

INTERFACE WITH THE ADC08061/08061 A.D.C.

Timing Diagrams

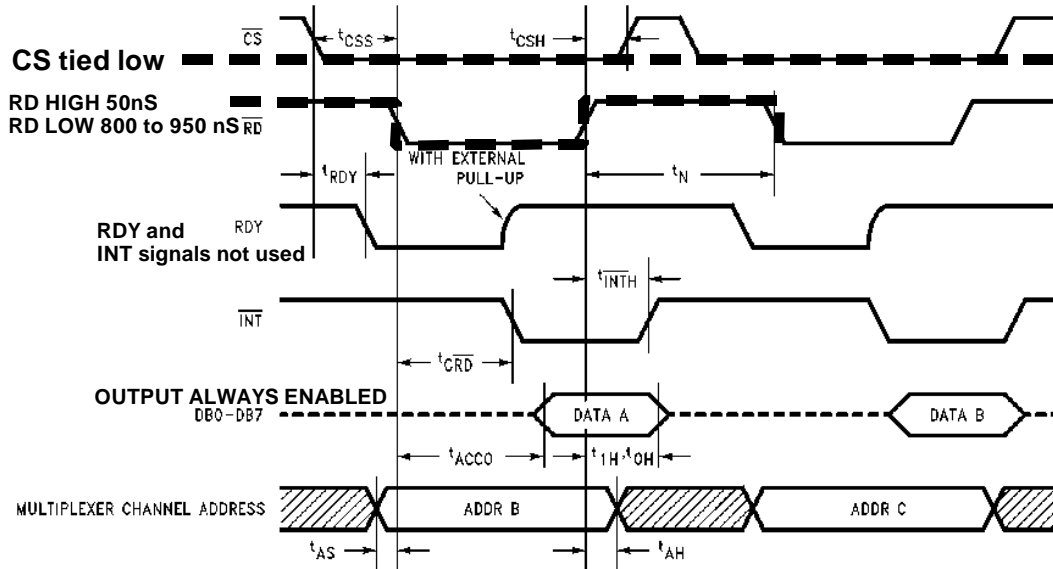


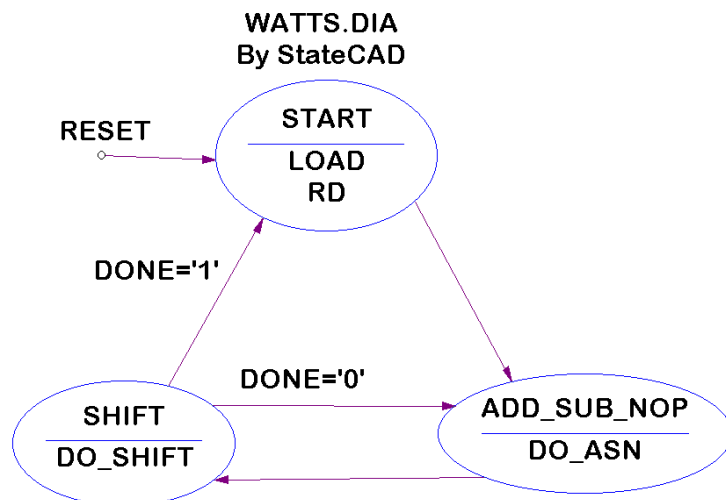
FIGURE 1. \overline{RD} Mode (Mode Pin is Low)

TL/H/1108E

We have no need to disable this device so CS will be tied low. This means that the data lines will always be enabled. We shall make \overline{RD} high for the 50ns minimum specified in the data sheets. We will assume it to be low for 800 to 950 ns. The specifications say it must be low for 640 ns typical, 900 ns worst case. We shall assume, for this example, that we will make sure that 800 ns operation is possible.

The state diagram for the control of the ADC and for doing the multiplication is below.

\overline{RD} will be high in the START state where the data from the ADC is input, the previous product is output and the registers are initialized for multiplication. Then the machine passes through the remaining two states, cycling through them 8 times to obtain the product. The entire process, then, takes 850ns. If 900 ns operation is required, an extra wait state is needed.



The VHDL code for the entire project is shown below. This code assumes to ADC08061 ADC's since they cost \$5.00 less and provide simultaneous conversions.

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
use work.watts;
entity watt is port(
    ADC_IN_1,ADC_IN_2:    in std_logic_vector(7 downto 0);
    CLK,RESET:          in std_logic;
    DAC_OUT:            out std_logic_vector(7 downto 0);
    RD:                 inout std_logic);
end;
architecture watmeter of watt is
---- DREG is used to hold the multiplicand
---- AREG is used to hold the multiplier initially, and the product
---- CREG is a 3-bit counter
    signal dreg: std_logic_vector(7 downto 0);
    signal areg: std_logic_vector(16 downto 0);
    signal creg: std_logic_vector(2 downto 0);
    signal DONE, DO_ASN, DO_SHIFT, LOAD: std_logic;
begin
cu: WATTS port map (CLK, DONE, RESET, DO_ASN, DO_SHIFT, LOAD, RD);
ck: process(clk, ADC_IN_1, ADC_IN_2, DREG, AREG) begin
    if clk'event and clk='1' then
        if RD='1' then ---- Load registers and DAC
            DAC_OUT<=AREG(15 downto 8);
            AREG(16 downto 9)<="00000000";
            AREG( 8 downto 1)<=ADC_IN_2;
            AREG( 0)<='0';
            DREG<=ADC_IN_1;
            CREG<="000";
        end if;
        if DO_ASN='1' then ---- add, subtract or do nothing.
            if AREG(1)='1' and AREG(0)='0'
                then AREG(16 downto 9)<=AREG(16 downto 9)-DREG; endif;
            if AREG(1)='0' and AREG(0)='1'
                then AREG(16 downto 9)<=AREG(16 downto 9)+DREG; endif;
            creg<=creg-1;
        end if;
        if DO_SHIFT='1' then ---- shift right
            AREG(15 downto 0)<=AREG(16 downto 1);
            areg(16)<=AREG(16);
        end if;
    end if;
    if CREG=0 then DONE<='1'; else DONE<='0'; end if;
end process;
end;
```

If we had a fast ADC, we could modify the wattmeter to work even faster. The state diagram for the faster model is shown below.

In this model, the add/sub state has been combined with the shift state. The addition or subtraction is performed and the result loaded back into the register shifted one bit to the right. Thus a total of only nine clock cycles are needed per multiplication (a load, 8 add/sub/shift states).

It may be of interest to know how fast we could really operate this wattmeter.

From the REPORT file, we obtain the following:

Worst Case Path Summary

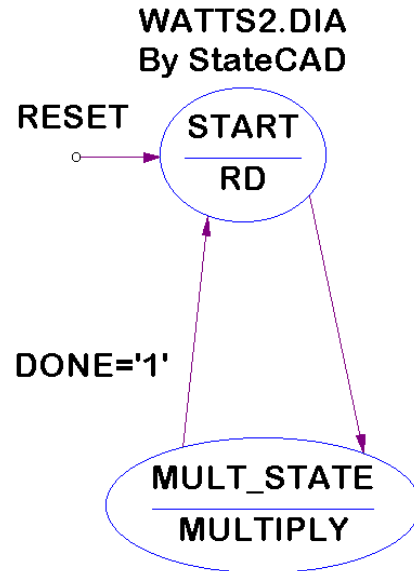
tS = 5.5 ns for areg_3.D
 tSCS = 26.0 ns for areg_15.T
 tCO = 6.5 ns for areg_10.C
 tPO = 16.0 ns for rd.AP

This says that we could clock the CPLD at 1/26 nS or almost 40 MHz. Suppose we used a 35 MHz clock, giving a period of 28.6 nS. We could do each multiply operation in 286 nS. (This includes making two RD states so that RD would be asserted for 57.2 nS, satisfying the 50 nS requirement).

This design using almost all a 372i CPLD.

CLOCK/LATCH ENABLE signals	1	2
Input REG/LATCH signals	0	36
Input PIN signals	4	4
Input PINs using I/O cells	4	4
Output PIN signals	28	28
Total PIN signals	37	37
Macrocells Used	57	64
Unique Product Terms	229	320

The VHDL file for this is shown on the next page.



```

library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
use work.watts2;
entity wattfast is port(
    ADC_IN_1,ADC_IN_2:    in std_logic_vector(7 downto 0);
    CLK,RESET:           in std_logic;
    DAC_OUT:             out std_logic_vector(7 downto 0);
    RD:                  inout std_logic);
end;
architecture watmeter of wattfast is
    signal dreg: std_logic_vector(7 downto 0);
    signal sum  std_logic_vector(16 downto 0);
    signal areg: std_logic_vector(16 downto 0);
    signal creg: std_logic_vector(2 downto 0);
    signal DONE, MULTIPLY, LOAD: std_logic;
begin
cu: WATTS2 port map (CLK, DONE, RESET, MULTIPLY, RD);
ck: process(clk, ADC_IN_1, ADC_IN_2, DREG, AREG, CREG) begin
    if clk'event and clk='1' then
        if RD='1' then
            DAC_OUT<=AREG(15 downto 8);
            AREG(16 downto 9)<="00000000";
            AREG( 8 downto 1)<=ADC_IN_2;
            AREG( 0)<='0';
            DREG<=ADC_IN_1;
            CREG<="111";
            end if;
        if MULTIPLY='1' then
            areg(16)<=sum(16);
            areg(15 downto 0)<=sum(16 downto 1);
            creg<=creg-1;
            end if;
        end if;
        if CREG=0 then DONE<='1'; else DONE<='0'; end if;
        sum<=areg(16 downto 0);
        if AREG(1)='1' and AREG(0)='0'
            then sum(16 downto 9)<=AREG(16 downto 9)-DREG(7 downto 0);
            end if;
        if AREG(1)='0' and AREG(0)='1'
            then sum(16 downto 9)<=AREG(16 downto 9)+DREG(7 downto 0);
            end if;
        end process;
end;

```

A simulation for this, using the same input as a previous simulation, is shown below.

