Fachhochschule Wiesbaden UNIVERSITY OF APPLIED SCIENCES
› INGENIEURWISSENSCHAFTEN

Fachhochschule Wiesbaden

Studienbereich
Informationstechnologie und Elektrotechnik

Studiengang
Fernsehtechnik und elektronische Medien

Diplomarbeit

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

vorgelegt von:

am (Stempel des Studienbereichs)

Referent/in: Professor Apfelbeck
Korreferent/in: Dipl.-Ing. Einanderer

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Ich versichere hiermit, diese Diplomarbeit selbständig angefertigt zu haben sowie, dass alle verwendeten Quellen und Hilfsmittel in der Arbeit angegeben sind.

Der Einsicht in diese Diplomarbeit und der Ausleihe eines Exemplares
stimme ich zu / stimme ich nicht zu*.

_____ _____

(Ort / Datum) (Unterschrift Studentin / Student)

Nur von der Betreuerin / von dem Betreuer auszufüllen :

Gegen die Einsicht in diese Diplomarbeit und gegen die Ausleihe eines Exemplars
wird / kein* Einspruch erhoben.

_____

(Unterschrift Betreuerin / Betreuer)

Begründung (bei Einspruch) :

_____

_____

_____

_____

* : Nichtzutreffendes bitte streichen.

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Vorwort

In diesem Bericht wird die Anbindung kryptografischer Komponente beschrieben. Diese Anbindung entsteht zwischen drei Komponenten : Microprocessor, Adapter und Hardware .Bei dieser Beschreibung wird VHDL_Text und State machinen verwendet um diese Anbindung leicht zu programmieren .Dadurch kann man die Anbindung leicht verstehen und programmieren, weil man die Änderung der Zustandsdiagramm deutlich sehen kann .Für diese Anbindung braucht man zwei Zustandsdiagramme , eines für Anbindung zwischen Microprocessor ,Adapter und Hardware in dieser Reihenfolge und das zweite für den Rückwärts (von Hardware bis Microprocessor).

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Inhaltsverzeichnis

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Einführung

Damit man diese Beschreibung verstehen zu können , sollte man die Funktion der Zuastandsdiagramme verstehen und einen Überblick über die Hardware-Programiersprache VHDL hat.

Was ist VHDL?

Very High Speed Integrated Circuit Hardware Description Language(auch VHSIC Hardware Description Language), kurz VHDL, ist eine Hardwarebeschreibungs-sprache, vergleichbar mit einer Programmiersprache, mit der es einfach möglich ist, komplizierte digitale Systeme zu beschreiben.Darüber hinaus gibt es sprachliche Erweiterungen in Form von VHDL_AMS, mit welcher auch analoge Systeme beschrieben werden können.

Bei VHDL arbeitet man nicht mit einzelnen elektronische Bauteilen, sondern beschreibt das gewünschte Verhalten einer Schaltung auf einer höheren Abstraktionsebene.VHDL ermöglicht das schnelle Entwickeln großer und komplexer Schaltungen (z.B. mikroprozessor mit über 20 Mio.Transistoren),die hohe Effizienz erfordern(zeitlich wie ökonomisch)und unterstützt den Entwickler bei allen Arbeiten.

So kann ein System simuliert, verifiziert und schließlich eine Netzliste erstellt werden.

Aus der Netzliste können Masken für die Herstellung von MPGAs(mask program-mable gate array) oder ähnlichen LSI(Large scale integration)-Chips produziert werden oder sie kann(nach Konvertierung in einen geeigneten Bitstream direkt in ein FPGA geladen werden.

Ein FPGA ist ein Programmierbarer Inegrierter Schalkreis (IC) der Digitaltechnik. Die Englische Abkürzung steht für Field Programmable Gate Array und kann als,,Vor Ort modifizierbarer Logikbaustein“übersetzt werden.In FPGAs können durch spezifische Konfiguration interner Strukturen die verschiedenartigsten Schaltungen gebildet werden.Diese reichen von geringer komplexität, wie z.B.einfacher Synchronzähler, bis zu hochkomplexen Schaltungen,wie Microprozessoren.FPGAs werden in allen Bereichen der Digitaltechnik eingesetzt, vor allem aber dort, wo es auf schnelle Signalverarbeitung und flexible Änderung der Schaltung ankommt, um beispielsweise nachträgliche Verbesserungen an den implementierten Funktionen vornehmen zu können, ohne dabei direkt die physikalische Hardware ändern zu müssen.

Mit FPGAs wurden kompakte anwenderspezifische Schaltungen in geringen Stückzahlen möglich.Heutzutage gestatten sie die preiswerte und flexible Fertigung Komplexer Systeme wie Mobilfunk-Basisstationen als Alternative zur teureren Auftragsfertigung durch Halbleiterhersteller.

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Neben den FPGAs existieren auch FPAAs(Field programmable Analog Array) die Keine digitaln, sondern analoge Funktionsblöcke enthalten,die vom Anwender programmiert und verschaltet werden können.

Funktionweise der Zustandsdiagramm(englisch state diagram):

Der Begriff Zustandsdiagramm bezeichnet die grafische Darstellung von Zuständen und daran anknüpfenden Zustandswechseln. Es ist eine der dreizehn Diagrammarten dieser Modellierungssprache für Software und andere Systeme.Es stellt einen endlichen Automaten in einer Sonderform grafisch dar und wird benutzt, um entweder das Verhalten eines Systems oder die zulässige Nutzung der Schnittstelle eines Systems zu spezifieren.

Ein Zustand modelliert eine Simulation, in der eine bestimmte unveränderliche Bedingung gilt. Meistens ist diese Invariante nur implizit gegeben, will man sie explizit formulieren, kann man sie als Einschränkung dem Zustand zuordnen.

Dem Zustand können drei Verhaltensspezifikationen, zum Beispiel in Form einer Aktivität oder einer Interaktion, zugeordnet werden:

- Ein Verhalten, das ausgeführt wird, wenn der Zustandsautomat in den Zustand eintritt(engl.entry behaviour)
- Ein Verhalten, das ausgeführt wird, wenn der Zustandsautomat den Zustand verlässt(engl.exit behaviour).
- Ein Verhalten, das ausgeführt wird , während sich der Zustandsautomat im zustand befindet (engl.doActivity).

  Graphisch wird ein Zustand meistens als Kreis oder Rechteck mit abgerundeten Ecken dargestellt, leicht unterschiedliche Darstellungsformen sind auch möglich.

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

1. Zustandsdiagramm für die Anbindung von Mikroprozessor zu Hardware:

Die erste vorstellung für die Anbindung zwischen Mikroprozessor, Adapter und Hardware wurde in einem Zustandsdiagramm dargestellt .Dies besteht aus drei zustände, die das Verhalten oder Funktion jedes Komponentes beschreibst (siehe das Bild unten):

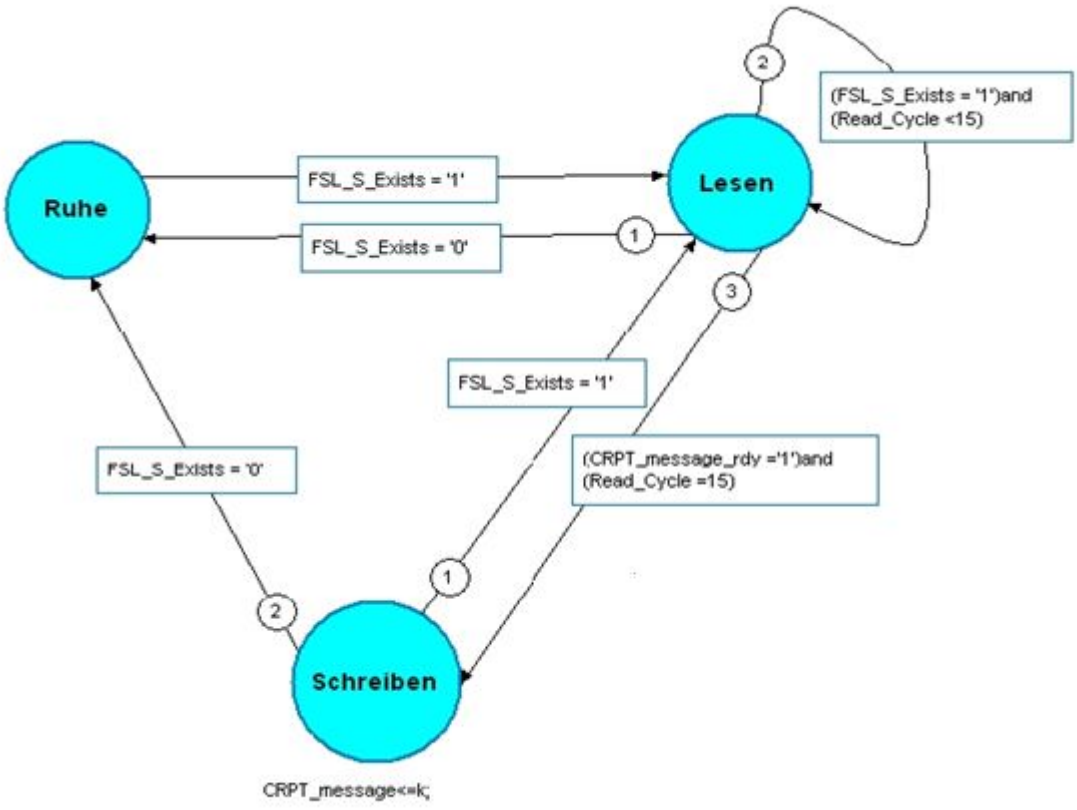


Abbildung 1.1: Zustandsdiagramm von Microprozessor zum Hardware

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

hier sind drei Zustände und die Verbindung dazwischen, der erste zustand ist die Ruhezustand (oder Mikroprozessor _zustand) der zweite ist das Lesen (oder Adapter_zustand) und der dritte ist das schreiben (Hardware_zustand). dazwischen gibt es Signale, die entweder auf 0 oder 1 eingesetzt sind und drücken das verhalten zwischen je zwei Zustände aus. Um die Bedeutung dieser Signale richtig zu verstehen, werden diese unten erklärt:

FSL_S_Exists: es bedeutet ob da Daten gibt (ZB: bei FSL_S_Exists ='1' es gibt Daten und bei FSL_S_Exists='0', keine Daten).

CRPT_message_rdy: bedeutet Hardware hat noch platz um andere Daten zu Schreiben.

CRPT_message: enthält die gesamten Daten, die in einem Speicher K gespeichert sind.

Read_Cycle: ist als variable definiert und wird für Zähler benutzt.

FSL_S_data: entspricht 32bits.

Der Speicher K: in dem befindet sich die gesamten Daten (512 Bits).

In diesem Zustandsdiagramm sollen 512 Bits zum Hardware geschickt und dort werden diese Bits auf 256 Bits reduziert, aber diese 512 sind nicht auf einmal gelesen sondern werden als 16 mal 32 Bits gelesen und gespeichert. Diese 32 Bits werden durch FSL_S_data ersetzt.

Wenn da Daten gibt es (FSL_S_Exists='1'), werden je 32 Bits zum Adapter geschickt , aber wenn FSL_S_Exists='0' ist, geht es wieder zum Ruhezustand weil da keine Daten enthalten sind.

Im Adapter werden jede 32 Bits gelesen und in einem Register K gespeichert ,der durch das Variable Read_cycle ersetzt ist . Dieser Zähler soll die 32 Bits 16-mal zählen dh soll 512 Bits insgesamt gespeichert werden. Deswegen hat man diesen Zähler im Adapter verwendet.

Wenn im Zustandsdiagramm oben im Bild Read_Cycle <15 ist bzw Adapter lesbar ist, bleibt der Zustand immer beim Lesen aber wenn Read_cycle=15 bzw im Adapter schon 512 Bits gespeichert sind ,werden diese zum Schreiben-zustand

geschickt aber das passiert nur wenn CRPT_message_rdy='1' ist (Hardware hat noch platz zu schreiben).

wenn alle 512 Bits in der Hardware sich befinden , werden diese in einem Register CRPT_message gespeichert(CRPT_message<=K).

1.1 Deklaration des Codes:

bei dem Simulieren hat dieses Zustandsdiagramm nicht funktioniert wie man das vorgestellt hat, deswegen sollte die Deklaration vieler Signale in der Entity geändert werden (siehe die Entity unten).

```
ENTITY crpt_adapter IS
 PORT(
CRPT_Clk          : IN     std_logic;
CRPT_Rst          : IN     std_logic;
CRPT_message      : OUT    std_ulogic_vector (511 DOWNTO 0);
CRPT_message_len  : OUT    std_ulogic_vector (9 DOWNTO 0);
CRPT_message_rdy  : in std_ulogic;
CRPT_next_message : in std_ulogic;
CRPT_result   : in  std_ulogic_vector (255 DOWNTO 0);
CRPT_result_rdy  : in std_ulogic
 FSL_Clk : IN  std_logic;
 FSL_Rst : IN  std_logic;
 FSL_S_Clk : OUT std_logic;
 FSL_S_Read : OUT std_logic;
 FSL_S_Data : IN std_ulogic_vector (0 TO 31);
 FSL_S_Control  : IN  std_logic;
 FSL_S_Exists  : IN std_logic;
```

```
FSL_M_Clk : OUT  std_logic;
FSL_M_Write : in  std_logic;
FSL_M_Data  : out  std_ulogic_vector (0 TO 31);
FSL_M_Control   : OUT    std_logic;
FSL_M_Full      : IN     std_logic  );
attribute SIGIS : string;
attribute SIGIS of FSL_Clk   : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";
END crpt_adapter ;
```

Zu dieser DeklarationsÄnderungen, die in Entity gemacht sind, sollten auch die Signale und variable definiert werden. Das kann man in Deklarationsblock machen auf dem klicken mit dem rechtmaus auf State Maschine Properties. (siehe das Fenster unten).

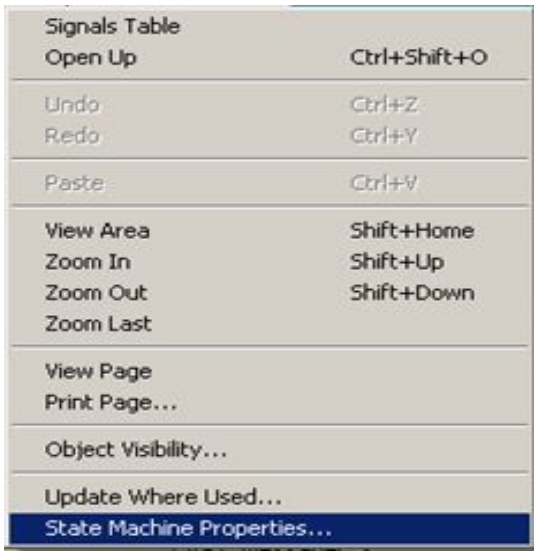


Abbildung1.2.1: State Machine Properties

Nachdem klicken auf State Maschine Properties kommt das zweite Fenster, wo man die Signale Definieren kann:

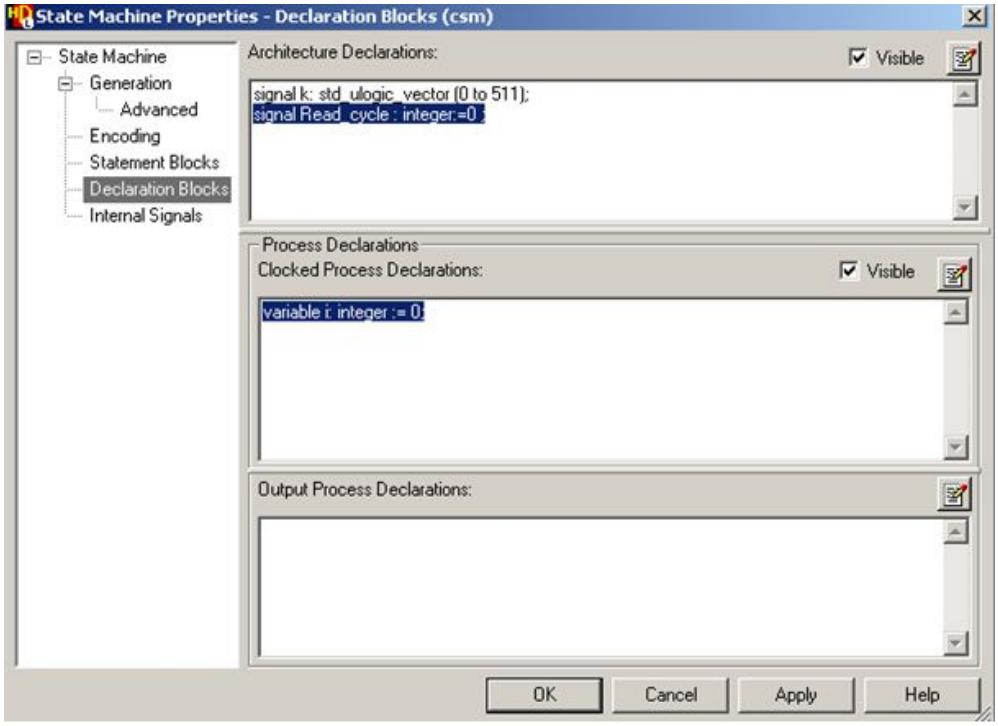


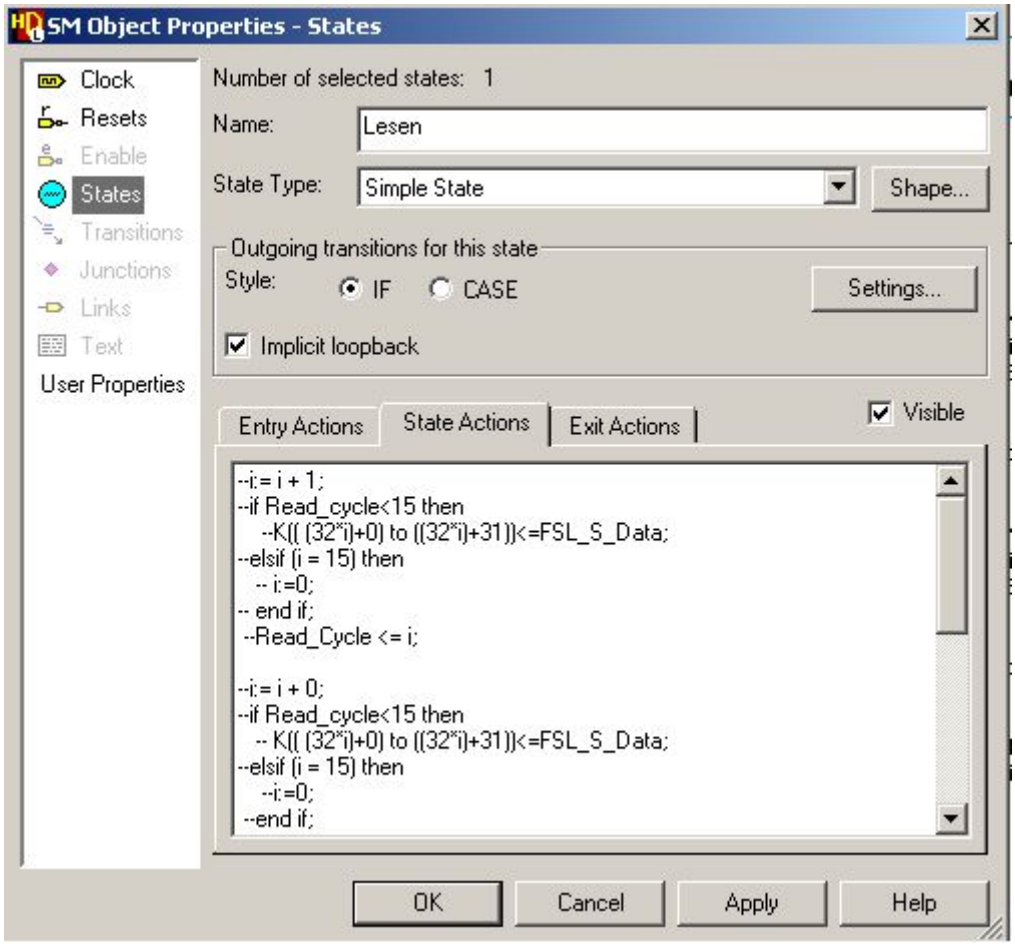
Abbildung 1.2.2: DeklarationsBlocks

In diesem fenster oder in Deklaration Blocks hat man die Möglichkeit die Signale in Architektur als lokale Signale zu deklarieren (in unserem Beispiel hat man der Speicher K als ulogin-vector(0 to 511) und der Variable Read_cycle als integer=0 definiert).Oder in Process_ Block (Zb Variable i:integer:=0).

1.2 Programmieren in Zustandsdiagrammramm:

Nach der Deklaration soll man innerhalb jeder State Maschine das beliebige programm schreiben . In dem fenster unter(zweimal klicken auf Lesen) kann man im Entry, State, oder Exit-Actions das programm schreiben.

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .



In unserem Beispiel hat man im lesen zustand ein kleines Programm geschrieben um die Funktion des Zählers zu beschreiben. Mit dem klicken zweimal auf lesen _zustand hat man dieses Programm im State Actions geschrieben:

```
i:=0;
Datenschleife: for i in 0 to 15 loop
i:=i+1;
--    Read_Cycle:=i;
--   K(( (32*i)+0) to ((32*i)+31))<=FSL_S_Data;
end loop;
 K(( (32*i)+0) to ((32*i)+31))<=FSL_S_Data;
```

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Das ist eine loopschleife von 0 bis 15, da soll jedes Mal 32bits gespeichert werden.

Aber bei der Simulation hat diese schleife nicht funktioniert und auch die Daten sind nicht unter K gespeichert wie man das vorgestellt hat, da hat man statt loopschleife einen Zähler benutzt um die Simulation zu funktionieren.

Dabei hat man mehrere versuche gemacht um den Zähler zu funktionieren:

```
--i:= i + 1;
--if Read_cycle<15 then
   --K((32*i)+0) to ((32*i)+31))<=FSL_S_Data;
--elsif (i = 15) then
  -- i:=0;
-- end if;
--Read_Cycle <= i;

--i:= i + 0;
--if Read_cycle<15 then
  -- K((32*i)+0) to ((32*i)+31))<=FSL_S_Data;
--elsif (i = 15) then
   --i:=0;
--end if;
-- Read_Cycle <= i;

i:= i + 1;
if Read_cycle<15 then
   --K((32*i)+0) to ((32*i)+31))<=FSL_S_Data;
elsif (i = 15) then
 i:=i+0;
 end if;
Read_Cycle <= i;

--i:= i + 1;
--if Read_cycle<15 then
   --K((32*i)+0) to ((32*i)+31))<=FSL_S_Data;
--elsif (i = 15) then
 -- i:=0;
--end if;
-- Read_Cycle <= i;
```

Dazu hat man auch die Deklaration der Signale und variable geändert ZB: Read_cycle wurde als Signal in Architekturen deklariert (Signal Read_cycle: integer:=0), und i als variable im Process definiert (variable i: integer:= 0). Das Signal FSL_S_Exists sollte auch als ((fsl_s_exists'event) and (fsl_s_exists='1')) geändert .Das richtige Zustandsdiagramm sieht jetzt so aus:

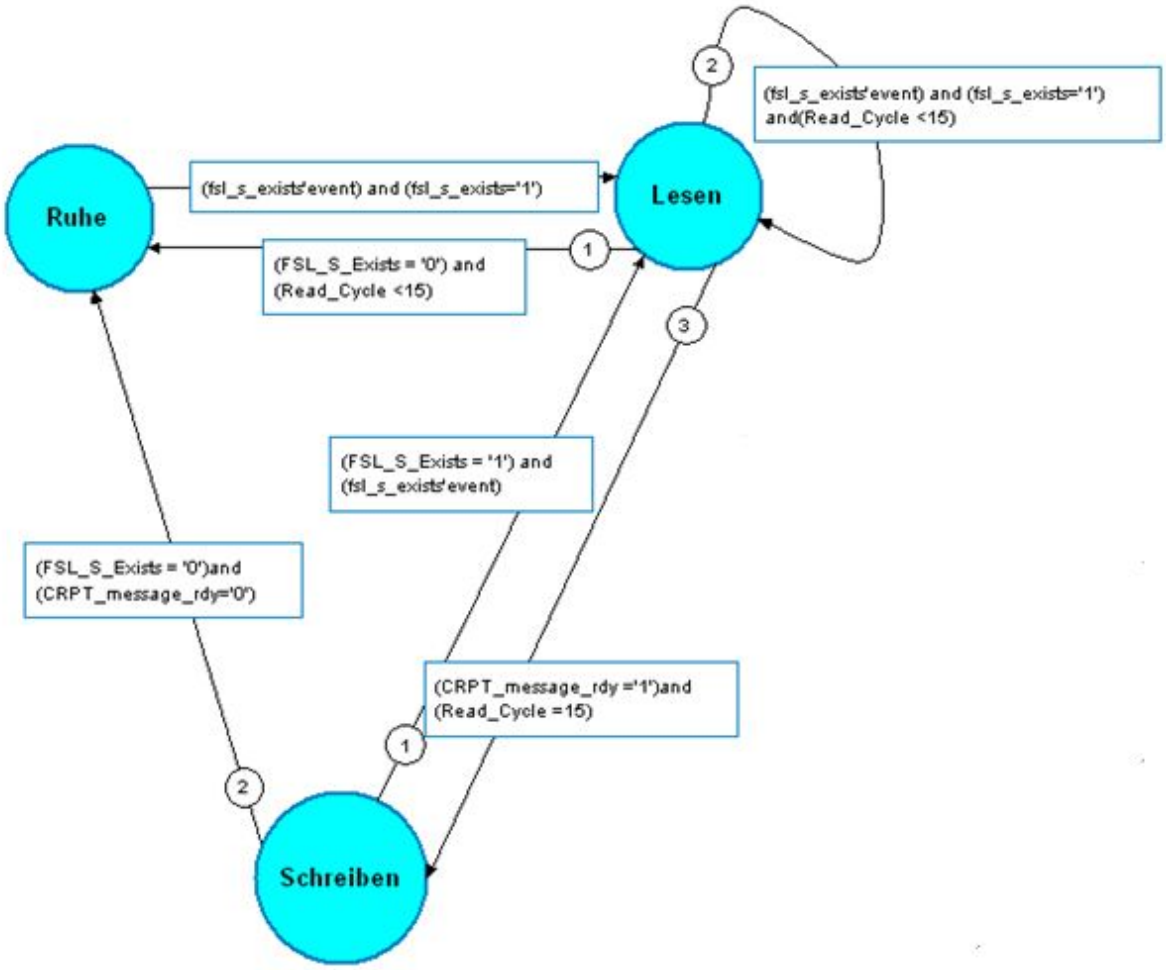


Abbildung 1.3: Zustandsdiagramm mit dem clock .

Nach der Deklarationsänderung hat die Simulation trotzdem nicht ganz funktioniert. Weil die Daten nicht im K gespeichert werden dh die Daten werden nicht zum Speicher k geschickt obwohl der Zähler gut funktioniert. Der Fehler war, dass das Programm, das im lesen zustand geschrieben ist, unter dem Clocked_process geschrieben wurde, aber das soll nicht sein und damit die Simulation funktioniert, musste man ein VHDL-text schreiben außerhalb des Zustandsdiagramms.

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

1.3 VHDL_Text1:

Das VHDL_Text ,das für dieses Zustandsdiagramm gedacht ist ,besteht aus Einer Entity und einer Architektur .Unter der Architektur gibt es Processe ,die das Verhalten jeder State maschine beschreibt

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;  -- haben wir geändert weil + net funkt.
ENTITY crpt_adapter IS
  PORT(
    CRPT_Clk : IN  std_logic;
    CRPT_Rst : IN std_logic;
    CRPT_message  : OUT std_ulogic_vector (511 DOWNTO 0);
    CRPT_message_len  : OUT std_ulogic_vector (9 DOWNTO 0);
    CRPT_message_rdy  : in  std_ulogic;
    CRPT_next_message : IN  std_ulogic;
    CRPT_result   : IN  std_logic_vector (255 DOWNTO 0);
    CRPT_result_rdy : IN  std_ulogic;
    FSL_Clk  : IN std_logic;
    FSL_Rst : IN  std_logic;
  FSL_S_Clk : OUT std_logic;
  FSL_S_Read  : OUTstd_logic;
```

```
  FSL_S_Data  : IN std_ulogic_vector (0 TO 31);
 FSL_S_Control : IN std_logic;
 FSL_S_Exists  : IN std_logic;
 FSL_M_Clk  : OUT std_logic;
 FSL_M_Write : OUT std_logic;
 FSL_M_Data  : OUT std_logic_vector (0 TO 31);
FSL_M_Control  : OUT std_logic;
FSL_M_Full  : IN  std_logi);
attribute SIGIS : string;
attribute SIGIS of FSL_Clk   : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";
END crpt_adapter ;
ARCHITECTURE fertig1 OF crpt_adapter IS
  signal k: std_ulogic_vector (0 to 511);
  signal Read_cycle : integer:=0 ;
  signal Time_out : integer:=0 ;
  TYPE STATE_TYPE IS (
    Ruhe,
    Lesen,
    Schreiben
  );
  SIGNAL current_state : STATE_TYPE;
  SIGNAL next_state : STATE_TYPE;
```

```
BEGIN
clocked_proc : PROCESS (
   CRPT_Clk,
   CRPT_Rst
)
variable d: integer := 0;
BEGIN
IF (CRPT_Rst = '1') THEN
current_state <= Ruhe;
ELSIF (CRPT_Clk'EVENT AND CRPT_Clk = '1') THEN
current_state <= next_state;
 ELSIF (Read_Cycle >0 and fsl_s_exists='0') THEN
 d:=d+1;
 Time_out <=d;
 END IF;
END PROCESS clocked_proc;
nextstate_proc : PROCESS (
   CRPT_message_rdy,
   FSL_S_Exists,
   Read_cycle,
   current_state
)
   BEGIN
   CASE current_state IS
```

```
 WHEN Ruhe =>
    IF ((fsl_s_exists'event) and (fsl_s_exists='1')) THEN
    next_state <= Lesen;
    ELSE
    next_state <= Ruhe;
    END IF;
  WHEN Lesen =>
    IF ( Time_out >50)   THEN
    next_state <= schreiben;
    ELSIF ((fsl_s_exists'event) and (fsl_s_exists='1')
    and(Read_Cycle <15)) THEN
    next_state <= Lesen;
    ELSIF ((CRPT_message_rdy ='1')and
    (Read_Cycle =15)) THEN
    next_state <= Schreiben;
   ELSE
   next_state <= Lesen;
   END IF;
  WHEN Schreiben =>
 IF ((FSL_S_Exists = '1') and
(fsl_s_exists'event)and (Time_out = 10)) THEN
next_state <= Lesen;
 ELSIF ((FSL_S_Exists = '0')and (fsl_s_exists'event)and (Time_out = 10)) THEN
 next_state <= Ruhe;
```

```
      ELSE
      next_state <= Schreiben;
      END IF;
      END CASE;
      END PROCESS nextstate_proc;
      states : process (
      FSL_S_Exists
     )
      variable i: integer := 0;
      Begin
      CASE current_state IS
      WHEN Lesen =>
     if Read_cycle<15 and ((FSL_S_Exists = '1') and
    (fsl_s_exists'event))then
      K(( (32*i)+0) to ((32*i)+31))<=FSL_S_Data;
      i:= i + 1;
      elsif (i = 16) then
      i:=0;
    elsif (CRPT_Clk'EVENT AND CRPT_Clk = '1') then
      end if;
      Read_Cycle <= i;
    WHEN Schreiben =>
if (CRPT_Clk'EVENT AND CRPT_Clk = '1') then
CRPT_message<=k;
```

```
 end if;
 WHEN OTHERS =>
 NULL;
END CASE;
END PROCESS states;
END fertig1;
```

1.4 Fehlerkorrektur im VHDL_Text:

Probleme 1:

Im Programm ist der Fehler aufgetreten, dass das Time_out der limit erreicht hat obwohl die Pakete noch am kommen sind siehe:

Bild :1p1

do File : 1-liest 15 mal und schreibt 512mbraus.

Also die lösung ist, dass man dieser Time_out erhöht. Natürlich ist das nicht die Perfekte Lösung denn am besten wäre es besser, wenn der Time_out jedes mal neu beginnt wenn ein Paket ankommt und erst da änfangt zu Zählen. Da man es leider noch nicht hinkriegt, wird der Time_out erstmal erhöht werden .

Time_out gleich 0 setzen ist schwere als gedacht denn d=0 im lesen :

```
WHEN Lesen =>
 if Read_cycle<15 and ((FSL_S_Exists = '1') and (fsl_s_exists'event))
then
K(( (32*i)+0) to ((32*i)+31))<=FSL_S_Data;
 i:= i + 1;
d:=0;
```

es hat aber das Problem leider nicht gelöst deswegen hat man versucht, das Time_out gleich 0 zu setzen sobald sie eine gewisse höhe erreicht, leider auch ohne erfolg.

```
ELSIF (Read_Cycle >0 and fsl_s_exists='0') THEN

d:=d+1;

 elsif (Time_out = 60)

then

  d:=0;

  END IF;
```

Vermute das Fehler liegt irgendwo im Clock Prozess denn während Time_out =60 ist ,ist auch Read_Cycle >0 und FSL_S_Exists = 0 also würde das programm den Ersten Elseif nehmen daher die lösung ist es :

```
elsif (Time_out = 60) then

                d:=0;
```

Nach oben zu verschieben vor der anderen Elsif bedingungen oder noch viel besser man benutzt Else, denn in allen anderen zuständen braucht man kein Time_out und somit Fängt das Time_out zu zählen nach das ankommen von jeden Paket neu siehe:

Bild : 1-p-2

do File : 1-liest 15mal und schreibt 512mb raus

nach der Korrektur dieses Fehlers , hat das Programm funktioniert und die Simulation sieht besser als vorher aus.

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

1.5 Simulation 1:

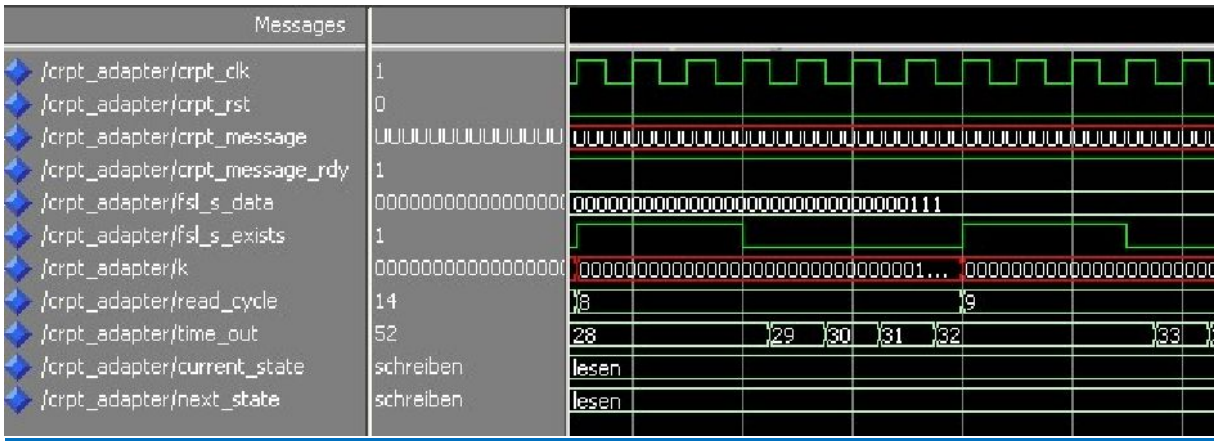


Abbildung 2.3.1 : Simulation für die Anbindung von Microprozessor zu Hardware.

Im Bild läuft die Simulation, hier sieht man, dass der Zähler richtig zählt ab 0,1,2….bis 15 und gleichzeitig werden die Daten von Crpt_message in k gespeichert dh jedes mal werden 32 Bits zum K geschickt ,wenn FSL_S_Exists ='1' und read_cycle<15, wechselt sich der lesen –zustand nicht erst wenn Read-cycle='15' dann springt es auf schreiben_Zustand .Aber wenn FSL_S_exists='0'dann geht es wieder zum ruhe-zustand .

2 Zustandsdiagramm für den Rückwärts:

Jetzt soll ein Zustandsdiagramm für den Rückwärts gemacht werden, bei dem Rückwärts hat man das Diagramm so vorgestellt:

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

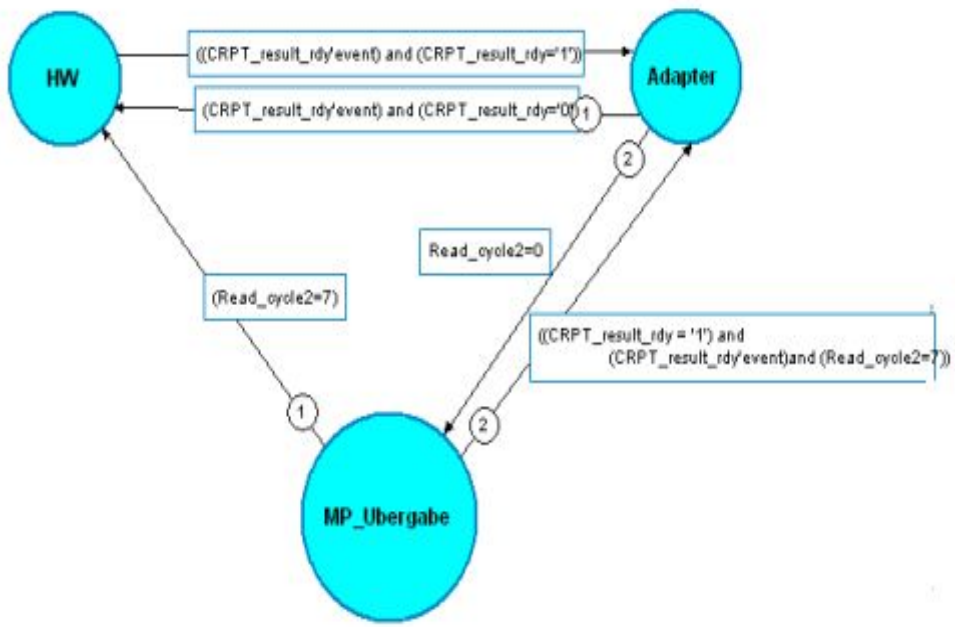


Abbildung **3.1:** Zustandsdiagramm von Hardware zum Microprozessor

Crpt_result_rdy='1' : es gibt Daten .

Crpt_result_rdy='0' : es gibt keine Daten.

Read_cycle2 : ein Signal wird für den Zähler benutzt.

Hier hat man keinen Fehler bei Deklaration gemacht. Nach den Problemen, die man bei dem ersten Zustandsdiagramm getroffen hat, werden diese bei dem Rückwärts_Zustandsdiagramm korrigiert Aber trotzdem gab es andere Probleme bei do file während der Simulation.

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

2.1 VHDL_text2 von dem Rückwärts

```
ENTITY crpt_adapter IS
  PORT(
    CRPT_Clk         : IN     std_logic;
    CRPT_Rst         : IN     std_logic;

    CRPT_message     : OUT    std_ulogic_vector (511 DOWNTO 0);
    CRPT_message_len : OUT    std_ulogic_vector (9 DOWNTO 0);
    CRPT_message_rdy : in     std_ulogic;     -- haben wir geändert von out zu in
    CRPT_next_message : IN    std_ulogic;
    CRPT_result      : in     std_ulogic_vector (255 DOWNTO 0);
    CRPT_result_rdy  : IN     std_ulogic;
    FSL_Clk          : IN     std_logic;
    FSL_Rst          : IN     std_logic;
    FSL_S_Clk        : OUT    std_logic;
    FSL_S_Read       : OUT    std_logic;
    FSL_S_Data       : IN     std_ulogic_vector (0 TO 31);
    FSL_S_Control    : IN     std_logic;
    FSL_S_Exists     : IN     std_logic;
    FSL_M_Clk        : OUT    std_logic;
    FSL_M_Write      : in     std_logic;
    FSL_M_Data       : out    std_ulogic_vector (0 TO 31);
    FSL_M_Control    : OUT    std_logic;
```

```
    FSL_M_Full       : IN     std_logic
  );

END crpt_adapter ;
 ARCHITECTURE fertig2 OF crpt_adapter IS
  signal m: std_ulogic_vector (0 to 255);
  signal Read_cycle2 : integer:=0 ;
  signal Time_out2 : integer:=0 ;
  TYPE STATE_TYPE IS (
    HW,
    Adapter,
    MP_Ubergabe
  );
  SIGNAL current_state : STATE_TYPE;
  SIGNAL next_state : STATE_TYPE;
  BEGIN
  clocked_proc : PROCESS (
    CRPT_Clk,
    CRPT_Rst
  )
 variable c: integer := 0;
  BEGIN
    IF (CRPT_Rst = '1') THEN
    current_state <= HW;
```

```
  ELSIF (CRPT_Clk'EVENT AND CRPT_Clk = '1') THEN
  current_state <= next_state;
 ELSIF Read_Cycle2 >0   THEN
c:=c+1;
 Time_out2 <=c;
 END IF;
END PROCESS clocked_proc;
 nextstate_proc : PROCESS (
 CRPT_next_message,
 CRPT_result_rdy,
 FSL_M_Write,
 Read_cycle2,
  current_state
)
  BEGIN
  CASE current_state IS
   WHEN HW =>
   IF ((CRPT_result_rdy'event) and (CRPT_result_rdy='1')) THEN
   next_state <= Adapter;
   ELSE
   next_state <= HW;
   END IF;
   WHEN Adapter =>
   IF ((CRPT_result_rdy'event) and (CRPT_result_rdy='0'))
```

```
THEN
 next_state <= HW;
        Elsif ( Read_cycle2=0) then
        next_state <= MP_Ubergabe;
         ELSE
         next_state <= Adapter;
         END IF;
         WHEN MP_Ubergabe =>
         IF  ((CRPT_result_rdy = '1') and
        (CRPT_result_rdy'event)and (Read_cycle2=7)) then
        next_state <= Adapter;
        ELSIF  (Read_cycle2=7) THEN
        next_state <= HW;
        END IF;
        WHEN OTHERS =>
        next_state <= HW;
        CASE;
        END PROCESS nextstate_proc;
        states : process (
        current_state,
        CRPT_Clk,
        CRPT_result_rdy
        )
       variable I: integer := 0;
```

```
        Begin
        CASE current_state IS
 WHEN Adapter =>
 m <= CRPT_result;
 WHEN MP_Ubergabe =>
 FSL_M_Data<=m(((32*L)+0) to ((32*L)+31));
 L:= L + 1;
 if (L = 8) then
 L:=0;
 end if;
 Read_Cycle2 <= L;
 WHEN HW =>
 Read_Cycle2 <=0;
 L:=0;
WHEN OTHERS =>
NULL;
END CASE;
END PROCESS states;
END ARCHITECTURE fertig2;
```

2.2 Simulation des zweiten Programmes

2.2.1 Fehlererkennung bei der Simulation :

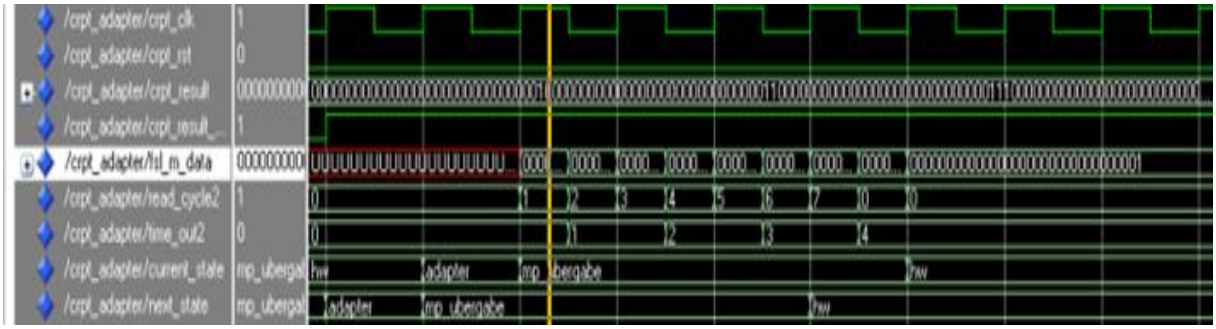
Also bei der 2 State maschine hat man ein Problem getroffen ,dass wenn man von Crpt_Result zu FSL_M_Data kopiert ,kriegt man 9 Pakete also ein Paket zu viel da wenn Read_Cycle2 von 7 auf 0 springt schreibt das Programm noch mal das erste Paket (siehe Bild 2_P_1):

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

Bild: To –do File : 2-p-1

Da in unser Problem die richtigen Dateien auf den dazu zugeordneten platz geschrieben werden kann Man dieses Problem vorerst vernachlässigen.

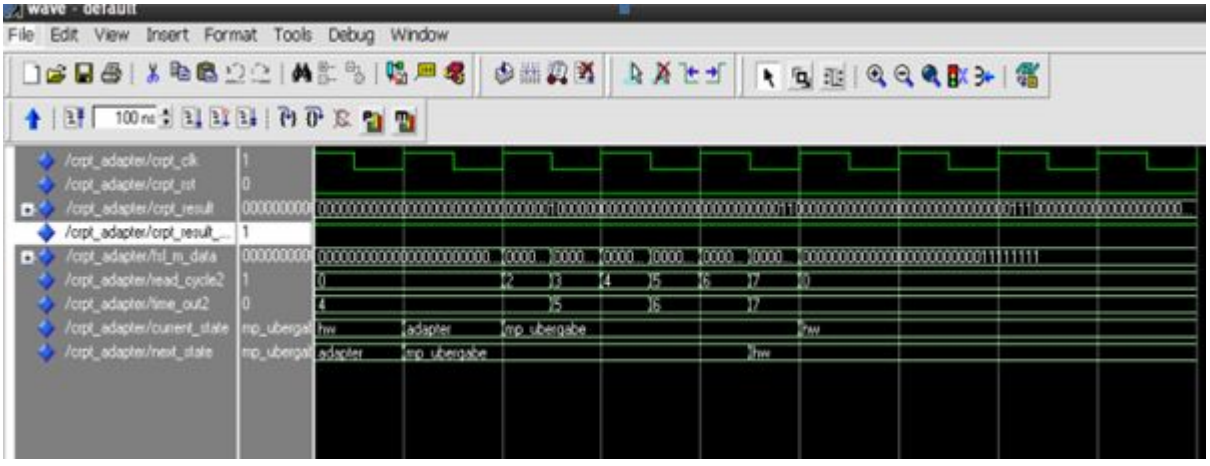
Simulation von Bild 2_P_1:



Wenn man aber neue Dateien schreiben möchte fängt Read_Cycle2 bei 2 an

Siehe Bild: do File : 2-p-2

Simulation von Bild 2_P_2:

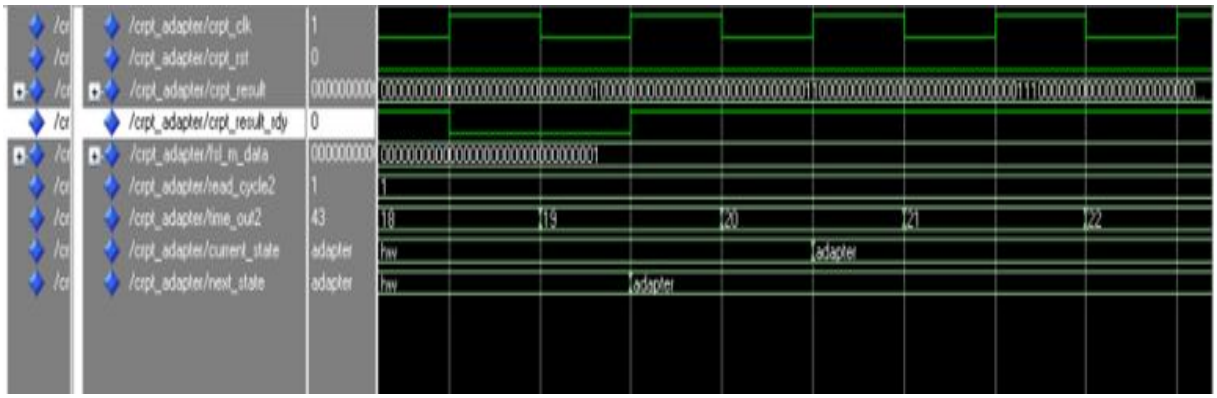


Gedachte Lösung ist: man setzt Read_Cycle2 gleich 0 im HW zustand damit im MP Übergabe zustand Read_Cycle2 mit 0 anfängt hat nicht funktioniert da L ja immer noch gleich 1 ist daher Setzt man L gleich 0 und nicht Read_Cycle2 da kommt das

Anbindung und Optimierung kryptografischer Komponente an einen Softprocessor auf einem FPGA mittels VHDL .

nächste Problem und zwar wird jetzt vom Adapter zustand nicht in den mp Übergabe zustand wechselt siehe:

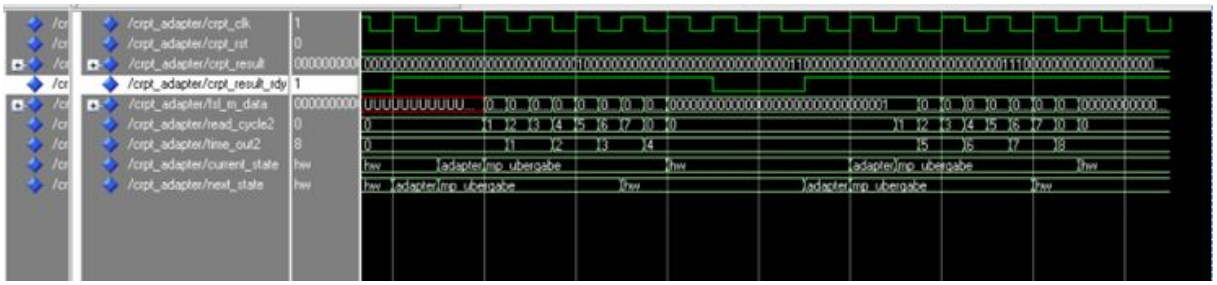
Simulation von Bild 2-p-3 :



So dieses Problem liegt daran das die Bedingung zum wechseln von Adapter zu MP Übergabe

Read_Cycle=0 ist also muss man auch Read_Cycle=0 setzen im HW zustand und siehe da es funktioniert :

Simulation von Bild: 2-p-4



jetzt funktioniert das Programm und man braucht kein Time_out2 aber man lässt es drin falls sich noch was ergibt .

Anbindung und Optimierung kryptografische Komponente an einen Softprocessor auf einem FPGA mittels VHDL .