Versuch Diktiergerät

Teilkomponente SRAM Speicher Controller

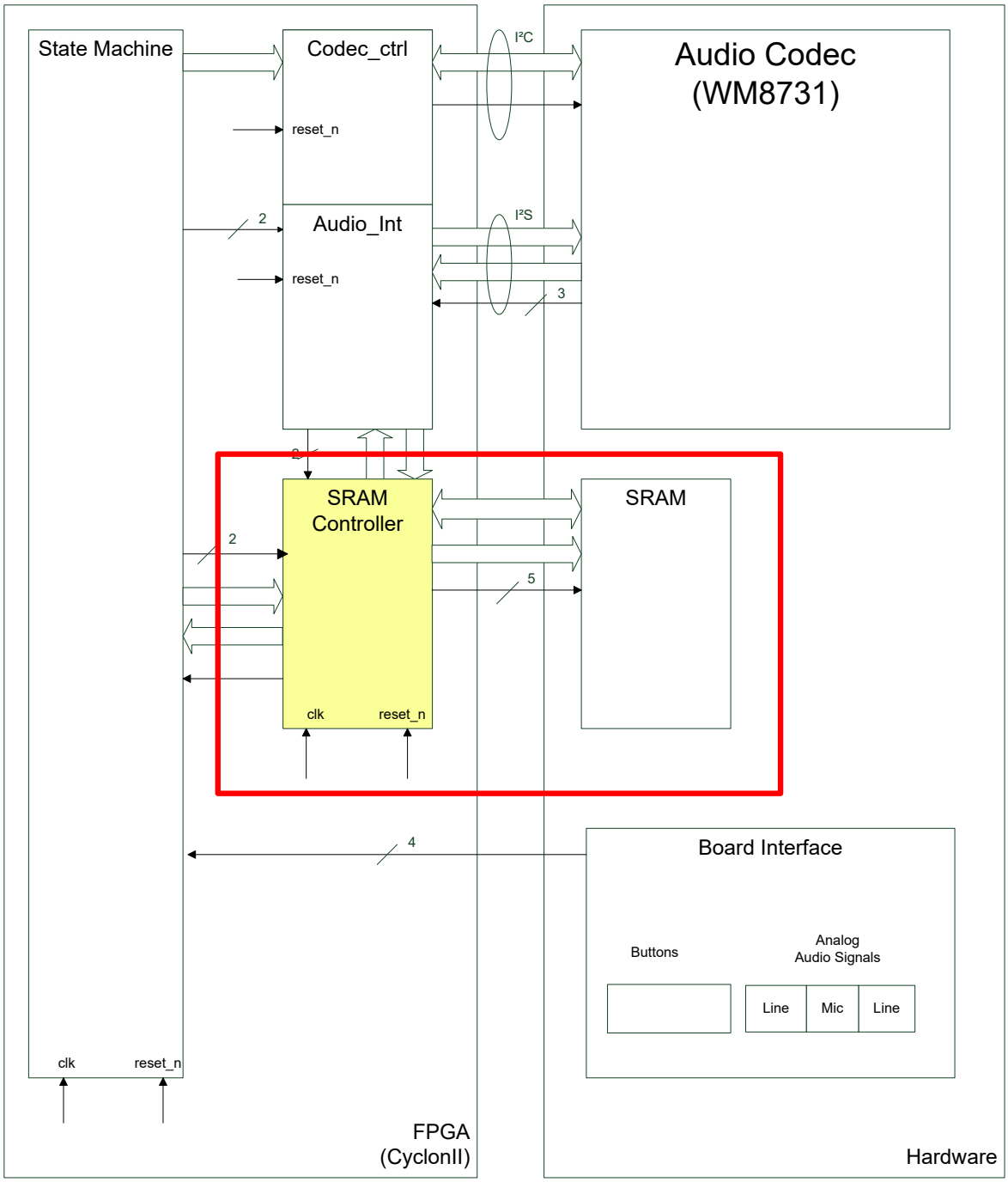


Abbildung 1: Blockschaltbild des Gesamtsystems Diktiergerät

Bei der in diesem Praktikum gestellten Aufgabe, handelt es sich um die Entwickelung eines Diktiergerätes. Es soll mit Hilfe der Hardware Beschreibungssprache VHDL eine Anbindung und Steuerung des Audio Codec WM8731 realisiert werden. Für die Aufnahme und spätere Wiedergabe des Audio Streams soll eine Speicherung der Audiodaten im SRAM Speicher implementiert werden. Der Audio Codec besitzt zwei Schnittstellen:

- eine Kontrollschnittstelle, über die man mit dem I²C Bus den Audio Codec konfigurieren kann und
- eine Datenschnittstelle die über das I²S Protokoll Audiodaten bereitstellt oder empfängt.

Das Gesamtsystem gliedert sich, wie in Abbildung 1 dargestellt, in folgende Teilkomponenten:

- Automat (FSM - Finite State Machine) zur Kontrolle der folgenden Teilkomponenten:
- Teilkomponente i2c: Audio Codec Control Interface (I²C)
- Teilkomponente i2s: Audio Codec Data Interface (I²S)
- Teilkomponente srctr: SRAM Controller

Im Rahmen des Praktikums wird die Teilkomponente „SRAM Controller“ implementiert.

Inhaltsverzeichnis

1. Einleitung und Überblick

Im Rahmen des Praktikums soll eine VHDL Schaltung entworfen werfen, die die Speicherung und den Lesevorgang der Audio Codec Daten im SRAM Chip ermöglicht.

Diese Teilkomponente (der Speicher Controller) ist in dem roten Kasten der Abbildung 1 enthalten. Der Speicher Controller selbst ist in mehrere Komponenten gegliedert. Die Entwicklung der Komponenten erstreckt sich über mehrere Praktikumstermine.

Um die Zusammenarbeit aller Teilkomponenten zu garantieren, ist es nötig die in der Schnittstellenbeschreibung vorgegebenen Schnittstellen genauestens einzuhalten. Da die zu speichernden Daten über eine 24bit breite Datenleitung übergeben werden, der Speicher jedoch „Byte adressiert“ (8bit) betrieben werden soll, wird es nötig sein diese 24bit Daten zwischen zu speichern und in mehreren Schreibzyklen in dem Speicher abzulegen.

Dasselbe Problem stellt sich umgekehrt beim Herauslesen von den Daten. Hierbei müssen mehrere Lesezyklen zu einem 24bit breiten Block zusammengefasst und übergeben werden sobald die Gegenkomponente bereit ist.

Folgendes sollen Sie mit dieser Aufgabe erlernen:

- VHDL-Kenntnisse:
  - Beschreibung von sequentiellen und kombinatorischen Prozesse,
  - Verwendung von arithmetischen Operatoren,
  - Erstellung einer Finite-State-Machine,
  - Verwendung von „Tri-State“ Signalen,
  - Anbindung des externen SRAM Chips.

- Simulation von VHDL-Beschreibungen:
  - Funktionale Simulation
  - Erstellung einer eigenen Testbench

1.1 Versuchsvorbereitung

Voraussetzung für diesen Versuch sind folgende Kenntnisse, die ggf. in den angegebenen

Skripten nachgeschlagen werden sollen:

- Die in den vorhergehenden Versuchen erlernten VHDL Fähigkeiten,
- Funktionalität und Beschreibung einer Finite-State-Machine in VHDL,
- das Verhalten und die Nutzung von Tri-State Signalen,
- Funktionsweise des anzusprechenden Speicher Chips, nachzuschlagen im Datenblatt des ISSI_61WV25616BLL Static RAM Chips (zu finden unter www.technik-emden.de/~rabe).

Im Versuch erhalten Sie eine fertige Testbench für das Teildesign des SRAM-Controlers und für den SRAM-Datenpfad. Die Architectures und Entities, sowie die Testbench für den SRAM-Adresspfad, werden dann im Praktikum erstellt.

1.2 Informationen

SRAM

Bei dem zu benutzenden Speicher handelt es sich um einen 512kb großen Wort-adressierten SRAM (1 Wort = 16 Bit). Dieser Speicher kann auch byteweise geschrieben und gelesen werden. Der Speicher besitzt unter anderen zwei Kontrolleingänge mit denen man das “upper“ und das “lower“ Byte der Wortadresse auswählen kann. Sind beide Kontrolleingänge gesetzt wird das ganze Wort geschrieben bzw. gelesen. Da es sich bei den Audiodaten um 24bit Daten handelt und die Zugriffsgeschwindigkeit keine Rolle spielt, wird der Speicherzugriff in drei 8bit Schreib- bzw. Lesezyklen aufgeteilt.

Schnittstelle des SRAM Chips

Die Schnittstelle vom Speicher sieht folgendermaßen aus:

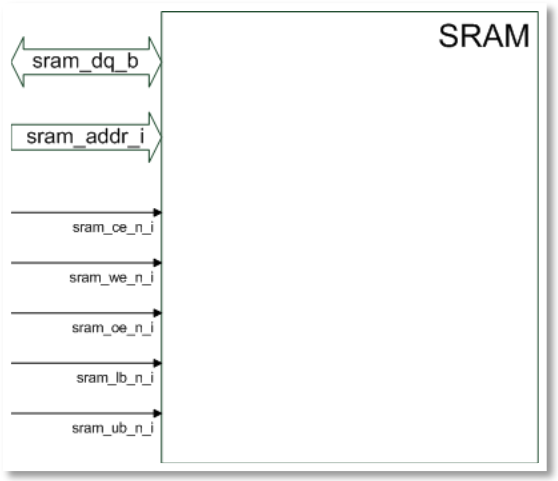


Abbildung 2: Schnittstellenbeschreibung des SRAM Speicherchips

Name und Datentyp	Name im Datasheet	Beschreibung
sram_ce_n_i std_ulogic	**CE**	Bei dem CE Eingang handelt es sich um einen „Chip Enable“ Eingang des SRAM Chips. Dieses Signal kann durchgehen Null und somit angeschaltet sein. Aus verschiedenen Gründen, wie z.B. um Energie zu sparen, macht es Sinn den Chip nur anzuschalten wenn gelesen oder geschrieben wird. (low aktiv)
sram_we_n_i std_ulogic	**WE**	Der „Write Enable“ Eingang dient zum Signalisieren, dass Daten auf den SRAM-Chip geschrieben werden sollen. (low aktiv)

Name und Datentyp	Name im Datasheet	Beschreibung
sram_oe_n_i std_ulogic	**OE**	Über den „Output Enable“ Eingang wird für den SRAM-Chip signalisiert, dass Daten gelesen werden sollen. Hiermit werden SRAM-intern die Ausgangstreiber auf den sram_dq_b-Bus gesteuert (low aktiv)
sram_ub_n_i std_ulogic	**UB**	Das UB Signal gibt an, dass das „Upper Byte“ an der vorliegenden Adresse beschrieben bzw. gelesen werden soll. (low aktiv)
sram_lb_n_i std_ulogic	**LB**	Mit dem LB Signal wird der Zugriff auf das „Lower Byte“ der vorliegenden Adresse gesetzt. (low aktiv)
sram_addr_i std_ulogic_vector (17 downto 0)	**A**	Über die Adresseingänge wird die Wortadresse, die beschrieben oder gelesen werden soll, an den Speicherchip angelegt.
sram_dp_b std_ulogic_vector (15 downto 0)	**I/O**	Die bidirektionalen In/Out Signale dienen als Daten Ein- und Ausgang. Hier liegen sowohl die Daten die in den Speicher geschrieben werden sollen, als auch die Daten die gelesen werden sollen an. Byteweisen Zugriff wird entweder erreicht durch setzten des “Upper“ Bytes oder “Lower“ Bytes. Bei gesetztem “Upper“ Byte werden nur die oberen 8bits benutzt und bei gesetztem “Lower“ Byte nur die unteren 8bits. Werden beide gesetzt werden die kompletten 16 Bit benutzt.

Um fehlerhafte SRAM-Zugriffe durch Laufzeitunterschiede an Adress- und Kontroll-Signalen zu vermeiden, werden für jeden Schreib-/Lesezugriff drei Takte genutzt:

- 1. Takt: Einstellen der Adresse, sram_ub_n_i/sram_lb_n_i-Signale und beim Schreiben der zu schreibenden Daten
- 2. Takt: Aktivierung des Lese- bzw. Schreibvorgangs durch sram_oe_n_i/ sram_we_n_i
- 3. Takt: Halten der Adresse, sram_ub_n_i/sram_lb_n_i-Signale und beim Schreiben der zu schreibenden Daten – die Signale sram_oe_n_i/ sram_we_n_i werden wieder deaktiviert.

Bei allen Signalen ist darauf zu achten, dass diese Signale nicht durch Laufzeitunterschiede spiken:

- Adresse: unabhängig vom Zustand immer die aktuell gespeicherte SRAM-Adresse verwenden,
- sram_ub_n_i/sram_lb_n_i: unabhängig vom Zustand dieses Signal aus dem untersten Adress-Bit generieren,
- zu treibende Daten: direkt ohne Multiplexer immer die unteren 8 Bits des 24-Bit-Datenregisters verwenden,
- sram_oe_n_i/ sram_we_n_i werden registered, um Spikes zu vermeiden (Abhängigkeit vom Zustand - sie erhalten deshalb den Namenszusatz _reg).

SRAM Kontrollsignalbelegung für den Zugriff

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	$\overline{LB}$	$\overline{UB}$	I/O PIN I/O0-I/O7	I/O PIN I/O8-I/O15	Vcc Current
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	Icc
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	D_{OUT}	High-Z	Icc
	H	L	L	H	L	High-Z	D_{OUT}	
	H	L	L	L	L	D_{OUT}	D_{OUT}	
Write	L	L	X	L	H	D_{IN}	High-Z	Icc
	L	L	X	H	L	High-Z	D_{IN}	
	L	L	X	L	L	D_{IN}	D_{IN}	

Abbildung 3: Zustandstabelle für den Zugriff auf den SRAM Speicherchip

Anmerkung zu Abb. 3: Diese Abbildung ist direkt aus der SRAM-Spezifikation kopiert. Die Signalpegel L und H sind in VHDL als 0 bzw. 1 zu beschreiben. „X“ steht für „DON’T CARE“ – d.h., Sie können sich hier für 0 oder 1 entscheiden.

Die Kontrollsignalbelegung muss für einen Zugriff auf den Speicher genau eingehalten werden, da der Speicherchip sonst falsch oder gar nicht reagiert. In Abbildung 3 sind die Kontrollsignalbelegungen für alle Speicherzugriffe dargestellt.

SRAM Timings

Da es sich bei diesen RAM Speichern um taktunabhängige Baugruppen handelt, spielt das Timing eine sehr wichtige Rolle. Wenn man zum Beispiel bei einem Lesevorgang eine Adresse an den Adresseingang (**A**) anlegt, steht kurze Zeit später (unabhängig vom Takt) das Datum schon bereit. Der Speicherchip benötigt eine bestimmte Zeit, um das angeforderte Datum an der Datenschnittstelle (**I/O**) bereitzustellen, wie in Abbildung 4 dargestellt ist.

In den Spezifikationen zu dem eingesetzten IS61WV25616BLL – 10 TL SRAM Speicherchip lassen sich solche und andere Werte über das Verhalten herausfinden.

Lesezugriff

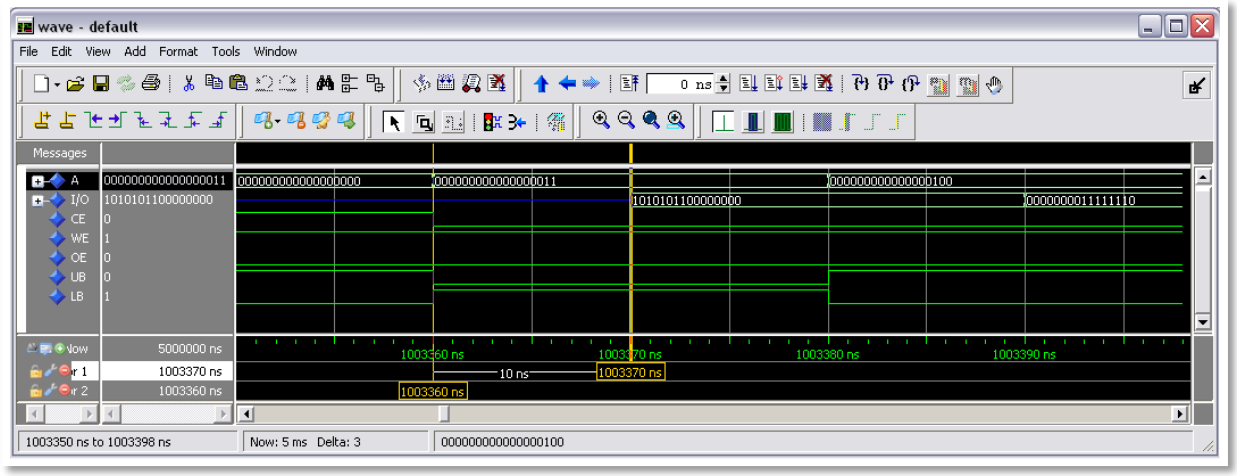


Abbildung 4 – Lesen vom SRAM: Die Signale CE, WE, OE, UB und LB sind low-active, was bedeutet das sie 0 sind wenn sie aktiviert sind. Um ein Datum aus dem Speicher zu lesen, müssen die Signale CE und OE mindestens aktiviert werden. Weiterhin muss mindestens eins der beiden Signale UB oder LB aktiviert werden, um zu bestimmen, welches Byte gelesen werden soll, sind beide aktiv wird ein ganzes Wort aus dem Speicher gelesen. Nach dem all diese Signale gesetzt sind und die Adresse an den Adresseingang(A) angelegt wurde, wird das Datum mit einer Hardware bedingten Verzögerung von max. 10 ns am Daten-Ein-/Ausgang(I/O) bereitgestellt. Dabei ist zu beachten, dass der Daten-Ein-/Ausgang ihres Controllers zu diesem Zeitpunkt „hochohmig" ist, um Daten empfangen zu können und die Hardware nicht zu beschädigen. Ein sogenanntes „Tri-State" Signal muss verwendet werden.

Schreibzugriff

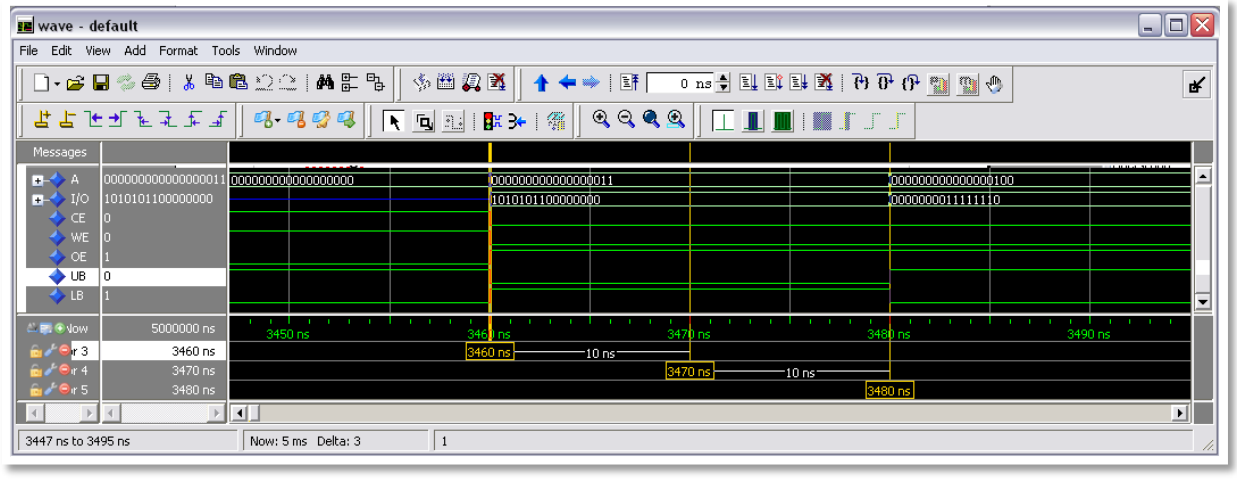


Abbildung 5 - Schreibzugriffe auf das SRAM: Es muss nur die Adresse angelegt werden und alle erforderlichen Kontroll-Signale gesetzt werden, damit das Datum an der gewünschten Stelle im SRAM gespeichert wird. Dies dauert allerdings auch seine Zeit, aber mit max. 10 ns liegt es weit unter der Periodendauer des Systemtakts und kann somit in einem Zyklus ihres Controllers abgearbeitet werden. Die Signale CE, WE und mindestens eins der Signale UB und LB müssen zusammen mit der Adresse anliegen. Mit UB und LB wird auch beim Schreibvorgang entschieden welche(s) Byte(s) geschrieben werden.

1.3 Schnittstelle der Teilkomponente SRAM-Controller

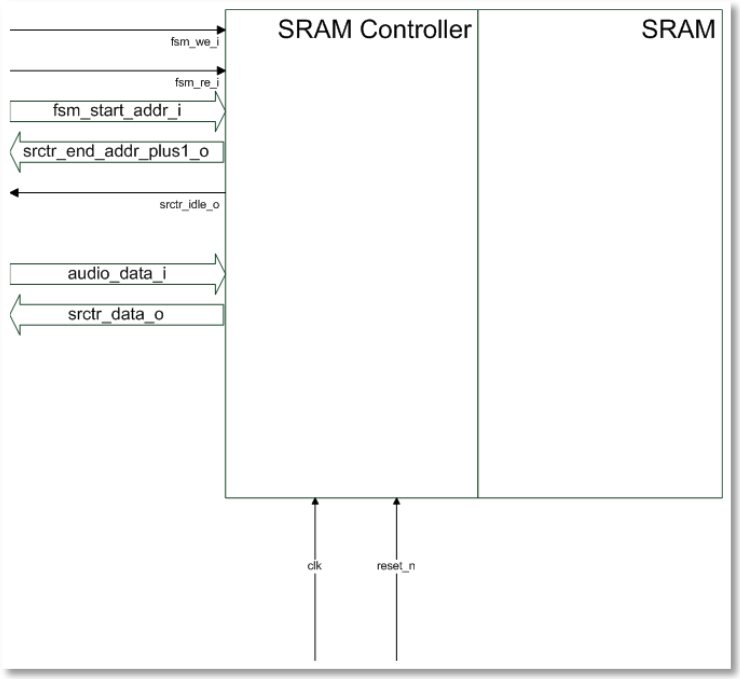


Abbildung 6: Schnittstellenbeschreibung des SRAM Speicher Controllers

NAME	Datentyp	Beschreibung
clk_i	**std_ulogic**	Systemtakt Eingang für die zu erstellende Teilkomponente.
reset_n_i	**std_ulogic**	Reset Eingang für die zu erstellende Teilkomponente. (low aktiv)
fsm_start_addr_i	**std_ulogic_vector (18 downto 0)**	Adresseingang von der (globalen) FSM, um die Startadresse für den Schreib- oder Lesevorgang des SRAM Controller festzulegen. 18 Bits stehen für die Wortadresse zur Verfügung und Bit 0 bestimmt ob das “upper“ oder das “lower“ Byte geschrieben / gelesen werden soll.
fsm_we_i, **fsm_re_i**	**std_ulogic,** **std_ulogic**	Über fsm_we_i wird signalisiert, dass Daten zur Speicherung bereit stehen. Mit fsm_re_i=‘1‘ wird ein Lesevorgang vom SRAM gestartet. Während eines aktiven Schreib- oder Lesevorgangs (dauer ja viele Takte) werden diese Signale ignoriert.

audio_data_i, **srctr_data_o**	**std_ulogic_vector (23 downto 0)** **std_ulogic_vector (23 downto 0)**	Datenleitungen für Ein- und Ausgabe des SRAM Controllers. audio_data_i ist der 24 Bit breite Dateneingang des Controllers und srctr_data_o der 24 Bit breite Ausgang.
srctr_idle_o	**std_ulogic**	Signalisiert, dass der SRAM Controller bereit ist, neue Daten zu verarbeiten. Ein neuer Zugriff kann mit der folgenden Taktflanke gestartet werden. Außerdem wird signalisiert, dass folgende Ausgangssignale gültig sind: • srctr_data_o (nach einem Lesevorgang) und • srctr_end_addr_plus1_o.
srctr_end_addr_plus1_o	**std_ulogic_vector (18 downto 0)**	Adressausgang zu der FSM, um die nächste freie Speicheradresse zu übergeben. Dies ist wichtig damit in der FSM nicht zusätzlich ein eigener Addierer implementiert werden muss, um die um 3 erhöhte Adresse für den nächsten Schreibzugriff zu berechnen.

1.4 Speicher Controller Timings

In den folgenden beiden Abbildungen wird das taktorientierte Zeitverhalten des Speichercontrolers für einen Schreib- und Lesezugriff gezeigt. Die Abbildungen enthalten die Simulationsergebnisse einer rein funktionalen Simulation – Verzögerungszeiten durch Gatterlaufzeiten sind also nicht berücksichtigt:

- Lesezugriff:
    - Vor der steigenden Taktflanke bei t=1300ns werden neue Audio-Daten angefordert. Der SRAM-Automat ist im Zustand fsm_idle (also bereit den nächsten Zugriff zu starten – dies wird durch das aktive Signal srctr_idle angezeigt) und der globale Automat setzt hierzu das Signal fsm_re auf '1'. Mit der steigenden Taktflanke wechselt der SRAM-Controller nun in den Zustand fsm_read_mem_11. Die Start-Adressen fsm_start_addr für den Lesevorgang wird vom SRAM-Controller in das Register srctr_addr_reg übernommen. Da das unterste Adress-Bit '0' ist, wird für den Lesevorgang das untere Byte selektiert (srctr_lb_reg_n='0').
    - Mit der nächsten steigenden Flanke (t=1320ns) wechselt der Automat des SRAM-Controllers in den Zustand fsm_read_mem_12. Mit diesem Wechsel wird das Signal srctr_oe_reg_n auf '0' gesetzt. Dadurch wird aus dem externen Speicher das Datum gelesen und auf den unteren 8 Bit von mem_data getrieben (Datum='FB').
    - Mit der folgenden steigenden Flanke (t=1340ns) wechselt der Automat des SRAM-Controllers in den Zustand fsm_read_mem_13. Gleichzeitig werden die Daten in die obersten 8 Bit des Datenregisters übernommen. Das Signal srctr_oe_reg_n kann hierfür wieder deaktiviert werden (→'1').
    - Mit der folgenden steigenden Flanke (t=1360ns) wechselt der Automat des SRAM-Controllers in den Zustand fsm_read_mem_21. Nun wird die Adresse um 1 erhöht (srctr_addr_reg='0001'). Entsprechend wird nun das Upper-Byte selektiert (srctr_ub_reg_n='0' und srctr_lb_reg_n='1').
    - Analog zum Lesevorgang für das erste Byte folgt nun das Lesen des 2. Bytes (es wird 00 gelesen). Beim Übergang vom Zustand fsm_read_mem_22 in den Zustand fsm_read_mem_23 (t=1400ns) wird das neu gelesene Byte wieder in die oberen 8 Bits des Daten-Registers (data_reg) übernommen. Damit die Daten nicht überschrieben werden, werden die vorher gelesenen Daten um 1 Byte nach rechts geschoben.
    - Mit der steigenden Taktflanke zum Zeitpunkt t=1420ns wird nun wieder die Adresse um 1 erhöht und der Lesevorgang des 3. Bytes gestartet.
    - Die kompletten 3 Bytes befinden sich im letzten Takt (Zustand fsm_read_mem_33) bereits im Daten-Register (data_reg='FD00FB'). Mit dem Signal srctr_idle wird signalisiert, dass mit der nächsten steigenden Flanke die Daten am Ausgang srctr_data zur Verfügung stehen. Außerdem kann dann bereits der nächste SRAM-Zugriff gestartet werden.
      Das Signal srctr_end_addr_plus1 gibt die Folgeadresse an, die für den nächsten Lesevorgang als Start-Adresse verwendet werden kann.

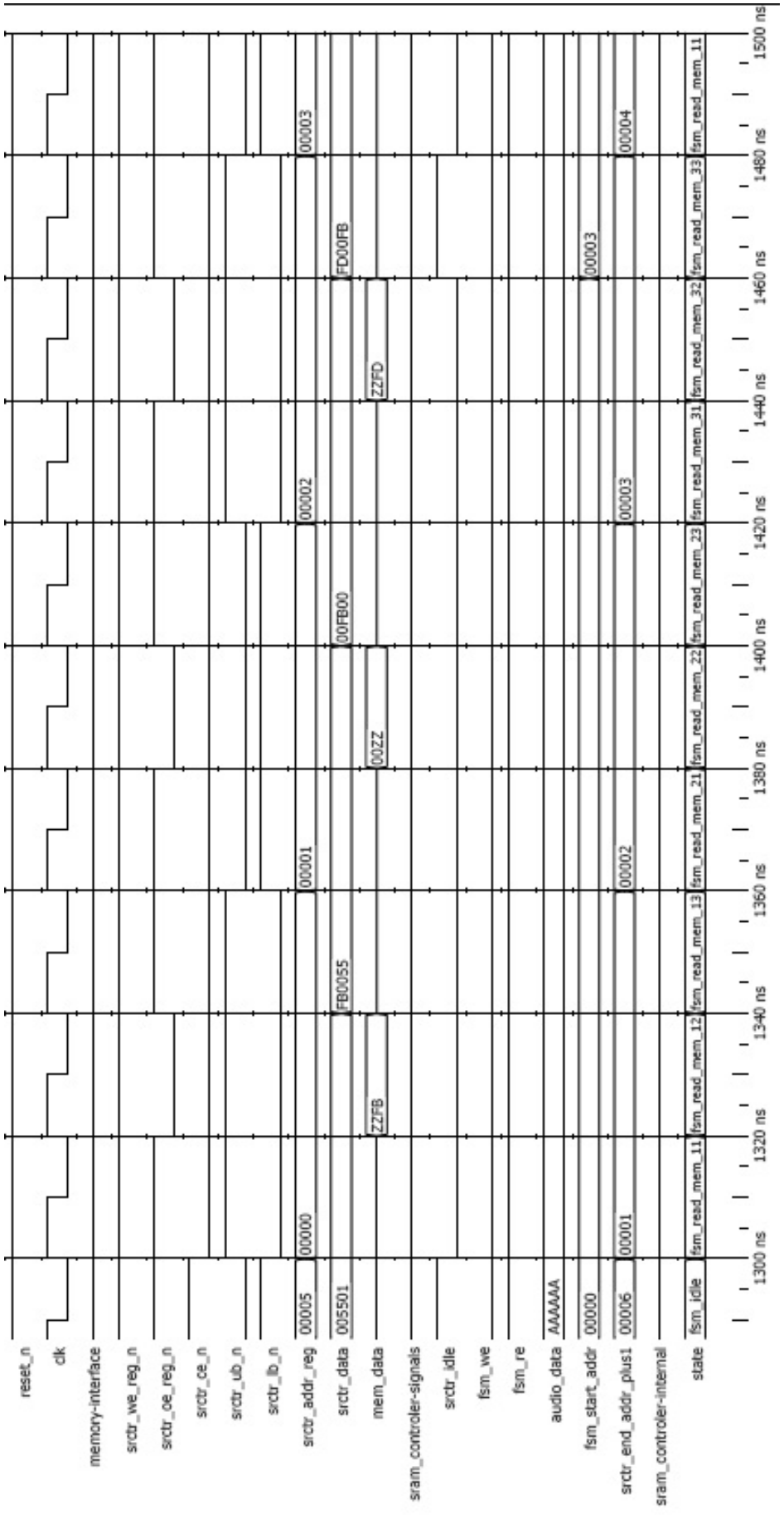


Abbildung 7: Nach dem Anlegen einer Adresse bei einem Lesezugriff kann das Datum nach 9 Taktzyklen abgeholt werden. srctr_idle_o signalisiert, dass das Datum mit der folgenden Taktflanke stabil an srctr_data_o anliegt und dass der Speicher Controller für den nächsten Zugriff bereit ist. Dieser kann mit der folgenden Taktflanke gestartet werden. Das Datum bleibt bis dahin gültig.

- Schreibzugriff:
  - Vor der steigenden Taktflanke bei t=100ns werden neue Audio-Daten zum schreiben in das SRAM bereit gestellt (audio_data='FD00FB'). Der SRAM-Automat ist im Zustand fsm_idle (also bereit den nächsten Zugriff zu starten – dies wird durch das aktive Signal srctr_idle angezeigt) und der globale Automat setzt hierzu das Signal fsm_we auf '1'. Mit der steigenden Taktflanke

wechselt der SRAM-Controller nun in den Zustand fsm_write_mem_11. Die Start-Adresse fsm_start_addr für den Schreibvorgang wird vom SRAM-Controller in das Register srctr_addr_reg übernommen. Da das unterste Adress-Bit '0' ist, wird für den Schreibvorgang das untere Byte selektiert (srctr_lb_reg_n='0'). Der Inhalt der unteren 8 Bits im Daten-Register (data_reg(7 downto 0)= 'FB') werden auf die unteren und oberen 8 Bit des bidirektionalen mem_data-Busses getrieben ('FBFB').

- Mit der nächsten steigenden Flanke (t=120ns) wechselt der Automat des SRAM-Controllers in den Zustand fsm_write_mem_12. Mit diesem Wechsel wird das Signal srctr_we_reg_n auf '0' gesetzt und damit der eigentliche Schreibvorgang gestartet.
- Mit der folgenden steigenden Flanke (t=140ns) wechselt der Automat des SRAM-Controllers in den Zustand fsm_write_mem_13. Das Signal srctr_we_reg_n wird hier wieder deaktiviert (→'1').
- Mit der folgenden steigenden Flanke (t=160ns) wechselt der Automat des SRAM-Controllers in den Zustand fsm_write_mem_21. Nun wird die Adresse um 1 erhöht (srctr_addr_reg=0001). Entsprechend wird nun das Upper-Byte selektiert (srctr_ub_reg_n='0' und srctr_lb_reg_n='1'). Um weiter die unteren 8 Bits des Daten-Registers (data_reg) als Quelle für die auf mem_data zu treibenden Daten verwenden zu können, wird das Datenregister um 1 Byte nach rechts geschoben. Das Rotieren der untersten 8 Bits auf die obersten 8 Bits von data_reg ist optional. Durch das Rotieren, wird nun das 2. Byte ('00') auf das Daten-Register getrieben.
- Analog zum Schreibvorgang für das erste Byte folgt nun das Schreiben des 2. Bytes (es wird '00' geschrieben). Beim Übergang vom Zustand fsm_read_mem_23 in den Zustand fsm_read_mem_31 (t=220ns) wird der Inhalt des Daten-Registers (data_reg) wieder um ein Byte nach rechts geschoben und die Adresse um 1 inkrementiert.
- Die kompletten 3 Bytes sind im letzten Takt (Zustand fsm_write_mem_33) im SRAM-Speicher abgelegt. Mit dem Signal srctr_idle wird signalisiert, dass mit der nächsten steigenden Flanke der nächste SRAM-Zugriff gestartet werden kann. Das Signal srctr_end_addr_plus1 gibt die Folgeadresse an, die für den nächsten Schreibvorgang als Start-Adresse verwendet werden kann.

Für die Ablaufsteuerung wird eine relativ einfache Zustandsfolge verwendet, um jeweils zum richtigen Zeitpunkt die richtigen Ausgaben generieren zu können. Das Zustandsfolgediagramm ist in der Abbildung 9 dargestellt.

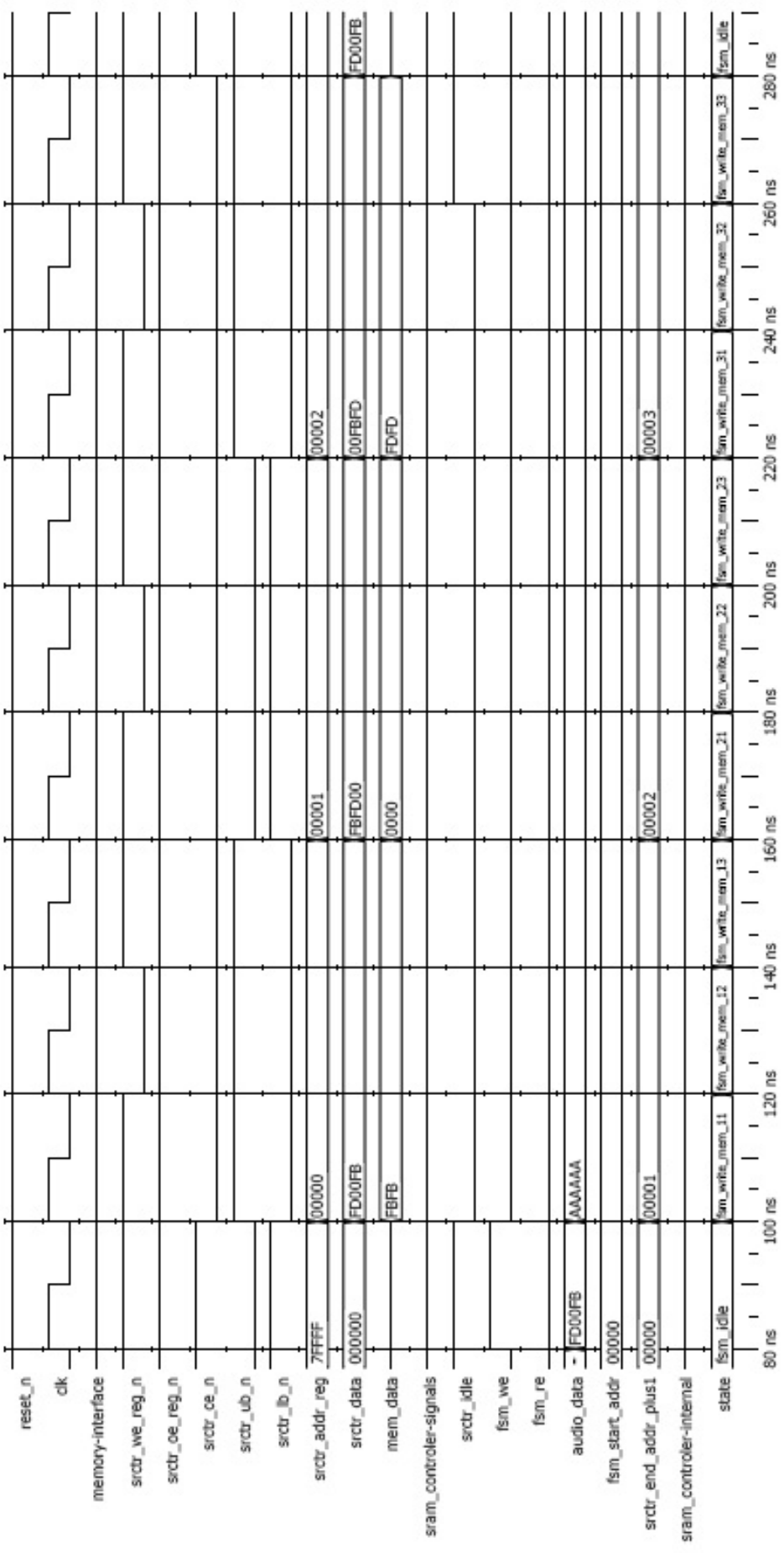
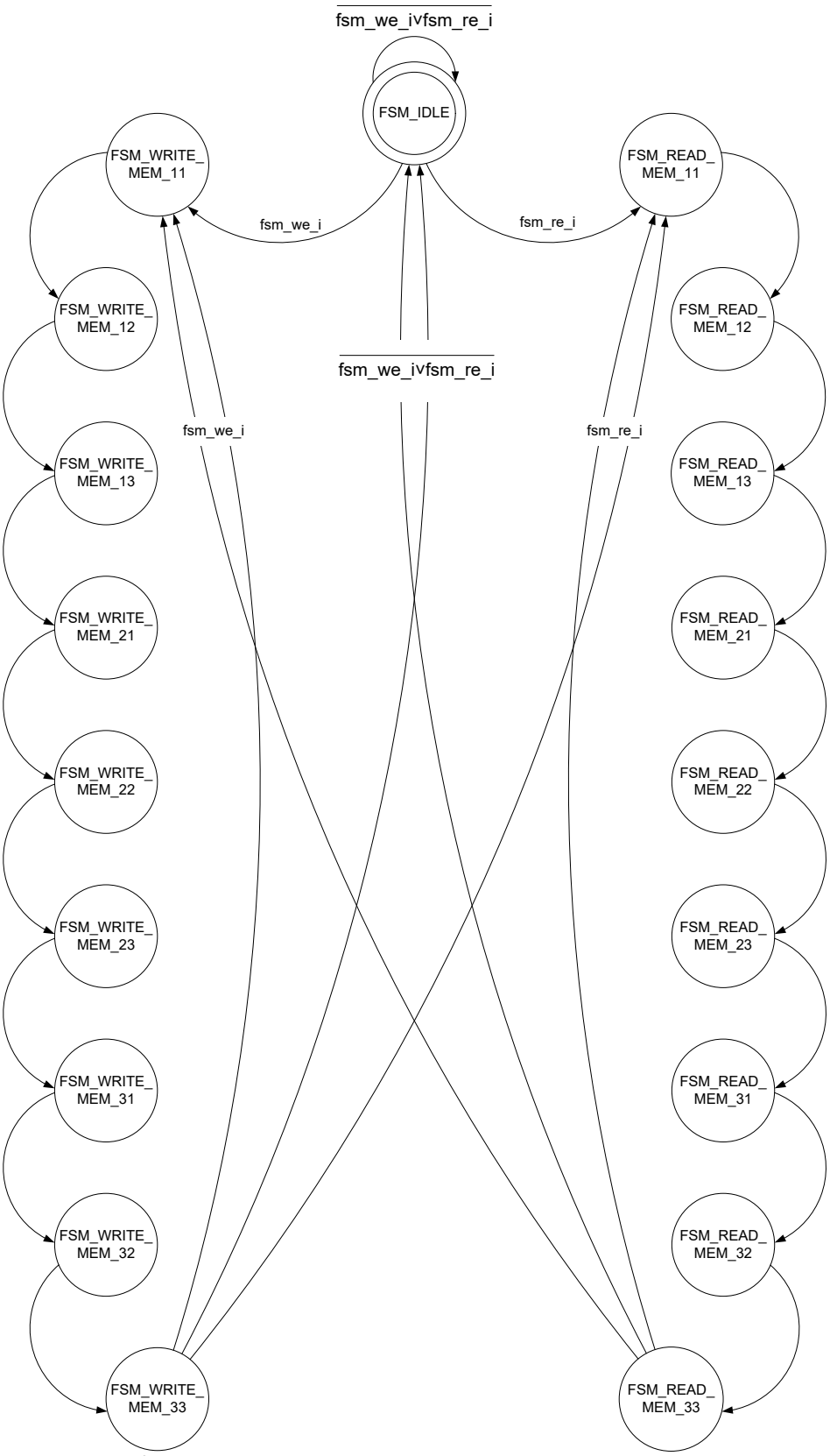


Abbildung 8: Bei Schreibzugriffen ist darauf zu achten das die Daten zusammen mit der Adresse und den Kontrollsignalen vor der Taktflanke stabil anliegen. Der Schreibzugriff benötigt 9 Taktzyklen. srctr_idle_o signalisiert, dass der Controller wieder bereit ist und mit der folgenden Taktflanke ein neuer Zugriff gestartet werden kann.



Anmerkung 1: alle Zustandübergangspfeile ohne Angabe einer Bedingung sind immer wahr
Anmerkung 2: es handelt sich hier um einen gemischten Moore/Mealy-Automaten. Die meisten Ausgaben hängen nur vom Zustand ab (also Moore-Automat). Die Bedingung für die Datenübernahme von audio_data_i greift jedoch zusätzlich auf fsm_we_i zurück. Die Ausgaben sind aus Gründen der Übersichtlichkeit in einer separaten Tabelle angegeben.

Abbildung 9 Finite State Maschine des SRAM-Controllers

1.3 Realisierung des SRAM-Controlers

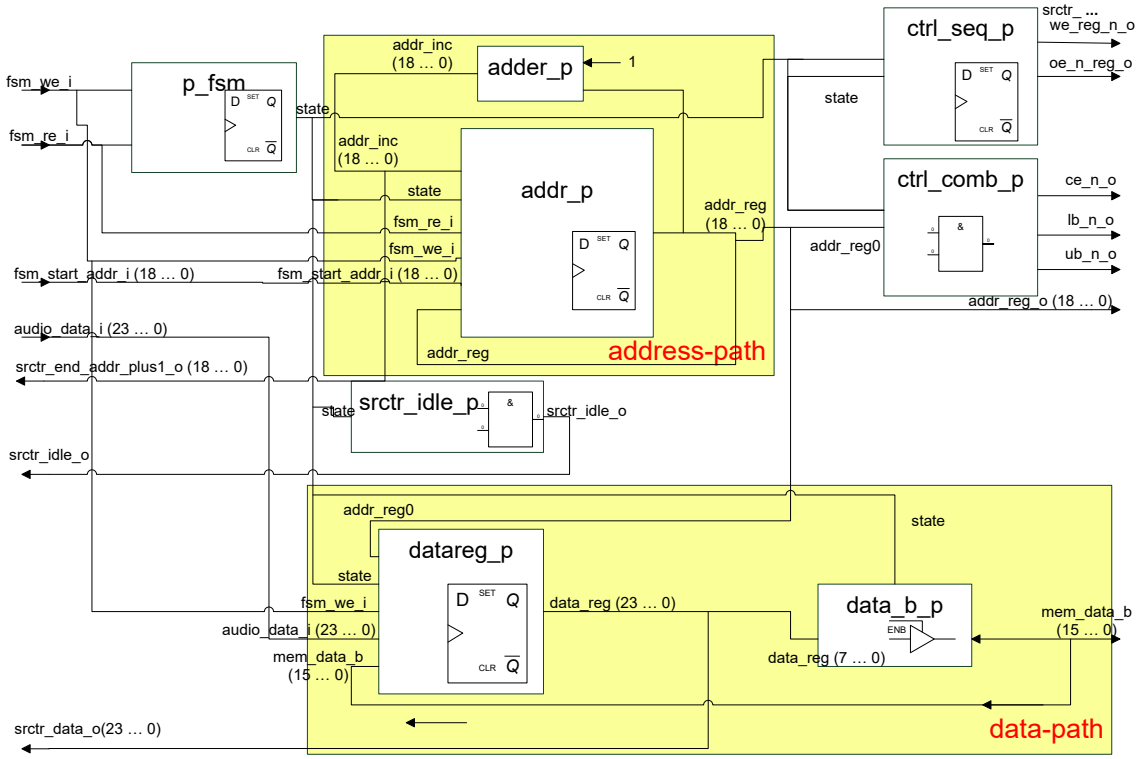
Bei komplexeren Aufgaben ist es meistens sinnvoll das Gesamtproblem in mehrere kleine Probleme aufzuteilen. Die einzelnen Teilaufgaben sind einfacher zu lösen und können später wieder zu einem Gesamtsystem zusammengesetzt werden. Ein guter Ansatz ist es mit den Ein- und Ausgabesignalen anzufangen. Diese kann man so gruppieren, dass alle Ausgänge die zur Kontrolle desselben externen Bauteils dienen, in einem Prozess beschrieben werden. Besondere Signale wie Tri-State-Signale sollten jedoch in einem gesonderten Prozess behandelt werden.

Die Steuerung der Komponente wird in der Regel immer durch eine Finite State Maschine (Automat) realisiert. In dieser FSM wird in Abhängigkeit vom aktuellen Zustand und der Kombination der Kontrolleingänge der nächste Zustand generiert.

Die Ausgänge der FSM werden aus dem aktuellen Zustand und eventuell den Eingangssignalen (Mealy/Moore-Automat) in der Ausgabefunktion berechnet.

Eine weitere, sinnvolle, Unterteilung ist es spezifische Operationen jeweils in separate Prozesse auszugliedern (z.B. Schieberegister, Adressinkrementierer, Counter und herunter geteilte Takte).

Die Aufteilung des SRAM-Controllers ist in der Abbildung 10 dargestellt.



Legende:

- weiße Blöcke mit Flipflop-Symbol: in VHDL als sequentieller Prozess zu realisieren
- weiße Blöcke mit UND-Gatter-Symbol: in VHDL als nebenläufige Signalzuweisung oder kombinatorischer Prozess zu realisieren
- gelb hinterlegte Bereiche: separate VHDL-Einheiten (entity+architecture)

Abbildung 10: Blockschaltbild des SRAM Speicher Controllers

Die Ein- und Ausgänge des SRAM-Controllers sind:

Ausgänge zum Speicherchip

srctr_we_reg_n_o, **srctr_ce_n_o**, **srctr_oe_reg_n_o**, **srctr_lb_n_o** und **srctr_ub_n_o** sind Kontrollausgänge die für die Steuerung des SRAM Chips Verantwortlich sind.

mem_data_b ist ein 16-Bit breiter bidirektionaler Ausgang zum Speicher um die Daten zu übergeben und zu Empfangen. Bidirektionale Ausgänge müssen mit einem Tri-State betrieben werden, da sie beim Empfang von Daten hochohmig sein müssen.

addr_reg_o ist ein 19-Bit breiter Ausgang zum Speicher, der die Adresse angibt. Das niederwertigste Bit wird hierbei nicht verwendet, weil die Information in den Signalen **srctr_lb_n_o** und **srctr_ub_n_o** enthalten ist.

Ausgänge der Komponente

srctr_idle_o – siehe obige Tabelle.

srctr_data_o ist ein 24 Bit breiter Ausgang – siehe obige Tabelle

srctr_end_addr_plus1_o ist ein 19 Bit breiter Ausgang – siehe obige Tabelle

Eingänge von der FSM

fsm_we_i und **fsm_re_i** sind die zwei Kontrolleingänge für die Komponente.

Eingänge vom Audio-Code-Modul

audio_data_i ist der 24 Bit breite Eingangs-Bus vom Audio-Codec-Modul

Der SRAM-Controler enthält 2 Sub-Komponenten:

- Daten-Pfad: enthält ein 24-Bit Datenregister und einen Prozess, um das TRI-State-Signal mem_data_b im Schreibfall zu treiben.
- Adress-Pfad: hier wird die aktuelle Adresse automatisch zum richtigen Zeitpunkt um 1 inkrementiert; zusätzlich wird die aktuelle Adresse+1 an den Ausgang srctr_end_addr_plus1_o gelegt.

Die dargestellten Prozesse bzw. nebenläufigen Signalzuweisungen sind in der folgenden Tabelle zusammen gefasst:

Prozessname	Typ	Beschreibung
datareg_p	Sequentiell	Shift Register (Byte-weise; abhängig vom State werden Daten übernommen oder geschoben (Details siehe unten))
data_b_p	Kombinatorisch	Schaltet den biderektionalen Datenbus mem_data_b auf hochohmig wenn Daten vom Speicher gelesen werden und treibt zu speichernde auf den Bus.
addr_p	Sequentiell	Erzeugt die nächste Speicheradresse
p_fsm	Sequentiell	Automatenrealisierung
srctr_idle_p	Kombinatorisch	Erzeugt aus dem aktuellen Zustand das srctr_idle_o Signal

Prozessname	Typ	Beschreibung
control_comb_p	Kombinatorisch	Erzeugt kombinatorisch aus addr_reg0 und dem aktuellen Zustand die restlichen Ausgabesignale, die für die Ansteuerung des SRAM-Speichers erforderlich sind.
control_seq_p	Sequentiell	Erzeugt aus dem aktuellen Zustand die Ausgabesignale oe_reg und we_reg, die für die Ansteuerung des SRAM-Speichers erforderlich sind.

Im Rahmen des Praktikums sollen nun zunächst die beiden Unterkomponenten data_path und address_path realisiert werden. Anschließend werden diese Einheiten in der srctr-Komponente selbst instanziiert und die fehlenden Prozesse ergänzt. Diese 3 Schritte werden in den nächsten 3 Abschnitten beschrieben.

2 Daten-Pfad

Diese VHDL-Komponente soll als erstes entworfen werden.

Im Daten-Pfad sind die Prozesse datareg_p und data_b_p dafür verantwortlich, dass in Abhängigkeit vom aktuellen Zustand (state_i) data_reg(23 downto 0) und mem_data_b(15 downto 0) die richtigen Werte speichern bzw. treiben. Die genaue Funktion der Prozesse ist im Folgenden erklärt.

datareg_p

In diesem sequentiellen Prozess werden 24 Daten-Bits gespeichert. Es werden dieselben Register für den Lese- und Schreibzugriff verwendet. Die Zugriffsweise unterscheidet sich jedoch:

- Beschreiben des Speichers mit Audio-Daten (24-Bit vom Audio-Codec-Modul):
  Die 24 Bit Daten werden in einem Schritt vom Audio-Codec-Modul übernommen (wenn fsm_we_i=‘1‘ und der Automat in einer der folgenden 3 Zustände ist: FSM_IDLE, FSM_WRITE_MEM_33, FSM_READ_MEM_33). Die Bits 0-7 werden jeweils auf die unteren und oberen Bits des mem_data_b-Signals getrieben. Nachdem das 1. Byte geschrieben wurde (also nach 3 Takten) wird das gesamte 24bit Register um 1 Byte nach rechts geschoben, damit die nächsten 8 Bits richtig positioniert sind, um von data_reg(7 downto 0) auf mem_data_b getrieben zu werden. Dieser Schiebe-Vorgang wird nach weiteren 3 Takten wiederholt, um die letzten 8 Bits zu schreiben.

  Schiebestruktur (Schreiben):

CLK Cycle	24 Bit (3 Byte) Schieberegister		
1-3	Byte2	Byte1	Byte0 → Speicher

CLK Cycle	24 Bit (3 Byte) Schieberegister		
4-6	Byte0[1]	Byte2	Byte1 → Speicher
7-9	Byte11	Byte01	Byte2 → Speicher

- Lesen von Audio-Daten (24-Bit vom SRAM für das Audio-Codec-Modul):
  Während des Lesezugriffs werden die aus dem Speicher gelesenen Bytes im obersten Byte des 24bit Registers gespeichert. Das 1. Byte wird mit der zweiten positiven Taktflanke übernommen und die übrigen Bytes optional um 1 Byte nach rechts geschoben. Das 2. Byte wird genau 3 Takte später übernommen und die übrigen Bytes zeitgleich um 1 Byte nach rechts geschoben. Entsprechend erfolgt auch das Lesen des 3 Bytes – wiederum 3 Takte später. Nach dem Auslesen der 3 Bytes liegen die Daten im Daten-Register (data_reg), das direkt mit dem Ausgang srctr_data_o verbunden ist und bleiben dort bis zum nächsten Zugriff liegen. Die Schieberichtung ist identisch mit der beim Speichern.
  Das unterste Adressbit gibt die Byte-Position innerhalb der 16 Datenbits an. Die Byte-Reihenfolge ist als Little-Endian definiert (also addr_reg0 -> lower byte -> Daten auf unteren 8 Bits des 16-Bit Adressbusses).

  Schiebestruktur (Lesen):

CLK Cycle	24 Bit (3 Byte) Schieberegister		
1-3	Speicher → Byte0	————	————
4-6	Speicher → Byte1	Byte0	————
7-9	Speicher → Byte2	Byte1	Byte0

data_b_p

Erzeugt ein Tristate Signal, das je nach Zustand den bidirektionalen Bus mem_data_b hochohmig schaltet:

State	mem_data_b=
FSM_WRITE_MEM_11/12/13, FSM_WRITE_MEM_21/22/23 oder FSM_WRITE_MEM_31/32/33	data_reg(7 downto 0) &[2] data_reg(7 downto 0)
Alle anderen States	'Z' - hochohmig

[1] Das Schieben dieses Bytes an diese Position ist optional.

[2] & → Verkettungssymbol in VHDL (hier data_reg wird auf die unteren und oberen 8 Bits des memdata_b-Busses getrieben.

2.1 Aufgabenstellung

1. Erstellen Sie die VHDL-Beschreibung für den Datenpfad (sram_controler_data): Entity+Architecture. Hierfür stehen Ihnen folgende VHDL-Beschreibungen zur Verfügung:
    - VHDL-Package sram_controler_pack: Definition des Datentyps fsm_t, damit dieser Typ in allen VHDL-Beschreibungen bekannt ist; außerdem wird dort noch der „+"-Operator für die Addition von 2 std_ulogic_vector-Werten (Interpretation als positive Binärzahl – ohne Vorzeichen) zur Verfügung gestellt.
    - VHDL-Testbench sram_controler_data_tb: Mit dieser VHDL-Testbench können Sie Ihre VHDL-Beschreibung zu Verifikationszwecken verifizieren.
2. Tipp zur VHDL-Beschreibung: Bei Signalzuweisungen ist darauf zu achten, dass die Datentypen „links" und „rechts" vom Zuweisungssymbol „<=" vom selben Typ sind. Sollte dies nicht der Fall sein, so müssen die Signale entsprechend konvertiert werden. Einige Typ-Konvertierungs-Funktionen finden Sie im VHDL-Package sram_controler_pack für die Realisierung der Funktion „+" („+"-Operator für std_ulogic).
3. Simulieren Sie Ihre Schaltung mit ModelSim und überprüfen Sie, dass die Simulation fehlerfrei durchläuft. Für die Simulation mit ModelSim sind die wichtigsten Bedienungs-Features im folgenden aufgelistet:
    - Anlegen eines Projekts: File->New->Project (wählen Sie als Project-Location das Verzeichnis aus, in dem die VHDL-Quellen liegen (Umlaute (ä,ö,ü…) im Pfad vermeiden); der Projektname ist beliebig
    - Über „add existing file" alle relevanten VHDL-Dateien zum Projekt hinzufügen
    - Compile->Compile Order auswählen und das Package an die erste Position setzen
    - Compile->Compile All auswählen
    - Bei Fehlern auf die rote Zeile im transcript-Fenster doppel-klicken und anschließend im Compile-Fenster erneut auf den Fehler doppel-klicken → der Fehler wird in der VHDL-Quelldatei angezeigt.
    - Nach der Korrektur von eventuellen Fehlern kann erneut alles kompiliert werden
    - Nachdem alle VHDL-Quellen erfolgreich übersetzt sind, kann die Simulation gestartet werden: Simulate->Start Simulation → in der Library work die zu simulierende Testbench auswählen
    - Sie können alle Signale der Testbench und Ihres VHDL-Designs mit folgendem Befehl (im transcript-Fenster) für die Darstellung im Waveform-Fenster aufzeichnen:
      log –r /*
    - Mit dem Befehl „run –all" (im transcript-Fenster) können Sie die Testbench simulieren. Die Simulation läuft mit diesem Befehl bis eine Assertion vom Level „Fatal Error" auftritt oder keine Events mehr anstehen.

- Über das Menü kann das Waveform-Fenster geöffnet werden: View->Wave Aus dem Objects-Fenster können direkt Signale in das Wave-Form gezogen werden. Sollten Variablen verwendet werden, so findet man diese, indem man das „Locals"-Fenster einblendet (über das Menü View…) – dann muss man natürlich den entsprechenden Prozess auswählen. Um Signale in einem anderen Format darzustellen (z.B. hexadezimal), markiert man die Signale und wählt über die rechte Maustaste die Option Radix.
- Im Waveform werden Fehler durch rote Dreiecke dargestellt. Fährt man mit der Maus hierüber, so wird die Fehlermeldung angezeigt.
- In dieser Testbench werden die erwarteten Ausgabewerte durch entsprechende Signale angezeigt. Dies erleichtert das Debugging.
- Das Waveform-Fenster kann über den Pfeil nach rechts oben (2. Symbol oben rechts in der Fenster-Ecke) aus dem Gesamt-Environment gelöst werden und dann auch auf dem gesamten Bildschirm dargestellt werden.
- Sollten Sie die Time-Line des Waveforms auf ns ändern wollen, so können Sie dies über „Wave->Wave Preferences" tun.
- Sollten Sie Änderungen am VHDL-Code vornehmen müssen, so können Sie nach der Speicherung der Änderung den Code erneut compilieren. Anschließend müssen Sie die Simulation wieder zurücksetzen. Dies geht am einfachsten im transcript-Fenster mit folgendem Befehl: „restart –f; run –all".
- Die Signal-Anordnung im Waveform-Fenster können Sie über File->Save Format abspeichern, um beim nächsten Mal gleich dieselbe Darstellung zu erhalten. Optional kann auch unter Datei → load im Waveform das im Ordner vorhandene wave.do verwendet werden.
- Sollten Sie Modelsim auf Ihrem eigenen Rechner installieren, so ist zu beachten, dass in der aktuellen Version die Bibliothek std_developerskit nicht mehr (offiziell) unterstützt wird – die VHDL-Sourcen sind jedoch verfügbar. Sie können die von mir vorcompilierte Bibliothek jedoch über meine WEB-Seite runterladen und das Verzeichnis einfach im Verzeichnis altera\10.0\std_developerskit drüber kopieren. Bei einem Versionswechsel ist diese Bibliothek jedoch wieder neu zu generieren…

4. Zu diesem Versuch ist der in Englisch kommentierte Quellcode abzugeben. Berücksichtigen Sie auch die Coding-Guidelines.

3 Adress-Pfad

In diesem Praktikums-Versuch sollen Sie den Adress-Pfad in VHDL beschreiben und hierfür eine eigene Testbench erstellen.

Der Adress-Pfad hat die Aufgabe die für den Speicherzugriff notwendigen Adressen zu generieren. Die 19-Bit-Start-Adresse wird zu Beginn des Speicherzugriffs übergeben, die dann 2 Mal zum richtigen Zeitpunkt um 1 inkrementiert werden muss. Für die Addition zweier std_ulogic_vector-Signale ist im Package der Plus-Operator als Funktion vorbereitet, so dass hier einfach 2 Zahlen vom Typ std_ulogic_vector addiert werden können.

Die um 1 inkrementierte aktuelle Adresse soll außerdem für das Ausgangssignal srctr_end_addr_plus1_o verwendet werden. Durch einen Reset soll die Adresse srctr_end_addr_plus1_o auf 0 initialisiert werden (Hinweis: das reset-Signal soll nicht in kombinatorischen Prozessen verwendet werden). In Abbildung 10 ist dargestellt, dass srctr_end_addr_plus1_o durch das Inkrementieren von addr_reg um 1 erzeugt wird. D.h., dass Sie sich die Frage stellen müssen, wie Sie addr_reg bei aktivem Reset belegen müssen, damit srctr_end_addr_plus1_o auf 0 initialisiert wird.

Die bei der nächsten steigenden Taktflanke zu speichernde Adresse ergibt sich wie folgt:

State + ggf. zusätzliche Bedingung	**Addr_reg =**
FSM_WRITE_MEM_13/23 oder FSM_READ_MEM_13/23	Addr_inc
(FSM_WRITE_MEM_33, FSM_READ_MEM_33 oder FSM_IDLE) und (fsm_re_i='1' oder fsm_we_i='1')	Fsm_start_addr_i
Alle anderen Fälle	Addr_reg

Eine Testbench besteht grundsätzlich aus einer leeren Entity und mindestens einer Architecture, in der die zu simulierende Schaltung instanziiert wird.

Außerdem muss natürlich der Takt und das Reset-Signal hier generiert werden. Sie können sich ggf. an der Datenpfad-Testbench hierfür Anregungen holen.

Weitere Eingabe-Signale sollten in einem Prozess generiert werden, der über wait-Statements sich mit dem Reset und dem Takt synchronisiert. Hier werden dann die weiteren Eingangssignale stimuliert und die zu erwartenden Ausgangssignale überprüft. Für die Überprüfung der Ausgangssignale verwendet man assert-Statements.

Um mehrfach verwendete Abläufe nur einmal beschreiben zu müssen, bietet sich die Verwendung von Prozeduren an. Auch hierzu finden Sie in der Datenpfad-Testbench einige Anregungen.

Einen Teil der zu stimulierenden Fälle können den folgenden Abbildungen entnommen werden:

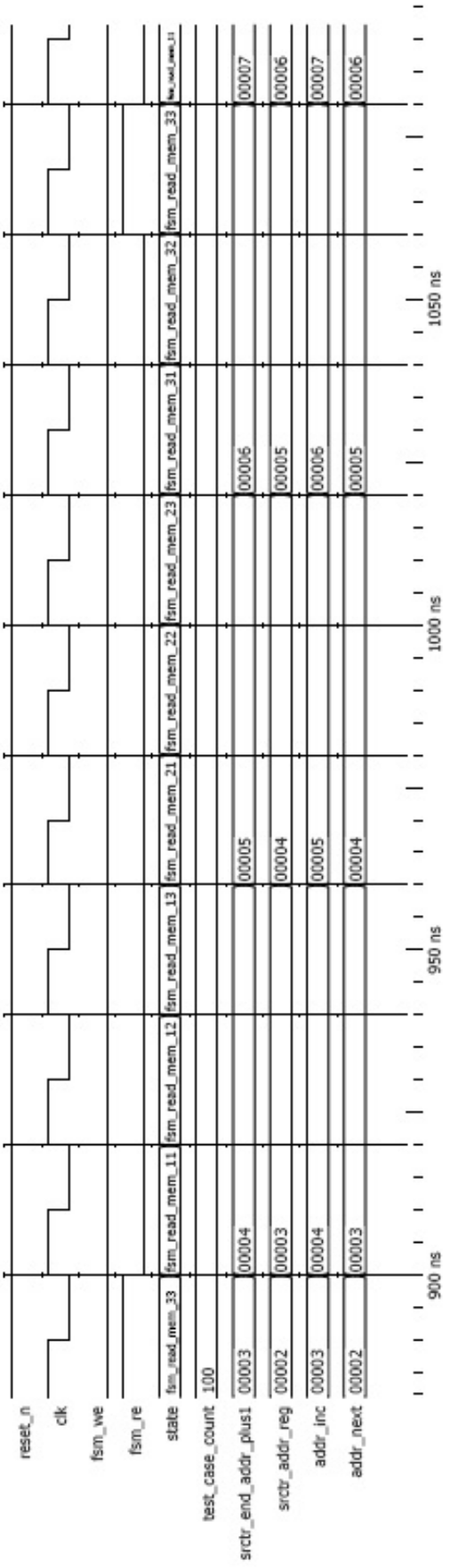


Abbildung 11 Waveform read Adress-Pfad

3.1

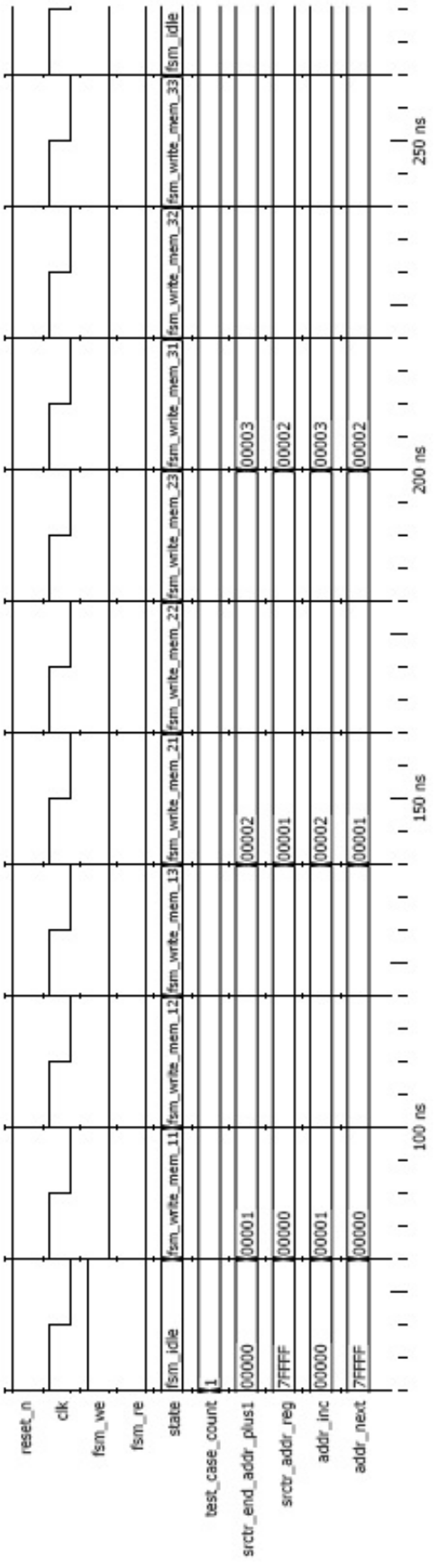


Abbildung 12 Waveform write Adress-Pfad

Aufgabenstellung

Die VHDL-Vorgaben enthalten 2 Verzeichnisse für diese Aufgabe:

- SRAM_simu_address1: Hier arbeiten Sie zunächst drin (bis zu Teilaufgabe 5): Erstellung VHDL-Code für den Adresspfad und Testbench-Erstellung
- SRAM_simu_address2: Hier sollen Sie Ihr VHDL-Design nochmal mit meiner Testbench verifizieren. Meine Testbench ist für Sie nicht lesbar, aber simulierbar.

Ihre Aufgaben:

1. Erstellen Sie die VHDL-Beschreibung für den Adress-Pfad (sram_controler_address – Vorlagenverzeichnis SRAM_simu_address1): Entity+Architecture. Hierfür steht Ihnen folgende VHDL-Beschreibung zur Verfügung:
   - VHDL-Package sram_controler_pack: Definition des Datentyps fsm_t, damit dieser Typ in allen VHDL-Beschreibungen bekannt ist; außerdem wird dort noch der „+"-Operator für die Addition von 2 std_ulogic_vector-Werten zur Verfügung.
2. Erstellen Sie eine Testbench, um Ihre Schaltung zu simulieren. Überlegen Sie sich hierfür welche unterschiedlichen Fälle Sie hierbei stimulieren und überprüfen müssen.
3. Simulieren Sie Ihre Schaltung mit ModelSim und überprüfen Sie, dass die Simulation fehlerfrei durchläuft.
4. Fügen Sie absichtlich einen Fehler in Ihren VHDL-Code ein, so dass die Testbench einen Fehler ausgibt. Stellen Sie den Fehler im Waveform dar (gehört auch in das Protokoll).
5. Tauschen Sie mit einer anderen Praktikumsgruppe die Testbench aus und überprüfen Sie, ob diese auch bei Ihnen fehlerfrei durchläuft. Diskutieren Sie ggf. Diskrepanzen.
6. Kopieren Sie Ihre Datei sram_controler_address.vhd vom Verzeichnis SRAM_simu_address1 in das Verzeichnis SRAM_simu_address2 und ergänzen Sie folgende 2 Zeilen in Ihrem Code:
   library project_lib;
   use project_lib.fsm_pack.all;
   Sollten Sie weitere Modifikationen im sram_controler_pack vorgenommen haben, so müssen Sie diese beiden Versionen der Dateien ebenfalls anpassen.
   Stellen Sie sicher, dass die im Verzeichnis SRAM_simu_address2 befindliche Testbench sram_controler_address_tb_hide mit Ihrem Design fehlerfrei simulierbar ist.

Hinweis: Die Bibliothek projekct_lib wurde für die Modelsimversion 10.1d kompiliert. Wenn Sie mit einer jüngeren Modelsimversion arbeiten, erhalten Sie unter Umständen folgende Fehlermeldung beim Starten der Simulation:

```
#  **  Fatal:  (vsim-3381)  Obsolete  library  format  for  design  unit.  Design  unit
…/project_lib.fsm_pack'
#    Time: 0 ps  Iteration: 0  Root: / File: NOFILE
# FATAL ERROR while loading design
# Error loading design
```

Gehen Sie in diesem Fall wie folgt vor: Verschieben Sie das project_lib-Verzeichnis in ein anderes Verzeichnis. Anschließend importieren Sie in Modelsim die project_lib-

Bibliothek (File→Import→Library). Beim Import wird die Library in das neue Format portiert.

7. Zu diesem Aufgabenteil ist der in Englisch kommentierte Quelltext abzugeben – idealerweise ergänzt um folgendes:
    - Bild von der Simulation mit Erklärung des Ablaufs (Screenshot)
    - Bild von der Simulation mit Fehler + Erklärung (Screenshot + kurze Beschreibung des Fehlers)

4 SRAM-Controller

Für den SRAM-Controler werden noch zusätzlich benötigt (siehe Abbildung 10: Blockschaltbild des SRAM Speicher Controllers):

p_fsm

Hier wird die Zustandsfolge des benötigten Automaten implementiert. Das Zustandsfolge-Diagramm ist weiter oben bereits dargestellt worden.
Der Automat soll in einem Block (siehe VHDL-Folien 211-214) beschrieben werden.

control_seq_p

Dieser Prozess ist sequentiell, um, Spikes beim Schalten der Signale zum schreiben in oder lesen vom SRAM-Speicher zu vermeiden. Da die srctr_signale low_active sind, müssen sie zum Aktivieren den Wert ‚0' haben. srctr_we_reg_n_o und srctr_oe_reg_n_o sind die Signale, die hier ihre Werte zugewiesen bekommen.

control_comb_p

Da die gesamten Signale schon im vorherigen Takt des Speicherzugriffs stabil anliegen, ist es an dieser Stelle nicht nötig, diese zu registern. srctr_ce_n_o kann für Energiesparzwecke im State FSM_IDLE deaktiviert werden, sonst muss es, wie oben beschrieben, aktiviert sein. Für den Zugriff auf immer ein Byte des Speichers ist es notwendig srctr_lb_n_o den invertierten Wert von srctr_ub_n_o zu zuweisen. Diese Werte kann man am LSB des addr_reg bestimmen, da dieses sich für jede Speicheradresse ändert.

srctr_idle_p

Dieser Prozess – oder die nebenläufige Signalzuweisung – generiert das Ausgabesignal srctr_idle_o aus dem aktuellen Zustand. Sinn dieses Signals ist es, zu signalisieren, dass der SRAM-Controler im nächsten Takt bereit ist, neue Daten zu schreiben oder zu lesen.

4.1 Aufgabenstellung

1. Erstellen sie die sram_controler-Komponente. Ergänzen Sie die fehlenden Prozesse und instanziieren Sie den Daten- und Adresspfad.
2. Simulieren Sie Ihre sram_controler-Komponente mit der zur Verfügung gestellten Testbench.
3. Binden Sie Ihren SRAM-Controler in die zur Verfügung gestellte Top-Level-Komponente des Diktiergeräts ein (siehe Abbildung 1). Die audio-codec-Komponente ist bereits implementiert und ist in der Top-Level-Komponente instanziiert. Synthetisieren Sie unter Verwendung der Quartus-Werkzeuge Ihre Schaltung für das

auf dem DE2-Board befindliche FPGA (Tipp: das entsprechende Pin-Assignment nicht vergessen).

4. Erproben Sie die Schaltung auf dem DE2-Board.
5. Zu diesem Versuch ist folgendes abzugeben:
    - Elektronisch: alle für das Diktiergerät notwendigen VHDL-Dateien als ZIP-Datei (über Moodle)
    - Eine kurze Demo Ihrer Implementierung

Quellenverzeichnis

256K x 16 HIGH SPEED ASYNCHRONOUS CMOS STATIC RAM

(ISSI_61WV25616.pdf)

Integrated Silicon Solution, Inc. – www.issi.com

„Entwicklung eines Nios II basierenden Audiodesigns mit Hilfe des Altera® DE2 Development and Education Boards“, Ersteller: **Nyberg, Ralph**

Labor für Rechnerarchitekturen und Programmierung peripherer Baugruppen der Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven, Prof. Dr.-Ing. Gerd von Cölln,

Nios Development Board Reference Manual

http://www.altera.com/literature/manual/mnl_nios2_board_cycloneII_2c35.pdf

Cyclone II Device Handbook

http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf

Nios II Processor Reference Handbook

http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Wolfson Microelectronics WM7831 / WM7831L Data Sheet

https://www.cirrus.com/en/pubs/proDatasheet/WM8731_v4.9.pdf

Nios II Development Kit Getting Started User Guide

http://www.altera.com/literature/ug/ug_nios2_getting_started.pdf