Vollsynchrone Mehrfunktionsregister

Mehrfunktionsregister können verschiedene Funktionen ausführen (Daten parallel übernehmen, in verschiedene Richtungen schieben, zählen usw.). Beim heutigen Stand der Technik wird eine vollsynchrone Arbeitsweise bevorzugt. Der Takt läuft durch. Alle Wirkungen sind mit kombinatorischen Verknüpfungen zu implementieren, die den Steuer- oder Dateneingängen der Flipflops vorgeschaltet sind[1]. Im allgemeinen Fall sind mehrere Funktionen F1, F2, F3 usw. vorgesehen, z. B. F1 = paralleles Laden, F2 = Linksschieben, F3 = Vorwärtszählen usw.

Mehrfunktionsregister mit D-Flipflops

Sind n Funktionen zu implementieren, werden jedem Flipflop ein Multiplexer oder Datenselektor mit n + 1 Eingängen vorgeschaltet (Abb. 1). Der zusätzliche Eingang dient der Selbsthaltung. An die verbleibenden n Eingänge sind die jeweiligen funktionellen Verbindungen und Netzwerke angeschlossen (Dateneingänge, Ausgänge benachbarter Flipflops (zum Schieben), Zählnetzwerke usw.). Ist keine Funktion auszuführen, so ist die Selbsthaltung zu aktiveren.

Vorrangregeln

Vorrangregeln sind mit Zusatzbeschaltungen[2] zu implementieren. Manchmal ist es von Vorteil, die kombinatorischen Verknüpfungen mit Schaltnetzen zu erledigen, die aus 2-zu-1-Multiplexern aufgebaut sind (Abb. 2). Dann ergibt sich die Vorrangregelung aus der Reihenfolge der Multiplexer. Je näher der Multiplexer dem Flipflopeingang ist, desto höher ist die Priorität des Auswahlsignals.

Mehrfunktionsregister mit DE-Flipflops

Werden DE-Flipflops eingesetzt, so entfällt die Rückführung (da sie schon im Innern der Flipflops vorgesehen ist). Statt dessen ist der Erlaubniseingang CE dann zu aktivieren, wenn eine Funktion auszuführen ist (z. B. über eine disjunktive Verknüpfung der Funktionsauswahlsignale; Abb. 3).

1: Es ist nicht zulässig, den Takt zu beeinflussen oder asynchrone Eingänge auszunutzen. Derartige vollsynchron wirkende Register ergeben sich auch, wenn man die Funktionsweise mit einer Verhaltensbeschreibung erfaßt (Verilog, VHDL usw.)

2: Das sind zumeist konjunktive Verknüpfungen mit den invertierten Signalen, die jeweils mit Vorrang wirken sollen (Inhibition).

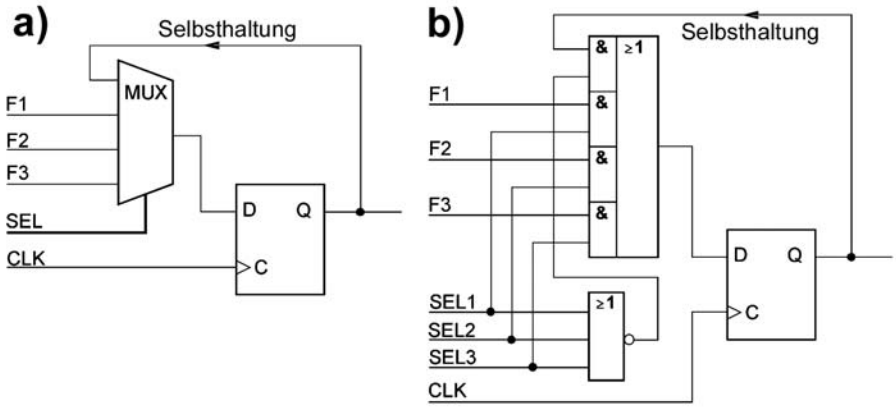


Abb. 1 Vollsynchrone Mehrfunktionsregister. a) mit Multiplexer (binär codierte Funktionsauswahl); b) mit Datenselektor (Funktionsauswahl 1 aus n). Die Funktionssignale F1, F2 usw. können Eingänge sein, aber auch Ausgänge anderer Flipflops oder Ausgänge von Verknüpfungsschaltungen, beispielsweise von Zählnetzwerken. Nähere Erläuterung im Text.

a) Funktionsauswahl über Multiplexer. Sind die SELECT-Eingänge mit Nullen belegt, so geschieht nichts, und die Datenbelegung bleibt erhalten (Selbsthaltung). Ansonsten wählt jede SELECT-Belegung eine bestimmte Funktion aus.

b) Funktionsauswahl über Datenselektor. Die Auswahleingänge SEL1, SEL2 usw. werden im 1-aus-n-Code angesteuert. Sind alle Auswahleingänge mit Nullen belegt, so wird über das NOR-Gatter die Rückführung aktiviert (Selbsthaltung).

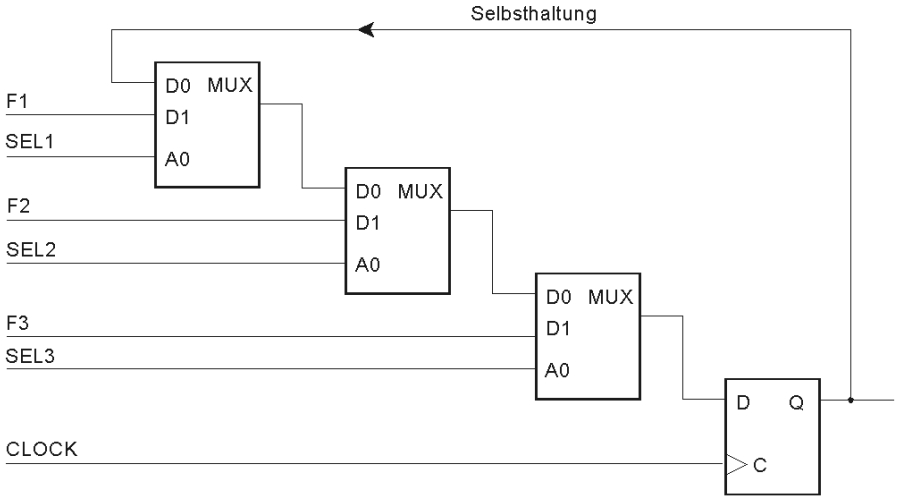


Abb. 2 Vollsynchrones Mehrfunktionsregister mit Vorrangsteuerung. Die Funktionen werden mit 2-zu1-Multiplexern ausgewählt. Je näher ein Multiplexer dem Flipflopeingang ist, desto höher ist die Priorität der jeweils ausgewählten Funktion. Im Beispiel hat F3 die höchste und F1 die niedrigste Priorität.

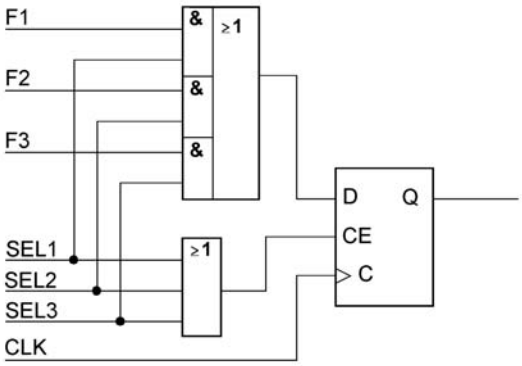
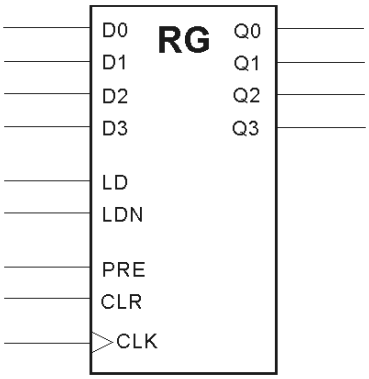


Abb. 3 Vollsynchrones Mehrfunktionsregister mit DE-Flipflops. Das Erlaubnissignal CE ist dann zu aktivieren, wenn eine Funktion auszuführen ist.

Entwurfsbeispiel

Es ist ein vollsynchrones 4-Bit-Register gemäß Abb. 4 zu entwerfen. Grundlage: D-Flipflops und beliebige Gatter.



Signal	Funktion
LD	Laden
LDN	Laden invertiert
PRE	Setzen (alle Stellen = 1)
CLR	Löschen (alle Stellen = 0)
–	Datenbelegung halten

Abb. 4 Die Entwurfsaufgabe.

Zunächst werden keine Vorrangregeln berücksichtigt. Es sind vier Funktionen. Demzufolge benötigen wir 4 + 1 = 5 UND-Gatter, denen eine ODER-Verknüpfung nachzuschalten ist (Abb. 5). Jede Funktion hat ein Steuersignal. deshalb ist nichts zu decodieren. Alle Steuersignale werden an ein NOR-Gatter angeschlossen. Dessen Ausgang dient dazu, die Rückführung auszuwählen (Selbsthaltung).

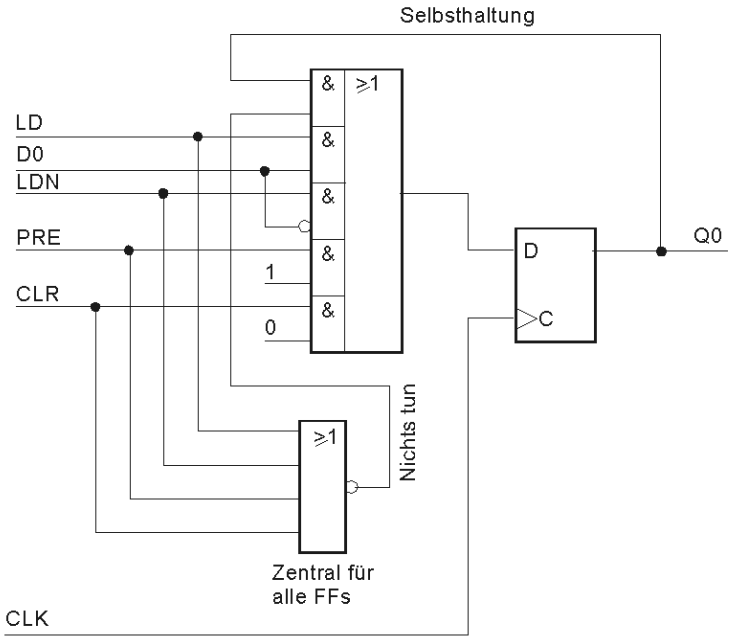


Abb. 5 Die erste Implementierung. Es ist Bitposition 0 dargestelöt. Alle anderen Bitpositionen sind gleichartig aufgebaut. Damit ist die Aufgabe eigentlich gelöst. Alles Weitere erledigt das Entwicklungssystem.

Zwei Vereinfachungsmöglichkeiten sind offensichtlich. Die UND-Verknüpfung des Steuersignals PRE ist nicht erforderlich, und das Steuersignal CLR muß gar nicht an die ODER-Verknüpfung angeschlossen werden (Abb. 6).

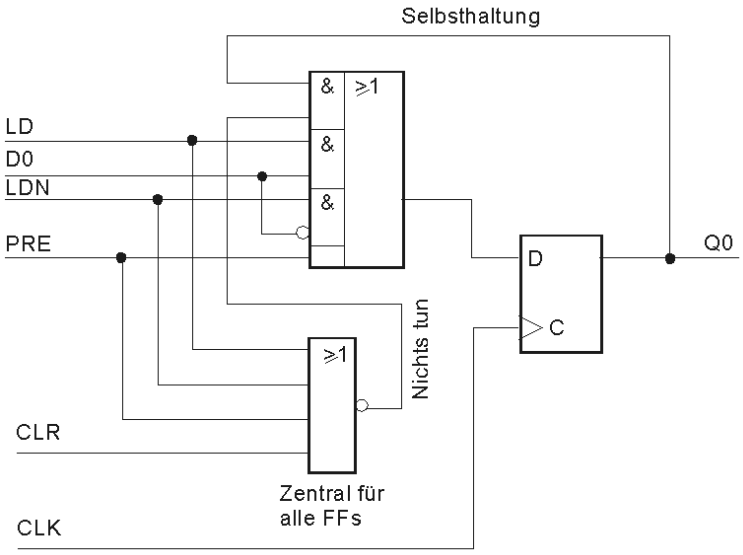


Abb. 6 Eine vereinfachte Implementierung.

Nun sollen die folgenden Vorrangregeln berücksichtigt werden:

- CLR dominiert über alle anderen Funktionen.
- CLR und PRE dominieren über die Ladefunktionen.
- LD dominiert über LDN.

Hierzu werden die UND-Verknüpfungen um die entsprechenden Inhibitionen erweitert (Abb. 7).

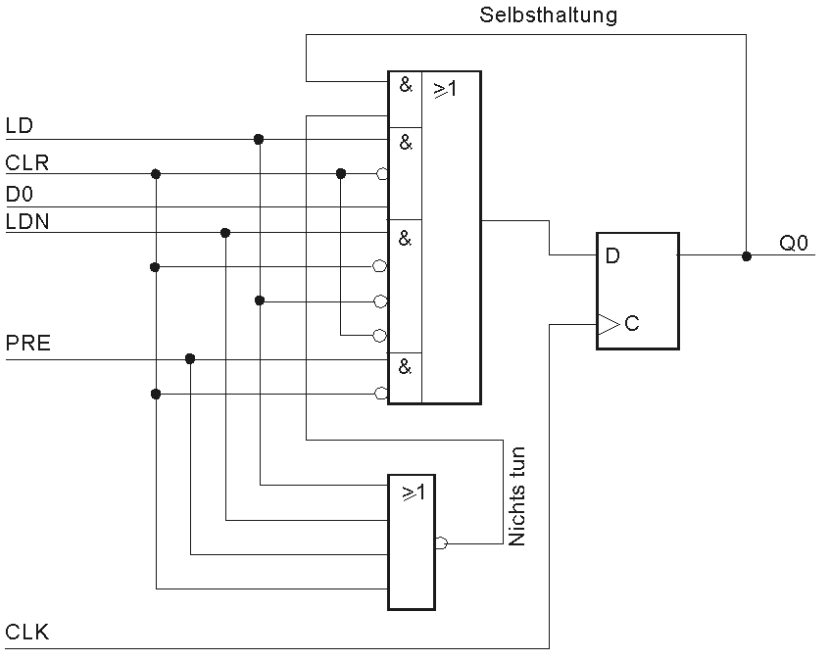


Abb. 7 So können die Vorrangregeln implementiert werden. Alle Signale, die inaktiv sein müssen, wenn die jeweilige Funktion ausgeführt werden soll, werden jeweils invertiert an die betreffenden UND-Gatter angeschlossen[3].

3: Im Beispiel kann darauf verzichtet werden, PRE invertiert auf die UND-Gatter der Signale LD und LDN einweirken zu lassen, weil die Preset-Funktion das Flipflop immer setzt und damit die Ladefunktionen einflußlos sind (Don't Cares). Diese Vereinfachung ist in Abb. 7 bereits eingeführt.

Das Beispiel als Verhaltensbeschreibung in Verilog (mit Vorrangregelung):

```
module registerbeispiel_01(Q, D, LOAD, LDN, PRE, CLR, CLK);

output [3:0] Q;
input [3:0] D;
input LOAD, LDN, PRE, CLR, CLK;
reg [3:0] Q;

always @ (posedge CLK)

    begin

        casex ({LOAD, LDN, PRE, CLR})

                4'bxxx1: Q <= 0;        // Löschen

                4'bxx10: Q <= 15;       // Setzen

                4'b1x00: Q <= D;        // Laden

                4'b0100: Q <= ~D;       // Negiert laden
        endcase

    end

endmodule
```

Dieses Schema kann beliebig abgeändert werden. Im folgenden werden die Steuersignale beibehalten, die Funktionen in der Case-Anweisung aber geändert.

a) Vorwärts und rückwärts zählen

```
        casex ({LOAD, LDN, PRE, CLR})

                4'bxxx1: Q <= 0;        // Löschen

                4'bxx10: Q <= Q + 1;    // Vorwärts

                4'b1x00: Q <= D;        // Laden

                4'b0100: Q <= Q - 1;    // Rückwärts
        endcase
```

b) Nach links und rechts schieben

```
casex ({LOAD, LDN, PRE, CLR})
        4'bxxx1: Q <= 0;            // Löschen
        4'bxx10: Q <= Q << 1;       // Linksschieben
        4'b1x00: Q <= D;            // Laden
        4'b0100: Q <= Q >> 1;       // Rechtsschieben
endcase
```

c) Nach links und rechts rotieren

```
casex ({LOAD, LDN, PRE, CLR})
        4'bxxx1: Q <= 0; //Löschen
        4'bxx10:                    // Linksrotieren
            begin
                Q[0] <= Q[3];
                Q[1] <= Q[0];
                Q[2] <= Q[1];
                Q[3] <= Q[2];
            end
        4'b1x00: Q <= D;             // Laden
        4'b0100:                    // Rechtsrotieren
            begin
                Q[0] <= Q[1];
                Q[1] <= Q[2];
                Q[2] <= Q[3];
                Q[3] <= Q[0];
            end
endcase
```