

# nanoSoC Configuration Manual

SoC Labs

December 16, 2024

**Additional information.**

You must run 'source set\_env.sh' from the accelerator-project directory every time you open a new terminal!

Preamble, copyrights licenses etc.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Summary . . . . .	3
1.2	Repository Structure . . . . .	3
1.3	IP . . . . .	4
1.3.1	Arm(R) . . . . .	4
1.3.2	SoCLabs . . . . .	4
1.3.3	Synopsys . . . . .	4
1.4	Environment Variables . . . . .	5
<b>2</b>	<b>Adding your IP</b>	<b>6</b>
2.1	Integrating your IP . . . . .	6
2.2	Configuring nanoSoC . . . . .	7
2.2.1	DMA config . . . . .	7
2.3	Mixed signal IP . . . . .	7
2.4	Synopsys IP . . . . .	8
<b>3</b>	<b>Simulating nanoSoC</b>	<b>9</b>
3.1	Supported Simulators . . . . .	9
3.2	Running Simulations . . . . .	9
3.3	Adding Tests . . . . .	9
3.3.1	Creating your C code . . . . .	10
3.3.2	Preloading expansion memories . . . . .	11
<b>4</b>	<b>FPGA Flow</b>	<b>12</b>
4.1	Summary . . . . .	12
4.2	Building the FPGA image . . . . .	12
4.3	Running test code on the FPGA . . . . .	12
<b>5</b>	<b>ASIC Flow</b>	<b>13</b>

# Chapter 1

## Introduction

### 1.1 Summary

### 1.2 Repository Structure

The nanosoc-tech repository should be included as part of the accelerator-project repository. To use nanosoc please make sure you have cloned the accelerator-project repository as this contains IP that nanosoc needs in order to run correctly.

You can clone the accelerator-project using the command:

```
git clone --recurse-submodules \
https://git.soton.ac.uk/soclabs/accelerator-project.git
```

Below is the repository structure showing all the dependancies from the top level accelerator-project.

- accelerator-project (top level)
  - accelerator-wrapper-tech
  - asic-lib-tech
  - fpga-lib-tech
  - generic-lib-tech
  - nanosoc-tech
    - \* sl-ams-tech
    - \* slcorem0-tech
    - \* sldma230-tech
    - \* sldm350-tech
    - \* socdebug-tech
    - \* synopsys-28nm-slm-integration
  - rtl-primitives-tech
  - soctools-flow

## 1.3 IP

The nanoSoC reusable SoC platform relies on IP's from certain vendors. In order to build the system you will need the following IP.

### 1.3.1 Arm(R)

Arm IP should be downloaded from Arm and placed into the recommended IP directories.

- Cortex-M0
- Corstone-101
- (Optional) PL230
- (Optional) DMA-350

### 1.3.2 SoCLabs

SoCLabs IP is all open domain and the repositories for these are automatically cloned into the nanosoc tech directories when you run a git clone.

- ASCII Debug Protocol Controller (SoCDebugTech Git)
- FT1248 Controller (SoCDebugTech Git)
- EXTIO Controller ( EXTIO8x4-axis Git)
- (Optional) PLL (for TSMC 65nm)
- (Optional) ADC (for TSMC 65nm)
- (Optional) DAC (for TSMC 65nm)

### 1.3.3 Synopsys

Synopsys IP is only used for chips that are taped out on a TSMC 28nm HPC+ node.

- (Optional) 3 GHz PLL
- (Optional) Silicon Lifetime Management

Additional information.

You must run 'source set\_env.sh' from the accelerator-project directory every time you open a new terminal!

## 1.4 Environment Variables

When you run 'source set\_env.sh' it sets up environment variables that are used by the tools (simulators, fpga, synthesis etc.). The main ones that are important for developing your IP are:

\$SOCLABS\_PROJECT\_DIR - points to the accelerator-project directory

\$ACCELERATOR\_DIR - points to your accelerator directory

\$ACCELERATOR\_DIR can be set up in the accelerator-project/env/dependency\_env.sh file using:

```
export ACCELERATOR\_DIR="$SOCLABS\_PROJECT\_DIR/**Your-IP-directory**"
```

## Chapter 2

# Adding your IP

In order to add the files for your IP there are 2 options. Either as a local version of your project or as a remote version.

1. Local version: you can just place your IP files into a new directory or in the system/src directory
2. Remote version: you can fork the accelerator-project git to your own git account, this allows you to add your IP either in the system/src directory or as a git submodule.

### 2.1 Integrating your IP

There are 2 steps to integrating your IP in nanosoc

1. Include your IP in the file list
2. Instantiate your IP in the system/src/accelerator\_subsystem.v file

For step 1. you can add paths to your files in the flist/project/accelerator.flist file. We recommend that you use the environment variables as mentioned in section 1.4. In order for fpga and asic flows to work properly you should split verilog and system verilog files into separate .flist files. We suggest adding an accelerator\_sv.flist to the accelerator-project/flist/project directory and adding the following to accelerator.flist

```
-f $SOCLABS_PROJECT_DIR/flist/project/accelerator_sv.flist
```

For step 2. you need to edit the accelerator\_subsystem.v file (found in accelerator-project/system/src/). The ports of this file are an AHB-lite port, 2x EXP\_DRQ (data request from accelerator to DMA), 2x EXP\_DLAST (last signal from DMA to accelerator), 4x EXP\_IRQ (Interrupts from accelerator to CPU), and some AXI stream interfaces (these are only there if the DMA350 is configured with stream interfaces)

#### Additional information.

Add the option `ACCELERATOR=yes` to include your accelerator when you run make commands!

## 2.2 Configuring nanoSoC

The nanoSoC reference design allows for some configuration flexibility. Most of these configuration options are in the `accelerator-project/nanosoc.config` file. In order to change this configuration, put a 'yes' next to the relevant options to include these options, otherwise leave it blank.

### 2.2.1 DMA config

The DMA options are fundamentally:

- 1x PL230
- 2x PL230
- 1x DMA350

More details on these DMA IP's are available from the Arm website.

The DMA-350 also has some extra configuration options

- `DMA350_SMALL` - Small configuration of DMA, 2 channels, no stream interface, no extended features
- `DMA350_DEFAULT` - Default configuration of DMA, 2 channels, stream interface, extended features
- `DMA350_BIG` - Big configuration of DMA, 3 channels, stream interface, extended features

If you use either the `SMALL` or `BIG` options for this, you must reconfigure the DMA-350. Follow the below steps:

1. `cd` to `accelerator-project/nanosoc_tech/nanosoc/sldma350_tech`
2. run `'make clean_ip'`
3. run `'make config_dma_ahb_small'` or `'make config_dma_ahb_big'`

## 2.3 Mixed signal IP

! Still under development !

You can also include mixed signal IP in this design. In order to do this you must also have the relevant IP for this



## 2.4 Synopsys IP

! Still under development !

If you are taping out with a TSMC 28nm node, and also have access to the Synopsys IP mentioned in section 1.3.3

## Chapter 3

# Simulating nanoSoC

### 3.1 Supported Simulators

- Mentor Graphics: QuestaSim
- Synopsys: VCS
- Cadence: Xcelium
- Icarus Verilog

### 3.2 Running Simulations

You can run make commands from the nanosoc\_tech directory to run the simulation.

```
make run SIM=x TESTNAME=y ACCELERATOR=yes
```

Where x=mti, vcs, xm, or iverilog for QuestaSim, VCS, Xcelium, or Icarus Verilog respectively. And y is the name of the test, the default test is hello (a hello world example).

Or to run the simulation in the GUI you can use:

```
make sim SIM=x TESTNAME=y ACCELERATOR=yes
```

Whilst the simulation is running, you should see the output from the UART std out channel in the console/terminal.

### 3.3 Adding Tests

To add your own testcodes to run on nanosoc in the simulation environment, you can add these to the accelerator-project/system/testcodes directory.

1. Create a new directory for your testcode

2. Create a .c source file with the same name as the directory
3. Copy the makefile from one of the example testcodes to your test code directory
4. Edit the TESTNAME variable in the new makefile to the name of your test
5. If you want to run any ADP code before your test, add an adp.cmp file (example in the adp\_v4\_cmd\_tests)
6. If you want to preload expansion memories, add an expam.l.hex and/or expam.h.hex
7. Add the name of your test to the accelerator-project/system/software.list.txt file

### 3.3.1 Creating your C code

The below code is a basic layout for your C code. It initialises the standard out channel over UART. You can then use printf statements that will output over UART, which will be printed on the console output.

The UartEndSimulation() function sends an escape character over UART, which the testbench will use to end the simulation.

Listing 3.1: Basic Template

```
#include "CMSDK_CM0.h"
#include "uart_stdout.h"
#include <stdio.h>

int main(void) {
    // Initialise the UART standard out channel
    UartStdOutInit();

    printf("Foo\n"); // Print over UART stdout

    /* Insert your code here */

    // End simulation by sending escape char
    UartEndSimulation();
    return 0;
}
```

For more detailed C code templates, please see the firmware in the accelerator-project/nanosoc\_tech/software/common/validation. These are also the testcodes used for validating nanoSoC.

### 3.3.2 Preloading expansion memories

You may want to load test vectors directly into the expansion memories to run your tests. Doing this can save space in the instruction memory space as you then don't have to load data in as arrays or vectors in your testcode. Instead you can use the simulator to automatically load these memories at the start of simulation.

To do this, simply add an "expram\_l.hex" file and/or "expram\_h.hex" file to your testcode directory. These files will then be loaded to the EXPRAM\_L or EXPRAM\_H region respectively. These can then be addressed in your testcode from 0x80000000 for EXPRAM\_L and 0x90000000 for EXPRAM\_H.

The expram\_l.hex files must be ASCII text files with a single byte per line. They will look very similar to the .hex files that are used to preload the instruction memory.

## Chapter 4

# FPGA Flow

### 4.1 Summary

### 4.2 Building the FPGA image

To build the FPGA image, run the below command from the nanosoc.tech directory:

```
make build_fpga ACCELERATOR=yes FPGA=x
```

Where x is either zcu104, mps3, kr260, kv260, z2. If you would like another FPGA target to be included please contact the soclabs team or raise an issue on the accelerator-project git.

### 4.3 Running test code on the FPGA

## Chapter 5

# ASIC Flow