Appendix A: Main Code and Modules

DNSLookup.sv

```
/* Name:
* Student ID: 12521589
* Purpose: A State Machine Depicting a DNS server webpage rendering
*/

module DNSLookup (
	input logic clk,
	input logic rst,
	input logic client_req,
	input logic [7:0] web_addr,
	output logic [15:0] webpage_idx_out,
	output logic [7:0] tld_addr_out,
	output logic [7:0] domain_ip_out,
	output logic [7:0] web_ip_out,
	output logic [7:0] exec_time,
	output logic ip_resolved,
	output logic client_res
);

	typedef enum logic [4:0] {IDLE, CLIENT_START, CLIENT_RESOLVER_REQ,
RESOLVER_ROOT_REQ, ROOT_RES,
		RESOLVER_TLD_REQ, TLD_RES, RESOLVER_DOMAIN_REQ, DOMAIN_RES, RESOLVER_RES,
CLIENT_SERVER_REQ, SERVER_RES, CACHING} enumstate;
	enumstate state, nextstate;

	logic count_en, count_rst;
	logic [3:0] count, web_ip_in;
	logic [7:0] tld_addr, domain_ip, web_ip;
	logic [15:0] cached_ip_map, webpage_idx;
	logic query_tld, query_domain, query_ip, query_data;

	counter ExecCounter(
		.clk(clk),
		.rst(count_rst),
		.en(count_en),
		.count(count)
	);

	TLDAddr WebAddrToTLDAddr (
		.in(web_addr),
		.out(tld_addr),
		.en(query_tld)
	);

	DomainIP TLDAddrToDomainIP (
		.in(tld_addr_out),
		.out(domain_ip),
		.en(query_domain)
	);

	WebIP DomainIPToWebIP (
		.in(domain_ip_out),
		.out(web_ip),
		.en(query_ip)
	);

	decoder416 WebIPToWebdata (
		.in(web_ip_in),
		.out(webpage_idx),
		.en(query_data)
	);

	always_ff @(posedge clk) begin
		if (rst) begin
			state <= IDLE;
		end
		else
			state <= nextstate;
	end

	always_comb begin
```

```
		case (state)
			IDLE: begin
				{count_rst, count_en} <= 2'b10;

				if (client_req) begin
					{client_res, ip_resolved} <= 2'b00;
					nextstate <= CLIENT_START;
				end
			end
			CLIENT_START: begin
				{count_rst, count_en} <= 2'b01;
				nextstate <= CLIENT_RESOLVER_REQ;
			end
			CLIENT_RESOLVER_REQ: begin

				if (cached_ip_map[7:0] == web_addr) begin
					nextstate <= CLIENT_SERVER_REQ;
				end
				else
					nextstate <= RESOLVER_ROOT_REQ;
			end
			RESOLVER_ROOT_REQ: begin
				query_tld <= 1'b1;
				tld_addr_out <= tld_addr;
				nextstate <= ROOT_RES;
			end
			ROOT_RES: begin
				query_tld <= 1'b0;
				nextstate <= RESOLVER_TLD_REQ;
			end
			RESOLVER_TLD_REQ: begin
				query_domain <= 1'b1;
				domain_ip_out <= domain_ip;
				nextstate <= TLD_RES;
			end
			TLD_RES: begin
				query_domain <= 1'b0;
				nextstate <= RESOLVER_DOMAIN_REQ;
			end
			RESOLVER_DOMAIN_REQ: begin
				query_ip <= 1'b1;
				web_ip_out <= web_ip;
				nextstate <= DOMAIN_RES;
			end
			DOMAIN_RES: begin
				query_ip <= 1'b0;
				nextstate <= RESOLVER_RES;
			end
			RESOLVER_RES: begin
				ip_resolved <= 1'b1;
				nextstate <= CLIENT_SERVER_REQ;
			end
			CLIENT_SERVER_REQ: begin
				query_data <= 1'b1;
				web_ip_in <= ip_resolved ? web_ip_out[3:0] :
cached_ip_map[11:8];
				webpage_idx_out <= webpage_idx;
				nextstate <= SERVER_RES;
			end
			SERVER_RES: begin
				query_data <= 1'b0;
				nextstate <= CACHING;
			end
			CACHING: begin
				cached_ip_map <= ip_resolved ? {web_ip_out, web_addr} :
cached_ip_map;

				exec_time <= count;
				client_res <= 1'b1;
				nextstate <= IDLE;
			end
			default: begin
				nextstate <= IDLE;
			end
		endcase
	end
endmodule
```

TLDAddr.sv

```
/* Name:
* Student ID: 12521589
* Purpose: A module simulating the process of querying for a TLD address
*/
module TLDAddr (
        input logic [7:0] in,
    output logic [7:0] out,
    input logic en
);

        assign out = en ? in>>2 : 8'bx;
endmodule
```

DomainIP.sv

```
/* Name:
* Student ID: 12521589
* Purpose: A module simulating the process of querying for a DomainIP
*/
module DomainIP (
        input logic [7:0] in,
    output logic [7:0] out,
    input logic en
);

        assign out = en ? in^8'b1111_1111 : 8'bx;
endmodule
```

WebIP.sv

```
/* Name:
* Student ID: 12521589
* Purpose: A module simulating the process of querying for a TLD address
*/
module WebIP (
        input logic [7:0] in,
    output logic [7:0] out,
    input logic en
);
        assign out = en ? {in[7:4]<<2, (in[3:0]>>2)^4'b1111} : 8'bx;
endmodule
```

counter.sv

```
/* Name:
* Student ID: 12521589
* Purpose: A simple up-counter
*/
module counter (
        input logic clk,
        input logic rst,
        input logic en,
        output logic[3:0] count
);

always_ff @(posedge clk) begin
        if (rst) begin
                count <= 0;
        end
```

```
        else if (en) begin
                count <= count + 1;
        end
end
endmodule
```

decoder416.sv

```
/* Name:
* Student ID: 12521589
* Purpose: A 4 to 16 decoder module to simulate web data to ip address mapping
*/

module decoder416 (
   input logic [3:0] in,
   output logic [15:0] out,
   input logic en
);
   parameter tmp = 16'b0000_0000_0000_0001;

   always_comb begin
      if (en) begin
         out = (in == 4'b0000) ? tmp   :
                     (in == 4'b0001) ? tmp<<1:
                     (in == 4'b0010) ? tmp<<2:
                     (in == 4'b0011) ? tmp<<3:
                     (in == 4'b0100) ? tmp<<4:
                     (in == 4'b0101) ? tmp<<5:
                     (in == 4'b0110) ? tmp<<6:
                     (in == 4'b0111) ? tmp<<7:
                     (in == 4'b1000) ? tmp<<8:
                     (in == 4'b1001) ? tmp<<9:
                     (in == 4'b1010) ? tmp<<10:
                     (in == 4'b1011) ? tmp<<11:
                     (in == 4'b1100) ? tmp<<12:
                     (in == 4'b1101) ? tmp<<13:
                     (in == 4'b1110) ? tmp<<14:
                     (in == 4'b1111) ? tmp<<15: 16'bx;
      end
      else
         out = 16'bx;
   end

endmodule
```

Appendix B: Testbench

```
// Specify simulation timescape
// Format: unit step / resolution
`timescale 1ns/1ps

module DNSLookup_tb();
        logic clk, rst, client_req;
        logic [7:0] web_addr, tld_addr, domain_ip, web_ip, exec_time;
        logic [15:0] webpage_idx;
        logic ip_resolved, client_res;

        // Instantiate the module to test
        DNSLookup DUT (
                .clk(clk),
                .rst(rst),
                .client_req(client_req),
```

```
                .web_addr(web_addr),
                .webpage_idx_out(webpage_idx),
                .tld_addr_out(tld_addr),
                .domain_ip_out(domain_ip),
                .web_ip_out(web_ip),
                .exec_time(exec_time),
                .ip_resolved(ip_resolved),
                .client_res(client_res)
        );

        // Initialize signals
        initial begin
                {clk, rst} = 2'b01;
                client_req = 1'b0;

        forever
        begin
                clk = ~clk;
                #5;
        end
        end


        // inputs
        initial begin
                #15;
                rst = 1'b0;
                #15;

                // query first web addr
                client_req = 1'b1;
                web_addr = 8'b1001_1010;
                #5;
                client_req = 1'b0;
                #140;

                // test that first web addr is cached by querying again
                client_req = 1'b1;
                web_addr = 8'b1001_1010;
                #5;
                client_req = 1'b0;
                #80

                // query second web addr, check that new address is cached
                client_req = 1'b1;
                web_addr = 8'b0011_1110;
                #5;
                client_req = 1'b0;
                #140;
        end
endmodule
```