Macro usage guideline.

Any parameterizable entity whose value is the same throughout the entire design and verification should be defined as a macro.
A macro is defined outside the verilog code with a macro definition compiler directive:

```
`define macro_name macro_value
```

Examples of parameterizable entities:
- bus width

```
`define ADDRBLENGTH 16
.....
wire[`ADDRBLENGTH-1:0] addrBus;
```

- bus field

```
`define ROMSELECT 15:13
.....
romSelect = addrBus[`ROMSELECT];
```

- FSM state code

```
`define START 3'b011
.....
state <= `START;
```

- constant value

```
`define JMPR 4'b1101
.....
if(opcode == `JMPR);
```

Macros can be cascaded, using macros for definition of other macros

```
`define ADDRBLENGTH 16
`define ROMSELECT `ADDRBLENGTH-1:`ADDRBLENGTH-3
```

Macros can be used in testbenches for input stimuli, monitor and checking, rendering tests independent of the actual parameter value.

```
for(
    addrmem = `ADDRBLENGTH'd0;
    addrmem < `ADDRBLENGTH'd1 << `ADDRBLENGTH;
    addrmem = addrmem + `ADDRBLENGTH'd1
)
// addrmem varies from 0 to 2^ADDRBLENGTH -1
```

To be visible throughout the entire project, all macro defines are kept in separate file with a suitable extension, for example *macro_file_name*.h, which would be included in compilation with all the verilog files that use that macros. Any verilog file that uses that macros should have in the header section, before the verilog code, an include compilation directive:

```
`include "macro_file_name.h"
```