```
library ieee;
use ieee.std logic 1164.all;
use ieee.std logic arith.all;
use ieee.std_logic_unsigned.all;

entity william is
---------------------------------------------------------------------------------------------------------------
port(KENDATAV,RD,WR,TAMPER,PANIC,DAJAR:in std logic;LCDCOM: inout std logic_vector (1 downto 0);ADDIN: inou
LCDENAB,KEYINT,KEYOUTEN,RAMOE,ADCEN,RAMWEN,TALRM,PALRM,DALRM,ALARM,ADCSTRT: inout std logic;CLK0:in std log
--IPT----------------------------------------------------------------------------------------------------------
attribute LOC:string;
attribute LOC of CLK0: signal is "P11";
attribute LOC of TAMPER:signal is "P43";----IPT from switch
attribute LOC of PANIC: signal is "P42";----IPT from switch
attribute LOC of DAJAR: signal is "P41";----IPT from switch
attribute LOC of ADDIN: signal is "P27 P28 P29 P30 P31 P36 P37 P38"; ----IPT from Address bus
attribute LOC of WR: signal is "P39";--IPT from /WR from MCU
attribute LOC of RD: signal is "P40";--IPT from /RD from MCU
attribute LOC of KENDATAV: signal is "P3";--IPT from Keypad Encoder
--OPT----------------------------------------------------------------------------------------------------------
attribute LOC of LCDCOM: signal is "P16 P15"; --OPT to LCD command pins RW and RS
attribute LOC of LCDENAB: signal is "P17"; --OPT to LCD Enable pin
attribute LOC of KEYINT: signal is "P2"; --OPT to MCU /INT0
attribute LOC of KEYOUTEN: signal is "P4";--OPT to Keypad Encoder
attribute LOC of RAMOE: signal is "P5";--OPT to RAM READ ENABLE
attribute LOC of RAMWEN: signal is "P6";--OPT to RAM WRITE ENABLE
attribute LOC of ADCEN: signal is "P7";--OPT to ADC out put enable
attribute LOC of TALRM: signal is "P19";--OPT code bit to MCU
attribute LOC of DALRM: signal is "P20";--OPT code bit to MCU
attribute LOC of PALRM: signal is "P21";--OPT code bit to MCU
attribute LOC of ALARM: signal is "P18";--OPT to Alarm Interrupt on MCU
attribute LOC of ADCSTRT: signal is "P24";--OPT ADC Start pin from MCU
end;
---------------------------------------------------------------------------------------------------------------
architecture behavioral of william is



--SIGNAL-------------------------------------------------------------------------------------------------------
signal tamperq:bit;
--signal tamperqnot:bit;
signal panicq: bit;
--signal panicqnot: bit;
signal dajarq: bit;
--signal dajarqnot: bit;
signal avaqnot: bit;
signal avaq: bit;
--signal swts: in std logic vector (0 to 3);
---------------------------------------------------------------------------------------------------------------
```

```
--The following are the VHDL D Flip Flops used to debounce
--the alarm activation switches

--This is the Tamper switch debounce. It uses the rising edge of
--CLK0 to determine if the q output can change or not.
--The qnot output is not used and is commented out in the signal section
begin
  process
  begin
        wait until rising edge(CLK0);--use tamperqnot
                if TAMPER='1' then
                        tamperq<='1';
                else
                        tamperq<='0';
                end if;


end process;
--Next is the panic switch debounce. It works in much the same way as the
--tamper debounce. Again, qnot is not used and is disregarded.
process
  begin
        wait until rising edge(CLK0);--use panicq
                if PANIC='1' then
                        panicq<='1';
                else
                        panicq<='0';
                end if;


end process;
--The Door ajar switch is exactly like the other two switches.
process
  begin
        wait until rising edge(CLK0);--use dajarqnot
                if DAJAR='1' then
                        dajarq<='1';
                else
                        dajarq<='0';
                end if;



end process;
-----------------------------------------------------------------------------------------------------------
--Alarm Decode to MCU
-----------------------------------------------------------------------------
```

```
--This section decodes the alarm switches to give a code word to the 8051.
--It uses the q outputs of the D Flip-Flop to control an output pin to the
--8051. The 8051 will check its pins to determine the alarm that was pushed.
--This is done for every alarm switch.
process(panicq,dajarq,tamperq)
begin
if tamperq='0' then TALRM<='1';
else TALRM<='0';
end if;

if panicq='1' then PALRM<='1';
else PALRM<='0';
end if;

if dajarq='0' then DALRM<='1';
else DALRM<='0';
end if;

if TALRM='1' or PALRM='1' or DALRM='1' then ALARM<='0';
else ALARM<='1';
end if;
end process;
-----------------------------------------------------------------------------
--LCD Address Decode

--This is the most complicated of the Address decodes. The LCD needs different
--commands to perform its functions. This is accomplished decoding the input address
--from the 8051. The commands that the LCD performs is determined by the input address.
--There are three different commands in use for the LCD shown by the three different
--states that LCDCOM can obtain (01 = write, 10 = read, 00 = command).
process (ADDIN)
begin
if ADDIN= "00010000" then LCDCOM<= "01";
elsif ADDIN= "00010001" then LCDCOM<= "10";
elsif ADDIN= "00010010" then LCDCOM<= "00";
else LCDCOM<="11";
end if;
end process;

--This portion is where the LCD enable gets clocked. The enable must be clocked after
--every command sent to it. To send this pulse, the CPLD simply XOR's the 8051 WR and RD
--signals together after it determines that LCDCOM received a proper command. Thus a
--small pulse is generated.
process (LCDCOM,WR,RD)
begin
if LCDCOM="01" then LCDENAB<=(WR xor RD);
elsif LCDCOM="10" then LCDENAB<=(WR xor RD);
elsif LCDCOM="00" then LCDENAB<=(WR xor RD);
else LCDENAB<='0';
```

```
end if;
end process;

--Keypad Address Decode

--This is the decode responsible for enabling the keypad encoder
--to place its information on the data bus. Note that this will only happen
--when the 8051 uses a movx command that will read ie: movx A,@DPTR
process (ADDIN,RD)
begin
if ADDIN="01000000" and RD='0' then KEYOUTEN<='0';
else KEYOUTEN<='1';
end if;
end process;

--The next process is a D Flip-Flop that helps to slow down the Key flutter coming
--from the encoder if the key is slowly let up. The qnot output from this
--Flip-flop directly drives the 8051 INT0.
process
  begin
        wait until rising edge(CLK0);
                if KENDATAV='1' then
                        avaq<='1';
                else
                        avaq<='0';
                end if;

                avaqnot <= not avaq;

end process;
process (avaqnot)
begin
if avaqnot='0' then KEYINT<='0';
else KEYINT<='1';
end if;
end process;

---ADC DECODE
--This process decodes the address from the 8051 to either start an ADC
--conversion, or to enable the output from the ADC0808CCN. The enable pulse
--is based on a simple Anding of the proper address and the RD signal from the
--8051. The start is a bit more complicated in that the pulse generated comes
--from the XOR of RD and WR (much like the LCD) due to the timing needed for
--the pulse.
process (ADDIN)
begin
if ADDIN="00110000" and RD='0' then ADCEN<='1';
else ADCEN<='0';
end if;
```

```
if ADDIN="00111000" then ADCSTRT<=(WR xor RD);
else ADCSTRT<='0';
end if;
end process;

---RAM DECODE
--This is a simple decode that enables the RAM chip based on the ANDing
--of the RD or the WR to the input address. If its RD thats ANDed, then
--the 8051 is calling for a read from the RAM. If it is WR that is ANDed,
--then the 8051 is trying to write to the RAM chip
process (ADDIN,RD)
begin
if ADDIN="00100000" and RD='0' then RAMOE<='0';
else RAMOE<='1';
end if;
end process;
process (ADDIN,WR)
begin
if ADDIN="00100000" and WR='0' then RAMWEN<='0';
else RAMWEN<='1';
end if;
end process;
end behavioral;
```