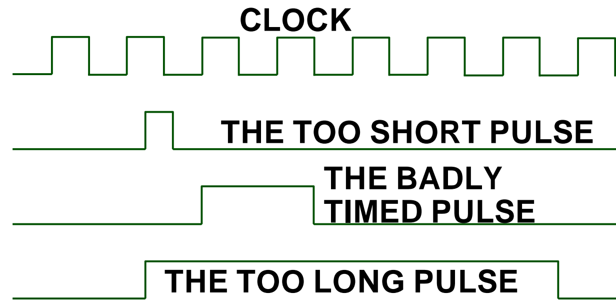
ASYNCHRONOUS INPUTS
TO
SYNCHRONOUS MACHINES

Ideally, all inputs to synchronous machines should be synchronized with the system clock so that setup and hold times are not violated. In practice, there are usually one or more asynchronous inputs that are not synchronized.

There are four problems with such signals. Three of these are illustrated to the right. The first is the input that is "too short" - it is so short that it may occur between clock events and is not detected. The second case is of a pulse that is sufficiently wide by does not meet the setup or hold time for the machine - the "badly timed" pulse. The last example is the pulse that is "too long". How a pulse can be too long will be discussed when we get to it. The fourth problem has to do with metastability and will be described as we go along.

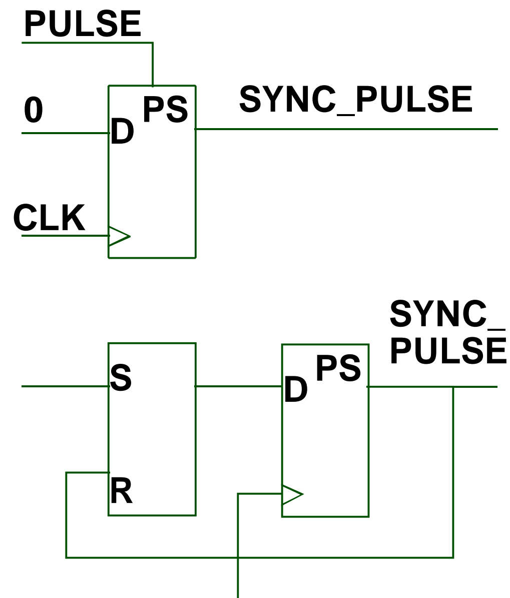


Most of these problems may be solved with analog or hybrid circuitry (such as "one shots") but it is not generally cost effective to do so today (in otherwise pure digital systems). They may be solved with discrete flip-flops, but this also is not usually a practical approach. We shall describe the solution in terms of discrete flip-flops (for better understanding) and then give the VLSI solutions (in VHDL).

Two solutions to the short pulse problem are shown to the right. They are, essentially, the same solution - just implemented different.

In the top circuit, the short pulse sets the clocked D flip-flop asynchronously. The synchronized pulse is then available at the next clock event which also clears the flip-flop. With discrete devices, this takes the fewest devices. This type of solution may be applicable in FPGA's which have flip-flops with individual presets and clears. It is not generally feasible in CPLD's which do not.

In the second case, the pulse sets an (asynchronous) SR latch. This is then clocked into the D flip-flop at the next clock event. Setting the D flip-flop clears the SR latch. This is more suitable for CPLD's.



While the above solutions solve the “shortness” problem, they still do not address the “badly timed” problem. To seek a solution, we first ask, what happens if an input to a D flip-flop if, for example, it changes during the setup or hold time? It may be set to 1. It my be set to 0. Either of these would be OK since the level of the input is undefined if it is changing. The problem, however, is two-fold.

First consider a state machine at state “000.” If X= '1', it is to go to state “011.” If X= '0' it is to go to state “110.” Now, if x is poorly timed, each flip-flop may be set to either 0 or 1. The result is that the next state may be 000, 001, 010, 011, 100, 101, 110, or 111 depending on what each of the three flip-flops do. The solution to this problem is to make sure than only one flip-flop is controlled by an asynchronous input. The solutions to the “too short” pulse automatically solve this problem. For the badly timed pulse which is not too short, all that is needed is a D flip-flop with the pulse connect to D. Either the flip-flop will catch the change or it will not and, in any case, will be valid. This, of course, adds an addition flip-flop (macrocell) and can be eliminated **if** state assignments can be chosen so that only one state variable depends on this ill-timed input.

VHDL solutions corresponding to the above are listed below.

```
library ieee;
use ieee.std_logic_1164.all;
entity too_short is port
    (p,y,clk : in bit;x: inout bit; z: out bit);
    end check;
-- p is a too short pulse, y a synchronous input.
architecture behavioral of check too_short is
    signal x: bit;
    begin
     process (p,clk) begin
        if p='1' then x<='1';
        elsif
        clk'event and clk='1' then
          x1<=p;
           p<=0;
          end if;
       end process;
      z<=x and y;
   end behavioral
```

The above solution corresponds to the first diagram above. It can not generally be used in CPLDs since they do not have flip-flops with individual presets or clears. It might be suitable for an FPGA. The solution below is for the second circuit and is suitable for CPLD’s (but not necessarily for FPGA’s).

```
library ieee;
use ieee.std_logic_1164.all;
entity too_short is port
    (p,y,clk : in bit;x: inout bit; z: out bit);
    end check;
-- p is a too short pulse, y a synchronous input.
architecture behavioral of check too_short is
    signal x: bit;
    begin
     x<=p or (x and not x1);
     process (p,clk) begin
        if clk'event and clk='1' then
          x1<=p;
          end if;
       end process;
      z<=x and y;
   end behavioral
```

One problem not yet described is the metastability problem (Sandige, pages 497-499). If the setup or hold time is not met, the flip-flop may actually reach one of **three** states: 0, 1, or metastable (in between 0 and 1). It will, eventually go to 0 or 1 but may remain in the metastable state for a number of nS. There are a number of solutions to this, but the one below is the most straight forward and most suitable for PLD's.

The diagram to the right is for one of the dedicated clock/input pins on the FLASH370 series CPLD.

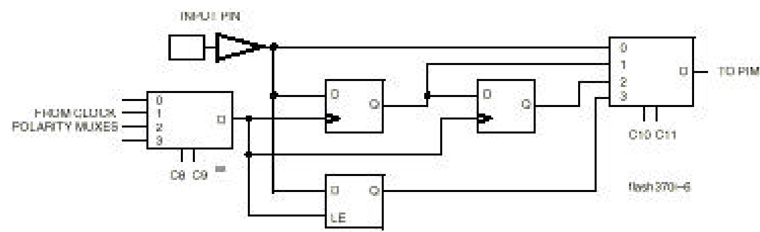


Figure 5. Input Pins

The input pin may be connected:

- directly to the inside of the CPLD (combinational input)
- to a D flip-flop (to act as a synchronizer as previously discussed)
- to a gated D latch.
- to two cascaded D flip-flops. The purpose of this last configuration is to eliminate metastability problems. The first D flip-flop may go into a metastable state for a time but will settle before the next clock pulse. Hence the output of the second flip-flop will be free of this type of problem

Note all CPLD's posses this feature (there are some such as the MACH215 which provides a single input synchronizer for all pins and there are many that provide none at all - output macrocells must be used.

Now, when such a feature is needed, how do we call it out in VHDL? Since this double registering feature is a device specific feature, VHDL provides no way of explicitly indicating that a signal should be treated in this manner. So what is the solution?

There would be no use in having a feature that could not be accessed. Cypress is the developer of both the CPLD and WARP2, so there **must** be some implicit means of specifying this.

With this thought in mind, it tmight be reasoned that the following example must be the way to do it.

```
library ieee;
use ieee.std_logic_1164.all;
entity check is port
    (x,y,clk : in bit; z: out bit);
    end check;
architecture doit of check is
    signal x1,x2: bit;
    begin
     process(clk) begin
      if clk'event and clk='1' then
          x2<=x1;
          x1<=x;
          z<=x2 and y;
          end if;
      end process;
   end doit;
```

By declaring x1 and x2 as internal nodes (i.e. they are not on the port list), the x2< = x1 and the x1< = x statement describe the double buffering of signal x. But if you programmed this into a 22V10 with no such input capability, x1 and x2 would be outputs on the PAL. The fitter for the FLASH370 series is smart enough to realize that this describing a double-register input. When it creates the JEDEC file and the REPORT file, signal x is double-registers as indicated by the following excepts from the REPORT file.

```
OPTIMIZATION OPTIONS       (13:13:43)

Messages:
  . . .
  Information: Signal x is converted to a Input Register.
  Information: Generating both D & T register equations for
     signal x2.D
  . . .

PARTITION LOGIC            (13:13:51)

Messages:
  . . .
  Information: Signal x2 forms Double Input Register at pin 35.
  . . .
```

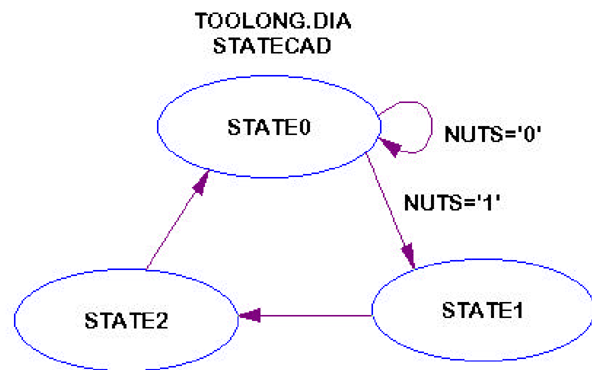
```
DESIGN EQUATIONS            (13:13:49)
  . . .
 x2.D  =  x.QI
 x2.C  =  clk
 x.CI  =  clk
 . . .
```

So, sure enough, the fitter program is smart enough to interpret the VHDL as intended.

THE TOO LONG PULSE

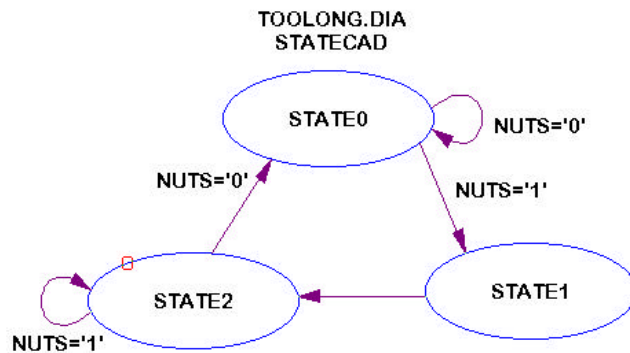
This situation frequently arises when signals come from push buttons. It also arises in other cases. Generally, the best solution is in the design of the state diagram. Consider the simple example below.

The machine is to wait in state 0 until input NUTS is asserted. It is then, presumably supposed to sequence through states 1 and 2 and then return to state 0 and wait for NUTS to be asserted <u>again</u>. But, if the NUTS input last for more than 3 clock cycles, the machine will continue to cycle.



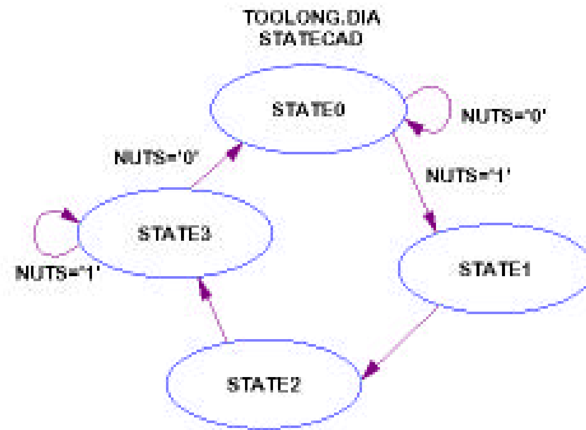
How do we make it so that it cycles only once for each assertion of NUTS? The best solution is the simple interlock shown below.

The unconditional transition from state 2 back to state 0 has been made to depend on NUTS= '0'. If NUTS is still asserted, the machine waits in state 2 until it is deasserted.



This is a simple fix to the problem but not always applicable. It may not be desirable for the machine to “hang” in state 2 waiting for NUTS to go to 0. In this case, the following solution is used.

An additional state is added (State 3). It is presumed that the output(s) in state 3 would be identical to those of state 0. In other words, states 0 and three are identical so far as the external operation of the machine is concerned.



What does this cost? It depends on the state assignment technique. If, as in this example, a binary encoded state assignment is used, going from 3 to 4 states costs nothing. Going from 4 to 5 states would cost an additional state variable (flip-flop/macrocell). If "one hot" encoding is used, an additional macrocell will be needed. In either case, allother solutions to the "too long input" are going to involve an additional macrocell anyway. This then is the most direct and safest solution. It is often used when two separate systems must communicate with each other (i.e., a computer and a modem). The process is there usually referred to as "handshaking"