WEEK:2

BabySoC Fundamentals

System on a Chip (SoC) Fundamentals

A **System on a Chip (SoC)** is a complete electronic system integrated onto a single, compact integrated circuit (chip). It acts like a mini-computer, combining all necessary components into one package, which offers significant advantages in terms of **space saving**, **energy efficiency**, **high performance**, and **cost-effectiveness**. SoCs are essential in modern portable electronics like **smartphones**, **smartwatches**, and **IoT devices**.

Key Components of a Typical SoC

A standard SoC integrates several critical functional blocks:

Component	Description
CPU (Central Processing Unit)	The "brain" that executes instructions, handles data processing, and manages core operations.
Memory	Includes **RAM** for temporary data and **ROM/Flash** for permanent storage.
GPU (Graphics Processing Unit)	Handles visuals, responsible for graphics, videos, and animations.
DSP (Digital Signal Processor)	Specialized for processing audio and video signals, used for tasks like noise reduction.
I/O Ports (Input/Output)	Connects the SoC to external components and peripherals (e.g., USB, camera, displays).
Power Management	Regulates power usage across the chip to ensure efficiency and extend battery life.
Interconnect	A bus or network that allows all the components to communicate rapidly.

SoCs can be categorized based on their core, such as **Microcontroller-based** (for low-power control tasks), **Microprocessor-based** (for running operating systems and complex applications), and **Application-Specific** (custom-designed for high-performance, specialized tasks).

The VSDBabySoC Educational Platform

The **VSDBabySoC** project serves as a compact, open-source model designed specifically for learning and experimentation in digital-analog interfacing and SoC concepts. It is implemented using **Sky130 technology** and is based on the **RISC-V** architecture, a simple and customizable open-source instruction set.

BabySoC’s Simplified Learning Model

VSDBabySoC is a simplified yet functional SoC designed to facilitate the simultaneous testing of three core open-source intellectual property (IP) blocks:

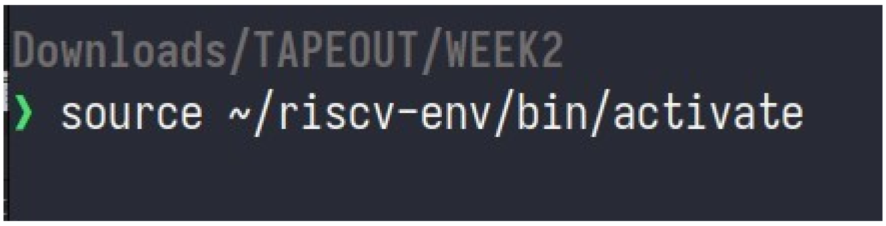
1. **RVMYTH (RISC-V CPU):** This is the processor core, handling data processing. It uses a register (specifically `r17`) to hold and cycle through digital values destined for analog conversion, generating continuous data streams.
2. **Phase-Locked Loop (PLL):** A crucial control system that generates a **stable and synchronized clock signal**. This is vital for coordinating the activities of the RVMYTH processor and the DAC, ensuring reliable timing and data integrity, which is a key concept in hardware design.
3. **10-bit Digital-to-Analog Converter (DAC):** This component is the focus of the digital-analog interfacing lesson. It receives the digital data from RVMYTH and **converts it into an analog signal**. This analog output can then be used to interface with external analog-accepting devices, such as TVs or mobile phones, for multimedia output (e.g., audio or video).

BabySoC in the Design Journey

The creation of BabySoC highlights the foundational stages of chip design:

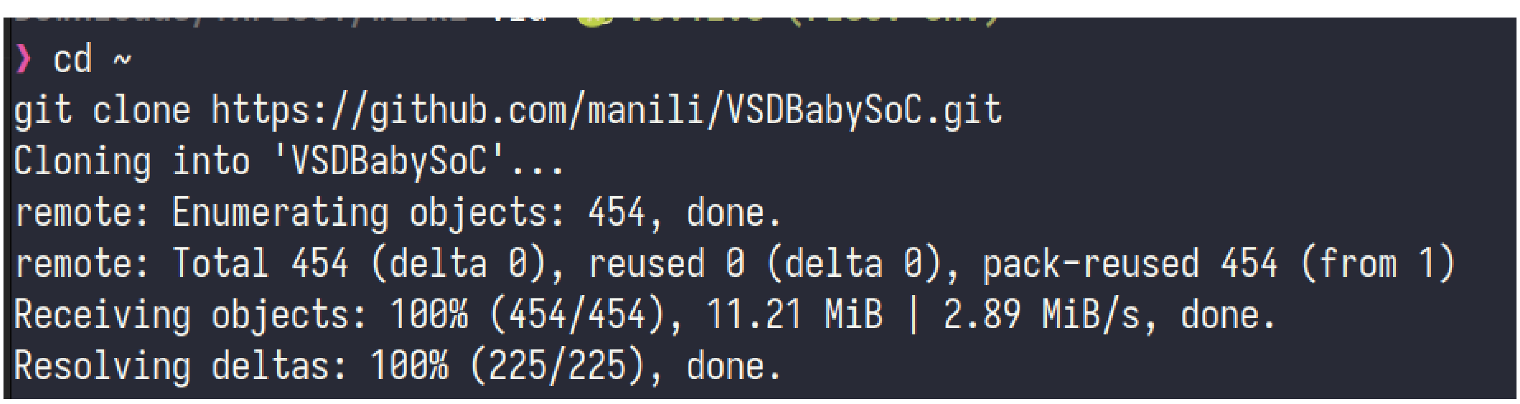
- **Learning Platform:** The project is a highly-documented educational platform focused on the practical application of digital-analog interfacing.
- **Testing Core IPs:** Its primary objective is to test and calibrate its analog components (the 10-bit DAC and PLL) alongside the RVMYTH microprocessor.
- **Functional Modelling:** Before progressing to advanced stages like **RTL (Register-Transfer Level)** and **physical design**, a functional model like BabySoC is essential. It allows for the testing of the core logic and the calibration of analog components, ensuring that the fundamental building blocks and their interactions (like clock synchronization via the PLL and data flow from the CPU to the DAC) perform as intended. By mastering this functional understanding, students gain a solid foundation before tackling the complexity of a full-scale SoC design flow.

ACTIVATE PYTHON VENV



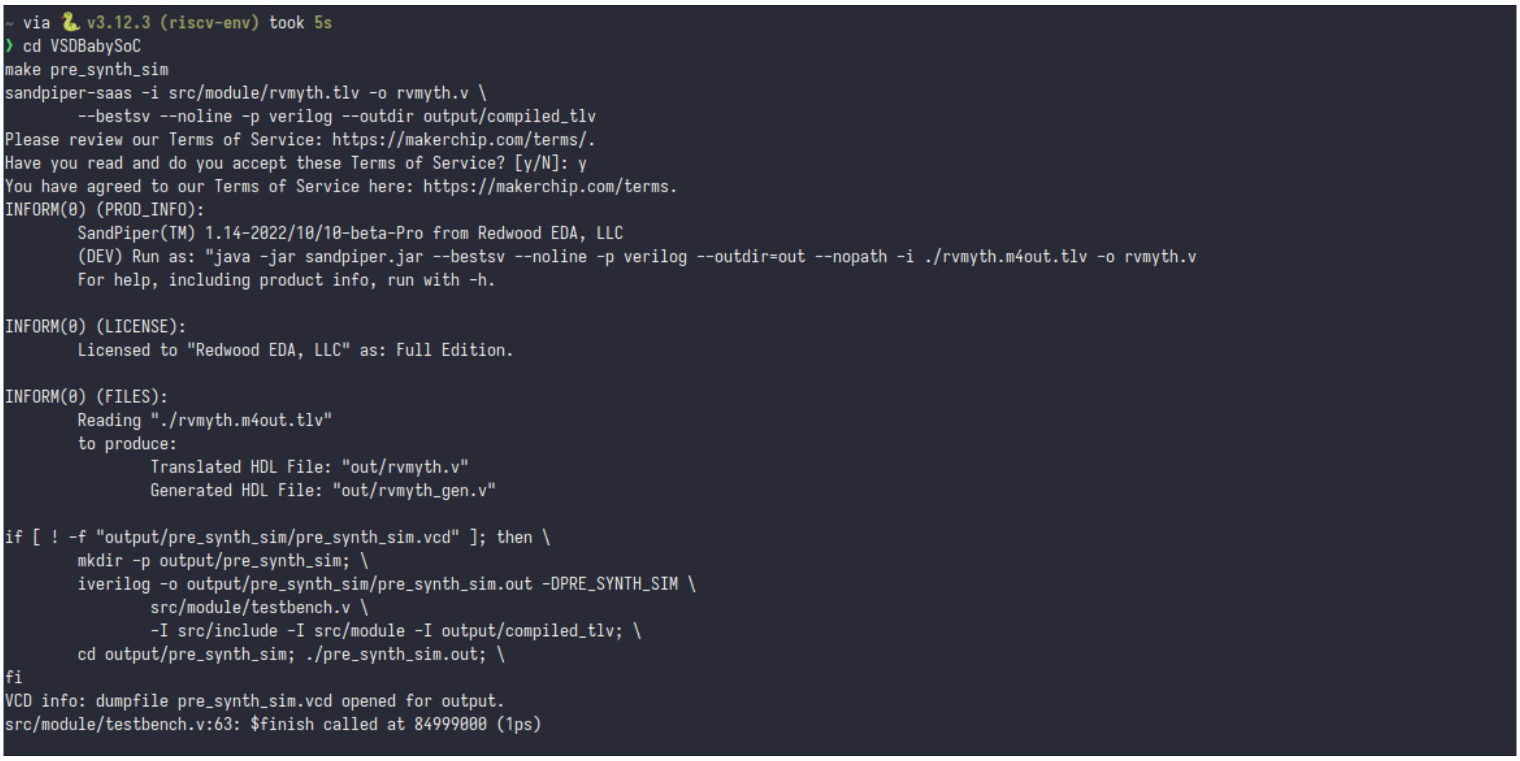
```
Downloads/TAPEOUT/WEEK2
❯ source ~/riscv-env/bin/activate
```

CLONE THE REPOSITORY



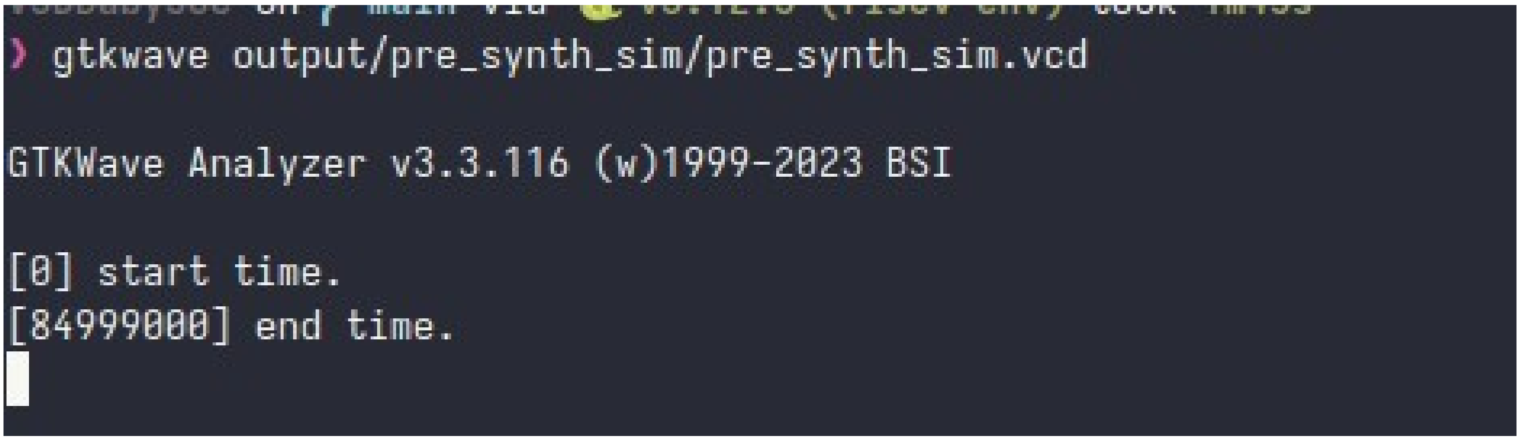
```
❯ cd ~
git clone https://github.com/manili/VSDBabySoC.git
Cloning into 'VSDBabySoC'...
remote: Enumerating objects: 454, done.
remote: Total 454 (delta 0), reused 0 (delta 0), pack-reused 454 (from 1)
Receiving objects: 100% (454/454), 11.21 MiB | 2.89 MiB/s, done.
Resolving deltas: 100% (225/225), done.
```

OPEN THE DOWNLOADED REPO
BUILD AND COMPILE THE FOLDERS



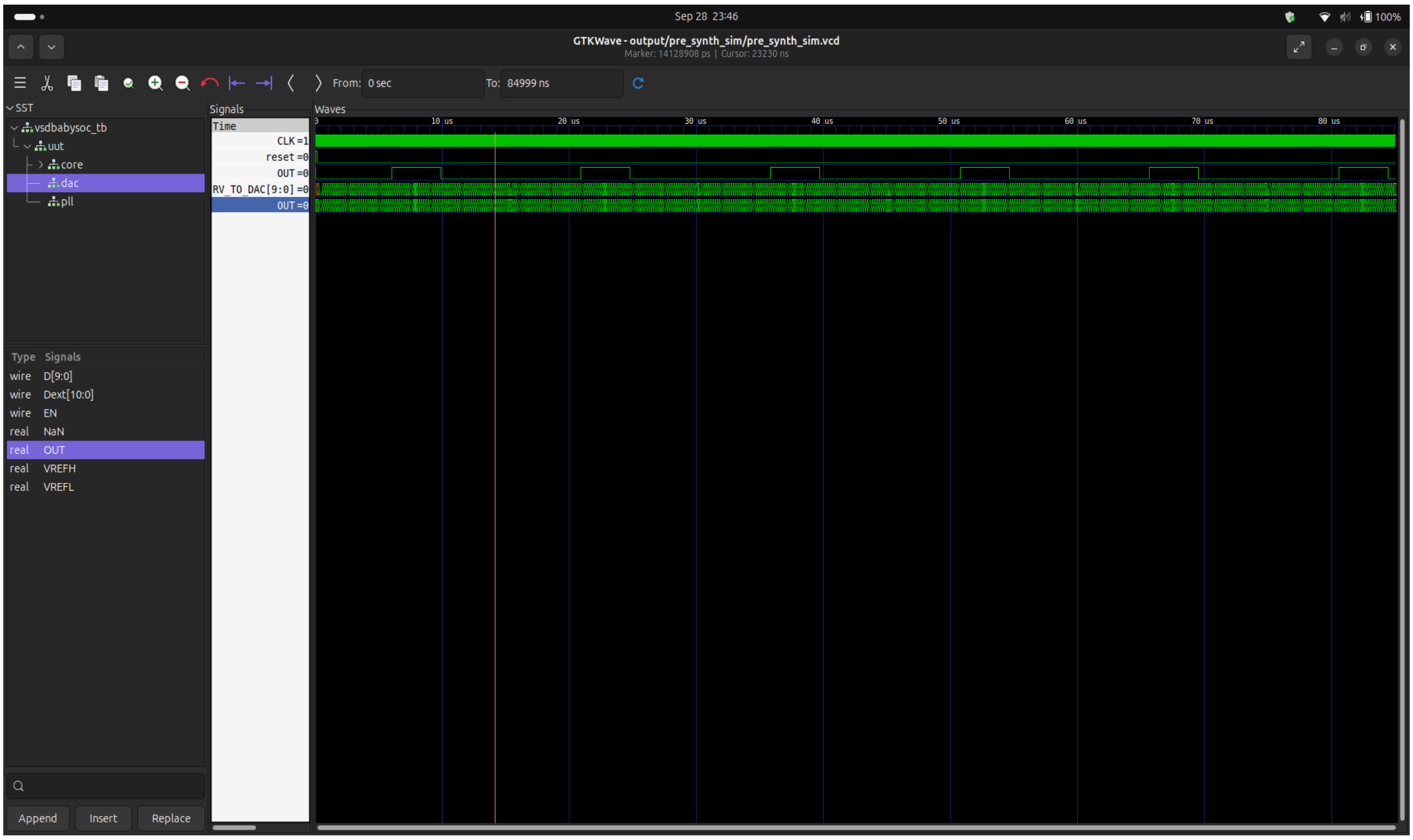
```
~ via 🐍 v3.12.3 (riscv-env) took 5s
❯ cd VSDBabySoC
make pre_synth_sim
sandpiper-saas -i src/module/rvmyth.tlv -o rvmyth.v \
        --bestsv --noline -p verilog --outdir output/compiled_tlv
Please review our Terms of Service: https://makerchip.com/terms/.
Have you read and do you accept these Terms of Service? [y/N]: y
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 --outdir=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 \
                -I src/include -I src/module -I output/compiled_tlv; \
        cd output/pre_synth_sim; ./pre_synth_sim.out; \
fi
VCD info: dumpfile pre_synth_sim.vcd opened for output.
src/module/testbench.v:63: $finish called at 84999000 (1ps)
```

VIEW .vdc USING GTKWAVE



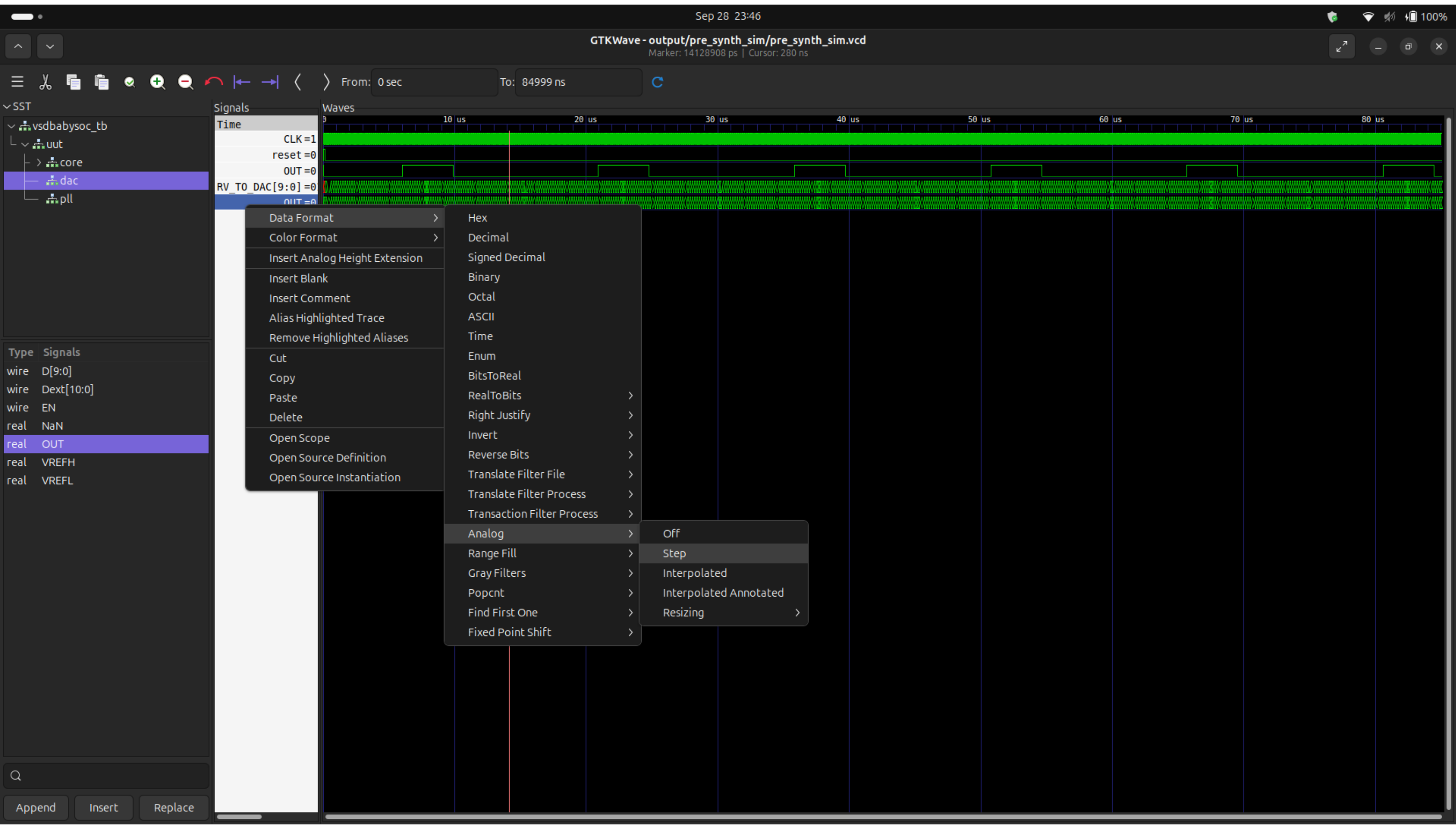
```
❯ gtkwave output/pre_synth_sim/pre_synth_sim.vcd

GTKWave Analyzer v3.3.116 (w)1999-2023 BSI

[0] start time.
[84999000] end time.
```

VIEW IN GTKWAVE, IMPORT CLK, RESET, OUT AND RV_TO_DAC[9:0] FROM vsdbabysox_tb and OUT FROM dac

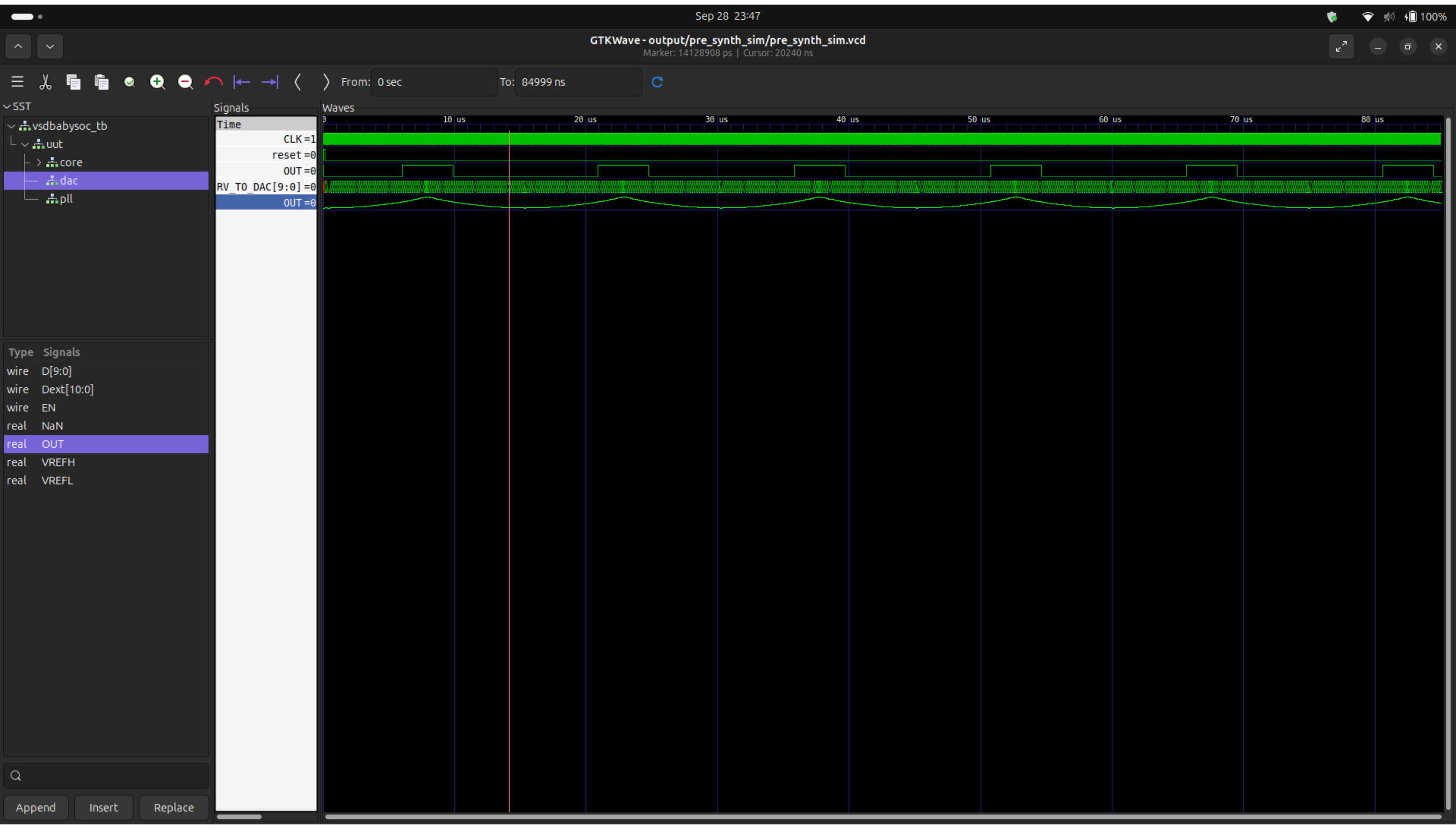


```
Sep 28 23:46
GTKWave - output/pre_synth_sim/pre_synth_sim.vcd
Marker: 14128908 ps | Cursor: 23230 ns
From: 0 sec    To: 84999 ns

SST
vsdbabysoc_tb
  uut
    core
    dac
    pll

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

Signals
Time
CLK =1
reset =0
OUT =0
RV_TO_DAC[9:0] =0
OUT =0
```

CONVERT DAC OUTPUT ‘OUT’ TO ANALOG (STEP)



INTERACTION BETWEEEN CLOCK, RESET, OUT OF CPU, DIGITAL INPUT(RV_TO_DAC) AND DAC ‘OUT’


Sep 28 23:47
GTKWave - output/pre_synth_sim/pre_synth_sim.vcd
Marker: 14128908 ps | Cursor: 20240 ns
From: 0 sec
To: 84999 ns
SST
vsdbabysoc_tb
uut
core
dac
pll
Signals
Time
CLK=1
reset=0
OUT=0
RV_TO_DAC[9:0]=0
OUT=0
Waves
Type Signals
wire D[9:0]
wire Dext[10:0]
wire EN
real NaN
real OUT
real VREFH
real VREFL
Append
Insert
Replace