M segments

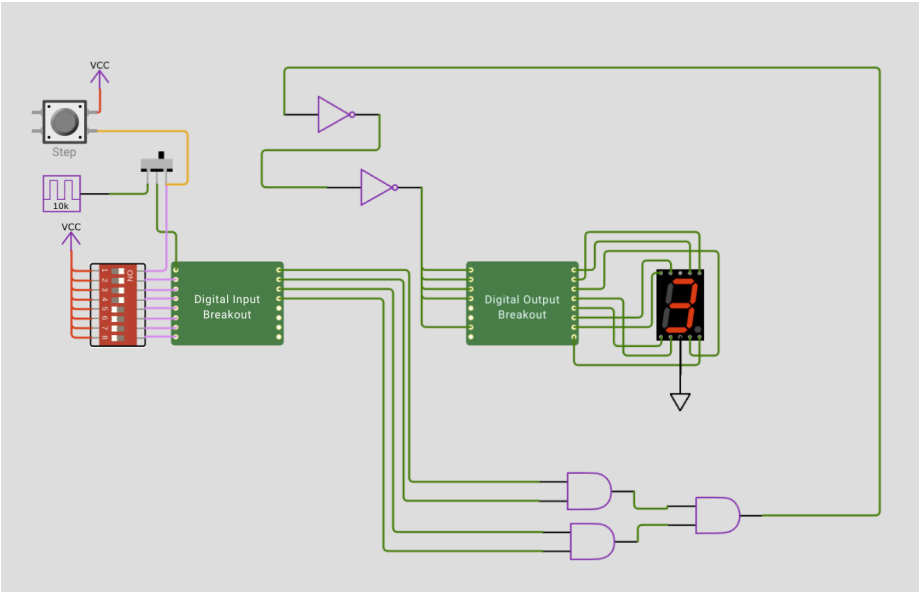

Figure 1: picture

- Author Matt Venn
- Description Setting the correct input will show a 3 on the display
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

AND gates connect to the 7 segment display

How to test

Turning on the first 4 inputs will show a 3 on the display

IO

#	Input	Output
0	input 1	segment a
1	input 2	segment b

#	Input	Output
2	input 3	segment c
3	input 4	segment d
4	none	segment e
5	none	segment f
6	none	segment g
7	none	dot

1-bit ALU

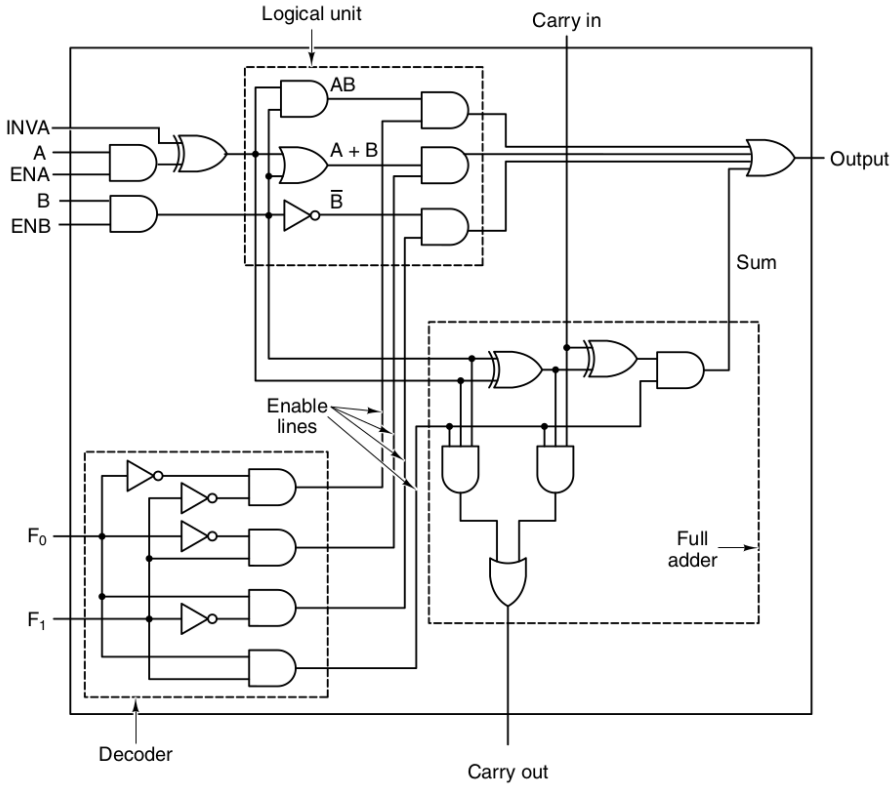

Figure 2: picture

- Author Leo Moser
- Description 1-bit ALU from the book `Structured Computer Organization: Andrew S. Tanenbaum`
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

The 1-bit ALU implements 4 different operations: AND, NOT, OR, ADD. The current operating mode can be selected via F0 and F1. F0=0 and F1=0 results in A AND B. F0=1 and F1=0 results in NOT B. F0=0 and F1=1 results in A OR B. F0=1 and F1=1 results in A ADD B. Where A and B are the inputs

for the operation. Additional inputs can change the way of operation: `ENA` and `ENB` enable/disable the respective input. `INVA` inverts A before applying the operation. `CIN` is used as input for the full adder. Multiple 1bit ALUs could be chained to create a wider ALU.

How to test

Set the operating mode via the DIP switches with F0 and F1. Next, set the input with A and B and enable both signals with ENA=1 and ENB=1. If you choose to invert A, set INVA to 1, otherwise to 0. For F0=1 and F1=1 you can set CIN as additional input for the ADD operation. The 7-segment display shows either a `0` or a `1` depending on the output. If the ADD operation is selected, the dot of the 7-segment display represents the COUT.

IO

#	Input	Output
0	CIN	segment a
1	INVA	segment b
2	A	segment c
3	ENA	segment d
4	B	segment e
5	ENB	segment f
6	F0	segment g
7	F1	COUT

Barrelshifter

- Author Johannes Hoff
- Description Shifts a 6 bit number up to 0-3 bits left
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

An 6 bit input value and 2 bit shift amount is provided, and the shifted value will be in the output

How to test

Choose an input value (like 6'b001010) and a shift amount (like 2'b10) and combine it into input pins (like 8'b00101010) and observe that the output is the shifted input (like 8'b10100000)

IO

#	Input	Output
0	bit 5 (most significant) of input value	bit 7 (most significant) of shifted value
1	bit 4 of input value	bit 6 of shifted value
2	bit 3 of input value	bit 5 of shifted value
3	bit 2 of input value	bit 4 of shifted value
4	bit 1 of input value	bit 3 of shifted value
5	bit 0 (least significant) of input value	bit 2 of shifted value
6	bit 5 (most significant) of shift amount	bit 1 of shifted value
7	bit 0 (least significant) of shift amount	bit 0 (least significant) of shifted value

PDM driver

- Author Harry Snell
- Description 5-bit pulse density modulation encoder (aka sigma-delta converter)
- GitHub project
- Wokwi project
- Extra docs
- Clock 200 Hz
- External hardware Clock source, switches for input. LED, RC circuit, logic analyser or oscilloscope to view output

How it works

The pdm_input is registered when write_en is high. The registered input is added to an accummulator on each cycle. The overflow bit of the sum is pdm_output.

How to test

Set reset low and write_en high, provide a clock (frequency not important) and put a 5-bit number on pdm_input and see how the average value on pdm_out changes

IO

#	Input	Output
0	clock	pdm_out
1	reset	pdm_out_n
2	write_en	none
3	pdm_input[0]	none
4	pdm_input[1]	none
5	pdm_input[2]	none
6	pdm_input[3]	none
7	pdm_input[4]	none

2x 1 to 4 Frequency Divider

- Author Seth Kerr
- Description A simple flip-flop based frequency divider
- GitHub project
- Wokwi project
- Extra docs
- Clock 16Hz Hz
- External hardware external clock sources are necessary to test all inputs. A simple adjustable astable 555 is recommended

How it works

The Frequency Divider works by taking 4 frequencies and selecting 2 with select pins as inputs and through a flip-flop chain, divides the input into 4 frequencies. The frequency breaks down to f/2, f/4, f/8, and f/16.

How to test

The most simple test of the frequency divider is to use the source clock on pin one, and attach an oscilloscope to the outputs and measure their frequency.

IO

#	Input	Output
0	clock (default f(1) input option)	f(1)/2 - Frequency 1 divided in half
1	f(1), 2 - second f(1) input option	f(1)/4 - Frequency 1 divided in quarters
2	f(2), 1 - default f(2) input option	f(1)/8 - Frequency 1 divided in eighths
3	f(2), 2 - second f(2) input option	f(1)/16 - Frequency 1 divided into sixteenths
4	f(1) select - selects which f(1) input frequency to use	f(2)/2 - Frequency 2 divided in half
5	f(2) select - selects which f(2) input frequency to use	f(2)/4 - Frequency 2 divided in quarters
6	none	f(2)/8 - Frequency 2 divided in eighths
7	none	f(2)/16 - Frequency 2 divided into sixteenths

BCD to Decimal Decoder

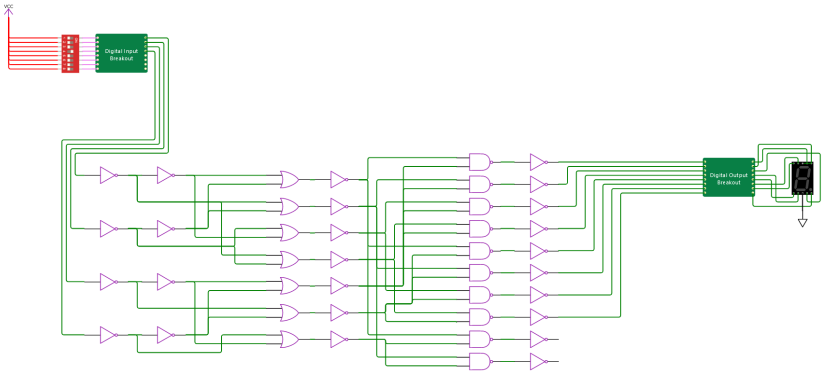

Figure 3: picture

- Author JinGen Lim
- Description Converts a BCD input into a decimal output
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

Accepts BCD through 4 input pins, and outputs the decimal equivalent (output pins 0 to 7). This is a functional clone of the DM7447, but does not include values 8 and 9 due to package constraints.

How to test

Write to the BCD input (IN0:0, IN1:2, IN2:4, IN3:8). One corresponding decimal output pin will be driven high.

IO

#	Input	Output
0	input 1 (BCD 1)	decimal output 0

#	Input	Output
1	input 2 (BCD 2)	decimal output 1
2	input 3 (BCD 4)	decimal output 2
3	input 4 (BCD 8)	decimal output 3
4	none	decimal output 4
5	none	decimal output 5
6	none	decimal output 6
7	none	decimal output 7

BCD to 7-Segment Decoder

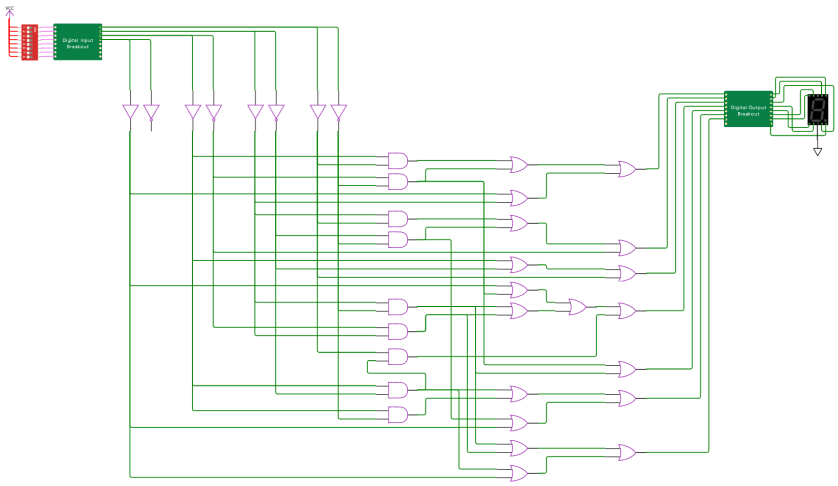

Figure 4: picture

- Author JinGen Lim
- Description Converts a BCD input into a 7-segment display output
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

The IC accepts four binary-coded decimal input signals, and generates a corresponding 7-segment output signal

How to test

Connect the segment outputs to a 7-segment display. Configure the input (IN0:0, IN1:2, IN2:4, IN3:8). The input value will be shown on the 7-segment display

IO

#	Input	Output
0	input 1 (BCD 1)	segment a
1	input 2 (BCD 2)	segment b

#	Input	Output
2	input 3 (BCD 4)	segment c
3	input 4 (BCD 8)	segment d
4	none	segment e
5	none	segment f
6	none	segment g
7	none	none

Barrel Shifter

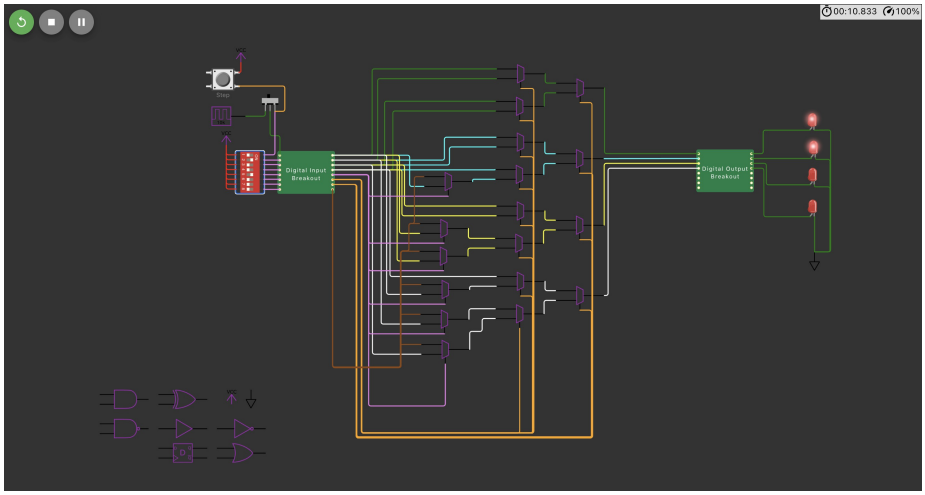

Figure 5: picture

- Author Shahzaib Kashif
- Description shifts the data n times, where n is the input provided via inputs
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

muxes connect to output

How to test

give input through input ports and toggle select pins to visualise shifting

IO

#	Input	Output
0	input 1	output bit 1
1	input 2	output bit 2
2	input 3	output bit 3
3	input 4	output bit 4
4	shift type	none

#	Input	Output
5	select 1	none
6	select 2	none
7	hardcode 0	none

Pseudo-random number generator

- Author Thomas Böhm thomas.bohm@gmail.com
- Description Pseudo-random number generator using a 16-bit Fibonacci linear-feedback shift register
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

16 flip flops are connected in a chain, and the output of some is XORed together and fed back into the first flip flop. The outputs that are XORed together are chosen in such a way as to give the longest possible cycle (2^16-1). All bits being zero is a special case and is treated separately (all negative outputs of the flip flops are ANDed together to generate a 1 as feedback). On each clock pulse (pin 1) one new bit is generated. Setting load_en (pin 3) to HIGH allows the loading of a user defined value through the data_in pin (pin2). On each clock pulse one bit is read into the flip flop chain. When load_en (pin 3) is set to LOW the computed feedback bit is fed back into the flip flops. The outputs of the last 8 flip flops are connected to the output pins. For each clock pulse a random bit is generated and the other 7 are shifted.

How to test

Set the switch for pin 1 so that the push button generates the clock. Press on it and see the output change on the hex display. Using pin 2 and 3 a custom value can be loaded into the flip flops.

IO

#	Input	Output
0	clock	random bit 0
1	data_in	random bit 1
2	load_en	random bit 2
3	none	random bit 3
4	none	random bit 4
5	none	random bit 5
6	none	random bit 6
7	none	random bit 7

BCD to 7 Segment Decoder

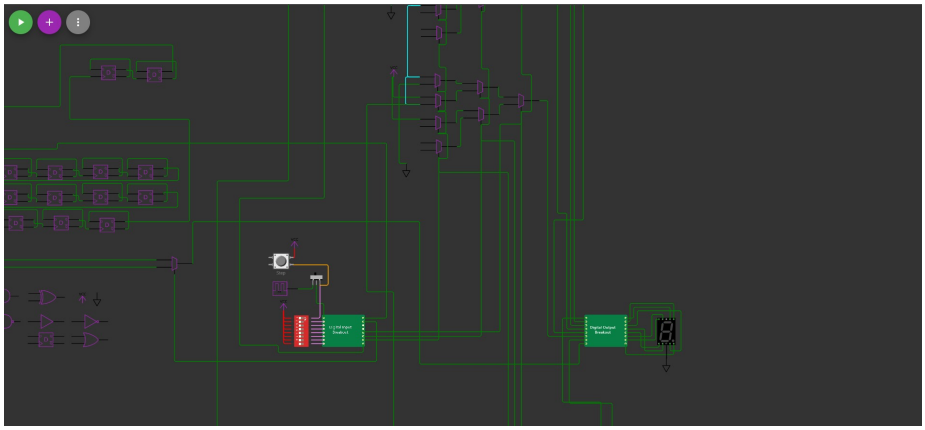

Figure 6: picture

- Author Pramit Pal
- Description Setting the DIP switches as a BCD Value displays the BCD Value 0-9 in the 7 segment display
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

Display BCD Value given by user with the 7 segment display

How to test

Turning on the last 4 input switches to display BCD Value

IO

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

#	Input	Output
7	input 0	dot

Fibonacci & Gold Code

- Author Daniel Estevez
- Description This project includes two independent designs: a design that calculates terms of the Fibonacci sequence and displays them in hex one character at a time on a 7-segment display, and a Gold code generator that generates the codes used by CCSDS X-band PN Delta-DOR.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware No external hardware is needed

How it works

The Fibonacci calculator uses 56-bit integers, so the terms of the Fibonacci sequence are displayed using 7 hex characters. Since the TinyTapeout PCB only has one 7-segment display, the terms of the Fibonacci sequence are displayed one hex character at a time, in LSB order. The dot of the 7-segment display lights up whenever the LSB is being displayed. On each clock cycle, 4-bits of the next Fibonacci term are calculated using a 4-bit adder, and 4-bits of the current term are displayed in the 7-segment display. The 7-segment display is ANDed with the project clock, so that the digits flash on the display. The Gold code generator computes a CCSDS X-band PN Delta-DOR Gold code one bit at a time using LFSRs. The output bit is shown on the 7-segment display dot. 6-bits of the second LFSR can be loaded in parallel using 6 project inputs in order to be able to generate different sequences. One of the project inputs is used to select whether the 7-segment display dot is driven by the Fibonacci calculator or by the Gold code generator.

How to test

The project can be tested by manually driving the clock using a push button or switch. Just by de-asserting the reset and driving the clock, the digits of the Fibonacci sequence terms should appear on the 7-segment display. The output select input needs to be set to Gold code (high level) in order to test the gold code generator. The load enable input (active-low), as well as the 6 inputs corresponding to the load for the B register can be used to select the sequence to generate. The load value can be set in the 6 load inputs, and then the load enable should be pulsed to perform the load. This can be done with the clock running or stopped, as the load enable is asynchronous. After the load enable is de-asserted, for each clock cycle a new bit of the Gold code sequence should appear in the 7-segment display dot.

IO

#	Input	Output
0	clock	{‘segment a’: ‘Fibonacci hex digit’}
1	output select (high selects Gold code; low selects Fibonacci LSB marker) & Gold code load value bit 0	{‘segment b’: ‘Fibonacci hex digit’}
2	Fibonacci reset (active-low; asynchronous) & Gold code load value bit 1	{‘segment c’: ‘Fibonacci hex digit’}
3	Gold code load enable (active-low; asynchronous)	{‘segment d’: ‘Fibonacci hex digit’}
4	Gold code load value bit 2	{‘segment e’: ‘Fibonacci hex digit’}
5	Gold code load value bit 3	{‘segment f’: ‘Fibonacci hex digit’}
6	Gold code load value bit 4	{‘segment g’: ‘Fibonacci hex digit’}
7	Gold code load value bit 5	{‘none’: ‘Gold code output / Fibonacci LSB digit marker’}

GPS C/A PRN Generator

- Author Adam Greig
- Description Generate any of the GPS C/A PRN sequences from PRN0 to PRN31
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware None

How it works

Instantiates the GPS G1 and G2 LFSRs to generate a pseudo-random sequence, then selects the G2 output taps based on the input signals to output the chosen sequence.

How to test

Apply clock to the in[0], pulse reset on in[1], choose a PRN between 0 and 31 using in[2:7], then the G1 sequence is emitted on out[0], the G2 sequence on out[1], and the selected PRN on out[2]. The first 20 bits of PRN2 are 11100100001110000011.

IO

#	Input	Output
0	clock	G1 subsequence
1	reset	G2 subsequence
2	prn[0]	Selected PRN
3	prn[1]	None
4	prn[2]	None
5	prn[3]	None
6	prn[4]	None
7	none	None

PDP-0: 4-bit CPU in the style of PDP-1/TX-0

- Author Tommy Thorn
- Description The tiny 4-bit CPU packs a 3b program counter, an accumulator, and 8 6b words.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware Besides interactining with the IOs, nothing is needed

How it works

The two top bits in each word form the opcode (load, store, add, branch-if-zero) while the remaining four are the immediate field that the opcode uses. Load and store only access the immediate field of the word. The IO implements a simple command protocol to reset, load data, load code, and run. The output are used for the PC and the Accumulator. The test bench shows how to load a fibonacci computing program.

How to test

Use the command protocol to load programs and run them (see test bench)

IO

#	Input	Output
0	clock	acc[0]
1	cmd[0]	acc[1]
2	cmd[1]	acc[2]
3	not used	acc[3]
4	cmdarg[0]	pc[0]
5	cmdarg[1]	pc[1]
6	cmdarg[2]	pc[2]
7	cmdarg[3]	Not used, wired to 0

Game of Life - Cell Neighbor Count

- Author Uri Shaked (Wokwi)
- Description Logic to decide about the fate of a cell in the game of life: die, stay alive, or spring to life
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

The 8 inputs represent the current state of the neighboring cells (0 = dead, 1 = alive). The fate of the cell is in bits 2 and 3 of the output. The cell will be alive if either bit 2 is on and it was alive in the previous generation, or if bit 3 is on. Otherwise, it'll die.

How to test

Connect 8 DIP switches to the 8 input pins, and LEDs to bits 2 and 3 of the output. Observe the value of the output bits: bit 2 should be on when either two or three of the DIP switches are on, and bit 3 should be on when exactly three DIP switches are on.

IO

#	Input	Output
0	in0	none
1	in1	none
2	in2	2 or 3 inputs are high
3	in3	exactly 3 inputs are high
4	in4	none
5	in5	none
6	in6	none
7	in7	none

Traffic Light FSM

- Author Christian Fibich
- Description FSM controlling two (red-yellow-green) traffic lights
- GitHub project
- Wokwi project
- Extra docs
- Clock 2 Hz
- External hardware 2 red LEDs, 2 yellow LEDs, 2 green LEDs and current limiting resistors

How it works

State machine that implements a typical Austrian traffic light: Red -> Red+Yellow -> Green -> Green Blinking -> Yellow -> Red. Generated using a hacked-together Verilog->Wokwi flow :D.

How to test

Starts in 'error' mode (yellow blinking). Switch SW1 (reset) to 1 and back to 0 to start operation. 'error' mode can be reached by toggling SW2.

IO

#	Input	Output
0	clock	red 1
1	reset	yellow 1
2	enter_error_mode	green 1
3	none	red 2
4	none	yellow 2
5	none	green 2
6	none	none
7	none	none

7 Segment Figure Eight

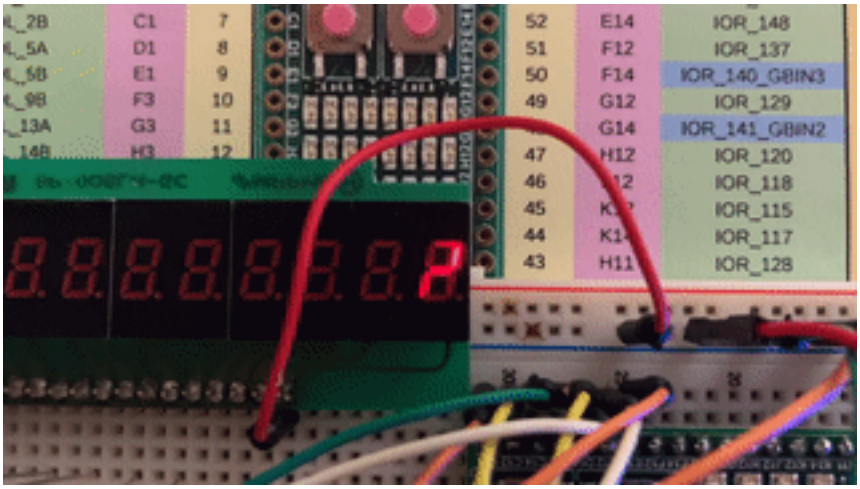

Figure 7: picture

- Author Christian Lyder Jacobsen
- Description
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

How to test

IO

#	Input	Output
0	clock	segment a
1	reset	segment b
2	speed lsb	segment c
3	speed	segment d
4	speed msb	segment e
5	tail	segment f
6	direction	segment g
7	led invert	none

Logic-2G57-2G58

- Author Sirawit Lappisatepun
- Description Replication of TI's Little Logic 1G57 and 1G58 configurable logic gates.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

This design replicates the circuit inside a TI configurable logic gates 74xx1G57 (and by including an inverted output, it will work as a 74xx1G58 as well). Since there are still I/O pins left, I included two of these configurables, and also one 74xx1G79 D Flip-Flop (again, an inverted output means this will also work as a 74xx1G80).

How to test

You could refer to TI's 1G79/1G80/1G57/1G58 datasheet to test the device according to the pinout listed below.

IO

#	Input	Output
0	dff_clock	dff_out
1	dff_data	dff_out_bar
2	gate1_in0	gate1_out
3	gate1_in1	gate1_out_bar
4	gate1_in2	gate2_out
5	gate2_in0	gate2_out_bar
6	gate2_in1	none
7	gate2_in2	none

Logic-2G97-2G98

- Author Sirawit Lappisatepun
- Description Replication of TI's Little Logic 1G97 and 1G98 configurable logic gates.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware

How it works

This design replicates the circuit inside a TI configurable logic gates 74xx1G97 (and by including an inverted output, it will work as a 74xx1G98 as well). Since there are still I/O pins left, I included two of these configurables, and also one 74xx1G79 D Flip-Flop (again, an inverted output means this will also work as a 74xx1G80).

How to test

You could refer to TI's 1G79/1G80/1G97/1G98 datasheet to test the device according to the pinout listed below.

IO

#	Input	Output
0	dff_clock	dff_out
1	dff_data	dff_out_bar
2	gate1_in0	gate1_out
3	gate1_in1	gate1_out_bar
4	gate1_in2	gate2_out
5	gate2_in0	gate2_out_bar
6	gate2_in1	none
7	gate2_in2	none

RGB LED Matrix Driver

- Author Matt M
- Description Drives a simple animation on SparkFun's RGB LED 8x8 matrix backpack
- GitHub project
- Wokwi project
- Extra docs
- Clock 12500 Hz
- External hardware RGB LED matrix backpack from SparkFun: https://www.sparkfun.com/products/retired/760

How it works

Implements an SPI master to drive an animation with overlapping green/blue waves and a moving white diagonal. Some 7-segment wires are used for a 'sanity check' animation.

How to test

Wire accordingly and use a clock up to 12.5 KHz. Asynchronous reset is synchronized to the clock.

IO

#	Input	Output
0	clock	SCLK
1	reset	MOSI
2	none	segment c
3	none	segment d
4	none	segment e
5	none	nCS
6	none	segment g
7	none	none (always high)

Digital padlock

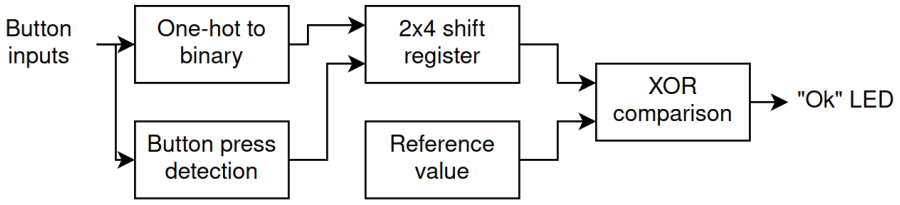

Figure 8: picture

- Author Jean THOMAS
- Description A 4-digit electronic padlock
- GitHub project
- Wokwi project
- Extra docs
- Clock 100 Hz
- External hardware

How it works

Each buttons press is detected by a rising edge detector, and each button press is decoded into a binary code. That binary code is stored in a shift-register which is continuously checked against a reference value ('the padlock code').

How to test

Connect a clock generator to the clock input, connect all four buttons with a debounce circuit - the buttons should act as active high.

IO

#	Input	Output
0	clock	none
1	Button A	none
2	Button B	none
3	Button C	none
4	Button D	none
5	none	none
6	none	Button press detected
7	none	Code valid

TinyIO

- Author Aidan Medcalf
- Description Tiny I/O expander with SPI interface
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware SPI driver

How it works

TinyIO takes 7 bits in as digital outputs, and sends 3 bits out from its digital inputs.

How to test

7-bit SPI transaction. Supply nCE, SIN and SCK.

IO

#	Input	Output
0	clock	out0
1	reset	out1
2	serial clock	out2
3	chip enable	out3
4	serial in	out4
5	in0	out5
6	in1	out6
7	in2	serial out

4-bit-asychronous multiplier

- Author Hirosh Dabui
- Description Asychronous 4-bit multiplier that return 8-bit
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware 2x4-switches and 8 leds

How it works

2x4bit operands and 8bit result

How to test

Feed the 4-bit-operands and get the multiplication

IO

#	Input	Output
0	input a0	output c[0]
1	input a1	output c[1]
2	input a2	output c[2]
3	input a0	output c[3]
4	input b1	output c[4]
5	input b2	output c[5]
6	input b3	output c[6]
7	input b3	output c[7]

Shiftregister 8 Bit

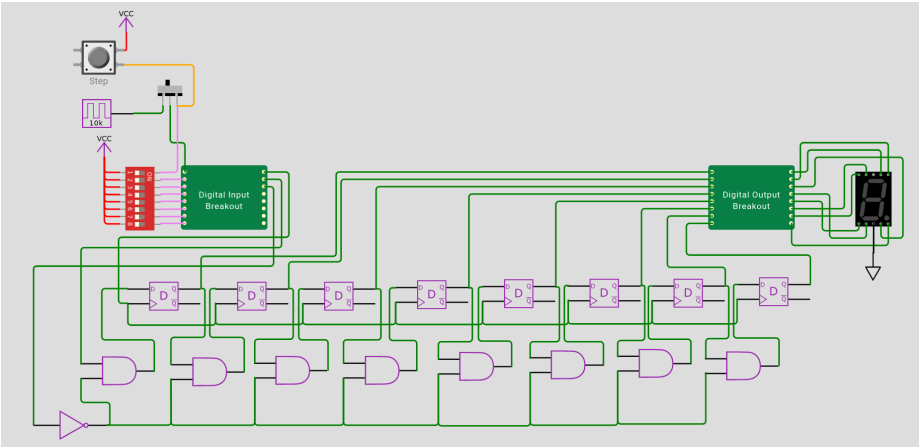

Figure 9: picture

- Author Thorsten Knoll
- Description A simple shiftregister with 8 bit depth, made from D-FlipFlops. Programmable with data and clk, reset. All 8 bits will be mapped to the output.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware A dipswitch and two button for the inputs. 8 LEDs connected to the outputs.

How it works

Programm the shiftregister with the data (IN0) and clk (IN1) inputs. With reset enabled, the FlipFlops will be cleared with the next risign edge on the clk. The outputs (OUT0 - OUT7) are driven by the shiftregister bits.

How to test

Each rising edge at the clk input pushs a new data bit into the reǵister. Reset happens with the next clk. See the complete state of the register at the 8 outputs.

IO

#	Input	Output
0	clk	out 0
1	data	out 1
2	reset	out 2
3	none	out 3
4	none	out 4
5	none	out 5
6	none	out 6
7	none	out 7

Shiftregister Challenge 40 Bit

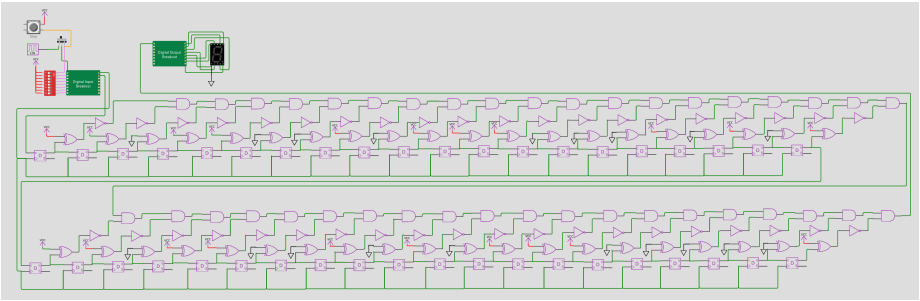

Figure 10: picture

- Author Thorsten Knoll
- Description The design is a 40 bit shiftregister with a hardcoded 40 bit number. The challenge is to find the correct 40 bit to enable the output to high. With all other numbers the output will be low.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware To test when knowing the correct 40 bit, only a dip-switch (data), a button (clk) and a LED (output) is needed. Without knowing the number it becomes the challenge and more hardware might be required.

How it works

Shift a 40 bit number into the chip with the two inputs data (IN0) and clk (IN1). If the shifted 40 bit match the hardcoded internal 40 bit, then and only then the output will become high. Having only the mikrochip without the design files, one might need reverse engineering and/or side channel attacks to fing the correct 40 bit.

How to test

Get the correct 40 bit from the design and shift them into the shiftregister. Each rising edge at the clk will push the next bit into the register. At the correct 40 bit, the output will enable high.

IO

#	Input	Output
0	data	output
1	clk	none
2	none	none
3	none	none
4	none	none
5	none	none
6	none	none
7	none	none

Figure 8 pattern generator

- Author todd1251
- Description Generates a figure 8 pattern on the 7-segment display
- GitHub project
- Wokwi project
- Extra docs
- Clock 1 Hz
- External hardware

How it works

How to test

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

Picture Printer

Figure 11: picture

- Author Miron Zadora
- Description Outputs the Edinburgh Hacklab logo pixel by pixel
- GitHub project
- Wokwi project
- Extra docs
- Clock 1000 Hz
- External hardware Something to decode and display the image. E.g. An arduino connected to the chip Clock and Pixel Output pins could be used to display the 1s and 0s coming from the chip as '@' and '.' characters in a serial console, putting a newline every 41 characters (41 is the width of the image)

How it works

It outputs the image pixel by pixel, line by line, left to right, top to bottom. 1 pixel per clock cycle. Image is 41 by 41 pixels. 1 = black pixel, 0 = white pixel.

How to test

Supply a clock to 1st input and hold 2nd input high for one clock cycle to reset. Everything in the design happens on the rising edge of the clock. Connect external hardware described below.

IO

#	Input	Output
0	Clock	Pixel output
1	Synchronous reset	none
2	none	none
3	none	none
4	none	none
5	none	none
6	none	none
7	none	none

4-bit ALU with 7 segment display decoder hexadecimal output

- Author Michael Gargano
- Description a 4-bit ALU with 8 different possible operations on an internal accumulator whose current 4-bit state is displayed in hexadecimal on the segmented display
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

3-bit decoder picks operation pass-through, bit-wise (NOT, AND, OR, XOR), two's complement, add, or subtract circuit based on selection and stores the result in a 4-bit accumulator, after the pressing the step button, the 4-bit to 7-segment display decoder circuits then take that value and display it hexadecimal

How to test

select one of the 8 alu operations [switches 4-2] (pass-through 0, bit-wise NOT 1, bit-wise AND 2, bit-wise OR 3, bit-wise XOR 4, add 5, two's compliment 6, subtract 7) then input a 4-bit number [switches 8-5], press step button to perform the selected computation and the display will indicate current 4-bit accumulator value in hex, dot indicates if a carry is output during addition or subtraction

IO

#	Input	Output
0	clock (or single step with step button)	segment a of hex output
1	alu operation selection bit 2	segment b of hex output
2	alu operation selection bit 1	segment c of hex output
3	alu operation selection bit 0	segment d of hex output
4	input bit 3	segment e of hex output
5	input bit 2	segment f of hex output
6	input bit 1	segment g of hex output
7	input bit 0	dot (indicates carry)

An optionally cumulative adder

- Author Michael Christen
- Description Increment with clock and add previous result or current A + B
- GitHub project
- Wokwi project
- Extra docs
- Clock 1 Hz
- External hardware LEDs and switches would be handy

How it works

A, B are 3 bits, there's a clock and a selctor to use A or (A + B)'

How to test

Toggle clock to run increment, swap between acumulating or just adding with switch 2; 3-5 are A, 6-8 are B

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

7-segment LED flasher

- Author Joseph Chiu
- Description Drives 7-segment LED display, alternating between NIC and JAC
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware Signals are assigned per the tinytapeout wokwi simulator and intended to run on the project PCB

How it works

Master clock is fed through a prescaler with four tap-points which feeds a 4-bit ripple counter (there are 6 total bits, but the top two bits are discarded). 2:1 muxes are chained to act like a 8:1 mux for each LED segment position. As the counter runs, this results in each segment being turned on or off as needed to render the display sequence (NIC JAC). The highest order bit is used to blink the decimal point on/off.

How to test

IN5 and IN6 selects the clock prescaler. OUT0-OUT7 are the LED segment outputs.

IO

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

tinytapeout-HELLo-3orLd-7seg

Figure 12: picture

- Author Rakesh Peter
- Description HELLo-3orLd Runner on 7 segment Display
- GitHub project
- Wokwi project
- Extra docs
- Clock 1 Hz
- External hardware none

How it works

BCD Counter with 7 seg Decoder

How to test

All toggle switches in zero position and clock switch on for auto runner. Individual BCD bits can be toggled using corresponding inputs with clock switch off.

IO

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

Wolf sheep cabbage river crossing puzzle ASIC design ( )

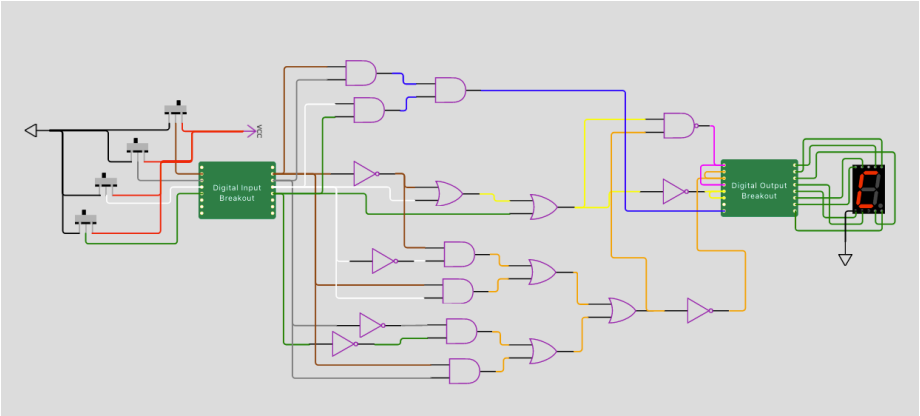

Figure 13: picture

- Author maehw
- Description Play the wolf, goat and cabbage puzzle interactively.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware Input switches and 7-segment display

How it works

Truth table with the game logic (hidden easter egg). The inputs are the positions of the farmer, wolf, goat and cabbage. The 7-segment display shows the status of the game (won or lost).

How to test

Slide the input switches, think, have a look at the 7-segment display.

IO

#	Input	Output
0	not connected because it is typically used for clocked designs and may be used in the future of this design	output signal ~E, i.e. the top and bottom segments light up, when the game is over due to an unattended situation on any river bank side
1	input signal F for the position of the farmer (/)	output signal ~R i.e. the top-right and bottom-right segments light up, to indicate an unattended situation on the right river bank (game over)
2	input signal W for the position of the wolf ( )	output signal ~R i.e. the top-right and bottom-right segments light up, to indicate an unattended situation on the right river bank (game over)
3	input signal G for the position of the goat ( )	output signal ~E, i.e. the top and bottom segments light up, when the game is over due to an unattended situation on any river bank side
4	input signal C for the position of the cabbage ( )	output signal ~L i.e. the top-left and bottom-left segments light up, to indicate an unattended situation on the left river bank (game over)
5	here be dragons or an easter egg	output signal ~L i.e. the top-left and bottom-left segments light up, to indicate an unattended situation on the left river bank (game over)
6	unused	here be dragons or an easter egg
7	unused	output signal A to light up the "dot LED" of the 7 segment display as an indicator that all objects have reached the right bank of the river and the game is won!

8x8 Bit Pattern Player

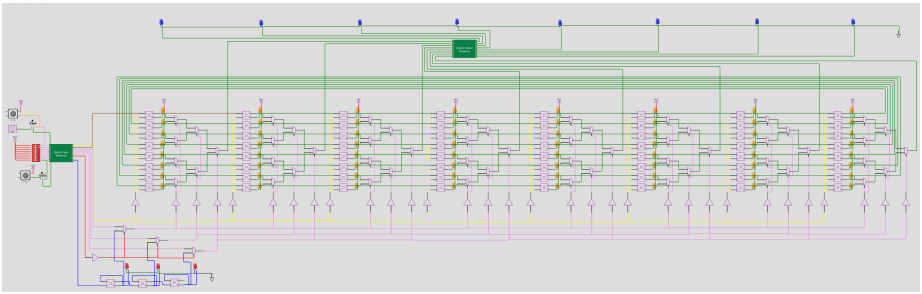

Figure 14: picture

- Author Thorsten Knoll
- Description 8x8 bit serial programmable, addressable and playable memory.
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware You could programm, address and play the 8x8 Bit Pattern Player with a breadboard, two clock buttons and some dipswitches on the input side. Add some LED to the output side. Just like the WOKWI simulation.

How it works

The 8x8 memory is a 64-bit shiftregister, consisting of 64 serial chained D-FlipFlops (data: IN0, clk_sr: IN1). 8 memoryslots of each 8 bit can be directly addressed via addresslines (3 bit: IN2, IN3, IN4) or from a clockdriven player (3 bit counter, clk_pl: IN7). A mode selector line (mode: IN5) sets the operation mode to addressing or to player. The 8 outputs are driven by the 8 bit of the addressed memoryslot.

How to test

Programm the memory: Start by filling the 64 bit shiftregister via data and clk_sr, each rising edge on clk_sr shifts a new data bit into the register. Select mode: Set mode input for direct addressing or clockdriven player. Address mode: Address a memoryslot via the three addresslines and watch the memoryslot at the outputs. Player mode: Each rising edge at clk_pl enables the next memoryslot to the outputs.

IO

#	Input	Output
0	data	bit 0
1	clk_sr	bit 1
2	address_0	bit 2
3	address_1	bit 3
4	address_2	bit 4
5	mode	bit 5
6	none	bit 6
7	clk_pl	bit 7

Figure of 8 orbit animation

- Author Rajarshi Roy
- Description Stepping using button will show a figure of 8 orbit animation on 7 segment display
- GitHub project
- Wokwi project
- Extra docs
- Clock 0 Hz
- External hardware None

How it works

Signal goes through ring shift register, each flop in shift register is connected to a segment in the display.

How to test

In button stepping mode, turn on switch 8 and press multiple times to fill figure of 8 with signal from switch 7, then toggle switch 7, step once, turn off switch 8, then keep stepping to see the orbit animation.

IO

#	Input	Output
0	clock	segment a
1	none	segment b
2	none	segment c
3	none	segment d
4	none	segment e
5	none	segment f
6	value of signal to enter into ring shift register when switch 8 is enabled	segment g
7	insert new signal from switch 7 into ring shift register	dot shows when switch 8 is enabled

Low-speed UART transmitter with limited character set loading

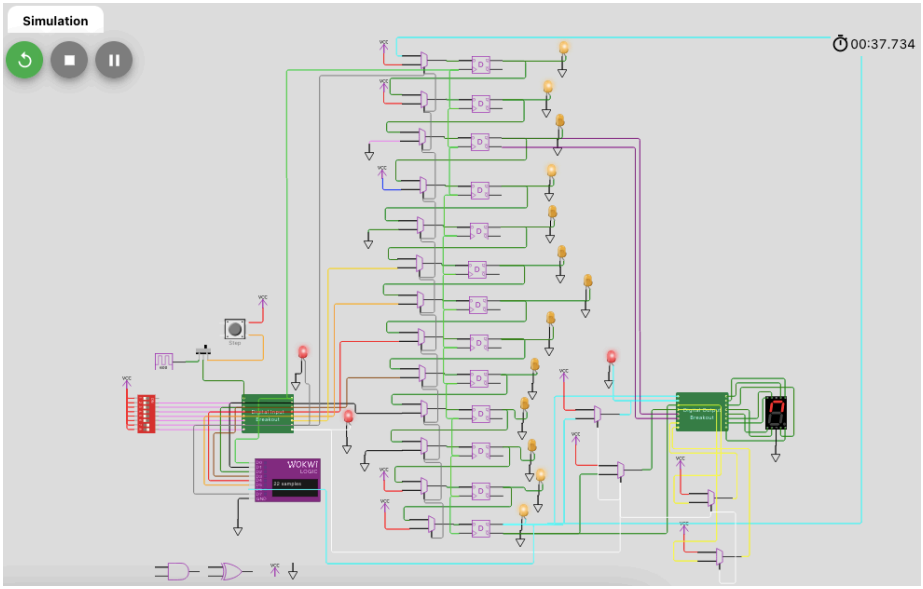

Figure 15: picture

- Author maehw
- Description 300(?) baud UART transmitter (8N1) with limited character set (0x40..0x5F; includes all capital letters in the ASCII table) loading.
- GitHub project
- Wokwi project
- Extra docs
- Clock 300 Hz
- External hardware UART receiver or oscilloscope or logic analyzer (optional)

How it works

The heart of the design is a 13 bit shift register (built from D flip-flops). When a word has been transmitted, it will be transmitted again and again until a new word is loaded into the shift register or the output is disabled (the word will keep on cycling internally).

How to test

Load a character into the design and attach a UART receiver (or oscilloscope or logic analyzer) on the output side.

IO

#	Input	Output
0	300 Hz input clock signal (or different value supported by the whole	UART (serial output pin, direct throughput)
1	bit b0 (the least significant bit) of the loaded and transmitted character	UART (serial output pin, gated by enable signal)
2	bit b1 of the loaded and transmitted character	UART (serial output pin, reverse polarity, direct throughput)
3	bit b2 of the loaded and transmitted character	UART (serial output pin, reverse polarity, gated by enable signal)
4	bit b3 of the loaded and transmitted character	UART (MSBit, direct throughput); typically set to 1 or can be used to sniffing the word cycling through the shift register)
5	bit b4 of the loaded and transmitted character	UART (MSBit, reverse polarity, direct throughput); same usage as above
6	load word into shift register from parallel input (IN1..IN5) (1) or cycle the existing word with start/stop bits around it (0)	UART (MSBit, gated by enable signal); typically set to 1 or can be used to sniffing the word cycling through the shift register)
7	{'output enable (for gated output signals)': '1 output is enabled, 0 output is disabled (permanently set to HIGH/1)'}	UART (MSBit, reverse polarity, gated by enable signal); same usage as above

LAB11

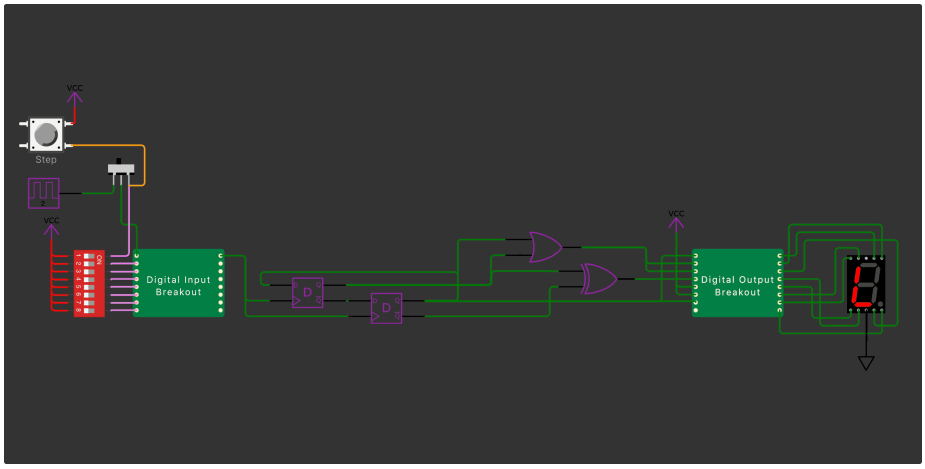

Figure 16: picture

- Author Thomas Zachariah
- Description Cycles through the characters of LAB11
- GitHub project
- Wokwi project
- Extra docs
- Clock 2 Hz
- External hardware None

How it works

Gates & flip-flops connected to the 7-segment display change the state of corresponding LED segments to form the next character, each cycle

How to test

Set to desired clock speed – characters are most readable at the lowest speed

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

#	Input	Output
6	none	segment g
7	none	none