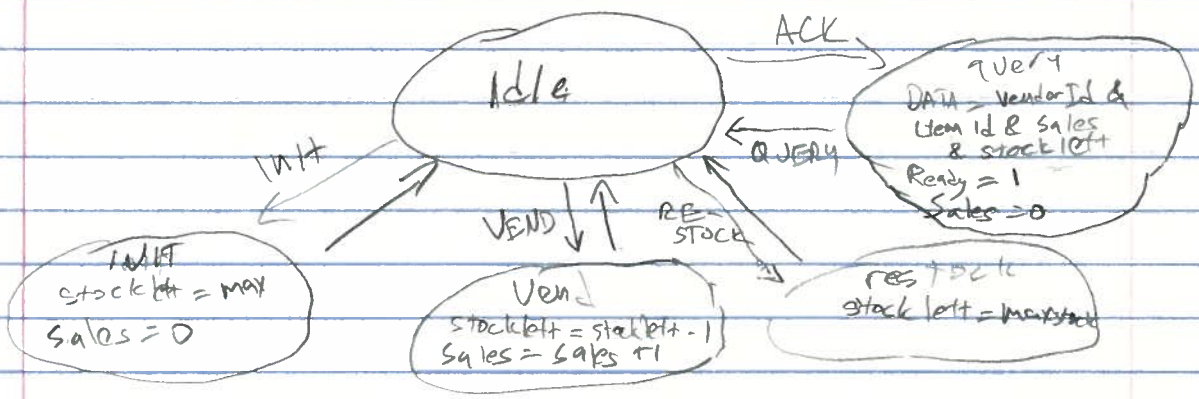
9 ~~10~~ / 10

Simulation?

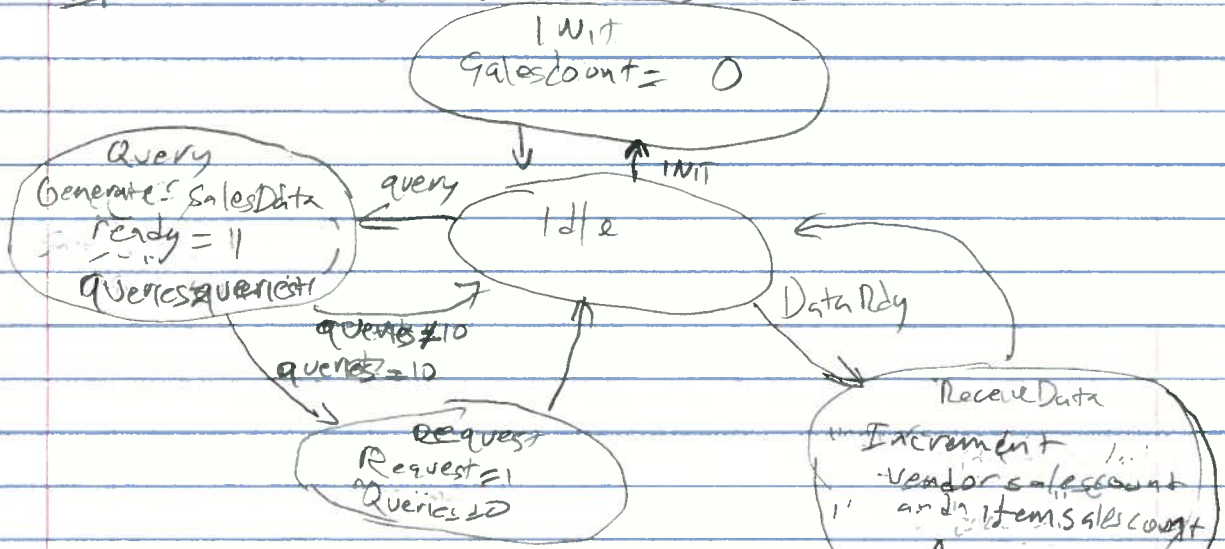
Dan Ports
6.111 PS5

I. Vendor FSM state diagram

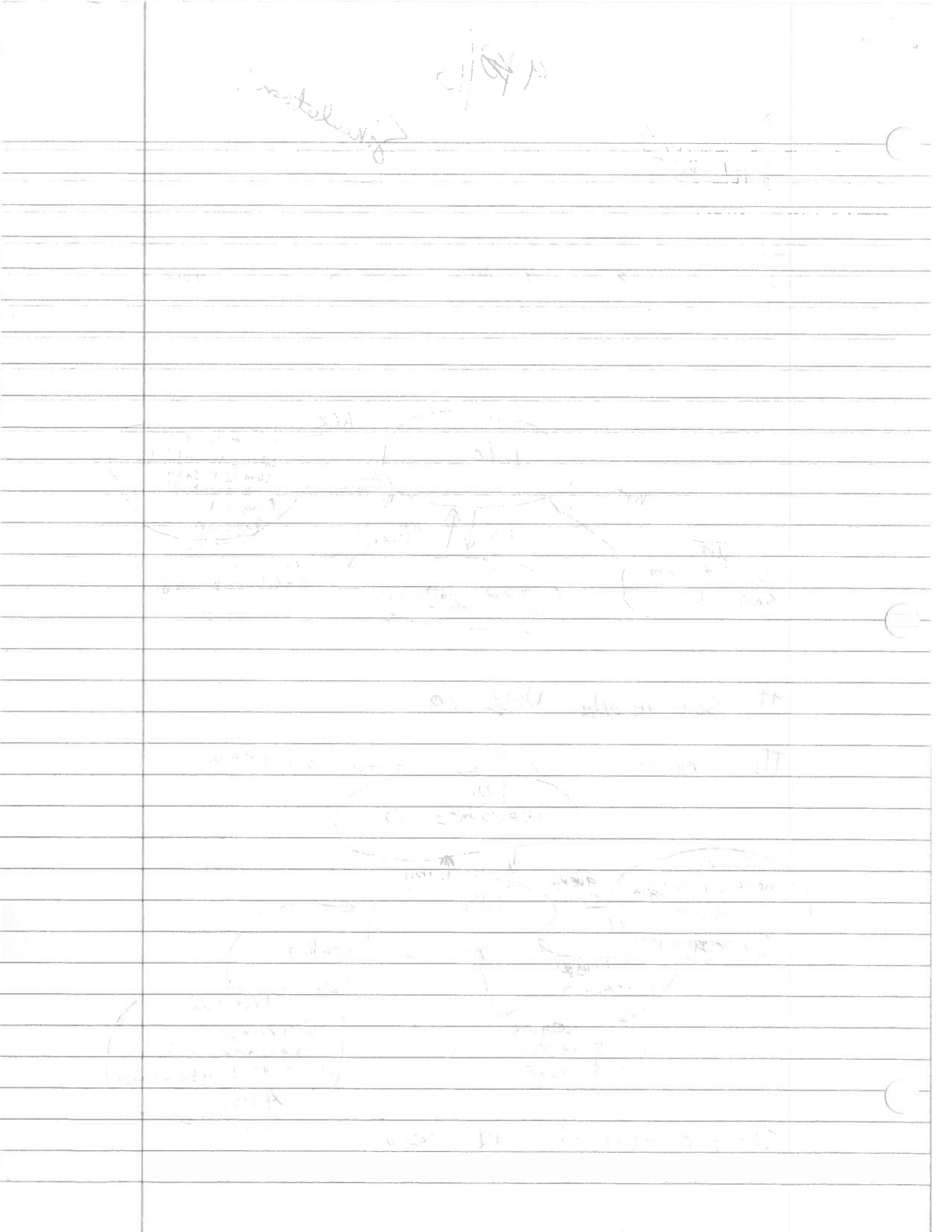


II. See attached VHDL code.

III Central controller state diagram



IV see attached VHDL code



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity vendor is

  generic (
    vendorid : integer := 0;
    itemid   : integer := 0);

  port (
    init, vend, restock, query, ack, clk : in  std_logic;
    ready                                : out std_logic;
    data                                 : out std_logic_vector(17 downto 0));

end vendor;

architecture behavioral of vendor is

  type StateType is (s_idle, s_init, s_vend, s_query, s_restock);
  signal p_s, n_s : StateType;
  signal vendoridvec : std_logic_vector(5 downto 0);
  signal itemidvec : std_logic_vector(2 downto 0);
  signal sales : std_logic_vector(4 downto 0);
  signal maxstock, stockleft : std_logic_vector(3 downto 0);

begin  -- behavioral

  -- purpose: generate next state
  -- type   : combinational
  -- inputs : p_s, init, vend, restock, query
  -- outputs: n_s
  nextstate: process (p_s, init, vend, restock, query)
  begin  -- process nextstate
    if p_s = s_idle then
      if init = '1' then
        n_s <=  s_init;
      elsif vend = '1' then
        n_s <=  s_vend;
      elsif restock = '1' then
        n_s <=  s_restock;
      elsif query = '1' then
        n_s <=  s_query;
      else
        n_s <=  s_idle;
      end if;
    elsif p_s = s_query then
      ready <= '1';
      if ack = '1' then
        n_s <=  s_idle;
      else
        n_s <=  s_query;
      end if;
    else
      n_s <=  s_idle;
    end if;
  end process nextstate;

  -- purpose: perform state transitions on clock edge and generate output
  -- type   : combinational
  -- inputs : clk
  -- outputs: p_s, n_s
  state_clocked: process (clk)
  begin  -- process state_clocked
    if rising_edge(clk) then
```

```
      p_s <= n_s;
      if n_s = s_init then
        stockleft <=  maxstock;
        sales <= (others => '0');
        vendoridvec <= conv_std_logic_vector(vendorid, 6);
        itemidvec <= conv_std_logic_vector(itemid, 3);
        case itemid is
          when 0 => maxstock <= "1100";
          when 1 => maxstock <= "1010";
          when 2 => maxstock <= "1000";
          when 3 => maxstock <= "1010";
          when 4 => maxstock <= "1100";
          when 5 => maxstock <= "1000";
          when others => null;
        end case;
      elsif n_s = s_vend then
        sales <= sales + 1;
        stockleft <= stockleft - 1;
      elsif n_s = s_restock then
        stockleft <= maxstock;
      elsif n_s = s_query then
        data <= vendoridvec & itemidvec & sales & stockleft;
        sales <= (others => '0');
      end if;
    end if;
  end process state_clocked;


end behavioral;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity centralcontrol is

  port (
    clk, init, query, datardy : in  std_logic;
    qid                       : in  std_logic_vector(7 downto 0);
    data                      : in  std_logic_vector(17 downto 0);
    ack, ready, request       : out std_logic;
    salesdata                 : out std_logic_vector(15 downto 0));

end centralcontrol;

architecture behavioral of centralcontrol is

  type salestype is array(63 downto 0) of std_logic_vector(15 downto 0);
  signal vendorsalescount, itemsalescount : salestype;

  type statetype is (s_idle, s_init, s_query, s_receivedata, s_request);
  signal p_s, n_s : statetype;

  signal queries : std_logic_vector(3 downto 0);

begin  -- behavioral

  -- purpose: generate next state
  -- type   : combinational
  -- inputs : p_s, query, datardy, init
  -- outputs: n_s
  next_state: process (p_s, query, datardy, init)
  begin  -- process next_state

    case p_s is
      when s_idle =>
        if init = '1' then
          n_s <=  s_init;
        elsif query = '1' then
          n_s <= s_query;
        elsif datardy = '1' then
          n_s <= s_receivedata;
        else
          n_s <= s_idle;
        end if;
      when s_init => n_s <= s_idle;
      when s_query => if queries = 10 then
          n_s <= s_request;
        else
          n_s <= s_idle;
        end if;
      when s_request =>  n_s <= s_idle;
      when s_receivedata => n_s <= s_idle;
      when others => n_s <= s_idle;
    end case;

  end process next_state;

  -- purpose: perform state transitions and generate outputs
  -- type   : combinational
  -- inputs : clk
  -- outputs: p_s, ready, salesdata, request, salescount, queries, ack
  state_clocked: process (clk)
  begin  -- process state_clocked
    p_s <=  n_s;
```

```
    case n_s is
      when s_idle => ready <= '0'; request <= '0'; ack <= '0';
      when s_init => ready <= '0'; request <= '0'; ack <= '0';
      when s_request => ready <= '0'; request <= '1'; ack <= '0';
                        queries <= (others => '0');
      when s_receivedata => ready <= '0'; request <= '0'; ack <= '1';
                            vendorsalescount(conv_integer(data(17 downto 11))) <= ve
ndorsalescount(conv_integer(data(17 downto 11))) + 1;
                            itemsalescount(conv_integer(data(10 downto 8))) <= items
alescount(conv_integer(data(10 downto 8))) + 1;
      when s_query => ready <=  '1'; request <= '0'; ack <= '0';
                      queries <= queries + 1;
                      if qid(7) = '1' then
                        salesdata <= itemsalescount(conv_integer(qid(5 downto 2)));
                      else
                        salesdata <= vendorsalescount(conv_integer(qid(5 downto 0)))
;
                      end if;
      when others => null;
    end case;

  end process state_clocked;


end behavioral;
```