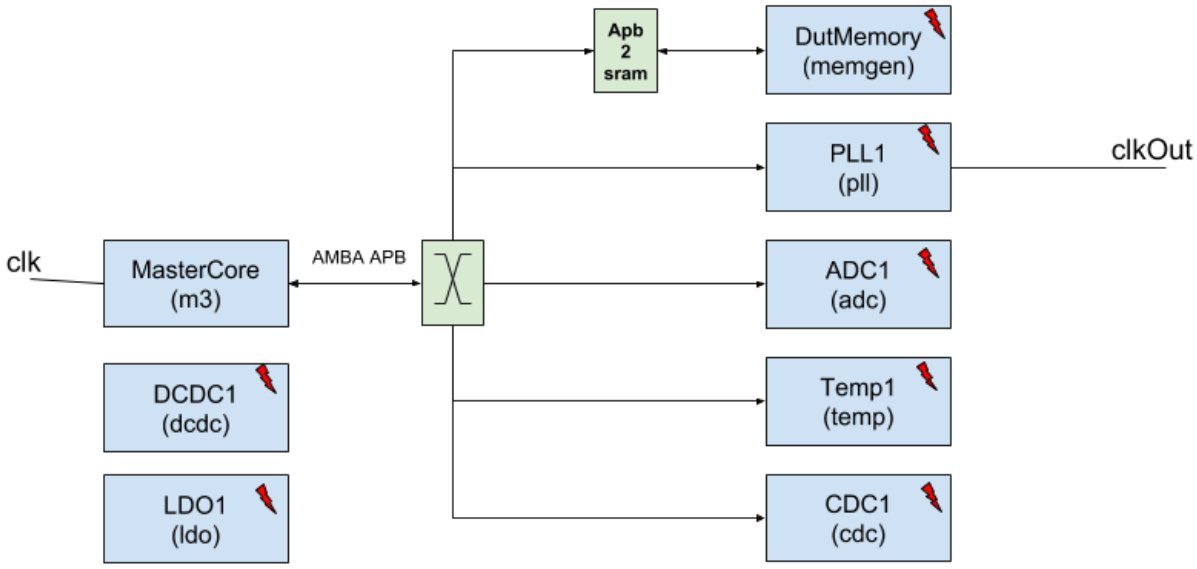
FASoC Walkthrough

https://fasoc.engin.umich.edu/

This purpose of this document is to establish the interfaces between the various components comprising the FASoC project and a guide to the process of generating and SoC

1. Initial Target Design

The selected target is a simple design that represents an SoC that using all of the various analog blocks proposed as part of the FASoC project.



2. High level user intent/specification

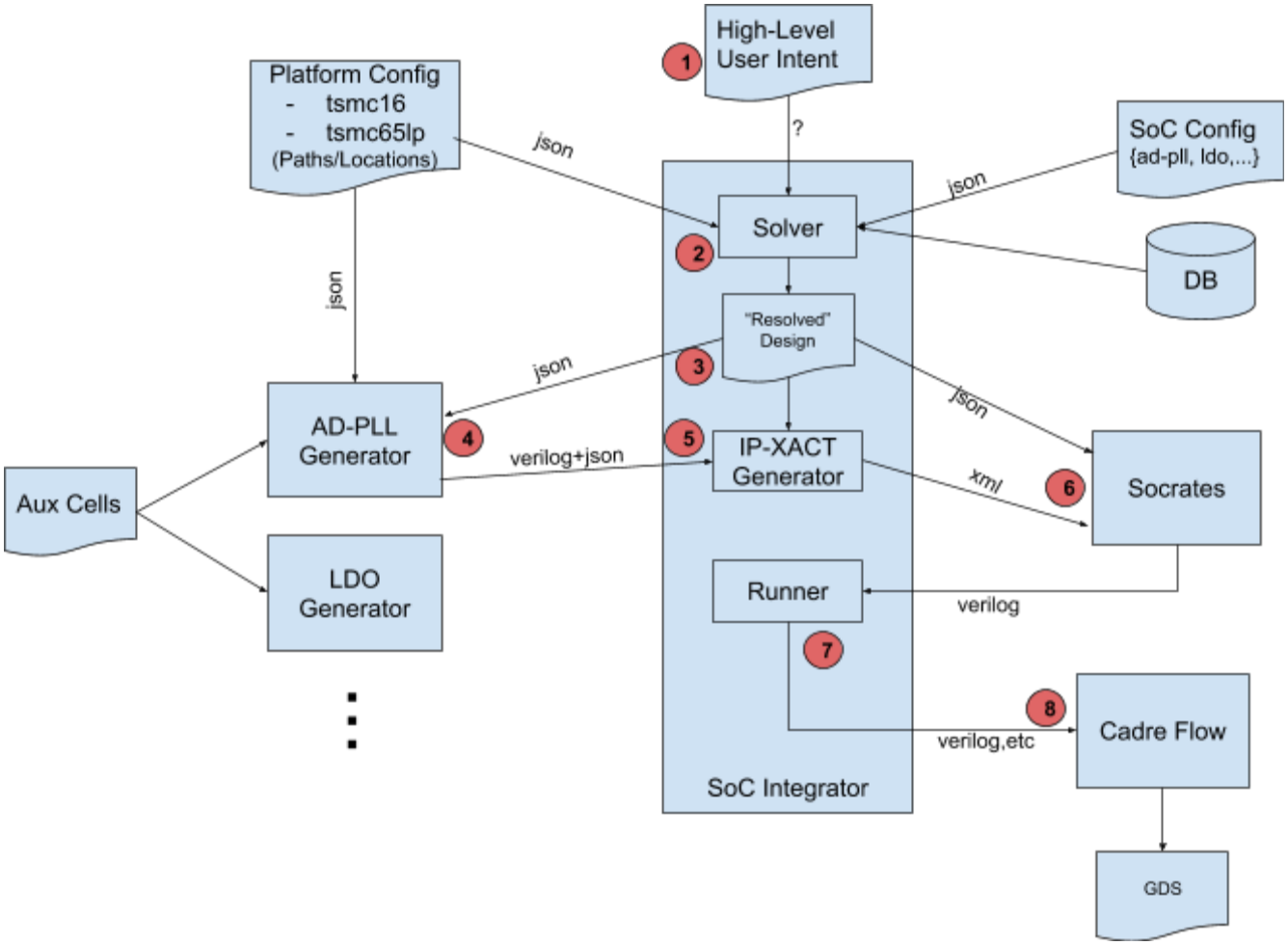
A key goal of the program is to have the user specify a “high-level” design intent, rather than explicit configuration and specifications for each analog block to be generated. For the target design above, the use will at least have to specify the following information:

- All of the major blocks (In blue). Detail specification/parameters not required
- At least the following connections:
    - Apb2Sram to DutMemory
- The input and output toplevel clocks

The crossbars and connections between the blocks can be resolved by the tools. More on this in section 4

3. SoC Integrator - Initialization

Initialize the SoC integrator with database and information about the different generators.



SoC Config (See FASoC_config.json)

This file describes all of the different generators available to the solver. It also describes their ports and interfaces for IP-XACT generation

FASoC_config.json

```
{
  "schema_version": 0.1,
  "generators": [
    {
      "type": "AD-PLL",
      "path": "./generators/ad-pll",
      "ports": [
        {
          "name": "clk",
          "direction": "in",
          "type": "clock"
        },
        {
          "name": "reset",
          "direction": "in",
          "type": "reset"
        },
        {
          "name": "clockOut",
          "direction": "out",
          "type": "clock"
        }
      ],
      "interfaces": [
        {
          "name": "config",
          "type": "apb-slave",
          "datawidth": 32,
          "addrwidth": 4
        }
      ]
    },
    {
      "type": "ADC",
      "path": "./generators/adc",
      "ports": [
        {
          "name": "clk",
          "direction": "in",
          "type": "clock"
        },
        {
          "name": "reset",
          "direction": "in",
          "type": "reset"
        }
      ],
      "interfaces": [
        {
```

Sample platform config (see platform_config.json)

This file describes the information about the processes and is intended to be used by the generators. It will have links to the various aux cells and information require paths to PDK/Cell information. This file will need to site/university specific.

```
{
  "schema_version": 0.1,
  "platforms": [
    {
      "name": "tsmc16",
      "nominal_voltage": 0.8,
      "aux_cells": {
        "AD-PLL": "/net/tretion/ad_aux.gds",
        "LDO": "/net/tretion/ldo.gds",
        "Temp": "/net/tretion/temp.gds"
      }
    }
  ]
}
```

Database

The format for the database is still to be decided. What's in the DB? Everything required to pass to socrates

- Power / Performance / Area
- Specifications
- Output files (see section 5)

4. SoC Integrator - Solver

The solver will need to perform the following tasks

1. Understand high-level user specifications: User specification can be in the same format as the resolved design (see below) but without the details (e.g. specifications, connections, crossbars, etc). Part of the user specification will also include an optimization directive (e.g area, power, runtime, noise, etc)
2. The solver will select modules and required specifications to meet the design requirements
3. The solver will have additional utilities and modules (e.g crossbars, interface width adapters, resets circuits, etc)

4. The solver will generate a “resolved” design with all of the specification details and connections. This format will capture all of the information about the design in detail. See resolved_design.json

resolved_design.json


```
{
  "schema_version": 0.1,
  "design_name": "Demo SoC",
  "tech": "tsmc65lp",
  "strategy":"power",
  "units": {
    "frequency": "hz",
    "voltage": "v",
    "current": "a",
    "length": "um"
  },
  "modules": [
    {
      "instance_name": "MasterCore",
      "generator": "socrates",
      "specifications": {
        "vendor": "arm.com",
        "library": "Cortex-M3",
        "name": "CORTEXM3",
        "version": "r2p1_1"
      },
      "parameters": {
        "MPU_PRESENT": true,
        "NUM_IRQ": 1
      }
    },
    {
      "instance_name": "DutCore",
      "generator": "socrates",
      "specifications": {
        "vendor": "arm.com",
        "library": "Cortex-M0",
        "name": "CORTEXM0",
        "version": "r2p1_1"
      },
      "parameters": {
        "MPU_PRESENT": true,
        "NUM_IRQ": 1
      }
    },
    {
      "instance_name": "CDC1",
      "generator": "CDC",
      "specifications": {
        "frequency": {
          "min": 2500000000,
          "max": 1300000000
        },
        "resolution": 8
      }
    },
    {
      "instance_name": "ADC1",
      "generator": "ADC",
      "specifications": {
        "frequency": {
          "min": 2500000000,
          "max": 1300000000
        },
        "resolution": 8
      }
    }
```

Perform high-level connections of clocks, resets and interfaces between blocks

resolved_design.json


```
1   {
2     "schema_version": 0.1,
3     "design_name": "Demo SoC",
4     "tech": "tsmc65lp",
5     "strategy":"power",
6     "units": {
7       "frequency": "hz",
8       "voltage": "v",
9       "current": "a",
10      "length": "um"
11    },
12    "modules": [ ...
245   ],
246   "connections": [
247     {
248       "type": "clock",
249       "from": {
250         "module": "toplevel",
251         "port": "clk"
252       },
253       "to": {
254         "module": "MasterCore",
255         "port": "clk"
256       },
257       "to": {
258         "module": "PLL1",
259         "port": "clk"
260       }
261     },
262     {
263       "type": "apb",
264       "from": {
265         "module": "MasterCore",
266         "port": "peripheral"
267       },
268       "to": {
269         "module": "CrossBar1",
270         "port": "MASTER_PORT_0"
271       }
272     },
273     {
274       "type": "apb",
275       "from": {
276         "module": "CrossBar1",
277         "port": "SLAVE_PORT_0"
278       },
279       "to": {
280         "module": "CDCD1",
281         "port": "config"
282       }
283     },
284     {
285       "type": "apb",
286       "from": {
287         "module": "CrossBar1",
288         "port": "SLAVE_PORT_1"
289       },
```

5. Check for unconnected blocks, generate warnings, etc. Can also be easily graphed using graphiz
6. Make calls to the analog block generators

5. Analog Block Generation

If the blocks are not found in the database, SoC integrator calls each analog block with a specification file as derived from the solver. The call to the solver will be in the following command line format:

```
{path_to_generator_in_cfg} -config pll_config.json -output
./output/dir -platform tsmc16
```

Where the specification file looks like (see generator_input.json)

generator_input.json

```
{
  "instance_name": "PLL1",
  "generator": "AD-PLL",
  "specifications": {
    "frequency": {
      "min": 2500000000,
      "max": 1300000000
    },
    "spot_phase_noise": 65,
    "freq_resolution": 1050000,
    "INL": 1
  }
}
```

The Analog Block outputs the following files in the output directory

```
test2.v
test2.result.json

# Option 1 (This is currently required for tapeout)
test2.gds
test2.lib
test2.db*
test2.lef
test2.cdl*

# Option 2 (In future, this will be the only requirement
#           once the aux cell generation is automated
test2.idf
```

The result.json contains information that has been achieved after characterization (see generator_output.json)

generator_result.json

```
{
  "instance_name": "PLL1",
  "generator": "AD-PLL",
  "specifications": {
    "frequency": {
      "min": 2500000000,
      "max": 1300000000
    },
    "spot_phase_noise": 65,
    "freq_resolution": 1050000,
    "INL": 1
  },
  "generator_results": {
    "platform": "tsmc65lp",
    "power": 10,
    "performance": 2400000000,
    "area": 2000,
    "frequency": {
      "min": 1100000000,
      "max": 2400000000
    },
    "spot_phase_noise": 62,
    "freq_resolution": 10500,
    "INL": 1
  }
}
```

The result is added to the database. Error if it doesn't meet specs

6. IP-XACT++ XML Generation

Generate IP-XACT++ xml for all blocks
Generate connection information/script for IP-XACT

7. Socrates++ - Generate Synth Files

The SoC Integrator will pass the following to socrates

- The resolved_design.json
- IP-XACT for individual blocks (except for ARM IP)

Socrates will generate

- Output verilog
- IP-XACT of complete SoC
- Potentially a picture of the SoC

- Reports and other collateral (power and area). Still in the works.

8. Cadre Flow - Synth + APR

Inputs will be the verilog from socrates, all of the different block folders. Output will be gds