

## Tarea 1

### Distribución de tiempo:

Búsqueda de información: 5 horas y 34 minutos.

Elaboración de los múltiples incisos: 2 horas y 23 minutos.

**Nota:** Esta tarea fue hecha a lo largo de varios días, los tiempos dados son la suma de dichas sesiones.

### Instalación y demostración de las herramientas:

Se tienen las figuras 1 y 2 para demostrar que se instalaron las herramientas de Icarus Verilog, GTKWave y Yosys.

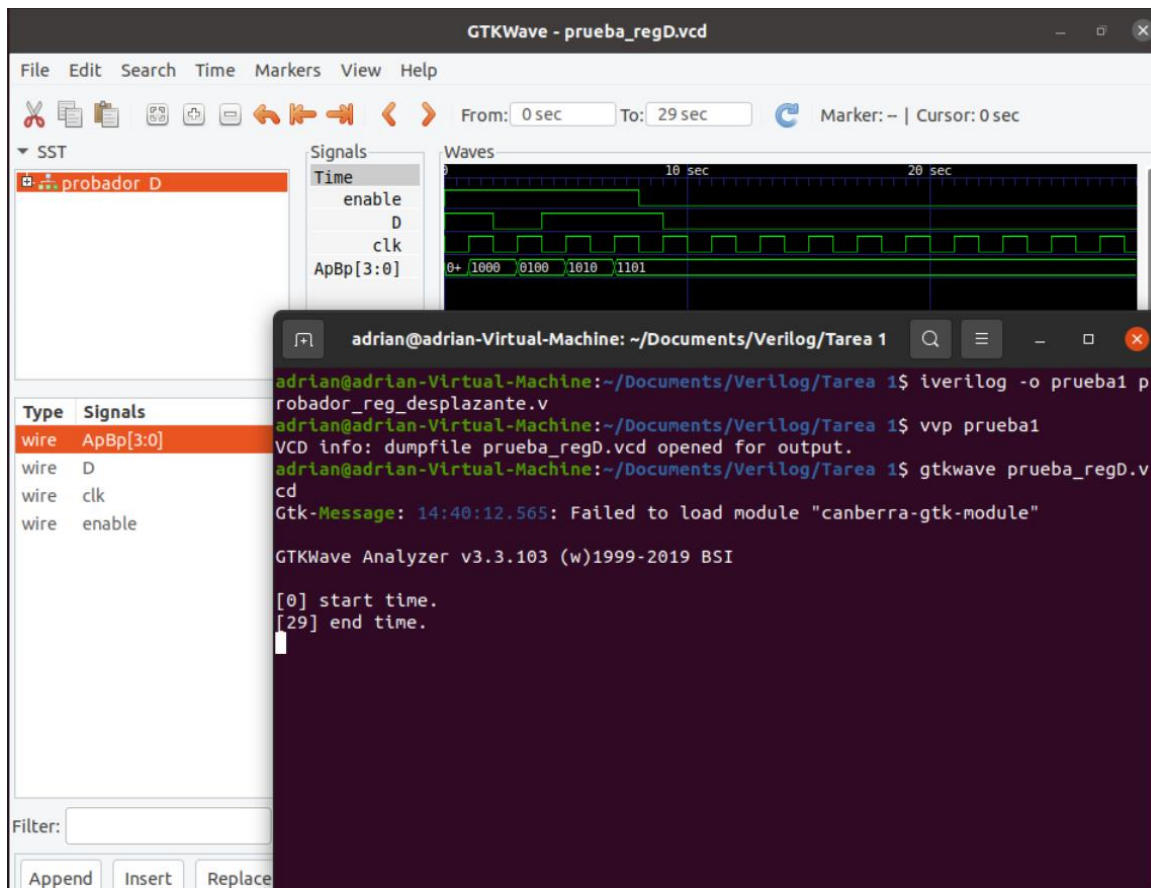


Figura 1. Utilización de Verilog y GTKWave

```
adrian@adrian-Virtual-Machine: ~  
-----  
yosys -- Yosys Open SYNthesis Suite  
  
Copyright (C) 2012 - 2019 Clifford Wolf <clifford@clifford.at>  
  
Permission to use, copy, modify, and/or distribute this software for any  
purpose with or without fee is hereby granted, provided that the above  
copyright notice and this permission notice appear in all copies.  
  
THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES  
WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF  
MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR  
ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES  
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN  
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF  
OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.  
-----  
  
Yosys 0.9 (git sha1 1979e0b)  
  
yosys>
```

Figura 2. Utilización de Yosys.

## Diseño del makefile:

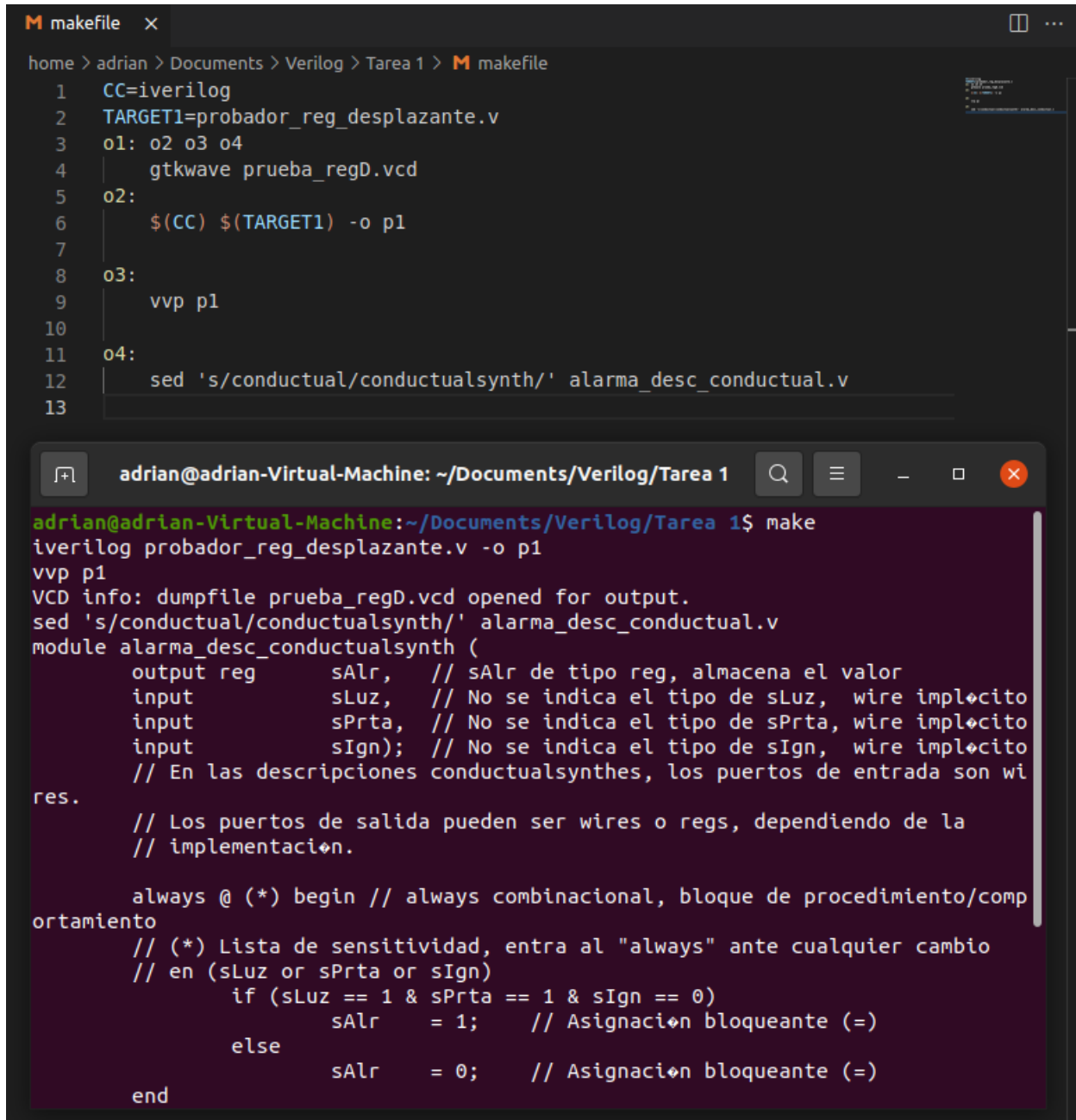
Se diseña un archivo makefile para automatizar el proceso de comprobación de resultados; el diseño propuesto se muestra en la figura 3.

```
makefile  
1 CC=iverilog  
2 TARGET1=probador_reg_desplazante.v  
3 o1: o2 o3  
4 gtwave prueba_regD.vcd  
5  
6 $(CC) $(TARGET1) -o p1  
7  
8 o3:  
9 vvp p1  
10  
  
GTKWave - prueba_regD.vcd  
File Edit Search Time Markers View Help  
From: 0 sec To: 29 sec Marker: - | Cursor: 6 sec  
SST  
probador D  
Type Signals  
wire ApBp[3:0]  
wire D  
wire clk  
wire enable  
-Signals-  
Time  
ApBp[3:0]  
D  
clk  
enable  
-Waves-  
0 8 4 A D  
9 sec 18 sec 27 sec  
adrian@adrian-Virtual-Machine: ~/Documents/Verilog/Tarea 1  
adrian@adrian-Virtual-Machine:~/Documents/Verilog/Tarea 1$ make  
iverilog probador_reg_desplazante.v -o p1  
vvp p1  
VCD info: dumpfile prueba_regD.vcd opened for output.  
gtkwave prueba_regD.vcd  
Gtk-Message: 20:56:04.477: Failed to load module "canberra-gtk-module"  
GTKWave Analyzer v3.3.103 (w)1999-2019 BSI  
[0] start time.  
[29] end time.
```

Figura 3. Diseño del makefile

## Comando sed:

El comando sed es un comando único de Unix que permite cambiar strings de un cierto archivo sin necesidad de abrir el mismo; en este caso, se cambia el nombre del módulo de modo que al final del nombre se añade la palabra “synth”. El uso de este se puede ver en la figura 4.



The screenshot shows a terminal window with a dark background. The top part shows a 'makefile' editor with the following content:

```
1 CC=iverilog
2 TARGET1=probador_reg_desplazante.v
3 o1: o2 o3 o4
4 | gtwave prueba_regD.vcd
5 o2:
6 | $(CC) $(TARGET1) -o p1
7 |
8 o3:
9 | vvp p1
10 |
11 o4:
12 | sed 's/conductual/conductualsynth/' alarma_desc_conductual.v
13 |
```

The bottom part shows a terminal window with the following output:

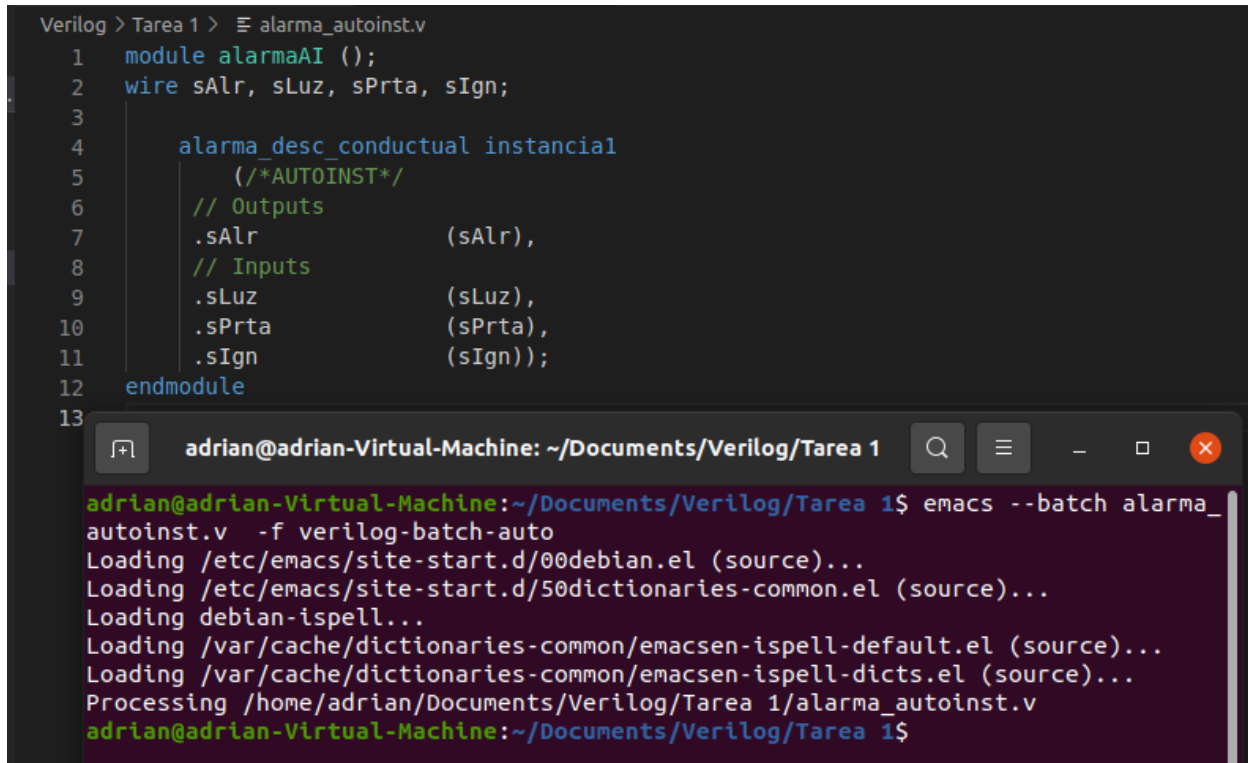
```
adrian@adrian-Virtual-Machine: ~/Documents/Verilog/Tarea 1
adrian@adrian-Virtual-Machine:~/Documents/Verilog/Tarea 1$ make
iverilog probador_reg_desplazante.v -o p1
vvp p1
VCD info: dumpfile prueba_regD.vcd opened for output.
sed 's/conductual/conductualsynth/' alarma_desc_conductual.v
module alarma_desc_conductualsynth (
    output reg      sAlr,    // sAlr de tipo reg, almacena el valor
    input           sLuz,    // No se indica el tipo de sLuz, wire implícito
    input           sPrta,   // No se indica el tipo de sPrta, wire implícito
    input           sIgn);   // No se indica el tipo de sIgn, wire implícito
    // En las descripciones conductualsynthes, los puertos de entrada son wi
res.
    // Los puertos de salida pueden ser wires o regs, dependiendo de la
    // implementación.

    always @ (*) begin // always combinacional, bloque de procedimiento/comp
ortamiento
        // (*) Lista de sensibilidad, entra al "always" ante cualquier cambio
        // en (sLuz or sPrta or sIgn)
        if (sLuz == 1 & sPrta == 1 & sIgn == 0)
            sAlr = 1;    // Asignación bloqueante (=)
        else
            sAlr = 0;    // Asignación bloqueante (=)
    end
```

Figura 4. Utilización del comando sed.

## AUTOINST de Emacs:

Con este complemento se puede llegar a realizar la instanciación de un módulo de manera automática; esta facilidad se utiliza más que todo para realizar el probador de manera rápida. En la figura 5 se puede ver la demostración de la función AUTOINST de Emacs.



The image shows a Verilog editor window at the top and a terminal window at the bottom. The Verilog code in the editor is as follows:

```
Verilog > Tarea 1 > alarma_autoinst.v
1 module alarmaAI ();
2 wire sAlr, sLuz, sPrta, sIgn;
3
4     alarma_desc_conductual instancial
5         (*AUTOINST*/
6         // Outputs
7         .sAlr          (sAlr),
8         // Inputs
9         .sLuz          (sLuz),
10        .sPrta         (sPrta),
11        .sIgn          (sIgn));
12 endmodule
13
```

The terminal window shows the command being executed and the output:

```
adrian@adrian-Virtual-Machine: ~/Documents/Verilog/Tarea 1
adrian@adrian-Virtual-Machine:~/Documents/Verilog/Tarea 1$ emacs --batch alarma_
autoinst.v -f verilog-batch-auto
Loading /etc/emacs/site-start.d/00debian.el (source)...
Loading /etc/emacs/site-start.d/50dictionaries-common.el (source)...
Loading debian-ispell...
Loading /var/cache/dictionaries-common/emacsen-ispell-default.el (source)...
Loading /var/cache/dictionaries-common/emacsen-ispell-dicts.el (source)...
Processing /home/adrian/Documents/Verilog/Tarea 1/alarma_autoinst.v
adrian@adrian-Virtual-Machine:~/Documents/Verilog/Tarea 1$
```

Figura 5. Demostración del uso de AUTOINST.