Low Power Implementation in the 2008.09 Release

Jonathan Dawes

12th May 2009

What is the purpose of this tutorial?

The aim is to:

- Make you comfortable with SAIF
- Demystify the tool's manipulations of SAIF
- Get the best activity information possible

Achieve the best possible dynamic power

Agenda

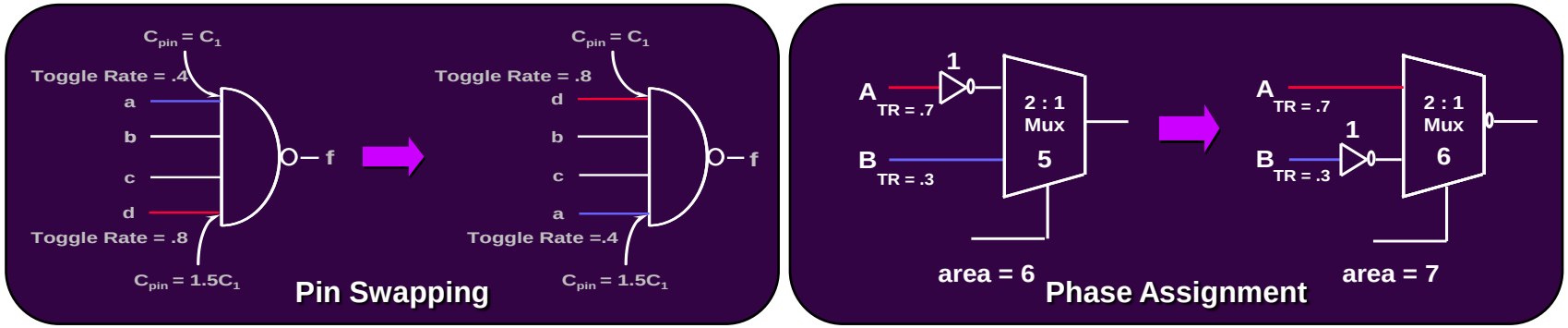
- Overview of power optimisation
- Activity information and SAIF
- How to generate good SAIF
- How to use it correctly through the flow
- What can we do if we don’t have good SAIF?
  - How to combine SAIF data from multiple sources including subblocks

What are the types of power usage?

- Leakage power
  - Cells that are not switching, but “leak” power
  - This reduces standby battery life
  - Address with Multi-Vt libraries
- Dynamic power
  - Power used by performing work
  - Charging up loads
  - This reduces the working-time of your battery
  - Addressed in many ways.

Optimisations for dynamic power during synthesis

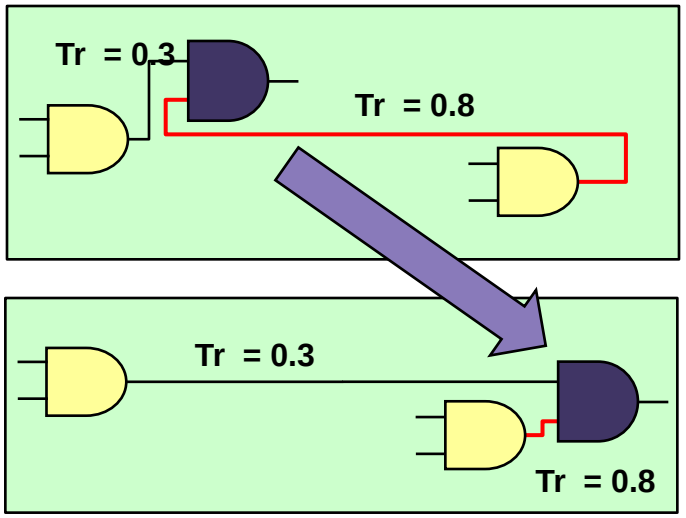
- Remap architecture to reduce power cost
- Bury high activity nets inside complex cells
- Swap high activity nets onto low-load pins



All these are performed automatically by DC

Optimisations for dynamic power during place & route

- Low Power Placement
  - shorten busy nets,
    so reducing the load



- optimize_pre_cts_power
  - Merge/Split/Remove/Optimize clock gating
  - Reduce the dynamic power of the clock tree
  - Reduce the dynamic power of data logic through improved clock gating
  - Physically aware clock gating

Activity information is the key

- Dynamic power is power used doing work
- We need to tell the tools what work looks like, and where it is done.
- This drives ***all*** dynamic power optimisation
- We must provide “activity” information for objects in the design.
- This is normally a SAIF file

Switching Activity Interchange Format

How good is your SAIF?

- Do you have the “perfect” SAIF file?
  - Accurately represents the real-world activity
  - Leaves no gaps or grey areas
  - Lucky you! Go and take a break!
- Or maybe not…
  - Can’t run gate-level sims early enough?
  - Don’t have a single toplevel testbench?
  - Need to combine data from multiple sources?
- Have you given up on dynamic power?

The worst case scenario

- The tools will assign “default” activity
  - 10% toggle rate wrt the fastest clock

- You can improve this by specifying more

```
set_switching_activity -static_probability
  0.5 -toggle_rate 0.2 [get_port in]
```

- The tool will propagate this data

The best activity information

- Use real activity information captured from an RTL or preferably gate-level simulation
- We will look at proven ways of doing this
- We will also look at combining data from multiple sources

- The aim is to get the very best activity data

Obsolete flows for generating SAIF

- RTL forward SAIF
  - rtl2saif is to be removed
- PLI and DPFLI

Generating SAIF in VCS

- RTL Simulation
  - Direct SAIF
  - VCD to SAIF
  - VPD to VCD to SAIF
- Gate Level Simulation

RTL Direct SAIF

- Advantages
  - Simple to use
  - Directly written out of VCS
  - No storage overhead
- Issues
  - System Verilog not well supported
  - Multi Dimensional Arrays not supported
  - This may result in low annotation due to gaps

RTL VCD to SAIF

- Advantages
  - This is the correct flow for System Verilog
- Issues
  - VCD file gets very large during simulation
  - Some SV constructs still being worked on.
    - Interfaces is the biggest gap
  - However, this will only create small gaps.
    - Propagation will back-fill lots of these

RTL VPD-VCD-SAIF

- Advantages
  - All the same benefits of VCD2SAIF
  - VPD is a lot smaller during simulation

- Issues
  - Still need to generate a VCD file on the fly
  - Same System verilog issues as VCD-SAIF
  - 3 step flow

Gate-level simulation

- Advantages
  - Simple
  - Maps directly to the gates you are optimising
  - Accuracy is much higher as real gates can be observed
- Issues
  - Gate level simulations may only be possible late in the flow
  - Long simulation runtimes

SAIF mapping

Why do we need SAIF mapping?

- Synthesis changes the names of many objects

- Netlist-driven tools need to match the RTL SAIF object names to gates
  - ICC
  - DC (multi-pass)
  - Primetime PX

DC recommended flows

- Initial compile
  - Read RTL and RTL SAIF
  - compile
  - Either
    - Write a gate-level SAIF out
  - OR,
    - Write out DC and PT-PX name maps
- Subsequent compile incremental flow (ASCII)
  - Read DC SAIF
  - OR use the name map from initial compile and read RTL SAIF

ICC recommended flow

- DDC retains the activity information
- Read netlist and RTL SAIF plus DC map
  - Also use -auto_map_names
  - Best coverage comes from using a map file ***and*** -auto_map_names*
- Optionally write a PTPX SAIF map out
  - May catch a few name changes

*Note read_saif will not accept -map_names at the same time as -auto_map_names. Always use –auto_map_names

PrimeTime – PX flow

- Read a netlist plus its related map file
- Read the RTL SAIF

Example Flow With RTL SAIF

An example DC script

```
saif_map -start
read design
read_saif -input rtl.saif -
  instance_name tb/chip -
  auto_map_names
```

- Must have auto_map_names
- Otherwise it will not annotate your SAIF
- No map file exists here

```
compile
```

DC post-compile script

```
saif_map -create_map -input
  rtl.saif -source_instance
  tb/chip
```

- Extract the information needed to write a name map file

```
saif_map -write_map map.ptpx -type
  ptpx
saif_map -write_map map.dc -type
  name_map
```

An example ICC script

- `saif_map -start`
- `saif_map -read_map map.dc`
- `read_saif -input rtl.saif -instance tb/chip -auto_map_names`
  - This combines the map file with auto name mapping

- `place/clock/route_opt -power`

- `saif_map -create_map -input rtl.saif -source_instance tb/chip`
- `saif_map -write_map map_icc.ptpx -type ptpx`

PT-PX with RTL SAIF

- Read and time design
- source map.ptpx
- read_saif rtl.saif -strip_path tb/chip

PT-PX with SAIF from DC

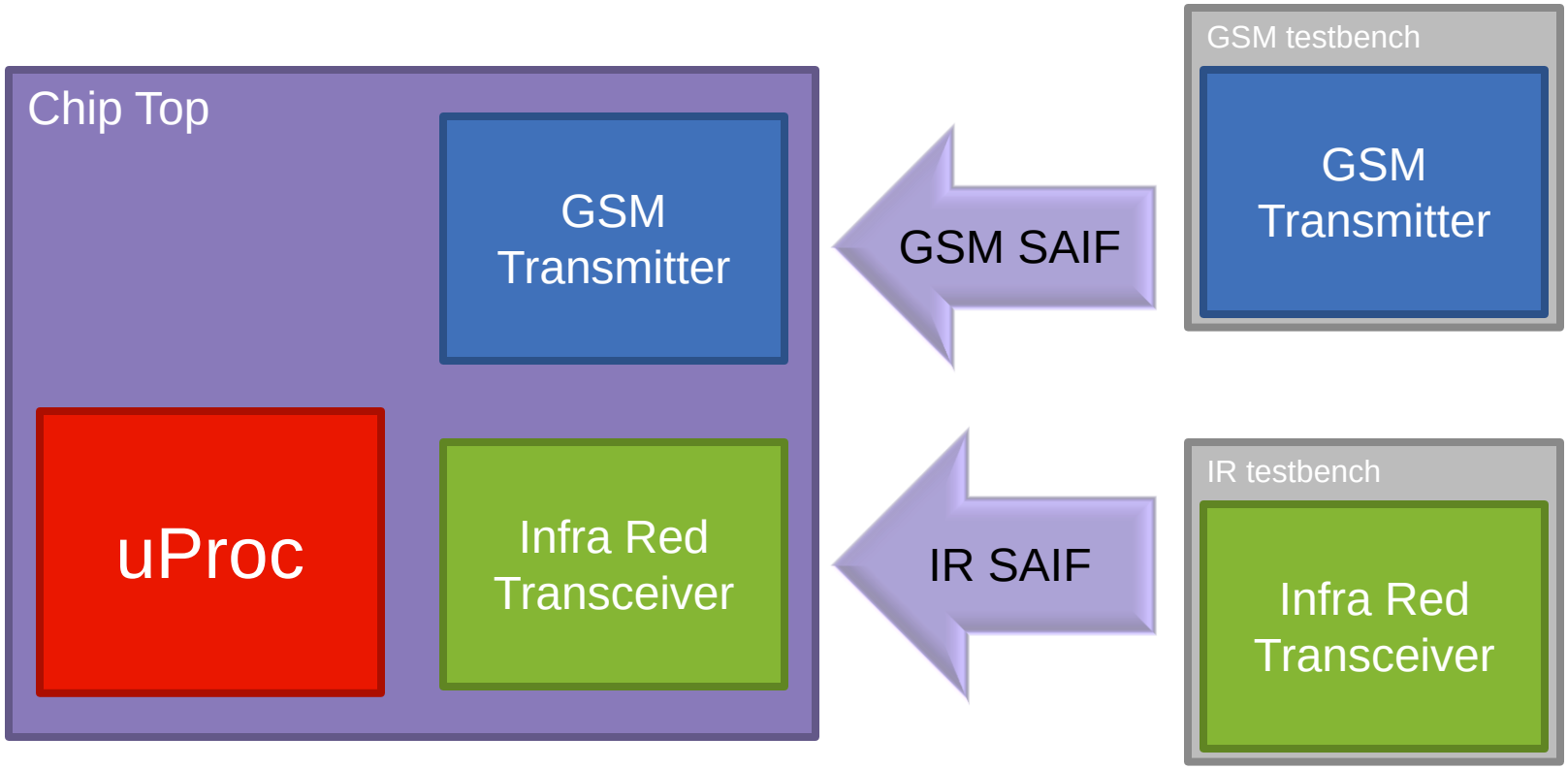
- DC and ICC can also write out SAIF
- This is then “gate-level” SAIF requiring no mapping
- However, the activity is still from the RTL simulation
  - Lower accuracy

Hierarchical SAIF

No decent top-level SAIF?

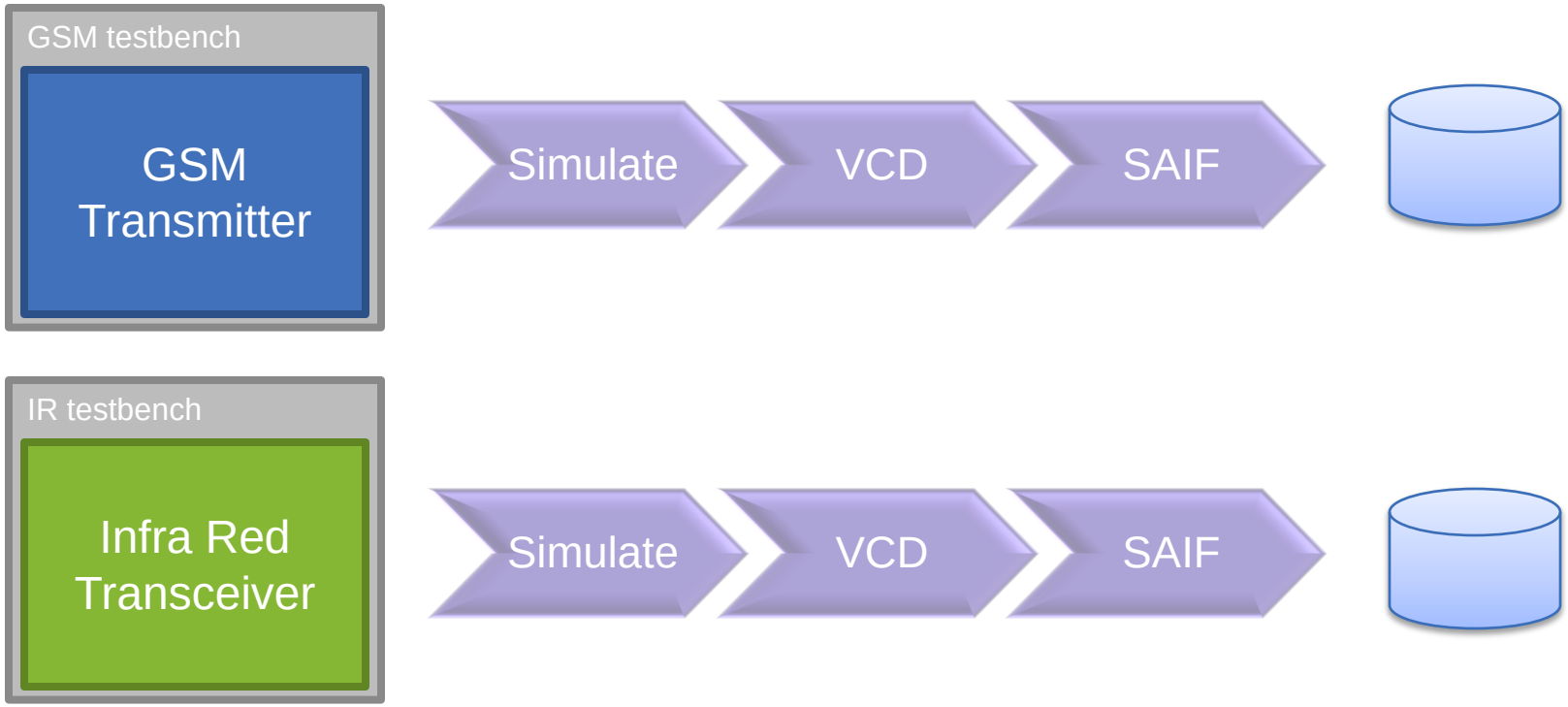
- Often it's not possible to put together one toplevel sim that shows worst case power
- Instead engineers would like to pull in module-level SAIF from smaller, block-level simulations
- This would allow you to create a chip “mode”
  - Based on certain blocks doing certain tasks
  - Can avoid mutually exclusive activities

Gathering data from multiple sources



Simulation

- Simulate each block in its own testbench
- Each testbench writes out VCD -> SAIF



How to use multiple SAIF files in DC

- Read in your SAIF

```
read_saif -auto_map_names -input blocka.saif
-instance_name tba/DUT -target_instance U_blocka

read_saif -auto_map_names -input blockb.saif
-instance_name tbb/DUT -target_instance U_blockb

read_saif -auto_map_names  -input  toplevel.saif
  -instance_name tb_top/DUT
```

  – Order is important here for annotation

- Compile your design.

Create name maps for each SAIF file

- Build the name maps from your SAIF files

```
saif_map -create_map -input blocka.saif
-source_instance tb_a/DUT -target_instance U_blocka

saif_map -create_map -input blockb.saif …

saif_map -create_map -input toplevel.saif …
```

Note we still use create_map for all blocks so that mapping is created for the names in your other SAIF files

Write out the name maps

```
saif_map -write_map ptpx.map -type ptpx
```

- Optionally write out a gate-level SAIF

```
write_saif -output toplevel_synth.saif
```

Using these SAIF files in ICC

```
saif_map -start
read verilog
saif_map -read_map map.dc
```

(Assuming flat DC compile)

Read in your SAIF

```
read_saif -auto_map_names -input blocka.saif -instance_name
   tba/DUT -target_instance U_blocka
read_saif -auto_map_names -input blockb.saif -instance_name
   tbb/DUT -target_instance U_blockb
read_saif -auto_map_names  -input  toplevel.saif
   -instance_name tb_top/DUT

place/clock/route_opt -power
```

Using this SAIF in PrimeTime-PX

```
source map.ptpx
read_saif  -strip_path tba/DUT -path
  U_blocka blocka.saif

read_saif -strip_path tbb/DUT -path U_blockb
  blockb.saif

read_saif rtl.saif -strip_path tb/chip
  toplevel.saif
```

What if I have different chip modes?

- Power usage can vary wildly with mode
  - 70% of time in standby
  - 20% of time receiving and processing
  - 10% of time transmitting a signal

- We are not likely to have a simulation!
- Instead we want to combine existing SAIF
- Merge them with appropriate weighting

How to merge SAIF files with weights

```
merge_saif –input_list {  \
        -input stby.saif –weight 70 \
        -input rx_process.saif –weight 20 \
        -input transmit.saif –weight 10} \
        [-simple_merge] –instance …
```

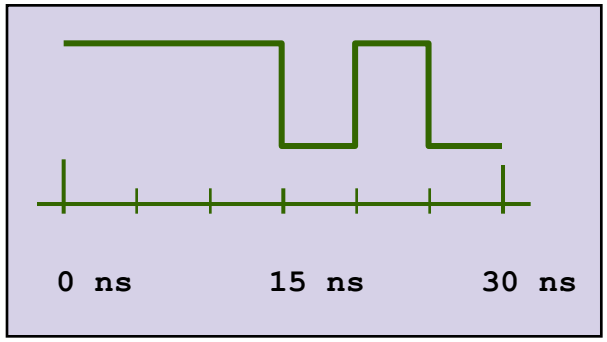
- Activity will be normalised over time
- Merged with the weighting specified

How is it calculated?

- Essentially the ratios of T0 and T1 are used
- The percentages are then applied to the new duration.

- Lets look at some terminology…

Switching activity terminology



So here (units ns):
TC = 3
T0 = 10
T1 = 20

Unused:
TR = 0.1
SP = 2/3

- Toggle Count (TC):
  A toggle is a logic transition (0→1 or 1→0). Toggle count (`TC`) is the number of toggles.
- Toggle Rate (Tr):
  Number of toggles per unit time (usually the simulation duration)
  `Tr = TC / duration`
- T1 , T0:
  Total time a signal is at logic 1, 0
- Static Probability (Sp):
  Probability of a logic 1 for a signal
  `Sp = T1 / duration`

Where is this info in a SAIF File?

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Mon May 17 02:33:48 2004")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-Scirocco-MX Power
   Compiler")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 10000.00)
(INSTANCE I_TOP
   (INSTANCE macinst
      (NET
       (z\[3\]
         (T0 6488) (T1 3493) (TX 18)
         (TC 26) (IG 0)
       )
```

- Each object has:
  - T0
  - T1
  - TC
  - (possibly TX)

So, how *is* a merge calculated?

TC = 5
T1 = 60
T0 = 140
Duration 200
Weight 80

Essentially add the ratios with weighting

$$T0_{merged} = T0_a/Duration_a * weight_a + T0_b/Duration_b * weight_b$$

TC=10
T1=80
T0=20
Duration 100
Weight 20

$$= 140/200 * 80\% + 80/100 * 20\%$$

$$= 0.6 \text{ or } 60\%$$

$$= 600 \text{ (new duration 1000)}$$

What about Xs?

TC = 5
T1 = 40
T0 = 140
TX = 20
Duration 200
Weight 80

Xs are removed from the ratio

$T0_a$ is still 140 but the duration is 180

So now $T0_{merged}$ =

$T0_a/Duration_a * weight_a$ +

$T0_b/Duration_b * weight_b$

TC=10
T1=80
T0=20
Duration 100
Weight 20

= (140/180) * 80% + (20/100) * 20%

= 662.22 (new duration 1000)

What about toggle counts?

TC = 5
T1 = 60
T0 = 140
Duration 200
Weight 80

TC=10
T1=80
T0=20
Duration 100
Weight 20

This works by expanding the durations then weighting

$TC_a = 5/200$

$= 25 / 1000$

$TC_b = 20/100$

$= 200 / 1000$

$Tc_{merged} = 25*80\% + 200*20\%$

$= 60$ (in 1000)

saif_merge –output results look odd?

- Actually the numbers are the same
- The duration specified is different
- $Duration_{merged} = duration_a * weight_a + duration_b * weight_b$

$= 200 * 80\% + 100 * 20\%$

$= 180$

A full description of the formula can be found at

https://solvnet.synopsys.com/retrieve/000360.html

What happens to incomplete SAIF?

- This is fine if all SAIF annotates all nets
- What if one SAIF only annotates a block?
  - All SAIFs must reference the same instance
  - However you may have built a toplevel SAIF by reading in a block-level SAIF only.

- What happens to the nets left out?

Default switching activity

- Nets not annotated by a given SAIF inherit the default switching activity
  - This is 0.1x the fastest clock (default)
  - If no clock defined, assume switches 10%

- This can cause a big jump in toggle count!
- Override with

```
set power_default_toggle_rate_type 0
```

What value do these nets assume?

- Unannotated nets now assumed constant

- The constant value is chosen as the opposite of the prevailing value
  - If saif_a says net is T1=70 T0=30
    - Assumes in saif_b this net is constant 0
  - If saif_a says T1=20, T0=80
    - Assumes in saif_b this net is constant 1

You are now SAIF experts!

- Use the best SAIF you can
- Gate level is more accurate and simpler
- You now have the correct name mapping flow for RTL SAIF
- You can create custom SAIF by
  - Merging 2 full files
  - Reading SAIF for hierarchical blocks
- Your optimisations will now deliver best dynamic power for your real chip

SYNOPSYS®

Predictable Success