Blaise Gassend
François-Marie Lefevere

Tuesday may 8th 2000

DUST

Developer's Universal SX Tool

User Manual

Introduction

Dust is a SX micro-controller simulator and an electronic circuit simulator made to work together in order to make complex SX applications easy to debug.
The simulator has an interactive graphical environment for debugging. The electronic circuit that you want to debug is defined in a native language.
Before creating circuits it is recommended that you read the section of the Technical Manual that discusses the name space and the simulation algorithm.
If you are unsure of a word, try the glossary.

File Format

General comments:

- The file is made up of definitions separated by semicolons.

The `#include` directive can be used to include external files.

- There are three different contexts in which definitions are accepted : The global context, the component definition context and the display box context.
- Except for components, symbols appearing in the global context are used to create startup windows.
- In the following, square brackets indicate optional items.

Commenting your circuits

C++ style comments are used. No nested comments.

Valid examples :
```
/* ... */
/*  /*  */
// ....
```
Bad example : `/* /* */ */`

Referring to a symbol in the namespace

To refer to a symbol you use the following notation :

```
PARENT1/PARENT2/.../PARENTn/NAME
```

The symbol is first searched for from the current location, then from the root of the tree, then from the main component.
In some cases (plugs and variables) it is possible to represent a whole range of symbols indexed by integers, using a compact notation :

Example : `MEM/11~10/VAL/1~0`

This is the same as `MEM/11/VAL/1:MEM/11/VAL/0:MEM/10/VAL/1:MEM/10/VAL/0`. This notation will be useful with plugs and variables.

Warning : There isn't much checking about names with double definitions. If a name is defined twice the behavior is unpredictable. So avoid naming two symbols the same way, and if things seem really strange, try changing names.

Including the contents of a file

Usage : `#include "filename"`
Context : at the start of any line
Example : `#include "flop.circuit"`

Includes the specified file. Can appear at the start of any line.

Defining a component type

Usage : `component NAME { CONTENTS };`
Context : global
Example : `component empty {};`

Defines the component type NAME.

Creating a component

Usage : `COMPONENTTYPE NAME1[,NAME2[...]];`
Context : component definition
Example : `and and1, and2;`

Inserts components of type `COMPONENTTYPE` with the names `NAME1, NAME2`, ...

Connecting pins together

Usage : `net [NAME] PIN1[-PIN2[...]];`
Context : component definition
Example : `net IN and1/IN-or2/IN;`

Connects the specified pins together with a net of name net. The pins can only come from symbols under the current symbol to avoid having nets connected to themselves. There might be cases in which the program doesn't detect that a net is referring to itself, try to avoid such situations.
Nets are considered to be pins, thus a *macrocomponent* can connect to the nets of its children.

Inserting a symbol defined elsewhere

Usage : `symbol [NAME] SYMBOL;`
Context : global
Example : `symbol OUT and1/out;`

Inserts a copy of the specified symbol. Is often useful for displaying a symbol in a `hbox` or `vbox`, but also in a *macrocomponent* to mirror a constituent component's symbol.

Grouping symbols for display purposes

Usage :
```
hbox [NAME] { CONTENTS };
vbox [NAME] { CONTENTS };
```
Context : global, symbol definition or display box
Example : `hbox { symbol AND1/OUT; symbol OR1/OUT; };`

Groups the listed symbol horizontally (`hbox`) or vertically (`vbox`) for the display. Using `hbox`es and `vbox`es it is easy to create a good looking interface for any specific circuit.

Grouping symbols for naming purposes

Usage : `plug [NAME] <SYMBOL1[:SYMBOL2:[...]]>;`
Context : global, symbol definition or display box
Example : `plug A <AND4/STATUS:AND3/STATUS:AND2/STATUS:AND1/STATUS>;`

In the preceding example it will now be possible to call `AND3/STATUS` by the name `A/3`. The following example shows that symbols put together in a plug can sometimes be named in a more flexible way :

Example : `plug B <A/3~1>;`

This notation is equivalent to `plug B <A/3:A/2:A/1:A/0>;`
Note that the numbers are in decreasing order. This is to be compatible with variables.
Note that it is possible to skip some numbers :

Example : `plug P <A:B::C>;`

In this case `P/1` will be a *nullsymbol*.
In some cases it can be interesting to give names to the different elements of a plug instead of numbers. To do this you have to define a `plugtype` :

Usage : `plugtype NAME <PATH1:PATH2:...:PATHn>;`
Context : global, symbol definition or display box
Example : `plugtype MYBUS <DATA/7~0:ADDRESS/15~0:READ:WRITE>;`

This will define a plug type in which the symbols will bear names instead of numbers. To create a symbol on such a model use the following syntax:

Usage : `PLUGTYPE [NAME] <SYMBOL1[:SYMBOL2[...]]>;`

Context : global
Example : `MYBUS MEMPLUG <MEM/DATA/7~0:MEM/ADDR/7~0:MEM/R:MEM/W>;`

Connecting many pins at a time with a bus

Usage : `bus [NAME] PLUG1[=PLUG2[...]];`
Context : component definition
Example : `bus MAINBUS MEMPLUG=PROCESSORPLUG;`

The various plugs must only be made up of pins. Corresponding pins will be connected. Note that the bus can later be used as a plug, just as a net can be used as a pin.
With bus it is only possible to connect plugs that are of exactly the same plug type. If you want to connect incompatible buses, you can use a `mixedbus`. Mixed buses are used in exactly the same way as busses between untyped plugs.

Defining your own variables.

Usage : `var [NAME] <BIT1[:BIT2[...]]>;`
Context : global, symbol definition or display box
Example : `var LENGTH <SCENIX/BANK_0/9~8>;`

Creates a variable by putting together bits from other variables. It is possible only to take some bits from the variables :

Example : `var FOO <BAR/4~3>;`

Takes bits 4 and 3 from `BAR` for variable `FOO`.
Note that bits are specified with most significant bit first.

This isn't very flexible, there are plenty of other things I want to choose!

Actually, most of the other things you want to choose are very component specific, so there is a general way of specifying them called attributes, that can adapt to a wide variety of situations.

Usage : `attribute1=value1[, attribute2=value2[...]];`
Context : global, symbol definition or display box
Example : `base = 10, Vcc = 6;`

The specified attribute will apply to all symbols in the current context appearing after its definition. It will also apply to all the children of these symbols.
It is also possible to specify an attribute that only applies to one symbol, here is an example :

Example : `variable FOO(base = 2) <BAR>;`

This attribute will be propagated to all of `FOO's` children in the symboltree. For *hbox* and *vbox* attributes that are specified here are not propagated to the children.
The values that an attribute can take are : integer, floating point, string, name and null. It is also possible for an attribute to take on the value of an attribute that was defined before it. In this way *macrocomponents* can receive parameters that they can distribute to their children.

Example : `base = 2; Vcc = 5.234; title = "Hello"; Vdd = ; child = AND1; f = %freq;`

In this example, base will have the value 2, `Vcc` will have the value 5.234, title will represent the string "Hello", `Vdd` will be undefined, child will hold the name `AND1` and `f` will contain the same value as `freq`.
It is possible to use metric multipliers when specifying floating point values. To do this, write an underscore followed by the multiplier you want. Valid multipliers are (in increasing order) : `a, f, p, n, µ (-or u), m, c, d, (blank), da, h, k, M, G, T, P.`

Example : f = 66_M

This sets the frequency to 66 Mhz.

Examples

The Dust distribution contains a number of examples that should allow you to become more familiar with this language.

Grammar

A full grammar of the language can be found at the end of this document if you want a more precise description.

Linux Interface

The following section applies to the linux version only.

Starting a session

Usage : `xdust source1 [source2 [...]]`

The program will parse the files in the order in which they appear. Only one file can contain a main component.
At startup, windows will be created for each symbol appearing in the global context. If no symbols are defined there, a window displaying the symboltree will appear.
Keep an eye open for error messages in the window you ran *xdust* from.

Running your circuit

Most of what follows can be done with the keyboard. Have a look at the right mouse button menu.

Resetting the circuit

The *Reset* button Resets the whole circuit.

One simulation step at a time

With the *One Tick* button, one simulation step is executed.

One SX instruction at a time

With the *One Trace* button, one SX instruction is executed. If there are many SXs, the first one to have finished an instruction stops the execution.

Skipping over calls

With the *One Step* button, one SX instruction is executed, except that this time calls are skipped over. If there are many SXs, the first one to have finished a step stops the execution.

Running continuously

There are many ways to run continuously. The simplest way is to use the *Go* button. The circuit is simulated without being updated. When you have had enough you can press *Stop* to stop.
If you want to have updates every once in a while, you can use the *Fast* button. The *Fast speed* edit box allows you to choose how many ticks go by between refreshes.
If you want to update after each Tick, Trace or Step, use the *Animate* buttons.
The *Slow* buttons leave a delay in miliseconds between Ticks, Traces or Steps that you can set with the *Slow delay* box.

Running to the cursor

If you want to run until the cursor is reached, use the *To Cursor* button.

Changing parameters

The right mouse button opens up a menu that allows you to interact with the selected symbol. Most of the functions that you find here have keyboard shortcuts that are indicated in the menu.

Creating an interface interactively.

By clicking on the *Edit Mode* button of a window, you get a new toolbar that allows you to modify the interface interactively. Some of the actions you can do in edit mode are also possible in run mode, most are not.

Selecting a symbol

When you move the mouse around the window, parts of it gets highlighted. The highlighted part is the selected symbol.

Moving a symbol to another position

Use the middle mouse button to drag the selected symbol from one place to the other. The symbol has to be dropped on the lines that appear between adjacent symbols.

Getting rid of a symbol

Drag the symbol onto the Trash Can button with the middle mouse button to get rid of it.

Creating a new displaybox

Drag the *New Horizontal* or *New Vertical* display box button to the desired place with the middle mouse button.

Inserting a project symbol

Drag the *New Project* symbol to the desired place with the middle mouse button.

Creating a new window

The *New Project, New Horizontal* and *New Vertical* buttons create a new window when clicked on. The New Window From Selection button creates a new window with the selected symbol when you drag onto it with the middle mouse button.

Saving a window

Once you have set up your interface the way you want it, you can export it using the E*xport* button. You may have problems exporting anonymous symbols. Boxes will loose their names.

Adding symbols from the project tree to your interface

You can add symbols from the project tree to your interface by dragging them with the left mouse button.

Inserting a symbol by using its name

You can use the *Insert Symbol* edit box to type in the name of a symbol. The box is grayed out when the symbol name is invalid. When you press Enter, the program tries to complete the name you typed. Pressing enter many times cycles through the different possibilities.
Once a valid name is typed in, you can drag it to the desired location with the left mouse button.

Windows interface

The following section applies to the Windows version only.

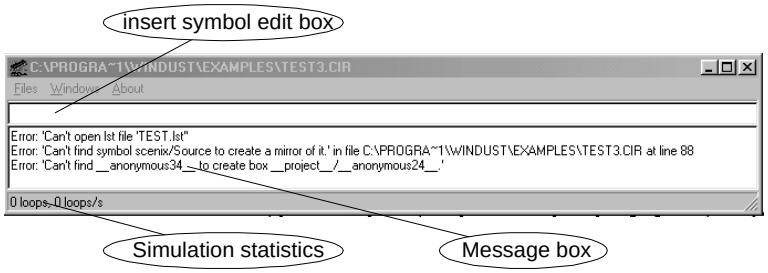
Starting a session

Usage : `windust [source]`

The program will parse the source file if there is any specified.
At startup, the program will display the main window and additional windows will be created for each symbol appearing in the global context.
You can also open a source file from then *Files*|*Open* menu in the Main window of the application.

Main Window

Once you have run Windust, this is the first windows that appears. A screenshot of this window if shown below:

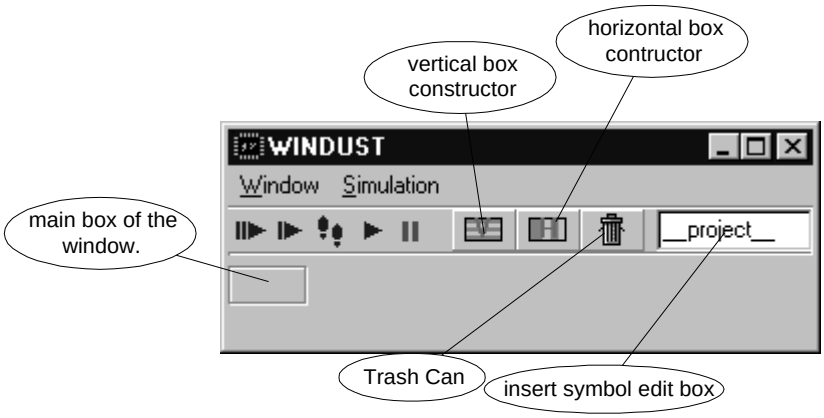


The *message box* displays error messages that occurs when a file is parsed.

Creating an interface interactively.

Creating a new *symbol view* window :

This can be done from the *Windows*|*New window* menu in the Main window. You will obtain the window below:



This window contains a main box where symbols can be dropped. If this box is destroyed the window is destroyed immediately.

Adding a new symbol :

There are two ways of adding symbols with the interface:

- using an *insert symbol edit box*.

Just type the name of the symbol you want to display in an *insert symbol edit box*, and drag it over a box or over another symbol. When you drop it, the new symbol is created and added in the window. (see the *Moving a symbol* section for more details about dropping symbols). If the name is invalid then an error message appears in the *message box* of the main window.

- using a *project widget*.

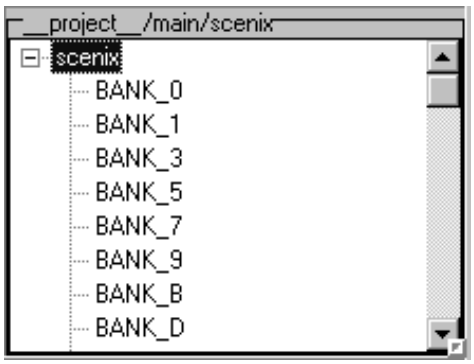
Just select the symbol you want to display in the tree and drag it over a box or over another symbol. (see the *Moving a symbol* section for more details about dropping symbols)

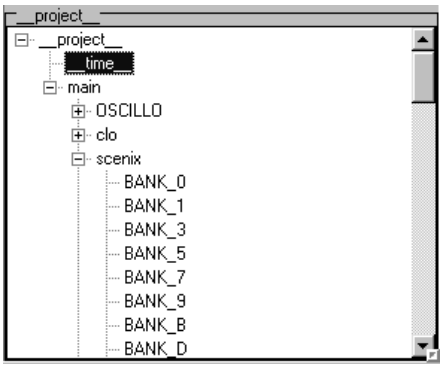
Creating a project widget :

A project widget is a widget containing a tree representing the hierarchy of the components, symbols and pins in the application.

If you type __project__ in an *insert symbol edit box* you will obtain, using drag & drop, a project widget containing the hierarchy of the whole simulation project.

If you type the name of a component that doesn't export a specific viewer, you will obtain a project widget whose root will be that component. Some examples of project widgets are shown below :





Creating a new horizontal/vertical box :

Horizontal and vertical boxes are fundamental widgets in the application. They are containers for all other widgets. A horizontal box puts all the widgets it contains in the same line while a vertical box will put all its widgets in a column.

To create a box, drag the *horizontal or vertical box constructor* button to the desired place.

Note : the type of the box (horizontal or vertical) can be modified after the creation of the box.

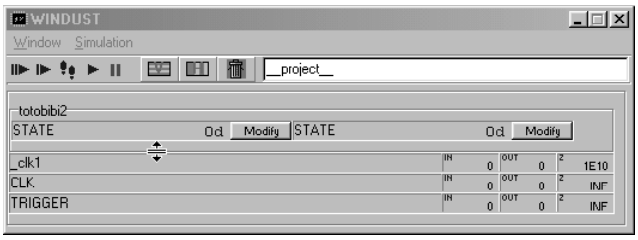
Deleting a symbol :

Drag the symbol onto a *Trash Can* button.

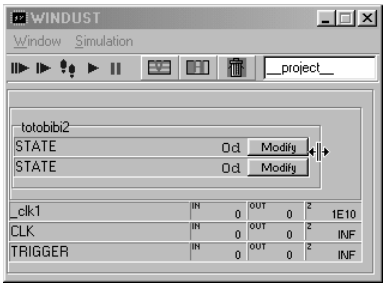
Note : Deleting the main box of a window will destroy it.

Moving a symbol :

Use the left mouse button to drag the symbol from one place to another.
When you drop the symbol onto a symbol other than a box, the symbol will be inserted after the symbol it is dropped on. When you drop the symbol over the title of a box, the symbol will be inserted in the box and will become the first item. If you drag the symbol over the bottom side or the right side of a box, the drag cursor will change of appearance. If you drop the symbol at this place, the symbol will be inserted after the box.



Insertion of a symbol after the *totobibi2* box. The *totobibi2* box is contained in a vertical box.



Insertion of a symbol after the *totobibi2* box. The *totobibi2* box is contained in a horizontal box.

Modifying display parameters of a symbol :

Each symbol has its own popup menu accessible through a right click on the symbol. With this menu, it is possible to modify parameters like :

- the length of the symbol name (full name, short name ...)
- the type of value being displayed (byte, word , dword, signed, unsigned)
- the type of box (horizontal, vertical)
- the base of the value (binary, octal, decimal, hexadecimal)

Modifying symbol sizes :

Symbols containing a small square at the bottom-right corner can be resized by dragging the square.
Boxes can be resized if the option *Auto size* in their popup menu is not checked.
Note : some sizes are imposed by the container :

- symbols contained in a vertical box will have their minimum width imposed by the minimum width of other symbols. To modify the width, you can modify the size of the container.
- symbols contained in a horizontal box will have their minimum height imposed by the minimum height of other symbols. To modify the height, you can modify the size of the container.

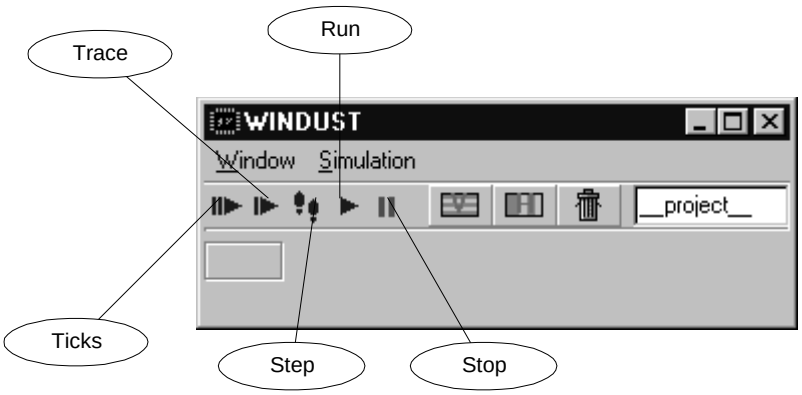
Saving a window or a box :

You can export the text description of a box to the clipboard using the popup menu of the box. The same operation can be done for a window, using the *Window|Export to clipboard menu*.

Running a circuit.

Execution control :

The execution of the simulation is controlled by the *Simulation* menu and by the buttons of the toolbar of a symbol view window.



Ticks runs one elementary step of the circuit simulation.
Trace will stop after the execution of each instruction of the micro-controller. (if there are several micro-controllers, the simulation will stop for each of them)
Step will stop after the execution of each instruction of the micro-controller but in case of a call, the simulation will stop after the return.
Run executes the simulation until the stop button is activated. Several refresh modes can be chosen in the *Simulation|Refresh rate* menu.
Note: The selection of the refresh mode can dramatically change the speed of the simulation.

Animate mode :

You can follow the execution of your circuit at a slow run speed with the *Simulate|animate* menu. In this mode, the simulation kernel will be called periodically with a period of *m* milliseconds between each call. The constant *m* can be changed with the *Simulate|Set animation speeds* option through the *Time by step* edit box.

Three options are available:
- By step.
- By trace.
- By ticks.

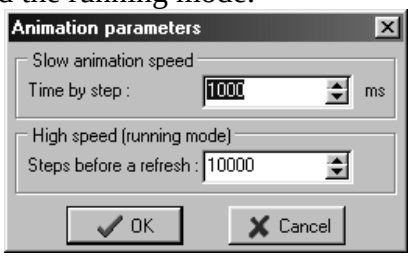
These options are the animate versions of the *step*, *trace* and *ticks* options presented above.
To stop the animation : use the stop button.

Refresh rate :

Three refresh rates are available in the *Simulate*|*Refresh rate* menu.
- *Normal* means that widgets will be refreshed every step.
- *Fast animation* means that widgets will be refreshed one time for *n* simulation ticks. The constant *n* can be changed with the *Simulate*|*Set animation speeds* option through the *Steps before a refresh* edit box.
- *No refresh* disables refresh.

Animation parameters :

This dialog box is accessible with the *Simulate*|*Set animation speeds* menu. It allows you to configurate the animate mode and the running mode.



See the refresh rate section and the Animate section for more details.

Breakpoints :

To set breakpoints, use the popup menu of disassembly widgets. At the present time, only simple breakpoints are available.

Pin types

Pin EDGE

Description :

Input pin with detection of high to low or low to high transition. Specific pin used in the sx28 component. The pin configuration is only accessible by the program of the micro-controller.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the pin : 0 or 1
DETECT	VARIABLE	A transition has been detected if this bit is set.

Attributes :

None.

Pin FLOATPIN

Description :

General analog output pin with null impedance.

Symbols :

None.

Attributes :

None.

Pin INCOMPPIN

Description :

Universal logic input pin.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the pin : 0 or 1

Attributes :

Name:	Type :	Default	Description :
Vcc	Real	5	
Vdd	Real	0	
Vtrig	Real	$\frac{Vcc+Vdd}{2}$	Voltage threshold between low level state and high level state.

Pin INPUT

Description :

5V logic trigger input pin with infinite impedance. The low to high threshold is set to 1V and the high to low threshold is set to 4V.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the pin : 0 or 1

Attributes :

None.

Pin OSCILLOINPUT

Description :

Oscilloscope input . Impedance 10G Ohms.

Symbols :

None.

Attributes :

None.

Pin OUTPIN

Description :

Universal logic output pin.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the pin : 0 or 1

Attributes :

Name:	Type :	Default	Description :
Vcc	Real	5	High level voltage.
Vdd	Real	0	Low level voltage.
Zout	Real	0	Output impedance.

Pin OUTPUT

Description :

5V logic output pin with 25 ohms impedance.

Symbols :

Name:	Type :	Description :
LATCH	VARIABLE	State of the pin : 0 or 1

Attributes :

None.

Pin SCENIXPIN

Description :

General IO pin for the SX micro-controller family.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the pin as input : 0 or 1.
TRIS	VARIABLE	Direction of the pin: 1 = input, 0 = output.
LATCH	VARIABLE	State of the pin as output : 0 or 1.
PULLUP	VARIABLE	State of the pull-up resistor : 1= no pull-up, 0 = 20Kohms pull-up resistor.

Attributes :

None.

Components

Component and

Description :

Two inputs AND gate.

Pins :

Name:	Type :	Description :
A, B	INCOMPPIN	Inputs.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pins attributes.

Component clock

Description :

Square clock.

Pins :

Name:	Type :	Description :
O	OUTPIN	Signal output.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the output pin : 0 or 1

Attributes :

Name:	Type :	Default	Description :
F	Real	$1^{E}6$	Frequency of the clock.
T	Real	1^{E}-6	Period of the clock.

Note : Pin levels can be defined using pins attributes.

Component flipflop

Description :

Bistable flip-flop

Pins :

Name:	Type :	Description :
S	INCOMPPIN	Set input.
R	INCOMPPIN	Reset input.
Q	OUTPIN	Output.
NQ	OUTPIN	Inverted output.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	State of the output pin : 0 or 1

Attributes :

No attribute defined for the flipflop, but pin levels can be defined using pins attributes.

Component motor

Description :

Linear motor driven by a PWM signal, with incremental quadrature encoder.

Pins :

Name:	Type :	Description :
PWM	INCOMPPIN	Pulse Width Modulation input.
DIR	INCOMPIN	Direction input.
Q1	OUTPIN	Quadrature output 1.
Q2	OUTPIN	Quadrature output 2
PosPin	FLOATPIN	Output pin giving a voltage corresponding to the position.
VelPin	FLOATPIN	Output pin giving a voltage corresponding to the velocity.
AccelPin	FLOATPIN	Output pin giving a voltage corresponding to the acceleration.

Symbols :

Name:	Type :	Description :
Pos	REALPTR	Position.
Vel	REALPTR	Velocity.

Accel	REALPTR	Acceleration.
ExtForce	REALPTR	External force.

Attributes :

Name:	Type :	Default	Description :
A, B	Real	1 1	Acceleration = A * STATE(PWM) * SIGN(DIR) - B * velocity
C	Real	1	Quadticks = x * C
Timestep	Real	.0001	Step size for numerical simulation of motion equations.

Note : Pin levels can be defined using pins attributes.

Component nand

Description :

Two inputs NAND gate.

Pins :

Name:	Type :	Description :
A, B	INCOMPPIN	Inputs.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pins attributes.

Component nor

Description :

Two inputs NOR gate.

Pins :

Name:	Type :	Description :
A, B	INCOMPPIN	Inputs.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pins attributes.

Component not

Description :

NOT gate.

Pins :

Name:	Type :	Description :
A	INCOMPPIN	Input.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pins attributes.

Component or

Description :

Two inputs OR gate.

Pins :

Name:	Type :	Description :
A, B	INCOMPPIN	Inputs.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pins attributes.

Component oscillo

Description :

Multi input oscilloscope. Any number of inputs can be used, just invoke a pin name starting with '_'. The oscilloscope starts sampling after a low to high transition on the Trigger input. It stops when the end of the memory is reached and restart when there is a new low to high transition on the Trigger input.

Pins :

Name:	Type :	Description :
CLK	INPUT	Sample clock. (Sampling on rising edge)
TRIGGER	INPUT	Trigger : start sampling after a low to high transition.
_XXXX	OSCILLOINPUT	Signal input pins.

Symbols :

This component exports a symbol which name is the same as the component name. The type of the symbol is OSCILLO.

Attributes :

Name:	Type :	Default	Description :
Memsize	Integer	1024	Number of samples that can be stored on each input.

Note: pin levels can be defined using pin's attributes

Component powerdc

Description :

Power supply.

Pins :

Name:	Type :	Description :
VCC	OUTPIN	Power output.

Symbols :

Name:	Type :	Description :
STATE	VARIABLE	Turn on/off the output of the power supply

Attributes :

No attribute defined for the gate, but pin levels can be defined using pins attributes. The output voltage can be configured using the following attribute:

Name:	Type :	Default	Description :
Vcc	Real	5	Voltage of the output pin.

Component sx28

Description :

8 bits RISC micro-controller.

Pins :

Name:	Type :	Description :
MCLR	INPUT	Master Clear reset input – active low
OSC1	INPUT	External clock source input
OSC2	OUTPUT	Not implemented yet.
RTCCPIN	EDGE	Input to Real Time Clock/Counter
RA0..RA3	SCENIX_PIN	Port A bi-directional I/O pins.
RB0	SCENIX_PIN	Port B bi-directional I/O pin; MIWU input; analog comparator Output
RB1	SCENIX_PIN	Port B bi-directional I/O Pin; MIWU input; analog comparator negative input
RB2	SCENIX_PIN	Port B bi-directional I/O pin; MIWU input; comparator Positive input
RB3..7	SCENIX_PIN	Port B bi-directional I/O pins; MIWU inputs
RC0..RC7	SCENIX_PIN	Port C bi-directional I/O pins.

Symbols :

Name :	Type :	Description :
STACK	ARRAY	Content of the stack used by the SX28 to manage CALL and RET functions. 2 or 8 levels, depending of the value of Fuse and Fusex.
ROM	ARRAY	Content of the program ROM. (not yet implemented on Linux platform)

BANK_0	ARRAY	Registers from 0x08 to 0x0F.
BANK_1	ARRAY	Registers from 0x10 to 0x1F.
BANK_3	ARRAY	Registers from 0x30 to 0x3F.
BANK_5	ARRAY	Registers from 0x50 to 0x5F.
BANK_7	ARRAY	Registers from 0x70 to 0x7F.
BANK_9	ARRAY	Registers from 0x90 to 0x9F.
BANK_B	ARRAY	Registers from 0xB0 to 0xBF.
BANK_D	ARRAY	Registers from 0xD0 to 0xDF.
BANK_E	ARRAY	Registers from 0xF0 to 0xFF.
Code	DISASSEMBLY	Disassembly window : display only lines containing opcodes. Lines are sorted by increasing ROM addresses.
Source	DISASSEMBLY	Disassembly window : display all the lines contained in the lst file. Lines are sorted by increasing number
PC	VARIABLE	Program Counter.
FUSE, FUSEX	VARIABLE	Registers containing the configuration of the SX.
C	VARIABLE	Carry bit.
DC	VARIABLE	Digital Carry bit.
Z	VARIABLE	Zero bit.
PD	VARIABLE	Power down bit.
TO	VARIABLE	Watch Dog Timeout bit.
PAGE	VARIABLE	Program memory page selection bits.
W	VARIABLE	Working register
INDF	VARIABLE	Indirect Addressing through FSR : the register location pointed to by FSR is accessed.
RTCC	VARIABLE	Real-Time Clock/Counter.
MODE	VARIABLE	Mode register
OPTION	VARIABLE	Option register
STATUS	VARIABLE	Contains the device status bits like C,Z,DC
FSR	VARIABLE	File Select Register.
RA, RB, RC	VARIABLE	Value of ports A,B,C inputs.
TRISA, TRISB, TRISC	VARIABLE	Direction of ports A, B, C
R_OUTA, R_OUTB, R_OUTC	VARIABLE	Value of ports A, B, C outputs
PLPA, PLPB, PLPC	VARIABLE	Pull-ups of ports A, B, C

Attributes :

Name :	Type :	Default	Description :

File	String	-	Name of the code file: supports both .HEX files and .LST files
Fuse	Integer	File value	Value of the Fuse configuration word.
Fusex	Integer	File value	Value of the Fusex configuration word.

Component xnor

Description :

Two inputs XNOR gate.

Pins :

Name:	Type :	Description :
A, B	INCOMPPIN	Inputs.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pin's attributes.

Component xor

Description :

Two inputs XOR gate.

Pins :

Name:	Type :	Description :
A, B	INCOMPPIN	Inputs.
O	OUTPIN	Output.

Symbols :

None.

Attributes :

No attribute defined for the gate, but pin levels can be defined using pin's attributes.

Symbols

Symbol ARRAY

Description :

Array of integer values.

Attributes :

Name:	Type :	Linux	Windows	Default	Description :
base	Integer	X	X	10	Base : 2 = binary , 8 = octal,10 = decimal, 16 = hexadecimal ...
columns	Integer	X	X	4	Number of columns.
datawidth	Integer	X	X	8	8 = bytes, 16 = 16-bits words, 32 = 32-bits wordsl
disptitle	String	X	X	-	title
fullname	Boolean	X	X	0	0 = short name, 1 = full name
height	Integer		X	169	Height of the array widget in pixels.
reduced	Boolean		X	0	0 = normal display mode, 1 = reduced mode (no title). This mode is used to put arrays together (example : to see the SX register memory)
signed	Boolean	X	X	0	0 = unsigned, 1 = signed.
title	Boolean	X	X	1	0 = no title, 1 = display title
width	Integer		X	257	Width of the array widget in pixels.

Symbol DISASSEMBLY

Description :

Displays disassembly code, breakpoints position and instruction pointer position.

Attributes :

Name:	Type :	Linux	Windows	Default	Description :
disptitle	String	X	X	-	title
Fullname	Boolean	X		0	0 = short name, 1 = full name
Height	Integer		X	153	Height of the disassembly widget in pixels.
Title	Boolean	X	X	1	0 = no title, 1 = display title
Width	Integer		X	265	Width of the disassembly widget in pixels.

Symbol HBOX

Description :

Horizontal widget container.

Attributes :

Name:	Type :	Linux	Windows	Default	Description :
Disptitle	String	X	X	-	title
Fullname	Boolean	X	X	4	0 = short name, 1 = full name
Title	Boolean	X	X	1	0 = no title, 1 = display title

Symbol REALPTR

Description :

Displays a real value.

Attributes :

Name:	Type :	Linux	Windows	Default	Description :
Disptitle	String	X	X	-	title
Fullname	Boolean	X	X	4	0 = short name, 1 = full name
Title	Boolean	X	X	1	0 = no title, 1 = display title

Symbol OSCILLO

Description :

Displays evolution in time of logic or analogic signals.

Attributes :

Name:	Type :	Linux	Windows	Default	Description :
disptitle	String	X	X	-	title
fullname	Boolean	X	X	0	0 = short name, 1 = full name
height	Integer		X	177	Height of the disassembly widget in pixels.
title	Boolean	X	X	1	0 = no title, 1 = display title
width	Integer		X	257	Width of the disassembly widget in pixels.

Symbol VARIABLE

Description :

Displays integer value.

Attributes :

Name:	Type :	Linux	Windows	Default	Description :

base	Integer	X	X	10	Base : 2 = binary , 8 = octal,10 = decimal, 16 = hexadecimal ...
disptitle	String	X	X	-	title
fullname	Boolean	X	X	0	0 = short name, 1 = full name
signed	Boolean	X	X	0	0 = unsigned, 1 = signed.
title	Boolean	X	X	1	0 = no title, 1 = display title

Symbol VBOX

Description :

Vertical widget container.

Attributes :

	Type :	Linux	Windows	Default	Description :
disptitle	String	X	X	-	title
fullname	Boolean	X	X	4	0 = short name, 1 = full name
title	Boolean	X	X	1	0 = no title, 1 = display title

Other symbols

__project__

Symbol that represents the whole hierarchy of the project.

__time__

Real value giving the simulation time.

Glossary

***pin* :** A *component* has a number of electrical contacts that are used to connect it with neighboring *components*. These contacts are *pins*. They are characterized by their input voltage, their output voltage and their output load.
component : The elementary building block of an electronic circuit.
***macrocomponent* :** *Component* that is made of other *components*.
***net* :** Electrical connection between a number of *pins*.
***widget* :** Building block of a graphical interface (buttons, editing fields, but also boxes that can contain a number of widgets).
***attribute* :** A name associated with a value that can be used to modify a *symbol* or a *widget*. The *attribute* class stores a list of *attributes* terminated by a *nullattribute*.
***nullattribute* :** Value that is used to pass an absence of *attributes*.
***symbol* :** Element of the *namespace*.
***variable* :** Collection of bits that can be visualized or modified together.
bus : Connection between *plugs* that connect the *pins* of one *plug* with corresponding *pins* of the other *plugs*.
***plug* :** Initially a group of *pins* that can be connected to other *plugs* with a *bus*. Was generalized to be able to group any type of *symbol* in the *namespace* in a convenient way.
***symbolconstructor* :** Object that stores the characteristics of a *symbol* before it can be built.
constructor : See *constructor*.
***componenttype* :** Object used to call the *constructor* of a *component*. Mainly useful for *macrocomponents*.
***macrocomponent* :** Object used to store the characteristics of a *macrocomponent* and construct it when it is needed.
***real* :** typedef that is used for floating point values. We used double, but this is probably an overkill.
***SX* :** High speed micro-controller from S*cenix* that this is all about.
***Scenix* :** The guys who developed the *SX*.
***symbolspace* :** A tree made with *symbols* at its nodes and its leaves that contains everything you can (and want?) to manipulate.
***namespace* :** An imaginary tree that bears the *names* of the *symbols* in *symbolspace*.
***name* :** Identifies the children of a *symbol* in *symbolspace*. Is represented by an int that is associated with a string by the *symboltable.*
***symbolname* :** Identifies a *symbol* with a relative or an absolute offset. Is represented by a list of integers and displayed as **name1/name2/name3**.
***symboltable* :** Class that stores a table of strings that are frequently used and that are usually the *names* of *symbols* or *attributes*. The strings are associated with integers. Positive or zero integers represent themselves, even negative integers represent a string, odd negative integers are for *names* that weren't specified (unnamed).

***list* :** Class that implements a simple linked list.

The grammar

Here is the full grammar that is used. It might help clarify details that I may have forgotten.

```
circuit : globaldefs EOF

globaldefs :
           | globaldef SEMICOLON globaldefs
           | SEMICOLON globaldefs

globaldef : componenttype
          | boxsymbol
          | plugtype
          | attributes

componenttype : COMPONENT NAME LEFT_CURLY componentdefs RIGHT_CURLY

componentdefs :
              | componentdef SEMICOLON componentdefs
              | SEMICOLON componentdefs

componentdef : netsymbol
             | componentsymbol
             | plugsymbol
             | bussymbol
             | boxsymbol
             | varsymbol
             | symbolcopysymbol
             | attributes

componentsymbol : NAME componentconstructorlist

componentconstructorlist : componentconstructor
                         | componentconstructor COMMA componentconstructorlist

componentconstructor : symbolname

boxsymbol : HBOX LEFT_CURLY boxdefs RIGHT_CURLY
          | VBOX LEFT_CURLY boxdefs RIGHT_CURLY
          | HBOX symbolname LEFT_CURLY boxdefs RIGHT_CURLY
          | VBOX symbolname LEFT_CURLY boxdefs RIGHT_CURLY

boxdefs :
        | boxdefs boxdef SEMICOLON
        | boxdefs attributes SEMICOLON

boxdef : boxsymbol
       | symbolcopysymbol
       | plugsymbol
       | varsymbol

plugsymbol : PLUG $$1 LEFT_POINTY namelist RIGHT_POINTY
           | PLUG $$2 symbolname LEFT_POINTY namelist RIGHT_POINTY
           | NAME LEFT_POINTY $$3 namelist RIGHT_POINTY
           | NAME symbolname LEFT_POINTY $$4 namelist RIGHT_POINTY

plugtype : PLUGTYPE NAME LEFT_POINTY namelist RIGHT_POINTY

varsymbol : VAR LEFT_POINTY namelist RIGHT_POINTY
```

```
          | VAR symbolname LEFT_POINTY namelist RIGHT_POINTY

bussymbol : BUS pluglist
          | BUS symbolname pluglist

pluglist : composedname
         | composedname EQUAL pluglist

symbolcopysymbol : SYMBOL symbolname composedname
                 | SYMBOL composedname

netsymbol : NET pinlist
          | NET symbolname pinlist

pinlist : composedname
        | composedname MINUS pinlist

namelist : namelistdef

namelistdef : namelistsymbolname
            | namelistsymbolname COLON namelistdef

namelistsymbolname :
                   | NAME
                   | NAME SLASH namelistsymbolnametail

namelistsymbolnametail : extendedname
                       | range
                       | extendedname SLASH namelistsymbolnametail
                       | range SLASH namelistsymbolnametail

range : INTEGER TILDA INTEGER

composedname : NAME
             | NAME SLASH composednametail

composednametail : extendedname
                 | extendedname SLASH composednametail

extendedname : NAME
             | INTEGER

symbolname : NAME
           | NAME LEFT_PAREN attributes RIGHT_PAREN

attributes : attribute
           | attribute COMMA attributes

attribute : NAME EQUAL INTEGER
          | NAME EQUAL REAL
          | NAME EQUAL composedname
          | NAME EQUAL STRING
          | NAME EQUAL PERCENT NAME
```