Revision History

Version	Author	Date	Changes
1.0	Altera Corporation	4/18/06	Initial version
1.1	Altera Corporation	5/22/06	Switched to dffeas

Table of Contents:

1. Overview

It is assumed that the reader of this document is familiar with the Stratix II and Cyclone II architectures and documentation.

Since Stratix II, lcell registers have been modeled separately from the combinational portion. Starting with Titan and Barracuda, we are pulling registers out of IO cells too. This is done mainly to improve IO timing analysis. A common register primitive will be used to model registers found in lcells and IO cells, which makes it unnecessary to destroy and create new atoms when we move registers between lcells and IO cells to improve performance during placement.

The existing Quartus primitive, *dffeas*, will be enhanced so it can be used to model WYSIWYG registers for Titan and Barracuda, as well as future new device families. Compared to the Stratix II lcell_ff wysiwyg and the Cyclone II lcell_ff wysiwyg, *dffeas*'s input and output port names are more well-known and compact, which should help users who are familiar with ASIC timing analysis tools such as PrimeTime and Quartus II's TimeQuest. Having a family-independent register primitive also greatly reduces modeling efforts.

Since registers that come from different blocks and different device families tend to use different set of secondary signals, family-aware and placement-aware legality checking is used to make sure that a register that is placed in a certain type of block or register-packed with a given block can be physically implemented on the actual hardware.

This document is a summary of the common register primitive *dffeas*.

2. Common Register Primitive

```
dffeas <primitive_name>
(
      .d(<register data source>),
      .clk(<clock source>),
      .clrn(<asynchronous clear source>),
      .prn(<asynchronous preset source>),
      .ena(<clock enable source>),
      .asdata(<register asynch/synch data source>),
      .aload(<asynchronous load source>),
      .sclr(<synchronous clear source>),
      .sload(<synchronous load source>),

      .q(<output0>)
);

defparam <primitive name>.power_up = <register power up mode>;
defparam <primitive name>.is_wysiwyg = <treat this primitive as
      WYSIWYG>;
```

2.1 Register Input Ports

<*primitive name*>**:** is the unique identifier for the register cell. This is any identifier name which is legal for the given description language (e.g. Verilog, VHDL, AHDL, etc.). This field is required.

.d(<*register data source*>**):** designates the data input to the register. It defaults to GND if unspecified.

.clk(<*clock source*>**):** designates the clock input to the register. This port is required if any of the following ports are used: .d, .sclr, .sload, .asdata, .ena. It defaults to GND if unspecified.

.clrn(<*asynchronous clear source*>**)** designates the asynchronous clear signal. This signal is active low and defaults to VCC when not specified.

.prn(<*asynchronous preset source*>**)** designates the asynchronous preset signal. This signal is active low and defaults to VCC when not specified.

.ena(<*clock enable source*>**),** designates the clock enable signal. Defaults to VCC when unspecified.

.asdata(<*asynchronous/synchronous load data source*>**),** designates the asynchronous or synchronous data signal for the logic cell. When .aload is high then the register is asynchronously loaded from .asdata. When .sload is high then the register is synchronously loaded from .asdata. Hence .asdata must be connected if .aload or .sload is connected. The signal defaults to VCC when unspecified.

.aload(<*asynchronous load source*>**),** designates the asynchronous load signal. When asserted the value of the .asdata port is applied to the register. Hence .asdata must be connected if .aload is connected. Defaults to GND when not specified.

.sclr(<*synchronous clear source*>**),** designates the synchronous clear signal for the logic cell. Defaults to GND if unspecified.

.sload(<*synchronous load source*>**),** designates the synchronous load signal for the logic cell. When .sload is high then the register is synchronously loaded from .asdata. Hence .asdata must be connected if .sload is connected. Defaults to GND when unspecified.

2.2 Register Output Ports

.q (<*output*>**)** is the output of this register.

2.3 Register Modes

<*power_up*> is one of *{high, low, dont_care}* and describes the power-up condition of the register. This field is optional and defaults to *dont_care*. Note that I/O registers are the only ones that have true *power_up* control, i.e., without need to do not-gate-pushback. If the register's *power_up* setting can not be honored for some reason, a warning message should be given to inform the user that the setting has been ignored.

<*is_wysiwyg*> is one of *{true, false}* and indicates whether this register primitive should be treated as WYSIWYG. This field is optional and defaults to *false*. If it is set to *true*, then no further optimizations will be done on this register by Quartus II and legality rules will be applied to the register to make sure the signals specified can be supported by the targeted architecture. For example, no emulation will be done to handle true asynchronous load connections; rather, a user error message will be issued whenever a true asynchronous load is detected.

2.4 Register Input Polarities and Default Values

Table 1 describes the polarity and programmable inversion ability of all the inputs. Signals which can be programably inverted can be provided in either polarity.

Table 1 -- Polarity of Register Inputs

Signal	*Polarity*	*Programmable Inversion*
.clk	Rising Edge	Yes
.d	Positive Logic	No
.clrn	Active Low	Yes
.prn	Active Low	Yes
.ena	Active High	Yes
.asdata	Positive Logic	No
.aload	Active High	Yes
.sclr	Active High	Yes
.sload	Active High	Yes

2.5 Register Control Signal Priority

There are several different signals that can cause the register to change its stored value. If more than one of these signals is active (logic 1), the highest priority signal determines the stored register state. The priority of these signals is described in Table 2. If none of these signals are active on a given cycle, the normal d input to the register sets the stored register value at the rising clock edge.

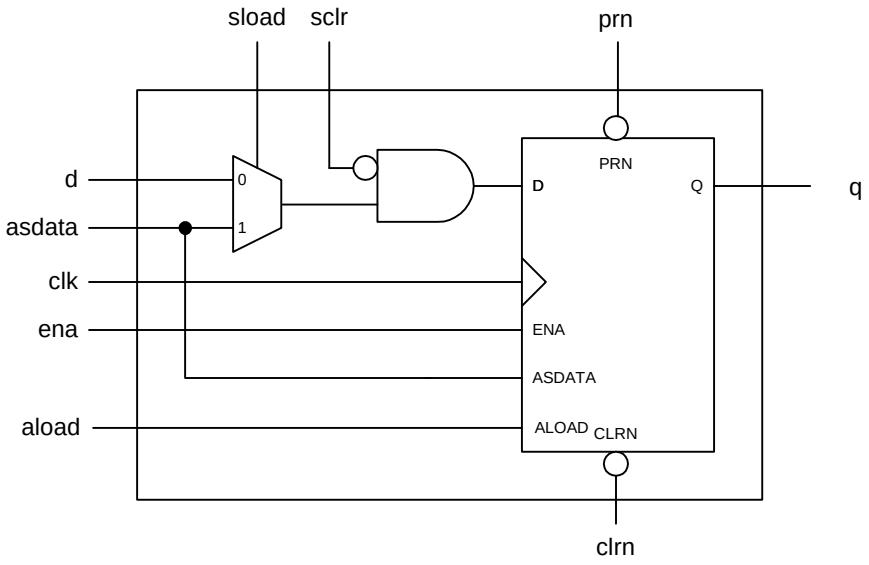
For example if clrn and sload were both asserted on the same clock cycle the clrn port would take precedence and clear the register since it has the first priority.

Table 2 -- Register Control Signal Priority

Signal	*Priority*
.clrn	1
.prn	2
.aload	3
.ena	4
.sclr	5
.sload	6

2.6 Register Block Diagram

Figure 1 – Register Primitive



2.7 Block Specific Constraints (Titan)

This section describes detailed constraints on a Titan register cell if it is placed in a specific block (LAB, LUTRAM, IO) or register packed with a specific block (MEAB, M-RAM, DSP).

2.7.1 LAB

The logic cell register in Titan can use all the secondary signals **except asynchronous load**. The asynchronous load capability of the Stratix II register has been removed.

2.7.2 RAM

2.7.2.1 LUTRAM

Input and output registers for LUTRAM are implemented with LAB registers so they have the same restrictions as the ones described in 2.7.1.

2.7.2.2 MEAB and M-RAM

Output registers contained in these blocks can not use asynchronous load, or synchronous load, asdata, or synchronous clear signals.

2.7.3 I/O

The I/O registers can not have true asynchronous load or synchronous load, but they can do preset both asynchronously and synchronously. We use asdata=1 to represent this functionality.

Asynchronous clear needs to be represented by the clrn port; synchronous clear needs to be represented via the sclr port. asdata=0 is not allowed.

Simultaneous use of asynchronous clear and preset are not supported. Similarly, simultaneous use of synchronous clear and preset are not supported either.

2.7.4 DSP

DSP registers can only have clock, clock enable, and asynchronous clear controls.

2.8 Block Specific Constraints (Barracuda)

This section describes detailed constraints on a Barracuda register cell if it is placed in a specific block (LAB, IO) or register packed with a specific block (MEAB, Multiplier).

2.8.1 LAB

The logic cell register in Barracuda can use all the secondary signals except asynchronous load.

2.8.2 MEAB

Output registers contained in these blocks can not use asynchronous load, or synchronous load, asdata, or synchronous clear signals.

2.8.3 I/O

The I/O registers can not have true asynchronous load or synchronous load, but they can do preset both asynchronously and synchronously. We use asdata=1 to represent this functionality.

Asynchronous clear needs to be represented by the clrn port; synchronous clear needs to be represented via the sclr port. asdata=0 is not allowed.

Simultaneous use of asynchronous clear and preset are not supported. Similarly, simultaneous use of synchronous clear and preset are not supported either.

2.8.4 Multiplier

Registers can only have clock, clock enable, and asynchronous clear controls.