Übung 3

> *Kurs: Parallele und verteilte eingebettete Systeme*
> *SS 2017*
> *VAK: 03-ME-712.06*
> *Veranstalter: Dr. Stefan Bosse*

Aufgabe 1: Matrixmultiplikation mit ConPro

Lesen sie die Folien im Abschnitt F9-F15 und Abschnitt J, insbesondere J7, J22-J31.

1. Implementieren Sie den sequenziellen Algorithmus und die verschiedenen parallelen Partitionierungen der Matrixmultiplikation in ConPro.

- A.) Sequenzieller Fall

```
VAR A,B,C: ARRAY [1 TO 10,1 TO 10] OF FLOAT;
FOR J = 1 TO 10 DO
 FOR I = 1 TO 10 DO
        FOR K = 1 TO 10 DO
         C[I,J] := A[I,K] * B[K,J];
        DONE
 DONE
DONE
```

- B.) Parallele Partitionierung Fall 1.

```
Object barrier: semaphore;
Process VE[i,j]:
 FOR k := 1 to q DO
        C[i,j] ? C[i,j] + A[i,k] • B[k,j];
 END FOR k;
 barrier.up();

Process MAIN:
 barrier.init(0);
 FOR ALL i,j := 1 to p,1 to r DO START Process VE[i,j];
 FOR N : = 1 to (p*r) DO barrier.down();
```

- C.) Parallele Partitionierung Fall 2.

```
Object barrier: semaphore;
Process VE[j]:
 FOR i := 1 to p DO
        FOR k := 1 to q DO
         C[i,j] ? C[i,j] + A[i,k] • B[k,j];
        END FOR k;
 END FOR i;
 barrier.up();

Process MAIN:
 barrier.init(0);
 FOR j := 1 to r DO START Process VE[j];
 FOR N := 1 to r DO barrier.down();
```

- D.) Parallele Partitionierung Fall 3.

```
Object barrier, barrier_t[p,q]: semaphore;
Process VE[i,j,k]:
 t[i,j,k] + A[i,k] • B[k,j];
 barrier_t[i,j].up();
-> Ein VE[i,j,k] ist Master
Process VE[i,j] ? V[i,j,k]:
 barrier[i,j].init(0);
 FOR N = 1 to q DO
        barrier_t[i,j].down();
 END FOR N;
 FOR k = 1 to q DO
        C[i,j] ? C[i,j] + t[i,j,k];
 END FOR k;
 barrier.up();
Process MAIN:
 barrier.init(0);
 FOR ALL i,j,k := 1 to p, 1 to q, 1 to r DO
 START Process VE[i,j,k];
 FOR N := 1 to (p*r) DO barrier.down();
```

- Benutzen Sie zunächst folgendes RAM Modell: `array A,B,C: var[16,16] of int[8]`, und folgendes ConPro Template:

```
open Core;
open Process;
open System;
open Reset;
open Ioport;
object sys : system;
sys.clock  (100000 kilohz);
sys.reset_level (0);
sys.reset_internal(1);

signal SW: logic[4];
reg LED: logic[8];

export LED,SW;

array A,B,C: var[16,16] of int[8];
-- array A,B,C: reg[16,16] of int[8];

process mult:
begin
  for i = 0 to 15 do
  begin
    for j = 0 to 15 do
    begin
      C.[i,j] <- 0;
      for k = 0 to 15 do
      begin
        C.[i,j] <- C.[i,j] + A.[i,k] * B.[k,j];
            end;
    end;
  end;
end;

process main:
begin
  reg sum:logic[8];
  reg temp:int[8];
  LED <- 0;
  for i = 0 to 15 do
  begin
    for j = 0 to 15 do
    begin
      A.[i,j] <- i+j;
      B.[i,j] <- i+j;
      C.[i,j] <- 0;
    end;
  end;
  mult.call();
  sum <- 0;
  for i = 0 to 15 do
  begin
    for j = 0 to 15 do
    begin
      temp <- C.[i,j];
      sum <- sum + to_logic(temp);
    end;
  end;
  LED <- sum;
end;
```

ConPro für Fall A

2. Synthetisieren Sie mittels ConPro die verschiedenen Designs zu VHDL und führen schließlich die Gate-level Synthese mit Xilinx ISE durch. Welcher Ressourcenbedarf ergibt sich für die verschiedenen Designs (FF Latches, FPGA Coverage, Cells).

- Dazu suchen Sie aus dem Xilinx ISE und dem Place&Route Bericht folgende Zeilen heraus, die zusammengefasst auch im Design Overview / Summary tabellarisch gelistet sind:
- Xilinix Synthesize Log:

  Advanced HDL Synthesis Report:

```
# RAMs
# Registers
Finale Register Report:
Flip-Flops : xxx
```

- Xilinx Implement design Log:

  Device utilization Summary:

```
Number of Slice Registers : xxx
Number of Slice LUT : xxx
```

- Design Summary

```
Device Utilization Summary:
Number of Slice Registers
Number of used as Flip FLops
Number of Slice LUTs : xxx / yyy (zz %)
```

- Welche Unterschiede ergeben sich für die verschiedenen Fälle A-D?
- Wie ist jeweils das Verhältnis Register / Kombinatorischer Logik?

3. Implementieren Sie die Matrizen A,B, und C mit folgenden Hardwareobjekten und vergleichen Sie den Ressourcenbedarf aus der Gate-level Synthese für die Fälle A-D mit:
   - `array A,B,C: var[16,16] of int[8]` (Drei RAMs, siehe Aufgabe 2)
   - `array A,B,C: reg[16,16] of int[8]` (256 Register)
   - `block ram; array A,B,C: var[16,16] of int[8] in ram` (Ein RAM)