

# Instalación de las herramientas para el curso.

Gabriel Fernando Araya Mora B80525  
*Escuela de Ingeniería Eléctrica, Universidad de Costa Rica*  
*Circuitos Digitales II (IE-0523)*

---

## 1. Tiempo de ejecución de la tarea:

- **Buscar información:** Siempre antes de empezar cualquier trabajo es importante tomarse un tiempo prudente para buscar información e investigar acerca del tema. Esto es especialmente cierto cuando se trata de programar. Como ya contaba con los programas de verilog y gtkwave de cursos anteriores, esto no significó mayor problema ; sin embargo, para el programa de yosys y AUTOINST me tomó aproximadamente 2 horas encontrar la información necesaria.
- **Usando la información:** Hecho esto entonces, y jugando un poco con los archivos de verilog dados por el profesor, me tomó aproximadamente 1 hora elaborar el makefile ya que hice pruebas con cada uno de los ejemplos y archivos para asegurar la comprensión del mismo y además que todo estuviese funcionando bien.
- **Elaboración del reporte:** La elaboración del reporte me tomó aproximadamente una hora, ya que construí una plantilla de L<sup>A</sup>T<sub>E</sub>X desde cero.
- **Total:** De lo anterior se saca que el total de horas que me tomó resolver la tarea es de 4h.

## 2. Instalación de las herramientas del curso (Verilog, GTKwave, Yosys)

El sistema operativo que se va a utilizar para todo el curso, será una distro de ubuntu de linux, lo que hace que instalar estos paquetes y programas sea muy sencillo.

### ■ Verilog:

```
\$ sudo apt install verilog
```

### ■ GtkWave:

```
\$ sudo apt install gtkwave
```

### ■ Yosys:

```
\$ sudo apt install yosys
```

### 3. Ejemplos de ejecución de los programas.

- **Verilog:** Verilog es un lenguaje de descripción de hardware usado para modelar sistemas digitales. El lenguaje, soporta el diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción (Conductual y Estructural).

Para compilar archivos de verilog se usa el siguiente comando en terminal UNIX:

```
\$ iverilog <filename.v>
```

El ejemplo de un módulo conductual se presenta a continuación. Aquí se programa pensando en cómo se comporta el circuito, es decir se usa la lógica y control de flujo normal de programación.

---

```
module alarma_desc_conductual (  
    output reg      sAlr,  
    input           sLuz,  
    input           sPrta,  
    input           sIgn);  
    always @ (*) begin  
        if (sLuz == 1 & sPrta == 1 & sIgn == 0)  
            sAlr = 1;  
        else  
            sAlr = 0;  
        end  
    endmodule
```

---

El ejemplo de un módulo estructural se presenta a continuación. Aquí se trabaja con lógica digital.

---

```
module alarma_desc_estructural (  
    output  sAlr,  
    input    sLuz,  
    input    sPrta,  
    input    sIgn);  
    not #1 gate1(x, sIgn);  
    and #1 gate2(sAlr,sLuz,sPrta,x);  
endmodule
```

---

- **GtkWave:** GtkWave es un visor de onda que sirve para ver de una forma gráfica cómo se comporta el circuito digital que se está diseñando. En la siguiente figura se muestra su funcionamiento.

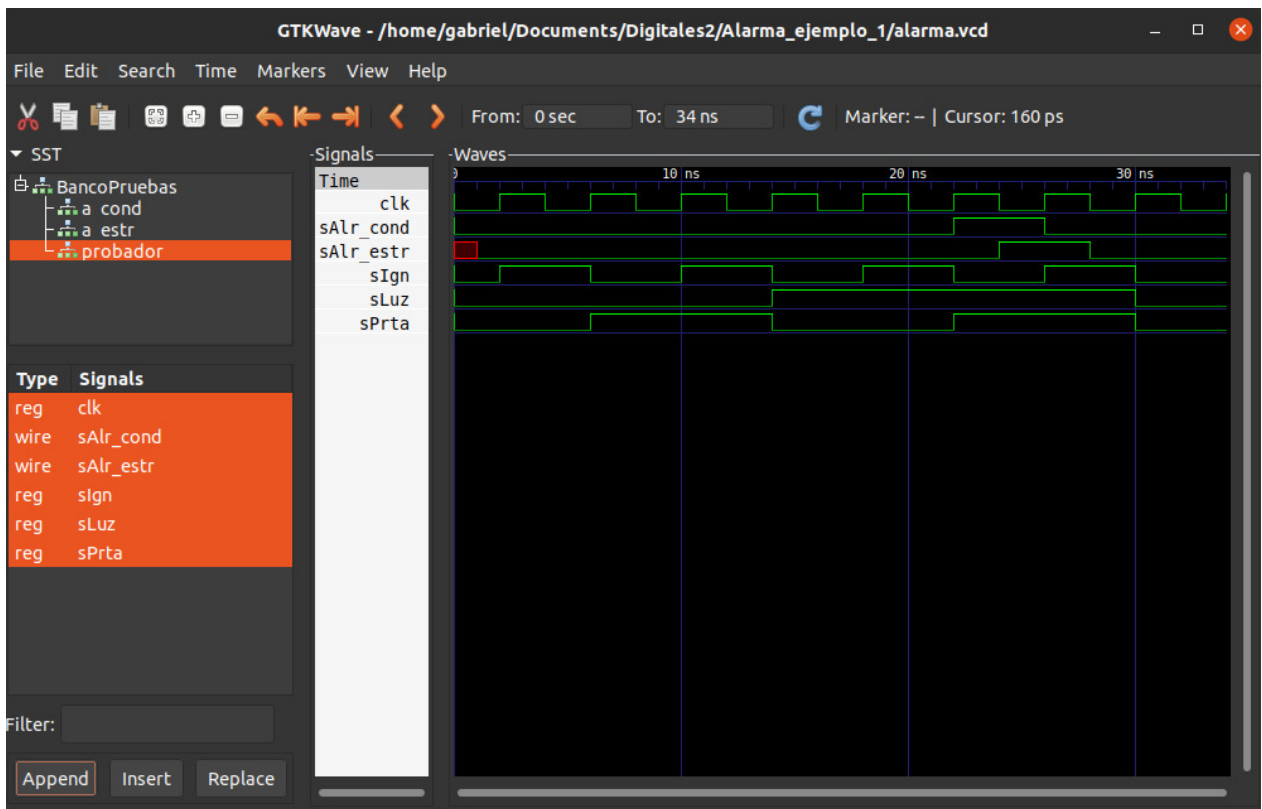


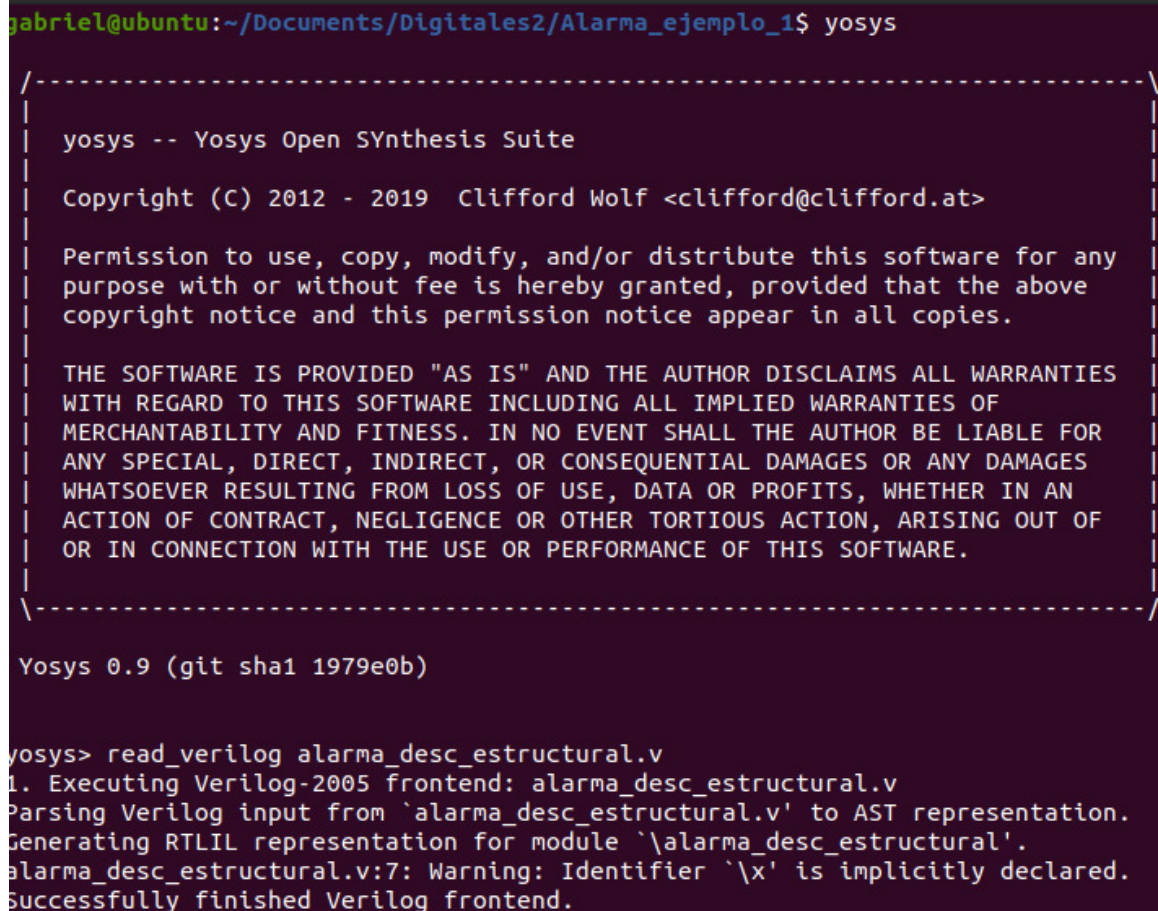
Figura 1: GtkWave

- **Yosys:** Yosys es un framework para la síntesis de módulos en código de verilog. Para usarlo se usan los siguientes comandos.

---

```
\$ yosys
\$ read_verilog <filename.v>
\$ opt # con el fin de optimizar procesos
\$ show
```

---



```
gabriel@ubuntu:~/Documents/Digitales2/Alarma_ejemplo_1$ yosys

/-----/
| 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> read_verilog alarma_desc_estructural.v
1. Executing Verilog-2005 frontend: alarma_desc_estructural.v
Parsing Verilog input from `alarma_desc_estructural.v' to AST representation.
Generating RTLIL representation for module `alarma_desc_estructural'.
alarma_desc_estructural.v:7: Warning: Identifier `x' is implicitly declared.
Successfully finished Verilog frontend.
```

Figura 2: Yosys

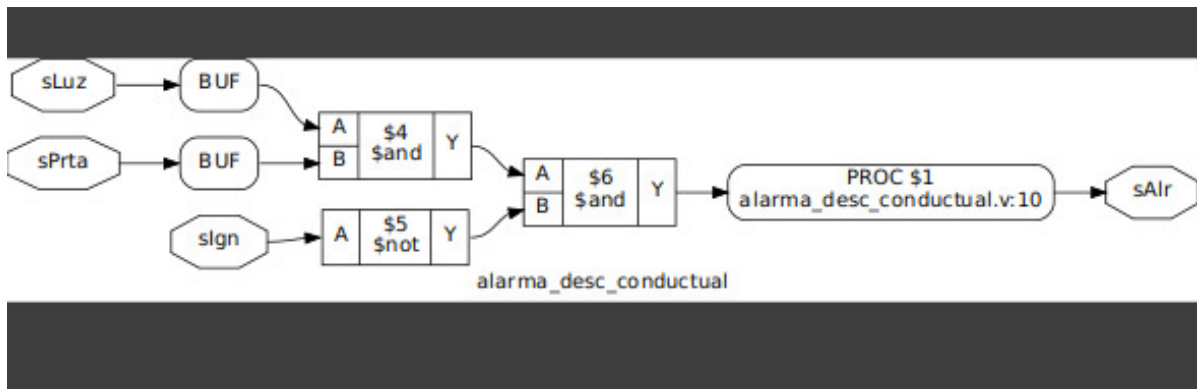


Figura 3: Yosys

#### 4. Makefile

El makefile es una herramienta con la cual es posible automatizar procesos como el compilado y la ejecución de código fuente o abrir o cerrar aplicaciones. Básicamente funciona como una lista de instrucciones bash que se ejecutan de manera secuencial. El código makefile requerido para esta tarea se muestra en seguida.

---

```

build: operador1 operador2 operador3 gtkwave1 yosys1 sed1 autoinst
    echo fin
operador1:
    iverilog BancoPrueba.v
operador2:
    ./a.out
operador3:
    rm a.out
gtkwave1:
    gtkwave --dump=alarma.vcd
yosys1:
    yosys
sed1:
    sed 's/alarma_desc_conductual/alarma_desc_conductual_synth/g' alarma_desc_conductual.v >
        alarma_desc_conductual_synth.v
autoinst:
    emacs --batch alarma_autoinst.v -f verilog-batch-auto

```

## 5. AUTOINST

El comando AUTOINST tiene como finalidad ahorrar tiempo al programador en escribir las entradas y salidas de los módulos que desea instanciar. Esto resulta especialmente útil cuando se quieren instanciar módulos con muchas entradas o muchas salidas o bien muchos módulos al mismo tiempo. Para hacer uso de la función AUTO se usa el siguiente comando bash

```
\$ emacs --batch <filename.v> -f verilog-batch-auto
```

---

```
`timescale          1ns          / 100ps
`include "alarma_desc_conductual.v"
`include "alarma_desc_estructural.v"
`include "probador.v"

module BancoPruebas; // Testbench
    wire sAlr_cond, sAlr_estr, sLuz, sPrta, sIgn;

    alarma_desc_conductual      a_cond(/*AUTOINST*/);
    alarma_desc_estructural    a_estr(/*AUTOINST*/);
    probador probador_(/*AUTOINST*/);
endmodule
```

---

Después de haber usado el comando bash anteriormente mencionado entonces se puede ver en el siguiente código como las instancias aparecen como por obra de magia.

---

```
`timescale          1ns          / 100ps
`include "alarma_desc_conductual.v"
`include "alarma_desc_estructural.v"
`include "probador.v"

module BancoPruebas; // Testbench
    wire sAlr_cond, sAlr_estr, sLuz, sPrta, sIgn;

    alarma_desc_conductual    a_cond(/*AUTOINST*/
                                    // Outputs
                                    .sAlr          (sAlr),
                                    // Inputs
                                    .sLuz          (sLuz),
                                    .sPrta          (sPrta),
                                    .sIgn          (sIgn));

    alarma_desc_estructural    a_estr(/*AUTOINST*/
                                    // Outputs
                                    .sAlr          (sAlr),
                                    // Inputs
                                    .sLuz          (sLuz),
                                    .sPrta          (sPrta),
                                    .sIgn          (sIgn));

    probador probador_(/*AUTOINST*/
                      // Outputs
                      .sLuz          (sLuz),
                      .sPrta          (sPrta),
                      .sIgn          (sIgn),
                      // Inputs
                      .sAlr_estr      (sAlr_estr),
                      .sAlr_cond      (sAlr_cond));

endmodule
```