```
MODULE SysDef;  (*For use with Portable Extended Oberon System*)
  IMPORT SYSTEM;
  CONST VERSION* = "SysDef Jul/18/2024";
(*System Definition Module provides addresses, constants, and simple procedures for
the major hardware interfacing. What is included: The major hardware of the main
board, such as memory size and addresses, CPU interface registers, or framebuffer
memory. What is not included: Minor or transient hardware such as PMOD modules,
temperature sensors, or other application specific hardware which is factored out
to their own subsystems.
The documentation section (after the module) lists all the constants from all
modules. This section will be eventually folded away.
Note: The same constants are hard coded in several modules, copied in the
documentation even when they repeat.
-----------------------------------------------------------------------------*)
(*The peripheral address map from PO.Computer page 13.
```

#	Unsigned	Signed	input	output	name in	name out
0	0FFFFFFC0H	-64	ms counter	reserved	TIMER	-
4	0FFFFFFC4H	-60	switches	LEDs	SWITCH	LEDS
8	0FFFFFFC8H	-56	RS-232 data	RS-232 data	RSDATA	RSDATA
12	0FFFFFFCCH	-52	RS-232 status	RS-232 control	RSSTATUS	RSCTRL
16	0FFFFFFD0H	-48	SPI data (SD, net)	SPI data (SD, net)	SPIDATA	SPIDATA
20	0FFFFFFD4H	-44	SPI status	SPI control	SPISTATUS	SPICTRL
24	0FFFFFFD8H	-40	PS/2 keyboard	--	KEYBRD	-
28	0FFFFFFDCH	-36	mouse	--	MOUSE	-

```
All these are negative numbers in Oberon, which does not provide unsigned integers.
Example: SPI data address calculated with Windows XP Calculator in DWORD display
mode. Switch to Dec. Enter "-48". Switch to Hex --> FFFFFFD0, same as above.

The memory-mapped peripheral address in software is FFFFFFxx. It is NOT used in
RISC5top.v, because the upper 8 address bits are NOT routed from RISC5. The actual
address used in RISC5top.v is rather 3FFFFxx. The prefix 3FFFF will change in
firmware, when more address bits are routed in Verilog. Fortunately, the software
memory map will stay the same. Addresses such as "-40" do not need to change.
-----------------------------------------------------------------------------*)
   (* Peripheral devices *)
   TIMER*     = -64;     (*input*)
   SWITCH*    = -60;     (*input*)
   LEDS*      = -60;     (*output*)
   RSDATA*    = -56;     (*input and output*)
   RSSTATUS*  = -52;     (*input*)
   RSCTRL*    = -52;     (*output*)
   SPIDATA*   = -48;     (*input and output*)
   SPISTATUS* = -44;     (*input*)
   SPICTRL*   = -44;     (*output*)
   KEYBRD*    = -40;     (*input. See WARNING *)
   MOUSE*     = -36;     (*input. See WARNING *)
(*WARNING. In Input.Mod these numbers were swapped: -40 <--> -36 *)
```

```
(*-------------------------------------------------------------------------------
The CPU register map from PO.System page 13 page 104.
The RISC processor features 16 registers (of 32 bits). R0 - R11 are used for
expression evaluation. R12 - R15 have fixed, system-wide usage. Note that "12" can
be either a register or a memory address, and these are not the same. Registers are
accessed with REG and LDREG, while memory is accessed with GET and PUT. *)
   (* CPU registers accessible with SYSTEM.REG and SYSTEM.LDREG *)
   REG12*  = 12;  (* Address of the module table MT (typically constant)*)
   REG13*  = 13;  (* Static Base address for variables in the current module SB*)
   REG14*  = 14;  (* Stack Pointer SP *)
   REG15*  = 15;  (* Return address LNK (fixed by RISC's BL instruction) *)
   MTOrg*  = 20H; (* Module Table actual mem address to be kept in MT (Reg 12) *)
   MT*   = REG12; (* Reg 12 alias is Module Table (R12 keeps the address of MT) *)
   SB*   = REG13; (* Reg 13 alias is Static Base *)
   SP*   = REG14; (* Reg 14 alias is Stack Pointer *)
   LNK*  = REG15; (* Reg 15 alias is LNK *)

(*Trap handling: PO.Applications page 77. Trap is installed in the first address of
the Module Table, that is MTOrg = 20H. This is because the module numbers start at
1. There is no module numbered 0. The 0-th location of the Module Table is thus
free. It is used to install the trap.*)
(*-------------------------------------------------------------------------------
Memory addresses from addr=0 to FFFF_FFFF. Boot parameters are near 0. Peripherals
are near the max address. Video RAM is below peripherals, but above MemLim. *)
   (* Memory map; PO.System page 104 *)
   STACKSIZE* = 8000H;     (* 32 kB. Was hardwired in Kernel.Init *)
   STACKORG*  = 80000H;    (* stackOrg = 524,288; half a megabyte. *)
   HEAPORG*   = STACKORG;  (* heapOrg = stackOrg *)
   FSoffset*  = STACKORG;  (* Not clear what it is *)
   MEMLIM*    = 0E7EF0H;   (* 1 MB minus 98,575 bytes of VRAM at the end*)
   VRAMORG*   = 0E7F00H;   (* start of memory-mapped display frame*)
   VRAMSIZE*  = 1024*768 DIV 8; (* 1024 x 768 pixel, monocolor display frame*)
   (* heapOrg   = stackOrg on page 104; this is where the heap starts *)
   (* heapLimit = MemLim on page 104; this is where the heap ends     *)
(*-------------------------------------------------------------------------------*)
(*Hardwired memory addresses 0, 4, 12, 16, 20, 24 are explained in PO.Applications,
chapter 14.1, page 74 (2nd part of the book "Project Oberon 2013 Edition"). Address
4 is an interrupt vector explained on page 2 of RISC5.Update.pdf. All these are
memory locations accessible with PUT and GET*)

   (* Memory locations accessible with SYSTEM.GET and SYSTEM.PUT *)
   SysStartAdr*   =  0; (*Branch instruction to the init body of Modules*)
   IntVecAdr*     =  4; (*Interrupt vector*)
   MemLimAdr*     = 12; (*The limit of available memory*)
   ModAllocAdr*   = 16; (*The address of the end of the module space*)
   ModRootAdr*    = 20; (*The current root of the links of loaded modules*)
   ModLimAdr*     = 24; (* The current limit of the module area*)

  (* Copy and paste the entire following block to the relevant modules *)

  (* BLOCK starts on the next line
  (* PO.Applications, chapter 14.1, page 74; RISC5.Update page 2 *)
  SysStartAdr = SysDef.SysStartAdr; (* =  0;*)
  IntVecAdr   = SysDef.IntVecAdr;   (* =  4; *)
  MemLimAdr   = SysDef.MemLimAdr;   (* = 12; *)
  ModAllocAdr = SysDef.ModAllocAdr; (* = 16; *)
  ModRootAdr  = SysDef.ModRootAdr;  (* = 20; *)
  ModLimAdr   = SysDef.ModLimAdr;   (* = 24; *)
  BLOCK ends on the previous line*)

(*-----------------------------------------------------------------*)
(* Auxiliary definitions gathered by inspecting various modules*)
   CARD0* = 1;      (* The single SD disk with Oberon File System*)
   SPIFAST* = 4;    (* SPI mode. To be verified. Fast is "2" in SCC.Mod *)
(*-----------------------------------------------------------------*)
(* Gaining access to registers at higher level w/o importing SYSTEM
will require a few regular exported procedures of the form GetReg,
LoadReg, Get, Put. These are defined below.*)

(* SYSTEM services can be used at higher level w/o importing SYSTEM.*)
(* These services cannot be imported by a standalone MODULE*
 Therefore, if you want to use SysDef.Mod for the BootLoader, then
the following must be commented out.*)

   (*The following four procedures are probably a bad idea.*)

   PROCEDURE LoadReg* (adr, value: INTEGER);
     BEGIN SYSTEM.LDREG (adr, value) END LoadReg;

   PROCEDURE GetReg* (adr: INTEGER): INTEGER;
     VAR value: INTEGER;
     BEGIN value := SYSTEM.REG (adr); RETURN value END GetReg;

   PROCEDURE PutMem* (adr, value: INTEGER);
     BEGIN SYSTEM.PUT (adr, value) END PutMem;

   PROCEDURE GetMem* (adr: INTEGER): INTEGER;
     VAR value: INTEGER;
     BEGIN value := SYSTEM.GET (adr); RETURN value END GetMem;
```

END SysDef.

```
⇨ (* From BootLoadDisk;  AP 1.11.2017 *)
   MT* = 12;              (* Module Table*)
   SP* = 14;              (* Stack Pointer ??*)
   LNK* = 15;             (* Link register ??*)
   MTOrg* = 20H;          (* Related to Module Table, to be explained*)
   MemLim* = 0E7EF0H;     (* 1 MB minus 98,575 bytes of VRAM at the end*)
   stackOrg* = 80000H;    (* 524,288. Not clear what it is *)
   spiData* = -48;        (* SPI data is available at this address*)
   spiCtrl* = -44;        (* SPI ctrl register at this address*)
   CARD0* = 1;            (* The single SD disk with Oberon File System*)
   SPIFAST* = 4;          (* SPI command code ?? To be explained*)
   FSoffset* = 80000H;    (* 256MB in 512-byte blocks; what does it mean??*)
   MEM12* = 12;           (* Mystery memory location 12 ??*)
   MEM24* = 24;           (* Mystery memory location 24 ??*)

(* From BootLoad; *NW 20.10.2013 / PR 4.2.2014 *)
(*This file contains two BootLoaders and a few other modules*)
   (* MT = 12; SP = 14; LNK = 15; already done above *)
   (* MTOrg = 20H; MemLim = 0E7EF0H; stackOrg = 80000H; done above *)
   swi* = -60;          (** DIP switch *)
   led* = -60;          (** LED register ? *)
   rsData* = -56;       (** Serial port *)
   rsCtrl* = -52;       (** Serial port *)
   (* spiData = -48; spiCtrl = -44;  *)
   (* CARD0 = 1; SPIFAST = 4;  *)
   (* FSoffset = 80000H; block offset  *)
(*WS: hardwired mem addresses 12, 16, 24 in LoadFromDisk and init section *)
(*WS: hardwired mem addres 4 in Load *)

(* From Clipboard; AP 1.6.17. These locations are a  mystery because they may be
pointing to non existing memory in the FPGA board. The (lack of) comments in
Clipboard.Mod does not say whether Clipboard works with the FPGA hardware.*)
(*BTW, Clipboard only handles Text as a sequence of CHAR. *)
   control = -24;       (* Mem location where the block length is put*)
   data = -20;          (* Addr of Block of memory to put the Text*)

(* From Display; AP 2.2.2020 *)
(* MemLim = 0E7EF0H =   95,0000 does not quite agree with the Display.base *)
   base*  = 0E7F00H;  (*95,0016 adr of 1024 x 768 pixel, monocolor display frame*)

(* From Graphics; AP 2.2.2020 *)
   NameLen* = 32;

(* From Input; NW 15.5.2013 Ceres-4 *)
   msAdr* = -40;        (** mouse *)
   kbdAdr* = -36;       (** keyboard *)

(* From Kernel; *NW/PR  11.4.86 / 27.12.95 / 4.2.2014 / AP 2.2.2020 *)
   SectorLength* = 1024;
   timer* = -64;
   (* FSoffset = 80000H;256MB in 512-byte blocks already defined*)
   mapsize* = 10000H;   (*1K sectors, 64MB what does it mean? *)

(* From Kernel1; AP 2.2.2020 *)
   (* All the constants are already defined in Kernel *)

(* From Modules; NW 20.10.2013 / 8.1.2019 / AP 2.2.2020 *)
(* Many constants in Modules seem private. They do not belong to the global SysDef.
There are two const candidates to be moved to SysDef. Both these literals "16" and
"20" are used in Modules.Init without a single word of explanation. Both these
locations are system-wide and need be moved to SysDef under proper names, after
understood what these are doing. *)
   (* MT* = 12;         already defined *)
   INIT16 = 16;         (* Mystery in procedure Modules.Init*)
   INIT20 = 20;         (* Mystery in procedure Modules.Init*)

(*From Oberon0; command interpreter / AP 2.2.20 Extended Oberon System.
Some constants are private and need not be moved. The ones below look rather global
and can / should be made system-wide. Some have been defined in other modules, but
under different names. These should be renamed to become common. In addition,
register "14" is hardcoded in Oberon0.Reset. *)
   swi = -60;           (*Already defined in BootLoad as "swi" *)
   stat = -52;          (*Already defined in BootLoad as "rsCtrl" *)
   maxCode = 8000;      (*Already defined in System *)
   REG14 = 14;           (*Stack pointer register hardcoded in Reset *)

(*BW screen in Oberon0: Since Oberon0 is a standalone piece, its screen
implementation can be different from the main system. In fact it is only used in
FillDisplay, which is a crude test of the video RAM by filling it with a pattern.
This is coded by hand. Oberon0 does not import Display (!!!). If the Display
implementation is modified then this crude video test will fall apart. What is the
point of testing the video RAM not using Display services? This test makes sense
only as long as Display is not changed. In general, the test should become a test
of the actual Display implementation rather than bypassing it. More general:
Oberon0 could / should become a more thorough test of hardware and firmware, using
the actual drivers rather than bypassing them. The place for the thorough display
HW/FW test is in Display, while Oberon0 is importing the test for execution.*)
   screenbase = 0E7F00H; (*Already defined in Display as "base" *)
   screensize = 1024*768 DIV 8; (*size of 1024 x 768 pixel monocolor display *)

(*From Oberon; JG 6.9.90 / 23.9.93 / 13.8.94 / 14.4.13 / NW 22.12.15 / AP 1.12.19*)
   BasicCycle = 20;      (*Used with GC, not documented*)
   REG14 = 14;           (*Stack pointer register hardcoded in Reset *)
   GCinterval = 1000;    (*GC called every 1 second, in init module part*)
   (*In real time software the GC should be under finer control than just 1 sec.*)

   (* ESC = 1BX; SETSTAR = 1AX; TAB = 9X; CR = 0DX;
   Defining these separately in every module is strange. Why not in one place?*)

(* From PCLink0; AP 2.2.20*)
(* No candidates for SysDef.Mod*)

(* From PCLink1; NW 25.7.2013*)
   data = -56;           (*Already defined in BootLoad as "rsData" *)
   stat = -52;           (*Already defined in BootLoad as "rsCtrl" *)

(* From RS232; NW 3.1.2012*)
   data = -56;           (*Already defined in BootLoad as "rsData" *)
   stat = -52;           (*Already defined in BootLoad as "rsCtrl" *)

(* From SCC; NW 13.11.87 / 22.8.90 PR 21.7.13 / 23.12.13 / AP 1.6.17*)
   swi = -60;            (*Already defined *)
   spiData = -48;        (*Already defined *)
   spiCtrl = -44;        (*Already defined *)
   spiFast = 2;          (*Surprise! SPIFAST was defined as "4" *)

(* From FileDir;  NW 12.1.86 / 23.8.90 / 15.8.2013 *)
(*Constants relate to the file system structure rather than to the board hardware*)

(* COMPILER and Linker / Loader *)

(* From ORB;  NW 25.6.2014  / 25.1.2020   in Oberon-07 / AP 2.2.20 *)
   versionkey* = 1;      (*Version mechanism is undocumented*)

(* From ORC;  Connection to RISC; NW 11.11.2013 / AP 11.11.17 *)
   stat = -52;           (*Already defined in BootLoad as "rsCtrl" *)

(* From ORG;  N.Wirth, 16.4.2016 / 4.4.2017 / 31.5.2019 / AP 2.2.20 *)
   WordSize* = 4;        (*Can it be any different?*)
   StkOrg0 = -64; VarOrg0 = 0; (*for RISC-0 only*)
   (*The hardwired registers should be *)
   MT = 12; SP = 14; LNK = 15;(*dedicated registers should be system wide*)

   (*The following are probably internal. *)
   maxCode = 8800; maxStrx = 3200; maxTD = 160;

(* From ORL;  Oberon boot linker/loader for RISC / AP 2.2.20 *)
   versionkey = 1X;      (*Version mechanism is undocumented*)
   versionkey0 = 0X;     (*Version mechanism is undocumented*)
   MT = 12;              (*dedicated register should be system wide*)
   DescSize = 84;        (*Probably internal and harmless*)
   MnLength = 32;        (*File name length should be system wide*)
(*Warning. In procedure Link* there are hardcoded numbers 12, 16, 20, 24. It is not
clear whether the literals have anything to do with similar literal CONST's
exported by other modules (e.g., MT). If this is not merely a coincidence then
there is a potential for bugs. *)

(* From ORP;  N. Wirth 1.7.97 / 31.5.2019 / AP 2.2.20 *)
(* No candidates for SysDef.Mod*)

(* From ORS;  NW 19.9.93 / 20.3.2017 *)
(* No candidates for SysDef.Mod*)

(* From ORTool;  NW 18.2.2013 / 12.4.2017 / AP 2.2.20 *)
(* No candidates for SysDef.Mod*)

(* From ORX;  Oberon boot converter for RISC / AP 2.2.20 *)
(* No candidates for SysDef.Mod*)

(* Other modules *)

(* From PIO;  NW 16.10.2014  PIC Input/Output for RISC *)
(* It is not clear what is the role of this module in the Oberon System*)
   gpio = -32;           (*Not in the table PO.Computer page 13*)
   gpoc = -28;           (*Not in the table PO.Computer page 13*)

(* From System;  JG 3.10.90 / NW 12.10.93 / NW 20.6.2016 / AP 3.2.20 *)
   (* TAB = 9X; CR = 0DX;control characters redefined in every module*)
   LNK = 15;             (*link register redefined again*)
   MTOrg = 20H;          (*Module Table redefined again*)
   TaskPeriod = 500; (*ms*)(*retry period for the deferred trap handler*)

(*In addition there are also hardwired constants in the System code. The 0 below is
 vicious, because it is the address rather than "just another zero"*)
   name: ARRAY 32 OF CHAR;(*RenameFiles, ExtractCode*)
   15                    (*Trap, Abort*)
   0, 20H                (*Init section of the module. 0 is vicious!*)

(* From System1;  AP 2.2.20 *)
(* No candidates for SysDef.Mod*)

(* From TextFrames;  JG 8.10.90 / NW 16.11.2015 / AP 2.2.20 *)
(* No candidates for SysDef.Mod*)
(* The only annoyances are BS = 8X; TAB = 9X; CR = 0DX; DEL = 7FX; *)
(* From Texts;  JG 21.11.90 / NW 11.7.90 / 24.12.95 / 22.11.10 / 18.11.2014 /
10.1.2019 / AP 2.2.20 *)
(* The usual annoyances are TAB = 9X; CR = 0DX; *)
(* No candidates for SysDef.Mod, except names of 32 chars hardwired in code*)
   FName: ARRAY 32 OF CHAR;(*Scanner, Load*)

(* From Tools;  NW 22.2.14 / AP 1.5.19 *)
(* In procedure Id there is a direct call to SYSTEM.H(1)to get processor ID. This
will not survive if the processor implementation is changed. *)
   1                      (*method to obtain processor ID*)

(*In this module TAB is not defined as a constant, but hardcoded as 9X, while there
is a comment next to it "TAB". The following are hardcoded in ShowFile, Convert, *)
   9X, 0DX, 0AX           (*  ShowFile, Convert, *)
   20H                    (*  ShowFile *)
   bunch of other literals(* Sector, Inspect*)

(* From Viewers;  JG 14.9.90 / NW 15.9.13 / AP 15.4.19 *)
(* No candidates for SysDef.Mod*)


(* It is not certain whether this module needs to execute any code.
Why yes: This module is at the very bottom of the tree. It defines
the system addresses. If these are also given values, then the good place
is here. In case it turns out that the registers are initialized only once,
then the values need not be exported at all, making the OS simpler
to interface. Namely, we would then avoid a double interface (both the
register content and exported constants.) We could just use the registers,
while the constants remain hidden.

Why not: At present, the system addresses are initialized in other modules. The
minimal use of SysDef.Mod is to provide the constant definitions, while all the
executable code stays where it is now.*)

BEGIN (* Init section*)
END SysDef.
⇦
```