Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Rahmen der Laborübung soll ein Taschenrechner, der die vier Grundrechenarten beherrscht, auf einem FPGA Board implementiert werden. Die Eingabe erfolgt mit Hilfe eines gewöhnlichen PS/2 Keypads. Die Ausgabe soll auf einem VGA-Monitor erfolgen.

Kapitel 2

Requirements

Req1: Der Taschenrechner soll die vier Grundrechenarten (Operatoren)

- Addition, +
- Subtraktion, -
- Multiplikation, *
- Division, /

beherrschen.

Req2: Bei der Berechung haben Multiplikation und Division Vorrang gegenüber Addition und Subtraktion.

Req3 Eine Zahl setzt sich ausschließlich aus den Ziffern '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' zusammen.

Req4: Als Operanden sind positive und negative Ganzzahlen erlaubt. Die Breite beträgt 32Bit. Mögliche Zahlen liegen daher im Wertebereich von -2^{31} bis $+2^{31}-1$.

Req5: Negative Operanden werden durch ein Minus-Zeichen vor der Zahl eingegeben. Fehlt das Vorzeichen, wird die Zahl als positiv interpretiert. Das heißt die Eingabe von positiven Zahlen erfolgt ohne Vorzeichen.

Req6: Der Aufbau einer Rechung sieht wie folgt aus:
Operand1 Operator Operand2 [**Operator Operand*N***]

Erklärung: Der Ausdruck von **Operand1** bis **Operand2** stellt die kleinstmögliche, gültige Rechnung da. Wenn man mehr als zwei Operanden auf einmal eingeben möchte, dann kann man die Minimalversion mehrmals um den Ausdruck in den eckigen Klammern erweitern.

Req7: In einer Zeile am Monitor können genau 70 Zeichen dargestellt werden. Die Eingabe eines Rechenausdruckes kann daher nicht mehr als 70 Zeichen betragen. Alles was länger als 70 Zeichen ist wird abgeschnitten (d.h. jeglicher Tastendruck, außer Backspace und Enter, wird ignoriet). Dadurch kann es zu ungültigen Ausdrücken kommen.

Req8: Die Auswertung eines Rechenausdrucks erfolgt folgendermaßen:
Es wird der Ausdruck von links nach rechts bis zum ersten Auftreten eines Mal oder Dividiert Operators durchsucht. Wenn ein solcher Operator gefunden wurde, wird das Ergebnis aus den Operanden die unmittelbar links und rechts vom Operator stehen berechnet, und anstelle dieses Ausdrucks eingefügt. Die Suche wird solange fortgesetzt, bis das Ende des Ausdrucks erreicht wurde. Danach befinden sich nur mehr Additionen und Subtraktionen im Ausdruck. Diese werden

dann anschließend ausgeführt.
Bevor die Auswertung startet, muss natürlich sichergestellt werden, das es sich um einen gültigen Rechenausdruck handelt.

Req9: Die Eingabe des Rechenausdrucks erfolgt über ein PS/2 Keyboard.

- Das Drücken der Backspace-Taste bewirkt das das letzte Zeichen gelöscht wird.
- Das Drücken der Enter-Taste startet die Berechnung.
- Weiters stehen noch die Ziffern von Null bis Neun sowie die Operatorsymbole zur Eingabe zur Verfügung.

Req10: Das Rechenergebnis eines gültigen Ausdrucks wird in der nächsten Zeile angezeigt. Der Curser positioniert sich aber gleich wieder in der Zeile darunter, um eine neue Rechnung entgegenzunehmen.

Req11: Die letzten 50 Rechenausdrücke (gültige wie auch ungültige) und deren Ergebnis werden in einem Zwischenspeicher abgelegt. Der Inhalt des Buffers wird über die RS232-Schnittstelle (8N1, 115200 baud) zum Computer übertragen (dabei wird der Inhalt aber nur gelesen, es findet keine Modifikation statt). Die Übertragung startet wenn

- der Knopf (welcher Knopf das genau sein wird haben wir noch nicht festgelegt) auf dem Development-Board gedrückt wird.
- auf Request vom Computer.

Req12: Der Inhalt des Buffers wird folgendermaßen auf dem Bildschirm dargestellt (Ausgabe beginnt in neuer Zeile):

Buffer Content: `<newline>`
Rechenausdruck 1 `<newline>`
Ergebnis von Rechenausdruck 1 `<newline>`
Rechenausdruck 2 `<newline>`
Ergebnis von Rechenausdruck 2 `<newline>`
...
Buffer End `<newline>` Bei ungültigen Rechenausdrücke wird anstatt des Ergebnisses die Fehlermeldung, aufgrund derer die Berechnung abgebrochen wurde, angezeigt.

Req13: Wenn ein ungültiger Rechenausdruck eingegeben wird, dann wird anstatt des Rechenergebnisses die Fehlermeldung *Invalid Input* ausgegeben.
Wann ist ein Audruck ungültig? - Wenn er nicht dem Aufbau von **Req1** bis **Req6** entspricht. Eine leere Zeile wird nicht als ungültig gewertet.

Req14 Das Ergebnis aller Berechnungen ist wieder eine 32Bit Zahl. Sollte diese Länge nicht ausreichen (z.B. kommt es bei einem Zwischenergebnis zu einem Überlauf), wird die Berechnung mit der Fehlermeldung *Overflow* beendet.

Req15 Alle Fehlermeldungen werden, gleich wie bei Rechenergebnissen, in der nächsten Zeile angezeigt, und der Cursor springt dann gleich wieder zur darauffolgenden.

Req16 Wird ein Backspace gedrückt, obwohl davor noch keine Eingabe erfolgte, wird dieser ignoriet.

Req17 Ungültige Rechenoperationen wie zum Beispiel Division durch 0, werden mit der Fehlermeldung *Error* quitiert.

Kapitel 3

High Level Design Description

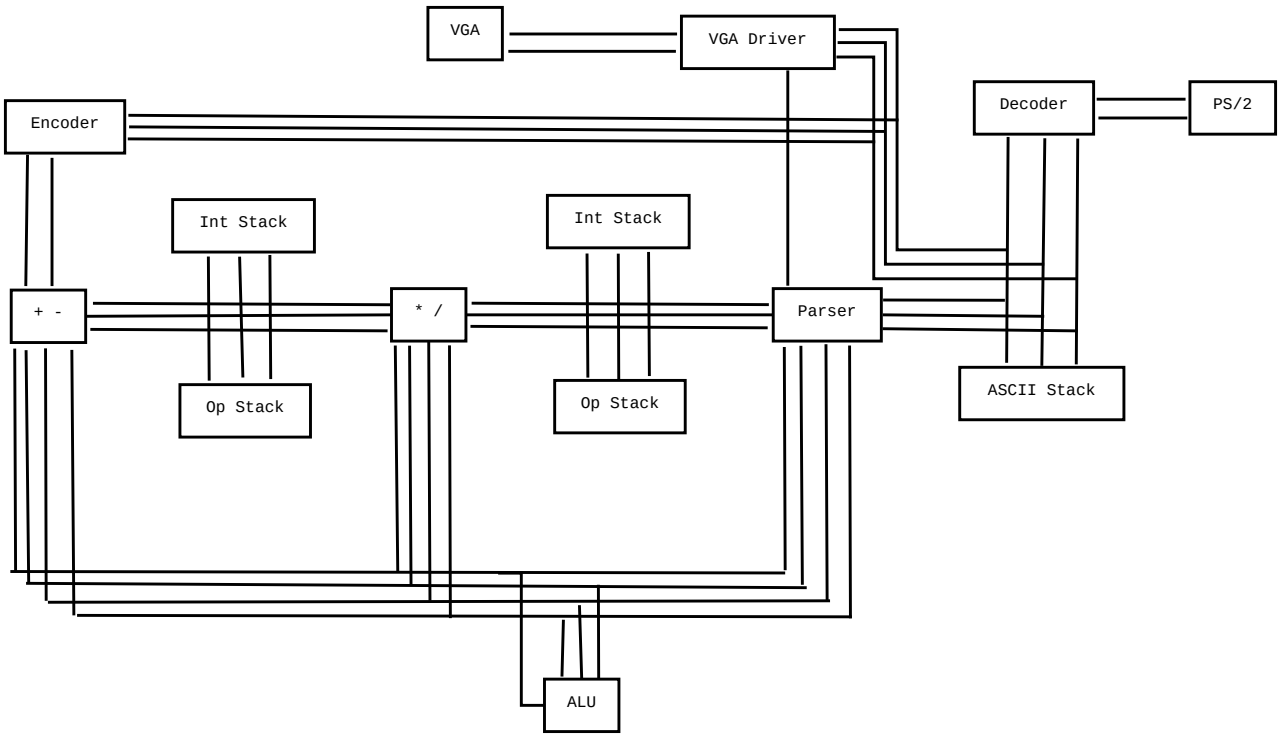

Abbildung 3.1: Übersicht über die Module die unser Design bilden. Diese Grafik spiegelt aber in keinster Weise die Implementierung wieder (z.B. da kein Tristate möglich). Sie soll lediglich den Prinzipaufbau vermitteln.

Abbildung 3.1 gibt einen Überblick über den Aufbau des Taschenrechners.

VGA Übernimmt die Darstellung von Zeichen auf dem Monitor.

VGA-Driver Nimmt von verschiedenen Komponenten Bits entgegen, und stellt sie auf dem Monitor dar. Die Darstellung erfolgt mithilfe des VGA Moduls.

PS/2 Übernimmt das Einlesen der Scancodes vom Keyboard.

Decoder Diese Einheit nimmt den Scancode vom PS/2-Modul entgegen wandelt sie in einen ASCII-Code um und legt sie in einem Stack (in der Zeichung dargestellt durch ASCII-Stack) ab.

Parser Der Parser stellt einerseits sicher, das es sich um einen validen Input handelt, und andererseits wird der Inputstring aufgeteilt auf einen Operatorstack und einen Operandenstack (wobei hier dann schon 32-Bit Integers gepusht werden).

***/ (MulDiv)** Diese Einheit berechnet die Ergebnisse aller Multiplikationen und Divisionen die im Inputstring vorkommen und legt die Ergebnisse wieder auf den Stack.

+- (SubAdd) Diese Einheit führt die verbleibenden Additionen und Subtraktionen durch und berechnet somit das Endergebnis.

Encoder Der Encoder wandelt das 32-Bit-Ergebnis wieder in eine Bitfolge um, die vom VGA-Driver interpretiert werden kann.

ALU ALU ist in diesem Fall eigentlich zu weit gegriffen, es führt praktisch die Division aus.

RS232 Übernimmt die Kommunikation mit dem Computer (nicht in Grafik).

Ringbuffer Speichert die letzten 50 Berechnungen (nicht in Grafik).

3.1 Externe Schnittstellen

3.2 Schnittstellen der Module

3.2.1 Stack (Tabelle 3.1)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
data_in	in	N	Bits, die auf den Stack gepusht werden sollen
enable	in	1	Starte mit der Ausführung
select	in	1	Zur Auswahl ob gepusht oder gepopt werden soll
data_out	out	N	Bits die vom Stack gelesen wurden
empty	out	1	Zeigt an, ob sich noch Daten auf dem Stack befinden

Tabelle 3.1: Signal description des **Stacks** (N = Bitbreite, D = Tiefe)

3.2.2 Decoder (Tabelle 3.2)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
ps2_data_in	in	8	Scancodes von der Tastatur
enable	in	1	Sperrt den Decoder damit keine neuen Daten während der Berechnung eingelesen werden können
new_data	in	1	Signalisiert, dass es neue Daten vom PS/2-Modul gibt
stack_data_out	out	8	Bits, die in den Stack geschrieben werden
stack_enable	out	1	Aktiviert den Stack
stack_select	out	1	Zur Auswahl der Operation
error	out	2	Error Code

Tabelle 3.2: Signal description des **Decoder**

3.2.3 Parser (Tabelle 3.3)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
data_in	in	8	Zeichen vom Stack
enable	in	1	Starte mit der Ausführung
operand_stack_out	out	32	Bits eines Integer-Operanden, die auf den Operanden-Stack gelegt werden sollen
operator_stack_out	out	2	Rechenoperator, der auf den Operator-Stack gelegt werden soll
data_stack_select	out	1	Zur Auswahl der Operation
data_stack_enable	out	1	
operand_stack_select	out	1	
operand_stack_enable	out	1	
operator_stack_select	out	1	
operator_stack_enable	out	1	
busy	out	1	Zeigt an, ob der Parser noch beschäftigt ist
error	out	2	Error Code

Tabelle 3.3: Signal description des **Parser**

3.2.4 MulDiv (Tabelle 3.4)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
enable	in	1	Starte mit der Ausführung
quotient	in	32	
div_busy	in	1	Zeigt an, ob das Ergebnis des Dividierers bereits fertig ist
operand_in_in	in	32	
operand_in_select	out	1	
operand_in_enable	out	1	
operator_in_in	in	2	
operator_in_select	out	1	
operator_in_enable	out	1	
operand_out_out	out	32	
operand_out_select	out	1	
operand_out_enable	out	1	
operator_out_out	out	1	
operator_out_select	out	1	
operator_out_enable	out	1	
divisor	out	32	
dividend	out	32	
busy	out	1	Zeigt an, ob der MulDiv noch beschäftigt ist
error	out	2	Error Code
error_in	in	1	Error Code, falls ein Fehler beim Divider aufgetreten ist

Tabelle 3.4: Signal description des **MulDiv**

3.2.5 Divider (Tabelle 3.5)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
enable	in	1	Starte mit der Ausführung
divisor	in	32	
dividend	in	32	
quotient	out	32	
remainder	out	32	
busy	out	1	
error	out	1	Error Code, wird direkt an den MulDiv geleitet

Tabelle 3.5: Signal description des **Divider**

3.2.6 SubAdd (Tabelle 3.6)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
enable	in	1	Starte mit der Ausführung
operand_stack_in	in	32	
operand_stack_select	out	1	
operand_stack_enable	out	1	
operator_stack_in	in	1	
operator_stack_select	out	1	
operator_stack_enable	out	1	
result	out	32	
busy	out	1	
error	out	2	Error Code

Tabelle 3.6: Signal description des **SubAdd**

3.2.7 Encoder (Tabelle 3.7)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
result	in	32	
enable	in	1	Starte mit der Ausführung
quotient	in	32	
remainder	in	32	
div_busy	in	1	
divisor	out	32	
dividend	out	32	
new_data	out	1	
data_out	out	8	
busy	out	1	

Tabelle 3.7: Signal description des **Encoder**

3.2.8 VGA-Driver (Tabelle 3.8)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
data_in	in	8	
enable	in	1	
new_data	in	1	
busy	out	1	
Schnittstellen zum VGA-IP-Core	out		

Tabelle 3.8: Signal description des **VGA-Driver**

3.2.9 Ringbuffer (Tabelle 3.9)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
data_in	in	8	
write_enable	in	1	
data_out	out	8	
next	in	1	
data_read	out	1	
data_ready	out	1	

Tabelle 3.9: Signal description des **Ringbuffer**

3.2.10 Button Driver (Tabelle 3.10)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	out	1	System reset signal
button_pin	in	1	
reset_pin	in	1	
button	out	1	

Tabelle 3.10: Signal description des **Button Drivers**

3.2.11 RS232 (Tabelle 3.11)

Signal name	*Direction*	*Signal width*	*Functionality*
sys_clk	in	1	System clock signal
sys_res	in	1	System reset signal
data_in	in	8	
data_read	in	1	
transfer_finish	in	1	Signalisiert, dass vom Buffer alle Daten gelesen wurden
RX	in	1	
btn	in	1	Button vom FPGA, initialisiert den Buffertransfer
next	out	1	Leitung zum Buffer, um nchstes Byte anzufordern
TX	out	1	
enable	in	1	
next	out	1	

Tabelle 3.11: Signal description des **RS232**

Kapitel 4

Detailed Design Description

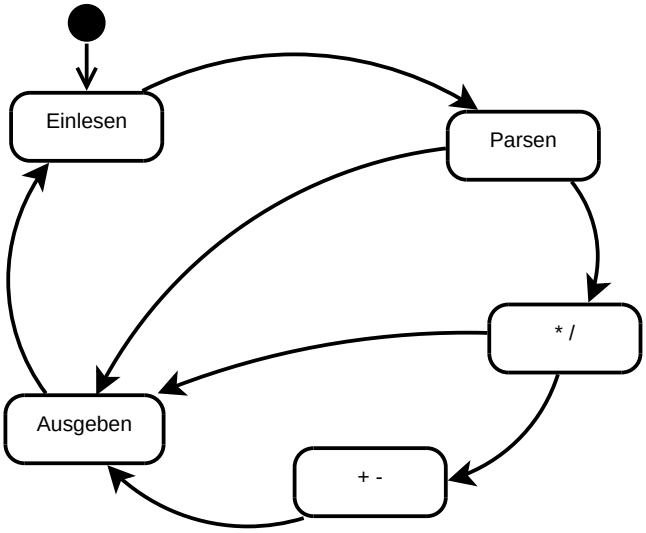

Abbildung 4.1: Grafik zeigt den Ablauf vom Eingeben einer Rechnung bis zum fertigen Ergebnis. Es wird zunächst der Input eingelesen. Nachdem der User das Auswerten des Ausdrucks anfordert, wird der Input auf Gültigkeit geprüft. Sollte es dabei zu Fehlern kommen, wird die Ausgabe einer entsprechenden Fehlermeldung erwirkt. Wenn der Ausdruck valide ist, werden zunächst alle Multiplikationen und Divisionen durchgeführt. Sollte es hier zu Fehlern kommen (z.B. Division durch 0), wird wieder eine Fehlermeldung generiert. Anschließend werden die verbleibenden Additionen und Subtraktionen ausgeführt. Wiederrum, sollte es zu Fehlern kommen (z.B. Overflow), wird eine Fehlermeldung erzeugt.

In nachfolgenden Diagrammen werden die Schritte, die in Abbildung 4.1 grob umrissen dargestellt wurden, näher beleuchtet.

4.1 Einlesen

Abbildung 4.2 veranschaulicht den Einlesevorgang. Es wird zunächst ein neuer Opcode vom PS/2-Modul entgegengenommen. Dieser wird in den entsprechenden ASCII-Code umgewandelt und dann einerseits in einem Stack abgelegt, und andererseits zum VGA-Driver weitergeleitet, damit der User auch seine Eingabe am Monitor sieht. Dieser Vorgang läuft solange, bis der User die Enter-Taste gedrückt hat (wenn natürlich vorher die 70 Zeichen Grenze erreicht wurde, wird jedes weitere Zeichen verworfen).

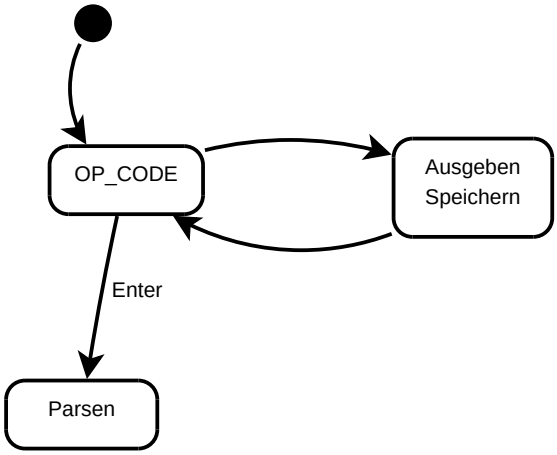

Abbildung 4.2: Einlesevorgang

4.2 Parsen

Abbildung 4.3 beschreibt den weiteren Ablauf, nachdem der User die Auswertung des eingegebenen Rechenaudrucks angefordert hat. Das Ganze lässt sich am besten anhand eines Beispiels erklären. Nehmen wir an, der User hat den String 1+2*3-5*6 eingegeben. Dann sieht sich der Parser mit folgendem Stack konfrontiert (Abbildung 4.4).

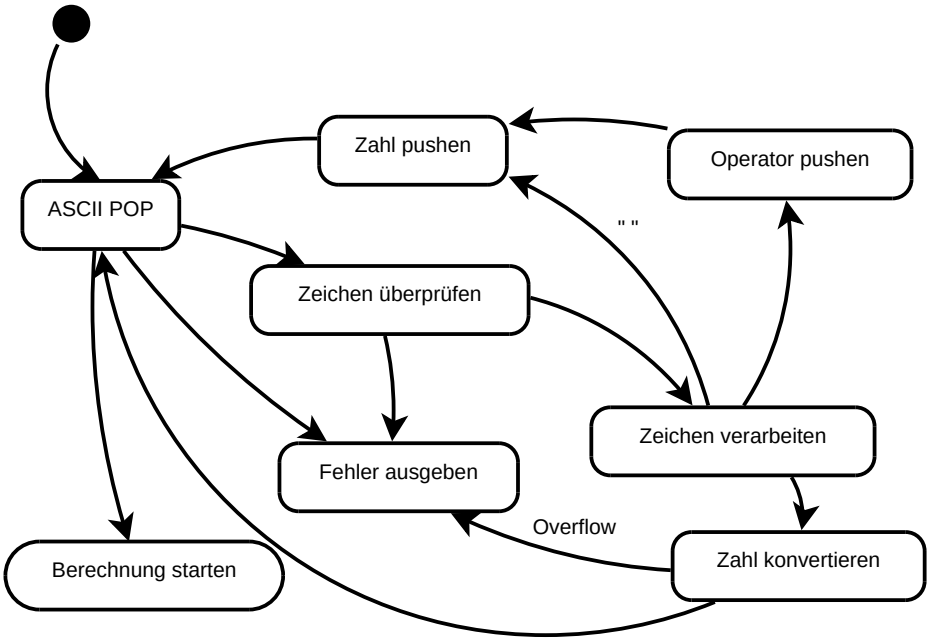

Abbildung 4.3: Vorgang beim Parsen des Inputs

Der Ablauf ist nun folgendermaßen. Es wird das ersten Zeichen gepopt (6er). Es wird überprüft ob das Zeichen das vom Stack gelesen wurde, auch gültig ist. Anschließend wird das Zeichen weiterverarbeitet, da es sich ja noch um ASCII Codes handelt. D.h. Ziffern müssen in 32-Bit Zahlen konvertiert werden (wie das Konvertieren genau abläuft, wird weiter unten beschrieben. Zunächst nur soviel; das Konvertieren einer Zahl kann sich über meherere Iterationen hinziehen, das Zwischenergebnis wird dabei in einer temporären 32-Bit Variable gehalten. Es ist immer eine Abwechslung von Multiplikation und Addition.). So keine Fehler aufgetreten sind, wird das nächste Zeichen vom ASCII-Stack geholt. In unserem Beispiel

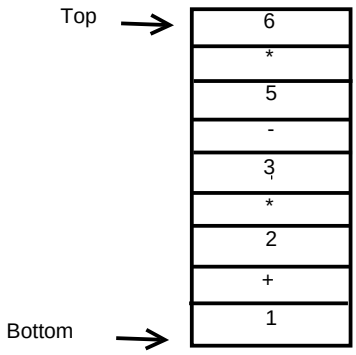

Abbildung 4.4: Stackinhalt

wäre das das *. Es wird wieder überprüft, ob das Zeichen gültig ist. Nachdem erkannt wurde, das es sich jetzt um einen Operator handelt, wird der Wert aus der temporären Variable auf einen seperaten Operanden-Stack gepusht. Da weiterhin keine Fehler aufgetreten sind, wird das nächste Zeichen geholt. Es ist der 5er. Da es sich um eine Ziffer handelt, wird die Umwandlung in einen 32-Bit Zahl eingeleitet. Das nächste Zeichen das vom Stack geholt wird ist wieder interessant, da es sich um ein Minus handelt. Dieses kann ja entweder als Vorzeichen, oder als Rechenzeichen fungieren. Da man das ja a priori nicht wissen kann, wird dieses Minus sofort dazu verwendet, die Zahl, die sich in der temporären Variable befindet, mit einem negativen Vorzeichen zu versehen. Sollte nun, das Zeichen nach dem Minus eine Ziffer sein (so wie es bei unserem Beispiel ist, Ziffer 3), wird als Operator ein + gepusht. Im Falle das wieder ein Minus kommt, wird eben das Minus auf den Stack gelegt.
Die restliche Abarbeitung erfolgt analog. Das Endergebnis sieht in jedem Fall wie in Abbildung 4.5 aus.

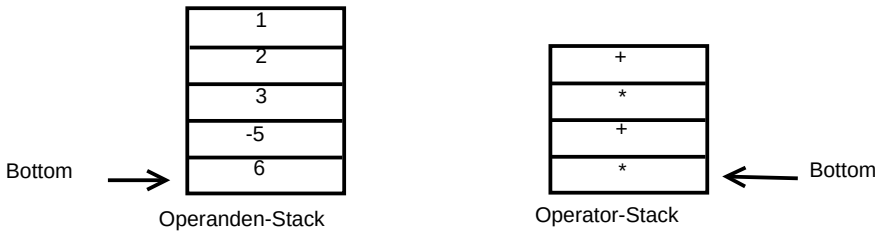

Abbildung 4.5: Endergebnis von Parser

4.3 MulDiv

Aufgrund der Tatsache, das Multiplikation und Division die beiden Operatoren mit der höchsten Priorität sind (zumindest von denen die unser Taschenrechner kann), werden diese als erstes behandelt.
Das heißt, es wird zunächst einmal vom Operanden-Stack die erste 32-Bit Zahl geholt. Anschließend wird der dazugehörige Rechenoperator vom Operator-Stack gepopt. Wenn wir uns wieder auf unser obiges Beispiel beziehen, wäre das die Zahl eins und der Operator +. Nun wird untersucht, handelt es sich bei dem Operator um ein Plus oder Minus, dann wird die Zahl, genauso wie der Operator, auf einen neuen Operanden- bzw. Operator-Stack gepusht. Wenn es aber ein Mal oder Dividiert ist, wird der zweite Operand gepopt, das Ergebnis berechnet. Dieses Ergebnis wird aber noch nicht auf den neuen Operanden-Stack gepusht. Denn, wenn nun wieder ein Mal oder Dividiert Operator käme, müsste die Zahl ja vorher wieder gepopt werden. Aus diesem Grund wird sie zwischengespeichert. Das Ergebnis wird erst dann gepusht, wenn der neue Operator ein Plus oder Minus ist. Weiters ist zu beachten, das wenn sich kein weiterer Operator mehr auf dem Operator-Stack befindet, die Ausführung beendet ist,

und ebenfalls das Zwischenergebnis gepusht werden kann (in der Zeichnung angedeutet durch EOF).

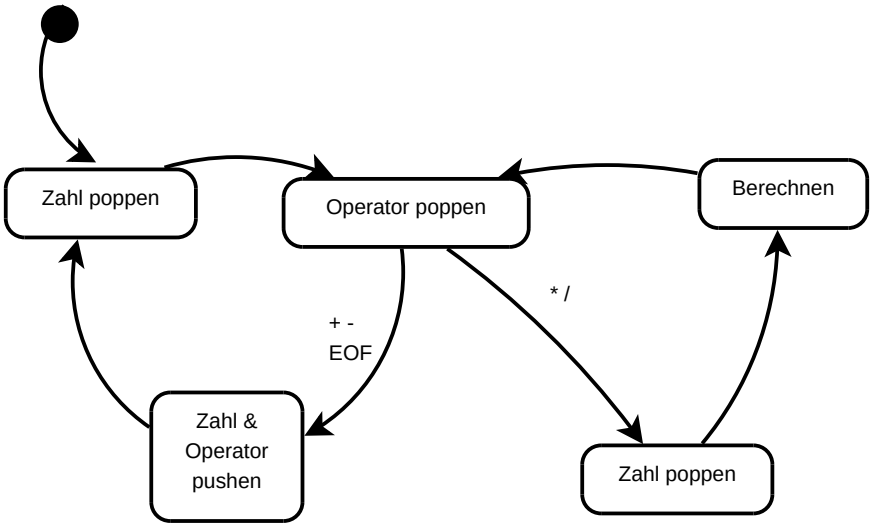

Abbildung 4.6: Multiplizier/Dividier-Modul

Das Endresultat für das obige Bespiel sind zwei neue Stacks die wie in Abbidlung 4.7 aussehen.

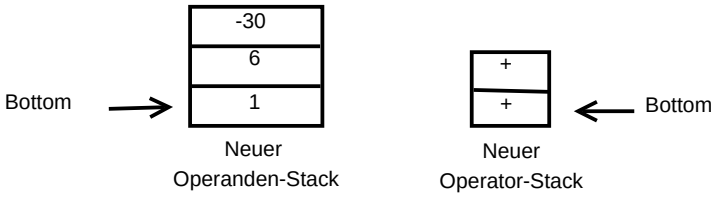

Abbildung 4.7: Ergebnis nach MulDiv-Modul

4.4 SubAdd

Dieses Modul führt nun die letzte Stufe der Berechnung aus. Das heißt, dieses Modul liefert das Endergebnis. Wie in Abbildung 4.8 zu sehen ist, wird ein Operand gepopt, dann ein Operator, das Ergebnis berechnet, zwischengespeichert, dann der neue Operator und Operand gepopt, ... Solange bis am Schluss eben nur mehr eine Zahl übrig bleibt.

4.5 Encoder

Die Aufgabe des Encoders (Abbildung 4.9) ist es nun, das Ergebnis, das vom SubAdd Modul geliefert wird, zurück in einen ASCII- String zu verwandeln, der am VGA angezeigt werden kann. Dazu wird der 32-Bit Integer fortlaufend durch zehn dividiert. Der Rest beinhaltet jeweils die letzte Ziffer, die in einen Character verwandelt und auf einen Stack gepusht wird (dadurch wird die Reihenfolge beim herauslesen wieder richtig). Mit dem Quotienten beginnt dann das Spiel wieder von vorne, solange er nicht selbst bereits kleiner als zehn ist.
Im Falle von negativen Zahlen, muss zuerst die Zahl positiv gemacht werden, und dann am Schluss das Vorzeichen ebenfalls am Stack gelegt werden.

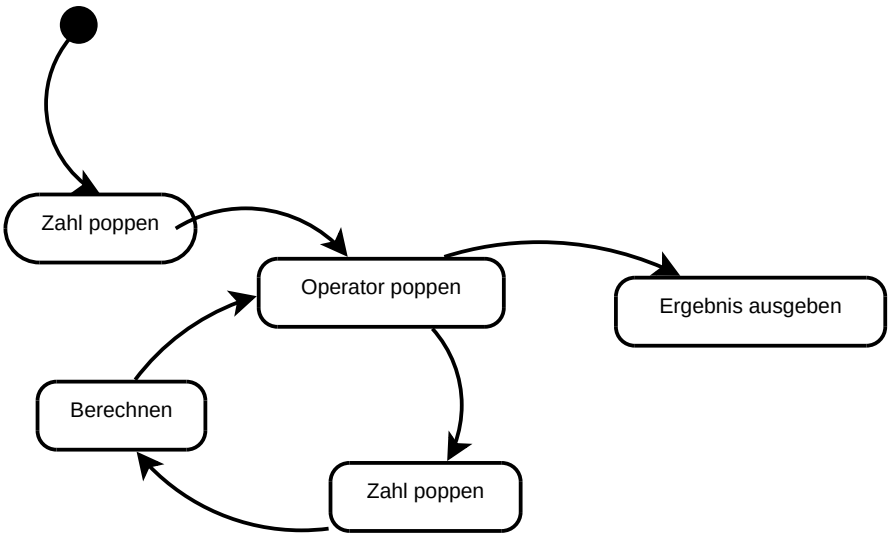

Abbildung 4.8: Addition/Subtraktions-Modul

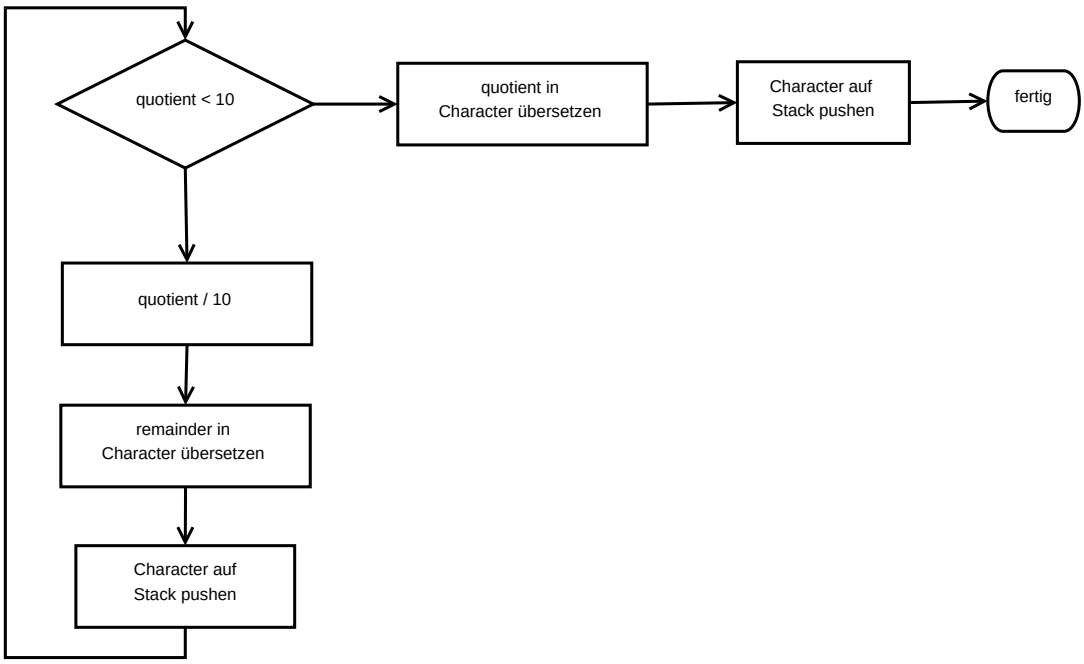

Abbildung 4.9: Encoder Prinzipaufbau

4.6 Dividierer

Bei allen Registern, die für die Implementierung des Dividierers notwendig sind (Rest, Divisor, Dividend,...) handelt es sich um 32 Bit Register. Der Ablauf der Division wird in Abbildung 4.10 gezeigt.

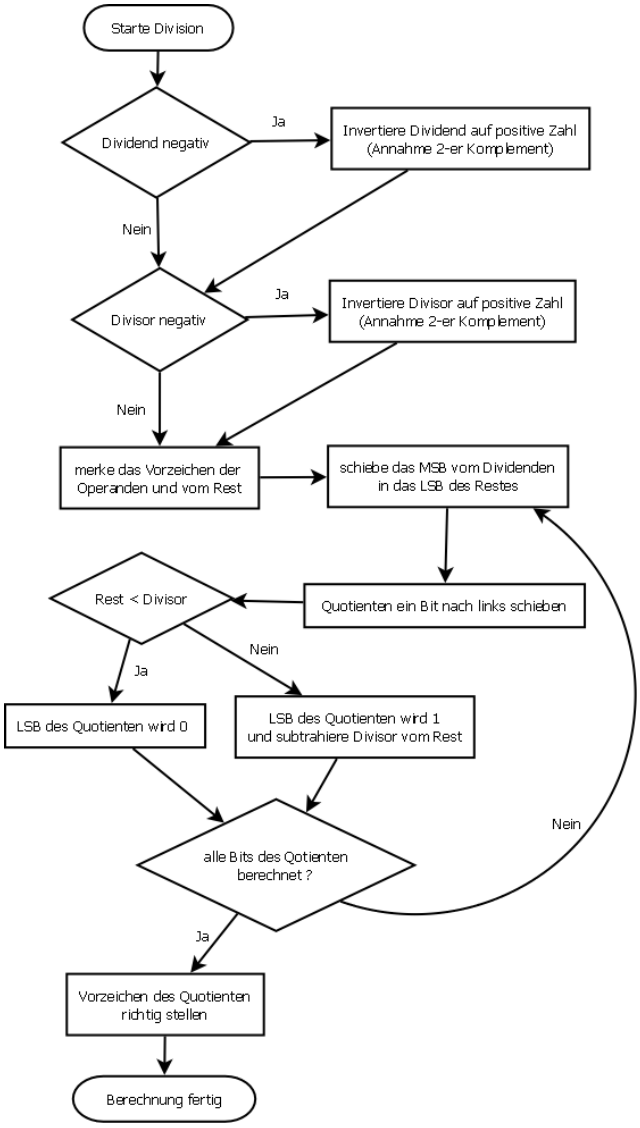

Abbildung 4.10: Dividierer Funktionsablauf

4.7 Umwandlen von ASCII to Integer

Das Umwandeln von ASCII-Wert zu Integer wir mithilfe einer Funktion gemacht welche in Listing 4.1 zu sehen ist. Hier wurde zur besseren Lesbarkeit der Typ Integer verwendet, in der Implementierung wird dann std_logic_vector für die Berechnung verwendet. Sobald man die einzelnen Ziffern hat kann man die Zahl zusammensetzen. Als Beispiel: Man hat die Zahlen 1 2 3 dann wird die Berechnung 3 + (10 *2) durchgeführt, das Ergebnis in ein 32-Tempregister geladen und danach Tempregister + (100 *1) berechnet, was 123 ergibt. Der Ergebnis-Wert 10 der to_int-Funktion kennzeichnet eine falsche Eingabe.

Listing 4.1: Beispielcode

```
function to_int(int: integer) return integer is
    variable c: integer;
  begin
       case int is
         when  48 => c := 0;
         when  49 => c := 1;
         when  50 => c := 2;
         when  51 => c := 3;
         when  52 => c := 4;
         when  53 => c := 5;
         when  54 => c := 6;
         when  55 => c := 7;
         when  56 => c := 8;
         when  57 => c := 9;
         when others => c := 10;
       end case;
    return c;
end to_int;
```

4.8 RS232 und Ringbuffer

Damit die Funktionalität des oben angeführten Ablaufs so wenig wie möglich von der Funktion des Zwischenspeicherns der Rechnungen betroffen ist, haben wir uns folgendes überlegt. Eigentlich muss ja der UART nur aus dem Buffer lesen und das gelesene an den Computer übertragen. Und der restliche Teil (genauer der Decoder wenn er die Zeichen aus dem ASCII-Stack holt und der Encoder wenn er das Ergebnis an den VGA-Driver schickt) muss nur in den Ringbuffer schreiben. D.h. dafür bietet sich ein Dualport-RAM an.
Der Ringbuffer selbst besteht intern aus zwei solchen RAMs. Im ersten wird der Inputstring gespeichert, im zweiten das Ergebnis. Bei dem der die Inputstrings speichert, gibt es ein zusätzliches Bit für jeden String, das mir darüber Auskunft gibt, ob für diesen String bereits ein zugehöriges Ergebnis vorliegt und somit zur Ausgabe herangezogen werden kann oder nicht.
Der Button vom FPGA-Board wird mit dem RS232 Modul verbunden, weil, egal ob der Request vom PC kommt oder der Button betätigt wird, das Resultat das selbe sein soll, nämlich das Versenden des Bufferinhalts.