Stefan Dumler
Friedrichshafener Str. 1
87439 Kempten
E-Mail: stdumler@web.de

Dipl.-Ing. (TU) Klaus-Eckart Schulz
Birnbaumring 64
13159 Berlin

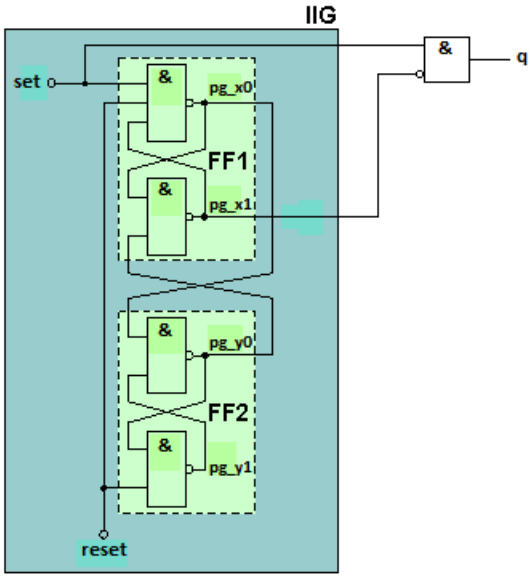
Kempten, 24. Mai 2018

Sehr geehrter Herr Schulz,

das auf Ihrer Webseite vorgestellte JK-Flipflop und das darin enthaltene ideale Impulsglied sind sehr interessante Schaltungen. Ich suchte schon seit längerem ein Impulstor, mit dem man Taktimpulse kontrolliert freigeben konnte. Hier war es ganz wichtig, dass der erste Taktimpuls nicht abgeschnitten wurde, wie es beispielsweise bei einem einfachen UND-Gatter der Fall ist.
Die Möglichkeit, ein mit zwei NICHT-UND-Gattern aufgebautes gewöhnliches RS-Flipflop zusammen mit einem UND-Gatter als Impulstor zu verwenden, habe ich ebenfalls in Betracht gezogen, erschien mir aber nicht als ideal. Ihr JK-FF sowie das von Tom Del Rosso in einem Forum vorgestellte JK-FF, welches Sie auf Ihrer Webseite ebenfalls erwähnten, habe ich mir deshalb genauer angesehen und getestet.

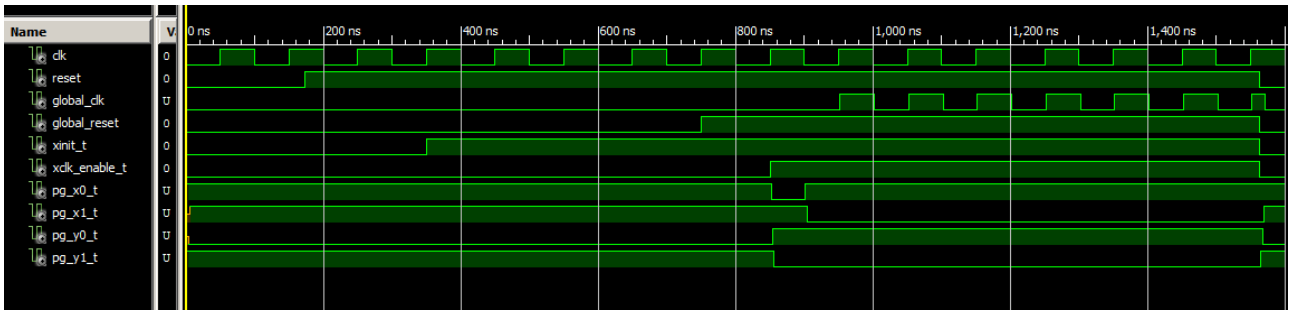
1. Ihr ideales Impulsglied (IIG):

In Ihrer Schaltung des neuen JK-FFs verwenden Sie ein ideales Impulsglied, welches ich genau für meine Zwecke verwenden konnte. Insbesondere habe ich damit ein kontrolliertes Freigeben von Taktsignalen für digitale Systeme realisieren können. Die Implementierung erfolgte als VHDL-Code in CPLDs und FPGAs. Hier der Schaltplan des modifizierten Impulsgliedes zur Verwendung als Impulstor:



Dazu habe ich Ihr ideales Impulsglied mit NICHT-UND-Gliedern aufgebaut und um ein UND-Gatter ergänzt. Das Taktsignal clk wird sowohl dem Impulsglied über den Eingang set als auch dem UND-Gatter zugeführt. Das Freigabesignal wird dagegen vom Ausgang pg_x1 abgeleitet und invertiert an den anderen Eingang des UND-Gatters gelegt.
Dem Impulsglied wird nun über den Eingang reset das taktsynchrone Freigabesignal zugeführt, welches wiederum von einem D-FF kommt. Am Ausgang q liegt das freigegebene Taktsignal.
Nachfolgend der vollständige VHDL-Code einschließlich des dazugehörigen Impulsdiagramms:

```
entity Clock_and_reset_pulse_starting_control is
   port  (  clk                                  : in STD_LOGIC;
             reset                              : in STD_LOGIC;
             global_clk, global_reset           : out STD_LOGIC;
             xinit_t, xclk_enable_t             : out STD_LOGIC;
             pg_x0_t, pg_x1_t, pg_y0_t, pg_y1_t : out STD_LOGIC
          );
end Clock_and_reset_pulse_starting_control;

architecture Behavior of Clock_and_reset_pulse_starting_control is
   constant gate_delay    : time := 2 ns;
begin
   INITIALIZE:    block
                     signal xclk_enable                                 : STD_LOGIC;
                     signal xinit                                       : STD_LOGIC_VECTOR (0 to 1);
                     signal xreset                                      : STD_LOGIC_VECTOR (0 to 3);
                     signal pg_x0, pg_x1, pg_y0, pg_y1                  : STD_LOGIC;
                     attribute NOREDUCE                                 : string;
                     attribute NOREDUCE of pg_x0, pg_x1, pg_y0, pg_y1   : signal is "TRUE";
                  begin
                     -- Synchronisieren des externen Rücksetzsignals mit dem externen Taktsignal
                     SYNCHRONIZE:  process( clk, reset )
                                   begin
                                      if reset = '0' then
                                         xinit <= (others => '0');
                                      elsif rising_edge( clk ) then
                                         xinit <= '1' & xinit(0);
                                      end if;
                                   end process SYNCHRONIZE;

                     -- Globales Rücksetzsignal verzögert erzeugen
                     RESET_DELAY:  process( clk, xinit(1) )
                                   begin
                                      if xinit(1) = '0' then
                                         xreset <= (others => '0');
                                      elsif rising_edge( clk ) then
                                         xreset <= '1' & xreset(0 to 2);
                                      end if;
                                   end process RESET_DELAY;
                     global_reset <= xreset(3);

                     -- Taktfreigabesignal verzögert erzeugen
                     CLOCK_ENABLE_DELAY: process( clk, xreset(3) )
                                         begin
                                            if xreset(3) = '0' then
                                               xclk_enable <= '0';
                                            elsif rising_edge( clk ) then
                                               xclk_enable <= '1';
                                            end if;
                                         end process CLOCK_ENABLE_DELAY;

                     -- Globales Taktsignal erzeugen
                     -- << Erweitertes Impulsglied als Impulstor >>
                     pg_x0 <= not ( clk and xclk_enable and pg_x1 )      after gate_delay;
                     pg_x1 <= not ( pg_x0 and pg_y0 )                   after gate_delay;
                     pg_y0 <= not ( pg_x0 and pg_y1 )                   after gate_delay;
                     pg_y1 <= not ( pg_y0 and xclk_enable )             after gate_delay;
                     global_clk <= clk and not pg_x1                    after gate_delay;

                     xinit_t <= xinit(1);
                     xclk_enable_t <= xclk_enable;
                     pg_x0_t <= pg_x0;
                     pg_x1_t <= pg_x1;
                     pg_y0_t <= pg_y0;
                     pg_y1_t <= pg_y1;
                  end block INITIALIZE;
end Behavior;
```



Deutlich erkennbar ist, dass der erste Taktimpuls in global_clk nicht abgeschnitten wird. Beim letzten Taktimpuls spielt dies dagegen keine Rolle und ist unkritisch, da global_reset bereits L-Signal angenommen und damit die weitere Schaltung zurückgesetzt hat.

Zur Funktion: Die ersten beiden FFs xinit(0) und (1) synchronisieren das externe reset-Signal mit dem externen Taktsignal clk. Damit ein metastabiles Verhalten im FF xinit(0) zu keinem unkontrollierten Verhalten der weiteren Schaltung führt, dient das nachfolgende zweite FF xinit(1), welches das H-Signal in xinit(0) taktsynchron aufnimmt. Hierdurch werden die Löscheingänge der FFs xreset taktsynchron freigegeben. Im Diagramm entspricht xinit_t dem Signal xinit(1).
Die darauffolgenden Taktflanken in clk setzen nun nacheinander alle Ausgänge von xreset auf H, da diese FFs als Schieberegister konfiguriert wurden. Bis xreset(3) (im Diagramm als global_reset beschriftet) H-Signal annimmt, vergehen vier Taktperioden. Dadurch entsteht eine gewünschte Verzögerung zwischen xinit(1) und xreset(3).
Sobald die nächste Taktflanke in clk eintrifft, erhält das Signal von FF xclk_enable, wegen xreset(3) = H, ebenfalls H-Signal. xclk_enable ist das Taktfreigabesignal und gibt das Impulsglied frei. xclk_enable ist im Diagramm als xclk_enable_t beschriftet.

pg_x0_t, pg_x1_t, pg_y0_t und pg_y1_t sind die Ausgänge innerhalb des Impulsgliedes. Durch die Freigabe über xclk_enable erhält das Signal pg_x1_t mit der fallenden Flanke L-Signal. Dies wird nun invertiert dem UND-Gatter weitergereicht, wodurch dieses aktiviert wird. Das Taktsignal clk kann somit ungehindert durch das UND-Gatter gelangen.

Aufgrund des zusätzlichen FFs und des Impulsgliedes entsteht zwischen xreset(3) und global_clk eine Verzögerung von zwei Taktperioden, was der zusätzlichen Sicherheit dient, bis die nachfolgende Schaltung durch global_reset sicher freigegeben wurde.
Nimmt reset nun wieder L-Signal an, werden zunächst die FFs xinit(0) und (1) zurückgesetzt (→ xinit_t = L) und somit auch die FFs xreset(0) bis (3) (→ global_reset = L). Dies führt dazu, dass das FF xclk_enable ebenso zurückgesetzt wird (→ xclk_enable_t = L) und dadurch das Impulsglied wegen pg_x1_t = H blockiert.

global_reset dient allgemein als systemweites Rücksetzsignal, global_clk ist das systemweite Taktsignal.

Im Quellcode dient die Konstante gate_delay dazu, den Gattern im Impulsglied eine realistische Durchlaufverzögerung zu geben. Eingestellt ist eine Gatterlaufzeit von 2 ns.

2. Ihr JK-FF:

In der alten Version hatten Sie Ihr JK-FF ohne den Freigabeeingang F dargestellt. In der neuen Version dagegen mit Freigabeeingang, wobei das Ausgangs-FF (pg_z0 / pg_z1) keine Verbindung mit diesem erhielt, also nur die vier linken Register (jeweils aus zwei NICHT-ODER-Gatter gebildet).
Ihr JK-FF in der neuen Version habe ich im VHDL-Code erstellt und im Xilinx-Simulator getestet und synthetisiert. Über gate_delay ist dazu wieder eine Gatterlaufzeit von 2 ns eingestellt. Der dazugehörige VHDL-Codeausschnitt lautet:

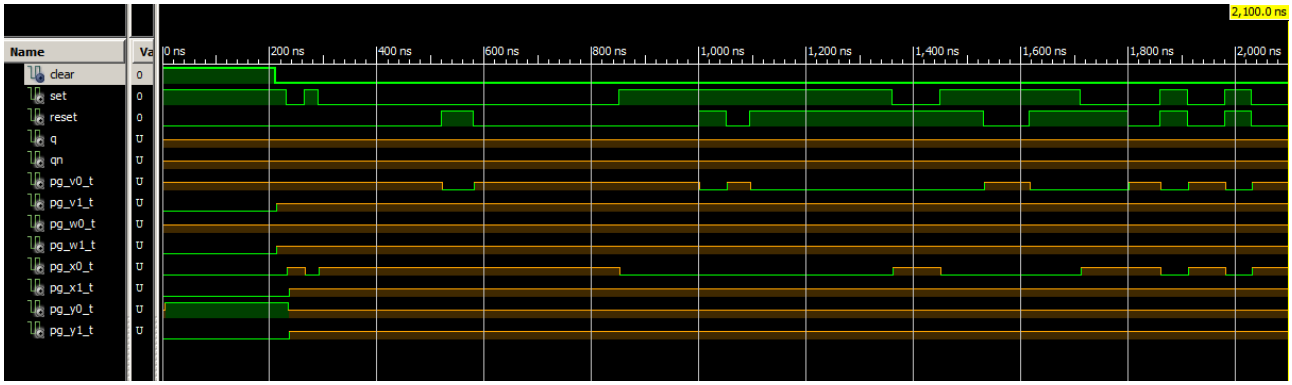
```
-- JK-FF Klaus-Eckart Schulz (neue Version):
pg_x0 <= not ( set or pg_z1 or pg_x1 ) after gate_delay;
pg_x1 <= not ( pg_x0 or clear or pg_y0 ) after gate_delay;

pg_y0 <= not ( pg_x0 or pg_y1 ) after gate_delay;
pg_y1 <= not ( pg_y0 or clear or pg_z1 ) after gate_delay;

pg_v0 <= not ( reset or pg_z0 or pg_v1 ) after gate_delay;
pg_v1 <= not ( pg_v0 or clear or pg_w0 ) after gate_delay;

pg_w0 <= not ( pg_v0 or pg_w1 ) after gate_delay;
pg_w1 <= not ( pg_w0 or clear or  pg_z0 ) after gate_delay;

pg_z0 <= not ( pg_x1 or pg_z1 ) after gate_delay;
pg_z1 <= not ( pg_z0 or pg_v1 ) after gate_delay;

pg_x0_t <= pg_x0;     pg_x1_t <= pg_x1;
pg_y0_t <= pg_y0;     pg_y1_t <= pg_y1;
pg_v0_t <= pg_v0;     pg_v1_t <= pg_v1;
pg_w0_t <= pg_w0;   pg_w1_t <= pg_w1;
q <= pg_z1;
qn <= pg_z0;
pg_set_t <= pg_set;
pg_reset_t <= pg_reset;
```

Die Testumgebung ist:

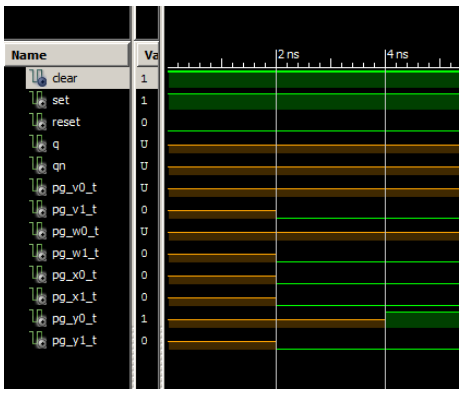
```
signal clear    : std_logic := '1';
signal set      : std_logic := '1';
signal reset    : std_logic := '0';

begin
   stim_proc: process
              begin
                 wait for 100 ns; -- hold reset state for 100 ns

                 clear <=   '0' after 110 ns, '1' after 2005 ns, '0' after 2080 ns;
                 set <=     '0' after 131 ns,
                            '1' after 165 ns, '0' after 190 ns, '1' after 750 ns, '0' after 1260 ns, '1' after 1350 ns, '0' after 1610 ns,
                            '1' after 1760 ns, '0' after 1810 ns, '1' after 1880 ns, '0' after 1930 ns, '1' after 2075 ns, '0' after 2120 ns;
                 reset <=   '1' after 420 ns, '0' after 480 ns, '1' after 900 ns, '0' after 950 ns,
                            '1' after 995 ns, '0' after 1430 ns, '1' after 1515 ns, '0' after 1700 ns,
                            '1' after 1760 ns, '0' after 1810 ns, '1' after 1880 ns, '0' after 1930 ns;
                 wait;
              end process;
end;
```

Für den Test habe ich die gleiche Impulsfolge verwendet, wie Sie sie auf Ihrer Webseite angegeben haben. Lediglich am Anfang wurde sie von mir geringfügig abgeändert, indem ich zu Beginn set = H und clear = H setzte. set erhält erst dann L-Signal, nachdem zunächst wieder clear = L geworden ist.

Bei der Simulation stellte ich fest, dass Ihr JK-FF (neue Version) aber undefinierte Zustände annimmt (siehe die beiden Impulsdiagramme unten).
Solange clear = H ist, zeigt eine Änderung an set keine Wirkung. Ohne diese Änderung in der Impulsfolge ergeben sich aber genauso undefinierte Zustände. Die Wirkung des Freigabeeinganges F (clear) ist statisch und blockiert das JK-FF sicher.

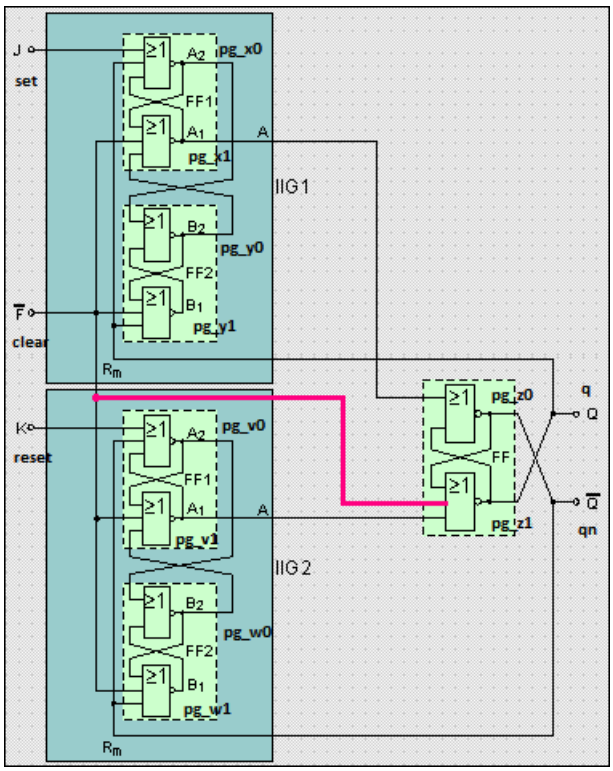


Das zweite Diagramm zeigt die Ausschnittsvergrößerung vom Zeitpunkt t = 0 ns bis t = 5 ns.



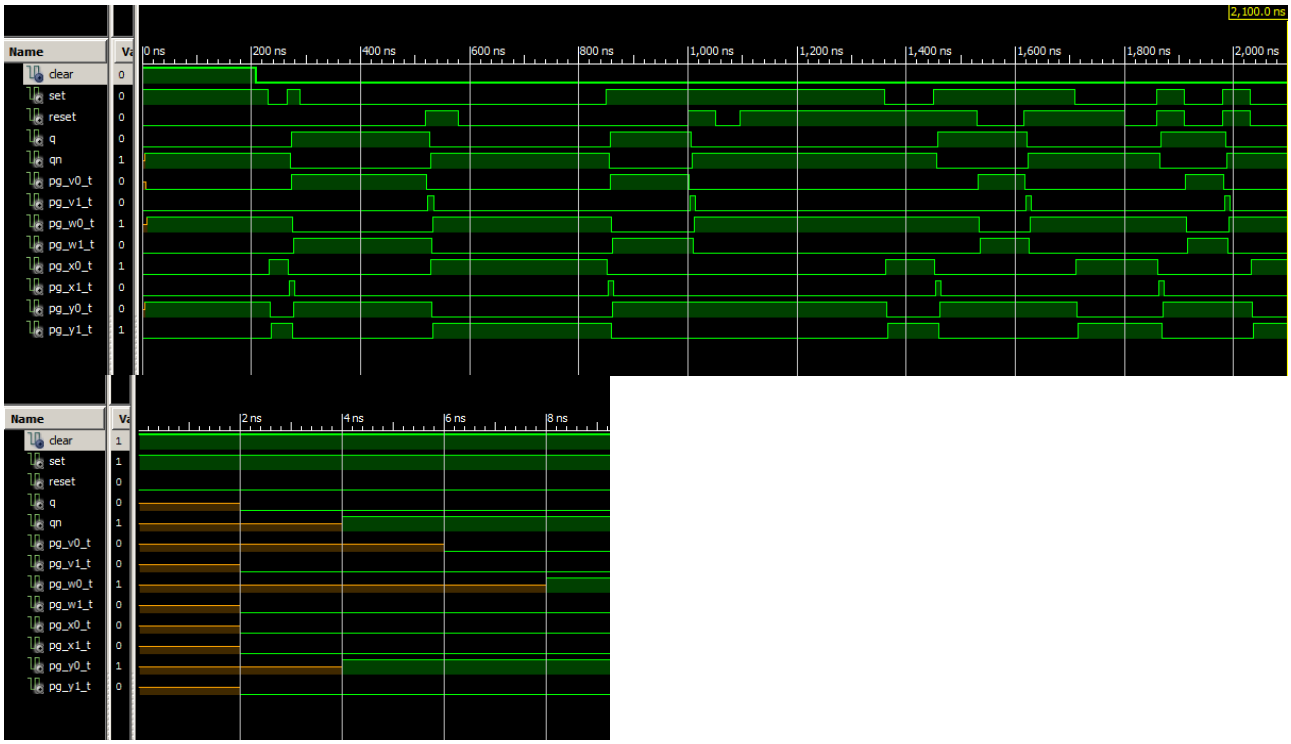
Die Ursache für die undefinierten Signalzustände bei der Simulation liegt sehr wahrscheinlich beim Ausgangs-FF (pg_z0 / pg_ z1), da der Simulator den Zustand zum Zeitpunkt t = 0 dort nicht eindeutig festlegen kann.

Mir fiel an Ihrer Schaltung dabei auf, dass nur die vier Eingangs-FFs vom Freigabesignal F angesteuert werden, das Ausgangs-FF (pg_z0 / pg_z1) dagegen nicht. Ich habe daraufhin den Freigabeeingang F mit einem zusätzlichen Eingang des NICHT-UND-Gatters pg_z1 verbunden. Im folgenden Schaltbild Ihres JK-FFs ist die zusätzliche Verbindung rot eingezeichnet:



Weist nun der Freigabeeingang F während des Einschaltens H-Signal auf, erhalten die Ausgänge der Eingangs-FFs pg_x1, pg_y1, pg_v1, pg_w1 sowie pg_z1 (= q) L-Signal. pg_z0 (= qn) erhält wegen pg_x1 = L und pg_z1 = L deshalb H-Signal.
Ist dabei gleichzeitig auch set = H, wird pg_x0 = L. Dadurch erhält pg_y0 dann H-Signal, weil auch pg_y1 = L ist.
Wenn dagegen set = L ist, wird pg_x0 = H, weil alle Eingänge dort L-Signal aufweisen (pg_z1 = q = L, pg_x1 = L). Dies führt dazu, dass auch pg_y0 = L ist. Beim Eingang reset ist es äquivalent.
Fehlt dagegen die Verbindung des Freigabeeinganges F zum Gatter pg_z1, ist es unbestimmt, welcher Ausgang des FFs, pg_z0 bzw. pg_z1, H-Signal annimmt, da über pg_x1 und pg_v1 L-Signale an dessen Eingänge gelangen. Dadurch kann der Simulator dort keine definierten Pegel festlegen.

Das Impulsdiagramm zeigt nun die definierten Signalzustände an den Ausgängen, wenn das Gatter pg_z1 zusätzlich ein H-Signal über den Freigabeeingang F (clear) erhält (darunter ist die Ausschnittsvergrößerung vom Zeitpunkt t = 0 ns bis t = 9 ns zu sehen):



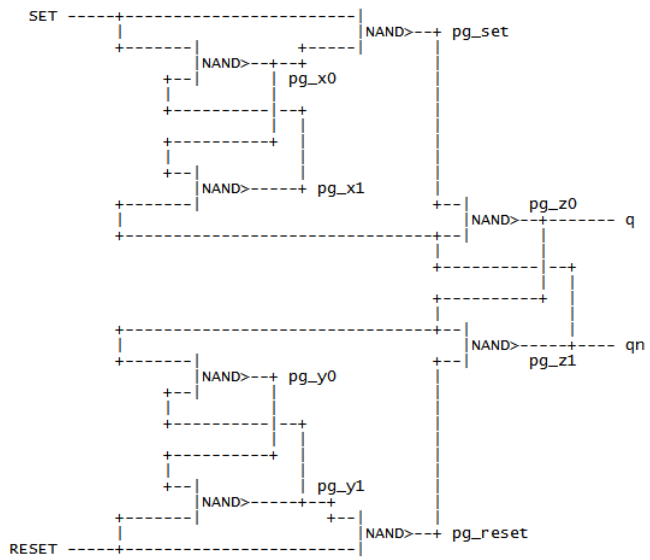
Die ersten 8 Nanosekunden benötigt das JK-FF bis zur vollständigen Stabilisierung aufgrund der Gatterlaufzeiten. Danach zeigt es ein definiertes und korrektes Verhalten. Es ist also wichtig, das Ausgangs-FF (pg_z0 / pg_z1) ebenso mit dem Freigabesignal F (clear) zurückzusetzen. Hier der VHDL-Codeausschnitt für das von mir ergänzte JK-FF:

```
-- JK-FF Klaus-Eckart Schulz (Ergänzung fett gedruckt):
pg_x0 <= not ( set or pg_z1 or pg_x1 ) after gate_delay;
pg_x1 <= not ( pg_x0 or clear or pg_y0 ) after gate_delay;

pg_y0 <= not ( pg_x0 or pg_y1 ) after gate_delay;
pg_y1 <= not ( pg_y0 or clear or pg_z1 ) after gate_delay;

pg_v0 <= not ( reset or pg_z0 or pg_v1 ) after gate_delay;
pg_v1 <= not ( pg_v0 or clear or pg_w0 ) after gate_delay;

pg_w0 <= not ( pg_v0 or pg_w1 ) after gate_delay;
pg_w1 <= not ( pg_w0 or clear or  pg_z0 ) after gate_delay;

pg_z0 <= not ( pg_x1 or pg_z1 ) after gate_delay;
pg_z1 <= not ( pg_z0 or clear or pg_v1 ) after gate_delay;

pg_x0_t <= pg_x0;    pg_x1_t <= pg_x1;
pg_y0_t <= pg_y0;    pg_y1_t <= pg_y1;
pg_v0_t <= pg_v0;    pg_v1_t <= pg_v1;
pg_w0_t <= pg_w0;   pg_w1_t <= pg_w1;
q <= pg_z1;
qn <= pg_z0;
pg_set_t <= pg_set;
pg_reset_t <= pg_reset;
```

2. Das JK-FF von Tom Del Rosso:

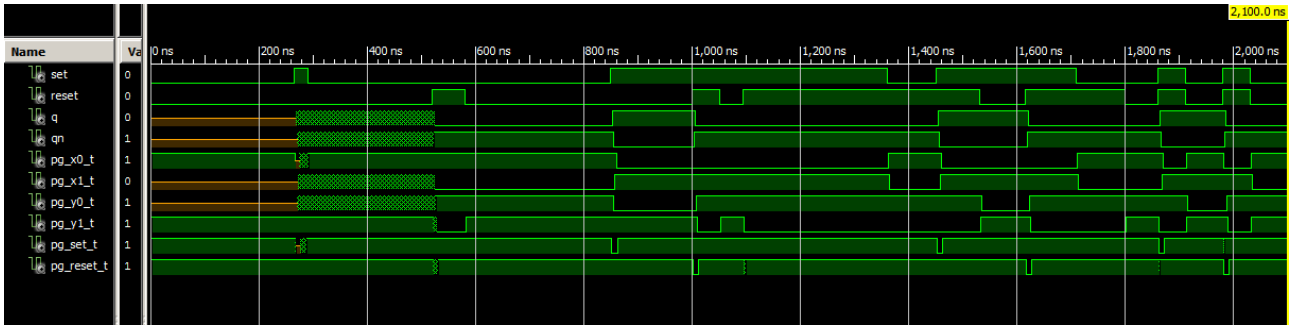
Auf Ihrer Webseite haben Sie auf ein JK-FF in einem Forum hingewiesen, welches dieselbe Funktion besitzt und mit zwei Gattern weniger auskommt. Grundsätzlich ist so etwas zu begrüßen. Ich habe mir deshalb dieses JK-FF von Tom Del Rosso genauer angesehen und analysiert. Dazu habe ich sie in VHDL nachgebaut und im Xilinx-Simulator getestet.

```
-- JK-FF Tom Del Rosso - Originalversion:
=========================================

  pg_x0 <= not ( pg_x1 and set ) after gate_delay;
  pg_x1 <= not ( pg_x0 and pg_z1 ) after gate_delay;

  pg_y0 <= not ( pg_z0 and pg_y1 ) after gate_delay;
  pg_y1 <= not ( pg_y0 and reset ) after gate_delay;

  pg_z0 <= not ( pg_z1 and pg_set ) after gate_delay;
  pg_z1 <= not ( pg_z0 and pg_reset ) after gate_delay;

  pg_set <= not ( set and pg_x0 ) after gate_delay;
  pg_reset <= not ( reset and pg_y1 ) after gate_delay;

  pg_x0_t <= pg_x0; pg_x1_t <= pg_x1;
  pg_y0_t <= pg_y0; pg_y1_t <= pg_y1;
  pg_v0_t <= pg_v0; pg_v1_t <= pg_v1;
  pg_w0_t <= pg_w0; pg_w1_t <= pg_w1;
  q <= pg_z0;
  qn <= pg_z1;
  pg_set_t <= pg_set;
  pg_reset_t <= pg_reset;
```



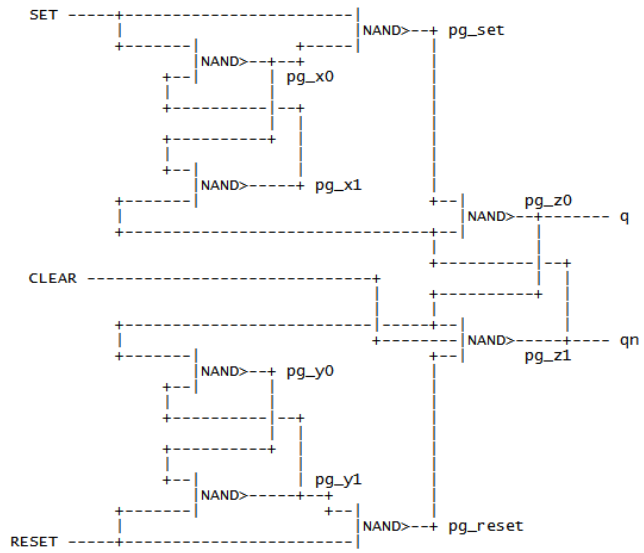
Oben sind der dazugehörige Quellcodeausschnitt sowie der Schaltplan, wie er von Tom Del Rosso im Forum dargestellt wird, zu sehen. Ich habe lediglich die Beschriftung abgeändert.

Die originale Version des JK-FFs von Tom Del Rosso zeigte bei der Simulation ein undefiniertes und zwischen den Zeitpunkten 260 ns und 520 ns ein instabiles und oszillierendes Verhalten (siehe Impulsdiagramm).
Dabei spielt es keine Rolle, ob set oder reset anfänglich mit H vorbelegt sind, die Oszillation besteht weiterhin. Auch hier ist die Ursache der undefinierten bzw. instabilen Signalzustände bei der Simulation sehr wahrscheinlich beim Ausgangs-FF (pg_z0 / pg_ z1) zu suchen, da der Simulator zum Zeitpunkt t = 0 ns nicht in der Lage ist, einen eindeutigen Zustand festzulegen.
Als Gatterlaufzeit sind über gate_delay wieder 2 ns eingestellt. Hier das resultierende Impulsdiagramm:

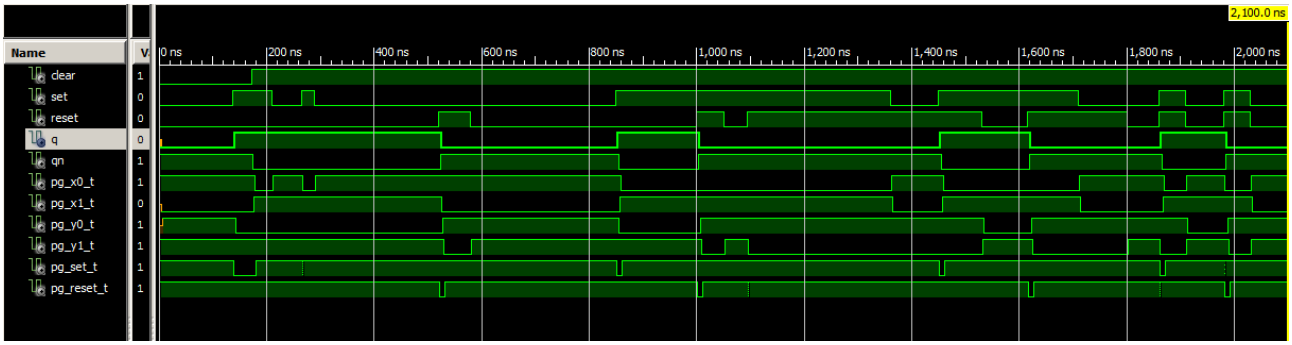


Basierend auf der Überlegung mit dem Freigabeeingang bei Ihrem JK-FF habe ich das Gatter pg_z1 vom Ausgangs-FF (pg_z0 / pg_z1) um einen weiteren Eingang ergänzt. Über diesen zusätzlichen Eingang wird jetzt das Freigabesignal clear zugeführt. Die Freigabe erfolgt erst dann, wenn clear = H ist. Damit sollte ein eindeutiger Zustand von Anfang gewährleistet sein. Nachfolgend der abgeänderte VHDL-Quellcodeausschnitt in der Version #1:

```
-- JK-FF Tom Del Rosso: Abgeänderte Version #1:
================================================

  pg_x0 <= not ( pg_x1 and set ) after gate_delay;
  pg_x1 <= not ( pg_x0 and pg_z1 ) after gate_delay;

  pg_y0 <= not ( pg_z0 and pg_y1 ) after gate_delay;
  pg_y1 <= not ( pg_y0 and reset ) after gate_delay;

  pg_z0 <= not ( pg_z1 and pg_set ) after gate_delay;
  pg_z1 <= not ( pg_z0 and pg_reset and clear ) after gate_delay;

  pg_set <= not ( set and pg_x0 ) after gate_delay;
  pg_reset <= not ( reset and pg_y1 ) after gate_delay;

  pg_x0_t <= pg_x0; pg_x1_t <= pg_x1;
  pg_y0_t <= pg_y0; pg_y1_t <= pg_y1;
  pg_v0_t <= pg_v0; pg_v1_t <= pg_v1;
  pg_w0_t <= pg_w0; pg_w1_t <= pg_w1;
  q <= pg_z0;
  qn <= pg_z1;
  pg_set_t <= pg_set;
  pg_reset_t <= pg_reset;
```

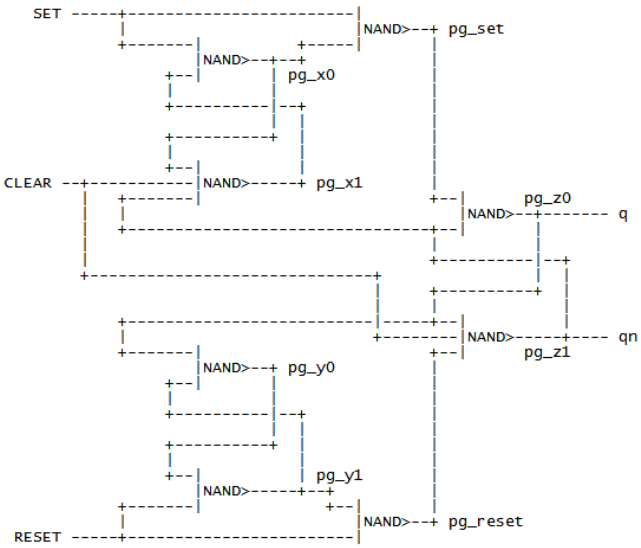


Die Impulsfolge in der Testumgebung habe ich etwas abgeändert, um die Wirkung des Freigabeeinganges clear zu demonstrieren. Dort erhält set bereits H-Signal, bevor clear = H ist, um danach wieder L-Signal anzunehmen. Es ergibt sich dadurch für Version #1 nun folgendes Impulsdiagramm:

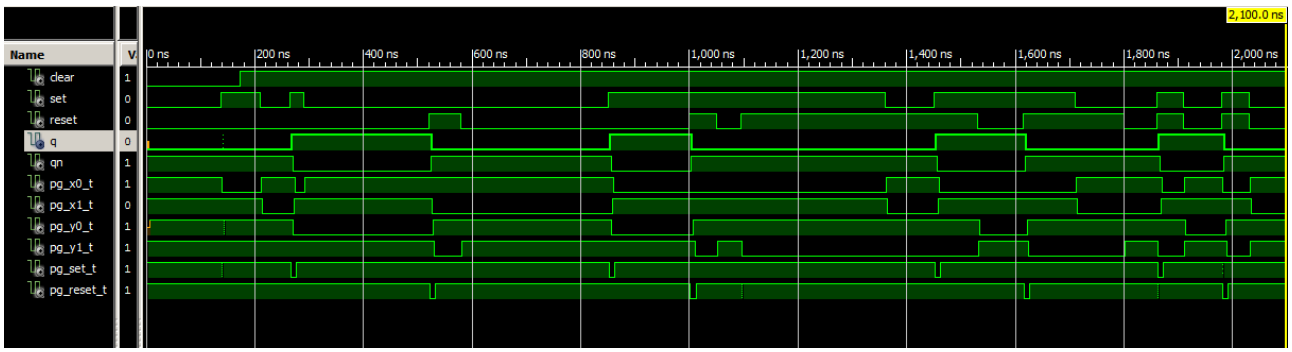


Das JK-FF arbeitet nun stabil und ohne Oszillationen. Lediglich das Ausgangssignal q zeigt ein fehlerhaftes Aussehen. Während clear = L ist, sollte auch q = L sein. Tatsächlich nimmt q H-Signal an, sobald an set die Änderung von L nach H erfolgt. qn erhält dagegen erst dann L-Signal, wenn clear = H ist. Beides ist aber nicht korrekt.
Da clear nur auf das Ausgangs-FF wirkt, nicht aber auf die Eingangs-FFs, liegt es nahe, das Gatter pg_x1 um einen weiteren Eingang zu erweitern und dort ebenfalls das Freigabesignal clear anzulegen. Der VHDL-Quellcodeausschnitt in der Version #2 lautet nun:

```
-- JK-FF Tom Del Rosso - Abgeänderte Version #2:
================================================

  pg_x0 <= not ( pg_x1 and set ) after gate_delay;
  pg_x1 <= not ( pg_x0 and pg_z1 and clear ) after gate_delay;

  pg_y0 <= not ( pg_z0 and pg_y1 ) after gate_delay;
  pg_y1 <= not ( pg_y0 and reset ) after gate_delay;

  pg_z0 <= not ( pg_z1 and pg_set ) after gate_delay;
  pg_z1 <= not ( pg_z0 and pg_reset and clear ) after gate_delay;

  pg_set <= not ( set and pg_x0 ) after gate_delay;
  pg_reset <= not ( reset and pg_y1 ) after gate_delay;

  pg_x0_t <= pg_x0; pg_x1_t <= pg_x1;
  pg_y0_t <= pg_y0; pg_y1_t <= pg_y1;
  pg_v0_t <= pg_v0; pg_v1_t <= pg_v1;
  pg_w0_t <= pg_w0; pg_w1_t <= pg_w1;
  q <= pg_z0;
  qn <= pg_z1;
  pg_set_t <= pg_set;
  pg_reset_t <= pg_reset;
```

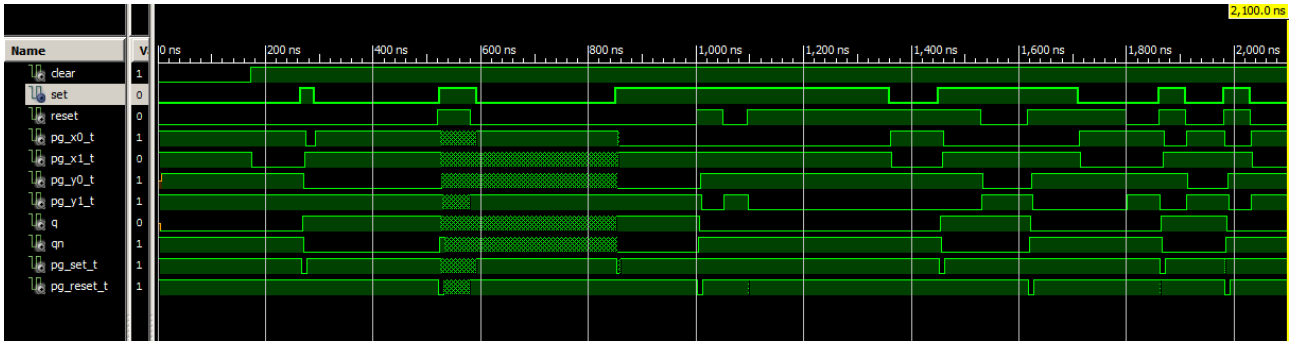


Nun zeigt das Impulsdiagramm der Version #2 die (nahezu) korrekten Signale q und qn:



Lediglich bei 137 ns ist auf q ein kurzer Glitch zu sehen. Vollkommen korrekt arbeitet diese Schaltung deshalb dennoch nicht. Sicherlich kann man diese noch weiter verbessern.

Darüber hinaus kann dieses JK-FF unter bestimmten Umständen auch instabil und zum Oszillieren gebracht werden. Das ist z. B. dann der Fall, wenn sowohl der Ausgang qn als auch das Eingangssignal set gleichzeitig von L nach H wechseln. Im nachfolgenden Impulsdiagramm ist dies im Zeitpunkt 524 ns der Fall. Dieses Verhalten tritt auch dann auf, wenn der Ausgang q und das Eingangssignal reset zur gleichen Zeit von L nach H wechseln.



Die Testumgebung für alle Versionen des JK-FFs von Tom Del Rosso lautet (versionsabhängige Änderungen sind darin *kursiv gedruckt*):

```
signal clear    : std_logic:= '0';
signal set      : std_logic := '0';
signal reset    : std_logic := '0';

begin
   stim_proc: process
              begin
                 wait for 100 ns; -- hold reset state for 100 ns.

                 clear <=   '1' after 73 ns, '0' after 2005 ns, '1' after 2080 ns;
                 set <=     '1' after 37 ns, '0' after 110 ns,  -- <<< nur bei den Versionen #1 und #2 einsetzen
                            '1' after 165 ns, '0' after 190 ns,
                            '1' after 424 ns, '0' after 493 ns,  -- <<< nur für den Fall qn und set = L → H  bei der Version #2 einsetzen
                            '1' after 750 ns, '0' after 1260 ns,
                            '1' after 1350 ns, '0' after 1610 ns,
                            '1' after 1760 ns, '0' after 1810 ns, '1' after 1880 ns, '0' after 1930 ns,
                            '1' after 2075 ns, '0' after 2120 ns;
                 reset <=   '1' after 52 ns, '0' after 93 ns,
                            '1' after 420 ns, '0' after 480 ns, '1' after 900 ns, '0' after 950 ns,
                            '1' after 995 ns, '0' after 1430 ns, '1' after 1515 ns, '0' after 1700 ns,
                            '1' after 1760 ns, '0' after 1810 ns, '1' after 1880 ns, '0' after 1930 ns;

                 wait;
              end process;
end;
```

Für alle oben angegebenen JK-FFs lautet der Deklarationsteil zum jeweiligen VHDL-Quellcodeausschnitt:

```
entity clk_enable is
   port  (  clear, set, reset                    : in STD_LOGIC;
            q, qn,
            pg_v0_t, pg_v1_t, pg_w0_t, pg_w1_t,
            pg_x0_t, pg_x1_t, pg_y0_t, pg_y1_t,
            pg_set_t, pg_reset_t                 : out STD_LOGIC
      );
end clk_enable;

architecture Behavior of clk_enable is
   signal      pg_v0, pg_v1, pg_w0, pg_w1,
               pg_x0, pg_x1, pg_y0, pg_y1,
               pg_z0, pg_z1, pg_set, pg_reset          : STD_LOGIC;
   attribute   NOREDUCE                                : string;
   attribute   NOREDUCE of  pg_v0, pg_v1, pg_w0, pg_w1,
                            pg_x0, pg_x1, pg_y0, pg_y1,
                            pg_z0, pg_z1, pg_set, pg_reset  : signal is "TRUE";
   constant    gate_delay                              : time := 2 ns;
begin
   <hier steht der jeweilige VHDL-Quellcodeausschnitt von einem der oben angegebenen JK-FFs>
end Behavior;
```

Das Attribut NOREDUCE dient dazu dem Compiler mitzuteilen, die Signale pg_v0, pg_v1, pg_w0, pg_w1, pg_x0, pg_x1, pg_y0, pg_y1, pg_z0, pg_z1, pg_set und pg_reset nicht wegzuoptimieren.

Fazit:
Baut man die JK-FFs (Ihres und das von Tom Del Rosso) mit TTL-Gattern nach, ergeben sich wieder bestimmte Zustände von Anfang an. Welcher Zustand aber nach dem Anlegen der Betriebsspannung vorherrscht, lässt sich auch dann nicht vorhersagen, solange keine definierte Freigabe erfolgt. Die Ursache liegt hier u. a. in den unvermeidlichen Bauteiltoleranzen als auch in der endlichen Anstiegsgeschwindigkeit der Betriebsspannung. Auch spielt die verwendete Schaltungstechnologie innerhalb der integrierten Schaltung eine große Rolle.

Ich empfehle meine Erweiterung in Ihrem JK-FF mit zu berücksichtigen, denn dadurch erhält Ihr – ohnehin schon geniales – Flipflop eine zusätzliche Verbesserung.
Die von Tom Del Rosso ist zwar genauso funktionstüchtig, letzten Endes aber nur mit den von mir gemachten Änderungen. Und es weist immer noch gewisse Schwächen auf, die u. U. störend sein können.
Ihr JK-FF ist dagegen sehr stabil und zuverlässig. Daher sollte es wirklich keine Rolle spielen, wenn Ihre Schaltung zwei Gatter mehr benötigt.

Mit besten Grüßen