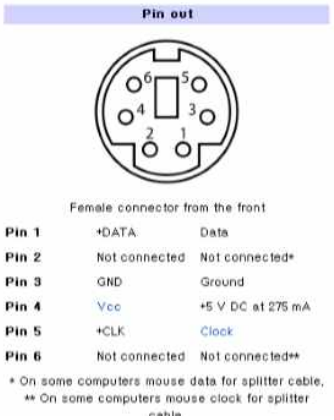
SOC실습

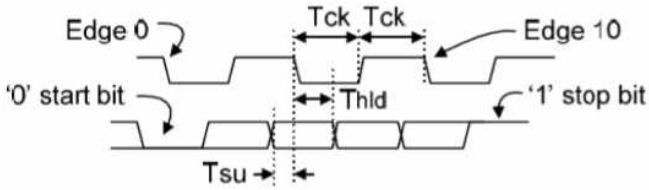
-PS2를 이용한 동기통신-

과 제 명 :PS2_동기통신
담당교수 : 최종성
학 과 :메카트로닉스공학과
학 번 : 2015132039
이 름 : 최영운
제 출 일 : 2019.10.10.

1. 주제 배경 이론

- Personal System/2 : IBM PC의 Keyboard와 마우스 입력 인터페이스
- CLK : 유효 데이터를 지시하기 위한 동기신호
  - 양방향 : 데이터 전송 측에서 CLK도 같이 전송
- DATA : SCAN Code 전송
- 실습에서는 Keyboard의 데이터 입력만 처리.





Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

PS2 선을 통한 키보드-컴퓨터 간의 통신을 해보고자 한다.
기본적으로 키보드에서 병렬로 데이터 값을 보내면 좋겠으나, 그러고자 하면 선의 개수가 증가하게 되어 비용이 비싸진다. 따라서 비용을 절감 하고자 직렬 전송을 선택하게 된다. 따라서 키보드 내에는 병렬 -> 직렬로 변환 시키는 모듈(칩)이 붙어 있으며, 컴퓨터 내에는 전송받은 직렬데이터 -> 병렬로 변환 시키는 모듈(칩)이 붙어있다.

또한 실제 키보드에 연결하여 PS2전송을 하기에는 어려움이 따르므로 키보드의 D키(23)를 눌렀다고 가정하여 설계를 진행하였다.

설계는 크게 두가지 부분으로써 키보드에서 전송하는 부분과
컴퓨터에서 받는 부분으로 나뉘어 설계한다.
동기 통신이므로 키보드에서 나오는 클럭을 컴퓨터에서 받는 부분과 동기 시켜 데이터를 검출하게 한다.

2. 소스코드 설명

1) PS2 KEYBOARD

```
1   library ieee;
2         use ieee.std_logic_1164.all;
3         use ieee.std_logic_arith.all;
4         use ieee.std_logic_unsigned.all;
5
6   entity ps2_keybord is
7        port(
8        nRst : in std_logic;
9        clk : in std_logic;
10       start_sig : in std_logic;
11       data : in std_logic_vector(7 downto 0);
12       ps2_clk : out std_logic;
13       ps2_data : out std_logic
14     );
15  end ps2_keybord;
16
17  architecture beh of ps2_keybord is
18
19    type state_type is(IDLE, START, SEND, PARITY, STOP);
20      signal state : state_type;
21      signal cnt : std_logic_vector(9 downto 0);
22      signal pclk : std_logic;
23      signal pclk_cnt : std_logic_vector(1 downto 0);
24      signal bit_cnt : std_logic_vector(2 downto 0);
25      signal temp_data : std_logic_vector(7 downto 0);
26      signal tx_data : std_logic_vector(7 downto 0);
27
28      signal start_d : std_logic;
29      signal flag : std_logic;
30
31  begin
32
33       process(nRst, clk)    --100MHZ? 80KHZ? ?? ?? ??
```

IEEE선언
라이브러리 선언

PS2_KEYbord의 입출력 포트 설정
(리셋, 클럭, 눌렸을때를 가정한 start시그널, 데이터)
(ps2클럭, ps2데이터)

내부 시그널들 모음

```
33      process(nRst, clk)    --100MHZ? 80KHZ? ?? ?? ??
34        begin
35          if(nRst = '0') then
36          cnt <= (others => '0');
37          pclk <= '0';
38        elsif rising_edge(clk) then  --625? ??? 1???.
39          if( cnt = 624) then
40          cnt <= (others => '0');
41          pclk <= not pclk;
42        else
43          cnt <= cnt + 1;
44        end if;
45      end if;
46    end process;
47
48    process(nRst,clk)
49      begin
50        if(nRst = '0') then
51        start_d <= '0';
52        flag <= '0';
53        temp_data <= (others => '0');
54      elsif rising_edge(clk) then
55        start_d <= start_sig;
56        if(start_d = '0')and(start_sig = '1') then    --
57
58        flag <= '1';
59        temp_data <= data;
60      elsif(state = START)then  --?? ??? flag  = '0'
61      flag <= '0';
62    end if;
63  end if;
64  end process;
65
```

리셋신호시 all클리어

624인 것은 주파수 분주를 통해 100MHZ인 시스템 클럭을 20KHZ로 쪼개서 사용하기 때문이다.
또한 그것을 통해 80KHZ분주도 하기 때문에 설정해 준다.

리셋동작

start_d와 start_sig가 있는 것은 --시그널을 정확히 동작 시키기 위해 1clk 딜레이를 통해 순간적인 and동작으로 검출해 낸다.
1clk이 딜레이시 기존 =1, 딜레이 신호 = 0 인부분에서 작동 시키는 원리

```
65
66      process(nRst,pclk)
67        begin
68          if(nRst = '0')then
69          state <= IDLE;  --?? ??? ???
70          pclk_cnt <= (others => '0');
71          bit_cnt <= (others => '0');
72          tx_data <= (others => '0');
73        elsif rising_edge(pclk) then
74          case state is
75          when IDLE =>
76            if (flag = '1') then
77              state <= START;
78            else
79              state <= IDLE;
80            end if;
81          pclk_cnt <= (others => '0');  --??? ?? ??? IDI
82          bit_cnt <= (others => '0');
83          tx_data <= (others => '0');
84        when START =>
85          if(pclk_cnt = 3) then
86          pclk_cnt <= (others => '0');  --4?? ? ?? ???
87          state <= SEND;
88        else
89          pclk_cnt <= pclk_cnt + 1;
90          state <= START;
91        end if;
92        tx_data <= temp_data;
93      when SEND =>
94        if(pclk_cnt = 3)then
95        pclk_cnt <= (others => '0');
96        if(bit_cnt = 7) then
97        bit_cnt <= (others => '0');
98          state <= PARITY;
99      else
100       tx_data <= '0' &tx_data(7 downto 1);  --??? ??
101       bit_cnt <= bit_cnt + 1;  --?? ?? ??? ?? 20KHZ
102       state <= SEND;
103     end if;
104     else
105       pclk_cnt <= pclk_cnt + 1;
106       state <= SEND;
107     end if;
108     when PARITY =>
109       if(pclk_cnt = 3)then
110       pclk_cnt <= (others => '0');
111       state <= STOP;
112     else
113       pclk_cnt <= pclk_cnt + 1;
114       state <= PARITY;
115     end if;
116     when STOP =>
117       if(pclk_cnt = 3) then
118       pclk_cnt <= (others => '0');
119       state <= IDLE;
120     else
121       pclk_cnt <= pclk_cnt + 1;
122       state <= STOP;
123     end if;
124     when others =>
125       state <= STOP;
126     end case;
127     end if;
```

대기(IDLE)상태에서는 스타트 Flag를 받을 때 까지 기다린다.

스타트 플레그를 입력 받으면 스타트 상태로 넘어간다.
아니라면 대기상태 지속.

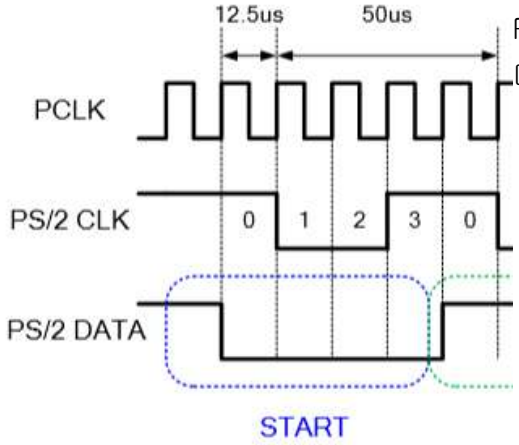
스타트 상태시 20MHZ로 분주된 클럭을 4개를 샌다. 그후 데이터 전송 모드로 넘어간다. 또한 가상공간에 저장된 데이터를 입력 받는다.

전송 상태시 한 비트씩 전송을 한다.
4클럭씩 총 8개의 비트를 전송한다.(1비트에 4클럭)

데이터 전송이 끝나면 페리티 비트를 전송시킨다.
(복잡하므로 이건 생략함)

전송이 끝났으므로 STOP상태로 넘어간 후 4클럭뒤 대기상태로 넘어간다.

```
128     end process;
129
130     ps2_clk <= '0'  when pclk_cnt >= 1 and pclk_cnt <= 2
131     else '1';  --?? ??? 0,3,?? 1 ,   1,2?? 0
132       ps2_data <= tx_data(0) when state = SEND else
133                         '0'   when state = START or state = PARITY else
134                         '1';
135   end beh;
```



PS2클럭의 1,2에서 하강상태인 것을 나타낸다.

2)PS2_RECEIVER

```
1   library ieee;
2      use ieee.std_logic_1164.all;
3      use ieee.std_logic_arith.all;
4      use ieee.std_logic_unsigned.all;
5
6    entity ps2_receiver is
7      port (
8        nRst : in std_logic;
9        clk : in std_logic;
10       ps2_clk : in std_logic;
11       ps2_data : in std_logic;
12       valid : out std_logic;
13       received_data : out std_logic_vector(7 downto 0)
14     );
15   end ps2_receiver;
16
17  architecture beh of ps2_receiver is
18
19      type state_type is(IDLE, START, RECEIVE, PARITY, STOP);
20      signal state : state_type;
21      signal ps2_clk_d : std_logic;
22      signal ps2_clk_det : std_logic;
23      signal ps2_data_d : std_logic;
24      signal ps2_data_det : std_logic;
25      signal bit_cnt : std_logic_vector(2 downto 0);
26      signal temp_data : std_logic_vector(7 downto 0);
27
28  begin
29
30       process(nRst, clk)
31         begin
32           if(nRst = '0') then
33            ps2_clk_d <= '0';
34            ps2_clk_det <= '0';
35            ps2_data_d <= '0';
36            ps2_data_det <= '0';
37          elsif rising_edge(clk) then
38          ps2_clk_d <= ps2_clk;
39          ps2_data_d <= ps2_data;
40            if( ps2_clk_d = '0')and(ps2_clk = '1') then
```

라이브러리 선언

입출력 포트 선언
(끝났다는 valid가 있음)

내부 시그널들 정리.

리셋동작과 상승엣지시 동작 정의

키보드와 같이 clk을 한클럭

```
41            ps2_clk_det <= '1';
42          else
43            ps2_clk_det <= '0';
44          end if;
45          if(ps2_data_d = '1')and(ps2_data = '0')then
46          ps2_data_det <= '1';
47        else
48          ps2_data_det <= '0';
49        end if;
50      end if;
51    end process;
52
53      process(nRst,clk)
54        begin
55          if(nRst = '0') then
56          state <= IDLE;
57          bit_cnt  <= (others => '0');
58          temp_data <= (others => '0');
59          received_data  <= (others => '0');
60          valid <= '0';
61        elsif rising_edge(clk) then
62          case state is
63        when IDLE =>
64          if (ps2_data_det = '1') then
65            state <= START;
66          else
67            state <= IDLE;
68          end if;
69        bit_cnt <= (others => '0');
70        temp_data <= (others => '0');
71        received_data  <= (others => '0');
72        valid <= '0';
73      when START =>
74        if(ps2_clk_det = '1') then
75        state <= RECEIVE;
76      else
77        state <= START;
78      end if;
79    when RECEIVE =>
80      if(ps2 clk det = '1')then
```

딜레이 시켜서 검출하게 한다.

각 상태에 따른 동작 설명
리셋시 모든 동작 0

대기상태에서 데이터 입력 감지 신호가 입력되면 start동작으로 전환.

스타트 신호시 4클럭이 지난 즉 한 비트가 지나가면 나오는 신호를 받아 받기 동작으로 넘어감

```
80      if(ps2_clk_det = '1')then
81          if(bit_cnt = 7) then
82          bit_cnt <= (others => '0');
83          state <= PARITY;
84          else
85          bit_cnt <= bit_cnt + 1;
86          state <= RECEIVE;
87    end if;
88    temp_data <= ps2_data & temp_data(7 downto 1);
89    else
90      state <= RECEIVE;
91    end if;
92    when PARITY =>
93      if(ps2_clk_det = '1')then
94      state <= STOP;
95    else
96      state <= PARITY;
97    end if;
98
99    when STOP =>
100     if(ps2_clk_det = '1') then
101     state <= IDLE;
102   else
103     state <= STOP;
104   end if;
105     received_data <= temp_data;
106     valid <= '1';
107   when others =>
108     state <= IDLE;
109   end case;
110   end if;
111   end process;
112
113   end beh;
114
```

받기 동작에서는 한비트씩 넘어오는 데이터를 총 8비트 받는다.

다음은 페리티 신호를 받고

정지 신호를 받으며 다시 대기상태로 돌아간다.

3) test_bench

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity tb_ps2_keybord is  end ;
7
8   architecture BEH of tb_ps2_keybord is
9
10     component  ps2_keybord is
11     port(
12       nRst : in std_logic;
13       clk : in std_logic;
14       start_sig : in std_logic;
15       data : in std_logic_vector(7 downto 0);
16       ps2_clk : out std_logic;
17       ps2_data : out std_logic
18     );
19     end component;
20
21     component  ps2_receiver is
22     port (
23       nRst : in std_logic;
24       clk : in std_logic;
25       ps2_clk : in std_logic;
26       ps2_data : in std_logic;
27       valid : out std_logic;
28       received_data : out std_logic_vector(7 downto 0)
29     );
30     end component;
31
32       signal signRst, sigclk, sigstart_sig : std_logic;
33       signal sigdata : std_logic_vector(7 downto 0);
34       signal sigps2_clk, sigps2_data : std_logic;
35       signal int_cnt : std_logic_vector(99 downto 0);
36       signal sigvalid : std_logic;
37       signal sigreceived_data : std_logic_vector(7 downto 0);
38
39    begin
40
```

라이브러리 선언

키보드와 receiver의 입출력 포트를 선언해 준다.

내부에서 사용될 시그널 들을 정의해 준다

```
41     process
42     begin
43       if(NOW = 0 ns) then
44        signRst <= '0', '1' after 100 us;
45      end if;
46       wait for 1 sec;
47       end process;
48
49       process
50    begin
51       sigclk <= '0', '1' after 5 ns;
52       wait for 10ns;
53       end process;
54
55       process(signRst, sigclk)
56         begin
57           if(signRst = '0') then
58           int_cnt <= (others => '0');
59         elsif(rising_edge(sigclk)) then
60           int_cnt <= int_cnt + 1;
61       end if;
62     end process;
63
64
65       sigstart_sig <= '1'when int_cnt = 200 else
66       '0';
67       sigdata <= "00100011" when ((int_cnt > 150)and(int_cnt < 250)) else
68             "00000000";
69
70   u_tb_ps2_keyboard : ps2_keybord
71   port map(
72      nRst => signRst,
73      clk => sigclk,
74      start_sig => sigstart_sig,
75      data => sigdata,
76      ps2_clk => sigps2_clk,
77      ps2_data => sigps2_data
78    );
79
80    u_tb_ps2_receiver : ps2_receiver
```

리셋을 정의해 준다 100us동안 리셋후 high 상태를 1초가량 유지해 준다
=>동작 시작 전 초기화

기준 클럭을 설정해 준다 100MHZ클럭이다.
작동시간은 10ns동안.

리셋시 0이며 클럭의 상승엣지마다 내부 시그널인 int_cnt가 상승한다

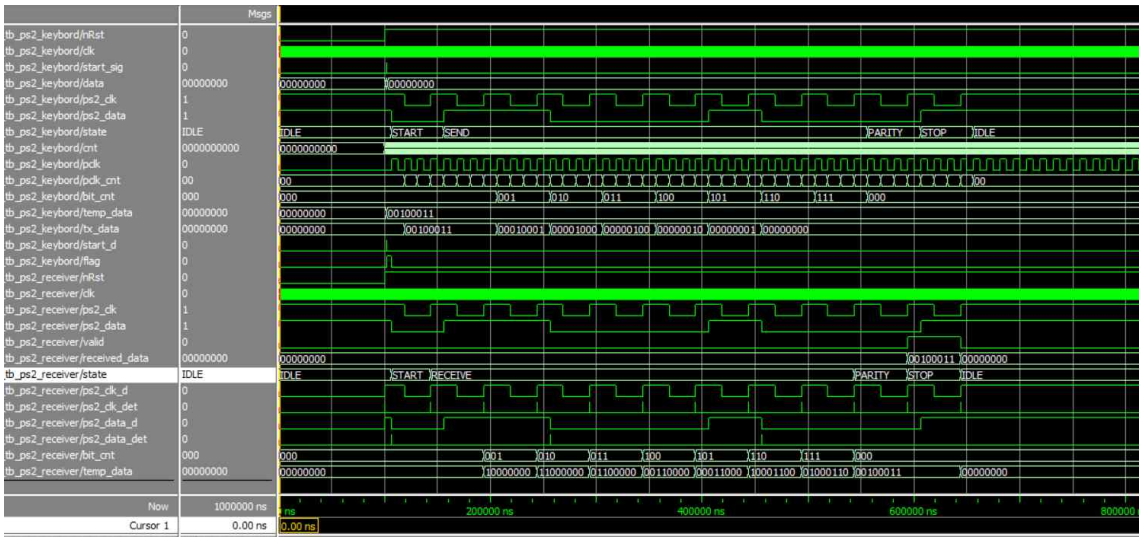
start시그널은 int_cnt가 200개일 때 작동한다.
데이터는 int_cnt가

```
81      port map(
82        nRst => signRst,
83        clk => sigclk,
84        ps2_clk => sigps2_clk,
85        ps2_data => sigps2_data,
86        valid => sigvalid,
87        received_data => sigreceived_data
88      );
89
90  end BEH;
```

150~250사이일 때 23을 보낸다.

각 포트를 키보드와 리시버에 연결한다.

3. 시뮬레이션 결과 및 설명



리셋 신호가 끝나고 start_signal이 들어오면 flag가 작동하고. 가상의 공간에 D키의 데이터인 “23”을 저장시킨다. 다음 클럭이 와서 flag를 확인하면 flag는 0으로 돌아가며 동작을 시작한다.

데이터를 보면 키보드의 데이터는 하나씩 뒤로 밀리며 사라지는 모습을 볼수 있으며 리시버의 데이터는 하나씩 뒤로 밀리며 나타나는 모습을 볼 수 있다. 따라서 데이터 전송이 되어 간다는 것을 확인 할 수 있다.

data_det와 clk_det를 통해 시작시 혹은 끝났을 시 상태를 검출하는 모습또한 볼 수 있다.

bit_cnt는 8개의 비트를 새어주는 역할을 하며 pclk_cnt는 한 비트 안의 4개의 클럭을 검출하는 것을 볼 수 있다.

전송이 완료된 후 리시버의 valid가 high가 되었다가 사라짐으로써 전송이 끝났다는 것을 알 수 있게 된다.