Lab 7: Using RAM

Wednesday, November 06, 2024 2:55 PM

Jack Burton

TAs: Cory and Karen

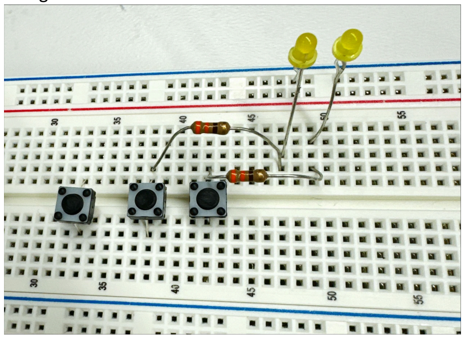
Summary: Uses iCE40's built-in RAM to create a visual looper which records a sequence of LEDs and then repeatedly plays it back from memory.

Bench: 11

Lab Journal

L1

Using two channels:



L2

I will sample 256 times per second, and the recording duration will be 8 seconds.
This is a total of 256 * 8 = 2048 samples, so I will need 2048 bits.

Instantiating Memory

L3

Created project on Lattice, added ramdp.vhd to the project, and then instantiated it as a submodule within top.vhd.

Counter

L4

1. Added HSOSC as a component to top.vhd just like in lab 6
2. Built 2-bit fastcounter module, which rolls over after 2 bits reach 11. This will roll over for each 2-bit looper sample.
3. Built second counter using provided code within top module, and a signal sample_counter.

Wiring Everything Up

L5

Connected each button to 5V, and then:
1. Connected the record button to a read_write input (std_logic)
2. Connected the two writing buttons to a write_value input (2-bit unsigned)
3. Connected LEDs (through 4.7k ohm resistors) to read_value output (2-bit unsigned)

After *much* troubleshooting, I realized I hadn't actually grounded the push-buttons.. But after grounding them, the LEDs were correctly playing back the recorded LED input.

L6

I added a signal to store the previous value of the record button, at the end of the if rising_clock. Then I added the conditional logic, which essentially just checked if the previous value and the current value differed. If either of those conditions was met, the sample counter was reset to 0. This successfully caused immediate replay.

L7

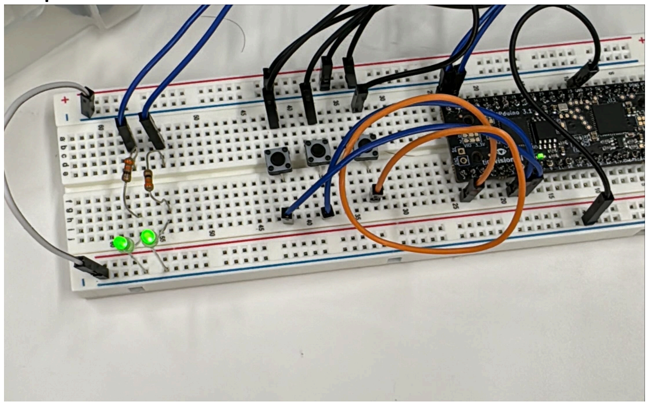
I added another signal to hold the sample_counter's saved value, and placed this in the conditional for when the record button is released. This should save the sample_counter's position when the record button is released

Then, within this same conditional, I added another conditional which checks if sample_counter reaches 0 (so if it reaches the end of the recording/sample size). Then this sets sample_counter to the sample_counter saved value signal, to loop back to the recording rather than continuing to iterate through the whole RAM.

This successfully works and allows me to record any length pattern and watch it playback endlessly.

Video of circuit working *(recording a small sample and then replaying it—my sample time is just large that's why there is the delay)*: https://drive.google.com/file/d/1AsiDn3Wka3SxJyyThhfPDm7_93RfrgkZ/view?usp=sharing

Completed Circuit:



VHDL Code

(ramdp.vhd was unmodified from the provided file)

Top.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity top is
port (
read_write : in std_logic; -- pin 2
write_value : in std_logic_vector(1 downto 0); -- pins 3 and 4
read_value : out std_logic_vector(1 downto 0) -- pins 28 and 38
);
end top;

architecture synth of top is
component ramdp is
  generic (
    WORD_SIZE : natural := 2; -- Bits per word (read/write block
size)
    N_WORDS : natural := 64; -- Number of words in the memory
    ADDR_WIDTH : natural := 6 -- This should be log2 of N_WORDS; see
the Big Guide to Memory for a way to eliminate this manual
calculation
   );
  port (
    clk : in std_logic;
    r_addr : in std_logic_vector(ADDR_WIDTH - 1 downto 0);
    r_data : out std_logic_vector(WORD_SIZE - 1 downto 0);
    w_addr : in std_logic_vector(ADDR_WIDTH - 1 downto 0);
    w_data : in std_logic_vector(WORD_SIZE - 1 downto 0);
    w_enable : in std_logic
  );
end component;

component HSOSC is
generic (
CLKHF_DIV : String := "0b00");
port (
CLKHFPU : in std_logic := 'X';
CLKHFEN : in std_logic := 'X';
CLKHF : out std_logic := 'X'
);
end component;

component fastcounter is
port(
clk : in std_logic;
result : out std_logic_vector(22 downto 0)
);
end component;

signal clk : std_logic;
signal sample_counter : unsigned(5 downto 0) := 6d"0";
signal intermediate_counter : std_logic_vector(22 downto 0);
signal read_write_prev : std_logic;
signal sample_counter_saved : unsigned(5 downto 0);
begin
osc : HSOSC generic map ( CLKHF_DIV => "0b00")
port map (CLKHFPU => '1',
CLKHFEN => '1',
CLKHF => clk);
fc : fastcounter port map (clk, intermediate_counter);
ram : ramdp port map (clk, w_enable => read_write, w_data =>
write_value, r_addr => std_logic_vector(sample_counter), w_addr =>
std_logic_vector(sample_counter), r_data => read_value);
process(clk) is
begin
if rising_edge(clk) then
if intermediate_counter  = 23d"0" then -- Counter just rolled over
sample_counter <= sample_counter + 1;
end if;
-- L6: If record button is pressed or released, reset sample_counter
(for immediate replay)
if (not read_write and read_write_prev) or (read_write and not
read_write_prev) then
sample_counter <= 6d"0";
end if;
read_write_prev <= read_write;
-- L7: Looping over recorded portion, not entire memory
if (read_write and not read_write_prev) then -- control button
released
sample_counter_saved <= sample_counter;
if sample_counter = 6d"0" then
sample_counter <= sample_counter_saved;
end if;
end if;
end if;
end process;
end;
```

<u>Fastcounter.vhd</u>

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity fastcounter is
port(
clk : in std_logic;
result : out std_logic_vector(22 downto 0)
);
end fastcounter;

architecture synth of fastcounter is
signal count : unsigned(22 downto 0);
begin
process (clk) begin
if rising_edge(clk) then
count <= count + 1;
end if;
end process;
result <= std_logic_vector(count);
end;
```