Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0499 - Proyecto Eléctrico
II ciclo 2024

Bitácora

Verificación funcional y diseño de la implementación física de un procesador PicoSoC

José Mario Navarro Bejarano B75398

25 de septiembre, 2024.

Índice

1. Semana 2

1.1. Sesión: lunes 19 de agosto (1 hora)*:

Se mantuvo una reunión con el profesor Gerardo Castro, en la cual se discutieron los términos generales del proyecto.
Se discutió el cronograma con el que se va a trabajar, incluyendo el documento requerido para entregar la próxima semana (test plan).

1.2. Sesión: miércoles 21 de agosto (2 horas)

Se asistió a clase presencial en donde se discutieron algunos temas del curso como el formato para algunos entregables y demás.

1.3. Sesión: viernes 23 de agosto: (6 horas)

La mitad de las horas fueron dedicadas a estudiar el DUT, el cual es un núcleo RISCV, por lo que fue necesario estudiar la especificación y entender la mayor cantidad de detalles posibles. La otra mitad del tiempo fue dedicado a realizar el documento test plan, con base en lo solicitado por el profesor y la especificación del DUT.

*Nota: En las sesiones con el profesor guía, se anotan varias mini-tareas que deben cumplirse antes de la próxima sesión. Sirven de recordatorio para las cosas que se deben realizar. - Montar el testplan. - Se decidió el cronograma, es necesario confeccionarlo formalmente. - Se podría realizar un repositorio en Github, pero no es urgente.

2. Semana 2

2.1. Sesión: lunes 19 de agosto (1 hora)*:

Se mantuvo una reunión con el profesor Gerardo Castro, en la cual se discutieron los términos generales del proyecto.
Se discutió el cronograma con el que se va a trabajar, incluyendo el documento requerido para entregar la próxima semana (test plan).

2.2. Sesión: miércoles 21 de agosto (2 horas)

Se asistió a clase presencial en donde se discutieron algunos temas del curso como el formato para algunos entregables y demás.

2.3. Sesión: viernes 23 de agosto: (6 horas)

La mitad de las horas fueron dedicadas a estudiar el DUT, el cual es un núcleo RISCV, por lo que fue necesario estudiar la especificación y entender la mayor cantidad de detalles posibles. La otra mitad del tiempo fue dedicado a realizar el documento test plan, con base en lo solicitado por el profesor y la especificación del DUT.

*Nota: En las sesiones con el profesor guía, se anotan varias mini-tareas que deben cumplirse antes de la próxima sesión. Sirven de recordatorio para las cosas que se deben realizar. - Montar el testplan. - Se decidió el cronograma, es necesario confeccionarlo formalmente. - Se podría realizar un repositorio en Github, pero no es urgente.

3. Semana 3

3.1. Sesión: lunes 27 de agosto (2 horas)*

Se tuvo una reunión con el profesor en donde se realizó la entrega de la primera versión del test plan.
Se discutieron detalles menores sobre la ejecución del proyecto.
Se acordó los archivos en cuales se debe seguir avanzando para entregar la próxima semana.

3.2. Sesión: domingo 8 de agosto (7 horas)

Se dedicó 1 hora a completar los archivos necesarios de la semana, como entregar el anteproyecto y crear y actualizar la bitácora con la información de las semanas anteriores.

*Nota: En las sesiones con el profesor guía, se anotan varias mini-tareas que deben cumplirse antes de la próxima sesión. Sirven de recordatorio para las cosas que se deben realizar. - Reemplazar "co-pro" por BFM en entorno de verificación. - Remover pruebas para C, E, Fence y System. - Investigar funciones do_compare y do_copy (para posible uso en scoreboard) - Investigar más sobre IRQ. - Explicar más cómo obtener datos del DUT en el diagrama de bloques. - Investigar test para comprobar compliance con RISC-V. - Investigar sobre CSR. - Buscar y usar un coding style. - Recordar escribir el header en cada documento y en inglés. - Se puede usar código en C para comprobar saltos de Branch. - Aclarar test para PCPI, la interfaz en específico.

4. Semana 4

4.1. Sesión: lunes 2 de septiembre (1 hora)*

Se tuvo una reunión con el profesor en donde se habló sobre los avances del testplan y sobre los resultados de la información que había que investigar.

4.2. Sesión: domingo 8 de setiembre (8 horas)

Se estuvo trabajando principalmente en agregar el header con la licencia utilizada y en crear un primer test sencillo que funcionara, para comprobar la utilidad del uso de BFM en lugar de una memoria simulada.

*Nota: En las sesiones con el profesor guía, se anotan varias mini-tareas que deben cumplirse antes de la próxima sesión. Sirven de recordatorio para las cosas que se deben realizar. - Investigar sobre alguna licencia para agregar al proyecto. - Investigar cómo acceder a los valores de los registros sin tener que sacar las señales como output (tal vez mediante operador punto) - Recordar agregar "Fix Me" cuando se hace un cambio temporal en el código. - Investigar e implementar UVC para reset, memoria y PCPI. Es decir, tener 3 o 4 driver, scoreboard, monitor... - Estudiar virtual sequencer. - Investigar sobre "manejo de registro" , "mapa de registros", "acceso a registros". UVM_Reg. - Investigar las capas de UVM (como UVM_Reg) - Reemplazar BFM en lugar de sim_mem. - Ya que se va a usar BFM para la memoria, revisar en UVM_driver cuando el DUT responder (¿handshake?) - Cambiar el diagrama de capas para hacerlo mas especifico y menos genérico (por ejemplo, agregar los 3 agents). - Agregar la sección de compliance en el testplan. - Tener prueba de concepto del BFM para la próxima reunión. - Revisar cómo funciona y cómo se hace la verificación formal del DUT.

5. Semana 5

5.1. Sesión: lunes 9 de setiembre (1 hora)*

En esta sesión se tuvo una reunión virtual con el profesor Gerardo Casto. Se mostró el avance que se realizó. Como el avance que se tenía era funcional, se solicitó para la próxima reunión poder realizar correctamente el envío de todas las instrucciones que se iban a verificar.

5.2. Sesión: miércoles 11 de setiembre (2 horas)

Se asistió a clases presenciales. En esta clase se discutió sobre la primera entrega de las bitácoras, y se mencionaron puntos de mejora. También se habló del cronograma del curso para las próximas semanas, esto incluye presentar el avance al profesor (25 de setiembre), Realizar el elevator pitch (2 de octubre) y la siguiente semana a esta la presentación en formato Pecha Kucha.

5.3. Sesión: domingo 15 de setiembre (6 horas)

El domingo se estuvo trabajando en mejorar el entorno de verificación que se tiene. Parte de las mejoras incluyen el envío correcto de todos los tipos de instrucciones que son soportadas por el DUT. En la figura 2 se muestra un fragmento del código que se utiliza para generar de forma aleatoria mediante restricciones, todos los espacios que conformar las instrucciones de RISC-V (como se muestra en la figura 1). De esta forma es posible enviar de forma ilimitada instrucciones constantemente al DUT, para poder realizar una verificación funcional más completa.

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Figura 1: Formatos de instrucciones en RISCV

```
43      // After selecting the type of instruction, generate the specific fields
44      // for each instruction
45      constraint constr_opcode_c{
46          // Type R instr
47          if      (type_instr==0) opcode inside{7'b0110011            };
48          // Type I instr
49          else if (type_instr==1) opcode inside{7'b0010011, 7'b0000011};
50          // Type S instr
51          else if (type_instr==2) opcode inside{7'b0100011            };
52          // Type B instr
53          else if (type_instr==3) opcode inside{7'b1100011            };
54          // Type U instr
55          else if (type_instr==4) opcode inside{7'b0110111, 7'b0010111};
56          // Type J instr
57          else if (type_instr==5) opcode inside{7'b1101111, 7'b1100111};
58      }
59      constraint constr_funct3_c{
60          if      (opcode==7'b1100111) funct3 inside{3'b000};
61          else if (opcode==7'b1100011) funct3 inside{3'b000, 3'b001, 3'b100, 3'b101, 3'b110, 3'b111};
62          else if (opcode==7'b0000011) funct3 inside{3'b000, 3'b001, 3'b010, 3'b100, 3'b101};
63          else if (opcode==7'b0100011) funct3 inside{3'b000, 3'b001, 3'b010};
64      }
```

Figura 2: Fragmento del código que muestra la forma en la que se generan los valores para formar instrucciones

Otra tarea que se realizó fue agregar una pequeña sección del código para la implementación de los virtual sequencer, los monitores y el scoreboard. No se agregan muestras del trabajo realizado ya que fue poco lo que se avanzó, se espera que en la siguiente semana se pueda tener esta parte lista.

*Nota: Mini tareas de la reunión con el profesor Guía: - Realizar correctamente el envío de todas las instrucciones al DUT. -Tener el entorno de verificación completo (incluyendo los 3 Agent y el virtual Sequencer)

6. Semana 6

6.1. Sesión: lunes 16 de setiembre (1 hora)

Se realizó la reunión semanal con el profesor guía. Básicamente se mostró el avance, y se mencionó que se debía finalizar el entorno, además de corregir el envío de instrucciones, ya que esta parte se estaba realizando de forma incorrecta en el driver.

6.2. Sesión: domingo 22 de setiembre(8 horas)

En esta sesión de trabajo, los dos objetivos principales eran corregir la forma en la que se enviaban las instrucciones desde el driver, ya que la forma en la que se estaba realizando era como si el driver se comportara como un monitor, lo cual es incorrecto en el entorno de verificación. El segundo objetivo era terminar de agregar los componentes de verificación que estaban pendientes, como el scoreboard y los monitores.

El primer objetivo se resolvió integrando algo llamado "driver reactivo". Lo que significa que el driver va a reaccionar a las solicitudes realizadas por el DUT. Para implementar esto, se agregó un monitor, y cuando estos detecten que el DUT desea realizar un acceso de memoria, y además este acceso es para obtener instrucciones, se envía la solicitud al sequencer, para que este puede enviar la instrucción al driver, y del driver al DUT.

Para el segundo objetivo, simplemente se terminaron de agregar los componentes de verificación faltantes, a modo de esqueleto, pues elementos como el scoreboard aún no realiza ninguna función.

En la figura 3 se muestra el diagrama completo que se tiene hasta el momento del entorno de verificación, en este se puede observar que ya los componentes faltantes se encuentran implementados, también se puede observar que existe un canal de comunicación desde el virtual sequencer hasta los sequencer, y además existe un canal de comunicación desde los monitores hasta los drivers de los 3 UVCs (memoria, reset y pcpi). Aunque quedan pendientes las asersiones, estas no representan un problema en la ejecución de la verificación.

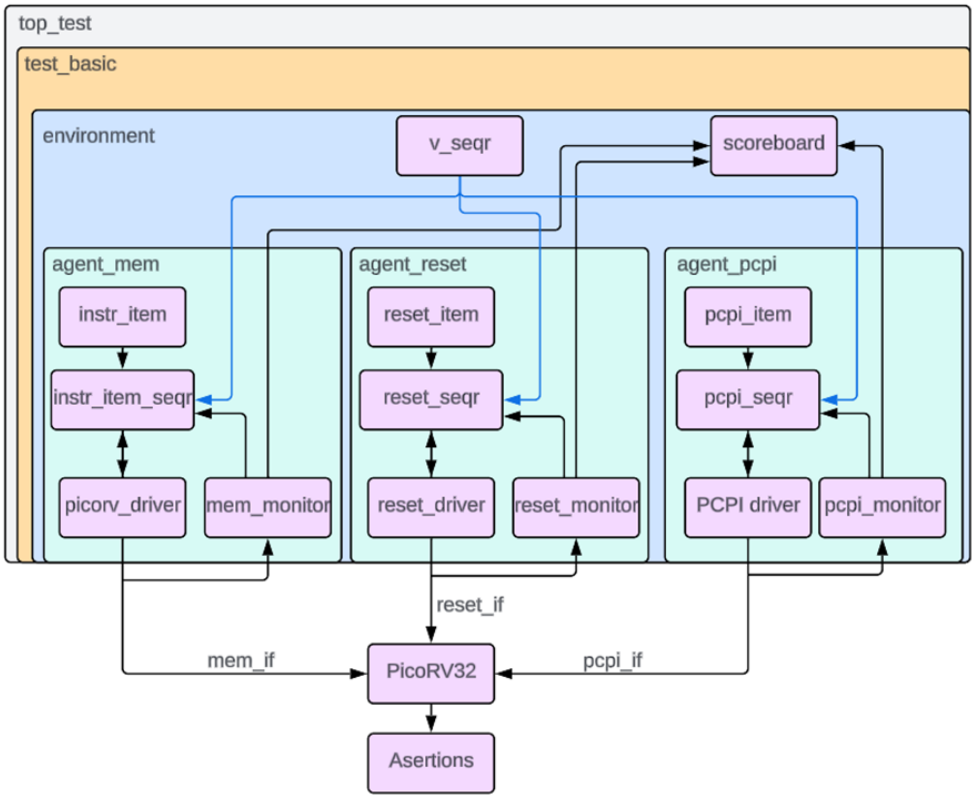


Figura 3: Diagrama de bloques completo del entorno de verificación que se está utilizando

7. Semana 7

7.1. Sesión: lunes 23 de setiembre (1 hora)*

En la reunión semanal con el profesor, se mostró el avance, y se decidió que para la próxima semana, se debe priorizar la implementación del modelo de referencia, ya que este puede presentar varias complicaciones.

7.2. Sesión: miércoles 25 de setiembre (2 horas)

Se asistió a clases presenciales, en donde se mostró el avances que se tiene con el proyecto hasta el momento al profesor.

7.3. Sesión: domingo 29 de setiembre (6 horas)

Ya que el objetivo principal es avanzar los más posible en el modelo de referencia, la mayoría del tiempo de la sesión de dedicó a esto. En la figura 4 se muestra parte del codigo que forma el item que contiene la información de los registros de propósito general que contiene el picorv. Este grupo de registros se envía al scoreboard cada vez que el monitor de memoria detecta que se produce algún cambio en el valor de algún registro. Por otro lado, el monitor también detecta cuando se envía alguna instrucción al DUT, y envía esta instrucción al scoreboard, de esta manera, con las instrucciones enviadas y el valor de los registros, el scoreboard puede realizar el proceso de comparación y comprobar si existen diferencias en los registros.

```
20  class reg_state extends uvm_sequence_item;
21      bit [31:0] reg_state_x1;
22      bit [31:0] reg_state_x2;
23      bit [31:0] reg_state_x3;
24      bit [31:0] reg_state_x4;
25      bit [31:0] reg_state_x5;
26      bit [31:0] reg_state_x6;
27      bit [31:0] reg_state_x7;
28      bit [31:0] reg_state_x8;
29      bit [31:0] reg_state_x9;
30      bit [31:0] reg_state_x10;
31      bit [31:0] reg_state_x11;
32      bit [31:0] reg_state_x12;
33      bit [31:0] reg_state_x13;
34      bit [31:0] reg_state_x14;
35      bit [31:0] reg_state_x15;
36      bit [31:0] reg_state_x16;
37      bit [31:0] reg_state_x17;
38      bit [31:0] reg_state_x18;
39      bit [31:0] reg_state_x19;
40      bit [31:0] reg_state_x20;
41      bit [31:0] reg_state_x21;
```

Figura 4: Parte del codigo del reg_state, que se envía al scoreboard cuando se detecta algún cambio.

En la imagen 5 se muestra una parte del código utilizado por el scoreboard para realizar la decodificación de las instrucciones y así poder manipular los registros de referencia, que se utilizan para comparar los valores del DUT.

```
60          case (instr.instr_dtc[6:0])
61              7'b0110111: begin
62              //    // LUI instr
63              //    ref_regs[instr.instr_dtc[11:7]] = {instr.instr_dtc[31:12], 12'b0};
64                  end
65              7'b0010011:begin
66                  case(instr.instr_dtc[14:12])
67                      //ADDI
68                      3'b000: begin
69                          temp_ref_regs.regs[instr.instr_dtc[11:7]] =
70                              {{20{instr.instr_dtc[31]}}, instr.instr_dtc[31:20]} +
71                               temp_ref_regs.regs[instr.instr_dtc[19:15]];
72                          regs_queue.push_back(temp_ref_regs);
73                          end
74                      default:;
75                  endcase
76              end
77              default:begin
78               `uvm_error("SCB", $sformatf("The regs are not the same"))
79              end
80          endcase
```

Figura 5: Sección del código que se encarga de realizar la decodificación de las instrucciones en el scoreboard.

También se muestra en la figura 6 una parte del código que utiliza el scoreboard cuando recibe el conjunto de valores de los registros reales del DUT, para poder compararlos con los de referencia.

```
91      virtual function void write_reg (reg_state reg_stte);
92          if (regs_queue.size()!=0)begin
93              ref_regs = regs_queue.pop_front();
94          end
95          reg_stte.print();
96          if (ref_regs.regs[1 ] != reg_stte.reg_state_x1  ||
97              ref_regs.regs[2 ] != reg_stte.reg_state_x2  ||
98              ref_regs.regs[3 ] != reg_stte.reg_state_x3  ||
99              ref_regs.regs[4 ] != reg_stte.reg_state_x4  ||
100             ref_regs.regs[5 ] != reg_stte.reg_state_x5  ||
101             ref_regs.regs[6 ] != reg_stte.reg_state_x6  ||
102             ref_regs.regs[7 ] != reg_stte.reg_state_x7  ||
103             ref_regs.regs[8 ] != reg_stte.reg_state_x8  ||
104             ref_regs.regs[9 ] != reg_stte.reg_state_x9  ||
105             ref_regs.regs[10] != reg_stte.reg_state_x10 ||
106             ref_regs.regs[11] != reg_stte.reg_state_x11 ||
107             ref_regs.regs[12] != reg_stte.reg_state_x12 ||
```

Figura 6: Sección del código que utiliza el scoreboard para comprobar si los registros reales son iguales a los registros de referencia.

*Nota: Mini tareas de la reunión con el profesor guía: -Agregar una memoria para almacenar la información que el DUT desea escribir. - Ver la cobertura funcional en Dsim Desktop. - Corregir el tamaño de los saltos en las instrucciones. - Corregir el código para que no haya que hacer reset cuando se levanta trap. - Modelar el acceso a memoria (tiempo de respuesta de 1 a 10 ciclos). - Crear un item que contenga los registros para enviarlos al scoreboard. - Priorizar el modelo de referencia a utilizar en el scoreboard. - Buscar más información sobre los CSR de RISCV. - Para la semana 9 tener el modelo de referencia listo. - Para la semana 10 tener el coverage listo. - Ir agregando algunas aserciones. - Revisar nuevamente lo de verificación formal.

8. Semana 8

8.1. Sesión: miércoles 4 de Octubre (2 horas)

En esta semana se asistió a clases presenciales del curso, en la cual se dio la presentación de los avances del proyecto frente a los compañeros del curso.

8.2. Sesión: domingo 06 de Octubre (7 horas)

En la reunión de la semana 7 con el profesor guía se indicó que se debía priorizar el avance en el modelo de referencia. También se indicó que se debía realizar una forma de simular la memoria para almacenar datos, esto con el objetivo de poder almacenar datos cuando se corra algún programa sobre el core. Con base en eso, en esta sesión se priorizó el diseño de una memoria o un lugar para almacenar los datos. En la figura 7 se muestra una parte del código que se implementó. En las lineas 21-22 se puede observar que se creó una estructura que contiene un array asociativo. Este array contiene numeros enteros como llaves y en cada uno almacena un byte de información, esto logra simular la forma en la que se almacenan datos en una memoria real.
Por otra parte en la linea 24 se escribe la clase que se instanciará luego en la memoria y que será en donde se escriban y lean los datos. Esta clase contiene algunas funciones, tal como se detalla a continuación:

- new: El constructor de la clase.
- write: Esta clase se utiliza para escribir los datos en la memoria. Esta función recibe una dirección de 32 bits y un byte de información. El monitor es el que se encarga de llamar a esta función con los parámetros correspondientes.
- read: (no se muestra). Esta función recibe una dirección de 32 bits y devuelve el correspondiente byte de información. Si la dirección no se ha escrito, se devuelve cero.
- read_all: (no se muestra). Esta función no recibe parámetros, Únicamente devuelve todo el array almacenado.

```
20  typedef struct{
21      bit [7:0] mem [int];
22  } storage_t;
23
24  class storage extends uvm_component;
25      `uvm_component_utils (storage)
26
27      //bit [7:0] mem [int];
28      storage_t stg_cl;
29      int addr_int;
30
31      function new (string name, uvm_component parent=null);
32          super.new (name, parent);
33    endfunction: new
34
35    function void write (int addr, bit [7:0] value);
36     stg_cl.mem[addr] = value;
37    endfunction: write
```

Figura 7: Sección del código que corresponde a la clase que contiene el almacenamiento.

El resto del tiempo se utilizó en agregar esta clase al monitor y realizar pruebas para comprobar el funcionamiento del código.

9. Semana 9

9.1. Sesión: domingo 13 de Octubre (9 horas)

En esta semana se trabajó exclusivamente añadiendo el resto de instrucciones al modelo de referencia. Esto es necesario porque a como está implementado este modelo, el scoreboard recibe la instrucción y la decodifica para tratar los datos con base en la instrucción. En la figura 8 se muestra otra parte del código del modelo, en este caso, se muestran las instrucciones de salto condicional.

```
493                 //BEQ
494                 3'b000: begin
495                     if (temp_ref_regs.regs[instr.rdata[24:20]]==
496                         temp_ref_regs.regs[instr.rdata[19:15]])begin
497                         exp_addr=
498                             {{20{instr.rdata[31]}},instr.rdata[7],
499                                 instr.rdata[30:25], instr.rdata[11:8], 1'b0}+
500                                 instr.mem_addr;
501                         exp_addr[1:0] = 2'b00;
502                         branch_pend=1;
503                     end else begin
504                         exp_addr+=4;
505                     end
506                     end
507                 //BNE
508                 3'b001: begin
509                     if (temp_ref_regs.regs[instr.rdata[24:20]]!=
510                         temp_ref_regs.regs[instr.rdata[19:15]])begin
511                         exp_addr=
512                             {{20{instr.rdata[31]}},instr.rdata[7],
513                                 instr.rdata[30:25], instr.rdata[11:8], 1'b0}+
514                                 instr.mem_addr;
515                         exp_addr[1:0] = 2'b00;
516                         branch_pend=1;
517                     end else begin
```

Figura 8: Algunas instrucciones de salto condicional agregadas al modelo de referencia.

El resto del tiempo se utilizó en agregar otras instrucciones y comprobar que la forma en la que se implementaron fuera el correcto.

10. Semana 10

10.1. Sesión: lunes 14 de Octubre (1 hora)*

Se mantuvo una reunión con el profesor guía en donde se mostró el avance con el que se cuenta.

10.2. Sesión: domingo 20 de Octubre (8 horas)

Parte del tiempo se invirtió en mejorar el coverage, el cual se puede utilizar para comprobar si ya se estimularon las funciones necesarias. En la figura 9 se muestra una parte del código que corresponde al coverage agregado. Se puede observar que lo que está comprobando es que se haya enviado cada instrucción. Queda pendiente agregar la parte para verificar que en cada instrucción se utilizan todos los registros como destinos y como fuente.

```
41      covergroup instr;
42          type_R: coverpoint vir_mem_if.mem_rdata {
43              //              {           imm          _ rd  _ opcode}
44              bins LUI    = {32'b????????????????????_?????_0110111};
45              bins AUIPC  = {32'b????????????????????_?????_0010111};
46              bins JAL    = {32'b????????????????????_?????_1101111};
47              //              {           imm  _rs1  _fu3_  rd _ opcode}
48              bins JALR   = {32'b???????????_?????_000_?????_1100111};
49              //              {      imm  _ rs2 _ rs1 _fu3_ imm _ opcode}
50              bins BEQ    = {32'b???????_?????_?????_000_?????_1100011};
51              bins BNE    = {32'b???????_?????_?????_001_?????_1100011};
52              bins BLT    = {32'b???????_?????_?????_100_?????_1100011};
53              bins BGE    = {32'b???????_?????_?????_101_?????_1100011};
54              bins BLTU   = {32'b???????_?????_?????_110_?????_1100011};
55              bins BGEU   = {32'b???????_?????_?????_111_?????_1100011};
56              //              {      imm       _ rs1 _fu3_ rd  _ opcode}
57              bins LB     = {32'b???????????_?????_000_?????_0000011};
58              bins LH     = {32'b???????????_?????_001_?????_0000011};
59              bins Lw     = {32'b???????????_?????_010_?????_0000011};
60              bins LBU    = {32'b???????????_?????_100_?????_0000011};
61              bins LHU    = {32'b???????????_?????_101_?????_0000011};
62              //              {      imm  _ rs2 _ rs1 _fu3_ imm _ opcode}
63              bins SB     = {32'b???????_?????_?????_000_?????_0100011};
```

Figura 9: Muestra del coverage implementado.

Al igual que en las semanas anteriores, parte del tiempo se utilizó en seguir agregando instrucciones al modelo de referencia. En esta ocasión se enfocó en agregar las instrucciones correspondientes al complemento M, es decir, las instrucciones de multiplicación y división.

*Nota: Mini tareas de la reunión con el profesor guía: - Terminar de chequear el correcto funcionamiento de las instrucciones I y M. -Completar el coverage. -Decidir si seguir implementado el driver para PCPI o en su lugar agregar las instrucciones C. -Investigar más lo de verificación formal.

11. Semana 11

11.1. Sesión: lunes 21 de Octubre (1 hora)

Se realizó una reunión con el profesor guía en donde se mostró el avance y se definieron las tareas a realizar para la próxima semana. Éstas tareas son:

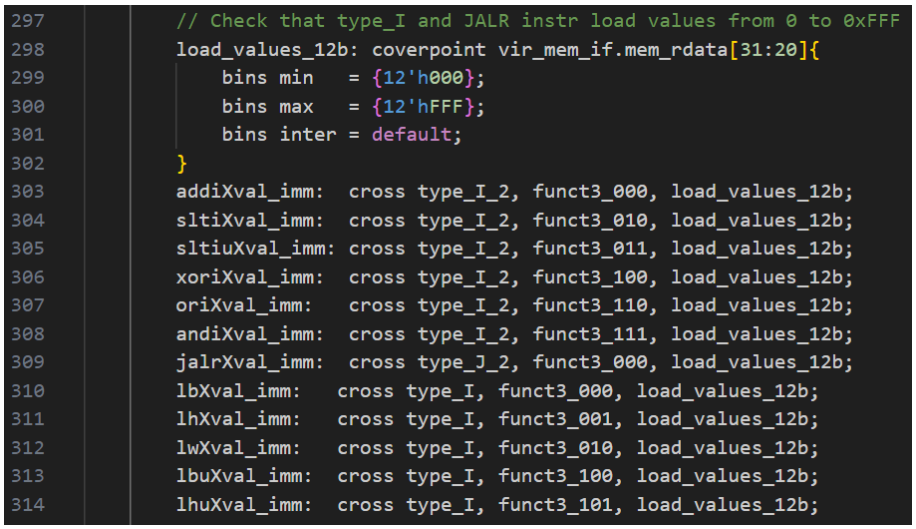
- Completar cobertura funcional. Incluyendo valores máximos y mínimos para las instrucciones que utilizan valores inmediatos.
- Finalizar el driver del PCPI.

11.2. Sesión: miércoles 23 de Octubre (2 horas)

Se realizó una presentación de los avances realizados hasta el momento del proyecto.

11.3. Sesión: domingo 27 de Octubre (6 horas)

Se realizaron los cambios solicitados por el profesor guía. En primer lugar, se finalizó de escribir la cobertura funcional, en la figura 10 se muestra una parte del código escrito para comprobar la cobertura funcional. En la linea 298-302 de esta figura se observa dónde se agrega la cobertura para comprobar que se cargaron valores inmediatos que son extremos, es decir, el valor máximo y el mínimo. Luego en las lineas 303-314 se crea la cobertura para comprobar que las instrucciones que utilizan esos valores inmediatos, se ejecutaron haciendo uso de esos valores extremos.



```
297         // Check that type_I and JALR instr load values from 0 to 0xFFF
298         load_values_12b: coverpoint vir_mem_if.mem_rdata[31:20]{
299             bins min   = {12'h000};
300             bins max   = {12'hFFF};
301             bins inter = default;
302         }
303         addiXval_imm:  cross type_I_2, funct3_000, load_values_12b;
304         sltiXval_imm:  cross type_I_2, funct3_010, load_values_12b;
305         sltiuXval_imm: cross type_I_2, funct3_011, load_values_12b;
306         xoriXval_imm:  cross type_I_2, funct3_100, load_values_12b;
307         oriXval_imm:   cross type_I_2, funct3_110, load_values_12b;
308         andiXval_imm:  cross type_I_2, funct3_111, load_values_12b;
309         jalrXval_imm:  cross type_J_2, funct3_000, load_values_12b;
310         lbXval_imm:   cross type_I, funct3_000, load_values_12b;
311         lhXval_imm:   cross type_I, funct3_001, load_values_12b;
312         lwXval_imm:   cross type_I, funct3_010, load_values_12b;
313         lbuXval_imm:  cross type_I, funct3_100, load_values_12b;
314         lhuXval_imm:  cross type_I, funct3_101, load_values_12b;
```

Figura 10: Muestra de la parte de la cobertura que se agregó ésta semana.

Otra cosa en la que se trabajó esta semana fue en la finalización de la parte del test que corresponde al PCPI. Esta interfaz PCPI es utilizada por el PicoRV para enviar instrucciones cuando detecta alguna que no puede resolver, como por ejemplo las instrucciones de multiplicación y división las cuales se realizan por unidades especiales separadas del core. Para comprobar el funcionamiento correcto de esta interfaz se agregó una instrucción falsa, la cual se envía al core, y luego este la coloca en la interfaz PCPI, en donde el driver PCPI la resuelve y devuelve los resultados al core. También se modificó el modelo de referencia para comprobar que la instrucción se ejecuta correctamente. En la figura 11 se muestra la sección del codigo que corresponde a la función send_response. Esta es la función del driver encargada de enviar la respuesta

al core cuando se detecta una instrucción falsa. También controla la señal wait, utilizada para indicarle al core que debe esperar en lo que se procesa la instrucción si se requiere más tiempo.

```
137     virtual task send_response(pcpi_item pcp_itm);
138         @(posedge intf_if.clk);
139         intf_if.pcpi_wait <=  (pcp_itm.wait_time>15) ? 1'b1 : 1'b0;
140         for(int i=0; i<=(pcp_itm.wait_time-5); i++) begin
141             @(posedge intf_if.clk);
142         end
143             intf_if.pcpi_wr <= pcp_itm.is_write;
144             intf_if.pcpi_rd <= pcp_itm.return_value;
145             intf_if.pcpi_ready <= 1;
146 
147         @ (negedge intf_if.clk);
148         @ (posedge intf_if.clk);
149         intf_if.pcpi_ready <= 0;
150     endtask: send_response
```

Figura 11: Función send_response del driver PCPI.

12. Semana 12

12.1. Sesión: lunes 28 de Octubre (1 hora)

Se mantuvo una reunión con el profesor guía en donde se mostraron los avances realizados en la semana anterior y se definieron nuevas tareas para realizar antes de la próxima reunión. Éstas tareas son:

- Agregar registro de las instrucciones que se enviaron al DUT.
- Agregar otro test que permita leer instrucciones desde un documento en lugar de generar éstas de forma aleatoria.
- Debido a que se complicó alcanzar el 100 % del coverage, investigar sobre cómo combinar test para lograr alcanzar esta cobertura total en varios test individuales.

12.2. Sesión: domingo 3 de Noviembre (8 horas)

En esta sesión lo primero que se realizó fue agregar el código necesario para poder guardar un registro de las instrucciones enviadas. En la figura 12 se muestra una captura de pantalla del documento que se genera luego de cada test, en donde se observa una lista que contiene, en hexadecimal, las instrucciones que se enviaron.

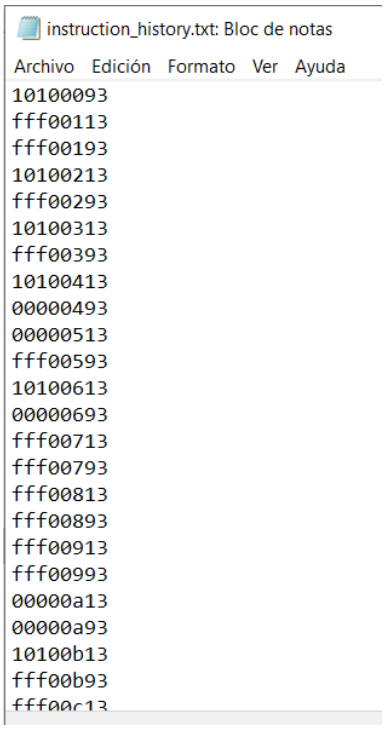


Figura 12: Muestra del documento que se guarda con el registro de instrucciones enviadas al DUT.

También se creó un test que en lugar de realizar una generación aleatoria de instrucciones, las lee de un documento previamente escrito. Para este caso, el documento a utilizar debe tener una estructura como el que se muestra en la figura 12. En la figura 13 se muestra parte del código que agregó en este test, en este caso lo que se observa es que el test lee las instrucciones de un documento y las guarda en la memoria simulada, para que luego cuando el DUT solicite ciertas instrucciones, estas se puedan acceder fácilmente y en el orden en que se escribieron.

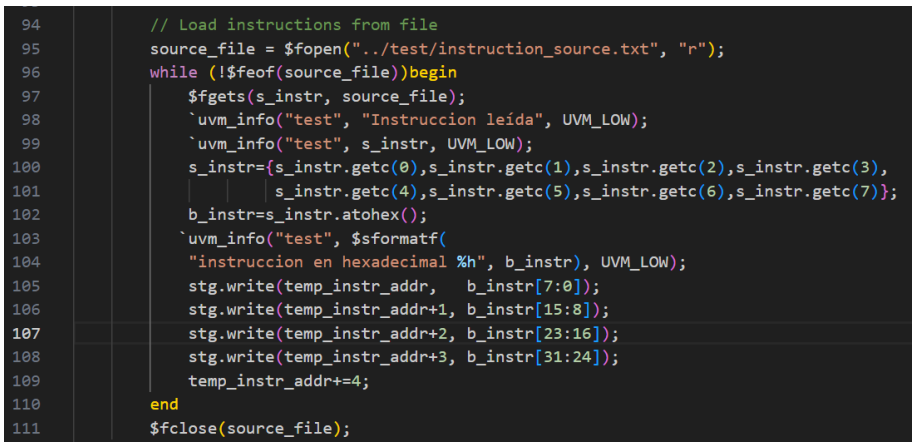


Figura 13: Muestra del código utilizado para leer instrucciones desde un documento .txt

13. Semana 13

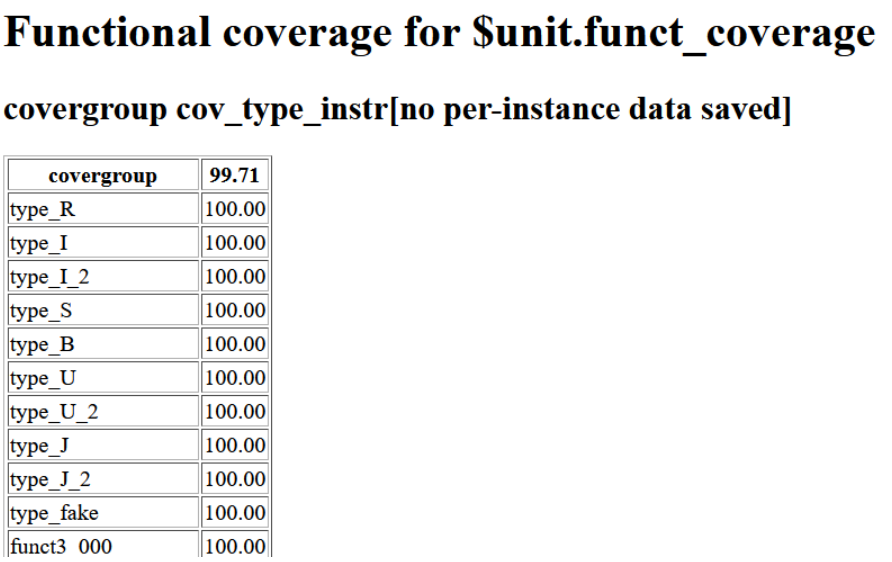
13.1. Sesión: lunes 4 de Noviembre (1 hora)

Se mantuvo una reunión con el profesor guía en donde se conversó sobre los avances que se tienen hasta el momento. También se definieron las próximas tareas que hay que realizar:

- Completar el documento de aserciones.
- Agregar los cambios que se han realizado al documento del testplan.

13.2. Sesión: domingo 10 de Noviembre (8 horas)

Parte del tiempo se dedicó a completar la documentación del testplan que se tiene. También se estuvieron realizado algunos cambios para poder alcanzar el 100 % de la cobertura funcional. En la figura 14 se muestra el documento que se genera con los resultados de la cobertura funcional obtenidos. Se puede observar que aun no se alcanza el valor deseado, sin embargo se continuarán realizado modificaciones para poder alcanzar la meta establecida.



Functional coverage for $unit.funct_coverage

covergroup cov_type_instr[no per-instance data saved]

covergroup	99.71
type_R	100.00
type_I	100.00
type_I_2	100.00
type_S	100.00
type_B	100.00
type_U	100.00
type_U_2	100.00
type_J	100.00
type_J_2	100.00
type_fake	100.00
funct3_000	100.00

Figura 14: Resultados de la cobertura funcional alcanzada.