Von POL nach VHDL

Frederik Grüll

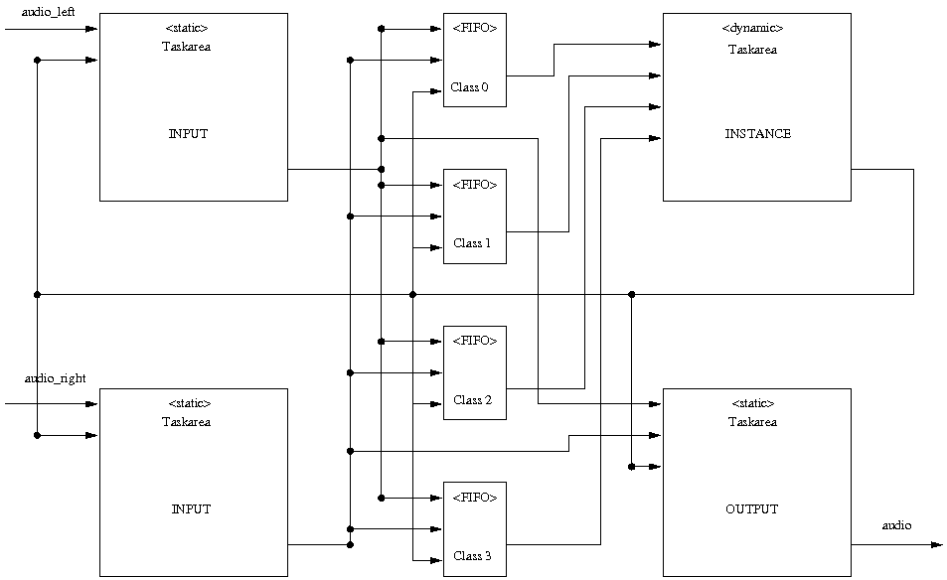
KIP

July 16, 2008

Übersicht

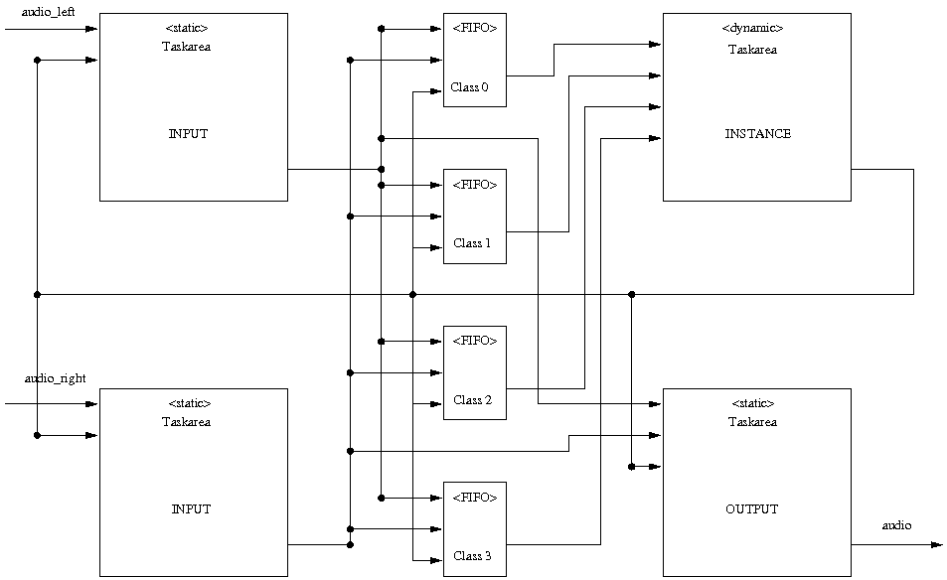
Norberts und Nicks Kommunikationsmatrix

- Kommunikation zwischen den Tasks



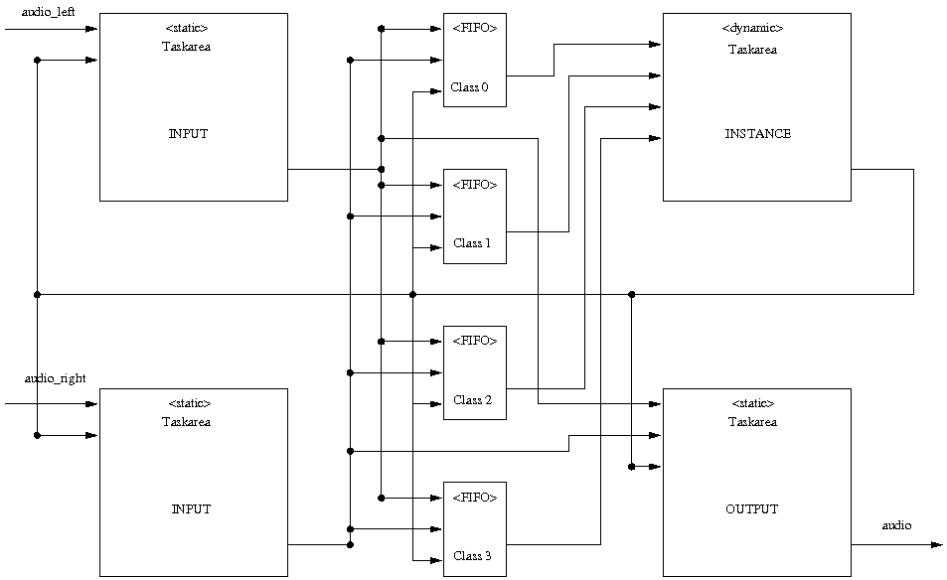
Norberts und Nicks Kommunikationsmatrix

- Kommunikation zwischen den Tasks
- Synchronisation



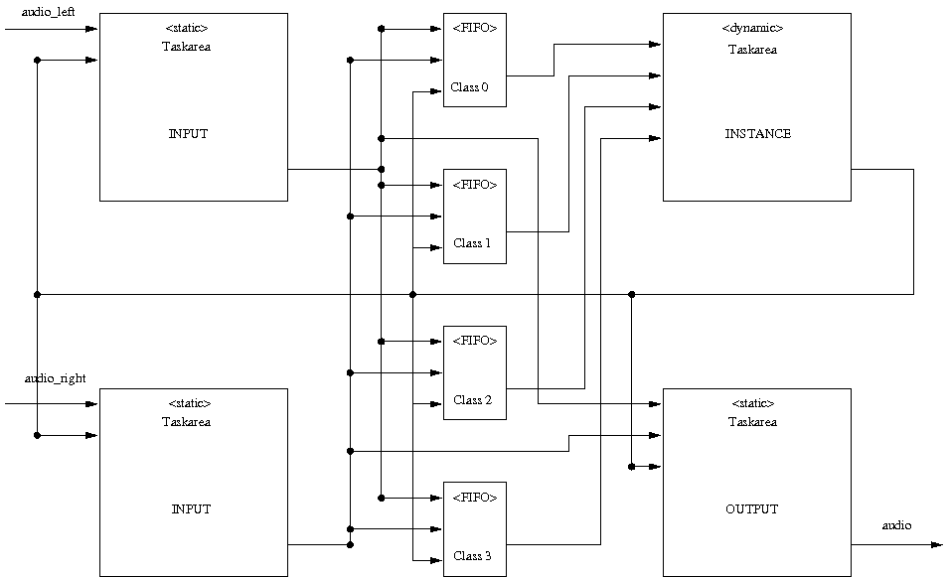
Norberts und Nicks Kommunikationsmatrix

- Kommunikation zwischen den Tasks
- Synchronisation
- Lagert Tasks aus und ein

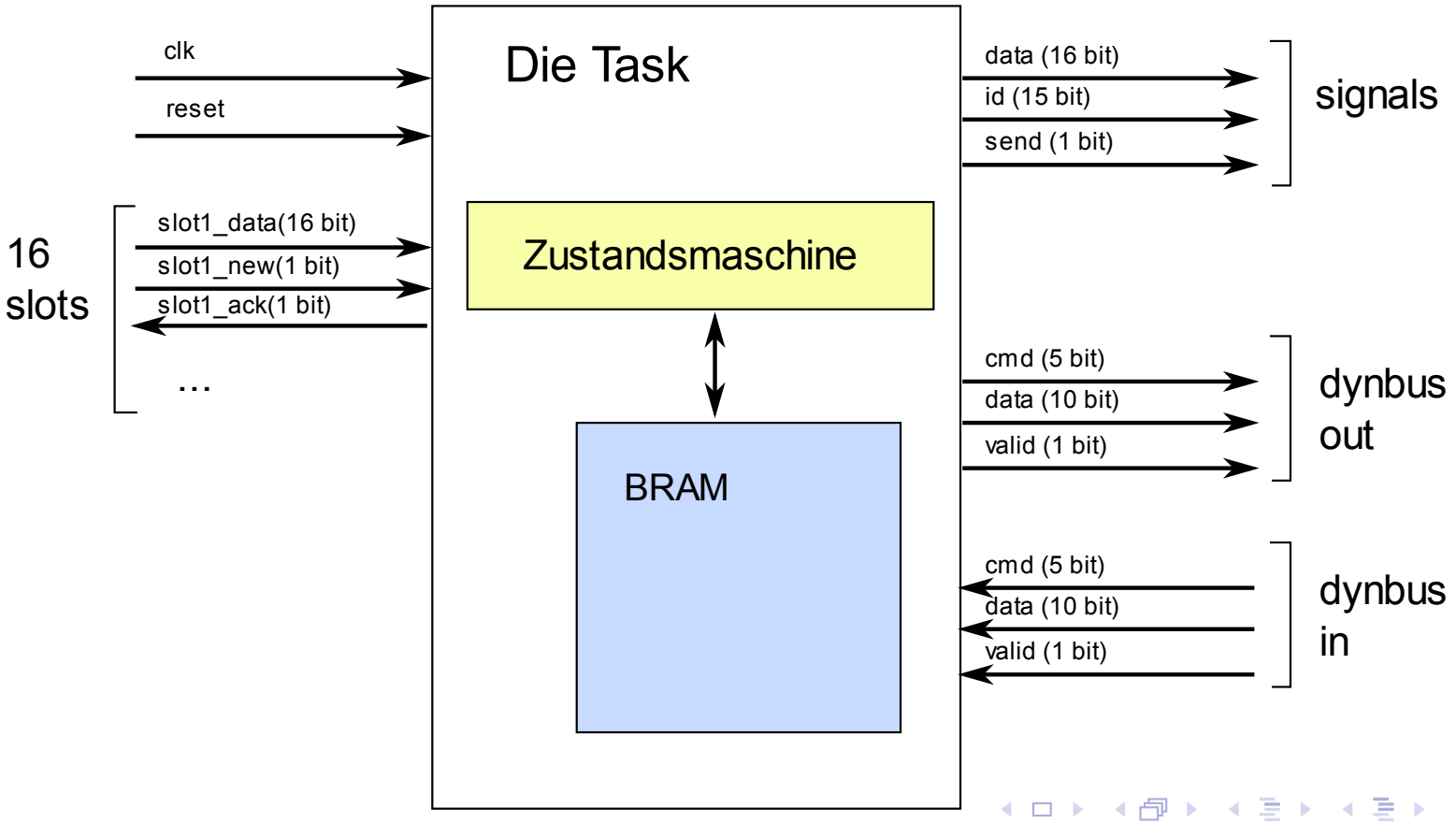


Norberts und Nicks Kommunikationsmatrix

- Kommunikation zwischen den Tasks
- Synchronisation
- Lagert Tasks aus und ein
- Überbrückt Rekonfiguration

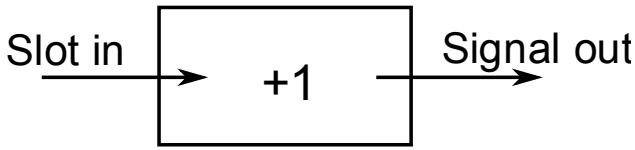


Interface zur Task



Eine Task in POL

```
class increment extends ParObj {
 int a, b;
 Slot in;
 Signal out;

 // Constructor
 increment()
 {
   a = 1;
 }

 // body
 void calc()
 {
   b = in.get();
   out.emit(a + b);
 }
}
```



Slots

```
Slot Sl;
a = Sl.get();
b = Sl.get(0);
```

- Teil des Klasseninterfaces
- erlauben den Empfang von Nachrichten

Slots

```
Slot Sl;
a = Sl.get();
b = Sl.get(0);
```

- Teil des Klasseninterfaces
- erlauben den Empfang von Nachrichten

Slot.get() blockiert, bis eine Nachricht für *Slot* eintrifft

Slots

```
Slot Sl;
a = Sl.get();
b = Sl.get(0);
```

- Teil des Klasseninterfaces
- erlauben den Empfang von Nachrichten

Slot.get() blockiert, bis eine Nachricht für *Slot* eintrifft

Slot.get(*default*) gibt falls vorhanden die Nachricht für `Slot` zurück, sonst *default*

Signals: emit()

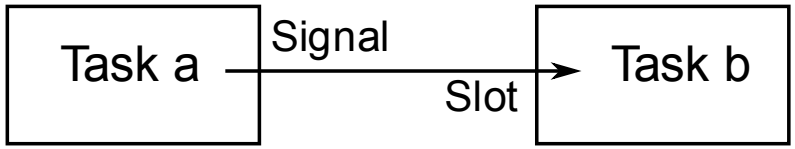
```
Signal Sig;
Sig.emit(a);
```

Signale sind wie Slots Teil des Klasseninterfaces und dienen dem Versand von Nachrichten.
Zu jedem POL-Signal gehört ein Bereich im BRAM, der die Empfänger speichert, die das Signal abonniert haben.

signal.emit(*value*) sendet *value* an alle Empfänger über die Matrix

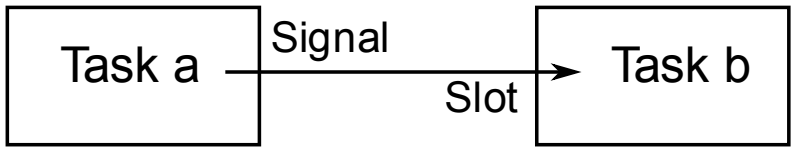
Signals: connect()

```
ClassA a;
ClassB b;
a.signal1.connect(b.slot1);
a.signal1.disconnect(b.slot1);
```



Signals: connect()

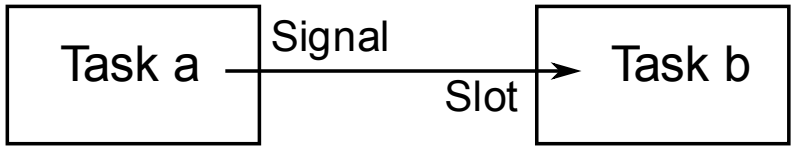
```
ClassA a;
ClassB b;
a.signal1.connect(b.slot1);
a.signal1.disconnect(b.slot1);
```



signal.connect(*slot*) Verbindet *signal* mit *slot*

Signals: connect()

```
ClassA a;
ClassB b;
a.signal1.connect(b.slot1);
a.signal1.disconnect(b.slot1);
```



signal.connect(*slot*) Verbindet *signal* mit *slot*

signal.connect(*slot*) Trennt *signal* von *slot*

Klassendefinitonen

```
class A1 extends Scheduler {

  class B1 extends ParObj {
  // ...
  }

  class B2 extends ParObj {
  // ...
  }

  //...
}
```

Aus jeder Klasse ergibt sich eine VHDL-Datei, benannt nach der Klasse. In diesem Fall `A1.vhd`, `B1.vhd` und `B2.vhd`.

new & finish

```
class A1 extends Scheduler {

  class B1 extends ParObj {
  // ...
  }

  void calc() {
    B1 b;
    b = new B1;
    //...
}
```

new & finish

```
class A1 extends Scheduler {

  class B1 extends ParObj {
  // ...
  }

  void calc() {
    B1 b;
    b = new B1;
    //...
}
```

new erzeugt eine neue Instanz einer Klasse und gibt eine Referenz darauf zuück.

new & finish

```
class A1 extends Scheduler {

  class B1 extends ParObj {
  // ...
  }

  void calc() {
    B1 b;
    b = new B1;
    //...
}
```

new erzeugt eine neue Instanz einer Klasse und gibt eine Referenz darauf zuück.

finish gibt die Instanz zur Auflösug frei (noch nicht implementiert)

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

bool: std_logic

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

bool: std_logic

int: std_logic_vector(15 downto 0)

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

bool: std_logic

int: std_logic_vector(15 downto 0)

- Jede Variable wird über ein eindeutiges Symbol identifiziert.

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

bool: std_logic

int: std_logic_vector(15 downto 0)

- Jede Variable wird über ein eindeutiges Symbol identifiziert.
- Geschachtelte Scopes mit sich überdeckendenn Variablen sind erlaubt

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

bool: std_logic

int: std_logic_vector(15 downto 0)

- Jede Variable wird über ein eindeutiges Symbol identifiziert.
- Geschachtelte Scopes mit sich überdeckendenn Variablen sind erlaubt
- Es gibt keine globalen Variablen

Deklarationen

```
bool b;
int i, j;
```

Variablen werden in VHDL zu Signals und Variables

bool: std_logic

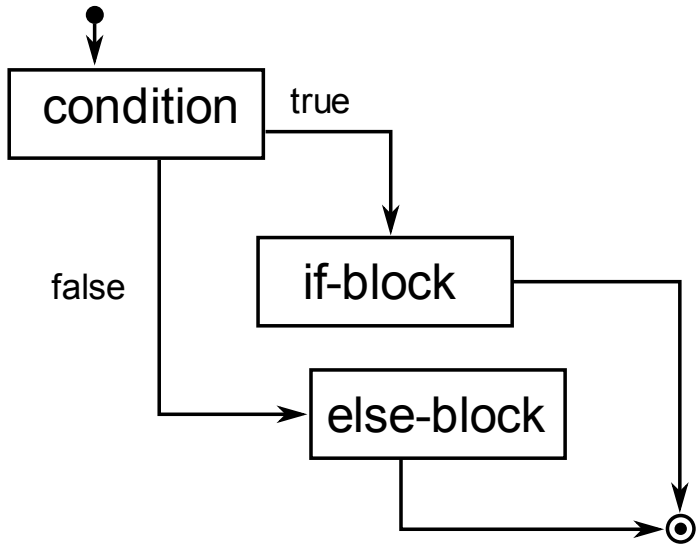
int: std_logic_vector(15 downto 0)

- Jede Variable wird über ein eindeutiges Symbol identifiziert.
- Geschachtelte Scopes mit sich überdeckendenn Variablen sind erlaubt
- Es gibt keine globalen Variablen
- Ein Objekt kann nur über die Matrix mit anderen Objekten kommunizieren

if-else

```
if (condition) {
  // if block...
} else {
  // else block
}
```

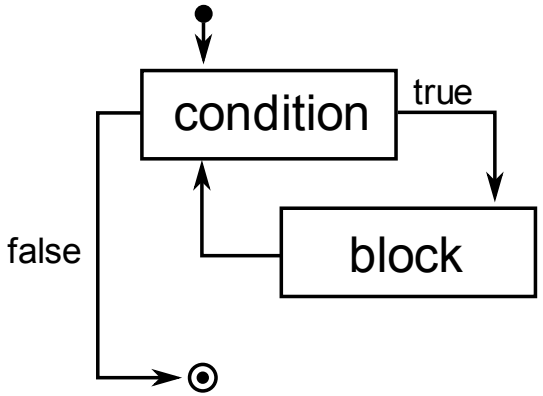
Die Bedingung kann aus meheren Zuständen bestehen, wenn sie ein get() enthält



while

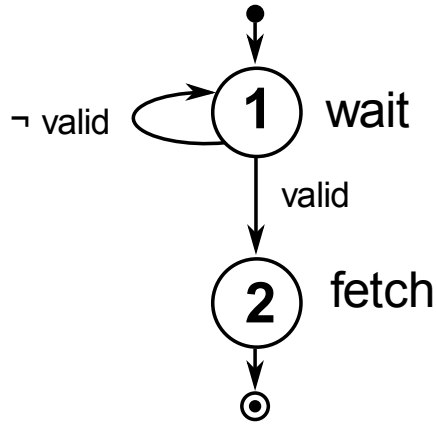
```
while (condition) {
  // block...
}
```

Analog zur if-Verzweigung



get()

1. Überprüfe, ob Daten anliegen
2. Bestätige Empfang



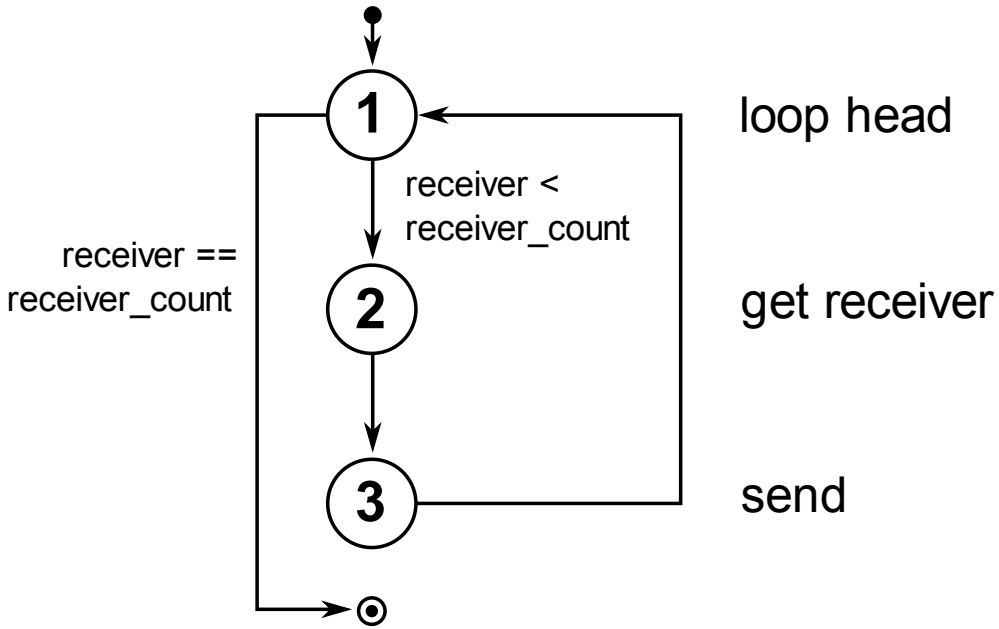
- Mehrere get() auf unterschiedliche Slots werden zusammengefassst.
- gleichen Slots: Hilfsregister

get(default) besteht nur aus dem zweiten Zustand und

```
slot_data when slot_valid = '1' else (default);
```

emit()

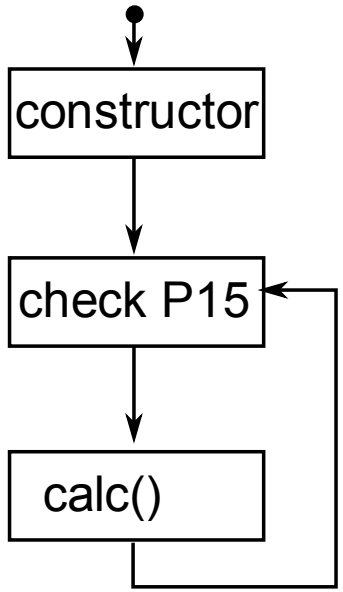
1. Überprüfe, ob alle Empfänger benachrichtigt wurden
2. Falls nicht, hole die Empfängeradresse aus dem BRAM
3. Sende das Datum und erhöhe die BRAM-Adresse um eins.



Klassen & Objekte

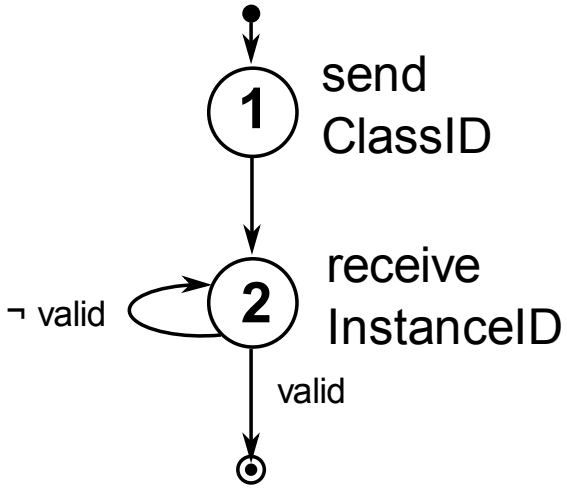
Klassen bestehen aus der calc()-Routine und evtl. einem Konstruktor

- Der Konstruktor wird nur zu Beginn ausgeführt
- Bevor calc() beginnt überprüft die Matrix Port 15 auf neue Nachricht von connect() oder disconnect()
- Danach beginnt der in calc() definierte Teil



new

1. Die gewünschte Klassen-ID wird über den DynBus gesendet
2. Die Zustandsmaschine wartet, bis die Instanz-ID des neuen Objekts in Empfang genommen werden kann.



Unterbrechbarkeit

Die Matrix kann die Task bitten, ihren Zustand im BRAM zu sichern und wird darauf komplett ins DRAM auslagern zu lassen.

Unterbrechbarkeit

Die Matrix kann die Task bitten, ihren Zustand im BRAM zu sichern und wird darauf komplett ins DRAM auslagern zu lassen.

- Während jedem blockierenden get()

Unterbrechbarkeit

Die Matrix kann die Task bitten, ihren Zustand im BRAM zu sichern und wird darauf komplett ins DRAM auslagern zu lassen.

- Während jedem blockierenden get()
- Am Ende von calc()

Unterbrechbarkeit

Die Matrix kann die Task bitten, ihren Zustand im BRAM zu sichern und wird darauf komplett ins DRAM auslagern zu lassen.

- Während jedem blockierenden get()
- Am Ende von calc()
- In jeder Schleife mindestens einmal

component-Schlüsselwort

```
include "sevensegment.vhd"

int nr;
component(sevensegment, in => nr);
```

Zusammen mit include würde component die Wiederverwendung von bestehendem VHDL-Code erlauben.

Anforderungen an die VHDL-Komponente:

component-Schlüsselwort

```
include "sevensegment.vhd"

int nr;
component(sevensegment, in => nr);
```

Zusammen mit include würde component die Wiederverwendung von bestehendem VHDL-Code erlauben.

Anforderungen an die VHDL-Komponente:

- Der Clock-Eingang heißt CLK, der Reset RST

component-Schlüsselwort

```
include "sevensegment.vhd"

int nr;
component(sevensegment, in => nr);
```

Zusammen mit include würde component die Wiederverwendung von bestehendem VHDL-Code erlauben.

Anforderungen an die VHDL-Komponente:

- Der Clock-Eingang heißt CLK, der Reset RST
- strobe_in ist ein Eingang, der der Komponenten anzeigt, dass Daten am Eingang anliegen

component-Schlüsselwort

```
include "sevensegment.vhd"

int nr;
component(sevensegment, in => nr);
```

Zusammen mit include würde component die Wiederverwendung von bestehendem VHDL-Code erlauben.

Anforderungen an die VHDL-Komponente:

- Der Clock-Eingang heißt CLK, der Reset RST
- strobe_in ist ein Eingang, der der Komponenten anzeigt, dass Daten am Eingang anliegen
- strobe_out zeigt der Task an, dass Daten am Ausgang anliegen

component-Schlüsselwort

```
include "sevensegment.vhd"

int nr;
component(sevensegment, in => nr);
```

Zusammen mit include würde component die Wiederverwendung von bestehendem VHDL-Code erlauben.

Anforderungen an die VHDL-Komponente:

- Der Clock-Eingang heißt CLK, der Reset RST
- strobe_in ist ein Eingang, der der Komponenten anzeigt, dass Daten am Eingang anliegen
- strobe_out zeigt der Task an, dass Daten am Ausgang anliegen
- Alle übrigen Ein- und Ausgänge haben 16 (int) oder 1 bit (bool)

import-Schlüsselwort

```
import Uxibo.SevenSegment;

class A extends ParObj {
  Signal nr;
  SevenSegement s;

  A() {
    nr.connect(SevenSegment.in);
  }

  calc() { ... }
}
```

Import würde die Verwendung von vorgefertigten POL-Modulen erlauben, die gegebenfalls unter Verwendung von *component* erstellt wurden.

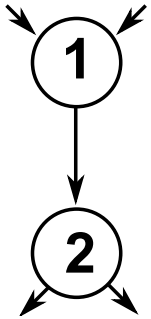
weitere Funktionen neben calc()

```
class A extends ParObj {
Slot nr;

bool even(int i) {
  (i == 0) ? return true : odd(i - 1);
}

bool odd(int i) {
  i == 0) ? return false : even(i - 1);
}

void calc() {
  if (even(nr.get()))
  ...
}
```

Benötigt einen Stack im BRAM.

Optimierung

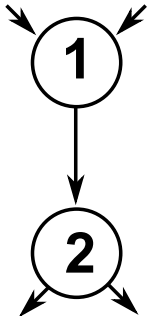
Bis jetzt belegt jedes POL-Statement mindestens einen Zustand.
Zwei Zustände S_1 und S_2 können verschmolzen werden, wenn gilt:



Optimierung

Bis jetzt belegt jedes POL-Statement mindestens einen Zustand.
Zwei Zustände S_1 und S_2 können verschmolzen werden, wenn gilt:

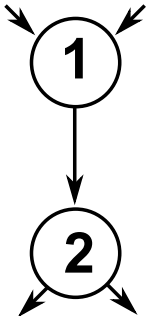
- S_1 ist der einzige Vorgänger von S_2 und S_2 ist der einzige Nachfolger von S_1



Optimierung

Bis jetzt belegt jedes POL-Statement mindestens einen Zustand.
Zwei Zustände S_1 und S_2 können verschmolzen werden, wenn gilt:

- S_1 ist der einzige Vorgänger von S_2 und S_2 ist der einzige Nachfolger von S_1
- S_2 liest und schreibt nichts, was S_1 geschrieben hat, einschließlich der Kommunikationsmatrix



Alles weitere kann in VHDL optimiert werden.

Ende

Vielen Dank