Abstract

The goal of this lab is to use VHDL programming to create an eight-bit parallel to serial transmitter circuit. The overall program includes an eight-bit parallel to serial transmitter and a serial to parallel receiver. The two are completely separate in that no internal variables or signals from one module are used in the other. The only connection to the two modules is through the extout and extin ports. The transmitter sends the clock signal at 1kHz, start signal, and the data (through the extout0, extout1, and extout2 respectively). The receiver receives the respective signals on extin0, extin1, and extin2. We were able to successfully send and receive data using the two modules.

The Process

The process of the lab was simple, on paper. All that needs to be done is create a module that sends data and a module that receives data. The issue is that the data is being sent in series using a right shift operation. The receiver does the same thing but "in reverse." With that in mind, the design of the transmitter would be more important, and the receiver design would just be a modified transmitter. The states for the transmitter are as seen in figure – 1. There are three states that were used: Load, Send Start Signal, and Finish Transmit. In the Load state, the input (on switches 0-7) would be loaded into our data vector. In the Send Start Signal state, the transmitter would send the start signal (from button 2) which is used to tell the receiver that data is about to be sent over. In the Finish Transmit state, the transmitter sends the bits from the data vector one bit at a time using a right shift operation. The receiver only required two states: Shift and Store (as seen in figure – 2). In the Shift state, the receiver takes in the data that's being sent from the transmitter and uses a right shift operation to place the data into the receiver's data vector. In the Store state, the receiver takes the filled data vector and displays the bits on LEDs 0-7. This leads to figure – 3, the top-level block diagram. It shows the connection between the two modules. That is, transmitter takes in data from the inputs (master clock, sw(0-7), and btn2) and sends them out of ports extout(0-2). The receiver takes in the data one extin(0-2) and displays them on led(0-7). A closer look at how the transmitter and receiver work is found in figure – 4 and figure – 5 respectively. In it, it can be seen that both the transmitter and the receiver use a 3 bit counter to tell the FSM control when to stop sending data.

Conclusions

This lab was a success in that the transmitter was able to successfully send data serially to the receiver. Using an oscilloscope, we found that the data was being sent in one clock cycle to the receiver. We did have some minor setbacks, (like almost frying the circuit by shorting the circuit to ground through the 3.3v port) but they didn't cause enough of an impact on the lab to require redesigns.

Figures

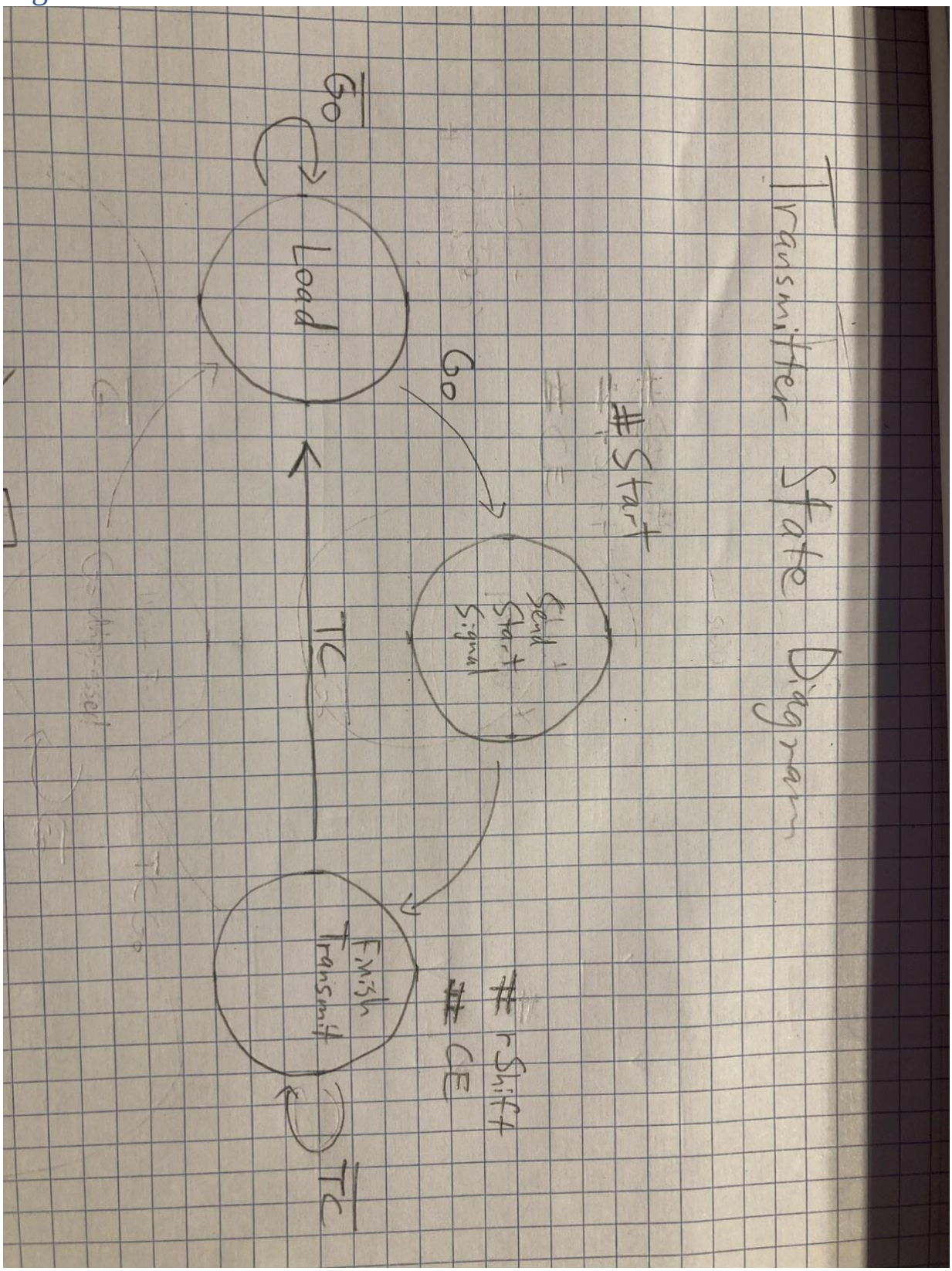

Figure 1 The transmitter state diagram

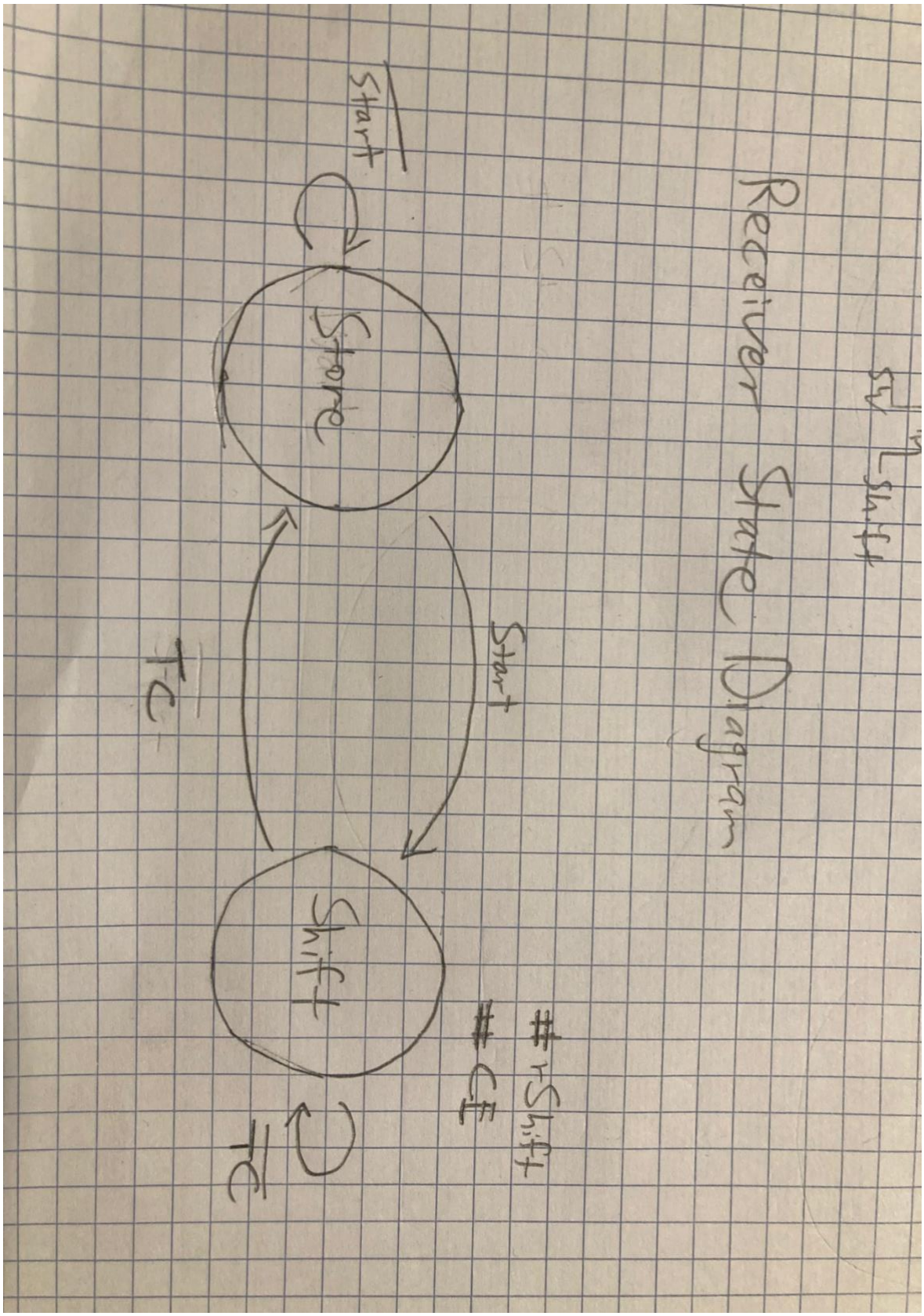

Figure 2 The receiver state diagram

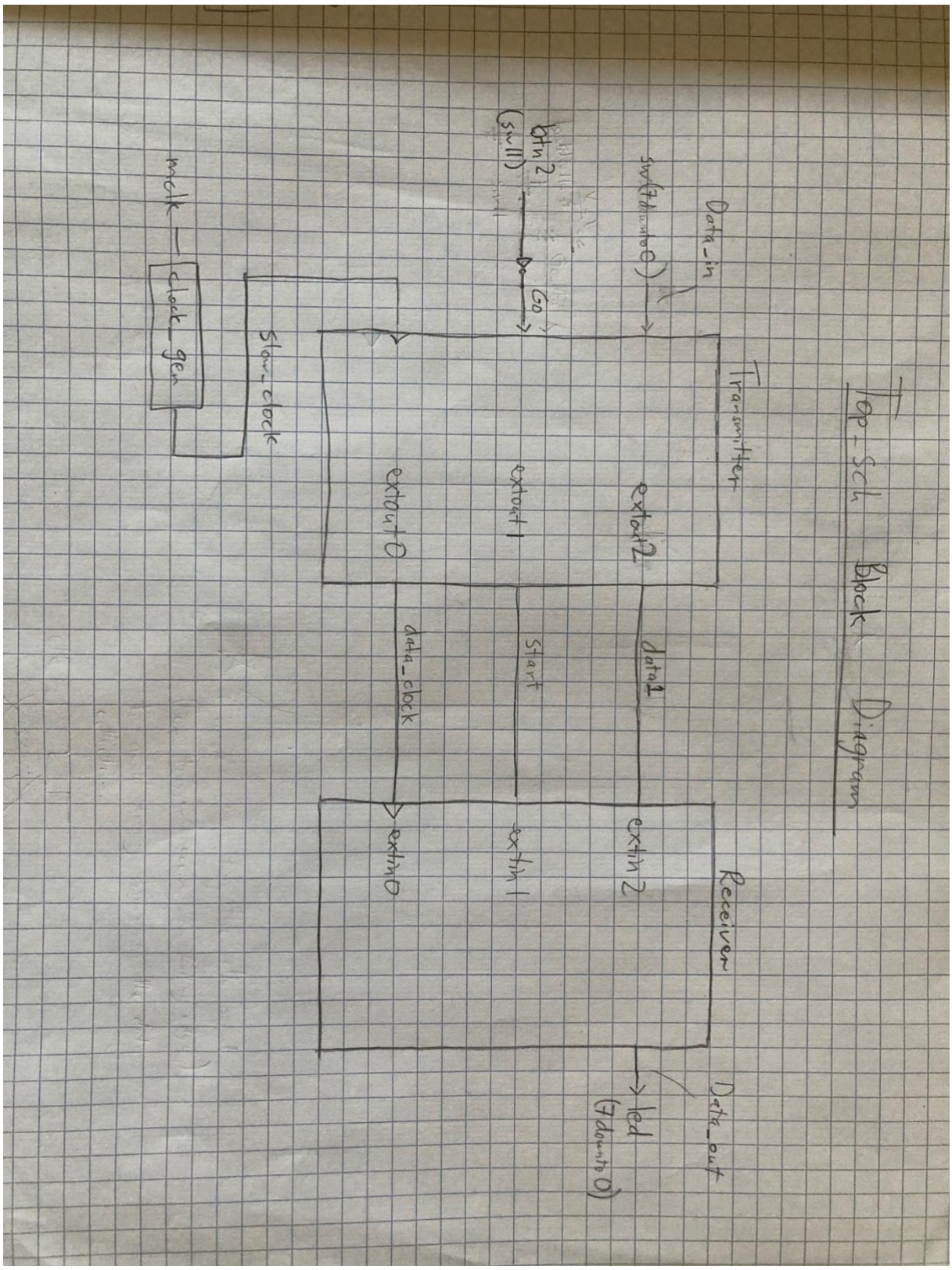

Figure 3 The top level block diagram

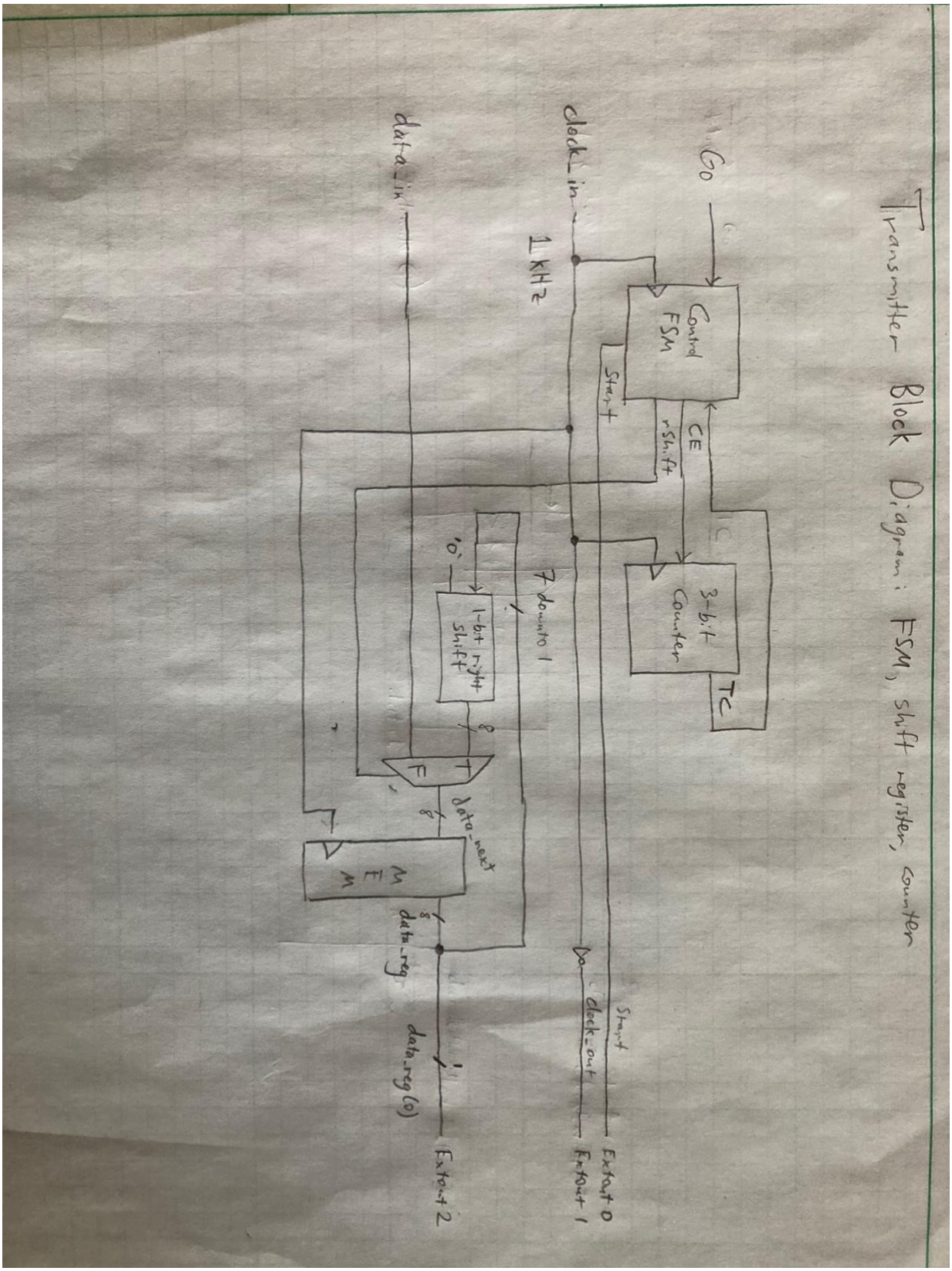

Figure 4 The block diagram for the transmitter

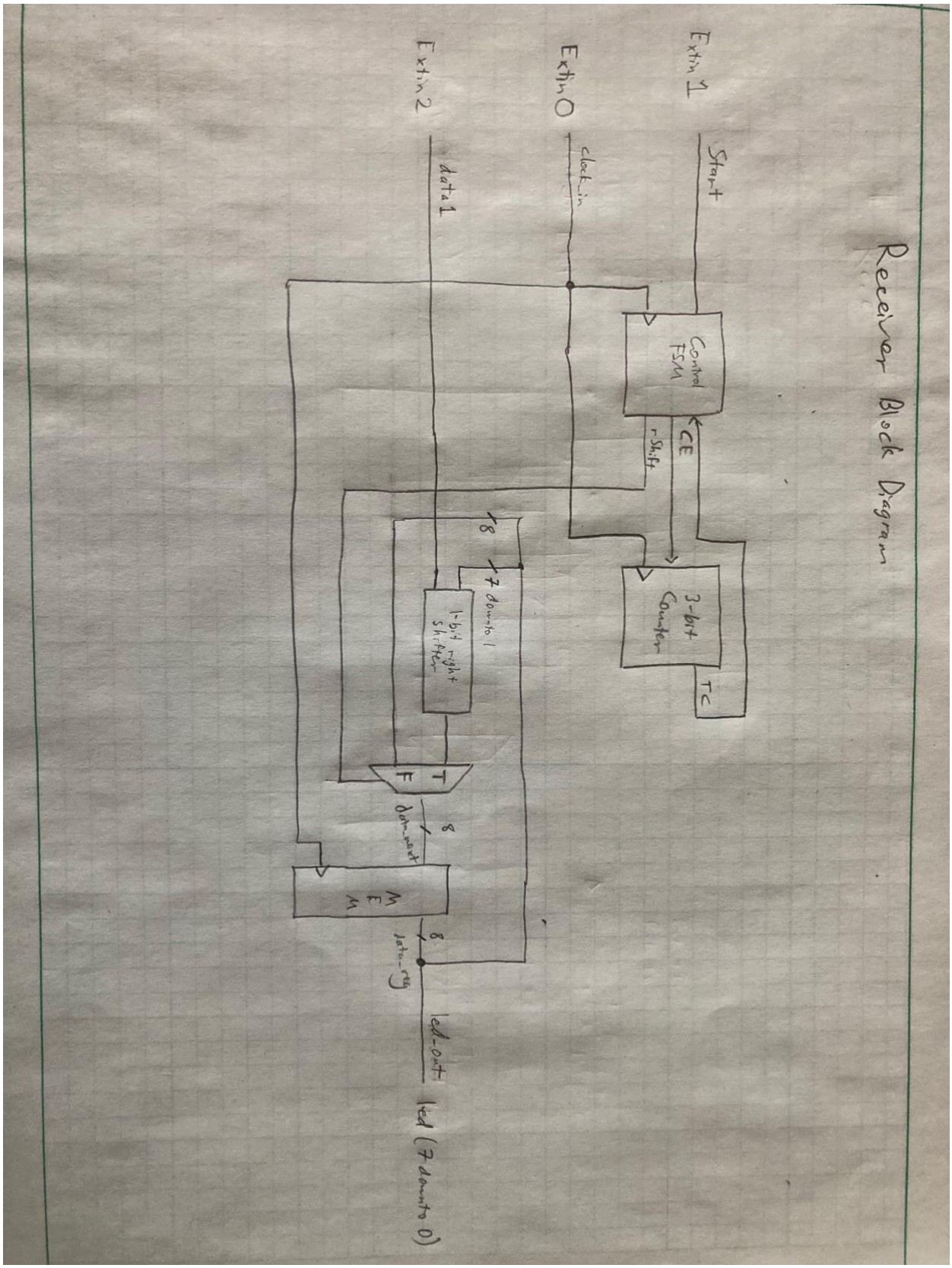

Figure 5 The receiver block diagram

The Code

```
----------------------------------------------------------------------------------
-- Company: Walla Walla University
-- Engineer: Eric Walsh & Nicholas Zimmerman
--
-- Create Date:    14:06:01 11/09/2021
-- Design Name:
-- Module Name:    Lab7_top_sch - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter3bitCycle is
	port(	clk : in std_logic;
			CE : in std_logic;	--CE triggers the full counting sequence
			TC : out std_logic );
end counter3bitCycle;

architecture Behavioral of counter3bitCycle is
	signal cnt_next, cnt_reg : unsigned(2 downto 0);
begin
	--DFF
	process(clk)
	begin
		if (clk'event and clk='1') then
			cnt_reg <= cnt_next;
		end if;
	end process;

	--N.S. Logic
		cnt_next <= cnt_reg+1 when (cnt_reg=0 nand CE='0') else cnt_reg;

	--Output Logic
		TC <= '1' when cnt_reg=7 else '0';
```

```
end Behavioral;
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity Lab7_top_sch is
	port(	--Clock
			mclk: in std_logic;

			--Transmitter
			sw: in std_logic_vector(7 downto 0);
			btn2: in std_logic;
			extout: out std_logic_vector(2 downto 0);

			--Receiver
			extin: in std_logic_vector(2 downto 0);
			led: out std_logic_vector(15 downto 0) );
end Lab7_top_sch;

architecture Structural of Lab7_top_sch is
	signal slow_clock: std_logic;
	--clock_gen
	signal clk_next, clk_reg : unsigned(14 downto 0);	--2^15=32768
	signal t_next, t_reg : std_logic;
	--Between Tx and Rx
	signal data1: std_logic;
	signal Start: std_logic;
	signal data_clock: std_logic;

	--Component declarations
		--Tx
		component Tx
			port(	data_in: in std_logic_vector(7 downto 0);
				Go: in std_logic;
				clock_in: in std_logic;

				data1: out std_logic;
				Start: out std_logic;
				clock_out: out std_logic;
				--debugging
```

```
                                led_out: out std_logic_vector(7 downto 0) ); --leds for
debugging
        end component;
        --Rx
        component Rx
                port(   data1: in std_logic;
                                Start: in std_logic;
                                clock_in: in std_logic;

                                led_out: out std_logic_vector(7 downto 0) );
        end component;
begin
        ----------------------------------------------------------------------------
        --Clock_gen    (50 MHz to 1 kHz)
        ----------------------------------------------------------------------------
        process(mclk)
        begin
                if (mclk'event and mclk='1') then
                        clk_reg <= clk_next;
                        t_reg <= t_next;                --TFF
                end if;
        end process;

        clk_next <= (others=>'0') when clk_reg=24999 else clk_reg+1;
        t_next <= (not(t_reg)) when clk_reg=24999 else t_reg;

        Clk_Buffer: BUFG                -- Buffered clock line
        port map ( I => t_reg, O => slow_clock);
        ----------------------------------------------------------------------------

        Transmitter: Tx
                port map (      --inputs
                                        data_in => sw,
                                        Go => not btn2,         --Buttons are high on logic
low so need inverting
                                        clock_in => slow_clock,
                                        --outputs
                                        data1 => extout(2),
                                        Start => extout(1),
                                        clock_out => extout(0),
                                        led_out => led(15 downto 8) );
        Receiver: Rx
                port map (      --inputs
                                        data1 => extin(2),
                                        Start => extin(1),
                                        clock_in => extin(0),
```

```
                                        --outputs
                                        led_out => led(7 downto 0) );

end Structural;
----------------------------------------------------------------------------------
-- Company: Walla Walla University
-- Engineer: Eric Walsh & Nicholas Zimmerman
--
-- Create Date:    14:32:32 11/09/2021
-- Design Name:
-- Module Name:    Tx - Structural
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tx is
        port(   data_in: in std_logic_vector(7 downto 0);
                        Go: in std_logic;
                        clock_in: in std_logic;

                        data1: out std_logic;
                        Start: out std_logic;
                        clock_out: out std_logic;       --data clock

                        --debugging
                        led_out: out std_logic_vector(7 downto 0) );
end Tx;

architecture Structural of Tx is
```

```
	signal rShift, CE, TC : std_logic;

	--Control FSM
	type state_type is (Load, StartTransmit, FinishTransmit);
	signal control_next, control_reg : state_type;

	--Data line
	signal data_next, data_reg: std_logic_vector(7 downto 0);

	--Component declarations
		--counter
		component counter3bitCycle is
			port(	clk : in std_logic;
					CE : in std_logic;
					TC : out std_logic );
		end component;
begin
	-----------------------------------------------------------------
	--Output clock inversion to run the receiver on
	--the falling clock edge
	-----------------------------------------------------------------
	clock_out <= not clock_in;
	-----------------------------------------------------------------
	--Control FSM
	-----------------------------------------------------------------
	process(clock_in)
	begin
		if(clock_in'event and clock_in='1') then
			control_reg <= control_next;
		end if;
	end process;

	--NS Logic
	process(control_reg, Go, TC)
	begin
		case control_reg is
			when Load =>
				if (Go='1') then
					control_next <= StartTransmit;
				else
					control_next <= Load;
				end if;
			when StartTransmit =>
				control_next <= FinishTransmit;
			when FinishTransmit =>
				if (TC='1') then
```

```
                        control_next <= Load;
                    else
                        control_next <= FinishTransmit;
                    end if;
        end case;
end process;

--Output Logic
process(control_reg)
begin
        Start <= '0';
        rShift <= '0';
        CE <= '0';

        case control_reg is
                when Load =>
                when StartTransmit =>
                        Start <= '1';
                when FinishTransmit =>
                        rShift <= '1';
                        CE <= '1';
        end case;
end process;
-------------------------------------------------------------------
--Counter
-------------------------------------------------------------------
counter: counter3bitCycle
        port map(       clk => clock_in,
                                CE => CE,
                                TC =>  TC );
-------------------------------------------------------------------
--Data line
-------------------------------------------------------------------
--DFF
process(clock_in)
begin
        if (clock_in'event and clock_in='1') then       --send data on the falling edge
                data_reg <= data_next;
        end if;
end process;

--N.S. Logic
data_next <= '0' & data_reg(7 downto 1) when rShift='1' else data_in;

--Output Logic
data1 <= data_reg(0);
```

```
        led_out <= data_reg;            --for degugging
        ----------------------------------------------------------------

end Structural;
----------------------------------------------------------------------------------
-- Company: Walla Walla University
-- Engineer: Eric Walsh & Nicholas Zimmerman
--
-- Create Date:    14:32:51 11/09/2021
-- Design Name:
-- Module Name:    Rx - Structural
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Rx is
        port(   data1: in std_logic;
                        Start: in std_logic;
                        clock_in: in std_logic;

                        led_out: out std_logic_vector(7 downto 0) );
end Rx;

architecture Structural of Rx is

        signal rShift, CE, TC : std_logic;

        --Control FSM
        type state_type is (Store, Shift);
```

```
	signal control_next, control_reg : state_type;

	--Data line
	signal data_next, data_reg: std_logic_vector(7 downto 0);

	--Component declarations
		--counter
		component counter3bitCycle is
			port(	clk : in std_logic;
					CE : in std_logic;
					TC : out std_logic );
		end component;
begin
	-----------------------------------------------------------------
	--Control FSM
	-----------------------------------------------------------------
	process(clock_in)
	begin
		if(clock_in'event and clock_in='1') then
			control_reg <= control_next;
		end if;
	end process;

	--NS Logic
	process(control_reg, Start, TC)
	begin
		case control_reg is
			when Store =>
				if (Start='1') then
					control_next <= Shift;
				else
					control_next <= Store;
				end if;
			when Shift =>
				if (TC='1') then
					control_next <= Store;
				else
					control_next <= Shift;
				end if;
		end case;
	end process;

	--Output Logic
	process(control_reg)
	begin
		rShift <= '0';
```

```
		CE <= '0';

		case control_reg is
			when Store =>
			when Shift =>
				rShift <= '1';
				CE <= '1';
		end case;
	end process;
	-----------------------------------------------------------------
	--Counter
	-----------------------------------------------------------------
	counter: counter3bitCycle
		port map(	clk => clock_in,
						CE => CE,
						TC =>  TC );
	-----------------------------------------------------------------
	--Data line
	-----------------------------------------------------------------
	--DFF
	process(clock_in)
	begin
		if (clock_in'event and clock_in='1') then
			data_reg <= data_next;
		end if;
	end process;

	--N.S. Logic
	data_next <= data1 & data_reg(7 downto 1) when rShift='1' else data_reg;

	--Output Logic
	led_out <= data_reg;		--LED values change while shifting
	-----------------------------------------------------------------

end Structural;
```