Overview

The VSDBabySoC is a simple SoC (System-on-Chip) design incorporating a RISC-V processor (rvmyth), a PLL (Phase-Locked Loop) module (pll), and a DAC (Digital-to-Analog Converter) module (dac). This project demonstrates integration of these IP cores and aims to simulate and verify the design behavior using pre-synthesis and post-synthesis simulations.

VSDBabySoC Modeling

Here we are going to model and simulate the VSDBabySoC using iverilog, then we will show the results using gtkwave tool. Some initial input signals will be fed into vsdbabysoc module that make the pll start generating the proper CLK for the circuit. The clock signal will make the rvmyth to execute instructions in its imem. As a result the register r17 will be filled with some values cycle by cycle. These values are used by dac core to provide the final output signal named OUT. So we have 3 main elements (IP cores) and a wrapper as an SoC and of-course there would be also a testbench module out there.

Project Structure

- src/include/ - Contains header files (*.vh) with necessary macros or parameter definitions.
- src/module/ - Contains Verilog files for each module in the SoC design.
- output/ - Directory where compiled outputs and simulation files will be generated.

```
VSDBabySoC/
 ├── src/
 │   ├── include/
 │   │   ├── sandpiper.vh
 │   │   └── other header files...
 │   ├── module/
 │   │   ├── vsdbabysoc.v     # Top-level module integrating all components
 │   │   ├── rvmyth.v         # RISC-V core module
 │   │   ├── avsdpll.v        # PLL module
 │   │   ├── avsddac.v        # DAC module
 │   │   └── testbench.v      # Testbench for simulation
 └── output/
 └── compiled_tlv/       # Holds compiled intermediate files if needed
```

RVMYTH modeling

As we mentioned in What is RVMYTH section, RVMYTH is designed and created by the TL-Verilog language. So we need a way for compile and trasform it to the Verilog language and use the result in our SoC. Here the sandpiper-saas could help us do the job.

Here is the repo we used as a reference to model the RVMYTH

PLL and DAC modeling

It is not possible to sythesis an analog design with Verilog, yet. But there is a chance to simulate it using real datatype. We will use the following repositories to model the PLL and DAC cores:

1. Here is the repo we used as a reference to model the PLL
2. Here is the repo we used as a reference to model the DAC

CAUTION: In the beginning of the project, we get our verilog model of the PLL from here. However, by proceeding the project to the physical design flow we realize that this model needs a little changes to become sufficient for a real IP core. So we changed it a little and created a new model named AVSDPLL based on this IP

Step by step modeling walkthrough

In this section we will walk through the whole process of modeling the VSDBabySoC in details. We will increase/decrease the digital output value and feed it to the DAC model so we can watch the changes on the SoC output. Please, note that the following commands are tested on the Ubuntu Bionic (18.04.5) platform and no other OSes.

1. First we need to install some important packages:

```
$ sudo apt install make python python3 python3-pip git iverilog gtkwave docker.io

$ sudo chmod 666 /var/run/docker.sock

$ cd ~

$ pip3 install pyyaml click sandpiper-saas
```

2. Now we can clone this repository in an arbitrary directory (we'll choose home directory here):

```
$ cd ~

$ git clone https://github.com/manili/VSDBabySoC.git
```

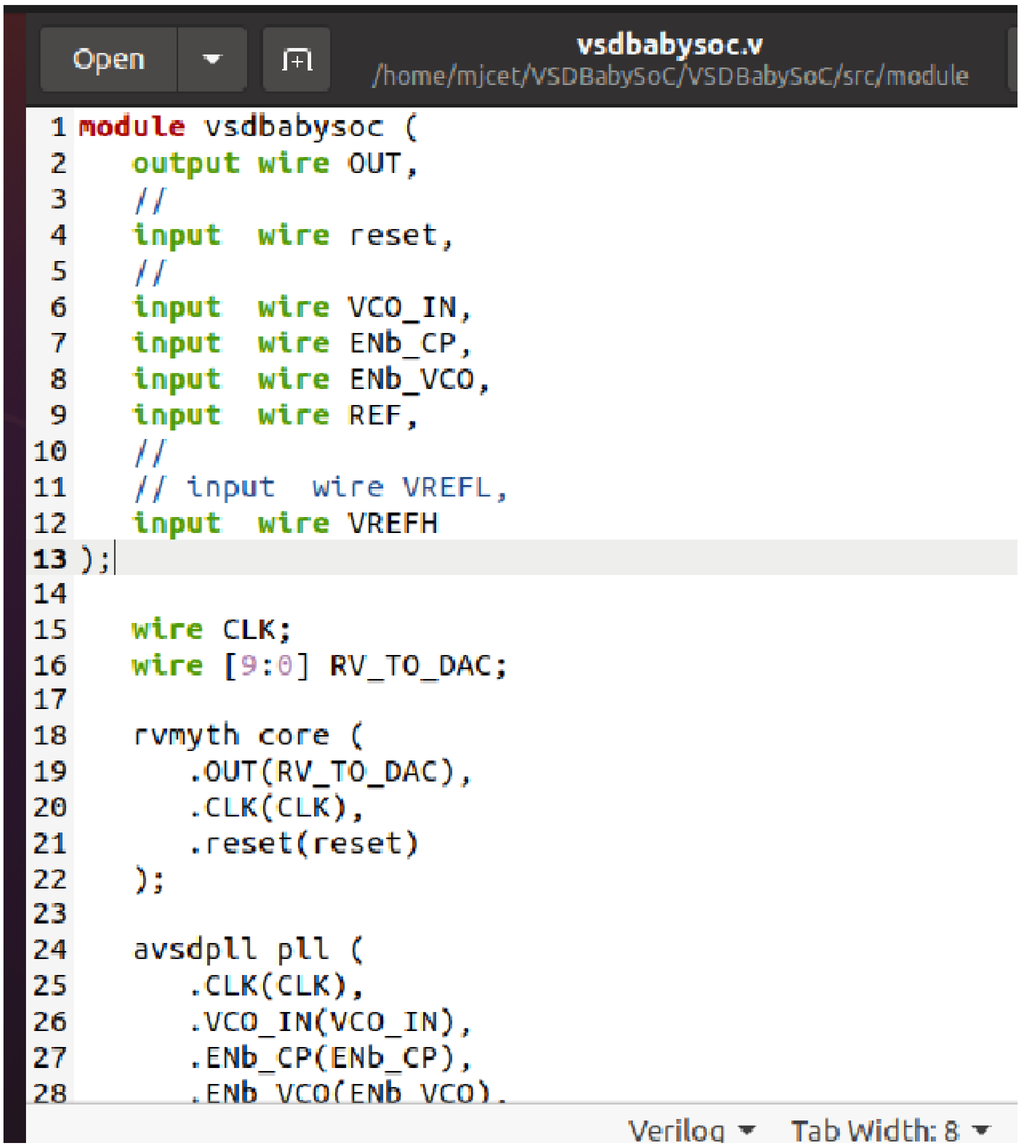
3. It's time to make the pre_synth_sim.vcd:

```
$ cd VSDBabySoC
$ make pre_synth_sim
```

The result of the simulation (i.e. pre_synth_sim.vcd) will be stored in the output/pre_synth_sim directory.

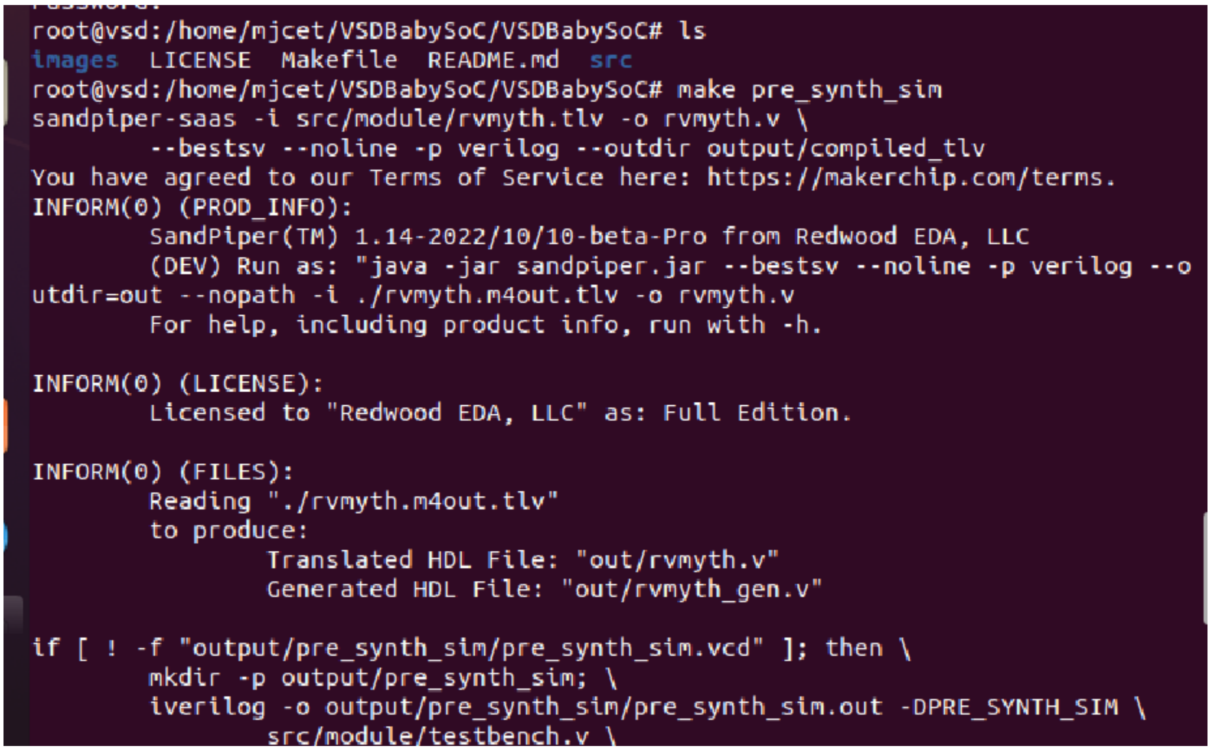
VSDBabySoC.v

```
cd VSDBabySoC
sandpiper-saas -i ./src/module/*.tlv -o rvmyth.v --bestsv --noline -p verilog --outdir ./src/module/
```



```
module vsdbabysoc (
   output wire OUT,
   //
   input  wire reset,
   //
   input  wire VCO_IN,
   input  wire ENb_CP,
   input  wire ENb_VCO,
   input  wire REF,
   //
   // input  wire VREFL,
   input  wire VREFH
);

   wire CLK;
   wire [9:0] RV_TO_DAC;

   rvmyth core (
      .OUT(RV_TO_DAC),
      .CLK(CLK),
      .reset(reset)
   );

   avsdpll pll (
      .CLK(CLK),
      .VCO_IN(VCO_IN),
      .ENb_CP(ENb_CP),
      .ENb_VCO(ENb_VCO),
```

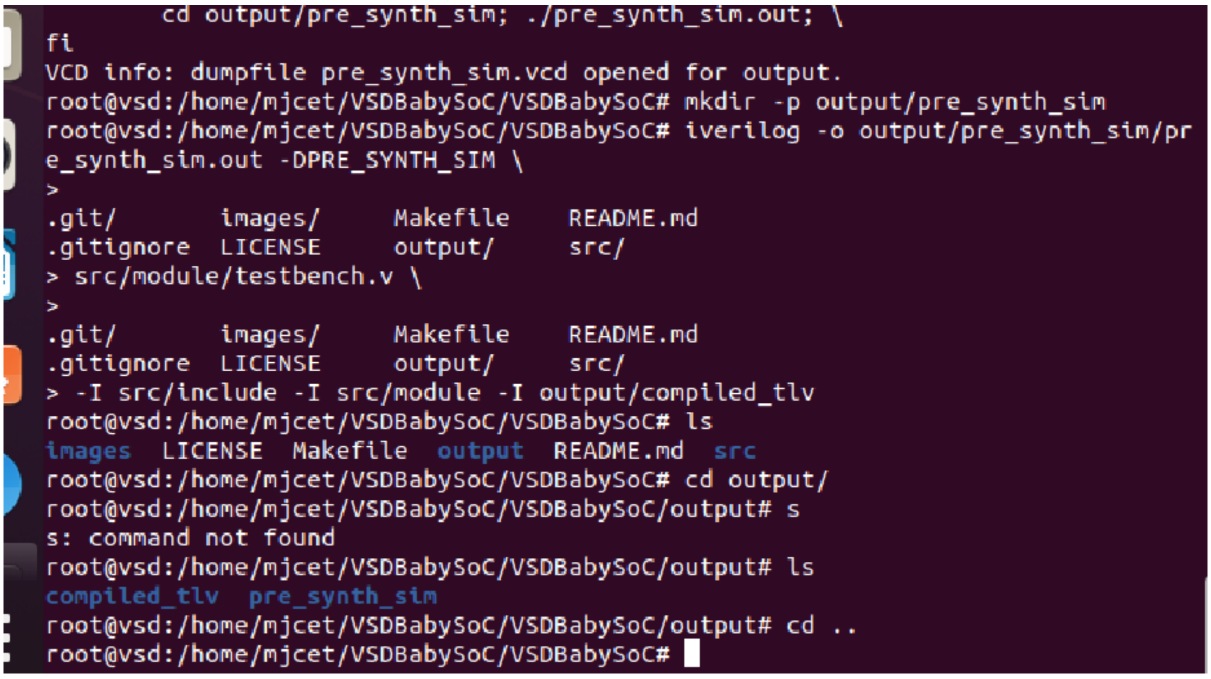
git clone https://github.com/manili/VSDBabySoC.git



```
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# ls
images  LICENSE  Makefile  README.md  src
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# make pre_synth_sim
sandpiper-saas -i src/module/rvmyth.tlv -o rvmyth.v \
        --bestsv --noline -p verilog --outdir output/compiled_tlv
You have agreed to our Terms of Service here: https://makerchip.com/terms.
INFORM(0) (PROD_INFO):
        SandPiper(TM) 1.14-2022/10/10-beta-Pro from Redwood EDA, LLC
        (DEV) Run as: "java -jar sandpiper.jar --bestsv --noline -p verilog --o
utdir=out --nopath -i ./rvmyth.m4out.tlv -o rvmyth.v
        For help, including product info, run with -h.

INFORM(0) (LICENSE):
        Licensed to "Redwood EDA, LLC" as: Full Edition.

INFORM(0) (FILES):
        Reading "./rvmyth.m4out.tlv"
        to produce:
                Translated HDL File: "out/rvmyth.v"
                Generated HDL File: "out/rvmyth_gen.v"

if [ ! -f "output/pre_synth_sim/pre_synth_sim.vcd" ]; then \
        mkdir -p output/pre_synth_sim; \
        iverilog -o output/pre_synth_sim/pre_synth_sim.out -DPRE_SYNTH_SIM \
                src/module/testbench.v \
```

make pre_synth_sim

The result of the simulation (i.e. pre_synth_sim.vcd) will be stored in the output/pre_synth_sim directory.

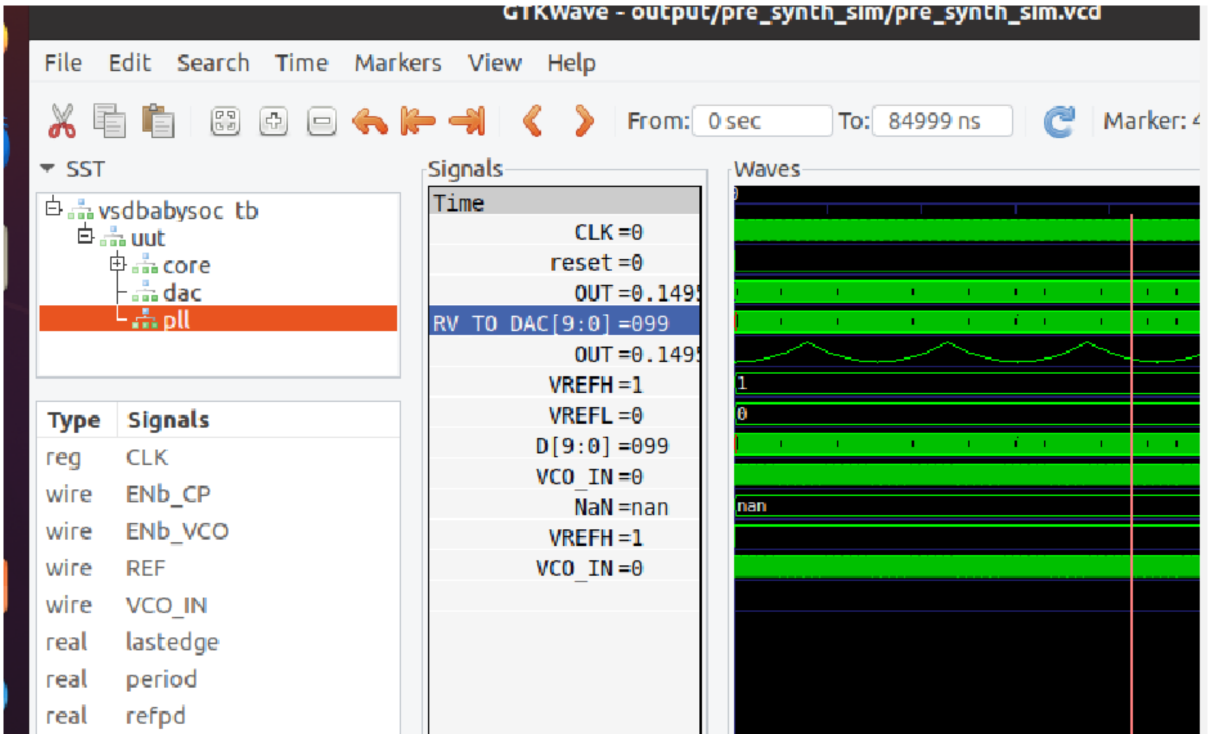


```
        cd output/pre_synth_sim; ./pre_synth_sim.out; \
fi
VCD info: dumpfile pre_synth_sim.vcd opened for output.
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# mkdir -p output/pre_synth_sim
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# iverilog -o output/pre_synth_sim/pr
e_synth_sim.out -DPRE_SYNTH_SIM \
>
.git/         images/     Makefile    README.md
.gitignore    LICENSE     output/     src/
> src/module/testbench.v \
>
.git/         images/     Makefile    README.md
.gitignore    LICENSE     output/     src/
> -I src/include -I src/module -I output/compiled_tlv
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# ls
images  LICENSE  Makefile  output  README.md  src
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# cd output/
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC/output# s
s: command not found
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC/output# ls
compiled_tlv  pre_synth_sim
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC/output# cd ..
root@vsd:/home/mjcet/VSDBabySoC/VSDBabySoC# 
```

We can see the waveforms by following command:

```
$ gtkwave output/pre_synth_sim/pre_synth_sim.vcd
```

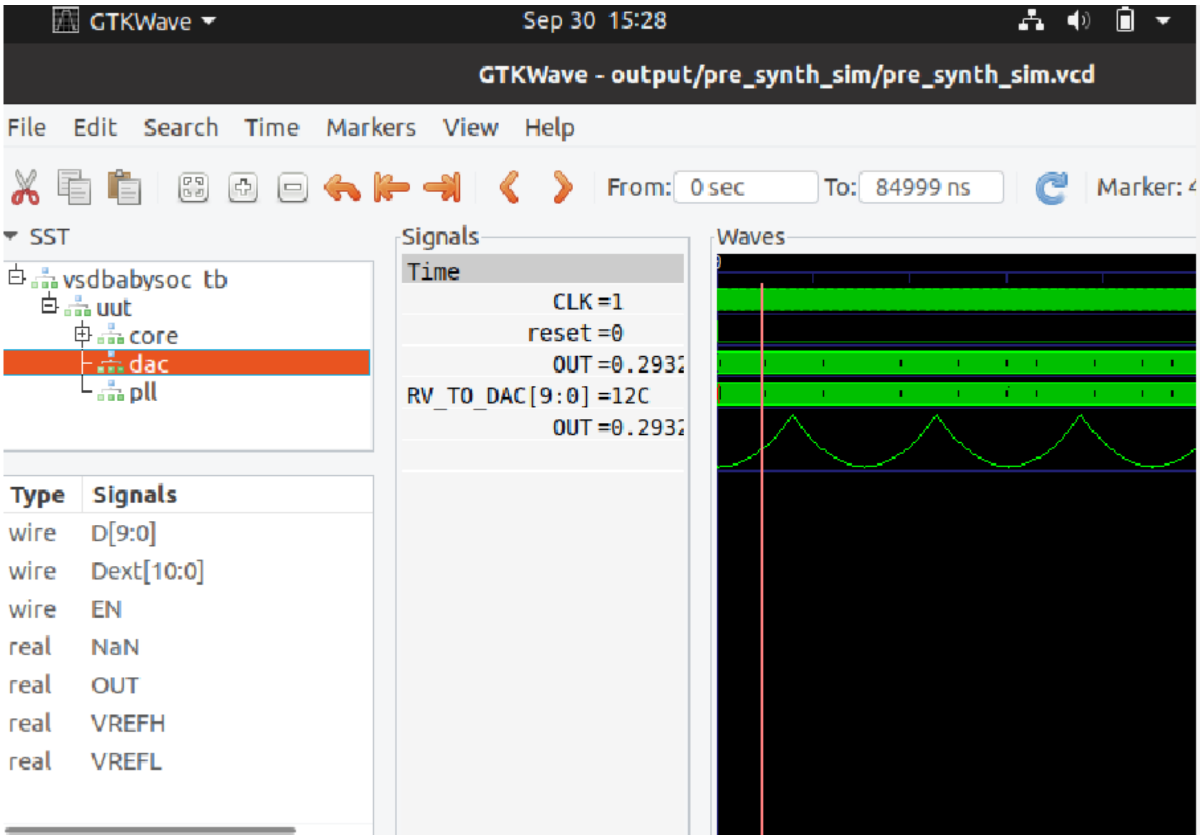
Two most important signals are CLK and OUT. The CLK signal is provided by the PLL and the OUT is the output of the DAC model. Here is the final result of the modeling process:



SST:
- vsdbabysoc tb
  - uut
    - core
    - dac
    - pll

Type	Signals
reg	CLK
wire	ENb_CP
wire	ENb_VCO
wire	REF
wire	VCO_IN
real	lastedge
real	period
real	refpd

```
Time
CLK =0
reset =0
OUT =0.149
RV TO DAC[9:0] =099
OUT =0.149
VREFH =1
VREFL =0
D[9:0] =099
VCO_IN =0
NaN =nan
VREFH =1
VCO_IN =0
```



Type	Signals
wire	D[9:0]
wire	Dext[10:0]
wire	EN
real	NaN
real	OUT
real	VREFH
real	VREFL

```
Time
CLK =1
reset =0
OUT =0.2932
RV_TO_DAC[9:0] =12C
OUT =0.2932
```

In this picture we can see the following signals:

- **CLK:** This is the input CLK signal of the RVMYTH core. This signal comes from the PLL, originally.
- **reset:** This is the input reset signal of the RVMYTH core. This signal comes from an external source, originally.
- **OUT:** This is the output OUT signal of the VSDBabySoC module. This signal comes from the DAC (due to simulation restrictions it behaves like a digital signal which is incorrect), originally.
- **RV_TO_DAC[9:0]:** This is the 10-bit output [9:0] OUT port of the RVMYTH core. This port comes from the RVMYTH register #17, originally.
- **OUT:** This is a real datatype wire which can simulate analog values. It is the output wire real OUT signal of the DAC module. This signal comes from the DAC, originally.