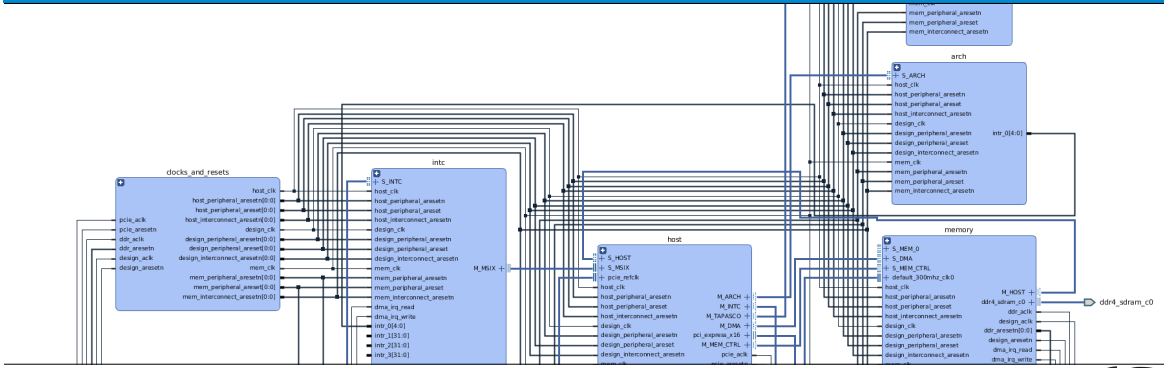
# Architekturen und Entwurf von Rechnersystemen
# Besprechung Theorieblatt 2

## Wintersemester 2022/2023

**Yannick Lavan**
Fachgebiet Eingebettete Systeme und ihre Anwendungen

TECHNISCHE UNIVERSITÄT DARMSTADT






Embedded Systems & Applications

# Intro

TECHNISCHE UNIVERSITÄT DARMSTADT

- Theorieübung 2:
  - Interfaces BSV ⇔ Verilog
  - Zeitverhalten

TECHNISCHE
UNIVERSITÄT
DARMSTADT

# Theorieübung 2

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Toolflow: BSV $\rightarrow$ Verilog $\rightarrow$ Netzliste

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Toolflow: BSV $\rightarrow$ Verilog $\rightarrow$ Netzliste
  - Genaueres Verständnis der Handshakes

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Toolflow: BSV $\rightarrow$ Verilog $\rightarrow$ Netzliste
  - Genaueres Verständnis der Handshakes
- Vorgehensweise:

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Toolflow: BSV $\rightarrow$ Verilog $\rightarrow$ Netzliste
  - Genaueres Verständnis der Handshakes
- Vorgehensweise:
  - Typ der Methode bestimmen $\Rightarrow$ Entsprechend benötigte Handshake-Signale (EN / RDY)

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

- Motivation:
  - Toolflow: BSV $\rightarrow$ Verilog $\rightarrow$ Netzliste
  - Genaueres Verständnis der Handshakes
- Vorgehensweise:
  - Typ der Methode bestimmen $\Rightarrow$ Entsprechend benötigte Handshake-Signale (EN / RDY)
  - Datensignale hinzufügen

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

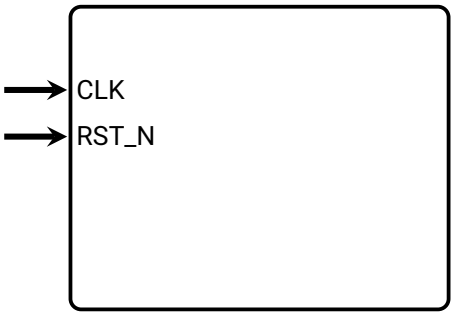
TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface Foo;
2      method Action putX(Int#(32) px);
3      method Bool getEven();
4  endinterface
5
6  module mkEven(Foo);
7      /*
8      Some fancy code...
9      */
10 endmodule
```

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

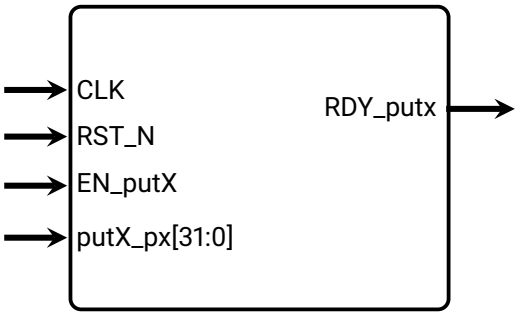
```
1  interface Foo;
2      method Action putX(Int#(32) px);
3      method Bool getEven();
4  endinterface
5
6  module mkEven(Foo);
7      /*
8      Some fancy code...
9      */
10 endmodule
```



# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

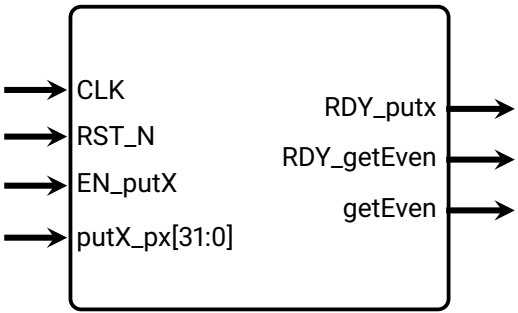
```
1  interface Foo;
2      method Action putX(Int#(32) px);
3      method Bool getEven();
4  endinterface
5
6  module mkEven(Foo);
7      /*
8      Some fancy code...
9      */
10 endmodule
```



# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  interface Foo;
2      method Action putX(Int#(32) px);
3      method Bool getEven();
4  endinterface
5
6  module mkEven(Foo);
7      /*
8      Some fancy code...
9      */
10 endmodule
```



# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 interface Bar;
2     (* always_ready, prefix="" *) method Action putX((* port="x" *)UInt#(32) px);
3     method UInt#(32) getY();
4 endinterface
5
6 module mkBar(Bar);
7     /* Hardware that solves all human problems... */
8 endmodule
```

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 interface Bar;
2     (* always_ready, prefix="" *) method Action putX((* port="x" *)UInt#(32) px);
3     method UInt#(32) getY();
4 endinterface
5
6 module mkBar(Bar);
7     /* Hardware that solves all human problems... */
8 endmodule
```

```
1 input CLK;
2 input RST_N;
```

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 interface Bar;
2     (* always_ready, prefix="" *) method Action putX((* port="x" *)UInt#(32) px);
3     method UInt#(32) getY();
4 endinterface
5
6 module mkBar(Bar);
7     /* Hardware that solves all human problems... */
8 endmodule
```

```
1 input CLK;
2 input RST_N;
3 // action method putX
4 input [31:0] x;
5 input EN_putX;
```

# Aufgabe 2.1.1: Bluespec Interface nach Verilog

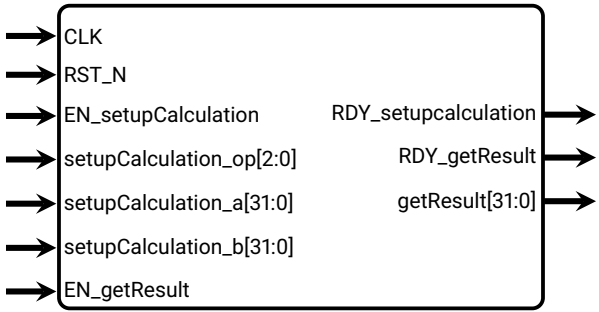
TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 interface Bar;
2     (* always_ready, prefix="" *) method Action putX((* port="x" *)UInt#(32) px);
3     method UInt#(32) getY();
4 endinterface
5
6 module mkBar(Bar);
7     /* Hardware that solves all human problems... */
8 endmodule
```

```
1 input CLK;
2 input RST_N;
3 // action method putX
4 input [31:0] x;
5 input EN_putX;
6 // value method getY
7 output [31:0] getY;
8 output RDY_getY;
```

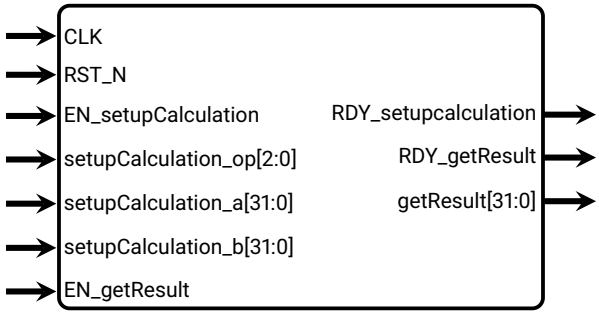
# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT



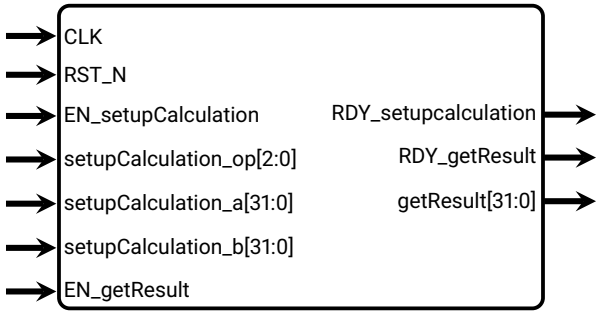

```
1 interface HelloALU;
2
3
4 endinterface
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface




```
1 interface HelloALU;
2    method Action setupCalculation(Bit#(3) op, Bit#(32) a, Bit#(32) b);
3
4 endinterface
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface




```
1 interface HelloALU;
2    method Action setupCalculation(Bit#(3) op, Bit#(32) a, Bit#(32) b);
3    method ActionValue#(Bit#(32)) getResult();
4 endinterface
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  input  CLK;
2  input  RST_N;
3
4  input  ack_pirq;
5  output interrupt;
6  input  valid_v;
7  output ready;
8  input  [15 : 0] set_px;
9  input  EN_get;
10 output [31 : 0] get;
11 output RDY_get;
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  input  CLK;
2  input  RST_N;
3
4  input  ack_pirq;
5  output interrupt;
6  input  valid_v;
7  output ready;
8  input  [15 : 0] set_px;
9  input  EN_get;
10 output [31 : 0] get;
11 output RDY_get;
```

```
1 interface FooBar;
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  input  CLK;
2  input  RST_N;
3
4  input  ack_pirq;
5  output interrupt;
6  input  valid_v;
7  output ready;
8  input  [15 : 0] set_px;
9  input  EN_get;
10 output [31 : 0] get;
11 output RDY_get;
```

```
1 interface FooBar;
2     (* always_enabled *)
3     method Action ack(Bool pirq);
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  input  CLK;
2  input  RST_N;
3
4  input  ack_pirq;
5  output interrupt;
6  input  valid_v;
7  output ready;
8  input  [15 : 0] set_px;
9  input  EN_get;
10 output [31 : 0] get;
11 output RDY_get;
```

```
1  interface FooBar;
2      (* always_enabled *)
3      method Action ack(Bool pirq);
4      (* always_ready *)
5      method Bool interrupt();
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  input  CLK;
2  input  RST_N;
3
4  input  ack_pirq;
5  output interrupt;
6  input  valid_v;
7  output ready;
8  input  [15 : 0] set_px;
9  input  EN_get;
10 output [31 : 0] get;
11 output RDY_get;
```

```
1 interface FooBar;
2     (* always_enabled *)
3     method Action ack(Bool pirq);
4     (* always_ready *)
5     method Bool interrupt();
6     (* always_enabled *)
7     method Action valid(Bool v);
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  input  CLK;
2  input  RST_N;
3
4  input  ack_pirq;
5  output interrupt;
6  input  valid_v;
7  output ready;
8  input  [15 : 0] set_px;
9  input  EN_get;
10 output [31 : 0] get;
11 output RDY_get;
```

```
1 interface FooBar;
2     (* always_enabled *)
3     method Action ack(Bool pirq);
4     (* always_ready *)
5     method Bool interrupt();
6     (* always_enabled *)
7     method Action valid(Bool v);
8     (* always_ready *)
9     method Bool ready();
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
input  CLK;
input  RST_N;

input  ack_pirq;
output interrupt;
input  valid_v;
output ready;
input  [15 : 0] set_px;
input  EN_get;
output [31 : 0] get;
output RDY_get;
```

```
interface FooBar;
    (* always_enabled *)
    method Action ack(Bool pirq);
    (* always_ready *)
    method Bool interrupt();
    (* always_enabled *)
    method Action valid(Bool v);
    (* always_ready *)
    method Bool ready();
    (* always_enabled *)
    method Action set(Int#(16) px);
```

# Aufgabe 2.1.2: Verilog nach Bluespec Interface

TECHNISCHE UNIVERSITÄT DARMSTADT

```
input  CLK;
input  RST_N;

input  ack_pirq;
output interrupt;
input  valid_v;
output ready;
input  [15 : 0] set_px;
input  EN_get;
output [31 : 0] get;
output RDY_get;
```

```
interface FooBar;
    (* always_enabled *)
    method Action ack(Bool pirq);
    (* always_ready *)
    method Bool interrupt();
    (* always_enabled *)
    method Action valid(Bool v);
    (* always_ready *)
    method Bool ready();
    (* always_enabled *)
    method Action set(Int#(16) px);
    method ActionValue#(Int#(32)) get();
endinterface
```

# Aufgabe 2.2.1: Zeitverhalten

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  module mkDeriver(Derivatives);
2      Wire#(Int#(16)) x <- mkDWire(0);
3      Wire#(Int#(32)) fx <- mkDWire(0);
4      Wire#(Int#(32)) dfx <- mkDWire(0);
5      Wire#(Int#(32)) x2 <- mkDWire(0);
6
7      rule foo;
8          x2 <= extend(x)*extend(x);
9      endrule
10
11     rule calcF;
12         fx <= 3*x2*extend(x) + 42;
13     endrule
```

```
1      rule calcDf;
2          dfx <= 9*x2;
3      endrule
4
5      method Action putX(Int#(16) px);
6          x <= px;
7      endmethod
8
9      method Int#(32) f();
10         return fx;
11     endmethod
12
13     method Int#(32) df();
14         return dfx;
15     endmethod
16 endmodule
```

# Aufgabe 2.2.1: Maximale Taktfrequenz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wir betrachten nur das gegebene Modul

# Aufgabe 2.2.1: Maximale Taktfrequenz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wir betrachten nur das gegebene Modul
  - Verzögerungen vor putX und nach f und df werden ignoriert

# Aufgabe 2.2.1: Maximale Taktfrequenz

TECHNISCHE UNIVERSITÄT DARMSTADT

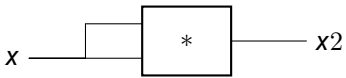
- Wir betrachten nur das gegebene Modul
  - Verzögerungen vor putX und nach f und df werden ignoriert
- Die **`WILL_FIRE`**-Bedingung aller drei Rules ist **`TRUE`**

# Aufgabe 2.2.1: Maximale Taktfrequenz

TECHNISCHE UNIVERSITÄT DARMSTADT

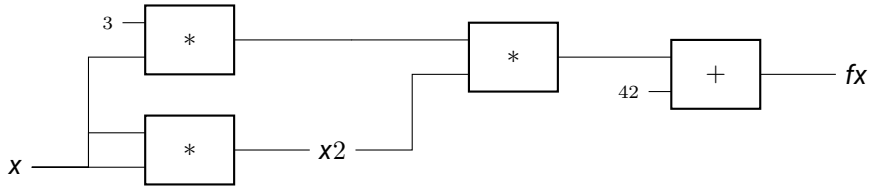
- Wir betrachten nur das gegebene Modul
  - Verzögerungen vor putX und nach f und df werden ignoriert
- Die `WILL_FIRE`-Bedingung aller drei Rules ist `TRUE`
  - Daher werden keine Multiplexer für die Wires benötigt

# Aufgabe 2.2.1: Maximale Taktfrequenz



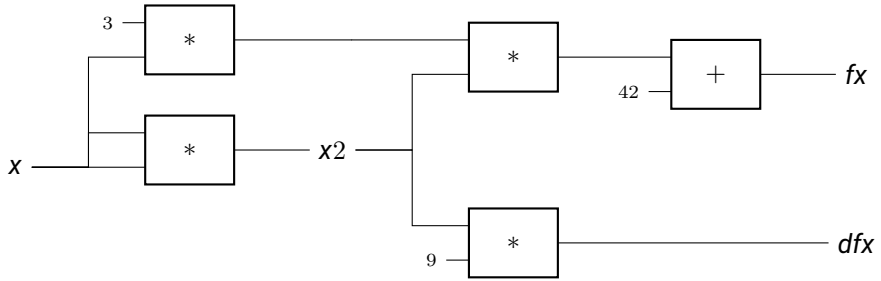

```
1   rule foo;
2       x2 <= extend(x)*extend(x);
3   endrule
```

# Aufgabe 2.2.1: Maximale Taktfrequenz




```
1   rule calcF;
2       fx <= 3*x2*extend(x) + 42;
3   endrule
```

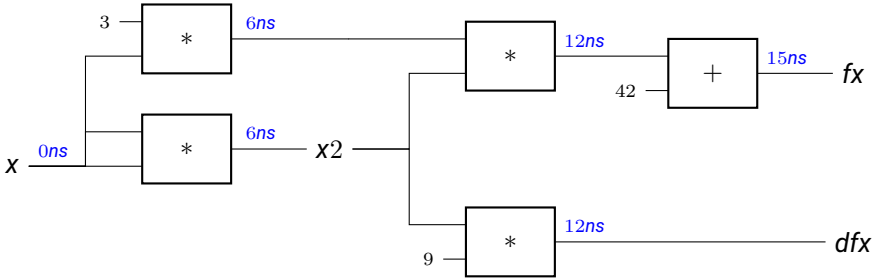
# Aufgabe 2.2.1: Maximale Taktfrequenz




```
1   rule calcDf;
2       dfx <= 9*x2;
3   endrule
```

# Aufgabe 2.2.1: Maximale Taktfrequenz

TECHNISCHE UNIVERSITÄT DARMSTADT



# Aufgabe 2.2.1: Maximale Taktfrequenz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kritischer Pfad ist 15 ns lang
- Maximale Taktfrequenz ist $\frac{1}{15\,\text{ns}} = 66.67\,\text{MHz}$

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Typen haben die Variablen?

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs
- Wir betrachten alle Rules und Methoden einzeln
  - Die put- und get-Methoden sind für Zeitverhalten nicht relevant
  - Die fin-Rules sind ebenfalls nicht relevant

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs
- Wir betrachten alle Rules und Methoden einzeln
  - Die put- und get-Methoden sind für Zeitverhalten nicht relevant
  - Die fin-Rules sind ebenfalls nicht relevant
  - Die Rules x_sqr, lin_y, sthZ schreiben in Wires
  - Die Rules calcF1, calcF2, calcF3 lesen aus Wires

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs
- Wir betrachten alle Rules und Methoden einzeln
  - Die put- und get-Methoden sind für Zeitverhalten nicht relevant
  - Die fin-Rules sind ebenfalls nicht relevant
  - Die Rules x_sqr, lin_y, sthZ schreiben in Wires
  - Die Rules calcF1, calcF2, calcF3 lesen aus Wires
  - Wire: Schreiben < Lesen!

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs
- Wir betrachten alle Rules und Methoden einzeln
  - Die put- und get-Methoden sind für Zeitverhalten nicht relevant
  - Die fin-Rules sind ebenfalls nicht relevant
  - Die Rules x_sqr, lin_y, sthZ schreiben in Wires
  - Die Rules calcF1, calcF2, calcF3 lesen aus Wires
  - Wire: Schreiben < Lesen!
  - Das Lesen eines Wires hat als Startverzögerung die (maximale) Verzögerung beim Schreiben dieses Wires.

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

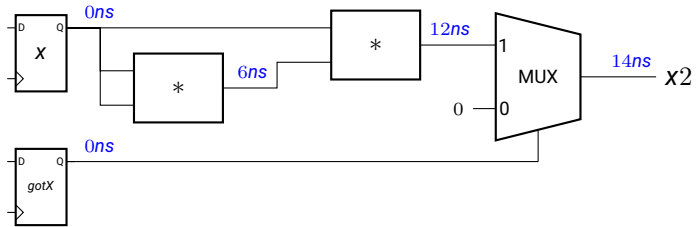
- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs
- Wir betrachten alle Rules und Methoden einzeln
  - Die put- und get-Methoden sind für Zeitverhalten nicht relevant
  - Die fin-Rules sind ebenfalls nicht relevant
  - Die Rules x_sqr, lin_y, sthZ schreiben in Wires
  - Die Rules calcF1, calcF2, calcF3 lesen aus Wires
  - Wire: Schreiben < Lesen!
  - Das Lesen eines Wires hat als Startverzögerung die (maximale) Verzögerung beim Schreiben dieses Wires.
- Die `WILL_FIRE`-Bedingung der Rules ist **nicht** `TRUE`

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

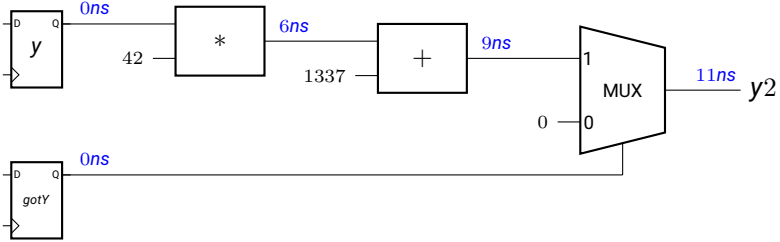
- Welche Typen haben die Variablen?
  - x2, y2, z2 sind Wires, alle anderen Regs
- Wir betrachten alle Rules und Methoden einzeln
  - Die put- und get-Methoden sind für Zeitverhalten nicht relevant
  - Die fin-Rules sind ebenfalls nicht relevant
  - Die Rules x_sqr, lin_y, sthZ schreiben in Wires
  - Die Rules calcF1, calcF2, calcF3 lesen aus Wires
  - Wire: Schreiben < Lesen!
  - Das Lesen eines Wires hat als Startverzögerung die (maximale) Verzögerung beim Schreiben dieses Wires.
- Die `WILL_FIRE`-Bedingung der Rules ist **nicht** `TRUE`
  - Wir benötigen Multiplexer für das Schreiben der Wires/Register

# Aufgabe 2.2.2: Verzögerung x_sqr



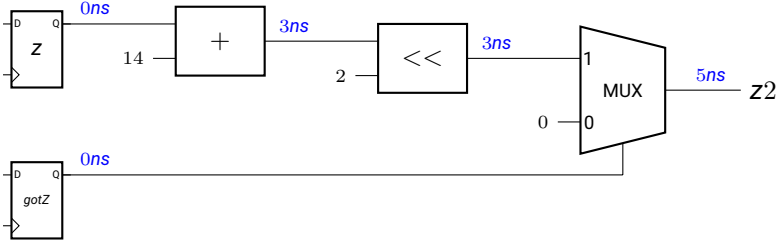

```
1       rule x_sqr (gotX);
2             x2 <= x * x * x;
3       endrule
```

# Aufgabe 2.2.2: Verzögerung lin_y



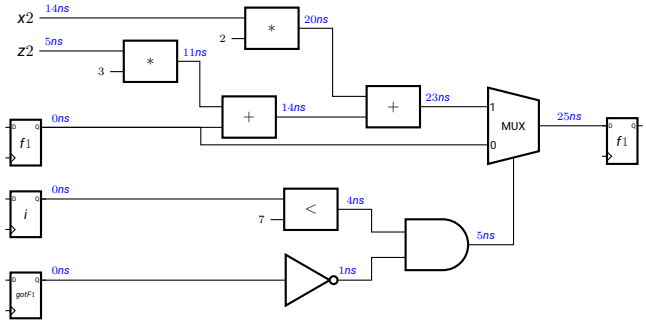

```
1       rule lin_y (gotY);
2           y2 <= 42 * y + 1337;
3       endrule
```

# Aufgabe 2.2.2: Verzögerung sthZ



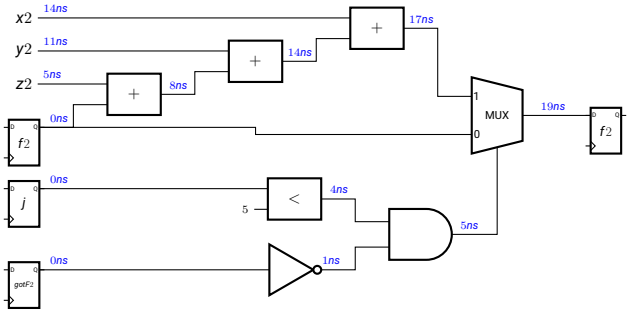

```
1   rule sthZ (gotZ);
2       z2 <= (z + 14) « 2;
3   endrule
```

# Aufgabe 2.2.2: Verzögerung calcF1



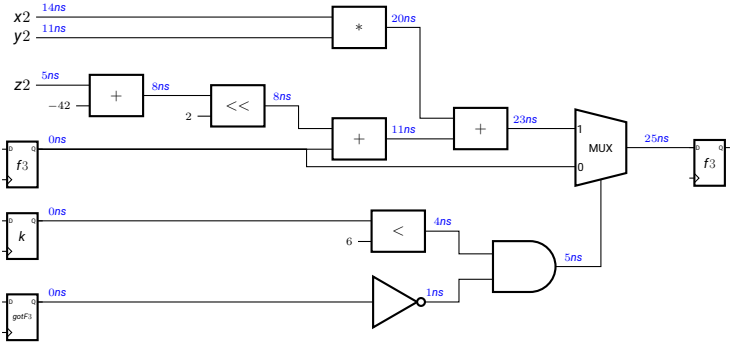

```
1       rule calcF1 (!gotF1 && i < 7);
2           f1 <= f1 + 2 * x2 + 3 * z2;
3           i <= i + 1;
4       endrule
```

# Aufgabe 2.2.2: Verzögerung calcF2




```
1       rule calcF2 (!gotF2 && j < 5);
2             f2 <= f2 + x2 + y2 + z2;
3             j <= j + 1;
4       endrule
```

# Aufgabe 2.2.2: Verzögerung calcF3




```
1       rule calcF3 (!gotF3 && k < 6);
2           f3 <= f3 + x2 * y2 + (z2 - 42) « 2;
3           k <= k + 1;
4       endrule
```

# Aufgabe 2.2.2: Maximale Taktfrequenz 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kritischer Pfad ist 25 ns lang (calcF1, calcF3)
- Maximale Taktfrequenz ist $\frac{1}{25\,\text{ns}} = 40\,\text{MHz}$

# Fragen zu Theorieblatt 2