

Lab 6 Report

BTU

CENG 342 Digital Systems

Cond	Encoding	Math	Description	Boolean Function
al	0000		Always (default)	1
mi	0001	< 0	Result Less than Zero	N
pl	0010	≥ 0	Result Less than or Equal to Zero	\overline{N}
eq	0011	$=$	Equal To	Z
ne	0100	\neq	Not Equal To	\overline{Z}
cs	0101		Carry Set	C
cc	0110		Carry Clear	\overline{C}
vs	0111		Overflow	V
vc	1000		No Overflow	\overline{V}
lt	1001	$<$	Less Than	$N \oplus V$
gt	1010	$>$	Greater Than	$\overline{Z}(N + V) + \overline{N}V$
le	1011	\leq	Less Than or Equal To	$Z + (N \oplus V)$
ge	1100	\geq	Greater than or Equal To	$NV + \overline{N}\overline{V}$
lo	0110	unsigned $<$	Lower Than	\overline{C}
hi	1101	unsigned $>$	Higher Than	$C\overline{Z}$
ls	1110	unsigned \leq	Lower Than or Same As	$\overline{C} + Z$
hs	0101	unsigned \geq	Higher Than or Same As	C

Samuel Donovan

Samantha Pfeiffer

2021-03-01

OVERVIEW

The purpose of this lab was to build a BTU (Branch Test Unit) that will be used by the CPU to conditionally execute instructions.

--- BTU ---

A BTU (Branch Test Unit) is used by the CPU to determine if an instruction should be executed based on its associated condition flags and the state of the CCR (Condition Code Register).

The BTU has 2 input buses, *Cond* and the CCR, and a single output bit, *R*.

Cond is a 4-bit bus whose value determines the boolean expression evaluated.

Our CCR currently has four bits (flags), *N*, *Z*, *C*, *V*. These flags are set by other subunits of the CPU; the BTU only reads them.

R is set based on the boolean expression evaluated, which is affected by both *Cond* and the CCR. When passed any unknown condition code, *R* is always set to 0.

The different condition flags available for the BTU are shown in **Table 1**. Note that some simplifications to the boolean functions have been made compared to the table given to us by our professor.

Table 1 BTU Condition Codes

Cond	Abbv.	Math	Description	Boolean Function
0000	al		Always	1
0001	mi	< 0	Result Negative (Minus)	N
0010	pl	≥ 0	Result Positive (Plus)	\overline{N}
0011	eq	=	Equal to	Z
0100	ne	\neq	Not Equal to	\overline{Z}
0101	cs		Carry Set	C
0110	cc		Carry Clear	\overline{C}
0111	vs		Overflow Set	V
1000	vc		No Overflow	\overline{V}
1001	lt	<	Less Than	$N \oplus V$
1010	gt	>	Greater Than	$N\overline{Z} + \overline{N}V$
1011	le	\leq	Less than or Equal to	$Z + (N \oplus V)$
1100	ge	\geq	Greater than or Equal to	$\overline{(N \oplus V)}$
0110	lo	unsigned <	Lower than	\overline{C}
1101	hi	unsigned >	Higher than	$C\overline{Z}$
1110	ls	unsigned \leq	Lower than or Same as	$\overline{C} + Z$
0101	hs	unsigned \geq	Higher than or Same as	C
XXXX	--		Others	0

DESIGN

All designs were made for a Nexys A7-100T (xc7a100tcsg324-1) FPGA board in VHDL using Vivado 2020.2.

--- PACKAGE ---

As was required by our instructor, a package file (**Appendix A**) was created to house a type definition for the CCR.

`flag_array` was defined to have four `STD_LOGIC` elements, `N`, `C`, `Z`, `V`, as is the case with our CCR. Two functions, `to_std_logic_vector` and `to_flag_array`, were also designed to ease the translation of our custom data type to and from an `STD_LOGIC_VECTOR`.

--- BTU ---

After defining our package, and having access to the `flag_array` data type, implementing the BTU was as simple as chaining a `with cond select` to map a different logical expression to `R` depending on `cond`.

To make the code more legible and easier to maintain, constants were defined for each of conditions.

The code for the BTU can be found in **Appendix B**.

--- TESTBENCH ---

As there were only 255 possible different combinations of the CCR and `cond`, two nested for loops were used to iterate through every possible value, and the output was manually verified to be correct.

The testbench can be found in **Appendix C**.

--- CONSTRAINTS ---

To synthesize this program, 8 switches were used as inputs, four switches for COND and four for the CCR. An LED was used to display the output. The constraints file can be found in **Appendix D**.

SIMULATION RESULTS

Figure 1 Simulation Results for cond= “0000” (al | Always)



Figure 2 Simulation Results for cond = “0001” (mi | Less than 0)

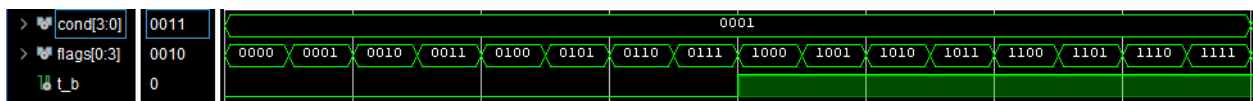


Figure 3 Simulation Results for cond = “0010” (pl | Greater or Equal to 0)

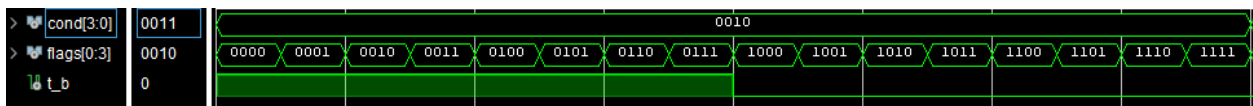


Figure 4 Simulation Results for cond = “0011” (eq | Equal to)

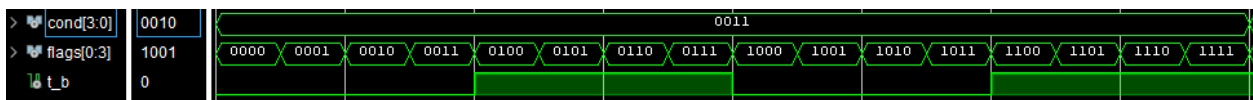


Figure 5 Simulation Results for cond = “0100” (ne | Not Equal to)

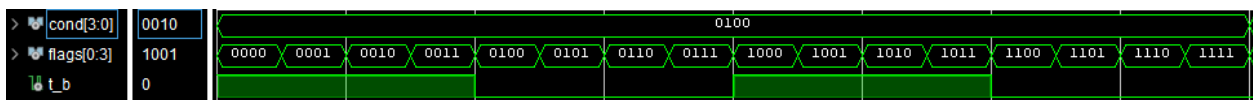


Figure 6 Simulation Results for cond = “0101” (cs | Carry Set)

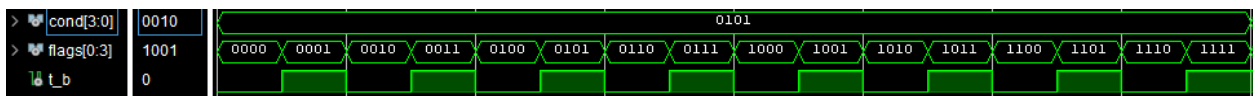


Figure 7 Simulation Results for cond = “0110” (cc | Carry Clear)



Figure 8 Simulation Results for cond = “0111” (vs | Overflow Set)

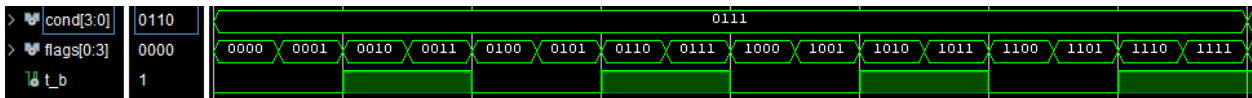


Figure 9 Simulation Results for cond = “1000” (vc | Overflow Clear)

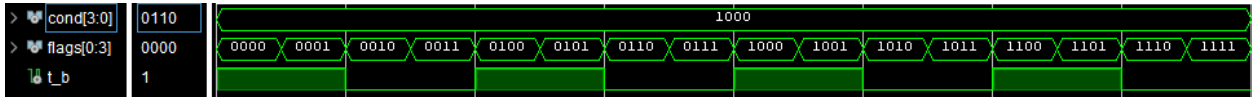


Figure 10 Simulation Results for cond = “1001” (lt | Less Than)

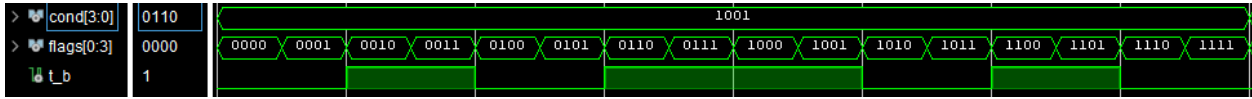


Figure 11 Simulation Results for cond = “1010” (gt | Greater Than)



Figure 12 Simulation Results for cond = “1011” (le | Less than or Equal to)

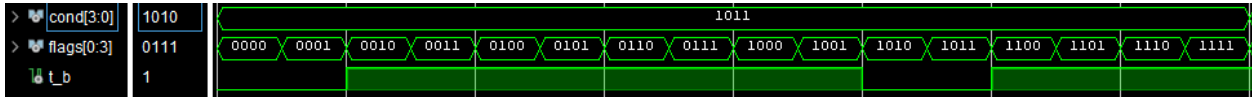


Figure 13 Simulation Results for cond = “1100” (ge | Greater than or Equal to)

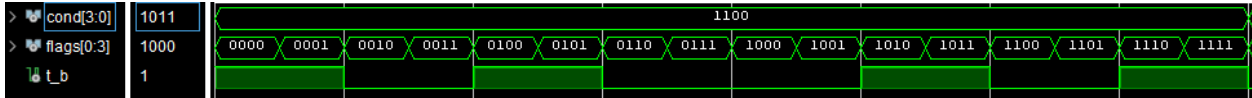


Figure 14 Simulation Results for cond = “1101” (lo | Lower than)

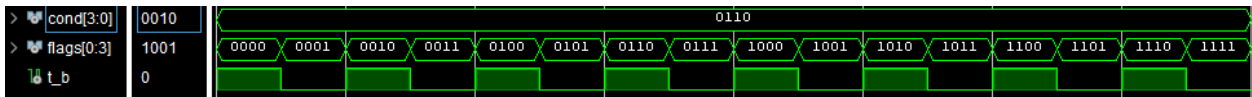


Figure 16 Simulation Results for cond = “1101” (hi | Higher than)

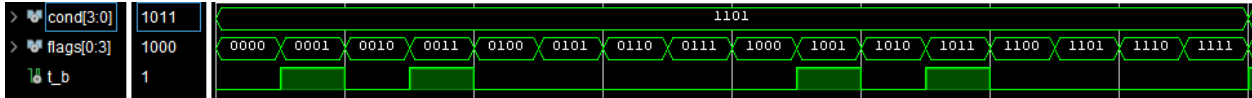


Figure 14 Simulation Results for cond = “1110” (ls | Lower than or Same as)

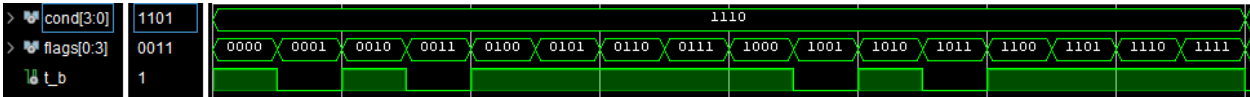


Figure 17 Simulation Results for cond = “0101” (hs | Higher than or Same as)

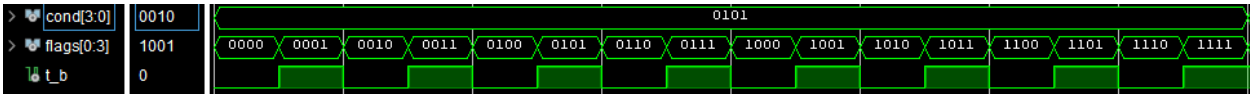
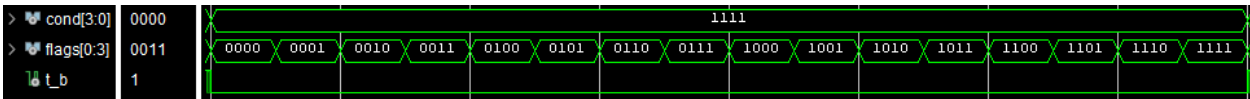

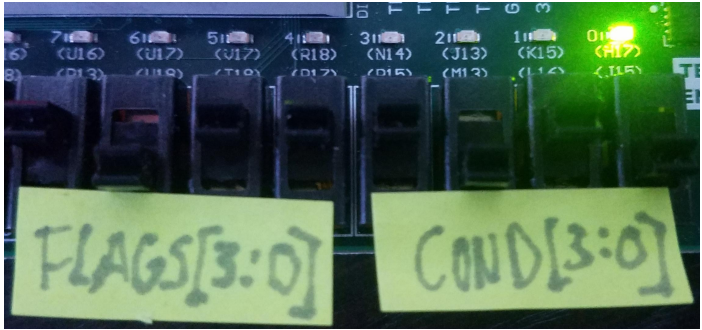
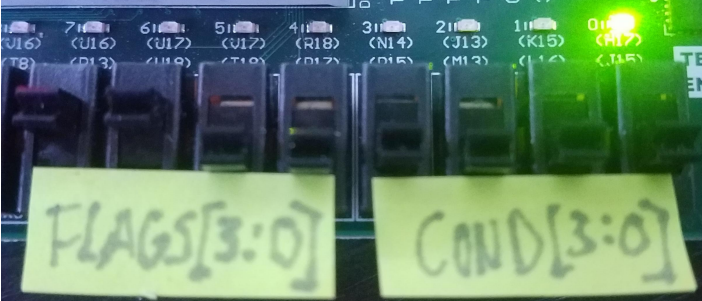


Figure 18 Simulation Results for cond = “1111” (unsupported)



SYNTHESIS RESULTS

After programming the board, all 256 input combinations were tested. For brevity, only three example images are shown.

Flags [N, Z, C, V]	Condition	Image
[0, 0, 1, 1]	0011 (eq) Z	
[1, 0, 1, 1]	1010 (gt) $N\bar{Z} + \bar{N}V$	
[1, 1, 0, 0]	1000 (vc) \bar{V}	

ISSUES ENCOUNTERED

It was initially determined that having a separate data type unnecessarily obfuscated our code, and reduced its reusability in the future, and so the CCR input was defined as an `STD_LOGIC_VECTOR`. However, after being informed in class that the desired product implements a new data type through the use of a package, we again re-wrote our code to comply.

Mistakes were also frequently made regarding the order of the *N*, *C*, *Z*, and *V* flags, as to which flags lied at which index. Perhaps it would be useful to include index constants in the package file for each of the flags.

CONCLUSIONS

Although tedious, each possible input combination was tested and manually verified to produce the proper output via simulation and synthesis.

While packages can be useful in many cases, especially for projects spanning multiple files, I fear that implementing our BTU with a package will stunt our progress when it comes to joining all our components into a CPU. This is especially true due to our previous modules (such as the ALU) that interact with the CCR not using this project's package, and having defined the CCR as a `STD_LOGIC_VECTOR` rather than a `flag_array`. Perhaps this will be mediated due to the casting functions defined in the package.

APPENDIX A: Flag_array Package VHDL Code (flag_package.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package flag_package is
    type flag_t is (N, Z, V, C);
    type flag_array is array (flag_t) of std_logic;
    function to_std_logic_vector(f : flag_array) return
std_logic_vector;
    function to_flag_array(f : std_logic_vector) return
flag_array;
end package flag_package;

package body flag_package is

    function to_std_logic_vector(f : flag_array) return
std_logic_vector is
        variable r: std_logic_vector(3 downto 0);
    begin
        r :=(f(N) & f(Z) & f(V) & f(C));
        return r;
    end function;

    function to_flag_array(f : std_logic_vector) return
flag_array is
        variable r: flag_array;
    begin
        r(N) :=f(3);
        r(Z) :=f(2);
        r(V) :=f(1);
        r(C) :=f(0);
        return r;
    end function;
end package body flag_package;
```

APPENDIX B: Branch Test Unit VHDL Code (btu.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.flag_package.all;

entity BTU is
Port (
    COND:          in std_logic_vector(3 downto 0);
    FLAGS:          in flag_array;
    TAKE_BRANCH:    out std_logic
);
end BTU;

architecture Behavioral of BTU is
    signal F: std_logic_vector(3 downto 0);
    signal N,Z,C,V,lt,gt,le,ge,hi,ls: std_logic;
begin
    F <= to_std_logic_vector(FLAGS);
    N <= F(3);
    Z <= F(2);
    V <= F(1);
    C <= F(0);
    lt <= N xor V;
    --gt <= (not(Z) and (N or V)) or (not(N) and V);
    gt <= (N AND NOT(Z)) OR (NOT(N) AND V);
    le <= Z or (N xor V);
    ge <= (N and V) or (not(N) and not(V));
    hi <= C and not(Z);
    ls <= not(C) or Z;

    with COND select
        TAKE_BRANCH <= '1'          when "0000",
                           N          when "0001",
                           not(N)     when "0010",
                           Z          when "0011",
                           not(Z)     when "0100",
                           C          when "0101",
                           not(C)     when "0110",
                           V          when "0111",
```

```
not(V)  when "1000",  
lt      when "1001",  
gt      when "1010",  
le      when "1011",  
ge      when "1100",  
hi      when "1101",  
ls      when "1110",  
'0'    when others;  
end Behavioral;
```

APPENDIX C: Branch Test Unit Testbench VHDL Code (btu_testbench.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.flag_package.all;

entity BTU_Testbench is
end BTU_Testbench;

architecture Behavioral of BTU_Testbench is
signal cond: std_logic_vector(3 downto 0);
signal flags: flag_array;
signal t_b: std_logic;
begin

testBTU: entity work.BTU (Behavioral)
    port map(COND=>cond, FLAGS=>flags, TAKE_BRANCH=>t_b);

test: process
variable i, j: integer;
begin
cond <= "0000";
for i in 0 to 15 loop
    cond <= std_logic_vector(TO_UNSIGNED(i,4));
    for j in 0 to 15 loop
        flags <=
to_flag_array(std_logic_vector(TO_UNSIGNED(j,4)));
        wait for 1ns;
    end loop;
end loop;

end process;

end Behavioral;
```

APPENDIX D: Constraints File (btu_constraints.xdc)

```
##Switches
set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 }
[get_ports { COND[0] }];
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 }
[get_ports { COND[1] }];
set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 }
[get_ports { COND[2] }];
set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 }
[get_ports { COND[3] }];
set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 }
[get_ports { FLAGS[3] }];
set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 }
[get_ports { FLAGS[2] }];
set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 }
[get_ports { FLAGS[1] }];
set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 }
[get_ports { FLAGS[0] }];

## LEDs
set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 }
[get_ports { TAKE_BRANCH }];
```