On-Chip-RAM erhöht die Logikkomplexität in FPGA (Teil 2)

Steven K. Knapp, Xilinx und Jutta Ramatschi, Metronik, München

152 Wie man RAM-Funktionen in unterschiedlichen FPGA realisiert, hängt von der Architektur des Logik-Grundbausteins ab. In den Baustein der XC4000-Serie von Xilinx wird ein RAM-Bereich mit demselben Look-up-Table aufgebaut, der auch die logischen Funktionen definiert. Jeder Logikblock der XC4000-Serie stellt entweder zwei 16x1-RAM oder ein 32x1-RAM (bzw. dementsprechend auch zwei 16x1-PROM oder 32x1-PROM) bereit.

Design Beispiele

Einige Schaltungsbeispiele können am ehesten verdeutlichen, wie kleine, über den Chip verteilte RAM-Funktionen anstelle von Flip-Flops die Logik einfach, weniger komplex machen. Das erste Design, das im vorigen Heft der Elektronik Informationen (H. 7/91) erläutert wurde, ist ein einfaches Schieberegister, das zweite, ebenfalls in Heft 7, ist ein Zähler-Array. Im dritten und vierten Beispiel wird gezeigt, wie man FIFO und LIFO mit Hilfe von Single-Port-RAM aufbaut. Das letzte Beispiel zeigt, wie man ein RAM als Register-File in einem High-speed DMA-Controller einsetzen kann.

Beispiel 3: First-In/First-Out-Speicher

Das On-Chip-RAM eignet sich auch hervorragend zum Aufbau von FIFO-Speichern. FIFO werden zur zeitlichen Abstimmung von Systemdaten eingesetzt. Man bezeichnet sie auch als elastische Speicher. FIFO werden dort eingesetzt, wo Systeme, die mit verschiedenen Taktraten arbeiten, für eine Datenübertragung aneinander angepaßt werden müssen. Auf der einen Seite werden Daten in das FIFO geschrieben, auf der anderen Seite werden diese Daten wieder ausgelesen, wobei diese zwei Funktionen mit verschiedenen Taktraten arbeiten. Da die beiden Taktfrequenzen unterschiedlich sind, muß permanent überwacht werden, ob der FIFO-Speicher voll oder leer ist. Sollte das FIFO leer sein, dann darf nicht mehr gelesen werden, und falls das FIFO voll ist, muß mit Schreib-Operationen so lange ausgesetzt werden, bis durch zwischenzeitliche Lese-Vorgänge wieder Platz im FIFO frei geworden ist. Ein typischer Anwendungsfall für einen FIFO-Speicher ist ein Datenpuffer für Drucker, auch Spooler genannt. Um zu vermeiden, daß ein schneller Systemprozessor auf den Drucker warten muß, bis dieser jedes einzelne Schriftzeichen gedruckt hat, werden die Druckdaten mit der hohen Taktgeschwindigkeit des Prozessors in den Zwischenspeicher (das FIFO) abgelegt und vom Drucker mit seiner geringeren Geschwindigkeit ausgelesen. Der Prozessor gewinnt so wertvolle Zeit für andere Aufgaben.

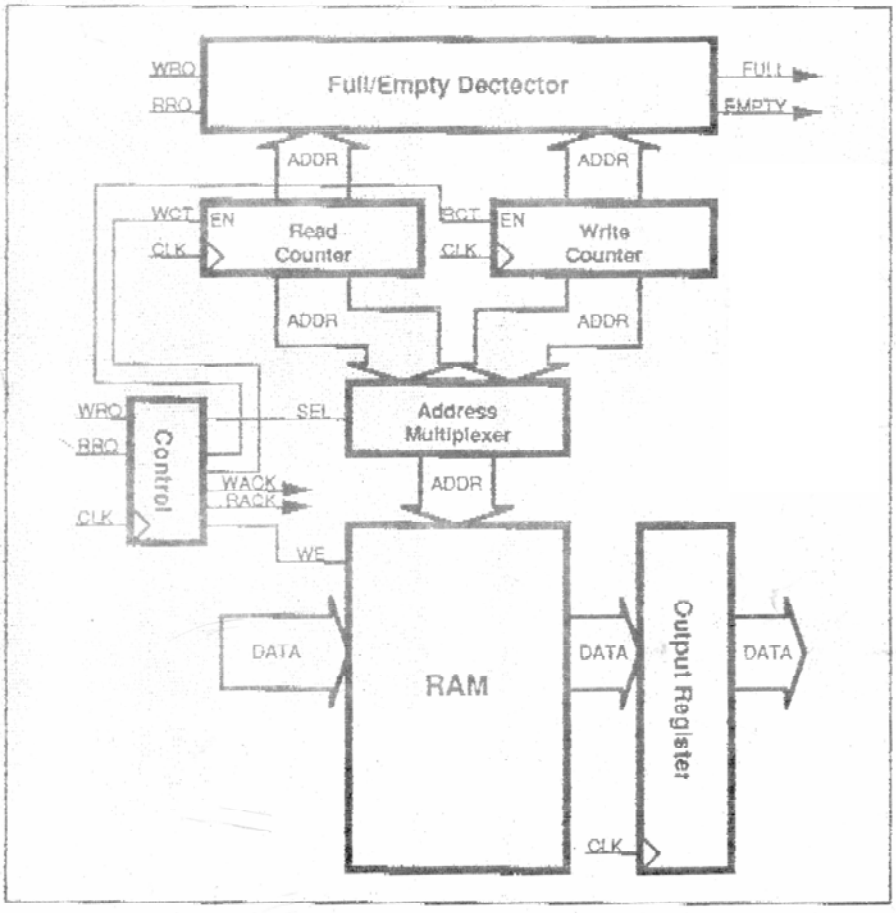

Bild 5. Der knifflige Teil dieser FIFO-Schaltung ist das Verhalten bei gleichzeitigem Schreib- und Lese-Request.

Normalerweise haben diese Druckerspeicher eine Speichertiefe von einigen 1000 Byte und können nur schwerlich in einen einzigen Chip integriert werden. Kleinere Pufferspeicher der FIFO allerdings finden oft Anwendung in der Telekommunikation. Sie sind meist nur 64x1 bis 64x9 Bit organisiert. Diese kleineren FIFO bieten sich geradezu dafür an, auf einem Chip - wie z.B. einem LCA - realisiert zu werden. Will man ein FIFO mit Single-port-RAM aufbauen, so muß man deren Struktur auf dual-port Architektur ändern. In einem Single-port-RAM wird zum Lesen und Schreiben der Daten ein einziges Adreß-Port verwendet; in einem Dual-port-Speicher, wie z.B. einem FIFO, gibt es dafür zwei getrennte Ports. Um einen Single-port-Speicher zu einem Dual-port-Speicher zu machen, müssen die beiden Adreß-Ports des Dual-port-Speichers zu einem einzigen Adreß-Port gemultiplext werden **(Bild 5)**. Die meisten Block. Nach dem Schreiben erhöht der Controller den Write-Counter auf die nächste Adresse. Ein Lesezugriff läuft nach dem gleichen Schema ab. Der knifflige Teil die-

ses Designs ist die Abstimmung bei gleichzeitigen Schreib- und Lesezugriffen. Da die Adreßleitungen des RAM gemultiplext sind, ist eine paralleler Zugriff nicht möglich; sie können nur nacheinander abgearbeitet werden. Kommen Anforderungen zum Schreiben und Lesen von Daten gleichzeitig an, so stellt die FIFO-Kontrollogik den Schreibvorgang zurück, um zuerst das Lesen komplett durchzuführen **(Bild 6)**. Beide Zugriffs-Anforderungen werden mit der ersten fallenden Flanke des Taktsignals erkannt. Der Controller löst einen Lesevorgang aus, und die Daten werden im Ausgangsregister abgelegt. Das Lesen dauert einen Taktzyklus. Da noch ein Schreibvorgang zu erledigen ist, ignoriert der Controller sämtliche neuen Lese- oder Schreibzugriffe, die bei der nächsten fallenden Taktflanke anliegen. Stattdessen initiiert er einen Schreibvorgang und speichert die Daten des FIFO. Ist das FIFO leer und gibt es gleichzeitig Schreib- und Lesezugriffs-Anforderungen, so wird der Schreibvorgang abgearbeitet, da ja noch keine Daten zum Lesen vorhanden sind. Das FIFO kann natürlich auch so konstruiert werden, daß nach dem Schreibvorgang der Lesezyklus zunächst noch ausgesetzt wird. Ob das Lesen oder das Schreiben den Vorrang bekommen, hängt gewöhnlich davon ab, welcher Vorgang das FIFO schneller füllt oder leert. Ein 32x16-FIFO belegt 33 Logikblöcke eines XC4000-Bausteins, 16 davon für die RAM-Funktion. Fünf Blöcke übernehmen die Kontrollfunktionen, neun die Lese/Schreib-Adreßzähler sowie den Adreßmultiplexer, und drei CLB realisieren die FIFO-voll/leer-Erkennung.

Beispiel 4: Last-In/First-Out Speicher

LIFO-Funktionen sind auch weitverbreitet, so z.B. als Stack im Mikroprozessor. Daten werden in den Stack geschrieben, wobei

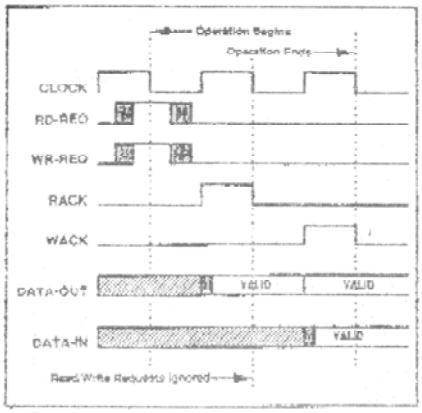

Bild 6. Bei gleichzeitigem Schreib- und Lese-Request muß das FIFO die beiden Vorgänge nacheinander bearbeiten.

das zuletzt angekommende Wort über einen Auf/Ab-Adreßzähler als erstes wieder gelesen wird. Der Aufbau eines LIFO ist einfacher als der eines FIFO. Normalerweise laufen Schreib- und Lesevorgänge in einem LIFO mit der gleichen Taktrate ab. Damit entfällt bei LIFO die komplizierte Abstimmungs-Logik der FIFO. Die einzige Schwierigkeit in einem LIFO-Design ist zu entscheiden, wann der Zähler nach oben oder nach unten zählen soll. In dem Design, das in **Bild 7** gezeigt wird, wird der Adreßzähler nach jedem Schreibvorgang erhöht und vor jedem Lesezugriff verringert. Damit schiebt jeder Schreibvorgang den Zähler auf die nächste RAM-Adresse, bis alle Bereiche beschrieben sind. Danach wird jede weitere Schreibanforderung ignoriert. Ein 16x16-Stack-Speicher belegt insgesamt nur 14 Logikblöcke. Acht für den Speicher, drei für den Adreßzähler und je einen für die Kontrollogik, die Speicher-voll- und die Speicher-leer-Erkennung.

Beispiel 5: Register-File in einem DMA-Controller

Das On-Chip-RAM kann auch als Register-File verwendet werden **(Bild 8)**. So wird in einem 16-kanaligen 32-Bit-DMA-Controller das RAM zum zeitlichen Multiplexen von Adreß- und Wortzählern benutzt. Length-

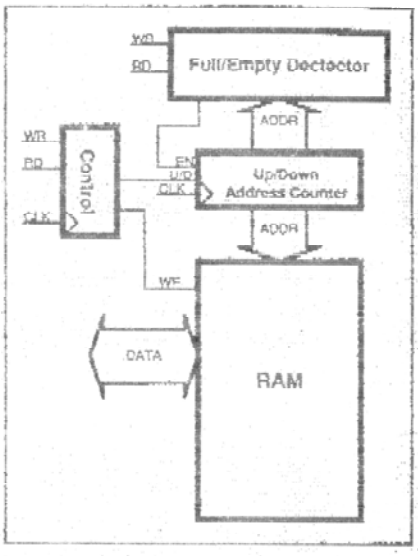

Bild 7. LIFO sind einfacher in der Kontrollogik als FIFO.

count und die Basisadresse der 16 Kanäle sind im RAM wie in einem Register-File gespeichert. In einem XC4000-Baustein belegt der 32x16-Speicher 16 Logikblöcke. Einige Bit des Basisadreßspeichers dienen dazu, Umfang (8-, 16- oder 32-Bit) und Art (Lese- oder Schreibvorgang) des Datentransfers festzulegen. Das 35x16-RAM für die Basisadreßdaten belegt 18 Logikblöcke. Mit Hilfe des On-Chip-RAM kann der DMA-Controller überlappende Datenübertragungen durchführen, d.h. eine gerade laufende Datenübertragung kann durch eine Übertragung mit höherer Priorität unterbrochen werden. Der Wort- und Adreßwert der unterbrochenen Datenübertragung wird ins RAM abgelegt und später zur Fortsetzung der ursprünglichen Übertragung wieder geladen. In diesem Beispiel ist eine Autoinitialisierung nicht vorgesehen, sie kann aber recht einfach hinzugefügt werden. Damit werden nach Abschluß der Datenübertragung der ursprüngliche Lengthcount und die Basisadresse wieder neu geladen. Hierfür verdoppelt man Lengthcount und Basisadresse. Dadurch benötigt man zwar die doppelte Anzahl an Logikblöcken (68 statt 34), doch in einem XC4005 mit typ. 5000 Gatterfunktionen hat der Anwender insgesamt 196 Logikblöcke.

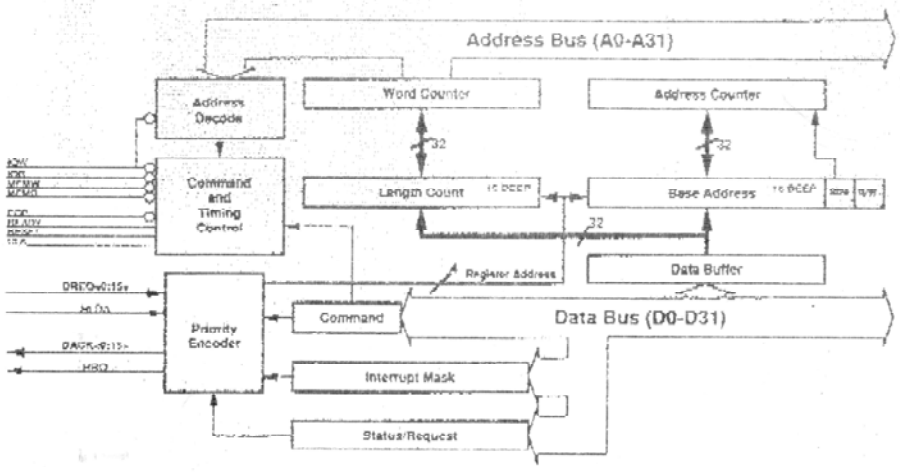

Bild 8. On-Chip-RAM wird als Register-File verwendet.