Spezifikation
Taschenrechner

Hardware Modeling SS 2010

Gruppe 11

Nick Mayerhofer
0726179

Lukas Petermann
0725146

Inhaltsverzeichnis

1 Hardware Modellierung LU

1 Einleitung

Dieses Dokument beschreibt die Spezifikationen eines einfachen Rechners der die vier Grundrechnungsarten unterstützt.
Eingaben werden über die Tastatur gemacht. Erlaubt sind die Zahlen '0'-'9' die vier Operationszeichen auf dem Ziffernblock und die Leer-, Backspace- und Entertaste vom normalen Block.
Die Ausgabe erfolgt über das Display. Jede Rechnung darf bis zu 70 Zeichen lang sein und das Ergebnis wird nach drücken der Entertaste in der nächsten Zeile ausgegeben. Sollte man bereits in der letzten Zeile befinden werden die restlichen Zeilen um eine Zeile nach oben verschoben.
Der Taschenrechner speichert die letzten 50 Rechnungen und kann diese auf Anfrage über das RS232 Interface auf einen anderen PC schicken. Implementiert ist auch eine Fehlerbehandlung um keine falschen Ergebnisse zu liefern. Abgedeckt ist die Division durch Null, Overflows von Zahlen und Ergebnissen und die ungültige Positionierung von Operanden.

2 Requirements

Req 1: Eingaben werden über die Tastatur gemacht und Zeilenweise am Bildschirm ausgegeben. Beim drücken der Enter Taste wird das Ergebnis in die nächste Zeile geschrieben. Sollte keine Zeile mehr frei sein werden die restlichen Zeilen nach oben verschoben und der letzte Eintrag am Bildschirm gelöscht.

Req 2: Die vier Grundrechnungsarten (+,-,*,/) müssen unterstützt werden.

Req 3: Eine gültige Zahl liegt zwischen $2^31 - 1$ und 2^31.

Req 4: Bei Divisionen wird, falls nötig, abgerundet um auf eine ganze Zahl zu kommen.

Req 5: Leerzeichen müssen eingegeben und beim Berechnen ignoriert werden.

Req 6: Die Backspace Taste löscht das letzte Zeichen und setzt den Curser zurück.

Req 7: Sollte kein Zeichen in der Rechnung stehen wird die Backspace Taste ignoriert.

Req 8: Multiplikation und Division wird vor Addition und Subtraktion berechnet. Sollten mehrere Punktrechnungen nacheinander ausgerechnet werden, werden diese nach der Reihenfolge ihrer Eingabe berechnet.

Req 9: Sollte die Rechnung bereits 70 Zeichen haben wird kein neues Zeichen akzeptiert.

Req 10: Der Fehler Division durch Null wird erkannt und die Fehlernachricht "Division durch Null" statt dem Ergebnis ausgegeben.

Req 11: Wenn zwei Zahlen, mit einem Leerzeichen getrennt, direkt nebeneinander stehen muss beim Berechnen ein Fehler erkannt und "ungültige Syntax" ausgegeben werden.

Req 12: Sollten zwischen zwei Zahlen zwei Operanden stehen, und der zweite Operand ist ein Minus, dann wird die zweite Zahl negativ behandelt und das richtige Ergebnis berechnet.

Req 13: Sollten zwischen zwei Zahlen zwei Operanden stehen, und der zweite Operand ist kein Minus, dann muss ein Fehler erkannt werden und "ungültige Syntax" ausgegeben werden.

Req 14: Sollte das erste Zeichen ein Minus-Operand sein wird die erste Zahl negativ behandelt und das richtige Ergebnis ausgegeben.

Req 15: Sollte das erste Zeichen kein Minus Operand sein muss der Fehler erkannt und "ungültige Syntax" ausgegeben werden.

Req 16: Eine Zahl die außerhalb des Wertebereichs ist wird erkannt und beim Berechnen die Fehlernachricht "Overflow" ausgegeben.

Req 17: Wenn das Ergebnis mehrerer Zahlen in irgendeinem Rechenschritt außerhalb des Wertebereichs ($2^31 - 1bis2^31$) liegt wird die Fehlernachricht "Overflow" ausgegeben.

Req 18: Bei Anforderung über die serielle Schnittstelle oder beim Drücken des Buttons auf dem Entwicklerboard werden die letzten 50 Berechnungen mit Ergebnissen über RS232 an den PC gesendet.

3 Testcase specification

TC1 **Eingabe:** 40+33 *2/5-7
Ausgabe:
deckt: Req1, Req2, Req4, Req5, Req8

TC2 **Eingabe:** 2147483648
Ausgabe: Overflow
deckt: Req2,

TC3 **Eingabe:** 45, dann 3*Backspace
Ausgabe: muss leer sein, kein Fehler durch löschen obwohl kein Zeichen da ist.
deckt: Req6, Req7

TC4 **Eingabe:** Eingabe von 71 Zeichen, Dann Backspace
Ausgabe: 71 Zeichen wird ignoriert und danach wird das 70Zeichen gelöscht und Zeiger nach hinten gesetzt
deckt: Req9

TC5 **Eingabe:** 10/0
Ausgabe: Division durch Null
deckt: Req10

TC6 **Eingabe:** 23 55
Ausgabe: ungültige Syntax
deckt: Req11

TC7 **Eingabe:** 20/-2
Ausgabe: -10
deckt: Req2, Req12

TC8 **Eingabe:** 20/*2
Ausgabe: ungültige Syntax
deckt: Req13

TC9 **Eingabe:** -15+5
Ausgabe: -10
deckt: Req14

TC10 **Eingabe:** *6/3
Ausgabe: ungültige Syntax
deckt: Req15

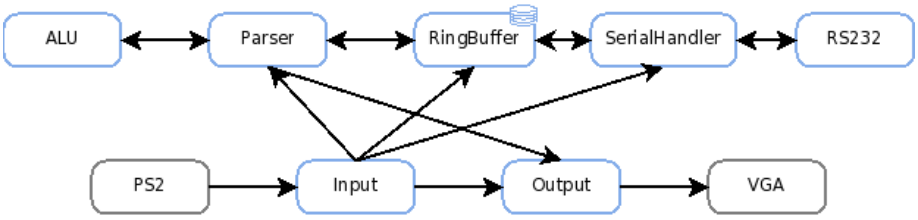
TC11 **Eingabe:** 2000000000*3
Ausgabe: Overflow
deckt: Req16

TC12 **Eingabe:** drücken des Buttons auf Entwicklerboard
Ausgabe: Pc empfängt die letzten 50 Rechnungen
deckt: Req18

4 High level design description

Wir haben den Rechner so entworfen, dass wir die einzelnen Aufgaben möglichst einfach in eigene Module kapseln können. Dies erleichtert die Implementierung, Wartung und spätere Erweiterung der Module. Wie diese miteinander kommunizieren wird in folgender Abbildung gezeigt.

Input: Wartet auf Scancodes vom PS2 Modul, bearbeitet diese und schickt, je nach Scancodes, ein ASCII Zeichen oder einen Befehle an eine andere Komponente weiter.



Output: Bei jedem neuen Zeichen oder nach dem Lösen einer Rechnung muss der Bildschirm aktualisiert werden. Der Output ist die Schnittstelle zwischen den anzuzeigenden Zeichen und der VGA Komponente, die den Bildschirm aktualisiert.

Parser: Der Parser wartet auf ein Signal, welches durch das Drücken der Enter Taste ausgelöst wird. Danach holt er sich vom Ringbuffer Modul die letzte Rechnung, zerlegt diese in seine Grundrechnungen und schickt jede dieser Rechnungen an die ALU. Nachdem die Berechnung gelöst wurde, wird das Ergebnis an den Ringbuffer und den Output geschickt.

ALU: Die "Arithmetic Logical Unit" löst die Grundrechnungsarten. Der Parser sendet zwei Operanden und den Operator an die ALU und bekommt das Ergebnis zurückgeschickt.

Ringbuffer: Dieses Modul speichert die letzten 50 Rechnungen samt Ergebnissen in einer Ringbuffer Struktur.

SerialHandler: Auf ein Signal vom RS232 oder vom Input Modul holt sich der SerialHandler den gesamten Speicher vom Ringbuffer und sendet ihn an die RS232 Komponente.

RS232: Dieses Modul hört auf dem RS232 Interface auf den Befehl alle Rechnungen zu schicken und leitet diesen Befehl an den SerialHandler weiter. Alle Daten die vom SerialHandler kommen werden über das Interface geschickt.

4.1 Externe Schnittstellen

Zwei Schnittstellen bekommen wir fertig compiliert und müssen nicht von uns implementiert werden.

PS2: Das PS2 Modul ist die Schnittstelle zwischen der Tastatur und dem Programm. Jeder Tastendruck sendet 1-3 Scancodes an unser Input Modul und muss daraus den richtige Befehl interpretieren.

VGA: Die VGA Komponente erlaubt einfache Kontrolle über den Bildschirm. Mit mehreren Befehlen kann der Bildschirm verändert werden.

4.2 Reset and Clock

Der Reset wurde low active gewählt.

Der VGA Controller wird über eine PLL auf 25.175 MHz getaktet. Alle weiteren Controller benützen die externe Clockfrequenz des Developer Boards von 33.33 MHz.

4.3 Logical Interfaces

In den folgenden Tabellen 1.1 bis 1.7 stehen zu jeder Komponente die dazu gehörigen Signale mit Richtung, Bitanzahl und einer kurzen Beschreibung der Signale.

Tabelle 1.1: **Input Modul**

Signal	Richtung	Bits	Beschreibung
ps2_new_data	in	1	Ist High wenn ein neuer Scancode gelesen werden kann.
ps2_data	in	8	Auf diesem Signal liegt der letzte Scancode
inp_new_data	out	1	Ist High wenn ein neuer gültiger ASCII Code eingegeben wurde.
inp_data	out	8	Der neue gültige ASCII Code.
inp_del	out	1	Beim Drücken der Backspace Taste für einen Zyklus auf High.
inp_sendRS232	out	1	Wird beim Button auf dem Entwicklerboard ausgelöst.
pars_start	out	1	Beim Drücken der Enter Taste wird der Parser gestartet.

4.4 Behavioural Interface

Ausgabe am Bildschirm

Der Bildschirm kann 80 Zeichen in einer Zeile darstellen und unsere Rechnungen können 70 Zeichen lang sein. Mit einem '=' Zeichen würden uns 9 Zeichen für das Ergebnis bleiben. Unser Ergebnis ist jedoch 32 Bit lang und liegt zwischen [2147483648,-2147483647].
Deswegen und auch wegen der Übersicht schreiben wir das Ergebnis in die nächste Zeile. Der Bildschirmhintergrund ist schwarz mit weißer Schrift. Die Rechnung beginnt in der ersten Zeile. Sollten keine Zeilen mehr frei sein rutschen alle Rechnungen um eine Zeile nach oben und die Älteste wird gelöscht.

Tabelle 1.2: **Output Modul**

Signal	Richtung	Bits	Beschreibung
vga_command	out	8	Befehl an das VGA Modul.
vga_command_data	out	32	Daten für den Befehl an die VGA.
vga_free	in	1	Signal von der VGA. Erlaubt neue Befehle.
inp_new_data	in	1	Ist High wenn ein neuer gültiger ASCII Code eingegeben wurde.
inp_data	in	8	Der neue gültige ASCII Code.
inp_del	in	1	Beim Drücken der Backspace Taste für einen Zyklus auf High.
pars_new_data	in	1	Vom Parser kann ein neuer ASCII Code gelesenw erden.
pars_data	in	32	Der neue ASCII Code.

Tabelle 1.3: **Parser Modul**

Signal	Richtung	Bits	Beschreibung
ps_start	in	1	Startet Berechnung.
calc_data	out	32	Erster Operand.
calc_data2	out	32	Zweiter Operand.
calc_operator	out	2	Operator für Berechnung.
calc_start	out	1	Startet Berechnung.
calc_finished	in	1	Berechnung fertig.
calc_result	in	32	Ergebnis.
calc_status	in	2	Status der Berechnung. Bei 0 fehlerfrei, sonst fehlerhaft.
pars_new_data	out	1	Vom Parser kann ein neuer ASCII Code gelesenw erden.
pars_data	out	8	Der neue ASCII Code.
rb_busy	in	1	Wenn der Buffer beschäftigt ist dürfen keine neuen Eingaben kommen.
rb_read_en	out	1	Eine neue Zeile wird angefordert.
rb_read_lineNr	out	6	Die neue Zeile die gelesen werden soll.
rb_read_data_rdy	in	1	Die neue Zeile kann gelesen werden.
rb_read_data	in	648	Die neue Zeile.

Tabelle 1.4: **ALU Modul**

Signal	Richtung	Bits	Beschreibung
calc_data	in	32	Erster Operand.
calc_data2	in	32	Zweiter Operand.
calc_operator	in	2	Operator für Berechnung.
calc_start	in	1	Startet Berechnung.
calc_finished	out	1	Berechnung fertig.
calc_result	out	32	Ergebnis.
calc_status	out	2	Status der Berechnung. Bei 0 fehlerfrei, sonst fehlerhaft.

Tabelle 1.5: **Ringbuffer Modul**

Signal	Richtung	Bits	Beschreibung
rb_busy	out	1	Wenn der Buffer beschäftigt ist dürfen keine neuen Eingaben kommen.
pars_new_data	in	1	Neue Daten von Parser.
pars_data	in	8	Der neue ASCII Code vom Parser.
inp_new_data	in	1	Neue Daten vom Input
inp_data	in	8	Der neue gültige ASCII Code vom Input.
inp_del	in	1	Ist kurz High wenn ein Zeichen gelöscht werdne soll.
rb_char_newline	in	1	Springt in die nächste Zeile.
rb_read_en	in	1	Eine neue Zeile wird angefordert.
rb_read_lineNr	in	6	Die neue Zeile die gelesen werden soll.
rb_read_data_rdy	out	1	Die neue Zeile kann gelesen werden.
rb_read_data	out	648	Die neue Zeile.

Ausgabe über RS232

Wird der Button am Developer board, oder eine bestimmte Taste am PC, gedrückt, wird der gesamte Verlauf der letzten 50 Rechnungen mit den Ergebnissen an den PC geschickt.

Umgang mit Overflows

Overflows werden von unserem ALU Modul abgefangen. Tritt ein Overflow auf wird die Berechnung beendet und eine Fehlernachricht in die History gespeichert.

Tabelle 1.6: **SerialHandler Modul**

Signal	Richtung	Bits	Beschreibung
inp_sendRS232	in	1	Initialisiert das Senden des gesamten Speichers.
rb_busy	in	1	Wenn der Buffer beschäftigt ist dürfen keine neuen Eingaben kommen.
rb_read_en	out	1	Eine neue Zeile wird angefordert.
rb_read_lineNr	out	6	Die neue Zeile die gelesen werden soll.
rb_read_data_rdy	in	1	Die neue Zeile kann gelesen werden.
rb_read_data	in	648	Die neue Zeile.
tx_rdy	in	1	Zum Senden muss rdy low sein.
tx_go	out	1	Startet Sendevorgang.
tx_data	out	8	Das zu sendende Byte.
rx_recv	in	1	Neues Byte wurde empfangen.
rx_data	in	8	Das neue Byte.

Tabelle 1.7: **RS232 Modul**

Signal	Richtung	Bits	Beschreibung
uart_rx	in	1	Die Receive Leitung des UART.
uart_tx	out	1	Die Transmit Leitung des UART.
tx_rdy	out	1	Zum Senden muss rdy low sein.
tx_go	in	1	Startet Sendevorgang.
tx_data	in	8	Das zu sendende Byte.
rx_recv	out	1	Neues Byte wurde empfangen.
rx_data	out	8	Das neue Byte.

Erlaubte Eingabe über Tastatur

Das Programm reagiert nur auf gedrückte und nicht auf losgelassene Tasten. Weiters verwenden wir die Scankeys der Zahlen und Operatoren vom NumPad und Enter, Leertaste und Backspace von der Haupttastatur.

ASCII	Scankey (Set2)	ASCII (hex)
0	0x70	30
1	0x69	31
2	0x72	32
3	0x7a	33
4	0x6b	34
5	0x73	35
6	0x74	36
7	0x6c	37
8	0x75	38
9	0x7d	39
+	0x79	2B
-	0x7b	2D
/	0xe0 0x4a	2F
*	0x7c	2A
Backspace	0x66	08
Enter	0x5a	0A
Space	0x29	20

Fehlerhafte Eingaben

Alle Tasten die nicht spezifiziert sind werden verworfen. Bei fehlerhaften Eingaben wird der Fehler vom Parser und der ALU abgefangen, die Zwischenergebnisse verworfen, die entsprechende Fehlernachricht am Bildschirm ausgegeben und in der History gespeichert.

Fehlernachrichten

Wir unterscheiden zwischen drei Fehlernachrichten:

Overflow Wenn bei irgendeiner Berechnung ein Overflow Fehler auftritt.

Division durch Null Sollte bei Irgendeiner Division im Zähler null stehen wird diese Nachricht ausgegeben.

falscher Syntax Zu falscher Syntax zählt ein Operand am Ende, zwei Operanden hintereinander oder eine Punktrechnung am Anfang.

4.5 Physical Interfaces

Die Physikalischen Interfaces der gesamten angeschlossenen Hardware lässt sich in nachfolgender Pintabelle ablesen.

Signal	Pin	Direction	Logic Level
sys_clk	N3	in	LVTTL
sys_res_n	AF17	in	LVTT
btn_a	A3	in	LVTTL
uart_cts	D20	out	LVTTL
uart_rts	D21	in	LVTTL
uart_txd	D22	out	LVTTL
uart_rxd	D23	in	LVTTL
ps2_data	E21	bidirec	LVTTL
ps2_clk	Y26	bidirect	LVTTL
vga_r0	E22	out	LVTTL
vga_r1	T4	out	LVTTL
vga_r2	T7	out	LVTTL
vga_g0	E23	out	LVTTL
vga_g1	T5	out	LVTTL
vga_g2	T24	out	LVTTL
vga_b0	E24	out	LVTTL
vga_b1	T6	out	LVTTL
vga_hsync_n	F1	out	LVTTL
vga_vsync_n	F2	out	LVTTL

5 Detailed design description

5.1 Input

Die Input Komponente empfängt die Scancodes vom PS2 Modul. Jedes Drücken der Tastatur löst ein bis drei Scancodes hintereinander aus. Hier wird zwischen zwei Tasten unterschieden. Normale Tasten senden einen Scancode beim Drücken und zwei Scancodes, wobei der erste immer 0xF0 ist, beim Loslassen einer Taste und Sondertasten, wie etwa die Enter Taste auf dem Nummernblock, die beim Drücken zwei und beim Loslassen drei Scancodes an die Input Komponente schickt. Die empfangenen Codes werden mit einem Lookup Table verglichen.

Unterscheidung der Empfangenen Daten:

0-9,+,-,*,/
- Wandeln der Scancodes in ASCII chars
- Speichern der Chars im RingBuffer
- Senden der Chars an den Output

Enter Senden an Parser und Output

Backspace Senden an RingBuffer und Output

Space Senden an RingBuffer und Output

Sonstige Alle anderen Scancodes werden verworfen.

Des Weiteren überwacht das Input Modul einen Button am development Board und sendet daraufhin eine Anfrage an den SerialHandler

5.2 Output

Das Output Modul bekommt vom Input und vom Parser Nachrichten

Vom Input wird jeder ASCII Code oder die Backspace Taste an den Output geschickt. Diese werden sofort an die VGA Komponente weiter gegeben und somit der Bildschirm aktualisiert. Hier wird auch überprüft ob schon 70 Zeichen in der Rechnung sind und reagiert dann dementsprechend nur noch auf die Backspace Taste die das letzte Zeichen löscht und den Cursor um eine Stelle nach hinten setzt.

Der Parser schickt das Ergebnis an den Output. Durch das Empfangen des Ergebnisses weiß die Komponente das die Rechnung vorbei ist. Es wird in die nächste Zeile gewechselt und danach ein '=' Zeichen und das Ergebnis dahinter auf den Bildschirm geschrieben. Danach wird wieder in die nächste Zeile gewechselt und der Cursor dorthin gesetzt. Somit kann die nächste Rechnung eingegeben werden.

Über drei Signale kommuniziert der Output mit dem VGA Modul. Das signalname signalisiert uns das das VGA Modul frei ist und neue Befehle gesendet werden können. Wenn neue Befehle an den Output geschickt werden speichert das Modul die Befehle zwischen und wartet bis die VGA Komponente frei ist. Mit signame1 wird er Befehl gesendet und signame2 hilft beim Übernehmen des Signals.

5.3 ALU

Die ALU wird vom Parser zum Lösen simpler Berechnungen genützt. An die beiden Daten Eingänge *calc_data* werden die Operanden und an *calc_operator* der Operator angelegt. Die Kodierung steht in folgender Tabelle:

calc_operator (binär)	Rechenoperation	Rechnungsart
00	Addieren	Strichrechnung
01	Subtrahieren	Strichrechnung
10	Multiplizieren	Punktrechnung
11	Dividieren	Punktrechnung

Nachdem alle für die Berechnung benötigten Daten anliegen, wird mit dem Signal *calc_start* die Berechnung gestartet. *calc_finished* sagt dem Parser, dass die Berechnung fertig ist. Sollte ein Fehler aufgetreten sein wird am *calc_error* ein Fehlercode gespeichert, der vom Parser überprüft werden muss. Der Fehlercode steht in folgender Tabelle:

calc_error (binär)	Fehler
00	Fehlerfrei, Ergebnis gültig
01	Division durch Null
10	Overflow
11	reserviert

Sollte kein Fehler vorgekommen sein, kann der Parser das Ergebnis auf *calc_data* ablesen.

5.4 Parser

Der Parser löst die Gleichung, indem er die Rechnung in für ihn lösbare simple Rechnungen zerlegt. Er wartet ihm durch den Input mit *char_EOL* gesagt wird die Rechnung zu lösen.
Der Parser holt sich dann vom Ringbuffer die momentane Zeile. Dann durchsucht er die Rechnung nach Punktrechnungen und löst diese der Reihe nach mit der ALU. Wenn keine mehr vorhanden sind wird die Rechnung erneut auf Strichrechnungen durchsucht und der Reihe nach gelöst bis das Ergebnis fest steht.
Sollte die ALU in einem Schritt einen Overflow oder einen Divison durch Null Fehler bekommen wird die gesamte Rechnung gestoppt und an den Ringbuffer und dem Output eine Fehlermeldung geschickt. Das selbe passiert wenn der Parser in einem Rechenschritt nicht die erwartete Syntax bekommt.

5.5 Ringbuffer

Der Ringbuffer ist grundsätzlich, wie der Name vermuten lässt, als Ringbuffer Struktur mit 50 Zeilen zu je 81 Characters realisert. Es gibt einen Zeiger der auf die momentage Zeile zeigt. In dieser Zeile steht die Rechnung die über die Tastatur eingegeben wurde und wird später vom Parser zur Berechnung geholt. Wenn das Ergebnis vom Parser zu der Rechnung hinzugefügt wird, wird automatisch der interne Zeiger auf die nächste Zeile verwiesen und der Inhalt der Zeile gelöscht, falls bereits mehr als 50 Rechnungen eingegeben wurden.
Der Ringbuffer bekommt vom Input, über *inp_new_data* und *inp_data*, und vom Parser, über *pars_new_data* und *pars_data*, einzellne Characters zugeschickt.
Diese Characters werden, solange nicht mehr als 70 Zeichen in einer Zeile sind, in den Speicher geschrieben. Vom Parser kriegt der Ringbuffer die einzellnen Zeichen des Ergebnisses. Das letzte Zeichen ist ein Sonderzeichen worauf der Ringbuffer die momemtane Zeile beendet und den internen Zeiger auf die nächste Zeile verweist und den Inhalt der Zeile löscht.
Wenn der Ringbuffer beschäftigt ist setzt er *rb_busy* auf High und signalisiert den anderen Komponenten das keine neuen Zeichen gesendet werden dürfen.

5.6 RS232

Das RS232 Modul wurde möglichst einfach gehalten. Es wartet bis es über das Signal *tx_go* vom SerialHandler der Befehl kommt neue Daten zu senden. Die Daten werden über *tx_data* gelesen und über den Bus gesendet. Außerdem hört die Komponente auf der Leitung ob Daten empfangen werden und signalisiert diese mit dem *rc_recv* Signal. Die vom RS232 Interface empfangenen Daten können dann über *rx_data* gelesen werden. Gesendet und empfangen wird mit 8 Datenbits, einem Stopbit und keinem Paritätsbit, bei einer Baudrate von 115‘200.