ProASIC^{PLUS} PLL Dynamic Reconfiguration Using JTAG

Introduction

The ProASIC^{PLUS} family devices provide two clock conditioning circuits. The clock conditioning circuits are located on the east and west sides of the device with PLL cores as the main component of each circuit. The clock conditioning circuits consist of delay and divider blocks to generate the desired outputs from the input reference clock. For more information on the clock conditioning circuits, refer to Actel's *Using ProASIC^{PLUS} Clock Conditioning Circuits* application note.

The delay and divider blocks are configured with default or user-defined values during the instantiation of the PLL macros in the design. The PLL cores can be configured both statically and dynamically. The static configuration is performed during the device programming with the configuration parameters identified in the design. However, the configuration of the PLLs can be modified during the operation of the device. Using PLL dynamic reconfiguration, a user can adjust his/her design's timing performance without the need to reprogram the device. JTAG pins serve as the ports for transferring the configuration data into the PLL cores during device operation. Using JTAG for PLL configuration also saves user I/O pins and logic.

PLL Macro Structure

The clock conditioning circuits on each side of the device contain a MUX architecture that transfers the configuration data into the PLL core. Figure 1 illustrates a simplified block diagram of the MUX architecture in the clock conditioning circuit.

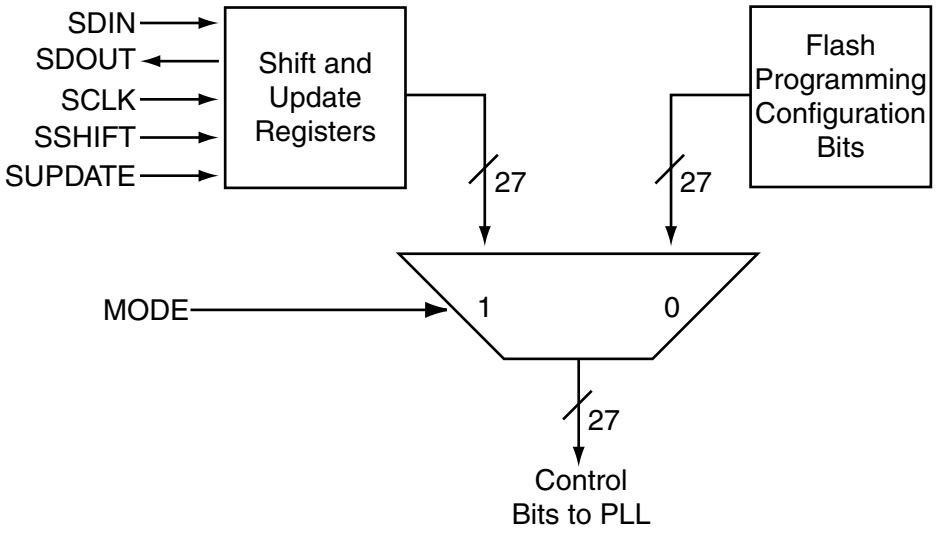
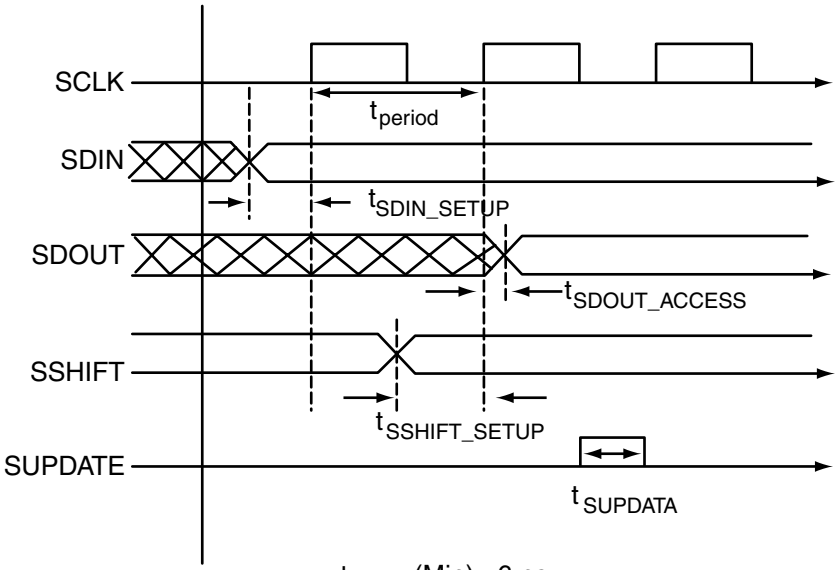


Figure 1 • *The Configuration MUX Architecture*

The control bits configure the PLL core. The control bits are either Flash configuration bits or the bits from the configuration register; the selection is made using the MODE pin shown in Figure 1 on page 1. The Flash configuration bits are the configuration bits associated with programmed Flash switches. These bits are used when the PLL is in static configuration mode. Once the device is programmed, these bits cannot be modified. However, the configuration register can be loaded with new configuration data without reprogramming the device if the MODE pin is at logic "high." There are 27 control bits to configure the different functions of the PLL core. The configuration register can be serially loaded with the new configuration data and programmed into the PLL using the following ports:

- SDIN: The configuration bits are serially loaded into the shift register through this port. The LSB of the configuration data bits should be loaded first. The minimum setup time requirement for SDIN with respect to the shift clock is 3ns. The minimum hold time requirement is 0ns.
- SDOUT: The shift register contents can be shifted out (LSB first) through this port using the shift operation. SDOUT maximum access time is 3ns after the positive edge of the shift clock.
- SCLK: This port should be driven by the shift clock. The maximum allowed frequency of SCLK is slightly higher than 16 MHz.
- SSHIFT: The active-high shift enable signal should drive this port. The configuration data will be shifted into the shift register if this signal is high. Once SSHIFT goes low, the data shifting will be halted. The minimum setup and hold time requirements for the SSHIFT signal are 3ns and 0ns, respectively.
- SUPDATE: The SUPDATE signal is an active-high signal. This signal is used when the shifting is completed, and the user wants to configure the PLL with the new configuration bits.

Figure 2 is the timing diagram for the above ports.



t_{period} (Min)= 6 ns

t_{SDIN_SETUP} (Min)= 3 ns

t_{SDOUT_ACCESS} (Max)= 3 ns

t_{SSHIFT_SETUP} (Min)= 3 ns

$t_{SUPDATA}$ (Min)= 2 ns

Figure 2 • *Timing Diagram*

By toggling SUPDATE, the contents of the configuration register will be latched and available at the input of the MUX architecture. To update the latched data with the contents of the shift register, the SUPDATE signal should be asserted for at least 2ns. The SUPDATE signal MUST be low during any clock cycle where SSHIFT is active. After the assertion of SUPDATE, it must go back to the low state until a new update is required.

Since the contents of the configuration shift register are latched into the input of the MUX, modifying the contents of the shift register via another shifting process will not affect the current configuration of the PLL unless the SUPDATE signal is toggled again.

Figure 3 illustrates the architecture and operation of the configuration shift register.

In order to access the configuration ports of the shift register (SDIN, SDOUT, SSHIFT, etc.), the user should instantiate the PLL macro PLLCORE in his design with the appropriate ports.

Actel recommends that users choose ACTgen to generate PLL macros with the required ports for dynamic configuration. In order to do so, the user should select "Dynamic" in the Configuration section of the ACTgen PLL macro generation GUI. This will generate both the PLL core and the configuration shift register/control bit MUX of Figure 3. The combined structure is shown in Figure 4 on page 4.

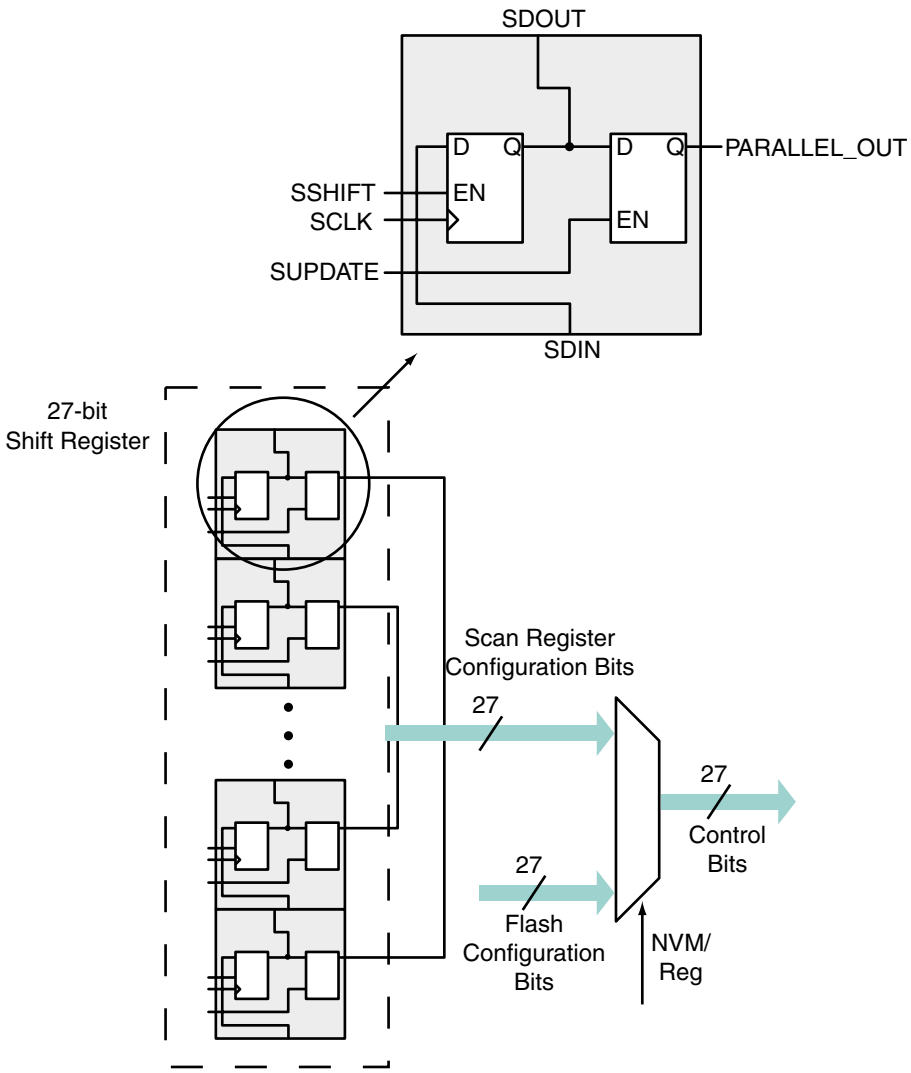


Figure 3 • *Architecture and Operation of the Configuration Shift Register*

Even if dynamic configuration is selected in ACTgen, the user still must specify the static configuration data for the PLL. The specified static configuration is used whenever the MODE signal is set to low and the PLL is required to function in the static mode. The static configuration data can be used as the default behavior of the PLL when required.

However, as specified in the *ProASIC^PLUS Flash Family FPGAs* datasheet, the legal combinations for the delay and the divider block values in the PLL structure are limited. All the legal values for the delay and the divider blocks and the feedback path multiplexing configuration signals are described in Actel's *Using ProASIC^PLUS Clock Conditioning Circuits* application note.

Users must familiarize themselves with the architecture of the PLL core and its input, output, and configuration ports in order to implement the desired delay and output frequency in the PLL structure. Figure 5 shows a simplified model of the ProASIC^PLUS PLL with configurable blocks and switches.

Each group of control bits is assigned a specific location in the 27-bit configuration shift register. Table 1 on page 5 shows the shift register structure.

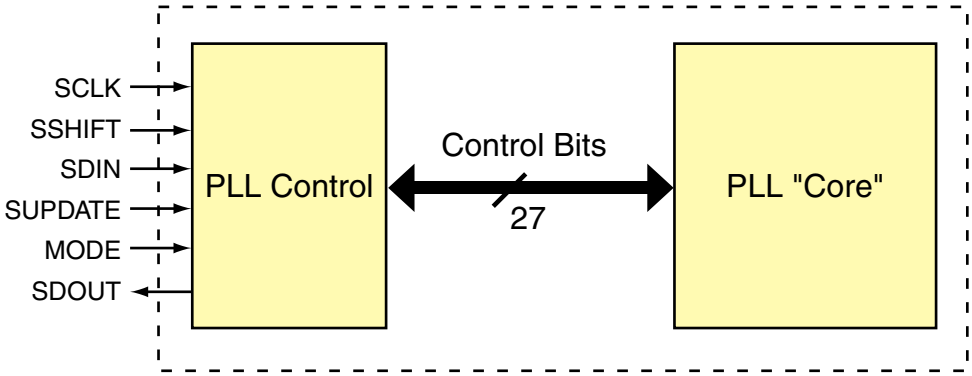


Figure 4 • *Dynamic ACTgen Selection of PLLCORE*

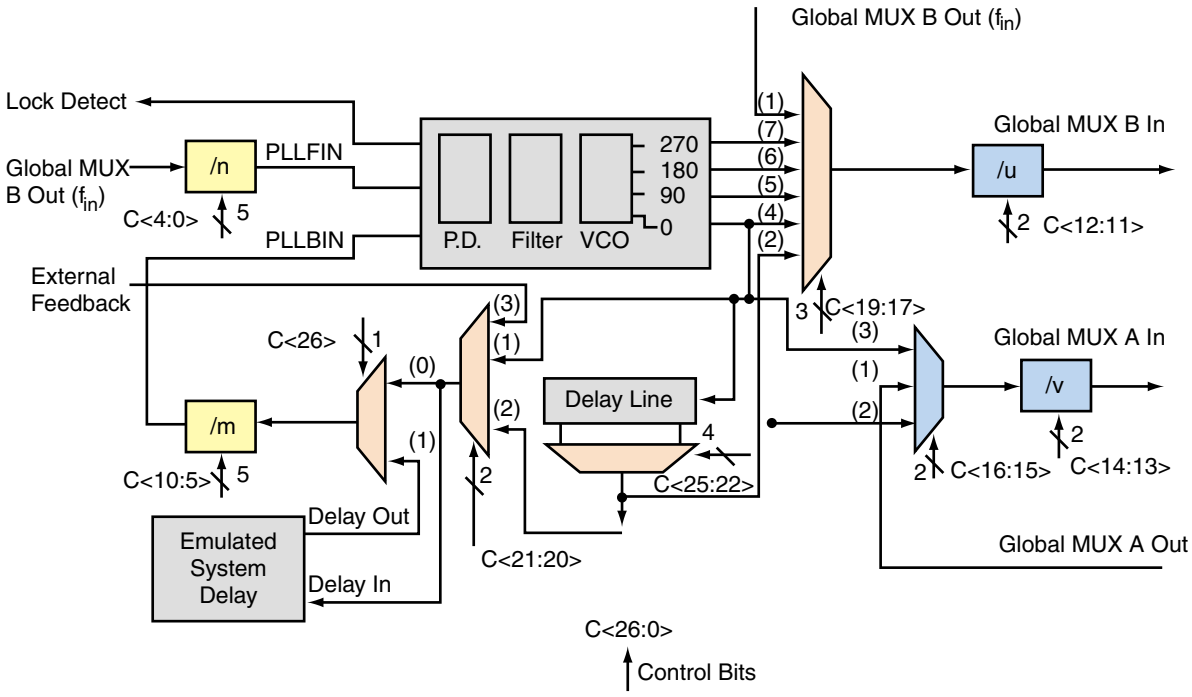


Figure 5 • *PLL Block Configuration via Control Bits*

Loading the Configuration Register

The most important part of PLL dynamic configuration is to load the shift register properly with the configuration bits. There are three different ways to access and load the configuration shift register:

- JTAG Interface
- Logic Core
- Specific I/O Tiles

The JTAG interface, described in the JTAG Interface section, is the most straightforward method. No additional I/O pins are required, since the APA JTAG TAP controller is used to control the loading of the PLL configuration shift register. If the logic core is employed, the user must design a module to provide the configuration data and control the shifting and updating of the PLL configuration shift register. In effect, this is a user-designed TAP controller, which requires additional chip resources. If specific I/O tiles are used for configuration, the user must provide the external equivalent of a TAP controller. This does not require additional core resources but does use pins.

Table 1 • *Configuration bits in the Shift Register*

Configuration Bits	Signal	Description
26	XDLYSEL	Mask-programmable delay select MUX
25-22	FBDLY[3:0]	Delay line tap select MUX
21-20	FBSEL[1:0]	Feedback source MUX
19-17	OBMUX[2:0]	"B" output MUX
16-15	OAMUX[1:0]	"A" output MUX
14-13	OADIV[1:0]	"A" output divider (/v)
12-11	OBDIV[1:0]	"B" output divider (/u)
10-5	FBDIV[5:0]	Feedback signal divider (/m)
4-0	FINDIV[4:0]	Input clock driver (/n)

JTAG Interface

ProASIC^PLUS devices provide a user-interface macro between the JTAG pins and the device core logic. This macro is called UJTAG. A user should instantiate the UJTAG macro in his design in order to access the configuration register ports via the JTAG pins. Figure 6 shows the UJTAG block and its ports.

The UJTAG Interface

The ProASIC^PLUS family provides an 8-bit instruction register, and most of these opcodes are available to users. The UJTAG interface provides the current instruction register value from the TAP and a number of data path control signals. Using these signals, simple user-defined logic can allow JTAG access to internal design registers (including the PLL configuration shift register). Note that the logic required for controlling the PLL requires about 17 tiles (excluding the instruction decoding).

The TDI, TMS, TDO, TCK, and TRST ports of UJTAG are only provided for design simulation purposes and should be treated as external signals in the design netlist. However, these ports should NOT be connected to any input or output buffers in the netlist. Table 2 on page 6 indicates all the UJTAG ports and their descriptions.

The URSTB will be asserted at power up, and a power-on-reset signal resets the TAP. The URSTB will stay asserted until the TAP is accessed externally.

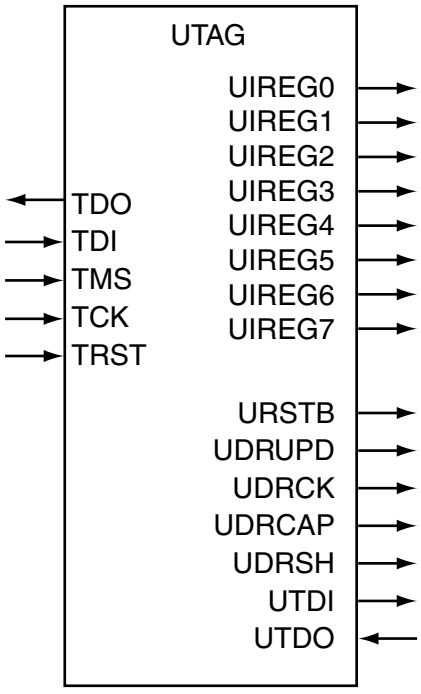


Figure 6 • *UJTAG Macro Block*

Actel recommends designers use a global segment for UDRCK to avoid skew problems (For more information on global spines in Flash products please refer to the *Efficient Use of ProASIC Clock Trees* application note).

Shifting the Configuration Data

In order to shift the configuration data into the PLL configuration shift register, data should be transferred from the user JTAG (UJTAG) block into the PLL block. For appropriate data transfer from the ProASIC PLUS TAP into the PLL, a user-defined interface block should be implemented in the design. The JTAG ports should be connected to the TDI, TDO, TMS, TCK, and TRST inputs of the UJTAG interface. The outputs of the UJTAG interface should be connected to the interface block. This allows the JTAG TAP controller to control the appropriate data transfer from UJTAG into the PLL.

One of the UJTAG output ports is UIREG [7:0]. This 8-bit bus carries the contents of the JTAG Instruction Register. The user-defined interface block must always decode the instruction register for the 'loading configuration data' instruction (this opcode can be defined by the user and can be any value between 16 and 127 inclusive, the rest are reserved codes), and the user's interface block should remain synchronous with the TAP controller and PLL macros. Figure 7 on page 8 shows the interconnection of the TAP controller with the UJTAG and the PLL macro.

Table 2 • UJTAG Interface Port Descriptions

UJTAG Port	Description	Notes
UIREG[7:0]	TAP Instruction Register contents	Current instruction
UTDI	TAP TDI	
UTDO	User TDO output	Sent to TAP TDO output MUX when IR is in user range
UDRSH	Data shift enable	High in Shift-DR TAP state
UDRUPD	Data latch update	High in Update-DR TAP state
URSTB	TAP reset	Low in Test-Logic-Reset TAP state
UDRCAP	Data capture enable	High in Capture-DR TAP state
UDRCK	TCK	

As previously mentioned, the opcode for PLL dynamic configuration is user selectable. As an example, consider the following sequence of opcodes:

Dynamic: 21Hex (Switch PLL to dynamic mode)

LOAD_PLL_CONFIG: 20Hex (Start Configuring the PLL configuration shift register)

Figure 8 on page 8 illustrates an example of a simple user interface implementation for configuring the PLL dynamically.

The following is a sample of HDL code to implement the design in Figure 8 on page 8:

Verilog

```
`define Load_PLL_Config 8'h20

`define Dynamic         8'h21

module pllint (UIR, UTDO, UTDI, UDRSH, UDRUPD,
UDRCAP, UDRCK, URSTB,

         SDIN, SDOUT, SSHIFT, SCLK, SUPDATE,
MODE);

input [7:0] UIR;

input UTDI, UDRSH, UDRUPD, UDRCAP, UDRCK,
URSTB;

output UTDO;

output SCLK, SUPDATE, SSHIFT, SDIN, MODE;

input  SDOUT;

reg SHREG;

assign SDIN = UTDI;

assign SUPDATE = (UIR == `Load_PLL_Config) &&
UDRUPD;

assign SSHIFT  = (UIR == `Load_PLL_Config) &&
UDRSH;

assign SCLK    = UDRCK;

reg UTDO;

always @(posedge UDRCK)

begin

  if (UDRCAP)

  begin : captureDR

    if (UIR == `Dynamic)

      SHREG <= MODE;

    else

      SHREG <= 0;

  end else if (UDRSH)

  begin : shiftDR

    SHREG <= UTDI;

  end

end

always @(UIR or SHREG or SDOUT)

begin : tdomux
```

```
  case (UIR)
     'Load_PLL_Config:  UTDO = SDOUT;
     default:           UTDO = SHREG;
  endcase
end
LDC md (.Q(MODE), .CLR(!URSTB), .D(SHREG),
.EN(UDRUPD && (UIR == 'Dynamic)));
endmodule
```

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
-- Entity Declaration
entity pllint is
  generic (Load_PLL_Config: std_logic_vector
:= x"20"; Dynamic: std_logic_vector :=
x"21");
  port (UIR: in std_logic_vector(7 downto 0);
        UTDI, UDRSH, UDRUPD, UDRCAP, UDRCK,
URSTB: in std_logic;
         SDOUT: in std_logic;
         SCLK, SUPDATE, SSHIFT, SDIN, MODE:
out std_logic;
         UTDO: out std_logic);
end pllint;
-- Architecture Definition
architecture rtl of pllint is
  signal SHREG, t_MODE, temp_en: std_logic;
  component LDC
    port (Q : out std_logic;
          CLR : in std_logic;
          EN : in std_logic;
          D : in std_logic);
  end component;
begin
  SDIN <= UTDI;
  SUPDATE <= UDRUPD when (UIR =
Load_PLL_Config) else '0';
 SSHIFT <= UDRSH when (UIR = Load_PLL_Config)
else '0';
  SCLK <= UDRCK;
  MODE <= t_MODE;
  temp_en <= UDRUPD when (UIR = Dynamic) else
'0';
captureDR: process (UDRCK)
begin
  if (UDRCK'event and UDRCK = '1') then
    if (UDRCAP='1') then
      if (UIR = Dynamic) then
        SHREG <= t_MODE;
      else
        SHREG <= '0';
      end if;
    elsif (UDRSH='1') then
      SHREG <= UTDI;
    end if;
  end if;
end process captureDR;
tdomux: process (UIR, SHREG, SDOUT)
begin
  case UIR is
    when Load_PLL_Config => UTDO <= SDOUT;
    when others => UTDO <= SHREG;
  end case;
end process tdomux;
md: LDC port map (Q => t_MODE,
                  CLR => NOT URSTB,
                  D => SHREG,
                  EN => temp_en);
end rtl;
```

More functionality (additional opcodes) can be added to the TAP controller. For example, by using additional opcodes, the user can configure each PLL individually.

PLL Behavior During Configuration

The output of the configuration register is latched into the PLL core. Therefore, the configuration of the PLL will not change while shifting the new configuration data into the register. In other words, the PLL configuration will not be changed in the dynamic mode unless the UPDATE signal is activated.

The PLL macro can switch between static and dynamic modes. This provides a fixed initial state (static mode), defined by Flash-programmed bits, for the user. For example, in case of any functional error in dynamic configuration, the PLL can always be reset to the static configuration mode. Switching between the static and dynamic modes is done using the MODE signal. When MODE is high, the PLL will be in the dynamic mode.

Once the PLL macro receives a pulse on its SUPDATE port, the new configuration bits will be loaded into the PLL core. If the PLL is already locked with the previous configuration

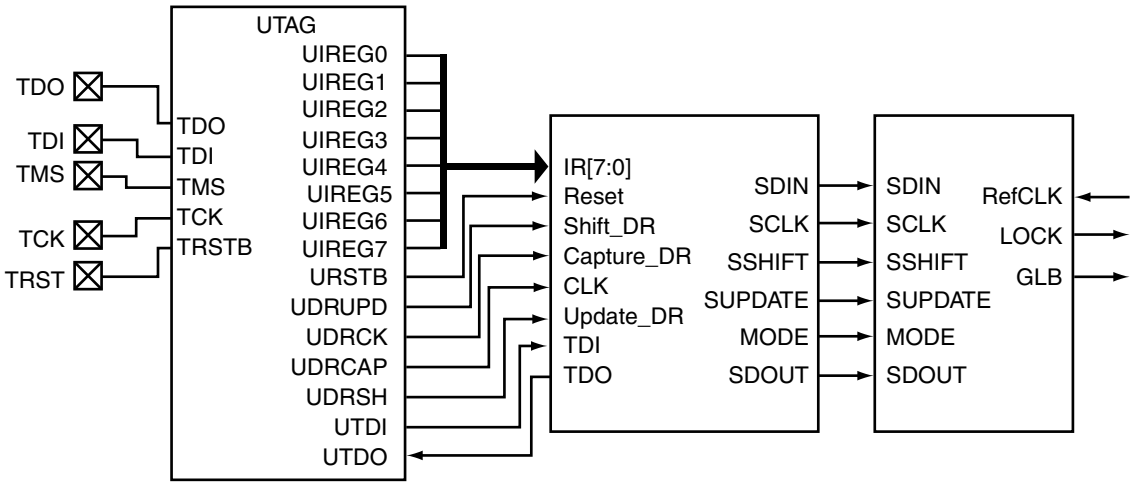


Figure 7 • *PLL and UJTAG Interfacing via User Interface Block*

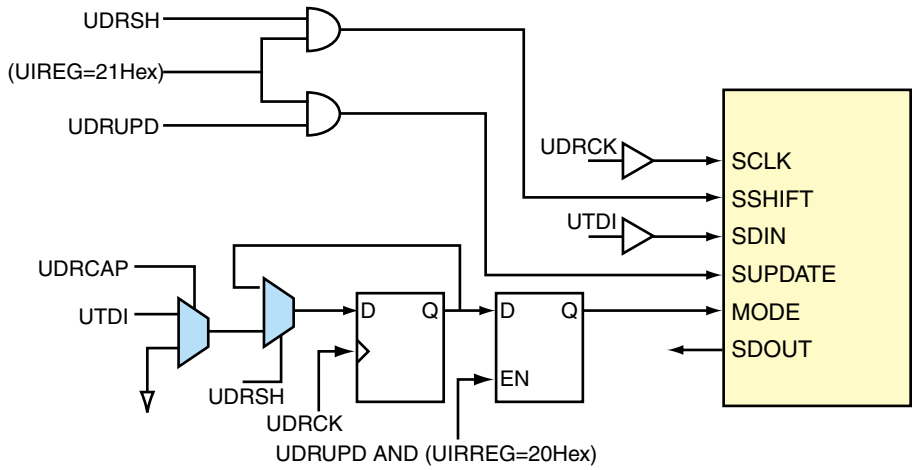


Figure 8 • *Sample Implementation of User Interface*

data, the LOCK output of the PLL will not go low when updating the configuration bits. In other words, once the PLL is locked for the first time, the LOCK output will remain high during dynamic configuration. If the user needs to use the LOCK signal as an enable input to the core logic to disable the design while the PLL is unlocked, the PLL should be reset first and then reconfigured. To reset the PLL, the configuration shift register should be serially loaded with "reset" inserted in the configuration word. Then the user needs to pulse the SUPDATE signal to load the configuration latches and to start the reset operation. Setting bits 21 and 20 (FBSEL [1:0]) of the shift register to "00" and loading the latches using UPDATE will set the PLL in reset operation. After reset, the original configuration word is reloaded into the shift register and another SUPDATE pulse is used to load the configuration latches and to end the reset operation. The user should note that a reset operation is NOT required for dynamic configuration. It is only used if the user needs to deactivate the LOCK output during each reconfiguration.

Conclusion

The ProASIC^PLUS family devices provide two PLL blocks. The PLL blocks can be configured during device operation, thus eliminating the need for completely reprogramming the device. The dynamic configuration bits are loaded into the serial-in/parallel-out shift register, provided in the clock conditioning circuit of each PLL. The contents of this shift register are latched into the PLL block. The best approach to serially load the configuration shift register is via a TAP controller. The JTAG ports can be used along with the user JTAG interface (UJTAG macro) to load the configuration shift register externally.

Using the dynamic configuration capability of the ProASIC^PLUS family PLLs helps designers adjust their chip-level and board-level timing performance based on changing operating conditions.

http://www.actel.com

Actel Europe Ltd.
Maxfli Court, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Tel: +44 (0)1276 401450
Fax: +44 (0)1276 401490

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
USA
Tel: (408) 739-1010
Fax: (408) 739-1540

Actel Asia-Pacific
EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Tel: +81-(0)3-3445-7671
Fax: +81-(0)3-3445-7668

5192719-0/8.02