POLITECNICO DI MILANO

Getting started with Bambu in one hour

Fabrizio Ferrandi <fabrizio.ferrandi@polimi.it >
Politecnico di Milano

Outline

- Introduction to bambu
- First example
- Compiler Based Optimizations, Tuning and Customization of Generated Accelerators
- Synthesis and optimization of memory accesses

Tutorial Material

- Slides and the VM with a pre-configured environment can be downloaded from:

https://panda.dei.polimi.it/?page_id=811

Update the git repository

- Change the directory to:

```
cd ~/PandA-bambu
```

- And then

```
git pull
```

- Switch to the FPT branch:

```
git checkout PNNL19_tutorial
```

First synthesis

- Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_pnnl_2019/intro/first
```

- Edit the file C by typing:

```
gedit module.c
```

```
unsigned short icrc1(unsigned short crc, unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);
    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

Run bambu

- Run the script by typing:

```
./bambu.sh
```

```
Summary of resources:
    - ASSIGN_SIGNED_FU: 2
    - IIconvert_expr_FU: 1
    - IUdata_converter_FU: 2
    - MUX_GATE: 2
    - UIdata_converter_FU: 3
    - UUdata_converter_FU: 1
    - bit_xor_expr_FU: 1
    - constant_value: 5
    - ge_expr_FU: 1
    - lshift_expr_FU: 1
    - ne_expr_FU: 1
    - plus_expr_FU: 1
    - read_cond_FU: 1
    - register_SE: 2
    - ui_bit_xor_expr_FU: 1
    - ui_cond_expr_FU: 1
    - ui_lshift_expr_FU: 1
Start reading vector           1's values from input file.

Value found for input crc: 0000000000001010
Value found for input onech: 00000010
Reading of vector values from input file completed. Simulation started.
Value found for output ex_return_port: 0010101001000010
 return_port = 10818   expected = 10818

Simulation ended after                  10 cycles.

Simulation completed with success

- HLS_output//simulation/testbench_icrc1_minimal_interface_tb.v:441: Verilog $finish
1. Simulation completed with SUCCESS; Execution time 10 cycles;
  Total cycles             : 10 cycles
  Number of executions     : 1
  Average execution        : 10 cycles
```

Simulation & Synthesis

- Testbench generated automatically
  - test_icrc1.xml
- Simulation and synthesis scripts generated automatically:
  - icrc1/simulate_icrc1_minimal_interface.sh
  - icrc1/synthesize_Synthesis_icrc1_minimal_interface.sh
- Verilog file generated at the end of the HLS step:
  - icrc1/icrc1.v

Analyze results

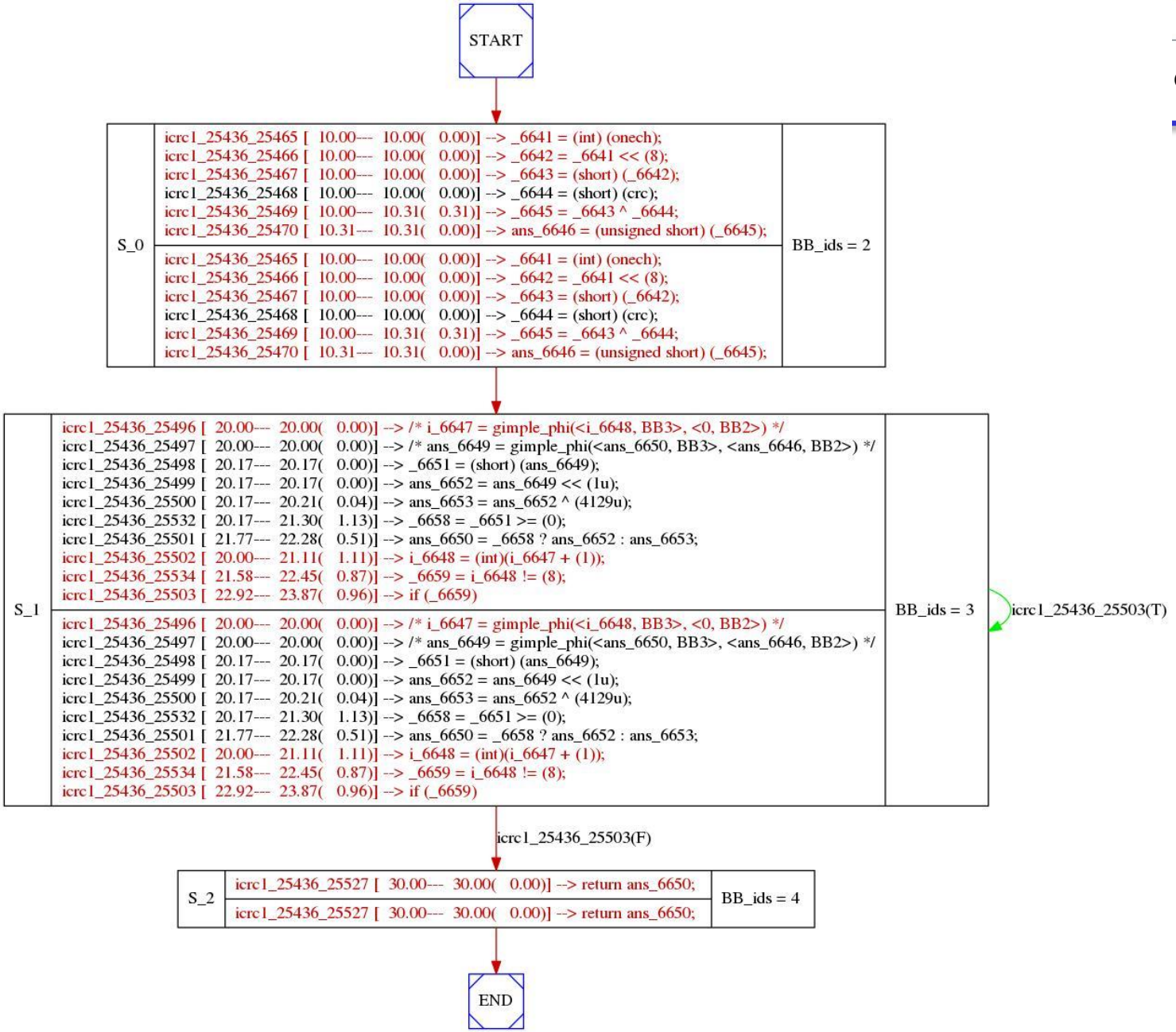
- Change directory to icrc1:

```
cd icrc1
```

- Display the FSM:

```
xdot HLS_output/dot/icrc1/HLS_STGraph.dot
xdot HLS_output/dot/icrc1/fsm.dot
```

FSM


START
S_0
icrc1_25436_25465 [10.00--- 10.00(0.00)] --> _6641 = (int) (onech);
icrc1_25436_25466 [10.00--- 10.00(0.00)] --> _6642 = _6641 << (8);
icrc1_25436_25467 [10.00--- 10.00(0.00)] --> _6643 = (short) (_6642);
icrc1_25436_25468 [10.00--- 10.00(0.00)] --> _6644 = (short) (crc);
icrc1_25436_25469 [10.00--- 10.31(0.31)] --> _6645 = _6643 ^ _6644;
icrc1_25436_25470 [10.31--- 10.31(0.00)] --> ans_6646 = (unsigned short) (_6645);
BB_ids = 2
S_1
icrc1_25436_25496 [20.00--- 20.00(0.00)] --> /* i_6647 = gimple_phi(<i_6648, BB3>, <0, BB2>) */
icrc1_25436_25497 [20.00--- 20.00(0.00)] --> /* ans_6649 = gimple_phi(<ans_6650, BB3>, <ans_6646, BB2>) */
icrc1_25436_25498 [20.17--- 20.17(0.00)] --> _6651 = (short) (ans_6649);
icrc1_25436_25499 [20.17--- 20.17(0.00)] --> ans_6652 = ans_6649 << (1u);
icrc1_25436_25500 [20.17--- 20.21(0.04)] --> ans_6653 = ans_6652 ^ (4129u);
icrc1_25436_25532 [20.17--- 21.30(1.13)] --> _6658 = _6651 >= (0);
icrc1_25436_25501 [21.77--- 22.28(0.51)] --> ans_6650 = _6658 ? ans_6652 : ans_6653;
icrc1_25436_25502 [20.00--- 21.11(1.11)] --> i_6648 = (int)(i_6647 + (1));
icrc1_25436_25534 [21.58--- 22.45(0.87)] --> _6659 = i_6648 != (8);
icrc1_25436_25503 [22.92--- 23.87(0.96)] --> if (_6659)
BB_ids = 3
icrc1_25436_25503(T)
icrc1_25436_25503(F)
S_2
icrc1_25436_25527 [30.00--- 30.00(0.00)] --> return ans_6650;
BB_ids = 4
END

Open the RTL Verilog


```
module icrc1_minimal_interface(clock, reset, start_port, crc, onech,
done_port, return_port);
  // IN
  input clock;
  input reset;
  input start_port;
  input [15:0] crc;
  input [7:0] onech;
  // OUT
  output done_port;
  output [15:0] return_port;
  // Component and signal declarations

  icrc1 icrc1_i0 (.done_port(done_port), .return_port(return_port),
.clock(clock), .reset(reset), .start_port(start_port), .crc(crc),
.onech(onech));

endmodule
```

Outline

- Compiler Based Optimizations, Tuning and Customization of Generated Accelerator
  - Tuning accelerators by means of optimizations
  - Math support

Improve Area/Performance of generated accelerators

- Performance and/or area of the generated accelerators can be improved by tuning the design flow
  - GCC/CLANG optimizations
  - Bambu IR optimizations
  - Bambu HLS algorithms
- Best design flow for every accelerator does not exist
  - Trade off between area and performance
  - Effects of the single optimizations can be different on the single accelerators
- Default:
  - Balanced area/performance trade off

Translation flow

- C→HDL without optimizations
  - GCC/CLANG optimizations are (mostly) disabled
  - Bambu IR optimizations are (mostly) disabled

```
-O0 --cfg-max-transformations=0 --no-chaining
```

- Can be exploited only when bambu is compiled with development support
- Useful for debugging

GCC/CLANG Optimizations

- Only GCC target independent optimizations are considered
- -O3 is not necessarily the best choice
  - Can improve performances
  - Can increment area
- User can tune this part of the flow:
  - Selecting optimization level:

```
-O0 or -O1 or -O2 or -O3 or -Os
```

  - Enabling/disabling single GCC/CLANG optimization:

```
-f<optimization> -fno-<optimization>
```

  - Tuning gcc parameters: --para

```
--param <name>=<value>
```

Effect of GCC Optimizations

- Results refer to other Bambu options set to default value

Opts	Cycles	Luts
O0	15764	11675
O1	7892	11052
O2	4679	10276
O3	3854	15679
O3 vectorize	3816	38553
O3 all inline	1327	13550

Bambu IR Analysis

- Collect information used by IR optimizations and High Level Synthesis
- Data flow analysis
  - Scalar: based on SSA
  - Aggregates: exploit GCC+Bambu alias analysis
- Graphs Computation
  - Call Graph, CFG, DFG, …
- Loops identification
- Bit Value Analysis
  - Compute for each SSA which bit are used and which bit are fixed

Bambu IR Optimizations

- Applied before HLS to the IR produced by GCC
- Two type of optimizations
  - Single instruction optimizations
  - Multiple instruction optimizations
  - Restructuring of Control Flow Graph
  - Fixing IR
- Sequences of optimizations can be applied multiple times
  - Fixed point iteration optimization flow

Single Instruction Optimization

- IR lowering – make single instructions more suitable to be implemented on FPGA
  - Expansion of multiplication by constant
  - Expansion of division by constant
  - Etc.
- Bit Value Optimization
  - Shrink operations to the only significant bits

Multiple Instruction Optimization

- Common Subexpression Elimination
- Dead Code Elimination
- Extract pattern (e.g., three input sum)
- LUT transformations
  - Merging multiple Boolean operations into a single LUT-based operation
- Cond Expr Restructuring

Restructuring of Control Flow Graph

- Speculation
- Code motion
- Merging of conditional branch
  - Creation of multiple target branch
- Basic Block Removal
  - Empty
  - Last

System of Difference Constraints

- Global scheduling based on ILP formulation
- Results are exploited to perform
  - Speculation
  - Code Motion

+ Improve performances of accelerators

- Potentially increment area of accelerators

- Increase High Level Synthesis time

Experimental setup

- Predefined design flows

```
--experimental-setup=<setup>
```

BAMBU-AREA: optimized for area

BAMBU-PERFORMANCE: optimized for performances

BAMBU-BALANCED: optimized for trade-off area/performance

BAMBU-AREA-MP, BAMBU-PERFORMANCE-MP, BAMBU-BALANCED-MP: enable support to true dual port memories

Default: BAMBU-BALANCED-MP

Constraints

- Bambu assumes infinite resources during High Level Synthesis
  - Produced solutions may not fit in the target device
- Area of generated solutions can be indirectly controlled by means of **constraints**
- User can constraint the number of available functional units in each function
  - E.g.: fix the number of available multiplier in each function
- Constraints are set by means of *XML file*

Example of constraints file


```
<?xml version="1.0"?>
  <constraints>
    <HLS_constraints>
      <tech_constraints fu_name="mult_expr_FU"
                        fu_library="STD_FU" n="8"/>
    </HLS_constraints>
  </constraints>
```

Integer Division Algorithms

- You can control how to implement integer divisions:

```
--hls-div=<implementation>
```

- Available implementations:
  - `none`: HDL based pipeline restoring division
  - **`nr1`** (default): C-based non restoring division with unrolling factor equal to 1
  - `nr2`: C-based non restoring division with unrolling factor equal to 2
  - `NR`: C-based Newton-Raphson division
  - `as`: C-based align divisor shift dividentd method

Floating point support

- Possible ways of implementing floating point ops:
  - Softfloat (default): customized faithfully rounded (nearest even) version of soft based implementation

```
--soft-float
```

  - Softfloat-subnormal: soft based implementation with support to subnormal

```
--softfloat-subnormal
```

  - Softfloat GCC: GCC soft based implementation

```
--soft-fp
```

  - Flopoco generated modules

```
--flopoco
```

Floating point Division Algorithms

- You can control how to implement floating point divisions:

```
--hls-fpdiv=<implementation>
```

- Available implementations:
  - **SRT4** (default): use a C-based Sweeney, Robertson, Tocher floating point division with radix 4
  - G: use a C-based Goldschmidt floating point division.

Libm versions

- Bambu exploits High Level Synthesis to generate accelerators implementing libm functions
- Two different versions of libm are available
  1. Faithfully rounding (default)
  2. Classical libm built integrating existing libm source code from glibc, newlib, uclibc and musl libraries.
     - Worse performances and area

First example – Evaluate GCC/CLANG optimizations

- Evaluate the effects of GCC/CLANG optimizations on the number of cycles of adpcm benchmark
  - Different level of optimizations
  - Vectorization
  - Different inlining

- aes from CHStone suite
  - Yuko Hara, Hiroyuki Tomiyama, Shinya Honda and Hiroaki Takada, "Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis", *Journal of Information Processing*, Vol. 17, pp.242-254, (2009).
- Target device: xc7z020-1clg484-VVD (Zynq with Vivado)
- Target clock period: 10 ns

Solution Hint


```
bambu adpcm.c -O0 --simulate
```

Solution


```
bambu adpcm.c -O0 --simulate

bambu adpcm.c -O2 --simulate --
compiler=I386_CLANG6

bambu adpcm.c -O2 --simulate

bambu adpcm.c -O3 --simulate

bambu adpcm.c -O3 --simulate
-finline-limit=1000000

bambu adpcm.c -O3 --simulate -ftree-vectorize
```

Second example – Use SDC Scheduling

- Check if SDC scheduling based optimizations further improve the best results obtained in the previous activity

Solution Hint


```
--speculative-sdc-scheduling
```

Solution


```
bambu adpcm.c -O3 --simulate
--speculative-sdc-scheduling
-finline-limit=1000000
```

Outline

- Optimization of the memory architecture
  - Explore different allocation policy
  - Multi-channel design space exploration
  - Control the Load/Store latency
  - Control asynchronous memories inference
  - Alignment hints
  - Customize memory layout

First example – memory allocation policy

Explore different allocation policy

```
--memory-allocation-policy=<type>
        Set the policy for memory allocation. Possible values for the <type>
        argument are the following:
            ALL_BRAM           - all objects that need to be stored in memory
                                 are allocated on BRAMs (default)
            LSS                - all local variables, static variables and
                                 strings are allocated on BRAMs
            GSS                - all global variables, static variables and
                                 strings are allocated on BRAMs
            NO_BRAM            - all objects that need to be stored in memory
                                 are allocated on an external memory
            EXT_PIPELINED_BRAM - all objects that need to be stored in memory
                                 are allocated on an external pipelined memory
```

Solution Hint


```
$ bambu adpcm.c --memory-allocation-policy=LSS
--clock-period=15 --simulate -v3
```

Look for the log section:

```
Memory allocation information:
    Variable external to the top module: test_result - 25438
- test_result
      Id: 25438
      Base Address: 1073741824
      Size: 400
      Is a Read Only Memory
      Used &(object)
      Number of functions in which is used: 1
      Maximum number of references per function: 1
      Maximum number of loads per function: 1
    …
```

Solution


```
$ bambu adpcm.c --memory-allocation-policy=ALL_BRAM
--clock-period=15 --simulate -v3

$ bambu adpcm.c --memory-allocation-policy=LSS
--clock-period=15 --simulate -v3

$ bambu adpcm.c --memory-allocation-policy=GSS
--clock-period=15 --simulate -v3

$ bambu adpcm.c --memory-allocation-policy=NO_BRAM
--clock-period=15 --simulate -v3

$ bambu adpcm.c
--memory-allocation-policy=EXT_PIPELINED_BRAM
--clock-period=15 --simulate -v3
```

Second example – Multi-channel design space exploration


```
--channels-type=<type>
      Set the type of memory connections.
      Possible values for <type> are:
          MEM_ACC_11 - the accesses to the memory have a single direct
                       connection or a single indirect connection (default)
          MEM_ACC_N1 - the accesses to the memory have n parallel direct
                       connections or a single indirect connection
          MEM_ACC_NN - the accesses to the memory have n parallel direct
                       connections or n parallel indirect connections

  --channels-number=<n>
      Define the number of parallel direct or indirect accesses.
```

- When BRAMs are involved only two ports at maximum could be given
- When option `--memory-allocation-policy=EXT_PIPELINED_BRAM` is given the number of channels could be greater than 2

Solution Hint


```
$ bambu adpcm.c --channels-type=MEM_ACC_NN --memory-
allocation-policy=EXT_PIPELINED_BRAM --channels-number=4
--clock-period=15 --simulate -v3
```

Look how long it take the simulation.

Consider `-fwhole-program` option

Third example – Control the Load/Store latency


```
--memory-ctrl-type=type
       Define which type of memory controller is used.
       Possible values for the <type> argument are the following:
           D00 - no extra delay (default)
           D10 - 1 clock cycle extra-delay for LOAD, 0 for STORE
           D11 - 1 clock cycle extra-delay for LOAD, 1 for STORE
           D21 - 2 clock cycle extra-delay for LOAD, 1 for STORE

--bram-high-latency=[3,4]
       Assume a 'high latency bram'-'faster clock frequency'
       block RAM memory based architectures:
       3 => LOAD(II=1,L=3) STORE(1).
       4 => LOAD(II=1,L=4) STORE(II=1,L=2).

--mem-delay-read=value
       Define the external memory latency when LOAD are performed (default 2).

--mem-delay-write=value
       Define the external memory latency when LOAD are performed (default 1).
```

Solution Hint


```
$ bambu mips.c --memory-ctrl-type=D21 --channels-
type=MEM_ACC_NN --memory-allocation-
policy=EXT_PIPELINED_BRAM --channels-number=4
--clock-period=15 --simulate -v3
```

Look how long it take the simulation.

```
$ bambu mips.c --bram-high-latency=4 --channels-
type=MEM_ACC_NN --clock-period=15 --simulate -v3
```

Look how long it take the simulation.

Fourth example – Control asynchronous memories inference


```
--do-not-use-asynchronous-memories
        Do not add asynchronous memories to the possible set
        of memories used by bambu during the memory allocation
        step.

--distram-threshold=value
        Define the threshold in bitsize used to infer
        DISTRIBUTED/ASYNCHRONOUS RAMs (default 256).
```

Solution Hint


```
$ bambu mips.c --do-not-use-asynchronous-memories -fwhole-program --clock-period=15 --simulate -v3
```

Look how long it take the simulation.

```
$ bambu mips.c --distram-threshold=1024 -fwhole-program --clock-period=15 --simulate -v3
```

Look how long it take the simulation.

Fifth – Alignment hints


```
--unaligned-access
        Use only memories supporting unaligned accesses.

--aligned-access
        Assume that all accesses are aligned and so only
        memories supporting aligned accesses are used.
```

Solution Hint


```
$ bambu mips.c --unaligned-access -fwhole-program --clock-period=15 --simulate -v3
```

Look how long it take the simulation.

```
$ bambu mips.c --aligned-access -fwhole-program --clock-period=15 --simulate -v3
```

Look how long it take the simulation.

Sixth example – customize memory layout


```
--base-address=address
     Define the starting address for objects allocated externally to the top
     module.

--initial-internal-address=address
     Define the starting address for the objects allocated internally to the
     top module.
```

Solution Hint


```
$ bambu mips.c --base-address=1024 --memory-allocation-
policy=LSS --clock-period=15 --simulate -v3
```

Look how long it take the simulation.

```
$ bambu mips.c --initial-internal-address=0 --base-
address=1024 --memory-allocation-policy=LSS --clock-
period=15 --simulate -v3
```

Look how long it take the simulation.

Other options


```
--sparse-memory[=on/off]
        Control how the memory allocation happens.
            on - allocate the data in addresses which reduce the decoding logic (default)
           off - allocate the data in a contiguous addresses.

--serialize-memory-accesses
        Serialize the memory accesses using the GCC virtual use-def chains
        without taking into account any alias analysis information.

--do-not-chain-memories
        When enabled LOADs and STOREs will not be chained with other
        operations.

--rom-duplication
        Assume that read-only memories can be duplicated in case timing requires.


--do-not-expose-globals
        All global variables are considered local to the compilation units.

--data-bus-bitsize=<bitsize>
        Set the bitsize of the external data bus.

--addr-bus-bitsize=<bitsize>
        Set the bitsize of the external address bus.
```