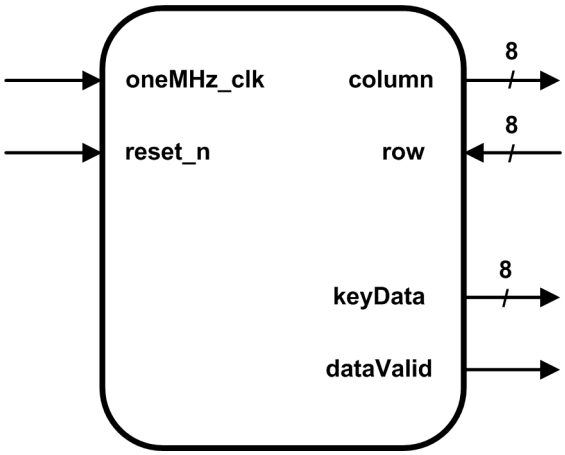
PERDYN11088™

KEYPAD INTERFACE PERIPHERAL

IP Reference Manual

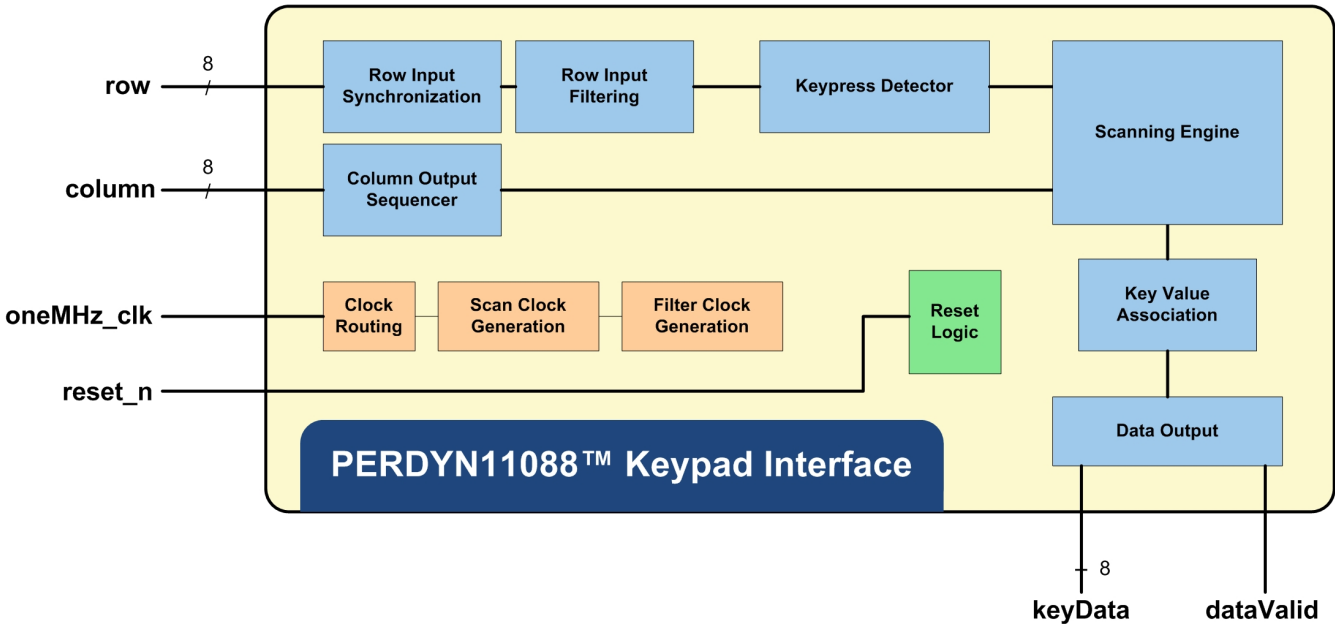
Overview

This peripheral core provides functionality for interfacing to a multiplexed mechanical keypad up to 8 rows by 8 columns in size. The switch closures may be membrane-style contacts, simple push-buttons, or nearly any other electrical contact. Our reduced EMI keypress algorithm is a great benefit over traditional scanning keypad interfaces, and built-in contact debouncing provides clean, reliable keypress data.



Key Features

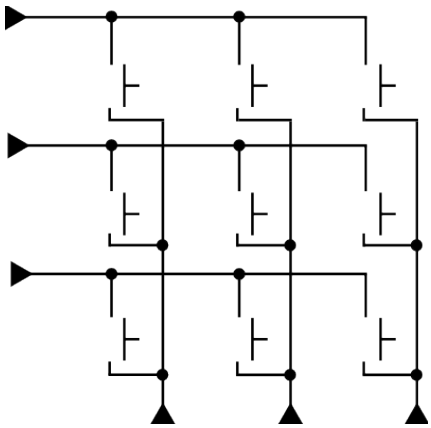
- ✔ **Connects multiplexed keypads up to 64 keys to a host processor**
- ✔ **Filters mechanical contact bounce**
- ✔ **Low-EMI keyscan algorithm aids compliance approval**
- ✔ **Off-the-shelf & custom interface wrappers available**
- ✔ **Verilog™ RTL synthesizeable in any vendor fabric**



Core Operation

The PERDYN11088 IP has application in an array of consumer and industrial applications, providing a reliable and convenient method of interfacing today's operator-interface keypads to their host processor. Whether the membrane keypad of a hand-held consumer product, the elastomeric or silicone rubber keypad of a rack-mounted instrument, or even an array of mechanical pushbuttons, the need is the same: to coordinate the interconnected rows and columns in such a way that key-presses are reliably detected and utilized.

No matter the physical size or shape of a keypad, the electrical row/column structure remains fundamentally the same. Pressing a button connects a particular row to a particular column, thus creating a circuit path between the two. The PERDYN11088 IP detects this event, processes the row and column information, and reports a unique value to the host – thereby indicating the specific key that has been pressed.



3x3 Multiplexed Keypad Schematic

4 Row by 3 Column (4x3) Keypad

A keypad's precise number of rows and columns is a design decision based largely upon the quantity of buttons needed and the row/column configuration. The PERDYN11088 IP can work with keypads having up to 8 rows and 8 columns in total, which allows for up to 64 buttons total. Although small keypads may be interfaced, more commonly keypads ranging from 12 to 36 buttons (typically arranged in a 3x4, 5x6 or 6x6 arrangement) are used with the PERDYN11088. As mentioned, keypads up to 64 buttons may be interfaced using the 8x8 arrangement.

The PERDYN11088 IP is typically used alongside a host micro-controller or microprocessor, relieving the processor of the burden of scanning for keypresses and debouncing pushbutton contacts. In addition, often a smaller footprint microcontroller may be utilized in the application due to the reduced I/O pin demand. Utilizing a serial interface wrapper on the PERDYN11088 IP reduces the required micro-controller pincount even further.

A synthesized 'soft' processor may also be utilized, allowing the PERDYN11088 IP to sit alongside the processor inside the FPGA. Generic processor I/O may be utilized to read keypress data from the PERDYN11088 IP, or a custom interface wrapper may be utilized to place the PERDYN11088 IP directly onto the processor's data bus.

Interfaces

The PERDYN11088 IP has 6 interface ports, as described below. Two ports (row, column) are for keypad connection; two more (keyData, dataValid) are for data output, and the remaining two (oneMHz_clk, reset_n) handle internal synchronization.

Interface	Width	Direction	Polarity	Description
row	8	Input	Active-high	Keypad matrix row connections
column	8	Output	Active-high	Keypad matrix column connections
keyData	1	Output	Active-high	Keypress data value
dataValid	6	Output	Active-high	Keypress data valid strobe
oneMHz_clk	1	Input	Active-high	IP clock
reset_n	1	Input	Active-low	IP reset

PERDYN11088 Interface Ports

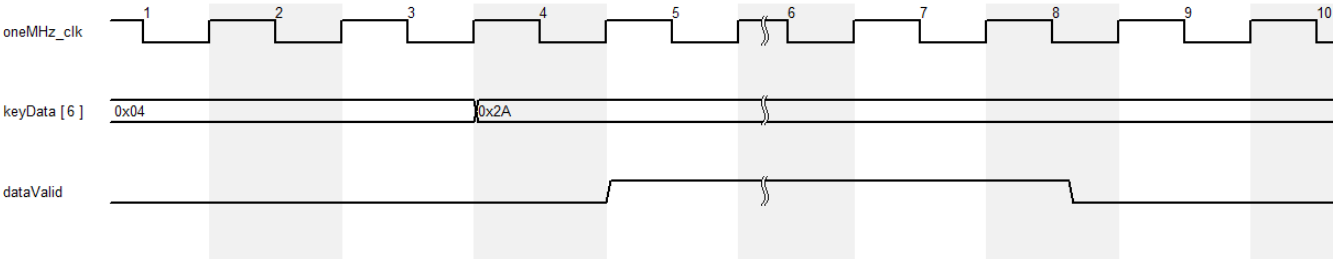
'row' is an 8-bit wide port designed for connection to a multiplexed keypad's rows. Up to eight rows (row 0 to row 7) may be connected to the PERDYN11088 IP. Smaller keypads having less rows may be connected, however a pullup resistor is required on each of the 8 rows. A value of 4.7K is typically utilized, however factors such as I/O voltage and keypad cable length may cause a designed to modify this value.

'column' is also an 8-bit wide port designed for connection to a multiplexed keypad's columns. Up to eight columns (column 0 to column 7) may be connected, and smaller keypads will utilized only a portion of the PERDYN11088 IP columns. All used column outputs must have a series resistor between the programmable logic device and the keypad to limit current when multiple keys are pressed. A resistance value of 330 Ohms is typically utilized, however factors such as I/O voltage and keypad cable length may cause a designed to modify this value.

'keyData' is a six-bit output, with a value representing one of the 64 possible key values ($00 through $3F). This port works in harmony with the *dataValid* strobe. The data value being output when the dataValid strobe occurs is the key value representing the most recently pressed key. See the *keyData* output table for additional details.

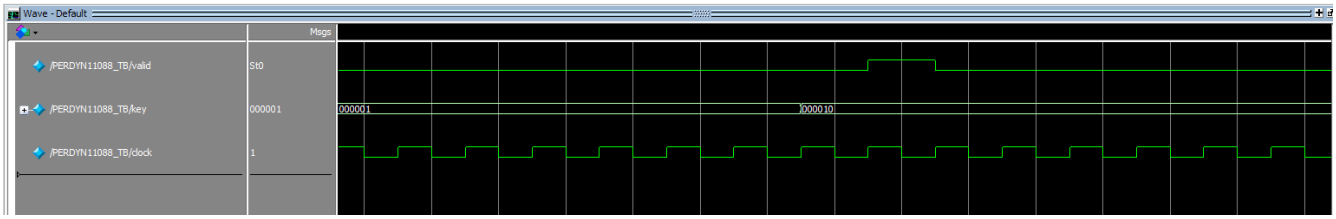
'dataValid' is an active-high data strobe indicating that *keyData* port data is valid and current. The *dataValid* strobe is held high the duration that the key is held, with a minimum of 2 microseconds for an extremely brief trigger event such as the testbench provides. Typically, the rising edge of *dataValid* indicates the start of the keypress, and the falling edge indicates the release of the key. As seen in the testbench waveform capture below, the keypress data is held beyond the duration of the *dataValid* pulse, so a processor interrupt triggered by the rising edge of *dataValid* would not need to be in a hurry to capture the data before it clears.

Shown below is a timing diagram illustrating the relationship between the *keyData* and *dataValid* output signals. Note that the rising edge of *keyData* occurs one clock cycle following the update of *keyData* with the new keypress information. The length of *dataValid's* high period is dependent upon how long the key is held.



Relative Timing of keyData and dataValid Output Signals

To illustrate, below is a screen capture from the PERDYN11088 testbench simulation. The PERDYN11088 signal '*dataValid*' is mapped to the testbench signal 'valid' (top waveform), and the PERDYN11088 signal '*keyData'* is mapped to testbench signal 'key' (middle waveform). Note the transition of *keyData (key)* from binary value '000001' to '000010', immediately followed by a strobe of the *dataValid* signal.



KeyData and dataValid Testbench Waveforms

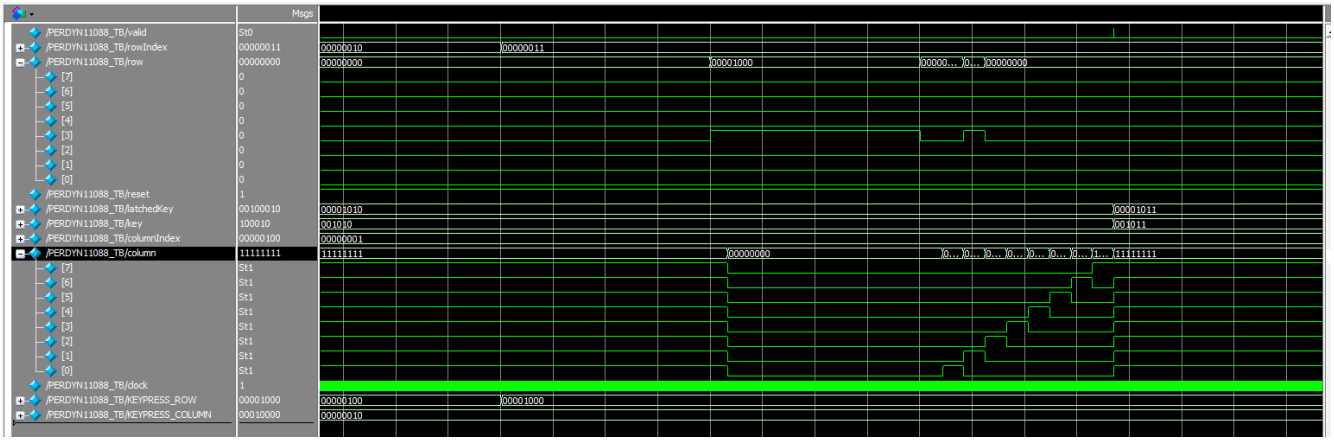
'oneMHz_clk' is the primary input clock for the PERDYN11088 IP. All timings mentioned in this reference manual assume a 1MHz input clock. If an extra clock PLL circuit is available in the programmable logic device, it is recommended to use it to generate this 1MHz clock, in phase with the primary system clock. If a PLL is not available, a programmable clock divider block (included with PERDYN11088 IP) may be used.

'reset_n' is a synchronous reset input used as a power-on reset or to restore the PERDYN11088 to a know condition. This input is an active-low signal, requiring a high input during ordinary (non-reset) operation.

Keypress Cycle

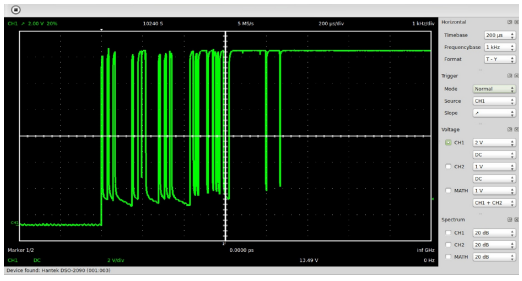
In an effort to reduce RF emission, the PERDYN11088 IP does not continuously scan the keypad as most keypad interfaces do. Instead, the column drivers hold a constant signal on the keypad columns, and the row inputs watch for this signal when a key is pressed – thus joining row to column. At this time, a single scan cycle is triggered, during which the precise key pressed is identified. The value associated with that key is output on the *keyData* port, and the *dataValid* signal is activated until the button is released.

The screen capture below illustrates a keypress cycle being simulated by the PERDYN11088 testbench. The testbench signal names are described later in this document. Notice the column drivers being held high, and then scanning one by one through all eight columns. Once this is complete, the precise key that was pressed is determined, and is reported on the output signals. Note that because this is a simulation, and not a human keypress, the *dataValid* (valid) signal is very brief. Typically this signal is longer because is lasts the duration of a human keypress.



Testbench Simulation Showing Keypress and Resulting Scan and Data Output

Contact Debounce



When two mechanical plates make contact, they bounce a few times before settling in to make good contact. This bouncing will be seen as multiple keypresses by a microcontroller, therefore the contacts must be 'debounced', or filtered, so that only a single switch closure is detected.

This oscilloscope waveform capture depicts a typical mechanical switch closure. It is clear to see the noise generated as the contacts begin to come together. The duration of the contact bounce noise depends upon several mechanical factors, but is

Typical Mechanical Contact Bounce

typically 1ms to 6ms in duration. Keypads with extremely noisy contact bounce, or with an extremely long transition period, may require customization to the PERDYN11088 IP.

Rows & Columns

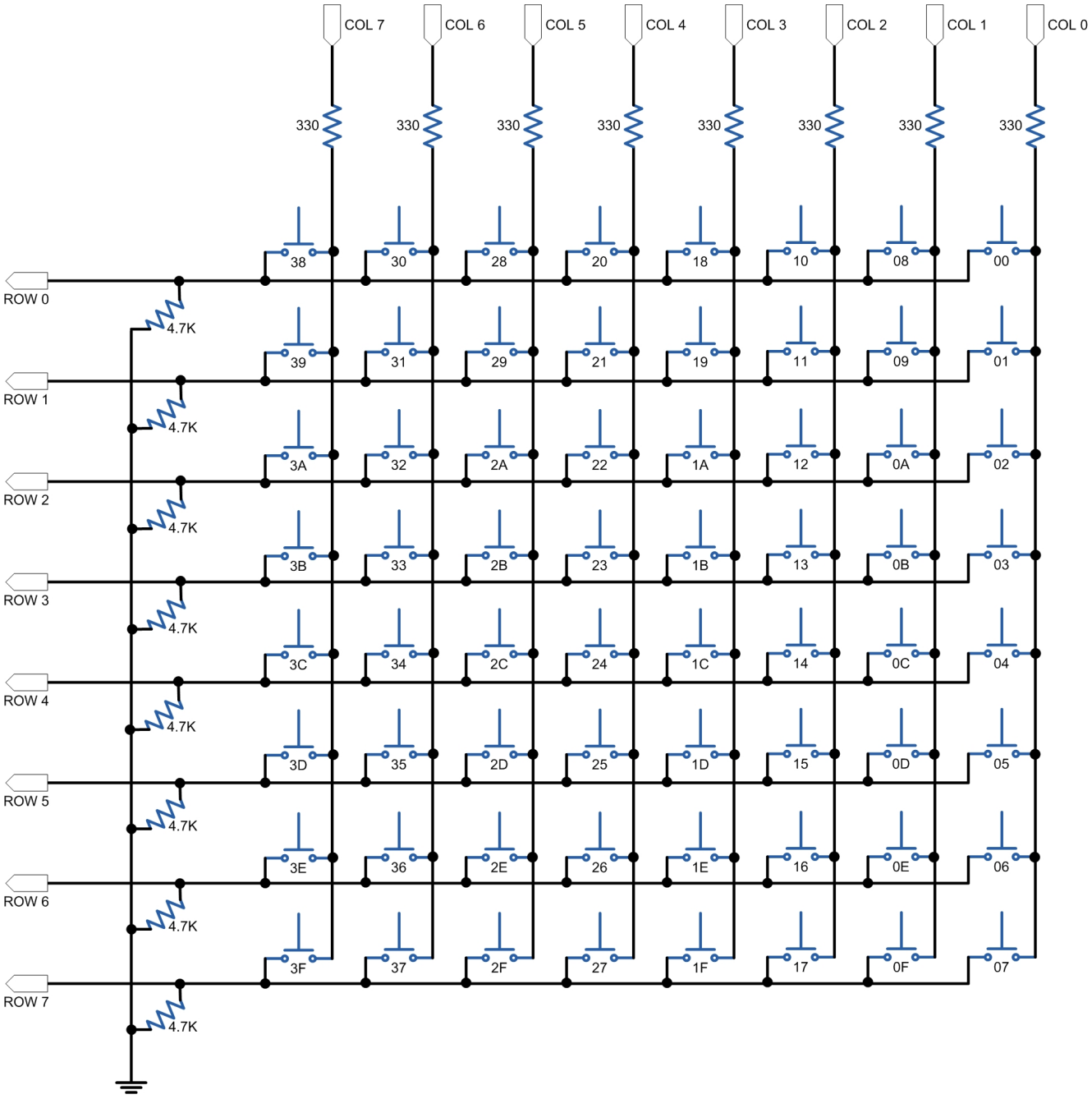
The following schematic illustrates connection of a 8 row x 8 column, 64-key keypad to the PERDYN11088 row and column interfaces. Two items of particular interest are the series resistors on each of the columns, and the pull-down resistors on each of the rows. The 330 Ohm series resistors are there to limit current from the column drivers in the event that two or more keys are pressed simultaneously (effectively shorting a high driver and a low driver to one another). The values of the series resistors should be chosen to limit the per-pin source and sink current for the programmable logic device that the PERDYN11088 IP is synthesized in. For example, suppose two keys on row 0 are being held. During the column scan, the column of one key will be driven high, while the column of another key is driven low. Suppose the I/O voltage is 3.3V, and the combined resistance through the loop is 660 Ohms (two 330 Ohm series resistors). The current flow in this example is 5mA, which is typically an acceptable value for a FPGAS or CPLD I/O pin. Calculated current should be conservative to handle the case that all keys in a particular row are depressed simultaneously; the effective loop resistance in this case would be approximately 377 Ohms (330 Ohms + Seven 330 Ohm resistors in parallel). For smaller keypad in which not every column of the PERDYN11088 is connected to the keypad, a series column resistor is not required for the unused columns. Alternately, unused column driver outputs do not need to be brought outside the FPGA, and may be left unconnected.

The 4.7K pull-down resistors on each of the row inputs prevent the inputs from floating when they are not being driven. All eight row pull-down resistors must be used, even for smaller keypads that do not make use of all eight PERDYN11088 row connections. If the unused row input connections are not brought external to the FPGA, they must be connected to ground for synthesis.

If the value of the 330 Ohm column series resistors is changed by a large amount, the value of the pulldown resistors should be recalculated to ensure that the voltage divider formed by a series resistor and a pulldown resistor while a key is being pressed keeps the voltage sent to the row inputs inside the acceptable 'high' input range for the FPGA I/O pin.

The PERDYN11088 is designed to receive only one keypress at a time, and multiple key-presses typically return the value associated with the higher row or column value(s). In addition, if three adjacent keys are pressed, a 'ghost key' effect can result due to the connection of the adjacent rows and columns – effectively incorporating a fourth 'ghost' key in the corner of the rectangle formed by the other three pressed keys. If a *Shift*, *Alt* (alternate), or *Ctrl* (control) type key is required to be pressed simultaneously with another key, this special key should not be incorporated into the key matrix and should be monitored by the host independently of the PERDYN11088. When the PERDYN11088 reports a keypress, the state of the special key(s) should be checked to see if alternate keypres functionality is required.

Note the hexadecimal value returned with each key location printed near each of the sixty-four keys on the schematic, further detailed by the key-map table below.



Row and Column Resistors Connected to the Keypad Matrix

	Column 7	Column 6	Column 5	Column 4	Column 3	Column 2	Column 1	Column 0
Row 0	Key 56: **$38**	Key 48: **$30**	Key 40: **$28**	Key 32: **$20**	Key 24: **$18**	Key 16: **$10**	Key 8: **$08**	Key 0: **$00**
Row 1	Key 57: **$39**	Key 49: **$31**	Key 41: **$29**	Key 33: **$21**	Key 25: **$19**	Key 17: **$11**	Key 9: **$09**	Key 1: **$01**
Row 2	Key 58: **$3A**	Key 50: **$32**	Key 42: **$2A**	Key 34: **$22**	Key 26: **$1A**	Key 18: **$12**	Key 10: **$0A**	Key 2: **$02**
Row 3	Key 59: **$3B**	Key 51: **$33**	Key 43: **$2B**	Key 35: **$23**	Key 27: **$1B**	Key 19: **$13**	Key 11: **$0B**	Key 3: **$03**
Row 4	Key 60: **$3C**	Key 52: **$34**	Key 44: **$2C**	Key 36: **$24**	Key 28: **$1C**	Key 20: **$14**	Key 12: **$0C**	Key 4: **$04**
Row 5	Key 61: **$3D**	Key 53: **$35**	Key 45: **$2D**	Key 37: **$25**	Key 29: **$1D**	Key 21: **$15**	Key 13: **$0D**	Key 5: **$05**
Row 6	Key 62: **$3E**	Key 54: **$36**	Key 46: **$2E**	Key 38: **$26**	Key 30: **$1E**	Key 22: **$16**	Key 14: **$0E**	Key 6: **$06**
Row 7	Key 63: **$3F**	Key 55: **$37**	Key 47: **$2F**	Key 39: **$27**	Key 31: **$1F**	Key 23: **$17**	Key 15: **$0F**	Key 7: **$07**

Values Returned by 'dataValid' Port

Hexadecimal values listed in the table above are returned by the PERDYN11088 IP upon each press of the corresponding key. The above values, ranging from $00 (binary 000000) to $3F (binary 111111) are output on the *dataValid* port, along with a corresponding pulse on the *keyData* output to signify a new keypress. In the event of a duplicate keypress, the value on the *dataValid* port will remain constant, but a second pulse on the *keyData* pin will be output upon the second duplicate keypress.

Synthesis in Various Devices

Synthesis results for the PERDYN11088 in the major FPGA vendors are listed below. Results are for general fitting and comparison only, and may very based on optimization settings. The PERDYN11088 is compact in size and fits easily in even a smaller CPLD or FPGA, making it an ideal peripheral for an FPGA-based SoC (system on chip) or a micro-controller companion IC.

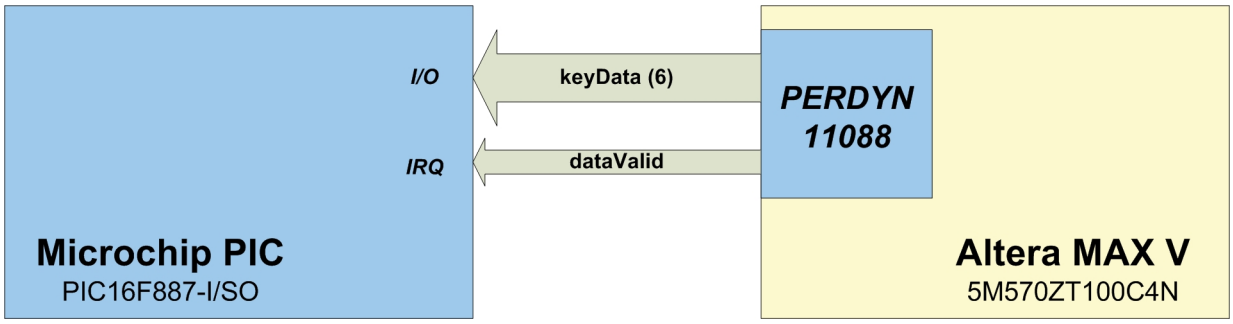
Manufacturer	IDE	Synthesis	Device	Fabric Area
Altera®	Quartus II® *v14.0*	Default	Max V® 5M570ZT100C4	238 Logic Elements
Xilinx®	Vivado® *v2013.3*	Default	Artix-7® XC7A75T-FTG256-2	LUT as logic: 144 LUT as memory: 0 LUT flip-flop pairs: 185
Xilinx®	ISE® *v14.7*	Default	Spartan 6® XC6SLX4-3TQG144	Slice registers: 112 Slice LUT: 74 LUT flip-flop pairs: 134
Microsemi®	Libero® *v11.4.1.17*	Default	ProASIC3® A3P030 48QFN -2	Core: 398 (Comb: 272 + Seq: 126)
Lattice®	Diamond® *v3.2*	Lattice LSE Engine	MachXO2® LCMXO2-1200HC-5TG144C	Slices: 71
Lattice®	Diamond® *v3.2*	Synplify Pro Engine	MachXO2® LCMXO2-1200HC-5TG144C	Slices: 73 Registers: 126 LUT4: 144
Lattice®	iCEcube 2® *v2014.12.27052*	Default	iCE40® iCE40LP1KCM121	LUT: 152 Registers: 126

Synthesis Results Across Multiple FPGA Vendor Tools

*** Altera, Xilinx, Microsemi, Lattice, Quartus II, Vivado, ISE, Libero, Diamond, iCEcube 2 and other product names are trademarks registered to their respective owners.*

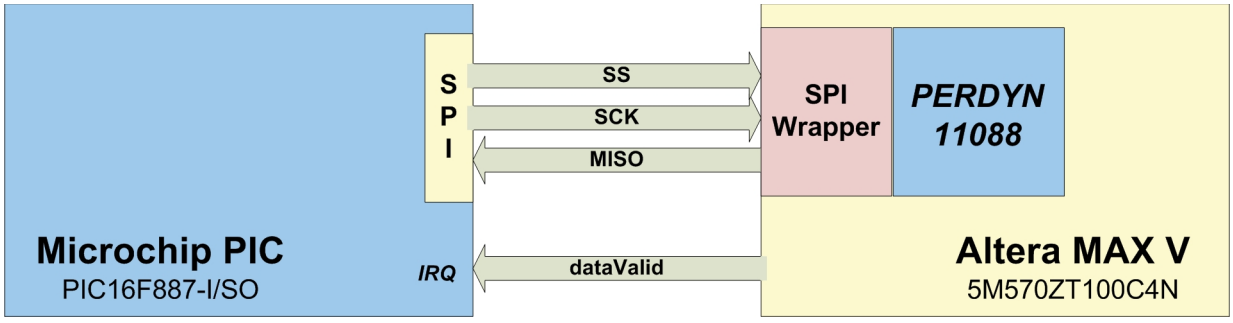
Microcontroller Companion IC

Peripheral Dynamics IP such as the PERDYN11088 is a great match for FPGA-based systems using soft processors inside the FPGA, as detailed in our '*Avalon® Bus Interface Application Note*' document. However, a second method of use is quickly gaining popularity among traditional hard-microcontroller design engineers. Pairing an FPGA alongside a microcontroller offers the microcontroller the benefits of enhanced peripheral support, increased I/O, and coordination of simultaneous events. The graphic below illustrates a typical connection between a standard hard microcontroller and a CPLD implementing the PERDYN11088 IP.



Notice that the *dataValid* signal is connected to an I/O pin being used as an interrupt input (IRQ), and the six lines of the *keyData* bus are connected to general purpose I/O to be read by the interrupt service routine. As seen in the Synthesis Results table above, the PERDYN11088 takes only a portion of the CPLD, leaving remaining logic available for other peripherals.

If your host microcontroller is low on I/O pins, an alternate interface method is available, as depicted in the graphic below. The PERDYN11088 ships with an optional SPI Wrapper, giving it a synchronous serial (SPI) interface. Nearly all modern microcontrollers contain a hardware SPI peripheral which may be used to receive keypress data from the PERDYN11088 via the standard MISO (master in, slave out), SCK (serial clock), and SS (slave select) signals. Notice that the *dataValid* signal is still used as a processor interrupt, telling the processor that a new keypress event has occurred.



*** Microchip, Altera, PIC, MAX V, SPI, and other product names are trademarks registered to their respective owners.*

Available Wrappers

As mentioned previously, the PERDYN11088 interface may be used as-is, or a front-end 'wrapper' may be added to implement an alternate interface. The document *Avalon® Bus Interface Application Note* describes a data bus wrapper that allows a processor using the Avalon bus to directly access the PERDYN11088 data using a simple bus read. Likewise, the SPI wrapper, described above, allows a serial interface to be used with the PERDYN11088. Both the Avalon bus wrapper and the SPI wrapper are included with the PERDYN11088 IP.

While not strictly required, a custom interface can greatly enhance or simplify an embedded design. Custom interfaces, implemented as 'wrappers' that convert the *keyData* and *dataValid* signals into various other signals, may be created to provide any number of custom interfaces such as I2C, APB3, Wishbone, etc. If desired, users are free to create a custom wrapper for their particular interface needs; alternately, our design services are available to assist in this regard.

Clock & Reset

The PERDYN11088 IP requires a one MHz clock signal to function properly. This clock signal, applied to the *oneMHz_clk* port, controls internal timing including contact synchronization and debounce as well as keypad scanning. If a spare hardware PLL circuit is available in the FPGA or CPLD that the core is being implemented in, it is recommended that the PLL be used to generate a 1 MHz clock.

If no PLL is available, a programmable clock divider is included with the PERDYN11088 IP. This Verilog file has an input parameter that must be set to the input clock frequency. Compile-time math adjusts the divisor size to produce a divider circuit that produces a 1 MHz clock that is relatively in-phase with the input clock. This generated clock may be used to supply the PERDYN11088 *oneMHz_clk* input port.

The reset input to the PERDYN11088 IP, *reset_n*, is a synchronous input designed to put the IP core into a know condition at powerup. It should be connected to an active-low FPGA / CPLD system reset signal. The *reset_n* input must be held high during ordinary operation.

Testbench

The following Verilog code serves as a testbench for the PERDYN11088 IP, instantiating the core and simulating a press of every key in sequence:

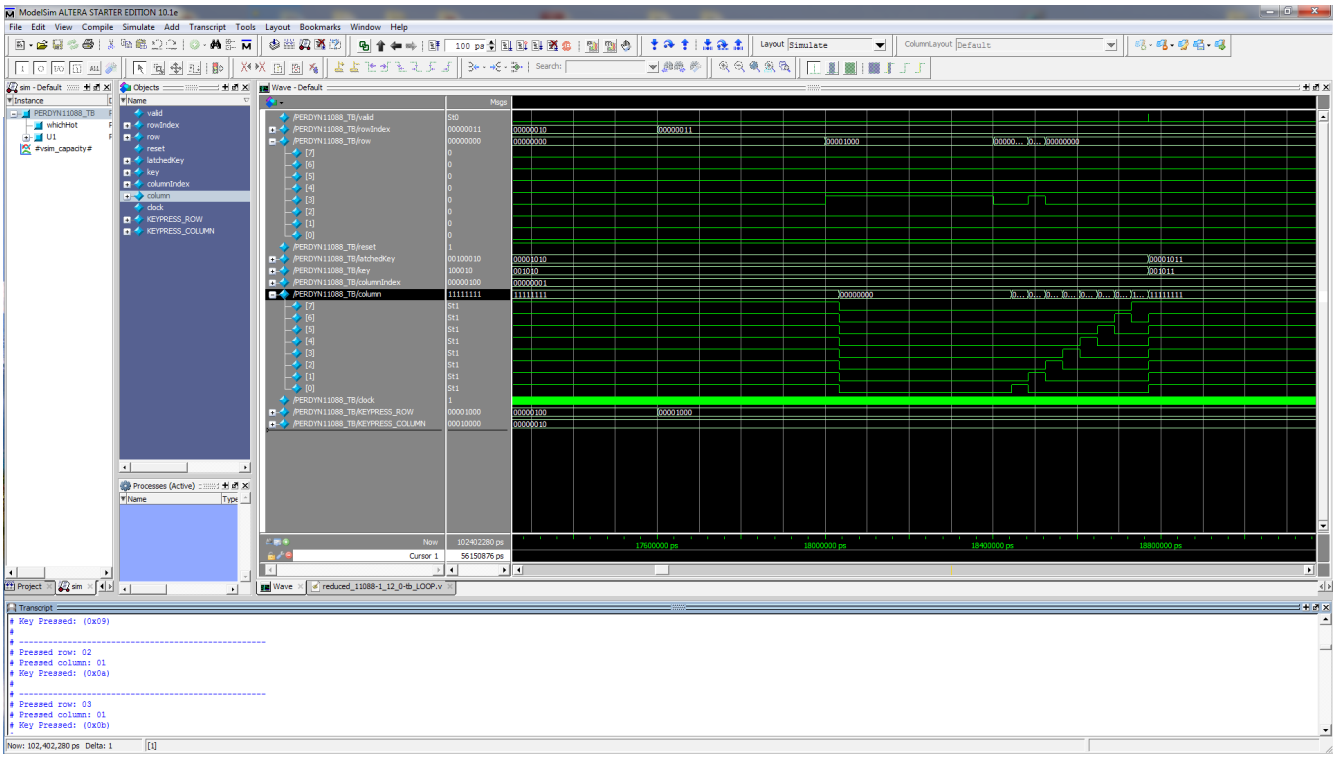
```
module PERDYN11088_TB;

// local signals
reg clock;                              // IP interface - core "oneMHz_clk" signal
reg reset;                              // IP Interface - core "reset_n" signal
wire [5:0] key;                         // IP Interface - keyData
wire [7:0] column;                      // IP Interface - keypad matrix columns
reg [7:0] row;                          // IP Interface - keypad matrix rows
wire valid;                             // IP Interface - dataValid

// local variables
reg [7:0] latchedKey;                   // returned value latched with valid strobe
reg [7:0] columnIndex;                  // outer loop index
reg [7:0] rowIndex;                     // inner loop index

reg [7:0] KEYPRESS_ROW = 8'b 0000_0001;
reg [7:0] KEYPRESS_COLUMN = 8'b 0000_0001;

//          Keypad Matrix Press Value Table
// ------------------------------------------------------------------------------------------------------------------
//
//                                         column
//                      7       6       5       4       3       2       1       0
//
//          0           $38     $30     $28     $20     $18     $10     $08     $00
//          1           $39     $31     $29     $21     $19     $11     $09     $01
//          2           $3A     $32     $2A     $22     $1A     $12     $0A     $02
//   row    3           $3B     $33     $2B     $23     $1B     $13     $0B     $03
//          4           $3C     $34     $2C     $24     $1C     $14     $0C     $04
//          5           $3D     $35     $2D     $25     $1D     $15     $0D     $05
//          6           $3E     $36     $2E     $26     $1E     $16     $0E     $06
//          7           $3F     $37     $2F     $27     $1F     $17     $0F     $07

// instantiate the PERDYN11088 core
PERDYN11088 U1(
        .oneMHz_clk(clock),
        .reset_n(reset),
        .row(row),
        .column(column),
        .keyData(key),
        .dataValid(valid)
);

initial
        begin
        clock = 0;                                      // begin with clock phase low
        reset = 0;                                      // reset the PERDYN11088 core
        repeat(200)
                #5 clock <= ~clock;

        reset = 1;                                      // disable PERDYN11088 reset
        // loop through each row and column to test all 64 key contact closures
        for (columnIndex=0;columnIndex<=7;columnIndex = columnIndex + 1)
```



```
                begin
                for (rowIndex=0;rowIndex<=7;rowIndex = rowIndex + 1)
                        begin
                        row <= 8'b00000000;

                        repeat(80000)                   // send clocks with no key pressed
                                #5 clock <= ~clock;

                        row <= column & KEYPRESS_ROW;   // blip a row to activate 'reduction OR' trigger

                        repeat(80000)                   // drop column drivers and wait for 'trigger' to clear
                                #5 clock <= ~clock;

                        // During scan, column outputs try each row.  When appropriate row matched, send column value
                        repeat(80000)
                                begin
                                #5 clock <= ~clock;

                                // Simulate continuity of a mechanical keypress
                                row = (column==KEYPRESS_COLUMN) ? KEYPRESS_ROW : 8'b00000000;

                                // If valid signal activates, capture the output data.
                                if (valid == 1'b1)
                                        latchedKey <= {2'b00,key};
                                end

                        row <= 8'b00000000;             // allow send of data and cycle completion

                        repeat(80000)
                                #5 clock <= ~clock;

                        #5 $display("\n---------------------------");
                        #5 $display ("Pressed row: %H",whichHot(KEYPRESS_ROW));
                        #5 $display ("Pressed column: %H",whichHot(KEYPRESS_COLUMN));
                        #5 $display ("Key Pressed: (0x%h)",latchedKey);

                        KEYPRESS_ROW = KEYPRESS_ROW << 1;

                        end         // inner loop

                KEYPRESS_COLUMN = KEYPRESS_COLUMN << 1;
                KEYPRESS_ROW = 8'b 0000_0001;           // reset for next round
                end         // outer loop
        end             // end initial

// ------------------------------------------------------------------------------------------------------------------------
// Row and column are defined as binary strings with one bit hot.
// This function converts this into a decimal, for instance bit 5 (0010_0000) results in '5' (rather than 0x32)
function [7:0] whichHot;
        input [7:0] inputVal;
        reg [3:0] i;  // needs to use values 0 through 8

        for (i=0;i<8;i=i+1)
                begin
                if (inputVal[i])
                        whichHot = i;
                end
endfunction
// ------------------------------------------------------------------------------------------------------------------------

endmodule
```


The testbench utilizes a six-wire bus named 'key' to hold the PERDYN11088 output data port *keyData*, and a single wire named 'valid' to pass the output data port *dataValid*. The wire 'clock' holds the PERDYN11088 *oneMHz_clk* clock signal, and 'reset' holds the active low reset signal *reset_n*. Eight-wire busses 'row' and 'column' complete the testbench signal list.

An outer loop cycles through the eight keypad rows, while an inner loop cycles through the eight keypad columns, and a keypress of each key is, in turn, simulated for the PERDYN11088 IP to detect. The testbench screen capture below illustrates the simulated keypress of key #11, which gives an output of $0B (binary 001011). Notice that row 3 and column 1, the coordinates of key #11, are currently active.



Testbench Screen Capture of Simulated Keypress

The testbench output, below, lists the row and column of each test performed, along with the data read back from the PERDYN11088 port *keyData* at the rising edge of the *dataValid* signal. Note that the testbench function 'whichHot' is used to convert the one-hot binary format of the simulated row and column keypresses into a decimal format more suitable for readability in the output report.


```
Pressed row: 00
Pressed column: 00
Key Pressed: (0x00)

----------------------------
Pressed row: 01
Pressed column: 00
Key Pressed: (0x01)

----------------------------
Pressed row: 02
Pressed column: 00
Key Pressed: (0x02)

----------------------------
Pressed row: 03
Pressed column: 00
Key Pressed: (0x03)

----------------------------
Pressed row: 04
Pressed column: 00
Key Pressed: (0x04)

----------------------------
Pressed row: 05
Pressed column: 00
Key Pressed: (0x05)

----------------------------
Pressed row: 06
Pressed column: 00
Key Pressed: (0x06)

----------------------------
Pressed row: 07
Pressed column: 00
Key Pressed: (0x07)

----------------------------
Pressed row: 00
Pressed column: 01
Key Pressed: (0x08)

----------------------------
Pressed row: 01
Pressed column: 01
Key Pressed: (0x09)

----------------------------
Pressed row: 02
Pressed column: 01
Key Pressed: (0x0a)

----------------------------
Pressed row: 03
Pressed column: 01
Key Pressed: (0x0b)

----------------------------
Pressed row: 04
Pressed column: 01
Key Pressed: (0x0c)

----------------------------
Pressed row: 05
Pressed column: 01
Key Pressed: (0x0d)

----------------------------
Pressed row: 06
Pressed column: 01
Key Pressed: (0x0e)

----------------------------
Pressed row: 07
Pressed column: 01
Key Pressed: (0x0f)
```

```
Pressed row: 00
Pressed column: 02
Key Pressed: (0x10)

----------------------------
Pressed row: 01
Pressed column: 02
Key Pressed: (0x11)

----------------------------
Pressed row: 02
Pressed column: 02
Key Pressed: (0x12)

----------------------------
Pressed row: 03
Pressed column: 02
Key Pressed: (0x13)

----------------------------
Pressed row: 04
Pressed column: 02
Key Pressed: (0x14)

----------------------------
Pressed row: 05
Pressed column: 02
Key Pressed: (0x15)

----------------------------
Pressed row: 06
Pressed column: 02
Key Pressed: (0x16)

----------------------------
Pressed row: 07
Pressed column: 02
Key Pressed: (0x17)

----------------------------
Pressed row: 00
Pressed column: 03
Key Pressed: (0x18)

----------------------------
Pressed row: 01
Pressed column: 03
Key Pressed: (0x19)

----------------------------
Pressed row: 02
Pressed column: 03
Key Pressed: (0x1a)

----------------------------
Pressed row: 03
Pressed column: 03
Key Pressed: (0x1b)

----------------------------
Pressed row: 04
Pressed column: 03
Key Pressed: (0x1c)

----------------------------
Pressed row: 05
Pressed column: 03
Key Pressed: (0x1d)

----------------------------
Pressed row: 06
Pressed column: 03
Key Pressed: (0x1e)

----------------------------
Pressed row: 07
Pressed column: 03
Key Pressed: (0x1f)
```

```
Pressed row: 00
Pressed column: 04
Key Pressed: (0x20)

----------------------------
Pressed row: 01
Pressed column: 04
Key Pressed: (0x21)

----------------------------
Pressed row: 02
Pressed column: 04
Key Pressed: (0x22)

----------------------------
Pressed row: 03
Pressed column: 04
Key Pressed: (0x23)

----------------------------
Pressed row: 04
Pressed column: 04
Key Pressed: (0x24)

----------------------------
Pressed row: 05
Pressed column: 04
Key Pressed: (0x25)

----------------------------
Pressed row: 06
Pressed column: 04
Key Pressed: (0x26)

----------------------------
Pressed row: 07
Pressed column: 04
Key Pressed: (0x27)

----------------------------
Pressed row: 00
Pressed column: 05
Key Pressed: (0x28)

----------------------------
Pressed row: 01
Pressed column: 05
Key Pressed: (0x29)

----------------------------
Pressed row: 02
Pressed column: 05
Key Pressed: (0x2a)

----------------------------
Pressed row: 03
Pressed column: 05
Key Pressed: (0x2b)

----------------------------
Pressed row: 04
Pressed column: 05
Key Pressed: (0x2c)

----------------------------
Pressed row: 05
Pressed column: 05
Key Pressed: (0x2d)

----------------------------
Pressed row: 06
Pressed column: 05
Key Pressed: (0x2e)

----------------------------
Pressed row: 07
Pressed column: 05
Key Pressed: (0x2f)
```

```
Pressed row: 00
Pressed column: 06
Key Pressed: (0x30)

----------------------------
Pressed row: 01
Pressed column: 06
Key Pressed: (0x31)

----------------------------
Pressed row: 02
Pressed column: 06
Key Pressed: (0x32)

----------------------------
Pressed row: 03
Pressed column: 06
Key Pressed: (0x33)

----------------------------
Pressed row: 04
Pressed column: 06
Key Pressed: (0x34)

----------------------------
Pressed row: 05
Pressed column: 06
Key Pressed: (0x35)

----------------------------
Pressed row: 06
Pressed column: 06
Key Pressed: (0x36)

----------------------------
Pressed row: 07
Pressed column: 06
Key Pressed: (0x37)

----------------------------
Pressed row: 00
Pressed column: 07
Key Pressed: (0x38)

----------------------------
Pressed row: 01
Pressed column: 07
Key Pressed: (0x39)

----------------------------
Pressed row: 02
Pressed column: 07
Key Pressed: (0x3a)

----------------------------
Pressed row: 03
Pressed column: 07
Key Pressed: (0x3b)

----------------------------
Pressed row: 04
Pressed column: 07
Key Pressed: (0x3c)

----------------------------
Pressed row: 05
Pressed column: 07
Key Pressed: (0x3d)

----------------------------
Pressed row: 06
Pressed column: 07
Key Pressed: (0x3e)

----------------------------
Pressed row: 07
Pressed column: 07
Key Pressed: (0x3f)
```

PERDYN11088 Testbench Output

IMPORTANT NOTICE

Peripheral Dynamics reserves the right to change minor performance and/or implementation specifications without notice. Customers are advised to obtain the latest versions of product specifications, which should be considered when evaluating a product's appropriateness for a particular use.

BY USING THIS PRODUCT, CUSTOMER AGREES THAT IN NO EVENT SHALL PERIPHERAL DYNAMICS, ITS PARENT COMPANY, SUBSIDIARIES, OR PARTNERS BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES AS A RESULT OF THE PERFORMANCE, OR FAILURE TO PERFORM, OF THIS PRODUCT. PERIPHERAL DYNAMICS MAKES NO OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

PERIPHERAL DYNAMICS PRODUCTS ARE NOT AUTHORIZED FOR USE IN LIFE SUPPORT DEVICES OR SYSTEMS. Life support devices or systems are those which are intended to support or sustain life and whose failure to perform can be reasonably expected to result in a significant injury or death to the user.