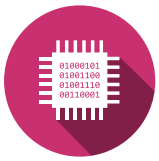


Programzähler

Labor Digitales Design

Inhalt

1 Ziel

Dieses Labor zeigt die Entwicklung eines Programm-Codes mit Hilfe eines Festwertspeichers (Read-Only Memory (ROM)) mit Hilfe der Erstellung eines Programmzählers Program Counter (PC).

Es wird auch das Zeichnen von hierarchischen Schaltkreisen geübt.

2 ROM zur Steuerung der ALU

2.1 Schaltung

Die Abbildung 1 zeigt eine vereinfachte Darstellung eines Prozessors, mit einer Arithmetic and Logical Unit (ALU), Register und einem Programmzähler.

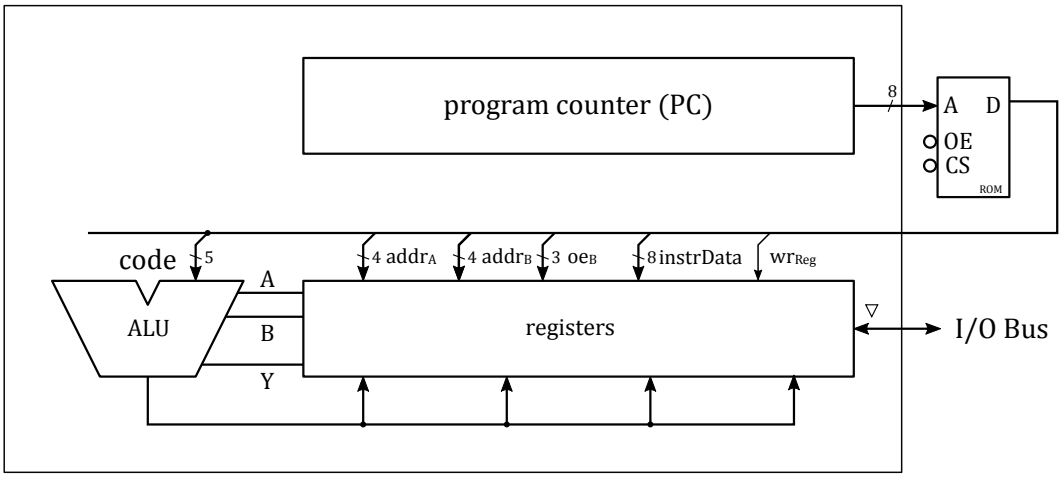


Abbildung 1 - ROM zur Steuerung der ALU und den Registern

Die Adresse der ROM wird von einem Programmzähler erstellt, welcher es erlaubt, den gespeicherten Code sequenziell zu lesen. Die Daten der ROM beinhalten Machine Instructions (MIs) welche aus den Steuersignale der ALU und der Register bestehen.

Der Eingang Output Enable (OE) steuert den hochohmigen Ausgang der ROM. Der Eingang Chip Select (CS) ist das Selektierungssignal der ROM. Beide müssen aktiv sein, damit der Baustein seine Daten am Ausgang bereitstellt.

2.2 Operationssequenz

Die Befehle werden in 2 Phasen durchgeführt, wie in Abbildung 2 dargestellt.

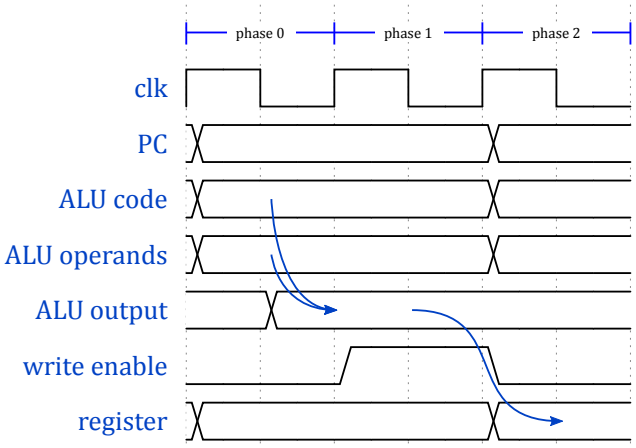
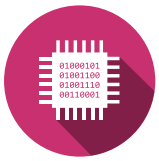


Abbildung 2 - Befehlsphasen

Der ALU Ausgang ist am Ende der Phase 0 stabil. Dieser Wert wird im selektiertem Register bei der steigenden Flanke am Ende der Phase 1 gespeichert.



3 Software-Erstellung eines seriellen Ports

Die Abbildung 3 gibt das zeitliche Verhalten der seriellen Übermittlung eines Datenwortes.

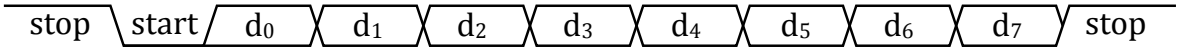


Abbildung 3 - Serielle Übermittlung

Die seriellen Daten werden auf dem niederwertigen Bit (Least Significant Bit (LSB)) des Prozessor-Datenbusses übermittelt.

3.1 Linearer Algorithmus

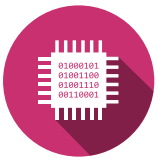
Der in der ROM gespeicherte Assemblerprogrammcode wird in Listing 1 gezeigt.

```
LOAD        s3, FF              ; load stop bit
OUTPUT      s3                  ; output stop bit
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s3, s3              ; no operation
LOAD        s0, 00              ; load start bit
OUTPUT      s0                  ; output start bit
INPUT       s1                  ; load word to send
OUTPUT      s1                  ; output word, LSB is considered
SR0         s1                  ; shift word, bit 1 -> LSB
OUTPUT      s1                  ; output bit 1
SR0         s1                  ; bit 2 -> LSB
OUTPUT      s1                  ; output bit 2
SR0         s1                  ; bit 3 -> LSB
OUTPUT      s1                  ; output bit 3
SR0         s1                  ; bit 4 -> LSB
OUTPUT      s1                  ; output bit 4
SR0         s1                  ; bit 5 -> LSB
OUTPUT      s1                  ; output bit 5
SR0         s1                  ; bit 6 -> LSB
OUTPUT      s1                  ; output bit 6
SR0         s1                  ; bit 7 -> LSB
OUTPUT      s1                  ; output bit 7
LOAD        s3, s3              ; no operation
OUTPUT      s3                  ; output stop bit
```

Listing 1 - Linearer Algorithmus

Im linearen Algorithmus wird eine Anweisung nach der anderen gelesen. Es gibt keine Schleifen oder Sprünge im Programm. Unser erster Programmzähler muss nur in der Lage sein, die Adresse um eins zu erhöhen.

Bevor Sie mit der Realisierung beginnen, stellen Sie sicher, dass Sie Abschnitt 4 gelesen haben, um zu erfahren wie man Blöcke, Komponenten und eine **For**-Schleife erstellt.



3.1.1 Implementierung

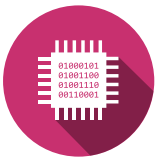
Beginnen Sie mit der Erstellung einer neuen Komponente eines 1-Bit-Zählers **`CNT/cnt_1bit`**. Dieser Zähler sollte bei der steigenden Flanke des **`clocks`** inkrementieren, wenn **`incPC = '1'`**. Beachten Sie, dass in unserem System **`incPC`** bei jeder zweiten Taktperiode aktiviert wird. Ignorieren Sie die Signale **`loadInstrAddress`** und **`instrAddress`** zum Laden eines neuen Wertes in den Zähler. Schliesslich zeichnen Sie einen **`n`**-Bit-Zähler unter Verwendung des **`FOR`**-Rahmens, um die **`n`** Bits des Zählers zu generieren.

- Erstellen Sie eine neue Komponente eines 1-Bit-Zählers **`CNT/cnt_1bit`**.
- Verwenden Sie im Komponenten **`CNT/programCounter`** den **`FOR`**-Rahmen und den **`CNT/cnt_1bit`**, um einen **`n`**-Bit-Zähler zu generieren.

3.1.2 Simulation

Simulieren Sie das System und überprüfen Sie die ordnungsgemässe Funktion des Zählers und des Prozessors.

Überprüfen Sie die ordnungsgemässe Funktion des Zählers, indem Sie den Testbank **`CNT_test/nanoProcess_tb`** mit der Simulationsdatei **`$SIMULATION_DIR/CNT1.do`** verwenden.



3.2 Algorithmus mit Schleifen

Der folgende Algorithmus erlaubt eine kompaktere Schreibweise des Programms, aber es verwendet Schleifen und Sprünge im Programm:

```
1  LOAD    s3, FF              ; load stop bit
2  OUTPUT  s3                  ; output stop bit
3  LOAD    s2, 04              ; initialize loop counter 3
4  SUB     s2, 01              ; decrement loop counter 4
5  JUMP    NZ  03              ; loop back if not end of count 5
6  LOAD    s0, 00              ; load start bit 6
7  OUTPUT  s0                  ; output start bit 7
8  LOAD    s2, 08              ; initialize loop counter 8
9  INPUT   s1                  ; load word to send 9
10 LOAD    s3, s3              ; no operation
11 OUTPUT  s1                  ; output word, LSB is considered
12 SR0     s1                  ; next bit -> LSB
13 SUB     s2, 01              ; decrement loop counter
14 JUMP    NZ  0A              ; loop back if not end of count
15 OUTPUT  s3                  ; output bit 1
```

Listing 2 - Algorithmus mit Schleifen

3.2.1 Erstellung

Um den Algorithmus mit Schlaufen durchzuführen, soll der PC neue Werte laden können. Die zwei Signale **`instrAddr`** und **`loadInstrAddr`** werden benutzt um einen neuen Wert zu laden.

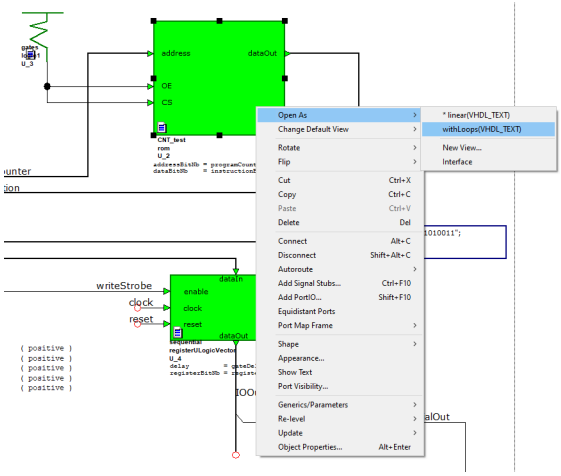
Ändern Sie die hierarchische Schaltung des Programmzählers, um das Laden eines neuen Wertes zu ermöglichen.

Ändern Sie den Programmzähler **`CNT/programCounter`**, um das Laden eines neuen Wertes zu ermöglichen.

3.2.2 Simulation

Simulieren Sie das System und überprüfen Sie den korrekten Betrieb des Zählers und des Prozessors. Ändern Sie die Standardansicht der ROM, um die Version mit Code inklusive Schleifen auszuwählen (**`withLoops`**).

1. **Klicken Sie mit der rechten Maustaste** auf die Komponente,
2. Wählen Sie **Open As**,
3. Klicken Sie auf **withLoops**

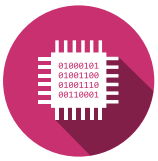


Überprüfen Sie die ordnungsgemässe Funktion des Zählers, indem Sie den Teststand **CNT_test/nanoProcess_tb** mit der Simulationsdatei **$SIMULATION_DIR/CNT2.do** verwenden.

3.3 Vergleich

Vergleichen Sie die beiden Algorithmen hinsichtlich der Übertragungsgeschwindigkeit (Baudrate) und der Codegrösse.

- Vergleichen Sie die Übertragungsgeschwindigkeit (Baudrate) der beiden Algorithmen.
- Vergleichen Sie die Grösse des Codes der beiden Algorithmen in Listing 1 und Listing 2.



4 Blöcke, Komponenten und `For` Schleifen

Im Rahmen des linearen Algorithmus ist der Programmzähler (PC) ein unidirektionaler Zähler. Er kann keinen neuen Wert laden. Sie müssen diesen Zähler so entwerfen, dass er parametrisierbar ist, sodass ein Zähler mit beliebiger Bitgrösse N einfach durch Ändern eines **`generic`**-Parameters erstellt werden kann.

Die folgenden Abschnitte führen Sie durch die Erstellung des Zählers mit Hilfe von **`FOR Generate`** Schleifen und der Erstellung einer wiederverwendbaren Komponente.

4.1 Block erstellen

Um eine neuen Block zu erstellen, müssen Sie ein neues Element hinzufügen, indem Sie auf die Schaltfläche **`Block hinzufügen`** klicken (siehe Abbildung 4).

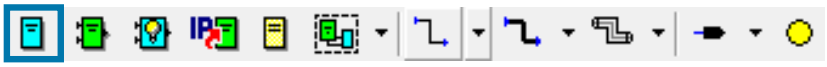


Abbildung 4 - Block hinzufügen

Nachdem Sie auf die Schaltfläche geklickt haben, kann ein neuer Block hinzugefügt werden, indem Sie auf das Diagramm klicken. Sobald der Block hinzugefügt ist, können die benötigten Input/Outputs (I/Os) verbunden werden:

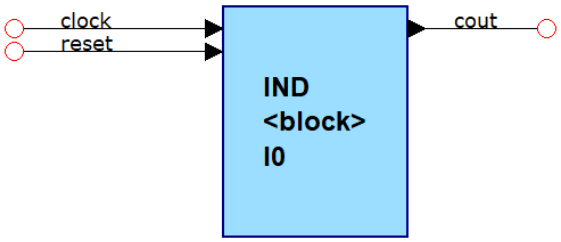
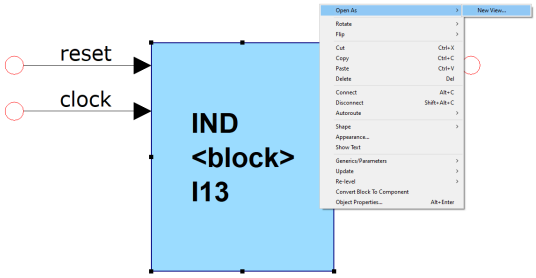


Abbildung 5 - Neuer Block verkabelt

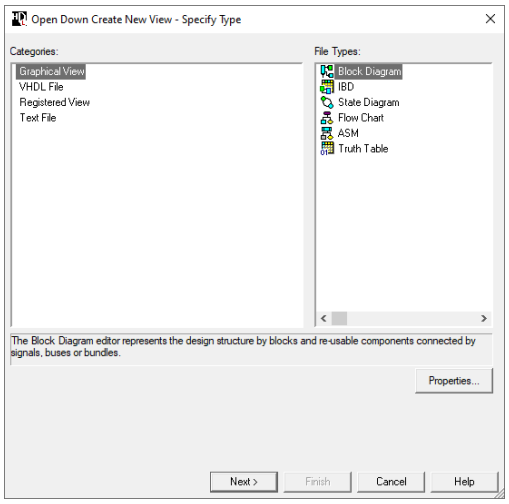
> ⚠ Ein **`blauer Block`** kann nicht kopiert und eingefügt werden, da er nur einmal existiert. Nur der **`grüne Block`** (Komponenten) kann kopiert und eingefügt werden.

Sobald die I/Os verkabelt sind, muss der Blocktyp ausgewählt werden (**`Block Diagram`**/**`State Diagram`**/**`VHDL file`**).

1. **`Klicken Sie mit der rechten Maustaste`** auf den Block
2. Wählen Sie **`Open As`**
3. Klicken Sie auf **`New View...`**



Das folgende Fenster öffnet sich und ermöglicht die Auswahl des zu erstellenden Blocktyps. Wählen Sie den gewünschten Blocktyp aus und klicken Sie auf **`Fertigstellen`**.



`Link`	`Blocktyp`	`Icon`
`Graphical View/Block Diagram`	Blockdiagramm	
`Graphical View/State Diagram`	Zustandsmaschine	
`VHDL File/Architecture`	VHDL-Code	

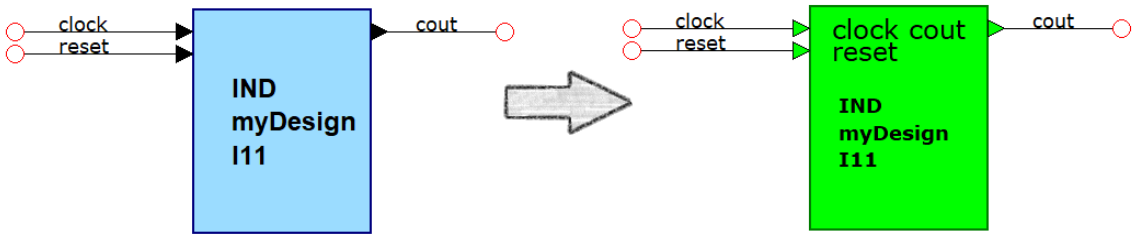
4.1.1 Blockdiagramm

- Wählen Sie **`Graphical View/Block Diagram`**
- Klicken Sie auf **`Next`**
- Geben Sie den Namen des Blocks unter **`Design unit`** ein
- Füllen Sie die Tabelle der I/Os aus
  - Stellen Sie sicher, dass die Typen korrekt sind
  - Definieren Sie die Grenzen für die Multi-Bit-Typen
- Klicken Sie auf „Finish“

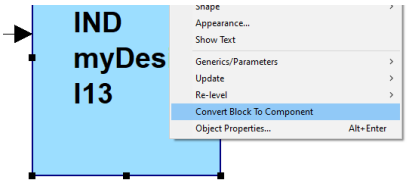
> Die I/Os können immer noch hinzugefügt, entfernt und geändert werden, wenn das Diagramm bearbeitet wird.

4.2 Block in Komponente umwandeln (blau zu grün)

Das Umwandeln des Blocks in eine Komponente (von blau zu grün) ermöglicht das Kopieren und Einfügen des Elements und macht ihn in der Projektbibliothek verfügbar. Blaue Blöcke sind ideal, um schnell eine Blockschnittstelle zu erstellen, während grüne Komponenten unerlässlich sind, um das Element an anderer Stelle im Projekt wiederzuverwenden.



1. **`Klicken Sie mit der rechten Maustaste`** auf den blauen Block
2. Klicken Sie auf **`Convert Block To Component`**

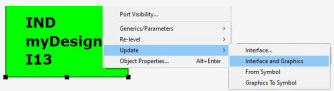


4.3 Schnittstelle der Komponente aktualisieren

Die Schnittstelle einer Komponente besteht aus mehreren Teilen: **`Eingänge`** und **`Ausgänge`**, **`Generics`** und **`Symbol`**. Diese verschiedenen Parameter können hinzugefügt, entfernt und geändert werden.

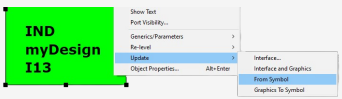
`Sobald die Schnittstelle der Komponente geändert wurde`

1. **`Klicken Sie mit der rechten Maustaste`** auf die Komponente
2. Wählen Sie **`Update`**
3. Klicken Sie auf **`Interface and Graphics`**



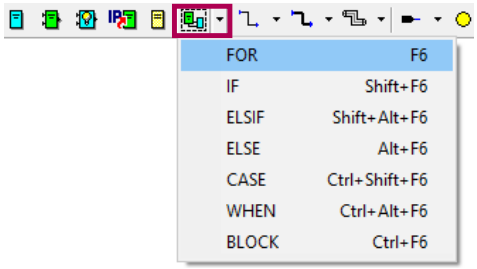
---

1. **`Klicken Sie mit der rechten Maustaste`** auf die Komponente
2. Wählen Sie **`Update`**
3. Klicken Sie auf **`From Symbol`**

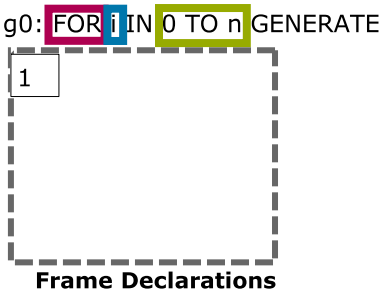


4.4 `For Generate`

Der **`FOR Generate`**-Rahmen ermöglicht es, mehrere Iterationen der gleichen Struktur zu erstellen. Um einen neuen Rahmen hinzuzufügen, klicken Sie auf die Schaltfläche **`add frame`** und wählen Sie **`FOR`**.



Wenn Sie einen **FOR**-Rahmen hinzufügen, wird das folgende Element im Diagramm gezeichnet.

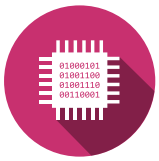


Part	Description
FOR	Typ des Blocks (**FOR**-Schleife)
i	Index der Schleife
0 TO n	Bereich der **FOR**-Schleife

Alle Signale und Komponenten innerhalb des **FOR**-Rahmens werden **n+1** mal kopiert. Der Index **i** geht von **0** bis **n**. *Ändern Sie n in eine Konstante oder ein* **generic**. Alle Signale innerhalb des **FOR**-Rahmens, die in allen Kopien eindeutig sind, sollten den Index **i** in ihrem Slice/Index verwenden.

Zum Beispiel: Der **clock** sollte überall gleich sein; aber ein Zähler sollte die einzelenen Bits b herausgeben. Zuerst **bit[0]**, dann **bit[1]**, dann **bit[2]** ... **bit[i]**.

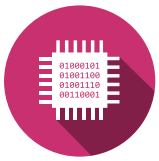
- Nur Signale und Komponenten (grüne Elemente) können in einer **FOR**-Schleife platziert werden.
- Es dürfen sich keine **In-Ports** oder **Out-Ports** innerhalb des Rahmens befinden.
- Es dürfen keine Blöcke (blaue Elemente) innerhalb der Rahmens platziert werden.



5 Checkout

Dies ist das Ende des Labors, Sie haben erfolgreich einen Programcounter implementiert welcher alle Instruktionen des μProzessors unterstützt. Bevor Sie das Labor verlassen, stellen Sie sicher, dass Sie die folgenden Aufgaben erledigt haben:

- ☐ Schaltungsentwurf
  - ☐ Der 1-Bit-Zähler Block **`CNT/cnt_1bit`** wurde erstellt.
  - ☐ Der Rahmen **`FOR`** wurde verwendet, um den 1-Bit Zähler Block zu duplizieren.
  - ☐ Der **`n`**-Bit-Zähler wurde erstellt.
  - ☐ Der Zähler wurde angepasst, um das Laden eines neuen Wertes zu ermöglichen.
- ☐ Simulationen
  - ☐ Die 2 Zähler wurden erfolgreich mit det Testbench **`CNT_test/nanoProcess_tb`** und den Simulationsdateien **`$SIMULATION_DIR/CNT1.do`** und **`$SIMULATION_DIR/CNT2.do`** getestet.
- ☐ Vergleich
  - ☐ Die Übertragungsgeschwindigkeit (baudrate) der beiden Algorithmen wurde verglichen.
  - ☐ Die Codegrösse der beiden Algorithmen wurde verglichen.
- ☐ Dokumentation und Projektdateien
  - ☐ Stellen Sie sicher, dass alle Schritte (Entwurf, Simulationen, Vergleich) gut in Ihrem Laborbericht dokumentiert sind.
  - ☐ Speichern Sie das Projekt auf einem USB-Stick oder im gemeinsamen Netzwerkordner (**`\\filer01.hevs.ch`**).
  - ☐ Teilen Sie die Dateien mit Ihrem Laborpartner, um die Kontinuität der Arbeit sicherzustellen.



Glossar

***ALU* – Arithmetic and Logical Unit** 2, 2, 2, 2

***CS* – Chip Select** 2

***I/O* – Input/Output** 7, 7, 8, 8

***LSB* – Least Significant Bit** 3

***MI* – Machine Instruction**: A machine instruction is a binary-coded operation that a processor can execute directly. It typically consists of an opcode, operands and immediates. 2

***OE* – Output Enable** 2

***PC* – Program Counter** 1, 5

***ROM* – Read-Only Memory** 1, 2, 2, 2, 2, 2, 3, 5