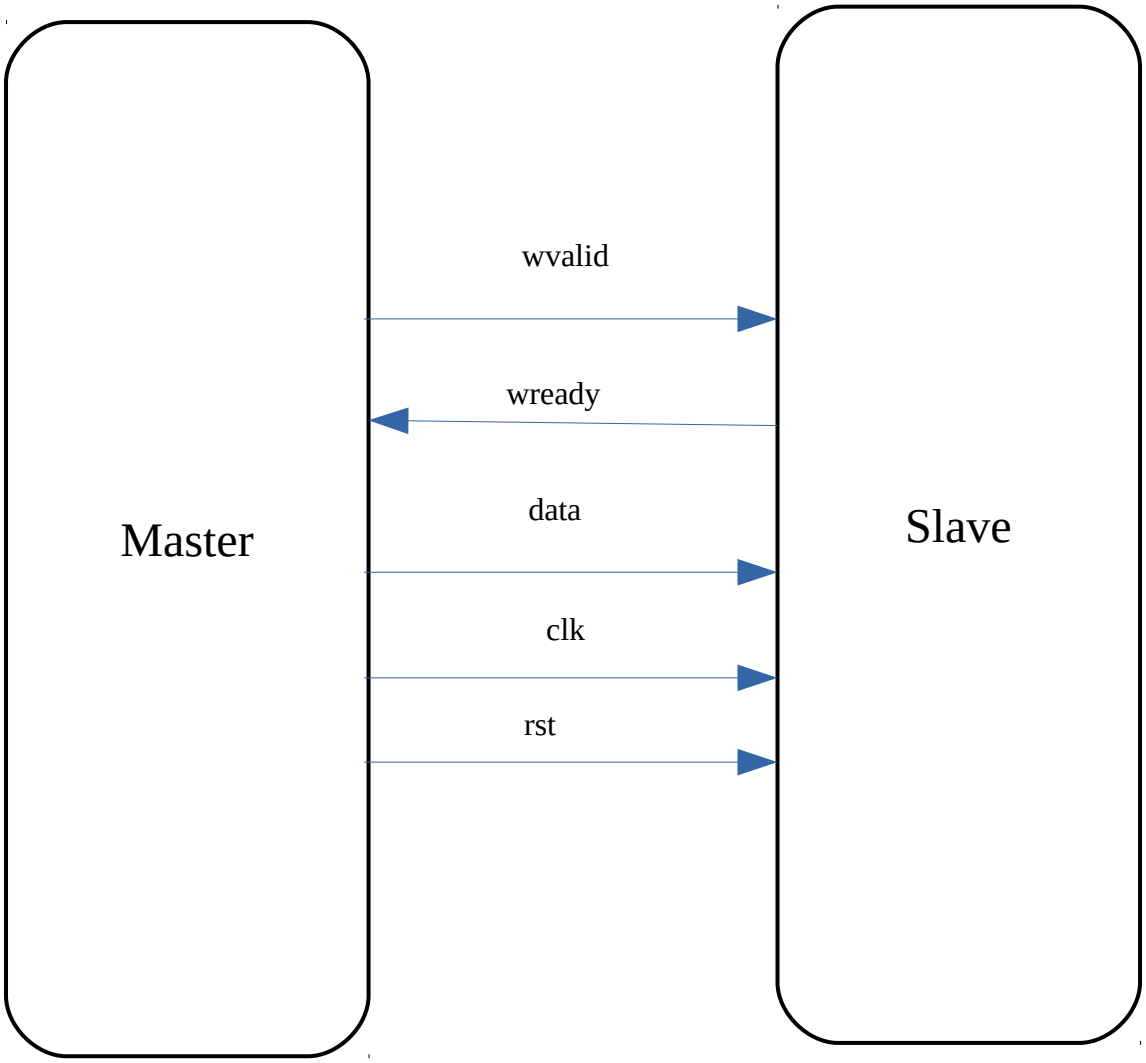
TD 0x01	Codesign		Bilel
	ready/valid handshake communications channel		

EX01:

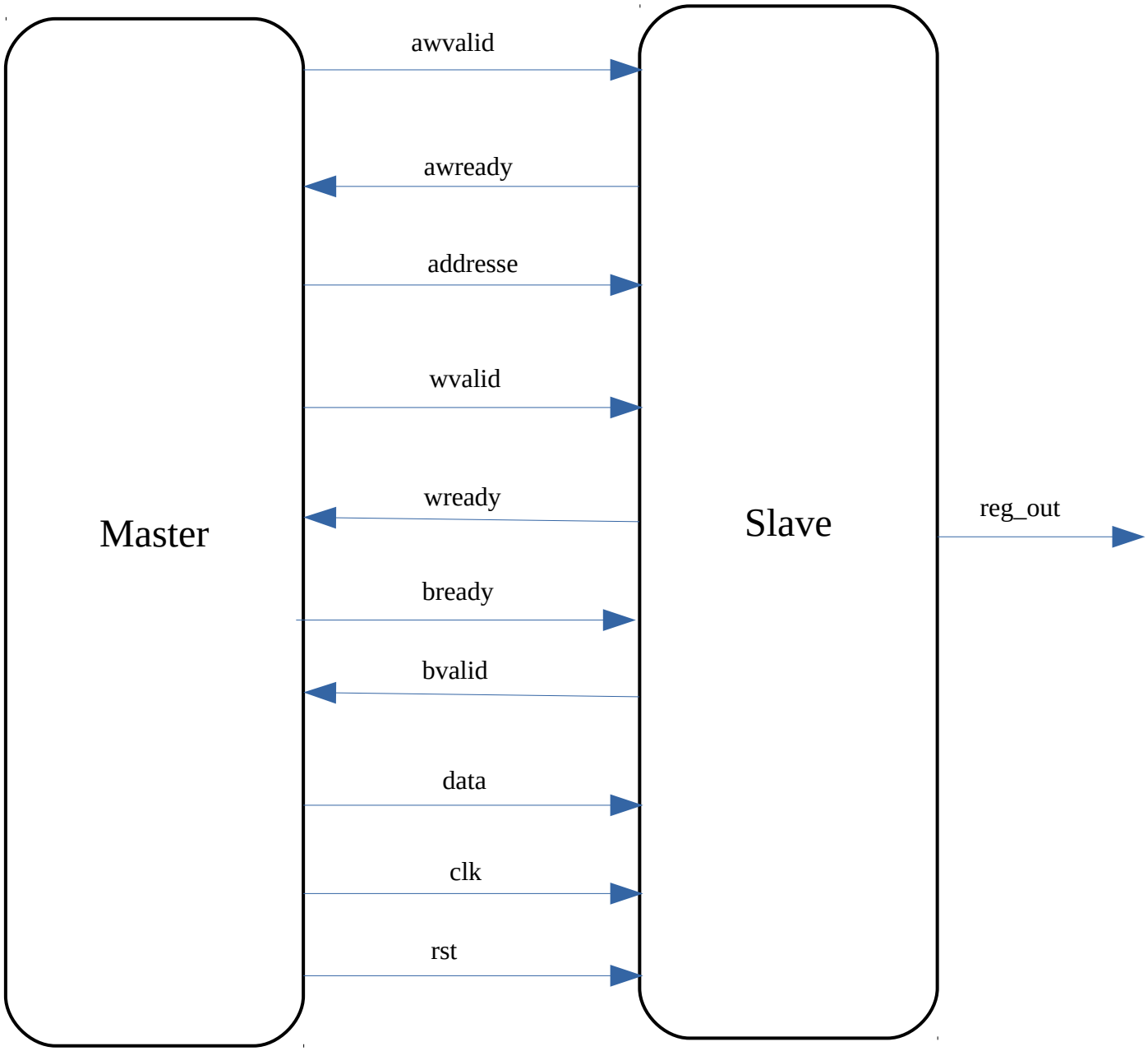
To understand ready/valid handshake lets suppose that we have a master and a slave modules, some data will be written into a slave register.
When both slave and master are ready to receive data and ready to send data respectively the transaction will occur.



- Extract the condition using the ready and valid signals to allow reading the data line?
- write a process that allows the reset of the bus lines?
- write a process to activate the read from the data line?
- write a process that reads the data and stock them in a register when the handshake occurs?
- write a testbench to validate your hdl module?

EX02:

We need to make a ready/valid handshake communications write channel for this purpose we will specify this channel as described bellow:



Bus ports

clk	Clock signal	input
rst	Reset signal	input
awready	Address write ready : generated by the slave to indicate that slave is ready for latching the address	Output
awvalid	Address write valid : generated by the master to indicate that the write address is valid.	Input
wready	Write ready: generated by the slave to indicate that slave is ready for latching the address	Output
wvalid	Write valid: generated by the master to indicate that the data is valid.	Input
data	Data to be written	input
address	Register Address where to perform the write operation	input
reg_out	Output the content of the first register	output
bready	Write response ready: generated by the master to indicate that it’s ready for write response.	input
bvalid	Write valid: generated by the master to indicate that the data is valid.	output

Bus datawidth : 8 bits
Slave data registers number: 2 register.

For this purpose we need 5 processes, the processes function is described bellow:

awready generation process:
awready is asserted for one clk clock cycle when both awvalid and wvalid are asserted(both equal to one). awready is de-asserted when reset is low.
slave is ready to accept write address when there is a valid write address and write data on the write address and data bus.

awaddr latching process:
a process should be used to used to latch the address when both
awvalid and wvalid are valid(both equal to one).

wready generation process:
wready is asserted for one clk clock cycle when both awvalid and wvalid are asserted. wready is de-asserted when reset is low.

Data writing implementation process:
Implement memory mapped register select and write logic generation,
The write data is accepted and written to memory mapped registers when awready, wvalid, wready and wvalid are asserted.

Use signals to store data in the 4 registers

```
signal reg1 : std_logic_vector(width-1 downto 0);

signal reg2 : std_logic_vector(width-1 downto 0);
```

NB : Use the following signals to output both awready and awvalid

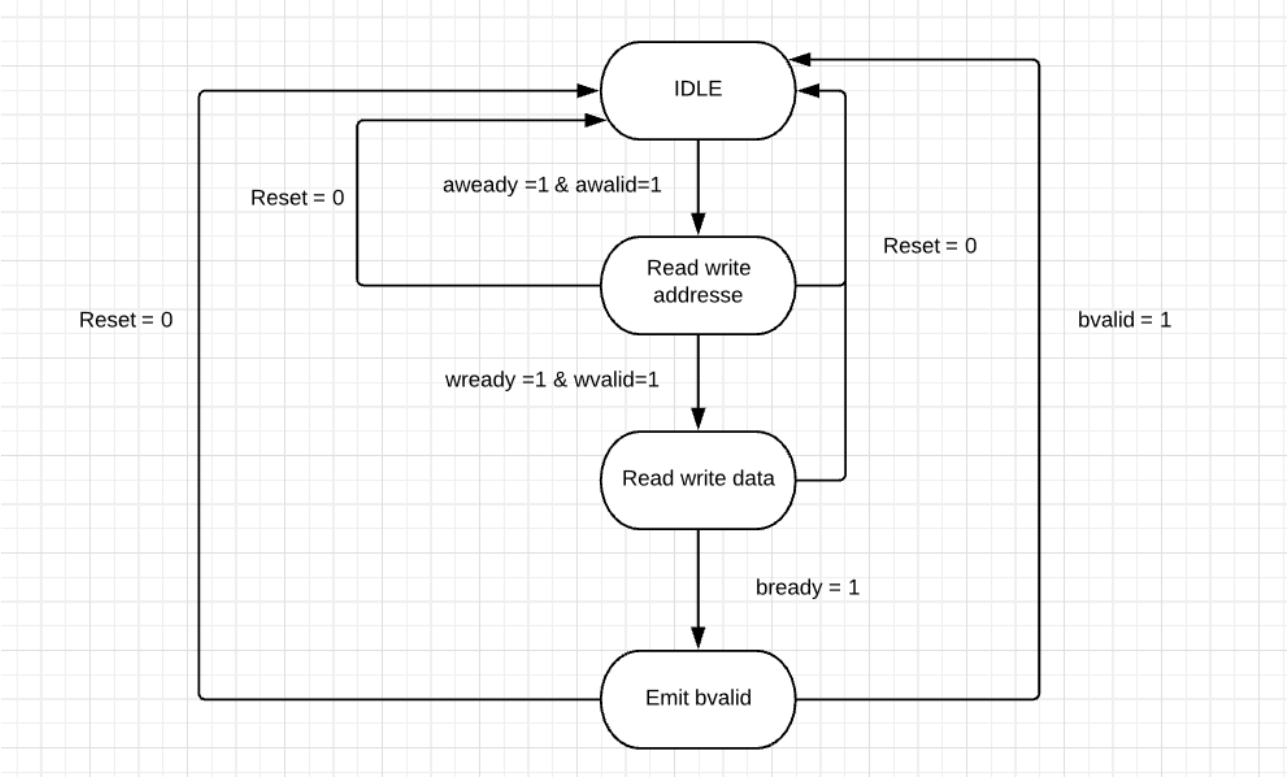
```
awready <= rvh_awready;
wready <= rvh_wready;
```

Write response generation process:
Implement write response logic generation

The write response and response valid signals are asserted by the slave when wready, wvalid, wready and wvalid are asserted.

This marks the acceptance of address and indicates the status of write transaction.

- The following FSM illustrates the mechanism of the interfaces



-extract the conditions to emit awready, wready ,and bvalid by the slave.

- write VHDL file to describe a salve component with a ready/valid handshake communications capability.

- write a testbench to test the communication bus.

NB:

- the following process to generate simulation clock

Constant ClockPeriod : TIME := 5 ns;

```
 -- Generate S_AXI_ACLK signal
 GENERATE_REFCLOCK : process
 begin
   wait for (ClockPeriod / 2);
   ClockCount:= ClockCount+1;
   tb_clk <= '1';
   wait for (ClockPeriod / 2);
   tb_clk <= '0';
 end process;
```

wait until and wait for can be used to wait for a condition event and for a certain simulation clock value respectively:

```
wait until (tb_awready and tb_wready) = '1';
wait for 5 ns;
```

- the following process could be used in your TB to simulate the behavior of a master trying to send some data through the interface when sendIt signal is high

```
send : PROCESS
 BEGIN
    tb_awvalid<='0';
    tb_wvalid<='0';

    loop
        wait until sendIt = '1';
        wait until tb_clk = '0';
            tb_awvalid<='1';
            --tb_awvalid<='0'; --let's mess a little bit with
our slave
            tb_wvalid<='1';  -- Master in ready
        wait until (tb_awready and tb_wready) = '1';
--Client ready to read address/data
           tb_bready<='1'; -- Master ready to receive
response
        wait until tb_bvalid = '1';  -- Write result valid
     wait until tb_clk = '0';
            tb_awvalid<='0';
            tb_wvalid<='0';
            tb_bready<='1';
        wait until tb_bvalid = '0';  -- All finished
            tb_bready<='0';
    end loop;
 END PROCESS send;
```