Übung 05

5.1.1 SR-Latch in VHDL

Eine der grundlegendsten sequentiellen Schaltungen ist das *SR-Latch*, dessen Schaltbild und Wahrheitstabelle in Abbildung 1 bzw. in Tabelle 1 dargestellt sind. Die Funktionsweise des SR-Latches kann aus dem Schaltbild und der Wahrheitstabelle abgelesen werden, insgesamt ergeben sich hierbei vier verschiedene Fälle:

- Fall I: R=1, S=0

  Wenn R den logischen Wert 1 besitzt, wird das NOR-Gatter N_1 die Ausgabe 0 am Ausgang Q erzeugen. Da sowohl Q als auch S den logischen Wert 0 haben, produziert das NOR-Gatter N_2 die Ausgabe 1 am Ausgang $\overline{Q}$.

- Fall II: R=0, S=1

  Wenn S den logischen Wert 1 besitzt, wird das NOR-Gatter N_2 die Ausgabe 0 am Ausgang $\overline{Q}$ erzeugen. Da sowohl $\overline{Q}$ als auch R den logischen Wert 0 haben, produziert das NOR-Gatter N_1 die Ausgabe 1 am Ausgang Q.

- Fall III: R=1, S=1

  Wenn R und S den logischen Wert 1 besitzen, produzieren die NOR-Gatter N_1 und N_2 die Ausgabe 0 an beiden Ausgängen Q und $\overline{Q}$.

- Fall IV: R=0, S=0

  Wenn R und S den logischen Wert 0 besitzen, sind die Ausgaben von N_1 und N_2 abhängig von den vorherigen Werten von Q und $\overline{Q}$, die als Q_{prev} und $\overline{Q}_{prev}$ angegeben sind.

Die Signale S und R des SR-Latches können also verwendet werden, um die Ausgabe Q (und dessen Negation $\overline{Q}$) zu setzen (*set*) und zu löschen (*reset*). Q wird auf den Wert 0 gesetzt wenn an R der logische Pegel 1 angelegt wird. Wenn an S der logische Pegel 1 angelegt wird, wird Q auf den Wert 1 gesetzt. Ein SR-Latch gleichzeitig setzen und löschen

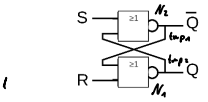


Abbildung 1: Schaltbild eines SR-Latches.

Fall	S	R	Q	$\overline{Q}$
IV	0	0	Q_{prev}	$\overline{Q}_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

Tabelle 1: Wahrheitstabelle eines SR-Latches.

zu wollen ist nicht sinnvoll, in diesem Fall (wenn also an beiden Eingängen S und R der logische Pegel 1 angelegt wird) befindet sich der Wert 0 an beiden Ausgängen Q and $\overline{Q}$. Dieser Zustand wird auch *irregulärer Zustand* genannt. Wenn an beiden Eingängen S und R der logische Pegel 0 angelegt wird, speichert das SR-Latch die vorherigen Werte von Q and $\overline{Q}$, also Q_{prev} und $\overline{Q}_{prev}$.

Aufgaben:

a. (2 Punkte) Sie haben in Übung 2 gelernt, wie man ein Logikgatter baut (z.B. AND, NAND, OR, NOT). Entwerfen Sie mit Hilfe dieser Bausteine ein *SR*-Latch mit Logikgatter gemäß der Wahrheitstabelle 1, indem Sie diesen in VHDL implementieren. Verwenden Sie zur Überprüfung Ihrer Implementierung die vorgegebene Testbench mit allen 4 Eingabe-Kombinationen von S und R (00, 01, 10 und 11).

b. (2 Punkte) Für Latches hängt das Verhalten nicht nur von den Eingaben ab, sondern auch von dem vorhergehenden Zustand. Erweitern Sie die Testbench um weitere Eingabe-Kombinationen um das Verhalten des SR Latches zu untersuchen, z.B. indem Sie Eingaben in anderer Reihenfolge ausführen. Beachten Sie dabei besonders die Eingabe 11 die zu dem *irregulärer Zustand* führt, beschreiben Sie wann dieser zu Problemen führt und wie sich dies auf die Simulation bzw auf die Ausgaben in GTKWave auswirkt.

c. (1 Punkte) Erklären Sie die Nachteile beziehungsweise allgemeinen Probleme von SR-Latches.

b. - Das Problem besteht in der Kombination von R=S=1 und anschließendem setzen von S=R=0.
- Hier liegt der kritische Zustand vor, da entweder das obere NOR-Gatter oder das untere NOR-Gatter nach deren Schnelligkeit entschieden werden kann, was Q bzw. $\overline{Q}$ für Ausgangssignale erhalten
- Hier ist also in der Simulation das Verhalten nicht deterministisch, also stoppt unsere Simulation "simulation stopped @ 107 ns -- stop-delta = 5000 \n" nach Überschreitung einer Anzahl von Delta-Zyklen und GTKWave zeigt den letzten stabilen Zustand an bei 107 ns.

c. Nachteile/Probleme: - Verhalten hängt von der Schaltzeit der Gatter ab
Je nachdem welches der NOR-Gatter schneller ist, kann am Ausgang ein nicht erwünschtes Ausgangssignal anliegen
also kann am Ausgang temporär ein instabiler Zwischenzustand anliegen
- $Q = \overline{Q} = 0$ ergibt logisch keinen Sinn ist daher irregulär/undefiniert für S=R=1
- kein Takt: jeglich anliegendes Eingangssignal wird sofort verarbeitet

5.1.2 D-Latch in VHDL

Ein weiterer häufiger Latch-Typ ist das *D-Latch*, das im Grunde genommen eine Erweiterung des SR-Latches ist. Wie schon das SR-Latch besitzt das D-Latch zwei Dateneingänge, wenn auch mit anderer Bedeutung: einen Dateneingang D und einen Takteingang CLK (für *clock*). Der Dateneingang D bestimmt den nächsten Ausgabewert des Latches, wohingegen der Takteingang kontrolliert, wann sich die Ausgabe ändern soll. Das Schaltbild und die Wahrheitstabelle sind in Abbildung 2 und Tabelle 2 dargestellt. Unter Zuhilfenahme dieser kann nachvollzogen werden, wie das D-Latch funktioniert:

Wenn an CLK der Logikpegel 0 anliegt, erzeugen die beiden AND-Gatter den Wert 0 an ihren Ausgängen und somit auch an S und R des verwendeten SR-Latches, was bedeutet, dass das D-Latch in diesem Fall die vorherigen Werte Q_{prev} und $\overline{Q}_{prev}$ unabhängig vom an D anliegenden Wert speichert. Wenn an CLK der Logikpegel 1 anliegt, wird der an D anliegende Wert gespeichert.

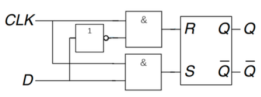


Abbildung 2: Schaltbild eines D-Latches.

CLK	D	Q	$\overline{Q}$
0	x	Q_{prev}	$\overline{Q}_{prev}$
1	0	0	1
1	1	1	0

Tabelle 2: Wahrheitstabelle eines D-Latches.

Aufgaben:

a. (2 Punkte) Implementieren Sie, basierend auf der Wahrheitstabelle 2 und Figure 2, ein D-Latch in VHDL. Verwenden Sie dabei einige der von Blatt 2 bekannten Logikgatter und den in Aufgabe 5.1.1 erstellten SR-Latch als Komponenten.

b. (1 Punkte) Testen Sie Ihre Implementierung mit allen Kombinationen von CLK und D (00, 01, 10 und 11).

c. (1 Punkte) Erklären Sie den Vorteil des D-Latches im Vergleich zum SR-Latch.

c. Vorteil: - Der irreguläre Zustand S = R = 1 wird verhindert, sodass kein irregulärer Zustand mehr anliegen kann.

↳ dies geschieht durch vorgeschaltete Logik vor dem R-S-Eingang, sodass niemals durch ein sich unterscheidendes D, sowohl am oberen als auch unteren AND-Gatter '1' anliegen kann.

- es resultiert, dass das Verhalten ggü. dem SR-Latch nun eindeutig und stabiler ist

Hinweis: Für D-Latch bzw. SR-Latch wurden jeweils seperate work-obj.cf und testbench.vcd angelegt, benannt work-obj93_d-latch.cf bzw. work-obj93_sr-latch.cf und testbench_d_latch.cf bzw. testbench_sr_latch.cf

5.2 D-Flip-Flop in VHDL (8 Punkte)

Zusätzlich zu den vorherigen Schaltwerken ist das D-Flip-Flop eine weitere grundlegende sequentielle Schaltung, die in der Lage ist, einen Wert zu speichern. Der Unterschied zwischen dem D-Flip-Flop und dem D-Latch ist, dass der Ausgang Q des D-Latches sich zu jeder Zeit ändern kann (sofern an CLK der logische Pegel 1 anliegt), wohingegen sich beim D-Flip-Flop der Wert am Ausgang Q nur zum Zeitpunkt einer aufsteigenden Taktflanke ändern kann. Zu allen anderen Zeitpunkten wird der Ausgabewert gespeichert.

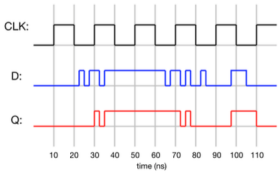


Abbildung 3: D-Latch Signalverlauf.

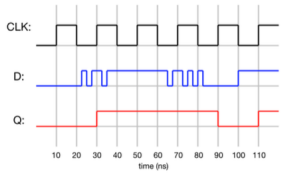


Abbildung 4: D-Flip-Flop Signalverlauf.

Um diesen Unterschied besser zu verstehen, ist ein Beispielszenario in den Abbildungen 3 und 4 dargestellt, bei dem der Signalverlauf für ein D-Latch, respektive D-Flip-Flop, gezeigt wird. Wie in Abbildung 3 zu sehen ist, ändert sich die Ausgabe des D-Latches nach 32 ns, weil der Eingabewert von D sich zu diesem Zeitpunkt ändert. Das D-Flip-Flop ändert seine Ausgabe zu diesem Zeitpunkt jedoch nicht, da keine aufsteigende Taktflanke vorliegt. Das D-Flip-Flop kann wie in Abbildung 5 dargestellt aus zwei hintereinandergeschalteten D-Latches mit komplementärem Taktsignal gebaut werden.

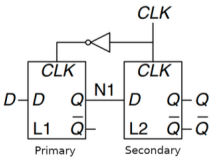


Abbildung 5: Schaltbild eines Primary-Secondary-D-Flip-Flops

Aufgaben:

a. (1 Punkte) Erklären Sie ausführlich, wie das Primary-Secondary-D-Flip-Flop in Abbildung 5 funktioniert.

b. (3 Punkte) Implementieren Sie ein D-Flip-Flop basierend auf der Wahrheitstabelle 3 und testen Sie Ihre Implementierung mit allen unterschiedlichen Kombinationen von CLK und D (00, 01, 10 und 11). Verwenden Sie dazu einige der von Blatt 2 bekannten Logikgatter und das D-Latch aus Aufgabe 5.1.2 als Komponenten.

a.

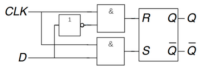


Abbildung 2: Schaltbild eines D-Latches.

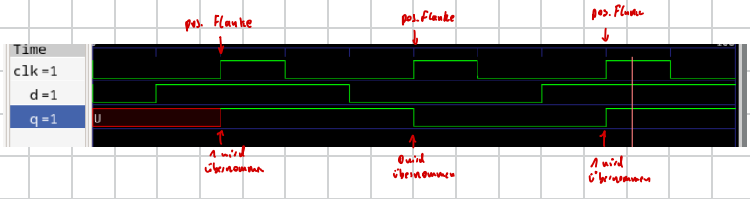
CLK	D	Q	$\bar{Q}$
0	x	Q_{prev}	$\bar{Q}_{prev}$
1	0	0	1
1	1	1	0

Tabelle 2: Wahrheitstabelle eines D-Latches.

- Wir haben also zwei D-Latches, wo das linke D-Latch das negierte Clock-Signal hat und das rechte D-Latch hat direkt das CLK-Signal anliegen.
- Es gilt: • 1. Nur wenn CLK = '0' anliegt, wir das Signal im linken D-Latch "aufgenommen" und liegt dann an N1 an
  → sofern nun also den Wert im Primary-Secondary-F-F ändern will, muss nun dies während CLK = '0' sein
  • 2. Wenn CLK = '1' gilt, ist der linke D-Latch "blockiert" also ist es egal was an D anliegt an N1 liegt immer Q an (Q wurde wie oben beschrieben gesetzt)
  → im rechten Latch liegt CLK = '1' an, somit wird das vorherige "aufgenommene" Signal übernommen und gespeichert

=) Der Zusammenhang zwischen CLK, dass im linken D-Latch "aufgenimmt" und dann bei CLK='1' "übernimmt" implementiert die positive Flanke

b.



CLK	D	Q	$\overline{Q}$
0	x	Q_{prev}	$\overline{Q}_{prev}$
1	0	0	1
1	1	1	0

Tabelle 3: Wahrheitstabelle eines D-Flip-Flops.

c. (1 Punkte) Damit ein Flip-Flop in einen definierten Startzustand gebracht werden kann, muss es ermöglicht werden, dieses zurücksetzen zu können (*Reset*). Erklären Sie den Unterschied zwischen Flip-Flops mit synchronem und asynchronem Reset.

d. (2 Punkte) Erweitern Sie das Flip-Flop, das Sie in Aufgabenteil b entworfen haben, um die Funktion eines synchronen Resets. Zeichnen Sie das Schaltbild dieses synchronen und resetbaren Flip-Flops und implementiert es in VHDL.

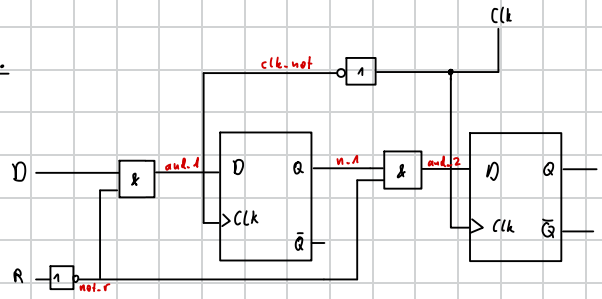
e. (1 Punkte) Testen Sie Ihre Implementierung mit allen Kombinationen von CLK und D (00, 01, 10, 11) und setzen Sie es danach auf den Startzustand (00) zurück.

c.

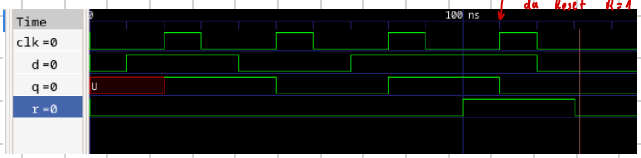
synchroner Reset:
- Der Reset erfolgt nur zusammen mit dem Takt, somit synchron zum nächsten Wechsel der Flanke des Taktes
- Zustände sind also abhängig vom Taktsignal, sodass Reset nur wirksam wird, wenn die nächste Clock-Flanke eintrifft

asynchroner Reset:
- Reset ist unabhängig vom Clock-Signal, d.h. FF wird sofort zurückgesetzt, sofern Reset-Signal anliegt

d.



e.



trotz dessen, dass D=1, gilt ist Q=0, da Reset R=1 gilt
=> reset wird also wie erwünscht bei pos. Flanke zurückgesetzt

5.3 JK-Flip-Flop in VHDL (4 Punkte)

Das JK-Flip-Flop ist ein weiterer Baustein, der ähnlich wie das D-Flip-Flop die Daten speichert und getaktet die Ausgabesignale verändern kann. Abbildung 6 zeigt das Schaltbild und die Wahrheitstabelle ist in Tabelle 4 angegeben.

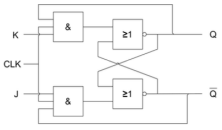


Abbildung 6: Schaltbild eines JK-Flip-Flops

J	K	Q	$\overline{Q}$
0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0
0	1	0	1
1	1	$\overline{Q}_{prev}$	Q_{prev}

Tabelle 4: Wahrheitstabelle eines JK-Flip-Flops.

Bisher wurden die Schaltungen im HaPra strukturell beschrieben, indem die Komponenten der Schaltung und deren Verbindungen beschrieben wurden. Es ist jedoch ebenfalls möglich, das Verhalten einer Schaltung zu beschreiben. In der beigelegten Datei *jk_flipflop.vhdl* geschieht dies für ein JK-Flip-Flop mit reset.

Aufgaben:

a. (1 Punkt) Beschreiben Sie den Unterschied zwischen einem JK-Flipflop und einem D-Flipflop.

b. (2 Punkt) Betrachten Sie die beigefügte Implementierung des JK-Flop-Flops und beschreiben Sie die Funktionsweise, um das Verhalten aus der Wahrheitstabelle zu implementieren.

c. (1 Punkt) Erstellen Sie eine Testbench und testen Sie jede Kombination aus Eingangssignalen sowie die Reset-Funktion, um die Funktionsweise zu überprüfen.

a.

- Ein D-FF besitzt ggü. des JK-FF, der 2 Eingänge besitzt
- D-FF wobei D für delay steht erzielt eines seits das abspeichern als auch das übergeben des D-Signals bei Q_{t+1} daher nach einem Takt
- JK-FF besitzt das Verhalten eines R-S-FF, wobei allerdings keine verbotenen Zustände existieren, sondern über diesen Schaltung vorher anliegende Signale invertiert

- Die Schaltung des D-FF ist einfacher, als die des JK-FF
- Wenn nun die Automatenfunktion für Q und das Setzen von Q' betrachtet, bleibt zudem das J-K weiter Freiheitsgrade

b.

```
entity jk_flipflop is
    port(
        j,k, clk, reset : in std_logic;
        Q : out std_logic
    );
end jk_flipflop;

architecture bh of jk_flipflop is
        signal q_int : std_logic;
    begin
        q <= q_int;
        process begin
            wait until rising_edge(clk); --wir betrachten nur positive Flanken d.h. den Übergang von CLK=0 zu CLK=1
                if (reset = '1') then     -- da reset-Signal zuerst definiert ist, hat 'reset' die höchste
                    q_int <= '0';         Priorität, somit wird Q falls reset aktiv d.h. reset = 1 auf 0 gesetzt
                elsif (j = '0' and k = '1') then -- Kill wird ausgeführt somit wird Q auf 0 gesetzt, unser Ausgang
                    q_int <= '0';                 signal Q wird somit auf Q=0 gesetzt
                elsif (j = '1' and k = '0') then -- Jump wird ausgeführt und Q auf 1 gesetzt also Q=1, falls
                    q_int <= '1';                 Kill-Signal mit 0 anliegt
                elsif (j = '1' and k = '1') then --  Q wird invertiert also wird die Toggle Funktion also
                    q_int <= not q_int;           der Zustandswechsel ausgeführt
                end if;
        end process;
end bh;
```

c.