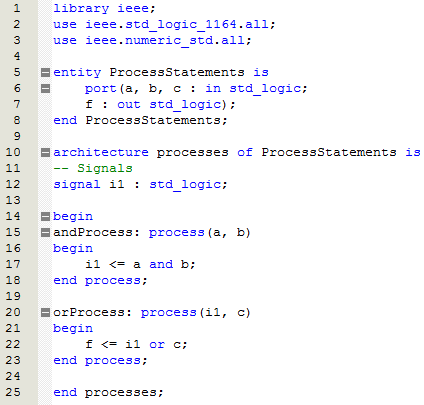
Exercise 5

## Process statements

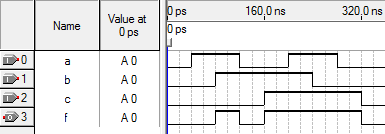
**1) Behavioural style beskrivelse til logisk kredsløb:**

Herunder ses koden til det logiske kredsløb:

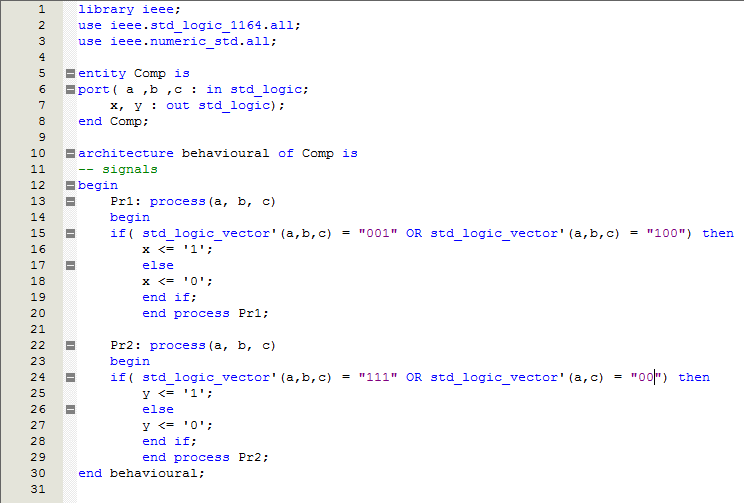


**2) Udførelse af funktionel simulation:**

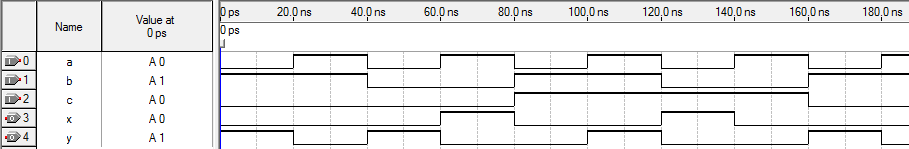
Herunder ses et udklip af den funktionelle simulation:



**3) Skriv kode til komponent, der opfylder sandhedstabel ved brug af to processer:**

****

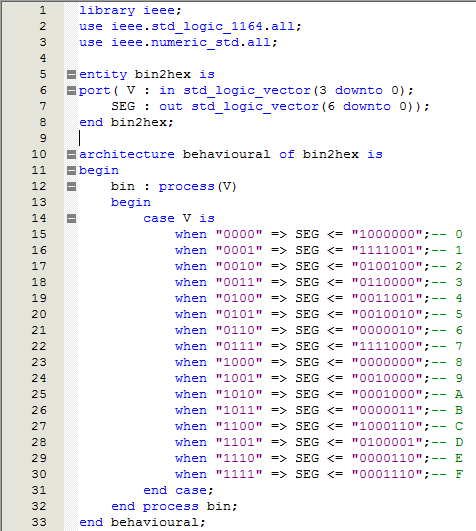
**4) Udfør en funktionel simulation:**

****

## Binary to 7-segment hexadecimal – with case statement

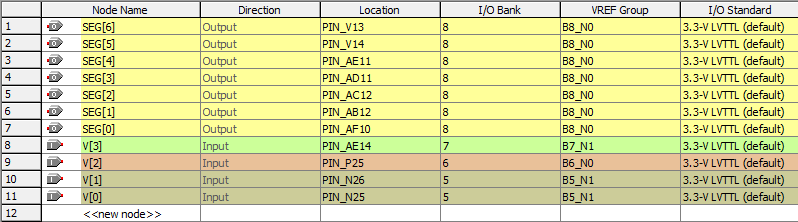
**1) Design en binær 7-segment hexadecimal converter vha. case-statements:**

Vi har designet vha. case-statements og fået følgende:



**2) Downloading og test på DE2 board:**

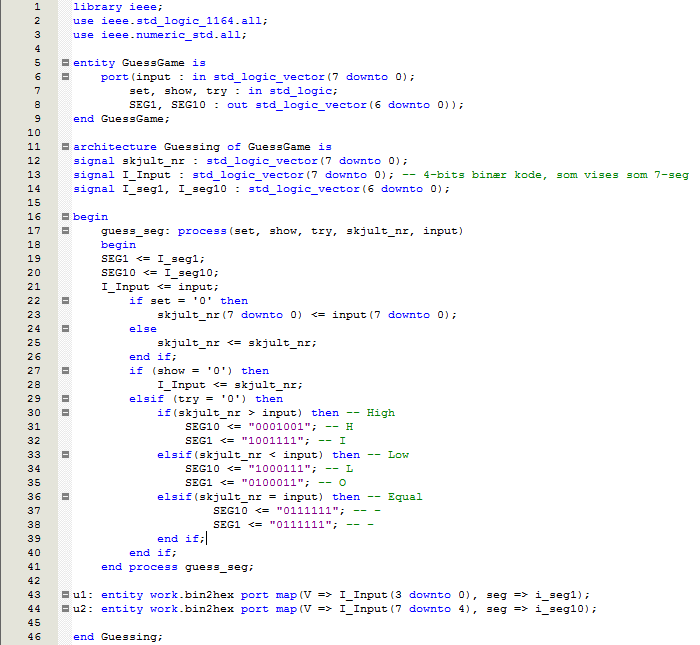
Vi har eftertestet dette og sat følgende pins:



## Guess a HEX number game

**1) Design spil med bestemte funktioner (jf. opgavebeskrivelsen):**

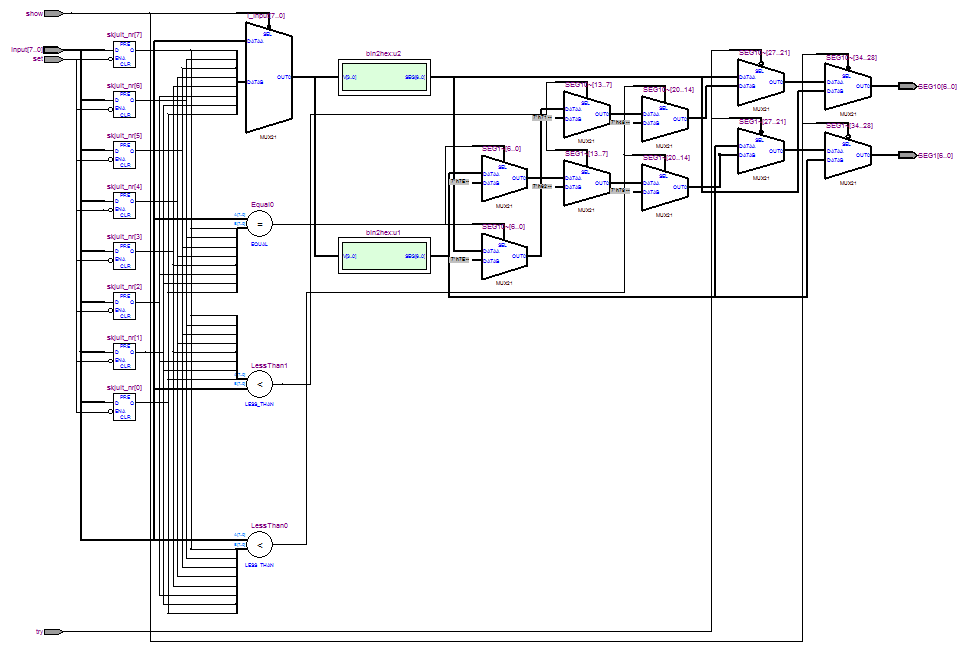
I denne opgave anvender vi entiteten bin2hex:



Det er herefter testet på DE2-boardet med de angivne pin-settings jf. vejledningen. Vi fik de forventede resultater efter flere test med spillet.

**2) Download og test designet, evt. latches i RTL:**

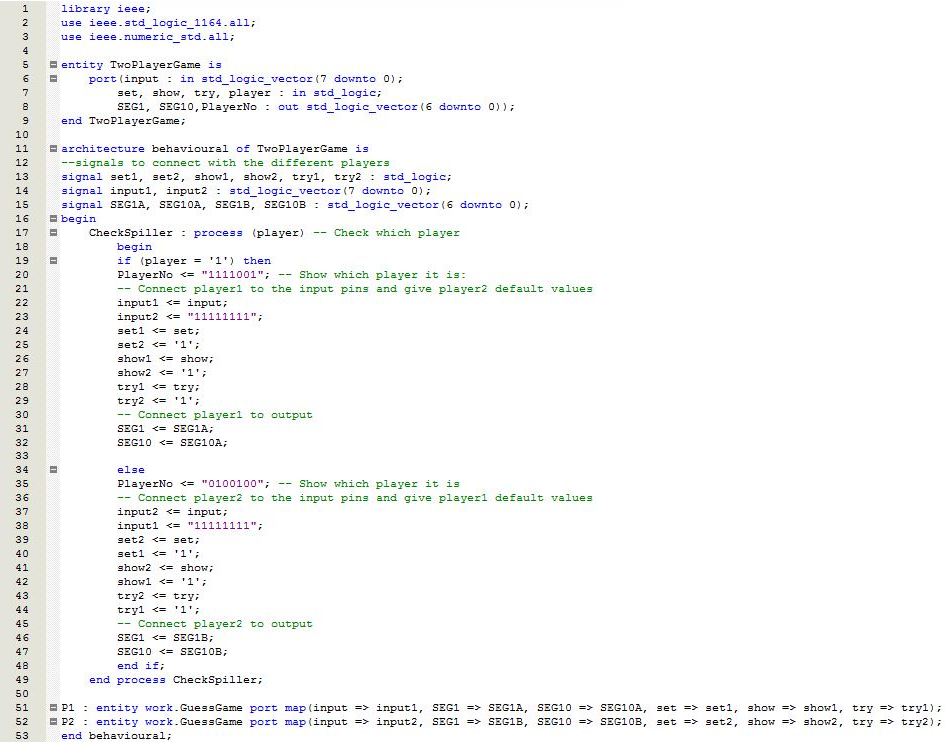
Vi har først indsat det i RTL, hvilket kan ses herunder:



Det fremgår af figuren, at der er latches, og det fremkom også, da man skulle tjekke syntaksen for koden. De dannes, således at det nummer, man vil gætte, kan gemmes.

**3) Udvidelse af designet til to spillere:**

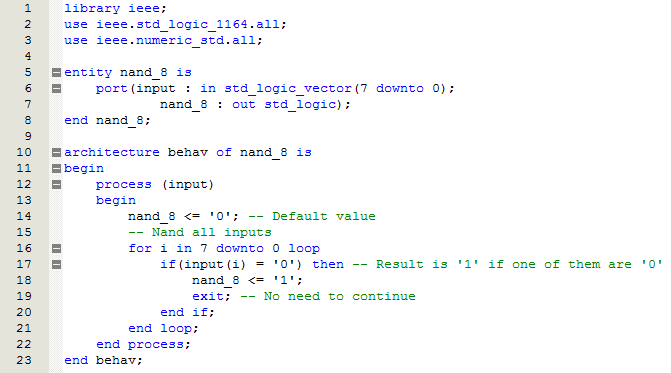
Vi har anvendt GuessGame-entiteten til dette spil, hvor vi har fået designet en relativ lang kode. Vi har derefter testet det på DE2-boardet og fået de forventede resultater.



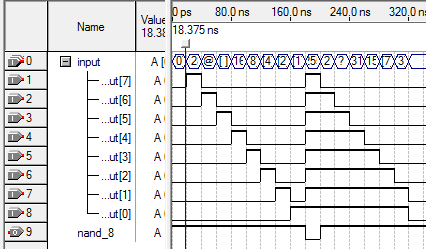
## 8 input NAND – using the for loop

**1) Skriv koden til en behavioural komponent, der bruger for-loop til at beskrive en otte input gate, og udfør en funktionel simulation:**

Her er koden til komponenten:

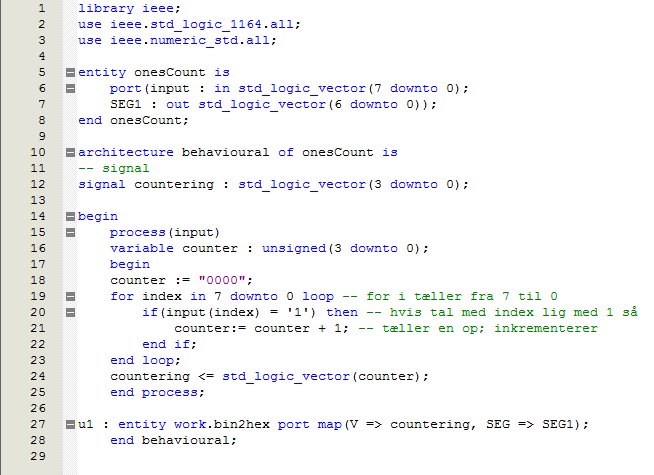


Her er et udklip til den funktionelle simulation:

****

## Count ’1’ – using the for loop

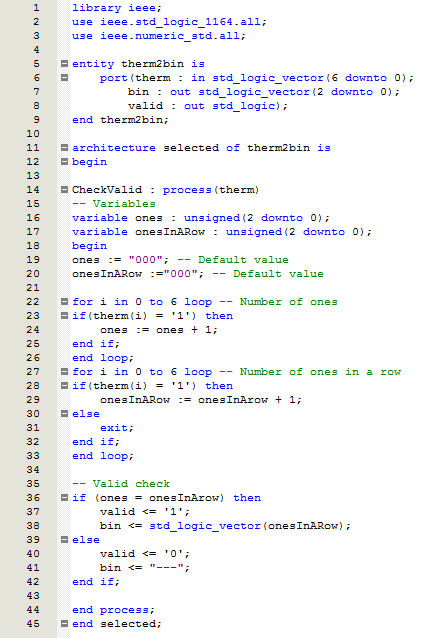
**1) Skriv koden til en behavioural component, der bruger en for-loop til at tælle antallet af ‘1’, der fremkommer ved input og bliver sat på 7-segmentets output, og test derefter designet:**



Vi har eftertestet det og har fået det forventede resultat, hvor den går igennem alle inputs i for-løkken og tæller alle de høje i en sum, som bliver vist på 7-segment-displayet.

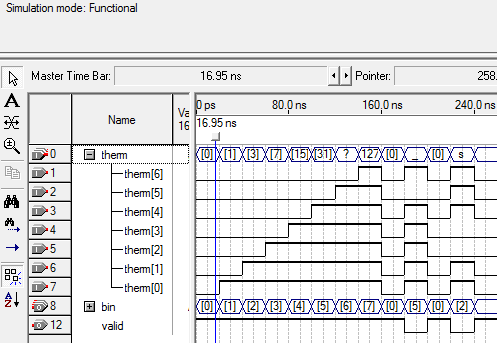
## Thermometer coding - optional

**1) Design komponent, der bruger loop-statements til at implementere et termometer til binær encoder:**



**2) Udfør funktionel simulation og test på DE2 board:**

Herunder ses et udklip af den funktionelle simulation:



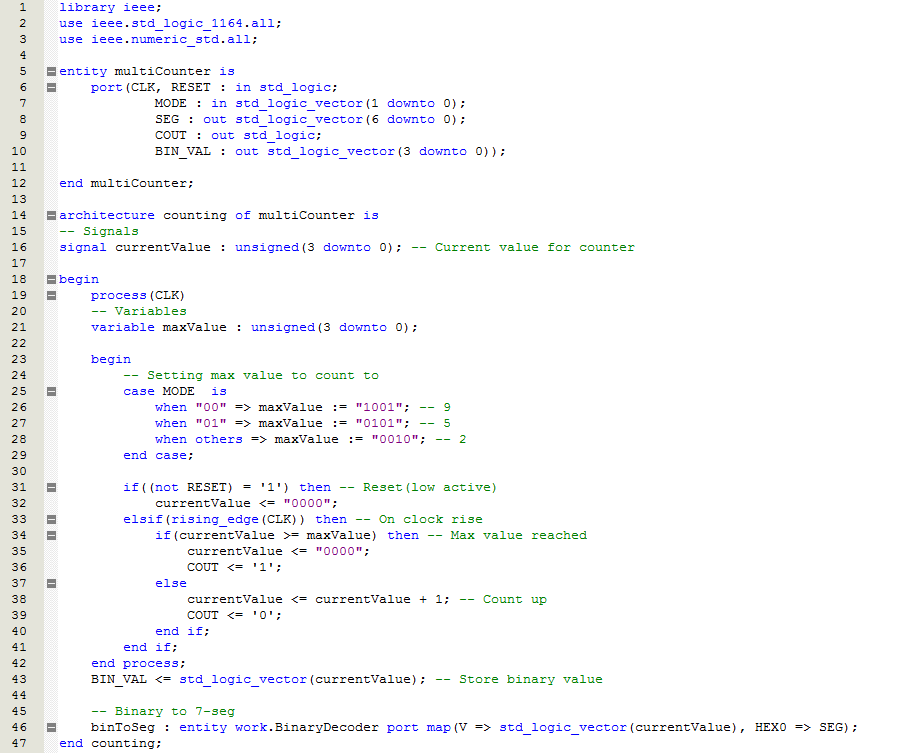
Vi har eftertestet det og fået det forventede resultat.

Exercise 6

## Counter – one digit

**1) Design binær cirkulær tæller, og brug 7-segment dekoder fra tidligere øvelse:**

Vi har brugt vores 7-segment dekoder fra tidligere øvelse til denne løsning:



**2) Download og test designet på DE2 board:**

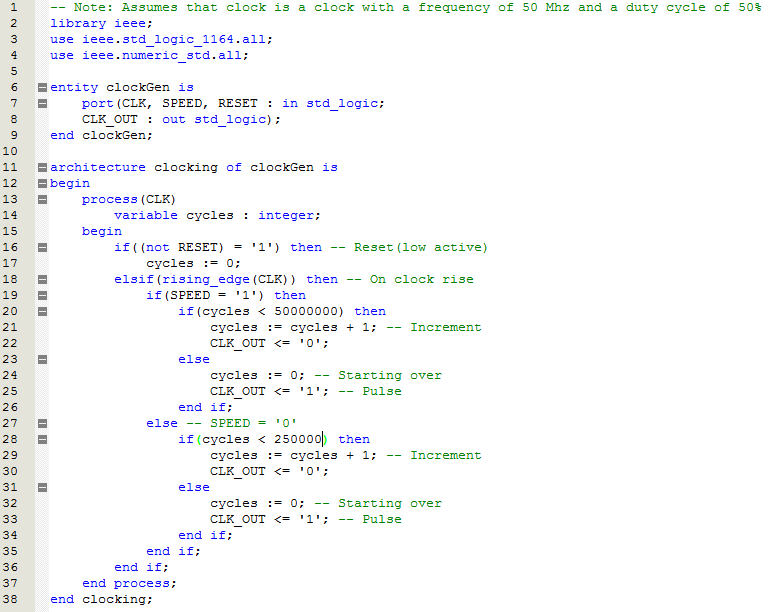
Vi har eftertestet det og fået det forventede resultat – vi har sat HEX0 som vores 7-segment.

## Watch – six digits

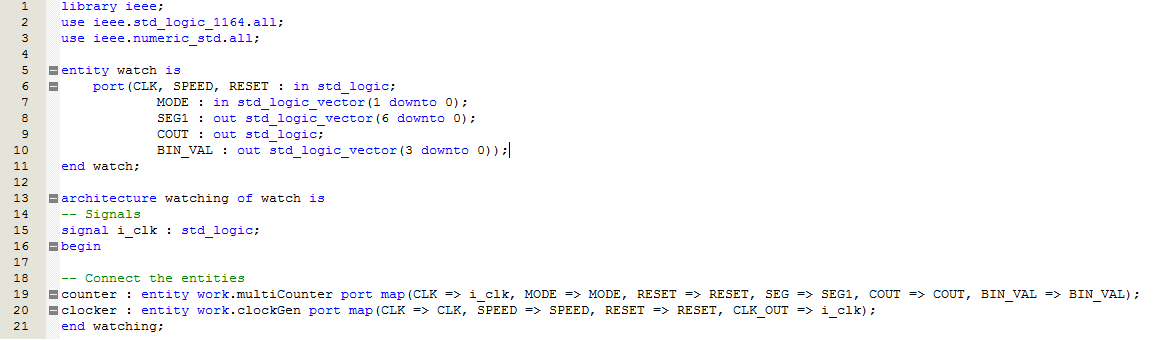
**1) Design en ny komponent med clockGen komponent, der genererer en clock pulsbredde pr. sekund, når SPEED er ‘1’ og 5 ms når SPEED er ’0’:**

Vi anvender entiteten fra tidligere øvelser af hhv. vores 7-segment dekoder og multiCounter til entiteten ’watch’:

Designet for clockGen:



Designet for watch:

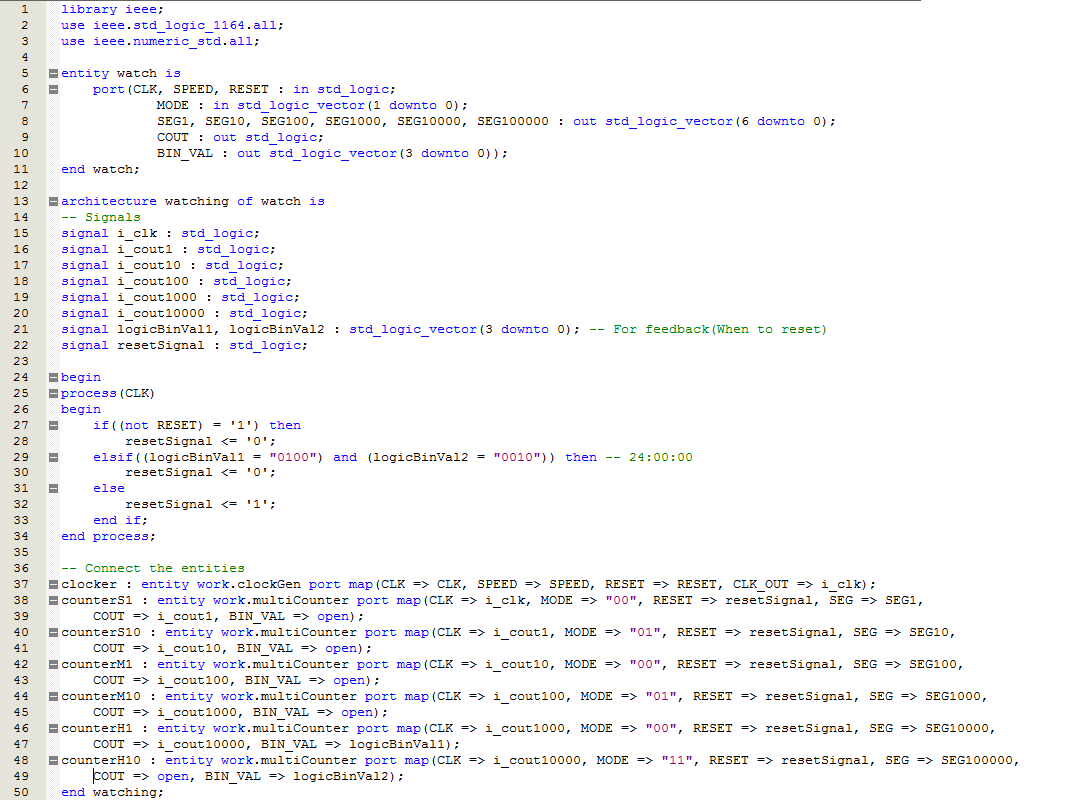


**2) Download og test designet på DE2 board:**

Vi har eftertestet det og fået det forventede resultat – man kan se den ”blinke” for SW[16], men ikke for SW[17], da det går for hurtigt (høj frekvens).

**3) Udvid designet, så det kan tage op til et seks cifret 24-timers ur:**

Designet er udvidet fra tidligere opgave, og derfor hedder entiteten stadig ’watch’:



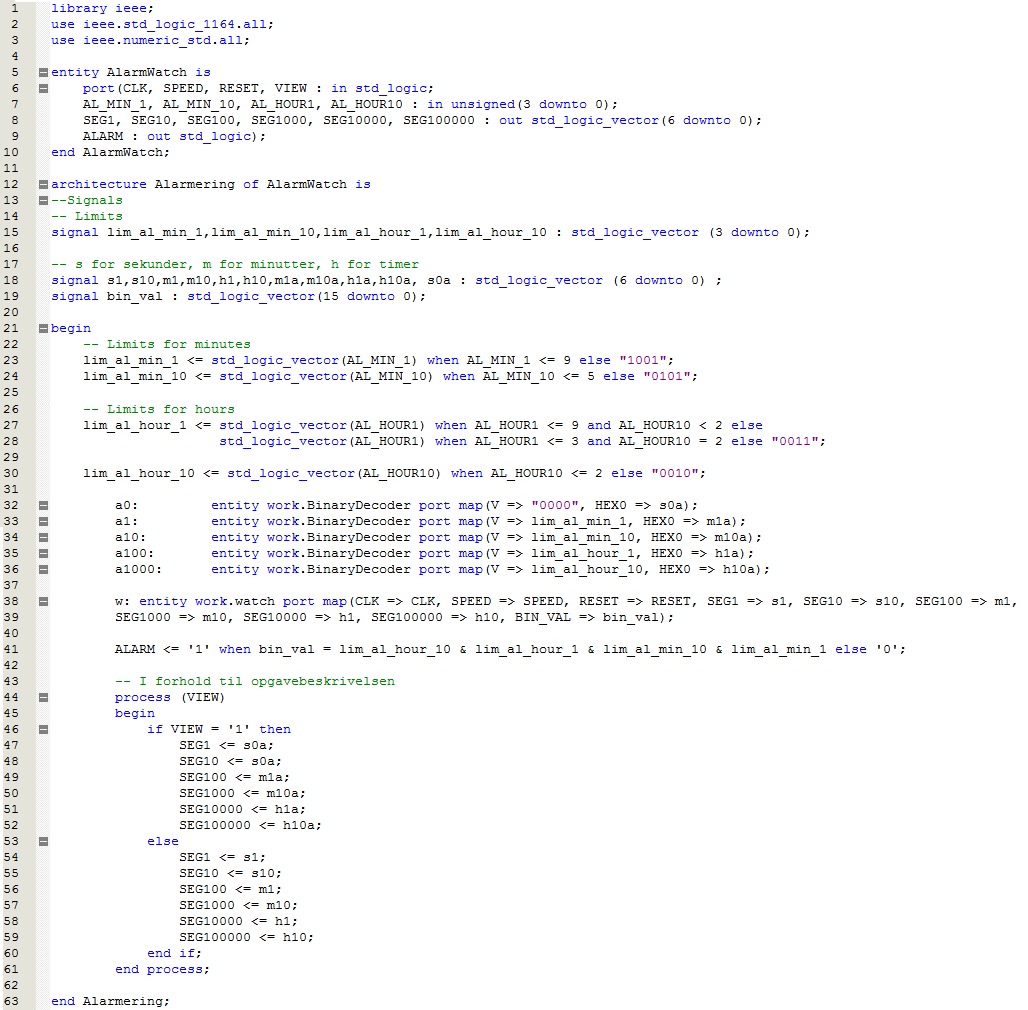
**4) Download og test designet på DE2 board:**

Vi har eftertestet det og fået det forventede resultat – vi ændrede på frekvensen for at se hurtigere på, hvad der sker, når man når 24:00:00, men ændrede det tilbage i koden efter testen på DE2 boardet. Her gik den blot direkte til 00:00:00 og startede med at tælle forfra.

## Alarm watch

**1) Udvid watch-komponenten med en alarm-funktion, så kun timer og minutter kan blive sat til alarmering:**

Entitet fra tidligere øvelser og denne øvelse er blevet brugt til at lave dette design – denne er også udvidet fra ’watch’-entiteten, og vi har fjernet inputtet ’MODE’ og kaldt entiteten for ’AlarmWatch’:



**2) Download og test designet på DE2 board:**

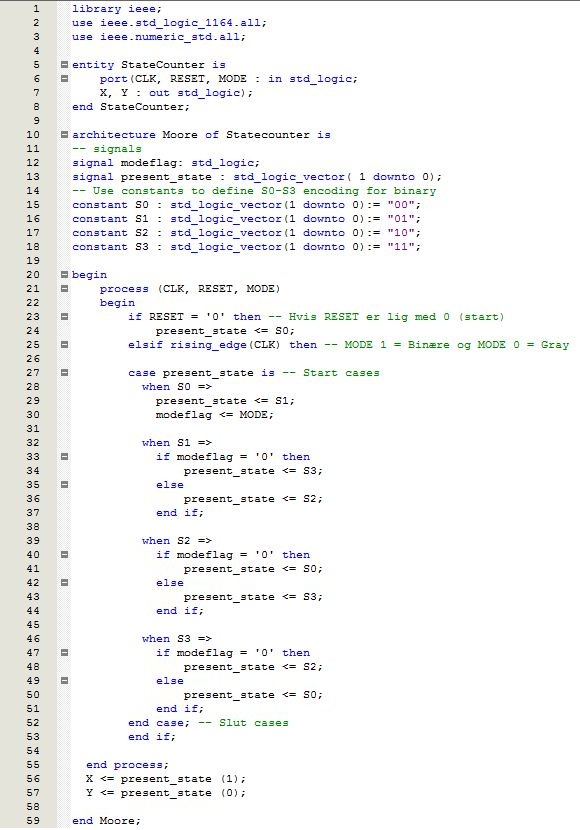
Vi fik det testet og fik det forventede resultat – Det fungerede således, at når vi havde sat den til en bestemt tid, så ville dioden LEDR0 lyse i et minut, når tiden var ens, hvorefter den slukker, men tiden kører videre.

Exercise 7

## State counter – simple Moore type

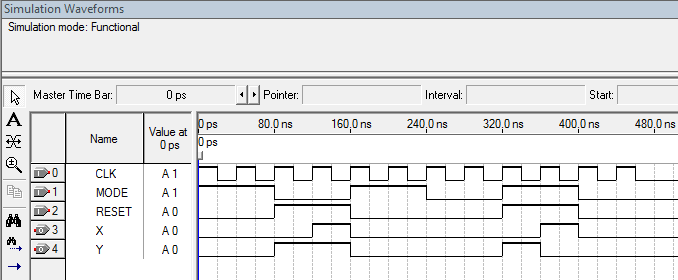
**1) Design en kontrolleret to bit binær/graycode counter:**

Herunder ses vores design:

****

**2) Udfør en funktionel simulation og test det på DE2 board:**

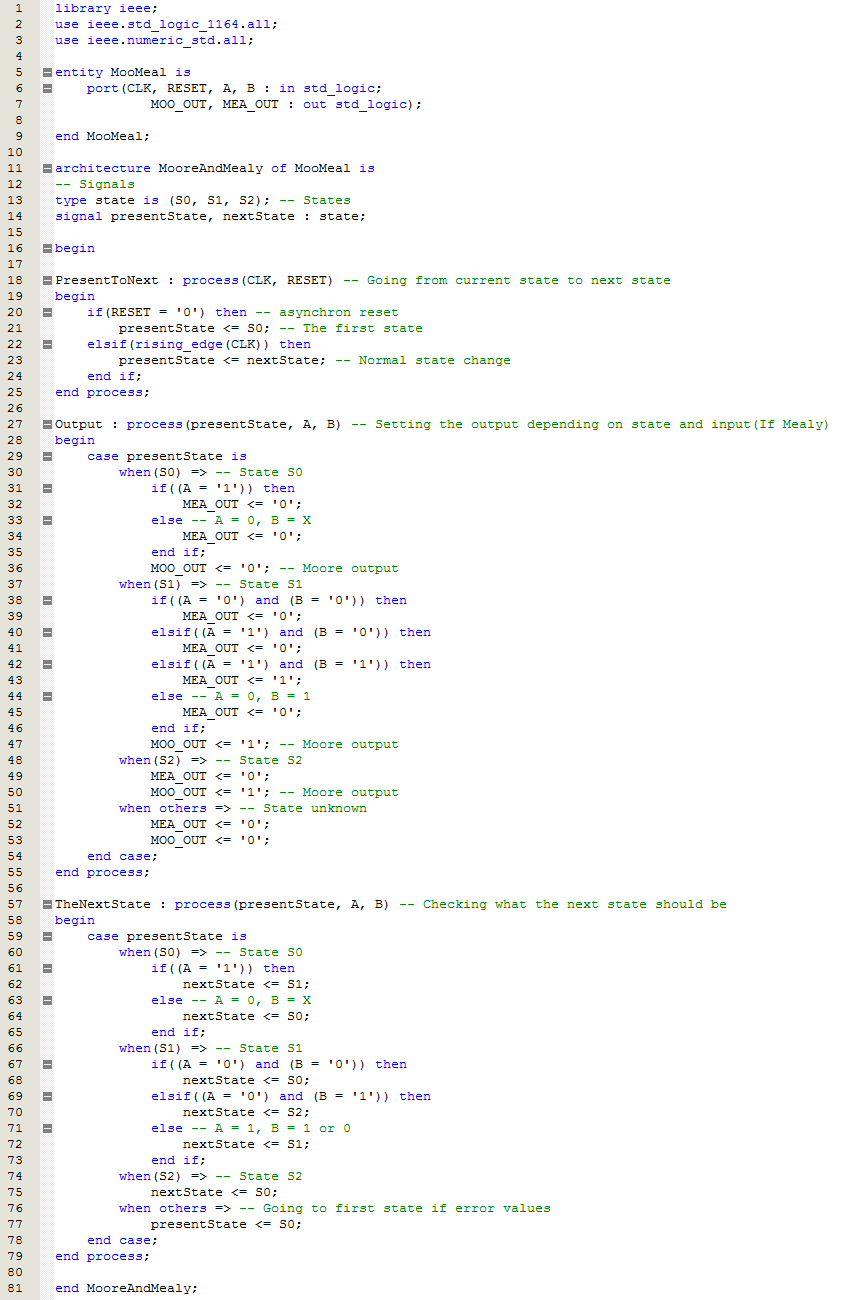
Den funktionelle simulation kan ses herunder:



Vi fik testet det og fik det forventede resultat.

## State diagram – all included

**1) Implementér state-diagrammet ved at bruge tre processer:**

****

**2) Download og test det på DE2 board:**

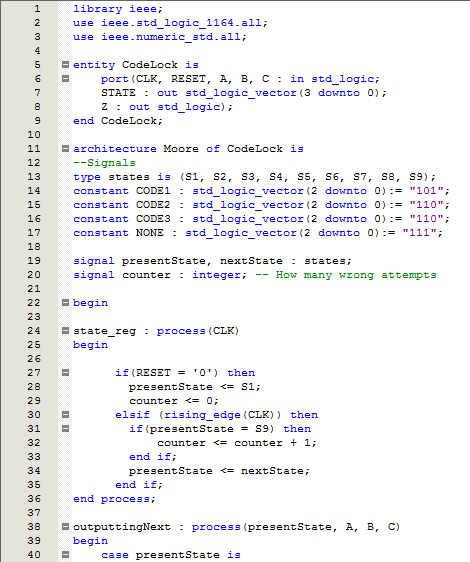
Vi fik testet det og fik det forventede resultat.

## Code lock

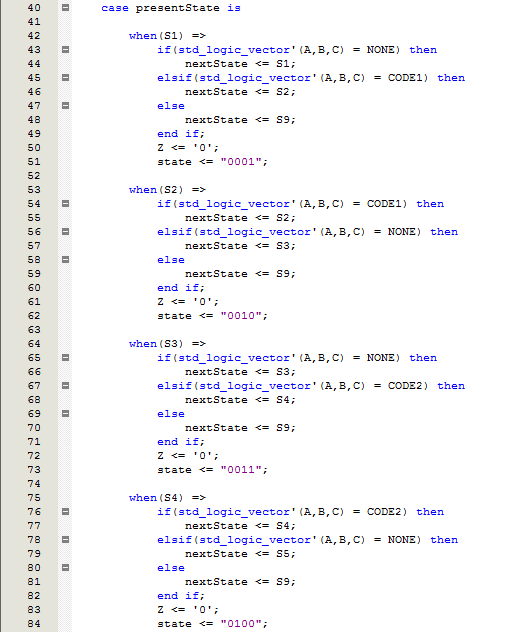
**1) Lav en state machine ved at bruge to processer og download det på DE2 board og test det:**

Herunder ses koden for de to processer opdelt i flere bider:

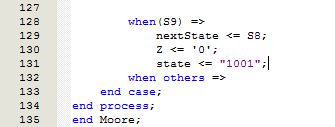
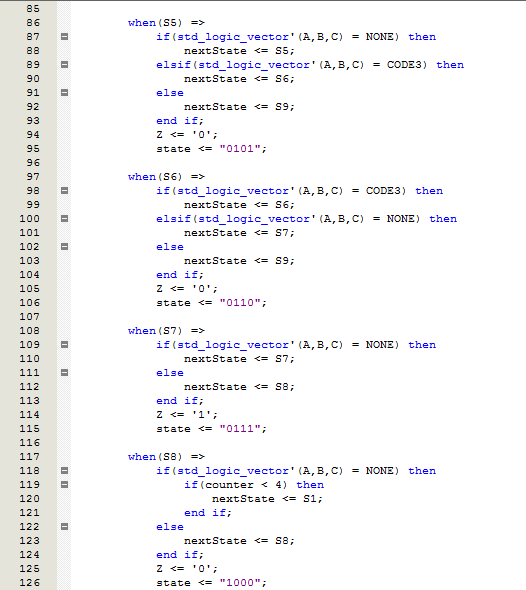
Del 1 af koden:



Del 2 af koden:



Del 3 af koden:



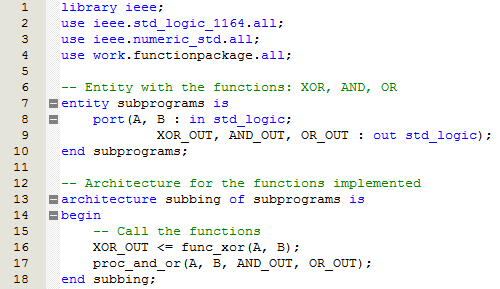
Vi fik herefter testet det på DE2-boardet og nåede frem til det forventede resultat. Vi har sat safe-state-setting på for at undgå, at man kommer ind i en udefineret state.

Exercise 8

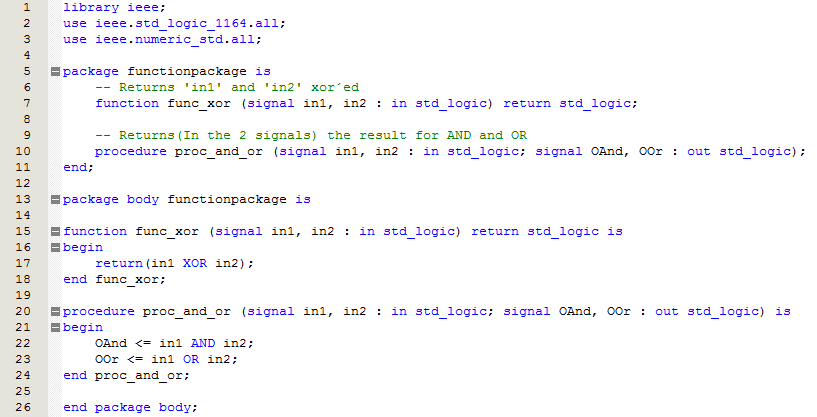
## A simple procedure and function

**1) Design komponenten angivet i øvelsesvejledningen og download det på DE2 boardet:**

Herunder ses koden for subprogram, som bruger funktionspakken:



Herunder ses koden for package:



**2) Gør procedurerne og funktionerne synlige for alle komponenter i projektet og download og test det nye design på DE2 boardet:**

Procedurerne og funktionerne er blevet gjort synlige, hvilket ses i koderne i punkt 1. Vi testede det på DE2 boardet og fik det forventede resultat.

## The Italian flag – on VGA

**1) Skriv en kort beskrivelse af skabelon-koden i projektet – kilder om VGA findes på campusnet:**

Skabelon-koden består af flere vigtige dele, og heraf er der f.eks. konstanter, der afgrænser billedets position og størrelse. De forskellige størrelser er oplyst i DE2\_usermanualen s. 39 i figur 4.13 horisontalt og 4.14 vertikalt. Endvidere er der den clock generator, der laver den rigtige frekvens.

Der er ’syncGenerator’, som står for at holde styr på:

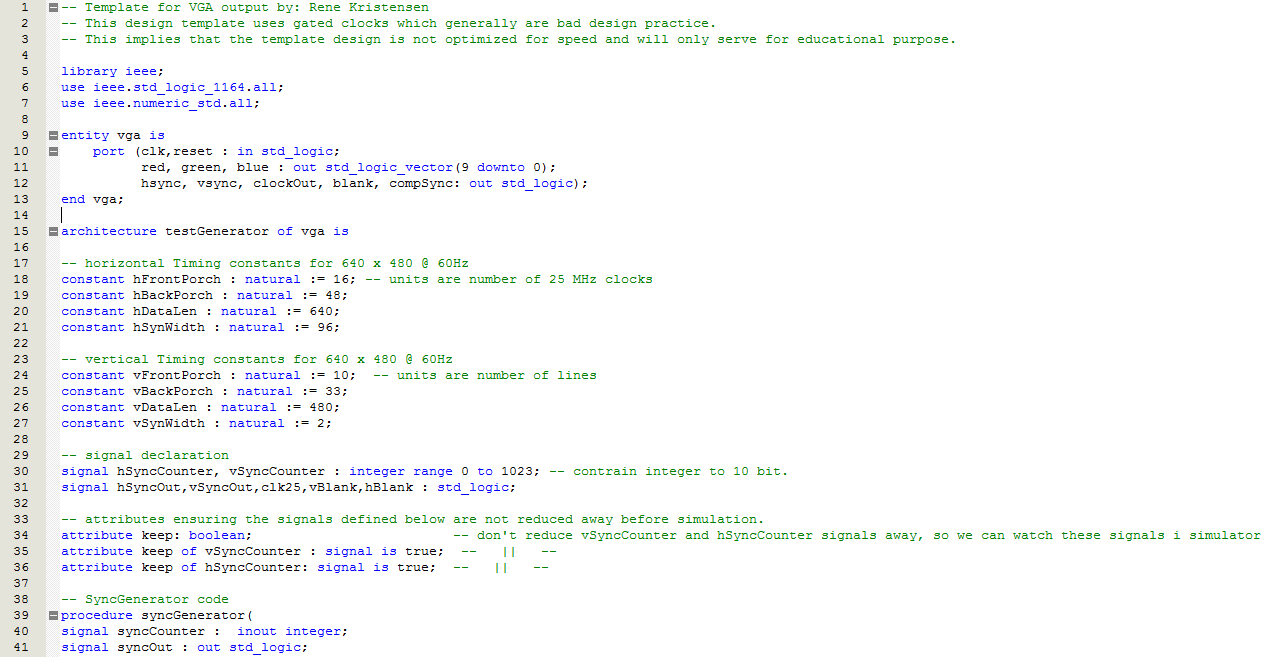
* Om sync counteren er max og skal resettes eller bare tælles op.
* Om der skal være blank out.
* Om der skal være sync out.

Selvfølgelig skal den der del, som står for at udskrive bestemte pixels ud fra, hvor på skærmen vi er ved at skrive også lige nævnes.

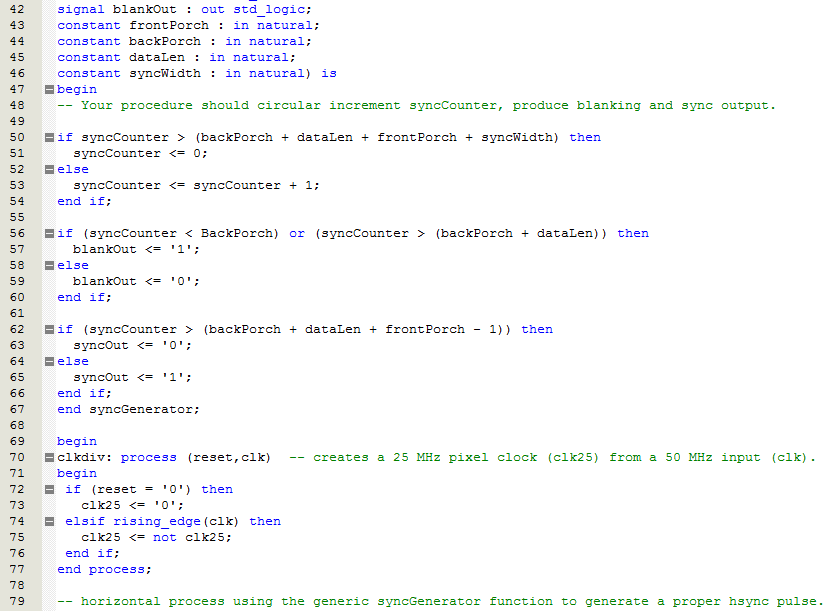
**2) Design den manglende procedure syncGenerator kaldet af to processer:**

Denne kode er opdelt i tre dele:

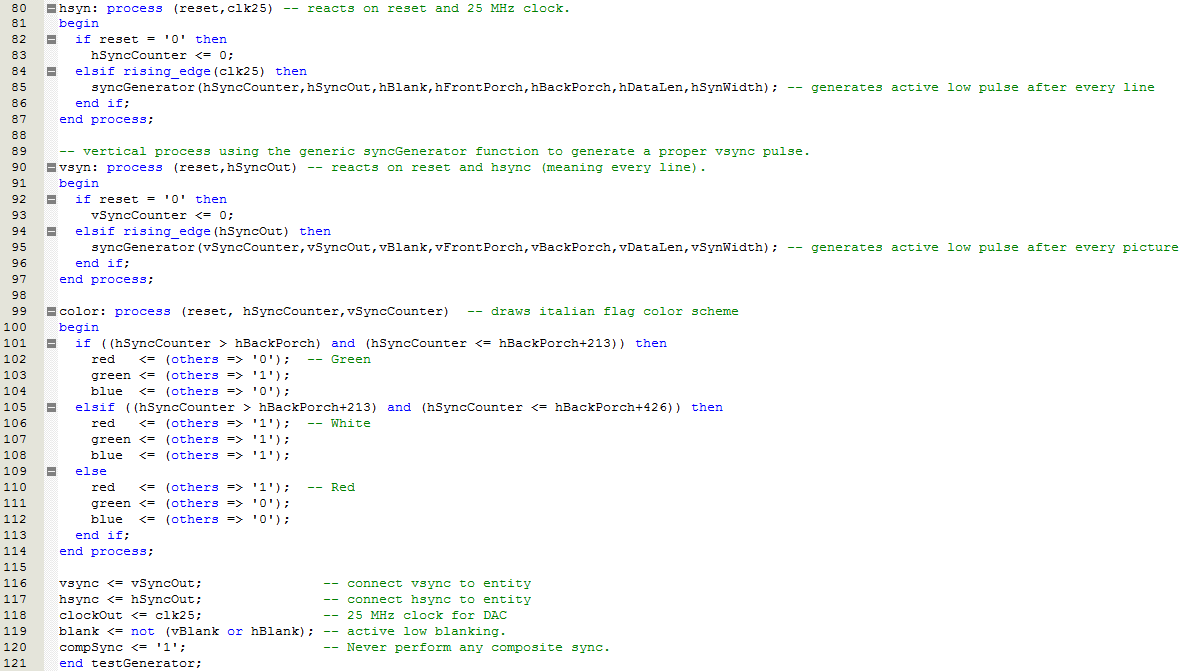
Del 1 af koden:



Del 2 af koden:



Del 3 af koden:



**2.1) Find klassen, mode og typen for hver parameter:**

Se punkt 2 i koden fra linje 38-67.

**2.2) Lav en tom procedure med de rigtige modes, klasser og typer i den erklærede del af arkitekturen:**

Se punkt 2 i koden fra linje 38-67.

**2.3) Design proceduren med funktionaliteten angivet i øvelsesvejledningen:**

Se punkt 2 i koden fra linje 38-67.

**3) Download og test projektet på DE2 boardet:**

Vi har downloadet det og testet det på DE2 boardet samt med DE2 Display med VGA-input udgivet af læreren, hvorved vi fik det forventede resultat.

Vi har efterfølgende ”leget” med koden og fået lavet det danske flag, hvilket vi har et bevis på herunder ☺ Koden (fra linje 99) for det danske flag står under billedet af det danske flag:



