VHDL-Praktikum

Aufgabe 3:
Minima-Rechenmaschine

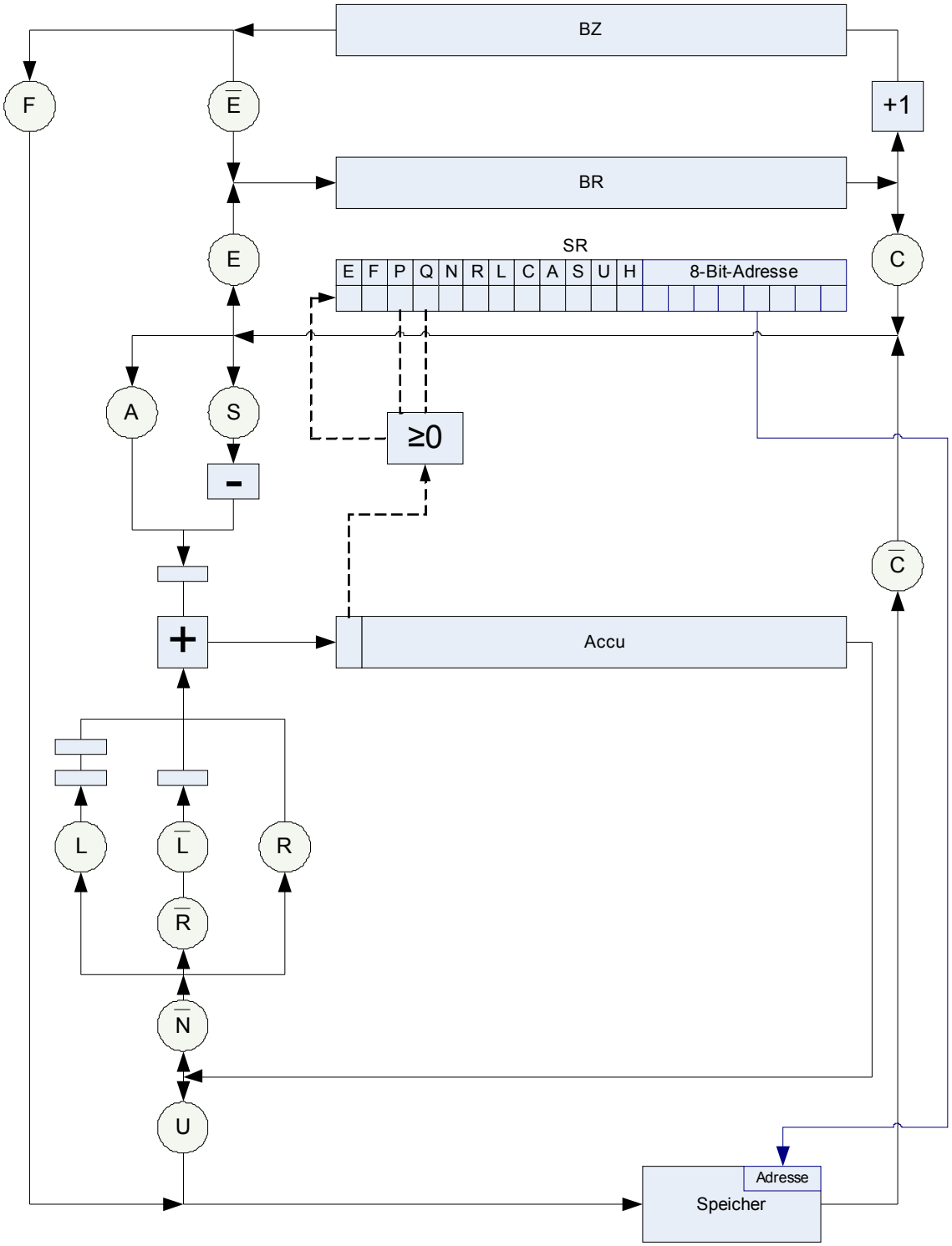
SS 2004

Andreas Schibilla (ii4900)

1. Inhaltsverzeichnis

2. Einleitung

Die Aufgabe besteht in der Nachbildung der seriellen Rechenmaschine „Minima“ in eine synthesefähige VHDL-Beschreibung, so dass sie in einen FPGA vom Typ FLEX10k10LC84-4 hineinpasst, wobei der Arbeitsspeicher in den Speicherblöcken dieses Bausteins untergebracht werden soll. Außerdem soll das Design auf das Testboard heruntergeladen und ausgetestet werden können. Die Rechnerstruktur der „Minima“ sei durch das nachfolgende Bild festgelegt:



3. Projektgliederung und Entwicklungstools

3.1 Projektstruktur und Erstellung

3.1.1 Die eingesetzten Tools

Für die Bearbeitung, Simulation und Synthese der VHDL-Beschreibung der Minima werden folgende Hard- und Softwarekomponenten vorausgesetzt:

Hardware
• Windows-kompatibler PC für Simulation und Programmierung • Testboard der FH-Wedel inkl. dem Baustein „FLEX10K10LC84-4“

Software
• Texteditor (z.B. SciTE oder Ultraedit) zum Editieren der VHDL-Dateien • Altera Quartus II 4.0 Web Edition • VHDL-Board-Programm zum Konfigurieren und Testen des Designs

3.1.2 Die Projektdateien

Im Verzeichnis **„\Aufgabe3\“** der CD-ROM befinden sich drei Unterverzeichnisse mit den Projektdateien der Minima.
In dem **Ordner „Komponenten“** liegen verschiedene Quartus-Projekte der einzelnen Bausteine (wie z.B. Befehlsregister oder Steuerregister des Steuerwerks) der Gesamtrechenmaschine. Sie können mit den jeweiligen .QPF-Dateien in Altera Quartus II geöffnet werden und beinhalten zum einen den entsprechenden VHDL-Code, zum anderen ist auch jeweils eine Simulation mittels Waveform möglich.
Der **Ordner „Minima“** enthält das komplette Projekt zur Minima-Rechenmaschine, in dem alle Komponenten zusammengeführt und eingebunden sind. Das Verzeichnis beinhaltet alle VHDL-Dateien, sowie zugehörige Dateien zur Konfiguration, Übersetzung, Simulation und Synthese mittels Altera Quartus II. Das Projekt kann durch Öffnen der Datei **„MINIMA.qpf“** geladen werden. Zum Projekt gehören die folgenden VHDL-Dateien:

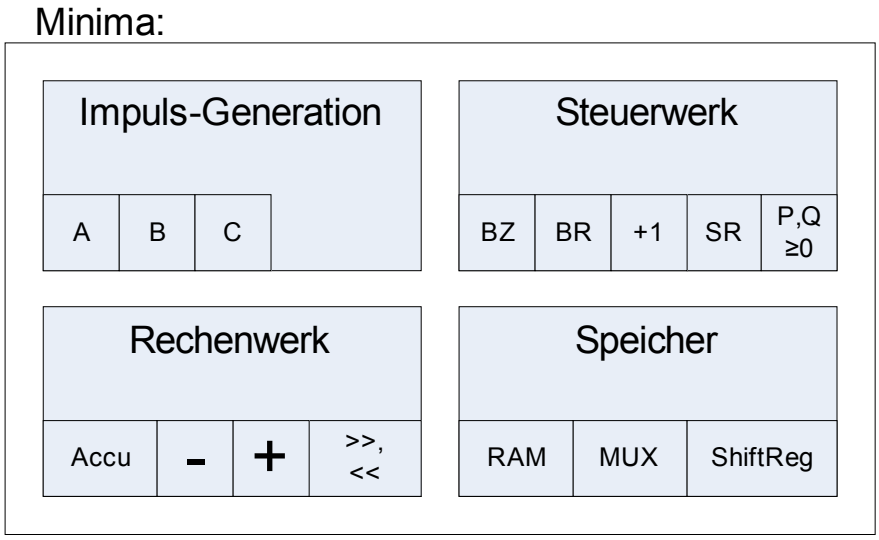
Minima-Zusammenfassung:	**Steuerwerk:**	**Speicher:**
MINIMA.VHD	BR.VHD	RAM.VHD
	BR_SHIFTREG.VHD	SHIFTREG.VHD
Rechenwerk:	BZ.VHD	SPEICHER.VHD
ACCU.VHD	PLUS_EINS.VHD	
ADDER.VHD	SR.VHD	**Impulse:**
ADDSUB.VHD	STEUERWERK.VHD	COUNTER.VHD
ALU.VHD		IMPULSE.VHD
SHIFTING.VHD		
VERZOEGERN.VHD		

Jede Datei enthält eine Entwurfseinheit mit Schnittstelle (Entity) und Innenleben (Architecture).

Das dritte **Verzeichnis „Debug“** enthält ebenfalls das komplette Projekt der Minima, jedoch sind in dieser Version zusätzliche Signale nach außen geführt, so dass sie in der Simulation angezeigt werden können. Da dies nur mit einem Baustein mit mehr Pins als der FLEX10k10LC84-4 möglich ist, wurde hier ein anderer, aber entsprechend ähnlicher Baustein der gleichen Familie gewählt.

4. Struktur und Aufbau der Minima

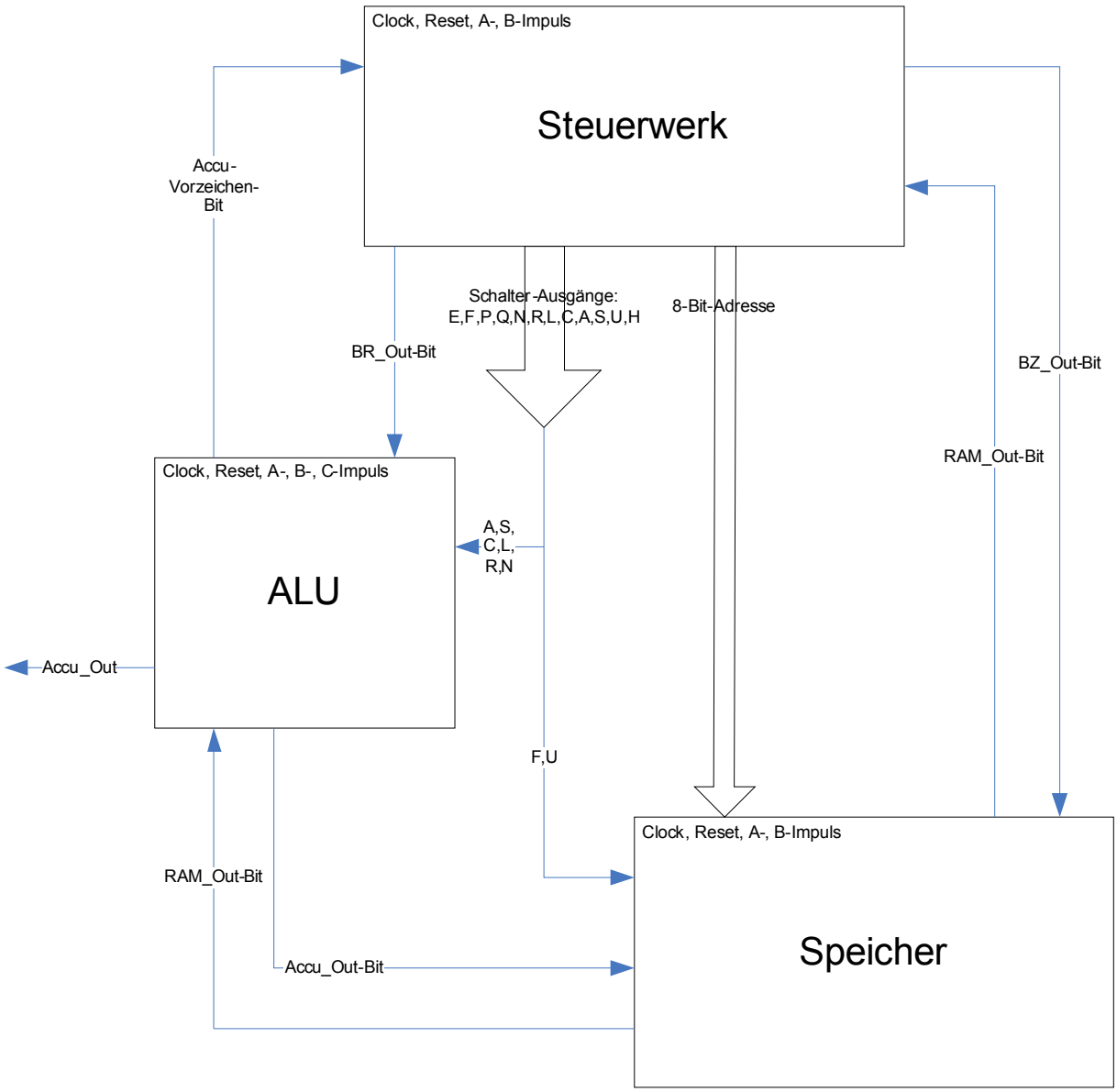
Die VHDL-Nachbildung der Rechenmaschine „Minima“ ist in vier größere Komponenten gegliedert, die in der untenstehender Abbildung gezeigt werden:



Auf der nächsten Seite ist der Zusammenhang zwischen den drei wichtigsten Modulen „Steuerwerk“, „Rechenwerk“ und „Speicher“ abgebildet. Alle drei Bausteine werden von außen über das Clock- und Reset-Signal beschaltet und „erhalten“ außerdem die drei Impulse A, B, C vom internen Modul der Impuls-Generation.
Die erste Komponente **Impulsgeneration** dient lediglich der Erzeugung der internen Impulse A,B,C in jeder Wortzeit. Die Impulse werden von den anderen Bausteinen verwendet und unterstützen die Steuerung der Abläufe verschiedener Operationen (→ siehe Kap. 5).
Die zweite Komponente, das **Steuerwerk**, enthält das Befehlsregister, den Befehlszähler und das Steuerregister. Hier wird die parallele Übernahme der Information aus dem Befehlsregister in das Steuerregister kontrolliert. Außerdem werden die Schiebeoperationen des Befehlsregisters und Befehlszählers passend zu den jeweiligen Taktzyklen gesteuert. Dazu gehört das Inkrementieren der Adresse vor dem Eingang im Befehlszähler, sowie das Hineinschieben der Bits von den richtigen Eingängen, je nach aktueller Schalterstellung des ausgeführten Befehls. Ferner wird bei bedingten Befehlen (Schalter P oder Q gesetzt) der Inhalt des Akkumulators ausgewertet. Hierzu ist das Accu-Vorzeichen-Bit vom Rechenwerk zum Steuerwerk gelegt. Damit die anderen Bausteine die aktuellen Schalterzustände des Steuerregisters abfragen können, sind diese wie in der Abbildung zu erkennen, nach außen geführt. Ebenso sind die Adressleitungen mit dem Speicher verbunden.
Die dritte Komponente umfasst das **Rechenwerk** der Maschine. Hier sind die Routinen zur Addition und Subtraktion eines Wortes im Speicher mit dem Akkumulator untergebracht. Die Module zum Links- und Rechtsschieben befinden sich ebenso im Rechenwerk.

Eine den Wortzeiten und Timing angepasste Steuerung überwacht die Operationen und verbindet die untergeordneten Bausteine. Dazu ist das Rechenwerk sowohl mit dem Steuerwerk als auch mit dem Speicher verbunden. Über das RAM_Out-Bit können Daten vom Speicher gelesen und über das Accu-Out-Bit vom Akkumulator in den Speicher geschrieben werden. Um Adressen vom Befehlsregister in den Akkumulator zu schieben, ist die Verbindung über das BR_Out-Bit hergestellt worden.
Die vierte Komponente beschreibt den **Speicher**, der in der VHDL-Beschreibung durch ein RAM modelliert wurde. Da der Zugriff auf das RAM parallel und nicht seriell erfolgen muss, ist hier eine Instanz mit Schieberegister zur Nachbildung des seriellen Datenstroms integriert. Eine State-Maschine in der Speicherkomponente verwaltet das Setzen der Speicheradresse, sowie Lade- und Schreibzugriffe. Über das BZ_Out-Bit vom Steuerwerk kann die Speicher-Komponente den Befehlszähler bei Unterprogrammsprüngen als Rückkehradresse im RAM (Adresse 5) ablegen.

Abbildung zum Aufbau und Struktur der Minima:



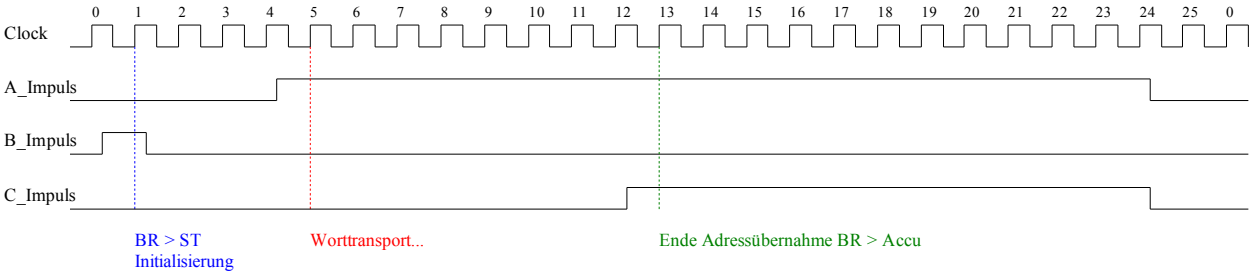
5. Erzeugen der internen Impulse: „E_IMPULSE“

5.1 Aufbau und Funktionsweise

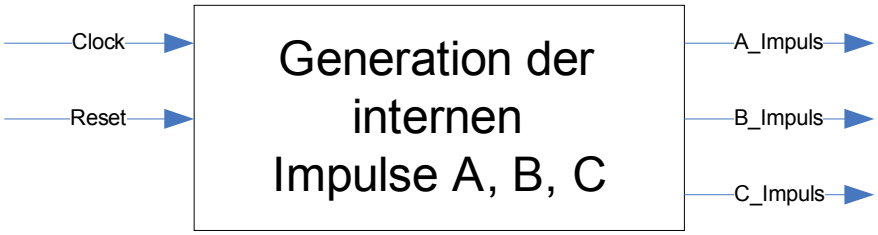
Die Arbeitsweise der Minima orientiert sich sowohl am Taktsignal als auch an den drei speziellen Steuerimpulsen A, B und C. Der zeitliche Ablauf besteht aus gleichartigen Zyklen, von denen jeder die Länge einer Wortzeit hat. Die Wortzeit besteht aus 26 Taktzeiten. Innerhalb einer Wortzeit steuern die Impulse folgende Operationen:

- **Impuls_A:**
  Steuert einen Worttransport zwischen den verschiedenen Registern bzw. Speicherzellen.
- **Impuls_B:**
  Steuert die parallele Übernahme der Informationen vom Befehlsregister ins Steuerregister vor der Ausführung von jedem Befehl. Außerdem leitet dieser Impuls unterschiedliche Initialisierungen im Steuer- bzw. Rechenwerk und in der Speichereinheit ein.
- **Impuls_C:**
  Steuert im Rechenwerk die Adressübernahme vom Befehlsregister in den Akkumulator.

Der genaue Verlauf der drei Impulse wird in der folgenden Grafik demonstriert:



Die VHDL-Beschreibung zur Erstellung der drei internen Impulse befindet sich in der Datei **„IMPULSE.VHD“**. Die Schnittstelle des Moduls wird in der untenstehenden Abbildung gezeigt:

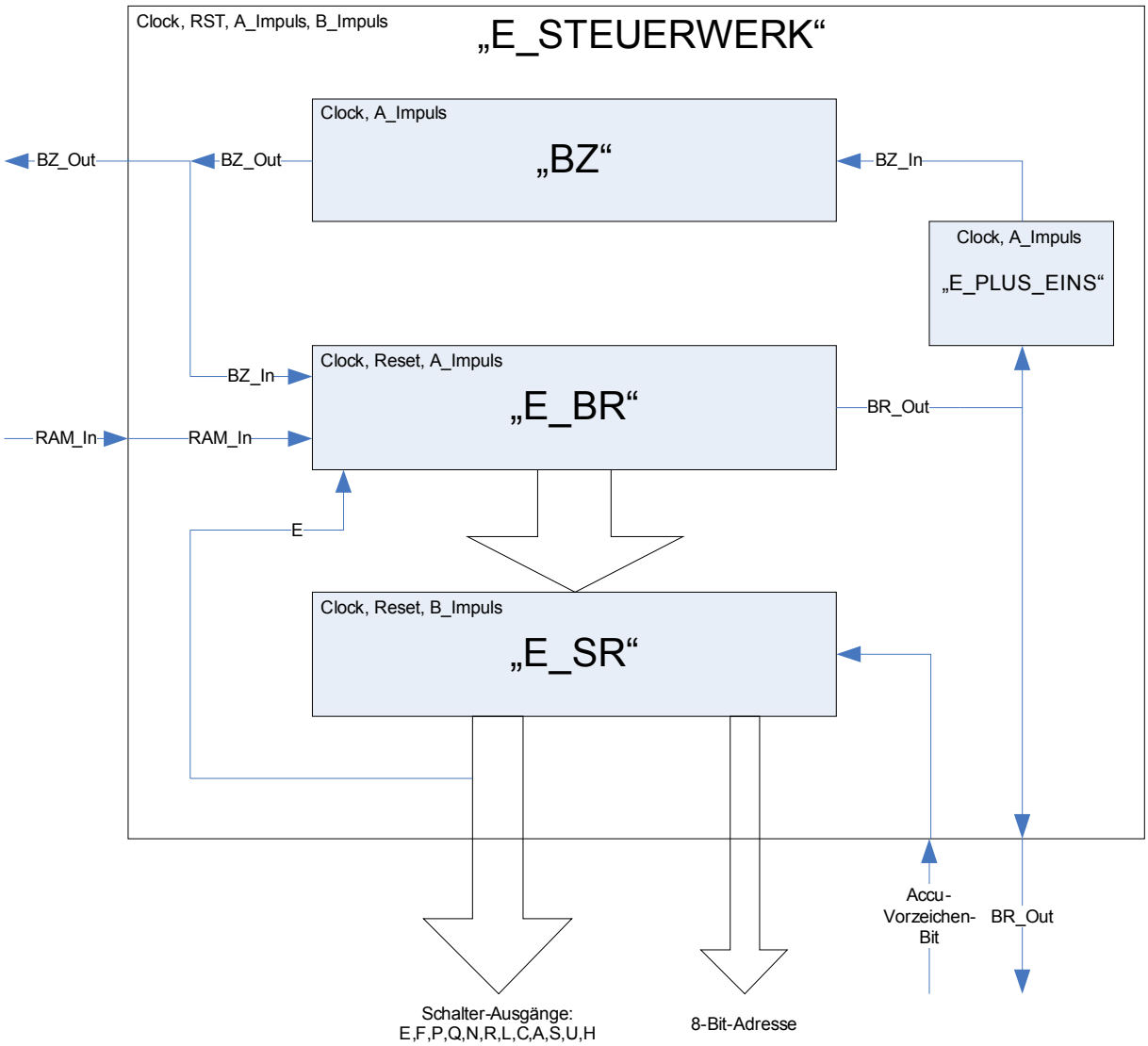


Das Clock-Signal am Eingang setzt einen Zähler innerhalb der Architektur in Gang, der bei jedem Takt-Impuls inkrementiert wird, bis die Wortzeit vollständig durchlaufen wurde. Der Zähler wurde mit dem MegaFunction-Wizard-Tool erstellt und befindet sich in der Datei **„COUNTER.VHD“**. Er ist auf 26 Einheiten (0-25) eingestellt und führt nach Erreichen der letzten Zahl automatisch einen Reset durch. Der jeweilige Zählerstand liegt ständig am Ausgang Q an. Abhängig vom Zählerstand werden die jeweiligen Pegel der drei Impulse im Prozess P2 erzeugt.

6. Das Steuerwerk: „E_STEUERWERK“

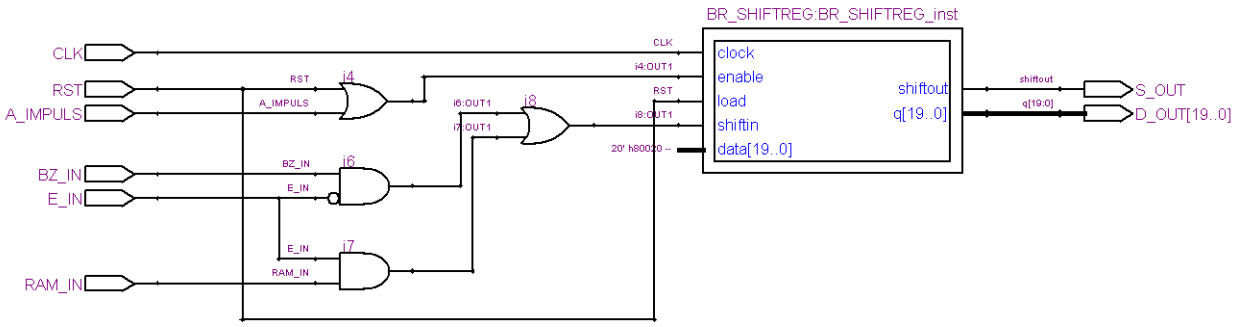
6.1 Aufbau und Funktionsweise

Das Steuerwerk besteht aus dem Befehlszähler, dem Befehlsregister, dem Steuerregister und dem Inkrementierer (+1 Addierer). Mit dem VHDL-Code aus der Datei **„STEUERWERK.VHD“** wird das Zusammenspiel dieser Komponenten sowie die Verbindungen untereinander beschrieben. Dazu werden die vier Bausteine jeweils mit Hilfe einer Port Map eingebunden und über interne Signale gekoppelt. Die folgende Abbildung veranschaulicht diese Verknüpfungen. Die typischen Signale, die den einzelnen Komponenten zusätzlich von außen zugeführt werden müssen, stehen jeweils oben links in der Ecke der Komponenten. Das Signal „RAM_In“ wird dem Steuerwerk vom Speicher und das Signal „Accu-Vorzeichen-Bit“ vom Rechenwerk zugeführt. Sowohl der Befehlszähler, als auch das Befehls- und Steuerregister werden durch Schieberegister in eigenen Komponenten repräsentiert, wobei die hinausgeschobenen Bits „BZ_Out“ und „BR_Out“ zusätzlich aus dem Steuerwerk hinausgelegt sind, da diese Signale in anderen Modulen benötigt werden.



6.2 Das Befehlsregister „E_BR"

Entity und Architektur des Befehlsregisters befinden sich in der Datei **„BR.VHD"**. Im wesentlichen besteht diese Komponente aus einem Schieberegister, welches mit Hilfe des MegaFunction-Wizard erstellt wurde und in der Datei **„BR_SHIFTREG.VHD"** zu finden ist. Es wird mit Hilfe einer Port Map eingebunden. Die untere Grafik zeigt das Synthese-Ergebnis und veranschaulicht die logischen Verknüpfungen der Eingangssignale:



Man erkennt auf der rechten Seite das Ergebnis bzw. den Ausgang des Befehlsregisters. Zum einen ist das gesamte Wort („D_OUT") nach außen geführt, was für die parallele Übernahme der Informationen ins Steuerregisters nötig ist. Zum anderen wird das letzte Bit bei jedem Takt des Worttransportes über die Leitung „S_OUT" hinausgeschoben, um so in den Akkumulator bzw. den Befehlszähler zu gelangen. Auf der Eingangsseite werden zunächst „Clock-„ und „Reset"-Signal direkt mit dem Schieberegister verbunden. Das „Enable"-Signal ist über ein OR-Gatter mit „Reset" und dem „A-Impuls" gekoppelt, da im Falle eines Resets ein neues Wort geladen werden muss („enable" und „load" auf high, das Wort liegt an „data" an). Während des Worttransportes (A-Impuls) soll das Schieben erfolgen (nur „enable" auf high). Das Eingangsbit für die Schiebeoperation („shiftin") stammt je nach Befehl vom Befehlszähler (BZ_IN) oder dem Speicher (RAM_IN):

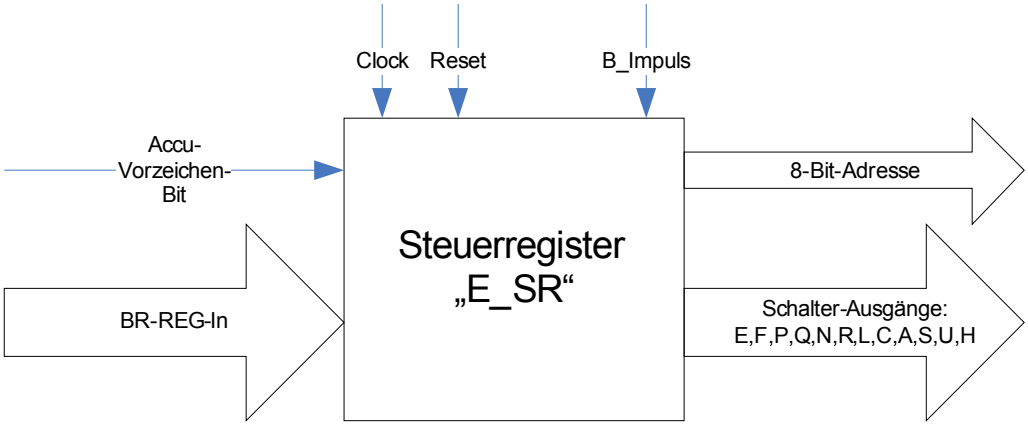
BZ_IN wenn $\overline{E}$

RAM_IN wenn $E \wedge \overline{C}$ (da der Befehl EC unsinnig ist, bleibt nur das E-Signal relevant).

6.3 Das Steuerregister „E_SR"

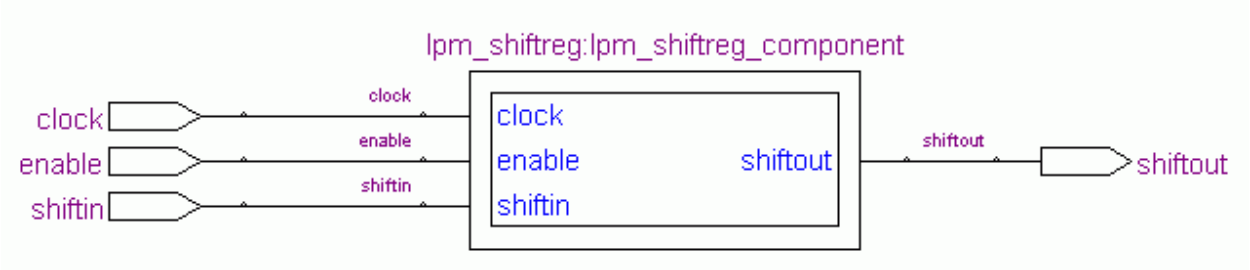
Das Steuerregister ist ein 20-Bit-Register. Die ersten 8 Bit repräsentieren die Adresse, die weiteren 12 Bit entsprechen den Befehlsschaltern der Minima. Das Register befindet sich in der Architektur „A_SR" in der Datei **„SR.VHD"**. Auf der nächsten Seite ist die Entity in einer Abbildung dargestellt. In der VHDL-Beschreibung wird zunächst mit Hilfe einer logischen Verknüpfung bestimmt, ob der aktuell anliegende Befehl (durch „REG_IN" gegeben) eine Bedingung (P oder Q) aufweist. Ist dies der Fall, so wird das Vorzeichen-Bit des Akkumulators („ACCUBIT") überprüft und ein internes Signal gesetzt („INT_OK"). Mit einer entsprechenden Konjunktion werden die Schalter-Bits zum Ausgang geführt. Da sich die Eingänge vom Befehlsregister („REG_IN") und Accu-Vorzeichen-Bit („ACCUBIT_IN") während des Schiebevorgangs ständig verändern, werden diese Signale intern zwischengespeichert („INT_REG" und „INT_ACCUBIT"). Das Initialisieren geschieht im Prozess „P1" mit Hilfe einer trivialen State-Maschine, gesteuert durch den B-Impuls der Minima.

Abbildung zur Entity „E_SR“ des Steuerregisters:



6.4 Der Befehlszähler „BZ“

Der Befehlszähler ist aus einem 20-Bit-Schieberegister aufgebaut und speichert den jeweils nächsten Befehl. Dazu wird während des Zyklus des Worttransportes am Eingang jeweils ein Bit des Befehlsregisters in das Schieberegister hinein geschoben. Der Adressanteil des Befehls wird dabei jedoch um Eins inkrementiert, so dass im Befehlszähler wirklich das richtige Fetch-Kommando zum folgenden Befehl steht. Das rechtsschiebende Register wurde mit dem MegaFunction-Wizard erstellt und in der Datei **„BZ.VHD“** abgelegt. Die Entity hat folgenden Aufbau:



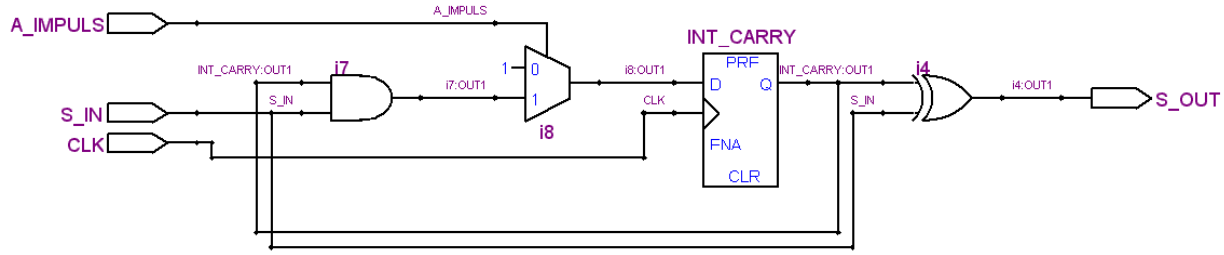
6.5 Das „Plus Eins“-Addierwerk „E_PLUS_EINS“

Bei jedem Takt während des Worttransport-Zyklus wird das vom Befehlsregister hinausgeschobene Bit über die „Plus Eins“ Inkrementier-Komponente in den Befehlszähler hineingeschoben. Die ersten 8-Bit des gesamten Wortes repräsentieren dabei die Adresse, welche im Befehlszähler um Eins erhöht stehen muss, damit dieser auf den nächsten zu ladenden Befehl zeigt. Die Aufgabe der „Plus Eins“-Komponente ist es also, den seriellen Datenstrom her vom Wert zu inkrementieren. Betrachtet man den Rechenvorgang, so stellt man fest, dass für jeden Rechenschritt genau eine AND- und eine XOR-Verknüpfung benutzt werden können, was im folgenden Beispiel demonstriert wird:

Beispiel zur Addition:

13+1=14 **1101** **+0001** **1110**	Es sind zwei Signale vorhanden, das Ergebnisbit für die aktuelle Stelle und ein internes Carry-Bit, welches mit 1 initialisiert werden muss: 1. Stelle: Ergebnis = **1** xor Carry = **0** Carry = **1** and Carry = 1 2. Stelle: Ergebnis = **0** xor Carry = **1** Carry = **0** and Carry = 0 3. Stelle: Ergebnis = **1** xor Carry = **1** Carry = **1** and Carry = 0 4. Stelle: Ergebnis = **1** xor Carry = **1** Carry = **1** and Carry = 0 **Eingabebits** **Ergebnisbits**

Die VHDL-Beschreibung ist in der Datei **„PLUS_EINS.VHD“** abgelegt. Das Synthese-Ergebnis dazu sieht wie folgt aus:



Da das aus dem Befehlsregister hinausgeschobene Bit bereits kurz nach jedem Takt-Impuls anliegt, kann die Addition und Ausgabe des Signals zum Befehlszähler direkt über ein XOR-Gatter unmittelbar und rechtzeitig vor dem nächsten Takt anliegen (Ergebnis „S_OUT“ = „S_IN“ xor „INT_CARRY“). Das Carry-Signal („INT_CARRY“) wird durch ein Flip Flop gespeichert und innerhalb des Prozesses „P1“ in Abhängigkeit vom A-Impuls gesetzt.

6.6 Simulation und Timing-Verhalten des Steuerwerks

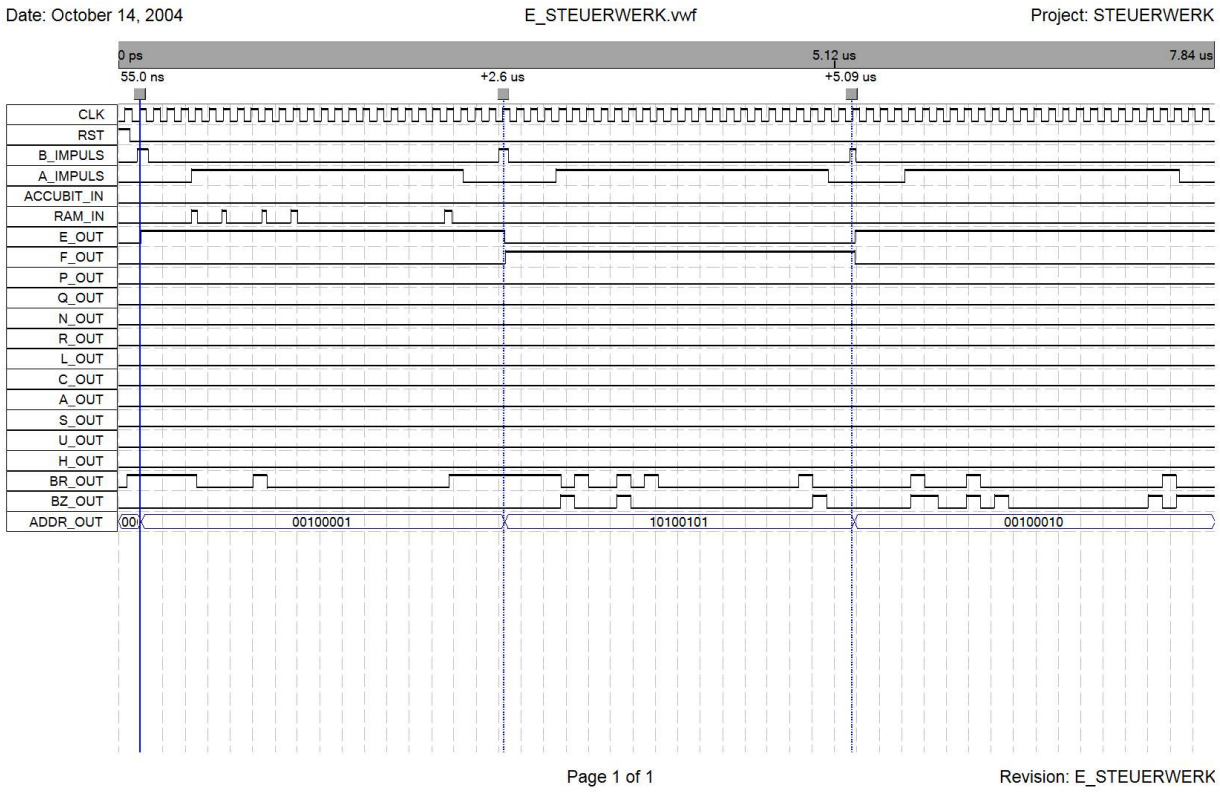
Die Grafik auf der nächsten Seite zeigt den genauen Verlauf der Ausführung von drei Befehlen im Steuerwerk. Die blauen Linien (Time Bar) zeigen jeweils den Beginn mit dem B_Impuls, der die Übernahme der Datenbits vom Befehlsregister ins Steuerregister einleitet. Man erkennt, dass jeweils etwas verzögert, nach der Propagation Delay Time, sowohl der Adressbus also auch die Steuer-Schalter für die aktuelle Befehlsausführung gesetzt werden. Während der ersten Wortzeit wird der Befehl „E $21“ ($100001_{bin} = 21_{hex}$) ausgeführt, womit ein neuer Befehl aus dem Speicher geladen wird (Fetch-Phase). Da der Befehl aus dem Speicher stammt, ist er hier in der Simulation am Signal „RAM_IN“ während der Worttransportphase (= Impuls_A ist high) seriell (LSB zuerst, MSB zuletzt) angegeben worden. Im Beispiel wurden folgende Bits angelegt:

1010 0101 0000 0000 0010

Da das LSB vorne steht, muss der Befehl rückwärts gelesen werden: „F $A5“. Die Bearbeitung wird in der zweiten Wortzeit durchgeführt.

Der B_Impuls leitet wieder den Worttransport vom Befehlsregister ins Steuerregister ein. Anschließend liegen die neue Adresse sowie die Schalterzustände an (nur F ist auf high). Dieses Kommando schiebt das Wort aus dem Befehlszähler (BZ_OUT) in den Speicher und gleichzeitig in das Befehlsregister, als Vorbereitung (Fetch-Phase) für den nächsten Befehl. Der dritte B_Impuls leitet eben diesen Ladevorgang ein. Es ist wie im ersten Fall nur das Bit „E“ gesetzt, die Adresse wurde jedoch inkrementiert: „E $22“.

Abbildung: Simulationsablauf von drei Befehlen im Steuerwerk

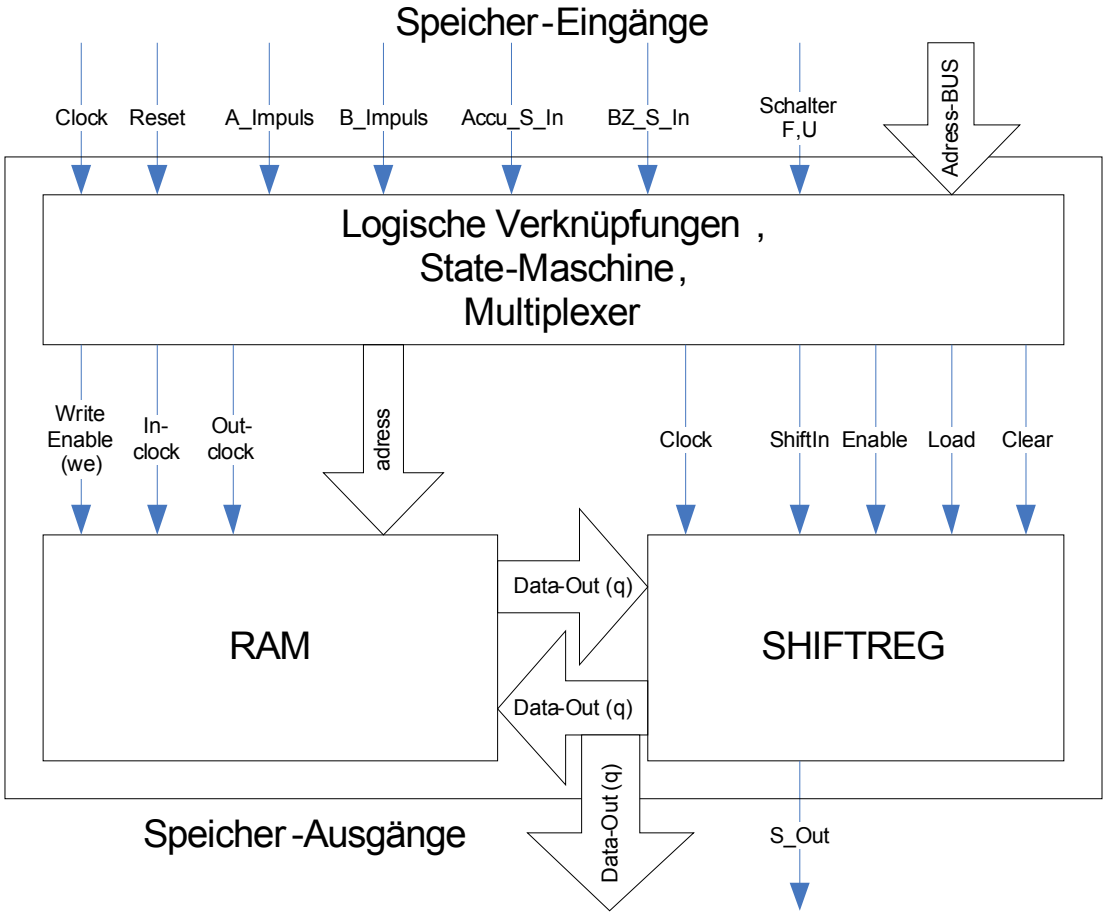


7. Der Speicher: „E_SPEICHER“

7.1 Aufbau und Organisation des Arbeitsspeichers „E_SPEICHER“

Der Trommelspeicher der Minima wird in der FPGA-Umsetzung durch ein RAM nachgebildet. Da ein RAM nur parallelen Zugriff auf die Speicherzellen gestattet, wird das serielle Verhalten mit Hilfe eines Schieberegisters und entsprechender State-Maschine imitiert. Auf dem Zielbaustein (FLEX10k10) stehen RAM-Bänke mit einer Gesamtkapazität von 6144 Bits zur Verfügung. Diese sind mit Hilfe der MegaFunction/LPM auf 256x20Bit konfiguriert. Die zugehörige Entity „RAM“ befindet sich in der Datei **„RAM.vhd“**. Außerdem ist das benötigte Schieberegister durch MegaFunctions/LPM erstellt worden. Die Entity lautet „SHIFTREG“ und ist in der Datei **„SHIFTREG.vhd“** zu finden. Die State-Maschine, die beide Komponenten verbindet, und den Speicher nach außen als Gesamteinheit präsentiert, steht in der Datei **„SPEICHER.VHD“**.

Die folgende Abbildung verdeutlicht den Aufbau bzw. die Struktur des Speichers:

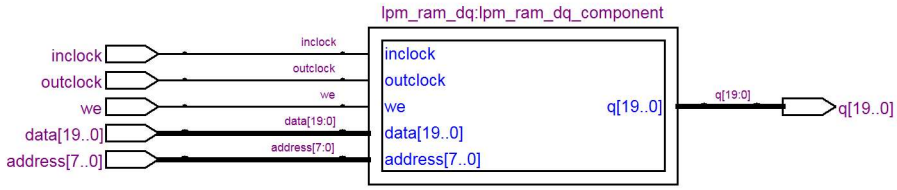


Die Steuersignale für den RAM-Baustein („we“, „inclock“, „outclock“) werden entsprechend des aktuellen Takts in der Wortzeit durch die State-Maschine generiert (→ siehe Kap. 7.4). Beim Ladevorgang wird über den Bus „q“ ein Wort vom Speicher in das Schieberegister geladen und dort seriell hinausgeschoben. Beim Speichervorgang werden die Bits hingegen erst in das Schiebregister geschoben und anschließend über den umgekehrten q-Bus ins RAM gebracht. Die im RAM verwendete Adresse („adress“) wird je nach Befehl/Schalterzustand auf den äußeren Adress-BUS gesetzt (der vom Steuerregister kommt) oder im Falle eines Unterprogrammsprungs auf den festen Wert 5 eingestellt.
Das Schieberegister „SHIFTREG“ wird ebenfalls durch die State-Maschine kontrolliert. Über das Signal „Shiftin“ wird das Bit angegeben, welches hineingeschoben werden soll. Es wird über die enthaltende Logik entweder dem Eingang „Accu_S_In“ (bei aktivem U-Schalter) oder „BZ_S_In“ (bei aktivem F-Schalter) zugewiesen. Über das „Load“-Signal wird das Schieberegister nach Auftreten des B-Impulses initialisiert und über „Enable“ kann der Schiebevorgang ein- bzw. ausgeschaltet werden.

7.2 Modellierung des Trommelspeichers aus dem RAM „RAM“

Mit Hilfe der MegaFunction „LPM_RAM_DQ“ kann auf den internen RAM-Speicher im FPGA zugegriffen werden. Die Komponente hat den unten gezeigten Aufbau und unterstützt zusätzlich die Vorgabe des Speicherinhalts durch eine Initialisierungsdatei.

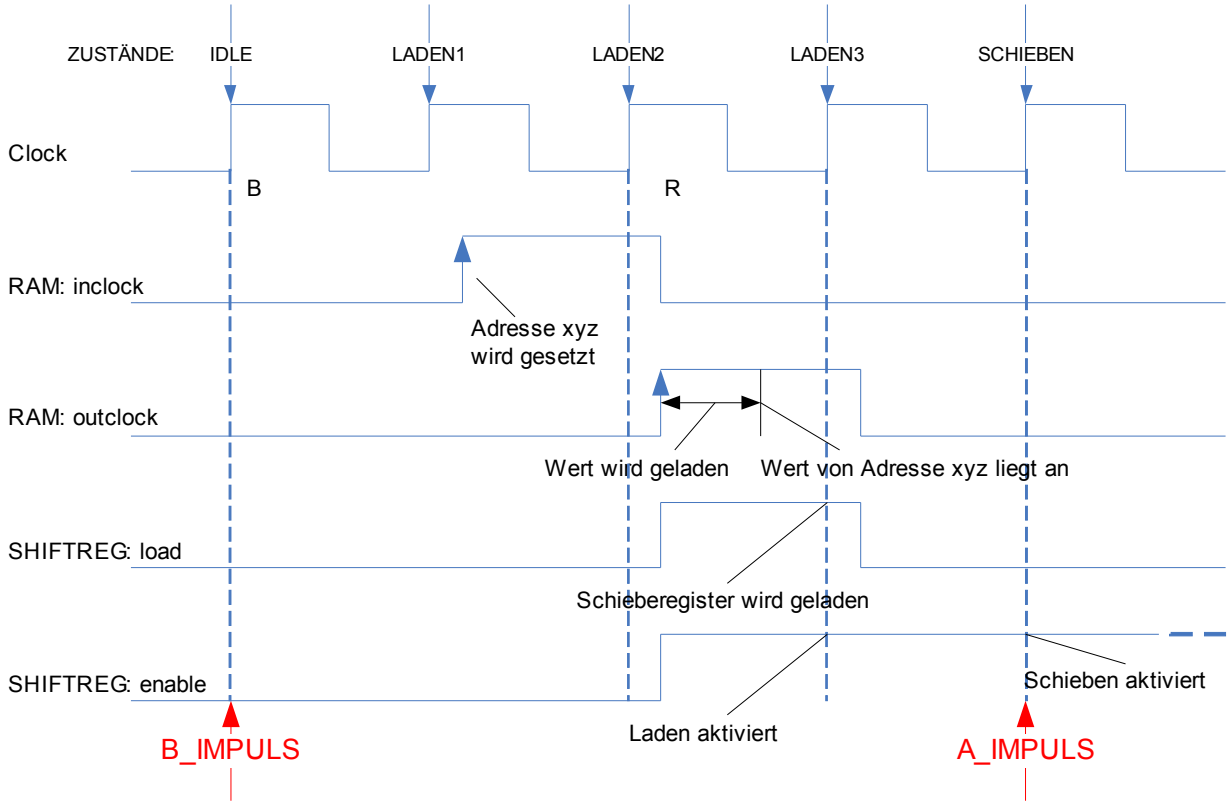
Die Initialisierungsdatei kann in der Datei „RAM.VHD“ über die GENERIC MAP „lpm_ram_dq_component“ unter dem Eintrag lpm_file => "SUBR.MIF" angegeben werden. Hier steht beispielsweise die Datei „SUBR.MIF“, die das auszuführende Beispiel-Programm mit Unterprogrammsprung binär codiert enthält.



7.3 Timing beim Lade- und Speichervorgang

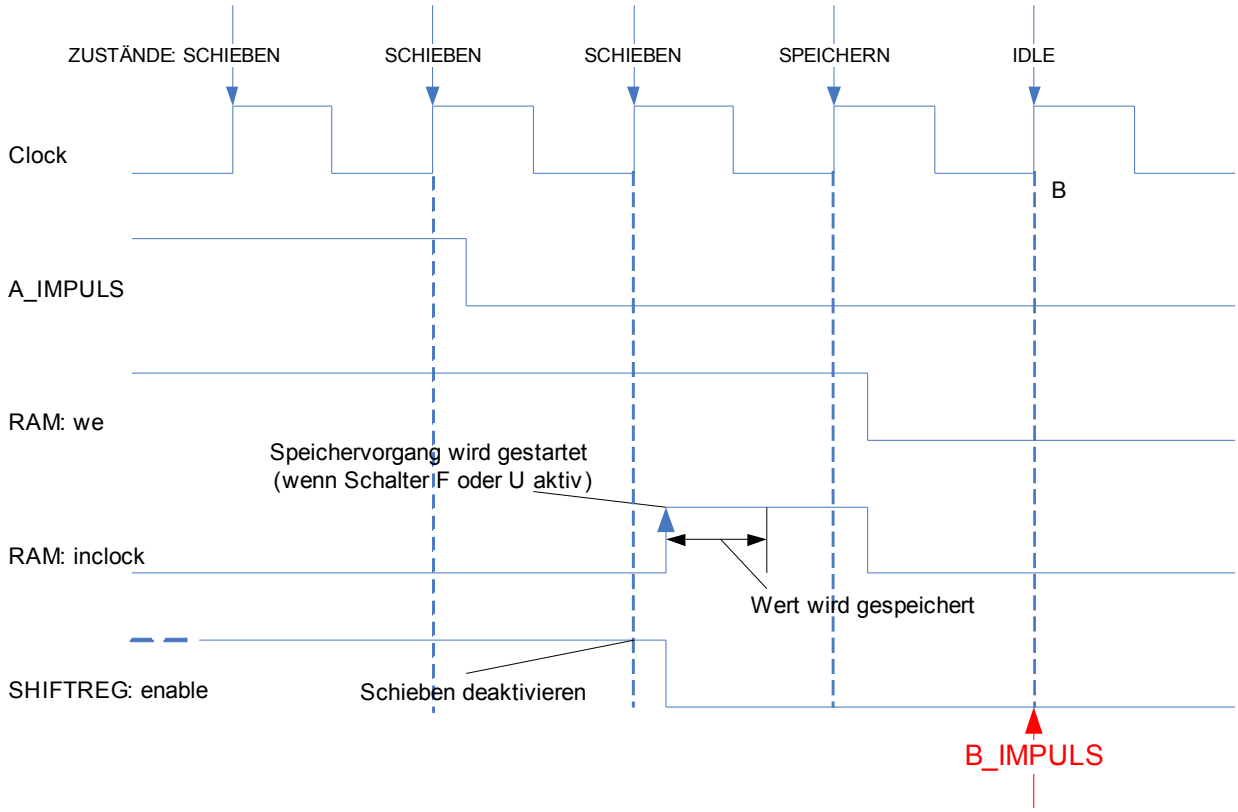
7.3.1 Der Ladevorgang

Das folgende Timing-Diagramm stellt den Signalverlauf zum Laden eines Wortes aus dem RAM-Speicher dar. Mit dem ersten Takt wird, durch den B-Impuls eingeleitet, die Information vom Befehlsregister in das Steuerregister übertragen. Der Zustand der State-Maschine wechselt von „IDLE“ auf „LADEN1“. Der zweite Takt aktiviert das „inclock-Signal“, was die aktuelle Adresse definiert. Anschließend wird mit dem nächsten Taktimpuls das Lesen vorbereitet und das Schieberegister zum Laden initialisiert („load“ und „enable“ auf high). Im Zustand LADEN3 ist somit sichergestellt, dass das gewünschte Wort aus dem RAM-Speicher auf dem Datenbus liegt. Das Wort wird in das Schieberegister übernommen. Mit dem fünften Takt ist der Zustand „SCHIEBEN“ erreicht, der A-Impuls liegt nun für die folgenden 20 Impulse des Schiebevorgangs für den Worttransport an.



7.3.2 Der Speichervorgang

Beinhaltet ein Befehl eines der Bits F oder U, so soll ein Wort in den RAM-Speicher abgelegt werden. Dazu wird wie beim Ladevorgang mit dem B-Impuls zunächst der Speicher adressiert (die Adresse mit „inclock“ definiert!), das alte Wort aus dem Speicher gelesen und während des Worttransportes bitweise hinausgeschoben. Gleichzeitig wird jedoch das Wort vom Befehlszähler (F=1) oder aus dem Akkumulator (U=1) in das Schieberegister des Speichers geladen. Das folgende Diagramm zeigt, wie anschließend, nach Ende des A-Impulses, die Steuersignale zum Speichern im RAM gesetzt werden müssen. Dieses Verhalten wird in der State-Maschine durchgeführt, die einzelnen Zustände sind oben angegeben:



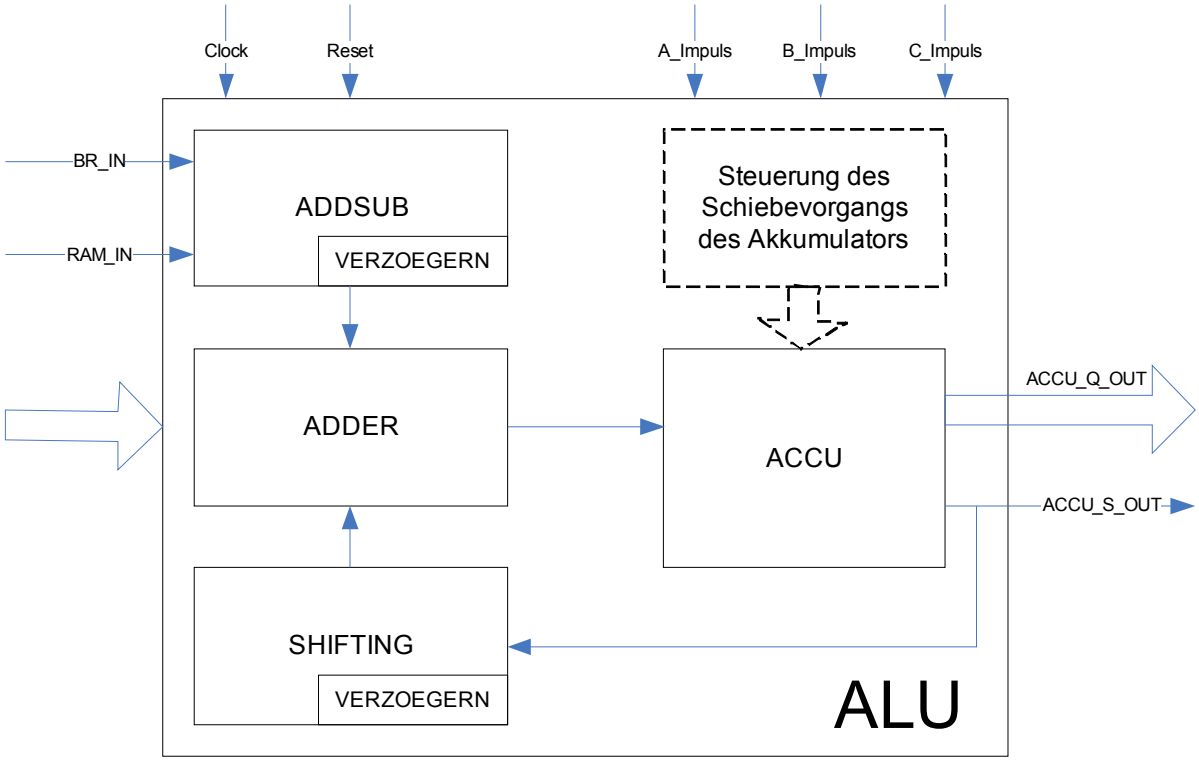
Der gezeigt zweite Takt entspricht dem letzten aktivem A-Impuls, also das Ende des Worttransports. Da die Änderungen des Schiebevorgangs immer erst verzögert auftreten, kann das Schieberegister ohne Probleme einen Takt länger aktiviert sein („SHIFTREG: enable“ wird erst mit dem letzten SCHIEBEN-Zustands-Takt auf low gesetzt).
Der eigentliche Speichervorgang im RAM wird mit der positiven Flanke des „inclock-Signals“ eingeleitet (im letzten Schiebezustand beim Wechsel zum Speichern-Zustand). Das Write-Enable (we) Signal ist während der gesamten Dauer des A-Impulses aktiv, auch wenn kein Speicherbefehl vorliegt (Schalter U,F nicht gesetzt) und wird erst nach dem letzten Schiebevorgang zurückgesetzt. Dies ist möglich, weil das Speichern nur erfolgt, wenn bei aktivem „we“ die positive Flanke von „inclock“ auftritt. Nachdem das Speichern erfolgt ist, wird das „inclock“- sowie „we“-Signal im nächsten Taktzyklus wieder auf Low gesetzt. Alle Zustände der State-Maschine sind nun erfolgreich durchlaufen und der neue B-Impuls veranlasst die Ausführung des nächsten Befehls, wieder im IDLE-Zustand beginnend.

8. Das Rechenwerk: „E_ALU“

8.1 Aufbau und Funktionsweise des Rechenwerks

Das Rechenwerk (ALU) wird in der VHDL-Datei **„ALU.VHD“** zusammengefasst. Dazu gehören die vier Komponenten „ADDSUB“, „ADDER“, „SHIFTING“ und „ACCU“, die alle über eine Port Map eingebunden werden. Die Komponenten „ADDSUB“ und „SHIFTING“ binden ihrerseits eine weiteres Modul „VERZOEGERN“ ein, welches in der Datei **„VERZOEGERN.VHD“** zu finden ist. Zur Steuerung des Schiebevorgangs im Akkumulator, ist neben dem A-Impuls eine weiter Instanz nötig, da aufgrund des Verzögerungselements vor der Bit-Addition ein zusätzlicher Schiebetakt erfolgen muss. Dies wird durch einen „verlängerten“ A-Impuls für den Akkumulator erreicht, dessen Signal den Namen „INT_IMPULS“ trägt und im Prozess „P1“ der Architektur „A_ALU“ erzeugt wird.

Das Rechenwerk hat die abgebildete Struktur:

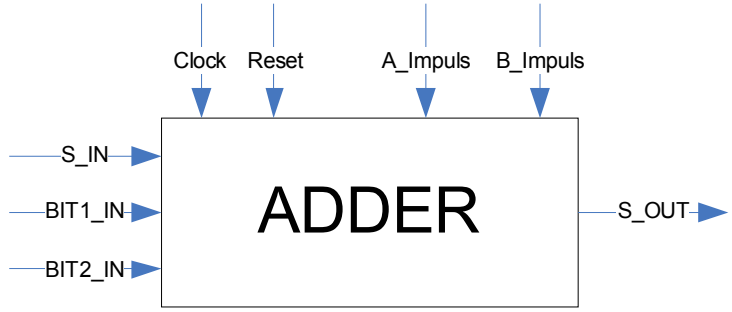


8.2 Der Akkumulator „Accu“

Der Akkumulator wird durch ein Schieberegister gebildet und befindet sich in der Datei **„ACCU.vhd“**. Das Schieberegister wurde mit dem MegaFunction-Wizard erstellt, als Komponente wurde „LPM_SHIFTREG“ mit einer Breite von 20 Bits benutzt. Der Baustein ist auf die Richtung rechtsschiebend eingestellt.

8.3 Das 2-Bit-Addierwerk „E_ADDER“

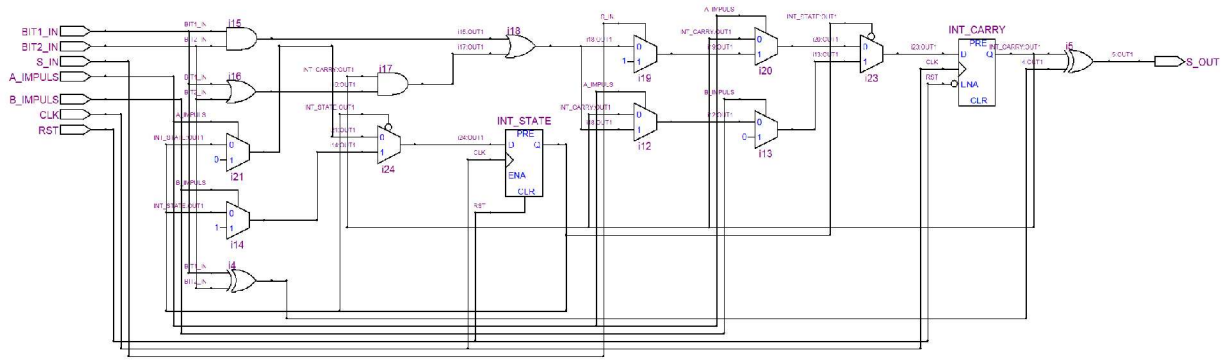
Der „Adder“ wird in der Datei **„ADDER.VHD“** beschrieben und enthält die Routine zur Addition von 2-Bits. Das erste Bit wird aus dem Speicher (bzw. dem BR) (ggf. über eine Komplementsbildung) in das Addierwerk geschoben. Das zweite Bit stammt von der Schiebeeinheit, die mit dem Akkumulator verbunden ist. Soll das Speicherbit nicht addiert, sondern subtrahiert werden, so muss zunächst die 1er Komplementbildung erfolgen. Dies geschieht bereits in der Komponente „ADDSUB“. Der Addierer muss diesen Wert jedoch noch um Eins erhöhen, damit das für die Subtraktion nötige 2er Komplement vorliegt. Der Subtraktionswunsch wird dem Adder über den Schalter S (S_IN) zugeführt. Der Aufbau der Entity sieht wie folgt aus:



Die bitweise Addition wird im VHDL-Code durch eine XOR-Verknüpfung beschrieben:

```
S_OUT <= (BIT1_IN xor BIT2_IN) xor INT_CARRY;
```

Diese Verknüpfung ist in im Synthese-Ergebnis deutlich wieder zu erkennen:



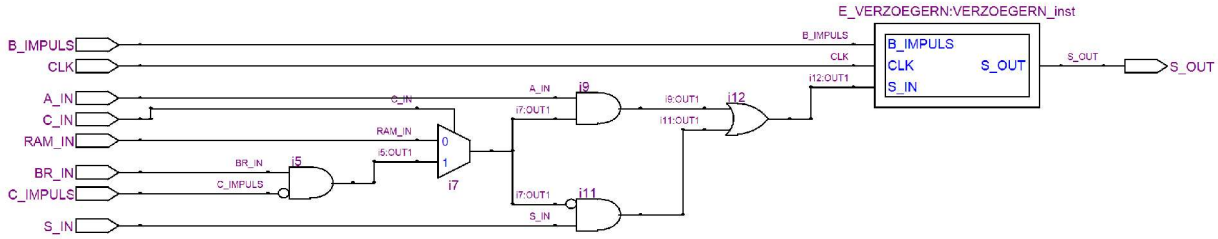
Die Initialisierung des Carry-Signals ist jedoch komplizierter, da erst nach dem B-Impuls und der gültigen Übernahme des Befehls in das Steuerregister feststeht, ob addiert (Schalter S=0) oder subtrahiert (Schalter S=1) werden soll. Der Vorgang wird mit Hilfe einer einfachen State-Maschine im Prozess „P1“ gelöst. Die erste Initialisierung (Zustand IDLE, B_IMPULS=1) erfolgt mit Null. Die anschließend für die Addition/Subtraktion gültige Vorgabe erfolgt im Zustand „SETCARRY“, in Abhängigkeit des Schalters S. Während den weiteren Additionsschritten im Zuge des Wort-Schiebevorgangs wird das Carry-Bit im Zustand „IDLE“ (B_IMPULS=0) entsprechend der AND-Verknüpfung definiert:

```
INT_CARRY <= (BIT1_IN and BIT2_IN) or (INT_CARRY and (BIT1_IN or BIT2_IN) );
```

Auch diese logische Anordnung ist im Syntheseergebnis wieder zu finden.

8.4 Subtraktion durch Komplementbildung „E_ADDSUB“

Diese Komponente hat die Aufgabe, je nach ausgeführtem Befehl das richtige Bit für den Addierer bereitzustellen. Das Bit stammt entweder aus dem Speicher „RAM_IN“ (Schalter C = 0) oder aus dem Befehlsregister „BR_IN“ (Schalter C = 1). Ist das Bit aus dem Befehlsregister gewählt, so ist außerdem zu beachten, dass nur die ersten acht Bits den Adressteil ausmachen, alle weiteren Bits also auf Null fallen müssen. Dies wird mit Hilfe des C_IMPULS bewerkstelligt, der nach den ersten acht Takten des Schiebevorgangs bis zum Ende aktiv wird. Das Syntheseergebnis zeigt sowohl die Schnittstelle dieser Komponente als auch den beschriebenen Zusammenhang:



Über den Multiplexer um C_IN wird zunächst das richtige Eingangsbit ausgewählt. Anschließend gibt es drei Möglichkeiten:

1. Ausgabebit ist gesetzt, wenn Schalter A aktiv und das Eingangsbit gesetzt ist
2. Ausgabebit ist gesetzt, wenn Schalter S aktiv und das Eingangsbit nicht gesetzt ist
3. Ausgabebit ist Null, weil Schalter S und Schalter A nicht gesetzt sind

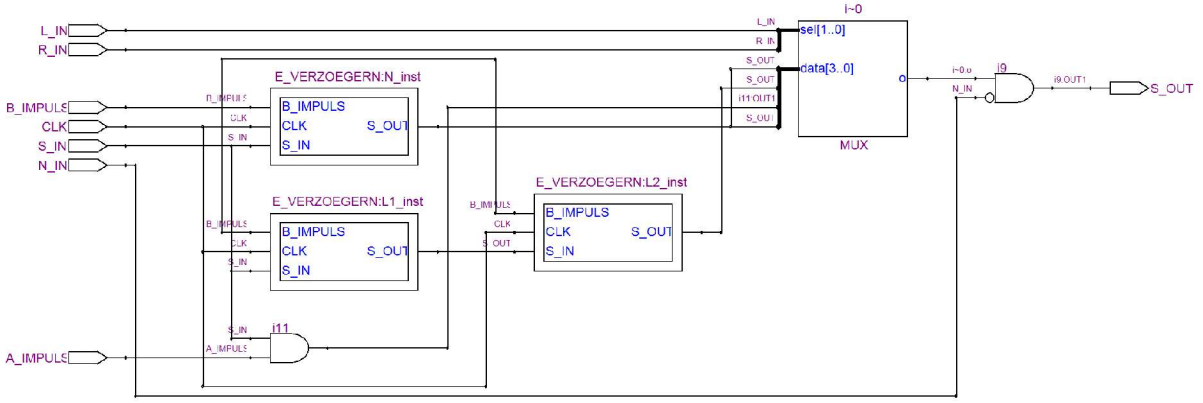
Diese Fälle sind in der VHDL-Beschreibung wie folgt codiert (INT_SUMMAND2 entspricht dem Ausgabebit, INT_SUMMAND1 dem Eingangsbit):

```
INT_SUMMAND2 <= (A_IN and INT_SUMMAND1) or (S_IN and (not INT_SUMMAND1) );
```

Diesen Zusammenhang findet man ebenfalls in der oberen Grafik in Form der Logik-Gatter wieder. Passend zum Schaltungsentwurf der Minima „fließt“ das Ergebnisbit anschließend in ein Verzögerungsglied, so dass der Wert um einen Takt verspätet anliegt.

8.5 Modul für Links- und Rechtsschiebeoperationen „E_SHIFTING“

Das folgende Syntheseergebnis zeigt die Struktur des Schiebewerks:



Die VHDL-Beschreibung zu dieser Komponente befindet sich in der Datei **„SHIFTING.VHD“**. Wie in der Zeichnung auf der vorigen Seite zu erkennen, besteht die Komponente im wesentlichen aus der Verknüpfung von drei Verzögerungsgliedern. Über einen Multiplexer wird der Weg über keinen, einen oder zwei Verzögerungsglieder bestimmt:

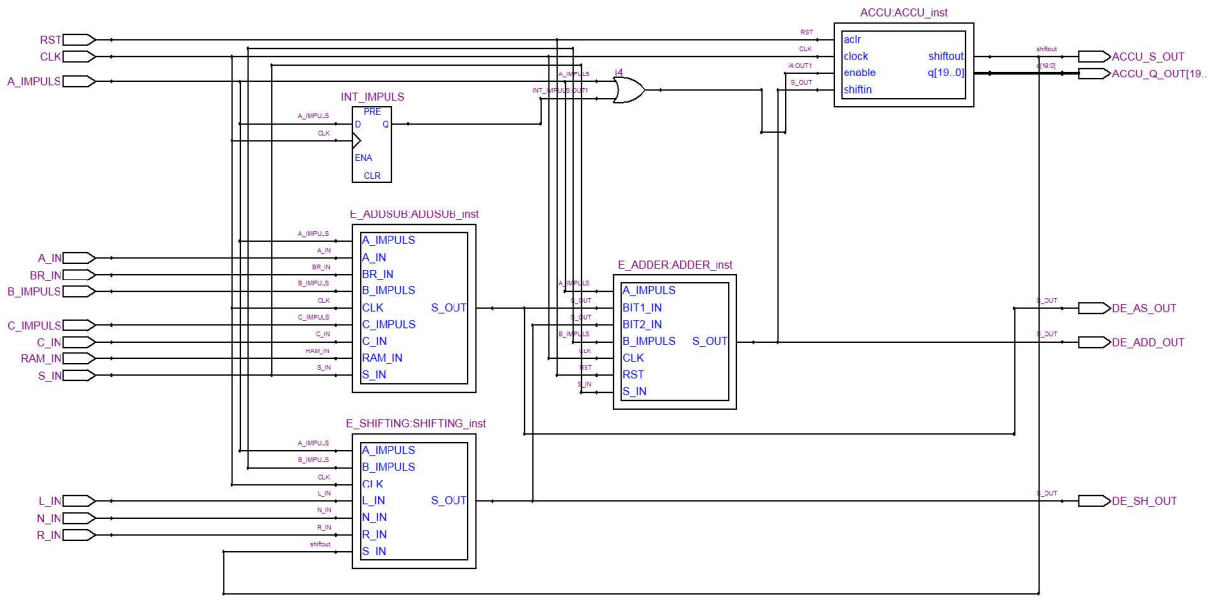
```
INT_TMP(0) <= R_IN;
INT_TMP(1) <= L_IN;

 with INT_TMP select
   S_OUT  <= (not N_IN) and INT_N_OUT           when "00",
             (not N_IN) and INT_N_OUT           when "11",
             (not N_IN) and (S_IN and A_IMPULS) when "01",
             (not N_IN) and INT_L2_OUT          when "10";
```

Die Schalter „L_IN“ und „R_IN“ legen dabei fest, welche Verzweigung aktuell gültig ist. Im Normalfall wird das Eingangssignal „S_IN“ um einen Takt verzögert. Soll nach links geschoben werden, sind zwei Takte nötig. Soll nach rechts geschobene werden, darf keine Verzögerung auftreten. Über den Schalter „N_IN“ ist außerdem ein „clearen“ des Eingangsignals möglich. Nur wenn es nicht gesetzt ist, kann das rechte UND-Gatter Eins als Ergebnis anzeigen.

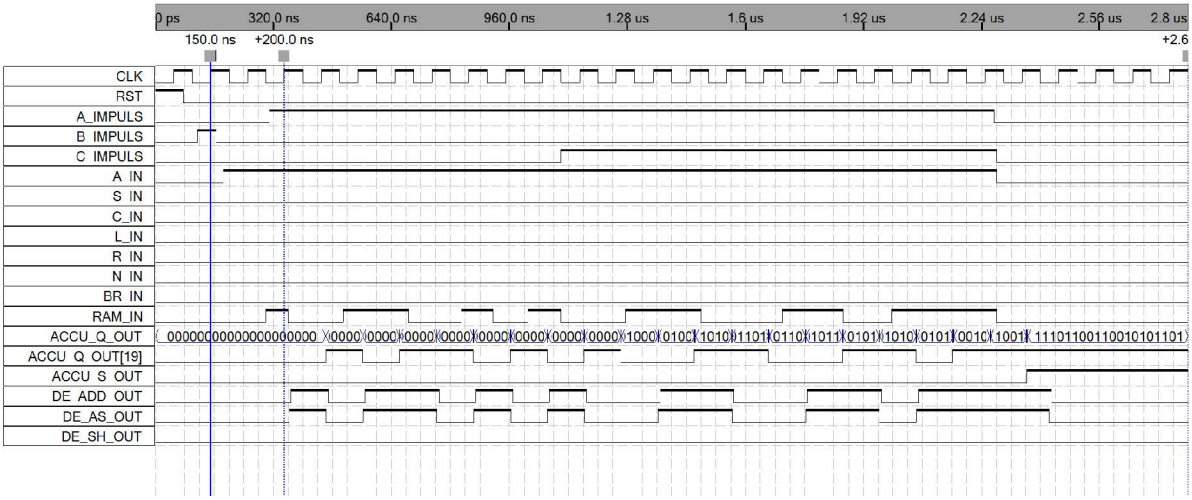
8.6 Zusammenfassung der Komponenten / Syntheseergebnis

Die untenstehende Abbildung zeigt noch einmal die Zusammenfassung aller Bausteine im Rechenwerk nach der Synthese. Der Prozess „P1“ für den Steuerimpuls für das Schieben des Akkumulators in der Hauptarchitektur „A_ALU“ wurde durch ein Flip Flop synthetisiert und wie im VHDL-Code beim Port Mapping angegeben mit dem A_IMPULS „OR-verknüpft“. Die übrige Verdrahtung verläuft ebenfalls wie in der VHDL-Beschreibung angegeben, zusätzliche Logik-Komponenten sind nicht notwendig.



8.7 Simulation und Timing-Verhalten

Dieses einfache Beispiel demonstriert das Timing-Verhalten des Rechenwerks für die Addition im Verlaufe einer simulierten Wortzeit. Der linke, blaue Strich zeigt den B-Impuls, der die Befehlsbearbeitung einleitet. Der rechte, blaue Strich steht am Anfang des A-Impulses, also zu Beginn des Worttransportes. Das Steuerbit A (für Addition) ist für die Ausführungsdauer aktiv und am Eingang „RAM_IN“ werden folgende Bits hineingeschoben: 1011 0101 0011 0011 0111.
Da der Akkumulator vor Beginn der Rechnung Null war, ist das Ergebnis der Addition gleich dem Eingangsstrom: 1110 1100 1100 1010 1101.
Während der Phase des Worttransportes (A_IMPULS=1) ist das verzögerte Schieben im Akkumulator sehr schön anhand des höchstwertigsten Bits (Vorzeichen-Bit: ACCU_Q_OUT [19]) zu erkennen. Das am Speicher angelegte Bit erscheint jeweils erst einen Takt verzögert.



9. Pinnzuweisung auf dem Testboard

Die Zuweisung der FLEX-Pins auf dem Testboard ist nach folgender Belegung vorgenommen:

8255 Chip 0

Port 0	***Port 1***	***Port 2 low***	***Port 2 high***
Flexpin 65 = Akku[19]	Flexpin 7 = Akku[11]	Flexpin 80 = Reset	Flexpin 73 = A_IMPULS
Flexpin 62 = Akku[18]	Flexpin 9 = Akku[10]		Flexpin 78 = B_IMPULS
Flexpin 64 = Akku[17]	Flexpin 11 = Akku[09]		
Flexpin 61 = Akku[16]	Flexpin 10 = Akku[08]		
Flexpin 67 = Akku[15]	Flexpin 8 = Akku[07]		
Flexpin 66 = Akku[14]	Flexpin 6 = Akku[06]		
Flexpin 70 = Akku[13]	Flexpin 3 = Akku[05]		
Flexpin 69 = Akku[12]	Flexpin 81 = Akku[04]		

8255 Chip 1

Port 0	***Port 1***		
Flexpin 23 = Akku[03]	Flexpin 52 = Cnt[4]		
Flexpin 22 = Akku[02]	Flexpin 54 = Cnt[3]		
Flexpin 25 = Akku[01]	Flexpin 59 = Cnt[2]		
Flexpin 24 = Akku[00]	Flexpin 60 = Cnt[1]		
	Flexpin 58 = Cnt[0]		

10. Funktionsbeispiele

10.1 Beispielprogramm mit Unterprogrammsprung

Dieses Beispielprogramm aus der Aufgabenstellung führt einfache Rechenoperationen Addition/Subtraktion/Shifting durch. Dabei wird auf Variablen im Speicher zugegriffen und auch das Ergebnis wird dort abgelegt. Außerdem wird ein Unterprogrammsprung demonstriert.
Soll das Programm ausgeführt bzw. simuliert werden, so muss die RAM-Initialisierungsdatei „SUBR.mif“ verwendet werden. Der entsprechende Eintrag ist in der Generic Map in „RAM.VHD“ vorzunehmen.

Name:	**Adresse:**	**Befehl/ Wert:**	**Beschreibung:**	**Bin:** EFPQNRLCASUH$$$$$$$$$	**Hex:**
X	$10	5	Speicherlatz für Variable X		5
Y	$11	3	Speicherlatz für Variable Y		3
Z	$12	15	Speicherlatz für Variable Z		F
H	$13	0	Speicherlatz für Ergebnis		0
	...			EFPQNRLCASUH$$$$$$$$	
START	$20	AN $10	Lade X in Akku	000010001000	8810
	$21	A $11	Addiere Y: <A>:=X+Y	000000001000	811
	$22	S $12	Subtr. Z: <A>:=X+Y-Z	000000000100	412
	$23	U $13	Speicher Ergebnis in Zelle H	00000000001000010011	213
	$24	EQF $30	Teste Vorzeichen: If <A> < 0 then CALL $30 Rückkehradd. In $05	11010000000000110000	D0030
	$25	ACL $19	Schiebe <A> links und addiere Konstante 25	00000011100000011001	3819
	$26	U $13	Speicher Erg. in Zelle H	00000000001000010011	213
	$27	H	Halt	00000000000100000000	100
	...			EFPQNRLCASUH$$$$$$$$	
SUBR	$30	E $31	Subroutineeinsprung	10000000000000110001	80031
	$31	NS $13	Wechsle Vorzeichen von <H>	00001000010000010011	8413
	$32	E $05	Return	10000000000000000101	80005

10.2 Beispielprogramm zum Generieren von Fibonacci-Zahlen

Das zweite Beispielprogramm demonstriert das Generieren von Fibonacci-Zahlen. Es arbeitet nach folgendem Prinzip:

Benutze Speicherzellen: x, y, sum

Init: x=0, y=1, sum=0

1) sum=x+y
2) x=y
3) y=sum

4) sum=x+y
5) x=y
6) y=sum
.
.
.

In Assembler:

```
-- sum = x + y
$1: load x
$2: add y
$3: store sum
-- x = y
$4: load x
$5: store y
-- y = sum
$6: load sum
$7: store y
-- schleifenbedingung
$8: jmp if acc>0 $1
$9: halt
```

Soll das Programm ausgeführt bzw. simuliert werden, so muss die RAM-Initialisierungsdatei „FIBO.mif“ verwendet werden. Der entsprechende Eintrag ist in der Generic Map in „RAM.VHD“ vorzunehmen.

Die Beschreibung der Befehle für die Minima-Nachbildung lautet:

Name:	**Adresse:**	**Befehl/ Wert:**	**Beschreibung:**	**Bin:** EFPQNRLCASUH$$$$$$$$$	**Hex:**
X	$10	0	Speicherlatz für Zahl 1		0
Y	$11	1	Speicherlatz für Zahl 2		1
ERG	$12	0	Speicherlatz für Ergebnis		0
	...			EFPQNRLCASUH$$$$$$$$	
START	$20	E $21	Springe zu $21	10000000000000100001	80021
	$21	AN $10	Lade X in Akku	00001000100000010000	8810
	$22	A $11	Addiere Y: <A>:=X+Y	00000000100000010001	811
	$23	U $12	Speicher Erg. in Zelle ERG	00000000001000010010	212
	$24	AN $11	Lade Y in Akku	00001000100000010001	8811
	$25	U $10	Speicher Akku in X	00000000001000010000	210
	$26	AN $12	Lade Ergebnis in Akku	00001000100000010010	8812
	$27	U $11	Speicher Akku in Y	00000000001000010001	211
	$28	EP $20	Teste Vorzeichen: Wenn Akku positiv, springe zu $20	10100000000000100000	A0020
SUBR	$29	H	Halt	00000000000100000000	100

11. Bewertung des Ergebnisses

In der folgenden Tabelle sind einige Informationen zum VHDL-Design der Minima-Nachbildung angegeben. Die Angaben beziehen sich auf die Nutzung des Tools Altera Quartus II unter Verwendung des Bausteins FLEX10K10LC84-4:

	Minima-Nachbildung:
Wortzeit:	26 Verarbeitungstakte - Takt 0-4: Initialisierung / Ladevorgang - Takt 5-24: Worttransport - Takt 25: Speichervorgang
Max. Frequenz:	37,31 MHz
Min. Clock Period:	26,800 ns
Logic Cells:	178
Memory Bits:	5120
Baustein-Auslastung	30 %

12. Anhang

12.1 CD-ROM mit Quelltext und Dokumentation (PDF)