Build a FIFO "Dipstick" With a CY7C371 CPLD

Programmable FIFO flags can simplify the design of a digital system. They do it by automatically indicating a status that can prevent overrun or underrun in an elastic FIFO buffer. Although many FIFOs are available with on-chip programmable flag functions, such features are not available on industry standard asynchronous FIFOs. And of those FIFOs that do have programmable flags, some do not allow the almost-empty and almost-full values to be programmed independently—or, in some cases, for the values to be programmed to any specific word boundary.

To get around the problem, this article gives you a method by which FIFOs of any size may be monitored by an external CPLD (complex programmable logic device) that generates all the flags necessary for most FIFO applications. This FIFO "dipstick" CPLD is, in effect, a measuring device that observes the level of data within a FIFO.

The design presented here lets you implement programmable flags for any size FIFO by simply changing values in its VHDL description. It's also easy to adapt a microprocessor port for applications that require dynamically alterable flags. And the design is adaptable to different FIFO applications as well, such as clocked FIFOs, FIFOs with asynchronously clocked ports, BiFIFOs, etc.

How to do it

A variable-length up-down counter is implemented in VHDL, to measure the exact level of data within a FIFO with asynchronously clocked ports. The number of bits required for the dipstick counter is dependent on the size of the FIFO; that is, FIFO depth = 2n , where *n* is the number of counter bits required.

For example, a 2K FIFO would require an 11-bit counter. The *n*th bit is necessary to prevent the dipstick counter from rolling over to zero when the last byte is written into the FIFO. In other words, the *n*th bit will be set only when the FIFO is completely full.

Due to the asynchronous nature of the read and write ports of a FIFO, it's necessary to implement a state machine to control the operation of the dipstick counter. The state machine must resolve the overlapping and nesting conditions that may occur with the FIFO_READ_L and FIFO_WRITE_L signals to the FIFO. For instance, multiple read pulses may occur within a single write pulse, read and write pulses may occur simultaneously, or read and write pulses may overlap by any amount of time.

The status of the almost-full and almost-empty flags is determined by simply comparing the dipstick counter value to preprogrammed levels and generating the appropriate combinatorial outputs. This method allows for the generation of any flag outputs required for a given application. The almost-full and almost-empty flags are the most typical levels required; they are used to determine greater-than-or-equal-to and less-than-or-equal-to specified levels, respectively. Many possibilities exist, however, such as an approximately-half-full flag, which could be used to add hysteresis to a FIFO's half-full value.

Synchronous FIFO

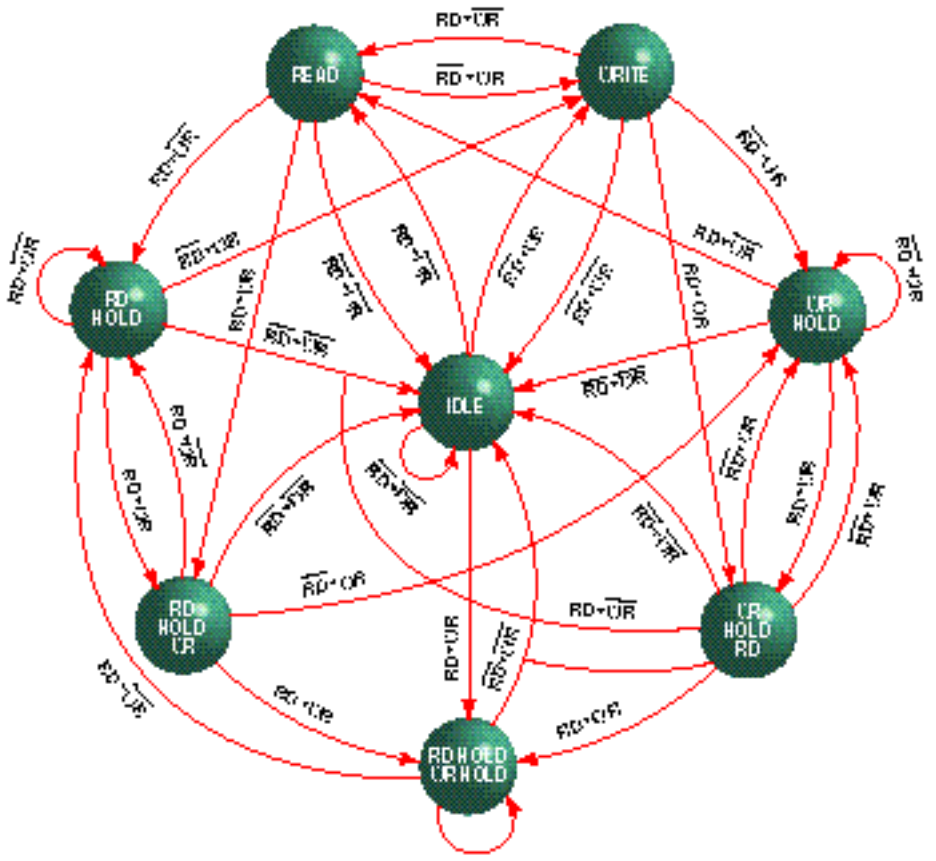
The VHDL implementation of the dipstick design, which uses a Cypress FLASH370™ CPLD, assumes that clocked circuitry controls both the read and write ports of the FIFO, and that the clocks for each port are synchronous to each other. The assumptions allow use of a single clock for the state machine and counter, and allow the read, write, and reset inputs to be used without any chance of a metastable event occurring.

A synchronous implementation such as this means that the "almost" flags will change state combinatorially within three clock cycles after the clock cycle that initiated the read or write. For instance, if a FIFO read is held active for two clock cycles followed by one clock cycle for read-recovery time, the updated almost-empty flag will be available during the read-recovery cycle.

Asynchronous FIFO

The read and write ports of a FIFO may be controlled by clocked circuitry with clocks that are asynchronous to each other. In this situation, the state machine and counter should be controlled by the one clock that best suits the application. That is, if it's required that the write port of the FIFO receive the almost-full flag immediately, then the write port clock should be used. If the read port of the FIFO is to receive the almost-empty flag immediately, the read port clock should be used.

In either case, the read or write input from the opposite port needs to be synchronized to the dipstick's clock before it is used as a state machine input. The CY7C371 is ideally suited for this because of its dedicated inputs, which can be configured as single- or double-registered, and a guaranteed 10-year MTBF. In addition, the port that is asynchronous to the dipstick's clock must also synchronize the almost-flags before use, to prevent metastability problems. A negative aspect of using the FIFO dipstick in this application is that additional delays are introduced be-tween a FIFO access and the "almost" flags status change. These additional delays may or may not be tolerable, depending on the application.



Under state-machine control. This eight-state finite state machine (FSM) observes the FIFO_READ_L and FIFO_WRITE_L inputs. Four states are counter-enabled; of these, two are count-up states (WRITE, RD_HOLD_WR) and two are count-down states (READ, WR_HOLD_RD). Three states are counter-disabled (RD_HOLD_WR_HOLD, RD_HOLD, WR_HOLD) and are needed for the FIFO_READ_L and FIFO_WRITE_L pulses, which are active for more than one clock cycle. Each state evaluates all four permutations of FIFO_READ_L and FIFO_WRITE_L to determine the next state. If neither signal is active, the FSM returns to the idle state. If a single signal goes active or stays active, the machine progresses to the appropriate state such that the counter-enabled states are active for a single clock cycle only, during each FIFO_READ_L and FIFO_WRITE_L pulse to the FIFO. If the latter signals go active on the same clock cycle, the FSM will avoid the counter-enabled states, thereby allowing the dipstick counter to remain constant. Because the dipstick counter remains cleared if FIFO_RESET_L is active, this signal is not required as an input to the state machine.

Warp2 VHDL implementation

The VHDL design used for the FIFO dipstick is completely behavioral. This high-level design methodology eliminates any need to describe device-specific implementations and it also allows for the most readability. Cypress's *Warp2* VHDL compiler will automatically synthesize the design into the low-level components necessary for a CY7C371.

Dipstick flags vs. FIFO flags

You should be aware of two differences between the FIFO dipstick de-sign and the use of a FIFO with programmable flags. First, the latency incurred between a FIFO access and the update of flag status may be prohibitive (see the Synchronous and Asynchronous FIFO Ports sections above). Second, the flag outputs of a FIFO will always go inactive based on a FIFO strobe going inactive, whereas the FIFO dipstick solution will always change flag states based on the strobes going active. ❖

For literature, visit the Cypress web site. See the appropriate site address (URL) for article 206 in the listing on the back cover.

Recent Articles

In the first quarter of 1996, Cypress engineers published a number of articles in the electronics press worldwide. They covered a range of interests, as the following sampling shows.

Programmable logic

"EPROMs No Flash in the Pan," by Naushik Desai; *Electronic Buyers' News*, 15 Jan.

"Heaven's Gate," by Al Graf; *Elrad* (Ger.), January.

"Save Logic in Magnitude Comparators," by Chris Jones; *Electronic Design*, 8 Jan.

"FPGA Configures Simple BCD Adder," by Chris Jones; *EDN*, 15 Feb.

"High-Performance Ethernet System Design Using FPGAs," by Greg Somer; *Elektronik Journal* (Ger.), 16 Feb.

"A Designer's Guide to VHDL Design and Verification," by Kevin Skahill; *Electronic Design*, 19 Feb.

DataCom / Logic

"Serializing Parallel Buses," by Craig Rich; *Communication System Design*, January.

"Reducing Jitter in PLL-Based Systems," by Eric Chan; *Selezione di Elettronica* (Italy), February.

"Faster Data Transfer Over a Crossbar," by David Horton; *New Electronics* (UK), March.

"How to Implement Your Own RACEway Interface," by David Horton; *VMEbus Systems*, Spring issue.

SRAMs / Cache

"Advantages of Cache Integration," by Mathew Arcoleo; *EDN*, 5 Jan.

"Modernize Your Memory Subsystem," by Paul Novell, Raymond Leong, and David Barringer; *Electronic Design*, 5 Feb.

"A Chipset with Integrated Cache," by Hideharu Miki and Kikuo Mita; *Transistor Technology* (Japan), March.