasPfaffeneder`

Can You Count Binary is a logic game designed for the Tiny Tapeout platform.
It runs entirely on the Tiny Tapeout PCB — no external hardware required!
Your goal is simple: convert decimal numbers displayed on the seven-segment display into their binary equivalents using the onboard DIP switches.
But hurry up - your time is limited, and the game gets faster as you play!

▸ How to Play

1. Power up your Tiny Tapeout PCB with this module loaded.
2. To start the game, bring all DIP switches except the 8th one in the **OFF** position.
3. A random 8-bit number appears on the seven-segment display.
   - The number is always shown as three digits (e.g. `123`, `045`, `007`, ...).
4. Convert the decimal number to binary and enter it using the DIP switches:
   - **Switch 8** = Least Significant Bit (2^0)
   - **Switch 1** = Most Significant Bit (2^7)
5. If your input is correct, a new random number will be displayed.
6. Be quick! You start with **30 seconds per number**, and the timer gets shorter as the game progresses.
   - If the timer runs out before you enter the correct value, the game ends.
7. When the game is over, your **final score** will be shown on the display.

⚙️ Top-Level I/O

Signal	Dir	W	Description

`ui_in[7:0]`	in	8	**DIP switches** (player input)
`uo_out[7:0]`	out	8	**Seven-segment display** (parallel 8-bit)
`uio_in[7:0]`	in	8	Unused
`clk`	in	1	System clock (1 kHz)
`rst_n`	in	1	Asynchronous reset (active-low)
`ena`	in	1	Always 1 on Tiny Tapeout

👉 Try it know on Wokwi

https://wokwi.com/projects/446871385453862913

🔌 External hardware

No external hardware is required. The Dip switches and the seven-segment display on the Tiny Tapeout PCB are used to play the game.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	DIP-Switch 0	7-Segment-Display A	unused
1	DIP-Switch 1	7-Segment-Display B	unused
2	DIP-Switch 2	7-Segment-Display C	unused
3	DIP-Switch 3	7-Segment-Display D	unused
4	DIP-Switch 4	7-Segment-Display E	unused
5	DIP-Switch 5	7-Segment-Display F	unused
6	DIP-Switch 6	7-Segment-Display G	unused
7	DIP-Switch 7	7-Segment-Display DP	unused

Relaxation oscillator

by **Matt Venn**

0259 Analog Project

`github.com/mattvenn/ttsky25b-analog-relax-oscillator`

"A relaxation oscillator"

How it works

In electronics a relaxation oscillator is a nonlinear electronic oscillator circuit that produces a nonsinusoidal repetitive output signal, such as a triangle wave or square wave.

The R&C have been chosen to make a 2MHz signal. An inverter after the oscillator makes a full swing square wave.

How to test

Measure the oscillator out on pin 0 (tbc, might cause issues due to the analog mux parasitics). Measure the square wave out on digital output pin 0.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	inverted output of oscillator	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	10	analog oscillator output

Tiny_Tapeout_Scoreboard

by **Lukasseirl**

0260 | 1 kHz | HDL Project

`github.com/Lukasseirl/Tiny_Tapeout_Scoreboard`

"Scoreboard for two players that can be controlled with two pushbuttons."

How it works

The Tiny Tapeout Scoreboard chip processes the input of two pushbuttons and controls two 7 segment displays. The purpose is to count and show a game score of two players. With the two pushbuttons you are able to set the points - button 1 is for player 1 and button 2 for player 2. A short press adds one point to the count, a long press (at least 1.5 seconds) decreases the score by 1. The two digit score is then presented via two 7-segment displays wich alternate between the score of player 1 and player 2. Therefore the scoreboard blinks 2 times with the text 'P1' or 'P2' and after the blinking the respective score of each player is shown for 2 seconds.

Furthermore, there is a pushbutton-processor that, not only recognises a long press of a push buthon, but has an debounce logik so that a single pushbutton press increases the score just by one - even if the button is bouncing.

How to test

The design can be testet with the given testbenches in the test folder. Each moudle has its own testbenchfile.

External hardware

- 2x external push-button with a pull up resistor and a ground connection.
- another 7 segment display or a 2 digit 7 segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	P1	Seg_A1	Seg_A2
1	P2	Seg_B1	Seg_B2
2	—	Seg_C1	Seg_C2
3	—	Seg_D1	Seg_D2

#	Input	Output	Bidirectional
4	—	Seg_E1	Seg_E2
5	—	Seg_F1	Seg_F2
6	—	Seg_G1	Seg_G2
7	—	Seg_H1	Seg_H2

Relax_Oscillator

by **UABC**

0261 Analog Project

`github.com/FloresCesarMX/relax_oscillator`

"relaxation_oscilator"

How it works

By changing the values of the capacitors it changes the frecuency

How to test

Using a Osciloscope.

External hardware

It doesn't use any external hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	ENABLE1	Q	—
1	ENABLE2	Qn	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	9	C1
1	8	C2

7SegDec

by **Tobias Graf**

0262 Wokwi Project

`github.com/Saibot2809/TinyTapout`
`wokwi.com/projects/442977493563151361`

"It is a 7 Segment Decoder"

How it works

It is a 7 Segent Decoder.

How to test

You have to switch S0 to S6. Each switch schoudl turn on another Segment.

External hardware

7 Segment DIsplay

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	CLK	OUT7	—

Oscillating Bones

by **Uri Shaked**

0263 Analog Project

`github.com/urish/tt10-oscillating-bones`

"A stylish ring oscillator built from SkullFET transistors"

How it works

A simple yet stylish ring oscillator that uses a chain of 21 SkullFET inverters to generate a square wave output. Based on simulation, the oscillator should have a frequency of around 169.2 MHz.

Pin	Expected frequency
osc_out	169.2 MHz
osc_div_2	84.6 MHz
osc_div_4	42.3 MHz
osc_div_8	21.1 MHz
osc_out_3v3	169.2 MHz

How to test

Connect an oscilloscope to one of `osc_out_3v3` or `osc_div_8` (uo_out pin 3) pin and enjoy the show. The remaining output pins, `osc_out`, `osc_div_2`, and `osc_div_4`, are likely to be changing too fast for GPIO pins to handle (the GPIO pins are limited to around 33 MHz), so you will probably not be able to observe them directly.

Simulation results

The following graph shows the output of the oscillator and the divided outputs. It was generated by running `make -C sim` and patiently waiting for the simulation to finish:

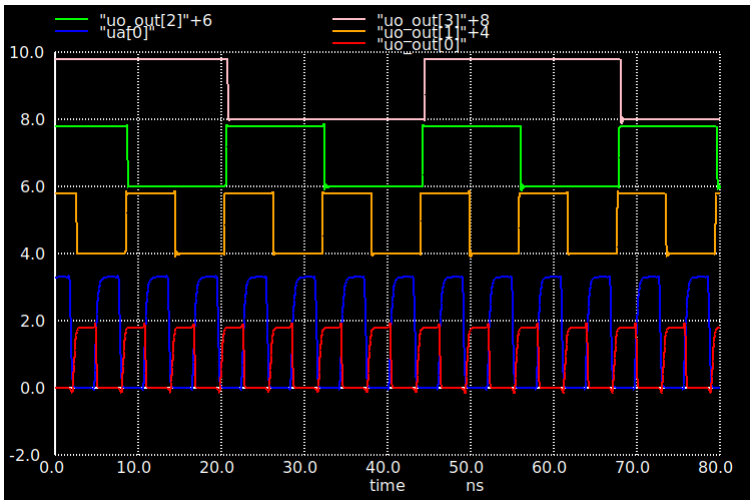


Figure 263.1: Simulation results

The outputs are shifted by 2 volts to make them easier to see in the graph. “uo_out[0]” is the main output of the oscillator, “ua[0]” is the 3v3 analog output, and “uo_out[1]”/“uo_out[2]”/“uo_out[3]” are the divided outputs.

Please note that the simulation results do not account for all parasitics, only the primary ones. Consequently, the actual frequency of the oscillator is likely to be lower than the simulated value.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	osc_out	—
1	—	osc_div_2	—
2	—	osc_div_4	—
3	—	osc_div_8	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	8	osc_out_3v3

7segCounter

by **MrF83**

0264 Wokwi Project

`github.com/MrF83/TinyTapeoutLinz`
`wokwi.com/projects/442976058838162433`

"Counter to 7 segment display."

How it works

TODO

How to test

TEST

External hardware

LOL

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input 1	out 1	—
1	input 2	out 2	—
2	input 3	out 3	—
3	input 4	out 4	—
4	—	out 5	—
5	—	out 6	—
6	—	out 7	—
7	—	—	—

Detail - Fibonacci design (MPW-2) – Designed by Konrad Rzeszutek Wilk. Illustrated by Máximo Balestrini.

ttsky25_se_opamps

by **Kithmin Wickremasinghe**

0265 Analog Project

github.com/SkillSurf/ttsky25_se_opamps

"TTSKY25b Low-power Single-ended Op-Amp tapeout from AICD Content"

How it works

In this project, we design and tape out a low power single-ended two-stage operational amplifier, using the SKY130 PDK, intended to function as a non-inverting unity-gain buffer for low-frequency analog signals (1–10 kHz).

The chip has the following connections:

- **Digital inputs (3 pins):** Select which design to enable
- **Analog inputs (2 pins):** Apply differential input signal
- **Analog outputs (1 pins):** Get the single ended output signal

Three different submission have been included in the chip:

- **ui_in[0]:** Lochana
- **ui_in[1]:** Rajinthan
- **ui_in[2]:** Nilakna and Pravindu

How to test

We plan to use the **AD03 board** to test the op-amp as a non-inverting unity-gain buffer. During testing, we will measure:

- Standard linearity
- Gain bandwidth product
- Offset voltage and offset current
- Overall power consumption

External hardware

The op-amp needs:

- An **external current source (Iref)** for biasing.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	en_1	—	—
1	en_2	—	—
2	en_3	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	10	Iref
1	7	Vm
2	9	Vp
3	8	Vout

Shift_Register

by **Simon Haeusler**

0266 Wokwi Project

`github.com/SmashHandTV/TinyTapeoutToShiftRegister`

`wokwi.com/projects/442977452121292801`

"Shift Register which runs along the outer ring of the seven segment."

How it works

setting In to high adds a bit to the shift register. by setting the INV input to high the display is inverted.

How to test

edit

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN_SHR	A	—
1	INV_RING	B	—
2	EN_SEG_MID	C	—
3	—	D	—
4	—	E	—
5	—	F	—
6	—	G	—
7	—	DP	—

2 Bit Counter

by **Nico Jankowitsch**

0268 Wokwi Project

`github.com/Nicojpg67/TinyTapeout2BitCounter`
`wokwi.com/projects/442977436543634433`

"a simple 2 bit counter"

How it works

Its a 2 Bit Cunter that shows the current number on a 7 segment

How to test

just flip on the clock

External hardware

7 segment

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	OUT0	—
1	—	OUT1	—
2	—	OUT2	—
3	—	OUT3	—
4	—	OUT4	—
5	—	OUT5	—
6	—	OUT6	—
7	—	OUT7	—

Tiny Flash ADC

by **Blake Mott**

0269 Analog Project

`github.com/vincentfusco/ttsky25b_tinyflash`

"3bit flash ADC with binary and thermometer output"

How it works

The 3bit flash ADC consists of (2^n)-1 comparators, which are supplied with the input voltage on the non-inverting input, and a scalled reference voltage from a resistor chain. This circuit is designed to operate from 0V-1V, using a vref = 1V.

How to test

For DC analysis, sweep the vin voltage from 0V-1V and observe the digital output counting up. For AC analysis, provide a sinusoidal signal (<100kHz for better performance) and overlay the output with the input signal (you can graph the difference between the two to quantify the performance).

External hardware

DC Power Supply, AC Function Generator, Oscilloscope, and possibly a simple microcontroller DIY board to automatically convert from binary output to decimal.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	bin_out_0	therm_out_5
1	—	bin_out_1	therm_out_6
2	—	bin_out_2	—
3	—	therm_out_0	—
4	—	therm_out_1	—
5	—	therm_out_2	—
6	—	therm_out_3	—
7	—	therm_out_4	—

Analog Pins

ua#	analog#	Description
0	7	vin
1	9	vref

Davids first Wokwi design

by **David Weissensteiner**

0270 | 1 Hz | Wokwi Project

`github.com/DavidWeissensteiner/TyniyTapoutDavidsFirstWokwiDesign`
`wokwi.com/projects/442977645972707329`

"8 Bit backward Counter"

How it works

The the Tickrate of the Clock it counts 8Bit backward.

How to test

Conect everything and look if it counts

External hardware

8 LEDs

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	Out1	—
1	—	Out2	—
2	—	Out3	—
3	—	Out4	—
4	—	Out5	—
5	—	Out6	—
6	—	Out7	—
7	—	Out8	—

Onchip - Ring VCO 11 stages x2

by **Jorge Angarita**

0271 Analog Project

github.com/Gior-gio/tt_um_Onchip_VCO

"Two identical 11-stages ring oscillator voltage controled oscillator"

How it works

The circuit is an 11-stage ring oscillator voltage-controlled oscillator (RVCO). The basic oscillating stage consists of current-starved inverters controlled by an external input, VCTRN. Additionally, there is a VRST pulse input that runs or stops the oscillator when it receives a low or high value. One of its phases is buffered and connected as an output.

The submission includes two oscillators. The first three analog pins are associated with the first oscillator in the following order: OUT, VCTRN, and VRST. The other three are associated with the second oscillator in the same order.

How to test

First, activate the oscillator by setting the corresponding VRST to GND. Then, set the VCTRN node to a relatively high analog value. For testing purposes, the test benches were done using Vctrn = 0.9 V. Then, measure the OUT node to obtain a pulse signal.

External hardware

- Voltage Source
- Oscilloscope

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	11	out1
1	6	vctrn1
2	10	vrst1
3	7	out2
4	9	vctrn2
5	8	vrst2

4-bit Johnson Counter

by **Nathan Jones**

0288 | 1 kHz | Wokwi Project

github.com/nathancharlesjones/tiny-tapeout_johnson-counter

wokwi.com/projects/446363782054494209

"Generates 4 offset square waves at 1/4 clock frequency"

How it works

A row of 4 DFFs are connected output to input (Q_n –> D_n+1). The last DFF is connected to the first via inverse Q.

How to test

Power on the IC. All outputs should be toggling at 1/4 of the input clock frequency.

A more in-depth test is to connect all four output to a logic analyzer. Verify that you see the following pattern when power is applied:

Q_1 1-1-1-1-0-0-0-0

Q_2 0-1-1-1-1-0-0-0

Q_3 0-0-1-1-1-1-0-0

Q_4 0-0-0-1-1-1-1-0

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	Q1	—
1	—	Q2	—
2	—	Q3	—
3	—	Q4	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

HAD TT

by **iSD**

0289 | 10 kHz | Wokwi Project

`github.com/iServeData/ttsky-wokwi-template`

`wokwi.com/projects/446363984157070337`

"Logic gates."

How it works

It works

How to test

We just run it.

External hardware

Can be anything. List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	t1 and	—
1	—	t1 and 2	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

test

by **Konkers**

0290 Wokwi Project

`github.com/konkers/tiny-tapeout-workshop`

`wokwi.com/projects/446363880417243137`

"test"

How it works

... Explain how your project works

How to test

... Explain how to use your project

External hardware

... List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	A	S	—
1	B	C	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny Tapeout Binary to Decimal

by **Nima Moravej**

0291 | 10 kHz | Wokwi Project

github.com/Jevarom/binary-decimal

wokwi.com/projects/446363731295014913

"Generate a decoder for a 8-bit decimal number to a 7-segment decimal digit"

How it works

Using simply AND and OR gates are connected to several inputs a, input b, input c, input d, input e

The goal is to get a 8-bit binary value into a 7-segment display decimal

How to test

input a and b	output a and b
0 0	0 1
0 1	0 0
1 0	0 0
1 1	1 0

input c	output c
0	1
1	0

input d and e	output d
0 1	1
1 0	1
0 0	0
1 1	1

External hardware

No external hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	input b	—	—
2	—	output b	—
3	—	—	—
4	input c	output c	—
5	—	—	—
6	input d	output d	—
7	input e	—	—

miloh-tinytapeout-supercon2025

by **r miloh**

0292 Wokwi Project

`github.com/miloh/miloh-tinytapeout-supercon`

`wokwi.com/projects/446907838117673985`

"TBD (after critical steps workshop at Supercon 2025!)"

How it works

The correct DIPSW sequence (coded, flip EOS) should create an 'A.' on a connected 8 segment display

How to test

Connect dipsw pins 1-8 to IO pins 0-7 Connect 8 segment display to the output

External hardware

5v psu one 8 pin dispwithc, one 8 segment led cell,

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	inbut b	output nand	—
2	—	output or	—
3	—	output clock div 16	—
4	—	output clock div 12	—
5	—	—	—
6	—	—	—
7	—	—	—

6502 Chip Selector

by **Brandon Ashworth**

0293 | 1 MHz | HDL Project

`github.com/brandonvfx/tiny_tapeout_class`

"Simple TT Project to select the different chips in Ben Eater's 6502 project."

How it works

Given the clock and high 5 bits of the address bus on Ben Eater's beardboard 6502, select set the correct lines to enable the different devices.

How to test

- ROM_CS - Active Low
- RAM_CS - Active Low
- RAM_OE - Active Low
- PERIPH_CS - Active Low
- VIA_CS - Active High
- URAT_CS - Active High
- SID_CS - Active Low

CLK	A11	A12	A13	A14	A15	ROM_CS	RAM_CS	RAM_OE	PERIPH_CS	VIA_CS	URAT_CS	SID_CS
0	0	0	0	0	0	1	0	0	1	0	0	1
1	0	0	0	0	0	1	1	0	1	0	0	1
0	0	0	0	0	1	0	1	0	1	0	0	1
0	0	0	0	1	1	0	1	1	1	0	0	1
0	0	0	0	1	0	1	0	1	0	0	0	1
0	0	0	1	1	0	1	0	1	0	1	0	1
0	0	1	0	1	0	1	0	1	0	0	1	1
0	1	0	0	1	0	1	0	1	0	0	0	0

External hardware

Ben Eater 6502 breaboard computer or LED bar graph.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	CLK	ROM_CS	—
1	A11	RAM_CS	—
2	A12	RAM_OE	—
3	A13	PERIPH_CS	—
4	A14	VIA_CS	—
5	A15	URAT_CS	—
6	—	SID_CS	—
7	—	—	—

JBs first chip

by **JB Morch**

0294 Wokwi Project

`github.com/jbmorch/tinytapeout`

`wokwi.com/projects/446645853372820481`

"8-bit counter"

How it works

The goal is to create an 8-line counter, similar to a TI CD4022B decade counter with 8 outputs.

How to test

Set enable to high and pulse the clock input with a square wave. NOTE the enable is not working. It will always be enabled.

External hardware

Use a microcontroller (like an Arduino or ESP32 module) to control the clock. Send the outputs to the individual segments of a 7-segment LED display (with the optional 8th segment that is the decimal point).

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Enable (n/a)	Output 0	Carry Out
1	—	Output 1	—
2	—	Output 2	—
3	—	Output 3	—
4	—	Output 4	—
5	—	Output 5	—
6	—	Output 6	—
7	—	Output 7	—

Digit Fidgit

by **Phantom**

0295 | 10 kHz | Wokwi Project

`github.com/phantomguerila/TTO-GDS`

`wokwi.com/projects/447138553926768641`

"Fidget with dip switches to make numbers on the 7-segment display."

How it works

Hidden in the logic are the numbers 0-9! Flip different combinations of dip switches to find the numbers 0-9. When the a correct combination is found, a number will display on the 7-segment display. If no number is shown (or the red dot on the bottom right is lit) then you have not found a valid combination. Combinations may be 1 to 8 digits long...all the dip...all the switch!

How to test

No Clue

External hardware

To be used with the Tiny Tapeout project PCB. Dip switches and 7-segment display are on TT-PCB.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input 1	Display Segment 1	—
1	Input 2	Display Segment 2	—
2	Input 3	Display Segment 3	—
3	Input 4	Display Segment 4	—
4	Input 5	Display Segment 5	—
5	Input 6	Display Segment 6	—
6	Input 7	Display Segment 7	—
7	Input 8	Display Segment 8	—

TinyTapeout

by **bignug13**

0296 Wokwi Project

`github.com/bignug13/TinyTapeout`
`wokwi.com/projects/446364165240923137`

"something to do with meshtastic"

How it works

I don't know yet. Something with Meshtastic.

How to test

I'll let you know when I know.

External hardware

Prolly a T-Echo

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	pin 1	output 1	—
1	pin 2	output 2	—
2	pin 3	output 3	—
3	pin 4	output 4	—
4	pin 5	output 5	—
5	pin 6	output 6	—
6	pin 7	output 7	—
7	pin 8	output 8	—

mysteryform

by **tc**

0297 Wokwi Project

`github.com/mystery-form/tc1`

`wokwi.com/projects/446363928119066625`

"segment configs by tc"

How it works

Various 'and', 'or', etc statements create different segments to illuminate from unique dip switch on/off configurations

How to test

TBD

External hardware

LED Display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	b	c
1	a	b	c
2	a	b	c
3	a	b	c
4	a	b	c
5	a	b	c
6	a	b	c
7	a	b	c

sparkhom SR flip flop

by **Serguey Parkhomovsky**

0298 Wokwi Project

github.com/sparkhom/wokwi

wokwi.com/projects/446363696828310529

"A SR flip flop"

How it works

A basic SR flip flop. Input 0 is R and Input 1 is S.

How to test

Pull input 1 high to set and input 0 high to reset.

External hardware

N/A

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	reset	Q	—
1	set	Q	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

LeviathonGds

by **Levi**

0299 | 1 Hz | Wokwi Project

`github.com/AstralVX/chip_tapeout`

`wokwi.com/projects/446363784753530881`

"Count from 0 to 7"

How it works

Converts a 3 bit binary coded decimal to the seven segment LCD.

It will print '0' to '7' and then loop, incrementing the digit every second.

Logic create using truth tables, sum of products, boolean expression simplification, Karnaugh mapping.

How to test

Turn clock switch on.

Hit reset button to start from 0.

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	out_0	—
1	—	out_1	—
2	—	out_2	—
3	—	out_3	—
4	—	out_4	—
5	—	out_5	—
6	—	out_6	—
7	—	out_7	—

DUMB-Turing

by **Yuanda Liu**

`0300` `50 Hz` `HDL Project`

`github.com/yliu-hashed/ttsky25b-dumb-turing`

"A Turing machine in Turing's original mathematical construction"

Background

Turing machines are mathematical models of computation. It is proposed by Alan Turing, and is used in the Church–Turing thesis to prove that anything computable can be computable by a Turing machine. This gives birth to the concept of Turing completeness. If a machine is Turing complete, it can compute anything.

Unfortunately, modern computers are not literal Turing machines. They are Turing complete, though, but it is different in structure from Turing's original design.

A Turing machine formally refers to a DFA (a meatly FSM) of finite size that has access to an infinite tape. The DFA takes the data on the current tape head as input, and based on the current state, takes a state transition while writing a symbol on the tape before deciding to move the tape head either left or right. By designing a set of state transitions, someone can perform any computation on the tape.

How it works

DUMB-Turing is a Turing machine in the Literal sense. It uses external SPI ROM and RAM for the transition table and (emulated) tape. This Turing machine has 128 states and an alphabet size of 256.

Transition Table

The transition table is modelled by an external SPI ROM attached to the UIO[3:0] pins. The SPI ROM used must be a 2-byte-addressed ROM.

The table is made up of 2-byte entries of transitions. Follow this formula to find the byte address of the entry of the current state and content on the tape.

```
addr = { state_number, tape_data, 1'b0 }
     = state_number * 512 + tape_data * 2
```

In each entry, the first byte dictates the tape data to be written, and the next byte dictates the tape-head movement direction and the next state.

Bits	Description
Byte0[7:0]	New tape data to write (before moving the tape head)
Byte1[7]	Tape head movement (left = 0, right = 1)
Byte1[6:0]	The next state

Since this Turing machine needs to use all 128 states, **you must attach an exactly 64K-byte SPI memory to this port!**

Special IO States

There are 2 special states at 0x7F and 0x7E. They behave like other states except for what is read or written on the tape.

State	Name	Description
0x7E	read	Read the UI[7:0] for making transitions instead of reading the tape
0x7F	write	Write the tape and output the new tape value to UO[7:0]

These states allow arbitrary usage of IO.

Tape Movement

One change I made that is inconsistent with Turing's mathematical description is the use of a circular tape. The SPI controller will wrap around the address space when the tape head moves past the limit below 0x0 or above 0xFFFF. This gives the illusion of a circular tape of 64K-bytes. The SPI memory needs to be attached to the UIO[7:4] pins. This port uses different SCK, MISO, and MOSI than the Transition Table.

This machine cannot be used with bigger SPI memory, as it will use 3 3-byte addresses instead of 2. However, this machine can work with smaller SPI memory, provided that the SPI memory ignores the upper used address bits.

To achieve the best reasonably achievable performance, DUMB-Turing uses a tape cache as well as a tape movement predictor. The cache is a write-through cache that saves 8 bytes of data near the tape head. This allows the Turing machine to compute on a small range of tape efficiently. Furthermore, the tape movement predictor uses movement history to predict how the tape is going to move next. This allows prefetching some tape before the Turing machine needs it. These optimizations are usually only featured in much more complicated processors. But since the memory access pattern of a tape is always linear, they can be integrated here in a much more "dumbed down" version.

How to test

This design is difficult to test, as it requires one or both SPI memory devices to be filled with data before execution begins.

I will provide updated instructions on how to build a Transition Table later.

External hardware

This design requires one 512Kbit ROM and one 512Kbit RAM. The ROM can be emulated by the on-board RP2040.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	gpio_out[0]	gpio_in[0]	state_spi_cs
1	gpio_out[1]	gpio_in[1]	state_spi_mosi
2	gpio_out[2]	gpio_in[2]	state_spi_miso
3	gpio_out[3]	gpio_in[3]	state_spi_sck
4	gpio_out[4]	gpio_in[4]	tape_spi_cs
5	gpio_out[5]	gpio_in[5]	tape_spi_mosi
6	gpio_out[6]	gpio_in[6]	tape_spi_miso
7	gpio_out[7]	gpio_in[7]	tape_spi_sck

AndNand

by **David Pulcifer**

0301 Wokwi Project

`github.com/DavidPulcifer/ttsky-wowki-template`
`wokwi.com/projects/446363829539267585`

"Output 1 is a not and gate, output 2 is an and gate"

How it works

Outputs NAND and AND from the same inputs

How to test

signal to a and or b connections

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	input a	a nand b	—
7	input b	a and b	—

my first wokwi design

by **Eric Zhang**

0302 Wokwi Project

github.com/Zhane4994/My-wowki

wokwi.com/projects/445163733427466241

"makes a clock controller"

How it works

NAND is connected to input 0 and 1 which is connected to output 0 AND is connected to input 2 and 3 which is connected to output 1 OR is connected to input 4 and 5 which is connected to output 2

How to test

Input a & b	Output NAND	Input c & d	Output AND	Input e & f	Output OR
0 0	1	0 0	0	0 0	0
0 1	1	0 1	0	0 1	1
1 0	1	1 0	0	1 0	1
1 1	0	1 1	1	1 1	1

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output nand	—
1	input b	output and	—
2	input c	output or	—
3	input d	—	—
4	input e	—	—
5	input f	—	—
6	—	—	—
7	—	—	—

Tiny Tapeout 20251031

by **Jacob Still**

0303 Wokwi Project

`github.com/jcstill/tinytapeout`

`wokwi.com/projects/446847576277095425`

"8x8 Shift Register Matrix Multiplier"

How it works

This is an 8 bit x 8 bit Multiplier that uses a shift register to injest data. The result of the calculation is parallel data out on the output pins.

How to test

Use `Data In` and `Clock` to shift in 2 bytes Y then X. MSB first.

```
     |1  1|1  0|0  0|0  0|
bit  |5  2|1  8|7  4|3  0|

53             |0011|0101|
77             |0100|1101|
=    ---------------------
4081 |0000|1111|1111|0001|
```

As of 2025-11-06, the output is paralel out. It will output the least significat byte of the result. I aim to add a buffer and a shift register for the output.

External hardware

clock signal data signals

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Data In	o0	o8
1	—	o1	o9
2	—	o2	o10
3	—	o3	o11
4	—	o4	o12
5	—	o5	o13
6	—	o6	o14
7	—	o7	o15

TinyTapeout Wokwi

by **Felix Feierabend**

0320 Wokwi Project

github.com/feierf/tt_wokwi

wokwi.com/projects/442984054310760449

"a simple 4:1 MUX"

How it works

A simple 4:2 MUX

How to test

I/O	Descriptionq
In1	Bit select Line 0
In2	Bit select Line 1
In3-6	Inputs
Output 1	Output

External hardware

not neccessary

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Bit select line 1	Out	—
1	Bit select line 2	—	—
2	In 1	—	—
3	In 2	—	—
4	In 3	—	—
5	In 4	—	—
6	—	—	—
7	—	—	—

8-Bit-CPU

by **Fabian Poell**

0322 | 50 MHz | HDL Project

`github.com/fabianp1704/8-Bit-CPU`

"A simple 8-bit-CPU."

How it works

This project implements a minimal 8-bit CPU consisting of a simple datapath and two finite-state machines.

How to test

Provide serial input data through the ui_in pins, then observe the 8-bit output.

External hardware

No external hardware is required.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Start (trigger serial input)	CPU_OUT[0]	Carry flag
1	Serial data input	CPU_OUT[1]	Zero flag
2	MUX_SEL0	CPU_OUT[2]	Overflow flag
3	MUX_SEL1	CPU_OUT[3]	Negative flag
4	MUX_SEL2	CPU_OUT[4]	—
5	MUX_ENABLE	CPU_OUT[5]	—
6	—	CPU_OUT[6]	—
7	—	CPU_OUT[7]	—

Hamming (7,4) Encoder and Decoder

by **Florian Berger**

0324 Wokwi Project

github.com/Nadere03/ttsky_wokwi_hamming74_encode_decode

wokwi.com/projects/442977579441572865

"An encoder and decoder for the hamming (7,4) error correction scheme"

How it works

This project is an encoder and decoder for the Hamming (7,4) code with automated single-bit error correction. Therefore, any multi-bit error are wrongly decoded. However, single-bit errors are corrected and the occurence of an error is reported by the scheme.

Input 7 selects between encode (0) and decode (1) operation. Output 7 reports the correction of a single-bit error in decode operation and is ignored for encoding. Inputs andd outputs 0-6 for the code word as described in the info.yaml.

How to test

For encoding, set the four data bits and observe the output word. For decoding apply a valid codeword with less than 2 errors and observe the correct output word.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Encode: Ignored, Decode: Parity 1	Parity 1	—
1	Encode: Ignored, Decode: Parity 2	Parity 2	—
2	Encode: Data 1, Decode: Data 1	Data 1	—
3	Encode: Ignored, Decode: Parity 3	Parity 3	—
4	Encode: Data 2, Decode: Data 2	Data 2	—

#	Input	Output	Bidirectional
5	Encode: Data 3, Decode: Data 3	Data 3	—
6	Encode: Data 4, Decode: Data 4	Data 4	—
7	Mode Selection, Encode: 0, Decode: 1	Encode: Ignored, Decode: Single-Bit Correction Occured (1)	—

Tiny Tapeout

by **daniel**

0326 Wokwi Project

`github.com/BreitenbaumerD/Test-Wokiwi`

`wokwi.com/projects/442977842875423745`

"null"

How it works

Turning Input 1 to 4 on

How to test

Start

External hardware

none List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input1	Output1	—
1	Input2	—	—
2	Input3	—	—
3	Input4	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Test

by **Robert**

0328 | 10 MHz | Wokwi Project

`github.com/robonaj/tiny-tapeout-workshop`
`wokwi.com/projects/442977476963131393`

"Random gates placed during workshop"

How it works

It does not do anything useful yet.

How to test

Power on and hope for the best.

External hardware

Microswitch for input, oscilloscope of output :)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	FF_in	FF_q	—
1	—	FF_qb	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	ring_enable	ring_out	—

Patrick first TT chip

by **Patrick Sieberer**

0329 | 10 kHz | Wokwi Project

github.com/siebsi/Patricks-first-TT-chip

wokwi.com/projects/442977463421808641

"Displaying P, if correct code is entered"

How it works

That is a secret

How to test

Try out different switch positions

External hardware

DIP switch, 7 seg display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	clk	out0	—
1	rst_n	out1	—
2	in0	out2	—
3	in1	out3	—
4	in2	—	—
5	in3	—	—
6	—	—	—
7	—	—	—

Polyphonic MIDI-Synth

by **Simon Grundner**

`0330` `3.5 MHz` `HDL Project`

`github.com/s-grundner/LVA-EIC-KV`

"Decodes MIDI and outputs several square waves to a max of 4 output pins, which can be mixed together externally"

How it works

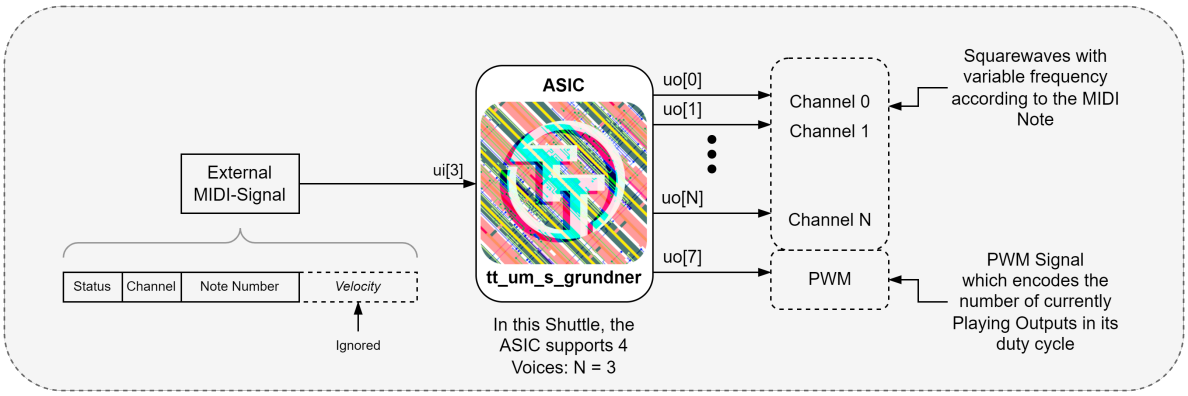


Figure 330.1: Abstract Blockdiagram

The Project generates **Square Waves** based on a MIDI note recieved on the input Pin. The oscillator stack is capable of synthesizing up to 4 voices simultaneously. Each voice is routed to a different output pin, allowing for external mixing of the signals. Each Oscillator responds to a **different MIDI-Channel**, which is necessary to simplify the voice allocation logic inside the ASIC. Additionally, a PWM signal is provided on a separate output pin, which encodes the number of currently active voices in its duty cycle. This PWM signal can be used to control the gain of the mixed output through lowpass filtering.

The number of voices is constrained by the available area. If further optimizations can be applied or another tile is added, you can change the number of voices by updating the `OSC_VOICES` parameter in `src/global.v` and see if the area is sufficient after regenerating the GDS.

How the MIDI protocol works

MIDI (Musical Instrument Digital Interface) is a technical standard that describes a protocol, digital interface, and connectors for connecting various electronic musical instruments and computers together.

MIDI messages are transmitted as a series of bytes. Each message typically consists of a status byte followed by one or more data bytes. The status byte

indicates the type of message and the MIDI channel it is associated with, while the data bytes contain additional information such as note number and velocity.

More on MIDI messages can be found here.

How the ASIC responds to Edge Cases

If a second note ON command is received for a voice that is already active, the ASIC will output the new frequency on the same output pin, effectively overriding the previous note. Releasing the previous note will not stop the sound. Only when the note OFF command of the currently active note is received, the oscillator will stop. This behaviour is called voice stealing.

If a channel is recieved outside of the voice range the behavior is as follows: Overflowing channels will eventually loop around when surpassing the size of the channel register. All inputs inbetween the max number of voices and the register size will be ignored.

The ASIC only responds to Note ON and Note OFF commands. Other MIDI commands such as Control Change, Pitch Bend etc. may result in undefined behavior.

The ASIC always expects 3 bytes per MIDI message. If the UART receiver gets out of sync, it may interpret random data as valid MIDI messages.

How to generate MIDI messages

There are multiple ways to feed the ASIC with MIDI messages:

1. Use a Microcontroller to Emulate a MIDI device. This is the easiest way to get started. You can use an e.g. the onboard RP2040 or any other microcontroller with UART capabilities.
2. Use a MIDI Controller (Piano, Pads) with a native MIDI-OUT DIN connector. You will need a PMOD as a physical layer to convert the differential signal to a single-ended UART signal.
3. Use a Computer to send MIDI Messages via a USB serial interface. You can use the RP2040 or an FTDI Chip and send raw MIDI messages over a serial connection.
4. If you have a Digital Audio Workstation (DAW) available, you can send MIDI messages from virtual instruments inside the DAW to a virtual MIDI Port.

Here are some software recommendations for option 4:

- Create a virtual MIDI Port on your Computer with loopMIDI by Tobias Erichsen (Free for personal use)
- Connect a virtual MIDI interface to a COM-Port with Hairless MIDI Serial (FOSS under GPLv2)
- Digital Audio Workstations:

- LMMS (FOSS under GPLv2)
- Ableton Live (Proprietary, Paid)

Internal Structure

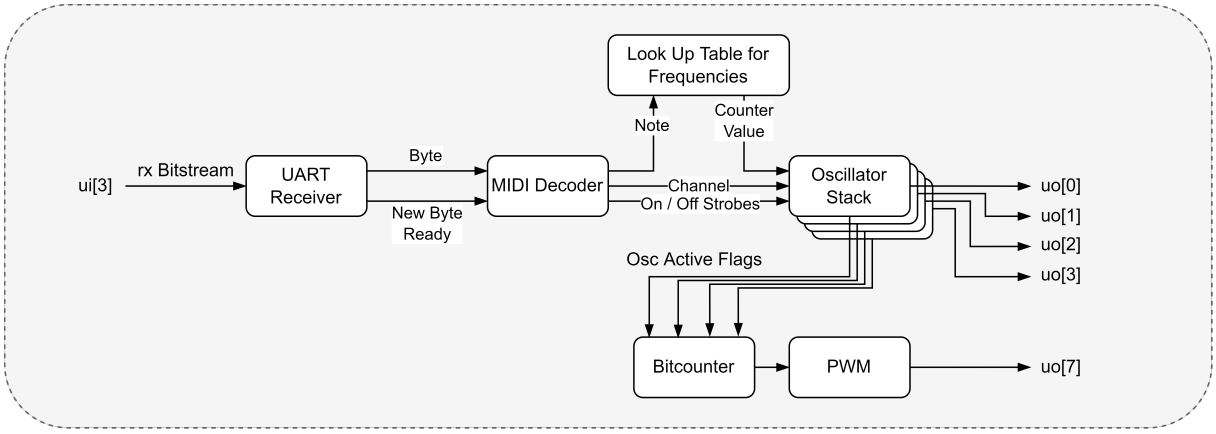


Figure 330.2: Internal Structure

How to test

To test the ASIC, feed it with Note-On-MIDI messages with one of the methods above. Observe the signals present on the output with an oscilloscope or logic analyzer. You should see square waves on the output pins corresponding to the notes played on the MIDI controller or sent from the DAW. Additionally, you can monitor the PWM output on pin 7 to see the number of active voices.

External hardware

In the simplest setup, a speaker with a driver circuit can be connected to an output pin to hear the synthesized audio.

- MIDI Source (MIDI Controller, DAW, Microcontroller)
- (Optional) External Mixing circuitry to combine the output signals from the oscillator outputs.
- Speaker with a driver

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	osc_0	—
1	—	osc_1	—
2	—	osc_2	—
3	midi_rx	osc_3	—
4	—	—	—

#	Input	Output	Bidirectional
5	—	—	—
6	—	—	—
7	—	pwm	—

Simple Logic-Gate Controlled LEDs

by **Valentin Spindler**

0331 | 1 Hz | Wokwi Project

`github.com/ValeriusO1/TinyTapeout`

`wokwi.com/projects/442983748797580289`

"LED - Lighting, Switch controlled"

How it works

You just connect up the switches and the 7-Seg Board and then the fun can start

How to test

The . - Point should blink at a frequency of 1Hz and you can do turrn on some Segments by using the bottom 4 switches

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	OUT0	—
1	—	—	—
2	—	—	—
3	—	OUT3	—
4	IN4	—	—
5	IN5	—	—
6	IN6	OUT6	—
7	IN7	OUT7	—

counter7seg

by **Cetinkaya Ozan**

0332 Wokwi Project

github.com/OzanBC/ozan

wokwi.com/projects/442987128372461569

"a counter with an 7seg display"

How it works

There is a simple counter build with FlipFlops whuch is triggered by a push-button and then the number displayed on a 7 segment display

How to test

push the button and see if it counts right

External hardware

a pushbutton and a 7 segment display is needed the pushbutton might be debounced by a condensator

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Button	Seg_A	—
1	—	Seg_B	—
2	—	Seg_C	—
3	—	Seg_D	—
4	—	Seg_E	—
5	—	Seg_F	—
6	—	Seg_G	—
7	—	Seg_H	—

jku-sky-LEDSpinner

by **Tim Tremetsberger**

0333 | 50 MHz | HDL Project

`github.com/timba307/jku-sky-LEDSpinner`

"LED Spinner on 7-Seg display with variable speeds"

Module: `tt_um_timba307_LEDSpinner`

Play Roulette using a 7-segment display hooked up to the chip. Using the TinyTapeout Demo Board, the built-in 7-segment can be used directly.

How it works

The "wheel" of the Roulette is compromised of the outer segments on the 7-segment display. When powering on, the wheel will instantly start spinning. The speed can be configured using `ui_in[3:0]`. These pins influence the size of the prescaler. The `prescaler.v` module generates ticks slow enough to enable seeing the rotating led segments. For every tick, a pseudo-random number is generated using a linear feedback shift register in the `rng.v` module. The `wheel.v` module controls the position of the active led on the ring and the stopping behaviour. If the user wants to stop the wheel `ui_in[7]` is activated. From this point on the wheel will use the random number to choose a stopping location and will spin till it reaches that location. Beforehand, the user takes a guess using the `uio_in[5:0]`. If the wheel lands on the correct guess (multiple guesses are also allowed), the `guess_eval.v` module will light up the point of the 7-seg display. The control of the 7-segment display is handled using the `seg_driver.v` module. After `ui_in[7]` is deactivated the wheel will continue to spin again and a new guess can be made.

Signal	Dir	W	Purpose
`ui_in[3:0]`	in	4	Set speed, only `4b0000`, `4b0001`, `4b0011`, `4b0111`, `4b1111` are allowed for 1 Hz, 2 Hz, 4 Hz, 8 Hz and 16 Hz respectively. Invalid config triggers 1 kHz speed for testing purposes.
`ui_in[7]`	in	1	Stop the wheel
`uio_in[0]`	in	1	Guess top segment
`uio_in[1]`	in	1	Guess top right segment
`uio_in[2]`	in	1	Guess bottom right segment
`uio_in[3]`	in	1	Guess bottom segment
`uio_in[4]`	in	1	Guess bottom left segment

`uio_in[5]`	in	1	Guess top left segment
`uo_out[6:0]`	out	7	7-segment outer ring
`uo_out[7]`	out	1	7-segment display point
`clk`	in	1	System clock
`rst_n`	in	1	Async reset (active-low)
`ena`	in	1	Always 1 on TinyTapeout

How to test

Every module has a dedicated testbench file for gtkwave plot generation. Run the `simulate.sh` script to automatically simulate the plots and open gtkwave.

External hardware

If the TinyTapeout Demo Board is used, just one additional set of dip-switches is needed for the bidirectional inputs. If the demo board is not used, then two sets of dip-switches and a 7-segment display is needed.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	speed_bit0	seg_top	guess_top
1	speed_bit1	seg_top_right	guess_top_right
2	speed_bit2	seg_bot_right	guess_bot_right
3	speed_bit3	seg_bot	guess_bot
4	—	seg_bot_left	guess_bot_left
5	—	seg_top_left	guess_top_left
6	—	seg_middle	—
7	stop_wheel	seg_dp	—

tt-test

by **Valentin Jochinger**

0334 Wokwi Project

`github.com/skiller-bat/tt-project`

`wokwi.com/projects/442977648674888705`

"Simple Test Project"

How it works

It's just a little test project connecting some inputs to outputs via 4 logic gates. The outputs are wired to the 7-segment display.

How to test

Just enable either input 3 or input 5 and the 7-segment display displays a letter. Input 7 causes the display to show the dot.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	—	—	output 7-segment display
2	—	—	output 7-segment display
3	—	—	output 7-segment display
4	—	—	output 7-segment display
5	—	—	—
6	—	—	—
7	—	—	output 7-segment display

4 Bit Counter

by **Tim Müllner**

0335 | 10 kHz | Wokwi Project

`github.com/timmuellner/tinytapeout4BitCounter`

`wokwi.com/projects/442977533467790337`

"4 Bit Binary Counter with T Flip Flops to divide the frequency"

How it works

11 flip flops are used to divide the clock speed, the new slower clock is connected to 4 D-Flip Flops which act as a 4 bit counter

How to test

connect to Clock

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	Output bit 1	—
1	—	Output bit 2	—
2	—	Output bit 3	—
3	—	Output bit 4	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Arko

by **Arko**

0416 | 50 MHz | HDL Project

`github.com/arkorobotics/ttsky-verilog-arko`

"Arko"

How it works

Does stuff and makes things happen, again

How to test

Do stuff and make the things do things

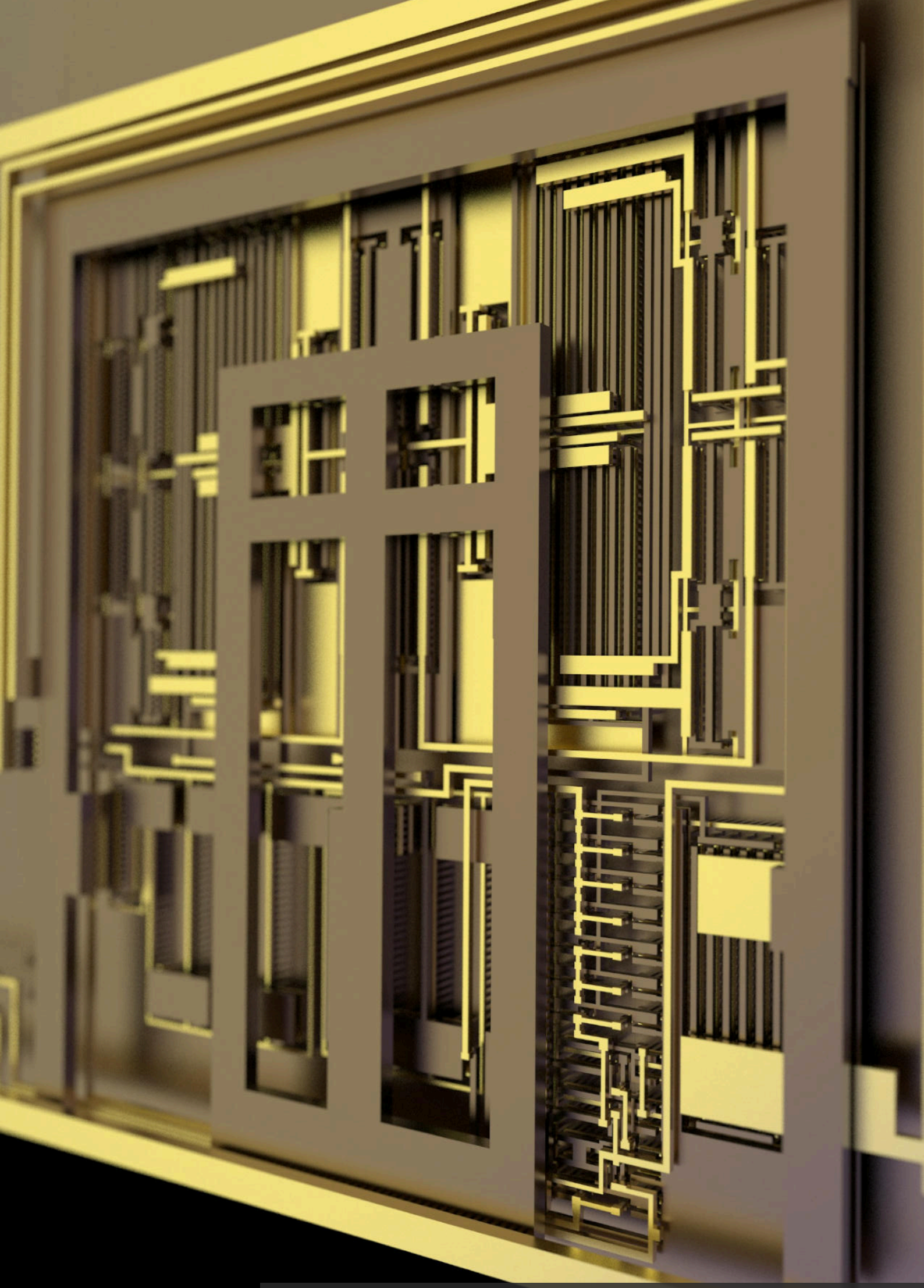
External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in0	out0	io0
1	in1	out1	io1
2	in2	out2	io2
3	in3	out3	io3
4	in4	out4	io4
5	in5	out5	io5
6	in6	out6	io6
7	in7	out7	io7



555 render (TT06) – Designed by Vincent Fusco. Illustrated by Máximo Balestrini.

4Bit_CPU

by **Maximilian Zaglmayr**

0417 | 10 MHz | HDL Project

`github.com/MaxZGit/Custom-4Bit-CPU`

> *"a small custom 4 bit cpu with 16 instructions and 16x4 Bit storage"*

How it works

TESTETSet

How to test

TESTSEt

External hardware

external

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	next_data_in_strb	OUT0
1	—	data_valid_strb	OUT1
2	EN_PROG	st_programm	OUT2
3	RX	st_fetch_i	OUT3
4	IN0	st_decode	INSTR_REG0
5	IN1	st_fetch_op	INSTR_REG1
6	IN2	st_fetch_mdr	INSTR_REG2
7	IN3	st_execute	INSTR_REG3

DETDFF (Tiny Tapeout)

by **Patrick D. Lloyd**

0418 Wokwi Project

github.com/pdlloyd/detdff_tiny-tapeout

wokwi.com/projects/446362347802682369

"4x Dual-Edge-Triggered D-Flip Flops for Tiny Tapeout"

How it works

This is a collection of four D-type flip-flops that each load & store the input, D, to output, Q, on *either* the rising or falling clock edge. There are also asynchronous RESET inputs.

Although there are inputs on the DETDFFs called "CLK", the application this circuit was designed for doesn't utilize a traditional periodic input waveform. The Tampercut project aims to use this circuit as a way to detect if the physical enclosure of a computer server is jostled, tilted, or tampered with. The idea is to connect various binary sensors to the CLK inputs, and set D to VCC (logic high). Then, when a sensor triggers (either active-high or active-low), even briefly, the logic-high value will be latched to the output, Q. This persistent state will remain on the output until the DETDFF is explicitly reset, either through the RST_N button, or by driving D low and manually clocking the sensor trigger. QN doesn't really serve much purpose except providing a complementary output on the 7-segment display.

How to test

This circuit does not make any use of the provided clock input, only the piano input switches and the RST_N button. The design won't build properly with a floating clock so there is a bogus inverter in the design not connected to anything to get the Github Actions to pass. The outputs, Q, of the DETDFFs and their complements, QN, are shown on the 7-segment display:

```
DETDFF0 (D0 / IN0, CLK0 / IN1) -> Q0  (Top Segment          /  A)
                               -> QN0 (Bottom Segment       /  D)

DETDFF1 (D1 / IN2, CLK1 / IN3) -> Q1  (Top Left Segment     /  F)
                               -> QN1 (Top Right Segment    /  B)

DETDFF2 (D2 / IN4, CLK2 / IN5) -> Q2  (Bottom Left Segment  /  E)
                               -> QN2 (Bottom Right Segment /  C)

DETDFF3 (D3 / IN6, CLK3 / IN7) -> Q3  (Dot LED              / DP)
                               -> QN3 (Middle Segment       /  G)
```

1. Inital conditions: All D pins should be set to ON (connecting the inputs to VCC). CLK pins can be set to ON (VCC) or unset to OFF (GND) arbitrarily. Press RST_N to zero out the flip-flop outputs. The display should show **F** (for **F**un).
2. Operation: Any toggleing of a CLK will cause the corresponding segment and its complement to toggle and stay in that toggled state until either a reset, or a logic-low on the D input is clocked in.

External hardware

No external hardware is required for testing, but take a look at Tampercut for more info on this project's application.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	D0	Q0	—
1	CLK0	QN0	—
2	D1	Q1	—
3	CLK1	QN1	—
4	D2	Q2	—
5	CLK2	QN2	—
6	D3	Q3	—
7	CLK3	QN3	—

j-initial-cycler

by **jordan richman**

0419 Wokwi Project

`github.com/jrichman-r7/tiny-tapeout-2025`
`wokwi.com/projects/446373432938861569`

"displays a J on 7-segment display when dipswitch 1 is active"

How it works

Displays a J when dipswitch 1 is on

How to test

Do the above

External hardware

LED Display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	—	—
1	input from button	on	—
2	—	on	—
3	—	on	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

wokwi-adder

by **Alexander Golubow**

0420 Wokwi Project

github.com/agolubow/tts-wokwi

wokwi.com/projects/446363844132307969

"hopefully this will add some numbers"

How it works

8 inputs are given, with the first 4 being the first number and second 4 being the second number. these numbers will then be added.

How to test

Use a truth table and see if it matches.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a0	—	—
1	input a1	—	—
2	input a2	—	—
3	input a3	—	—
4	input b0	—	—
5	input b1	—	—
6	input b2	—	—
7	input b3	—	—

TinyTapeout2

by **Kevin**

0421 Wokwi Project

`github.com/ArigganKnight/TinyTapeout`

`wokwi.com/projects/446368563224770561`

"Programmable Logic Function"

How it works

It uses input 4-7 to populate a LUT to implement a simple logic function.

How to test

Configure it by presenting a truth table on inputs 4-7. Give it inputs on 0 and 1, and it'll give you the output on 0.

External hardware

There is none.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	q	—
1	b	—	—
2	—	—	—
3	—	—	—
4	v00	—	—
5	v01	—	—
6	v10	—	—
7	v11	—	—

OctoPWM

by **Camden Miller**

0422 | 10 MHz | HDL Project

`github.com/camdenmil/tinytapeout_sky25b`

"8 channel PWM driver controlled over SPI"

How it works

This is an 8 channel PWM driver which is controlled over SPI. It was designed for a 10MHz clock but other input frequencies should also work.

SPI CS and both enable inputs are active-low. The SPI interface expects a 16 bit word per transaction. SPI polarity/phase CPOL=0, CPHA=0.

The PWM driver takes an input clock and increments a 10 bit counter by one each clock cycle. The driver also has a 10 bit user-configurable compare threshold register. Driver output is high when the counter starts at zero. When the counter exceeds the value in the threshold register, the driver output will go low until the counter resets. There are 8 instances of the PWM driver in this design.

The system clock passes through a clock divider which can be configured to divide the clock frequency by any power of 2 from 2^0 - 2^15 before it is passed into the PWM drivers.

The SPI interface resets when CS is asserted so it must be held low for the entirety of the transaction. The design expects one 16 bit word per transaction of the following form.

Bits	15:12	11:10	9:0
Field	Write Address	Not used	Write Value

The PWM drivers are address 0-7, the specific address corresponding to which `uo_out` pin it drives. Address 8 is the clock divider which accepts only the lower 4 bits of the write value (the upper 6 bits have no effect for the clock divider). When 16 bits are clocked in, the write will immediately take effect. When the compare threshold is changed for a channel, it will set its output low for the remainder of the cycle until the counter resets to zero after which the newly-written compare threshold will take effect on waveform generation. All 8 PWM drivers are fixed to be in-phase with each other. When writting to the clock divider, the lower 4 bits of the value defines the division factor N in a 2^N divisor.

The reset pin will reset all PWM compare thresholds to zero (so 0% duty cycle) and set the clock divider to 0 (i.e. no division).

`ui_in0` is a global active-low output enable pin. Pull this pin high to disable output for all the PWM drivers (note that this does not stop the internal counters).

`uio_out7` is the internal clock divider output. This signal can be toggled on/off with the active-low enable pin at `ui_in1`.

I made this in one night with no HDL experience and the testbench is woefully lacking so hopefully it works ;)

How to test

Make sure `ui_in0` is low and feed in a 10MHz clock to `clk`. Hold the `rst_n` pin low for at least one clock cycle after selecting the design with the TT mux. Do a simple 16 bit transaction to one of the PWM drivers over the SPI bus to set its duty cycle (the POR reset compare threshold is zero). To match testbench behavior, assert CS at least one system clock cycle before shifting in the first bit with SCK/MISO. For example, to set PWM0 to 50% duty cycle, write `0x01FF`. After writing that value, there should be a 50% duty cycle waveform on `uo_out`.

External hardware

No external hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	out_enable	PWM 0	SPI CS
1	clk_div_out_enable	PWM 1	—
2	—	PWM 2	SPI MISO
3	—	PWM 3	SPI SCK
4	—	PWM 4	—
5	—	PWM 5	—
6	—	PWM 6	—
7	—	PWM 7	clock_div_out

Jon_And_Angie_08281999

by **Jon**

0423 Wokwi Project

`github.com/ghangaskan/JonsSecondTinyTapeout`

`wokwi.com/projects/446359782214875137`

"Celebrates our marriage covenant by crystalizing our union"

How it works

There are two modes of operation. The dip switches can be used to drive a single letter state. OR you can use the button as a clock to cycle through the letters in order.

How to test

press the clock button and read the seven segment display It should cycle through: JONANDANgIE

External hardware

Seven segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	J	—	—
1	O	—	—
2	dash	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Sleepy Chip

by **Ron Sardarian**

0424 | 50 MHz | Wokwi Project

github.com/teafella/sleepy_tapeout

wokwi.com/projects/0

"SPI-controlled 3-waveform synthesizer with streaming mode and delta-sigma DAC"

How it works

Sleepy Chip is a dual-mode digital audio synthesizer featuring a 3-waveform oscillator with streaming capability and delta-sigma DAC output. It provides both traditional synthesis (oscillator + waveform mixing) and direct sample streaming in a compact 1x1 Tiny Tapeout tile.

Architecture

```
SPI Interface          24-bit Phase          Waveform
Mode
(3 pins)               Accumulator           Generators
Selection
                       (DDS core)
uio[0]: MOSI     ┌─→   ┌──────────┐          ┌──────────┐
┌──────┐
uio[1]: SCK      |     | Frequency|──────────→| Square   |──┐     |
|
uio[2]: CS       |     | Register |          | Sawtooth |    ├──→|
MUX  |
                 |     |          |          | Triangle |    |    |
|
     ┌──────────┴─┐  | 24-bit   |          └──────────┘    |    | OSC/
|
     | SPI RX     |  | 50 MHz   |                          |
|STREAM|─┐
     | Registers  |  └──────────┘          ┌──────────┐    |    |
| |
     | (8 regs)   |                        | Mixer    |──┘
└──────┘ |
     └────────────┘                       | (3-ch)   |
|
           |                                └──────────┘
|
           |
|
           | Streaming Sample
|
```

```
            | Register (0x10)
──────────────────────────────────────────────┘
            |
|
            |                                   ┌──────────┐
┌──────────┐ |
            | Volume Register                   | Volume   |    | Delta-
| |
            └──────────────────────────────→| Control  |◄──| Sigma DAC
|◄┘
                                                | 8-level  |    | (1-
bit)    |
                                                └─────────┘
└────┬────┘
                                                                        |
                                                                        ▼

uo_out[0]
                                                                 (Audio
Output)
```

Features

Dual Operating Modes:

1. **Oscillator Mode** - Traditional synthesis with 3 waveforms:
   - Square wave (variable duty cycle 0-100%)
   - Sawtooth wave (linear ramp, all harmonics)
   - Triangle wave (symmetric fold, odd harmonics)
   - Individual waveform enable/disable
   - 3-channel mixer with saturation protection
2. **Streaming Mode** - Direct sample playback:
   - 8-bit sample input via SPI
   - Direct to volume control and DAC
   - Suitable for PCM streaming or custom waveforms

Common Features:

- 24-bit phase accumulator (DDS) - 2.98 Hz resolution, <1 cent musical accuracy
- 8-level volume control - Instant bit-shift implementation (mute to full)
- First-order delta-sigma DAC - 1-bit 50 MHz PDM output
- SPI register interface - Mode 0 (CPOL=0, CPHA=0), RX-only

Register Map

Address	Name	Bits	Description

0x00	Control	[0]: OSC_EN[1]: STREAM_MODE[2]: SW_GATE[3]: SQUARE_EN[4]: SAW_EN[5]: TRI_EN	Oscillator enable, mode selection, waveform enables
0x02	Freq Low	[7:0]	Frequency control word low byte
0x03	Freq Mid	[7:0]	Frequency control word middle byte
0x04	Freq High	[7:0]	Frequency control word high byte (24-bit total)
0x05	Duty Cycle	[7:0]	Square wave PWM duty cycle (0x00=0%, 0xFF=100%)
0x06	Volume	[7:5]	Volume level (8 discrete levels, bit-shift based)
0x10	Stream Sample	[7:0]	Streaming mode sample value (0x00-0xFF)
0x12	Status	[0]: GATE[1]: OSC_RUN	Read-only status register

Technical Specifications

Phase Accumulator:

- Resolution: 24-bit (16,777,216 steps/cycle)
- Frequency Range: 2.98 Hz to 25 MHz
- Musical Accuracy: <1 cent error across full audio range
- Example: 440 Hz (A4) = 0x024000

Volume Control:

- 8 discrete levels based on top 3 bits of volume register
- Bit-shift implementation (instant response, area-optimized)
- Levels: mute, 1/8, 1/4, 3/8, 1/2, 5/8, 3/4, full

Delta-Sigma DAC:

- Type: First-order with error feedback
- Output Rate: 50 MHz (1-bit PDM)
- External Filter: Simple RC low-pass (Fc 25 kHz recommended)

Resource Usage:

- Total: 870 instances (63% of 1x1 tile)
- SPI Interface: 45 cells
- Phase Accumulator: 60 cells
- Waveform Generators: 103 cells
- Delta-Sigma DAC: 50 cells

Pin Configuration

Dedicated Inputs (`ui_in[7:0]`)

Pin	Name	Function	Description
`ui_in[0]`	GATE	Gate Input	Hardware gate trigger (OR'd with software gate)
`ui_in[1]`	HW_RST	Reset	Hardware reset (active high, AND'd with rst_n)
`ui_in[7:2]`	-	Reserved	Unused (future expansion)

Dedicated Outputs (`uo_out[7:0]`)

Pin	Name	Function	Description
`uo_out[0]`	**DAC_OUT**	**Audio**	**1-bit delta-sigma audio output**
`uo_out[1]`	GATE_LED	Status	Gate status indicator (HW gate OR SW gate)
`uo_out[2]`	OSC_RUN	Status	Oscillator running indicator
`uo_out[3]`	SYNC	Debug	Sync pulse (phase MSB, frequency/ waveform visualization)
`uo_out[7:4]`	-	Reserved	Unused (future expansion)

Bidirectional IOs (`uio[7:0]`)

Pin	Name	Function	Description
`uio[0]`	MOSI	SPI Data	SPI Master Out Slave In (data from controller)
`uio[1]`	SCK	SPI Clock	SPI clock input from controller
`uio[2]`	CS	SPI Select	SPI chip select (active low)
`uio[7:3]`	-	Reserved	Unused (future expansion)

Note: All bidirectional pins are configured as inputs (SPI is RX-only, no MISO output to save area).

How to test

Basic Setup

1. **Power On**: Apply 50 MHz clock to the chip
2. **SPI Controller**: Connect microcontroller or SPI master to:
   - `uio[0]` - MOSI
   - `uio[1]` - SCK
   - `uio[2]` - CS (active low)
3. **Audio Output**: Connect `uo_out[0]` (DAC_OUT) to RC low-pass filter

Oscillator Mode Test

```
# SPI write function: write_register(address, data)

# 1. Enable oscillator with sawtooth waveform
write_register(0x00, 0b00010001)  # OSC_EN=1, STREAM=0, SAW_EN=1

# 2. Set frequency to 440 Hz (A4 note)
write_register(0x02, 0x00)  # Freq low
write_register(0x03, 0x40)  # Freq mid
write_register(0x04, 0x02)  # Freq high = 0x024000

# 3. Set volume to full
write_register(0x06, 0xFF)

# Result: Clean 440 Hz sawtooth wave on audio output
```

Streaming Mode Test

```
# 1. Switch to streaming mode
write_register(0x00, 0b00000010)  # OSC_EN=0, STREAM_MODE=1

# 2. Stream different sample values
for sample in [0x00, 0x40, 0x80, 0xC0, 0xFF]:
    write_register(0x10, sample)
    time.sleep(0.001)  # 1ms per sample = 1 kHz sample rate

# 3. Set volume
write_register(0x06, 0x80)  # 50% volume

# Result: Direct sample playback at your chosen rate
```

Expected Results

Oscillator Mode:

- **440 Hz tone** at test frequency
- **Different timbres** per waveform:
  - Square: Bright, hollow sound (odd harmonics)
  - Sawtooth: Bright, buzzy sound (all harmonics)
  - Triangle: Mellow, pure sound (odd harmonics, softer)

- **Multiple waveforms** can be mixed by enabling multiple EN bits

Streaming Mode:

- **Direct control** of DAC output value
- **Custom waveforms** by streaming sample sequences
- **PCM playback** at your chosen sample rate

Volume Control

Test all 8 volume levels:

```
volumes = [0x00, 0x20, 0x40, 0x60, 0x80, 0xA0, 0xC0, 0xFF]
names = ["Mute", "1/8", "1/4", "3/8", "1/2", "5/8", "3/4", "Full"]

for vol, name in zip(volumes, names):
    write_register(0x06, vol)
    print(f"Volume: {name}")
    time.sleep(0.5)
```

External hardware

Minimum Setup

RC Low-Pass Filter (required):

```
DAC_OUT (uo_out[0]) ──┬── 10kΩ ──┬── to audio amp
                      │          │
                     GND       680pF
                                 │
                                GND
```

- R = 10kΩ
- C = 680pF
- Fc ≈ 23 kHz
- Removes 50 MHz carrier from PDM signal

Audio Amplifier (recommended):

- Simple op-amp buffer (TL072, LM358) or
- LM386 audio amplifier
- Speaker (8Ω) or headphones

SPI Controller (required):

- Microcontroller (Arduino, ESP32, STM32, etc.)
- Raspberry Pi
- Or any SPI master device
- Mode 0 (CPOL=0, CPHA=0)
- Up to 1 MHz SPI clock speed

Filter Options

Alternative cutoff frequencies:

- **15 kHz**: R=10kΩ, C=1nF (more filtering, softer sound)
- **25 kHz**: R=10kΩ, C=680pF (recommended, balanced)
- **35 kHz**: R=10kΩ, C=470pF (less filtering, brighter sound)

Full System Example

```
┌──────────────┐
│ Arduino/     │ MOSI ──→ uio[0]
│ Raspberry Pi │ SCK  ──→ uio[1]
│ (SPI Master) │ CS   ──→ uio[2]
└──────────────┘


┌──────────────┐
│ Sleepy Chip  │ uo[0] ──→ RC Filter ──→ LM386 ──→ Speaker
│ (TT09)       │ uo[1] ──→ GATE LED
│              │ uo[2] ──→ OSC LED
└──────────────┘
```

Design Verification

All modules validated with comprehensive testbenches:

- ✅ **SPI RX Interface**: All 8 registers, burst writes, read-only status
- ✅ **Phase Accumulator**: Frequency accuracy <0.25% @ 440Hz
- ✅ **Waveform Generators**: Square, sawtooth, triangle verified
- ✅ **Mixer**: 3-channel mixing with saturation protection
- ✅ **Volume Control**: All 8 levels verified in both modes
- ✅ **Streaming Mode**: Sample updates, mode switching
- ✅ **Delta-Sigma DAC**: Output modulation working correctly
- ✅ **End-to-End**: 12 comprehensive integration tests passing

SPI Protocol

Mode: SPI Mode 0 (CPOL=0, CPHA=0) **Speed**: Up to 1 MHz SCK **Format**: 2-byte transactions: [address][data]

Basic Write:

1. Assert CS (low)
2. Clock out address byte (8 bits, MSB first)
3. Clock out data byte (8 bits, MSB first)
4. Deassert CS (high)

Burst Write (auto-increment):

1. Assert CS (low)
2. Clock out address byte
3. Clock out data byte 1
4. Clock out data byte 2 (writes to address+1)
5. Clock out data byte 3 (writes to address+2)

6. Deassert CS when done

Frequency Calculation

To calculate frequency word for a desired frequency:

```
freq_word = (desired_freq_Hz * 2^24) / 50_MHz
```

Examples:

- **440 Hz (A4)**: 0x024000 (147,456)
- **1000 Hz**: 0x051EB8 (335,544)
- **C4 (261.63 Hz)**: 0x015820 (88,096)
- **A3 (220 Hz)**: 0x012000 (73,728)

Use Cases

1. **Musical Synthesizer** - Traditional subtractive synthesis with waveform mixing
2. **Sample Player** - Stream custom waveforms or short samples via SPI
3. **Sound Effects** - Generate tones, sweeps, and effects for games/devices
4. **Test Equipment** - Programmable audio frequency generator
5. **Educational** - Learn digital synthesis, DSP, and SPI communication
6. **MIDI Controller** - Microcontroller reads MIDI, controls synth via SPI

References

- Source code: `src/` directory
- Test benches: `test/` directory
- Tested with Icarus Verilog and Cocotb
- Clock: 50 MHz
- Technology: Sky130 PDK (via Tiny Tapeout 09)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	GATE	DAC_OUT	SPI_MOSI
1	HW_RST	GATE_LED	SPI_SCK
2	—	OSC_RUN	SPI_CS
3	—	SYNC	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Seven Segment Display Driver

by **Benjamin Meyer**

0425 Wokwi Project

github.com/CrypticXVZ/BenWokwiTinyTapeOut

wokwi.com/projects/446364777751946241

"Takes in a binary number and displays it on the Seven segment display"

How it works

This works by inputing a binary number 0-9 and turning it into a signal for a 7 segment display ## How to test

To test put any binary number 0-9

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	Segment A	—
1	—	Segment B	—
2	—	Segment C	—
3	—	Segment D	—
4	Input 4	Segment E	—
5	Input 3	Segment F	—
6	Input 2	Segment G	—
7	Input 1	DP	—

Simple Tapped Delay Line

by **Carson**

0426 Wokwi Project

`github.com/carson-mpspace/tinytapeout-demo`

`wokwi.com/projects/446363771447095297`

"Tapped Delay line formed with inverters and FFs"

How it works

It's a tapped delay line! Useful perhaps for measuring time between start and stop signals. I have no idea how fast the signal will propagate through the delay line formed with the inverters, so this is a good characterisation test for future projects.

How to test

Launch a rising edge on start and some time after, launch it on stop, and then flick through the mux'es to see how far the start signal has travelled down the delay line!

External hardware

Signal generator and possibly a FPGA would be nice

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	start	tap0	—
1	stop	tap1	—
2	mux0	tap2	—
3	mux1	tap3	—
4	mux2	tap4	—
5	ff_r	tap5	—
6	—	tap6	—
7	—	tap7	—

First WOWKI Project

by **Andrew McGRegor**

0427 Wokwi Project

github.com/MightyKingDrew/supreme-happiness
wokwi.com/projects/446363833426340865

"Testing Wowki"

How it works

There is an AND gate on 1 and two connected to an LED.

How to test

I don't know.

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	1	—	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Lights-Out

by **Brent Mellor**

0428 | 1 kHz | HDL Project

`github.com/BMellor/tiny-tapeout-project`

"Lights-Out game using 3x3 multiplexed LED's and buttons"

This project implements a simple 3x3 Lights-Out game in Verilog.

The goal of the game is to turn all LEDs off. Pressing a button toggles that LED and its neighboring LEDs.

1. Reset → board is empty.
2. Press any button → random board loads.
3. Press buttons to toggle LEDs.
4. When all LEDs are off → DONE goes high.
5. Press any button again → a new random game starts.

- **Wokwi Simulation:** https://wokwi.com/projects/446364136220529665
- **Breakout Board KiCAD:** https://github.com/BMellor/tiny-tapeout-breakout

How it works

1. High-Level Overview

The design includes:

- A 3x3 LED matrix
- 3 row button inputs (matrix-scanned to detect 9 buttons)
- LED multiplexing logic
- Button debouncing
- A 16-bit LFSR for random game generation
- Lights-Out toggle logic
- A DONE signal when all LEDs are off

2. LED Matrix Representation

The game board is stored in:

```
reg [8:0] leds;
```

This 9-bit register represents the 3x3 grid in row-major order:

```
Index layout:

0 1 2
3 4 5
6 7 8
```

If a bit is:
`1` → LED is ON
`0` → LED is OFF

3. LED Multiplexing (Column Scanning)
The design uses column multiplexing:

```
reg [1:0] active_col;
```

- active_col cycles 0→1→2 repeatedly.
- Only one column is active at a time.
- Row outputs are driven according to:
  - The active column
  - The stored LED state

This allows a 3×3 grid to be driven using:

- 3 row lines
- 3 column lines

Because scanning happens at 1kHz, the display appears continuously lit.

4. DONE Signal (Win Detection)
DONE is assigned as:

```
DONE = !(|leds);
```

If any LED is on, DONE = 0.
If all LEDs are off, DONE = 1.

The player wins when the board is completely cleared.

5. Button Matrix and Debouncing
There are 3 physical button inputs:

```
BTN_ROW0
BTN_ROW1
BTN_ROW2
```

Because columns are scanned, each row line corresponds to a different button depending on the active column:

Column 0 → buttons 0, 3, 6
Column 1 → buttons 1, 4, 7
Column 2 → buttons 2, 5, 8

Each of the 9 buttons has a 16-bit shift register:

```
reg [15:0] btn_shift[0:8];
```

These are used for debouncing.

How it works:

- On each clock cycle, the current button value shifts into the register.
- If the register becomes `16'h7FFF` (15 stable 1s after a 0), a clean rising-edge press is detected.
- When this happens, btn_debounced[i] is set to 1 for one clock cycle.

This produces a single clean pulse per valid press.

6. Random Game Generation (LFSR)

A 16-bit Linear Feedback Shift Register (LFSR) runs continuously. Each clock cycle, the LFSR shifts and computes a new feedback bit.

Polynomial:
`x^16 + x^14 + x^13 + x^11`

When the board is empty (leds == 0) and any button is pressed, the lower 9 bits of the LFSR are loaded as a new random board.

```
leds <= lfsr[8:0];
```

This allows a new game to start after a win.

7. Lights-Out Toggle Logic

Each button has a predefined toggle mask such as:

```
localparam [8:0] TOGGLE_MASK0 = 9'b000001011;
```

Each mask determines:

- The pressed LED
- Its neighboring LEDs (up, down, left, right)

When a button is pressed the XOR operation flips the selected bits.

```
leds <= leds ^ TOGGLE_MASKx;
```

The board update is the XOR of:

- The current board
- The mask for any pressed button

Generally only one button press is processed at a time due to debouncing.

External Hardware

There is a pre-designed breakout board at the following repo: https://github.com/BMellor/tiny-tapeout-breakout

I/O Pin Description

Pin Name	Signal Name	Function	Connection / Usage
IN_0	BTN_ROW0	Row 0 button input for scanned button matrix.	Connect to row 0 of 3x3 button matrix

			(pull-down resistor required).
IN_1	BTN_ROW1	Row 1 button input for scanned button matrix.	Connect to row 1 of 3x3 button matrix (pull-down resistor required).
IN_2	BTN_ROW2	Row 2 button input for scanned button matrix.	Connect to row 2 of 3x3 button matrix (pull-down resistor required).
OUT_0	LED_ROW0	Drives LED row 0 of the matrix.	Connect to row 0 of LED matrix (Cathode of LED's through current-limiting resistor).
OUT_1	LED_ROW1	Drives LED row 1 of the matrix.	Connect to row 1 of LED matrix (Cathode of LED's through current-limiting resistor).
OUT_2	LED_ROW2	Drives LED row 2 of the matrix.	Connect to row 2 of LED matrix (Cathode of LED's through current-limiting resistor).
OUT_3	COL0	Column 0 select for LED drive and button scan.	Connect to column 0 of LED matrix (Anode of LED's) and button matrix.
OUT_4	COL1	Column 1 select for LED drive and button scan.	Connect to column 1 of LED matrix (Anode of LED's) and button matrix.
OUT_5	COL2	Column 2 select for LED drive and button scan.	Connect to column 2 of LED matrix (Anode of LED's) and button matrix.
OUT_6	DONE	Indicates win condition (all LEDs off).	Connect to anode of indicator LED

Notes

- The LED and button matrix share the same 3 column lines (COL0–COL2).
- Add pull-down resistors on button row inputs to prevent them from floating when the buttons aren't pressed.
- Add current limiting resistors on LED row outputs, not the columns.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	BTN_ROW0	LED_ROW0	—
1	BTN_ROW1	LED_ROW1	—
2	BTN_ROW2	LED_ROW2	—
3	—	COL0	—
4	—	COL1	—
5	—	COL2	—
6	—	DONE	—
7	—	—	—

My Tiny Tapeout

by **Shawn Hymel**

429 Wokwi Project

github.com/ShawnHymel/tiny-tapeout-workshop

wokwi.com/projects/446363830909757441

"My first Tiny Tapeout"

How it works

2-bit 7 segment driver. It converts a 2-bit binary (0-3) to 7-segment display pin outputs.

How to test

Inputs 0 and 1 correspond to input bits 0 and 1. Output pins connect to 7-segment display:

0: a 1: b 2: c 3: d 4: e 5: f 6: g 7: dp

External hardware

Drive it with whatever you want.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in0	out0	—
1	—	out1	—
2	—	out2	—
3	—	out3	—
4	—	out4	—
5	—	out5	—
6	—	out6	—
7	in7	out7	—

Tiny Tapeout Workshop Supercon 25

by **Boris Nimcevic**

0430 | 2 Hz | Wokwi Project

`github.com/borisnimcevic/tinytapeout-supercon25`

`wokwi.com/projects/446364133350576129`

"Test - update later"

How it works

This design prints my name on the 7-segment display. (The last letter is '.', just so there is an obvious break to the name, but there is always something lit up, so it is easier for debugging)

Prints boris, dario, ivan and natasa.

How to test

IDK, will add later

External hardware

IDK, will add later

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	input b	output b	—
2	input c	output c	—
3	input d	output d	—
4	input e	output e	—
5	input f	output f	—
6	input g	output g	—
7	input h	output h	—

Fastest Finger First

by **Hannes Páll Þórðarson**

0431 | 50 MHz | Wokwi Project

github.com/v207/tinytapeout

wokwi.com/projects/446363696538901505

"8 player Fastest Finger Game"

8-Player Fastest Finger First Game Author: Hannes Pall Thordarson Wokwi Project: https://wokwi.com/projects/446363696538901505 Description This project is an 8-player Fastest Finger First game, perfect for a large quiz or game show.

It's built using only standard digital logic gates (like AND, OR, NOT) and D-type flip-flops. There is no microcontroller or CPU; the entire game logic is implemented in pure hardware.

When the game is reset, the first player to press their button will light their corresponding LED. This action simultaneously locks out all other players, preventing them from registering a press until the game is reset again.

How it Works The design uses a clever combinatorial feedback loop to manage the game state. Each of the 8 players has an identical logic cell, which consists of one D-flip-flop, one AND gate, and one OR gate.

A single, global "Game Armed" signal is used to lock out players.

Global Lock Logic: The "Game Armed" signal is generated by feeding all 8 flip-flop outputs (Q0...Q7) into an 8-input OR gate. The output of this OR gate is then inverted with a NOT gate.

Game_Armed = NOT (Q0 OR Q1 OR ... OR Q7) Player Latch Logic: For each player, the D-input of their flip-flop is controlled by their OR gate. This OR gate's inputs are:

The flip-flop's own Q output (to keep itself "on" once set). The output of their AND gate. Player Input Logic: Each player's AND gate checks for two conditions:

Is the player's button being pressed? Is the "Game Armed" signal high (meaning no one has won yet)? Game Sequence:

Reset: The rst_n signal clears all 8 flip-flops. All Q outputs are 0. Game Armed: Because all Q outputs are 0, the 8-input OR gate outputs 0. The NOT gate inverts this, making the "Game Armed" signal 1. The game is now live. Player 1 Presses: Player 1's AND gate sees Button_1 = 1 and Game_Armed = 1, so its output becomes 1. This 1 passes through the OR gate to Player 1's D-input.

First Clock Edge: On the next clock, Player 1's flip-flop latches the 1, and Q1 becomes 1. Player 1's LED turns on. Lockout: Instantly, the 8-input OR gate sees Q1 is 1, and its output goes to 1. The NOT gate inverts this, and the "Game Armed" signal drops to 0. Game Locked: This 0 signal is fed to all 8 player AND gates, disabling them. Even if Player 2 presses their button, their AND gate will output 0. Winner Stays Lit: Player 1's light stays on because their D-input is fed by their OR gate, which is now receiving a 1 from its own Q1 output. How to Test Assert the rst_n pin (set to 0). De-assert the rst_n pin (set to 1). The game is now armed. Press any of the 8 player buttons (ui_in[0] through ui_in[7]). The corresponding LED pin (uo_out[0] through uo_out[7]) will go high. Try pressing the other player buttons. They will have no effect, as the game is locked. Go back to step 1 to start a new round. Pinout This design uses the 8 dedicated inputs (ui_in) for the players and the 8 dedicated outputs (uo_out) for the LEDs. The uio_ (bidirectional) pins are not used.

ui_in (Inputs) uo_out (Outputs) ui_in[0]: Player 1 Button uo_out[0]: Player 1 LED ui_in[1]: Player 2 Button uo_out[1]: Player 2 LED ui_in[2]: Player 3 Button uo_out[2]: Player 3 LED ui_in[3]: Player 4 Button uo_out[3]: Player 4 LED ui_in[4]: Player 5 Button uo_out[4]: Player 5 LED ui_in[5]: Player 6 Button uo_out[5]: Player 6 LED ui_in[6]: Player 7 Button uo_out[6]: Player 7 LED ui_in[7]: Player 8 Button uo_out[7]: Player 8 LED Dedicated Chip Pins:

clk: The main clock signal used by all flip-flops. rst_n: The active-low reset signal to start a new game.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	Input b	output b	—
2	input c	output c	—
3	input d	output d	—
4	input e	output e	—
5	input f	output f	—
6	input g	output g	—
7	input h	output h	—

Brainf*ck ASIC

by **Matt Prock**

448 | 40 MHz | HDL Project

`github.com/map588/tt_um_brainfck_asic`

*"A Brainf*ck interpreter on a chip! Uses external SPI RAM for the tape, 2-wire serial for user I/O and interrupt-driven bracket handling via MCU collaboration."*

How it works

Main Module

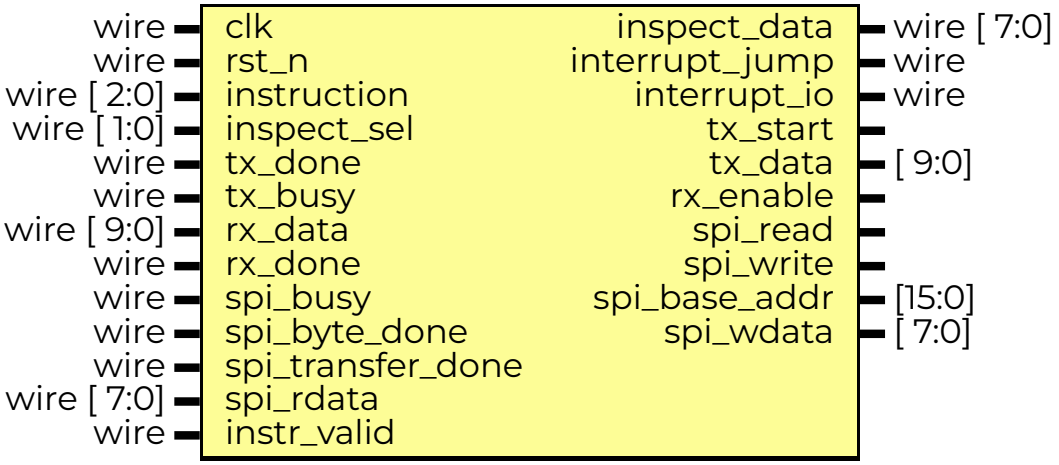


Figure 448.1: Module

The ASIC executes Brainfuck instructions using the RP2040 for user I/O and SPI "tape". Each 3-bit instruction is decoded into data operations (+/-), pointer operations (</>), I/O operations (,/.), or bracket operations ([/]).

The design features a 9-byte data cache with SPI RAM backend for the full 1024-byte tape. When the pointer moves beyond the cached window, the ASIC writes old data to SPI and fetches new data in 5-byte bursts.

The 8-deep bracket stack supports nested loops with interrupt-driven jump handling. Bracket operations collaborate with the MCU:

- [with data=0: ASIC triggers `interrupt_jump` and enables RX. MCU sends PC of matching] via 10-bit serial.
- [with data≠0: ASIC pushes PC to bracket stack and continues (enter loop).
-] with data=0: ASIC pops bracket stack and continues (exit loop).
-] with data≠0: ASIC triggers `interrupt_jump` and transmits top of bracket stack via 10-bit serial TX. MCU resumes from that PC (loop back).

Two interrupt signals notify the MCU: `interrupt_jump` for bracket operations requiring PC transmission, and `interrupt_io` for user I/O. Communication between the BF ASIC and RP2040 occurs over two 2-wire serial interfaces (10-bit RX/TX for PC values and data) and SPI for the "tape" memory. The MCU monitors interrupts to handle I/O and jump requests.

Main State machine

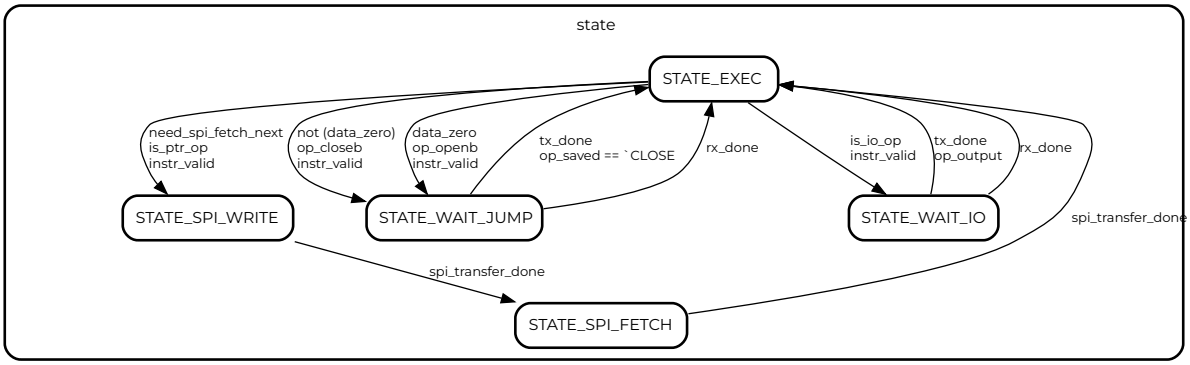


Figure 448.2: State Machine

How to test

- Write some BF (Like this echo loop ,[.,])
- Use the following encoding to translate it to what the BF_ASIC can understand: `'-' => "000"` `'+' => "001"` `'<' => "010"` `'>' => "011"` `'[' => "100"` `']' => "101"` `',' => "110"` `'.' => "111"` Heres a python script to process .b files: link
- Connect to the pico via UART for I/O
- `++++++++++[>+++++++>++++++++++>+++<<<-]>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>++++++++++.`
- hello world!

External hardware

- The TT I/O board including the RP2040 / RP2350
- RP2xx0 with this firmware installed
- A protoboard, wires, and patience

Credit for SPI firmware: https://github.com/MichaelBell/spi-ram-emu

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	instruction[0]	tx_bit	spi_cs
1	instruction[1]	tx_clk	spi_mosi
2	instruction[2]	interrupt_io	spi_miso
3	instr_valid	interrupt_jump	spi_sck
4	rx_clk	inspect_data[0]	inspect_data[4]
5	rx_bit	inspect_data[1]	inspect_data[5]
6	inspect_sel[0]	inspect_data[2]	inspect_data[6]
7	inspect_sel[1]	inspect_data[3]	inspect_data[7]

VGA Screensaver with embedded bitmap ROM

by **Uri Shaked**

`0450` `25.175 MHz` `HDL Project`

`github.com/urish/tt-rom-vga-screensaver`

"64x64x6bpp Tiny Tapeout Logo bouncing around the screen (640x480, TinyVGA Pmod)"

How it works

Displays a bouncing 64x64x6bpp Tiny Tapeout logo on the screen:

Figure 450.1: Tiny Tapeout screensaver

How to test

Connect to a VGA monitor. Set the following inputs to configure the design:

- `tile` (ui_in[0]) to repeat the logo and tile it across the screen
- `double` (ui_in[1]) to double the logo size to 128x128

If you have a Gamepad Pmod connected, you can also use the following controls:

- Start button: start/pause bouncing
- Left/right/up/down: change the bouncing direction (if bouncing) or move the logo around the screen (if paused)

External hardware

- Tiny VGA Pmod
- Optional: Gamepad Pmod

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	tile	R1	—
1	double	G1	—

#	Input	Output	Bidirectional
2	—	B1	—
3	—	VSync	—
4	gamepad_latch	R0	—
5	gamepad_clk	G0	—
6	gamepad_data	B0	—
7	—	HSync	—

My (S)VGA Playground

by **rej**

0452 | 25.175 MHz | HDL Project

`github.com/rejunity/tt-vga-sandbox`

"(S)VGA experiments"

How it works

My playground for (S)VGA experiments

How to test

Run it!

External hardware

VGA PMOD, VGA monitor

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	R1	—
1	—	G1	—
2	—	B1	—
3	—	VSync	—
4	—	R0	—
5	—	G0	—
6	—	B0	—
7	—	HSync	—

ASAP CPU v1

by **Aaron Libokuohai Bovensiepen**

0454 | 1 MHz | HDL Project

`github.com/libokuohai/ttsky-2025-asap-cpu`

"A SAP-1 inspired CPU with a fibonacci program"

How it works

This project is a minimal SAP-1 style CPU with an 8-bit data path and a 4-bit address space (16 bytes). The micro-architecture includes:

- **Program Counter (PC)**: 4-bit, increments during fetch, can be loaded on jumps.
- **Instruction Register (IR)**: holds opcode (upper 4 bits) and operand/address (lower 4 bits).
- **Registers**: A and B (8-bit), plus a 2-bit **FLAGS** register (`Z`, `C`).
- **ALU**: 8-bit adder with two's-complement subtraction; updates `Z` (zero) and `C` (carry).
- **Memory**: 16×8 array used for code and data (addresses 0x0–0xF).
- **Control**: a simple multi-cycle controller steps through fetch/decode/execute stages and asserts control lines (e.g., `CO`, `MI`, `RO`, `IO`, `II`, `AI`, `AO`, `BI`, `EO`, `SU`, `FI`, `OI`, `JP`, `CE`, `HT`).

Program: the built-in demo computes successive Fibonacci numbers by alternating loads/adds/stores and outputs via the `OUT` instruction (mirrored to the 7-segment via `bin_to_bcd` → `seven_seg`).
Display: the 8-bit OUT value is converted to BCD (0–255) and scanned across three digits. Segment lines are active-high; digit enables are active-low.
Clocks: a counter divides the external `clk` to make (1) a **CPU clock** (slow enough to watch values change) and (2) a faster **scan clock**. `uo[7]` exposes a divided-down **heartbeat** so you can verify liveness on a scope/LED.

The project was influenced by:

- SAP-1 Design (Digital Computer Electronics by A. P. Malvino and J. A. Brown)
- Ben Eater (https://www.youtube.com/@BenEater)
- Austin Morlan's FPGA work (https://austinmorlan.com/posts/8bit_breadboard_fpga/)
- Jason Kaufmann's Tiny Eater 8 Bit (https://github.com/jasonkaufmann/tt07-beneater8bit)

How to test

Wire the outputs like this:

- uo[6] -> 1 k–2.2 kΩ -> SEG_A
- uo[5] -> 1 k–2.2 kΩ -> SEG_B
- uo[4] -> 1 k–2.2 kΩ -> SEG_C
- uo[3] -> 1 k–2.2 kΩ -> SEG_D
- uo[2] -> 1 k–2.2 kΩ -> SEG_E
- uo[1] -> 1 k–2.2 kΩ -> SEG_F
- uo[0] -> 1 k–2.2 kΩ -> SEG_G
- uio[0] -> DIG0 cathode (EN_N)
- uio[1] -> DIG1 cathode (EN_N)
- uio[2] -> DIG2 cathode (EN_N)
- uio[3] -> DIG3 cathode (EN_N, unused/off by default)

Let the project run and you should see the fibonacci numbers being counted up to 233.

External hardware

- three 1-digit common-cathode 7-segment modules
- breadboard and jumpers

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	RESET (manual, active-high)	SEG_G	DIG0_EN_N
1	—	SEG_F	DIG1_EN_N
2	—	SEG_E	DIG2_EN_N
3	—	SEG_D	DIG3_EN_N (unused/off by default)
4	—	SEG_C	—
5	—	SEG_B	—
6	—	SEG_A	—
7	—	HEARTBEAT	—

First Wokwi design

by **Ilyass**

0456 | 10 kHz | Wokwi Project

github.com/ilyassjaadari/project-test
wokwi.com/projects/442977495241922561

"the logic circuit is testing for Wokwi"

How it works

the design is a test for Wokwi platform

How to test

it only a test for WOkwi

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	IN7	OUT7	—

Tiny Tapeout Wokwi Bsp

by **Leon**

0458 | 1 kHz | Wokwi Project

github.com/Le0n19/Tiny-Tapeout-Wokwi-Bsp

wokwi.com/projects/442977534651634689

"GDS"

How it works

The gates xnor, inv, and are connected with each other to the outputs 0-7

How to test

Click on play and test all the buttons out to see which combination works out
Click on the reset button if something is wrong

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	button 1	and inv xnor	—
1	button 2	and inv xnor	—
2	button 3	and inv and	—
3	button 4	and inv and	—
4	button 5	and inv and	—
5	button 6	and inv and	—
6	button 7	inv xnor	—
7	button 8	inv xnor	—

PWM_LED_dimmer

by **Michael Pichler**

0460 | 50 MHz | HDL Project

`github.com/MP5-git/PWM_LED_dimmer`

"Uses Buttons to change the brightness of an LED"

How it works

1. Use the first two buttons to adjust the brightness of the Dot on the 7-Seg display. The number displayed shows the current brightness level.

How to test

1. Reset the system.
2. Press the first button to increase brightness.
3. Press the second button to decrease brightness.
4. Observe the LED dot getting brighter or dimmer, while the display updates to show the new brightness level.

External hardware

No external hardware required.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	INC	SEG_A	—
1	DEC	SEG_B	—
2	—	SEG_C	—
3	—	SEG_D	—
4	—	SEG_E	—
5	—	SEG_F	—
6	—	SEG_G	—
7	—	SEG_H	—

tt_classic_vga_clock_by_TT

by **A Person**

0462 | 31.5 MHz | HDL Project

github.com/Some-Other-Random-Person/tt_classic_vga_clock_by_TT

"Crude implementation of an analog/classic clock to be displayed via VGA"

How it works

This project is a crude implementation of an analog/classic clock to be displayed on a VGA monitor.

The working principle is as follows:
The time is counted with the clock source for the chip. The clock input must be 31.5 MHz as this is also used for the VGA output. Due to space concerns, the calculation of the angles for the clockhands is performed with a CORDIC algorithm producing sine / cosine results. The pixels for drawing/rendering the clockhands can be derived from the sine / cosine values.
Additionally the clock has an alarm function, which is realized by comparing a set alarm time to the current time and activating an output upon a match. The output waveform is intended to be used with a buzzer, however this requires an external buffer/driver to avoid possibly loading the chip beyond its design limit.
The VGA output is realized according to the common VGA standard and the pinout supports the TinyTapeout VGA PMOD for easy interfacing.
The time and alarm time, as well as alarm on/off can be set using external pushbuttons.

How to test

Set the clock frequency to 31.5 MHz and initialize using reset. Connect a VGA monitor (ideally using the TinyTapeout VGA PMOD).
Connect the pushbuttons (5x) with pulldown resistors and connect a buzzer or something similar for the alarm signal (output is a 3.15 kHz square wave pulsed on/off at 1 Hz and a duty cycle of 50%.
Enjoy the extremely crude clock. - Hopefully :)

External hardware

The TinyTapeout VGA PMOD is recommended for the VGA interface.
Regarding the pushbuttons there is no specific requirement - debouncing measures are implemented, however practical bouncing behaviour will have to be specifically tested.

The alarm output is intended to be used with a buzzer but this may be a realm for creative external circuitry.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Hours adjust input	black white	buzzer output
1	Minutes adjust input	black white	—
2	Seconds adjust input	black white	—
3	Alarm time adjust input	vsync	—
4	Alarm on off toggle input	black white	—
5	—	black white	—
6	—	black white	—
7	—	hsync	—

kitt_scanner

by **yorimichi**

0481 | 10 MHz | HDL Project

`github.com/yorimichiykhm/ttsky-um_yorimichi_kittscanner`

"K.I.T.T scanner light controller"

How it works

Figure 481.1: alt text

This project is K.I.T.T. scanner lights controller.

Pin	Name	I/O	Descrption
ui[0]	ENA	I	enable
ui[2:1]	MODE[1:0]	I	mode
ui[3]	SPEED	I	speed select
ui[4]	OINV	I	inverse mode
ui[5]	OSEL	I	out select
uo[7:0]	LEDOUT[7:0]	O	LED K or A
uio[7:0]	monitor[7:0]	O	test monitor

How to test

1. Initial ENA = L.
2. Set MODE, SPEED, OINV and OSEL to select operation mode.
3. ENA = H to activate.
4. LEDs are controlled.
5. ENA = L to de-activate

SPEED

SPEED	Description
0	Normal speed
1	1.5x speed

MODE

MODE[1:0]	Description
0	Normal mode
1	Mode 1
2	Mode 2
3	Mode 3

OINV

□□□□□□□□□

OINV	Description
0	A-common mode
1	K-common mode

OSEL

OSEL	Description
0	Non-dimming mode
1	Dimming mode

External hardware

- 8 LEDs for K.I.T.T. scanner lights
- Switches for mode configuration and a start trigger

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	ENA	LEDOUT[0]	PWMOUT[0]
1	MODE[0]	LEDOUT[1]	PWMOUT[1]
2	MODE[1]	LEDOUT[2]	PWMOUT[2]
3	SPEED	LEDOUT[3]	PWMOUT[3]
4	OINV	LEDOUT[4]	PWMOUT[4]
5	OSEL	LEDOUT[5]	PWMOUT[5]
6	—	LEDOUT[6]	PWMOUT[6]
7	—	LEDOUT[7]	PWMOUT[7]

SIC-1 8-bit SUBLEQ Single Instruction Computer

by **Uri Shaked**

0482 HDL Project

`github.com/urish/tt-subleq-sic1`

"Hardware implementation of the 8-bit Single Instruction Computer"

How it works

SIC-1 is an 8-bit Single Instruction computer. The only instruction it supports is SUBLEQ: Subtract and Branch if Less than or Equal to Zero. The instruction has three operands: A, B, and C. The instruction subtracts the value at address B from the value at address A and stores the result at address A. If the result is less than or equal to zero, the instruction jumps to address C. Otherwise, it proceeds to the next instruction.

Memory map

The SIC-1 computer has an address space of 256 bytes, and and 8-bit program counter. The first 253 bytes are used for the program memory, and the last 3 bytes are used for input, output, and for halting the computer:

Address	Label	Read	Write
253	@IN	`ui` pins	Ignored
254	@OUT	Returns 0	`uo` pins
255	@HALT	Returns 0	Ignored

Setting the program counter to 253, 254, or 255 will halt the computer.

Each instruction is 3 bytes long, and the program counter is incremented by 3 after each instruction, except when a branch is taken.

For more information, check out the SIC-1 Assembly Language Reference.

Execution cycle

Each instruction takes two cycles to execute, regardless of whether a branch is taken or not. The execution of an instruction is divided into the following stages:

1. a. Write result: writes the result of the previous instruction back to memory. b. Read instruction: reads A, B, and C from the memory at the address pointed to by the program counter (PC).

2. Read data: Reads the values at addresses A and B and calculate the result of valA - valB. If the result is less than or equal to zero, set the PC to C. Otherwise, increment the PC by 3 to point to the next instruction.

The pseudocode for the execution cycle is as follows:

```
(1) result = valA - valB
    next_PC = result <= 0 ? C : PC + 3
    memory[A] <= result
    A <= memory[next_PC]
    B <= memory[next_PC+1]
    C <= memory[next_PC+2]
    PC <= next_PC
(2) valA <= memory[A]
    valB <= memory[B]
```

Where `valA` and `valB` are internal registers that hold the values read from memory at addresses A and B, respectively.

The <= symbol indicates a memory or register write operation, while the = symbol indicates a combinational assignment.

Control signals

The `uio` pins are used to load a program into the computer, and to control the computer:

uio pin	Name	Type	Description
0	run	input	Start the computer
1	halted	output	Computer has halted
2	set_pc	input	Set the program counter to the value on ui pins
3	load_data	input	Load the value from the ui pins into the memory addressed by PC
4	out_strobe	output	Pulsed for one clock cycle when the computer writes to @OUT (`uo` pins)
5	dbg[0]	input	Debug select bit 0
6	dbg[1]	input	Debug select bit 1
7	dbg[2]	input	Debug select bit 2

Debug interface

The `dbg` pins are used to expose internal signals for debugging on the `uo` pins. When the `dbg` pins are set to 0, the `uo` pins will output the @OUT value. For other values of `dbg`, the `uo` pins will output the following signals:

dbg[2:0]	Signal

0	None
1	PC
2	A
3	B
4	C
5	valA
6	result (valA - valB)
7	state (2 bits)

The `state` signal is a 2-bit value that represents the current state of the computer, corresponding to the execution cycle stages described above (1 and 2), and a 2-bit value of 0 when the computer is halted.

Programming the SIC-1

You can use the online SIC-1 app to compile and simulate your SIC-1 programs. Click on "Run game" and then "Apply for the job", close the "Electronic mail" popup. Paste the code and click on "Compile" (on the bottom left). You'll see the compiled code in the "Memory" window on the right, and will be able to step through the code.

To load a program and run a program, follow this sequence:

1. Set the `ui` pins to 0 (target address)
2. Pulse the the `load_pc` pin for a single clock cycle
3. Set the `ui` pins to the value you want to load
4. Pulse the `load_data` pin for a single clock cycle
5. Repeat steps 3-4 until you have loaded the entire program
6. Set the `ui` pins to the address you want to start at (usually 0)
7. Pulse the `set_pc` pin for a single clock cycle
8. Set the `run` pin to 1. The computer will start running the program, and the `halted` pin will go high when the program is done.

If you want to step through the program, you can pulse the `run` pin to advance one instruction at a time.

Reading the internal memory

You can read the internal memory of the SIC-1 while the program is halted. To do this, follow these steps:

1. Set the `run` pin to 0 to halt the computer, and wait for the `halted` pin to go high.
2. Set the `ui` pins to the address you want to read.
3. Pulse the the `load_pc` pin for a single clock cycle.
4. Set the `dbg` pins to `010` (binary for 2) to select the "A" debug signal.

5. Read the value on the `uo` pins.

The value on the `uo` pins will be the value stored in the internal memory at the address you specified.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in[0]	out[0]	run
1	in[1]	out[1]	halted
2	in[2]	out[2]	set_pc
3	in[3]	out[3]	load_data
4	in[4]	out[4]	out_strobe
5	in[5]	out[5]	dbg[0]
6	in[6]	out[6]	dbg[1]
7	in[7]	out[7]	dbg[2]

Classic 8-bit era Programmable Sound Generator AY-3-8913

by **Eric Farrow - ReJ aka Renaldas Zioma**

0483 | 2 MHz | HDL Project

`github.com/devmonkZA/devmonk-tto-ay8913`

> *"The AY-3-8913 is a 3-voice programmable sound generator (PSG) chip from General Instruments. The AY-3-8913 is a smaller variant of AY-3-8910 or its analog YM2149."*

How it works

This Verilog implementation is a replica of the classical **AY-3-8913** programmable sound generator. With roughly a 1500 logic gates this design fits on a **single tile** of the TinyTapeout.

The goals of this project

1. closely replicate the behavior and eventually the complete **design of the original** AY-3-891x with builtin DACs
2. provide a readable and well documented code for educational and hardware **preservation** purposes
3. leverage the **modern fabrication** process

A significant effort was put into a thorough **test suite** for regression testing and validation against the original chip behavior.

Chip technical capabilities

- **3 square wave** tone generators
- A single **white noise** generator
- A single **envelope** generator able to produce 10 different shapes
- Chip is capable to produce a range of waves from a **30 Hz** to **125 kHz**, defined by **12-bit** registers.
- **16** different volume levels

Registers The behavior of the AY-3-891x is defined by 14 registers.

Register	Bits used	Function	Description
0	xxxxxxxx	Channel A Tone	8-bit fine frequency
1	xxxx	—//—	4-bit coarse frequency
2	xxxxxxxx	Channel B Tone	8-bit fine frequency
3	xxxx	—//—	4-bit coarse frequency

4	xxxxxxxx	Channel C Tone	8-bit fine frequency
5	xxxx	—//—	4-bit coarse frequency
6	...xxxxx	Noise	5-bit noise frequency
7	..CBACBA	Mixer	Tone and/or Noise per channel
8	...xxxxx	Channel A Volume	Envelope enable or 4-bit amplitude
9	...xxxxx	Channel B Volume	Envelope enable or 4-bit amplitude
10	...xxxxx	Channel C Volume	Envelope enable or 4-bit amplitude
11	xxxxxxxx	Envelope	8-bit fine frequency
12	xxxxxxxx	—//—	8-bit coarse frequency
13	xxxx	Envelope Shape	4-bit shape control

Square wave tone generators Square waves are produced by counting down the 12-bit counters. Counter counts up from 0. Once the corresponding register value is reached, counter is reset and the output bit of the channel is flipped producing square waves.

Noise generator Noise is produced with 17-bit Linear-feedback Shift Register (LFSR) that flips the output bit pseudo randomly. The shift rate of the LFSR register is controller by the 5-bit counter.

Envelope The envelope shape is controlled with 4-bit register, but can take only 10 distinct patterns. The speed of the envelope is controlled with 16-bit counter. Only a single envelope is produced that can be shared by any combination of the channels.

Volume Each of the three AY-3-891x channels have dedicated DAC that converts 16 levels of volume to analog output. Volume levels are 3 dB apart in AY-3-891x.

Historical use of the AY-3-891x

The AY-3-891x family of programmable sound generators was introduced by General Instrument in 1978. Soon Yamaha Corporation licensed and released a very similar chip under YM2149 name.

Both variants of the AY-3-891x and YM2149 were broadly used in home computers, game consoles and arcade machines in the early 80ies.

- home computers: Apple II Mockingboard sound card, Amstrad CPC, Atari ST, Oric-1, Sharp X1, MSX, ZX Spectrum 128/+2/+3
- game consoles: Intellivision, Vectrex, Amstrad GX4000
- arcade machines: Frogger, 1942, Spy Hunter and etc.

The AY-3-891x chip family competed with the similar Texas Instruments SN76489.

The original pinout of the AY-3-8913

The **AY-3-8913** was a 24-pin package release of the AY-3-8910 with a number of internal pins left simply unconnected. The goal of AY-3-8913 was to reduce complexity for the designer and reduce the foot print on the PCB. Otherwise the functionality of the chip is identical to AY-3-8910 and AY-3-8912.

```
          ,--._.--.
   GND ---|1    24|<-- /cs*
  BDIR -->|2    23|<--  a8*
   BC1 -->|3    22|<-- /a9*
   DA7 <->|4    21|<-- /RESET
   DA6 <->|5    20|<-- CLOCK
   DA5 <->|6    19|--- GND
   DA4 <->|7    18|--> CHANNEL C OUT
   DA3 <->|8    17|--> CHANNEL A OUT
   DA2 <->|9    16|    not connected
   DA1 <->|10   15|--> CHANNEL B OUT
   DA0 <->|11   14|<-- test*
 test* <--|12   13|<-- VCC
          `-------'
   * -- omitted from this Verilog implementation
```

Difference from the original hardware

This Verilog implementation is a completely digital and synchronous design that differs from the original AY-3-8913 design which incorporated internal DACs and analog outputs.

Audio signal output While the original chip had no summation The module provides two alternative outputs for the generated audio signal:

1. digital 8-bit audio output suitable for external Digital to Analog Converter (DAC)
2. pseudo analog output through Pulse Width Modulation (PWM)

Master output channel In contrast to the original chip which had only separate channel outputs, this implementation also provides an optional summation of the channels into a single master output.

No DC offset This implementation produces output 0/1 waveforms without DC offset.

No /A8, A9 and /CS pins The combination of **/A8**, **A9** and **/CS** pins orginially were intended to select a specific sound chip out the larger array of devices connected to the same bus. In this implementation this mechanism is omitted for simplicity, **/A8**, **A9** and **/CS** are considered to be tied **low** and chip behaves as always enabled.

Synchronous reset and single phase clock The original design employed 2 phases of the clock and asynchronous reset mechanism for operation of the registers.

To make it easier to synthesize and test on FPGAs this implementation uses single clock phase and synchronous reset for registers.

The reverse engineered AY-3-891x

This implementation would not be possible without the reverse engineered schematics and analysis based on decapped AY-3-8910 and AY-3-8914 chips.

Explain how your project works

How to test

Summary of commands to communicate with the chip

The AY-3-8913 is programmed by updating its internal registers via the data bus. Below is a short summary of the communication protocol of AY-3-891x. Please consult AY-3-891x Technical Manual for more information.

BDIR	BC1	Bus state description
0	0	Bus is inactive
0	1	(Not implemented)
1	0	Write bus value to the previously latched register #
1	1	Latch bus value as the destination register #

Latch register address First, put the destination register adress on the bus of the chip and latch it by pulling both **BDIR** and **BC1** pins **high**.

Write data to register Put the desired value on the bus of the chip. Pull **BC1** pin **low** while keeping **BDIR** pin **high** to write the value of the bus to the latched register address.

Inactivate bus by pulling both **BDIR** and **BC1** pins **low**.

Register	Format	Description	Parameters
0,2,4	`ffffffff`	A/B/C tone period	`f` - low bits
1,3,5	`0000FFFF`	—//—	`F` - high bits
6	`000fffff`	Noise period	`f` - noise period
7	`00CBAcba`	Noise / tone per channel	`CBA` - noise off,
—	—	—	`cba` - tone off
8,9,10	`000Evvvv`	A/B/C volume	`E` - envelope on,
—	—	—	`v` - volume level

11	`ffffffff`	Envelope period	`f` - low bits
12	`FFFFFFFF`	—//—	`F` - high bits
13	`0000caAh`	Envelope Shape	`c` - continue, `a` - attack, `A` - alternate, `h` - hold

Note frequency

Use the following formula to calculate the 12-bit period value for a particular note:

$$toneperiod_{cycles} = clock_{frequency} / \left(16_{cycles} * note_{frequency}\right)$$

For example 12-bit period that plays 440 Hz note on a chip clocked at 2 MHz would be:

$$toneperiod_{cycles} = 2000000Hz / \left(16_{cycles} * 440Hz\right) = 284 = 11C_{hex}$$

An example to play a note at a maximum volume

BDIR	BC1	DA7..DA0	Explanation
1	1	`xxxx0000`	**Latch** tone A coarse register address $0 = 0000_{bin}$
1	0	`xxxx0001`	**Write** high 4-bits of the 440 Hz note $1 = 0001_{bin}$
1	1	`xxxx0001`	**Latch** tone A fine register address $1_{dec} = 0001_{bin}$
1	0	`00011100`	**Write** low 8-bits of the note $1C_{hex} = 00011100_{bin}$
1	1	`xxxx1000`	**Latch** channel A volume register address $8 = 1000_{bin}$
1	0	`xxx01111`	**Write** maximum volume level $15_{dec} = 1111_{bin}$ with the envelope disabled

```
Timing diagram

CLK   ____      ____      ____      ____      ____
____
   __/    `____/    `____/    `____/    `____/    `____/
`____ ...
    |         |         |         |         |         |
    |         |         |         |         |         |

BDIR ______    ______    ______    ______    ______    ______
   _/      `__/      `__/      `__/      `__/      `__/      `__

BC1  _______             _______             ________
   _/       `___________/       `__________/        `___________

DA7..DA0_____  ________  ________  ________  ________  ________
```

```
_/  0000  `/xxxx0001`/  0001  `/00011100`/  1000  `/xxx01111`
   latch     write     latch      write    latch
```

Externally configurable clock divider

SEL1	SEL0	Description	Clock frequency
0	0	Standard mode, clock divided by 8	1.7 .. 2.0 MHz
1	1	—//—	1.7 .. 2.0 MHz
0	1	New mode for TT05, no clock divider	250 .. 500 kHZ
1	0	New mode for TT05, clock div. 128	25 .. 50 MHz

SEL1	SEL0	Formula to calculate the 12-bit tone period value for a note
0	0	$clock_{frequency}/(16_{cycles}*note_{frequency})$
1	1	—//—
0	1	$clock_{frequency}/(2_{cycles}*note_{frequency})$
1	0	$clock_{frequency}/(128_{cycles}*note_{frequency})$

External hardware

The data bus of the AY-3-8913 chip has to be connected to microcontroller and receive a regular stream of commands. The AY-3-8913 produces audio output and has to be connected to a speaker. There are several ways how the overall schematics can be established.

8-bit parallel output via DAC One option is to connect off the shelf data parallel Digital to Analog Converter (DAC) for example Digilent R2R Pmod to the output pins and route the resulting analog audio to piezo speaker or amplifier.

```
uController             AY-3-8913
,---------.            ,---._.---.
|         |    2 Mhz ->|CLK  SEL0|<-- 0
|    GPIOx|----------->|BC1  SEL1|<-- 0
|    GPIOx|----------->|BDI      |         ,----------.
|    GPIOx|----------->|DA0  OUT0|-------->|LSB       |
|    GPIOx|----------->|DA1  OUT1|-------->|          |
|    GPIOx|----------->|DA2  OUT2|-------->|   pDAC   |  Headphones
|    GPIOx|----------->|DA3  OUT3|-------->|    or    |      or
|    GPIOx|----------->|DA4  OUT4|-------->| RESISTOR |
Buzzer
|    GPIOx|----------->|DA5  OUT5|-------->|  ladder  |         /|
|    GPIOx|----------->|DA6  OUT6|-------->|          |     .--/ |
|    GPIOx|----------->|DA7  OUT7|-------->|MSB       |-----|    |
`---------'            `---------'         `----------'     `--` |
                                                             |  `|
```

```
                                                                  |
                                                            GND ---
```

AUDIO OUT through RC filter Another option is to use the Pulse Width Modulated (PWM) AUDIO OUT pin that combines 4 channels with the Resistor-Capacitor based low-pass filter or better the Operation Amplifier (Op-amp) & Capacitor based integrator:

```
uController             AY-3-8913
,---------.            ,---._.---.
|         |    2 Mhz ->|CLK  SEL0|<-- 0
|    GPIOx|----------->|BC1  SEL1|<-- 0
|    GPIOx|----------->|BDIR     |
|    GPIOx|----------->|DA0      |
|    GPIOx|----------->|DA1      |
|    GPIOx|----------->|DA2      |          C1
|    GPIOx|----------->|DA3      |     ,----||----.
|    GPIOx|----------->|DA4      |     |          |
|    GPIOx|----------->|DA5      |     |  Op-amp  |
Speaker
|    GPIOx|----------->|DA6 AUDIO|     |   |X     |            /|
|    GPIOx|----------->|DA7  OUT |-----+---|-X    |   C2   .--/ |
`---------'            `---------'          |  }---+---||---|    |
                                         ,--|+/             `--` |
                                         |  |/               |  `|
                                         |                   |
                                     GND ---             GND ---
```

Separate channels through the Op-amp The third option is to externally combine 4 channels with the Operational Amplifier and low-pass filter:

```
uController             AY-3-8913
,---------.            ,---._.---.
|         |    2 Mhz ->|CLK  SEL0|<-- 0
|    GPIOx|----------->|BC1  SEL1|<-- 0
|    GPIOx|----------->|BDIR     |
|    GPIOx|----------->|DA0      |
|    GPIOx|----------->|DA1      |
|    GPIOx|----------->|DA2      |           C1
|    GPIOx|----------->|DA3      |      ,----||----.
|    GPIOx|----------->|DA4      |      |          |
|    GPIOx|----------->|DA5     A|---.  |  Op-amp  |        Speaker
|    GPIOx|----------->|DA6     B|---+  |   |X     |            /|
|    GPIOx|----------->|DA7     C|---+--+---|-X    |   C2   .--/ |
`---------'            `---------'          |  }---+---||---|    |
                                         ,--|+/             `--` |
                                         |  |/               |  `|
                                         |                   |
                                     GND ---             GND ---
```

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	DA0 - multiplexed data/ address bus LSB	audio out (PWM)	(in) BC1 bus control
1	DA1 - multiplexed data/ address bus	digita audio LSB	(in) BDIR bus direction
2	DA2 - multiplexed data/ address bus	digita audio	(in) SEL0 clock divider
3	DA3 - multiplexed data/ address bus	digita audio	(in) SEL1 clock divider
4	DA4 - multiplexed data/ address bus	digita audio	(out) channel A (PWM)
5	DA5 - multiplexed data/ address bus	digita audio	(out) channel B (PWM)
6	DA6 - multiplexed data/ address bus	digita audio	(out) channel C (PWM)
7	DA7 - multiplexed data/ address bus MSB	digita audio MSB	(out) AUDIO OUT master (PWM)

8 Bit SAR ADC

by **Eric Estevez**

0484 Analog Project

`github.com/eestevez3/ttsky_SAR_ADC`

"Convert analog signal to digital"

How it works

The goal of the project is to make an 8-bit resolution SAR ADC similar to a group project I did in college.

The design currently does not have full rail-to-rail detection without precise timing. If I can make a good feedback system for the comparator while not relying on timing restrictions, I will update the design.

Without any requirements for recalibration, the circuit detects and outputs voltages accurately for the system approximately in the range of 0.8V to VCC. The sampling circuit will hold the analog input value until the positive edge of its clock cycle which feeds its output to the comparator. The comparator outputs a value based on if the DAC output is higher or lower than the sampled input (if DAC is higher, it outputs low; if DAC is lower, it outputs high). The output is fed to the digital logic block which checks whether the comparison bit needs to change so that the value outputted by the block converges to the sampled analog signal.

How to test

Input an analog signal to the analog input pin. Because I currently don't have a clock divider, there will need to be two clock signals, one in the digital logic clock input, and another that's at least 8 times as slow to the clock pin for the sampled input. The enable signal and calibration signal can be used to tune the comparator for low input signals, but I currently do not have a good enough feedback system to track it accurately. Currently they are just used to reset the comparator signal levels. Without those signals changing according to the input level, the comparator can only output accurately above approximately 0.8V.

External hardware

Signal generator, oscilloscope

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	comparator enable	sar bit 0	sar clock
1	comparator calibration	sar bit 1	comp done
2	—	sar bit 2	—
3	—	sar bit 3	—
4	—	sar bit 4	—
5	—	sar bit 5	—
6	—	sar bit 6	—
7	—	sar bit 7	—

Analog Pins

ua#	analog#	Description
0	4	sample in

Audio Transient Shaper

by **Mark Renker**

0485 | 24 MHz | HDL Project

`github.com/mark-renker/transient_shaper`

"Real-time transient shaper: enhances/reduces attack/sustain by envelope modulation"

How it works

This transient shaper processes audio signals in real-time by independently controlling the attack (initial transient) and sustain (decay) portions of the signal using dual envelope followers.

Architecture

The design implements three main components:

1. **Fast Envelope Follower**: Tracks rapid changes in the audio signal with a time constant of 0.25 (attack detection)
2. **Slow Envelope Follower**: Tracks gradual changes with a time constant of 0.125 (sustain detection)
3. **Modulation Stage**: Applies independent gain control to attack and sustain components

Signal Flow

- 6-bit audio input is fed to both envelope followers simultaneously
- Fast envelope captures transient peaks for attack enhancement
- Slow envelope tracks sustained signal energy
- Each envelope can be boosted by 50% when enabled via control bits
- Final output combines original signal with envelope-derived gain modulation

Processing Parameters

- **Clock frequency**: 24 MHz
- **Attack time constant**: 0.25 (fast response)
- **Sustain time constant**: 0.125 (slow response)
- **Boost amount**: 50% (right-shift by 1)
- **Bit depth**: 6-bit input, 8-bit output

How to test

Input Configuration

The 8-bit input (`ui_in`) is mapped as follows:

- **Bits [5:0]**: Audio input (6-bit unsigned, 0-63 range)
- **Bit [6]**: Sustain amount (0=bypass, 1=boost sustain)
- **Bit [7]**: Attack amount (0=bypass, 1=boost attack)

Test Procedure

1. **Passthrough Test**: Set `ui_in = 0b00XXXXXX` (attack/sustain off) and verify output equals input
2. **Attack Enhancement**: Feed an impulse signal (high→low transition) with `ui_in[7]=1` to see transient boost
3. **Sustain Control**: Apply a constant signal with `ui_in[6]=1` to verify sustained level boost
4. **Combined Mode**: Enable both bits for maximum dynamic processing

Example Test Vectors

```
ui_in = 0b00100000 → Passthrough (audio=32, no boost)
ui_in = 0b10100000 → Attack boost enabled (audio=32)
ui_in = 0b01100000 → Sustain boost enabled (audio=32)
ui_in = 0b11111111 → Full boost (audio=63, both enabled)
```

Clock & Reset

- Apply 24 MHz clock to `clk`
- Assert `rst_n` low for at least 10 cycles to reset
- Set `ena` high to enable processing

External hardware

This design operates entirely in the digital domain and does not require external hardware. For practical audio applications, you would need:

- **ADC**: To convert analog audio input to 6-bit digital (e.g., parallel flash ADC)
- **DAC**: To convert 8-bit digital output back to analog audio (e.g., R-2R ladder)
- **Anti-aliasing filter**: Input low-pass filter for frequencies above 1 MHz
- **Reconstruction filter**: Output low-pass filter to smooth the digital-to-analog conversion

Optional but recommended:

- Input level control (potentiometer + voltage divider)
- Output amplifier/buffer for driving speakers or headphones

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	audio_in[0]	audio_out[0]	—

#	Input	Output	Bidirectional
1	audio_in[1]	audio_out[1]	—
2	audio_in[2]	audio_out[2]	—
3	audio_in[3]	audio_out[3]	—
4	audio_in[4]	audio_out[4]	—
5	audio_in[5]	audio_out[5]	—
6	sustain_amt	audio_out[6]	—
7	attack_amt	audio_out[7]	—

OnChip - Bandgap Reference

by **Jeison Acevedo** & **Ramon Sarmiento**

0486 Analog Project

`github.com/RamonSsc/tt_um_Onchip_BandGap`

"BandGap Voltage Reference 1.23V"

How it works

The presented circuit is a bandgap voltage reference based on the Beta amplifier topology, which incorporates an operational transconductance amplifier (OTA) — in this case, a folded cascode configuration. The typical output voltage is approximately 1.23 V

How to test

To test the circuit, both output nodes (Vg and Vref) must be measured. The first node includes a buffer implemented using a replica of the internal folded cascode, while the second node is connected directly.

External hardware

DC voltage source (VDD)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	1	vref
1	3	vsense

ua#	analog#	Description
2	2	ibias

Baud-Rate PAM4 Clock and Data Recovery (CDR)

by **Soumobrata Ghosh**

0487 | 50 MHz | HDL Project

`github.com/Soumobrata/Baud_Rate_CDR`

"A baud-rate PAM4 CDR with Mueller–Müller phase detector, PI loop filter, and digital DCO."

Author: Soumobrata Ghosh
Project type: Mixed-signal digital CDR core implemented fully in synthesizable Verilog
Clock: 50 MHz
Tile size: 1 × 1

Overview

This design implements a baud-rate PAM4 Clock-and-Data-Recovery (CDR)** architecture entirely in digital logic.
It follows a classical Mueller–Müller phase detector (MMPD) structure with a proportional-integral (PI) loop filter and a digitally controlled oscillator (DCO) that generates a symbol-rate strobe.

The core tracks the phase of a four-level (PAM-4) input data stream and aligns its internal sampling instant to incoming data transitions — a technique used in **high-speed SerDes, Ethernet, and memory interface receivers.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	DATA[0]	X[4]	—
1	DATA[1]	X[5]	—
2	DATA[2]	X[6]	—
3	DATA[3]	X[7]	—
4	DATA[4]	S[0]	—
5	DATA[5]	S[1]	—
6	DATA[6]	S[2]	—
7	DATA[7]	Sample_en	—

DreiMann

by **Benjamin Lehner**

0489 | 50 kHz | HDL Project

`github.com/b2nch/jku-sky-DreiMann`

> *"DreiMann is a game that requires two dice and can be played in a group of at least three people. It is often used as a party game. This implementation replaces the dice and displays the current move on a seven-segment display."*

How it works

DreiMann is a game that requires two dice and can be played in a group of at least three people. It is often used as a party game. This implementation replaces the dice and displays the current move on a seven-segment display.

A move consists of rolling two dice. To do this, one person must press both buttons. The result is shown on the 7-segment display.

Possible dice combinations and their display:

- Both dice show the same number (double): the number is displayed (1-6)
- "21" - one of the dice is 1 and the other is 2: all LEDs on the display light up
- Exactly one 3 - one of the dice shows 3 and the other does not: the 3 horizontal lines on the display light up
- None of the above dice combinations: only the middle line on the display lights up

Game explanation:

- At the beginning, there is no DreiMann.
- During the round, both dice are rolled in turn (both buttons pressed).
- If a double is rolled (number 1 to 6 on the display), the person who rolled the result can give the number of minus points shown to another person.
- If a "21" is rolled (all LEDs light up), each person in the round receives one minus point.
- If there is no DreiMann, rolling a 3 makes you the DreiMann (the 3 horizontal lines light up). You remain the DreiMann until you roll exactly a 3 on your turn.
- If there is already a DreiMann and another person rolls a 3, the current DreiMann receives a minus point. Minus points from a double (see above) may not be given to the DreiMann.

If a different combination of dice is rolled (middle line), nothing happens. After rolling a result, it is the next person's turn.

Have fun playing!

How to test

Connect the hardware. Get together with a few friends and try playing the game. Don't forget to have fun.

External hardware

Two buttons (pin 0 and pin 1) are required to execute the dice rolls. To execute a dice roll, the pin must be set to HIGH (1). By default, the pin should be set to LOW (0).

An additional switch can be connected to pin 2, which can be used to switch between common cathode and common anode of the 7-segment display. By default (LOW), this display is set to common anode.

For visualization, a 7-segment display must be connected to the output (pin 0 to pin 7).

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	roll_dice_1	segment_0	—
1	roll_dice_2	segment_1	—
2	switch_common_cathode_anode_7segmentdisplay	segment_2	—
3	—	segment_3	—
4	—	segment_4	—
5	—	segment_5	—
6	—	segment_6	—
7	—	segment_7	—

Time Domain ADC

by **M. Taufiqul Huda**

0490 | 1 MHz | Analog Project

github.com/mthudaa/its_10b_tadc

"Time Domain Comparator Based ADC"

Low-Power SAR-ADC for PPG Signal Acquisition

This repository contains the design files and simulation results for a 10-bit SAR-ADC in 180nm CMOS, optimized for low-power biomedical applications like Photoplethysmogram (PPG) signal acquisition.

Project Overview

The primary goal of this project is to design an efficient and accurate ADC for low-power wearable devices. The design focuses on a 10-bit monotonic SAR-ADC architecture utilizing a **time-domain comparator** to achieve power consumption in the microwatt range.

The final post-layout design is ready for fabrication, achieving a power consumption of **23.25 µW** and an Effective Number of Bits (ENOB) of **6.44 bits**. The design has been verified to accurately acquire PPG signals, making it suitable for biomedical applications.

Key Specifications

The final specifications from the post-layout simulation are as follows:

Parameter	Tipikal	Unit
Technology	CMOS 180nm	-
Supply Voltage	1.80	V
Resolution	10	bits
Input Voltage Range	0 - 1.8	V
Conversion Time	5.2	µs
Power Consumption	**23.25**	**µW**
ENOB	**6.44**	**bits**
SNR	41.03	dB
THD	−50.11	dB
INL	−3 / +3	LSB

DNL	−1 / +4	LSB

Architecture Details

The design is based on the **Monotonic Switching SAR-ADC** architecture, which is known for its power efficiency and smaller capacitor array requirements.

The core innovation lies in the **Time Domain Comparator**, which is based on a Voltage-Controlled Delay Line (VCDL). This approach significantly reduces power consumption compared to traditional dynamic comparators.

Key Components:

1. **Track/Hold (T/H) Switch**: A Bootstrapped Switch for input signal sampling.
2. **Capacitive DAC (CDAC)**: Generates the reference voltage for comparison.
3. **Time Domain Comparator**: The low-power VCDL-based comparator.
4. **SAR Logic**: Controls the successive approximation algorithm.

✍️ How to Cite

If you use this design or information from this project, please cite the original proceedings:

```
M. T. Huda, A. N. Irfansyah, and Tasripan, "Desain SAR-ADC Berbasis
Time Domain Comparator Rendah Daya Untuk Perekaman Sinyal
Photoplethysmogram Menggunakan Teknologi CMOS," Departemen Teknik
Elektro, Institut Teknologi Sepuluh Nopember, Surabaya, 2025.
```

Resources

This project is based on the research and findings presented in the paper above. For further technical details on the implementation and simulation results, please refer to the full document.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	EN	CKO	DATA[1]
1	—	DATA[8]	DATA[0]
2	—	DATA[7]	—
3	—	DATA[6]	—
4	—	DATA[5]	—

#	Input	Output	Bidirectional
5	—	DATA[4]	—
6	—	DATA[3]	—
7	—	DATA[2]	—

Analog Pins

ua#	analog#	Description
0	1	VCM
1	3	VIN
2	2	VIP

JK flip-flop

by **Eric Ulteig**

0491 | 10 kHz | Wokwi Project

`github.com/beird0/TTSKY25b_JK_flip-flop`
`wokwi.com/projects/446424200424416257`

"Basic use of logic gates"

How it works

J K Flip-Flop

How to test

The flip-flop's clock is IN1, J and K are IN0 and IN2, respectfully.

External hardware

Outputs are OUT0 and OUT1

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	J	Q	—
1	Clock	Q not	—
2	K	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

60/50Hz AC clock

by **Will Whang**

0493 | 60 Hz | HDL Project

`github.com/will127534/TinyTapeOut_SuperCon2025`

"Simple Wall Clock using Mains AC signal"

This design turns a **50/60 Hz AC tick** into a stable **HH:MM:SS** clock with optional **PPS (pulse-per-second) discipline**. It drives **six 7-segment digits statically** using a **shared `{a,b,c,d,e,f,g}` bus** and **six latch-enable strobes**—ideal for simple external latches or 7-seg driver boards.

How it works

Top-level interface (TinyTapeout-style wrapper `tt_um_willwhang`)

Inputs (`ui_in`)

- `ui_in[0]` – **PPS** (optional). A rising edge realigns the second boundary (must be ≥ 1 AC tick wide).
- `ui_in[1]` – **set_mode** (1 = set/freeze, 0 = run). In set mode, time doesn't advance and *no cascades* occur.
- `ui_in[2]` – **inc_hours** (debounced, edges only in set mode).
- `ui_in[3]` – **inc_minutes** (debounced, edges only in set mode).
- `ui_in[4]` – **inc_seconds** (debounced, edges only in set mode).
- `ui_in[5]` – **ac50_sel** (0 = 60 Hz, 1 = 50 Hz).
- `ui_in[6]` – **hour_12h** (0 = 24 h display, 1 = 12 h display, PM flag active).
- `ui_in[7]` – spare.

Clock & reset

- `clk` – **AC tick**: one clean logic pulse per mains cycle (50/60 Hz).
  > Provide a zero-cross or threshold detector that yields exactly **one** tick per cycle. If your detector produces **120 Hz** (half-cycles, e.g., H11AA1), divide by 2 before feeding `clk`.
- `rst_n` – active-low reset.

Outputs

- `uo_out[6:0]` – **shared 7-seg bus** `{a,b,c,d,e,f,g}` (no decimal points).
- `uo_out[7]` – **colon LED** (toggles each second).
- `uio_out[5:0]` – **6 latch-enable strobes** in this order: `{Ht, Ho, Mt, Mo, St, So}`
  (Hour-tens, Hour-ones, Minute-tens, Minute-ones, Second-tens, Second-ones).

- `uio_out[6]` – **PM** indicator (1 = PM when `hour_12h=1` and hour ≥ 12).
- `uio_out[7]` – **1 Hz pulse** (one `clk`-tick wide each second boundary).
- `uio_oe` – all 1's (we always drive `uio_out`).

Segment polarity defaults to **active-high** (common-cathode). The RTL has parameters to flip polarity if needed.

Timebase & discipline

- A small divider counts **AC ticks** to create **1 Hz**:
  - 60 Hz mode: counts 0..59
  - 50 Hz mode: counts 0..49
- A **combinational `sec_tick`** goes high **on the very tick** the divider hits its terminal count (or when **PPS** rises). This drives:
  - seconds/minutes/hours BCD increments,
  - colon toggle,
  - the 1 Hz pulse pin.
- **PPS alignment:** when a PPS rising edge is seen, we **reset the divider** and assert `sec_tick` immediately, aligning second boundaries to PPS.

Setting time (no cascade)

- With `set_mode=1`, the divider is frozen; `inc_*` buttons (debounced) directly wrap their field:
  - seconds 59→00 **does not** increment minutes
  - minutes 59→00 **does not** increment hours
  - hours 23→00
- With `set_mode=0`, normal cascades occur on rollovers.

12-hour display

- Internally time is 24 h. For 12 h display:
  - `00:xx` shows **`12:xx` AM**
  - `13..23` map to **`1..11` PM**
  - `uio_out[6]` (PM) = 1 when hour ≥ 12 in 12 h mode.

Static 7-seg via latches

- The design **does not multiplex** current. Instead, on **each AC tick** we: 1) drive `{a..g}` for the next digit, 2) pulse that digit's **LE** (latch-enable).
- Over six ticks (≈100 ms @ 60 Hz), all digits are refreshed—plenty fast for persistence with latches, and no digit current sharing on the FPGA.

How to test

Simulation (Icarus Verilog)

1) Put `main.v` and `tb.v` in the same directory.
2) Build & run: ```bash iverilog -o sim.vvp main.v tb.v vvp sim.vvp # Optional: gtkwave tb.vcd

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	pps input	7-seg display a	Digit 1
1	set mode	7-seg display b	Digit 2
2	increase hours	7-seg display c	Digit 3
3	increase minutes	7-seg display d	Digit 4
4	increase seconds	7-seg display e	Digit 5
5	50/60Hz selection	7-seg display f	Digit 6
6	hours in 12h display	7-seg display g	PM indicator
7	—	colon LED	PPS out

Tiny Whisper

by **Simon Dorrer**, **Jonathan Hager**, **Matthias Jung** & **Harald Pretl**

494 | 56 MHz | Analog Project

`github.com/iic-jku/tt-sky-jku-jmu-TinyWhisper`

> *"This chip implements a transmitter for the wireless long-reach protocol called WSPR (weak signal propagation reporter) in the shortwave frequency range (3 MHz to 30 MHz), containing a CORDIC, a DS-Modulator, and LO Generation (digital core), as well as an analog IQ Modulator."*

How it works

ToDo (see https://github.com/iic-jku/TinyWhisper)

How to test

ToDo (see https://github.com/iic-jku/TinyWhisper)

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

- SPI Master
- UART to USB Breakout Board
- RF Power Amplifier + Filters + Antenna

Project Pinout

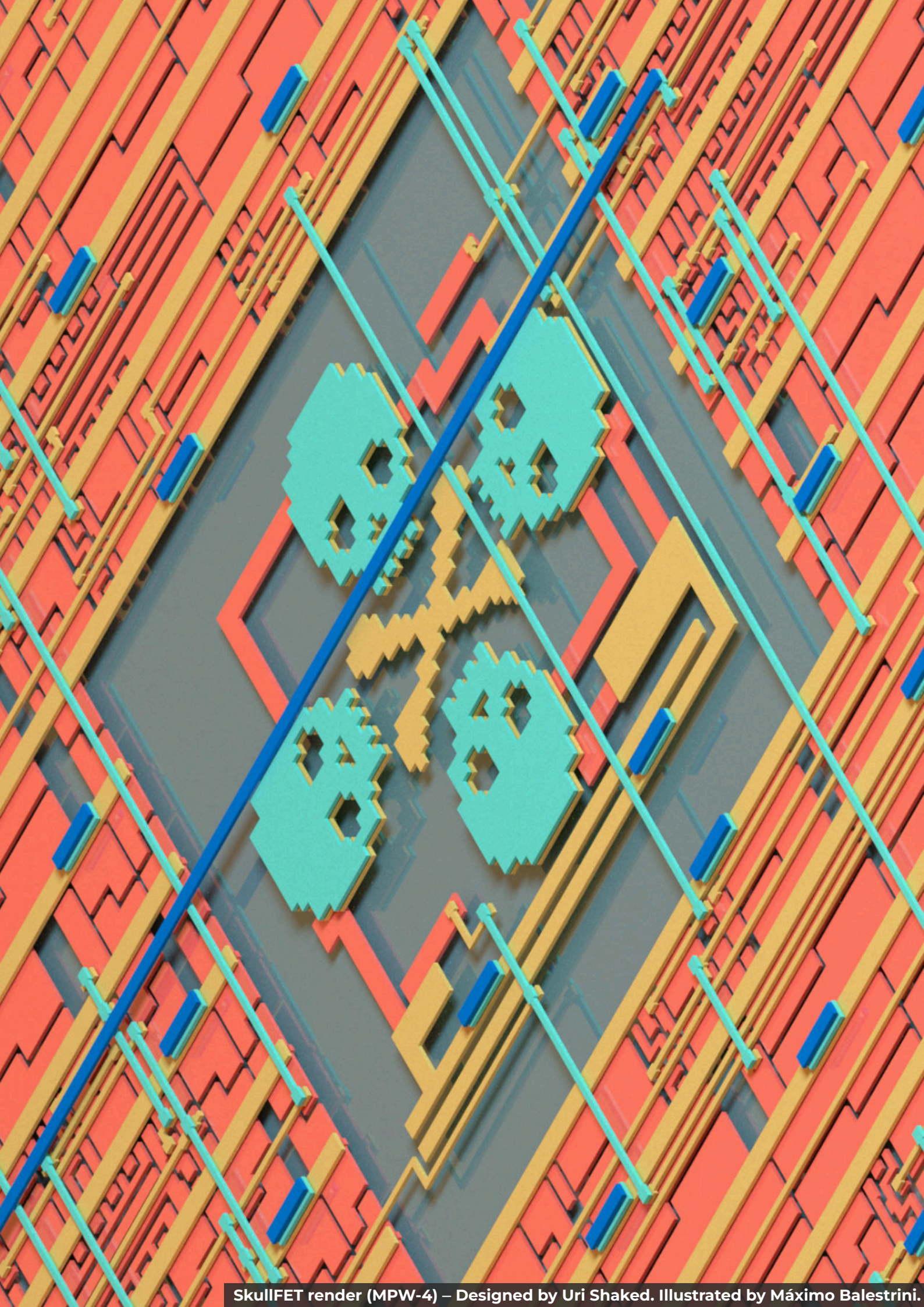
Digital Pins

#	Input	Output	Bidirectional
0	LO_QX_i	—	—
1	LO_Q_i	—	—
2	LO_IX_i	—	—
3	LO_I_i	—	—
4	afe_en_i	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	0	vout_RF

ua#	analog#	Description
1	4	vinn_Q
2	1	vinp_Q
3	3	vinn_I
4	2	vinp_I



SkullFET render (MPW-4) – Designed by Uri Shaked. Illustrated by Máximo Balestrini.

Noise shaping delta-sigma DAC

by **Toivo Henningsson**

495 | 50 MHz | HDL Project

`github.com/toivoh/ttsky25b-noise-shaping-ds-dac`

"16 bit digital noise shaping single pin output delta-sigma DAC"

How it works

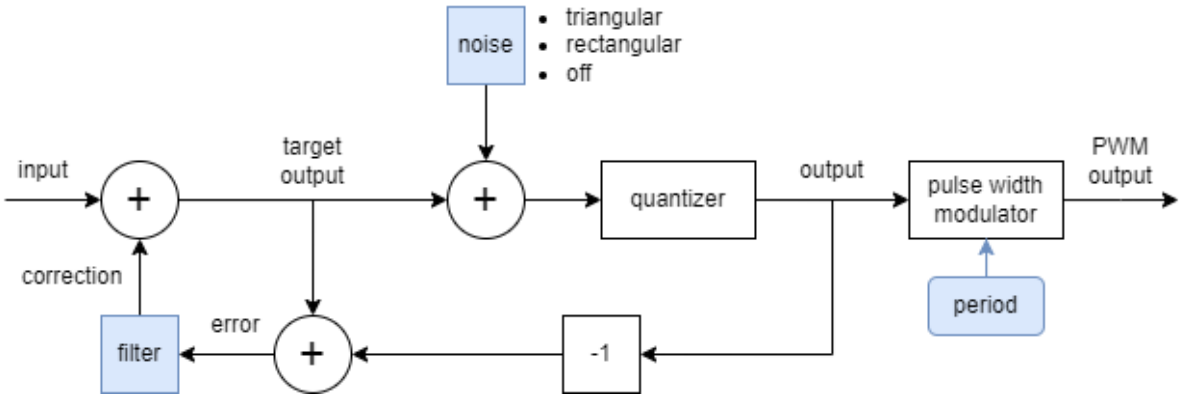


Figure 495.1: block diagram

The aim of this project is to test if it is possible to output 16 bit audio using a single digital output from a pure digital Tiny Tapeout design. For this purpose, it implements a noise-shaping delta-sigma converter that feeds into a pulse width modulator to create the digital output signal.

The design can be driven with sample data (eg using the PIO functionality in the RP2040/RP2350 microcontroller on the demo board). There is also a built-in triangle wave generator for testing without needing to feed sample data.

Operating principle

Assume that we want to output an audio signal at 48 kHz from a design that is clocked at 50 MHz, using single digital output pin fed by a flip-flop. That gives us about 1040 cycles to output each sample. (This design can go up to 66.66 MHz, but the main target is to get a working output at 50 MHz). If we output each sample by controlling the average over a 1040 cycle period, we get a little more than 10 bits of resolution. This can for instance be done with pulse width modulation (PWM).

One way to look at it is that we are taking the original unquantized output signal and adding noise to it so that it becomes a quantized signal with a resolution that can actually be output. The target can be a one bit signal sampled at 50 MHz, or a signal with higher bit depth and lower frequency output using PWM for instance (we will aim for the latter). There is a least

amount of noise we need to add to make the output signal quantized, but the noise doesn't have to be added equally across the frequency spectrum. This design uses a noise shaping sigma-delta modulator to shape the quantization noise to put most of it above the audible frequency range.

Example

Let's use a 4th order filter (the highest order that seems to be useful with a clock frequency of 50 MHz in this setup). We will consider two kinds of error:

- The *quantization error* is the difference between the target output and the actual output
- The *quantization noise* is the difference between the input signal and the actual output

(see the figure at the top.) At each sample, we will make a quantization error. Assume (for the purpose of illustration) that we make a quantization error of 1 in the first sample. The 4th order filter will add corrections to the following samples 4 samples, with values `-4, 6, -4, 1`. Taken together, the sequence of disturbances caused by the first quantization error of 1 becomes

```
1, -4, 6, -4, 1
```

where the the first 1 comes from the quantization noise in the first sample, and the rest is corrections added by the filter.

This sequence is the impulse response of a filter `(1 - z^-1)^4` with 4 zeros at `z = 1` (zero frequency), and has very little low frequency content. Adding the corrections will add more energy to the quantization noise (in this case, the variance will be 70x higher, `1^2+4^2+6^2+4^2+1^2 = 70`), but the low frequency components of the quantization noise will be greatly reduced by the corrections.

The corrections can not be applied exactly because there will be quantization errors in the following samples as well, but these errors can be corrected in the same way. The quantization error is bounded, and the bound doesn't grow due to the corrections, so the corrections will be bounded as well. If the quantizer rounds the output to the nearest integer, the magnitude of the quantization error is `<= 0.5`, and the 4th order filter adds noise of up to `+-7.5` output steps to each sample.

The output of the delta-sigma modulator is fed to a pulse width modulator with programmable period. The PWM period sets the range of the output signal. We have to make sure that the period is long enough to fit the variation of both the quantization noise and the input signal. Setting the PWM period is a tradeoff between output resolution and range on one hand, and output frequency on the other:

- If the magnitude of the of the input signal becomes too low compared to the noise, the noise will dominate and the effective resolution will suffer.
- The longer we make the PWM period, the lower the filter's sampling frequency becomes, moving more of the high frequency quantization noise into the audible range.

In this case, a period of 47 cycles may be appropriate, to allow an output swing of 0-31 from the input signal combined with 0-14 (if we add 7.5) for the noise. 45 cycles would be enough for this purpose, but since rising and falling edges may propagate at different speeds throught the TT multiplexer, we also want to avoid hitting 0% or 100% duty cycle to keep one rising and one falling edge per pulse in the output signal.

Adding additional noise to the quantization error

The default behavior is to let the quantizer calculate the output by rounding the target output to the nearest integer. The hope is that the quantization error should behave as white noise. Then, even if some of it feeds through as audible noise at (higher) audible frequencies, it is just a little added noise. Round to nearest quantization produces the smallest quantization error, but the statistical properties may be unpredictable. If the quantization error ends up including prominent frequency components that change with the input signal, it could create objectionable artifacts in some cases.

TODO: rectangular and triangular noise.

Interface

The design contains a number of 16 bit registers. To write to one,

- Make sure that `data_part_in = 1`
- Set `data_in[7:0]` to the low byte
- Set `data_part_in = 0`
- Wait at least 3 cycles for the low byte to be buffered
- Set `data_in[7:0]` to the high byte
- Set `addr[2:0]` to the desired register address
- Set `data_part_in = 1`
- Wait 3 cycles for the register to be written

Registers:

Register	Field	Name	Description
0	`reg0[15:0]`	`u`	input signal
1	`reg1[6:0]`	`period_minus_1`	—
—	`reg1[10:8]`	`u_rshift`	input shift
—	`reg1[12]`	`reset_lfsr`	for testing
—	`reg1[13]`	`force_err_0`	for testing

—	`reg1[15:14]`	`pwm_mode`	—
2	`reg2[3:2]`	`filter_choice`	—
—	`reg2[11:8]`	`n_decorrelate`	noise decorrelation
—	`reg2[15:14]`	`noise_mode`	—
3	`reg3[13:0]`	`delta_u`	for triangle generator

Other pins

- `sample_toggle_out` toggles each time `pulse_divider+1` PWM pulses have been output. This can be used to time transfers of new input signal values `u`.
- `echo_out` echoes the value of `echo_in`. This could be used to test how a signal that is routed next to `dac_out` in the TT chip influences the noise performance.

How to test

External hardware

This project needs some way to use the `dac_out` output signal, and preferably to low pass filter it to filter out the high frequency noise. Mike's audio Pmod (https://github.com/MichaelBell/tt-audio-pmod) can be used to convert the output to an audio signal for listening and includes a low pass filter to reduce frequencies above the audible range.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data_in[0]	pulse_width[0]	sample_toggle_out
1	data_in[1]	pulse_width[1]	addr_in[0]
2	data_in[2]	pulse_width[2]	addr_in[1]
3	data_in[3]	pulse_width[3]	addr_in[2]
4	data_in[4]	pulse_width[4]	data_part_in
5	data_in[5]	pulse_width[5]	echo_in
6	data_in[6]	pulse_width[6]	echo_out
7	data_in[7]	pulse_width[7]	dac_out

FemtoRV register file test

by **Mike Bell**

0514 | 50 MHz | HDL Project

`github.com/MichaelBell/ttsky25b-femtorv-soc`

"Simple FemtoRV SoC using tnt's register file"

How it works

Modified FemtoRV using tnt's register file and the QSPI controller from TinyQV.

Instructions are read using QSPI from Flash, and a QSPI PSRAM is used for memory. The QSPI clock and data lines are shared between the flash and the RAM, so only one can be accessed simultaneously.

Code can only be executed from flash. Data can be read from flash and RAM, and written to RAM.

The SoC includes a UART and an SPI controller.

Address map

Address range	Device
0x0000000 - 0x0FFFFFF	Flash
0x1000000 - 0x17FFFFF	RAM A
0x1800000 - 0x1FFFFFF	RAM B
0x8000000 - 0x8000007	GPIO
0x8000010 - 0x800001F	UART
0x8000020 - 0x8000027	SPI
0x8000028 - 0x800002B	PWM
0x8000034 - 0x800003B	GAME

GPIO

Register	Address	Description
OUT	0x8000000 (W)	Control out0-7, if the corresponding bit in SEL is high
OUT	0x8000000 (R)	Reads the current state of out0-7
IN	0x8000004 (R)	Reads the current state of in0-7

SEL	0x800000C (R/W)	Bits 0-7 enable general purpose output on the corresponding bit on out0-7. Bit 8 enables PWM output on out7, bit 9 enables PWM output on io7.

UART

Register	Address	Description
DATA	0x8000010 (W)	Transmits the byte
DATA	0x8000010 (R)	Reads any received byte
STATUS	0x8000014 (R)	Bit 0 indicates whether the UART TX is busy, bytes should not be written to the data register while this bit is set. Bit 1 indicates whether a received byte is available to be read.

Debug UART (Transmit only)

Register	Address	Description
DATA	0x8000018 (W)	Transmits the byte
STATUS	0x800001C (R)	Bit 0 indicates whether the UART TX is busy, bytes should not be written to the data register while this bit is set.

SPI

Register	Address	Description
DATA	0x8000020 (W)	Transmits the byte in bits 7-0, bit 8 is set if this is the last byte of the transaction, bit 9 controls Data/Command on out3
DATA	0x8000020 (R)	Reads the last received byte
CONFIG	0x8000024 (W)	The low 4 bits set the clock divisor for the SPI clock to 2*(value + 1), bit 8 adds half a cycle to the read latency when set
STATUS	0x8000024 (R)	Bit 0 indicates whether the SPI is busy, bytes should not be written or read from the data register while this bit is set.

PWM

Register	Address	Description
LEVEL	0x8000028 (W)	Set the PWM output level (0-255)

GAME

Register	Address	Description
Controller 1	0x80000034 (R)	Controller 1 state
Controller 2	0x80000038 (R)	Controller 2 state

Controller state is in the low 12 bits of the register, in order (MSB to LSB): b, y, select, start, up, down, left, right, a, x, l, r

How to test

Load an image into flash and then select the design.

Reset the design as follows:

- Set rst_n high and then low to ensure the design sees a falling edge of rst_n. The bidirectional IOs are all set to inputs while rst_n is low.
- Program the flash and leave flash in continuous read mode, and the PSRAMs in QPI mode
- Drive all the QSPI CS high and set SD1:SD0 to the read latency of the QSPI flash and PSRAM in cycles.
- Clock at least 8 times and stop with clock high
- Release all the QSPI lines
- Set rst_n high
- Set clock low
- Start clocking normally

Based on the observed latencies from tt06 testing, at the target 50MHz clock a read latency of 2 is likely required. The maximum supported latency is 3.

The above should all be handled by some MicroPython scripts for the RP2 on the TT demo board.

Build programs using the riscv32-unknown-elf toolchain.

External hardware

The design is intended to be used with this QSPI PMOD on the bidirectional PMOD. This has a 16MB flash and 2 8MB RAMs.

The UART is on the correct pins to be used with the hardware UART on the RP2 on the demo board.

The SPI controller is intended to make it easy to drive an ST7789 LCD display (more details to be added).

It may be useful to have buttons to use on the GPIO inputs.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	UART TX	Flash CS
1	—	UART RTS	SD0
2	SPI MISO	SPI DC	SD1
3	—	SPI MOSI	SCK
4	Game controller latch	SPI CS	SD2
5	Game controller clock	SPI SCK	SD3
6	Game controller data	Debug UART TX	RAM A CS
7	UART RX	PWM	RAM B CS / PWM

Analog 8 bit 3.3v R2R DAC

by **Matt Venn**

0516 Analog Project

`github.com/mattvenn/tt08-analog-r2r-dac-3v3`

"A simple 8 bit DAC with a sine waveform driver and 3.3v output"

How it works

A simple 8 bit R2R DAC. Driven externally or by an digitally generated sine waveform generator.

3.3v output is achieved with level shifting drivers.

How to test

Drive externally

Set the `external data` input high to provide the DAC with external data.

Then drive the 8 inputs and observe the analog output.

Drive with internal sawtooth wave generator

Set the `external data` input low to enable the sine generator. A sine wave should be seen on the analog output. Everytime the sine counter is at 0, digital output 0 should go high for one clock.

To change the frequency, set the inputs and then raise the 'load divider' input.

External hardware

A multimeter to measure the output voltage on analog pin 0.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	bit 0	count zero	external data
1	bit 1	—	load divider
2	bit 2	—	—
3	bit 3	—	—
4	bit 4	—	—
5	bit 5	—	—
6	bit 6	—	—

#	Input	Output	Bidirectional
7	bit 7	—	—

Analog Pins

ua#	analog#	Description
0	7	DAC output

TT25 3B DAC and RingO

by **Luke McCubbin**

0518 Analog Project

`github.com/LukeMcCubbin/TT_LMcCubbin_DAC_OSC`

"First Tapeout Run using Harald Pretl and Jakob Ratschenberger's tt05 analog design"

How it works

Credit to Harald Pretl and Jakob Ratschenberger, Institute for Integrated Circuits, Johannes Kepler University, Linz, Austria for the design of this circuit for TinyTapeout 5 Analog test (500kHz ring oscillator and 3-bit R-2R DAC) .

Nominal supply voltage: 1.8V

Two test outputs:

```
Clock output (rail-rail) at ca. 500kHz
DAC voltage vout=1.8V / 8 * in (0...7)
```

How to test

Apply digital bit input to input bits 0-2 and measure output with voltmeter/osciloscope

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	dac0	—	—
1	dac1	—	—
2	dac2	—	—
3	tg_dac	—	—
4	—	—	—
5	tg_ro	—	—
6	tg_short	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	11	clk
1	6	dac

3V3 Opamp and DRAM cell

by **Emilian Miron**

0520 Analog Project

`github.com/diferential/ttsky25b-analog-mix`

"Simple opamp with 3.3v power and DRAM cell"

How it works

A simple 8 bit R2R DAC. Driven externally or by an digitally generated sine waveform generator.

3.3v output is achieved with level shifting drivers.

How to test

Drive externally

Set the `external data` input high to provide the DAC with external data.

Then drive the 8 inputs and observe the analog output.

Drive with internal sawtooth wave generator

Set the `external data` input low to enable the sine generator. A sine wave should be seen on the analog output. Everytime the sine counter is at 0, digital output 0 should go high for one clock.

To change the frequency, set the inputs and then raise the 'load divider' input.

External hardware

A multimeter to measure the output voltage on analog pin 0.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Set to 0 - bias control	—	—
1	Set to 1 - bias control	—	—
2	WR	—	—
3	RD	—	—
4	WRB	—	—
5	RDB	—	—
6	—	—	—

#	Input	Output	Bidirectional
7	—	—	—

Analog Pins

ua#	analog#	Description
0	11	VOUT
1	10	IN_N in parallel with BL
2	6	IN_P in parallel with BLB

mini mosbius

by **andrew kang**

0526 Analog Project

`github.com/mattvenn/ttsky25b-minimosbius`

"a mosbius style chip in a 3x2 tt slot"

How it works

A mini MOSbius in a 3x2 tiny tapeout slot.

Contains the following blocks (x2) NMOS, programmable width (x2) PMOS, programmable width (x1) N-type differential pair, programmable tail current (x1) P-type differential pair, programmable tail current (x1) N-type 5T-OTA, programmable tail current (x2) N-type current sources (referenced to input bias current), programmable current (x2) P-type current sources (referenced to input bias current), programmable current

6 busses, 5 of which can connect to external package pins, are available to interconnect the blocks.

Each of the 6 busses contains A and B sides, which can be electrically shorted (single A/B bus) or isolated (separate A and B busses).

Contains 196 transmission gates to interconnect the blocks. Programming is done by a shift register.

Generating the programming file is WIP.

How to test

Provide a 100uA bias current into the "ibias" pin.

Toggle "rst_n" low then release to high to set shift register back to zero.

Hold "enable" low until shift register programming is done. This gates the shift register output to prevent random states until programming is done.

Clock in 196 bits to the shift register using "clk" and "dat_in". Sampled on posedge of "clk".

When programming is completed, assert "enable" high to set transmission gates.

Analog I/O is on "bus5" through "bus1" pins.

"data_out" is typically unused, except for debugging.

External hardware

Raspberry Pi or digital pattern generator to program the device.

Application dependent equipment to interface with the “bus5”-“bus1” pins.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	dat_in	dat_out	—
1	enable	—	—
2	rst_n	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Analog Pins

ua#	analog#	Description
0	11	bus1
1	7	bus2
2	6	bus3
3	10	bus4
4	9	bus5
5	8	ibias

myTTproject

by **alexlowl**

0544 | 50 kHz | HDL Project

`github.com/alexlowl/jku-sky-myTTproject`

"A Knight Rider-style LED chaser with PWM brightness control. Speed and pause are adjustable using three debounced buttons. A bargraph is used as LED bar."

How it works

A Knight Rider-style LED chaser with PWM brightness control. Speed and pause are adjustable using three debounced buttons. A bargraph is used as LED bar.

How to test

Reset. Start with pause button. LEDs start moving. Use faster/slower buttons to adjust the movement speed. Use pause button to freeze the movement, as well as to continue.

External hardware

3x push button + 1x reset button 1x bargraph or 8x LED

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	pause	LED0	—
1	faster	LED1	—
2	slower	LED2	—
3	—	LED3	—
4	—	LED4	—
5	—	LED5	—
6	—	LED6	—
7	—	LED7	—

7-Segment-Animations

by **Armin Hartl**

0545 | 10 MHz | HDL Project

`github.com/akaArmin/TT9-7-Segment-Animations`

"Many different Animations on an 7-Segment-Display"

How it works

Simplified is this project just a counter, which speed can be changed, combined with animations for a 7-segment display, which also can be switched through.

How to test

The default setting should display the numbers 0 to 9, which should change every second. First of you need to add 4 buttons at the first 4 inputpins, with which you can change the speed and animation. Test the design be by pressing the different input buttons and seeing if the speed or respectively the animation changes.

External hardware

You might need a breadboard and buttons for the controls, as well as a 7-segment display if not available.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	btn1_incAni	segment a1	—
1	btn2_decAni	segment b2	—
2	btn3_SpeedDown	segment c3	—
3	btn4_SpeedUP	segment d4	—
4	—	segment e5	—
5	—	segment f6	—
6	—	segment g7	—
7	—	—	—

Tiny Calculator

by **Andy Gong**

546 HDL Project

`github.com/AG2048/tiny-calculator-asic`

"A tiny calculator chip with built-in I/O drivers"

How it works

This project is to function as a tiny 16-bit calculator that can directly read keypad inputs from an external connected 4x4 keypad (for 0x0 to 0xF input) and operation keys (`+ - * / = AC (neg)`). The output will be decoded to be used with 5 common-cathode 7-seg display, with the support of external shift registers with serial input and shift clock (optionally with a storage register and output enable active low pin).

How to test

Connect the keypad IO pins to a physical keypad with key presses being shorts between row and col pins, and wire the operation keys as active-high buttons with pull-down resistors. Connect the output pins to a shift register which the output pins are connected to the anodes of a common-cathode 7-seg display.

Alternatively, the pins can all be connected to a microcontroller that simulates the pins' functionalities.

External hardware

- 4x4 keypad with numbers 0x0 to 0xF, with button presses being shorts between corresponding row and col pins. The keypad should have 8 wires, 4 for columns and 4 for rows.
  - Diodes are recommended to prevent shorts when multiple keys are pressed.
- 6 active-high buttons with pull-down resistors for operations: `+`, `-`, `*`, `/`, `=`, `AC`, `(neg)`.
- Shift Registers (optionally with active-low output enable pin, and storage register)
- Common-cathode 7-seg displays.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	i_BL0	o_WL0	o_ADD
1	i_BL1	o_WL1	o_SUB
2	i_BL2	o_WL2	o_MUL
3	i_BL3	o_WL3	o_DIV
4	i_ADD	o_SER	i_EQ
5	i_SUB	o_SRCLK	i_AC
6	i_MUL	o_RCLK	i_NEG
7	i_DIV	o_OE_n	i_NEG_EN

instrumented_ring_oscillator_two

by **Jeremy Mickelsen**

547 HDL Project

`github.com/AnotherPseudonym/instrumented_ring_oscillator_two`

"A ring oscillator with a selectable number of stages and initial state."

How it works

Preface: This is probably not a component you want if you want a reliable end device. This is intended to allow studying the evolution of high-frequency "modes" which are generally very undesirable.

This project uses ring oscillators with muxes on the inputs to allow setting an initial state or "seed". This can be configured using a clock (in3) and data (in2) similar to SPI (positive edge clocks the data in. The in0 line is the enable to start the oscillator running, and in1 is a HOLD line that blocks one stage so that the normal long period can be obtained. in7:in4 select the number of stages (2*n + 5). In order to have selectable stages without a really big mux (which would have a very different propagation speed than the other stages), two muxes per stage are used, some of them bypassing some of the chain to get the desired number of muxes. This diagram shows the short mux paths as pipes ("|").

Note that when less than 25 stages are used, all inverters are still driven, but some outputs are not used. Note that the seed state is a FIFO fed in at the little end - it's always updatable (though it's state should not impact operation).

How to test

0. Hook up an analyzer / scope to the output & bidirectional channels.16 phases are driven out.
1. Select the number of stages (in7:in4).
2. If desired, seed the initial state using in3 (clk), in2 (data). The input is a shift register clocked in on the rising edge.
3. Drive enable (in0) high and watch the chaos to see if it stabilizes to the longest frequency, or if high frequency modes persist.
4. The hold (in1) can be briefly driven to get to the longest frequency.

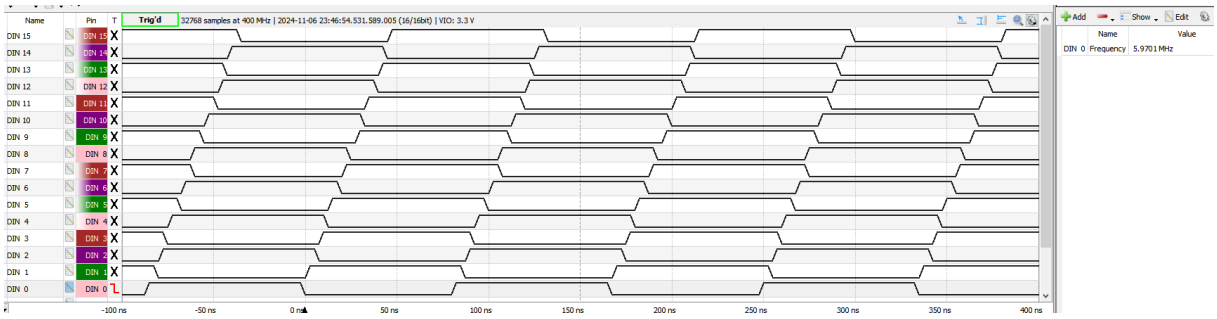


Figure 547.1: An example pattern showing only the fundamental or slowest mode.

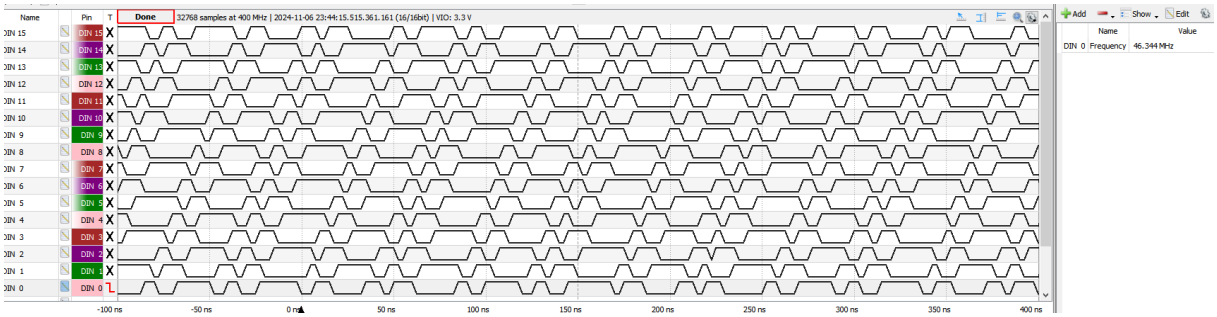


Figure 547.2: An example pattern showing higher frequency patterns or modes.

External hardware

A logic analyzer will probably be the most useful tool for this - For FPGA testing, I used a Digilent Digital Discovery (DD) with this projects outputs going to DD channels 0-15, and using DD channels 24-31 to drive the project inputs.

A multi-channel oscilloscope might also be interesting to use with this, though since the signals go through output buffers the output might not be fully representative of the RO signal.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	enable	phase[0]	phase[8]
1	hold	phase[1]	phase[9]
2	bdat	phase[2]	phase[10]
3	bclk	phase[3]	phase[11]
4	n_stages[0]	phase[4]	phase[12]
5	n_stages[1]	phase[5]	phase[13]
6	n_stages[2]	phase[6]	phase[14]

#	Input	Output	Bidirectional
7	n_stages[3]	phase[7]	phase[15]

bytebeat

by **proppy**

0548 | 8 kHz | HDL Project

`github.com/proppy/ttsky25-bytebeat`

> *"Attempt implement the formula from one of the original [bytebeat](http://countercomplex.blogspot.com/2011/10/algorithmic-symphonies-from-one-line-of.html) [video](https://www.youtube.com/watch?v=tCRPUv8V22o) in hardware."*

How it works

The main module accept parameters from 4x 4-bit parameters buses and generate PCM samples according to the following formula: `((t*a)&(t>>b))|((t*c)&(t>>d))`.

Derivative of this project can easily be created by editing the formula in `src/bytebeat.x` and using the XLS: Accelerated HW Synthesis toolkit to regenerate the Verilog code.

How to test

- Tweak parameters pins using a absolute encoders
- Feed the data coming from the sample bus to a DAC

External hardware

- 8-bit parallel DAC (ex: AD558)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	param a bit 0/3	pcm sample bit 0/7	param c bit 0/3
1	param a bit 1/3	pcm sample bit 1/7	param c bit 1/3
2	param a bit 2/3	pcm sample bit 2/7	param c bit 2/3
3	param a bit 3/3	pcm sample bit 3/7	param c bit 3/3
4	param b bit 0/3	pcm sample bit 4/7	param d bit 0/3
5	param b bit 1/3	pcm sample bit 5/7	param d bit 1/3
6	param b bit 2/3	pcm sample bit 6/7	param d bit 2/3
7	param b bit 3/3	pcm sample bit 7/7	param d bit 3/3

Flip the Flop

by **Eric Dynowski**

0549 | 10 kHz | Wokwi Project

`github.com/brokebit/tinytapeout-supercon`

`wokwi.com/projects/446364830841352193`

"128 Daisy Chained D Flip Flops"

How it works

We use magic mixed with stuff

How to test

Do all the right things in the right places and the right time

External hardware

You need two encabultors. One connected to the input and one on to process the output.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	FlipFlop_Input	FlipFlop_Output	—
1	SW	—	—
2	SW	—	—
3	SW	—	—
4	Sw	—	—
5	Sw	—	—
6	Sw	—	—
7	Sw	—	—

6bit ALU

by **Jaipriya Narahari**

0550 | 100 MHz | HDL Project

`github.com/jaipriyanh/Alu_6bit_16op`

"ALU is 6-bit,synchronous with 16 bit opcode"

Project Title: 6-bit Signed ALU with TinyTapeout Wrapper Institution: FH Kärnten Course: Digital-1 Program: Integrated Systems & Circuit Design

How it works

This design is a 6-bit signed (two's-complement) ALU. Inputs (A, B, opcode) are sampled on a rising clock edge; RESULT/ZERO/OVF are registered and appear on the next rising edge (1-cycle latency). Signed input operand range: −32…+31. Reset: rst_n is active-low, asynchronous. While low, the ALU holds RESULT=0, ZERO=1, OVF=0. Enable: ena is unused in this design (wrapper passes it through; it does not gate logic).

ALU core: For each clock cycle, one of 16 operations is selected by opcode. ZERO flag is 1 when the computed result equals 0. OVF is a signed overflow flag (asserted for ADD/SUB/NEG/INC/DEC under two's-complement rules). It is 0 for logic, shifts, MUL, DIV (Signed integer division, if B = 0, the result is 0 (ZERO=1, OVF=0)).

Control truth table (opcode –> operation)

Opcode	Operation	Result	Notes
`0000`	PASSA	`A`	—
`0001`	ADD	`A + B`	`OVF` on signed overflow
`0010`	SUB	`A - B`	`OVF` on signed overflow
`0011`	MUL	`(A * B) mod 64`	truncate to 6 bits, `OVF=0`
`0100`	DIV	`A / B`	if `B=0` → `0`, `ZERO=1`, `OVF=0`
`0101`	AND	`A and B`	bitwise
`0110`	OR	`A or B`	bitwise
`0111`	XOR	`A xor B`	bitwise
`1000`	LSL1	`A << 1`	logical left shift
`1001`	ASR1	`A >> 1`	arithmetic right shift (keeps sign)
`1010`	NOT	`not A`	bitwise
`1011`	NEG	`-A`	`OVF` when `A = -32`

`1100`	INC	`A + 1`	`OVF` when `A = +31` (wraps to −32)
`1101`	DEC	`A − 1`	`OVF` when `A = −32` (wraps to +31)
`1110`	SLT	`1` if `A < B` else `0`	signed compare
`1111`	XNOR	`not (A xor B)`	bitwise

Mapping of ports: Input port: ui_in[5:0]=A, ui_in[7:6]=opcode[1:0],

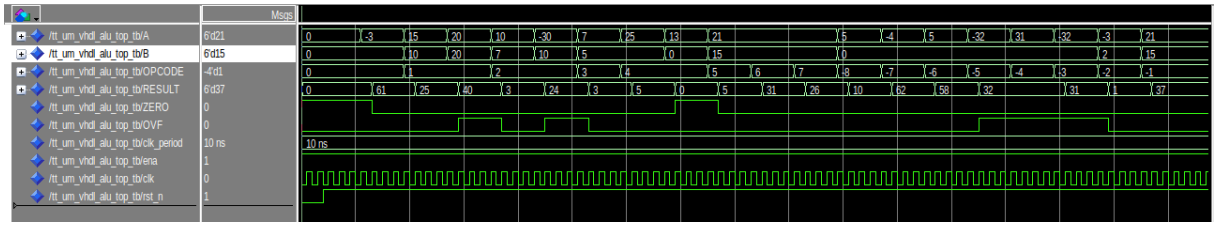
Bidirectional port : uio_in[7:2]=B, uio_in[1:0]=opcode[3:2]

Output port : uo_out[7:2]=RES[5:0] uo_out[0]=OVF, uo_out[1]=ZERO

How to test

To test the design, expect the following results:

ModelSim simulatuion results:



| # | Op | Opcode (dec) | A | B | Expected RESULT | ZERO | OVF | | :-: | :—- | :————-: | :-: | :-: | :————-: | :–: | :-: | | 1 | PASSA | 0 | −3 | 0 | −3 | 0 | 0 | | 2 | ADD | 1 | 15 | 10 | 25 | 0 | 0 | | 3 | ADD | 1 | 20 | 20 | −24 | 0 | 1 | | 4 | SUB | 2 | 10 | 7 | 3 | 0 | 0 | | 5 | SUB | 2 | −30 | 10 | 24 | 0 | 1 | | 6 | MUL | 3 | 7 | 5 | −29 | 0 | 0 | | 7 | DIV | 4 | 25 | 5 | 5 | 0 | 0 | | 8 | DIV | 4 | 13 | 0 | 0 | 1 | 0 | | 9 | AND | 5 | 21 | 15 | 5 | 0 | 0 | | 10 | OR | 6 | 21 | 15 | 31 | 0 | 0 | | 11 | XOR | 7 | 21 | 15 | 26 | 0 | 0 | | 12 | LSL1 | 8 | 5 | 0 | 10 | 0 | 0 | | 13 | ASR1 | 9 | −4 | 0 | −2 | 0 | 0 | | 14 | NOT | 10 | 5 | 0 | −6 | 0 | 0 | | 15 | NEG | 11 | −32 | 0 | −32 | 0 | 1 | | 16 | INC | 12 | 31 | 0 | −32 | 0 | 1 | | 17 | DEC | 13 | −32 | 0 | 31 | 0 | 1 | | 18 | SLT | 14 | −3 | 2 | 1 | 0 | 0 | | 19 | XNOR | 15 | 21 | 15 | −27 | 0 | 0 |

External hardware

clock source. Reset button for rst_n (active-low)

Inputs as switches (16 total)

2× 8-switch:

UI bank → ui[5:0] = A[5:0], ui[7:6] = opcode[1:0] UIO bank → uio[7:2] = B[5:0], uio[1:0] = opcode[3:2]

Outputs as LEDs (8 total)

uo[7:2] → RESULT[5:0], uo[1] → ZERO, uo[0] → OVF

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	A0 (op1[0])	OVF (signed overflow flag)	OP3 (opcode[3])
1	A1 (op1[1])	ZERO (result==0)	OP2 (opcode[2])
2	A2 (op1[2])	RES0 (result[0])	B0 (op2[0])
3	A3 (op1[3])	RES1 (result[1])	B1 (op2[1])
4	A4 (op1[4])	RES2 (result[2])	B2 (op2[2])
5	A5 (op1[5])	RES3 (result[3])	B3 (op2[3])
6	OP1 (opcode[1])	RES4 (result[4])	B4 (op2[4])
7	OP0 (opcode[0])	RES5 (result[5])	B5 (op2[5])

4-bit CLA

by **Wei Zhang**

0551 HDL Project

github.com/Electom/SKY25b_CLA_4bits

"A 4 bit carry look-ahead adder"

How it works

This is a 4-bit CLA. It can be used to construct the adder with higher bits.

How to test

The design has 3 input ports: a, b and ci It has 2 output ports: s and co a: an addend. b: the other addend. ci: the carry signal for the input. s: the output sum. co: the carry signal for the output.

This project was tested by an U250 FPGA.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any No external hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a[0]	s[0]	ci
1	a[1]	s[1]	—
2	a[2]	s[2]	—
3	a[3]	s[3]	—
4	b[0]	co	—
5	b[1]	—	—
6	b[2]	—	—
7	b[3]	—	—

FSM

by **G.kahlouche**

0552 | 100 MHz | HDL Project

`github.com/GhilasIC/Digital1.1`

"dd"

How it works

It's a Moore FSM with spaced change/refund pulses and a small datapath for credit/selection.

States: st_select → st_vend → (st_mc_pulse ↔ st_mc_gap)* → st_select st_select → (st_refund_pulse ↔ st_refund_gap)* → st_select

Inputs

- coin (1 bit): rising-edge–counted +1 credit per pulse (capped at 7).
- btn (4 bits): user command sampled on the transition "0000" → non-zero 0000 → wait (no command) 0001 → cancel (clear selection, keep credit) 0010 → refund (return all credit as spaced pulses) 0011..1111 → product codes for items 1..13 (internal idx 0..12)

Selection & Pricing

Prices depend on the selected index (sel_idx): idx 0..3 → price 1, idx 4..7 → price 2, idx 8..12 → price 3 When a valid selection exists and credit ≥ price, the FSM vends: Credit is debited by price in st_vend. If credit remains, it enters the make-change sequence (spaced pulses). Otherwise, it returns to st_select.

Refund vs. Make-change

Refund (btn="0010") returns all current credit as pulses with 1-cycle gaps. Make-change happens after a vend if debit left credit > 0; same pulse/gap pattern.

Outputs (Moore)

- dispense = '1' only while in dispense_product (exactly one clock cycle).
- change = '1' only while in return_change (exactly one clock cycle).
- product_num (4 bits) latches the 1..13 product number on entry to st_vend, stays stable until the next coin or button press (or reset). This aligns naturally with the dispense_product strobe.

Timing/Reset reset is asynchronous, active-high: immediately sends the FSM to idle. State updates on the rising edge of clk. Inputs are sampled synchro-

nously. The “one-cycle pulse” behavior comes from those terminal states automatically returning to idle on the next clock.

How to test

Scenario 1 —
Input: 1 coin (coin 1-cycle pulse), select btn in 0011

Expect: dispense_product=‘1’ for 1 cycle, product_num= (10)10 = (1010)2 “until next scenario/btn press” and change=‘0’.

Scenario 2 —
Input : 2 coins (2 pulses each coin for 1-cycle pulse), select btn in 0111

Expect: dispense_product=‘1’ for 1 cycle, product_num= (5)10 = (101)2 “until next scenario/btn press” and change=‘0’.

Scenario 3 —
Input : 2 coins (2 pulses each coin for 1-cycle pulse), select btn in 1011

Expect: dispense_product=‘1’ for 1 cycle, (11)10 = (1011)2 “until next scenario/btn press” and change=‘0’.

Scenario 4 —
Input : 3 coins (3 pulses, each coin for 1-cycle pulse), select btn in 0011 (price product = 1 coin)

Expect: dispense_product=‘1’ for 1 cycle, product_num = (1)10 = (0001)2 “until next scenario/btn press” and change=‘1’ then ‘0’ then ‘1’ (2 pulses for 2 coins) .

Scenario 5 — Refund
Input : 3 coins (3 pulses, each coin for 1-cycle pulse), select btn in 0010 (to return change)

Expect: dispense_product=‘0’, product_num = (0)10 = (0000)2 “until next scenario/btn press” and change=‘1’ then ‘0’ then ‘1’ then ‘0’ then ‘1’ (3 pulses for 3 coins) .

Scenario 6 —
Input : 2 coins (2 pulses, each coin for 1-cycle pulse), select btn in 1111 (product price= 3 coins) ##### Expect: Nothing happens, dispense_product = product_num = change = 0 Then press btn = ‘0001’ to cancel the transaction, and then press btn = 0100 (product price = 1) ##### Expect: dispense_product=‘1’, product_num = (2)10 = (0010)2 ” and change=‘1’ then ‘0’ (1 pulse for 1 coin) .

External hardware

Inputs:

- coin → 1x pushbutton or DIP (debounced)
- btn(3:0) → 4x switches to issue 0001 (cancel), 0010 (refund), and 0011..1111 (products)

Outputs:

- 1x LED for dispense_product (short flash on vend)
- 1x LED for change
- 4x LEDs for product_num (optional)

Clock & reset: board oscillator + synchronous active-high reset generation.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	clk	dispense	—
1	reset	change	—
2	coin	product_num(0)	—
3	btn (0)	product_num(1)	—
4	btn (1)	product_num(2)	—
5	btn (2)	product_num(3)	—
6	btn (3)	—	—
7	—	—	—

Tiny Pong

by **Andrew Chen**

0553 | 25 MHz | HDL Project

`github.com/AxC1271/TinyPong`

"VGA Display Driver for Pong Game"

How It Works

The `TinyPong` chip is an ASIC meant to run a simple single-player Pong game on a 640x480 Hz. This is meant to be a fun and cool demo project for the Hacker Fab club at Case Western Reserve University.

How To Test

We're aiming to have the chip have a VGA interface so you can plug that VGA port into any monitor to see the Pong game happen in action. For extra safety, assert the `rst_n` pin low temporarily to "start" the chip.

External Hardware

Some external hardware to test this chip would be a USB-C cable (and a laptop to program the chip), VGA port for interfacing, and a 640x480 display to see the pong game work.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Paddle Up Button	VGA HSYNC	—
1	Paddle Down Button	VGA VSYNC	—
2	—	VGA RED bit 0	—
3	—	VGA RED bit 1	—
4	—	VGA GREEN bit 0	—
5	—	VGA GREEN bit 1	—
6	—	VGA BLUE bit 0	—
7	—	VGA BLUE bit 1	—

PWM Generator

by **Shyam Sunder Mandapally**

0554 | 10 MHz | HDL Project

`github.com/mshyam001/pwm_with_demux_logic`

"PWM Generator with demux logic"

10-bit PWM with Demux Logic

Institution: FH Kärnten
Course: Digital-1
Program: Integrated Systems & Circuit Design

How it works

This design is a **10-bit PWM generator** with a **compact configuration demux** and **simultaneously update** mechanism. **PWM core**
A 10-bit up-counter runs from 0 to reload_active.
On the clock where cnt = set_active, the output goes **high**.
On the clock where cnt = clr_active, the output goes **low**.
When cnt = reload_active, the counter wraps to 0.

Control truth table

- sel="00" + wr=1 → write **SET** shadow with data_i
- sel="01" + wr=1 → write **CLEAR** shadow with data_i
- sel="10" + wr=1 → write **RELOAD** shadow with data_i
- commit=1 → copy all shadows to actives in the pwm_module

Frequency and duty (plain text formulas):

- PWM frequency = f_clk / (reload + 1)
- High time (for set < clr) = (clr - set) / (reload + 1) of the period
- If set > clr, the high window wraps around the end of the period.

  Config demux + double buffering
  Three shadow registers (set_shadow, clr_shadow, reload_shadow) are written via a single shared 10-bit bus:

  - sel = "00" → write **SET**
  - sel = "01" → write **CLEAR**
  - sel = "10" → write **RELOAD**
  - wr = 1 stores data_i into the selected **shadow**.
  - commit = 1 copies **all shadows** into the **active** registers (*_active) in the same clock edge.

(Same-cycle wr=1 and commit=1 is supported: the just-written value is applied.)

Reset/enable
Internally, reset is gated with ena (res_ni_g <= rst_n and ena), so the design stays in reset while the tile isn't enabled.

TinyTapeout pin map (10-bit data path)
pin map (10-bit data path)

- ui_in[7:0] → data_i[7:0] (LSBs)
- uio_in[5:4] → data_i[9:8] (MSBs)
- uio_in[1:0] → sel (00=SET, 01=CLEAR, 10=RELOAD, 11 ignored)
- uio_in[2] → wr (1-cycle strobe)
- uio_in[3] → commit (1-cycle strobe)
- uo_out[0] ← pwm_o (active-high PWM). Other uo_out bits are 0.
- uio_out is driven 0 and uio_oe is 0 (uio pins used as inputs only). > Note: reload must be **≥ 1**. If reload = 0, the counter would stall (the RTL asserts a sim-time warning).

How to test

1. **Bring the core out of reset**
   - Hold ena = 1 and rst_n = 1. Keep uio_in[3:0] low until you're ready to program.
2. **Program the three fields (example for 9.77 kHz @ 10 MHz clk, 50% duty)**
   - Target values: reload = 1023, set = 0, clear = 512.
   - Data bus wiring: data_i[9:8] = uio_in[5:4], data_i[7:0] = ui_in[7:0].

   Write RELOAD
   - Put data_i = 1023 (0x3FF): set uio_in[5:4]="11", ui_in="11111111".
   - Set sel="10" → uio_in[1:0]="10".
   - Pulse wr: uio_in[2]=1 for one clock, then back to 0.

   Write SET
   - Put data_i = 0: uio_in[5:4]="00", ui_in="00000000".
   - sel="00", pulse wr for one clock.

   Write CLEAR
   - Put data_i = 512 (0x200): uio_in[5:4]="10", ui_in="00000000".
   - sel="01", pulse wr for one clock.

   Commit Simultaneously
   - Pulse commit: uio_in[3]=1 for one clock.
3. **Observe the output**
   - uo_out[0] toggles with period 1024 / fclk and 50% duty in this example.

4. **Inputs type for different Requirements**
   - Increase duty: raise clear (for set=0 case).
   - Shift phase: move set away from 0.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data_i[0] (LSB)	pwm_o (active-high PWM)	sel[0]
1	data_i[1]	—	sel[1]
2	data_i[2]	—	wr (strobe)
3	data_i[3]	—	commit (strobe)
4	data_i[4]	—	data_i[8]
5	data_i[5]	—	data_i[9] (MSB)
6	data_i[6]	—	—
7	data_i[7]	—	—

Negative Nature #4

by **bleeptrack**

0555 HDL Project

`github.com/bleeptrack/negative-nature04`

"generative voronoi pattern for negative space in silicon"

How it works

Silicon art on met4.

How to test

No test, just looks :)

External hardware

Nothing needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	dummy	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

lime_cic_filter

by **E Rijnbeek**

556 | 32 MHz | HDL Project

`github.com/erijnbeek/lime_cic_filter`

"Cascaded Integrator-Comb (CIC) Filter"

How it works

Disclaimer, this "How it works" section is generated by AI based on the code comments, and manually checked.

This project implements two CIC (Cascaded Integrator-Comb) filters designed for processing sigma-delta modulator bitstreams. The design features both first-order and second-order CIC filters with comprehensive debugging capabilities.

Normal Operation

The CIC filter takes a sigma-delta modulator bitstream as input and decimates it to produce a lower sample rate digital output. The module includes two different CIC implementations:

1. **First Order CIC Filter**: Uses a 10-bit register width with decimation factor of 10
2. **Second Order CIC Filter**: Uses a 20-bit register width with decimation factor of 10

Input Signals (ui_in)

- **ui_in[0]**: Modulator data input - the sigma-delta bitstream to be filtered
- **ui_in[1]**: CIC filter select - selects between first-order (0) and second-order (1) CIC filters
- **ui_in[2]**: UIO input mode for testing - when high, allows uio pins to be used as inputs for debugging
- **ui_in[3]**: Reserved for future use
- **ui_in[7:4]**: Debug select - 4-bit selector providing 16 different debug modes per filter

Output Signals

- **uo_out[7:0]**: Dedicated outputs - provides filtered data or debug information based on debug select
- **uio_out[7:0]**: Bidirectional IOs used as outputs - provides additional debug information
- **uio_oe[7:0]**: IO enable signals - controls direction of uio pins (normally all outputs except in a specific test mode)

Debug System Overview

The debug system is intelligently split based on the CIC filter selection (ui_in[1]):

First Order CIC Debug Modes (ui_in[1] = 0)

When the first-order CIC is selected, the debug modes focus on its internal signals:

- **0x0**: CIC1 output data - MSB on uo_out, LSB + clock on uio_out
- **0x1**: CIC1 integrator output - MSB on uo_out, LSB + clock on uio_out
- **0x2**: CIC1 comb output - MSB on uo_out, LSB + clock on uio_out
- **0x3**: Input monitoring - displays ui_in values on both outputs
- **0x4**: IO monitoring - displays uio_in on uo_out when in input test mode
- **0x5**: Status and control signals - shows rst_n, clk, ena, and cic_output_clk
- **0x8**: CIC1 output data LSB - focuses on least significant bits with clock
- **0x9**: CIC1 integrator LSB - integrator LSB with clock
- **0xA**: CIC1 comb LSB - comb LSB with clock
- **0xB**: CIC1 decimation counter - MSBs on uo_out, LSBs + clock on uio_out
- **0x6-0x7, 0xC-0xE**: Reserved for future CIC1 debug options
- **0xF**: Reserved (0x4B on uo_out, 0x00 on uio_out)

Second Order CIC Debug Modes (ui_in[1] = 1)

When the second-order CIC is selected, the debug modes provide access to its more complex internal state:

- **0x0**: CIC2 output data MSB - top 8 bits on uo_out, middle bits + clock on uio_out
- **0x1**: CIC2 output data LSB - bits [14:7] on uo_out, bits [6:0] + clock on uio_out
- **0x2**: CIC2 integrator1 MSB - first integrator stage MSB + middle bits
- **0x3**: CIC2 integrator1 LSB - first integrator stage LSB
- **0x4**: CIC2 integrator2 MSB - second integrator stage MSB + middle bits
- **0x5**: CIC2 integrator2 LSB - second integrator stage LSB
- **0x6**: CIC2 comb1 MSB - first comb stage MSB + middle bits
- **0x7**: CIC2 comb1 LSB - first comb stage LSB
- **0x8**: CIC2 comb2 MSB - second comb stage MSB + middle bits
- **0x9**: CIC2 comb2 LSB - second comb stage LSB
- **0xB**: CIC2 decimation counter - MSBs on uo_out, LSBs + clock on uio_out
- **0xA**: Reserved for future CIC2 debug options
- **0xC**: Reserved (0x4C on uo_out, 0x00 on uio_out)
- **0xD**: Reserved (0x69 on uo_out, 0x00 on uio_out)
- **0xE**: Reserved (0x6D on uo_out, 0x00 on uio_out)
- **0xF**: Reserved (0x65 on uo_out, 0x00 on uio_out)

Filter Architecture

Both CIC filters implement the classic integrator-comb architecture:

- **Integrator stages** run at the high input sample rate and accumulate the input signal
- **Decimation** reduces the sample rate by the decimation factor (10)
- **Comb stages** run at the decimated rate and implement the differentiator function

The first-order CIC has one integrator and one comb stage, while the second-order CIC has two of each, providing better filtering characteristics at the cost of increased bit width requirements.

The filtered output represents the average value of the sigma-delta bitstream over the decimation period, effectively converting the 1-bit high-rate stream into a multi-bit lower-rate digital signal.

How to test

This section I will provide a test plan.

Test 1: Smoke test, bringup

- set 'hF0 on the input, 'h4b will appear on the output

Test 2: Clipping test.

- set 'h01 on the input and let the clock running with reset released
- uo_out will become 'hFF, two MSB's of uoi_out will become 1
- set 'h00 on the input and let the clock running with reset released
- all bits mentioned above will become 0.

Test 3: Static input.

- put a PWM on the input, of which the frequency is higher than f_clk/1024, but lower than f_clk.
- Output will take value between min and max proportional to the duty cycle.

Test 4: Sinewave input.

TBD

External hardware

No external hardware needed.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	MOD_IN	OUT_MSB0	OUT_LSB0
1	CIC_SEL	OUT_MSB1	OUT_LSB1
2	UIO_TEST	OUT_MSB2	OUT_LSB2
3	RESERVED	OUT_MSB3	OUT_LSB3
4	DEBUG_SEL0	OUT_MSB4	OUT_LSB4
5	DEBUG_SEL1	OUT_MSB5	OUT_LSB5
6	DEBUG_SEL2	OUT_MSB6	OUT_LSB6
7	DEBUG_SEL3	OUT_MSB7	OUT_LSB7

Negative Nature #3

by **bleeptrack**

0557 HDL Project

`github.com/bleeptrack/negative-nature03`

"generative voronoi pattern for negative space in silicon"

How it works

Silicon art on met4.

How to test

No test, just looks :)

External hardware

Nothing needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	dummy	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

tinytapeout test gates

by **gavin m**

558 Wokwi Project

`github.com/PandaInPajamas/tinytapeoutexample`

`wokwi.com/projects/444123325639097345`

"some logic gates for testing"

How it works

inputs 0 and 1 are connected to the first and gate going to output 0. inputs 2 and 3 are connected to the second and gate going to output 1. inputs 4 and 5 are connected to the first nand gate going to output 2. inputs 6 and 7 are connected to the second nand gate going to output 3.

How to test

Input A	Input B	AND Output	NAND Output
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

External hardware

none

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and 1	—
1	input b	output and 2	—
2	input c	output nand 1	—
3	input d	output nand 2	—
4	input e	—	—
5	input f	—	—
6	input g	—	—
7	input h	—	—

Negative Nature #2

by **bleeptrack**

0559 HDL Project

`github.com/bleeptrack/negative-nature02`

"generative voronoi pattern for negative space in silicon"

How it works

Silicon art on met4.

How to test

No test, just looks :)

External hardware

Nothing needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	dummy	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

TTSKY25B Register File Validation

by **Sylvain Munaut**

0576 HDL Project

github.com/smunaut/ttsky25b-rf-validation

"Validation design for register file macro"

The register file

The register file macro being tested is an optimized SRAM macro organized as 32 words of 32 bits and offers three access ports : One write port and two read ports.

It's been designed for TinyTapeout especially. It's compact enough (88% of the area of a single tile) and doesn't use any DRC waiver/special rules.

The interface looks like this :

```
module rf_top (
    input  wire [31:0] w_data,
    input  wire  [4:0] w_addr,
    input  wire        w_ena,
    input  wire  [4:0] ra_addr,
    input  wire  [4:0] rb_addr,
    output reg  [31:0] ra_data,
    output reg  [31:0] rb_data,
    input  wire        clk
);
```

The internal working of the macro are out-of-scope for this document.

The validation design

This particular documentation focuses on this specific design which is meant to validate the register file macro, both from a functionality stand point but also its timings.

This is the high-level overview of the design :

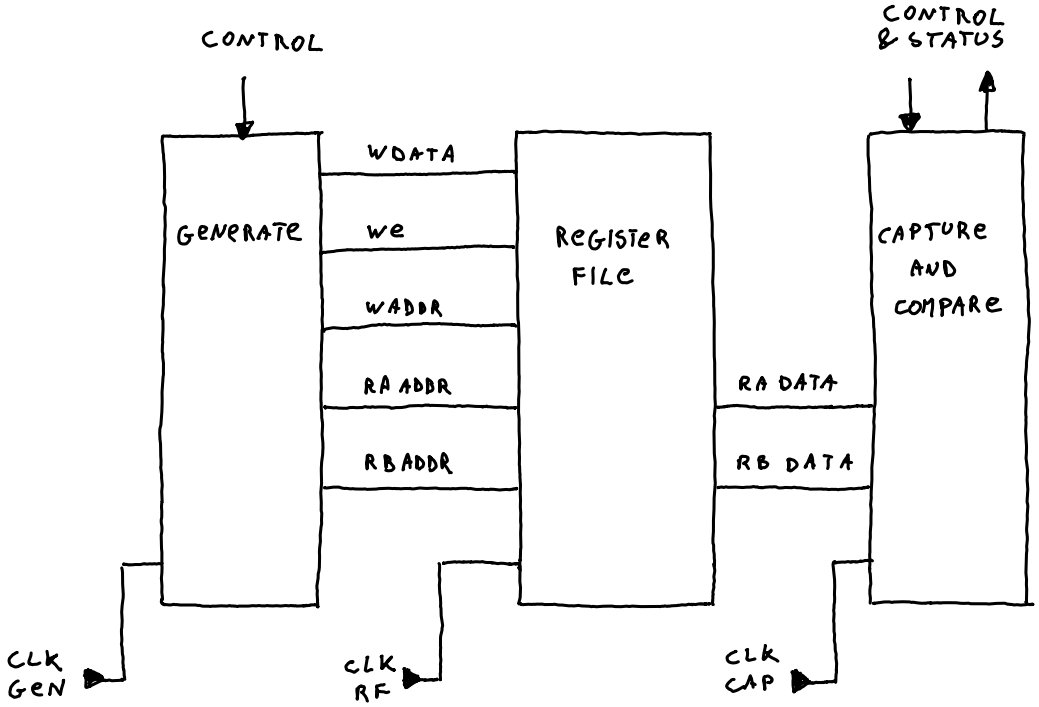


Figure 576.1: Validation design overview

The main blocks are :

- A generator block that controls all the inputs to the register file, feeding all the read/write addresses and write data.
- The register file itself.
- A capture and compare block that captures all the outputs of the register file and can compare them either to a known value or against a LFSR sequence.

Each of those blocks is clocked independently which should allow to test the various timings (setup/hold/clock-to-out/retain) by phase shifting them against each other.

In addition, there is a small “feedback” block that can select 4 timing critical signals and feed them back to an output through a common path. This should allow to calibrate out any skew in the input path of those signals to make more precise timing measurements.

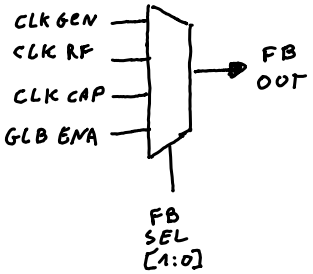


Figure 576.2: Feedback mux

Generator

The high level overview of the generator block is show below :

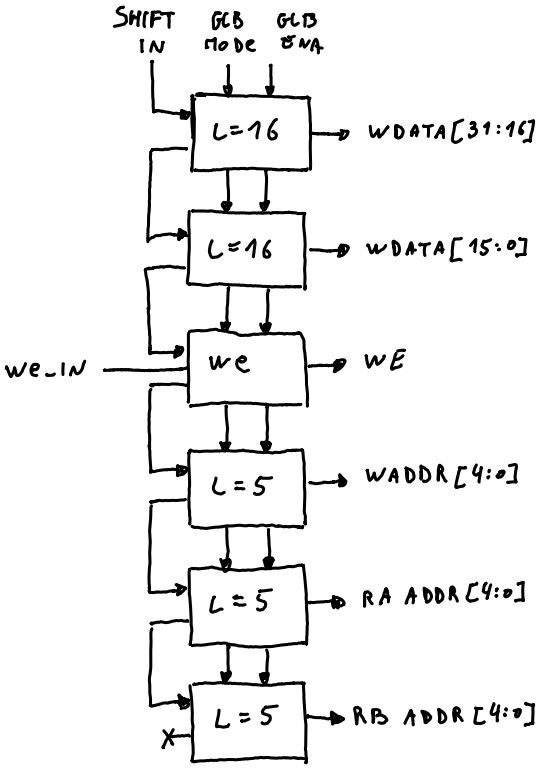


Figure 576.3: Generate block: Overview

It's composed of 6 blocks, each with an internal shift register chain. They are chained together in a single chain, as shown, new data being fed to the MSB first.

There are two types of block :

- One "common" type of variable length to generate write data and read/write addresses.
- One special type to generate the write enable signal specifically.

The common block internally looks like this :

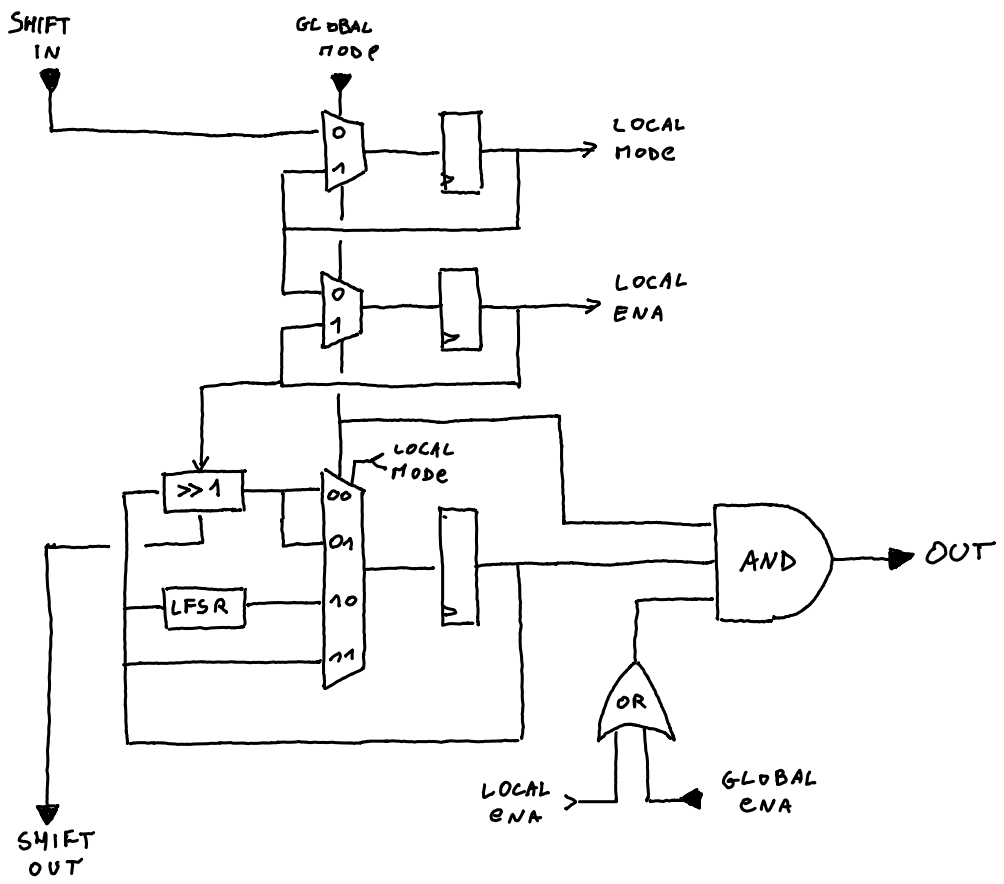


Figure 576.4: Generate block: Group diagram

It first starts with two independent bit elements in the chain, and then follows a multi bit element that can be either 5 or 16 bits long.

When the `global_mode` signal is low, the whole block acts as a shift register which allows to load arbitrary values anywhere in the chain. Note that when the signal is low, the output of the block is also masked and forced to zero. This avoid having the register file see randomly changing inputs while loading new values.

When the `global_mode` signal is high then the two first bits don't change any more and they instead define two control signals local to this specific block : `local_mode` and `local_ena`.

The first one `local_mode` defines how the block acts when clocked. Either the value stays the same across clock cycle, remaining whatever was loaded serially. Or the values goes through a LFSR. The polynomial is either `0xD008` or `0x14` depending on the length (both being maximum length). Note that like any LFSR if you pre-loaded zero, the value will remain zero. And that if you pre-loaded any other value, then zero will never be generated.

The second control bit `local_ena` can force this group to be unmasked even if the global masking signal `global_ena` is not active. This can be used to independently test the setup/hold times requirements of the data/addresses,

forcing all the groups to be active except the one under test, and then using `global_ena` to unmask the group under test slightly before/after clock edge.

The WE signal is generate a bit differently :

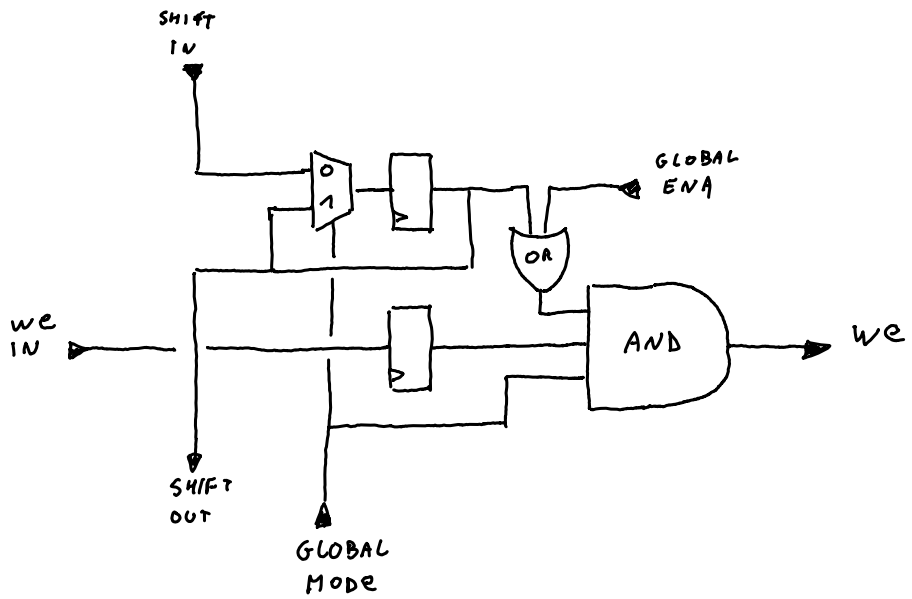


Figure 576.5: Generate block: WE diagram

There is only one control bit `local_ena` that operates the same way as for the common block. But the actual value is fed through a dedicated input signal `we_in`.

Capture

The capture block is very simple. It's composed of two blocks chained together, each handling 16 bits from read port A and 16 bits from read port B.

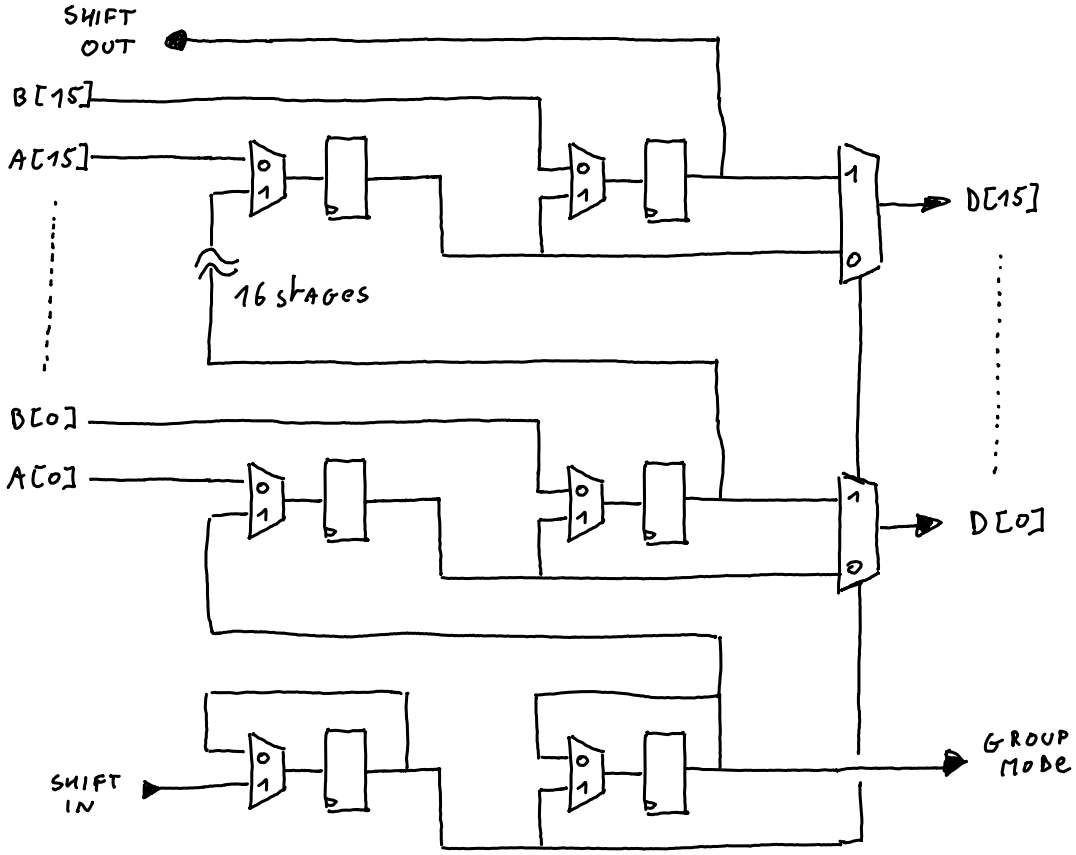


Figure 576.6: Capture block

The actual shift chain is actually composed of 34 bits, the two lower bits are control bits and then 32 bits of data, with the two read port interlaced.

There is a `cap_sel` signal that controls if the capture block is operating in "shift" mode, where the data that was previously captured can be shifted out, but also new data can be shifted in from the outside. This way you can both read what was captured, but also load controlled data into the chain.

If the shift mode is not enabled, when `cap_sel` is low, then the two control bits are fixed and don't change on the clock edge. The data bits are simply captured directly from the register file data outputs.

The output of the capture block is selected by one of those control bits. It only outputs either the values capture from read port A or of read port B. This is simply because the following compare block was a bit too large and I couldn't fit two of them, so automatic comparison tests are limited to a single port at a time.

Compare

The compare block is depicted below.

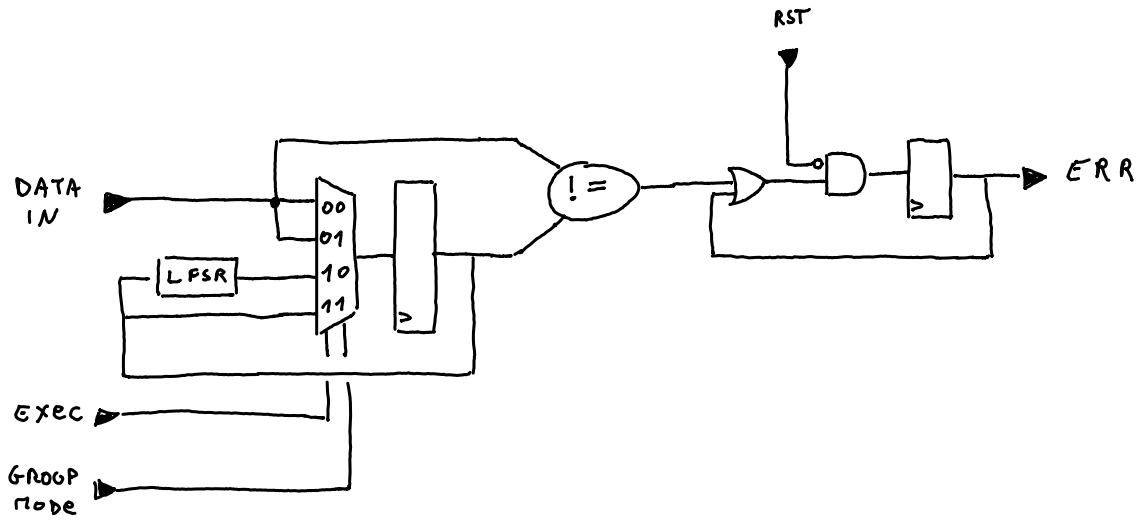


Figure 576.7: Compare block

Its purpose is to be able to directly compare captured data with expected value. Those can either be a fixed value, or a LFSR sequence, the same as generated by the generate block.

There is a group local signal `group_mode` that comes from the capture block and selects the operation mode, either fixed value or LFSR.

The two other control signals `rst` and `exec` are global. The former just resets the latching error signal to zero. The second one select if we're pre-loading the LFSR or comparison value, for initialization purposes, or if we're executing the comparison.

Typically, you'd preload some value in the capture stage using shift register, then clock once with `exec` set to zero to configure the compare stage and then automated comparison can begin.

There are some subtleties to the exact sequence to follow to properly synchronize generate and compare stage to properly automate long tests, but it is possible (tested in a verilog test bench).

How to test

No test procedure has been written yet. It's probable that at some point some custom test script for the demo board will be written to run functional tests but this is not done yet.

External hardware

The design itself doesn't use any external hardware and nothing should be connected to the PMODs of the demo board.

Basic functional testing is possible with the default demo board, controlling signals through the pico microcontroller.

However to do any advanced timing testing, measuring setup/hold times and such, very fine timing control will be needed. A scope will be needed to measure the skew between the critical control signals using the “feedback” mechanism and then some FPGA hardware will need to be used to have precise timing control of the signals sent to the design to make measurements.

The exact testing platform is still TBD.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	cmp_rst	cap_dat_out	—
1	cmp_exec	k_zero	clk_cap
2	cap_sel	cmp_err_msb	—
3	gen_dat_in	cmp_err_lsb	clk_gen
4	we_in	k_zero	—
5	cap_dat_in	k_zero	clk_rf
6	gen_glb_ena	fb_out	fb_sel[0]
7	gen_glb_mode	k_zero	fb_sel[1]

rc_servo_motor_xy

by **Yavar Hadi**

0578 | 50 MHz | Wokwi Project

`github.com/yavarhadi/rc_servo_motor_xy`

`wokwi.com/projects/0`

"rc servo motor chip "

How it works

This design is a two-channel RC-servo pulse generator. Two asynchronous comparator inputs (`ui_in[0]` = X, `ui_in[1]` = Y) are sampled and processed by the DUT. For each axis the logic produces a standard servo PWM frame with a period of 20 ms and a pulse width between 1.0 ms and 2.0 ms. The system clock is the TinyTapeout `clk` (50 MHz). Reset is active-low (`rst_n`) and outputs are forced low when `ena` is deasserted. The outputs are: `uo_out[0]` = PWM_X and `uo_out[1]` = PWM_Y. All other pins are unused and tied low.

How to test

1. Program the TT board with the bitstream containing this project and select it (`ena` high).
2. Drive `ui_in[0]` and `ui_in[1]` with digital levels to emulate the comparator results:
   - a higher duty of high time represents a larger tilt/command.
3. Observe `uo_out[0]` and `uo_out[1]` on a scope. You should see a 20 ms frame where the pulse width varies roughly 1.0–2.0 ms according to the inputs.
4. In simulation (Verilog or cocotb), toggle `ui_in[0]`/`ui_in[1]` periodically and check the measured pulse widths on `uo_out[0]`/`uo_out[1]`. Keep `rst_n=1` and provide a 50 MHz clock.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	comp_async_x_i	pwm_pin_x_o	—
1	comp_async_y_i	pwm_pin_y_o	—

#	Input	Output	Bidirectional
2	—	pwm_x_o	—
3	—	pwm_y_o	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

LEDs Racer

by **KerCrafter**

0580 | 50 MHz | HDL Project

`github.com/KerCrafter/FPGA-LEDs-Racer`

"Funny electronic game with WS2812B LEDs"

How it works

A fun little game, 4 players (Red, Blue, Green, Yellow) around a table, each with an arcade button, must challenge each other by pressing their button as fast as possible.

How to test

Pressing the button causes their led to progress to the center circle. The first to reach the center wins.

External hardware

I'm putting here the material I used for the project, but of course if it's compatible this list can be adapted.

- 4 x **Arcade buttons** (https://www.amazon.fr/EG-STARTS-Nouveau-Boutons-poussoirs-lumineux/dp/B01MSNXLN0/)
- 1 x **LEDs Circles (with WS2812B Leds)** (https://www.amazon.fr/Treedix-anneaux-WS2812B-adressable-Raspberry/dp/B0CD3DYJRK/)

Should be connected to the FPGA inputs with a pull down resistor

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	BLUE_BTN	LEDS_LINE	—
1	RED_BTN	TP_SCREEN_0	—
2	GREEN_BTN	TP_SCREEN_1	—
3	YELLOW_BTN	TP_BLUE_READY_TO_PLAY	—
4	—	TP_RED_READY_TO_PLAY	—
5	—	TP_GREEN_READY_TO_PLAY	—
6	—	TP_YELLOW_READY_TO_PLAY	—
7	—	TP_UPDATE_FRAME	—

VGA Screensaver with Tiny Tapeout Logo

by **Uri Shaked**

582 | 25.175 MHz | HDL Project

`github.com/TinyTapeout/tt-logo-screensaver`

"Tiny Tapeout Logo bouncing around the screen (640x480, TinyVGA Pmod)"

How it works

Displays a bouncing Tiny Tapeout logo on the screen, with animated color gradient.

Figure 582.1: Tiny Tapeout screensaver

How to test

Connect to a VGA monitor. Set the following inputs to configure the design:

- `tile` (ui_in[0]) to repeat the logo and tile it across the screen,
- `solid_color` (ui_in[1]) to use a solid color instead of an animated gradient.

If you have a Gamepad Pmod connected, you can also use the following controls:

- Start button: start/pause bouncing

- Left/right/up/down: change the bouncing direction (if bouncing) or move the logo around the screen (if paused)

External hardware

- Tiny VGA Pmod
- Optional: Gamepad Pmod

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	tile	R1	—
1	solid_color	G1	—
2	—	B1	—
3	—	VSync	—
4	gamepad_latch	R0	—
5	gamepad_clk	G0	—
6	gamepad_data	B0	—
7	—	HSync	—

Tiny Shader v2

by **Leo Moser**

584 | 50.35 MHz | HDL Project

`github.com/mole99/ttsky25b-tiny-shader`

"With Tiny Shader you can write a small program to create different images and even animations."

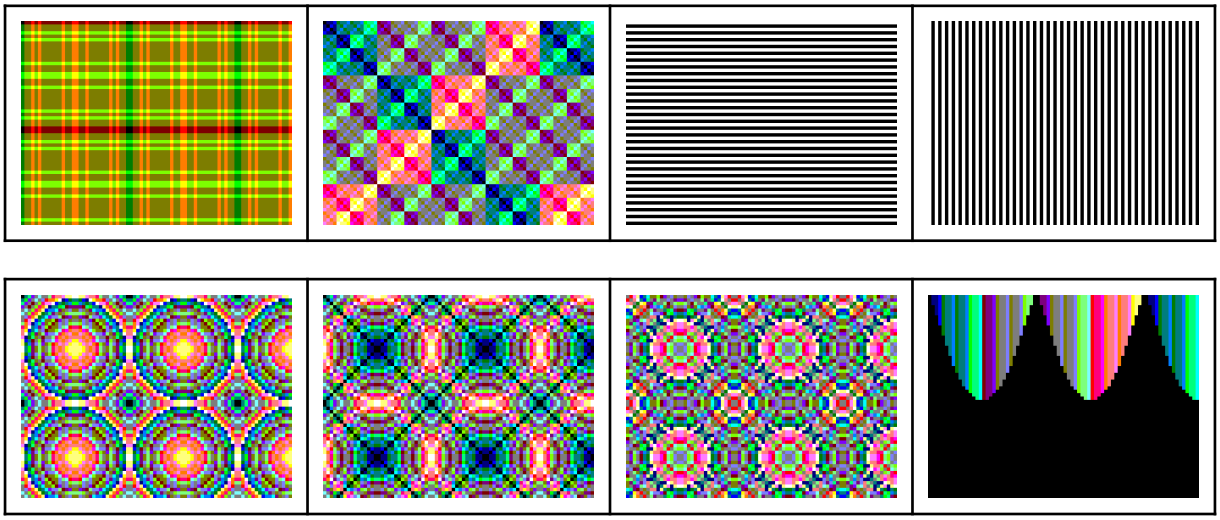
How it works

Modern GPUs use fragment shaders to determine the final color for each pixel. Thousands of shading units run in parallel to speed up this process and ensure that a high FPS ratio can be achieved.

Tiny Shader mimics such a shading unit and executes a shader with 16 instructions for each pixel. No framebuffer is used, the color values are generated on the fly. Tiny Shader also offers an SPI interface via which a new shader can be loaded. The final result can be viewed via the VGA output at 640x480 @ 60 Hz, although at an internal resolution of 80x60 pixel.

Examples

These images and many more can be generated with Tiny Shader. Note, that shaders can even be animated by accessing the `TIME0` or `TIME1` register.



The shader for the last image is shown here:

```
# Shader to display a rainbow colored sine wave

# Clear R3
CLEAR R3

# Get the sine value for x and add the user value
```

```
GETX R0
TIME1 R1
ADD R0 R1

# Set default color to R0
SETRGB R0

# Get the sine value for R0
SINE R0
HALF R0

# Get y coord
GETY R1

# If the sine value is greater
# or equal y, set color to black
IFGE R1
SETRGB R3
```

Architecture

Tiny Shader has four (mostly) general purpose registers, REG0 to REG3. REG0 is special in a way as it is the target or destination register for some instructions. All registers are 6 bit wide.

Input

The shader has four sources to get input from:

- `X` - X position of the current pixel
- `Y` - Y position of the current pixel
- `TIME0` - Increments at 7.5 Hz, before it overflow it counts down again.
- `TIME1` - Increments at 60 Hz, before it overflow it counts down again.

Output

The goal of the shader is to determine the final output color:

- `RGB` - The output color for the current pixel. Channel R, G and B can be set individually. If not set, the color of the previous pixel is used.

Sine Look Up Table

Tiny Shader contains a LUT with 16 6-bit sine values for a quarter of a sine wave. When accesing the LUT, the entries are automatically mirrored to form one half of a sine wave with a total of 32 6-bit values.

Instructions

The following instructions are supported by Tiny Shader. A program consists of 16 instructions and is executed for each pixel individually. The shader execution engine runs at double the VGA frequency. Therefore, the actual resolution is one eighth of the VGA resolution (80x60 pixel).

Output

Instruction	Operation	Description
SETRGB RA	RGB <= RA	Set the output color to the value of the specified register.
SETR RA	R <= RA[1:0]	Set the red channel of the output color to the lower two bits of the specified register.
SETG RA	G <= RA[1:0]	Set the green channel of the output color to the lower two bits of the specified register.
SETB RA	B <= RA[1:0]	Set the blue channel of the output color to the lower two bits of the specified register.

Input

Instruction	Operation	Description
GETX RA	RA <= X	Set the specified register to the x position of the current pixel.
GETY RA	RA <= Y	Set the specified register to the y position of the current pixel.
GETTIME0 RA	RA <= TIME0	Set the specified register to the current time0 value, increments at 7.5Hz and counts down down again.
GETTIME1 RA	RA <= TIME1	Set the specified register to the current time1 value, increments at 60Hz and counts down down again.

Branches

Instruction	Operation	Description
IFEQ RA	TAKE <= RA == R0	Execute the next instruction if RA equals R0.
IFNE RA	TAKE <= RA != R0	Execute the next instruction if RA does not equal R0.
IFGE RA	TAKE <= RA >= R0	Execute the next instruction if RA is greater then or equal R0.
IFLT RA	TAKE <= RA < R0	Execute the next instruction if RA is less than R0.

Arithmetic

Instruction	Operation	Description
DOUBLE RA	RA <= RA * 2	Double the value of RA.

HALF RA	RA <= RA / 2	Half the value of RA.
ADD RA RB	RA <= RA + RB	Add RA and RB, result written into RA.

Load

Instruction	Operation	Description
CLEAR RA	RA <= 0	Clear RA by writing 0.
LDI IMMEDIATE	RA <= IMMEDIATE	Load an immediate value into RA.

Special

Instruction	Operation	Description
SINE RA	RA <= SINE[R0[4:0]]	Get the sine value for R0 and write into RA. The sine value LUT has 32 entries.

Boolean

Instruction	Operation	Description
AND RA RB	RA <= RA & RB	Boolean AND of RA and RB, result written into RA.
OR RA RB	RA <= RA | RB	Boolean OR of RA and RB, result written into RA.
NOT RA RB	RA <= RB	Invert all bits of RB, result written into RA.
XOR RA RB	RA <= RA ^ RB	XOR of RA and RB, result written into RA.

Move

Instruction	Operation	Description
MOV RA RB	RA <= RB	Move value of RB into RA.

Shift

Instruction	Operation	Description
SHIFTL RA RB	RA <= RA << RB	Shift RA with RB to the left, result written into RA.
SHIFTR RA RB	RA <= RA >> RB	Shift RA with RB to the right, result written into RA.

Pseudo

Instruction	Operation	Description
NOP	R0 <= R0 & R0	No operation.

How to test

First set the clock to 50.25 MHz (2*25.175 MHz) and reset the design. For a simple test, simply connect a Tiny VGA to the output Pmod. A shader is loaded by default and an image should be displayed via VGA.

For advanced features, connect an SPI controller to the bidir pmod. You can shift in new data into the shader memory one byte at a time. If you set `ui[0]` (`pause_execute`) to 1 the shader will stop execution and you can safely shift in new data. Set it back to 0 for the shader to resume operation. This way you can load a new shader program. Have fun!

Experimental Features

Tiny Shader v2 has two additional signals: `double_instr` (`ui[1]`) and `half_resolution` (`ui[2]`).

In reality, the 16-instruction long shift register is a 32-instruction long shift register with a middle tap. By default, only the first 16 instructions are used. If `double_instr`, however, is pulled high, the shift register is rewired to use the full 32 instructions. Since one pixel is still 16 instructions, this means you can upload two different 16-instruction long shaders and each shader is executed for every other pixel, creating a dithering effect.

This effect is shown here:

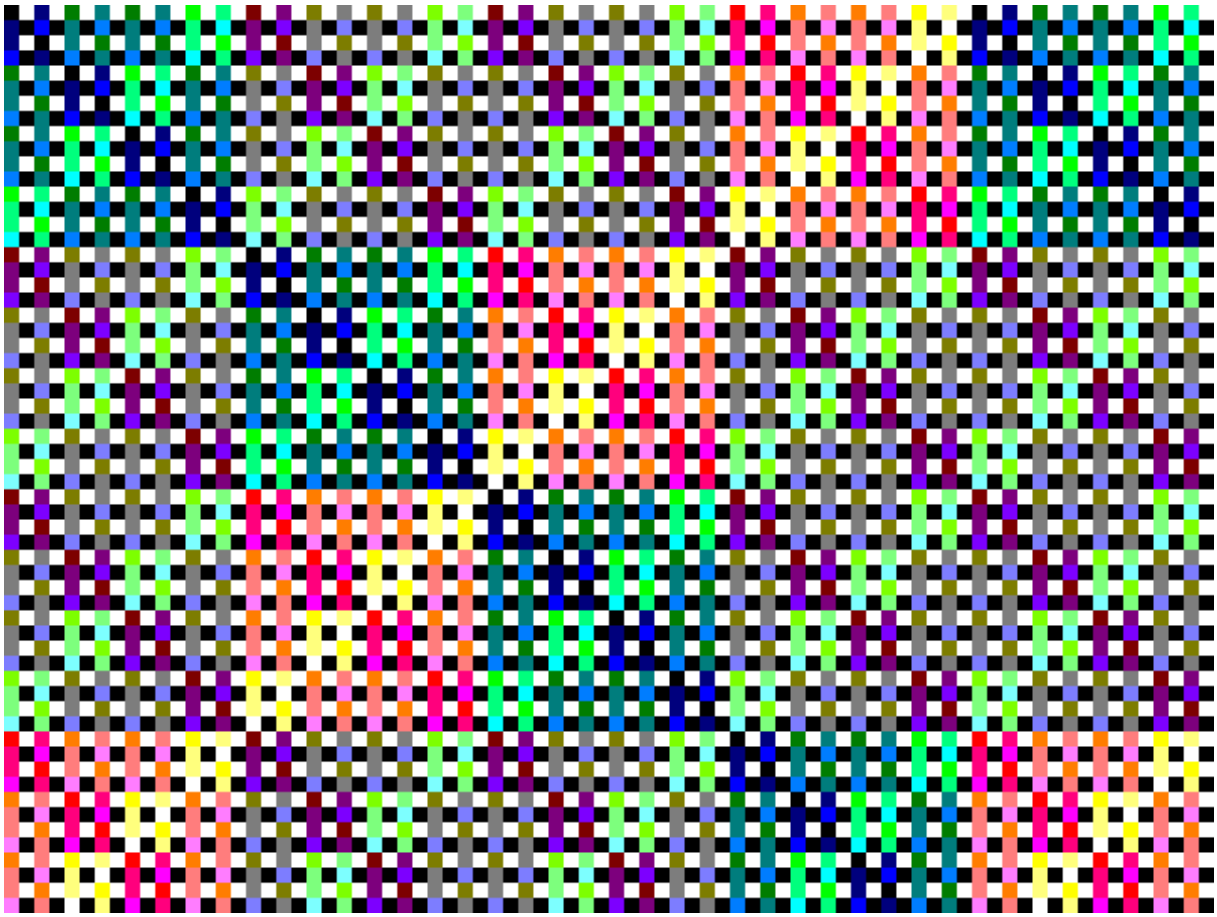


Figure 584.1: default_double.png

If `half_resolution` is also pulled high, each pixel uses 32 instructions. This halves the output resolution but enables 32-instruction shaders to be executed for a single pixel.

If `half_resolution` is high, but `double_instr` is low, then the the output resolution is simply halved with no other apparent effects. In reality, the 16-instruction shader is executed for each pixel twice. Perhaps this could be used to create a cool effect?

External hardware

- Tiny VGA or similar VGA Pmod
- Optional: SPI controller to upload new shaders

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	pause_execute	R[1]	spi_cs
1	double_instr	G[1]	spi_mosi
2	half_resolution	B[1]	spi_miso

#	Input	Output	Bidirectional
3	—	vsync	spi_sclk
4	—	R[0]	next_vertical
5	—	G[0]	next_frame
6	—	B[0]	—
7	—	hsync	—

TTSKY25b-RISCV-Core

by **David Awad**

#586 HDL Project

`github.com/davidawad/TTSKY25b-RISCV-Core`

"a copy of underserved as a memento from the course"

How it works

This project implements a minimal 32-bit RISC-V processor core (RV32I) using the SERV architecture - one of the world's smallest RISC-V CPUs. The design is optimized for area efficiency, using a bit-serial architecture that processes instructions one bit at a time over 32 clock cycles.

Core Components

SERV CPU Core - A bit-serial RISC-V processor that implements the base integer instruction set (RV32I). The core features:

- 32-bit instruction processing
- Bit-serial arithmetic logic unit (ALU)
- Custom shift register-based register file (5 registers)
- No CSR (Control and Status Registers) support for minimal area
- No compressed instruction support
- No multiply/divide unit

SPI Flash Interface - The `spimemio` module provides access to external SPI flash memory where the RISC-V program is stored. This allows the small chip to execute larger programs stored off-chip.

GPIO Controller - The `subservient_gpio` module provides 5 general-purpose output pins that can be controlled by the RISC-V software, allowing the processor to interact with external hardware.

Register File - Instead of using traditional SRAM, this design uses a shift register implementation (`rf_shift_reg`) with only 5 registers to minimize silicon area.

Data Flow

1. CPU fetches instructions from external SPI flash via the instruction bus
2. Instructions are decoded and executed bit-serially over 32 cycles
3. Data memory accesses go through the data bus, which can access either GPIO registers or the SPI flash
4. GPIO writes allow the program to control the 5 output pins

How to test

Hardware Requirements

- **External SPI Flash** (e.g., W25Q80, AT25SF081, or similar)
  - Connect to pins as follows:
    - Flash CS# → `uo_out[6]`
    - Flash CLK → `uo_out[5]`
    - Flash MOSI → `uo_out[7]`
    - Flash MISO → `ui_in[7]`

Programming the Flash

Before testing, you need to program the SPI flash with RISC-V machine code:

1. Write a simple RISC-V assembly program (RV32I instruction set)
2. Compile to raw binary using `riscv32-unknown-elf-gcc` and `objcopy`
3. Program the binary to SPI flash starting at address 0x000000

Example Test Program

A minimal blinker program that toggles GPIO pins:

```
# Simple LED blinker for GPIO pins
.section .text
.global _start
_start:
    li t0, 0x00     # Load 0 into t0
loop:
    sw t0, 0(x0)    # Write to GPIO (address 0)
    addi t0, t0, 1  # Increment
    j loop          # Repeat
```

Expected Behavior

1. On power-up and reset release, the CPU should start fetching from address 0x00000000
2. GPIO pins `uo_out[4:0]` should change according to your program
3. SPI flash signals should show:
   - Clock toggling on `uo_out[5]`
   - CS# going low during reads on `uo_out[6]`
   - Data on MOSI/MISO (`uo_out[7]` and `ui_in[7]`)

Simulation Testing

The project includes a testbench (`test/tb.v` and `test/test.py`) but these currently test the simple adder example. To properly test the RISC-V core:

1. Create a small test program binary
2. Modify the testbench to load this binary into a simulated SPI flash
3. Run simulation and verify GPIO outputs match expected program behavior
4. Check that instruction fetches occur correctly from flash

Debugging Tips

- Monitor SPI flash signals to verify instruction fetches
- Start with very simple programs (e.g., just write to GPIO)
- Use a logic analyzer to capture SPI transactions
- Remember: Each instruction takes 32+ clock cycles due to bit-serial execution

External hardware

Required:

- **SPI Flash Memory** (e.g., Winbond W25Q80, Adesto AT25SF081, or compatible)
  - Minimum 1Mbit (128KB) recommended
  - 3.3V compatible
  - Standard SPI interface

Optional:

- **LEDs** connected to GPIO pins `uo_out[4:0]` through current-limiting resistors (1kΩ suggested)
- **Logic analyzer** for debugging SPI flash communication

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	gpio0	—
1	—	gpio1	—
2	—	gpio2	—
3	—	gpio3	—
4	—	gpio4	—
5	—	sclk	—
6	—	cs_n	—
7	—	mosi	—

VGA Screensaver with Zero to ASIC Logo

by **Matt Venn** & **Louis Ledoux**

0588 | 25.175 MHz | HDL Project

`github.com/Bynaryman/ttihp0p3-z2a-logo-screensaver`

"VGA Zero to ASIC Logo bouncing around the screen meta shaped chip"

How it works

This design drives a 640×480 VGA output and displays a bouncing "Zero to ASIC" logo. It supports a tiled mode and an animated color gradient. Optionally, a Gamepad Pmod can control motion.

Figure 588.1: Tiny Tapeout screensaver

How to test

Connect to a VGA monitor. Set the following inputs to configure the design:

- `tile` (ui_in[0]) to repeat the logo and tile it across the screen,
- `solid_color` (ui_in[1]) to use a solid color instead of an animated gradient.

If you have a Gamepad Pmod connected, you can also use the following controls:

- Start button: start/pause bouncing

- Left/right/up/down: change the bouncing direction (if bouncing) or move the logo around the screen (if paused)

External hardware

- Tiny VGA Pmod
- Optional: Gamepad Pmod

Harden flow with artwork (advanced)

We extend LibreLane's Classic flow by inserting a custom ODB Python step before global placement. This step can generate placement blockages from a bitmap to "shape" placement density and create chip art.

- Base: LibreLane Classic flow
- Extension: inserts `Odb.ApplyArt` just before `OpenROAD.GlobalPlacementSkipIO`
- Implementation: lightweight LibreLane plugin packaged in a small Docker overlay

Quick start

1) Build the custom LibreLane image (adds the plugin + Pillow):

- `docker build -t my-librelane:mock -f docker/librelane-mock/Dockerfile .`

2) Select the image and choose an input image (PNG/JPG):

- `export LIBRELANE_IMAGE_OVERRIDE=my-librelane:mock`
- `export TT_ART_IMAGE="$PWD/art/zero-to-asic-logo.png"`

Optional knobs:

- `TT_ART_GRID` (int, default 40): grid columns; rows follow image aspect
- `TT_ART_THRESHOLD` (0..255, default 128): luminance threshold
- `TT_ART_INVERT` (1/true/yes): invert selection
- `TT_ART_AREA_PCT` (0..100, default 20): scale art area within the core

3) Run harden with the art step enabled:

- `./tt/tt_tool.py --harden --with-art`

4) Inspect results:

- Step log: `runs/wokwi/<NN>-odb-applyart/odb-applyart.log`
- Preview: `runs/wokwi/<NN>-odb-applyart/art_preview.png`

Notes:

- If container env vars aren't forwarded, the tool writes `art_config.json` in the project root and the step reads it automatically.
- The art step fails when no image is provided, to avoid silent fallbacks.

What the step does

- Reads the OpenDB after floorplan
- Rasterizes the input image to a grid
- For selected cells, creates soft placement blockages in the core bounding box (scaled by `TT_ART_AREA_PCT`)
- Writes the ODB back in-place and saves a small preview image

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	tile	R1	—
1	solid_color	G1	—
2	—	B1	—
3	—	VSync	—
4	gamepad_latch	R0	—
5	gamepad_clk	G0	—
6	gamepad_data	B0	—
7	—	HSync	—

Verilog Meetup Template Project Example

by **Yuri Panchul** & **with credits to others**

0590 | 25 MHz | HDL Project

`github.com/yuri-panchul/ttsky-verilog-calpoly-slo-final`

> *"An ASIC project compatible with an FPGA project from BGM (basic-graphics-music) infrastructure. A test for the Verilog Meetup variant of the Tiny Tapeout Verilog Project Template. Prepared for the meetup at California Polytechnic State University, San Luis Obispo in October 2025."*

How it works

It receives sound level from an INMP441 microphone, runs sound recognition, displays the note on a 7-segment display of a TM1638-based interface module and draws color patterns on a VGA screen using Tiny VGA interface module.

How to test

See "How it works".

External hardware

See "How it works".

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	switch [0]	VGA red [1]	INMP441 mic I2S l/r
1	switch [1]	VGA green [1]	INMP441 mic I2S ws
2	switch [2]	VGA blue [1]	INMP441 mic I2S sck
3	switch [3]	VGA vsync	INMP441 mic I2S sd
4	switch [4]	VGA red [0]	output sticky_failure
5	switch [5]	VGA green [0]	TM1638 inout dio
6	switch [6]	VGA blue [0]	TM1638 output clk
7	UART rx	VGA hsync	TM1638 output stb

Electronic Voting Machine

by **Tarun B**

0608 | 10 Hz | HDL Project

github.com/BhanuVasanthButti/EVM

"A simple EVM depicting the practical behaviour"

How it works

The electronic voting machine (EVM) design counts votes for three candidates. Each candidate has one input button — pressing it increments that candidate's vote count. When the vote_over signal is asserted, the voting process ends, and results for all three candidates are displayed as 6-bit binary outputs (o_count1, o_count2, o_count3).A reset (rst) signal clears all counts, preparing the system for a new voting session.

How to test

The testbench (tb_voting_machine) generates a clock and reset, then simulates button presses for all three candidates. Each press and release represents a vote, and after several votes, the vote_over signal is asserted to stop voting. Finally, it displays the total vote counts for each candidate and dumps all signals into a VCD file for waveform viewing.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a[0]	b[0]	c[0]
1	a[1]	b[1]	c[1]
2	a[2]	b[2]	c[2]
3	a[3]	b[3]	c[3]
4	a[4]	b[4]	c[4]
5	a[5]	b[5]	c[5]
6	a[6]	b[6]	c[6]
7	a[7]	b[7]	c[7]

tt-wokwi

by **Maximilian Greger**

0609 | 10 kHz | Wokwi Project

`github.com/GregMa220214/tt-wokwi-set1`
`wokwi.com/projects/442987636779898881`

"just trying somethings out in Wokwi and trying to use a dsr Flip Flop"

How it works

It was created to test the dsr Flip Flop.

How to test

Use the Dip switches to set 1 or nothing

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input xor	—	—
1	input xor	q from dsr Flip Flop	—
2	—	q from dsr Flip Flop	—
3	—	—	—
4	input to Reset	—	—
5	input to Data	—	—
6	—	—	—
7	—	—	—

AXIS_UART_TTO

by **K Shreenidhi**, **Punith Roy S** & **Krishna Negi**

0610 | 50 MHz | HDL Project

github.com/KShreeni/AXIS_UART_TTO

"Data generated by the AXI-Stream master is pushed into the FIFO. From the FIFO,to the UART transmitter,and to the UART receiver which reconstructs the parallel data, and outputs it."

How it works

This project bridges a parallel AXI-Stream interface with a serial UART communication protocol. Data is received from an AXI master, stored in an 8-byte synchronous FIFO to handle clock domain and speed mismatch, then serialized and transmitted over UART. A loopback UART receiver is integrated to validate data integrity.

Flow of data:

1. AXI Input Stage: AXI master asserts axis_valid with data on axis_data. When m_axis_ready is high, data is accepted and written to FIFO.
2. FIFO Buffering: FIFO stores incoming bytes until the UART transmitter is ready to send. Read and write can occur simultaneously.
3. UART Transmission:

a) Each byte is serialized at 115200 baud.

b) Start bit (0), data bits (8), parity bit, and stop bit (1) are sent.

c) tx_data_ready indicates when UART can accept new data.

4. UART Reception:

a) Receiver detects start bit, samples bits at mid-baud intervals.

b) Extracts data, verifies parity, and asserts rx_valid.

c) If parity check fails, parity_err is asserted for one clock.

5. Loopback Communication: The transmitted UART data (uart_tx) is internally connected to the receiver's rx line, allowing full system verification.

How to test

You can test the design in two ways:

1. Simulation (Testbench)

Provide sequential 8-bit ASCII values to the AXI input (axis_data, axis_valid).

Monitor UART TX output to see data serialized.

Observe rx_data and rx_valid to verify that the UART receiver reconstructs the same data.

2. Hardware

Use a UART terminal at 115200 baud rate.

Send data through the AXI interface (or mapped IO pins).

View the corresponding UART output on TX pin.

Since the design includes loopback, received data is automatically validated and made available on output pins.

A successful test shows the same data at rx_data that was sent via axis_data.

External hardware

No external hardware is necessary to verify core functionality.

The project is designed to be fully testable on-chip using the internal UART loopback.

External UART or LEDs can be added only if real-time debugging is required.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a[0]	b[0]	c[0]
1	a[1]	b[1]	c[1]
2	a[2]	b[2]	c[2]
3	a[3]	b[3]	c[3]
4	a[4]	b[4]	c[4]
5	a[5]	b[5]	c[5]
6	a[6]	b[6]	c[6]
7	a[7]	b[7]	c[7]

Numbers

by **Maxi Nigl**

0611 | 10 kHz | Wokwi Project

github.com/Maxskill07/Chip

wokwi.com/projects/442977691824856065

"Nummern anzeigen"

How it works

Press buttons for numbers

How to test

press buttons ## External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	1	—	—
1	2	—	—
2	3	—	—
3	4	—	—
4	5	—	—
5	6	—	—
6	7	—	—
7	8	—	—

Numerically Controlled Oscillator

by **Snevin Leoneel Dsouza**, **Kishan Revankar** & **Suhan Rai K**

0612 | 50 MHz | HDL Project

github.com/snevindsouza/NCO_TTO

"A versatile NCO, capable of generating a diverse range of waveforms such as sine, cosine, triangular, square, gaussian chirplet and ecg waves."

How it works

The project implements a fully digital NCO/DDS core that generates different waveform outputs (sine, cosine, triangle, sawtooth, square, ramp) using a phase accumulator and a waveform lookup ROM. Each clock cycle, the phase accumulator adds a frequency tuning word (FTW) to determine the waveform phase. This phase value is then mapped to an amplitude using the ROM. The resulting 8-bit digital waveform is sent to the output, either as parallel digital data or through a PWM/resistor-ladder DAC for analog output. The waveform type is selected using simple control pins.

How to test

We can test the design in simulation or on hardware. In simulation, run the provided testbench to verify frequency sweep, waveform selection, and edge cases. On hardware, connect the 8-bit output to an external R-2R resistor ladder DAC and view the generated waveform on an oscilloscope. Change the waveform select pins to observe different shapes (sine, triangle, etc.) and vary the FTW to adjust frequency.

External hardware

R-2R Resistor Ladder DAC – converts the 8-bit digital output to an analog waveform.

Oscilloscope – used to visualize the generated waveforms.

Optional LEDs or logic analyzer – to observe the digital output pattern directly.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a[0]	b[0]	c[0]

#	Input	Output	Bidirectional
1	a[1]	b[1]	c[1]
2	a[2]	b[2]	c[2]
3	a[3]	b[3]	c[3]
4	a[4]	b[4]	c[4]
5	a[5]	b[5]	c[5]
6	a[6]	b[6]	c[6]
7	a[7]	b[7]	c[7]

VGA clock render – Designed by Matt Venn. Illustrated by Máximo Balestrini.

Siebensegment_Chip

by **Mörzinger**

0613 | 10 kHz | Wokwi Project

`github.com/M1chiM/Siebensegment`

`wokwi.com/projects/442988211554861057`

"Siebensegmentanzeige"

How it works

By pressing a number the number should light up

How to test

just try out

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	INT0	OUT0	—
1	INT1	OUT1	—
2	INT2	OUT2	—
3	INT3	OUT3	—
4	INT4	OUT4	—
5	INT5	OUT5	—
6	INT6	OUT6	—
7	INT7	OUT7	—

Morse Code Detector

by **Yagnadatt Sarangi**

0614 | 2 Hz | HDL Project

github.com/yphoenixs/morse_code_tto

"The Morse Code Converter project translates Morse code sequences of dots and dashes into English alphabets (A–Z) and digits (0–9)."

How it works

This project simulates a Morse Code Transmitter and Receiver system implemented in Verilog HDL. It uses two Finite State Machines (FSMs) — one for transmission (trans_fsm) and another for reception (rec_fsm) — to encode and decode Morse code for alphabets (a–z) and digits (0–9).

Transmitter FSM (trans_fsm):

Converts button inputs (dot_inp, dash_inp, char_space_inp, word_space_inp) into a 3-bit encoded serial Morse signal (p_out).

Uses states like IDLE, DOT, DASH, CHAR_SPACE, and WORD_SPACE to send Morse sequences with proper timing gaps between symbols, characters, and words.

Receiver FSM (rec_fsm):

Samples the transmitted serial input (p_in) and reconstructs it into the corresponding 8-bit ASCII character (s_out).

Decoding is based on internal lookup tables that map Morse sequences to characters.

The two FSMs operate synchronously through the system clock (clk), ensuring correct timing and reset behavior through the active-low reset (rst).

How to test

Apply a reset pulse (rst = 0 → 1) to initialize both FSMs.

Provide input signals on the transmitter side:

dot_inp = 1 for a dot.

dash_inp = 1 for a dash.

char_space_inp = 1 for character spacing.

word_space_inp = 1 for word spacing.

Observe the transmitted Morse output (p_out) and verify it transitions according to the input sequence.

The receiver FSM should decode this serial Morse pattern and output the correct ASCII character on s_out.

Example: For Morse input corresponding to the letter A (dot, dash and character space), the output should be s_out = 8'h61 (ASCII for 'a').

External hardware

16x2 LCD FPGA Board

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a[0]	b[0]	c[0]
1	a[1]	b[1]	c[1]
2	a[2]	b[2]	c[2]
3	a[3]	b[3]	c[3]
4	a[4]	b[4]	c[4]
5	a[5]	b[5]	c[5]
6	a[6]	b[6]	c[6]
7	a[7]	b[7]	c[7]

Kleine Tapeout Vorlagen-Kopie

by **Aut_Lanzelot**

0615 Wokwi Project

github.com/AutLanzelot/7-Segment-Anzeige
wokwi.com/projects/442982978057266177

"7-Segment-Anzeige"

How it works

Press the switch

How to test

Press the switch ## External hardware

idk what to do

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	IN7	OUT7	—

Tiny Hardware Authentication Engine

by **Raghul G**, **Swedha Pragalya V** & **Antony Bacil C**

`616` `50 MHz` `HDL Project`

`github.com/raghul-g1322/tiny_hardware_authentication_tto`

"A Tiny Hardware Authentication Engine that generates a 4-digit OTP using a 16-bit LFSR with time-bound validity and multi-attempt lockout. The design is lightweight, Tiny Tapeout friendly, and observable on FPGA using LEDs and 7-segment displays."

How it works

The **Tiny Hardware OTP Authenticator** is a self-contained digital authentication system implemented on the **Tiny Tapeout** platform.
It generates, stores, and verifies a **4-digit (16-bit) One-Time Password (OTP)** using a **Linear Feedback Shift Register (LFSR)** and a **Finite State Machine (FSM)** that controls all authentication, retry, and timeout operations.

Functional Flow

1. The **LFSR** runs continuously, generating a new 16-bit pseudo-random OTP on every clock cycle.
2. When the **`otp_latch`** signal is pulsed, the current 16-bit value from the LFSR is stored as the active OTP for that session.
3. The **user** enters a 4-digit OTP attempt using `user_in[3:0]`, one digit at a time.
4. Each digit is captured when **`user_latch`** is pulsed.
5. Once all 4 digits are entered, the FSM compares the entered OTP with the latched OTP and determines the outcome:
   - ✅ **U (Unlocked):** Correct OTP entered
   - ❌ **L (Locked):** Incorrect OTP (3 failed attempts)
   - ⏰ **E (Expired):** 50 seconds of inactivity
6. The system also includes a **display driver** that alternates between OTP and system status every 2 seconds.

Display Information

- **Right 4 digits:** User-entered OTP (`USER_OTP`)
- **Left 4 digits:** Toggles every 2 seconds between:
  - The generated LFSR OTP (for reference)
  - The status format **"A#-L/U/E"**, where:
    - **A#** → Attempt number (A1, A2, A3)
    - **L/U/E** → Lock / Unlock / Expire status

All information (LFSR OTP, user OTP, attempt count, and state) is shown on **8 multiplexed 7-segment displays**, making this a standalone hardware authenticator.

Finite State Machine (FSM)

The FSM manages OTP generation, user input, comparison, and timeout handling.

State Encoding

State	Code	Description
IDLE	`2'b00`	Default/reset state; initializes all signals.
GENERATE_OTP	`2'b01`	Reads OTP from LFSR; stores when `otp_latch` is asserted.
ENTER_OTP	`2'b10`	Accepts 4-digit user input; detects expiry after 50s inactivity.
UNLOCK	`2'b11`	Compares OTPs and determines lock/unlock; increments attempt counter.

State Transitions

From	Condition	To
IDLE	`reset = 1`	`GENERATE_OTP`
GENERATE_OTP	`otp_latch = 1`	`ENTER_OTP`
ENTER_OTP	4 user entries complete	`UNLOCK`
ENTER_OTP	Timeout > 50s	`IDLE (EXPIRED)`
UNLOCK	OTP match	`IDLE (UNLOCKED)` after 5s
UNLOCK	OTP mismatch < 3 attempts	`ENTER_OTP`
UNLOCK	3rd mismatch	`IDLE (LOCKED)` after 5s

At any state, if `reset = 0`, the system resets to `IDLE`.

Timing and Clock Details

Signal	Frequency	Description
clk (Master Clock)	50 MHz	Drives LFSR and FSM logic

Display Multiplexing Clock	2 kHz (0.5 ms)	Controls 7-segment display scanning
Display Toggle Clock	0.5 Hz (2 sec)	Toggles between OTP and status display
Expire Timer	Derived from master clock	Detects 50-second inactivity
Hold Timer	Derived from master clock	Holds result for 5 seconds for readability

How to test

1. **Power on / Reset** → The LFSR starts generating random 16-bit OTPs.
2. **Latch OTP:** Pulse `otp_latch` to store the current OTP for authentication.
3. **Enter User OTP:**
   - Use `user_in[3:0]` to set each 4-bit digit (0–9).
   - Pulse `user_latch` to confirm each digit entry.
4. After entering all four digits:
   - FSM compares the OTP and displays the result on the **left 4 displays**:
     - `A#-U` → Unlocked
     - `A#-L` → Locked
     - `A#-E` → Expired
5. The system automatically returns to **IDLE** after 5 seconds of displaying results.
6. Reset the system or latch a new OTP to restart.

External hardware

Component	Description	Purpose
8× Seven-Segment Display	Common cathode, connected to `lfsr_out`, `user_out`, and `an`	Displays OTP and user input/status
Push Buttons / Switches	Connected to `otp_latch` and `user_latch`	For OTP latch and digit entry
Clock (50 MHz)	Onboard or external	Main system timing
Reset Button	Connected to `reset_n`	Resets FSM and regenerates OTP

Tip: During simulation, display values can be viewed in waveform format instead of physical LEDs.

Signal Specifications

Signal	Direction	Width	Description
`clk`	Input	1	System clock
`reset_n`	Input	1	Active-low asynchronous reset
`user_in`	Input	4	4-bit user digit input
`otp_latch`	Input	1	Captures current LFSR OTP
`user_latch`	Input	1	Captures user-entered digit
`lfsr_out`	Output	7	Decoded LFSR output for 7-segment display
`user_out`	Output	7	Decoded user output for 7-segment display
`an`	Output	2	Anode control for multiplexed display

End of Document

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	user_in[0]	lfsr_out[0]	user_out[0]
1	user_in[1]	lfsr_out[1]	user_out[1]
2	user_in[2]	lfsr_out[2]	user_out[2]
3	user_in[3]	lfsr_out[3]	user_out[3]
4	otp_latch	lfsr_out[4]	user_out[4]
5	user_latch	lfsr_out[5]	user_out[5]
6	Unused	lfsr_out[6]	user_out[6]
7	Unused	an[1]	an[0]

Tiny Tapeout meli

by **Meli**

0617 | 1 kHz | Wokwi Project

github.com/MelanieBauer-creator/meli

wokwi.com/projects/442977531497515009

"Test some logic gates"

How it works

Explain how your project works ..

How to test

Explain how to use your project ..

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any ..

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OP And	—
1	IN1	—	—
2	IN2	OP Or	—
3	IN3	OP Nand	—
4	IN4	OP And	—
5	IN5	—	—
6	IN6	—	—
7	IN7	—	—

Binary Counter

by **TJ Johnson**

0618 | 1 Hz | Wokwi Project

github.com/FlyTJ/TinyTapeout

wokwi.com/projects/446992736864284673

> *"Binary counter that runs on internal 1Hz clock and outputs high logic on each of the outputs to light LEDS to show a binary representation of a counter. Note that upon boot, the lights might already be in some sequence as I could not figure out to issue a reset on the D Flipflops"*

How it works

The built in 1 Hz clock sends a pulse to a D Flip Flop array that counts in binary with outputs being hooked up to LEDs. Note that some of the flipflops might be in an unknown state so might be lit upon powerup. I could not figure out how to reset the D Flipflop upon powerup.

How to test

Simply power up the unit, and make sure you have at least one LED hooked up to the first output. You can connect up to 8 LEDS - for all of the outputs to show a binary counter.

External hardware

None other than the LEDs and possible resistors for each LED to limit current.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	LED Bit 1	—
1	—	LED Bit 2	—
2	—	LED Bit 3	—
3	—	LED Bit 4	—
4	—	LED Bit 5	—
5	—	LED Bit 6	—
6	—	LED Bit 7	—
7	—	LED Bit 8	—

Template

by **Me**

0619 | 10 kHz | Wokwi Project

`github.com/rajani-25-byte/ttsky-wokwi-template`
`wokwi.com/projects/442985098452614145`

"required"

How it works

created

How to test

by giving inputs

External hardware

no hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in0	out0	—
1	in1	out1	—
2	in2	out2	—
3	in3	out3	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Manchester Encode/Decode with built-in 5X PLL

by **patrick-lin-git**

0620 | 25 MHz | HDL Project

`github.com/patrick-lin-git/ttsky-verilog-template`

"Manchester Encode/Decode"

Manchester Encode/Decode

```
### Author:       Patrick Lin  (https://github.com/patrick-lin-git)
### Description:  Machester Encode/Decode
### Language:     Systemverilog
```

How it works

This chip perform machester encode or decode function that conver a paralle 8 bit of data to serial bit stream. Here below is the pin function:

INPUT:

```
CLK100M   100MHz Clock is needed when DIR pin is 0. if DIR is 1,
keep this input low.
DIR       Control data conversion direction, 1: Encode (TX) / 0:
Decode (RX)
HALT      Control Halt or normal, 1: STOP Paralle/Serial
Conversion / 0: Normal Function, do auto convert repeatly
BIST      1: Enter BIST Mode / 0: Normal Mode
RXD       if DIR is 0, this should connect the counter part chip.
Keep low if DIR is 1.
```

OUTPUT:

```
DBGO      Internal Debug Singal Output
TXD       Valid when DIR is 1, Keep floating if DIR is 0.
```

How to test

During BIST Mode, all bi-directional IO pin will become input. DBGO[0] will refect the compare error if TX to Rx loopback doesn't match.

External hardware

```
This chip require two clock
    25MHz
    100MHz
```

8 Bi-directional pins might be input or output mode depends on the DIR pin setting.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	CLK100M	DBG_OUT0	IO0
1	DIR	DBG_OUT1	IO1
2	HALT	DBG_OUT2	IO2
3	BIST	DBG_OUT3	IO3
4	PLL_EN	DBG_OUT4	IO4
5	DBG_SEL0	DBG_OUT5	IO5
6	DBG_SEL1	DBG_OUT6	IO6
7	MCHT_RXD	MCHT_TXD	IO7

Tiny tapeout test

by **Fabian Wunsch**

0621 Wokwi Project

github.com/fabi321/tt-01

wokwi.com/projects/442983461306274817

"Test the tapeout process"

How it works

My project works by working

How to test

You can use it by using it

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input 0	Output 0	—
1	Input 1	Output 1	—
2	Input 2	Output 2	—
3	Input 3	Output 3	—
4	Input 4	Output 4	—
5	Input 5	Output 5	—
6	Input 6	Output 6	—
7	Input 7	Output 7	—

Negative Nature #1

by **bleeptrack**

0622 HDL Project

`github.com/bleeptrack/negative-nature01`

"generative voronoi pattern for negative space in silicon"

How it works

Silicon art on met4.

How to test

No test, just looks :)

External hardware

Nothing needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	dummy	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny Tapeout Template Hammer

by **Hammer Fabian**

0623 | 10 kHz | Wokwi Project

github.com/fabianhammer/Hammer1

wokwi.com/projects/442983243536903169

"sign the numbers from 0 to 9 on an 8-segment display xxx"

How it works

sign numbers

How to test

buttons

External hardware

8 buttons

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	i0	o0	x0
1	i1	o1	x1
2	i2	o2	x3
3	i3	o3	x4
4	i4	o4	x5
5	i5	o5	x6
6	i6	o6	x7
7	i7	o7	x8

SimProc (Simple Processor)

by **Saptarshi Talukdar**

0642 HDL Project

github.com/ieeeuoft-asic/simproc

"8-bit Process that can be programmed via UART"

How it works

Send 3 byte packets through UART, program SimProc, read memory data / SimProc data through UART

How to test

Send commands using the UART RX pin (using a USB to UART conveter), receive data through the UART TX pin. Set UIO7-UIO0 to set the BAUD rate of the internal UART module.

External hardware

A USB to UART converter or any UART device.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	uart rx	uart tx	clk per bit 0
1	—	halt flag	clk per bit 1
2	—	done flag	clk per bit 2
3	—	—	clk per bit 3
4	—	—	clk per bit 4
5	—	—	clk per bit 5
6	—	—	clk per bit 6
7	—	—	clk per bit 7

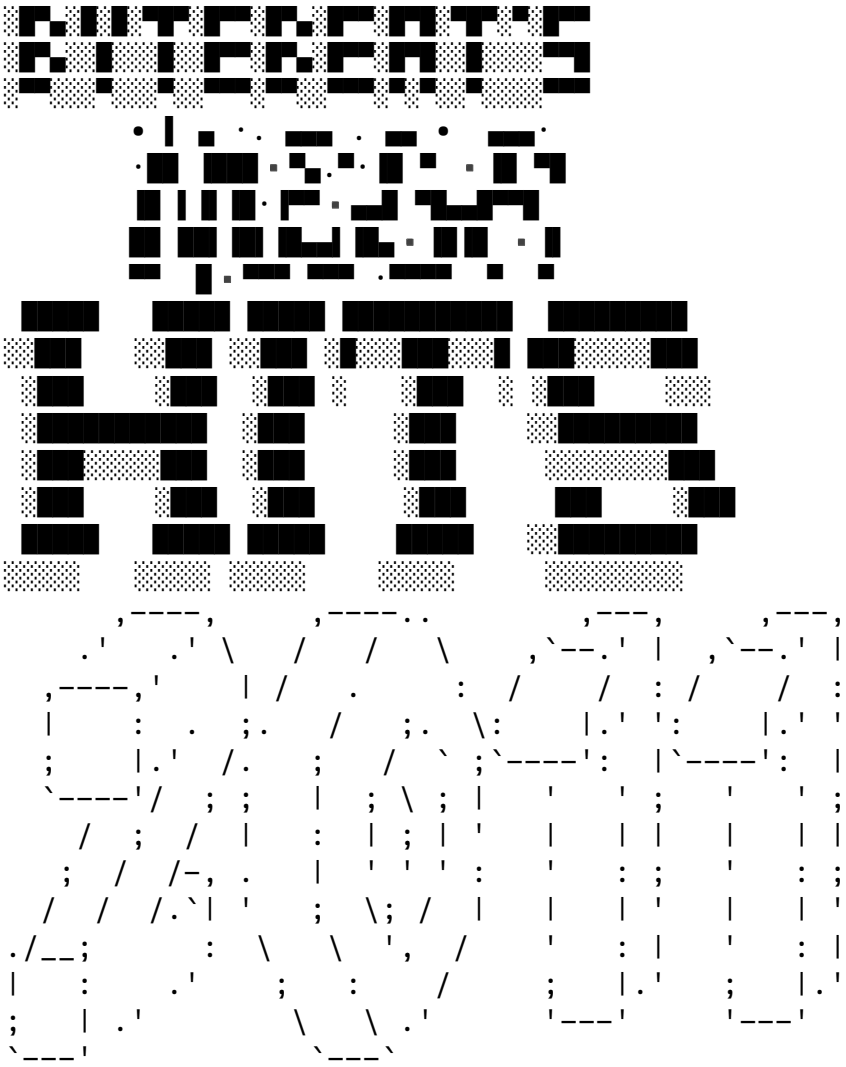
megabytebeat

by **proppy**

0648 | 2.048 MHz | HDL Project

`github.com/proppy/ttsky25-megabytebeat`

"Best-of compilation of bytebeat 2011's greatest hits on-a-chip."



Extended Play of some of the 2011's BYTEBEAT greatest hits on-a-chip:

Pin	Track	Artist
`out0`	the 42 melody	people on irc
`out1`	fractal trees	danharaj
`out2`	untitled	droid
`out3`	a tune to share	Niklas_Roy
`out7`	sierpinski harmony	miiro

How it works

The main module accept parameters from 4x 4-bit parameters buses and generate PWM audio signal on each output pins according to the following bytebeat formulas:

Pin	Formula
`out0`	`t*({a,b}&t>>{c,d})`
`out1`	`t|t%{a,b}|t%(1+{c,d})`
`out2`	`t>>a&b?t>>c:-t>>d`
`out3`	`t*(t>>9|t>>13)&16`
`out7`	`t*a&(t>>b)|t*3&(t*c>>d)`

How to test

- Connect a speaker to the pin you want to "play".
- Tweak parameters pins using binary rotary switches.

External hardware

- Rotaty switches w/ 16 positions.
- Speakers or TinyTapeout Audio PMOD if you want to hear `out7`.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	param a bit 0/3	track #0 PWM output	param c bit 0/3
1	param a bit 1/3	N/A	param c bit 1/3
2	param a bit 2/3	N/A	param c bit 2/3
3	param a bit 3/3	track #3 PWM output	param c bit 3/3
4	param b bit 0/3	N/A	param d bit 0/3
5	param b bit 1/3	N/A	param d bit 1/3
6	param b bit 2/3	N/A	param d bit 2/3
7	param b bit 3/3	track #7 PWM output	param d bit 3/3

CEJMU WSPR Encoder

by **Matthias Jung DL9MJ** & **Jonathan Hager DK7JH**

0654 | 56 MHz | HDL Project

`github.com/CEJMU/tt-WhisperEncoder`

"WSPR encoder that takes call, locator and power and outputs a PWM that can be used for transmitting the WSPR symbols."

How it works

Please see the documentation PDF in this repo that will be uploaded after the submission for more detailed info!

The design is programmed using UART. You may use the Python script in this repo. WSPR only needs the callsign, locator and transmit power, which is then turned into an array of 162 symbols. The transmission is achieved with a 4-FSK. The frequency for each symbol can also be programmed using UART.

How to test

Run the Python script to set the registers and then set start_transmission to 1. The chip will output sine and cosine with a PWM that can be turned into a proper sine/cosine by using a low-pass filter. The iq output signals should be used together with the TinyWhisper analog project on this shuttle or alternatively by providing the filters extra.

External hardware

You somehow need to convert the serial data produced by Python to UART. You may use a simple USB-UART bridge.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	UART rx (115200 baud)	encoding_valid (indicates that start_transmission may be pressed)	ix (output)
1	start_transmission	cos_pwm	current symbol[0]
2	unused	cos_pwm_neg	current symbol[1]
3	unused	sin_pwm	unused
4	unused	sin_pwm_neg	unused
5	unused	i	unused

#	Input	Output	Bidirectional
6	unused	q	unused
7	unused	ix	unused

JKU Matt Venn workshop simple project

by **Martin Kulovits**

0672 | 10 Hz | Wokwi Project

github.com/DjLemonfresh/WorkshopJKU
wokwi.com/projects/442977682425413633

"flipflop and shift register"

How it works

A simple Flipflop from NOR Gates and A 3 bit shift register from D-Flipflop

How to test

Use the pins to set/unset and checkt shift register output

External hardware

not necessary, simple logic analyser or LEDS will do

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	SET	Q	—
1	—	/Q	—
2	RESET	—	—
3	—	—	—
4	SHIFT/LOAD	—	—
5	IN0	Shift_out	—
6	IN1	—	—
7	IN3	—	—

Secret Password

by **Ivan**

0673 Wokwi Project

`github.com/Ichen-ai/tinyTapeoutTemplate`
`wokwi.com/projects/445163606906219521`

"Enter secret password to recieve output"

How it works

Secret password, enter correct combination to recieve output

How to test

Current password: 1234

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	1	1	—
1	2	2	—
2	3	3	—
3	4	4	—
4	—	5	—
5	—	6	—
6	—	7	—
7	—	8	—

Test

by **KV**

0674 Wokwi Project

`github.com/klausv001/Test`

`wokwi.com/projects/442984695522496513`

"Test für Submission"

How it works

only as test

How to test

test usage

External hardware

Test

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in1	—	—
1	in2	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Four-bit adder tiny tapeout

by **Aditya Choudhuri** & **Duke Liu**

0675 Wokwi Project

github.com/TimLiu2019/TinyTapeoutGDSVerison1

wokwi.com/projects/445163800203964417

"Outputs the number of inputs that are on."

How it works

The inputs correspond to bits, and it's combined with half adders and full adders to produce a four-bit added output.

How to test

Set the inputs on or off and check if the output represents their sum in binary.

Where out0 is LSB, and out2 is the MSB.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	inputA	The 2^0 output of the sum of how many of the inputs are on, represented as a binary number.	—
1	inputB	The 2^1 output of the sum of how many of the inputs are on, represented as a binary number.	—
2	inputC	The 2^2 output of the sum of how many of the inputs are on, represented as a binary number.	—
3	inputD	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

4 bit adder

by **J Parker Jones**

676 Wokwi Project

`github.com/ParkerJones567/tinytapeout2025`
`wokwi.com/projects/442987174502478849`

"its a 4 bit adder"

How it works

2 4 bit inputs, 1 5 bit output ## How to test

4 bit adder

External hardware

none

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a3	—	—
1	a2	—	—
2	a1	—	—
3	a0	c4	—
4	b3	c3	—
5	b2	c2	—
6	b1	c1	—
7	b0	c0	—

Tiny Takeout Test Gates

by **Ms Harris**

0677 Wokwi Project

`github.com/MsJAHarris/tinyTapeout`

`wokwi.com/projects/445175272109059073`

"test logic gates"

How it works

This is a basic T- flip flop - input from 2 and gates to 2 Xor's

How to test

pin 6 is straight connection to LED to check power pin 0,1 are output input is 2 and 3 input is 1 and 7

External hardware

List external hardware buttons, LEDs, resistors

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	ANDA	NOR1	—
1	ANDC	NOR2	—
2	ANDB	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	ToLEDCtr	—
7	ANDD	—	—

TinyTapeoutTest

by **HILG**

0678 | 10 kHz | Wokwi Project

`github.com/muagln/TinyTapeout`
`wokwi.com/projects/442987226036856833`

"Test"

How it works

some words

How to test

some words

External hardware

some words

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output nand	—
2	—	output ok	—
3	—	output clock div 16	—
4	—	output clock div 12	—
5	—	—	—
6	—	—	—
7	—	—	—

not a dumpster fire

by **liam eddi**

0679 Wokwi Project

`github.com/liamkmcm-ux/vct-game-changers-`
`wokwi.com/projects/445163636148924417`

" numbers"

How it works

you flick the switch and the light one of the lights will turns on

How to test

flip all the switch's and the entire seven segment display will turn on

External hardware

LED display, DIP switch 8

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input or1:A Input or1:B	output pin:A	—
1	Input or3:A Input or3:B	output pin:B	—
2	Input or4:A Input or4:B	output pin:C	—
3	Input or5:A Input or5:B	output pin:D	—
4	Input or6:A Input or6:B	output pin:E	—
5	Input or7:A Input or7:B	output pin:F	—
6	Input or8:A Input or8:B	output pin:G	—
7	Input or9:A Input or9:B	output pin:H	—

Digital_gate

by **Fatemeh**

0680 | 100 kHz | Wokwi Project

github.com/abbassif/TT_Fatemeh

wokwi.com/projects/442977548950021121

"test project"

How it works

AND, NAND and OR Gates are connected to input a and b. 16 flip flops are used to divide the clock speed with 2 outputs, one at 12th flip flop and one at 16th. Explain how your project works

How to test

set the input and outputs match with the expected result: Explain how to use your project

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output nand	—
2	—	output or	—
3	—	output clk div 16	—
4	—	output clk div 12	—
5	—	—	—
6	—	—	—
7	—	—	—

fdfs

by **afsdf**

0681 | 1 GHz | Wokwi Project

github.com/simonwaid/tt2

wokwi.com/projects/442978871257096193

"Synchronous Counter"

How it works

3 Bit synchronous counter

How to test

Attach a logic a analyzer

External hardware

Clock, logic analyzer

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	obit0	—
1	—	obit1	—
2	—	obit2	—
3	ibit3	obit3	—
4	ibit4	obit4	—
5	ibit5	obit5	—
6	ibit6	obit6	—
7	ibit7	obit7	—

in-81004-1 silicon

by **Torgeir Trøite**

0682 HDL Project

`github.com/torgeir-a-troite/tt_um_in-81004-1-silicon`

"Silicon Design Evolution"

How it works

Doing a gradual evolutuion of the template to perform more and more tasks

How to test

Well, we will see how to test this

External hardware

No external hardware needed

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	i_data_0	o_data_0	—
1	i_data_1	o_data_1	—
2	i_data_2	o_data_2	—
3	i_data_3	o_data_3	—
4	i_data_4	o_data_4	—
5	i_data_5	o_data_5	—
6	i_data_6	o_data_6	—
7	i_data_7	o_data_7	—

Tiny Tapeout Test Gates

by **Rafael Viertlmayr**

0683 Wokwi Project

`github.com/rafael-vy/Wokwi-Template`

`wokwi.com/projects/442977503814034433`

"???"

How it works

Explain how your project works AND and OR and NOT are connected and if you press buttons the Lights change ## How to test

Explain how to use your project ???

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0 and	—
1	IN1	OUT1 and	—
2	IN2	OUT2 or	—
3	IN3	OUT3 or	—
4	IN4	OUT4 not	—
5	IN5	OUT5 nand	—
6	IN6	OUT6 nand	—
7	IN7	OUT7 not	—

Arjun's First Template Copy

by **Arjun**

0684 | 10 kHz | Wokwi Project

github.com/ArjunG255/First_GDS

wokwi.com/projects/442982637762839553

"First Handson"

How it works

Toggle the input pins to create a value

How to test

Toggle the input pins to create a value

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	input b	output b	—
2	input c	output c	—
3	input d	output d	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

QEI (Quadrature Encoder Interface)

by **Jake Drew**

#685 HDL Project

`github.com/jakedrew/ttsky-verilog-qei`

"Decodes A/B quadrature input. Outputs last step direction and 15-bit count."

How it works

Quadrature x4 decoder with 15-bit up/down count and last-step direction.

- **Inputs:** `A=ui_in[0]`, `B=ui_in[1]` (two-stage sync to `clk`)
- **Decoding:** `00→01→11→10→00` = +1 per edge; reverse = −1
- **Outputs:**
  - `uo_out[7]` — DIR (1 = forward, 0 = backward)
  - `uo_out[6:0]` — COUNT `[6:0]`
  - `uio_out[7:0]` — COUNT `[14:7]`

Notes:

- DIR reports the **last** step direction; it is not a signed count.
- The exposed count bits give you 15 LSBs across `uio_out` and `uo_out[6:0]`.

How to test

On hardware

1. Drive encoder A/B into `ui[0]` and `ui[1]`.
2. Read:
   - `uo[7]` → DIR (1 = forward, 0 = backward)
   - `uo[6:0]` → `count[6:0]`
   - `uio[7:0]` → `count[14:7]`
3. Turn the encoder forward: count increases and `uo[7]=1`. Reverse: count decreases and `uo[7]=0`.

Simulation (local)

- Run: `make -C test clean test`
- View waveform: `gtkwave test/tb.vcd`

External hardware

- If your encoder is **single-ended**, connect A/B directly (3.3V logic).
- If your encoder is **differential** (A/A', B/B'), use an RS-422/line-receiver to convert to 3.3V CMOS before `ui[0:1]`.
- Index (Z) is not used in this design.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	A	COUNT0	COUNT7
1	B	COUNT1	COUNT8
2	—	COUNT2	COUNT9
3	—	COUNT3	COUNT10
4	—	COUNT4	COUNT11
5	—	COUNT5	COUNT12
6	—	COUNT6	COUNT13
7	—	DIR	COUNT14

TestGDS

by **Daniel**

0686 | 10 kHz | Wokwi Project

`github.com/danicerhol/tinyTapeOut`

`wokwi.com/projects/442979151350678529`

"AND"

How it works

It is an AND.

How to test

Just run.

And wait.

External hardware

adedaede

S List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in_a	out	—
1	in_b	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny Turing Machine

by **Christoph Menz**

0687 HDL Project

`github.com/chrimenz/tiny-turing-machine`

> *"A tiny Turing machine with configurable state transition table"*

How it works

This project implements a **Turing Machine** on an ASIC with the following features:

Core Components

- **8-bit tape** with configurable wrap-around mode
- **4 states** (S0-S3) with **unary alphabet** (Blank/0 and 1)
- **State Transition Table (STT)** with 4 banks:
  - Bank 0: **Unary Incrementer** - Adds one "1" to a string of ones
  - Bank 1: **Bit Flipper** - Inverts all bits from left to right
  - Bank 2: **Wall-E** - Carries Bits from right side to left bound
  - Bank 3: **Custom** - User-configurable STT

Operating Modes

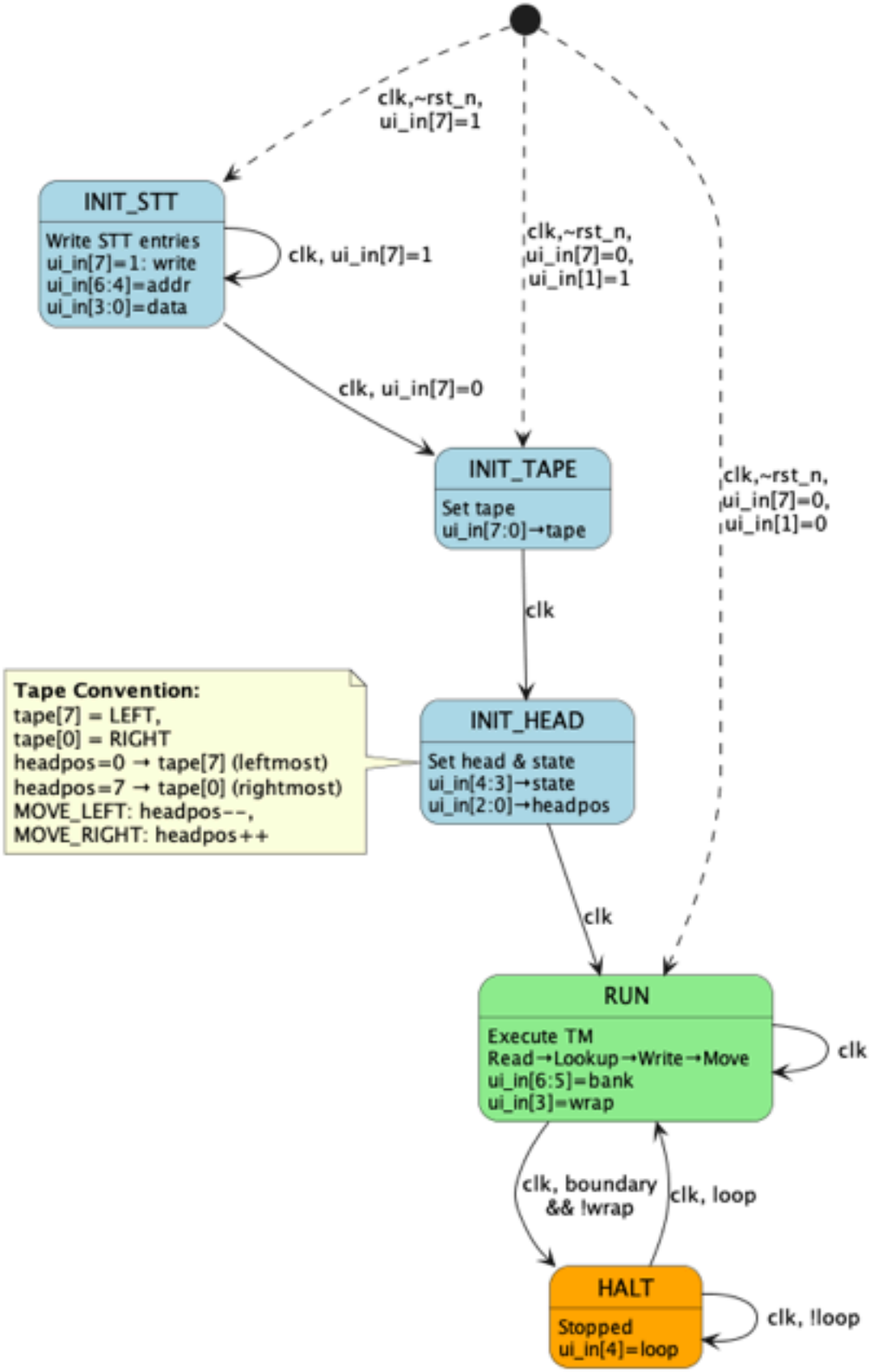


Figure 687.1: Operating Modes

The machine operates in 5 distinct modes:

- **INIT**: Initial state after reset, loading configuration
- **WRITE_TAPE**: Writing initial tape values
- **WRITE_HEAD**: Setting head position and initial state
- **RUN**: Executing the Turing Machine program
- **HALT**: Machine stopped at boundary (unless wrap mode enabled)

Output Routing

Two 8-bit outputs are available:

- **uo_out**: Normally shows tape, or status when swapped
- **uio_out**: Normally shows status, or tape when swapped

Status format (8 bits):

- `[7:5]`: Mode (0-4)
- `[4:3]`: Current state (0-3)
- `[2:0]`: Head position (0-7)

Head position: The status output can be switched to a (one hot) decoded head position output by setting `ui_in[0]=1`.

Swap control: `ui_in[2]=1` swaps the outputs.

How to test

Quick Test (Preset 0: Unary Incrementer)

1. **Reset with all inputs low**: `ui_in = 00000000`
   - Loads Bank 0 (Unary Incrementer) preset
   - Initial tape: `11100000` (three 1s on left)
   - Head at position 0 (leftmost)
   - State 0
2. **Clock slowly** (manual or low frequency)
   - Observe `uo_out` (tape) changing
   - Expected behavior:
     - Head moves right through the 1s
     - Finds first 0, writes 1
     - Result: `11110000` (four 1s)
     - Machine halts at right boundary
3. **Observe outputs**:
   - `uo_out`: Shows tape pattern (LEDs on bits 7-4)
   - `uio_out[7:5]`: Mode (should be 3=RUN, then 4=HALT)
   - `uio_out[2:0]`: Head position (counts 0→7)

Advanced Tests

Test Loop Mode (Auto-restart):

1. Reset: `ui_in = 00010000` (loop enabled, bit 4=1)
2. Clock continuously

3. Machine will halt, then automatically restart with initial tape
4. Observe repeating pattern

Test Wrap Mode (Endless tape):

1. Reset: `ui_in = 00001000` (wrap enabled, bit 3=1)
2. Clock continuously
3. Head wraps around at boundaries, never halts
4. Useful for Bank 1 (Bit Flipper) to see full tape processed

Test Custom STT:

1. Reset: `ui_in = 10000000` (manual mode, bit 7=1)
2. Write 8 STT entries:
   - Example: `ui_in = 10000011` writes `0011` (S0,1,R) to address 0
   - Increment address bits [6:4] for each entry
3. Exit config: `ui_in = 00000000`
4. Set tape: `ui_in = 10101010`
5. Next clock: Set head/state: `ui_in = 00000011` (state 0, position 3)
6. Run: `ui_in = 01100000` (Bank 3, no wrap/loop)

Test Output Swap:

1. Start any preset
2. Toggle `ui_in[2]` between 0 and 1
3. Observe tape ↔ status swapping between `uo_out` and `uio_out`

External hardware

For Tape Visualization:

- **LED PMOD** with 8 LEDs in a row
- **7-Segement-Display:** might also be useful but not as intuitive

For Status Display:

- **LED PMOD** with 8 LEDs in a row
- Display mode(0-4), head position (0-7) and current state (0-3)
- With switch `ui_in[0]=1` the one-hot decoded head position is shown

For Manual Control:

- **DIP Switch**
  - Connect to `ui_in`
  - Allows manual control of all settings:
    - Switch 7: Config/Auto mode
    - Switches 6-5: Bank selection
    - Switch 4: Loop enable
    - Switch 3: Wrap enable
    - Switch 2: Output swap

    - Switch 1: Manual tape init
    - Switch 0: Head position decode
  - Depending on mode allows also STT, Tape, Head and State configuration
- Clock button (for configuration and manual mode/state transistion)
- Reset button

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0_HEAD_DECODE	TP_ST_0	ST_TP_0
1	IN1_TAPE_INIT	TP_ST_1	ST_TP_1
2	IN2_OUTPUT_SWAP	TP_ST_2	ST_TP_2
3	IN3_WRAP_EN	TP_ST_3	ST_TP_3
4	IN4_LOOP_EN	TP_ST_4	ST_TP_4
5	IN5_BANK_SEL0	TP_ST_5	ST_TP_5
6	IN6_BANK_SEL1	TP_ST_6	ST_TP_6
7	IN7_CUSTOM_AUTO	TP_ST_7	ST_TP_7

TinyQV Risc-V SoC

by **Michael Bell**

0708 | 64 MHz | HDL Project

`github.com/Almas00222/ttsky25b-tinyQV`

"A Risc-V SoC for Tiny Tapeout"

How it works

TinyQV is a small Risc-V SoC, implementing the RV32EC instruction set plus the Zcb and Zicond extensions, with a couple of caveats:

- Addresses are 28-bits
- Program addresses are 24-bits
- gp is hardcoded to 0x1000400, tp is hardcoded to 0x8000000.

Instructions are read using QSPI from Flash, and a QSPI PSRAM is used for memory. The QSPI clock and data lines are shared between the flash and the RAM, so only one can be accessed simultaneously.

Code can only be executed from flash. Data can be read from flash and RAM, and written to RAM.

The SoC includes a UART and an SPI controller.

Address map

Address range	Device
0x0000000 - 0x0FFFFFF	Flash
0x1000000 - 0x17FFFFF	RAM A
0x1800000 - 0x1FFFFFF	RAM B
0x7FFFF00 - 0x7FFFFFF	Internal RAM (32 bytes, wrapped)
0x8000000 - 0x8000007	GPIO
0x8000010 - 0x800001F	UART
0x8000020 - 0x8000027	SPI
0x8000028 - 0x800002B	PWM
0x8000030 - 0x8000033	DEBUG
0x8000034 - 0x800003B	TIME
0x8000040 - 0x8000047	GAME

GPIO

Register	Address	Description
OUT	0x8000000 (W)	Control out0-7, if the corresponding bit in SEL is high
OUT	0x8000000 (R)	Reads the current state of out0-7
IN	0x8000004 (R)	Reads the current state of in0-7
SEL	0x800000C (R/W)	Bits 0-7 enable general purpose output on the corresponding bit on out0-7. Bit 8 enables PWM output on out7, bit 9 enables PWM output on io7.

UART

Register	Address	Description
DATA	0x8000010 (W)	Transmits the byte
DATA	0x8000010 (R)	Reads any received byte
STATUS	0x8000014 (R)	Bit 0 indicates whether the UART TX is busy, bytes should not be written to the data register while this bit is set. Bit 1 indicates whether a received byte is available to be read.

Debug UART (Transmit only)

Register	Address	Description
DATA	0x8000018 (W)	Transmits the byte
STATUS	0x800001C (R)	Bit 0 indicates whether the UART TX is busy, bytes should not be written to the data register while this bit is set.

SPI

Register	Address	Description
DATA	0x8000020 (W)	Transmits the byte in bits 7-0, bit 8 is set if this is the last byte of the transaction, bit 9 controls Data/Command on out3
DATA	0x8000020 (R)	Reads the last received byte
CONFIG	0x8000024 (W)	The low 4 bits set the clock divisor for the SPI clock to 2*(value + 1), bit 8 adds half a cycle to the read latency when set

STATUS	0x8000024 (R)	Bit 0 indicates whether the SPI is busy, bytes should not be written or read from the data register while this bit is set.

PWM

Register	Address	Description
LEVEL	0x8000028 (W)	Set the PWM output level (0-255)

DEBUG

See debug docs

TIME

Register	Address	Description
MTIME	0x8000034 (RW)	Get/set the 1MHz time count
MTIMECMP	0x8000038 (RW)	Get/set the time to trigger the timer interrupt

GAME

Register	Address	Description
Controller 1	0x80000040 (R)	Controller 1 state
Controller 2	0x80000044 (R)	Controller 2 state

Controller state is in the low 12 bits of the register, in order (MSB to LSB): b, y, select, start, up, down, left, right, a, x, l, r

How to test

Load an image into flash and then select the design.

Reset the design as follows:

- Set rst_n high and then low to ensure the design sees a falling edge of rst_n. The bidirectional IOs are all set to inputs while rst_n is low.
- Program the flash and leave flash in continuous read mode, and the PSRAMs in QPI mode
- Drive all the QSPI CS high and set SD1:SD0 to the read latency of the QSPI flash and PSRAM in cycles.
- Clock at least 8 times and stop with clock high
- Release all the QSPI lines
- Set rst_n high
- Set clock low
- Start clocking normally

Based on the observed latencies from tt3p5 testing, at the target 64MHz clock a read latency of 2 or 3 is likely required. The maximum supported latency is currently 3, but should get up to 5 to have a chance at running at faster clock speeds.

The above should all be handled by some MicroPython scripts for the RP2040 on the TT demo PC.

Build programs using the riscv32-unknown-elf toolchain and the tinyQV-sdk, some examples are here.

External hardware

The design is intended to be used with this QSPI PMOD on the bidirectional PMOD. This has a 16MB flash and 2 8MB RAMs.

The UART is on the correct pins to be used with the hardware UART on the RP2040 on the demo board.

The SPI controller is intended to make it easy to drive an ST7789 LCD display (more details to be added).

It may be useful to have buttons to use on the GPIO inputs.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Interrupt 0	UART TX	Flash CS
1	Interrupt 1	UART RTS	SD0
2	SPI MISO	SPI DC	SD1
3	1 MHz clock for time	SPI MOSI	SCK
4	Game controller latch	SPI CS	SD2
5	Game controller clock	SPI SCK	SD3
6	Game controller data	Debug UART TX	RAM A CS
7	UART RX	Debug signal / PWM	RAM B CS / PWM

2048 sliding tile puzzle game (VGA)

by **Uri Shaked**

0712 | 25.175 MHz | HDL Project

`github.com/urish/tt-2048-game`

"Slide numbered tiles on a grid to combine them to create a tile with the number 2048."

How it works

2048 is a single-player sliding tile puzzle video game. Your goal is to slide numbered tiles on a grid to combine them and create a tile with the number 2048. The game is won when a tile with the number 2048 appears on the board, hence the name of the game. The game is lost when the board is full and no more moves can be made.

The game is played on a 4x4 grid, with numbered tiles that slide when a player moves them using `ui_in` pins or using a SNES compatible controller along with the Gamepad Pmod.

The game starts with two tiles with the number 2 on the board. The player can move the tiles in four directions: up, down, left, and right. When the player moves the tiles in a direction, the tiles slide as far as they can in that direction until they hit the edge of the board or another tile. If two tiles with the same number collide, they merge into a single tile with the sum of the two numbers. The resulting tile cannot merge with another tile again in the same move.

How to test

Use the `ui_in` pins to move the tiles on the board:

`ui_in` pin	Direction
0	Up
1	Down
2	Left
3	Right

Or use a SNES compatible controller along with the Gamepad Pmod. The game will automatically detect the presence of the Pmod and switch to controller input mode.

After resetting the game, you will see a jumping "2048" animation on the screen. Press any of the `ui_in[3:0]` pins (or the gamepad buttons) to start

the game. The game will start with two tiles with the number 2 on the board. Use the `ui_in` pins (or the gamepad buttons) to move the tiles in the desired direction. The game will end when the board is full and no more moves can be made.

The game offers two color themes: modern and retro. You can switch between the two themes using the `select` button on the gamepad or by setting both `ui_in[4]` and `ui_in[5]` to `1`.

Setting `ui_in[7]` to `1` will enter unit test mode. In this mode, the game displays a colorful rectangle on the top of the screen, and accepts debug commands on the `uio` pins. Check out the test bench for more information.

External hardware

- TinyVGA Pmod
- Optional: Gamepad Pmod

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	btn_up	R1	debug_cmd
1	btn_down	G1	debug_cmd
2	btn_left	B1	debug_cmd
3	btn_right	VSync	debug_cmd
4	gamepad_latch	R0	debug_data
5	gamepad_clk	G0	debug_data
6	gamepad_data	B0	debug_data
7	debug_mode	HSync	debug_data

TV-B-Gone-EU (ROM Macro variant)

by **Embelon**

0716 | 8 MHz | HDL Project

`github.com/smunaut/ttsky25a-tv-b-gone-eu`

"TV-B-Gone shuts down many TVs by sending IR codes"

How it works

Project drives IR LEDs to send series of TV codes for different protocols used in EU to shut down TVs remotely. Can work in one of two modes:

- sending series of TV codes in a loop (when Loop_Forever input is driven high)
- sending series of TV codes only once, after activation of Start_Button_N input (when Loop_Forever input is driven low)

There are two variants on this shuttle. This one uses a few ROM macros to store the codes to send instead of relying on yosys to generate combinatorial logic to store the codes. This drastically reduces the project size but those macros are not silicon-proven at the time of this tapeout.

How to test

For proper work, module needs 8MHz clock (to be generated by RPi Pico) External IR LED should be connected to IR_LED_OUT or IR_LED_OUT_N through transistor to increase driving current. For NPN transistor, IR_LED_OUT should be used. For PNP transistor, IR_LED_OUT_N should be used. There are two status outputs (active low) that can drive LEDs: ACTIVE and FAILURE. To setup module for sending TV codes in a loop, Loop_Forever input should be driven high. To make it sending only one serie of TV codes triggered by Start_Button_N input (active low), Loop_Forever should be driven low.

External hardware

External IR LED is needed with driving transistor. External button can be connected to Start_Button_N input (driving pin to ground), with pull up resistor to 3V3. Optional diodes can be connected to ACTIVE_LED_N and FAILURE_LED_N, those are active low so should be connected via resistors to 3V3.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Start_Button_N	IR_LED_OUT	—
1	Loop_Forever	IR_LED_OUT_N	—
2	—	ACTIVE_LED_N	—
3	—	FAILURE_LED_N	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Cynergy4MIE SPI Slave Component

by **Sergejs Umnovs**

0718 | 20 MHz | HDL Project

`github.com/Entritarus/tt_sergejsumnovs_spi_slave`

"An SPI slave demo component for the Cynergy4MIE project (and also my master's thesis). It has GPIO, LFSR and PWM."

How it works

This projects consists of an SPI slave and other components. The SPI slave accepts an 8-bit address and an 8-bit data value. Write transactions always have bit 7 of the address set. Write transactions always have bit 7 of the address cleared. There should be a pause between address and data transactions of at least 3 system clock cycles. The project includes a register map of 21 registers, GPIO module, LFSR with serial output, timer module with PWM and capture on rising and falling edges.

Example:

Read transaction. Read data from timer capture register 1.

```
|00001111|pause|11101001|
 address          data
```

Write transaction. Enable PWM and start the timer.

```
|10000001|pause|00000101|
 address          data
```

How to test

Simulation

This project uses cocotb and Icarus Verilog. Even though it is a VHDL project, it was transpiled using GHDL, so you can use Verilog simulation tools. Just run `make` in the `test` directory.

Register map

Address	Description
0x00	Timer control
0x01	Timer top
0x02	Timer clock divisor
0x03	Timer capture control

0x04	GPIO read data
0x05	GPIO write data
0x06	GPIO direction
0x07	LFSR polynome
0x08	LFSR control
0x09	LFSR state input
0x0a	PWM 0 top
0x0b	PWM 1 top
0x0c	PWM 2 top
0x0d	PWM 3 top
0x0e	Capture 0 register
0x0f	Capture 1 register
0x10	Capture 2 register
0x11	Capture 3 register
0x12	Capture 4 register
0x13	Capture 5 register
0x14	Capture 6 register
0x15	Capture 7 register

Description of each register

All registers are 0 after reset by default.

Timer control (0x00)

You can enable or disable features of the timer through this register.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	Unused	Unused	Timer overflow control	Capture control	PWM control	Direction control	Timer enable
—	—	—	—	—	—	—	—
—	—	—	'1' - reset on top	'1' - enable capture	'1' - enable PWM	'1' - count down	'1' - enable timer
—	—	—	'0' - reset on overflow	'0' - disable capture	'0' - disable PWM	'0' - count up	'0' - disable timer

Timer top (0x01)

Timer top value. If bit 4 of the timer control register is set, then timer will count up to this value and roll back to 0.

Timer clock divisor (0x02)

Divides the clock of the timer by (value + 1). By default it is 0, so divisor is 1.

Timer capture control (0x03)

Controls the edges at which the time of events will be saved

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Capture 7 edge	Capture 6 edge	Capture 5 edge	Capture 4 edge	Capture 3 edge	Capture 2 edge	Capture 1 edge	Capture 0 edge
‘1’ - rising edge	‘1’ - rising edge	‘1’ - rising edge	‘1’ - rising edge	‘1’ - rising edge	‘1’ - rising edge	‘1’ - rising edge	‘1’ - rising edge
‘0’ - falling edge	‘0’ - falling edge	‘0’ - falling edge	‘0’ - falling edge	‘0’ - falling edge	‘0’ - falling edge	‘0’ - falling edge	‘0’ - falling edge

GPIO read data (0x04)

GPIO input register.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	Unused	Unused	Unused	GPIO pin 3	GPIO pin 2	GPIO pin 1	GPIO pin 0
—	—	—	—				
—	—	—	—	‘1’ - pin high	‘1’ - pin high	‘1’ - pin high	‘1’ - pin high
—	—	—	—	‘0’ - pin low	‘0’ - pin low	‘0’ - pin low	‘0’ - pin low

GPIO write data (0x05)

GPIO output register.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	Unused	Unused	Unused	GPIO pin 3	GPIO pin 2	GPIO pin 1	GPIO pin 0
—	—	—	—				

—	—	—	—	‘1’ - set high	‘1’ - set high	‘1’ - set high	‘1’ - set high
—	—	—	—	‘0’ - set low	‘0’ - set low	‘0’ - set low	‘0’ - set low

GPIO direction (0x06)

GPIO direction register. GPIO is connected to uio port of the Tiny Tapeout IC. By default all pins are inputs.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	Unused	Unused	Unused	GPIO pin 3	GPIO pin 2	GPIO pin 1	GPIO pin 0
—	—	—	—	—	—	—	—
—	—	—	—	‘1’ - output	‘1’ - output	‘1’ - output	‘1’ - output
—	—	—	—	‘0’ - input	‘0’ - input	‘0’ - input	‘0’ - input

LFSR polynome (0x07)

LFSR polynome register. Max L polynome for an 8 bit LFSR is 0x63.

LFSR control (0x08)

This register controls the LFSR.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	Unused	Unused	Unused	Unused	Unused	Load enable	LFSR enable
—	—	—	—	—	—	—	—
—	—	—	—	—	—	‘1’ - load inital value	‘1’ - enable LFSR
—	—	—	—	—	—	‘0’ - don’t load inital value	‘0’ - disable LFSR

LFSR state input (0x09)

You can set the initial state through this register.

PWM tops (0x0a – 0x0d)
You can set tops for the PWM or use these as general purpose registers.

Capture registers (0x0e – 0x15)
These registers hold the time of event edges happened at capture pins.

How to use

How to use the Timer.

- Starting the timer
    1. Set the clock divisor through the register 0x02 (optional).
    2. Set the top value through the register 0x01 (optional).
    3. Configure bits 4 and 1 of the register 0x00 as you wish.
    4. Start the timer by setting bit 0 of the register 0x00.
- Using PWM
    1. Set PWM tops (registers 0x0a – 0x0d).
    2. Set bit 2 of the register 0x00
    3. Start the timer.
- Using event capturing
    1. Set bit 3 of the register 0x00
    2. Configure capture edges in the register 0x03
    3. Read captured times from registers 0x0e – 0x15

How to use the GPIO.

1. Configure the register 0x06 as you wish.
2. Use register 0x05 to set digital output.
3. Use register 0x05 to read digital input.

How to use the LFSR.

1. Load the initial value into register 0x09.
2. Load the polynome into register 0x09 (I suggest 0x63, because it is a max L polynome).
3. Set set bits 0 and 1 of the register 0x08 (enable the LFSR and enable loading).
4. Clear bit 1 of the register 0x08 (disable loading).
5. The LFSR will how cycle through all states resulting from the loaded polynome.

External hardware

The pins of the SPI slave are assigned so that you could use PMOD SPI master. Any other USB-to-SPI master should suffice. You can also use LEDs and buttons for GPIO.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Capture 0	CS	PWM 0
1	Capture 1	MOSI	PWM 1
2	Capture 2	MISO	PWM 2
3	Capture 3	SCK	PWM 3
4	Capture 4	GPIO 0	LFSR
5	Capture 5	GPIO 1	NC
6	Capture 6	GPIO 2	NC
7	Capture 7	GPIO 3	NC

Tiny Tapeout Template Copy

by **Echocrft**

0736 | 1 Hz | Wokwi Project

github.com/Echocrft/7_Segment-Anzeige
wokwi.com/projects/442987801460891649

"7 Sgment Anzeige in einzelbetaetigung"

How it works

If you put a High on the right on one of the Inputs

How to test

just use it for 7 segment

External hardware

7 Segmente Anzeige

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	/
1	IN1	OUT1	/
2	IN2	OUT3	/
3	IN3	OUT4	/
4	IN4	OUT4	/
5	IN5	OUT5	/
6	IN6	OUT6	/
7	IN7	OUT7	/

VGA clock render – Designed by Matt Venn. Illustrated by Máximo Balestrini.

Tiny Tapeout Template Project

by **Claudia Rossi**

0737 | 10 kHz | Wokwi Project

github.com/claudiajrossi08-debug/Hardware-Hack-day
wokwi.com/projects/445172222101072897

"create the letter f i guess"

How it works

I don't know

How to test

I don't know

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	—	—
2	—	—	—
3	—	—	—
4	input c	output c	—
5	input d	output d	—
6	input e	output e	—
7	input f	output f	—

TinyBF

by **René Hahn**

0738 | 50 MHz | HDL Project

github.com/renehahn/TinyBF

"A tiny Brainfuck interpreter in Verilog"

How it works

TinyBF is a complete hardware implementation of a Brainfuck interpreter designed to fit within the constraints of a single Tiny Tapeout tile. The design is a fully functional CPU with integrated UART I/O, executing a hardcoded demonstration program that showcases all major Brainfuck operations.

Architecture

The system consists of five main components:

1. **Control Unit** - An 11-state finite state machine (FSM) that serves as the CPU core. It fetches instructions, decodes them, and orchestrates all operations including memory access and I/O. The FSM uses binary encoding for optimal gate count.
2. **Program Memory (ROM)** - 16×8-bit read-only memory containing a fixed demonstration program. Each instruction is encoded as an 8-bit value: 3 bits for the opcode and 5 bits for a signed argument (enabling optimizations like `+5` instead of five separate `+` instructions).
3. **Tape Memory (RAM)** - 8×8-bit synchronous RAM representing the Brainfuck data tape with 8 cells. This is the working memory where Brainfuck programs manipulate data.
4. **UART Subsystem** - Includes both transmitter and receiver modules operating at 38400 baud. The UART handles Brainfuck's I/O commands: `.` (output) sends bytes via TX, and `,` (input) receives bytes via RX. A baud rate generator provides precise timing.
5. **Reset Synchronizer** - Ensures clean reset propagation across clock domains to prevent metastability issues.

Hardcoded Program

The ROM contains a UART-based case converter that demonstrates input, arithmetic, loops, and output (16 instructions):

```
Address | Instruction | Description
--------|-------------|------------
0       | ,           | Read character from UART into cell[0]
1       | >           | Move to cell[1]
```

```
2       | +10          | cell[1] = 10 (newline character)
3       | <            | Back to cell[0]
4       | [ +6         | Jump forward 6 if cell[0] == 0 (to address
10)
5       | -15          | Subtract 15 from cell[0]
6       | -15          | Subtract 15 from cell[0] (total -30)
7       | -2           | Subtract 2 from cell[0] (total -32)
8       | .            | Output cell[0] via UART
9       | ,            | Read next character
10      | ] -6         | Jump back -6 if cell[0] != 0 (to address 4)
11      | >            | Move to cell[1]
12      | .            | Output newline (cell[1] = 10)
13-15   | HALT         | End of program
```

Program behavior: Reads ASCII characters from UART RX (`,` command). For each non-null character, subtracts 32 (via −15, −15, −2) to convert lowercase to uppercase, then outputs via UART TX (`.` command). On null terminator (0x00), exits loop and outputs newline (0x0A).

Example: Input `"abc"` → Output `"ABC\n"`

- ‘a’ (0x61 = 97) → −32 → ‘A’ (0x41 = 65)
- ‘b’ (0x62 = 98) → −32 → ‘B’ (0x42 = 66)
- ‘c’ (0x63 = 99) → −32 → ‘C’ (0x43 = 67)
- null (0x00) → exit loop → output ‘’ (0x0A)

Instruction Set

TinyBF implements all eight Brainfuck commands plus an optimized instruction encoding:

Opcode	Command	Description	Argument
000	>	Increment data pointer	Signed offset (-16 to +15)
001	<	Decrement data pointer	Signed offset (-16 to +15)
010	+	Increment cell value	Amount (0 to 31)
011	−	Decrement cell value	Amount (0 to 31)
100	.	Output cell via UART	N/A
101	,	Input from UART to cell	N/A
110	[	Jump forward if zero	PC-relative offset
111	]	Jump backward if non-zero	PC-relative offset

Special: The instruction `0x00` acts as a HALT, cleanly stopping program execution.

The 5-bit argument field enables compact encoding of common patterns. For example, incrementing a cell by 5 requires just one instruction instead of five, reducing both program size and execution time.

Memory Timing

The tape memory uses synchronous reads with 1-cycle latency. The control unit explicitly manages this through dedicated wait states: when initiating a read, the FSM transitions through a WAIT state before the data becomes valid, ensuring correct synchronization without combinational paths through memory.

The program ROM provides registered outputs with 1-cycle latency, maintaining timing consistency across the design.

How to test

Pin Configuration

Inputs:

- `ui[0]` - UART RX: Serial input for Brainfuck `,` command (38400 baud, 8N1)
- `ui[1]` - START: Pulse high to begin program execution from address 0
- `ui[2]` - HALT: Pulse high to stop execution immediately

Outputs:

- `uo[0]` - UART TX: Serial output for Brainfuck `.` command (38400 baud, 8N1)
- `uo[1]` - CPU_BUSY: High when CPU is actively executing
- `uo[5:2]` - Program counter bits [3:0]: Current instruction address (0-15)
- `uo[7:6]` - Cell value bits [6:5]: Upper 2 bits of current cell

Bidirectional (configured as outputs):

- `uio[2:0]` - Data pointer [2:0]: Current tape position (0-7)
- `uio[7:3]` - Cell value bits [4:0]: Lower 5 bits of current cell

Testing Procedure

1. **Power-up and Reset**: Apply power and ensure `rst_n` is asserted low, then released high. The CPU will enter IDLE state. The ROM program is immediately available (no initialization delay).

2. **Start Execution**: Pulse the START input (`ui[1]`) high for at least one clock cycle. The CPU will begin executing the hardcoded ROM program.

3. **Expected Behavior**:

   - **UART Input**: Program waits for character input on UART RX
   - **Case conversion**: Converts lowercase ASCII to uppercase (subtracts 32)

- **UART output**: Outputs converted characters via UART TX
- **Loop termination**: Exits on null character (0x00), outputs newline (0x0A)

4. **Monitor Execution**:
   - Watch `CPU_BUSY` (`uo[1]`) to see when the program is running
   - Observe the program counter on `uo[5:2]` cycling through addresses 0-15
   - Monitor the data pointer on `uio[2:0]` switching between cells 0 and 1
   - Track cell values on `{uo[7:6], uio[7:3]}` showing ASCII codes during conversion
5. **UART Communication**:
   - Connect a UART terminal to `ui[0]` (RX) and `uo[0]` (TX) at 38400 baud, 8N1 format
   - Send lowercase characters like `"abc"` followed by null terminator (0x00)
   - You should receive uppercase output `"ABC\n"`
   - The program demonstrates interactive UART I/O with both `,` (input) and `.` (output) commands
6. **Program Restart**: To run the program again, pulse START (`ui[1]`) or reset the system.

External hardware

Required:

- UART controller for serial communication
  - Connect TinyBF's TX (`uo[0]`) to converter's RX
  - Connect TinyBF's RX (`ui[0]`) to converter's TX
  - Configure terminal software for 38400 baud, 8 data bits, no parity, 1 stop bit (8N1)

Optional:

- Logic analyzer or oscilloscope to monitor debug outputs (program counter, data pointer, cell values)
- Push button for manual START/HALT control

Note: The program is hardcoded in ROM and cannot be changed without re-synthesizing the design. This reduces die area but makes the design a demonstration platform rather than a general-purpose Brainfuck interpreter.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	UART_RX	UART_TX	DP[0]
1	START	CPU_BUSY	DP[1]
2	HALT	PC[0]	DP[2]
3	—	PC[1]	CELL[0]
4	—	PC[2]	CELL[1]
5	—	PC[3]	CELL[2]
6	—	CELL[5]	CELL[3]
7	—	CELL[6]	CELL[4]

Tiny Tapeout Binary to 7SD

by **Ayama Inque**

0739 Wokwi Project

github.com/AInque/urban-meme

wokwi.com/projects/445163763917969409

"Why do you need my discord lol?"

How it works

-# When none of the inputs are 1, the 7 segment display will have a 0 on it.

How to test

Change the values around. They'll only do something when all of them are off.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output 1	—
1	input b	output 2	—
2	input c	output 3	—
3	input d	output 4	—
4	—	output 5	—
5	—	output 6	—
6	—	output 7	—
7	—	—	—

WokWi Test

by **Richard Feichtinger**

0740 Wokwi Project

`github.com/rfeichtinger/Richard_Tiny_Tapeout`
`wokwi.com/projects/442977585335625729`

"Test Design"

How it works

test

How to test

test it

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	input b	output b	—
2	input c	output c	—
3	input d	output d	—
4	—	output e	—
5	—	output f	—
6	input g	output g	—
7	input h	output h	—

Anushka and Tina Wokwi

by **Anushka** & **Tina**

0741 Wokwi Project

github.com/tinachen1111/t-and-a

wokwi.com/projects/445172941748296705

"The frist two stwitches activate the the circuit to display one and all other pins display each segemnt of the light."

How it works

Our project can display any numbers by turning on the correct switches but we added a shortcut to display the number one.

External hardware

LED Display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	candle	—
1	input b	apple	—
2	input c	bank	—
3	inpput d	dark	—
4	input e	entire	—
5	input f	finger	—
6	input g	grass	—
7	input h	horse	—

Wokwi Template

by **Jonas Lehner**

0742 Wokwi Project

github.com/LehnerJonas/Tapeout-Workshop

wokwi.com/projects/442977456053457921

"number output"

How it works

Explain how your project works

If you switch the Lever 1 to 8 the 7-segment-LED will put out the number with the lever reset you will reset the led's. If you activate more than one lever there wont happen anything

How to test

Explain how to use your project If you switch the Lever 1 to 8 the 7-segment-LED will put out the number with the lever reset you will reset the led's. If you activate more than one lever there wont happen anything

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	reset button	and inverter	—
1	1 lever	and and or	—
2	2 lever	and and or	—
3	3 lever	and and or	—
4	4 lever	and and or	—
5	5 lever	and and or	—
6	6 lever	and and or	—
7	7 lever	and and xor	—

Tahmid and Sunny's First Wokwi Design

by **Tahmid Monir** & **Sunny Lee**

0743 | 10 kHz | Wokwi Project

github.com/Tahmid-M10/Tahmid-Repository-with-Sunny-Lee

wokwi.com/projects/445163964149915649

"Test some logic gates and a clock divider."

How it works

And gates are connected to input a and b. The gates are connected to a digital clock, which gets one input when both switches (1, 2) are on.

How to test

Toggle the 1st and 2nd switches to see how the and gate works.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	ouput and	—
1	input b	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

ShilpaTinytapeout

by **ShilpaShyam**

0744 | 10 kHz | Wokwi Project

`github.com/Shilpa-777/ShilpaWorkshop`

`wokwi.com/projects/442979336364610561`

"Test some logic gates and a clock divider"

How it works

AND, NAND and OR GATES are connected to the input a and b

16 flip flops are used to divide the clock speed with 2 outputs one at the 12th flip flop and one at the 16th ## How to test

Set the inputs and check the outputs match with the expected results

External hardware

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output 0	—
1	input b	output 1	—
2	—	output 2	—
3	—	output 3	—
4	—	output 4	—
5	—	output 5	—
6	—	output 6	—
7	—	output 7	—

RHYTHM LOOPER and RANDOMIZER 9000

by **william kennedy**

0745 | 2 Hz | Wokwi Project

`github.com/zumdar/tinytapeout_random_and_sequencer`

`wokwi.com/projects/446363834407809025`

"a simple rhythm sequencer looper and a rAnDomiZeR. Uses a ring counter and LFSR"

This is a little rhythm looper machine and randomizer machine.

The idea is for it to be used to make music or art installations!

How it works

There are 8 cascading D-type flip flops that create a ring counter OR a LFSR (Linear Feedback Shift Register), which will generate a psuedo-random bitstream (and casade that down the 8 flip flops).

Inputs

- SW1 selects between the LOOPER and the RANDOMIZER:
  - LOW = LOOPER (Ring Counter)
  - HIGH = RANDOMIZER (Linear Feedback Shift Register or LFSR) NOTE!: you need some bits in the sequencer for the randomizer to work. (A "seed")
- SW2 = LOOPER INPUT
  - tap on and off to create new ring counter pattern
- SW3 = CLEAR
  - HIGH = hold to clear the ring counter pattern

Outputs

- There are 8 outputs, each one corresponding to an output of a flip flop on the ring counter

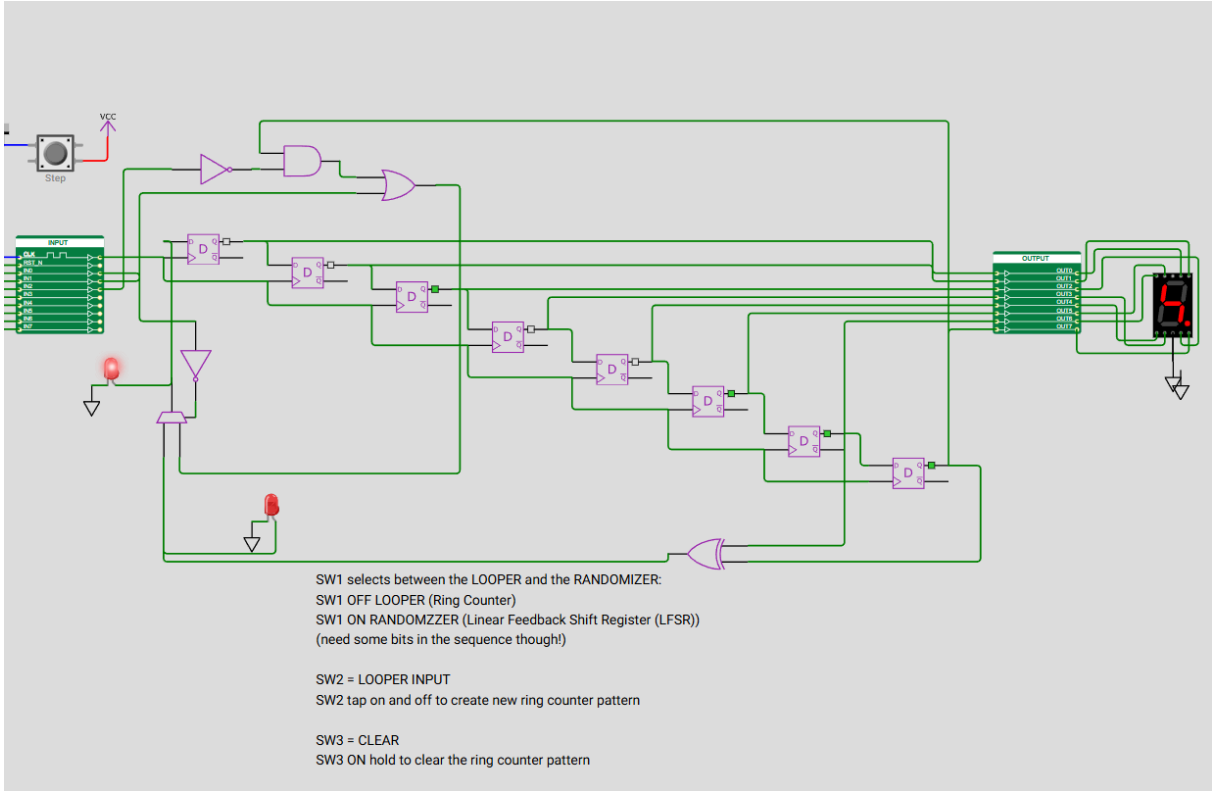


Figure 745.1: wokwi screen shot

How to test

- Select the project on the chip!
- All switches OFF
- HOLD SW3 to clear any resididual pattern
- hold SW2 for 1 second, release
- should have a short looping pattern on any of the OUTPUTS
- Switch SW1 to HIGH
- Now the pattern should be randomized

External hardware

Tiny Tapeout GPIO specs

Inputs

- NEED 3 switches
  - pulldown to GND
  - LOW = 0V
  - HIGH = 3.3

Outputs

- Drive strength (source/sink) 4 mA
- Outputs could hook up to a motor with MOSFET output circuitry

- Outputs could hook up trigger sounds, like drum samples, short one shot analog drums, gate oscillator sounds etc

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	switch between OFF = looper (ring counter) and ON = randomizer (LFSR)	OUTPUT1	—
1	toggle ON and OFF to make a new loop on the ring counter	OUTPUT2	—
2	hold to clear pattern (ON), or leave (OFF) to input your own rhythm loop using SW1	OUTPUT3	—
3	—	OUTPUT4	—
4	—	OUTPUT5	—
5	—	OUTPUT6	—
6	—	OUTPUT7	—
7	—	OUTPUT8	—

PWM_selector

by **Delia Hurtado**, **Ruben Velasquez**, **Gerardo Aerreola** & **Julio Ramirez**

0746 | 10 MHz | HDL Project

`github.com/DeliaMariaHC/PWM_selector`

> *"PWM_module generates a PWM (Pulse-Width Modulation) signal using a counter and a comparator. The output duty-cycle resolution is 12.5%."*

How it works

PWM_module generates a PWM (Pulse-Width Modulation) signal using a counter and a comparator. The output duty-cycle resolution is 12.5%. • Inputs: clock=In[4] , enable=In[3], speed[2:0]=In[2:0]. Output: PWM=Out[0]. • In[2:0] (speed[2:0]) determines the Duty Cicle at Output PWM signal • When enable=1, counter and output go to 0. • When enable=0, a 5-bit counter (0...31) increments on each rising clock edge; • Because the counter is 5 bits wide, the period of the PWM output signal is determined by 32 clock periods applied to the clock (In[4]). • A combinational block maps speed to a target pulse width (width) in steps of 4: 0, 4, 8, 12, 16, 20, 24, 28, which correspond to 0, 12.5, 25, 37.5, 50, 62.5, 75, 87.5% over a 32-cycle period. • On each output cycle, PWM = 1 while counter < width, otherwise 0. Result: a carrier at f_clk / 32 with duty cycle set by In[2:0]. Notes: • The default case sets width=0 as a safe fallback, so out-of-range speed values force the output low.

How to test

The combination of logic states on the Input[2:0] signals determines the value of speed and therefore the output duty cycle, as indicated in the following table. Connect an oscilloscope at Out[0] in order to measure PWM output signal

External hardware

3 10K pull-down resistors conected from In[2:0] to GND. 3 switches connected from Input[0:2] to VDD. An oscilloscope connected to Out[0].

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	speed[0]	PWM	—

#	Input	Output	Bidirectional
1	speed[1]	—	—
2	speed[2]	—	—
3	enable	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

SKY130 technology characterization

by **Andreas Madner**

0747 HDL Project

`github.com/analogandi/ttsky-um-technology-characterization`

"First attempt in characterizing the SKY130 technology and perform aging experiments"

How it works

Will be updated later

How to test

Will be updated later

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	unused	unused	unused
1	unused	unused	unused
2	unused	unused	unused
3	unused	unused	unused
4	unused	unused	unused
5	unused	unused	unused
6	unused	unused	unused
7	unused	unused	unused

WokwiPWM

by **Ken Pettit**

0748 | 10 kHz | Wokwi Project

github.com/kdp1965/tt-um-wokwipwm

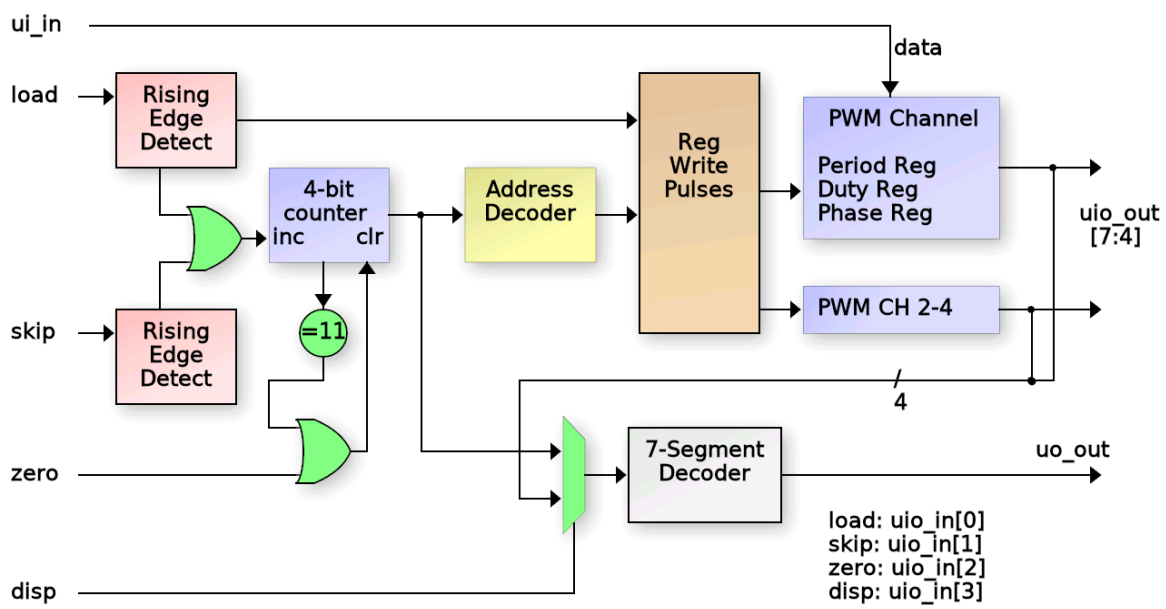
wokwi.com/projects/445338187869298689

"A 4-channel PWM impelemented in PWM gates"

Wokwi 4-Channel PWM

As it says, this is a 4-channel PWM coded in Wokwi. Each channel has 3 8-bit values to control their frequency, duty cycle and relative (starting) phase to each other. Values are loaded by setting 8-bit data in on ui_in[7:0] and loading to the current address which auto-increments.

Block diagram



How it works

After reset, there is an 4-bit counter that is the address where the next 8-bit input data at in[7:0] will be loaded. The 'load' signal loads the 8-bit input to the register at that current counter address and then increments the counter.

The 'skip' input will skip an address without loading it. And the 'zero' input will reset the counter back to zero.

None of the ‘load’, ‘skip’, ‘zero’ or ‘disp’ inputs are debounced, but they are riging edge detected (at least the load and skip are).

The current count (register address) is displayed on the 7-segment display via a 4-to-7 decoder. Each of 4 PWM channels use two 8-bit control registers to set frequency and duty cycle. The first (lowest address) register controls the duty cycle and the second (highest address) controls the period of that channel. Only channels with non-zero period value will operate (the counter does not count when period_val is zero). For PWM channels 2-4, there is an additional 8-bit register (address 8-10) to control the starting phase of that channel relative to channel 1. This means that if channels 2-4 are to be used, channel 1 must also be used.

Program PWM channel control registers to create independent PWMs:

Address Meaning 0x0 PWM1 duty 0x1 PWM1 period 0x2 PWM2 duty 0x3 PWM2 period 0x4 PWM3 duty 0x5 PWM3 period 0x5 PWM4 duty 0x7 PWM4 period 0x8 PWM2 phase 0x9 PWM3 phase 0xa PWM4 phase 0xb Control BIT0: BIT1: BIT2: BIT3: Write ‘1’ to clear and synchronize all PWM channels. Self clearing.

How to test

1. Start the clock and reset the circuit.
2. Set the DIP switches so only switch 7 is on (8’h40)
3. Press ‘load’. The display should now show ‘1’.
4. Set the DIP switches so only switch 8 is on (8’h80).
5. Press ‘load’. The display should show ‘2’ and the first LED should start blinking at about 50% rate.
6. Continue programming values for the other 3 PWMs by setting DIP switch value and pressing ‘load’ even values (duty cycle) must be less that the associated odd address value (period).
7. When the 7-segment display shows “8”, this is the phase offset value for PWM channel 2 relative to channel 1. Loading a value here will force a known phase for channel 2 WHEN THE CHANNELS ARE SYNCHRONIZED.
8. Load additional relative phase values for PWM channels 3 and 4 at the next two addresses (9 and A).
9. To synchronize the PWM channels, write a value of 4 to address B.

External hardware

LEDs should be connected to the PWM channel outputs. The 7-Segement display shows the current register address to be written.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	D0	seg_a	load
1	D1	seg_b	skip
2	D2	seg_c	zero
3	D3	seg_d	disp
4	D4	seg_e	pwm_0
5	D5	seg_f	pwm_1
6	D6	seg_g	pwm_2
7	D7	seg_dp	pwm_3

Bouncing squares

by **parabolaguay-boop** & **BeefJoerky**

0749 | 25.175 MHz | HDL Project

`github.com/parabolaguay-boop/bouncing-squares`

"Renders five bouncing squares on a VGA output"

How it works

It's a screensaver with five squares

How to test

Plug into VGA port.

External hardware

Monitor (old).

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	R1	—
1	—	G1	—
2	—	B1	—
3	—	VSync	—
4	—	R0	—
5	—	G0	—
6	—	B0	—
7	—	HSync	—

TicTacToe

by **Jonathan**

0750 | 100 Hz | Wokwi Project

`github.com/joni1912/ttsky-wokwi-TicTacToe`

`wokwi.com/projects/447051835034957825`

"You can play TicTacToe with two players using RGB Leds and buttons"

How it works

This project implements a Tic-Tac-Toe game using multiplexed RGB LEDs. Each of the 9 game fields is displayed using a common-cathode RGB LED. When a player presses a button assigned to a cell, the logic evaluates the current game state and updates only that cell. When a winning combination is detected, the logic evaluates the grid and drives an additional RGB LED to show the winner's color.

How to test

Wire the circuit according to the Wokwi design (push-buttons, RGB LEDs, connectors). Set the simulation clock to approximately 100 Hz (or faster if preferred). Press any of the 9 buttons to place a move. The LED for that position lights up in the corresponding player color. Continue placing moves alternately. When a player wins, the winner LED displays their color.

External hardware

9× push-buttons, 10× RGB LEDs (common-cathode), resistors

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Button_1	LED_1_RED	ROW_1_GND
1	Button_2	LED_4_RED	ROW_2_GND
2	Button_3	LED_7_RED	ROW_3_GND
3	Button_4	LED_1_BLUE	Button_9
4	Button_5	LED_4_BLUE	—
5	Button_6	LED_7_BLUE	—
6	Button_7	LED_WinnerCheck_RED	—
7	Button_8	LED_WinnerCheck_BLUE	—

MORSEASCII

by **Fatima** & **Tasnim**

0751 HDL Project

github.com/tasal722/Chipp

"Translates morse to ASCII"

How it works

Push buttons in morse letters, it will then translate to ASCII

How to test

Put the word you want to decode in the testprogram and it will work..

External hardware

Button for input, led for output

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input	ASCII[0]	—
1	—	ASCII[1]	—
2	—	ASCII[2]	—
3	—	ASCII[3]	—
4	—	ASCII[4]	—
5	—	ASCII[5]	—
6	—	ASCII[6]	—
7	—	ASCII[7]	—

YM2413 FM synthesis audio chip

by **Sehyeon Kim**, **IKA Microelectronics Inc.** & **Renaldas Zioma (TT/ASIC)**

0774 | 3.57954 MHz | HDL Project

`github.com/rejunity/tt-ym2413-IKAOPLL`

"The YM2413 is the OPL (FM Operator Type-L) series FM synthesis audio chip built using IKA OPLL core implementation."

How it works

TODO, but meanwhile see https://github.com/ika-musume/IKAOPLL/blob/main/README.md

External hardware

Use either

a) Audio PMOD on BIRIDIRECTIONAL pins

b) R2D DAC on OUTPUT pins

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	D0 data bus	DAC output **(LSB)**	(in) A0 register latch
1	D1 data bus	DAC output	(in) /CS chip select **(active low)**
2	D2 data bus	DAC output	(in) /WR write **(active low)**
3	D3 data bus	DAC output	—
4	D4 data bus	DAC output	—
5	D5 data bus	DAC output	—
6	D6 data bus	DAC output	—
7	D7 data bus	DAC output **(MSB)**	(out) PWM output **(Audio PMOD compatible)**

PSM_asic

by **Cristian Valenti**

0782 | 40 MHz | HDL Project

github.com/CristianValenti/Pseudo_Softmax

"It is the implementation of a reduced softmax function for neural networks"

How it works

My project is an 8-bit Pseudo SoftMax accelerators for neural networks. It is basically an activation function at the end of the network that ouputs the results of the computation.

How to test

A behavioural test of the softmax is complex as it's tipical inputs are output of plenty of layers of neurons. Since this particular design has been ripped off from an actually working network, we are pretty sure of its beahviour. The way we tested out is more related to our concerns about the physical implementation of the chip. So what we did was take our Openlane output (nl.v) and provide random inputs to both the netlist and the verilog design and compare it.

This methodology will be implemented on this reporitory in a further release.

External hardware

Since it is an hardware accelerator it doesn't need any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Data_in[0]	Data_out[0]	Sel_out[0]
1	Data_in[1]	Data_out[1]	Sel_out[1]
2	Data_in[2]	Data_out[2]	Sel_out[2]
3	Data_in[3]	Data_out[3]	Sel_out[3]
4	Data_in[4]	Data_out[4]	Sel_in[0]
5	Data_in[5]	Data_out[5]	Sel_in[1]
6	Data_in[6]	Data_out[6]	Sel_in[2]
7	Data_in[7]	Data_out[7]	Sel_in[3]

tt-tinytone

by **Jakob Schaumberger**

0800 | 12 MHz | HDL Project

`github.com/JakobSchaumberger/tt-tinytone`

"PWM tone-generator for a piezo buzzer"

How it works

PWM tone-generator for a piezo buzzer

How to test

Connect piezo buzzer to output_0

External hardware

Piezo buzzer

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	sound_o	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny RAM DFF 2r1w

by **Darryl Miles**

0801 | 10 MHz | HDL Project

github.com/dlmiles/tt09-dffram-2r1w

"RAM made from DFF 2r1w (32x8)"

How it works

This is a really bad implementation of RAM that uses standard verilog to implement a dual-port-read single-port-write RAM using D-type flipflops.

- DO_A Data Out Port-A
- DI_A Data In Port-A
- DO_B Data Out Port-B
- AD_A Address Port-A
- AD_B Address Port-B
- LOHI_A Nibble (4bit) select Port-A
- LOHI_B Nibble (4bit) select Port-B
- W_EN Write Enable (Port-A implied)

2 pages of 16 bytes (8-bits) is the total storage. The high 1-bit of address are set via RST_N configuration, see below. the low 4-bits of address are supplied on the signal lines.

How to test

The external ports are as you would expect for a RAM module, similar to other ram modules based on the pin descriptions.

Memory reads occur all the time, there is no read-enable. Only port-A can be used to write into. The RST_N does not change the contents of the RAM storage area.

The RST_N release (posedge) is used to latch some additional configuration bits, so the following values are significant and can only be changed by clocking RST_N with a posedge which causes capture:

- uio_in[0] ADDRHI 1-bit to change the RAM page that can be accessed. This is a way to fill out the TT 1x1 tile space a little and allow the upper storage area to be accessible.
- uio_in[3:1] unused
- uio_in[4] READ_BUFFERED_A enable this will enable a synchronous output buffer register on the PORT-A to be enable, so the read value becomes available at the next cycle (pipelined) and held between cycles.

If this works as expected this makes the port output asynchronous or synchronous.

- uio_in[5] READ_BUFFERED_B enable, same as above but for PORT-B.
- uio_in[6] WRITE_THROUGH this will activate a MUX bypass that has the effect of implementing a READ_AFTER_WRITE policy, so the currently written value is also the value found at the output port. When inactive (set logic 0) a READ_BEFORE_WRITE policy should be in effect. TODO check this works as expected when READ_BUFFERED_A is active.
- uio_in[7] unused, due to it also being the WRITE-ENABLE bit when in normal operations so allowing the CLK to run freely across reset and an unwanted write occurring.

External hardware

The standard PCB and RP2040 can be used to access. I expect a Micro Python interface to follow in an update.

Future areas to explore

Write a custom placement and wiring router to perform better placement and congestion architecture so that RAM size and WORD WIDTH. This would perform placement into the standard cell track layout so it can be run as a first pass to pack a solution into a design. Ideally leaving the external signals accessible at the edges of the area.

This might allow packing of any width, any depth, single/dual port (as options into the placement process) allowing for consistent size estimations to be made.

It seems when standard placement is left to solve this problem you don't get a result that scales with increased area usage. TODO some research into exactly what occurs in those scenarios, it is expected this maybe due to wiring congestion problem of cells just being in the wrong place / locality and requiring a lot of wiring to get a solution.

I pick dual-port-read support as that should provide a harder problem to solve as a single-port-read needs less wiring.

NOTES

These details relate to the original tt09 version and OpenLane used at the time.

PL_TARGET_DENSITY_PCT=95%
PL_RESIZER_HOLD_SLACK_MARGIN=0.08

GRT_RESIZER_HOLD_SLACK_MARGIN=0.03
CLOCK_PERIOD=100 (10MHz)

- 32x8 3 slew, 26 fanout vio, +106 buffers, resized 646, +16 tie, +238 hold buffers, No room for 156 instances.
- 28x8 1 slew, 36 fanout vio, +124 buffers, resized 763, +16 tie, +201 hold buffers, No room for 23 instances.
- 26x8 1 slew, 28 fanout vio, +105 buffers, resized 646, +16 tie, +191 hold buffers,
  10091 vio, 6289 vio after 6th, did not get much better, 6H to 4025 (incomplete pass)
- 24x8 0 slew, 26 fanout vio, +97 buffers, resized 632, +16 tie, +176 hold buffers,
  9084 vio, 5305 vio after 6th, best 2134 vio after 24th
- 22x8 0 slew, 20 fanout vio, +82 buffers, resized 555, +16 tie, +171 hold buffers,
  7079 vio, 3583 vio after 6th, best 428 vio after 29th, 6H to 427
- 20x8 4 slew, +101 buffers, +101 buffers, resized 649, +16 tie, +151 hold buffers,
  6622 vio, 3390 vio after 6th, best 991 vio after 24th
- 18x8 2 slew, 23 fanout vio, +72 buffers, resized 496, +16 tie, +138 hold buffers,
  4379 vio, 1299 vio after 6th, 0 vio after 43rd,
  SUCCESS
- 16x8 0 slew, 24 fanout vio, +76 buffers, resized 506, +16 tie, +120 hold buffers,
  4802 vio, 1631 vio after 6th, 1 vio after 56th, 6H to 64th

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	DI_A[0]	DO_A[0]	AD_B[0] (in)
1	DI_A[1]	DO_A[1]	AD_B[1] (in)
2	DI_A[2]	DO_A[2]	AD_B[2] (in)
3	DI_A[3]	DO_A[3]	AD_B[3] (in)
4	AD_A[0]	DO_B[0]	LOHI_A (in)
5	AD_A[1]	DO_B[1]	LOHI_B (in)
6	AD_A[2]	DO_B[2]	—
7	AD_A[3]	DO_B[3]	W_EN (in)

FIR Filter

by **Hampus Malmberg**

0802 HDL Project

`github.com/hammal/vhdl_filter`

"An FIR Filter"

How it works

A finite impulse response (FIR) filtered signal is computed by applying a 5 tap FIR filter convolution to the last fed 5 input signals. Each clock period triggers a new computation as the input is shifted into the internal 5 delay element buffer. Optionally, the filter coefficients can be set by feeding them on the input in reverse order (for five clock cycles) after which a '1' on the load signal triggers a coefficient update.

How to test

See test.py. Essentially the coefficients are first fed in during 5 clock cycles after which we can perform filtering by applying inputs and receiving the filtered signal on the output. Note that a new input sample is expected and the resulting filter output is computed for every following clock cycle.

External hardware

None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input signal (0)	output signal (0)	load signal
1	input signal (1)	output signal (1)	—
2	input signal (2)	output signal (2)	—
3	input signal (3)	output signal (3)	—
4	input signal (4)	output signal (4)	—
5	input signal (5)	output signal (5)	—
6	input signal (6)	output signal (6)	—
7	input signal (7)	output signal (7)	—

NCO

by **Pantelis**

0803 | 50 MHz | HDL Project

`github.com/pantelis300/tt_um_pantelis300_nco`

"Numerically Controlled Oscillator"

How it works

Numerically Controlled Oscillator (NCO) with linear-interpolated sine output.

- 8-bit `freq_ctrl` controls the output frequency
- 8-bit `dac_out` drives a DAC
- Uses 16-bit phase accumulator + sine ROM + linear interpolation between samples.

Frequency relationship (with `PHASE_WIDTH` = 16, `ROM_ADDR_BITS` = 6) f_out = (freq_ctrl / 2^PHASE_WIDTH) * f_clk

It's a basic block for Direct Digital Synthesis (DDS).

How to test

On the board connct the `uo_out` on an 8-bit DAC. The output of the DAC connected to an oscilloscope should give you a sinewave of controlled frequency (`freq_ctrl` needs to be in the range [1, 255]).

Run the bit-accurate python model

You can run the python model to get an idea of the expected output by observing the generated plots:

```
$ cd python_nco_model
$ python3 nco_model.py
$ eog nco_qwave_waveforms.png
```

Run Simulation

```
$ cd sim
$ make waves
```

You can observe in gtkwave the `dac_out` output using the analog data format.

External hardware

8-bit DAC connected at the output.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	freq_ctrl_0	dac_out_0	—
1	freq_ctrl_1	dac_out_1	—
2	freq_ctrl_2	dac_out_2	—
3	freq_ctrl_3	dac_out_3	—
4	freq_ctrl_4	dac_out_4	—
5	freq_ctrl_5	dac_out_5	—
6	freq_ctrl_6	dac_out_6	—
7	freq_ctrl_7	dac_out_7	—

hardcoded sequencer

by **Kasper** & **Mikael**

0804 | 4.8 MHz | HDL Project

`github.com/magmusson/bygg_ett_gratis_chip`

"a squarewave synth sequencer that only follows a specific progression, you can set the tempo"

How it works

it plays a sequencer, plug and play. use the inputs to increase/decrease tempo. you need a speaker for it to play

How to test

set a clock to 4,8Mhz and connect a speaker and it should work.

External hardware

a speaker is needed to play the music.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	tempo up	output to speaker	—
1	tempo down	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Dual-Channel PWM with SPI Control + Extra Test Logic

by **Nicolás Romero** & **Eduardo Holguín**

6805 | 1 kHz | HDL Project

`github.com/EduHolg/Dual_PWM_Generator`

"Two complementary PWM outputs per channel with SPI-configurable duty and alignment. Unused pins repurposed as gate-level test signals for debugging."

How it works

This design creates **two independent PWM channels** (each with a complementary output), and their duty cycles + alignment modes are set by an SPI interface.

- UI pins:
  - `ui_in[0]` = internal chip clock (also routed to an output pin for probing)
  - `ui_in[1]` = SPI SCLK
  - `ui_in[2]` = SPI MOSI
  - `ui_in[3]` = SPI SS (active low)
  - `ui_in[4]` = async reset (active high)
  - `ui_in[5]` = extra test logic input A
  - `ui_in[6]` = extra test logic input B
  - `ui_in[7]` = extra test logic input C (currently unused in logic)

Internally each PWM channel can work in:

- **edge–aligned** mode: up counter
- **center–aligned** mode: up/down counter (triangular)

Duty is 4 bits (0–15).
SPI writes 5 bits per channel: 4 bits duty + 1 bit mode.
Order: first 5 bits configure PWM1, the next 5 bits configure PWM2.

In addition to the PWM outputs, **all other unused outputs** (uo + uio) have been assigned to simple logic operations based on A/B test inputs. This allows validating silicon gates easily with a scope or logic analyzer. Also, an internal divided clock is made available at one pin for slower visual debugging.

How to test

1) Drive a stable clock into `ui_in[0]` (tens of kHz or low MHz recommended).
2) Apply **ui_in[4] = 1** for 1 cycle to reset, then back to 0 to run.
3) With SPI in **MODE 1** (CPOL=0, CPHA=1):

- Pull SS low
- Shift MSB-first:

channel	data	meaning
PWM1	DDDD M	duty (4) + mode (1)
PWM2	DDDD M	duty (4) + mode (1)

`M=0` → edge aligned
`M=1` → center aligned

Example: set PWM1=8/15 center and PWM2=4/15 edge

- send: `1000 1` then `0100 0`

4) Observe the four PWM outputs on the logic analyzer or scope.
They will update every time a full 5-bit frame is clocked into each channel.

5) Place test vectors on `ui_in[5]` and `ui_in[6]` (A/B) and observe all the gate-level outputs to verify correct logic function on silicon.

External hardware

No external hardware is required for basic validation.

(If desired, PWM can drive H-bridge gate drivers, LEDs, etc., but not required for functional demonstration.)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	CLK_g	PWM1	A NAND B
1	SCLK_g	PWM1_N	A NOR B
2	MOSI_g	PWM2	A XNOR B
3	SS_g (active-low)	PWM2_N	NOT A
4	RESET_g (active-high)	A AND B	NOT B
5	A (test input)	A OR B	A AND (B)
6	B (test input)	A XOR B	(A) AND B
7	C (test input/spare)	CLK out (raw)	VDD

Mastermind

by **Unt-C**

0806 | 100 kHz | HDL Project

`github.com/maxha465/tt-gameonchip`

"Epiiiiic game recreating the classic Mastermind"

How it works

A simple random number generator assigns random values for the correct "colors" which are then compared to the guesses that the users have made to determine how many "colors" are correct and in the correct position (referred to as correct) as well as how many "colors" are correct but in the wrong position (referred to as half correct).

How to test

Start the program by resetting. Proceed by guessing the correct "color" by guessing four (zero through three) 3-bit binary numbers, then comparing them to the right answers by pressing submit. There are 16 turns to guess the correct "color" combination. You get hints by knowing how many "colors" are correct and how many "colors" are half correct. When the turns counts down to zero, you lose and have to manually reset to be able to play another round.

External hardware

The game is best enjoyed by connecting seven-segment displays to the outputs of correct, half correct and turns.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	zero_0	correct_0	two_2
1	zero_1	correct_1	three_0
2	zero_2	correct_2	three_1
3	one_0	half_correct_0	three_2
4	one_1	shalf_correct_1	submit
5	one_2	half_correct_2	turns_0
6	two_0	turns_3	turns_1
7	two_1	—	turns_2

test

by **Timo Reiter**

0807 | 1 kHz | Wokwi Project

github.com/timoreiter/Workshop

wokwi.com/projects/442988784492711937

"Test"

How it works

?

How to test

?

External hardware

?

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output a	—
1	input b	output b	—
2	input c	output c	—
3	input d	output d	—
4	input e	output e	—
5	input f	output f	—
6	input g	output g	—
7	input h	output h	—

React Chip

by **saksh156**

0808 HDL Project

github.com/Red2trappy/muxen_proj

"A chip for a simple reaction test game."

How it works

It works!

How to test

Send in data!

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	b	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

cirucuit

by **Abd-ur-rehman** & **abd ur rahim**

0809 Wokwi Project

github.com/abdurstar389/circuit

wokwi.com/projects/445254916601240577

"shows the intial for my name"

How it works

You can change switches to make your intials show up Explain how your project works It uses a basic lcd ## How to test See if the leds light up Explain how to use your project uses the switches ## External hardware Led display List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	rx	—	—
1	tx	—	—
2	ou1	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	ou3	—	—
7	ou9	—	—

RagnarMegaRaknare

by **Lucas Nilsson**, **Josua Kärnsund** & **Petter Ljungkvist**

0810 HDL Project

github.com/LucasNilsson954/RagnarMegaRaknare

"A Mega Silly Calculator"

How it works

This is a calculator which can take in a four bit operand followed by an operation (addition, subtraction, multiplication or division) then another four bit number. The result will be outputted in an eight bit number. Division rounds upwards. Using zero as an operand is not supported, because Ragnar disapproves of the concept of zero, because no one deserves to be a zero.

How to test

Connect a hexadecimal keypad to the circuit then do your favorite calculations. Bits 0-3 are used for operands and operations while bit 4 is used for strobe, which activates shortly after pressing a button.

External hardware

Some displays to display the result.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Number/operation	Result	—
1	Number/operation	Result	—
2	Number/operation	Result	—
3	Number/operation	Result	—
4	Strobe	Result	—
5	—	Result	—
6	—	Result	—
7	—	Result	—

Tiny Tapeout Test Gates

by **Mobin Alimohammadi**

0811 | 10 kHz | Wokwi Project

`github.com/MobinMakes/Tiny-Tapeout-GDS`

`wokwi.com/projects/445254913718704129`

"Demo for logic gates and a clock divider"

How it works

AND, NAND gates connected to input a and b.

NOT gate connected to input d.

Two NOR gates connected in series with one feedback loop from output 7 and inputs from g and h.

Logic behavior

AND gate: Y_AND = a · b

NAND gate: Y_NAND = ¬(a · b)

NOT gate: Y_NOTd = ¬d

NOR latch: Inputs: g → Set (S), h → Reset (R) Outputs: Q (main), $\overline{Q}$ (complement) Characteristic equations: Q = NOR(R, $\overline{Q}$) $\overline{Q}$ = NOR(S, Q)

a	b	d	Y_AND	Y_NAND	Y_NOTd
0	0	0	0	1	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	1	0	1
1	1	1	1	0	0

SR latch (g → S, h → R) | S (g) | R (h) | Q_next | $\overline{Q}$_next | Description | | :—: | :—: | :—-: | :——: | ——————— | | 0 | 0 | Q_prev | ¬Q_prev | Hold (no change) | | 1 | 0 | 1 | 0 | Set | | 0 | 1 | 0 | 1 | Reset | | 1 | 1 | X | X | Forbidden / Undefined |

How to test

Power & setup Connect all inputs to VCC or GND through pull-up/pull-down resistors. Use pushbuttons or switches for a, b, d, g, and h. Verify basic gates Toggle a and b to confirm AND/NAND output behavior. Toggle d to confirm the NOT gate output. Test the SR latch With g=0, h=0, the latch should hold its previous state. With g=1, h=0, it sets Q=1. With g=0, h=1, it resets Q=0. Avoid g=1, h=1 (undefined state).

Observe outputs Use LEDs or a 7-segment display to visualize logic states. Q may drive one LED; Y_AND, Y_NAND, Y_NOTd can drive others.

External hardware

Momentary pushbuttons for inputs (a, b, d, g, h) Pull-up resistors (10 kΩ typical) LEDs or a 7-segment display for output indication Breadboard, jumper wires, regulated VCC supply

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input a	output and	—
1	input b	output nand	—
2	input c	output not	—
3	input d	output not	—
4	input e	output	—
5	input f	output	—
6	input g	output	—
7	input h	output nor feedback	—

Uart and Temperature sensor

by **riyasls**

0812 | 20 Hz | HDL Project

`github.com/riyasls/tt_uart_temp_sens`

> *"This project implements a convert a pwm digital serial signal to a parallel value and transmits it over UART (Tx) at the configured baud rate."*

Project_Tittle => Uart_Temperature sensor

Who

1.hiral Patadiya 2.Riya soni 3.Divayang Rana

what

making uart_temperature sensor

Short Discription

=> This project implements a convert a pwm digital serial signal to a paralle value and trasmits it over uart(Tx) at the configured baud rate.

How it works

The PWM signal is generated by the temperature sensor, and this module is designed to count the number of clock cycles during which the PWM signal remains LOW. The resulting count value is then transmitted to the UART receiver via the UART TX pin using the 115200 baud rate.

Block Diagram

pinout summary

Dependenices => This design instantiates one submodule: Uart_Temp

How to test

The input PWM signal is provided on the input pin ui_in[0], and the UART TX output is connected to the output pin uo_out[0]. To capture the transmitted value at the UART receiver, configure your serial terminal (e.g., Tera Term, PuTTY, etc.) with the following settings: Baud Rate: 115200, Data Bits: 8, Parity: None, Stop Bits: 1, Flow Control: None, The design operates with a clock frequency of 50 MHz.The values that shown on the serial terminal is in the

Hexadecimal form. The low time period = (1/clock frequency) * (counter value shown in the serial terminal in decimal).

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	pwm input for uart module	UART TX output	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Tiny Tapeouts Chip Design

by **Daya Thomas**

0813 Wokwi Project

github.com/dayasusanthomas/Chip-Design

wokwi.com/projects/445255035084055553

"Testing Logic Gates"

How it works

Tests OR Gates in a simple circuit ## How to test Test through inputs and outputs

Input a| Output AND|
0 | 0 |

External hardware

Not applicable

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	1	1	—
1	2	1	—
2	3	1	—
3	4	output or	—
4	input a	1	—
5	input a	1	—
6	5	1	—
7	6	1	—

vga_projekt_ed_nkpng

by **ED2 nkpng**

0814 HDL Project

`github.com/wlinne/vga_projekt_ed_nkpng`

"VGA color test image"

How it works

Two FSM modules, two synchronous counters and timing constants defined in 640 x 480 px VGA standard specification form a driver which generates output to display a test image for the three color channels.

How to test

Connect external hardware for the vga port, power on, connect vga compatible device.

External hardware

TinyVGA Pmod https://store.tinytapeout.com/products/TinyVGA-Pmod-p678647356

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	b	—
1	—	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

xorshift

by **fkdajlfas**

0815 HDL Project

`github.com/oscargus/byggenchip`

"An implementation of XORshift"

How it works

This is an 8bit VHDL implementation of Tzarc's XORShift. It is a counter based random number generator which based on the previous output and the count of generated numbers generates a new ouput.

How to test

This can be tested by running a known good correct implementation of XORShift and comparing the output with the output of the manual clocked circuit.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in0	out0	—
1	in1	out1	—
2	in2	out2	—
3	in3	out3	—
4	in4	out4	—
5	in5	out5	—
6	in6	out6	—
7	in7	out7	—

KianV RISC-V RV32E Baremetal SoC

by **Dipl.-Ing. Hirosh Dabui**

0838 | 50 MHz | HDL Project

`github.com/by17s/RISCV-KianV-BareMetalStyle`

"A baremetal RISC-V RV32E ASIC with audio, spi, uart"

How it works

After implementing a KianV uLinux TT06, I felt like implementing a KianV bare metal edition, which is an RV32E RISC-V32 SoC. This SoC is equipped with a UART, qspi memory controller (psram/flash), a generic SPI interface, and a sigma-delta emulator for playing audio files. In the firmware folder, the kernelboot.c and crt0.S files display all hardware registers and their initialization in the code.

How to test

First, one must build the toolchain for an RV32E, as you can see here:

```
sudo apt-get update
sudo apt-get install autoconf automake autotools-dev curl python3
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex
texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev
git clone --recursive https://github.com/riscv/riscv-gnu-toolchain.
git
cd riscv-gnu-toolchain
./configure --prefix=/opt/riscv32e --with-arch=rv32e --with-
abi=ilp32e
make
export PATH=/opt/riscv32e/bin:$PATH
```

The following hardware addresses are given:

```
#define LSR_DR 0x01
#define LSR_TEMT 0x40
#define LSR_THRE 0x20
#define PWM_ADDR (IO_BASE + 0x14)
#define REG_DIV (IO_BASE + 0x10)
#define SPI_DIV (IO_BASE + 0x500010)
#define UART_LSR (IO_BASE + 0x5)
#define UART_RX (IO_BASE)
#define UART_TX (IO_BASE)
```

The use of the registers can be determined from the C, linker script and assembly program. The CPU starts to execute the instruction stored in the NOR Flash at an offset of 1MiB. When the chip comes into my hands, I will

provide demos that I test on the chip, including audio playback with appropriate documentation.

External hardware

It's very important to use the PMOD Flash + PSRAM. We only use 8MB of PSRAM address space.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	uart_rx	spi_cen0	ce0 flash
1	spi_sio1_so_miso0	spi_sclk0	sio0
2	—	spi_sio0_si_mosi0	sio1
3	—	pwm_o	sck
4	—	uart_tx	sd2
5	—	led[0]	sd3
6	—	led[1]	cs1 psram
7	—	led[2]	always high

SPELL

by **Uri Shaked**

0844 | 10 MHz | HDL Project

github.com/urish/tt-spell

"SPELL is a minimal, cryptic, stack-based programming language crafted for The Skull CTF"

How it works

SPELL is a minimal, stack-based programming language created for The Skull CTF.

The language is defined by the following cryptic piece of Arduino code:

```
void spell() {

                  uint8_t*a,pc=16,sp=0,
               s[32]={0},op;while(!0){op=
            EEPROM.read(pc);switch(+op){case
          ',':delay(s[sp-1]);sp--;break;case'>':
         s[sp-1]>>=1|1;break;case'<':s[sp-1]<<=1;
        break;case'=':pc=s[sp-1]-1;sp--;break;case
       '@':if(s[sp-2]){s[sp-2]--;pc=s[sp-1]-1;sp+=
      1;}sp-=2;break;case'&':s[sp-2]&=s[sp-1];sp-=1;
      break;case'|':s[sp-2]|=s[sp-1];sp-=1;break;case
    '^':s[sp-2]^=s[sp-1];sp--;break;case'+':s[sp-2]+=
   s[sp-1];sp=sp-1;break;case'-':s[sp-2]-=s[sp-1];sp--;
   break;case'2':s[sp]=s[sp-1];sp=sp+1;break;case'?':s[
 sp-1]=EEPROM.         read(s[sp-1]|0        );break;case
  "!!!"[0]:             666,EEPROM              .write(s
   [sp-1]                ,s[sp-2]                );sp=+
    sp-02;               ;break;                 1;case
    "Arr"[               1]:  s[+               sp-1]=
    *(char*)            (s[+   sp-1            ]);break
      ;case'w':*   (char*)(     s[+sp-1])  =s[sp-+2];
        sp-=2;break;case+       'x':s[sp] =s[sp-1
           ];s[sp-1]=s[sp   +    -2];s[sp-2]=s[
            0|sp];break;    ;;    case"zzz"[0
             ]:sleep();"   Arrr  ";break;case
             255  :return;;  default:s  [sp]
              =+   op;sp+=    1,1   ;}pc=
               +    pc  +      1;      %>

}
```

This design is an hardware implementation of SPELL with the following features:

- 256 bytes of program memory (volatile, simulates EEPROM)
- 32 bytes of stack memory
- 32 bytes of data memory
- 8 bidirectional pins and up to 8 output-only pins

Initially, all the program memory is filled with `0xFF`, and the stack and data memory are filled with `0x00`. The program counter is set to `0x00`, and the stack pointer is set to `0x00`.

To load a program or inspect the internal state, the design provides access to the following registers via a simple serial interface:

Address	Register name	Description
0x00	PC	Program counter
0x01	SP	Stack pointer
0x02	EXEC	Execute-in-place (write-only)
0x03	STACK	Stack access (read the top value, or push a value)

The serial interface is implemented using a shift register, which is controlled by the following signals:

Pin	Type	Description
`reg_sel`	input	Select the register to read/write
`load`	input	Load the selected register with the value from the shift register
`dump`	input	Dump the selected register value to the shift register
`shift_in`	input	Serial data input
`shift_out`	output	Serial data output (when `porta[3]` is disabled)

When `load` is high, the value from the shift register is loaded into the selected register. When `dump` is high, the value of the selected register is dumped into the shift register, and can be read after two clock cycles by reading `shift_out` (MSB first).

For example, if you want to read the value of the program counter (PC), you would:

1. Set `reg_sel` to 0x00 and set `dump` to 1
2. Wait for two clock cycles for the first bit (MSB) to appear on `shift_out`.
3. Read the remaining bits from `shift_out` on each clock cycle.

To write a value to the program counter, you would:

1. Write the value to the shift register, one bit at a time, starting with the **MSB**.
2. Set `reg_sel` to 0x00 and set `load` to 1.
3. Wait for a single clock cycle for the value to be loaded.

Writing an opcode to the `EXEC` register will execute the opcode in place, without modifying the program counter (unless the opcode is a jump instruction).

The `STACK` register is used to push a value onto the stack or read the top value from the stack (for debugging purposes).

Data memory and registers

The data memory space is divided into two regions:

Address range	Description
0x00 - 0x1F	General-purpose data storage (data memory)
0x20 - 0x5F	I/O and control registers

Other addresses are reserved for future use, and should not be accessed.

The following registers are available in the data memory space:

Address	Name	Description
0x36	PINB	Read the value of the `portb` pins, or toggle the output when written to
0x37	DDRB	Set the direction of the `portb` pins (0 = input, 1 = output)
0x38	PORTB	Write to the `portb` pins
0x39	PINA	Toggle the output on `porta` pins (write only; read returns 0x00)
0x3A	DDRA	Enables of the `porta` pins (0 = disabled, 1 = output)
0x3B	PORTA	Write to the `porta` (output only) pins

For example, to toggle the value of the `portb[2]` (`uio[2]`) pin, you would write `0x04` to the `PINB` register.

The `porta[3:0]` pins are also used for debug output, and their function is determined by the `DDRA` register:

Output pin	DDRA[n] == 0	DDRA[n] == 1
0	`sleep`	`porta[0]`
1	`stop`	`porta[1]`
2	`wait_delay`	`porta[2]`
3	`shift_out`	`porta[3]`

4	0	`porta[4]`
5	0	`porta[5]`
6	0	`porta[6]`
7	0	`porta[7]`

How to test

To test SPELL, you need to load a program into the program memory and execute it. You can load the program by repeatedly executing the following steps for each byte of the program:

1. Write the byte to the top of the stack (using the `STACK` register)
2. Write the address of the byte in the program memory to top of the stack
3. Write the opcode `!` to the `EXEC` register

After loading the program, you can execute it by writing the address of the first byte in the program memory to the `PC` register, and then pulsing the `run` signal.

Test programs

The following program will spell "SPELL" on the Tiny Tapeout demo board's 7-segment display: (see what we did there?)

```
[127, 58, 119, 0, 129, 57, 57, 244, 62, 116, 109, 50, 0, 38, 94,
59, 119, 250, 44, 0, 59, 119, 25, 44, 11, 64, 3, 61]
```

The program bytes should be loaded into the program memory starting at address 0.

And of course, the obligatory blink, rapidly blinking an LED connected to the `uio[0]` pin:

```
[1, 55, 119, 1, 54, 119, 250, 44, 3, 61]
```

External hardware

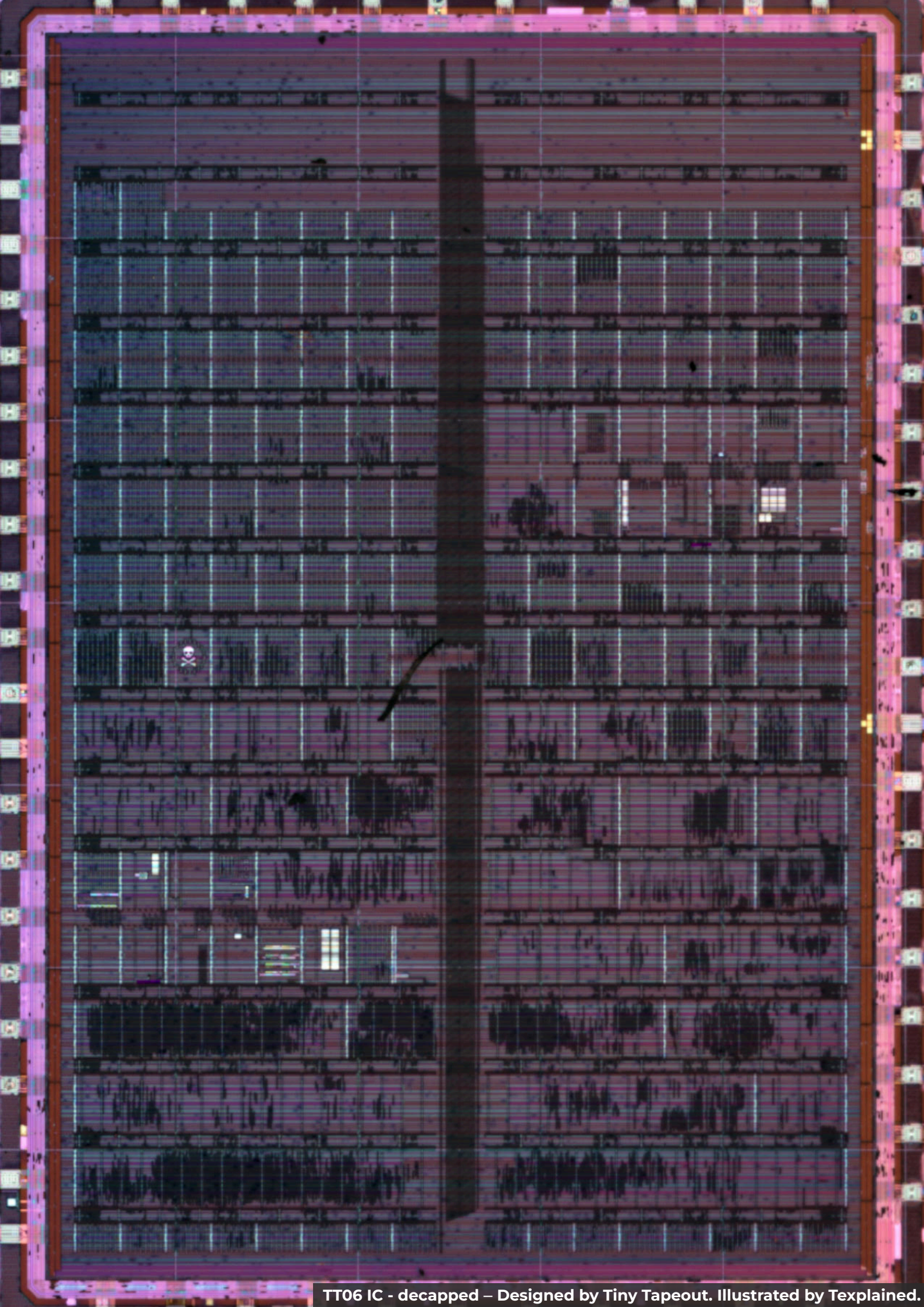
None

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	run	sleep/porta[0]	portb[0]
1	step	stop/porta[1]	portb[1]
2	load	wait_delay/porta[2]	portb[2]
3	dump	shift_out/porta[3]	portb[3]

#	Input	Output	Bidirectional
4	shift_in	porta[4]	portb[4]
5	reg_sel[0]	porta[5]	portb[5]
6	reg_sel[1]	porta[6]	portb[6]
7	—	porta[7]	portb[7]



TT06 IC - decapped – Designed by Tiny Tapeout. Illustrated by Texplained.

VGA Pride

by **Rebecca G. Bettencourt**

846 HDL Project

`github.com/RebeccaRGB/tt-vga-pride`

"A VGA demo for showing pride flags"

How it works

Displays pride flags on the screen.

To add another flag, create a flag.v file and add it to `src/flag_index.v`, `test/Makefile`, and `info.yaml`, using the existing flags as examples.

How to test

Connect to a VGA monitor. Set the following inputs to change the displayed flag:

- `ui_in[7]` to display the first flag
- `ui_in[6]` to display the next flag
- `ui_in[5]` to display the previous flag
- `ui_in[4]` to display the flag whose index is on `uio_in`

Index	Flag
0	Rainbow flag, 6 stripes
1	Rainbow flag, 7 stripes
2	Rainbow flag, 8 stripes
3	Rainbow flag, 9 stripes
4	Philadelphia rainbow flag
5	Progress rainbow flag
6	Progress rainbow flag 2021 version
7	Trans pride flag
8	Abrosexual pride flag
9	Aceflux pride flag
10	Aegosexual pride flag
11	Agender pride flag
12	Androgyne pride flag
13	Androsexual pride flag
14	Aporagender pride flag

15	Aroace pride flag
16	Aroflux pride flag
17	Aromantic pride flag
18	Asexual pride flag
19	Aspec pride flag
20	Bigender pride flag (pink purple white purple blue)
21	Bigender pride flag (blue white purple white pink)
22	Bigender pride flag (pink yellow white purple blue)
23	Bisexual pride flag
24	Ceterosexual pride flag
25	Demiandrogyne pride flag (pink purple blue)
26	Demiandrogyne pride flag (green white green)
27	Demiboy pride flag
28	Demifluid pride flag
29	Demiflux pride flag
30	Demigender pride flag
31	Demigirl pride flag
32	Demiromantic pride flag
33	Demisexual pride flag
34	Disability rights flag (gold silver bronze tricolor)
35	Disability rainbow flag
36	Gender-neutral pride flag
37	Genderfluid pride flag
38	Genderflux pride flag
39	Genderqueer pride flag
40	Greygender pride flag
41	Greysexual pride flag
42	Gynosexual pride flag
43	Intersex pride flag (purple circle)
44	Intersex pride flag (blue/pink gradient)
45	Thislesbianlife lesbian pride flag (pink and red)
46	Sadlesbeandisaster lesbian pride flag, 7 stripes (orange and pink)
47	Sadlesbeandisaster lesbian pride flag, 5 stripes (orange and pink)
48	Lydiandragon lesbian pride flag (violet crocus dill rose)

49	Maya Kern lesbian pride flag (violet rose crocus dill)
50	RebeccaRGB femme lesbian pride flag (violet lavender pink rose)
51	Littleender pride flag
52	Maverique pride flag
53	Leonis Ignis MLM pride flag (brown and blue)
54	Vincian MLM pride flag, 7 stripes (green and blue)
55	Vincian MLM pride flag, 5 stripes (green and blue)
56	Vincian MLM pride flag (light blue and light green)
57	Multigender pride flag
58	Multisexual pride flag
59	Neptunic pride flag
60	Neutrois pride flag
61	Nonbinary pride flag
62	Objectum pride flag
63	Omnisexual pride flag
64	Pangender pride flag
65	Pansexual pride flag
66	Polyamory pride flag (blue, red, black with yellow pi)
67	Polyamory pride flag (blue, magenta, purple with yellow heart)
68	Polygender pride flag
69	Polysexual pride flag
70	Pomosexual pride flag
71	Proculsexual pride flag
72	IBM PS/2 pride flag
73	Queer pride flag
74	Trains pride flag (*Train Landscape*, Ellsworth Kelly, 1953)
75	Transfeminine pride flag
76	Transmasculine pride flag
77	Transneutral pride flag
78	Trigender pride flag
79	Unlabeled pride flag
80	Uranic pride flag
81	Voidpunk pride flag
82	Dorian Rutherford butch lesbian pride flag (blue and purple)

83	Butchspace butch lesbian pride flag (orange and yellow)
84	Sapiosexual pride flag (green brown blue)
85	Sapiosexual pride flag (white pink blue)

External hardware

TinyVGA PMOD

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	address mode	R1	A0
1	—	G1	A1
2	—	B1	A2
3	—	VSync	A3
4	set	R0	A4
5	prev	G0	A5
6	next	B0	A6
7	reset	HSync	A7

JulkeGussinatorn 2.0

by **hamgr699 thegu842**

0864 | 10 MHz | HDL Project

`github.com/Hhulke/JulkeGussinatorn-2.0`

“:)”

How it works

You enter normal or encrypted morse code signal by sending ‘1’ to input 0 for a short morse dot. If you send a ‘1’ to input 1 you get a long morse dot . After each letter send ‘1’ to input 2 to lock in that letter (for a space just send ‘1’ to input 2). Show morse code by sending ‘1’ to input 3. Encrypt by sending ‘1’ to input 4. Decrypt by sending ‘1’ to input 5.

How to test

You enter normal or encrypted morse code. After each letter send ‘1’ to input 2 to lock in that letter (for a space just send ‘1’ to input 2). Show morse code by sending ‘1’ to input 3. Encrypt by sending ‘1’ to input 4. Decrypt by sending ‘1’ to input 5.

With a led on output you can see the morse code that is saved on the device

External hardware

1 LED for morse code output

6 buttons for input of short and long morse dot, next letter, show morse code, encrypt and decrypt

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Kort morse	LED	—
1	Lång morse	—	—
2	Nästa	—	—
3	Spela	—	—
4	Kryptera	—	—
5	Dekryptera	—	—
6	x	—	—
7	x	—	—

I2C Slave

by **Markus Pommermayr**

0865 | 100 kHz | HDL Project

`github.com/PomMarkus/TinyTapeout`

> *"only write I2C Slave (Adr: 0x8E) wit 3 control registers (0x00 - 0x02) and 8 Data Registers (0x08 - 0x0F). Control registers control direct passthrough / stepper mode and operating speed."*

How it works

This project is a i2c slave for write only (address: 0x8E) with 3 control registers (0x00 - 0x02) and 8 data registers (0x08 - 0x0F).

Register configuration: CTRL0 register:

-
-
-
-

CTRL0[4-7], CTRL1[0-7] and CTRL2[0-7] set a compare value for the 1MHz counter (LSB: CTRL0[4], MSB CTRL2[7])

Description: The saved byte inside the 11 registers can be changed via i2c.

- When CTRL00 is HIGH, the Direct Troughput Mode is enabled. This means, that the 8 input pins are directly mapped to the output pins. It also blocks the Stepper from operation
- When CTRL00 is LOW, the Direct Trouput Mode is dissabled
- When CTRL01 is HIGH, the Stepper is enabled when CTRL00 is LOW.
- When CTRL01 is LOW, the Stepper gets reset to step0
- step0 activates nothing
- step1 to step8 enable the output of data1 to data8. Only one output can be turned on at a time.
- step0 follows after step8

With the reigsters CTRL0[4-7], CTRL1[0-7] and CTRL2[0-7] a compare value for a counter can be defined. The internal clock operates with 1MHz. When the compare value is reached, the counter resets and the stepper moves one step forward.

How to test

an i2c master is mandatory to test this project

External hardware

- i2c Master
- 8 switches for the input
- 8 leds for the output (7-segment display for example)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data in 0	data out 0	SDA
1	data in 1	data out 1	SCL
2	data in 2	data out 2	—
3	data in 3	data out 3	—
4	data in 4	data out 4	—
5	data in 5	data out 5	—
6	data in 6	data out 6	—
7	data in 7	data out 7	—

7-Segment Counter

by **Angel Viera**

0866 | 1 Hz | HDL Project

`github.com/UncleGravity/tinytapeout`

"Counts from 0 to 9 once per second on a 7-segment display"

How it works

This project implements a simple decimal counter that counts from 0 to 9 and displays the current count on a 7-segment display.

The design uses a 4-bit counter that increments on each clock edge. When the counter reaches 9, it automatically wraps back to 0 on the next clock cycle. The counter value is decoded into 7-segment display patterns using a combinational logic decoder.

The design expects a 1 Hz clock input (configured via `clock_hz: 1` in `info.yaml`).

How to test

1. Connect a common-cathode 7-segment display to the output pins (uo[0] through uo[6])
2. Provide a 1 Hz clock signal (or configure Tiny Tapeout to provide it automatically)
3. Assert reset (rst_n low) to initialize the counter to 0
4. Release reset (rst_n high) and observe the counter counting from 0 to 9 repeatedly
5. Each digit should be displayed for 1 second before incrementing to the next

Pin Mapping

- **uo[0]**: Segment A (top)
- **uo[1]**: Segment B (top right)
- **uo[2]**: Segment C (bottom right)
- **uo[3]**: Segment D (bottom)
- **uo[4]**: Segment E (bottom left)
- **uo[5]**: Segment F (top left)
- **uo[6]**: Segment G (middle)
- **uo[7]**: Decimal point (always off)

External hardware

You will need:

- **1x Common-cathode 7-segment display** (or a TinyTapeout 7-segment display board)

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	ui_unused_0	Segment A	uio_unused_0
1	ui_unused_1	Segment B	uio_unused_1
2	ui_unused_2	Segment C	uio_unused_2
3	ui_unused_3	Segment D	uio_unused_3
4	ui_unused_4	Segment E	uio_unused_4
5	ui_unused_5	Segment F	uio_unused_5
6	ui_unused_6	Segment G	uio_unused_6
7	ui_unused_7	Decimal Point	uio_unused_7

EZ Reservoir

by **kinako71-2**

0867 HDL Project

github.com/kinako71-2/TTSKY25b

"Lightweight linear reservoir ASIC implemented in Verilog."

How it works

The neurons are imagined to be arranged in a 4 × 4 grid, with each neuron connected to its four neighboring neurons.
In reality, neurons [0, 0] and [3, 0] cannot be physically connected, but here it is assumed that they are connected.

At each time step, the state of each neuron is updated based on the input value and the states of its neighbors.

The network structure described above is shown in the figure below.

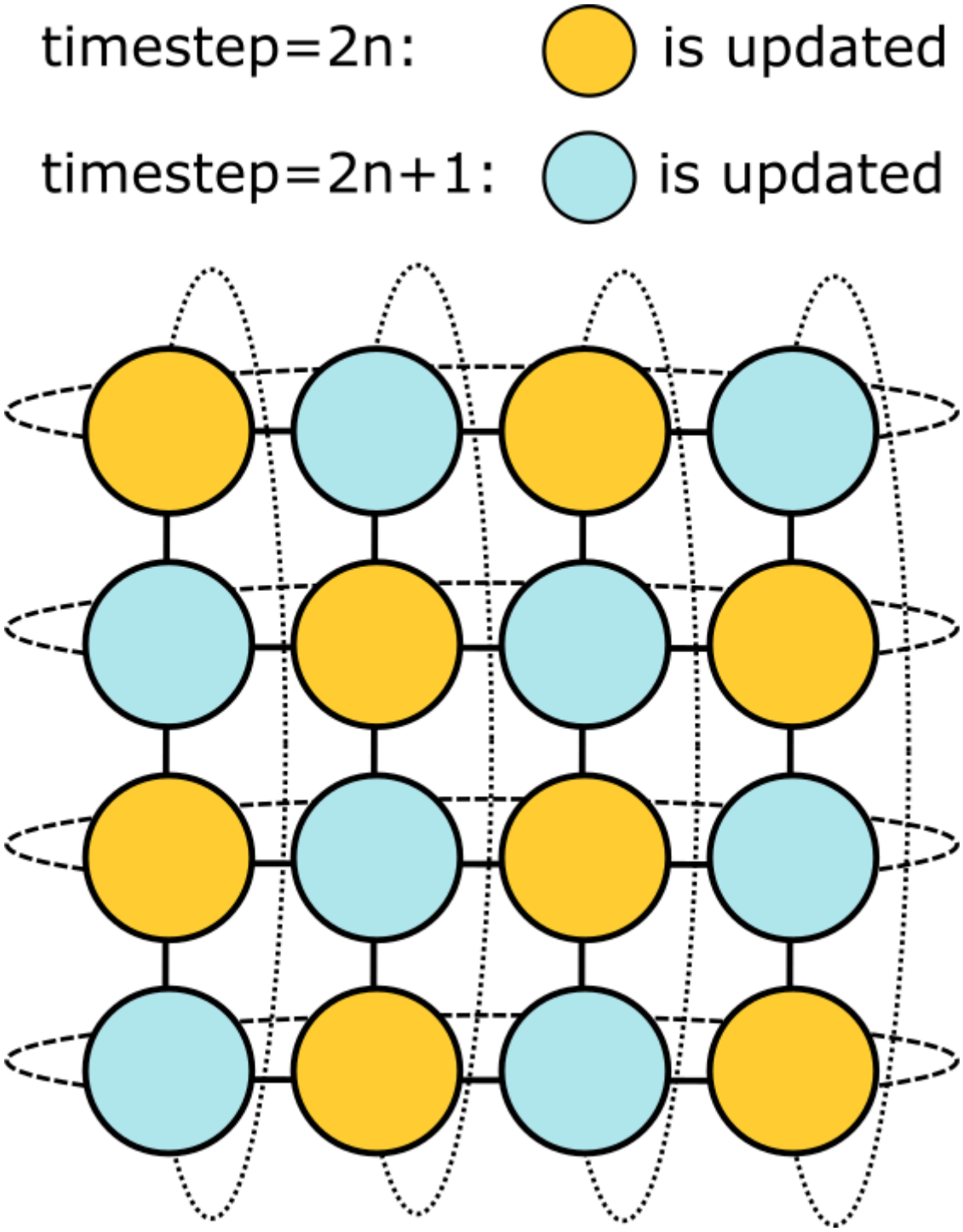


Figure 867.1: Reservoir structure

Here, the yellow neurons represent **odd neurons**, and the blue neurons represent **even neurons**, as mentioned in the pin explanation.
Odd and even neurons are updated alternately, following a checkerboard pattern.

The inputs, states, and weights are all **8-bit**, which might reduce accuracy. Moreover, the absence of a nonlinear transformation (e.g., *tanh*) might further degrade performance, making it different from the original reservoir computing.

How to test

Simulation was originally conducted using Julia.
The given inputs and the corresponding ideal outputs are provided as text files.
The test bench checks whether the circuit reproduces these results.

External hardware

A PCB board is sufficient.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input to reservoir [0]	Output from odd neuron [0]	Output from even neuron [0]
1	Input to reservoir [1]	Output from odd neuron [1]	Output from even neuron [1]
2	Input to reservoir [2]	Output from odd neuron [2]	Output from even neuron [2]
3	Input to reservoir [3]	Output from odd neuron [3]	Output from even neuron [3]
4	Input to reservoir [4]	Output from odd neuron [4]	Output from even neuron [4]
5	Input to reservoir [5]	Output from odd neuron [5]	Output from even neuron [5]
6	Input to reservoir [6]	Output from odd neuron [6]	Output from even neuron [6]
7	Input to reservoir [7]	Output from odd neuron [7]	Output from even neuron [7]

Andrew chip design

by **Andrew Marjanski**

0868 Wokwi Project

`github.com/andycandy15123/Andrew-Chip-design`

`wokwi.com/projects/445256658591419393`

"my project is very simple and essentially just a test of a button and a couple switches."

How it works

When the 1 switch is on, the clock is a 7. When the button is pressed, its also a 7 but looks different then the switch one When the 2 switch is on, the 1 is halved so only one piece of led is on. ## How to test

When the 1 switch is on, the clock is a 7. When the button is pressed, its also a 7 but looks different then the switch one When the 2 switch is on, the 1 is halved so only one piece of led is on. ## External hardware Push button seven segment display DIP switch 8

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	—	—
1	When this is one, number is 7. When off is 1	—	—
2	when on, 1 is halved	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Morse Code Detector (With Serial RX)

by **Yagnadatt Sarangi**

0869 | 2 Hz | HDL Project

`github.com/KerCrafter/morse_code_tto`

"The Morse Code Converter project translates Morse code sequences of dots and dashes into English alphabets (A–Z) and digits (0–9)."

How it works

This project simulates a Morse Code Transmitter and Receiver system implemented in Verilog HDL. It uses two Finite State Machines (FSMs) — one for transmission (trans_fsm) and another for reception (rec_fsm) — to encode and decode Morse code for alphabets (a–z) and digits (0–9).

Transmitter FSM (trans_fsm):

Converts button inputs (dot_inp, dash_inp, char_space_inp, word_space_inp) into a 3-bit encoded serial Morse signal (p_out).

Uses states like IDLE, DOT, DASH, CHAR_SPACE, and WORD_SPACE to send Morse sequences with proper timing gaps between symbols, characters, and words.

Receiver FSM (rec_fsm):

Samples the transmitted serial input (p_in) and reconstructs it into the corresponding 8-bit ASCII character (s_out).

Decoding is based on internal lookup tables that map Morse sequences to characters.

The two FSMs operate synchronously through the system clock (clk), ensuring correct timing and reset behavior through the active-low reset (rst).

How to test

Apply a reset pulse (rst = 0 → 1) to initialize both FSMs.

Provide input signals on the transmitter side:

dot_inp = 1 for a dot.

dash_inp = 1 for a dash.

char_space_inp = 1 for character spacing.

word_space_inp = 1 for word spacing.

Observe the transmitted Morse output (p_out) and verify it transitions according to the input sequence.

The receiver FSM should decode this serial Morse pattern and output the correct ASCII character on s_out.

Example: For Morse input corresponding to the letter A (dot, dash and character space), the output should be s_out = 8'h61 (ASCII for 'a').

External hardware

16x2 LCD FPGA Board

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	DOT_INP	CHAR_CODE_OUT[0]	—
1	DASH_INP	CHAR_CODE_OUT[1]	—
2	CHAR_SPACE_INP	CHAR_CODE_OUT[2]	—
3	WORD_SPACE_INP	CHAR_CODE_OUT[3]	—
4	—	CHAR_CODE_OUT[4]	—
5	—	CHAR_CODE_OUT[5]	—
6	—	CHAR_CODE_OUT[6]	—
7	RX_CW	CHAR_CODE_OUT[7]	—

Counter

by **Isak Hadi och Izabella Hagberg**

0870 HDL Project

`github.com/oscargus/ttsky-vhdl-makeAchip-izaha684`

"Saturating 4-bit counter with seven-segment output"

How it works

This is a four-bit counter with seven-segment display output. The direction is determined by input pin 0 and it will saturate at 15 when counting up and 0 when counting down.

How to test

Connect a clock and use input pin 0 to control direction. Seven-segment is reversed with binary point as the MSB.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	direction [count up || down]	7seg a	—
1	—	7seg b	—
2	—	7seg c	—
3	—	7seg d	—
4	—	7seg e	—
5	—	7seg f	—
6	—	7seg g	—
7	—	7seg h (dot)	—

dyno-tt

by **Mathis Salmen**

0871 | 50 MHz | HDL Project

`github.com/mathis-s/dyno-tt`

"Test project synthesized with Dyno-SV, a new open-source synthesis tool."

How it works

Design was synthesized to gate level using Dyno-SV by hand, using the command below

```
dyno-sv --liberty=sky130_fd_sc_hd.lib project_rtl.v -o=src/
project.v
```

How to test

Check if correct values are being output.

External hardware

None!

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input word bit 0	idx of first set bit, bit 0	idx of third set bit, bit 0
1	input word bit 1	idx of first set bit, bit 1	idx of third set bit, bit 1
2	input word bit 2	idx of first set bit, bit 2	idx of third set bit, bit 2
3	input word bit 3	idx of first set bit, valid	idx of third set bit, valid
4	input word bit 4	idx of second set bit, bit 0	counter bit 0
5	input word bit 5	idx of second set bit, bit 1	counter bit 1
6	input word bit 6	idx of second set bit, bit 2	input registered (1) or comb (0)
7	input word bit 7	idx of second set bit, valid	output registered (1) or comb (0)

Anton's-Hardware-Hack

by **Anton Tran**

0872 Wokwi Project

`github.com/anton02tran/Anton-s-Hardware-Hackday-Project`

`wokwi.com/projects/445254959452357633`

"Changes lights on the display."

How it works

When switch 0 or 1 are on, the number 2 is displayed on the sev_seg screen, when switch 5 and 6 are on, the . is displayed on the sev_seg screen. The other inputs/outputs I am not sure it how works.

How to test

I do not know how to test my project.

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	or1:A	or1: OUT	—
1	or1:B	or1: OUT	—
2	ttin:IN2	—	—
3	ttin:IN3	—	—
4	—	or1: OUT	—
5	and1: A	ttout:5	—
6	and2: B	or1: OUT	—
7	—	and1: OUT	—

4 hour CPU

by **Johannes Hoff**

0873 HDL Project

`github.com/johshoff/ttsky-25b`

"Last minute submission of a CPU"

How it works

A very simple CPU, mostly inspired by Ben Eater and the game turing complete. Started 4 hours before submission deadline for no good reason.

How to test

Use RP2xxx on testing board as a memory controller and run programs.

External hardware

RP2040/RP2350 on test board

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	8 bit input bus	8 bit output bus	n/a
1	8 bit input bus	8 bit output bus	n/a
2	8 bit input bus	8 bit output bus	n/a
3	8 bit input bus	8 bit output bus	n/a
4	8 bit input bus	8 bit output bus	n/a
5	8 bit input bus	8 bit output bus	n/a
6	8 bit input bus	8 bit output bus	n/a
7	8 bit input bus	8 bit output bus	n/a

SignalSorter

by **Jonathan Skorokhod-Adamov**

0874 Wokwi Project

github.com/goob30/UofT_Workshop_Chip

wokwi.com/projects/445256643197274113

"Detects and sorts signals of 2 systems with 2 units each, and detects if one or both units are sending a signal. To be used for error reporting."

How it works

Designed for error/failure reporting.

Receives signal from 4 units divided into 2 groups, A and B (0, 1 and 2, 3 respectively). Outputs 4 values from 0 - 3 depending on if either unit on group A or B has a signal (output 0 and 1) and if both units on groups A and B has a signal (output 2 and 3). The circuit is NO.

How to test

Simulate in Wokwi, and test DIP switches 1 through 4.

External hardware

N/A

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input 0	a partial	—
1	input 1	b partial	—
2	input 2	a full	—
3	input 3	b full	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Flappy VGA

by **Daniel Robinson**

0875 | 25 MHz | HDL Project

`github.com/ArsenoL/ICdesign`

"A simple flappy bird clone with a button input and 640x480 VGA output. 25MHz clock required."

How it works

There are three main modules in the design. The vgaControl module takes in the clock and outputs the horizontal and vertical sync signals, and provides the current pixel coordinate to the bitGen module. The gameControl module takes the button input and updates the game state (bird position, pipe position, hole position, score, etc) each frame. The bitGen module takes in the game state and pixel coordinate and outputs the color that the current pixel should be.

How to test

The clock input should be set to 25MHz (or 25.179MHz, either should be close enough). ui_in[0] should be connected to a debounced button that goes low when pressed. The VGA output is compatible with the Tiny VGA PMOD (https://tinytapeout.com/specs/pinouts/). Once everything is connected, a reset may need to be triggered before normal operation. An image with a yellow square, green pillars, and blue background should appear. Pressing the button should cause the bird to flap and start moving towards the pipes. The goal is to go through the gap in the pipes. Your score will count up in binary on the bidirectional pins each time you successfully make it through a pipe.

External Hardware

Some kind of VGA adapter needed. Compatible with Tiny VGA PMOD. Also needs a debounced button that goes low when pressed on ui_in[0].

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Button	R1	score[0]
1	—	G1	score[1]

#	Input	Output	Bidirectional
2	—	B1	score[2]
3	—	vsync	score[3]
4	—	R0	score[4]
5	—	G0	score[5]
6	—	B0	score[6]
7	—	hsync	score[7]

The Adder

by **Darius Pirnia**

0876 | 10 kHz | Wokwi Project

`github.com/D-man22/circuit-microchip`

`wokwi.com/projects/445265826672030721`

"this adds numbers in the first 4 and second 4 bits of input"

How it works

It adds 2 4 bit input together

How to test

You can mess arround with the swithces which is kinda fun to see what turns on!

External hardware

I used a bit of components like 1 resistor, 2 buttons, 1 2-input switch, 1 switch array

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	bit 1-1	out bit 1	—
1	bit 1-2	out bit 2	—
2	bit 1-3	out bit 3	—
3	bit 1-4	out bit 4	—
4	bit 2-1	out bit carry	—
5	bit 2-2	—	—
6	bit 2-3	—	—
7	bit 2-4	—	—

LFSR Driven Cryptography

by **Sai Surya**

0877 | 12.5 kHz | HDL Project

github.com/Sai-222777/TinyTapeout_Cipher

"A two-channel XOR stream cipher with fully programmable 32/16/8/4-bit Galois LFSRs"

Background

What is an XOR stream Cipher?

An XOR stream cipher is a type of symmetric-key encryption algorithm that encrypts data one bit or byte at a time, in a continuous stream. In a stream cipher, the encryption key is combined with a stream of pseudo-random bits or bytes to generate a sequence of cipher symbols, which are then combined with the plaintext message using the XOR (exclusive OR) operation to generate the ciphertext.

Unlike block ciphers, which encrypt fixed-size blocks of plaintext at a time, stream ciphers can encrypt data of any size in a continuous stream, making them well-suited for real-time communication applications such as wireless communications or internet telephony. Stream ciphers are also often faster and more efficient than block ciphers, since they require less computation and memory.

However, stream ciphers can be vulnerable to certain types of attacks, such as known-plaintext attacks or stream cipher resynchronization attacks, if the key stream is not properly generated or the same key stream is used multiple times.

Galois LFSR

An LFSR is a shift register whose output bit is a linear function of its previous state. In a Galois LFSR, the linear function is implemented using exclusive-OR (XOR) operations between selected bits of the register. The XOR feedback taps should be chosen in a way that generates a maximal length sequence, which is a pseudo-random sequence of bits that has the longest possible period for a given register length.

The Galois LFSR operates by shifting the bits in the register to the right, with the rightmost bit being discarded and a new bit being added at the leftmost position. The new bit is generated by performing XOR operations between selected bits of the register, as determined by the feedback taps. The sequence of bits generated by the LFSR is periodic, with the period being determined by the size of the register and the feedback taps used.

How it works

The design has two channels, one for **encryption** and another for **decryption**, each controlled by a separate enable pin. The plaintext or ciphertext bitstream is XORed with a bit derived from a register variable generated by a **Linear Feedback Shift Register (LFSR)**.

The LFSR is **programmable/customizable** to operate in **32-bit, 16-bit, 8-bit, or 4-bit modes**, depending on the select lines `sel0` and `sel1`. This provides flexibility in choosing the level of security and performance trade-off.

During **transmission**, the plaintext (`tx_p`) is encrypted into `tx_e`. For **reception**, the encrypted stream (`rx_e`) is decrypted back into the original plaintext (`rx_p`) using the same LFSR sequence. Debug outputs (`dbg_tx_p`, `dbg_rx_e`) are available for testing but are disabled by default.

The configuration input (`cfg_i`) and enable (`cfg_en`) allow dynamic loading of the 130-bit serial shift register, ensuring the LFSR sequence can be customized at runtime. The `cfg_o` pin provides read-back capability for verification.

How to test

A testbench is provided at `../test/tb_testing.v`.

- The primary input is assigned to the variable **`custom_input`**.
- The bit-width of this input can be adjusted by modifying the **`size`** variable.

Update these variables in the testbench, run the simulation.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
`0`	sel0 (Used for LSFR configuration)	tx_e (Encrypted bitstream for transmission)	—
1	sel1 (Used for LSFR configuration)	rx_p (Decrypted bitstream for reception)	—
2	tx_p (Plaintext bitstream for transmission)	dgb_tx_p (Decrypted transmit bitstream, debug pin disabled by default)	—
3	rx_e (Encrypted bitstream for reception)	dgb_rx_e (Encrypted receive bitstream, debug pin disabled by default)	—

#	Input	Output	Bidirectional
4	cfg_en (Active high configuration enable for the 130-bit serial shift register)	cfg_o (Configuration output from the 130-bit shift register)	—
5	cfg_i (Configuration input to the 130-bit serial shift register)	heartbeat0 (Bit [7] from the heartbeat counter)	—
6	tx_en (Transmit channel enable)	heartbeat1 (Bit [8] from the heartbeat counter)	—
7	rx_en (Receive channel enable)	heartbeat2 (Bit [9] from the heartbeat counter)	—

Tiny Tapeout Hradware Workshop Vaibhav

by **Vaibhav S Krishnan**

0878 | 10 kHz | Wokwi Project

github.com/VIP-10/Hardware-Hacks-Workshop

wokwi.com/projects/445175605912766465

"Test some logic gates and a clock divider"

How it works

This circuit simply takes each input signal and inverts it — if the input is ON (1), the output turns OFF (0), and vice versa. It's basically a set of OR gates used as a template to test and build more complex digital logic circuits.

How to test

Toggle the input switches (the left-side green blocks) and watch what happens on the output side (right-side green blocks / display). See what the default behavior is (likely simple inverters).

Replace the inverters with your own logic: for example, make a small gate network that implements a function (e.g., A AND B, or a 2-to-4 decoder).

Use a 7-segment display if available in the template to show numbers/letters based on inputs (there are example tutorials of BCD to 7-segment decoding)

External hardware

No external hadrware is used

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	d	d	—
1	d	—	—
2	—	—	—
3	—	—	—
4	—	—	—
5	—	—	—
6	—	—	—

#	Input	Output	Bidirectional
7	—	—	—

Verilog Multistage Oscillator with Enable and Counter

by **Remington Holder**, **Sherry Yu**, **Rohin Kumar**, **Julia Ke**, **Jeda Williams** & **Crystal Jiang**

0879 HDL Project

github.com/RemingtonHolder/tapeout_verilog

"ring oscillator with externally-selectable length, enable, and counter"

What is this?

This project is a Ring Oscillator with an enable and a built in 15 bit counter. At its the ring oscillator was build upon the work done by Anton Maurovic in tt09-ring-osc3 but was modified greatly to add extra features, inverters, and implement a fuller founder with a richer I/O scheme. The ring oscillator at the core also hakes use of the "tap-select" system designed by Anton Maurovic to change how many inverters are in the chain.

This uses Verilog to instantiate a ring of (an odd number of) inverter cells, then hooks them into each other to make an oscillator. At the end of this chain there is an and gate that works to kind of "enable" the output of the oscillator. After this there is a counter that iterates a count in a 15 bit register on the rising edge of the oscilation pulse. When the oscillator is turned off and the 0th ui_in pin is set to zero, the contents of the 15 bits of the counter are output over 15 of the output I/O Pins: [7:1] of the uo_out pin set for the LSB bits starting from pin one and moving to most significant near 7 and then continuing into [7:0] uio_out pin set for the MSB bits continuing from LSB at 0 to msb at 7. When enable is turned high again the counter is cleared and then begins its count of the oscillations once again.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	tap[0]	out[0]	—
1	tap[1]	out[1]	—
2	tap[2]	out[2]	—
3	—	out[3]	—
4	—	out[4]	—
5	—	out[5]	—

#	Input	Output	Bidirectional
6	—	out[6]	—
7	—	out[7]	—

Template copy

by **Patrick Kratky**

0897 | 10 kHz | Wokwi Project

`github.com/Patrick25100/GDS_create`

`wokwi.com/projects/442983567899298817`

"Test 7Segment display with Switches"

How it works

test

How to test

test

External hardware

switches, 7Segment

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT2	—
3	IN3	OUT3	—
4	IN4	OUT4	—
5	IN5	OUT5	—
6	IN6	OUT6	—
7	IN7	OUT7	—

Auberger tiny tapeout

by **Auberger Moritz**

0899 Wokwi Project

`github.com/Link00001/Tiny-Tapeout-auberger`

`wokwi.com/projects/442977465750697985`

"broken counter"

How it works

With a button you can turn on the four outputs one after another and then with two presses turn them out again.

How to test

Press the button at least six times.

External hardware

You ned something that generates current a button and something to display the outputs.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	CLK	OUT1	—
1	VDD	OUT2	—
2	—	OUT3	—
3	—	OUT4	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

7-segment-games

by **Michael Schurz**

0901 | 10 MHz | HDL Project

`github.com/userbanana47/7-segment-games`

"three selectable minigames with button input and 7-segment output"

How it works

few minigames with 7 segment output

How to test

press different buttons

External hardware

8 Buttons and a 7 segment display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	btn1	segment a1	—
1	btn2	segment b2	—
2	btn3	segment c3	—
3	btn4	segment d4	—
4	—	segment e5	—
5	—	segment f6	—
6	—	segment g7	—
7	btn_game_switch	—	—

Paircheck

by **Stefan V. Fellner**

0903 Wokwi Project

`github.com/FellSt220132/ttsky-wowki-fellner`

`wokwi.com/projects/442977476137901057`

"Checks if only one Pair of Switches is on (may change later if possible)"

How it works

Gates check if a Pair (1/2, 3/4, ...) is ON. Then the logic filters out, only lighting up the Display with "0" if there is only one pair is on.

How to test

The Switches are flipped manually in the Wowki Online Project, it should work like that on the chip, too

External hardware

The 7 Segment Display from the Template is used

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Input1	—	Output if ONE Pair is on
1	Input2	—	-//-
2	Input3	—	-//-
3	Input4	—	-//-
4	Input5	—	-//-
5	Input6	—	-//-
6	Input7	—	—
7	Input8	—	-//-

Chip

by **Hauser**

0905 Wokwi Project

`github.com/leo1hauser/Chip`

`wokwi.com/projects/442985235137668097`

"Chip for 7 Segment"

How it works

Through pressing the the right switch the right nummer should light up

How to test

For a seven segment display

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	INT1	OUT1	—
1	INT2	OUT2	—
2	INT3	OUT3	—
3	INT4	OUT4	—
4	INT5	OUT5	—
5	INT6	OUT6	—
6	INT7	OUT7	—
7	INT8	OUT8	—

Lauflicht_HTL_Leonding

by **Laurin Roider**

0907 | 10 kHz | Wokwi Project

github.com/HackerGecko/Lauflicht_HTL_Leonding

wokwi.com/projects/442980274206980097

"Ein Lauflicht"

How it works

Divides the 10kHz CLK into 2.44 Hz and into 0.3 Hz. With this diversion I connected a flank detector and flipflops together that it creats a 8bit chasing light.

How to test

Connect the output pins with leds and look at it.

External hardware

leds, resistors

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	—	LED0	—
1	—	LED1	—
2	—	LED2	—
3	—	LED3	—
4	—	LED4	—
5	—	LED5	—
6	—	LED6	—
7	—	LED7	—

My first Wokwi project

by **Yevhenii Konyshev**

0909 Wokwi Project

`github.com/ykonyshev/my-wokwi-project-from-tiny-tapeout`

`wokwi.com/projects/442977603880750081`

"A project that shows the first letter of my name when the correct number code is inputed"

How it works

My project requires a 7-segment display connected to the output of the circuit on the first 7 out pins. One needs to input the correct 8-digit binary number, the number being 137 decimal and 10001001 binary in my case, and set the input pins to the corresponding state to see the letter "Y" - the first letter of my name light up on the 7-segment display. 1 being a corresponding to high pin and 0 corresponding to a low pin.

How to test

To test my project one has to connect a 7-segment display to the output with the pins A-G connected to outputs 0-7. Afterwards, set the input pins to the correct values corresponding to the binary representation of the decimal number 137 with the lowest digit corresponding to the lowest-numbered pin: 1st digit - 0th pin, etc.

External hardware

A 7-segment display.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	1st digit of the binary number code	Pin connected to the SEG_A pin of the 7-segment display	—
1	2nd digit of the binary number code	Pin connected to the SEG_B pin of the 7-segment display	—

#	Input	Output	Bidirectional
2	3rd digit of the binary number code	Pin connected to the SEG_C pin of the 7-segment display	—
3	4th digit of the binary number code	Pin connected to the SEG_D pin of the 7-segment display	—
4	5th digit of the binary number code	Pin connected to the SEG_E pin of the 7-segment display	—
5	6th digit of the binary number code	Pin connected to the SEG_F pin of the 7-segment display	—
6	7th digit of the binary number code	Pin connected to the SEG_G pin of the 7-segment display	—
7	8th digit of the binary number code	—	—

KianV uLinux SoC

by **Hirosh Dabui**

0910 HDL Project

`github.com/TinyTapeout/KianV-RV32IMA-RISC-V-uLinux-SoC`

"A RISC-V ASIC that can boot µLinux."

How it works

32-bit RISC-V IMA processor, capable of booting Linux. Features 16 MiB of external SPI flash memory, 16 MiB of external PSRAM (8 MiB per bank), a UART peripheral, and a SPI peripheral.

System Memory Map

The system memory map is as follows:

Address	Size	Purpose
0x10000000	0x14	UART Peripheral
0x10500000	0x14	SPI Peripheral
0x10600000	0x0c	GPIO Peripheral
0x11100000	0x04	Reset / HALT control
0x20000000	16 MiB	SPI Flash
0x80000000	16 MiB	PSRAM

The system boots from the SPI flash memory. After reset, the CPU starts executing code from 0x20100000 (corresponding to the offset 0x100000 into the SPI flash memory), where the bootloader is expected to be.

UART Peripheral registers

Address	Name	Description
0x10000000	UART_DATA	Write to transmit, read to receive
0x10000005	UART_LSR	UART line status register
0x10000010	UART_DIV	Clock divider for UART baud rate

SPI Peripheral registers

Address	Name	Description
0x10500000	SPI_CTRL0	SPI Peripheral Control
0x10500004	SPI_DATA0	SPI Data

0x10500010	SPI_DIV	Clock divider for SPI peripheral

GPIO Peripheral registers

Address	Name	Description
0x10600000	GPIO_UO_EN	Enable bits for uo_out pins
0x10600004	GPIO_UO_OUT	Write to uo_out pins
0x10600008	GPIO_UI_IN	Read from ui_in pins(read-only)

CPU control register

Address	Name	Description
0x11100000	CPU_RESET	Write 0x7777 to reset the CPU, 0x5555 to halt the CPU.

How to test

Build the system image as described in the kianRiscV repo and load it into the SPI flash memory:

Flash offset	File name	Description
0x100000	`bootloader.bin`	Bootloader
0x180000	`kianv.dtb`	Device Tree Blob
0x200000	`Image`	Linux kernel + rootfs

The system runs at 30 MHz, with a maximum tested speed of 34.5 MHz.

External hardware

QSPI Pmod - can be purchased from the Tiny Tapeout store.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	gpio[0]	spi_cen0	ce0 flash
1	gpio[1]	spi_sclk0	sio0
2	spi_sio1_so_miso0	spi_sio0_si_mosi0	sio1
3	uart_rx	gpio[3]	sck
4	gpio[4]	uart_tx	sd2
5	gpio[5]	gpio[5]	sd3
6	gpio[6]	gpio[6]	cs1 psram

#	Input	Output	Bidirectional
7	gpio[7]	gpio[7]	cs2 psram

TimosChip

by **Timo Friesenecker**

0911 Wokwi Project

`github.com/FrieTi220044/AstroWorkShopChipDesign`

`wokwi.com/projects/442983115801432065`

"does smth"

How it works

bla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla blabla bla bla

How to test

bla bla bla

External hardware

bla bla bla

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input 0 makes a line and connected to an and	—	—
1	input 1 is also connected to the and	—	—
2	input 2 is connected to output 2	output from input 2	—
3	—	output and	—
4	—	—	—
5	—	output	—
6	—	—	—
7	—	output	—

Tiny Tapeout

by **Michael Miko**

0961 Wokwi Project

github.com/MichiMik0/Testing-WOKWI
wokwi.com/projects/442977846716359681

"no description yet"

How it works

it just works by existing

How to test

just do it :)

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	input 1	output 1	—
1	input 2	output 2	—
2	input 3	output 3	—
3	input 4	output 4	—
4	input 5	output 5	—
5	input 6	output 6	—
6	input 7	output 7	—
7	input 8	output 8	—

Tiny Tapeout Test

by **Alexander Binder**

0963 Wokwi Project

github.com/Alebi304/Tiny-Tapeout

wokwi.com/projects/442983760106473473

"Test 7Segment display and switches"

How it works

test

How to test

test

External hardware

7Segment, Buttons

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN0	OUT0	—
1	IN1	OUT1	—
2	IN2	OUT7	—
3	IN3	—	—
4	IN4	—	—
5	IN5	—	—
6	—	—	—
7	—	—	—

Traffic Light Controller

by **Maximilian Kernmaier**

0965 | 1 MHz | HDL Project

`github.com/Max00Ker/TinyTapeoutV1`

"A digital traffic light controller with pedestrian request handling and MAX7219 display interface."

How it works

This project implements a **digital traffic light controller** with integrated **pedestrian request logic** and **MAX7219 LED display output**.
The system coordinates car and pedestrian lights with precise timing, button debouncing, and a visual countdown before the pedestrian green phase.

In addition to the countdown, the LED matrices display **different smiley faces** corresponding to the current pedestrian signal:

- 😀 **Happy face** for green
- 😐 **Neutral face** for blinking green
- ☹ **Sad face** for red

If pedestrians have to wait during the red phase, they can trigger an **early green** phase by pressing the request button. The countdown will start immediately, and the request indicator light will turn on once the system acknowledges the input.
When the pedestrian phase is about to activate, the final **9 seconds** are displayed on the LED matrices driven by the MAX7219 interface.

The design operates on a **1 MHz input clock**. To verify its behavior, the circuit was simulated using **GTKWave** and additionally validated in **Wokwi** with a custom chip module (see the figure below).
This provides high confidence that the design will also function correctly when fabricated — or at least, that's the hope 🙂

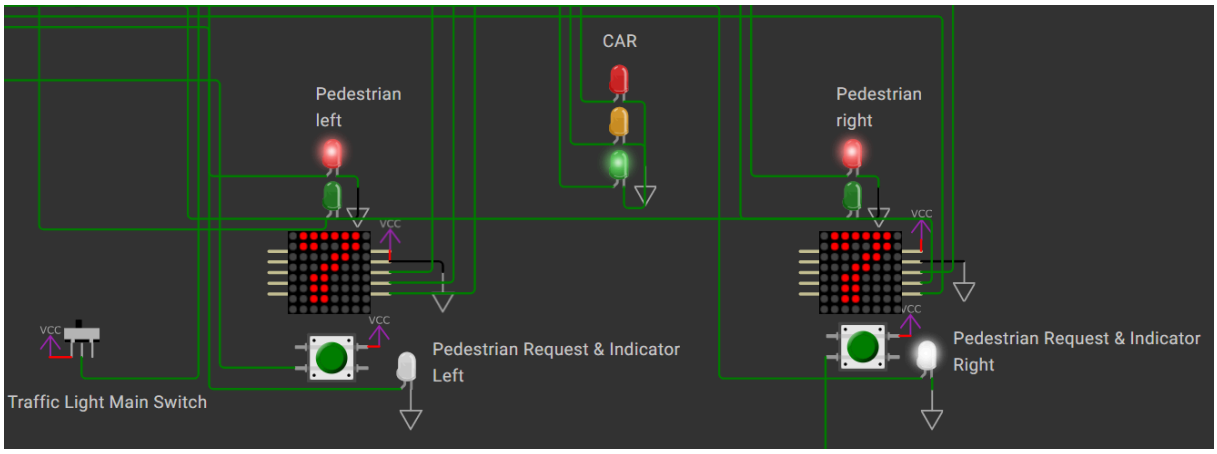


Figure 965.1: WokwiSimulation

The design is entirely written in **Verilog** and organized into several modules:

- A **Finite State Machine (FSM)** controlling light transitions and timing
- A **clock-enable divider** generating precise timing intervals
- A **dedicated MAX7219 driver** for configuration and data transfer
- A **SPI master** handling communication with the MAX7219

The full Verilog implementation and testbenches are available on GitHub.

This project demonstrates how real-world traffic-light logic can be implemented purely in Verilog —
including timing control, state management, and SPI-driven display output, all within a compact TinyTapeout design.

How to test

1. Connect LEDs to the car and pedestrian light output pins (`uo[0]`–`uo[6]`):
   - `uo[0]` → Car red
   - `uo[1]` → Car yellow
   - `uo[2]` → Car green
   - `uo[3]` → Pedestrian red (right)
   - `uo[4]` → Pedestrian green (right)
   - `uo[5]` → Pedestrian red (left)
   - `uo[6]` → Pedestrian green (left)
2. Connect switches or pushbuttons to the input pins:
   - `ui[0]` → Main ON/OFF switch
   - `ui[1]` → Pedestrian request (left)
   - `ui[2]` → Pedestrian request (right)
3. Connect two **MAX7219** LED modules via `uio[0:5]`:
   - `uio[0]` → DIN (left module)
   - `uio[1]` → CLK (left module)
   - `uio[2]` → CS (left module)
   - `uio[3]` → DIN (right module)

- `uio[4]` → CLK (right module)
- `uio[5]` → CS (right module)
- `uio[6]` → Pedestrian request indicator (left)
- `uio[7]` → Pedestrian request indicator (right)

4. Power on the system and observe the **automatic light sequence** and the **countdown timing** on the displays.
   Also verify the operation of the **request indicators** and the **main power switch**.
5. The design can also be simulated using **GTKWave** together with the testbench `traffic_light_tb.v` located in the `test` folder.
   Make sure the global define `SIM` is enabled before running the simulation.

External hardware

- LEDs for car, pedestrian, and early-request indicators
- Pushbuttons for pedestrian requests
- A toggle switch for the main power control
- Two **MAX7219 LED matrix modules (FC16 type)** for the visual countdown display

---

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	Main ON/OFF Switch	Car red light	DIN (MAX7219 left)
1	Pedestrian request button (left)	Car yellow light	CS (MAX7219 left)
2	Pedestrian request button (right)	Car green light	SCLK (MAX7219 left)
3	—	Pedestrian red light (left)	DIN (MAX7219 right)
4	—	Pedestrian green light (left)	CS (MAX7219 right)
5	—	Pedestrian red light (right)	SCLK (MAX7219 right)
6	—	Pedestrian green light (right)	Pedestrian request indicator (left)
7	—	—	Pedestrian request indicator (right)

Piffl first Wokwi project

by **Piffl Lorenz**

0967 Wokwi Project

`github.com/PifflL/tinyTapeoutInitial`

`wokwi.com/projects/442983721200099329`

"It is a logic gate based currcuit, where, if you choose to flip the correct switches you get the initial of my first name L on the 7-Seg."

How it works

I use the DIP Switch 8. If you switch on the first, second, fith, eighth switch you change something in the currcuit. The first end second go into a XOR-gate. The fifth and eighth go into an AND-gate. The outpit of those two go into an AND-gate as well. The output of this one goes to the fourth to sixth output to give out my initial L on the Seven Segment Display. ## How to test

Switch on and of the four mantioned switches. As long as only one of the first switches and both of the later ones are active you should get the correct output.

External hardware

DIP Switch 8, Seven Segment Display

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	IN1	—	—
1	IN2	—	—
2	IN3	—	—
3	IN4	OUT3	—
4	IN5	OUT4	—
5	IN6	OUT5	—
6	IN7	—	—
7	IN8	—	—

8-Bit ALU

by **Timon Strassern**

0969 | 50 MHz | HDL Project

github.com/StrasTi/8-Bit-ALU

"8-Bit arithmetic logic unit perfomring arithmetic and bitwise operations on integer binary numbers."

How it works

8-Bit ALU with state machine to read A and B inputs using the 8 digital pins. Operation control over the In/Out pins. Status out over the In/Out pins.

How to test

TODO: Test explanation.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	in0	ou0	s0
1	in1	ou1	s1
2	in2	ou2	carry_borrow
3	in3	ou3	f0
4	in4	ou4	f1
5	in5	ou5	f2
6	in6	ou6	f3
7	in7	ou7	f4

KianV uLinux RISC-V regfile edition

by **Hirosh Dabui**

0970 HDL Project

`github.com/TinyTapeout/ttsky25b-kianv-linux-soc-with-regfile`

"A RISC-V ASIC that can boot µLinux, including a 32x32 register file macro from tnt."

How it works

32-bit RISC-V IMA processor, capable of booting Linux. Features 16 MiB of external SPI flash memory, 16 MiB of external PSRAM (8 MiB per bank), a UART peripheral, and a SPI peripheral.

System Memory Map

The system memory map is as follows:

Address	Size	Purpose
0x10000000	0x14	UART Peripheral
0x10500000	0x14	SPI Peripheral
0x10600000	0x0c	GPIO Peripheral
0x11100000	0x04	Reset / HALT control
0x20000000	16 MiB	SPI Flash
0x80000000	16 MiB	PSRAM

The system boots from the SPI flash memory. After reset, the CPU starts executing code from 0x20100000 (corresponding to the offset 0x100000 into the SPI flash memory), where the bootloader is expected to be.

UART Peripheral registers

Address	Name	Description
0x10000000	UART_DATA	Write to transmit, read to receive
0x10000005	UART_LSR	UART line status register
0x10000010	UART_DIV	Clock divider for UART baud rate

SPI Peripheral registers

Address	Name	Description
0x10500000	SPI_CTRL0	SPI Peripheral Control

0x10500004	SPI_DATA0	SPI Data
0x10500010	SPI_DIV	Clock divider for SPI peripheral

GPIO Peripheral registers

Address	Name	Description
0x10600000	GPIO_UO_EN	Enable bits for uo_out pins
0x10600004	GPIO_UO_OUT	Write to uo_out pins
0x10600008	GPIO_UI_IN	Read from ui_in pins(read-only)

CPU control register

Address	Name	Description
0x11100000	CPU_RESET	Write 0x7777 to reset the CPU, 0x5555 to halt the CPU.

How to test

Build the system image as described in the kianRiscV repo and load it into the SPI flash memory:

Flash offset	File name	Description
0x100000	`bootloader.bin`	Bootloader
0x180000	`kianv.dtb`	Device Tree Blob
0x200000	`Image`	Linux kernel + rootfs

The system runs at 30 MHz, with a maximum tested speed of 34.5 MHz.

External hardware

QSPI Pmod - can be purchased from the Tiny Tapeout store.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	gpio[0]	spi_cen0	ce0 flash
1	gpio[1]	spi_sclk0	sio0
2	spi_sio1_so_miso0	spi_sio0_si_mosi0	sio1
3	uart_rx	gpio[3]	sck
4	gpio[4]	uart_tx	sd2
5	gpio[5]	gpio[5]	sd3

#	Input	Output	Bidirectional
6	gpio[6]	gpio[6]	cs1 psram
7	gpio[7]	gpio[7]	cs2 psram

simple adder and subtractor

by **Alexander Brandstetter**

0971 Wokwi Project

github.com/Aether-Monarch/Tiny-Tapeout-Chip

wokwi.com/projects/442977509323262977

"full adder and full subtactor with 2 4bit registers"

How it works

2 4 bit registers fed into a full adder and a full subtractor, the output is chosen via mux

How to test

in input 0-3 give the data in binary, swich registers with input 4, choose the output with input 7

External hardware

none required, however to see the output leds are quite nice, to choose the input switches are also a qol feature

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	data in 1	data out 1	—
1	data in 2	data out 2	—
2	data in 3	data out 3	—
3	data in 4	data out 4	—
4	select register to write 1 or 2, logic 1 means write register 1, logic 0 means write register 2	data out 5, for addition	—
5	—	—	—
6	—	—	—
7	choose between add and subtract, add is logic 1, subtract is logic 0	overflow for subract, meaning below 0	—

The Ultimate ASIC

by **Clyde Laforge**

0973 Wokwi Project

github.com/Scafir/ttsky-ultimate-asic

wokwi.com/projects/442977852139597825

"The ultimate ASIC - does what any machine should"

How it works

Very simple - as any good device should, it simply turns itself off whenever it is turned on.

How to test

Press the button, which will turn on the corresponding LED. The device should sweep the LEDs and remove the offender.

External hardware

8 press buttons for the input. 8 leds for the output.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	button0	led0	—
1	button1	led1	—
2	button2	led2	—
3	button3	led3	—
4	button4	led4	—
5	button5	led5	—
6	button6	led6	—
7	button7	led7	—

TinyTapeStation

by **James Ashie Kotey**

0974 | 25 MHz | HDL Project

`github.com/Sheffield-Chip-Design-Team/TinyTapeStation`

"The Tiny Tapestation is SHARC most ambition project to date .. TODO"

How it works

TapeStation is played using a SNES or NES controller connected to the gamepad peripheral. The gamepad peripheral reads the controller's digital inputs and exposes them to the core, which maps buttons to game actions and supports standard retro layouts and simple remapping.

The Console has a single pre-loaded game called 'Gnome Knight'. The aim of Gnome Knight is to defeat a serepentine dragon terrorising your village. With your trusty sword, smite the dragon (A button) and avoid his cunning attacks (D-pad) to protect the village's sheep and restore peace to the land.

The combination of authentic controller input, low-latency VGA video, and discrete sound output provides a complete retro gaming experience!

How to test

- Connect the SNES/NES controller to the gamepad PMOD port.
- Connect the VGA peripheral cable to a monitor.
- Connect audio output from the sound peripheral to speakers or an amp.
- Power the system; the core detects peripherals, and uses the controller inputs for play.

External hardware

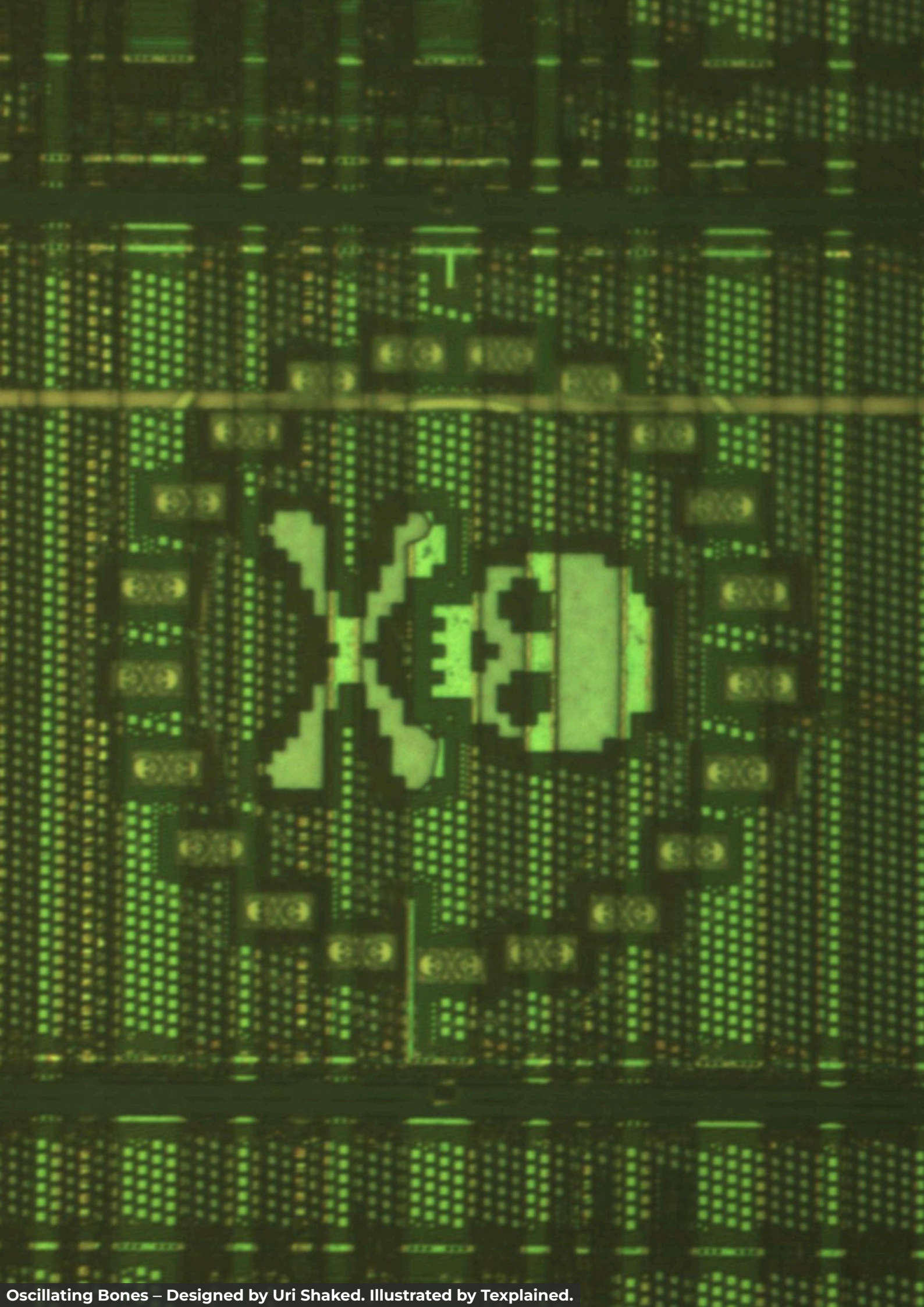
List external hardware used in your project (e.g. PMOD, LED display, etc), if any The Gamepad PMOD can be used to connect a SNES-compatible gamepad. A NES Controller can be used by directly connecting the relevant pins to the board pins. The VGA peripheral outputs 640x480, 60fps video at to any compatible monitor. The audio peripheral provides analogue audio (mono/stereo) to an on-board buzzer or 3.5mm headphone jack.

Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	NES_DATA	R_HI	—

#	Input	Output	Bidirectional
1	—	G_HI	NES_CLK
2	—	B_HI	NES_LATCH
3	—	V_SYNC	—
4	SNES_PMOD_Latch	R_LO	—
5	SNES_PMOD_Clk	G_LO	—
6	SNES_PMOD_Data	B_LO	—
7	—	H_SYNC	PWM



Oscillating Bones – Designed by Uri Shaked. Illustrated by Texplained.

Test project

by **Yuusha**

0975 | 1 kHz | Wokwi Project

`github.com/yuushabio/tiny-tapeout-gds-export`
`wokwi.com/projects/442984105081776129`

"This is a test project"

How it works

This is a test project

How to test

There are not io pins

External hardware

List external hardware used in your project (e.g. PMOD, LED display, etc), if any

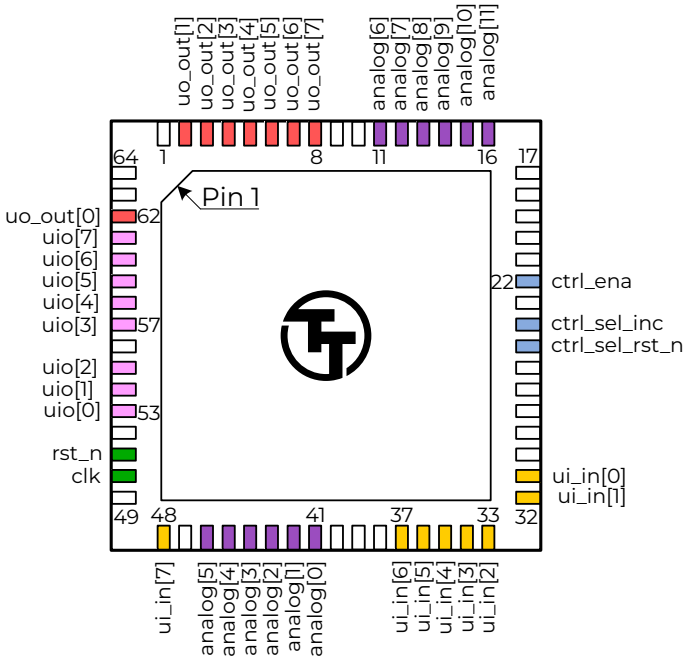
Project Pinout

Digital Pins

#	Input	Output	Bidirectional
0	a	nota	—
1	b	notb	—
2	c	notc	—
3	d	notd	—
4	—	—	—
5	—	—	—
6	—	—	—
7	—	—	—

Pinout

The chip is packaged in a 64-pin QFN package. The pinout is shown below.



Bottom View

Note: You will receive the chip mounted on a breakout board (github.com/tinytapeout/breakout-pcb). The pinout is provided for advanced users, as most users will not need to solder the chip directly.

The Tiny Tapeout Multiplexer

Overview

The Tiny Tapeout Multiplexer distributes a single set of user IOs to multiple user designs. It is the backbone of the Tiny Tapeout chip.

It has the following features:

- 10 dedicated inputs
- 8 dedicated outputs
- 8 bidirectional outputs
- Supports up to 512 user designs (32 mux units, each with up to 16 designs)
- Designs can have different sizes. The basic unit is called a tile, and each design can occupy up to 16 tiles.

Operation

The multiplexer consists of three main units:

1. The controller — used to set the address of the active design
2. The spine — a bus that connects the controller with all the mux units
3. Mux units — connects the spine to individual user designs

The Controller

The mux controller has 3 input lines:

Input	Description
`ena`	Sent as-is (buffered) to the downstream mux units
`sel_rst_n`	Resets the internal address counter to 0 (active low)
`sel_inc`	Increments the internal address counter by 1

It outputs the address of the currently selected design on the `si_sel` port of the spine (see below).

For instance, to select the design at address 12, you need to pulse `sel_rst_n` low, and then pulse `sel_inc` 12 times:

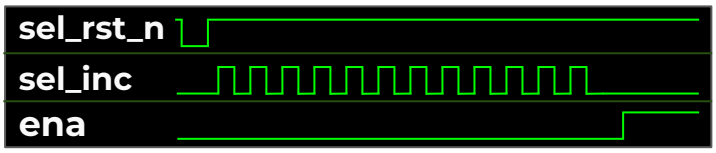


Figure 1: Mux signals for activating the design at address 12

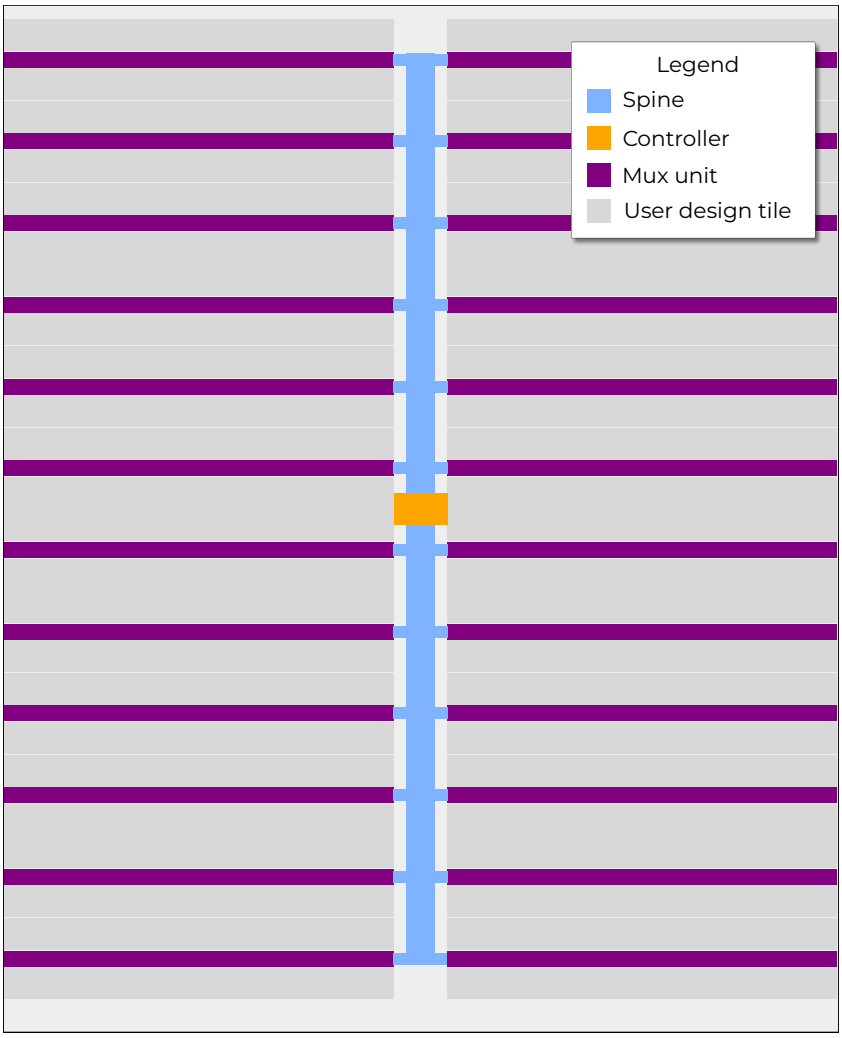


Figure 2: Mux Diagram

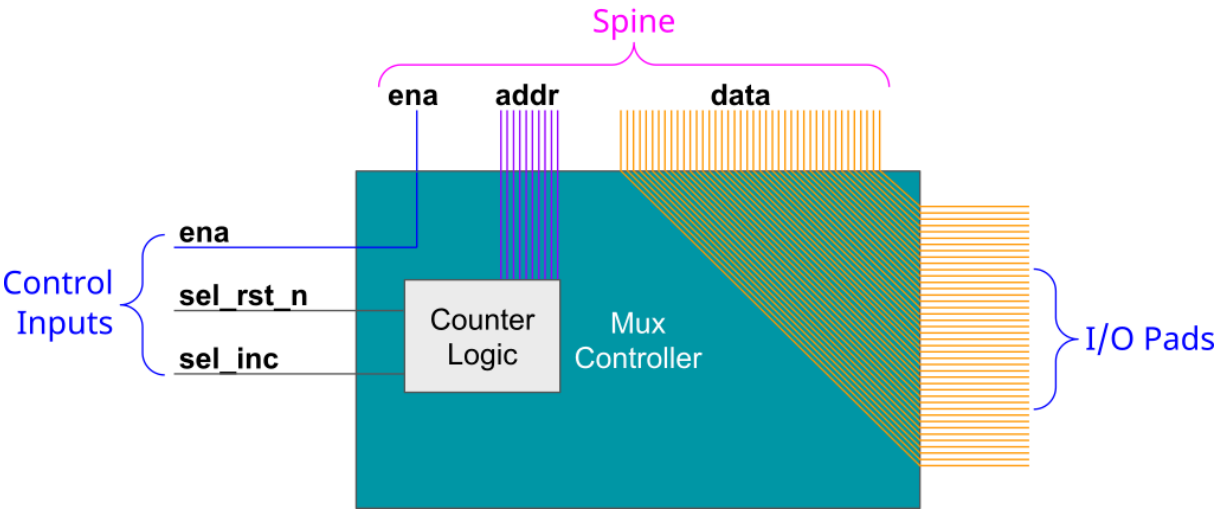


Figure 3: Mux Controller Diagram

Internally, the controller is just a chain of 10 D-flip-flops. The `sel_inc` signal is connected to the clock of the first flip-flop, and the output of each flip-flop is connected to the clock of the next flip-flop. The `sel_rst_n` signal is connected to the reset of all flip-flops.

The following Wokwi project demonstrates this setup: wokwi.com/projects/36434780766. It contains an Arduino Nano that decodes the currently selected mux address and displays it on a 7-segment display. Click on the button labeled `RST_N` to reset the counter, and click on the button labeled `INC` to increment the counter.

The Spine

The controller and all muxes are connected together through the spine. The spine has the following signals going to it:

From controller to mux:

- `si_ena` — the `ena` input
- `si_sel` — selected design address (10 bits)
- `ui_in` — user clock, user `rst_n`, user_inputs (10 bits)
- `uio_in` — bidirectional I/O inputs (8 bits)

From mux to controller:

- `uo_out` — user outputs (8 bits)
- `uio_oe` — bidirectional I/O output enable (8 bits)
- `uio_out` — bidirectional I/O outputs (8 bits)

The only signal which is actually generated by the controller is `si_sel` (using `sel_rst_n` and `sel_inc`, as explained above). The other signals are just going through from/to chip I/O pads.

The Multiplexer

Each mux branch is connected to up to 16 designs. It also has 5 bits of a hard-coded address (each unit gets assigned a different address, based on its position on the die).

The mux implements the following logic: If `si_ena` is 1, and `si_sel` matches the mux address, we know the mux is active. Then, it activates the specific user design port that matches the remaining bits of `si_sel`.

For the active design:

- `clk`, `rst_n`, `ui_in`, `uio_in` are connected to the respective pins coming from the spine (through a buffer)
- `uo_out`, `uio_oe`, `uio_out` are connected to the respective pins going out to the spine (through a tristate buffer)

For all others, inactive designs (including all designs in inactive muxes):

- `clk`, `rst_n`, `ui_in`, `uio_in` are all tied to zero
- `uo_out`, `uio_oe`, `uio_out` are disconnected from the spine (the tristate buffer output enable is disabled)

Pinout

QFN64 Pin	Function	Signal
1	Mux Control	`ctrl_ena`
2	Mux Control	`ctrl_sel_inc`
3	Mux Control	`ctrl_sel_rst_n`
4	Reserved	(none)
5	Reserved	(none)
6	Reserved	(none)
7	Reserved	(none)
8	Reserved	(none)
9	Output	`uo_out[0]`
10	Output	`uo_out[1]`
11	Output	`uo_out[2]`
12	Output	`uo_out[3]`
13	Output	`uo_out[4]`
14	Output	`uo_out[5]`
15	Output	`uo_out[6]`
16	Output	`uo_out[7]`
17	Power	VDD IO
18	Ground	GND IO
19	Analog	`analog[0]`
20	Analog	`analog[1]`
21	Analog	`analog[2]`
22	Analog	`analog[3]`
23	Power	VAA Analog
24	Ground	GND Analog
25	Analog	`analog[4]`
26	Analog	`analog[5]`
27	Analog	`analog[6]`
28	Analog	`analog[7]`
29	Ground	VDD Core
30	Power	VDD Core
31	Ground	GND IO
32	Power	VDD IO
33	Bidirectional	`uio[0]`
34	Bidirectional	`uio[1]`
35	Bidirectional	`uio[2]`

QFN64 Pin	Function	Signal
36	Bidirectional	`uio[3]`
37	Bidirectional	`uio[4]`
38	Bidirectional	`uio[5]`
39	Bidirectional	`uio[6]`
40	Bidirectional	`uio[7]`
41	Input	`ui_in[0]`
42	Input	`ui_in[1]`
43	Input	`ui_in[2]`
44	Input	`ui_in[3]`
45	Input	`ui_in[4]`
46	Input	`ui_in[5]`
47	Input	`ui_in[6]`
48	Input	`ui_in[7]`
49	Input	`rst_n` †
50	Input	`clk` †
51	Ground	GND IO
52	Power	VDD IO
53	Analog	`analog[8]`
54	Analog	`analog[9]`
55	Analog	`analog[10]`
56	Analog	`analog[11]`
57	Ground	GND Analog
58	Power	VDD Analog
59	Analog	`analog[12]`
60	Analog	`analog[13]`
61	Analog	`analog[14]`
62	Analog	`analog[15]`
63	Ground	GND Core
64	Power	VDD Core

† Internally, there's no difference between `clk`, `rst_n` and `ui_in` pins. They are all just bits in the `pad_ui_in` bus. However, we use different names to make it easier to understand the purpose of each signal.

Special Thanks

Thank you to ChipFoundry for sponsoring this shuttle.

Team

Tiny Tapeout would not be possible without a lot of people helping. We would especially like to thank:

- **Uri Shaked** for Wokwi development and lots more
- **Patrick Deegan** for PCBs, software, documentation and lots more
- **Sylvain Munaut** for help with scan chain improvements
- **Mike Thompson** and **Mitch Bailey** for verification expertise
- **Tim Edwards** and **Harald Pretl** for ASIC expertise
- **Jix** for formal verification support
- **Proppy** for help with GitHub actions
- **Maximo Balestrini** for all the amazing renders and the interactive GDS viewer
- **James Rosenthal** for coming up with digital design examples
- All the **people who took part in TinyTapeout 01** and volunteered time to improve docs and test the flow
- The **team at YosysHQ** and **all the other open source EDA tool makes**
- **Jeff** and the **Efabless Team** for running the shuttles and providing OpenLane and sponsorship
- **Tim Ansell** and **Google** for supporting the open source silicon movement
- **Zero to ASIC course community** for all your support
- **Jeremy Birch** for help with STA

Using This Datasheet

Structure

Projects are ordered by their mux address, in ascending order. Documentation is user-provided from their GitHub repositories and are merged into the final shuttle once the deadline is reached.

In general, each project should contain:

- The user-provided title & a list of authors
- A link to the GitHub repository used for submission
- A link to the Wokwi project (if applicable)
- A "How it works" section
- A "How to test" section
- An "External hardware" section (if applicable)
- A pinout table for both digital & analog designs

Badges

This datasheet uses "badges" to quickly convey some information about the content. These badges are explained in the table below.

Badge	Description
Artwork	Used to showcase artwork from our community.
0123 0423/2	Mux address of the project, in decimal. For microtile designs, their sub-address is placed after the forward slash. In this example, it would be 2.
25.175 MHz	Clock frequency of the project. May be truncated from actual value or omitted completely.
HDL Project Wokwi Project Analog Project	Project type, indicating if it was made with a HDL, Wokwi, or if it is analog.
Medium Danger High Danger	Indicates the risk that the project presents to the ASIC. Medium danger projects can damage the ASIC under certain conditions, whilst high danger projects *will* damage the ASIC.

Callouts

In addition to `Medium Danger` and `High Danger` badges being used, a callout is placed before the project documentation begins to alert the user.

A callout for `Medium Danger` may look something like:

This project will damage the ASIC under certain conditions.

There is an error in the schematic which may lead to ASIC failure under certain clocking conditions.

Similarly, a callout for `High Danger` may look something like:

This project will damage the ASIC.

There is an error in the schematic which may cause permanent damage when powered on in a certain configuration.

Should there be a project that poses a danger, the callout will explain the reasoning behind the danger level.

Callouts may also provide some additional information, and look something like so:

Information

Silicon melts at 1414°C, and boils at 3265°C. Don't let your chip get too hot!

Figures & Footnotes

Numbering for figures and footnotes within the "Project" chapter is formed by combining the address of the project with the current figure number. For example, the second figure for a project with an address of 256 will be captioned with "Figure 256.2". Likewise, the third footnote for a project of address 128 will be shown as "128.3".

The numbering outside of the "Project" chapter resumes as normal, being formatted with a simple number, e.g. "Figure 3".

Updates

This datasheet is intended to be a living and breathing document. Please update your projects' datasheet with new information if you have it, by creating a pull request against the shuttle repository.

Where is *your* design?

Go from idea to chip design in minutes, without breaking the bank.

Interested and want to get your own design manufactured? Visit our website and check out our educational material and previous submissions!

How?

New to this? Use our basic Wokwi template to see what's possible. If you're ready for more, use our advanced Wokwi template and unlock some extra pins.

Know Verilog and CocoTB? Get stuck in with our HDL templates.

When?

Multiple shuttles are run per year, meaning you've got an opportunity to manufacture your design at any time.

Stuck? Need help? Want inspiration?

Come chat to us and our community on Discord! Scan the QR code below.

Website	Digital design guide	Discord server
tinytapeout.com	tinytapeout.com/digital_design	tinytapeout.com/discord