Tutorium zu Computer-Engineering im SS19 Termin 2

Jakob Otto

HAW Hamburg

9. April 2019



Ablauf

Praktikum

- ► Ideen zum Aufbau
- ► Trial-subtraction-Verfahren
- Testen







Aufgabenzettel





Addierer/Subtrahierer?!

- Ihr dürft einen Addierer und einen Subtrahierer nutzen.
 - der Addierer ist für das Inkrementieren des Zustands gedacht
 - der Subtrahierer für den Algorithmus
- Denkt an Aufgabe 3 aus DT und steuert den Addierer/Subtrahierer.

4 / 14



NICHT!!

```
something: process (...) is
  -- paar variablen
begin
  if (select v = '0') then
    res v := a + b;
  else
   res v := b + c;
  end if;
  -- ggf. interpretation
 res_s <= res_v;
end process;
```

5 / 14

Addierer/Subtrahierer?!

BESSER.

```
something: process (...) is
  -- paar variablen
begin
  if (select_v = '0') then
    op_a := a;
   op b := b;
  else
   op a := b;
   op b := c;
  end if;
 res_v := op_a + op_b
  -- ggf. interpretation.
 res_s <= res_v;
```

kleiner Tipp zum Code

- Versucht NICHT den Code modular zu gestalten
- d.H. schreibt einen (zwei mit sequlo) Prozess, der die State machine beinhaltet
- Modularisieren ist eine gute Sache, aber macht das Ziel von 40 Zuständen schwer erreichbar...





Eingaben

- Bei Eingabe von:

braucht ihr keine Berechnung zu starten. Eingabe einfach als Ergebnis durchreichen

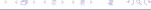
8 / 14



Trial-subtraction

Definitionen der Variablen:

- $u \rightarrow positive Zahl (die Eingabe)$
- ullet $s o \mathsf{Approximation}$ der Wurzel
- $s' \rightarrow \text{die "neue" Approximation}$
- ullet r o der mögliche Rest
- $r' \rightarrow$ ein möglicher "anderer" Rest
- $d \rightarrow \text{Differenz} \rightarrow 2^n$



Trial-subtraction

Formeln:

$$r' = r - 2^{n}(2s + 2^{n})$$

 $s' = s + 2^{n} \leftarrow \text{Hierzu gleich mehr!}$





Algorithmus:

- wenn:
 - $ightharpoonup r' >= 0 \rightarrow s'$ berechnen und s = s'
 - $ightharpoonup r' < 0 \rightarrow s = s$ bleibt also.
- \odot sobald $r' = 0 \rightarrow ENDE$





und nun in code?!

```
Denkt an:
r' = r - 2^n(2s + 2^n)
delta_v := to_stdlogicvector("10000000000000" srl n_v);
operandB_v := to_stdlogicvector(to_bitvector(
        (s v(msbPos-1 downto 0) & '0') or (delta v)) srl n v);
operandA_v := r_v;
result_v := opA_v - opB_v;
-- weiter interpretieren
```



 Otto (HAW Hamburg)
 CE Tutorium
 9. April 2019
 12 / 14

wichtig:

- Bevor ihr eine Berechnung startet:
- Eingabe auf VZ prüfen und ggf. positiv machen. \rightarrow (0 Wert) **VZ merken!**
- ullet Nach der Berechnung Ergebnis wieder Negativ machen o (0- Ergebnis)





Testframe

```
for i in -32768 to 32767 loop
  x s <= std logic_vector(to_signed(i, x_s'length));</pre>
  reg s <= '1';
  -- warten, dass Berechnung gestartet wurde
  wait for fullClockCycle;
  req_s <= '0';
  -- warten auf Ende der Berechnung
  wait for 18 * fullClockCycle;
end loop;
```



