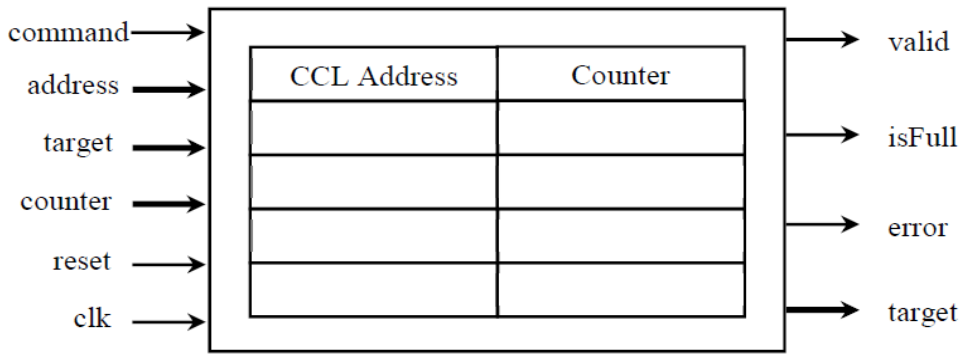
CCLU verification plan

The CCLU is a counter-controlled loop unit that is stack-based and it executes the loop for a known number of iterations, the basic instruction: **ccl target, Rs.** There's also a break command which terminates the current loop and a continue command which terminates the current iteration of the loop and starts the next iteration (if there's one!).



RTL Input :

1. Command: is an operation command size of this variable with two bits
   - 00 - do nothing
   - 01 - A ccl or a continue instruction needs to be processed.
   - 10- A break instruction that terminates the current ccl instruction prematurely
   - 11- Error
2. address - is a vector of 32 bits
3. target - the target address of the current control instruction (ccl, break, or continue)
4. counter: the initial number of loop iterations. It is fetched from the register file. It is considered for the ccl instruction first iteration. It is ignored, otherwise.
5. reset: 2-bit single that deletes all elements of the stack.
6. clk: the system clock

RTL Output :

1. valid: "1" means the value of the output target address is valid.
2. isFull:"1" indicate the stack of the CLLU is Full.
3. error: "1" indicates that there is some error.
4. target: target address of the current control instruction.

1. **Verification level**: Since we know a bit about the internal mechanism of the unit we will deterministically verify (unit verification) the stack since it's the main part of this unit, but also verify the top level (inputs and outputs). Here's a brief explanation of unit verification:
   Unit-level, This level focuses on verifying the individual components and the functionality of the CCLU unit in isolation. It includes verifying the correct operation of stack management, command interpretation, address handling, target address generation, counter decrementing, and stack extension handling. It may involve creating test benches and stimuli to exercise different scenarios and corner cases for each component.

2. **Functions under test:**
   These are the critical functions under test:

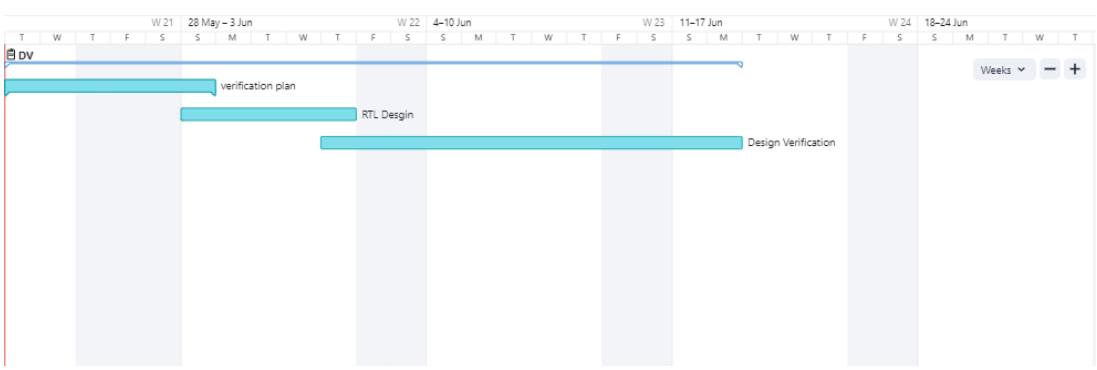
- When the input signal is 00/11 nothing is added or removed from the stack.
- When it's 01 we compare the input address with the top element in the stack:
  - They're equal:
    - If the counter is 1: then remove.
    - The counter>1: then decrease by one.
  - Not equal AND counter>1 AND stack not full: add a new element to stack.
- When the 10 top elements of the stack are removed.
- Target Address Generation:
  - Verify that the CCLU correctly generates the target address for the current control instruction (ccl, break, or continue).
  - Verify that the target address is set to the input target address for new CCL instructions or break instructions.

There are also some secondary functions that we can test:

- Memory is full —> isFull output bit on + error bits is raised.
- 11 input signal or counter input is 0 —> error output bit is on + valid bit is 0.
- In case of no errors input target address == output target address and the valid bit is raised.
- When the reset bit is 1 and the command bits are 01 / 10 —> raise the error bit.

3. **Resource requirements**: a team of 3 people will verify the RTL design.
4. **Required tools**:
   - Simulation engines.
   - Waveform viewer.
   - Verification software environment tools like EDA playground.

5. **Schedule**:
   By Sunday we need to finish the verification plan
   first week-design RTL
   Second and third weeks - Verify the design



6. **Specific test and methods**: will use gray box verification and our strategy is deterministic driven tests so we can check the data using I/O data correctness and timing rules.

7. **Completion criteria**: when the bug rate drops and we cover most cases testing will be done.

8. **Test scenarios:**

- Clocking tests:
  - Reset input signal is 1 —> empty stack in one clock cycle

Cycle clock	cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
1	01	1016	1000	1	1	0	-	0	0	1000	0
2	01	1016	1000	1	1	0	-	0	1	1000	0

  - Any error is raised at the same time as the error

Cycle clock	cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
1	11	1016	1000	1	1	0	-	0	1	1000	0

  - When there's a valid execution of the operation —> in the next clock cycle valid bit is raised and the input target becomes the output target.
- Counter tests:
  If the counter is 0 raise the error output bit

cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1016	1000	0	1	16	-	0	1	1000	0

- If the counter is 1 nothing happens in the stack and just the valid and target outputs are changed.

cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1016	1000	1	1	16	-	1	0	1000	0

s

1000:L1:Instruction 1;

1004:Instruction 2;

1008:Instruction 3;

1012:ccl L1, Rs; Rs; #Rs = 1

Input					New Internal State		Output			
cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1012	1000	1	0	0	-	1	0	1000	0

- Command line tests:
  - Test if all 4 values are executed correctly

1000:L1:Instruction 1;

1004:Instruction 2;

1008:Instruction 3;

1012:ccl L1, Rs; Rs; #Rs = 2

1. The command is 01:

Input					Expected New Internal State		Expected Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1012	1000	3	0	1	2	1	0	1000	0
01	1012	1000	-	0	1	1	1	0	1000	0
01	1012	1000	-	0	0	-	1	0	1000	0

2. The command is 11:

Input					Expected New Internal State		Expected Output			
cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
11	1012	1000	3	0	1	2	0	1	1000	0

3. The command is 10:

Input					Expected New Internal State		Expected Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1012	1000	3	0	1	2	1	0	1000	0
10	1012	1000	-	0	0	0	1	0	1000	0

4. The command is 00:

Input					Expected New Internal State		Expected Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1016	1000	3	0	1	2	1	0	1000	0
00	-	1000	-	-	1	2	0	0	100	0

- Corner tests:
  - Raise the reset bit with 11/10/01 command

1000:L1:Instruction 1;

1004:Instruction 2;

1008:Instruction 3;

1012:ccl L1, Rs; Rs; #Rs = 2

1. The command is 01:

Input					Expected New Internal State		Expected Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1012	1000	3	0	1	2	1	0	1000	0
01	1012	1000	-	1	0	0	0	0	0	0

2. The command is 11:

Input					Expected New Internal State		Expected Output			
cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
11	1012	1000	3	1	0	0	0	1	0	0

3. The command is 10:

Input					Expected New Internal State		Expected Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
10	1012	1000	3	1	0	0	0	1	0	0

- Try to add a new CCL loop when the stack is full. Let's say that the stack is 3 memory blocks long:
  1000:L1:Instruction 1;

  1004:L2:Instruction 2;

  1008:L3:Instruction 3;

  10012:L4:Instruction 4;

  1016:ccl L1, Rs; #Rs = 2

  1020:ccl L2, Rs; #Rs = 2

  1024:ccl L3, Rs; #Rs = 2

  1028:ccl L4, Rs; #Rs = 2

Input					Expected New Internal State		Expected Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1016	1000	2	0	1	2	1	0	1000	0
01	1020	1004	2	0	2	2	1	0	1004	0
01	1024	1008	2	0	3	2	1	0	1008	0
01	1028	1012	2	0	-	-	0	1	0	1

- Enter many continue instructions after one another until iterations become 1

Input					New Internal State		Output			
cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1000	2000	5	0	0	-	1	0	2000	0

- The counter entered is zero / very large number of iterations (counter >>)

Input					New Internal State		Output			
cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1000	1000	0	0	0	-	0	1	1000	0

- Try to reset an empty stack

Input					New Internal State		Output			
cmd	address	target	count	rst	Stack Size	count	V	Err	Target	Full
01	1000	1000	5	0	1	1	1	0	1000	0
00	-	-	-	1	0	-	0	0	0	0

- 10 instruction (break instruction) and the stack is empty

Input					New Internal State		Output			
cmd	Address	target	count	rst	Stack Size	count	V	Err	Target	Full
10	1012	1000	0	0	0	-	0	0	1000	0

9. **Risks and Dependencies**:

   The risk of dependencies on specific tools, simulators, or development environments that may impact the CCLU's design, verification, or integration process. The dependencies are: Ensure that the necessary tools and environments are available, properly configured, and compatible with the CCLU design and verification flows. Have contingency plans in place if any tool or environment becomes unavailable or incompatible.