# Automatenbeschreibungen in VHDL

## 1 Einführung

Obwohl ein Moore-Automat ein sehr einfaches strukturelles Modell besitzt (Zustandsspeicher, Zustandsübergangsfunktion, Ausgabefunktion), gibt es verschiedene Möglichkeiten, den Automaten in VHDL zu beschreiben. Je nach Beschreibung kann sich dann sogar das zeitliche Verhalten erheblich ändern. Anhand eines kleinen Beispiels soll der Einfluss der Beschreibung auf das Zeitverhalten, speziell der Ausgangssignale, gezeigt werden.

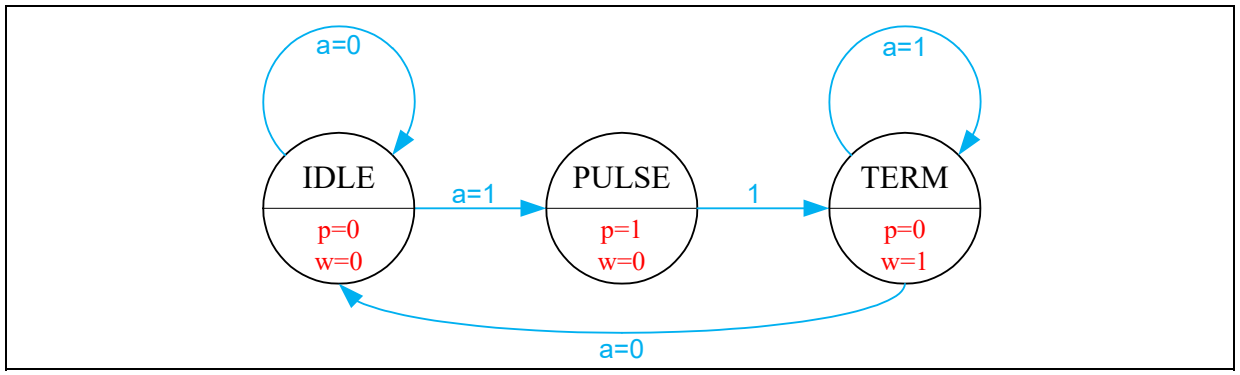


**Abbildung 1: Zustandsübergangsdiagramm**

Abbildung 1 zeigt das Zustandsübergangsdiagramm des Automaten. Er hat die Aufgabe, bei einem Wechsel des Eingangssignals *a* einen kurzen Impuls am Ausgang *p* zu erzeugen. Zur Illustration einiger Effekte soll er auch noch mit dem Ausgangssignal *w* anzeigen, dass er derzeit auf das Zurücksetzen des Eingangssignals *a* wartet.

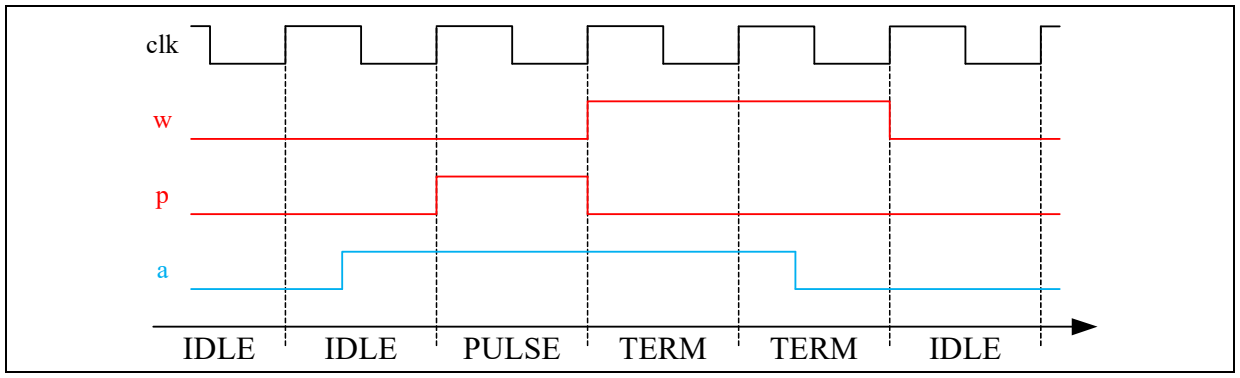


**Abbildung 2: Idealisierter Signalverlauf**

Abbildung 2 zeigt den erwünschten Signalverlauf für einen Zyklus. Zu Beginn soll sich der Automat im Wartezustand IDLE befinden. Das geschieht ohnehin, wenn das Signal *a* länger als zwei Taktperioden auf 0 liegt, alternativ kann ein Rücksetzsignal vorgesehen werden. Der Automat reagiert auf die steigenden Taktflanken und die Wechsel am Eingangssignal finden immer in einem zulässigen Zeitbereich statt (Setup- und Holdzeiten werden nie verletzt).

Im Versuch V6 (SPI-Automat) könnte das Signal *p* beispielsweise dem Signal *ldc* (*load command*) entsprechen. Dann würde der Automat im Zustand *IDLE* auf das achte Datenbit warten und im Zustand *TERM* prüfen, ob es sich um ein gültiges Kommando handelt.

# 2 Umsetzungen in VHDL

Im Folgenden werden fünf unterschiedliche VHDL-Beschreibungen vorgestellt. Alle Automaten werden mit derselben Eingangssequenz simuliert. Das jeweilige Ergebnis finden Sie im Anhang. Für alle Beschreibungen wurde dieselbe *entity* (Listing 1) verwendet.

```
entity automat is
  port
  (
    clk: in  std_logic;
    a,r: in  std_logic;
    p,w: out std_logic
  );
end;
```

**Listing 1: Gemeinsame Entity**

Das Eingangssignal r ist ein asynchrones Rücksetzsignal, mit dem der Automat in den Zustand IDLE gebracht wird. In den folgenden *architectures* werden Sie bei Signalzuweisungen oft eine Klausel *after xx ns* finden. Diese Zeitangabe hat nur für die Simulation eine Bedeutung. Die Signalzuweisung findet erst nach der angegebenen Zeit statt.

## 2.1 Direkte Umsetzung der Struktur

Da ein Automat zwei rein kombinatorische Funktionen und einen taktflankengesteuerten Speicher enthält, kann er mit einem *process* und zwei Funktionen beschrieben werden. Dies wurde in Listing 2 gemacht.

```
architecture a1 of automat is
  type states is (idle, pulse, term);
  signal ns, cs: states;
  signal nsi, nst: states;
begin

  -- Zustandsspeicher
  process(clk, r)
  begin
    if (r='0')
    then
      cs <= idle after 5ns;
    elsif (rising_edge(clk))
    then
      cs <= ns after 5ns;
    end if;
  end process;

  -- Zustandsübergangsfunktion (gesamt)
  with cs select
  ns <= nsi  after 10ns when idle,
        term after 10ns when pulse,
        nst  after 10ns when term;

  -- Zustandsübergangsfunktion (pro Zustand, Hilfssignale)
  nsi <= idle after 10ns when (a='0') else pulse after 10ns;
  nst <= term after 10ns when (a='1') else idle  after 10ns;

  -- Ausgabefunktion
  with cs select
  p <= '1' after 10ns when pulse,
       '0' after 20ns when others;

  with cs select
  w <= '1' after 10ns when term,
       '0' after 20ns when others;

end;
```

**Listing 2: Strukturnahe Beschreibung**

Das Ergebnis ist häufig unbefriedigend oder sogar unbrauchbar. Der Grund ist, dass zunächst mit einer steigenden Taktflanke der neue Zustand eingenommen wird und danach erst die

Ausgabefunktion die neuen Ausgabewerte bestimmen kann. Das führt zunächst zu einer mehr oder weniger großen Verzögerung vom Zustandswechsel zu den Ausgabewerten. Schlimmer ist aber, dass bei den Ausgabewerten Spikes (transiente falsche Ergebnisse, siehe Vorlesung) auftreten können. Das kann bei der weiteren Verwendung der Ausgabewerte zu Folgefehlern führen.
In Abbildung 3 ist ein ungünstiger Fall rot markiert. Der Zustandswechsel von *PULSE* nach *TERM* erfolgte bei *t=300 ns*. Für eine kurze Zeit erscheint an den Ausgängen eine ungültige Wertekombination, weil die Laufzeiten der Signale (*with/select*-Anweisung) unterschiedlich ist. Zudem dauert es vergleichsweise lang, bis sich nach dem Zustandswechsel die endgültige Belegung der Ausgänge einstellt. Das grundsätzliche Zeitverhalten ist dagegen korrekt.

## 2.2 Zusammenfassung des Speichers mit der ZÜ-Funktion

Da in mit nebenläufigen Anweisungen nicht geschachtelt werden kann, müssen für komplexere Funktionen Hilfssignale definiert werden. Zieht man die Übergangsfunktion mit in den Speicherprozess (Listing 3), dann kann man den Folgezustand sehr viel bequemer mit einer *case/when*-Anweisung berechnen.

```
architecture a2 of automat is
  type states is (idle, pulse, term);
  signal cs: states;
begin

  -- Zustandsübergangsfunktion und Zustandsspeicher
  process(clk, r)
  begin
    if (r='0')
    then
      cs <= idle after 5ns;
    elsif (rising_edge(clk))
    then
      case cs is
        when idle =>  if (a='0') then cs <= idle after 5ns;
                                 else cs<=pulse after 5ns;
                      end if;

        when pulse => cs <= term;

        when term =>  if (a='1') then cs <= term after 5ns;
                                 else cs <= idle after 5ns;
                      end if;
      end case;
    end if;
  end process;


  -- Ausgabefunktion
  with cs select
  p <= '1' after 10ns when pulse,
       '0' after 20ns when others;

  with cs select
  w <= '1' after 10ns when term,
       '0' after 20ns when others;

end;
```

**Listing 3: Berechnung des Folgezustands im *process***

Diese Beschreibung ist bequemer. Da die Ausgabefunktion weiterhin erst nach dem Zustandswechsel berechnet wird, bleibt der prinzipielle Nachteil der ersten Methode bestehen. Daher unterscheiden sich die Zeitverläufe der Ausgabesignale auch nicht von der ersten Version.

## 2.3 Ein-Prozess-Darstellung

Nun kann man auf die Idee kommen, auch noch die Ausgabefunktion mit in den getakteten Speicherprozess aufzunehmen. Diese Variante (Listing 4) führt zu einer sehr kompakten und leicht verständlichen Form. Man berechnet in jedem Zustand den Folgezustand und definiert dort auch gleich die Ausgaben, die zu diesem Zustand gehören sollen (die Reihenfolge spielt keine Rolle).

```
architecture a3 of automat is
  type states is (idle, pulse, term);
  signal cs: states;

begin

  -- Zustandsübergangsfunktion, Zustandsspeicher und Ausgabefunktion
  process(clk, r)
  begin
    if (r='0')
    then
      cs <= idle after 5ns;
    elsif (rising_edge(clk))
    then
      case cs is
        when idle =>  if (a='0') then cs <= idle  after 5ns;
                                 else cs <= pulse after 5ns;
                      end if;
                      p<='0'  after 5ns;
                      w<='0'  after 5ns;

        when pulse => cs <= term;
                      p<='1'  after 5ns;
                      w<='0'  after 5ns;

        when term =>  if (a='1') then cs <= term  after 5ns;
                                 else cs <= idle  after 5ns;
                      end if;
                      p<='0'  after 5ns;
                      w<='1'  after 5ns;
      end case;
    end if;
  end process;

end;
```

**Listing 4: Ein-Prozess-Darstellung eines Moore-Automaten**

Diese Variante hat jedoch ein schwerwiegendes Problem: Die Ausgaben erfolgen gegenüber den Zustandswechseln **um einen ganzen Takt verschoben**. Das mag in manchen Fällen unerheblich sein, weil die Sequenz (Zustands- und damit Ausgabefolge) ja weiterhin stimmt, aber für die Zusammenarbeit eines Automaten mit einem anderen ist es nicht akzeptabel.
Der Grund für dieses merkwürdige Verhalten liegt in einer schon erwähnten Spezialität von VHDL bei der Zuweisung von Signalen innerhalb eines getakteten *process*. Signalzuweisungen werden in einem solchen *process* **erst mit der nächsten Taktflanke** ausgeführt. Bis dahin behalten alle Signale ihren Wert. Man kann sogar ein- und demselben Signal nacheinander verschiedene Werte zuweisen, ohne dass dies (wie in der nebenläufigen Umgebung) zu einem Problem wird. Es wird immer nur die letzte Zuweisung ausgeführt.
Angenommen, der Automat sei jetzt gerade im Zustand *PULSE*. Dann wird mit der **nächsten** Taktflanke der Zustand *TERM* eingenommen (*cs <= term*) und mit dieser Taktflanke dann auch *p<='1'* und *w<='0'* gesetzt. Der Automat wird während des folgenden Taktes also im Zustand *TERM* sein, aber die Ausgaben in dieser Zeit gehören zu dem Zustand *PULSE*.
Sie sehen diese Verschiebung in Abbildung 4Abbildung 1 rot markiert. Die anderen Automatenbeschreibungen erzeugen dagegen die Ausgaben passend zu den internen Zuständen.
Der technische Hintergrund ist, dass Signale in getakteten Prozessen immer über ein Flipflop mit dem Prozesstakt laufen, d.h. man hat pro Signal einen weiteren Speicher.

## 2.4 Ein-Prozess-Darstellung mit Look Ahead

Das Problem der verzögerten Ausgabe kann man umgehen, indem man die gewünschten Ausgabesignale einen Zustand früher berechnet, so dass Zustandswechsel und Ausgabewechsel wieder synchron laufen. Von daher kommt der Name "Look Ahead": vorausschauend". Listing 5 zeigt eine entsprechende Beschreibung in VHDL.

```
architecture a4 of automat is
  type states is (idle, pulse, term);
  signal cs: states;

begin

-- Zustandsübergangsfunktion, Zustandsspeicher und Ausgabefunktion
-- mit look-ahead
  process(clk, r)
  begin
    if (r='0')
    then
      cs <= idle;
    elsif (rising_edge(clk))
    then
      case cs is
        when idle => if (a='0') then cs <= idle after 5ns; p<='0' after 5ns;w<='0' after 5ns;
                                else cs <= pulse after 5ns;p<='1' after 5ns;w<='0' after 5ns;
                     end if;

        when pulse => cs <= term  after 5ns;
                      p<='0' after 5ns; w<='1' after 5ns;

        when term => if (a='1') then cs <= term after 5ns;p<='0' after 5ns; w<='1' after 5ns;
                                else cs <= idle after 5ns;p<='0' after 5ns; w<='0' after 5ns;
                     end if;

      end case;
    end if;
  end process;

end;
```

**Listing 5: Ein-Prozess-Darstellung mit Look Ahead**

Diese Darstellung funktioniert bereits sehr gut. Die Ausgaben laufen synchron mit den inneren Zuständen und entsprechen damit den Entwurfsvorgaben. Zudem wird die Berechnung der neuen Ausgaben ja jetzt von der tatsächlichen Ausgabe durch das eingefügte Flipflop abgekoppelt. Die Berechnung selber wird, wie in den Methoden 1 und 2, pro Signal unterschiedlich lang dauern. Die neuen Werte werden aber erst mit der nächsten Taktflanke alle zur gleichen Zeit in die Flipflops übernommen. Damit werden sie auch **gleichzeitig** und **nach kurzer Zeit** (die Verzögerungszeit des Flipflops) nach der Taktflanke erscheinen.
Sie können das in Abbildung 3 sehr gut an den Signalen *p4* und *w4* sehen.
Diese Beschreibung scheint also für die Umsetzung sehr gut geeignet zu sein. Problematisch ist allerdings, dass in der Beschreibung die Ausgabewerte für den Zustand *x* an allen Vorgängerzuständen des Zustands *x* angegeben werden muss. Das ist erstens nicht sehr logisch und zweitens fehleranfällig - ändert sich die Ausgabebelegung des Zustands *x*, dann muss man diese Änderung ggf. an mehreren Stellen nachziehen.

## 2.5 Drei-Prozess-Darstellung

Die vierte Methode liefert ein sehr gutes Ergebnis, nur die Beschreibung ist ungünstig. Wenn man nun die Idee, dass man **sowohl den Folgezustand als auch die Folgeausgänge** in Flipflops speichert, konsequent umsetzt, dann bietet sich die Trennung in einen Speicherprozess und zwei Berechnungsfunktionen wie in den Methoden 1 bis 3 an.
In Listing 6 ist zunächst neu, dass es jetzt explizit einen aktuellen Zustand *cs* und einen Folgezustand *ns* gibt und dazu passend die aktuellen Ausgänge *p*, *w* und die Folgeausgänge *pn*, *wn*.

```
architecture a5 of automat is
  type states is (idle, pulse, term);
  signal cs, ns: states;
  signal pn, wn: std_logic;

begin
  -- Zustandsspeicher und Ausgabe der vorbereiten Ausgangssignale
  process(clk, r)
  begin
    if (r='0')
    then
      cs <= idle after 5ns;
      p<='0' after 5ns; w<='0' after 5ns;
    elsif (rising_edge(clk))
    then
      cs <= ns after 5ns;
      p <= pn after 5ns;
      w <= wn after 5ns;
    end if;
  end process;

  -- Zustandsübergangsfunktion
  process(cs,a)
  begin
    case cs is
      when idle =>    if (a='0')
                      then ns <= idle after 5ns;
                      else ns <= pulse after 5ns;
                      end if;

      when pulse =>   ns <= term;

      when term =>    if (a='1')
                      then ns <= term after 5ns;
                      else ns <= idle after 5ns;
                      end if;
    end case;
  end process;

  -- Ausgabefunktion
  process(ns)
  begin
    case ns is
      when idle =>    pn <='0' after 20ns; wn <= '0' after 20ns;
      when pulse =>   pn <='1' after 10ns; wn <= '0' after 20ns;
      when term =>    pn <='0' after 20ns; wn <= '1' after 10ns;
    end case;
  end process;

end;
```

**Listing 6: Drei-Prozess-Darstellung**

Zustände und der Ausgabesignale wechseln taktsynchron im ersten *process*.

Die Zustandsübergangsfunktion ist ebenfalls als *process* beschrieben. Sie hängt vom aktuellen Zustand und den Eingängen ab - daher die Signale *cs* und *a* in der Empfindlichkeitsliste.

Die Ausgabefunktion hängt aber hier vom Folgezustand *ns* ab. Jetzt kann man die Ausgabewerte wieder übersichtlich und an der richtigen Stelle (bei den jeweiligen Zuständen) angeben. Der Signalverlauf *p5*, *w5* ist mit dem von Methode 4 identisch (Abbildung 3).

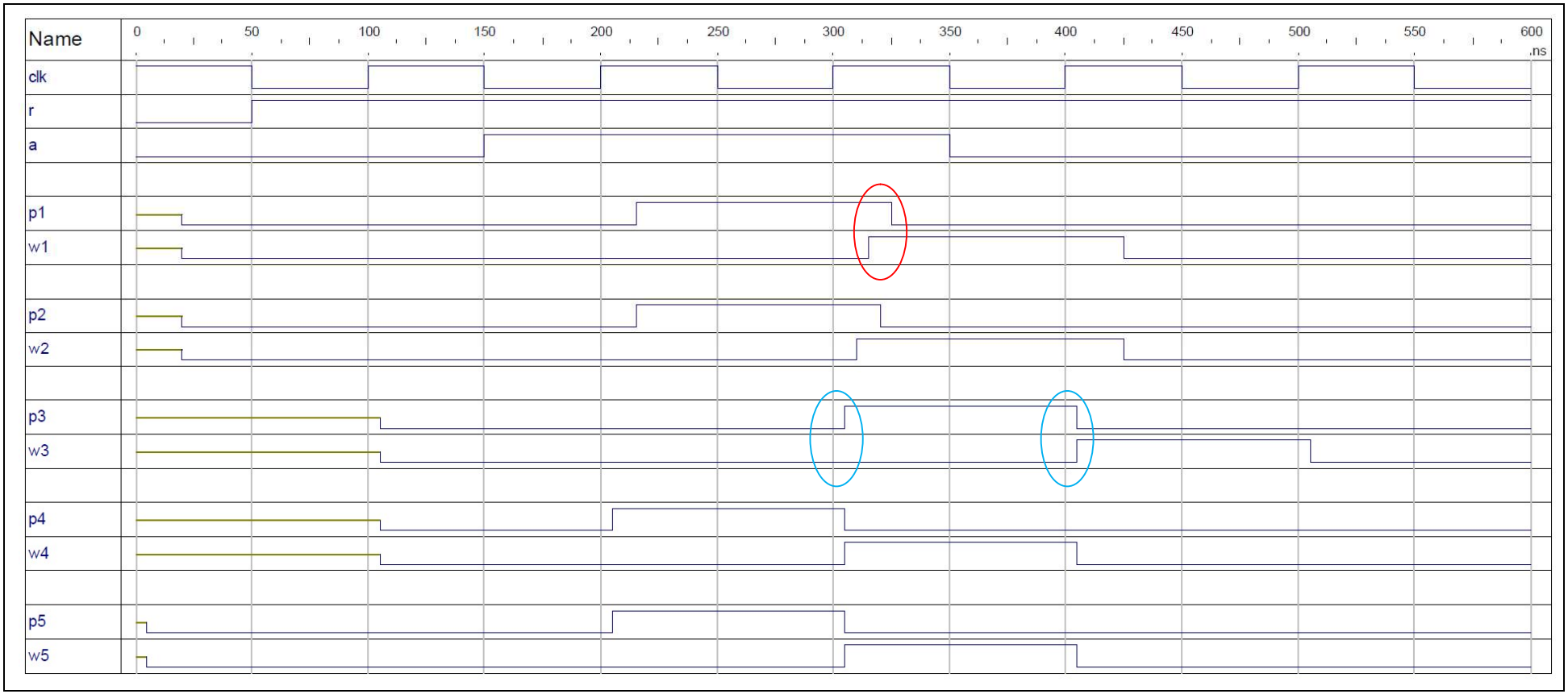


**Abbildung 3: Signalverläufe (Gruppierung nach Automat 1-5)**

Die Verzögerung von der steigenden Taktflanke zum Ausgang eines Flipflops beträgt hier einheitlich 5 ns. Diese Verzögerung sehen Sie z.B. an den hellblau markierten Stellen. Die Verzögerung durch Kombinatorik (Gatter) beträgt hier 10 ns für einen Wechsel von 0 nach 1 und 20 ns für den Wechsel von 1 nach 0. Solche ungleichen Verzögerungszeiten sind auch in der realen Schaltung möglich, weil beispielsweise Transistoren unterschiedlich schnell schalten können.
Die rot markierte Stelle zeigt einen unerwünschten Effekt: Für eine kurze Zeit erscheint am Ausgang eine Wertekombination (*p=1*, *w=1*), die im Automatenentwurf gar nicht vorgesehen ist. Das kann nachfolgende Schaltungen stören.

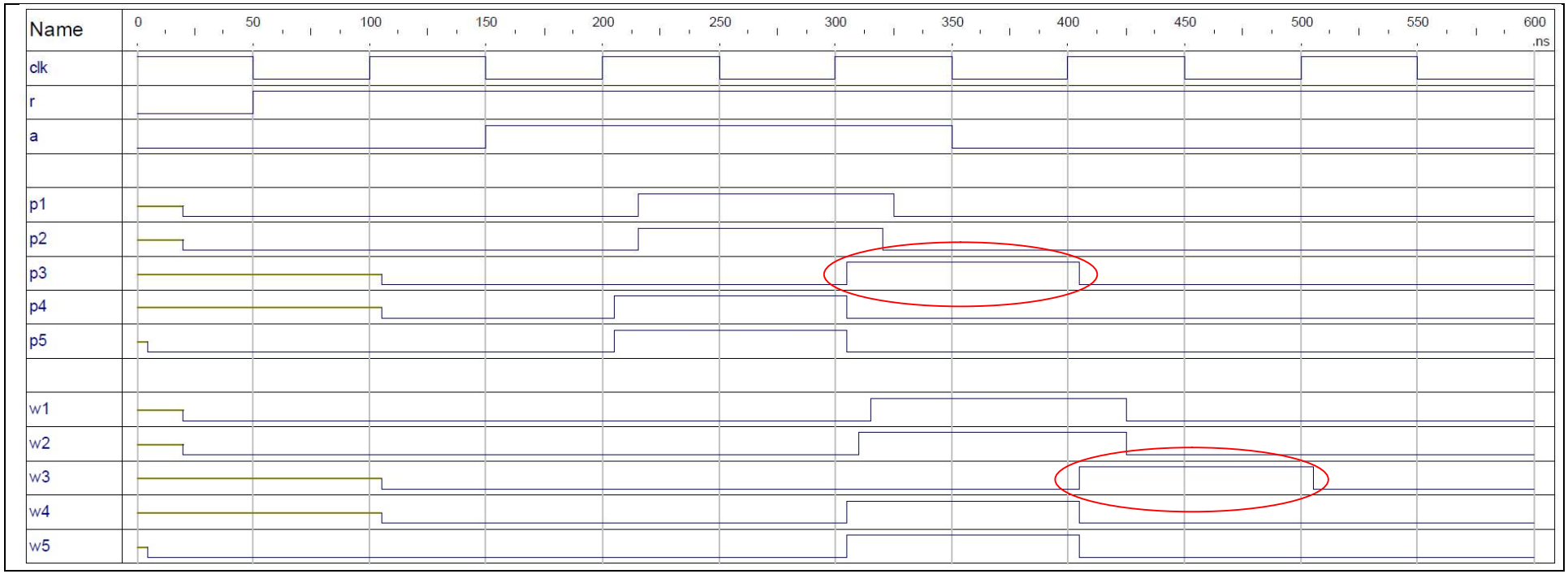


**Abbildung 4: Signalverläufe (gruppiert nach Ausgangssignal p und w)**

Hier sehen Sie die Verschiebung der Ausgabesignale durch einen zusätzlichen Ausgabespeicher. Die rot markierten Signale gehören zu dem Zustand, den der Automat im **Takt davor** eingenommen hat. Sämtliche Ausgabesignale sind um einen Takt verschoben. Für die Sequenz hat das keine Bedeutung, aber die Reaktion eines von diesen Signalen abhängigen Moduls kommen natürlich dann immer um einen Takt verzögert. Wenn der Automat also schon im nächsten Takt die Antwort eines solchen Moduls auswerten will, dann wird es zu einem Fehler kommen.
Zum Vergleich: Die anderen Automatenbeschreibungen (Signale *p1-p3, p5* bzw. *w1-w3, w5*) liefern die Ausgaben einen Takt früher.