Поведенческо (behavioral), структурно (structural) и данново (dataflow) описание на СГИС чрез използване на VHDL. Симулиране на работата на създадените електронни модули чрез програмата ModelSim

При описание на архитектурата (принципа на действие) на даден цифров блок или система, с помощта на езика VHDL , е възможно да се използват три начина за нейното създаване. Те се наричат и стилове на VHDL програмата. Трите стила, които се използват при езика VHDL са поведенческо (behavioral) описание, структурно (structural) и по начина на разпространение на потока от данни – данново (dataflow). Разбира се трите начина, които са възможни да се използват за описание при използване на езика VHDL могат да се използват и съвместно при конкретна програма. На следващите редове, ще се разгледат поотделно характерните черти за всеки от трите стила на програмиране с използване на езика VHDL.

1. Поведенческо, структурно и данново описание

Поведенческото описание, което ние използваме на лабораторните упражнения при създаване на комбинационни и последователностни цифрови модули. За него е характерно, че се описва поведението на реализирания модул т.е. неговата реакция на входни въздействия (сигнали). Той се доближава до познанията които имаме с езиците за описание на програмната част – използва се алгоритмичния начин за създаване на конкретна програма, използва се конструкцията IF THEN ELSE, CASE конструкцията и др. които се срещат и при програмните езици. Обикновено информация за състоянието на сигналите във времето не се използва, но тя може лесно да се включи чрез различните типове закъснения. Основен момент при поведенческия начин на описание на архитектурата е задаването на процеса. При описанието на процеса се определят сигналите, които стартират (запускат) неговото действие, както и последователността от действия , които се извършват в процеса. Тук искаме да припомним, че може да има и неявно зададени процеси (без ключовата дума процес), както и това че процесите се изпълняват паралелно (конкурентно) един с друг, а последователно се изпълняват действията включени в структурата на процеса. Поведенческия начин на описание се характеризира като най-абстрактния модел на проектирания цифров блок в сравнение с другите начини на описание – структурно и данново. За него е характерно използването на подпрограми и пакети, които подпомагат целия процес на проектирането.

Вторият начин на описание на архитектурата на проектираното изделие е т.нар. структурен стил при VHDL моделирането. При него моделираната система се изгражда от отделни елементи, като са посочени и връзките между тях. Създадените предварително елементи могат да бъдат включени в отделна библиотека и да се използват при различни цифрови системи. При проектирането тук се използва начина отдолу нагоре (bottom-up) като се започва от конкретни цифрови елементи (ниско ниво) и се върви към тяхното свързване и образуване на нови цифрови системи (по-високо ниво). Този метод на VHDL проектиране ще бъде демонстриран с изграждането на система от два компонента (логически елементи), като проектираната система ще бъде реализирана, проверена и синтезирана (създадена принципната електрическа схема) на следващите лабораторни упражнения.

Третият стил (начин) на описване на архитектурата на проектираното изделие е да се проследи какви промени претърпява потока от данни, който се придвижва от началото на цифровата система към нейния край т.е. това е т.нар. dataflow (даннов) начин на програмиране с езика VHDL. За него е характерно, че принципа на действие се описва с паралелни (конкурентни) изрази използващи логическите оператори. За да можем да опишем по този начин една система е необходимо да представим цифровата система във вид на сравнително прости по своето устройство комбинационни и последователностни елементи. Този метод се

използва за описание при не много сложни цифрови системи.

2. Примери за трите вида възможни описания чрез VHDL

За да се придобие по-голяма яснота за трите възможни начина за проектиране на архитектурата на цифровата система, ще бъде даден пример със описание на елемента полусуматор използвайки трите стила при VHDL програмирането.

Първо ще започнем с определяне на принципа на действието на полусуматора (формиране на спецификацията на полусуматора). Неговата блокова схема е дадена на фигурата по-долу.

където, с x и y са отбелязани двете еднобитови събираеми, сигнала enable (разрешение) както подсказва името му служи за разрешение на работата на полусуматора, а на изхода result (резултат) се получава резултата от събирането на операндите x и y. На изхода carry (пренос) се получава преноса в резултат на събирането (ако има такъв).

Спецификацията на полусуматора е следната:

- Входовете и изходите на полусуматора са с големина (размер) от един бит.
- Когато на входа enable имаме високо логическо ниво (логическа единица) , се разрешава работата на полусуматора т.е. сумирането на двата операнда и евентуално формиране на сигнал за пренос.
- Когато на входа enable имаме подадено ниско логическо ниво (логическа нула) изходните сигнали също преминават в ниско логическо ниво.

След като приключихме със спецификацията на полусуматора преминаваме към създаването на съответната VHDL програма. Първата стъпка при нейното написване е описанието на VHDL единицата (entity) , като след ключовата дума port се описват входовете и изходите на цифровия блок (неговия интерфейс). Това е направено със следните няколко реда.

```
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
port ( x, y , enable : in std_logic;
carry, result : out std_logic);
end half_adder;
```

Тук ще кажем, че създадената част от VHDL програмата трябва да се включи при съответното описание на архитектурата на полусуматора.

Първото описание, което ще направим, ще използва поведенческия стил на VHDL програмиране. Програмата е показана по-долу.

```
architecture of beh of half_adder is begin
 process (x, y, enable)
 begin
 if enable =’1’ then result <= x xor y; carry <= x and y;
 else result <= ‘0’; carry <= ‘0’; end if;
 end process;
 end half_adder;
```

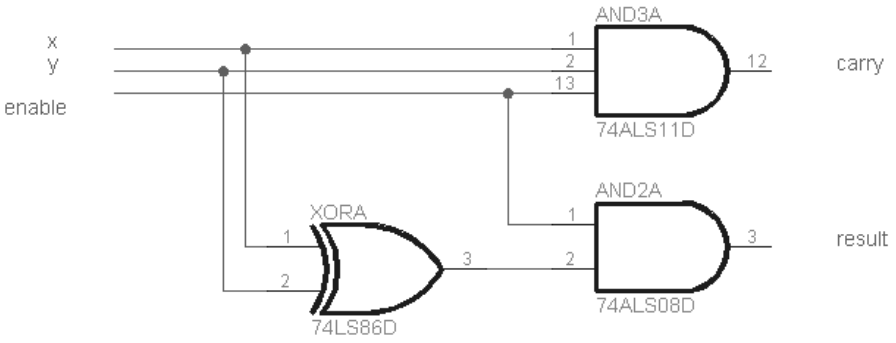
Изразите включени в поведенческото описание на алгоритъма изпълняват предварително определената спецификация на модула, както и неговата таблица на истинност. Създаденият поведенчески модел като е използван езика VHDL може да се симулира т.е. да се провери неговата работоспособност използвайки VHDL симулаторна програма като например ModelSim.

Второто описание на архитектурата на същия модул ще бъде направено в dataflow (описание на потока от данни). Ще бъдат използвани логически функции и ще бъдат формирани два (колкото са изходите) конкурентни израза. Самото описание е показано по-долу.

```
architecture dataflow1 of half_adder is begin
result <= enable and (x xor y);
carry <= enable and (x and y);
end dataflow1;
```

Искам да припомня, че при всяка промяна на някой от входните сигнали x, y или enable се изчисляват паралелно стойностите за изходните сигнали result и carry, т.е. тук имаме зададени неявно два процеса.

Като трети последен начин на описание на архитектурата на полусуматора ще бъде използвано структурното описание т.е. използване на отделни компоненти (в случая те са логически елементи). Преди това да бъде направено, ще бъде представена принципната електрическа схема на блока полусуматор. Тя е показана по-долу.



Както се вижда от схемата по-горе тя се състои от един двувходов логически елемент от тип И (AND2A), един тривходов логически елемент И (AND3A) и един двувходов логически елемент сума по модул 2 (XORA). По-долу е направено описание на архитектурата на полусуматора състоящ се от тези три елемента.

```
architecture structural1 of half_adder is
 component and2
 port (in0, in1 : in std_logic;
 out0 : out std_logic);
 end component;

 component and3
 port (in0, in1, in2 : in std_logic;
 out0 : out std_logic);
 end component;

 component xor2
 port (in0, in1 : in std_logic;
 out0 : out std_logic);
 end component;

 for all : and2 use entity gate_lib.and2_nty(and2_a);
 for all : and3 use entity gate_lib.and3_nty(and3_a);
 for all : xor2 use entity gate_lib.xor2_nty(xor2_a);

 signal xor_res : std_logic;

 begin

 a0: and 2 port map (enable, xor_res, result);
 a1: and3 port map (x, y, enable, carry);
 x0: xor2 port map (x, y, xor_res);
 end structural1;
```

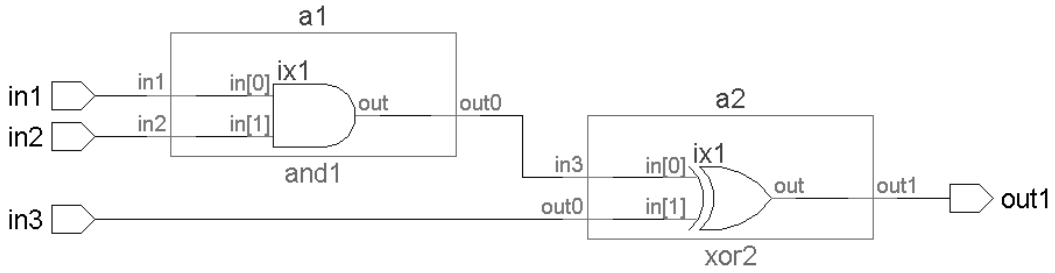
Анализирайки тази VHDL програма правят впечатление следните неща:

- Използване на ключовата дума component (компонент). След тази ключова дума се представят всички логически елементи включени в цифровата система, като се задават техните входове и изходи.

- Архитектурата на всеки един от компонентите трябва да се зададе предварително и да се включи в библиотека или да се зададе в този файл.

- Използвайки ключовата дума port map (карта на изводите) се извършва свързване на всеки един от сигналите на компонента със сигналите в цифровата схема т.е. прави се едно съответствие на сигналите, една схема на свързване, като вътрешния сигнал с име xor_res е сигнала , който свързва изхода на елемента сума по модул 2 с входа на двувходовия логически елемент И.

С оглед по-доброто изясняване на структурния начин за описание на архитектурата на дадена цифрова система е даден пример, който ще бъде използван на лабораторните упражнения. По долу е показан резултата от направения синтез, на базата на използваната програма на VHDL. Реализираната електронна схема се състои от два двувходови логически елемента И и сума по модул 2 свързани по определен начин.



Програмата на езика VHDL, довела до синтеза на горната схема е следната.

```
library ieee;
use ieee.std_logic_1164.all;

entity twocomp is
port (in1: in std_logic;
in2: in std_logic; in3: in std_logic; out1: out std_logic); end twocomp ;
architecture  structure of twocomp is component       and1
port(in1: in std_logic;
in2: in std_logic;
out0: out std_logic);
end component;

component xor2
port( in3: in std_logic; out0 : in std_logic; out1: out std_logic);
end component                                ;

signal out0: std_logic;

begin
a1: and1 port map (in1, in2, out0);
a2: xor2 port map (out0, in3, out1);
end structure;

library ieee;
use ieee.std_logic_1164.all;

entity and1 is
port (in1: in std_logic;
in2: in std_logic; out0: out std_logic);
end and1;

architecture   dataflow1 of and1 is begin
out0 <= in1 and in2;
end dataflow1;
```

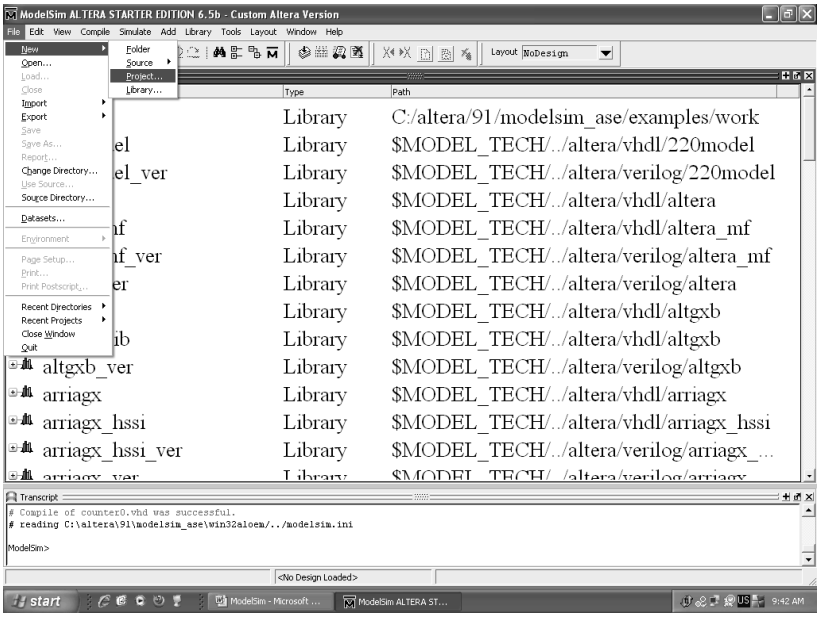
```
library ieee;
use ieee.std_logic_1164.all;


entity xor2 is
port (in3: in std_logic; out0: in std_logic; out1: out std_logic);
end xor2;

architecture      dataflow2 of xor2 is begin
 out1 <= in3 xor out0;
end dataflow2;
```

3. Симулиране на работата на създадените електронни модули чрез програмата ModelSim

VHDL симулатора ModelSim ALTERA STARTER EDITION 6.5b е създаден от американската фирма Mentor Graphics® специално за друга американска фирма Altera Corporation – един от най-големите производители в световен мащаб на програмируеми интегрални схеми и програмни средства за симулиране, проектиране и реализиране на създадените проекти.



Работата със симулатора е описано в неговото помощно меню (**Help**) на английски език. В следващите редове са представени накратко основните неща, които трябва да се знаят и използват при работата с ModelSim ALTERA STARTER EDITION 6.5b.

В началото на работата трябва да се създаде проект и да се прибави към него файл.

Както се вижда от фигура по-горе това става като се избере падащото меню **File** и от него командата **New / Project**.

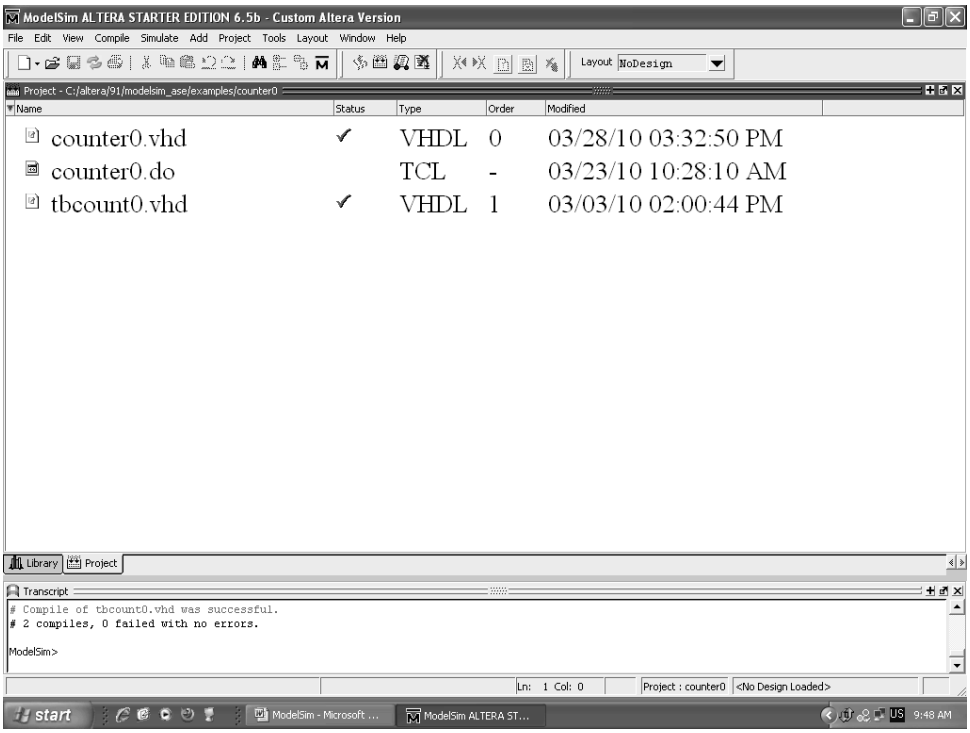
При създаването на проекта трябва да се запише неговото име в правоъгълника под **Project Name**. Необходимо е да се зададе папката в която ще бъде записан проекта (**Project Location**) (по-добре е да не се променя предварително избраната папка), като името на библиотеката **(work**) не е необходимо да се променя (**Default Library Name**). За завършването на последователността по създаването на проект трябва да се щракне с левия бутон на мишката върху бутона **OK**. След

създаването на проекта трябва да се добави файл написан на езика VHDL към него. Това може да се извърши като от появилия се прозорец с име **Add items to the Project** трябва да се щракне с левия бутон на мишката върху бутона с име **Create new File**, ако нямаме предварително създаден файл. Трябва да се укаже името на файла в правоъгълника **File Name**, типа на файла (написан на езика VHDL) и да се щракне с левия бутон на мишката върху бутона **OK.**

След това като се щракне с десния бутон на мишката върху името на файла от появилото се контекстно меню трябва да се избере командата **Edit** за да започне въвеждането на програмата на езика VHDL (сорс файл). Ако е необходимо да се увеличи размера на шрифта на сорс (първичен) файла трябва да се постъпи по следния начин. От падащото меню **Tools** на симулатора трябва да се избере командата **Edit Preferences** и от появилия се прозорец да се избере **By Window / Source Windows**. След това трябва да се избере нов размер на шрифта, чрез щракване с левия бутон на мишката върху **Choose** от правоъгълника с име **Fonts**. След избора на подходящия размер на шрифта направените промени се съхраняват като се щракне с левия бутон на мишката върху бутона **Apply**.

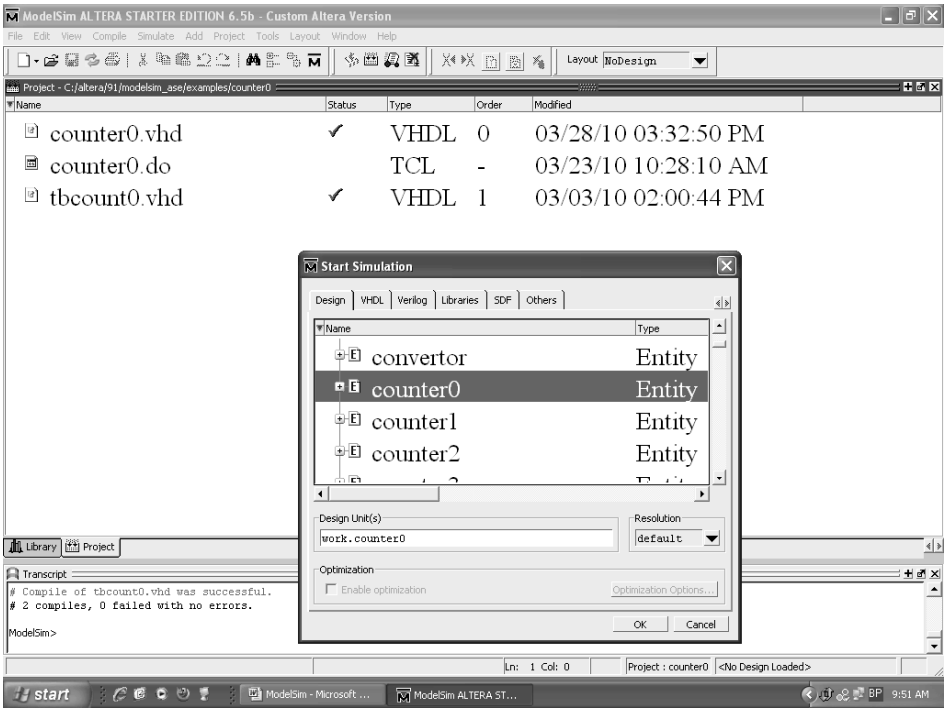
Въведеният файл на езика VHDL трябва да се компилира или чрез падащото меню или чрез помощния бутон с име **Compile**. Трябва компилираното да бъде успешно, което се вижда от изписания в зелен цвят надпис **Compile of** (име на файла) **was successful**. Такъв пример е показан на фигурата по-долу.

Ако има грешки при компилирането те трябва да се отстранят.



След успешното компилиране може да се премине към симулиране (проверка) на работоспособността на описания с езика VHDL модул. Това става като се използва падащото меню **Simulate** и командата **Simulate.**

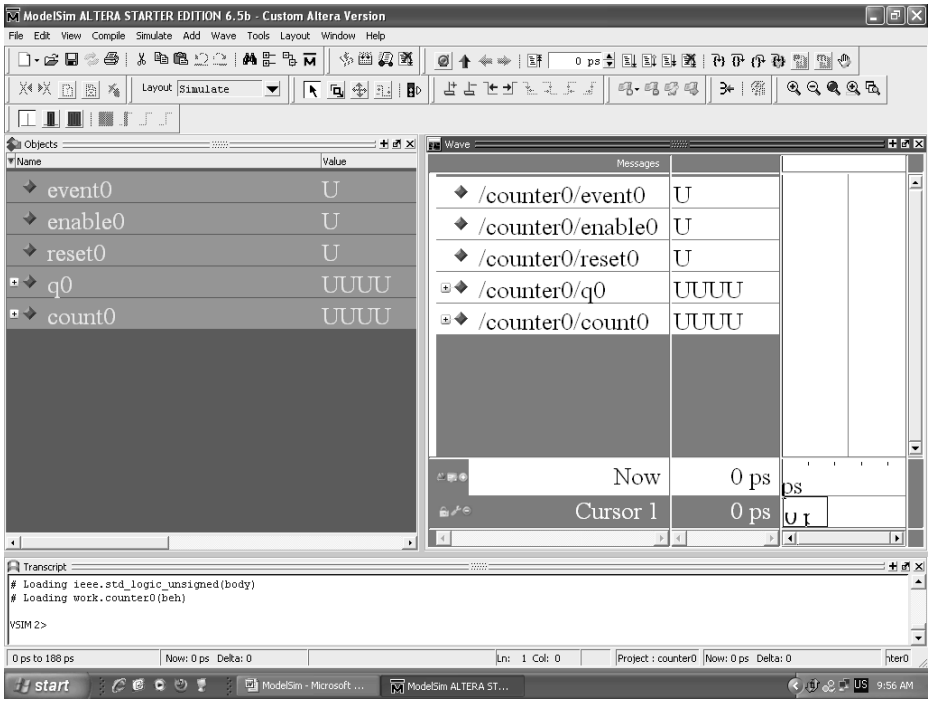
.



След това от появилия се прозорец с име **Simulate** трябва от библиотеката Work да се избере VHDL единицата (entity) (или върху архитектурата, ако имаме няколко такива) подлежаща на симулиране и да се щракне с левия бутон на мишката върху бутона **ОК** както е показано на фигурата по-горе..

За да можем да наблюдаваме резултатите получени в резултат от симулиране на работата на създадения цифров модул трябва от падащото меню **Edit** на симулатора да се щракне с левия бутон на мишката последователно върху командите **Objects** и **Wave**. Трябва и двата отворени прозореца да се виждат едновременно на екрана. Маркират се сигналите, които ще наблюдаваме от прозореца **Objects** и с помощта на мишката по метода **Drag. and Drop** се поставят в прозореца **Wave**, както е показано на фигурата по-долу.

След това започва задаването на стойностите на входните сигнали. Това може да стане като чрез щракване с десния бутон на мишката върху името на даден сигнал намиращ се в прозореца **Wave** и от контексното меню да се избере командата **Force**. От контексното меню може да се избере и команда Clock за задаване на тактов сигнал.



Симулацията започва (стартира) като от падащото меню **Simulate** се избира командата **Run / Run 100 pS**. Времето за симулиране (в случая 100 pS) може да се промени чрез съответния помощен бутон. На прозореца **Wave** може да се проверят логическите стойности на сигналите, добавени в този прозорец. Сравнявайки ги с тези от таблицата на истинност за всеки цифров модул, можем да направим заключение за правилната му работа.

Освен, чрез командата **Force** и командата **Clock**, тестовите въздействия могат да се опишат предварително в файл от тип .do. Съдържанието на подобен файл е показано по-долу.

```
add wave Event0
add wave Enable0
add wave Reset0
add wave Q0
force -freeze Event0 0 0, 1 {10ns} -r 20ns
force Reset0 1
force Enable0 0
run 5 ns
force Reset0 0
force Enable0 0
run 5 ns
force Enable0 1
run 350 ns
```

Чрез командата **add wave** последвана от името на даден сигнал се извършва добавянето му в прозереца **Wave**. Чрез f**orce** се извършва задаване на логическо ниво, а чрез **run** се стартира симулацията на електронния модул за указано време. Чрез командата **force –freeze** се задава тактов сигнал.