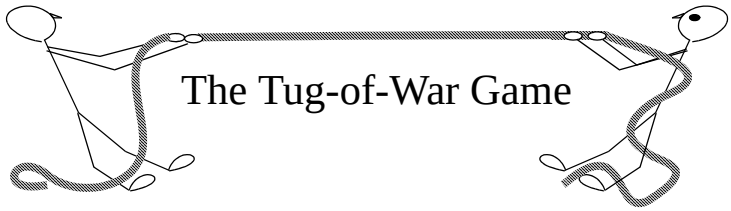
CARLETON UNIVERSITY

Deparment of Electronics

ELEC 3500 **Digital Electronics** **October 17, 2006**

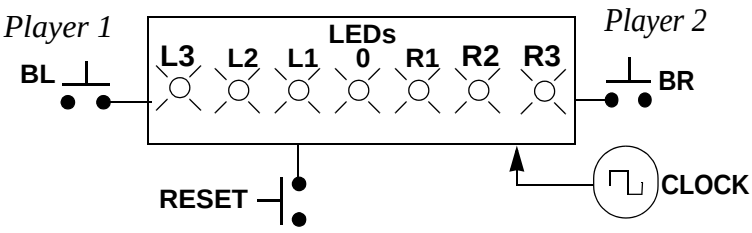


Rev 11.

1.0 The Game : Overview

The players of this game see a row of 7 LEDs represented by the numbers L3 through R3 in FIGURE 1

FIGURE 1.
The tug of war machine. It contains 3 push-buttons, 7 light-emitting diodes (LEDs) and 1 field-programmable gate array (FPGA)



After pushing reset, all the LEDs should come on for a second and then go off. This is the get ready signal. After a random time the middle LED comes on again. Then each player will try to push his button before the other player does. The position of the lit LED will move toward the fastest button pusher.

Thus if player 2 is fastest, LED 0 will go out and LED R1 will come on. This is the end of **round** one. After a second the LEDs will all go out. This is the get ready signal for round two. After a random time LED R1 will come on (assuming player 2 won the first round). Again each player will try to push his button first, and again the light will shift toward the fastest button pusher. The game is won when one player makes the position of the lit LED move off the end of the display.

If a player jumps the light, i.e. he pushes his button while the LEDs are off, then the light will come on immediately and it will be shifted one position away from him as a penelty of an illegal move.

The FAVOR-THE-LOSER circuit gives a slow starter a chance. When a player is about to win (the R3 or L3) position) the light will jump back two spaces (to positions R1 or L1) if he loses, but only one position if he jumps-the-light.

The circuit must be able to distinguish the winner within two gate delays (a few nanoseconds). Further the design must be fair. For example, the design should not assign ties to one of the players.

2.0 The Field Programmable Gate Array (FPGA)

These are a collection of several hundred flip-flops, and several thousand gates, all collected in one integrated circuit. Also in the integrated circuit are a large number of wires which run across different parts of the integrated circuit. There are also many electrically controlled switches which connect the flip-flops and gates to the wires, and the wires to each other. By opening and closing the switches, one can build almost any digital circuit unless the circuit needs more gates or flip-flops than there are in the FPGA.

Setting theses connection switches might be quite a job, Fortunately there is a computer program that does all the work. Just connect the FPGA to the serial port of a PC, run the program, and in about five seconds, your circuit is built. It sure beats stripping wires!

Your assignment is to design a Field-Programmable Gate Array (FPGA) to perform the tug-of-war game. All the logic will be on one IC. Only the push buttons, LEDs, LED drivers and the oscillator will be external.

3.0 Good Design Practice

The following practices shall be followed in your design:

a. Do not gate the clock. The gate will cause the flip-flops controlled by the gated clock to flip later than the ones directly connected. Use enabled flip-flops, and gate the enable instead of the clock.

b. Use only ONE clock for all timing. Divide its output to get slower timing signals. Then use a pulse fed into the clock-enable (CE) flip-flop input to give a lower effective clock rate.

c. Two related (passing through common logic) asynchronous input signals must be stable near the clock edge on which they are read.

d. Do not allow one asynchronous input to change two state variables (flip-flops) together. If the clock changes just as the input changes you get a race. Instead clock the input into a single D flip-flop. This makes the input into a synchronous signal which can safely initiate multiple-variable state changes.

e. Do not use asynchronous preset or clear in counters and shift registers. The clear signal may clear itself before clearing all the flip-flops. Asynchronous clear may, and should, be used to clear all flip-flops during power-on restart.

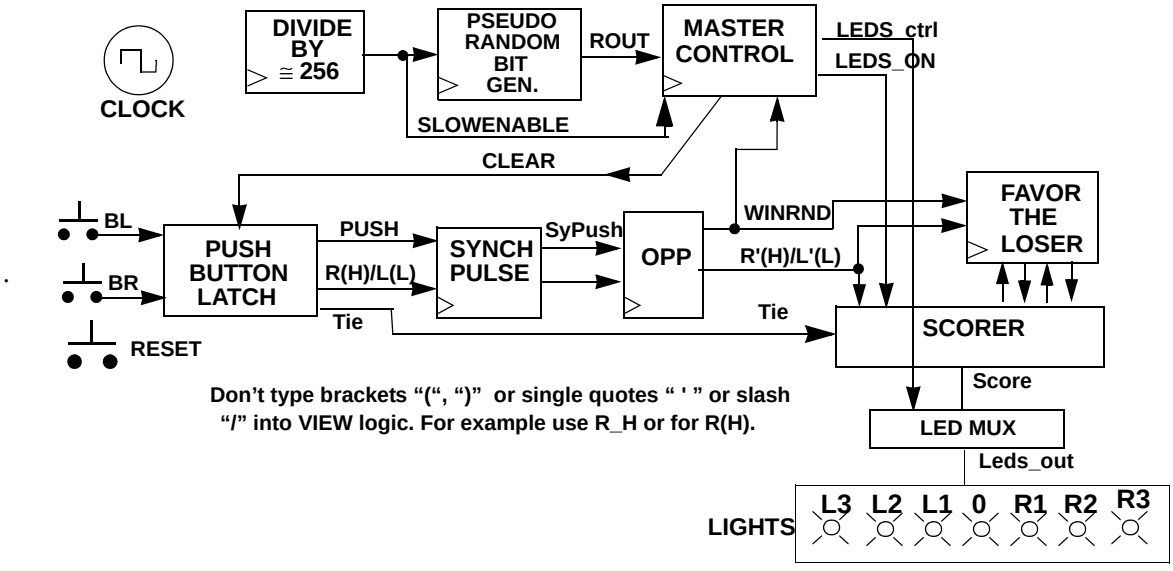
f. Any asynchronous circuit which latches, i.e. your push button circuit, must be checked for hazards and races.

g. Any unclocked latches in your circuit are asynchronous. If you place cross coupled NAND gates in the synchronous part of your circuit, you have created an asynchronous island in a supposedly synchronous circuit. You must then check for hazards, races and essential hazards at all latch inputs.[1]

4.0 Initial Planning

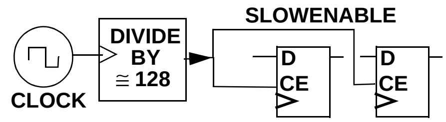
As with most specifications, there are a lot of subspecifications buried in the design document. You should read through the complete document and list these before you start serious design. .

Figure 2. A possible block diagram for the game. The blocks are completely synchronous and run from a common clock, except for the PUSH BUTTON LATCH which is asynchronous.



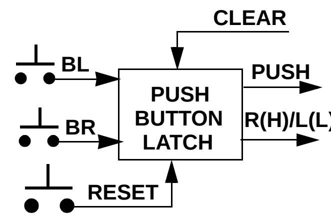
1. There is another reason for not placing any of the above asynchronous concepts into a synchronous circuit. The testing method (Scan Testing) used for modern digital circuits will not work with embedded asynchronous elements.

4.1 Clock and Clock Divider



The master clock should be about 500 Hz. This is fast enough so humans will not see appreciable delay, but it is slow enough so a divide by 256 can give the 1/2 second timing humans can see. The divide by 256 gives a pseudo-clock, a pulse one-clock-cycle wide every 2^8 or 256 clock cycles., called SLOWENABLE.  . This pulse can enable/disable the flip-flops so they can only change at a 1/2 sec rate, even though they are clocked at 500 Hz.

4.2 Push Button Latch



This must be asynchronous since it must tell which button was pushed first within a couple of gate delays. It debounces the buttons by merely latching on the first contact of the first button pushed. It should hold the signal until reset by the CLEAR input. It has three inputs: BL, BR and CLEAR.

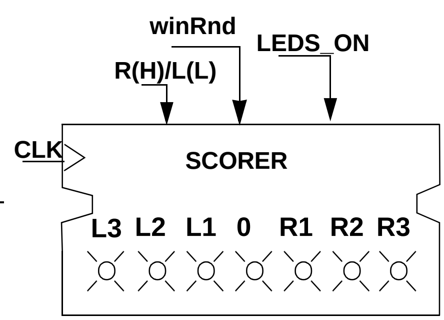
4.3 The FAVOR-THE-LOSER circuit

When a player is in the L3 or R3 LED positions and the other player wins, the lights will jump back two positions. The block diagram above seems to suggest that this is a separate block. Another, probably better, way is to build it into the scorer. We would recommend you build the Favor the loser circuit into the Scorer. Note it only shifts back one position if the winner jumps-the-light.

4.4 The SCORER

It is suggested that the SCORER have three inputs:

WinRnd.............. a one clock-cycle pulse which comes after a button push. It tells that somebody has won the round.
R(H)/L(L) ≡ R/L. tells which button was pushed first. Right if high , left if low.
LEDS_ON............ tells if a LED is lit and thus can be used to tell if a player pushes his button ahead of time (jumps the light) The score will move left or right according to the formula



Right = WinRnd·MR.

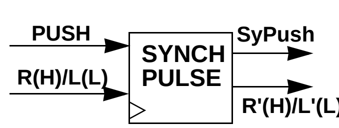
Left = WinRnd·$\overline{MR}$

Where MR............move right = 1; if R/L,LEDS_ON =1, 1, or 0,0
0; if R/L,LEDS_ON =0, 1, or 1,0.

Thus MR signifies a right move - if (the right button was pressed) and (the player did not jump the light),or (the left button was pressed) and (the player jumped the light).

4.5 The Synchronizer Circuit

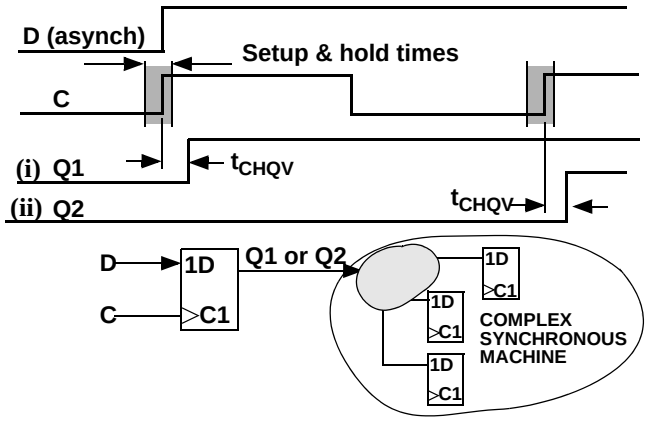
The output from the PUSH BUTTON LATCH is asynchronous, that is it may change at any time including on the clock edge. The SCORER may **not** count correctly if its inputs change very near the clock edge.



As you will learn in the lectures, inputs to an FSM's D flip-flops must never change near the clock edge. Near means inside the fli-flops setup and hold times. If the D inputs changes near the clock edge, one does not know whether the flip-flops will capture the old D value, the new D value, or some will capture the old and some the new.

The PBL output must be synchronized, so it cannot change on the clock edge. This is done by passing it through a D flip-flop. The clock-to-output delay, t_{CHQV}, inside the flip-flop will delay Q till after the clock edge. Thus Q acts as a slightly delayed synchronous D. FIGURE 2 shows how this happens.

FIGURE 2. How a D flip-flop synchronizes a signal. If the D input changes inside the flip-flop setup or hold times, the signal may:
i) be captured, in which case it appears as the Q_1 output, rising a safe distance after the clock edge.
ii) have the capture delayed until the next clock edge. Then it will appear as Q_2, which is delayed, but still a safe distance from the active clock edge. Either Q1 or Q2 is a safe, synchronized signal to feed to multiple flip-flops in a synchronous machine.
iii) go metastable which is not a problem with very slow clocks like this one.



The tentative design suggests two input signals:

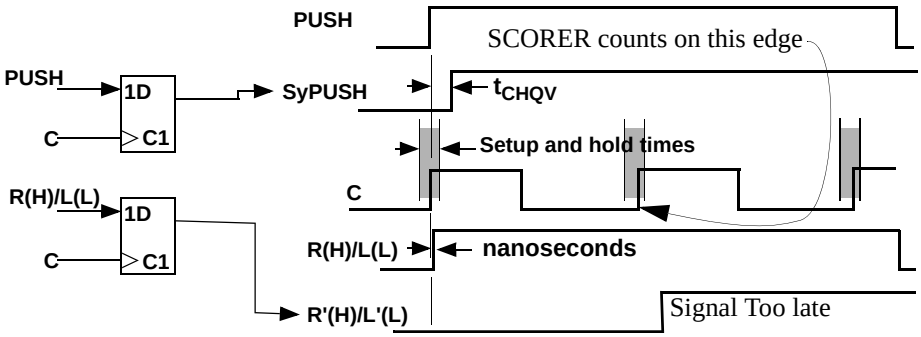
PUSH(H)....is latched true as soon as any button is pushed.

R(H)/L(L)...tells which button was pushed first, the right button or the left. .

While one asynchronous signal can be synchronized by passing it through a D flip-flop, two logically related asynchronous signals cannot. They will appear to be properly synchronized **until they both** change on the same clock edge. Then one may be captured by its D flip-flop but the other may not.

The problems can be seen by looking at PUSH and R(H)/L(L) in FIGURE 3 The two input signals PUSH and R(H)/L(L) are logically related and change at about the same time. Suppose PUSH and R(H)/L(L) both change inside the setup and hold time. Then the Synchronizer circuit might latch the correct PUSH and old incorrect value of R(H)/L(L). On the next clock cycle R'(H)/L'(L) would be correct but it is too late! The SCORER has already moved to the wrong light.

FIGURE 3. The signals PUSH and R(H)/L(L) are logically related and occur only nanoseconds apart. Suppose PUSH is captured on one cycle, but R(H)/L(L) is not captured until the next cycle. Then left will appear to have won because R/L = 0 on the next clock edge when SCORER is enabled.



The cure is simple. Latch only PUSH; do not latch R(H)/L(L). See FIGURE 5 on page 5. We know when PUSH goes high, R(H)/L(L) is at worst nanoseconds behind it. The SCORER reacts, not to the clock edge where PUSH is captured, but on the next (2nd) clock edge. By that time R(H)/L(L) will be correct for sure

Alternately, PUSH might not be captured and then SyPush would be delayed a clock cycle. See FIGURE 4. Then the SCORER would not react until the 3rd clock edge. By that time R(H)/L(L) will have been correct for two cycles..

FIGURE 4. This time PUSH was too late for the 1st clock edge and was captured on the 2nd as SyPush.

R(H)/L(L) does not go through a flip-flop and rises within nanoseconds of PUSH rising.remains correct until cleared.

The SCORER reacts to SyPush on the 3rd clock cycle. It counts in a direction given by R(H)/L(L). Both signals are stable at the 3rd clock edge.

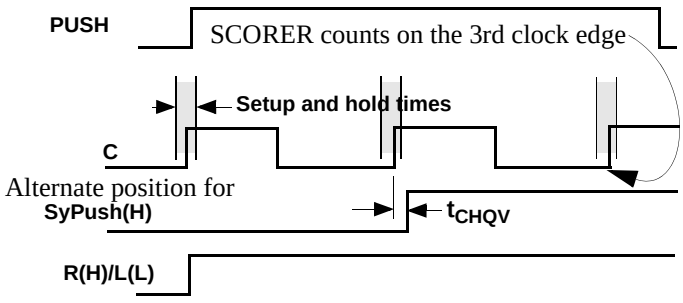
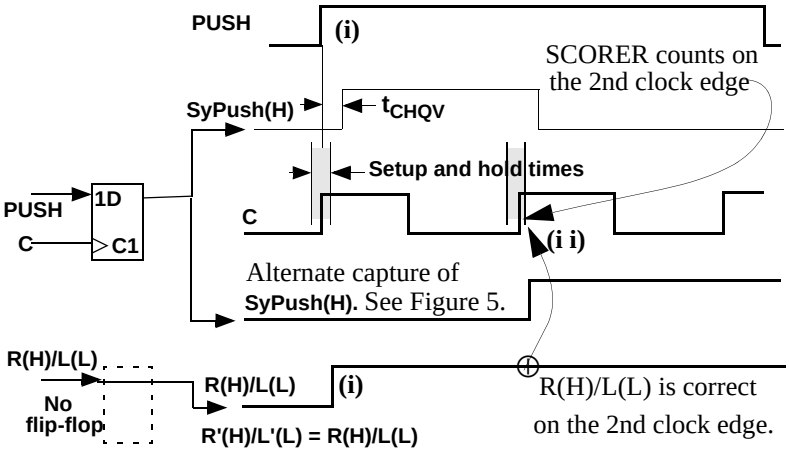


FIGURE 5. The signals PUSH and R(H)/L(L) are logically related and occur only nanoseconds apart.

(i) Suppose PUSH rises very near the clock edge, SyPush will rise either on the same clock edge or the next one. However, R(H)/L(L) will be correct within nanoseconds of PUSH rising, and will remain correct until cleared.

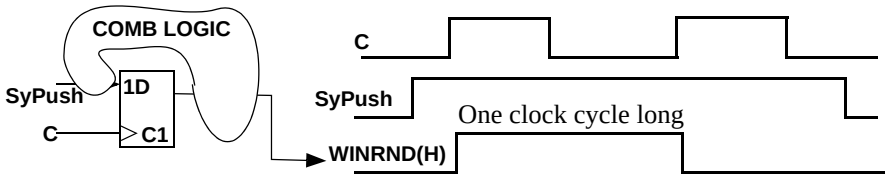
(ii) The SCORER reacts to SyPush on the 2nd cycle. It counts up or down according to R(H)/L(L) which is stable long before the 2nd clock edge.



4.6 The OPP (One-Pulse-Per-Push) Circuit

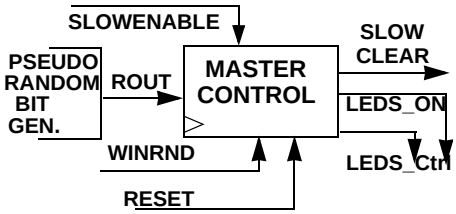
The SCORER must count only once, be it up or down, for each round played on the game. Otherwise each push will count for many wins. The OPP circuit insures this by giving out one-pulse-per-push (round). The synchronized push signal, SyPush, will initiate an output signal WINRND which will last exactly one clock cycle.

FIGURE 6. The WINRND signal lasts a single clock cycle so one round of the game will only move the SCORER once.



4.7 The Pseudorandom Counter

This circuit creates ROUT, a string of 1s and 0s which is unpredictable to the players. The time between bits should be about 1 second.



4.8 The Master Controller

The controller times the action of the game as follows:

a. Someone pushes a button either as a win or a jumped light.
b. The LEDs will come on immediately after a button is pushed.
c. The LEDs stay on for 1/2 to 1 second so it is clear who won; under 1/2 is too short, longer would be boring
d. Then all the LEDs go out for a random time, minimum of 1/2 second.
e. A single LEDs come back on after this random time.

4.8.1 Slow Players and Gloating Players

If the players are slow, the controller circuit should keep the lights on until a push occurs. It also holds the light on an additional 1/2 to 1 second after a push so the movement of the lights can be observed and the winner can gloat.

4.8.2 Jumping the Light

If a player “jumps the light”, the light should change immediately. However the timing is the same as for a legitimate push. Also the controller should wait the “gloat period” and start the next round.

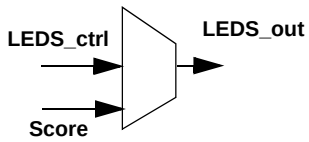
4.8.3 Clear

After the LEDs go off, one must clear the push button latch. The master controller send out a CLEAR pulse to do this. This arms the push buttons quickly and allows detecting any player “jumping the light” at anytime except the first clock cycle of the dark period (about 1/250 of a second).

4.9 Led Mux

Led Mux is used to choose between whether the display lights should show all lights ON or all lights OFF or display the Score at that point of the Game depending on t

LEDS_Ctrl signal from the master controller.



5.0 Implementation details.

5.1 The SCORER Circuit

If you decide to add extras, like separate win states, flashing lights, music, or a siren, you are free to do so however design them into your state machine now. Do not stick patches on at the end!

5.2 The Master Controller

The inputs signals shown for this subcircuit are:

ROUT..... the bit string from the pseudorandom bit generator.
WINRND.....goes high for one clock cycle after a button is pushed.
SLOWENABLE.... which acts like a slow clock

The output signals might be:

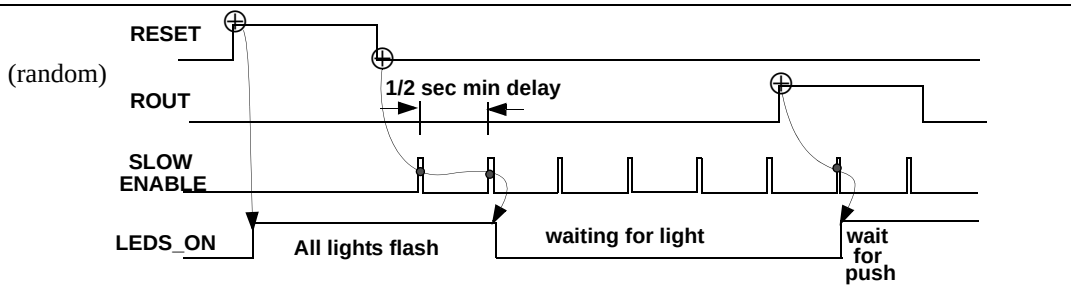
LEDS_ON....to say one or more LEDs are on.

LEDS_Ctrl.... controls the LED MUX to show the appropriate lights
CLEAR..which generates CLEAR signal to clear the PUSH-BUTTON LATCH.

5.2.1 Initial Lamp Check

On RESET, all the LEDs should immediately come on, and stay on for 1 sec after RESET is released. This shows that all the lights work. All LEDs should then go out and the center LED should come back on after the next rising edge of ROUT.

FIGURE 7. The game after the initial reset. Here LEDS_ON comes on again, a Moore machine delay after the rising edge of ROUT. Note SLOW ENABLE does not run during RESET.



5.2.2 The Mid-Game

Consider midgame play as in FIGURE 8.

(a) When the ROUT signal goes high it raises LEDS_ON on the next SLOWENABLE. This turns on a LED and signals the players to push. Note ROUT rises just after pulse hence it cannot cause a reaction until pulse B.

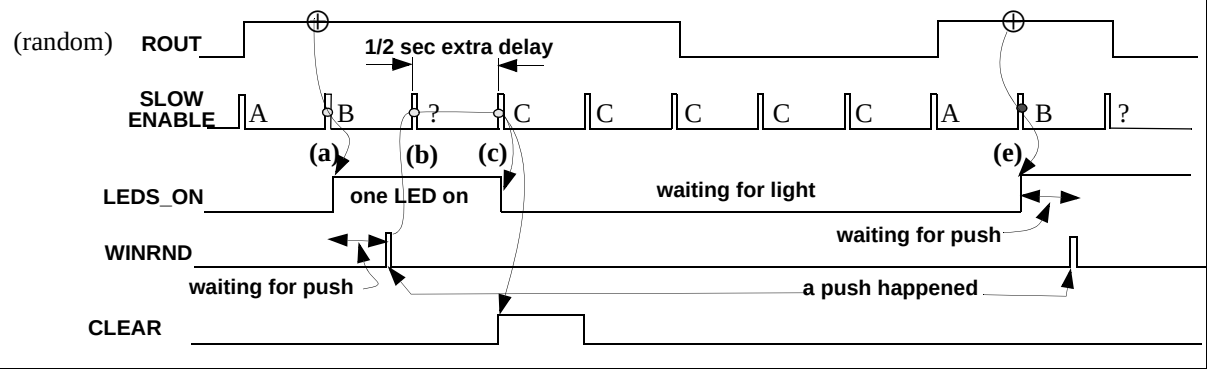
(b) Pushes generate a WINRND signal which moves the light and will eventually turn off LEDS_ON. However LEDS_ON stays on for at least 1 second after WINRND, so the players can see who won, and remove their fingers.

(c) After the 1 sec extra delay, LEDS_ON will go off even though ROUT may stay high.

(d) The Master controller generates CLEAR which clears the push-button latch.

(e) LEDS_ON should wait a short time before coming on again. If ROUT stays high it would come back on the next SLOW ENABLE.

FIGURE 8. .The midgame after one or more rounds.
(a) ROUT rising turns on the LEDs, the signal to the players to push. (b) A push raises the WINRND pulse, which will, after 1/2 to 1 sec, (c) lower LEDS_ON and pulse SLOWCLEAR. (d) SLOWCLEAR in turn will pulse FASTCLEAR. (e) The next LEDS_ON will wait till the next rising edge of ROUT.



5.3 Pseudorandom Sequence Generators

FIGURE 11 on page 8 shows a widely used circuit which generates a sequence of bits. The output may be taken from any of Q1,Q2.Q3 or Q4. While the sequence is periodic, inside of its 15 cycle period, it passes many of the statistical tests for randomness. Notice it is a 15, not a 16,cycle FSM. The 16th state is all zeros. IF IT GETS INTO THAT STATE IT WILL NEVER COME OUT.

These pseudorandom shift registers can be made in any length but the most efficient ones have N flip-flops and have sequence length 2^N-1. The final state is all zeros and not used.

Connections for sequences with periods up to 511 bits are shown in FIGURE 10 .

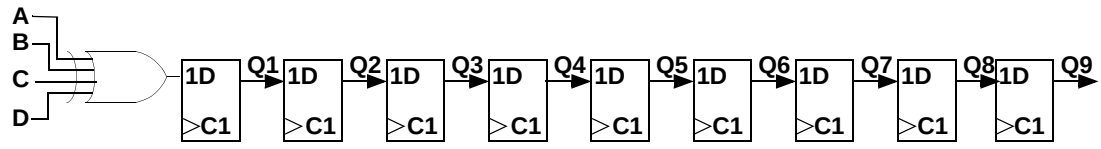


FIGURE 10 A general pseudorandom bit stream generator. The number of flip-flops used depends on the length of sequence desired. Thus, for a 15 bit sequence remove all but 4 flip-flops. For a 31 bit sequence, remove all but 5 flip-flops, etc. See FIGURE 11 and FIGURE 12

SEQUENCE LENGTHS FOR FIGURE 10

Link	15	31	63	127	255	511
A to	Q3	Q3	Q5	Q6	Q2	Q5
B to	Q4	Q5	Q6	Q7	Q3	Q9
C to	Omit	Omit	Omit	Omit	Q4	Omit
D to	Omit	Omit	Omit	Omit	Q8	Omit

Examples using this table are shown in FIGURE 11 and especially FIGURE 12.

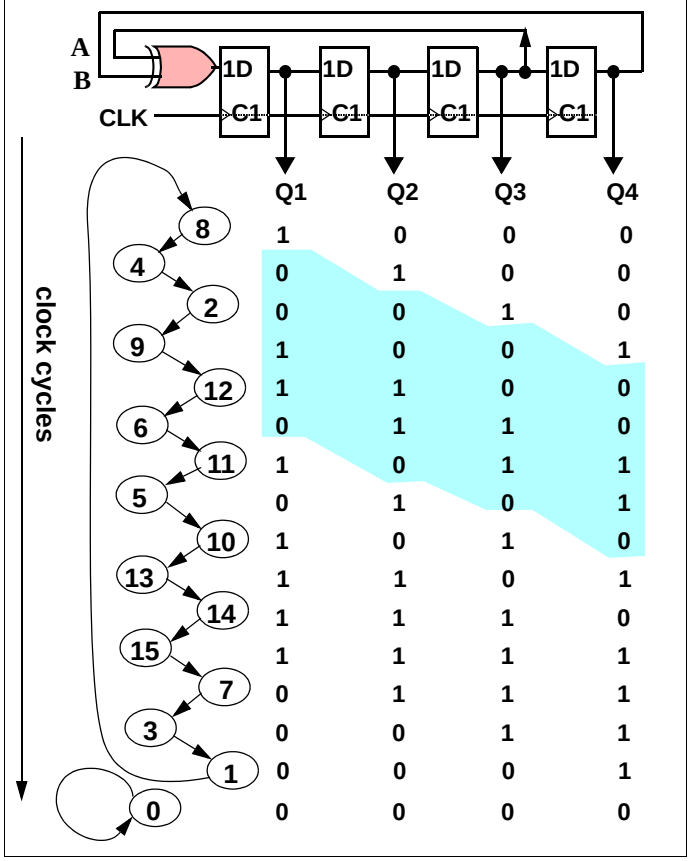
Note that the generator must not be all zeros initially or it will not start, i.e. this is a state machine which will lock up in the 0000 state!

Note cycle 0 and 15 are the same The sequence is always of length 2^N-1 because the flip-flops can never be all zeros!

Verilog code to imply the random number generator on the right, is given below.

```
.reg [4:1] Q;
always @(posedge clk or posedge clr)
   begin if (clr) Q<=4'd8;
      // Must not reset state to zero.
      else if (clk)
         begin
         Q[4:2]<=Q[3:1];        //(line 7)
          Q[1] <= Q[1]^Q[4];  //(line 8)
        end
/*     The "<=" is called a nonblocking
assignment. The Qs on the right  all use
the values they had at the begin.
   Thus  line 8 uses the original Q[4] and
not the revised value from line 7.*/
```

FIGURE 11 A circuit made from XOR and FFs which gives repeatable "random" number sequences. This example can be constructed from the general example of FIGURE 10 by connecting A to Q3 and B to Q4



5.4 Generating the SLOWENABLE

This circuit must generate a single pulse every 256 clock pulses. One needs a 8 bit counter which gives a carry pulse (TC) out every time it rolls over from say 11111111 -> 00000000. Counters are trivial to generate in Verilog.

5.5 The Basis of the Design of the Push Button Circuit

The push button circuit shall be as fair as modern technology can make it.

i) It must tell who pushed first within a time equal to the propagation delay of two inverters.

ii) It shall have no theoretical bias toward one player. Because of the difference in delays between gates and/ or lead lengths, it is impossible not to have some bias in the circuit after construction.

iii) If your circuit goes into a tie state, it shall either:

a. have equal theoretical probability of exiting toward either player.

b. leave the light stationary for that round.

iv) It should not depend on which player releases their button first. In a poor design, this can happen in the case of a tie, which causes the winner to be determined by the bounce properties of the push buttons.

5.6 The Design of the Synchronizer Circuit

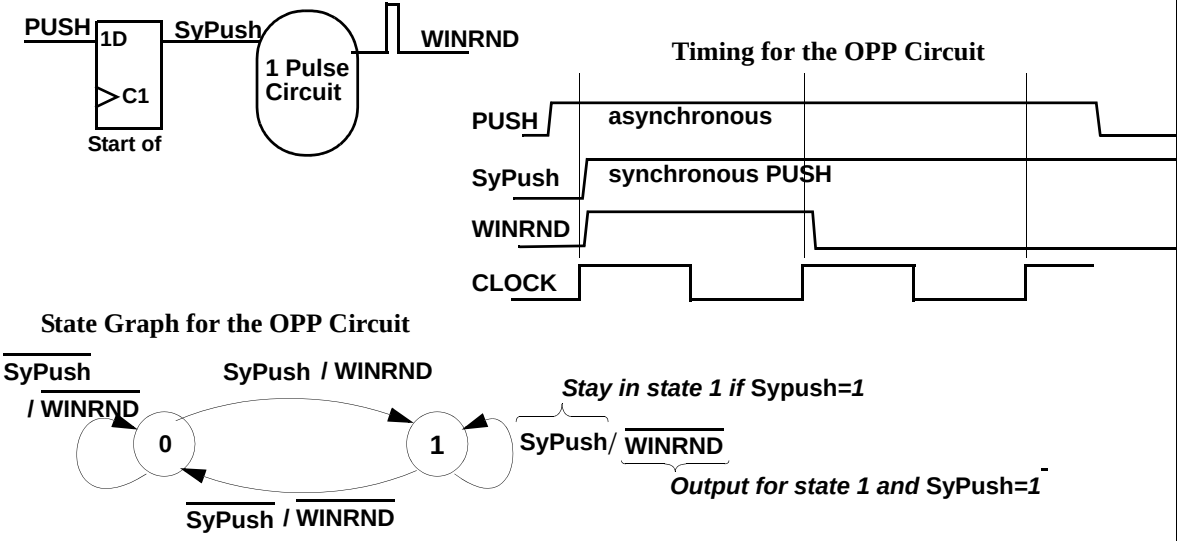
The synchronizer circuit takes the asynchronous PUSH input which may change anytime and makes it into SyPush which cannot change near the clock edge. That is during the setup or hold times of the flip-flops.

Logically independent asynchronous input signals should be sent through a single D flip-flop before they enter the rest of the synchronous machine. But this will not work for two logically-dependent signals. See Sect 4.5.

5.7 The Design of the OPP (One-Pulse-per-Push) Circuit

It gives out a single synchronous pulse one clock cycle long. It always gives one pulse, and only one pulse.

FIGURE 12 The sychronizer D flip-flop and a the OPP circuit. The OPP circuit is simpler to implement as a Mealy machine, as shown by its state graph.



More Details on Implementation will be on the Cover page for each lab, don't forget to check them!!!