情報科学実験Ａ　レポート４

氏名　山久保孝亮
所属　大阪大学基礎工学部情報科学科ソフトウェア科学コース
学籍番号　09B22084
提出日　2024 年 1 月 25 日
担当教員　繁田　浩功/桝井　晃基

1　実験ペアの名前および学籍番号

学籍番号	名前
09B22083	安川雄輝

表 1: ペアの学籍番号と名前

2　送受信回路のステート・マシンの概要図

送受信回路のステート・マシンの概要図は以下の図 1, 図 2 のようになった.

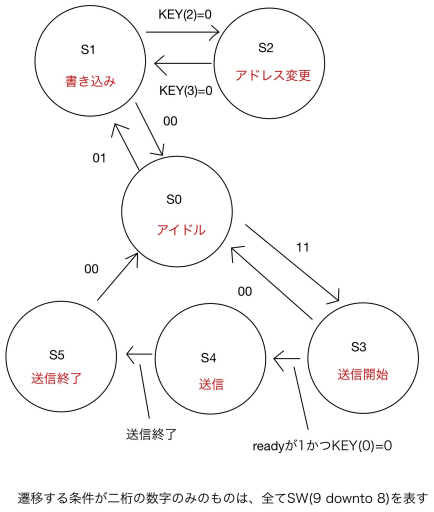

図 1: 送信回路のステート図

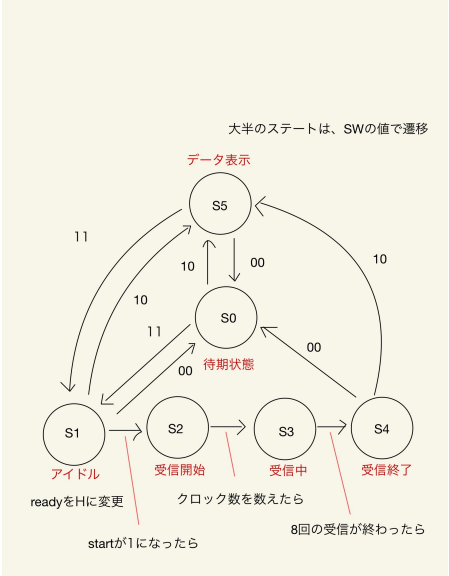

図 2: 受信回路のステート図

3　送受信回路の設計内容

3.1　送信回路の設計

送信回路の設計にあたって使用した変数の名前とその役割の対応は以下の表 2 のとおりである.

以下では図 1 のそれぞれのステートにおける処理を記述する. また,Listing1 は 3.1 の末尾に添付したのは送信回路の

3.1.1　アイドル状態

s0 であるアイドル状態について記述する.32 ビットのベクトルである SW の 8 から 9 ビット目を参照しその値によって次の遷移先を決定する.01 なら s1 に遷移し,11 なら s3 に遷移する. それ以外であれば s0 にとどまり続ける. また,s1 に遷移する際には wadr を 000 に,key2_frag を 0 に,write_in を 0000 に初期化する. これらの変数の具体的な働きは後述する. また, このときに KEY(0) が 0 であるかどうかを判定し 0 であれば key2_frag を 1 としているがこれはテストベンチの KEY の切り替わりをステートの遷移後に計測しているとデータを書き込むのに間に合わなかったので追加した.

3.1.2 書き込み状態

s1 である書き込み状態について記述する.KEY(0) が 0 になっている, 即ち KEY(0) が押されると key2_frag という変数が 1 となる. この key2_frag という変数は KEY(2) が押されたことを記憶しておく変数である. そしてこの key2_frag が 1 であれば key2_frag,KEY(3) が押されたかどうかを判定する key3_frag を 0 にして write_in に現在の din の値を格納する.din は ram1 の入力であり一度 write_in に退避させておくことで次のアドレス変更状態にいる際にデータが変更されても書き込むデータ自体は KEY(2) を押したタイミングでのデータであり続けるからである. そしてその後 s2 に遷移する. また,s0 の時と同様に SW の 8 から 9 ビットが 00 であれば s0 に遷移し, 以上どの分岐にも当てはまらない場合には s1 にとどまり続ける.

3.1.3 アドレス変更状態

s2 であるアドレス変更状態について記述する.KEY(2) や KEY(3) が押されるとそれぞれの frag の変数が 1 となる. key3_frag が 1 であるかどうかを判定し,1 であれば key3_frag を 0 にし wadr を 1 だけ増やす. これにより加算される前の wadr の値に対応するデータの値が確定する. そして分岐に当てはまらなければ s2 にとどまり続ける.

3.1.4 送信開始状態

s3 である送信開始について記述する.SW の 8 から 9 ビットが 00 であれば s0 に遷移する. 次に受信側が送ってくる ready が 1 であるかどうかを判定する. ready が 1 でなければ s3 にとどまり続け以降の判定は行わない.ready が 1 のときは KEY(0) が 0 かどうかを判定する.0 であれば radr を 000 に,count を 1 に初期化して s4 に遷移する.

3.1.5 送信状態

s4 である送信状態について記述する.

3.1.6 送信終了状態

s5 である送信終了状態について記述する.SW の 8 から 9 ビットが 00 であれば radr,clk_tx_last,count,count_send,count_clk を初期化して s0 に遷移する. それ以外の時は s5 にとどまり続ける.

Listing 1: 送信回路のコード

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use IEEE.NUMERIC_STD.all;
5
6  entity tx is
7    generic(N: integer := 32;
8            K: integer := 4;
9            W: integer := 3);
10   port(
11     CLOCK_50, RESET_N: in std_logic;
12     -- KEY(0): start button
13     -- KEY(2): write enable
14     -- KEY(3): address increment
```

```
15      KEY: in std_logic_vector(3 downto 0);
16      -- SW(9 downto 8): mode
17      -- SW(3 downto 0): din
18      SW: in std_logic_vector(9 downto 0);
19      -- GPIO_1(3 downto 0): data bits
20      -- GPIO_1(4): start bit
21      -- GPIO_1(5): ready bit
22      GPIO_1: inout std_logic_vector (35 downto 0);
23      -- LEDR: for debug
24      LEDR: out std_logic_vector (9 downto 0);
25      HEX0, HEX1, HEX2, HEX3, HEX4, HEX5: out std_logic_vector(6 downto 0));
26  end tx;
27
28  architecture rtl of tx is
29    constant cnt_max: std_logic_vector(31 downto 0):= X"000003FF";
30    type state_type is (s0, s1, s2, s3 , s4 , s5);
31    signal state: state_type;
32    signal sout: std_logic_vector (3 downto 0);
33    signal clk, xrst: std_logic;
34    signal enable: std_logic;
35    signal clk_tx: std_logic;
36    signal start: std_logic;
37    signal ready: std_logic;
38    signal we: std_logic;
39    signal wee: std_logic;
40    signal wadr, radr, adr: std_logic_vector (2 downto 0);
41    signal din, dout: std_logic_vector (3 downto 0);
42    signal debug0, debug1, debug2: std_logic_vector (3 downto 0);
43    signal enter_1, enter_3, enter_5: std_logic_vector (3 downto 0);
44    signal count_send, count_read, count_write, count_clk, count : integer := 0;
45    signal clk_tx_last, finish,key2_frag, key3_frag: std_logic;
46    signal write_in, first: std_logic_vector (3 downto 0);
47
48    component clock_gen
49      generic(N: integer);
50      port(clk, xrst: in std_logic;
51           enable: in std_logic;
52           cnt_max: in std_logic_vector (N-1 downto 0);
53           clk_tx: out std_logic);
54    end component;
55
56    component ram_WxK
57      generic(K: integer;
58              W: integer);
59      port(clk: in std_logic;
60           din: in std_logic_vector (K-1 downto 0);
61           wadr: in std_logic_vector (W-1 downto 0);
62           radr: in std_logic_vector (W-1 downto 0);
63           we: in std_logic;
64           dout: out std_logic_vector (K-1 downto 0));
65    end component;
66
```

```
67   component seven_seg_decoder is
68     port(clk: in std_logic;
69          xrst: in std_logic;
70          din: in std_logic_vector(3 downto 0);
71          dout: out std_logic_vector(6 downto 0));
72   end component;
73
74 begin
75   clk <= CLOCK_50;
76   xrst <= RESET_N;
77   din <= SW(3 downto 0);
78   we <= not KEY(2);
79   clk_tx_last <= '0';
80   ready <= GPIO_1(5);
81
82   cg1: clock_gen generic map(N => N) port map(clk => clk, xrst => xrst, enable =>
         enable, cnt_max => cnt_max, clk_tx => clk_tx);
83   ram1: ram_WxK generic map(K => K, W => W) port map(clk => clk, din => write_in, wadr
          => wadr, radr => radr, we => key3_frag, dout => dout);
84   ssd0: seven_seg_decoder port map(clk => CLOCK_50, xrst => RESET_N, din => "0000",
         dout => HEX0);
85   ssd1: seven_seg_decoder port map(clk => CLOCK_50, xrst => RESET_N, din => enter_1,
         dout => HEX1);
86   ssd2: seven_seg_decoder port map(clk => CLOCK_50, xrst => RESET_N, din => "0000",
         dout => HEX2);
87   ssd3: seven_seg_decoder port map(clk => CLOCK_50, xrst => RESET_N, din => enter_3,
         dout => HEX3);
88   ssd4: seven_seg_decoder port map(clk => CLOCK_50, xrst => RESET_N, din => "0000",
         dout => HEX4);
89   ssd5: seven_seg_decoder port map(clk => CLOCK_50, xrst => RESET_N, din => enter_5,
         dout => HEX5);
90
91 process(xrst,clk)
92 begin
93   if(xrst = '0')then
94     state <= s0;
95     enable <= '0';
96     wadr <= "000";
97     radr <= "000";
98     finish <= '0';
99     start <= '0';
100
101  elsif(clk'event and clk = '1')then
102    case state is
103      when s0 =>
104        enable <= '0';
105            finish <= '0';
106
107        if(sw(9 downto 8) = "01")then
108            wadr <= "000";
109            key2_frag <= '0';
110            write_in <= "0000";
```

```
111             if(KEY(2) = '0')then
112                 key2_frag <= '1';
113                 end if;
114             state <= s1;
115           elsif(sw(9 downto 8) = "11")then
116             state <= s3;
117           else
118             state <= s0;
119           end if;
120
121
122         when s1 =>
123           if(KEY(2) = '0')then
124             key2_frag <= '1';
125           end if;
126           if(key2_frag = '1')then
127             key3_frag <= '0';
128             key2_frag <= '0';
129             write_in <= din;
130            if(KEY(3) = '0')then
131             key3_frag <= '1';
132           end if;
133                 state <= s2;
134             elsif(sw(9 downto 8) = "00")then
135           state <= s0;
136             else
137                 state <= s1;
138             end if;
139
140         when s2 =>
141           if(KEY(3) = '0')then
142             key3_frag <= '1';
143           end if;
144           if(KEY(2) = '0')then
145             key2_frag <= '1';
146           end if;
147           if(key3_frag = '1')then
148             key3_frag <= '0';
149             wadr <= wadr + 1;
150             state <= s1;
151           else
152             state <= s2;
153           end if;
154
155         when s3 =>
156               if(sw(9 downto 8) = "00")then
157                 state <= s0;
158               elsif(ready = '1')then
159               if(KEY(0) = '0')then
160                 GPIO_1(4) <= '1';
161                 radr <= "000";
162                 count <= 1;
```

```
163                 state <= s4;
164             end if;
165         else
166               state <= s3;
167                 end if;
168 
169 
170       when s4 =>
171         if(finish = '0')then
172         enable <= '1';
173         if(clk_tx = '1')then
174             if(clk_tx_last = '0')then
175                 clk_tx_last <= '1';
176                 start <= '0';
177                 finish <= '1';
178                 enable <= '0';
179             end if;
180         else
181             start <= '1';
182             clk_tx_last <= '0';
183             count_clk <= count_clk + 1;
184         end if;
185         elsif(count_send = 8)then
186             count_send <= 0;
187             state <= s5;
188         else
189             count <= count + 1;
190             if(count = count_clk -1)then
191                 radr <= radr + 1;
192                 enter_1 <= dout;
193                 enter_3 <= "0" & radr;
194                 enter_5 <= "000" & GPIO_1(4);
195                 count_send <= count_send + 1;
196                 count <= 0;
197             end if;
198             state <= s4;
199         end if;
200 
201     when s5 =>
202         if(SW(9 downto 8) = "00")then
203             adr <= "000";
204             clk_tx_last <= '0';
205             count <= 0;
206             count_send <= 0;
207             count_clk <= 0;
208             state <= s0;
209         else
210             state <= s5;
211         end if;
212       end case;
213     end if;
214 GPIO_1(4) <= start;
```

```
215 GPIO_1(0) <= enter_1(0);
216 GPIO_1(1) <= enter_1(1);
217 GPIO_1(2) <= enter_1(2);
218 GPIO_1(3) <= enter_1(3);
219 end process;
220 end rtl;
```

3.2 受信回路の設計

4 テストベンチによる動作確認内容・結果

4.1 送信回路の動作結果

4.1.1 データ書き込みの動作

以下の図の

4.1.2 データ送信の動作

4.2 受信回路の動作結果

5 非同期通信回路作成における注意点などの考察

6 教員・TA による動作確認時刻

1 月 23 日時分