BCD to 7-segment encoder

- Author maehw
- Description Encode binary coded decimals (BCD) in the range 0..9 to 7-segment display control signals
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware Input switches and 7-segment display (should be on the PCB)

How it works

The design has been fully generated using https://github.com/maehw/wokwi-lookup-table-generator using a truth table (https://github.com/maehw/wokwi-lookup-table-generator/blob/main/demos/bcd_7segment_lut.logic.json). The truth table describes the translation of binary coded decimal (BCD) numbers to wokwi 7-segment display (https://docs.wokwi.com/parts/wokwi-7segment). Valid BCD input values are in the range 0..9, other values will show a blank display.

How to test

Control the input switches on the PCB and check the digit displayed on the 7-segment display.

IO

#	Input	Output
0	w	segment a
1	x	segment b
2	y	segment c
3	z	segment d
4	none	segment e
5	none	segment f
6	none	segment g
7	none	none

chase the beat

- Author Emil J Tywoniak
- Description Tap twice to the beat, the outputs will chase the beat. Or generate some audio noise!
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware A button on the tap input, a switch on the mode input, LEDs on the 8 outputs, and audio output on the first output. Don't just connect headphones or speakers directly! It could fry the circuit, you need some sort of amplifier.

How it works

The second button press sets a ceiling value for the 1kHz counter. When the counter hits that value, it barrel-shifts the outputs by one bit. When the mode pin isn't asserted, the first output pin emits digital noise generated by a LFSR

How to test

Set 1kHz clock on first input. After reset, set mode to 1, tap the tap button twice within one second. The outputs should set to the beat

IO

#	Input	Output
0	clk	o_0 - LED or noise output
1	rst	o_1 - LED
2	tap	o_2 - LED
3	mode	o_3 - LED
4	none	o_4 - LED
5	none	o_5 - LED
6	none	o_6 - LED
7	none	o_7 - LED

LUTRAM

- Author Luis Ardila
- Description LUTRAM with 4 bit address and 8 bit output preloaded with a binary to 7 segments decoder, sadly it was too big for 0-F, so now it is 0-9?
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

uses the address bits to pull from memory the value to be displayed on the 7 segments, content of the memory can be updated via a clock and data pins, reset button will revert to default info, you would need to issue one clock cycle to load the default info

How to test

clk, data, rst, nc, address [4:0]

IO

#	Input	Output
0	clock	segment a
1	data	segment b
2	reset	segment c
3	nc	segment d
4	address bit 3	segment e
5	address bit 2	segment f
6	address bit 1	segment g
7	address bit 0	segment pd

Configurable SR

- Author Greg Steiert
- Description Two configurable gates driving an SR flip-flop
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware none

How it works

Connect to each of the configurable gates to control the SR FF

How to test

set up inputs like 1G99 configurable gate to drive SR FF

IO

#	Input	Output
0	SA	Smux
1	SB	Sxor
2	Ssel	Q
3	Sinv	none
4	RA	Rmux
5	RB	Rxor
6	Rsel	Qbar
7	Rinv	none

tiny-fir

- Author Tom Schucker
- Description 4bit 2-stage FIR filter
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware Arduino or FPGA

How it works

Multiplies the input by the tap coefficient for each stage and outputs the sum of all stages

How to test

Load tap coefficients by setting the value and pulsing 2 times, then repeat for second tap. Change input value each clock to run filter. Select signals change output to debug 00(normal) 01(output of mult 2) 10(tap values in mem) 11(output of mult 1). FIR output discards least significant bit due to output limitations

IO

#	Input	Output
0	clock	fir1/mult0/tap10
1	data0/tap0	fir2/mult1/tap11
2	data1/tap1	fir3/mult2/tap12
3	data2/tap2	fir4/mult3/tap13
4	data3/tap3	fir5/mult4/tap20
5	select0	fir6/mult5/tap21
6	select1	fir7/mult6/tap22
7	loadpulse	fir8/mult7/tap23

Stream Integrator

- Author William Moyes
- Description A silicon implementation of a simple optical computation
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

It is possible to generate a pseudorandom bit sequence optomechanically using four loops of punched paper tape. Each of the four tape loops, labeled A, B, C, and D, encodes one bit of information per linear position using a tape specific hole pattern. The patterns are TapeA_0=XOOO, TapeA_1=OXOO, TapeB_0=OOXO, TapeB_1=OOOX, TapeC_0=OOXX, TapeC_1=XXOO, TapeD_0=OXOX, TapeD_1=XOXO, where O is a hole, and X is filled. The pseudorandom sequence is obtained by physically stacking the four tapes together at a single linear point, and observing if light can pass through any of the four hole positions. If all four hole positions are blocked, a 0 is generated. If any of the four holes allows light to pass, a 1 is generated. The next bit is obtained by advancing all four tapes by one linear position and repeating the observation. By using the specified bit encoding patterns, the expression (C ? A : B) ^ D is calculated. If all four tapes are punched with randomly chosen 1 and 0 patterns, and each tape's length is relatively prime to the other tape lengths, then a maximum generator period is obtained. This TinyTapeout-02 minimal project was inspired by the paper tape pseudorandom bit sequence generator. It implements the core (C ? A : B) ^ D operation electrically instead of optomechanically. An extra ^ E term is added for ease of use.

How to test

Run through the 32 possible input patterns, and verify the expected output value is observed. Counting from 00000 to 111111, where IN0 is the LSB (i.e. Tape A), and IN4 (i.e. Extra E) is the MSB should yield the pattern: 01010011101011001010110001010011.

IO

#	Input	Output
0	Value from Tape A	Output
1	Value from Tape B	none
2	Value from Tape C	none
3	Value from Tape D	none

#	Input	Output
4	Extra term XORed with generator output	none
5	none	none
6	none	none
7	none	none

small FFT

- Author Rice Shelley
- Description Computes a small fft
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware

How it works

Takes 4 4-bit signed inputs (real integer numbers) and outputs 4 6-bit complex numbers

How to test

after reset, use the write enable signal to write 4 inputs. Read the output for the computer FFT.

IO

#	Input	Output
0	clock	rd_idx_zero
1	reset	none
2	wrEn	data_out_0
3	none	data_out_1
4	data_in_0	data_out_2
5	data_in_1	data_out_3
6	data_in_2	data_out_4
7	data_in_3	data_out_5

Avalon Semiconductors '5401' 4-bit Microprocessor

- Author Tholin
- Description 4-bit CPU capable of addressing 4096 bytes program memory and 256 words data memory. Hopefully capable of more complex computation than previous CPU submissions.
- GitHub project
- Wokwi project
- Extra docs
- Clock 2 Hz
- External hardware TODO

How it works

TODO

How to test

TODO

IO

#	Input	Output
0	CLK	D0_RR0
1	RST	D1_RR1
2	D0	D2_RR2
3	D1	D3_RR3
4	D2	D4_MAR
5	D3	D5_WRITE
6	EF0	D6_JMP
7	EF1	D7_I

Ising Spin

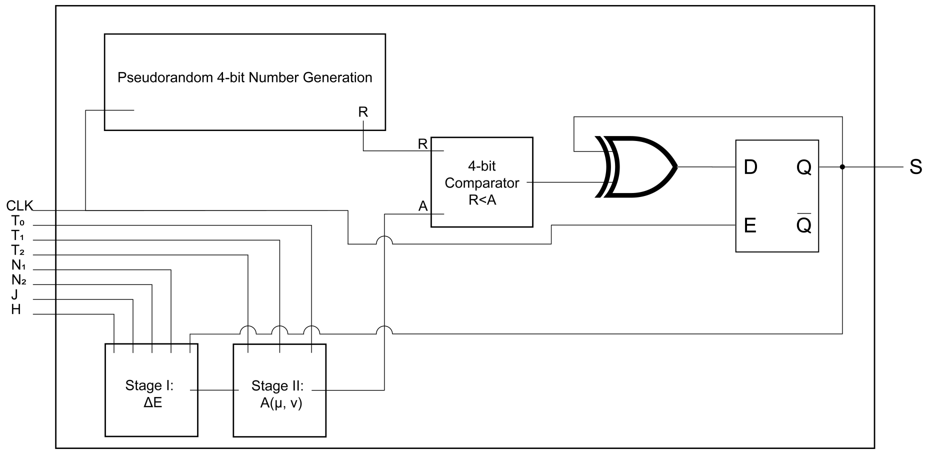

Figure 1: picture

- Author Seppe Van Dyck
- Description Circuit that simulates 4 spins of an Ising chain in an external field.
- GitHub project
- Wokwi project
- Extra docs
- Clock 20 Hz
- External hardware

How it works

It runs the Metropolis-Hastings monte-carlo algorithm to simulate 4 Ising spins in a linear chain with two external neighbours and an external field. Every monte-carlo step (10 clock cycles) a random number is created through a 32-bit LFSR and is compared to an 8-bit representations of the acceptance probability of a random spin flip.

How to test

When you only enable one of the neighbours and leave everything else 0, the system will evolve into a ground state with every other spin pointing up.

IO

#	Input	Output
0	clock, clock input.	segment a, Spin 0 (1 if spin is up).
1	T0, LSB of the 3-bit temperature representation.	segment b, Spin 1 (1 if spin is up).
2	T1, Middle bit of the 3-bit temperature.	segment c, Spin 2 (1 if spin is up).
3	T2, MSB of the 3-bit temperature.	segment d, Spin 3 (1 if spin is up).
4	N1, Value of neighbour 1 (up/1 or down/0).	segment e, Neighbour 2 (1 if spin is up).
5	N2, Value of neighbour 2 (up/1 or down/0).	segment f, Neighbour 1 (1 if spin is up).
6	J, The sign of the NN coupling constant J (+/1 or -/0).	none
7	H, Value of the coupling to the external field H (1 or 0).	segment h, blinks every Monte Carlo step.

Trafficlight

- Author Jens Schleusner
- Description A state machine to control german traffic lights at an intersection.
- GitHub project
- Wokwi project
- Extra docs
- Clock 1 Hz
- External hardware Inverter for pedestrian red signals + LEDs for the signals

How it works

A state machine generates signals for vehicle and pedestrian traffic lights at an intersection. Shows blinking yellow light for side street in reset state.

How to test

Provide a clock. Generate a reset signal to reset the intersection to all-red.

IO

#	Input	Output
0	clock	main street red
1	reset	main street yellow
2	none	main street green
3	none	main street pedestrian green
4	none	side street red
5	none	side street yellow
6	none	side street green
7	none	side street pedestrian green

8x8 SRAM

- Author James Ross
- Description Write to and Read from 8 addressable 8-bit words of memory
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

Write Enable (WE) pin high while passing 4-bits low data, 4-bits high data into an 8-bit temporary shift register. Then Commit high while passing a 3-bit address places the register value into memory. While Output Enable (OE) high, a 3-bit address places the data from memory into the temporary register returns 8-bit register to output data interface. The highest 4th bit of the address is ignored. Fast memset, such as zeroing memory, can be performed with Commit high while passing a new address per clock cycle.

How to test

After reset, you can write values into memory and read

IO

#	Input	Output
0	input[0]	clk
1	input[1]	we
2	input[2]	oe
3	input[3]	commit
4	input[4]	addr[0]/high[0]/low[0]
5	input[5]	addr[1]/high[1]/low[1]
6	input[6]	addr[2]/high[2]/low[2]
7	input[7]	xxxxxxx/high[3]/low[3]

TinySensor

- Author Justin Pelan
- Description Using external hardware photodiodes as inputs, display light intensity on the 7-segment display
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware Breadboard, resistors, photodiodes, specific part# TBD

How it works

inputs 1 - 7 will be connected to external photodiodes to read either a '0' or '1', inputs will be added together and displayed on the 7-segment display

How to test

the dip switches can be used in place of external hw, simply throw the switches and the total number should show up on the 7-segment display

IO

#	Input	Output
0	clock	segment a
1	reset	segment b
2	none	segment c
3	none	segment d
4	none	segment e
5	none	segment f
6	none	segment g
7	none	none

binary clock

- Author Azdle
- Description A binary clock using multiplexed LEDs
- GitHub project
- Wokwi project
- Extra docs
- Clock 100 Hz
- External hardware

How it works

Using the internal clock, minutes and hours are counted in registers with an overflow comparision. An overflow in one, triggers a rising edge on the clock input of the successive register. The values of each register are connected to the input to a multiplexer, which is able to control 16 LEDs using just the 8 outputs.

How to test

after reset, the output shows the current time in BCD H:M when applied to a multiplexed array of LEDs

IO

#	Input	Output
0	clock	col 0
1	reset	col 1
2	none	col 2
3	none	col 3
4	none	row 0
5	none	row 2
6	none	row 3
7	none	row 4

The McCoy 8-bit Microprocessor

- Author Aidan Good
- Description Custom RISC-V inspired microprocessor capable of simple arithmatic, branching, and jumps through a custom ISA.
- GitHub project
- Wokwi project
- Extra docs
- Clock None Hz
- External hardware TODO

How it works

uses a register and some combinational logic

How to test

TODO

IO

#	Input	Output
0	clk	out7
1	reset	out6
2	in5	out5
3	in4	out4
4	in3	out3
5	in2	out2
6	in1	out1
7	in0	out0

CPU

- Author Ryan C
- Description 8bit CPU
- GitHub project
- Wokwi project
- Extra docs
- Clock 100 Hz
- External hardware todo

How it works

todo

How to test

todo

IO

#	Input	Output
0	clock	segment a
1	reset	segment b
2	none	segment c
3	none	segment d
4	none	segment e
5	none	segment f
6	none	segment g
7	none	none

s4t2-fpga

- Author Jan Gray
- Description super slow serial SRAM FPGA
- GitHub project
- Wokwi project
- Extra docs
- Clock 100000 Hz
- External hardware 23LC1024 serial SRAM

How it works

receives nybble stream of LUTs' config data, serially evaluates LUTs

How to test

pending

IO

#	Input	Output
0	clk	luts[0]
1	rst	luts[1]
2	si[0]	luts[2]
3	si[1]	luts[3]
4	si[2]	luts[4]
5	si[3]	luts[5]
6	none	luts[6]
7	none	luts[7]

Duty Controler

- Author Marcelo Pouso / Miguel Correia
- Description Increase/Decrease a duty cycle of square signal
- GitHub project
- Wokwi project
- Extra docs
- Clock 12500 Hz
- External hardware You are gonna need a 12.5Khz clock signal generator, and 2 bottoms for incremental and decremental inputs. An oscilloscope to see the output PWM 1.2KHZ signal.

How it works

Enter a square clock of 12.5Khz, and change Its duty cycle by pressing increase or decrease bottom. the change will be in steps of 10%. We have in mind the problem of bouncing bottom, and this was implemented for incremental and decremental bottom.

How to test

You need connect a signal clock of 12.5Khz in the io_in[0] port, a reset active high in io_in[1], a incremental signal in io_in[2], and the decremental signal in io_in[3]. The output signal will be in the io_out[0] port and the negate output in io_out[1]. The signal output will have a frecuency of 10Khz. When you press the incremental input bottom then the signal will increment by 10% Its duty cycle, and when you press the decremental input bottom you will see that the output signal decrement by 10%.

IO

#	Input	Output
0	{'clock': 'Clock input of 12.5Khz.'}	{'pwm': 'PWM output signal of input clk/10 = 1.2Khz.'}
1	{'reset': 'Reset active high.'}	{'pwm_neg': 'PWM negate output signal.'}
2	{'increase': 'This will increase the final duty cycle of the output signal by 10%.'}	{'increase': 'Syncronized signal from increase input.'}
3	{'decrease': 'This will decrease the final duty cycle of the output signal by 10%.'}	{'decrease': 'Syncronized signal from decrease input.'}
4	{'disable_debouncer': 'Disable debouncer for the increase / decrease inputs.'}	none

#	Input	Output
5	none	none
6	none	none
7	none	none

Power supply sequencer

- Author Jon Klein
- Description Sequentially enable and disable channels with configurable delay
- GitHub project
- Wokwi project
- Extra docs
- Clock 12500 Hz
- External hardware None, but could be useful for GaAs amplifiers or other circuits which need sequenced power supplies.

How it works

Counters and registers control and track the state of channel activations. The delay input sets the counter threshold.

How to test

After reset, bring enable high to enable channels sequentially, starting with channel 0. Bring enable low to switch off channels sequentially, starting with channel 7.

IO

#	Input	Output
0	clock	channel 0
1	reset	channel 1
2	enable	channel 2
3	delay0	channel 3
4	delay1	channel 4
5	delay2	channel 5
6	delay3	channel 6
7	delay4	channel 7

Matrix display

- Author Chris
- Description Display stuff on matrix display
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware

How it works

Uses SK9822 display

How to test

Need SK9822 display

IO

#	Input	Output
0	clock	segment a
1	reset	segment b
2	none	segment c
3	none	segment d
4	none	segment e
5	none	segment f
6	none	segment g
7	none	none

HD74480 Clock & UART transmitter

- Author Tom Keddie
- Description Displays a clock on a attached HD74480 & transmits a uart string
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware HD74480 & Serial Receiver

How it works

See https://github.com/TomKeddie/tinytapeout-2022-2/blob/main/doc/README.md

How to test

See https://github.com/TomKeddie/tinytapeout-2022-2/blob/main/doc/README.md

IO

#	Input	Output
0	clock	lcd D4
1	reset	lcd D5
2	none	lcd D6
3	none	lcd D7
4	none	lcd EN
5	none	lcd RS
6	none	none
7	none	uart tx output

SIMON Cipher

- Author Fraser Price
- Description Simon32/64 Encryption
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware

How it works

Encrypts data by sending it through a feistel network for 32 rounds where it is combined with the round subkey and the last round. Data is entered into the core via shift registers.

How to test

Set shift high and shift data in lsb first, 4 bits at a time. Shift in 96 bits, 32 being data and 64 being the key, with the plaintext being shifted in first. Eg if the plaintext was 32'h65656877 and key was 64'h1918111009080100, then 96'h191811100908010065656877 would be shifted in. Once bits have been shifted in, bring shift low, wait 32 clock cycles then set it high again. The ciphertext will be shifted out lsb first.

IO

#	Input	Output
0	clock	data_out[0]
1	shift	data_out[1]
2	data_in[0]	data_out[2]
3	data_in[1]	data_out[3]
4	data_in[2]	segment e
5	data_in[3]	segment f
6	none	segment g
7	none	none