Creation of a protocol driver

Practical lesson 1, Advanced Programmable Logic Systems.

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hguzman@us.es

Creation of a protocol driver

The objective of this lesson is to create a VHDL entity, not necessarily synthesizable, to facilitate the verification of a digital design. This entity receives a 32-bit data word along with a field indicating that the data is valid.

A VHDL package is provided on the course page that describes a `record` containing this data. Remember that, in order to use it, in addition to adding it to the project, you must include it using the statement `use work.protocol_common.all;` in the `LIBRARY` section of the VHDL files that need to use it. Adding code to the package is permitted if deemed necessary or appropriate.

Driver operation:

The driver must receive as inputs a clock signal, `clk`, and an `input_tran` input of the `record` type previously mentioned. Its outputs must be the protocol signals: `data`, `ena`, `startp`, and `endp`. The entity's ports are described in the following table:

Name	Data type	Direction	Function
`input_tran`	`protocol_type`	`in`	Input transaction
`clk`	`std_ulogic`	`in`	Clock signal, active on rising edge. Changes in the outputs must be synchronous with this clock.
`data`	`std_ulogic_vector` (width depends on the student's ID number)	`out`	Output data
`ena`	`std_ulogic`	`out`	Protocol enable signal
`startp`	`std_ulogic`	`out`	Transmission start signal
`endp`	`std_ulogic`	`out`	Transmission end signal

When the `valid` field of `input_tran` is ‘1’, the driver must send the data that is currently in the `data` field of `input_tran`, and must send it according to the protocol described in the next section. When a transaction needs to be sent, the module using the driver will set `input_tran.valid` to ‘1’ during a single clock cycle.

Protocol description:

The protocol consists in sending a 32-bit data word, serialized into blocks of 1 to 16 bits, which are sent sequentially (meaning one block at a time).

When data is to be sent, the `ena` signal must be activated. Following this, the `startp` signal must be enabled for one clock cycle, after which the data must be sent in the correct order through the `data` port. When no data is being sent, the `data` output must be set to high impedance (`'Z'`). Finally, the `endp` signal must be enabled for one clock cycle and the `ena` signal must be disabled. Between any of these steps, there may be wait cycles, as described in the following section.

Personalization with DNI / NIE:

Using the digits of the student's DNI / NIE:

d7 / char1	d6	d5	d4	d3	d2	d1	d0	char0

The following protocol parameters will be calculated:

Width of serialized data (width of data blocks):
d0 modulo 5:

- 0: 16-bit width
- 1: 8-bit width
- 2: 4-bit width
- 3: 2-bit width
- 4: 1-bit width

Order of serialized data:
d1 is:

- Even: data blocks are sent starting with the most significant and ending with the least significant.
- Odd: data blocks are sent starting with the least significant and ending with the most significant.

Polarity of the `ena` signal:
d2 is:

- Even: `ena` is active low
- Odd: `ena` is active high

Polarities of the `startp` and `endp` signals:
d3 is:

- Even: active high
- Odd: active low

Cycles, after activating `ena`, that must be waited before activating the `startp` signal:
d3

Cycles, after activating `startp`, that must be waited before assigning the value of the first data block to `data`:
d4

Duration, in cycles, of each data block:

d5 * 10 + d4

If this calculation results in a value of 0, a value of 1 must be used instead. The value in `data` must remain constant during this number of cycles.

Cycles, counting from the end of the last cycle of the last data block, that must be waited before activating the `endp` signal:
d6

Cycles, after activating the `endp` signal, that must be waited before disabling `ena`:
d6 + d5

Minimum cycles, after disabling `ena`, that must be waited before reactivating `ena` if a new transaction is to be sent:
d5 + d4

Realization and evaluation of the practical lesson:

Students must develop a protocol driver adapted to their student's ID number (NIF/NIE/etc). It is recommended to draw waveform diagrams of the resulting protocol before writing the VHDL code.

For the development of the driver, one option is to use processes without a sensitivity list that use the `wait` statement. The `wait` statement can be used in the following ways:

```
wait for <time>; -- Waits for the specified time, for example: wait for 10 ns;

wait on <sensitivity list> -- Waits for one of the signals specified in the sensitivity list to change, for example: wait on enable, disable;

wait until <condition> -- Waits for one of the signals in the condition to change and for the condition to be met, for example: wait until enable = '1';

wait; -- Without further arguments, this statement waits indefinitely, preventing the process from terminating.
```

Enhancements to the basic functionality will be considered when grading, such as:

- Reporting when a bus transaction starts and ends using the `report` statement.
- Considering what happens if the driver receives a new transaction while it is busy and implementing a solution.

A report on the practical exercise must be prepared, describing the work performed and demonstrating the correct implementation of the protocol driver. The report must include calculations of the protocol parameters based on the student's ID number and the expected waveform. For this purpose, https://wavedrom.com/editor.html is a good resource to generate waveform diagrams. The demonstration of functionality must be based on simulations and screen captures of the resulting waveforms. In addition to the report, all developed code must be submitted, including testbenches and the `protocol_common` package.