Centro universitario: CUCEI

Materia: Seminario de solución de problemas de arquitectura de computadoras

Alumno: Ramos Preciado Alan Josafat

Código: 218130165

Carrera: Ingeniería en informática (INNI)

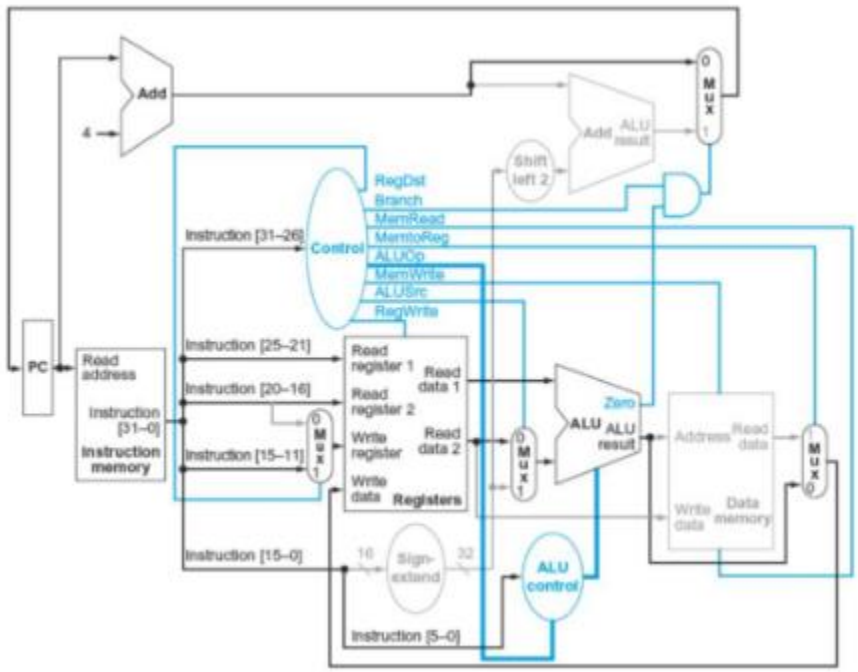
título de investigación: fase 1

Reporte de fase 1

Se tienen los siguientes modulos.

pc.v	Verilog	0	11/16/2024 11:38:40 ...
mux.v	Verilog	1	11/16/2024 03:02:03 ...
instructionM.v	Verilog	2	11/16/2024 11:19:35 ...
control.v	Verilog	3	11/16/2024 09:59:38 ...
registers.v	Verilog	4	11/16/2024 11:20:18 ...
alucontrol.v	Verilog	5	11/16/2024 03:43:24 ...
alu.v	Verilog	6	11/16/2024 09:57:11 ...
datam.v	Verilog	7	11/23/2024 09:11:21 ...
cand.v	Verilog	8	11/23/2024 08:59:07 ...
fase1.v	Verilog	9	11/16/2024 11:42:05 ...
mux5b.v	Verilog	10	11/16/2024 11:27:40 ...

Que son los que tenemos en nuestro diagrama.



Probaremos cada uno de ellos.

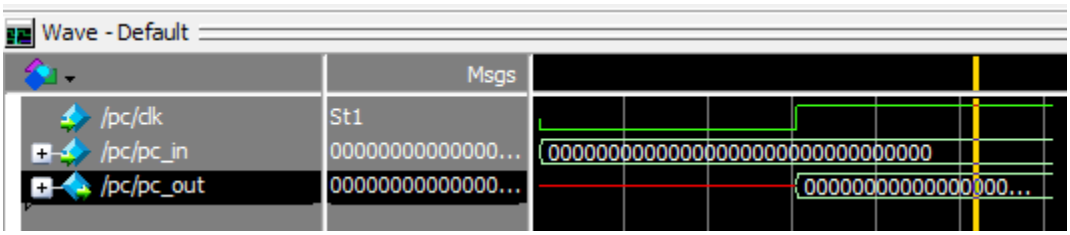
Empezando con pc

instructionM.v | pc.v | fase1.v

C: > Users > alan_ > Desktop > Fase1 > pc.v

```
1   `timescale 1ns/1ns
2
3   module pc(
4       input clk,
5       input [31:0] pc_in,
6       output reg [31:0] pc_out
7   );
8
9   always @(posedge clk)
10  begin
11      pc_out = pc_in;
12  end
13
14  endmodule
```

Donde se comprueba que funciona como se espera.



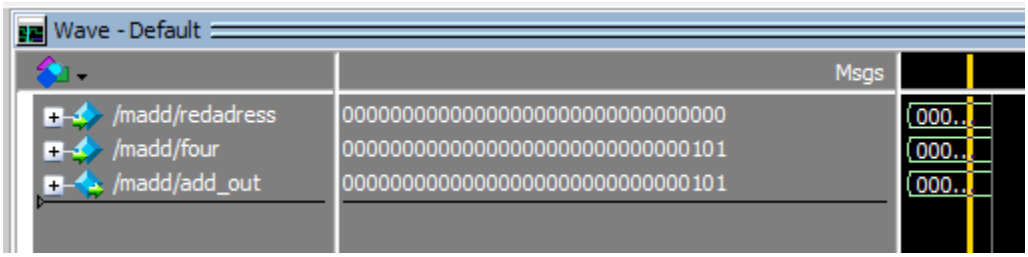
Al cambiar el valor de clk se da el valor de entrada al de salida.

El siguiente será el modulo madd

C: > Users > alan_ > Desktop > Fase1 > madd.v

```
1   `timescale 1ns/1ns
2
3   module cand(
4       input [31:0] madd_in,
5       input [31:0] madd_four,
6       output reg [31:0] out1  // 'out1' ahora es un registro
7   );
8
9       // Bloque always que se ejecuta cuando 'madd_in' o 'madd_four' cambian
10      always @ * begin
11          out1 = madd_in + madd_four;
12      end
13
14  endmodule
```

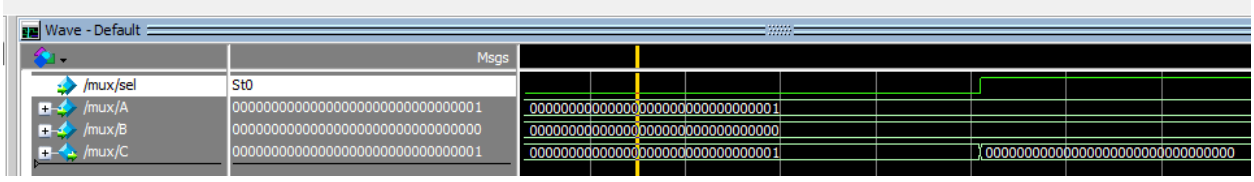
Comprobamos.



Ahora con el mux

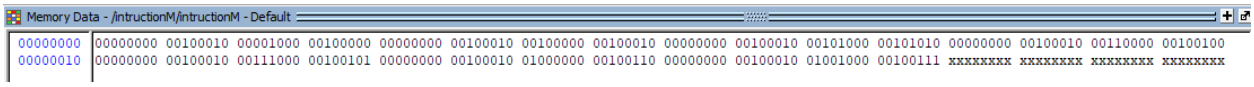
C: > Users > alan_ > Desktop > Fase1 > mux.v

```
1   timescale 1ns/1ns
2
3   module mux(
4
5       //inputs
6       input sel,
7       input [31:0] A,
8       input [31:0] B,
9
10      //outputs
11      output reg[31:0] C
12
13  );
14
15  always @*
16  begin
17      if(sel)
18      begin
19          C=B;
20      end
21      else
22      begin
23          C=A;
24      end
25  end
26
27  endmodule
```

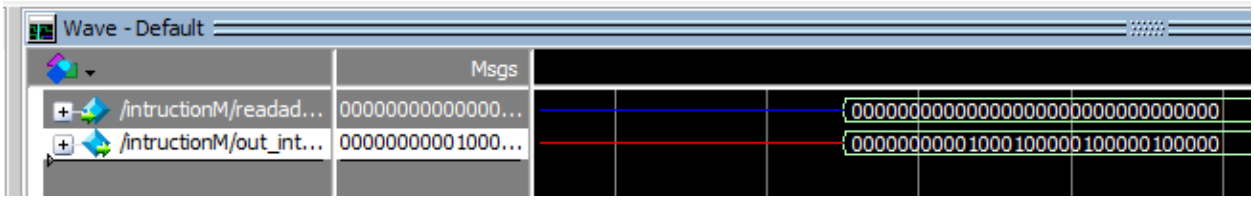


Ahora para que sea mas facil solo se mostrar las campturas de los waves de cada modulo haciendo el proceso anterior y comprabando su funcionamiento correcto.

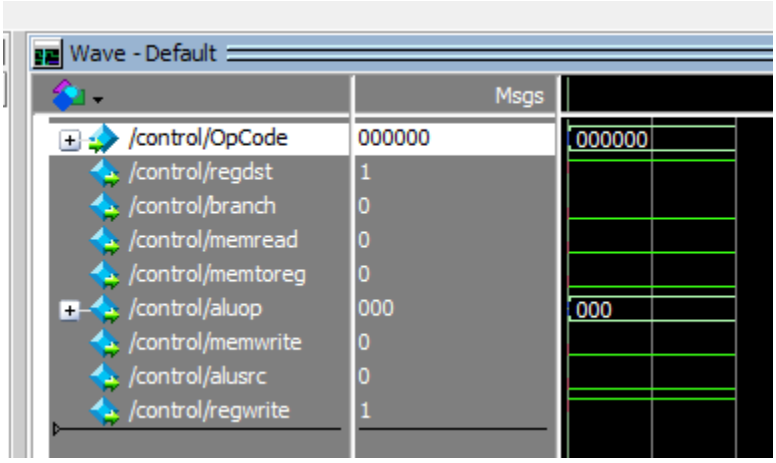
En lo siguiente comprabamos que nuestra memoria esta dividida en espacios de 8bit



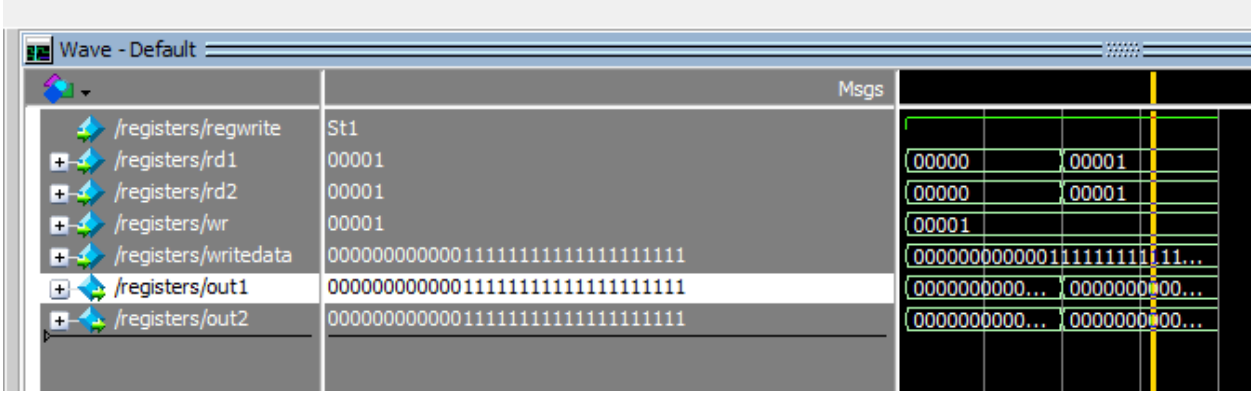
En los cuales nuestra salida es la concatenación.



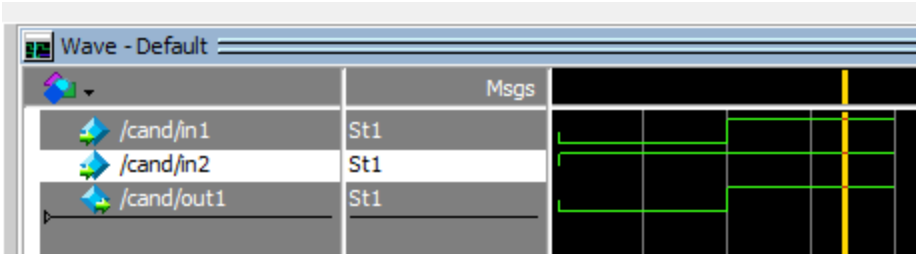
Nuestra posición 0 concatela los valores de las direcciones 0,1,2 y 3
modulo contron



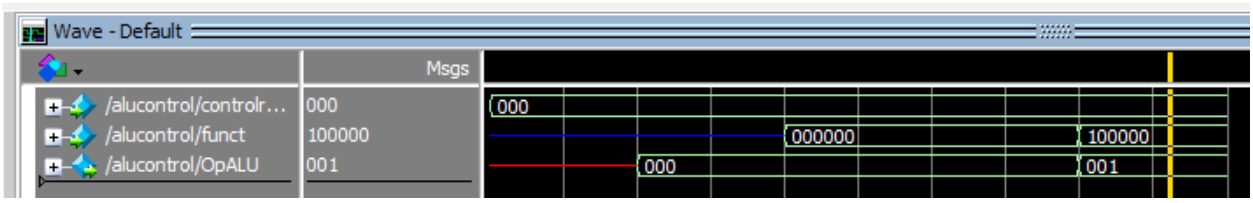
Registros



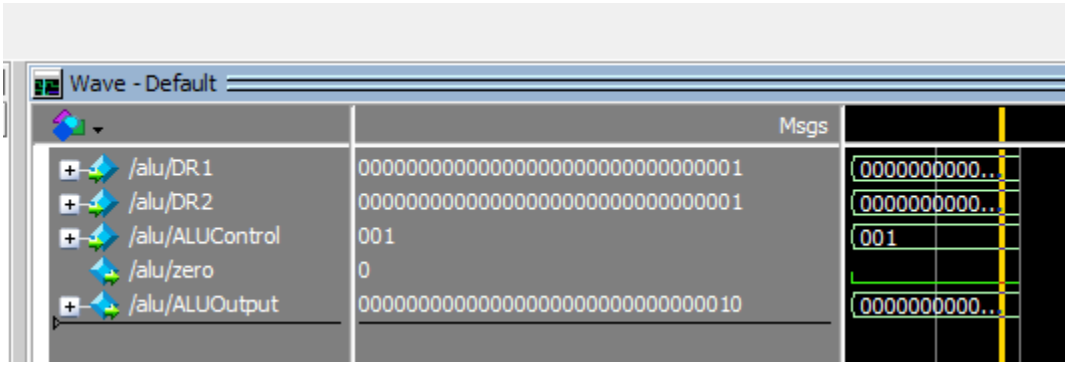
And



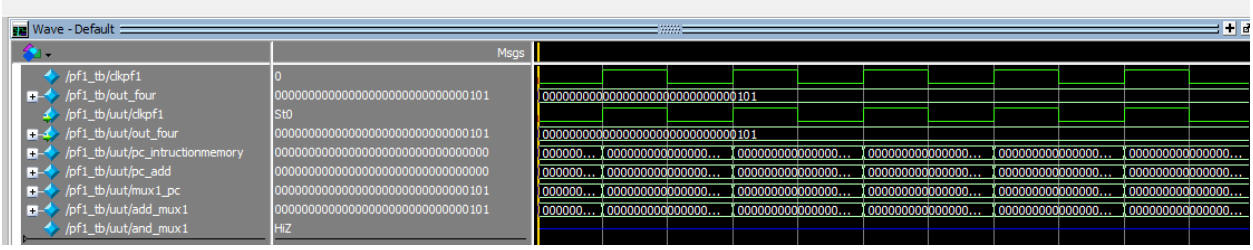
Alucontrol



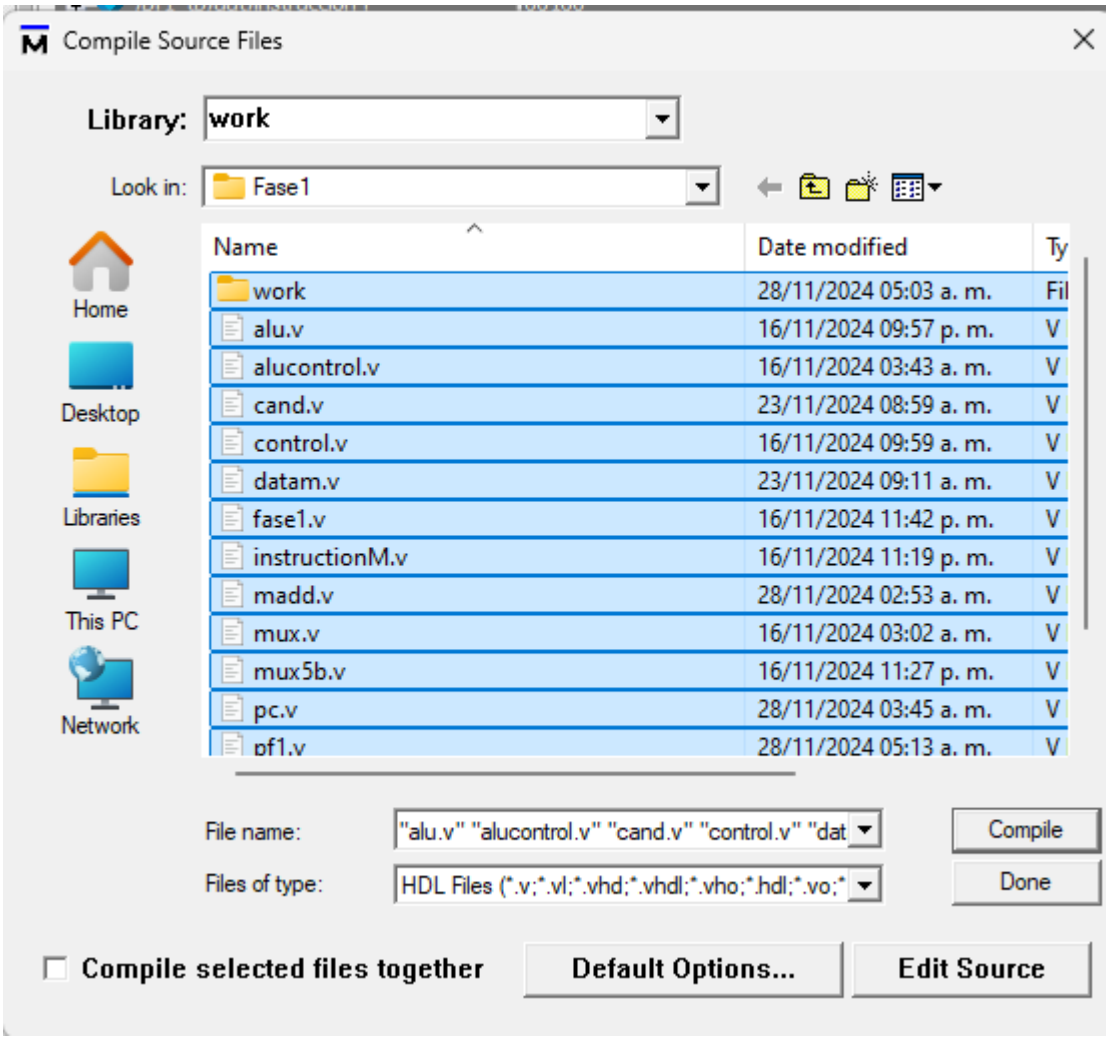
Alu



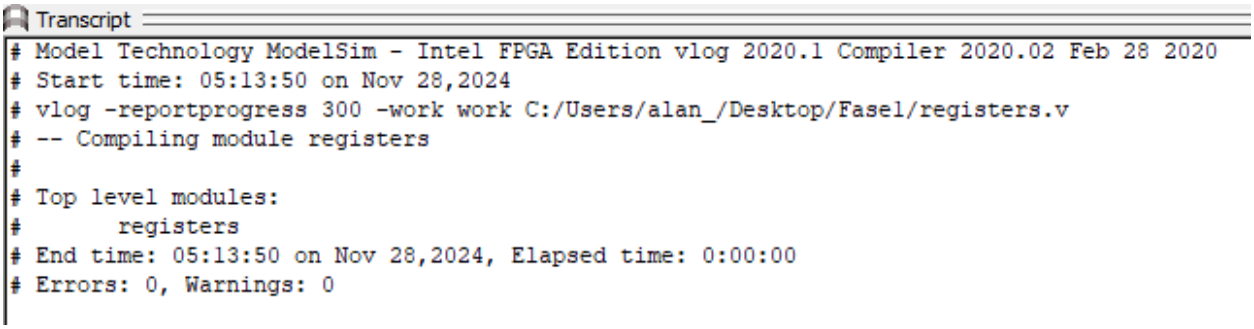
Aquí probaremos el pc, add, mux para comprobar que funcionen los 3 juntos



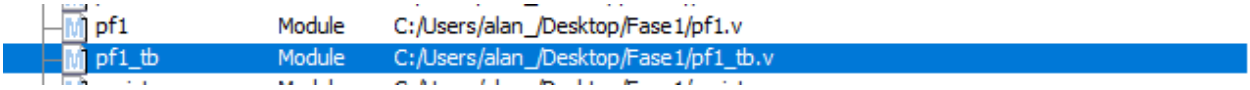
Después de compilar todo



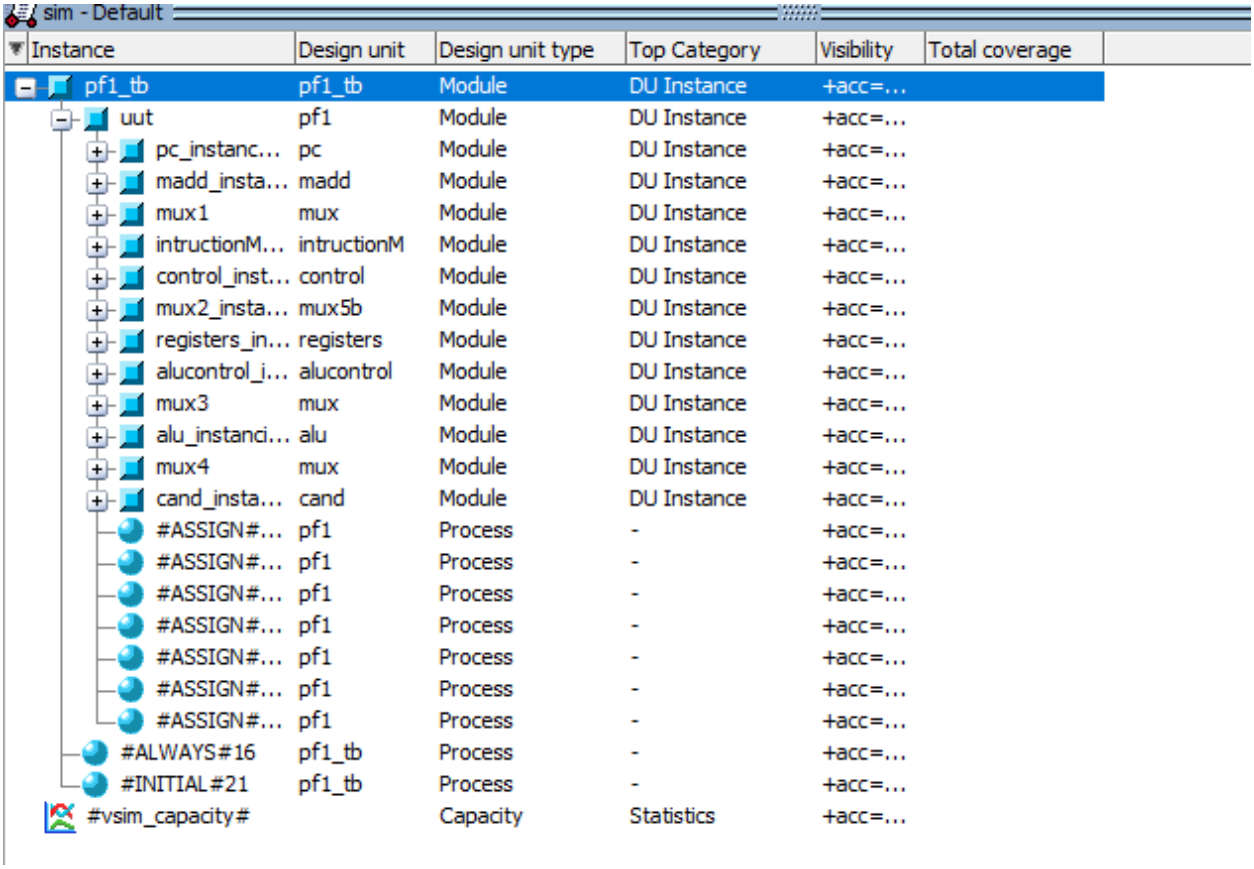
Revisamos si existe algún error



Simulamos el tb de la fase 1



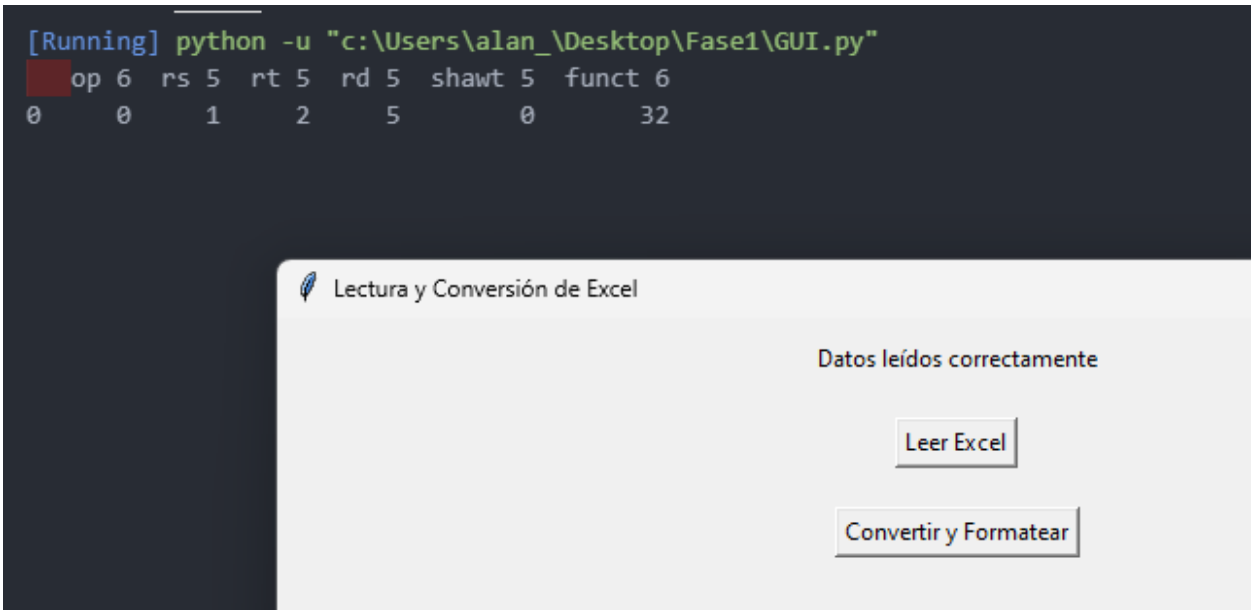
Agregamos al wave



Al parecer todo funciona pero tenemos que hacer una prueba

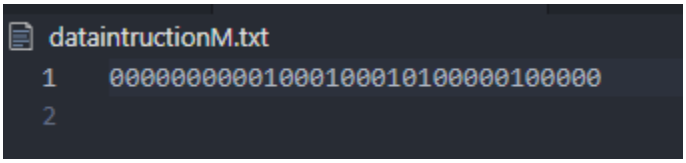
	A	B	C	D	E	F
1	op 6	rs 5	rt 5	rd 5	shawt 5	funct 6
2	0	1	2	5	0	32

Esa instrucción

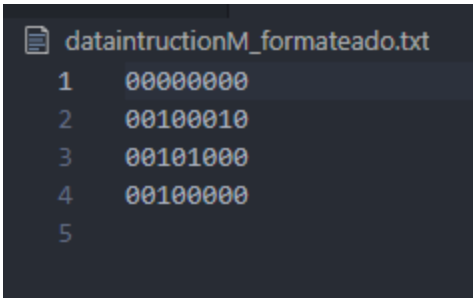


usamos nuestro codificador

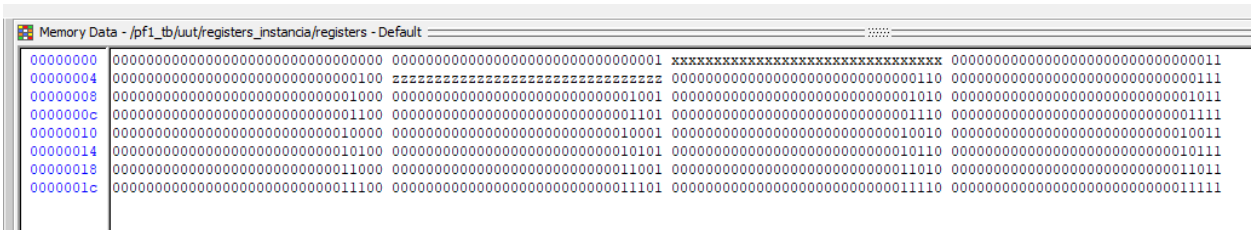
Nos genera nuestra instrucción



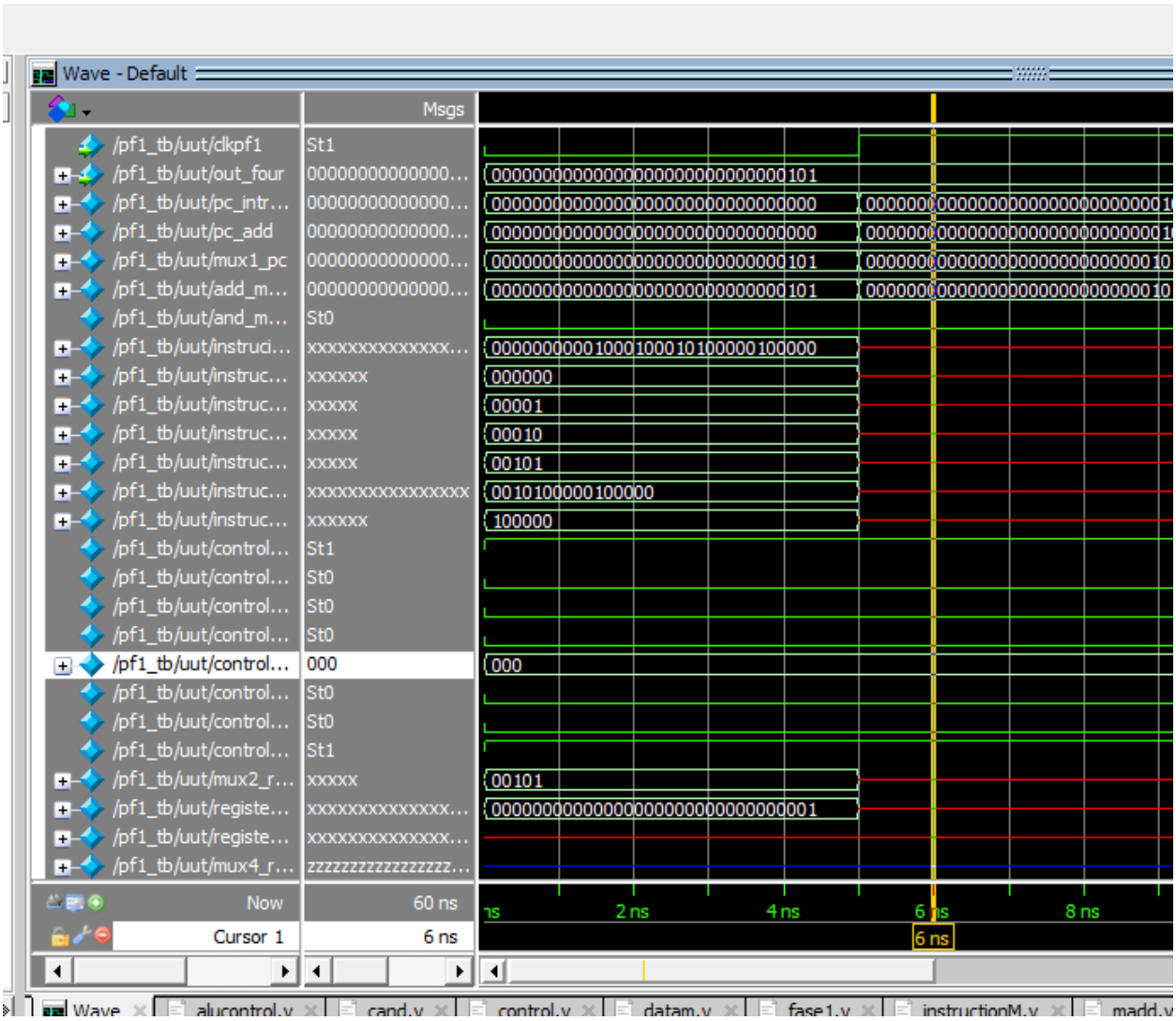
La hacemos de 8bits



Y al simular nos dio el siguiente error



Guardo valores inválidos



Conclusión

Se vuelve mas sencillo hacer las instrucciones con el codificador y termina siendo algo complicado hacer los módulos que reúnen casi toda los mini módulos.