Introduction

This technical note describes the architectural features of the ispMACH® 4000ZE ultra low power devices and how they can be implemented using Lattice’s ispLEVER® Classic design software. The features discussed in this document include:

1. On-chip oscillator and timer
2. Power Guard
3. Individual I/O bus maintenance

On-Chip Oscillator

An internal oscillator and timer is provided for use in miscellaneous housekeeping functions such as watchdog heartbeats, digital de-glitch circuits and control state machines. The oscillator is disabled by default to save power. The oscillator and timer primitive can be used directly in schematic, or instantiated using HDL (ABEL, Verilog, or VHDL). Figure 1 illustrates the functional block diagram of the oscillator and timer.

Figure 1. Oscillator and Timer

![Figure 1. Oscillator and Timer](image)

The software oscillator primitive (OSCTIMER) is shown in Figure 2.

Figure 2. Software Oscillator Primitive

![Figure 2. Software Oscillator Primitive](image)

Table 1. OSCTIMER Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSCOUT</td>
<td>Output</td>
<td>Oscillator out (nominal frequency 5MHz +/- 30%)</td>
</tr>
<tr>
<td>TIMEROUT</td>
<td>Output</td>
<td>Oscillator frequency divided by an attribute TIMER_DIV = (128, 1024, 1048576), default is 128.</td>
</tr>
<tr>
<td>TIMERRES</td>
<td>Input</td>
<td>Resets the timer.</td>
</tr>
<tr>
<td>DYNOSCDIS</td>
<td>Input</td>
<td>Disables the oscillator. Saves AC power</td>
</tr>
</tbody>
</table>

Note: At least one of the two outputs are required.
Advanced Features of the
ispMACH 4000ZE Family

On-Chip Oscillator HDL Usage
Below are Verilog, VHDL, and ABEL definitions of the OSCTIMER. For additional information on designs that use the oscillator, reference examples are included with the ispLEVER Classic design tool.

**Verilog**

OSCTIMER Declaration:

```verilog
module osctimer(DYNOSCDIS, TIMERRES, OSCOUT, TIMEROUT);

parameter TIMER_DIV = "128";

input  DYNOSCDIS;
input  TIMERRES;
output OSCOUT;
output TIMEROUT;

endmodule
```

OSCTIMER Parameter Declaration and Instantiation:

```verilog
defparam I1.TIMER_DIV = "1024",

osctimer I1 (.DYNOSCDIS(osc_dis), .TIMERRES(tmr_rst),
    .OSCOUT(osc_out), .TIMEROUT(tmr_out));
```

**VHDL**

Library Instantiation:

```vhdl
library lattice;
use lattice.components.all;
```

OSCTIMER and Attribute Declaration:

```vhdl
component osctimer
    generic (TIMER_DIV : string);
    port( DYNOSCDIS : in  std_logic;
          TIMERRES : in  std_logic;
          OSCOUT  : out std_logic;
          TIMEROUT : out std_logic);
end component;
```

OSCTIMER Instantiation:

```vhdl
I1: OSCTIMER
    generic map (TIMER_DIV => "1024")
    port map ( DYNOSCDIS => osc_dis,
               TIMERRES => tmr_rst,
               OSCOUT  => osc_out,
               TIMEROUT => tmr_out);
```

end component;
ABEL

Library Instantiation:

   library 'lattice';

OSCTIMER Declaration:

   XLAT_OSCTIMER(DYNOSCDIS, TIMERRES, OSCOUT, TIMEROUT, 128);

OSCTIMER Instantiation:

   I1 OSCTIMER(osc_dis, rst, osc_out, tmr_out);

On-Chip Oscillator in the Constraint Editor

The ispLEVER Classic Constraint Editor includes an OSCTIMER Attribute Sheet. This sheet gives the user the ability to view signals, connected to the oscillator and set the divider value (TIMER_DIV).

Table 2. Oscillator in the Constraint Editor

<table>
<thead>
<tr>
<th>Type</th>
<th>Oscillator Clock</th>
<th>Timer Clock</th>
<th>Oscillator Disable</th>
<th>Timer Reset</th>
<th>Timer Divide</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSCTIMER</td>
<td>TOUT_c</td>
<td>TOUT2_c</td>
<td>RST_c</td>
<td>RST2_c</td>
<td>128</td>
</tr>
</tbody>
</table>

On-Chip Oscillator in the Report File

The OSCTIMER section of the report file includes all connections and properties of the oscillator.

OSCTIMER_Summary

~~~~~~~~~~~~~~

OSCTIMER:   Pin/Node
OSCTIMER Instance Name     I1
Dynamic Disable Signal     osc_RST
Timer Reset Signal         osc_RST2
Oscillator Output Clock    mfb A-15  TOUT_c
Timer Output Clock         mfb G-15  TOUT2_c

Oscillator Output Clock Frequency  5.0000 MHz
Timer Output Clock Frequency    39.0625 KHz
Timer Divider                  128

Power Guard

During the system inactive state, ignoring the CPLD array logic from external input signal changes based on a “system inactive” signal is an excellent way to reduce power and increase battery life. This feature is called Power Guard in the ispMACH 4000ZE family. When all the inputs have Power Guard turned on the power consumption of the device is close to standby current of the device. The Power Guard can be used directly in schematic entry, or instantiated using HDL (ABEL, Verilog or VHDL). A bus with Power Guard can be generated using the Module Manager. Figure 3 shows a detailed architectural view of the I/O cell including the Power Guard block.
The software Power Guard primitive is shown in Figure 4.

**Figure 4. Power Guard Primitive**

**Table 3. Power Guard Signals**

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>INPUT</td>
<td>Signal coming from an input or I/O pad. This pin cannot have any fanout and connects only between the pad and the Power Guard.</td>
</tr>
<tr>
<td>E</td>
<td>INPUT</td>
<td>The “ENABLE” input that is tied to BIE (Block Input Enable). The BIE signal is driven directly from an I/O or through logic such as an OE signal.</td>
</tr>
<tr>
<td>Q</td>
<td>OUTPUT</td>
<td>This is the output of the Power Guard that drives toward the Global Routing Pool. When E=1, the output Q is driven by the input D.</td>
</tr>
</tbody>
</table>
Power Guard HDL Usage
Below are Verilog, VHDL, and ABEL definitions of the Power Guard option. For designs that use Power Guard, refer to the examples included with the ispLEVER Classic design tool.

Verilog

Power Guard Declaration:

```verilog
module PG_example(D, E, Q);

    input  D;
    input  E;
    output Q;

endmodule
```

Power Guard Instantiation:

```verilog
PG I1 (.D(Input_sig),.E(Enable_sig), .Q(Temp_sig));
```

VHDL

Library Instantiation:

```vhdl
library lattice;
use lattice.components.all;
```

Power Guard Declaration:

```vhdl
component PG
port( D : in  std_logic;
    E : in std_logic;
    Q : out  std_logic);
end component;
```

Power Guard Instantiation:

```vhdl
I1: PG
    port map ( D => Input_sig,
               E => Enable_sig,
               Q => Temp_sig);
```

ABEL

Library Instantiation:

```abel
"library 'lattice';
```

Power Guard Declaration:

```abel
XLAT_PG(D, E, Q);
```

OSCTIMER Instantiation:

```abel
I1 PG(in0, ie, d0);
```
Power Guard Module Manager Usage

The Module Manager can be used to generate a bus of Power Guard primitives, which can be instantiated in the target design. The only option is to specify the width of the Power Guard bus. Figure 5 is a screen shot of the Power Guard Module Manager GUI.

Figure 5. Power Guard Module Manager GUI

Power Guard Usage Rules

There are several important guidelines to keep in mind when using Power Guard:

1. Within a block, the Power Guard enables must be connected to the same.
2. If the BIE signal feeds itself, the Fitter will produce an error.
3. If the BIE signal is generated via internal logic and the source of the logic is the Power Guard output, then the Fitter will produce a warning.

Power Guard in the Report File

I/Os which have Power Guard enabled can be identified in the Fitter Report by looking at the Power Guard Enable column. The signal in the Power Guard Enable is the logical net name which drives the Power Guard Enable port. Below is an example of the Fitter Report that includes the Power Guard information in the right-hand column:

<table>
<thead>
<tr>
<th>Pin No</th>
<th>Pin Type</th>
<th>Bank Number</th>
<th>GLB</th>
<th>Assigned</th>
<th>Signal Type</th>
<th>Signal Name</th>
<th>Power Guard Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TDI</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>I/O/OE</td>
<td>0</td>
<td>A5</td>
<td></td>
<td>LVCMOS18</td>
<td>Input a_4</td>
<td>PG_E_node</td>
</tr>
<tr>
<td>3</td>
<td>I/O</td>
<td>0</td>
<td>A6</td>
<td></td>
<td>LVCMOS18</td>
<td>Input rst</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>I/O/OE</td>
<td>0</td>
<td>A7</td>
<td></td>
<td>LVCMOS18</td>
<td>Input a_5</td>
<td>PG_E_node</td>
</tr>
<tr>
<td>5</td>
<td>GNDIO0</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>VCCIO0</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Individual I/O Bus Maintenance

The ispMACH4000ZE I/Os have individual programmable I/O bus maintenance options. The four options for the I/O are programmable pull-up, pull-down, bus keeper and off. The I/O bus maintenance can be set using the ispLEVER Constraints Editor or using the HDL attribute PULL set to either “UP”, “DOWN”, “HOLD”, or “OFF”.

---

---
Advanced Features of the ispMACH 4000ZE Family

Technical Support Assistance
Hotline: 1-800-LATTICE (North America)
          +1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2008</td>
<td>01.0</td>
<td>Initial release.</td>
</tr>
<tr>
<td>April 2008</td>
<td>01.1</td>
<td>Updated On-board Oscillator and Timer diagram. Replaced “I/O Termination” with “Individual I/O Bus Maintenance”.</td>
</tr>
</tbody>
</table>