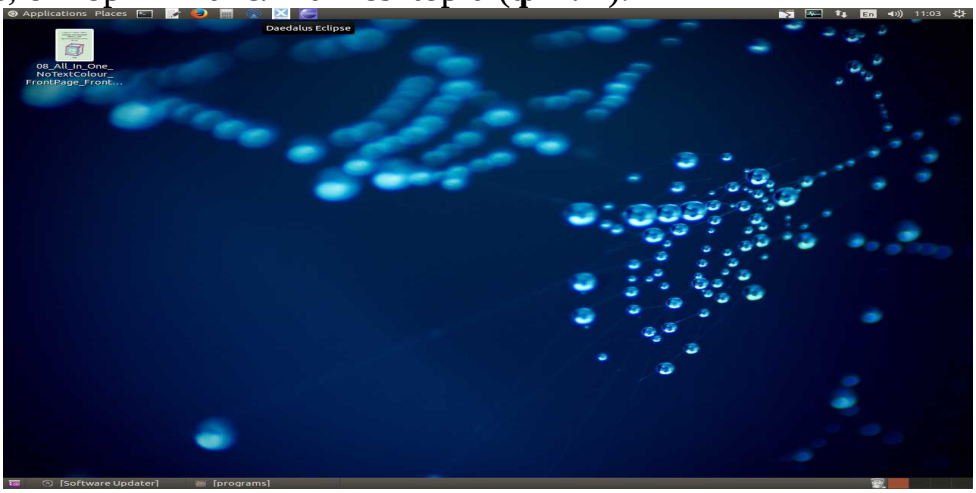
# Проектиране на вградени системи
# Лабораторно упражнение №6

## Създаване на Daedalus проект. Визуализиране на процесна мрежа.

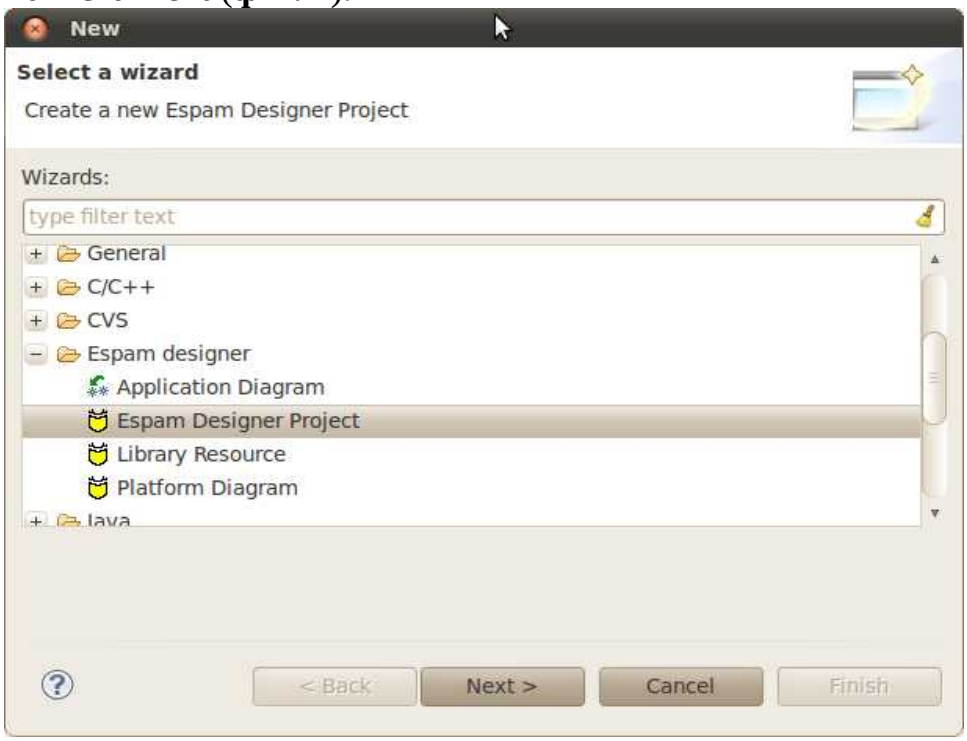
==========

**1.** Стартира се развойната среда Eclipse Indigo, модифицирана за работа с Daedalus програмите, от горния панел на Desktop-а (**фиг. 1**).




**Фиг. 1** – Стартиране на развойната среда Eclipse.

**2.** Създава се нов проект File → New → Other → Espam Designer → Espam Designer Project и се натиска Next (**фиг. 2**).

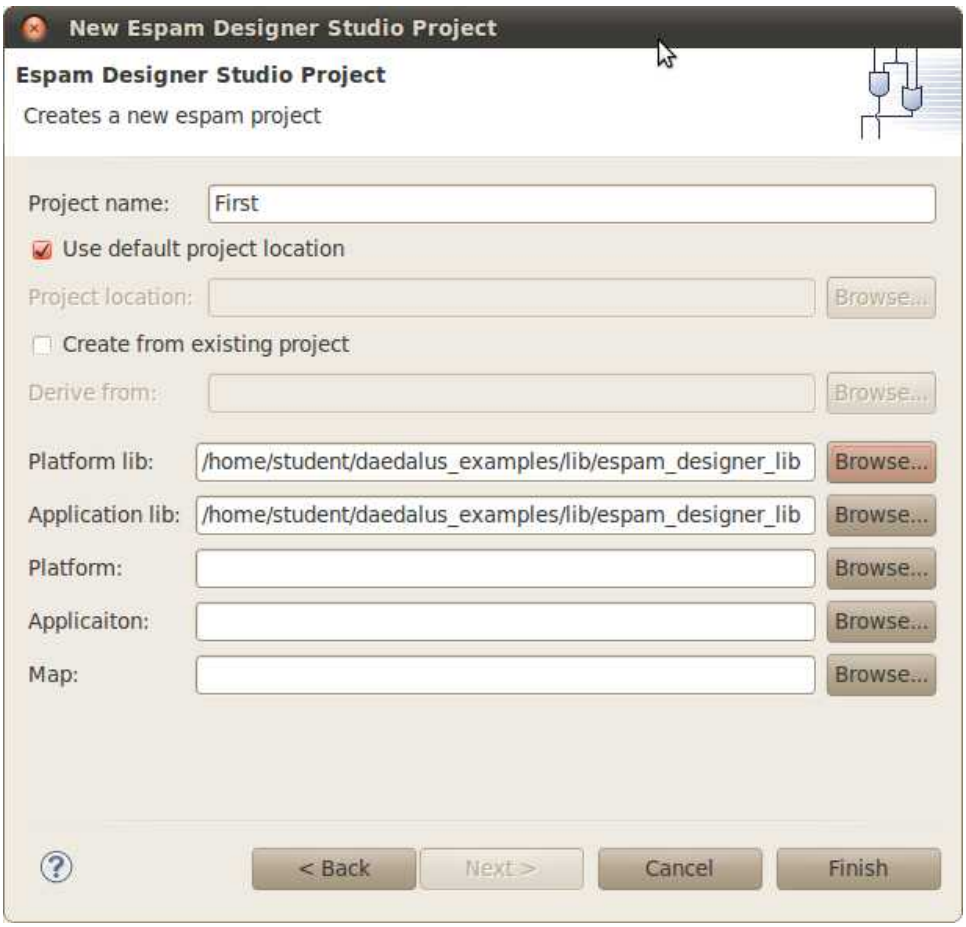


**Фиг. 2** – Създаване на Espam проект.

**3.** Задава се име на проекта “first” и се оказва пътя до библиотеките, които ще бъдат използвани за синтеза. Това са библиотеките в директорията (**фиг. 3**):

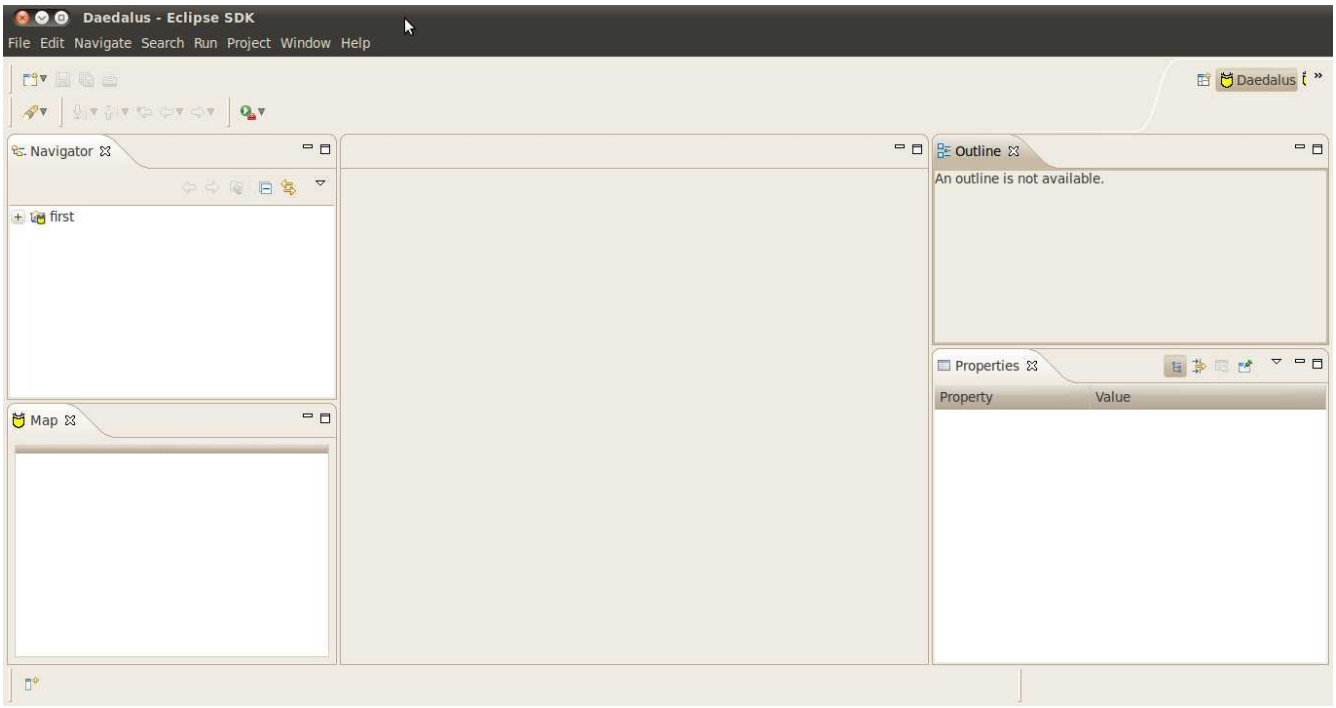
/home/user/programs/daedalus_gui/daedalus_examples/lib/espam_designer_lib

Полетата Platform, Application и Map се оставят празни. Натиска се бутон Finish.




**Фиг. 3** – Указване на пътя до използваните библиотеки.

**4.** Ако създаването на проекта е преминало успешно ще се отвори средата Eclipse с Daedalus перспектива (разположение), както е показано на **фиг. 4**.




**Фиг. 4** – Daedalus перспектива на средата Eclipse.

**5.** Избира се току-що създадения проект от дървото с проекти, натиска се десен бутон → Import → File System → Next → From Directory: Browse → /home/user/programs/daedalus_gui/daedalus_examples/daedalus_template/ → OK → поставят се отметки на всички файлове вдясно (empty.map, empty.pla, Makefile, run.sh) → натиска се + на директорията daedalus_template → поставя се отметка на субдиректорията sequential вляво (daedalus_template/sequential) → Finish.

**6.** Избира се директорията sequential. Файлът project.c се преименува на first.c с бутон F2. Този файл е top-level и съдържа функцията main( ).
Допълнителни функции на програмата могат да бъдат добавени от различни хедърни и сорс файлове с директивата #include. Такива са файловете project_func.h и project_func.c.

Преименуват се project_func.h и project_func.c съответно на first_func.h и first_func.c. Отваря се файлът sources и се коригира променливата:

PRJ_NAME = first

**7.** Във файла first.c се добавя директивата #include “first_func.h”.

**8.** От дървото на проекта се променят имената на файловете empty.map и empty.pla съответно на first.map и first.pla. Файлът first.pla съдържа информация за броя и вида на микропроцесорите, допълнителни IP елементи (например комуникационни модули) и връзките между тях. В настоящото лабораторно упражнение няма да се синтезира реална система, която да се зареди на FPGA. Следователно това е един празен файл, който в случая ни е необходим единствено за успешното пускане на симулациите. По същата причина файлът, отговарящ за разпределянето на процесите от паралелната програма върху отделните микропроцесори, first.map, е празен.

**9.** От дървото на проекта се отваря сорс файлът first.c от директорията sequential. Въвежда се код на програма за изчисление сумата на числата от 1 до 10. Нека резултатът да се изписва в терминала с функцията printf( ).

**10.** Написаната програма е последователна, т.е. в нея няма нишки. Компилира се от Run → External Tools → Compile Sequential. Това е първата стъпка от създаването на MPSoC. Наблюдава се компилацията в Console, разположена вдясно на екрана. Ако няма съобщение за грешка може да се премине към стартиране на програмата.

**ВНИМАНИЕ**: ако менюто на External tools е празно, изберете File → Import → Run/Debug → Launch configurations → From Directory: → Browse → /home/user/programs/daedalus_gui/daedalus_launch_configs → OK → в левия прозорец сложете тикче на директорията daedalus_launch_configs → Finish. След това Run → External tools → Organize favourites → подредете ги в нарастващ ред.

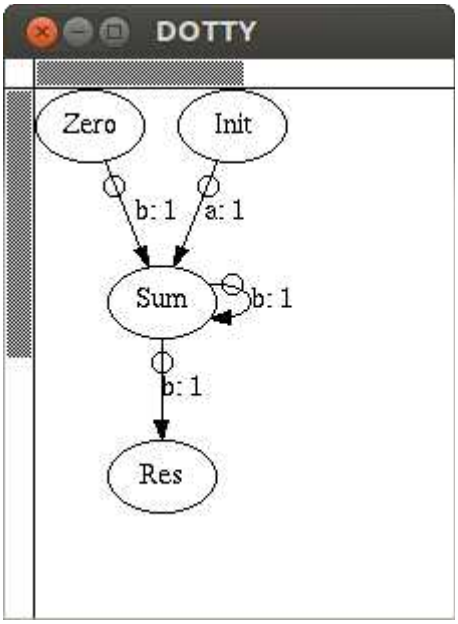
**11.** Стартира се последователната програма. За целта се избира Run → External Tools → Run Sequential. В Console трябва да се види очаквания резултат.

**12.** Следва да се генерира паралелен код от последователната програма. Това става с помощта на програмата PNgen. Избира се Run → External Tools → Generate KPN (Kahn Process Network).

**13.** Резултатите от генерирания паралелен код могат да бъдат визуализирани. Избира се Run → External Tools → Visualize PN, което трябва да покаже процесната мрежа на **фиг. 5**.
Всеки процес от нашия паралелен код ще бъде изобразяван с елипса, а връзките между процесите (т.е. комуникацията между процесорите) – със стрелки. От хардуерна гледна точка стрелките обозначават FIFO буфери, при които след четене данните се губят. Това означава, че променливата, записана във FIFO буфера, може да бъде прочетена само веднъж.
Когато даден процесор (от многопроцесорната система) чете от FIFO буфер, ако той е празен, процесорът ще бъде оставен да изчаква. Когато даден процесор записва във FIFO буфер, ако той е пълен, процесорът ще бъде оставен да изчаква. Това е т.нар. blocking read/write.
Спазването на горните условия гарантира правилната комуникация между процесорите. Благодарение на тези буфери се изключва необходимостта от използването на нишки и синхронизирането на комуникацията между тях с мютекси, семафори и т.н., което значително улеснява програмирането.

На този етап най-вероятно написаната програма няма да бъде успешно трансформира-на в паралелен код, защото не са спазени Daedalus *правилата за писане на статичен код*. За да може една последователна програма да бъде преобразувана в паралелна, тя трябва да отговаря на тези правила. На Desktop-а се намира PDF файл с всичките правила.

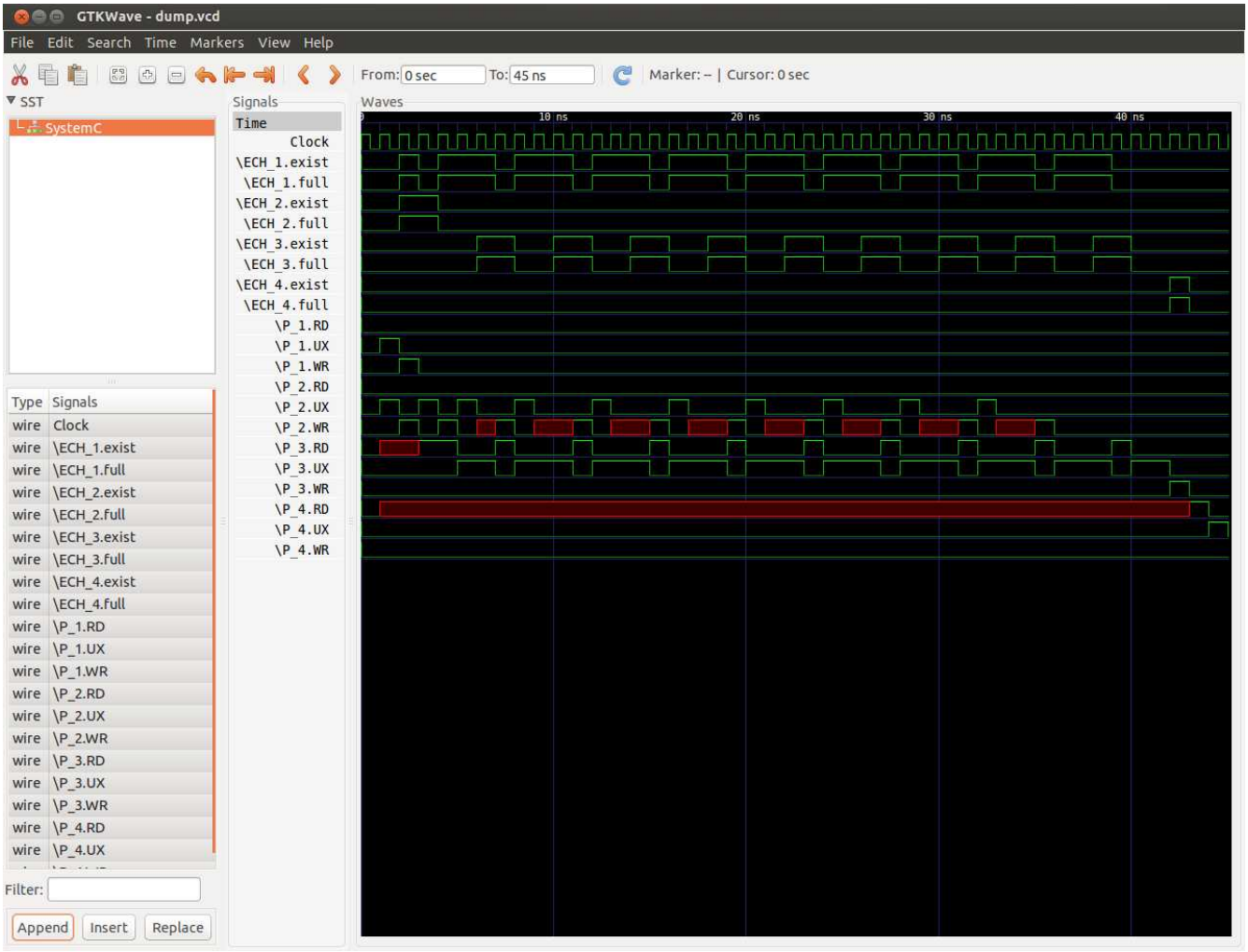
**ВНИМАНИЕ:** В края на лабораторното е дадена програма, която отговаря на тези правила.




**Фиг. 5** – Процесна мрежа на паралелната програма.

**14.** Използват се SystemC untimed и SystemC timed симулации, които могат да бъдат оприличени на DC и Transient анализ при електронните схеми. Untimed симулацията се използва за проверка коректността на паралелната програма – тя трябва да даде същия краен резултат, както последователната програма. Timed симулацията се използва за проверка на коректността на програмата при взети под внимание latency параметри на дадена реална система (например време за запис във FIFO буфер и др.), както и за визуализиране на комуникацията между различните процеси във времето. Измерва се в процесорни цикли, откъдето може да се трансформира във време (ако се знае един процесорен цикъл за колко части от секундата се изпълнява). Пример за такава симулация е дадена на **фиг. 6**.

P_1, P_2, P_3 и P_4 са четири примерни процеса. RD означава четене от FIFO буфер, WR – запис, а UX – execute или изпълнение на процеса. Червената графика означава блокиран read/write опит (поради характера на FIFO буферите, пояснени в предишната точка).




**Фиг. 6** – SystemC Timed примерна симулация на паралелен код.

```
/*****************************************************
****************** first.c ***********************
*****************************************************/

#include "first_func.h"

#define N 10

int main(void)
{
   int i;
   int a[N];
   int b;
```

```
   Zero( &b );

   for( i=0; i<N; i++) {
    Init( &a[i]);
    Sum( &a[i], &b, &b );
   }

   Res( &b );

   return(0);
}

/***********************************************
***************** first_func.c ****************
***********************************************/
#include <stdio.h>
#include "first_func.h"

void Zero( int *b ) {
   *b=0;
}

void Init( int *a ) {
   static int i=1;
   *a=i++;
}

void Sum( const int *a, const int *in_b, int *out_b ) {
    *out_b = *in_b + *a;
}

void Res( const int *b )   {
   printf("Result: b = %d\n", *b);
}

/***********************************************
***************** first_func.h ****************
***********************************************/
```

```
#ifndef FIRST_FUNC_H
#define FIRST_FUNC_H

void Zero( int *b );
void Init( int *a );
void Sum( const int *a, const int *in_b, int *out_b );
void Res( const int *b );

#endif
```

*  *  *

[1] Stefanov T., Deprettere E., Nikolov H., Marinov M., Popov A., *„Embedded Systems: components, modelling, design and case study“*. TU-Sofia, 2012, ISBN 978-954-438-975-8.

[2] Bogdanov L., H. Nikolov, T. Stefanov. *„Embedded Multi-processor Systems-on-Chip: Laboratory Experiments and Users Manual”*, TU-Sofia, 2013, ISBN 978-619-167-034-5.

инж. д-р Христо Николов (LIACS), гл. ас. д-р инж. Любомир Богданов, 2021 г.