1., The program and the hardware environment

The resource map file describes in detail the physical characteristics of the network of processors available to operating system. It is used by the network manager, when loading operating system onto the network for the first time, and by the task force manager, when deciding how to allocate tasks to processors. The resource map file includes the following information:

- the number of processors
- the processor types
- how the processors are interconnected
- how much external memory

The network manager use this information to determine which processor need to have the operating system nucleus loaded onto them. The task force manager uses this information to work out how to load the network, as specified in the resource map file, with the resources

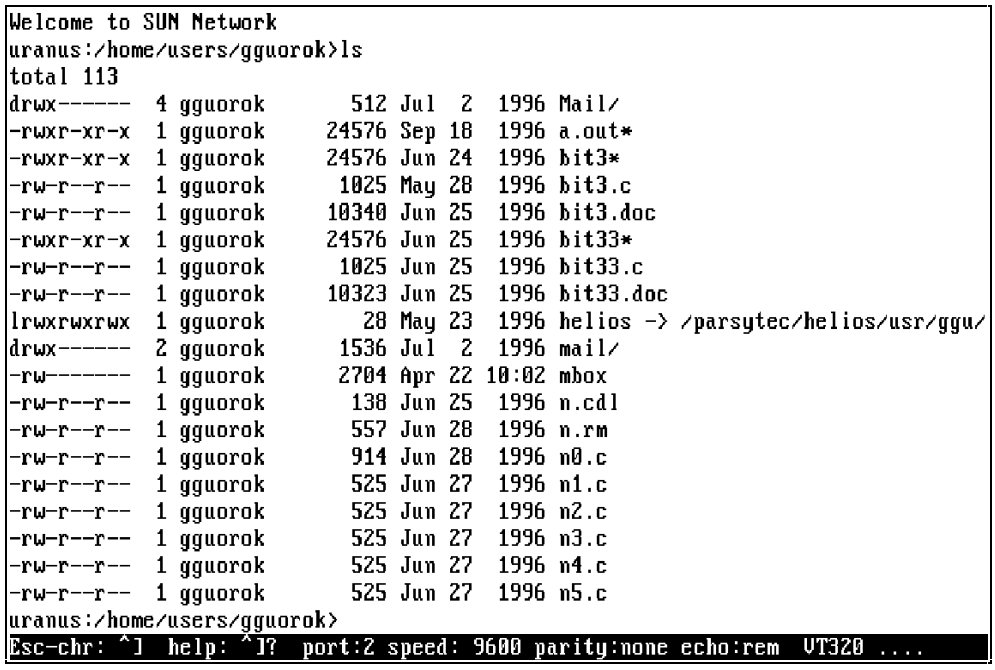

Figure 1 The home screen via *telnet*

requested by the task force, as specified in the CDL file. The resource map file must be written when the network is first put

together and amended if the network is reconfigured. It must be compiled by the utility program (*rmgen*) before it can be used by the network manager. This program simply converts textual resource map file into a more efficient binary representation. Resource map file consist of a set of network descriptions called subnets. Each subnet consist of a set of

```
uranus:/home/users/gguorok/helios>ls n*.*
-rw-r--r--  1 gguorok      1923 Jun 27  1996 n.asm
-rw-r--r--  1 gguorok       138 Jun 27  1996 n.cdl
-rw-r--r--  1 gguorok       557 Jun 28  1996 n.rm
-rw-r--r--  1 gguorok       914 Jun 28  1996 n0.c
-rw-r--r--  1 gguorok       525 Jun 27  1996 n1.c
-rw-r--r--  1 gguorok       525 Jun 27  1996 n2.c
-rw-r--r--  1 gguorok       525 Jun 27  1996 n3.c
-rw-r--r--  1 gguorok       525 Jun 27  1996 n4.c
-rw-r--r--  1 gguorok       525 Jun 27  1996 n5.c
-rw-r--r--  1 gguorok      3864 Jun 28  1996 nn.doc
uranus:/home/users/gguorok/helios>
uranus:/home/users/gguorok/helios>
```

Figure 1 The "HELIOS" screen via *telnet*

processor descriptions. A processor description is a terminal or a subnet description. It is the terminal descriptions which describe the resource available to a particular processor and how the processor and how the processors in the network are connected. The subnet construct enables complicated networks to be divided into a set of related subnetworks. It is entirely up to you how you decide to divide your networks. You may decide to have one large network or several small networks intended for different applications. for example, a network of processors which is shared between several users might be divided into a number of subnetworks, one per user. It is always a good idea to draw a picture of our network before attempting to write the research map file. **Figure 1,2** (5) [1]

The component distribution language (CDL) is used to defined groups of related tasks called task forces. A CDL definition consist of two parts:

[1] Se The References on page 16

- a description of how the tasks are connected together
(i. e., how the communicate with each other).
- an optional description of the resource requirements for each task.

The CDL task force descriptions are used by the task force manager when loading task forces onto the network. The task force manager attends to reconcile the research requirements us specified in the CDL file, with the network resources us specified in the resource map file. The task force manager also ensures the tasks are correctly connected together no matter how the tasks are physically distributed across the network.. CDL allows any task force to be described in a language and network independent way. The language independent aspect of CDL enables task forces to include task forces written in more than one language. The network independent aspect of CDL enables task forces to be run on a single processor network as easily as on a multiprocessor network without any change to the CDL description. It is the task force manager which decides where to place the tasks on the available processors and which ensures that all the communication pipes are correctly connected. 5

2., The programming method

The many transputer containing parallel machines are very useful solving computational hunger problems, having inherent parallelism, such as weather forecast, modeling, image processing, ... and so on. An other application could be to make experiments with different structures, before their have a final hardware realization. In this paper we describe two worked examples from the field of signal processing, illustrating both types of problems mentioned above. The examples are keeping single and stupid in order to help understand how to analyze and parallelism the problem as well as how to work

with the Parsitec HELIOS parallel operating system. The programs were written in '**c**' programming language.

2., The digital noise

For this topology and program was designed to generate digital noise, a sequence of pseudo-random numbers. The circuit consists of a shift register with a feedback parity bit from the first and last bits in the shift register to produce a maximal length sequence of non-repeating strings. For a n-bit shift register, this algorithm will generate a sequence of 2^n - 1 different strings (excluding the null string which would be a permanent state) of length *n* before returning to a previous state and repeating the sequence. The order of the strings is apparently random.

$$N_{pattern}=2^{(n-1)}$$

n; length of the digital shift register. The input is one clock connect at the end of the circuit and the single output is connected to the opposite end. The circuit design consists of several major components: The shift register, logical feedback exclusive or gates. (Figure 1.)

The Inverted XOR gate at the input end of the circuit accepts the feedback lines from the first and last RS master-slave flip-flops in the shift register. The RESET circuit, at the power on, set to the start position the shiftregisters. The CLOCK generator made the clock impulse.

Since the shift register consists of RS flip-flops, the non-inverted output is sent to the input of the first flip-flop in the shift register. An inverted XOR gate was used because the design of the RS flip-flops made having both a non-inverted and an inverted feedback line from

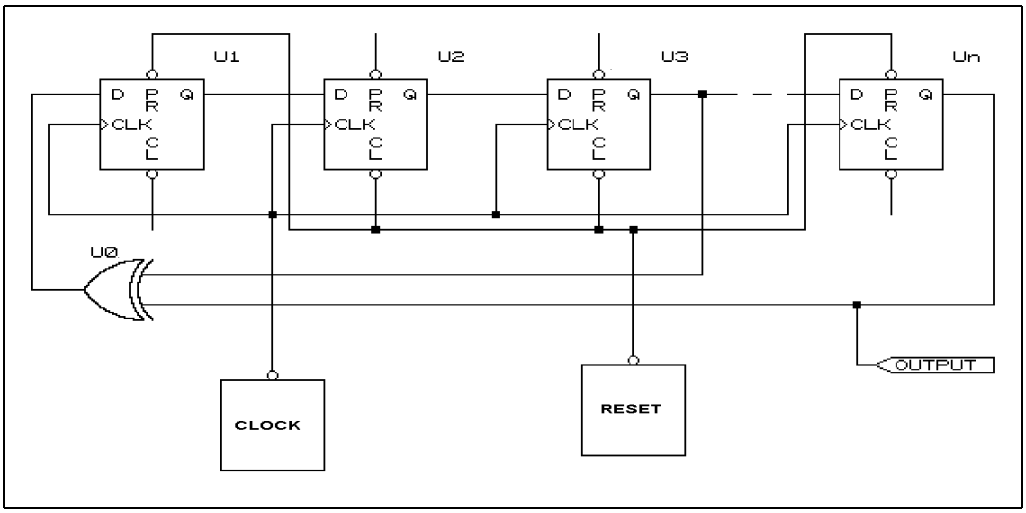

Figure 3 The well know noise generator circuit

different bits in the shift register on the same side of the circuit very inefficient. Figure 3.

The circuit consists of a shift register with a feedback parity bit from the first and last bits in the shift register to produce a maximal length sequence of non-repeating strings. For a n-bit shift register, this algorithm will generate a sequence of 2^n - 1 different strings (excluding the null string which would be a permanent state) of length n before returning to a previous state and repeating the sequence. The order of the strings is apparently random. There are limited values of n that will produce maximal length sequences.

There are limited values of *n* that will produce maximal length sequences. The circuit was designed with the in transzputer in HELIOS machine. The circuit is designed linearly because the "shift register" can be efficiently laid out in a linear fashion.

3., Realization on serial machine

```
/*************************************************************
**/
/*                                                           */
/*    Noise generator program on a 32 bit machine (Athens 27.05.1996)
*/
/*
*/
/*************************************************************
**/


main()
{
int mar, and, zso,
  boc, geo, cip;

boc=1300;                       /* count of cycle, length of the
                          "noise dump", duration of noise */

mar=123456789;                          /* onset of pattern */
    while (boc!= 0)               /* loop */
    {
    boc--;
    and=mar<<1;          /* left shift */
    geo=and&(1<<29);

    /* and relation with 00010000000000000000000000000000 */

    geo=geo>>29;

    /* transform of the result to 0000000000000000000000000000000x */

    zso=and&(1<<30);

    /* and relation with 00100000000000000000000000000000 */
```

```
zso=zso>>30;

/* transform of the result to 00000000000000000000000000000000x */

cip=geo^zso;          /* exor relation */
mar=and|cip;          /* overrun if result is one */
printf ("%x", and); /* result */
```

Figure 4 The ordinary serial program in *C* implementation

eb79a2a1d6f34543ade68a875bcd152eb79a2a4d6f34548
ade
21526b79a2a4e6f34549ede68a93cbcd1527a79a2a4f6f34
54
8cd1527b09a2a4f6234549ec468a93d8ad1527b14a2a4f62
a4
93d8ae1527b15c2a4f62b8549ec572a93d8ae6527b15cea
4f6
4573e93d8ae7e27b15cfc4f62b9fa9ec573f63d8ae7ec7b15
cf
e6c573f68d8a1e7ed0b15cfda262b9fb46c573f68c8ae7ed
1a1
fb468573f68d124ae7ed1a65cfda34eb9fb469e73f68d3ee7
ed
f89fb469f23f681d3e47ed311a7cafda34f94fb469f28f68d3e
5
a34f940b469f28268d13e506d1a7ca0ca34f941a469f28368
d3
a0dc34f9411b869f28372d3e506e4a7ca0dca4f941b969f28
37
c7ca0dcbaf9141b974f28372e8e506e5d0ca0dcba0941b9
742
6e5d0aa0dcba1641b9742e8372e85e6e5d0bcdcba1781b
974
06e5d10bc2dcba1784b9742f0a72e85e16e5d0bc2ccba17
858
85e1645d0bc2caba1785967412f0b2ee85e165cd0bc2cb8
a17
2e685e165cebc2cb91c178597382f0b2e705e165ce2bc2c
b9c
0b2e71ce165c1e38c2cb9c70859738e2b2e71c4165ce388
2cb
e22b2e71c46615ce388ecb9c711c9738e23a2e71c4745ce
388
738e23aee71c1475cce388eb89c711d7238e23ae471c475
cae
1d72188e23ae521c475ca4388eb948711d7292e23ae524c4
7
9211d72921423ae5248475ca4928eb949261d72924c3ae52
4
b9149264d72924c8ae5249925ca49326b949264e72924c9e
e

Figure 5 Any part the result of the sequential programs

4., Realization on parallel machine

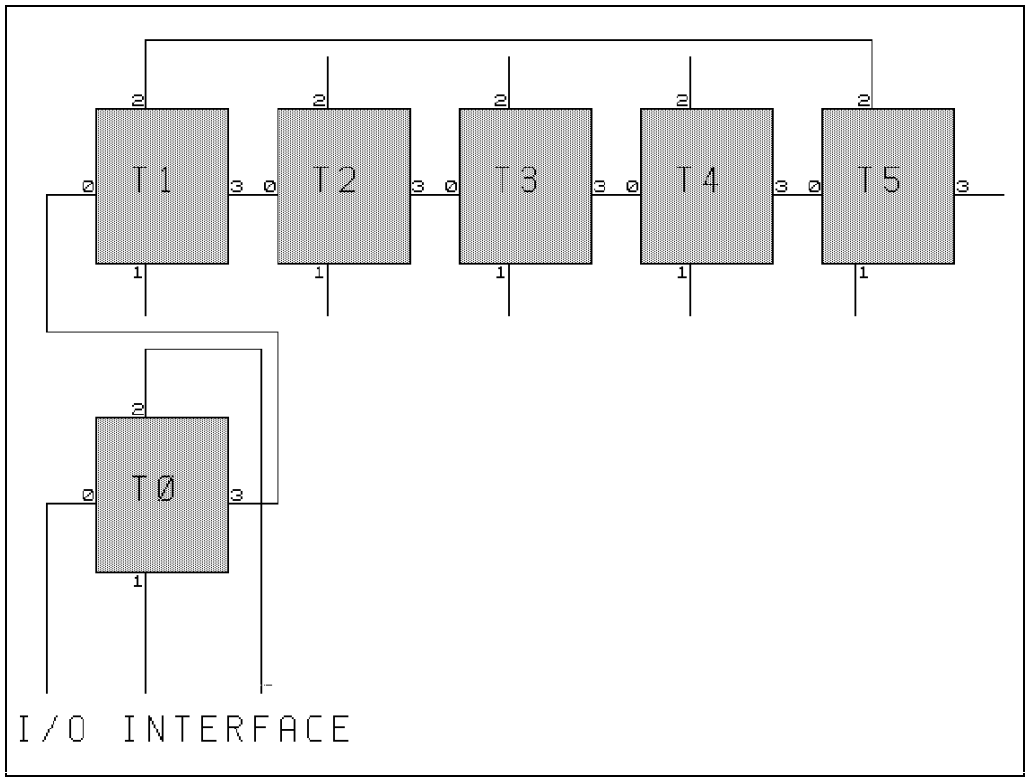

Figure 6 The realization on transputer network

```
component test0 {memory #100000;}
component test1 {memory #100000;}
component test2 {memory #100000;}
test0 (<> test1, <> test2)
```

Figure 7 The ordinary CDL segment

```
subnet /n {

CONTROL Rst_Anl [/n/00];

terminal 00 { ~IO,,~05,~01;     SYSTEM;
            Mnode Rst_Anl [pa_ra.d];
            memory #400000;
            ptype T800;
            }
terminal 01 { ,~00,,~02; HELIOS;
            memory #100000;
            ptype T800;
            }
terminal 02 { ,~01,,~03;          HELIOS;
            memory #100000;
            ptype T800;
            }
terminal 03 { ,~02,,~04; HELIOS;
            memory #100000;
            ptype T800;
            }
terminal 04 { ,~03,,~05; HELIOS;
            memory #100000;
            ptype T800;
            }
terminal 05 { ,~04,,~00; HELIOS;
            memory #100000;
            ptype T800;
            }
terminal IO { ~00; IO; }
}
```

Figure 8 The topology definition

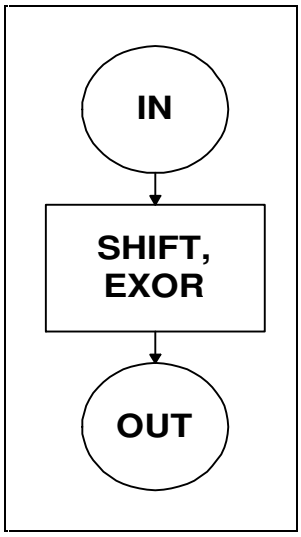

Figure 9 The chart of the slave segment

```
/*************************************************/
/*                                               */
/*        Noise generator program (n2.c) to transputer_02     */
/*                                               */
/*************************************************/


#include <stdio.h>
#include <posix.h>

int main()

{
int result;
char and, zso;

 result=read(0,&and,1);  /*read from link one byte*/
 zso=and<<1;             /*shift left with one bit*/
 result=write(1,&zso,1);  /*one byte write to link*/
}
```

Figure 10 The SLAVE segment

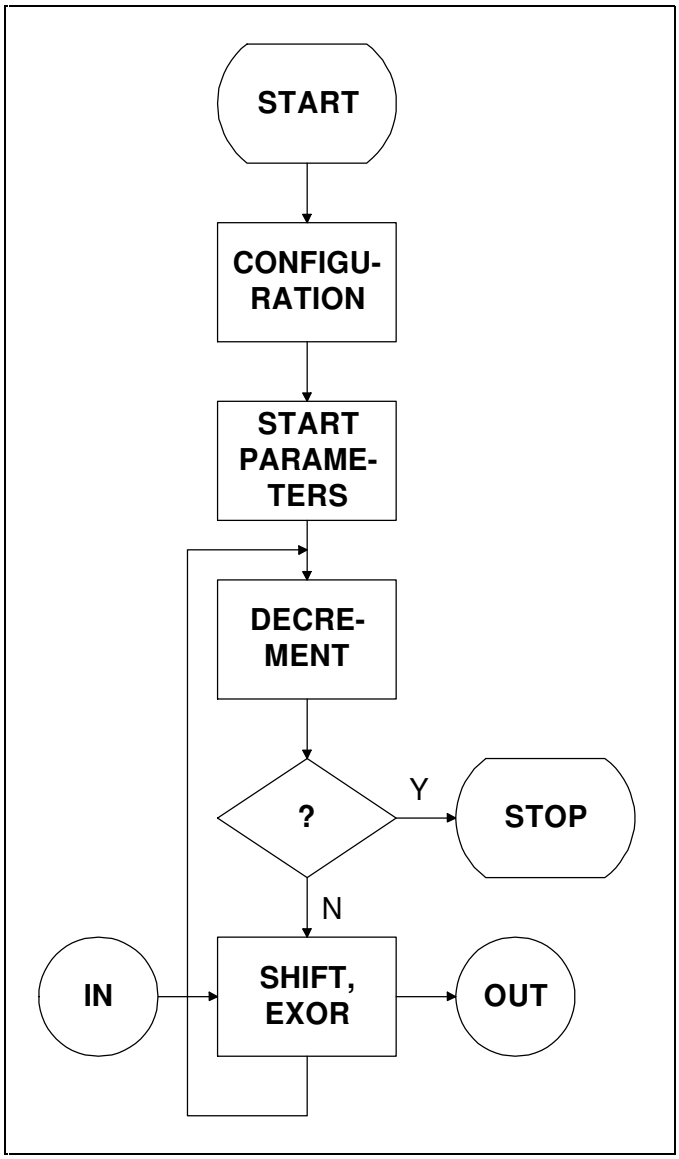

Figure 11 The chart of the main segment

```
/*********************************************************/
/*                                                       */
/*     Noise generator program on a transputer machine   */
/*              master program (n0.c) to transputer_00   */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <posix.h>
```

```
int main(void)

{
int boc, geo, cip, mar,zso,and, result;
char cmar, cand, czso;
boc=33;                 /*count of cycle, length of the pattern*/
mar=1234;                       /*onset of pattern*/
   result = write(5,&cmar,4);
   while (boc!= 0)              /*loop*/
   {
   boc--;
   result = read(4,&cand,4);

and=zso;
   and=and<<1;          /*left shift*/
   geo=and&(1<<29);
   geo=geo>>29;

   zso=and&(1<<30);
   zso=zso>>30;
   cip=geo^zso;         /*exor relation*/
   zso=and|cip;         /*overrun if result is one*/
   result = write(5,&czso,4);
   printf ("%x\n", zso);        /*result*/
   }
}
```

Figure 12 The MASTER program

```
uranus:/home/users/gguorok>
uranus:/home/users/gguorok>a.out
eb79a2a1d6f34543ade68a875bcd152eb79a2a4d6f34548ade68a925bcd1526b79a2a4e6f34549e>
Esc-chr: ^]  help: ^]?  port:2 speed: 9600 parity:none echo:rem  VT320 ....
```

Figure 13 The "result" screen via *telnet*

6., References

1.,Digital Signal Processing
Abraham Peled, Bede Liu
John Willey & sons Toronto 1976

2.,Párhuzamos és konkurens programozás sok-transzputeres rendszereken
P. Kacsuk, Sz. Ferenczi
BME, Mérnöktovábbképzô Intézet 1993 Budapest

3.,Introduction to Digital Filters
T. J. Terrel
The Macmillan Press LTD 1980 New York

4.,The CDL Guide
Perihelion Software LTD
Distributed Software Limited, 670 Aztec West
Bristol BS12 4SD, UK

5.,The Helios Parallel Programming Tutorial
Bill Noble, Rachel Ganz of Vardas and Bart Veer
Distributed Software LTD, UK 1990