Programmable Peripheral Application Note 033

Keypad Interface to PSD4XX/5XX with Autoscanning

By Ching Lee

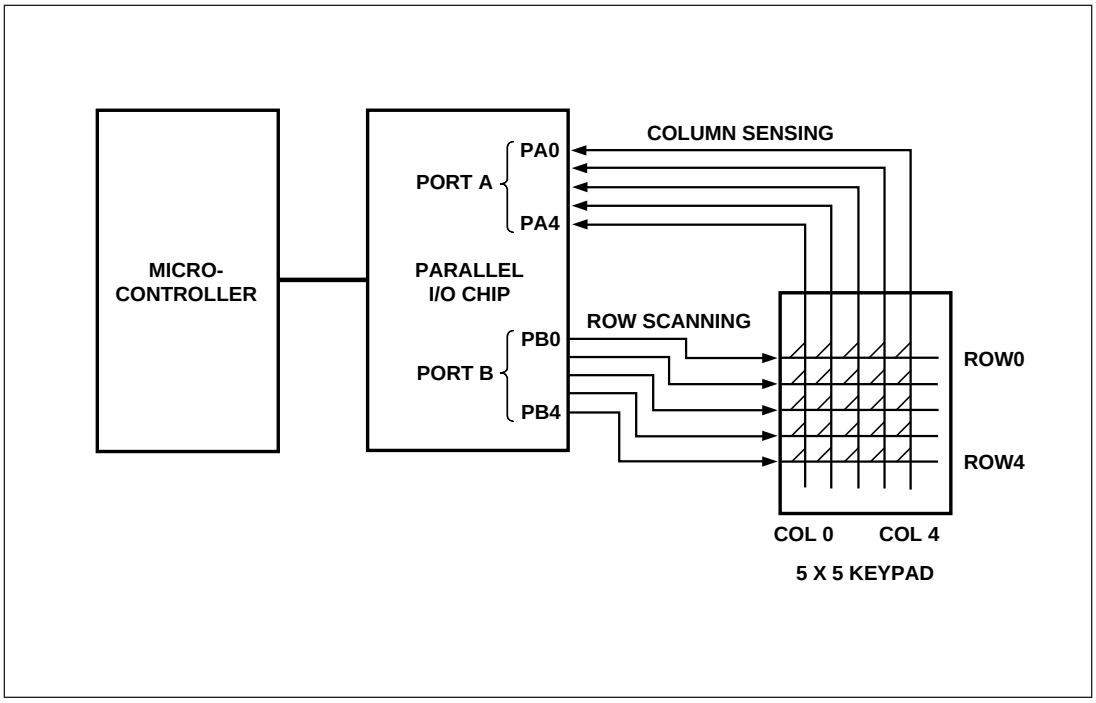
Introduction

The integration of complex PLD and I/O functions in the PSD4XX/5XX is well suited to the implementation of I/O interface logic such as a keypad controller. This application note describes how to take advantage of this PSD4XX/5XX feature to design an efficient and power saving keypad interface.

Typical Keypad Interface

A keypad consists of a matrix of pressure or touch activated switches. Figure 1 shows a typical keypad interface using a PIO (parallel I/O) chip. It is assumed that the keypad has internal pull ups for the rows and columns. The keypad has 25 keys, and is arranged in a 5 (row) x 5 (column) matrix. In this example, Port B is configured as an output port (PB0 – PB4) and driving logic "0" to the 5 row inputs of the keypad. Port A is configured as an input port (PA0 – PA4). PA0 – PA4 are normally pulled high by internal keypad resistors until one of the keys is pressed. For example, if key [3,1] (row 3, column 1) is pressed, then the "0" on PB3 is passed through the closed switch to column 1.

Figure 1. Keypad Interface



Typical Keypad Interface (Cont.)

Detection of the key closure usually involves the following steps:

- ❏ The microcontroller program continues to poll Port A to determine if any of the inputs are low. If data on Port A is switched from “1F” (no keys are pressed) to “17” (PA3 is low), the microcontroller can then identify that one of the keys in column 1 is pressed.
- ❏ To eliminate erroneous read due to key switch bouncing, the software executes a delay routine and reads Port A again after the column inputs are stable.
- ❏ After a key closure from column 1 is detected, the microcontroller reverses the direction bits of Port A and Port B. Now Port A acts as an output port and Port B as an input port. Port A drives back “17” to the column inputs.
- ❏ The microcontroller then reads Port B which acts as an input port for the rows. If it reads “17” (PB3 is low), then it can identify that the key common to row 3 and column 1 (key [3,1]) is pressed. This can be done through a look up table.

This keypad interface technique can also be implemented in the PSD4XX/5XX by connecting the rows and columns to the I/O ports as described above. The microcontroller must be always active and must keep on polling the Ports for keypad input.

A More Efficient Keypad Interface Implementation

The major overhead of the above keypad interface is:

- ❏ The microcontroller must poll the port at a fixed frequency, thus reduce the processor performance.
- ❏ The microcontroller must remain active and consumes power even when the keypad is idle.

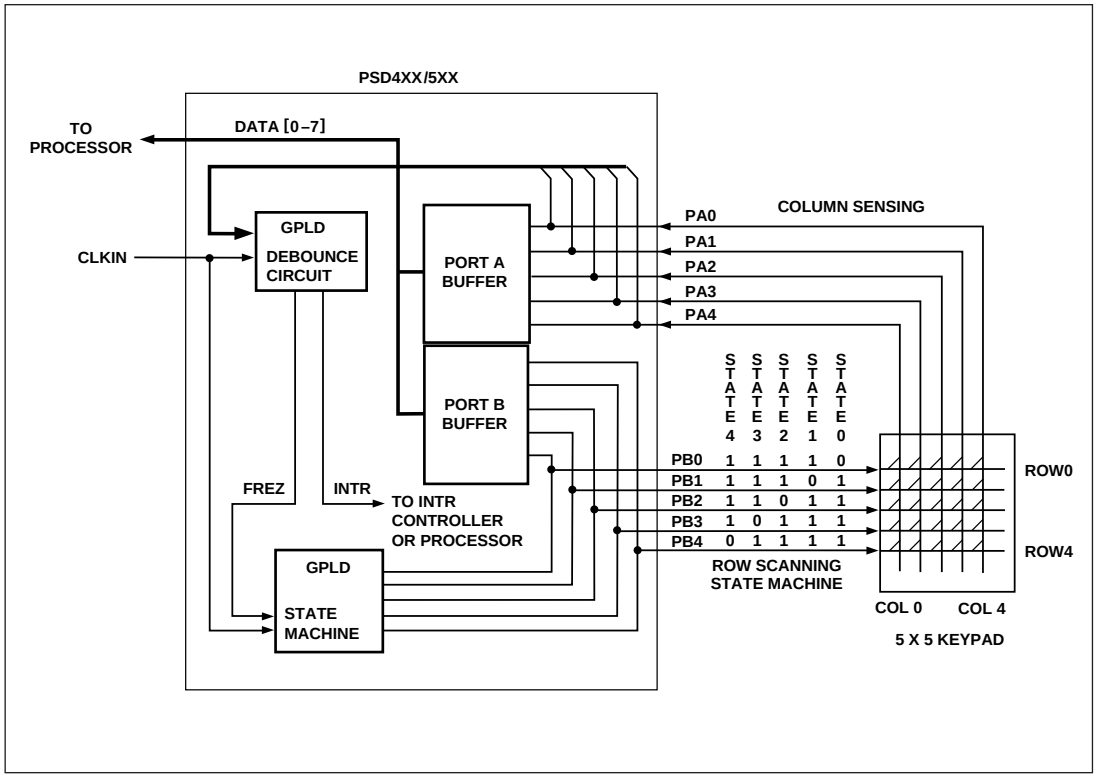
A more efficient way of interfacing to a keypad which reduces the above overhead is described here. The PSD device will perform the interface function automatically by:

- ❏ Implementing a hardware debounce circuit in the GPLD of the PSD4XX/5XX, replacing software debouncing.
- ❏ Implementing a state machine in the GPLD to scan the rows of the keypad automatically, replacing software polling.
- ❏ Setting Port A as a column input port and Port B as a scan output port.
- ❏ Generating an interrupt to the microcontroller only when a key is pressed.

The concept of this design is shown in the block diagram in Figure 2. The block diagram shows only the I/O Ports and GPLD portion of the PSD4XX/5XX which are used in the keypad interface. The following paragraphs describe the PSD configuration and GPLD logic function.

A More Efficient Keypad Interface Implementation (Cont.)

Figure 2. PSD Implementation



PSD I/O Port Configuration

Port B is configured as an output port for the GPLD. Outputs of the scanning state machine are routed to Port B and are connected to the row inputs of the keypad. The outputs of the state machine can be read by the microcontroller via the Port B Buffer (Data In Register or Macrocell Out Register).

Port A is configured as an input port for the GPLD and is connected to the column outputs of the keypad. The column outputs can also be read by the microcontroller via the Data In Register of Port A.

GPLD Logic Implementation

The GPLD implements both a debounce circuit and a scanning state machine. Both functions can be fitted in the PB macrocells and can run on the same input clock (clkin). The state machine is clocked by the rising edge of clkin, while the debounce circuit uses the falling edge of clkin.

The Debounce Circuit

The bounces on the keypad column outputs due to switch opening/closing can lead to an erroneous result. The debounce circuit performs two functions:

- Generates a “freeze” signal when a key is pressed. This signal, frez, is used to stop the state machine until the key is released. The ABEL equation is

  frez := !(col0 * col1 * col2 * col3 * col4);

- Generates an interrupt, “intr”, to the microcontroller when the column outputs stay low for two (or more) consecutive clocks. This is to ensure that the inputs are stable before interrupting the microcontroller. The ABEL equation is

  intr := frez * !(col0 * col1 * col2 * col3 * col4);

The clock input to the debounce circuit can be derived from the system clock, but the clock period should be larger than the switch bounce time.

The Scanning State Machine

The state machine does the keypad scanning by sending a “running 0” pattern to the row inputs at the rising edge of the input clock via Port B. For a 5 row keypad, the “running 0” patterns at each clock are:

Clock	Row 0	Row 1	Row 2	Row 3	Row 4
1	0	1	1	1	1
2	1	0	1	1	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	0
6	0	1	1	1	1
7	1	0	1	1	1

The pattern is repeated every five clocks. The sequence of events when a key [3,1] (row 3, column 1) is pressed at clock 2 are:

- At clock 2: Key [3,1] is pressed. The “0” in the pattern (row 1) is not passed to column 1 output.
- At clock 3: The “0” in the pattern (row 2) is not passed to column 1 output.
- At clock 4: The “0” in the pattern (row 3) is passed to column 1 output via the closed/pressed key [3,1].
- At the falling edge of clock 4, the “0” causes the debounce circuit to generate the “frez” signal and freezes the state machine.
- At the next clock, if column inputs are stable and remain low, the debounce circuit generates an interrupt which wakes up the microcontroller.
- The microcontroller reads Port A. The column inputs are “17h” which indicates a key in column 1 was pressed.
- The microcontroller reads the output of the state machine (“running 0” pattern). The value is “1Dh”. This indicates a key in row 3 was pressed.
- By using a look up table, the microcontroller identifies the pressed key to be key [3,1]. The microcontroller puts itself back to power down/sleep mode.
- The state machine remains in a stop condition until the pressed key is released. After the key is released, the state machine returns to generating the “running 0” pattern.

The Scanning State Machine (Cont.)

The state machine has 5 states and you can assign the "running 0" pattern as the state value. The operation of the state machine, including the debounce circuit, is described in ABEL as follows:

```
"state values (running 0 pattern)
   sreset  = ^b00000;
   scanr0  = ^b11110;
   scanr1  = ^b11101;
   scanr2  = ^b11011;
   scanr3  = ^b10111;
   scanr4  = ^b01111;

frez := !(col0 * col1 * col2 * col3 * col4);  active high

intr  := frez * !(col0 * col1 * col2 * col3 * col4); active high

"frez is active when key is pressed

rowreg.c  = clk;     "scanning clk  =  clk
rowreg.re = !rst;    "clear registers at reset

state_diagram rowreg;

state sreset:     goto scanr0;
state scanr0:     if  !frez  then  scanr1  else  scanr0;
state scanr1:     if  !frez  then  scanr2  else  scanr1;
state scanr2:     if  !frez  then  scanr3  else  scanr2;
state scanr3:     if  !frez  then  scanr4  else  scanr3;
state scanr4:     if  !frez  then  scanr0  else  scanr4;

"if no frez, state machine runs continuously
```

Implement The Keypad Interface In The PSD4XX/5XX

This Keypad design can be implemented in any of the PSD4XX/5XX devices. There are two ways to implement the keypad row scanning function:

- Use the state machine as described above. This approach is restricted to a keypad with a few rows. As the number of rows increase, the number of product terms required by the state machine also increases and soon there will not be enough product terms. The ABEL file which defines the GPLD logic function of this implementation, keya.abl, is shown in Appendix A.

- Use a circular shift register to generate the "running 0" pattern instead of a state machine. The shift register needs only one product term per output and can interface to keypads with large row counts. During reset, the register is set/preset with the "running 0" pattern (11110). After reset, the "0" in the pattern is shifted and repeated between the row inputs. The clock input to the shift register is "anded" with the frez signal and will stop shifting after a key is pressed. The ABEL file of this implementation is shown in Appendix B.

A stimulus file, keypad.stl, which simulates the keypad operation is included in Appendix C. The stimulus file shows the steps required to set up Port A and the reading of column and row values by the microcontroller after a key is pressed.

The PSD4XX/5XX frees up valuable I/O ports on the microcontroller, and off-loads some of the keypad software overhead. The resulting design allows better utilization of microcontroller resources.

Appendix A. KEYA.ABL File

```
module keya
title 'test:keyboard autoscanning, 80C196 bus interface';

"Input signals

col0, col1, col2, col3, col4 pin 27,26,25,24,23; "key bd column inputs

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

clkin, rst pin 42, 40;

"PLD output signals.

csiop, rs0, es0, es1, es2, es3 node;     "More outputs using reserved names.
intr pin;                                "key board interrupt
frez node;
nclkin node;                             "reverse of clkin

row0, row1, row2, row3, row4 pin 50,49,48,47,46;    "row scanning outputs
row0, row1, row2, row3, row4 is type 'buffer, reg_d';


"Definitions

rowreg  =  [row4, row3, row2, row1, row0];

"state values
        sreset   =  ^b00000;
        scanr0  =  ^b11110;
        scanr1  =  ^b11101;
        scanr2  =  ^b11011;
        scanr3  =  ^b10111;
        scanr4  =  ^b01111;

c  = .c. ;    " Clock pulse definition
X  = .x. ;    " Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];
```

Appendix A. KEYA.ABL File (Cont.)

```
equations

csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF); " 256 block
rs0   = (Address <= ^h087FF) & (Address >= ^h08000);  "2k block
es0   = (Address <= ^h01FFF) & (Address >= ^h00000);  "32KB block

frez  := !(col0 * col1 * col2 * col3 * col4);   "active high frez
intr  := frez * !(col0 * col1 * col2 * col3 * col4);   "active high intr


"intr is active when key is pressed

nclkin     = !clkin;  "reverse clkin for debounce circuit
frez.c     = nclkin;  intr.c = nclkin;
rowreg.c   = clkin;  "scanning clk = clkin
frez.re    = !rst;  intr.re = !rst;
rowreg.re  = !rst;  "reg. clear input

state_diagram rowreg;

state sreset:   goto scanr0;
state scanr0:  if !frez then scanr1 else scanr0;
state scanr1:  if !frez then scanr2 else scanr1;
state scanr2:  if !frez then scanr3 else scanr2;
state scanr3:  if !frez then scanr4 else scanr3;
state scanr4:  if !frez then scanr0 else scanr4;

"if no interrupt, state machine runs continously

test_vectors

([clkin,  rst, col0, col1, col2, col3, col4] –> [row0, row1, row2, row3, row4, intr])
 [ c,    0 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 0 , 0 , 0 , 0 , 1 ];
 [ c,    0 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 0 , 0 , 0 , 0 , 1 ];
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 1 ];
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 0 , 1 , 1 , 1 , 1 ];
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 0 , 1 , 1 , 1 ];
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 1 , 0 , 1 , 1 ];
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 1 , 1 , 0 , 1 ];
 "key (1,1) is pressed/closed
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 1 ];
 [ c,    1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
"column (col1) detects  key is pressed, intr is generated. Scanning stops
"until intr goes away
 [ c,    1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
 [ c,    1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
 [ c,    1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
"
"
"MCU reads column inputs and scanning outputs, determined key (1,1) has been
"closed. Later key (1,1) is released, intr becomes inactive and scanning resumes
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 0 , 1 , 1 , 1 , 1 ];
 [ c,    1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 0 , 1 , 1 , 1 ];


END
```

Appendix B. KEYB.ABL File

```
module keyb
title 'test:keyboard autoscanning, 80C196 bus interface';

"Input signals

col0, col1, col2, col3, col4 pin 27,26,25,24,23;   "column inputs, Port A

"Address lines, using reserved names.

a15,a14,a13,a12,a11,a10,a9,a8,a1,a0 pin;

clkin, rst pin 42, 40;


"PLD output signals.

csiop, rs0, es0, es1, es2, es3 node;                    "More outputs using reserved names.
intr pin                                                "key board interrupt
frez node;
nclkin node;                                            "reverse of clkin
row0, row1, row2, row3, row4 pin 50, 49, 48, 47, 46;    "row scanning outputs
row0, row1, row2, row3, row4 is type 'buffer, reg_d';


"Definitions

rowreg  =  [row4, row3, row2, row1, row0];

c          = .c. ;  "Clock pulse definition
X          = .x. ;  "Don't care
Address = [a15,a14,a13,a12,a11,a10,a9,a8,X,X,X,X,X,X,a1,a0];

equations

csiop = (Address >= ^h0C000) & (Address <= ^h0C0FF); "256 block
rs0    = (Address >= ^h08000) & (Address <= ^h087FF); "2k block
es0    = (Address >= ^h00000) & (Address <= ^h01FFF); "8KB block
es1    = (Address >= ^h02000) & (Address <= ^h03FFF); "8KB block


frez  :=  !(col0 * col1 * col2 * col3 * col4);  "active high frez
intr   :=  frez * !(col0 * col1 * col2 * col3 * col4);  "active high intr


"frez/intr is active when key is pressed
```

Appendix B. KEYB.ABL File (Cont.)

```
nclkin    = !clkin; "reverse clkin for debounce circuit
frez.c    = nclkin; intr.c = nclkin;
frez.re   = !rst; intr.re = !rst;
rowreg.c = clkin & !frez;  "scanning clk = clkin if no frez

row0.re  = !rst    ;  "set row registers initial value to 11110
row1.pr  = !rst    ;  "PSD macrocell has active high reset
row2.pr  = !rst    ;
row3.pr  = !rst    ;
row4.pr  = !rst    ;

row0.d   = row4.q ;  "5-bit shift register
row1.d   = row0.q ;  "shifting stops if frez is active
row2.d   = row1.q ;
row3.d   = row2.q ;
row4.d   = row3.q ;


"if no frez, shift register runs continously


test_vectors

([clkin,  rst, col0, col1, col2, col3, col4] –> [row0, row1, row2, row3, row4, intr])
 [ c,     0 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 1 ];
 [ c,     0 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 1 ];
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 0 , 1 , 1 , 1 , 1 ];
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 0 , 1 , 1 , 1 ];
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 1 , 0 , 1 , 1 ];
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 1 , 1 , 0 , 1 ];
"key (1,1) is pressed/closed
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 1 ];
 [ c,     1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
"column (col1) detects  key is pressed, intr is generated. Scanning stops
"until intr goes away
 [ c,     1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
 [ c,     1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
 [ c,     1 , 1 , 0 , 1 , 1 , 1 ] –> [ 0 , 1 , 1 , 1 , 1 , 0 ];
"
"
"MCU reads column inputs and scanning outputs, determined key (1,1) has been
"closed. Later key (1,1) is released, intr becomes inactive and scanning resumes
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 0 , 1 , 1 , 1 , 1 ];
 [ c,     1 , 1 , 1 , 1 , 1 , 1 ] –> [ 1 , 1 , 0 , 1 , 1 , 1 ];


END
```

Appendix C. KEYPAD.STL File

```
//auto scanning simulation
//start scanning, press key, read port A (column) and port B (row)


//+++++++++++++++++++++++++++++++++++++++++++++++++
//  Defining tasks to simplify the stimulus file
//+++++++++++++++++++++++++++++++++++++++++++++++++

task write (addr_bus,bhe_value,data_in);  // 80196 write bus cycle

input  [15:0] addr_bus;
input  [15:0] data_in;
input  bhe_value;

      begin

      #20   ale   = 1;  //Latch the address lines
      #20   adio =  addr_bus;  //Set-up the right address
            bhe  =  bhe_value;
      #20   ale   = 0;  //Ale inactive

      #20   adio =  data_in;  // Write operation

      #40   wr    = 0;  // Write pulse
      #100  wr    = 1;  // Write ends
      #10   adio =  Z16;  bhe = Z;
      end

endtask


task read (addr_bus);    //80196 read bus cycle

input [15:0] addr_bus;

      begin

      #20   ale   = 1;  //Latch the address lines
      #20   adio =  addr_bus;  //Set-up the right address
            bhe  =  0;
      #20   ale   = 0;  //Ale inactive

      #20   adio =  Z16;  // Float Address bus

      #40   rd    = 0;  // Rd pulse
      #100  rd    = 1;  // Rd ends
      #10   bhe  =  Z;
      end

endtask

reg [4:0] column;
assign  {col4, col3, col2, col1, col0}  =  column;
assign  {row4, row3, row2, row1, row0}  =  row;
reg intr, frez;
initial
```

Appendix C. KEYPAD.STL File (Cont.)

```
begin

    rst     = 0;   //generate reset
    wr      = 1;  rd  =  1;   //initialize control signal
    ale     = 0;  bhe  =  1;
    adio    = 16'bz;  //initialize addr/data bus
    intr    = 'bz;  frez  =  'bz;
    clkin   = 0;  pd5  =  0;  pd6  =  0;  pd7  =  0;   //init not used port pins
                  pa5  =  0;  pa6  =  0;  pa7  =  0;
    row     = 5'bz;
    column  = 5'b11111;
    csi     = 0;    //set PSD5XX chip select low

#300  rst   =  1;    //after 500ns, rst inactive

//write and read to the sram, verify bus interface is ok
write ('h8476,0,'h5a27);

//read sram,word
read ('h8476);       //Word-read operation

//write Port A Control Register, configure Port as I/O
write ('hC002,1,'hff);

//write Port A Direction Register, configure Port A as input
write ('hC006,1,'h00);


#635  column  =  'b11101;    //press key (3,1)  --  row 3, column 1
                             //state machine is freezed
                             //intr is generated to the MCU

//MCU reads Port A Data In Reg.  (column inputs)
read ('hc000);

//MCU reads Port B Macrocell Out Reg.  (state machine row pattern)
read ('hc00d);


#500  column  =  'b11111;    //key is released
                             state machine resumes operation

end

always
    #200 clkin  =  ~clkin;
```