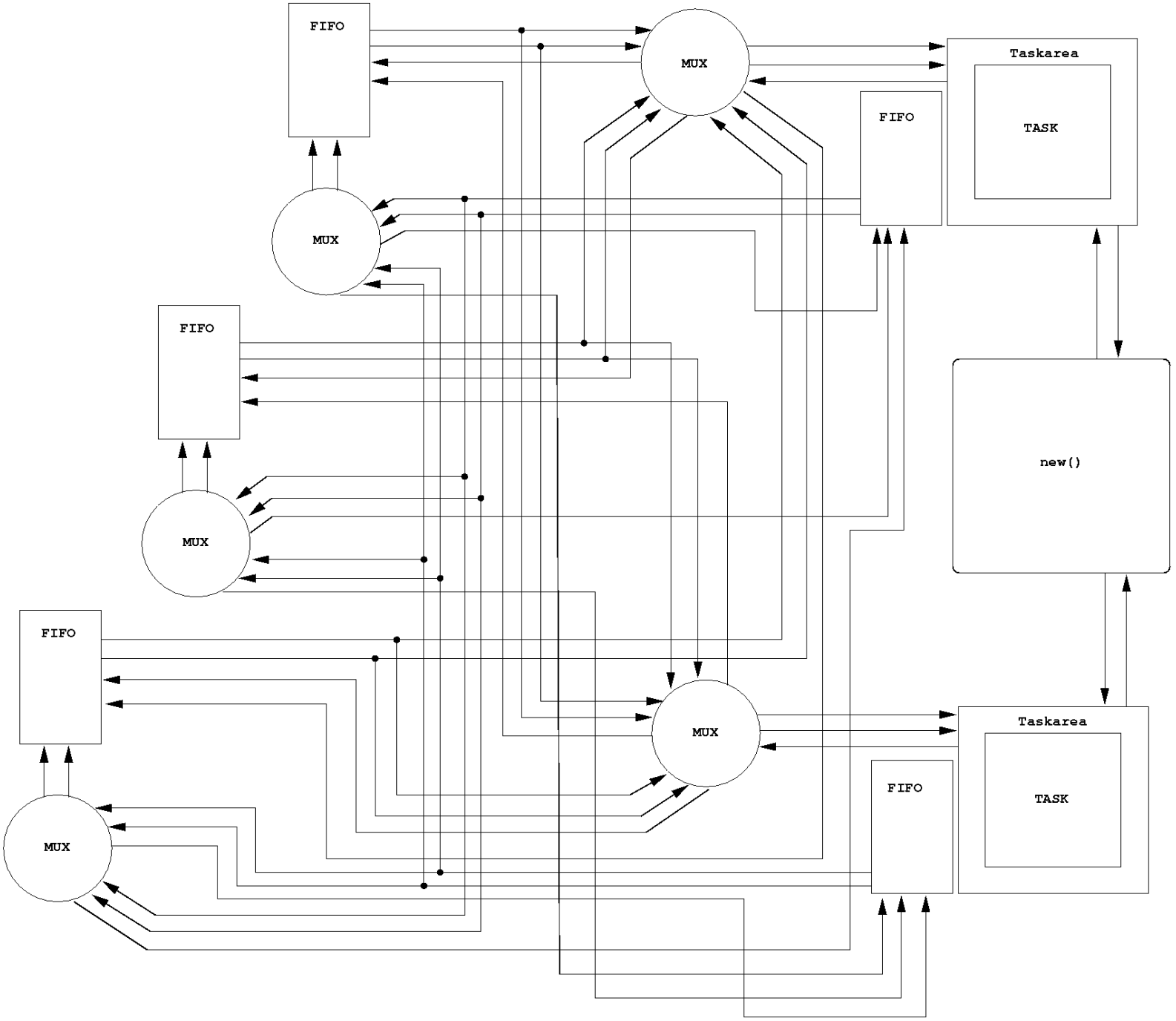
KIP Heidelberg

Norbert Abel

POL - Spezification 0.1

DPR-Architektur



DPR-Architektur - Kommunikation

- Für jede Task-Area gibt es eine Ausgangs-FIFO und einen Eingangs-MUX
- Für jede Klasse gibt es einen Klassen-MUX und eine Klassen-FIFO

- Daten werden mit Klassen-ID + Instanz-ID + Register-ID in die Ausgangs-FIFO geschrieben
- Die Klassen-MUX holen sich (möglichst am Stück) die Daten aus einer Ausgangs-FIFO, welche die Entsprechende Klasse adressiert und schreiben die Daten in ihre Klassen-FIFO.
- Die Eingangs-MUX holen die Daten aus den Klassen-FIFOs, die an das in ihrer Task-Area befindliche Objekt gehen.
- Lokale Daten werden in taskarea-internen BRAMs gespeichert, welche mittels Rekonfiguration gesichert / initialisiert werden

DPR-Architektur - Rekonfiguration

- Es gibt eine “kurze Rekonfiguration” und eine “lange Rekonfiguration”

- Die **kurze Rekonfiguration** entspricht einem Kontextwechsel bei unveränderter Hardware
- Nur die BRAM-Inhalte werden ausgetauscht und ein Reset gezogen
- Die kurze Rekonfiguration wird von den Anliegenden Eingangs-Daten gesteuert.
  Es wird immer die Instanz geladen, an die die Daten gehen.

- Die **lange Rekonfiguration** entspricht einer vollständigen Rekonfiguration.
- Es werden die Hardware und der Kontext gewechselt.
- Die lange Rekonfiguration wird vom Scheduler ausgelöst.
- Die Entscheidung, was wann geladen wird,
  hängt vor allem vom Füllgrad der Klassen-FIFOs ab

DPR-Architektur - Spezifikationen

- Jede Klasse hat eine ID (4 bit) - das System hat die Klassen-ID 0xf
- Jedes Objekt hat eine Instanz-ID (6 bit)
- Jedes Objekt hat Eingangsregister (Channels) (4 bit)

- Alle Variablen sind Integer à 16 bit
- Lesezugriffe sind (aus Task-Sicht) blockierend

Effektivität der Rekonfiguration:

--->Datensortierung

(sets und calc() zu einer Instanz zusammenfassen, so dass sie in der FIFO direkt hintereinander kommen -> Reduzierung der Zahl der kurzen Rekonfigurationen)

2 Möglichkeiten:

SW: POL-Compiler
HW: Klassen-MUX holt Set-Daten nur von 1 Objekt bis calc()-Signal kommt.
--->Ist das realisierbar? Was ist mit Konstruktor-Sets?

Pipelining:

Die Verdrahtungsvariante des POL-FPL-Papers in Fig. 3 ist ungeschickt, da hier ein 4. Objekt (aus der 3. calc()-methode) erzeugt wird, welche die Zugriffe sequentialisiert.

Das automatische Übersetzen von Ausgangssignal-IDs auf Klassen- und Objekt-IDs funktioniert nicht zusammen mit dem dynamischen Anlegen von Objekten. Bei letzterem wird nämlich eine Instanz-ID zurückgegeben, welche vom Task für die weiteren Zugriffe verwendet wird.

---> Die Parallelisierung muss auf POL-Ebene passieren:

```
class Adder extends ParObj {
    private int a,b, mode, s;
    setter a;
    setter b;
    setter mode;
    getter s;
    void calc() {
        s = a + b;
        if (m==0) Multipliers<0>.set_a(s);
        if (m==0) Multipliers<1>.set_a(s);
    }
}
```

Pipelining:

Eine automatische Parallelisierung / Pipelining ist nicht denkbar, da der POL-Programmierer auch folgendes bauen könnte:

```
Adders<0>.set_a(X1);
Adders<0>.set_b(X2);
Adders<0>.calc();
Adders<1>.set_a(Y1);
Adders<1>.set_b(Y2);
Adders<1>.calc();
Multipliers<0>.set_a(Adders<0>.get-s());
Multipliers<0>.set_b(Adders<1>.get-s());
P1 = Multipliers<0>.get_p() + X1;
```

2 Aufgaben:

1. Neue Objekte Anlegen

POL: Adders.add(new Adder());

```
DynBus_OP <= “01”;
DynBus_wID <= KlassenID & “0000”;
dann:
if (DynBus_OK = '1') then
   NeueInstanzID <= DynBus_rID;
end if;
```

2. Objekte löschen

POL: Adder<0>.finish();

```
DynBus_OP <= “10”;
DynBus_wID <= KlassenID & InstanzID;
dann:
if (DynBus_OK = '1') then
   weiter...
end if;
```

DPR-Architektur - Offene Fragen

- HW: Klassen-MUX holt Set-Daten nur von 1 Objekt bis calc()-Signal kommt.
  --->Ist das realisierbar? Was ist mit Konstruktor-Sets?

- Wie soll die Objekt-Handler-Verwaltung im Detail passieren?

DPR-Architektur - Aufgaben

Nick:

- Architektur prüfen
- Spezifikation erweitern / vervollständigen
- Busse und Bus-Protokolle spezifizieren

Frederik:

- “Gutes” POL-Beispiel aufschreiben (Addierer / Multiplizierer)
- Entsprechenden VHDL-Code von Hand generieren

Andreas:

- Entsprechenden Java-Code von Hand generieren

Alle:

- Offene Fragen aufschreiben
- Repository ergänzen

REPOSITORY

- FPL-Paper “POL” (Norbert)
- Java-VHDL-Beispiel (Norbert)