# JDemoComPlus

Technisches Handbuch

## Einleitung

JDemoComPlus ist ein Simulator für die in der Vorlesung Rechnerarchitektur vorgestellte DemoComPlus-Beispielarchitektur. Beim Design der Softwarearchitektur waren 3 Grundgedanken maßgeblich: die Komponenten der Hardwarearchitektur sollten so realitätsnah wie möglich abgebildet werden, gleichzeitig stark grafisch orientiert arbeiten und dazu in einer Art Baukastensystem für erweiterte Architekturen wiederverwendbar sein.

## Design allgemein

Die DemoComPlus-Komponenten wurden von der Swing-Klasse JPanel abgeleitet und die HW-Funktionalität somit direkt an die grafische Komponente gekoppelt. Ein Register z.B. stellt also nicht nur den DemoComPlus-Hardwareteil dar, sondern enthält gleichzeitig alles was zur Anzeige seines Zustandes nötig ist. Dadurch lassen sich die Klassen und ihre Subklassen leicht in grafische Oberflächen integrieren, durch die direkte Abbildung aller Komponenten als in sich abgeschlossene Elemente wird ein sehr intuitiver Programmierstil ermöglicht und die resultierende Anwendung ist leicht erweiterbar.

Um die Verteilung des Systemtaktes und der Kontrollsignale an die Komponenten zu realisieren wurden entsprechende Interfaces entworfen, die von den Komponenten implementiert werden. Der Systemtakt des JDemoComPlus wird direkt von der Hauptanwendung generiert und an die Komponenten verteilt.

Intern verwendet der Simulator zum Speichern von Registerinhalten oder zur Durchführung von Rechenoperationen Integer-Variablen, diese werden bei Beadrf zur Darstellung in Binärwerte in 2-Complement-Darstellung umgewandelt.

# Hauptkomponenten

## Der Systemtakt

Ein grundsätzliches Problem bei der Verteilung des Systemtaktes und der Simulation der Funktionen einer CPU ist, daß alle Komponenten den Takt zur selben Zeit erhalten sollten, jedoch mit Software-Routinen nur sequentiell Programmteile abgearbeitet werden können. Da die Komponenten zum Teil sehr komplex miteinander interagieren treten leicht Probleme auf, wenn eine Komponente beispielsweise Daten vom Systembus lesen möchte, eine andere Komponente jedoch genau die gewünschte Information noch nicht auf den Bus angelegt hat, weil sie in der Sequenz erst später abgearbeitet wird.

Um diesem Problem zu begegnen wurde eine ansteigende bzw. abfallende Taktflanke jeweils in zwei Abschnitte geteilt, einen Abschnitt für „datenbereitstellende“ Funktionen und einen für „datennutzende“ Funktionen. Der Taktverteiler durchläuft zuerst komplett den ersten Abschnitt, danach erst den zweiten. Somit ist die Reihenfolge der Abarbeitung nicht mehr wesentlich, nachdem beide Abschnitte durchlaufen wurden ist die Taktflanke beendet ohne Konflikte zu erzeugen.

Die Aufrufmethoden der Abarbeitungsfunktionen wurde im Interface ClockConsumer zusammengefasst, welches von allen Komponenten die an den Systemtakt angeschlossen sind implementieren:

```
public interface ClockConsumer {
	// steigende Taktflanke
	public void clk_raise_start();
	public void clk_raise_end();

	//fallende Taktflanke
	public void clk_lower_start();
	public void clk_lower_end();
}
```

Ein Register, der vom Bus lesen und auf den Bus schreiben kann verwendet beispielsweise folgende Implementation dieser Methoden (im PseudoCode):

```
public void clk_raise_start() {
            if(„Schreibesignal liegt an“) {
              Bus.grab();
              }
      }

public void clk_raise_end() {
        if („Ladesignal liegt an“){
          inhalt=Bus.read();
          }
        }
      }

public void clk_lower_start() {
            if(busGrabbed){
              Bus.release();
              busGrabbed=false;
          }
      }
```

## Die Kontrollsignale

Um die Kontrollsignale der Ablaufsteuerung des DemoComPlus an die Komponenten zu bringen wurde ein Interface SignalConsumer entworfen. Es enthält zum einen eine Methode um die Kontrollsignale selbst weiterzugeben, aber auch eine Methode um ein bestimmtes Signal an eine bestimmte Funktion einer Komponente zu binden. Alle Komponenten erhalten alle Kontrollsignale, werten aber nur die für sie Relevanten aus.

```
public interface SignalConsumer {
   public void setSignal(boolean[] sig);
   public void linkSignal(int signal, int target);
}
```

Wieder am Beispiel des Lese-Schreibregisters eine Referenzimplementation:

```
public void linkSignal(int signal, int target) {
      if(signal==“schreibsignal“)
         writeenable=target;
      if(signal==“ladesignal)
         loadenable=target;
   }
```

Die Variable target enthält hierbei einen Index auf das auszuwertende Signal. Beispielaufrufe um die Signale zu verbinden:

```
ram.linkSignal(Memory.CE,Control.Ce);
ram.linkSignal(Memory.WE,Control.We);
ir.linkSignal(Register.RG_LOAD,Control.Li);
or.linkSignal(Register.RG_LOAD,Control.Lo);
```

Mit der folgenden Methode wird bei allen Komponenten das Signal übergeben:

```
public void setSignal(boolean[] sig) {
      this.signal=sig;
}
```

## Der Datenbus

Der Datenbus des DemoComPlus wurde ebenfalls als grafische Komponente implementiert, um die Bewegungen von Daten auf dem Bus verständlich darstellen zu können. Die implementierende Klasse ist stark vom Aufbau der Hauptanwendung abhängig und kann daher nur begrenzt in anderen Anwendungen weiterverwendet werden, da die visuelle Komponente entsprechend angepasst werden muss.

Der Bus verfügt über öffentliche Methoden, über die eine Komponente vom Bus lesen kann, um den Bus für den Schreibzugriff öffnen und um ihn wieder zu schliessen.

```
Public void grab(Buswriter r) {
        ...
        }

public void release() {
        ...
        }

public int read() {
        return inhalt;
        }
```

Obwohl die Bus-Klasse relativ umfangreich ist enthält sie ansonsten nur Funktionalität um für die Anzeige zu sorgen.

# Die ALU

Die ALU des JDemoComPlus implementiert das Verhalten und die Operationen der DemoComPlus-ALU, mit dem Unterschied dass die ALU erst dann eine Operation ausführt, wenn sie mit dem Bus verbunden wird, wohingegen die hardwired DemoComPlus-ALU jederzeit rechnet und bei Bedarf das jeweils aktuelle Ergebnis auf den Bus legt.

Erhält die ALU das Signal zur Datenübergabe, so wird ein Teil des aktuellen Inhaltes des Instruction-Register als Befehlscode interpretiert und die entsprechende Operation ausführt, das Ergebnis wird sofort auf den Bus gelegt.

Im Gegensatz zur DemoComPlus-ALU erhält die JDemoComPlus-ALU ihren Befehlscode direkt durch die drei niederwertigsten Bits des aktuellen Opcodes im Instruction-Register, das führt zu einem lesenden Zugriff auf den Instruction-Register beim Durchführen einer Operation, wobei die DemoComPlus-ALU ihren Befehlscode von der Ablaufsteuerung erhält.

Das direkte auslesen des Instruction-Registers hat vor allem den praktischen Hintergrund, dass der Steuerbefehl nicht durch die Ablaufsteuerung weitergereicht werden muss und diese somit einfacher zu implementieren war, da sich weniger Abhängigkeiten zwischen den Klassen ergeben.

Für den Endanwender kommt dieser Unterschied kaum zum Tragen, der einzige Unterschied für den Betrachter ist dass die ALU nur dann eine Aktivität anzeigt wenn sie von Belang ist, und dass in diesem Moment ein Zugriff auf den Instruction-Register erfolgt und angezeigt wird anstatt eines Steuerwortes von der Ablaufsteuerung.

# Die Ablaufsteuerung

Die Ablaufsteuerung des JDemoComPlus entspricht in ihrem Verhalten derer des DemoComPlus, jedoch wurden auch hier intern aus programmiertechnischen Gründen die Abläufe leicht verändert.

Die Ablaufsteuerung durchläuft zwischen 4 (z.B. OUT) und 8 (z.B. ADD) Maschinenzyklen für die Abarbeitung eines Befehls, zusätzliche Wartezyklen nach „kurzen“ Befehlen existieren nicht. Der aktuelle Maschinenzyklus wird von der ABL angezeigt.

Im Grunde entsprechen die nacheinander ausgeführten Schritte denen des DemoComPlus, jedoch wird z.B. bereits in Maschinenzyklus M2 in der Befehl-Holen-Phase der PC inkrementiert.

Die Ablaufsteuerung zeigt die aktuelle Phase der Befehlsausführung an, es wird in der aktuellen Implementation unterschieden zwischen Befehl-holen (fetch), dekodieren (decode) und ausführen (execute), wobei die Ausführen-Phase das holen eventuell benötigter Operanden (operand) und ein eventuelles Zurückschreiben von Ergebnissen (writeback) einschliesst.

Da die hardwired Ablaufsteuerung ohne Zeitverlust dekodiert, ergibt sich ein überlappen der decode-/execute-Phasen, Maschinenzyklus M4 stellt sowohl Decode- als auch Beginn der Execute-Phase dar, was entsprechend angezeigt wird.

Die von der ABL erzeugten Steuerwörter werden über das bereits erläuterte Interface SignalConsumer an die beteiligten Komponenten weitergeleitet, sie entsprechen denen des DemoComPlus und werden angezeigt wenn sie aktiv sind.