Redesigning Power Systems on a Single Chip Micro Mote with Berkeley Analog Generator Low Dropout Series Regulator Generation

Jackson Paddock

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-124
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-124.html

May 14, 2021

Redesigning Power Systems on SCμM with BAG LDO Generation

by Jackson Paddock

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

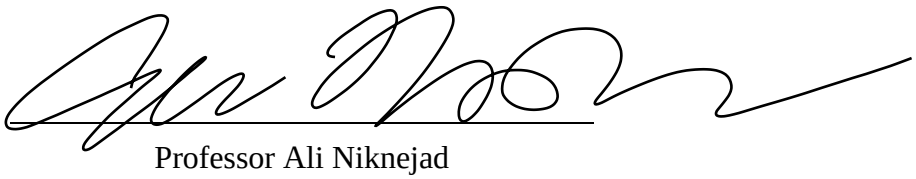
Committee:

Professor Kristofer S.J. Pister
Research Advisor

5/13/21

(Date)

* * * * * * *



Professor Ali Niknejad
Second Reader

5/14/2021

(Date)

Abstract

Redesigning Power Systems on a Single Chip Micro Mote with Berkeley Analog Generator Low Dropout Series Regulator Generation

by

Jackson Paddock

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Kristofer S.J. Pister, Chair

The Single Chip Micro Mote (SCµM) is a crystal-free radio chip with an on-board CPU developed at UC Berkeley in the Swarm Lab. This chip was designed to function as the brain of an untethered microrobot with no external components other than a power source required. SCµM also features an optical programmer so that no cables are even needed to program it. With its size and functionality, SCµM has the potential to allow a swarm of microrobots to communicate and perform complex tasks in tandem.

Like every other circuit, SCµM needs power to operate, and one of the most common power regulation circuits is an LDO (low dropout series regulator), which unlike many other DC-DC converter designs, does not require any switching or inductors, which take up a lot of area. This circuit is relatively simple, requiring only a voltage amplifier and one additional transistor, but the challenges in design come from balancing stability, accuracy, and area. The goal of this project was to write a process-independent script that will automatically design such a circuit quickly and accurately in any technology while requiring minimal manual adjustment on the part of the designer. Due to the prevalence of LDOs in integrated circuit chips, the script discussed in this report has many real-world applications.

To my parents and my brother, and my friends from BareStage.

Thank you for supporting me in everything I do and making my life that much brighter.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to offer many thanks to my advisor, Professor Kristofer Pister for guiding me through this program and always having more confidence in me than I at times have have in myself. Without him, I would not be where I am today and for that I would like to offer my utmost gratitude.

Thank you to Lydia Lee, who has been my mentor on this project and has always given me the guidance I needed when I was unsure of how to proceed with my work. Thank you for your endless support and patience.

Thank you to Alex Moreno, Austin Patel, David Burnett, Fil Maksimovic, and the rest of the SCµM team for all the advice and knowledge you have given me in the short time I've worked with you.

Chapter 1

Background on SCμM Power Systems

1.1 Power Domains

SCμM has eight power domains as listed in Table 1.2, each of which is supplied by an LDO connected to the off-chip battery voltage. There are also four different bandgap references, inside the blocks for the digital LDO, optical/alwayson, radio divider, and radio LO. Each of these LDOs is connected to the battery voltage for power, which is nominally at 1.5V, but can be brought down to 1.2V. The chip also uses an external VDDIO to supply the level shifters in the pad ring and GPIO to connect to chip-level inputs and outputs.

1.2 Power Consumption

The power consumption of the LDOs themselves are listed in Table 1.2. Each LDO reference voltage is set by a constant bandgap reference current running through a tunable resistor. The resistor for each LDO is the same and can be tuned to achieve discrete voltages between 0.8V and 1.2V. With a reference current of 500nA or 1μA, the resistor ranges between 950kΩ and 1.7MΩ. The LDOs that consume the most power in the amplifier, the Always On and Sensor ADC LDOs, also have the largest biasing networks, which are partially for the enable/disable bits that control whether or not the LDOs supply power to their loads.

State	Clock Rate	Average Current Consumption
Normal	5MHz	350μA
Radio on	5MHz	1.6mA
Low power	78kHz	200μA

Table 1.1: SCμM 3C current draw in different operating states [3].

[1]Better performance was seen in practice with a slightly higher reference voltage.

Domain	Operating Voltage			Amplifier Quiescent Current			Sub-Blocks
	Min	Nom	Max	Min	Nom	Max	
Digital (VDDD)		1V			4.763µA		Cortex, RAM
Auxiliary		1V			4.763µA		Auxiliary Digital Circuits, GPIO Levelshifters
Always On	0.8V		1.2V	7.12µA		7.24µA	Osc. DAC, Optical Receiver
Sensor ADC	0.8V		1.2V	7.16µA		7.16µA	ADC
Radio IF	0.8V	0.8V[1]	1.2V	4.85µA	4.85µA	5.32µA	Radio IF
Radio LO		0.8V			5.98µA		Radio LO
Radio PA		0.8V			4.84µA		Radio TX PA
Radio Divider		0.8V			760nA		Radio Divider

Table 1.2: SCµM power domains [2].

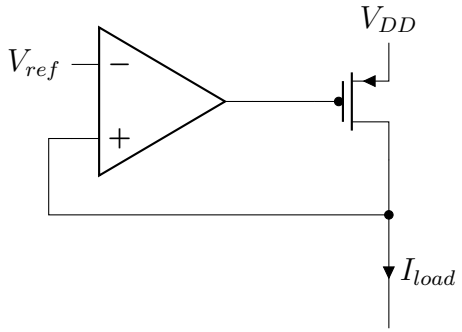


Figure 1.1: LDO with PMOS series device.

Combined, the regulators consume 40.5µA, which is more than 10% and 20% of the total operating current in the normal and low power states respectively (Table 1.1). Even in the radio on state, this current is not negligible. Disabling some LDOs may reduce this current consumption, but the LDO quiescent current is still high compared to the results seen in Section 2.4.

The dominant pole for the radio's IF, LO, and PA LDOs was placed at the LDO output to reduce the high frequency noise at the output and accommodate the sensitivity of the attached circuits. The rest of the LDOs have the dominant pole at the output of the amplifier. A dominant amplifier pole can lead to lower current consumption as discussed in Sections 2.2 and 2.3. The radio divider LDO does clearly have lower current than the radio LDOs with dominant output poles, but there is a lot of room for improvement with the other LDOs in terms of power efficiency. The current consumption of the digital, auxiliary, always on,

and ADC LDOs should be much closer to that of the radio divider with the right design, if not even lower. With the varying requirements for nominal output voltages and currents on each LDO, and the sensitivity of the radio LDOs excepting the divider LDO, the Berkeley Analog Generator (BAG), discussed in the next chapter, becomes a useful tool for efficiently reducing power consumption and catering to the specifications of each LDO.

For SCµM, reducing leakage current is a very important goal. For any application where the power supply is coming from a battery and not a lab bench, any reduction in power will allow SCµM to run for a longer time, increasing the scope of potential applications.

Chapter 2

LDO Design with BAG

2.1 Berkeley Analog Generator (BAG)

The Berkeley Analog Generator (BAG) [1] is in infrastructure that supports a collection of process-independent design scripts which, when given a set of input parameters, will automatically design a circuit to meet those parameters down to the transistor level. These scripts can make the design process of a circuit block significantly faster and have the potential to design more efficient circuit blocks than would practically be created by hand. This chapter discusses a BAG script in BAG2 created to design an LDO optimized for low power consumption in any process, and potential applications for these designs.

This chapter describes the process the design script in Appendix D uses to generate LDOs and provides examples of its results. The LDO design script has been validated in both the TSMC 65nm LP and TSMC 28nm processes. The BAG framework and scripts used are process-independent, but the process information for the technologies used is protected under nondisclosure agreements. All process-specific information is located on private servers hosted by the Berkeley Wireless Research Center (BWRC). The process information for TSMC 65nm LP and TSMC 28nm is included in the library paths of the respective BAG workspace repositories described in Appendix A. For non-BWRC readers, a template for establishing the correct hooks can be found at:

`https://github.com/ucb-art/BAG2_cds_ff_mpt`.

2.2 LDO Design

The performance parameters this report uses to design an LDO are the static output error, load regulation, noise rejection, and stability. Physical parameters used to determine transistor sizings and bias points are the load voltage, load current draw, load capacitance, power supply, and amplifier reference current. For this report and the associated design script, a 5T amplifier topology with an NMOS input pair was used. All diagrams in this section and LDOs designed with BAG use a PMOS series device. Although an NMOS series device is

possible, it may be difficult to keep in saturation unless the output common mode of the amplifier is near the supply voltage. This issue is discussed in more detail in Section 2.3.

In this circuit, the two main poles are at the output of the amplifier and at the output of the LDO. Although there are other capacitors that contribute to the frequency response of the LDO, for the purposes of this section the amplifier and series device are assumed to only have a single pole each. There is also a zero created by the capacitor (a combination of explicit and parasitic) between the amplifier output of the amplifier and the LDO output with the effective resistance it sees at $1/g_{m,ser}$ considered in this section. The zero may affect the phase margin at higher values of C_{amp} or higher gain from the series device, but the phase margin, derived from the loop gain in Figure 2.1 and given by Equation 2.2, is primarily determined by how far away the amplier and LDO output poles are. The unity gain frequency ω_0 of the loop gain in Equation 2.1 is affected by the DC gain as well as the poles and zeros.

$$LoopGain = \frac{A_{amp}A_{ser}(1+\frac{j\omega}{\omega_z})}{(1+\frac{j\omega}{\omega_{amp}})(1+\frac{j\omega}{\omega_{out}})} \tag{2.1}$$

$$PM = 180^\circ + \arctan\left(\frac{\omega_0}{\omega_z}\right) - \arctan\left(\frac{\omega_0}{\omega_{amp}}\right) - \arctan\left(\frac{\omega_0}{\omega_{out}}\right) \tag{2.2}$$

An unknown load capacitance can pose challenges for ensuring stability, for example if the load capacitance is higher than expected and the amplifier pole is dominant, the poles in Equation 2.1 move closer together, leading to instability if they become too close. This problem and possible solutions are discussed in more detail in Section 2.3.

After choosing a bias point for the gate of the series device (any voltage that keeps the transistor in saturation), the bias points for a 5T differential amplifier topology as shown in Figure 2.4 are entirely determined assuming a constant V* for the devices in the amplifier. The BAG LDO design script makes this assumption to simplify the choices it makes, but setting the amplifier input pair's V* higher than the amplifier load pair's V* improves noise and matching.

Modeling the circuit as a block diagram like in Figure 2.1, the relationship between the reference and the output of the LDO with a PMOS series is given by Equation 2.3, and the corresponding static error, expressed as a unitless fraction between the difference between the reference and output voltages and the reference voltage throughout this report, is given by Equation 2.4 where $A_{ser} = g_{m,ser}r_{o,ser} > 0$.

$$A_{amp}(-A_{ser})(V_{out} - V_{ref}) = V_{out}$$

$$\frac{V_{out}}{V_{ref}} = \frac{A_{amp}A_{ser}}{A_{amp}A_{ser}+1} \tag{2.3}$$

$$\mathcal{E}_s = 1 - \frac{V_{out}}{V_{ref}} = \frac{1}{A_{amp}A_{ser}+1} \tag{2.4}$$

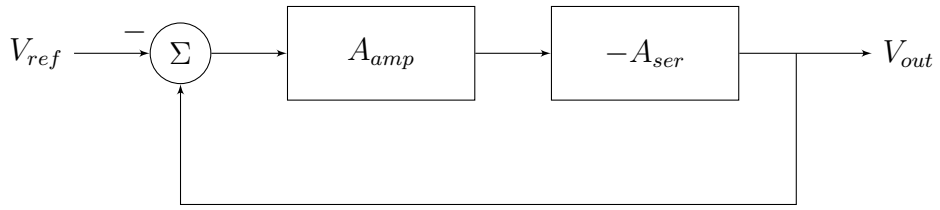


Figure 2.1: Block diagram of LDO feedback loop for a PMOS series device.

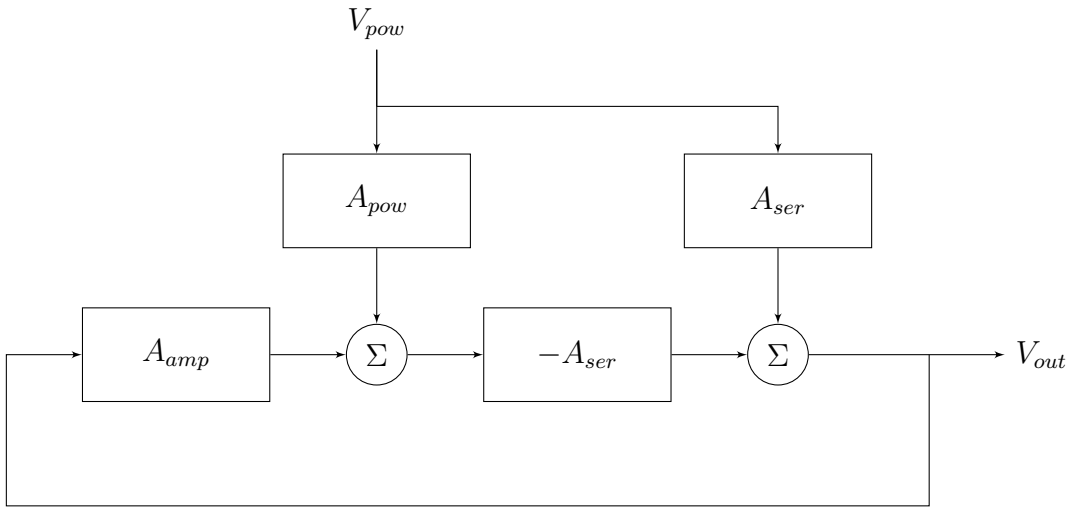


Figure 2.2: Block diagram of LDO power supply gain loop for a PMOS series device.

For high amplifier and series device gains, the static error in Equation 2.4 will be very small and the actual static error will be dominated by the device offsets of the amplifier devices. The LDO design script in Appendix D does not account for this in the approximations it makes.

Modifying Figure 2.1 slightly to find the gain from the power supply to the output through both the amplifier and the series device produces Figure 2.2, which in combination with Equation 2.3 can be used to find the PSRR as in Equation 2.5. For a 5T differential amplifier, A_{pow} approaches 1 as the tail resistance increases.

$$-A_{ser}(A_{amp}V_{out} + A_{pow}V_{pow}) + A_{ser}V_{pow} = V_{out}$$

$$PSRR = \frac{V_{pow}}{V_{out}} \cdot \frac{V_{out}}{V_{ref}} = \frac{A_{amp}A_{ser} + 1}{A_{ser}(1 - A_{pow})} \cdot \frac{A_{amp}A_{ser}}{A_{amp}A_{ser} + 1} = \frac{A_{amp}}{1 - A_{pow}} \tag{2.5}$$

Considering the amplifier output pole as a part of A_{amp} and the LDO output pole as a part of A_{ser} and that A_{pow} is relatively independent of frequency near the amplifier and

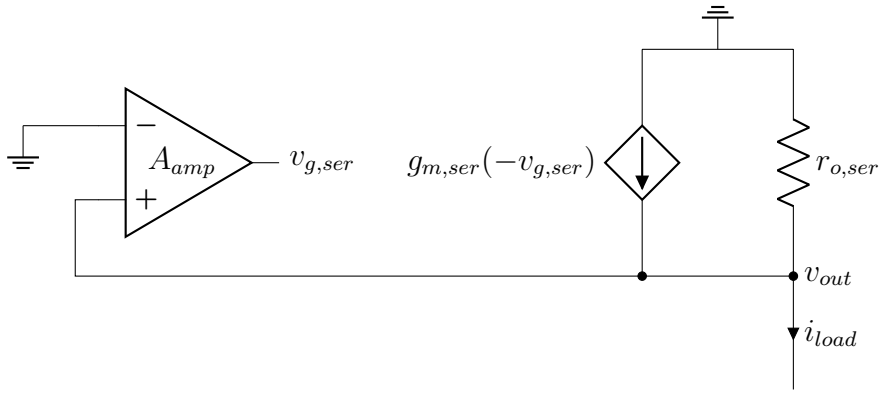


Figure 2.3: Small signal model for LDO load regulation calculation at low frequency with a PMOS series device.

output poles, the dominant pole and bandwidth of the PSRR equation is the amplifier output pole.

To calculate load regulation for a peak load current variation of 10% from the nominal value, which is expressed as a unitless fraction between the resulting peak to peak regulator output voltage and the average regulator output voltage, the small signal model with an abstracted amplifier in Figure 2.3 provides Equation 2.6.

$$v_{g,ser} = A_{amp} v_{out}$$

$$\frac{v_{out}}{r_{o,ser}} = i_{load} - g_m v_{g,ser} = i_{load} - g_m A_{amp} v_{out}$$

$$v_{out} = \frac{r_{o,ser}}{1 + A_{amp} A_{ser}} i_{load}$$

$$LoadReg = \frac{\Delta v_{out}}{V_{out}} = \frac{\Delta i_{load} r_{o,ser}}{V_{out}(1 + A_{amp} A_{ser})} = \frac{0.2\, I_{load} r_{o,ser}}{V_{out}(1 + A_{amp} A_{ser})} \tag{2.6}$$

From these equations, a higher amplifier and series device gain improves static error, PSRR, and load regulation. A lower series device output resistance also improves the load regulation, but shorter channel devices also have a lower gain. For $A_{amp} A_{ser} >> 1$, the load regulation equation becomes dependent on $1/g_{m,ser}$. A shorter channel length also decreases the capacitance at the load of the amplifier, bringing the PSRR bandwidth higher. Because the amplifier pole sees the output resistance of the amplifier, larger devices and smaller effective widths will decrease the amplifier pole frequency and PSRR bandwidth, and decrease the power consumed in the amplifier. The PSRR will also be increased by the higher gain from longer channel devices in the amplifier. Conversely, shorter channel devices and larger effective widths will increase the amplifier pole frequency and PSRR bandwidth, and increase the power consumed in the amplifier. The PSRR will also decrease with shorter channel amplifier devices.

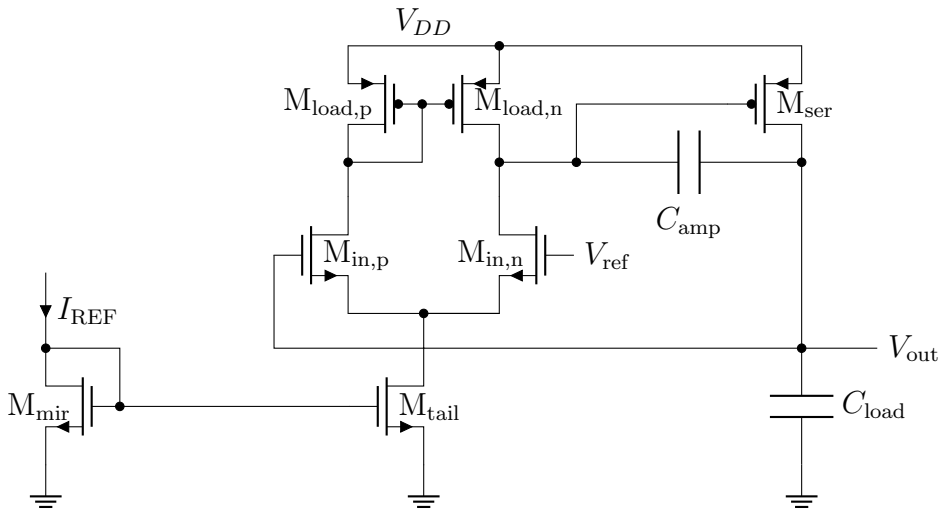


Figure 2.4: LDO with 5T differential amplifier topology and PMOS series device.

2.3 Script Design Approach

The file `regulator_ldo_series.py` defines `bag2_analog__regulator_ldo_series_dsn` as a subclass of the `DesignModule` class, which given a set of input parameters described in Table 2.1 will design an LDO with either an n-type or p-type series device and a 5T differential amplifier with an n-type input pair for the amplifier as shown in Figure 2.4. See Appendix C for an example input YAML file, and Appendix D for the script itself. Although this script has the option of using an NMOS series device, using a PMOS is recommended. Unless the supply voltage is high or the device thresholds are very low, using an NMOS series device may be difficult. The minimum series device gate voltage to keep the device out of the subthreshold region is the device's threshold voltage plus the regulator output voltage, which may need to exceed the supply voltage to provide the nominal load current. A different amplifier topology with a greater output swing would be more suited to using an NMOS series device.

The inputs to this script are listed with descriptions in Table 2.1. The output is a single dictionary, containing more dictionaries listed in Table 2.2 of the component sizes of the particular design.

The first part of the LDO that the code designs is the series device. Because the current is known, the drain and source voltages are known (the supply voltage and the output voltage, depending on what type of device it is), and the channel length is known, the only unknowns are the device's effective width and gate voltage. By setting the gate voltage at some value between the bounds that keep all devices in saturation, the effective width is easily determined by a ratio of the total load current to the device current at the base width. Since the transistor data is simulated with a fixed finger width, this code assumes that the

Parameter	Description
specfile_dict	Transistor database spec file names for each device
ser_type	n or p for type of series device
vdd	Supply voltage in volts
vout	Reference voltage to regulate the output to
iload	Bias current of series device, in amperes
iref	Reference current for amplifier biasing, in amperes
iampmax	Maximum amplifier current, in amperes
cload	Load capacitance from the output of the LDO to ground
cdecap	Maximum additional capacitance added to circuit
rsource	Resistance from the power supply, in ohms
err	Maximum percent static error at output (as decimal)
psrr	Minimum power supply rejection ratio (dB, 20*log10(dVdd/dVout))
psrr_fbw	Minimum bandwidth for power supply rejection roll-off
pm	Minimum phase margin for the large feedback loop, in degrees
loadreg	Maximum absolute change in output voltage given change in output current
load_pole	True to ensure dominant pole is at theregulator output
v_res	Resolution of voltage bias point sweeps, in volts
sim_env	Simulation environment
l_dict	Transistor channel length dictionary
th_dict	Transistor flavor dictionary

Table 2.1: LDO design script input parameters.

Parameter	Description
w_dict	Dictionary of transistor channel width per finger for each device
l_dict	Dictionary of transistor channel length for each device
nf_dict	Dictionary of transistor number of fingers for each device
th_dict	Dictionary of transistor flavor for each device
type_dict	Dictionary of transistor type, n or p, for each device
cap_dict	Dictionary of values of C_{amp} and C_{load}

Table 2.2: LDO design script output dictionary keys.

device currents, capacitances, and transconductances scale linearly with increased effective width. Additionally, without any information on minimum or maximum finger widths of the devices used, this code assumes that the base finger width is the minimum value, and twice that width is the maximum finger width. This assumption also ensures that any effective width greater than the minimum finger width is possible, with accuracy of device matching only limited to the how well the model matches the devices and what scale of resolution is possible for fabrication. The class function `resize_op(op,wm)` takes in a dictionary of transistor parameters and scales them up by the factor `wm` that the effective width increases by. In pseudocode, this first layer of the design is:

best LDO design ← None, $I_{amp,best} = \infty$
for $V_{g,ser}$ where M_{ser} in saturation **do**
  size M_{ser} for I_{load}
  design amplifier(specs)
  if $I_{amp,new} < I_{amp,best}$ **then**
    best LDO design ← new LDO design, $I_{amp,new}$
  end if
end for
return best LDO design

To simplify the calculations, all amplifier stage devices ($M_{load,p}$, $M_{load,n}$, $M_{in,p}$, $M_{in,n}$, and M_{tail}) are assumed to have the same V*. With the series device gate voltage set, the output common mode voltage of the amplifier is also set. Because the gate and drain of the amplifier load pair are tied together on $M_{load,p}$, all bias voltages for the amplifier devices are set by V*. The only parameter that is not fixed for these devices at this point is the effective width, which will be determined later.

The drains of $M_{in,p}$ and $M_{in,n}$ are tied to the series device gate voltage as previously set in the sweep of the series device gate, and the gates are tied to V_{ref}. V* is already determined for the devices in this amplifier stage, so a sweep of the input pair source voltage will determine what value will best match their V* with the load devices.

The only remaining unknown bias point is the tail device gate, which can be determined through a sweep to match V* as with the input devices. Anticipating that an LDO produced by this script will be fabricated and therefore require layout, this script also sizes M_{mir} to ensure that they have the same finger width to promote better matching between devices. Also with layout in mind, each device is later sized to have an even number of fingers so a common centroid pattern is possible. In the case where the minimum number of fingers desired is some number other than two, for example if M_{mir} has a large number of fingers and M_{tail} is at its minimum, but the fingers will not fit in a single row given the area prescribed for the LDO on a chip, the script can easily be modified to set a new guaranteed factor of the number of fingers for a device. For the M_{tail}, line 156 in the code in Appendix D, which guarantees an even number of fingers, can be changed from:

```
156 nf_tail = int(2*max(((2*op_load['ibias'])//Id_tail)+1,((2*op_in['ibias'
      ])//Id_tail)+1))
```

to the following to guarantee that the number of fingers is a multiple of four:

```
156 nf_tail = int(4*max(((2*op_load['ibias'])//Id_tail)+1,((2*op_in['ibias'
      ])//Id_tail)+1))
```

For M_{mir}, lines 147 through 149 can be changed from:

```
147 m_mir = iref/(2*op_mir['ibias'])
148 wm_mir = (m_mir%1 + 1)
149 nf_mir = 2*int(m_mir)
```

to the following to guarantee that the number of fingers is a multiple of four:

```
147 m_mir = iref/(4*op_mir['ibias'])
148 wm_mir = (m_mir%1 + 1)
149 nf_mir = 4*int(m_mir)
```

For $M_{load,p}$, $M_{load,n}$, $M_{in,p}$, and $M_{in,n}$, the guaranteed factor can be modified similarly to line 149 on lines 163 and 167 for the load devices and input devices respectively, additionally changing the multipliers within the `max()` function in line 153 from '2' to the desired factors. While these modifications will always produce a potential sizing if the load, input, or tail device factors are changed, increasing the guaranteed factor for the number of mirror device fingers may increase the total current at a given bias point past the value of $iref$, and that bias point will be ignored.

With the bias current $iref$ as an input, the effective width of M_{mir} is uniquely determined (as long as $iref$ is larger than twice the base finger width current at the bias point). All of the amplifier devices excluding the mirror are then set at minimum effective width based on which device has the largest base current, accounting for the fact that the tail device sinks the current of both input and load devices. Up to this point, the amplifier design in pseudocode (with ranges on the for loops to keep devices in saturation) is:

$V^*_{amp} \leftarrow V^*_{load}$
for $V \in [0, V_{ref} - V_{th,in}]$ **do**
  if $V^*_{in} = V^*_{amp}$ **then**
    $V_{s,in} \leftarrow V$
  end if
end for
for $V \in [V_{th,tail}, V_{s,in} + V_{th,tail}]$ **do**
  if $V^*_{in} = V^*_{amp}$ **then**
    $V_{g,tail} \leftarrow V$
  end if
end for
size M_{mir} for I_{ref}
$W_{finger,tail} \leftarrow W_{finger,mir}$
$I_{tail} \leftarrow 2 * max(I_{tail,finger}, 2 * I_{in,finger}, 2 * I_{load,finger})$
size M_{tail} for I_{tail}
size M_{in}, M_{load} for $I_{tail}/2$

The final part of the amplifier design checks that specs are met and adds capacitors to make the feedback loop stable. The specs that are most significantly affected by adding capacitors and sizing up the amplifier devices are the phase margin and the power supply rejection ratio bandwidth. There are two places to put the dominant pole in this circuit: at the output of the amplifier and at the output of the LDO. The primary advantages of placing the dominant pole at the output of the amplifier are low power and a lower value of explicit capacitance added to the circuit, which can take up a lot of space on a chip. Power consumption is generally lower with a dominant amplifier pole because the amplifier output resistance is highest with minimally sized devices, and a higher resistance means a lower pole, even with less capacitance than may be necessary for a dominant pole at the output of the LDO. The disadvantages of a dominant amplifier pole are that the PSRR bandwidth follows this pole closely, so a lower pole that is necessary for stability kills higher frequency noise rejection, and if the estimate of the LDO load capacitance is too low, the poles will be closer than expected and may lead to instability. In case the LDO load capacitance is unknown, one of the inputs to the design script `load_pole` ensures that the dominant pole is placed at the LDO output and the attempt to set it at the amplifier output is skipped.

If a load pole at the amplifier output is allowed, the script begins by checking if the LDO meets the specs without any additional capacitance, then increases the capacitor tied to the amplifier output until either all specs are met, the PSRR bandwidth drops below the minimum spec, or the maximum allowable capacitance is added. In the case where the spec is not met or the dominant pole is placed at the output of the LDO, the script adds the maximum allowable capacitance to the LDO output to bring that pole as low as possible. This has little effect on the PSRR bandwidth, which is still tied to the amplifier output pole. If the phase margin and PSRR bandwidth are still too low, the script begins increasing the effective width of the amplifier devices to decrease the output resistance and drive the amplifier pole higher. This excludes the tail mirror device because the reference current it takes is fixed and does not affect the pole.

With the tail finger width fixed so it matches the reference current mirror, the only way to increase its effective width is by adding fingers. This script does so two fingers at a time to promote symmetry in layout as mentioned before. If the desired factor for the tail device is a number other than two, changing the number of fingers added to the number of tail fingers `nf_tail` on line 237 to that factor will ensure the proper number of fingers. At this point in the loop, the tail device is resized but the amplifier load and input devices also have to be resized relative to the tail. Simply adding fingers to these devices will not be enough to match their currents because they don't necessarily have the same finger widths as the tail or each other, so they are resized relative to the new tail bias current. This process continues until either a solution is found, or the maximum amplifier current is exceeded. The pseudocode for upsizing transistors and ensuring stability is:

if specs met **then**
  return amplifier design
end if

if not load_pole **then**
  $C_{load} \leftarrow 0$
  for $C_{amp} <$ max decap **do**
    if specs met **then**
      return amplifier design
    end if
  end for
end if
$C_{load} \leftarrow C_{max}$
$C_{amp} \leftarrow 0$
while $I_{tail} < I_{max}$ **do**
  if specs met **then**
    return amplifier design
  end if
  $nf\_tail \leftarrow nf\_tail + 2$
  size M_{in}, M_{load} for $I_{tail}/2$
end while
return no design

Once the amplifier at a bias point is designed, the function `op_compare(op1,op2)` is called in `meet_spec(**params)` to determine whether to keep or ignore the current iteration. This function is set to compare only the amplifier bias current but can be changed to optimize for any other performance parameter specified in the design input file. Line 335 also sets the maximum amplifier bias current to the bias current of the best amplifier design found up to that point. This line may be slightly redundant considering that the previous line calling `op_compare(op1,op2)` also compares the amplifier bias currents and will always choose the new design because of the assignment in line 335, but calling `op_compare(op1,op2)` allows for the optimized parameter to be swapped out more easily, and the assignment of the maximum bias current may shorten the run time of the while loop on line 213 in the amplifier design if the bias current is already greater than will be considered for the final design.

This script utilizes the LTICircuit class as defined in `BAG_framework/data/lti.py`, which can use the transistor parameters obtained for the design and solve the small signal equations of the circuit model when the nodes each component connects to are defined. The resulting circuit model can be used to calculate the transfer function between any two nodes with voltage or current inputs, giving the gain and bandwidth of a particular transfer function. Using this class to calculate the circuit performance parameters is advantageous because it allows for more rigorous calculations of small signal circuits (i.e. including all parasitic capacitors) and can easily be used to model more complicated amplifier topologies quickly without having to derive precise transfer functions by hand. LTICircuit also does not require a license to use as other circuit modeling software like Cadence Virtuoso does.

Reference Voltage	1V
Load Current	1mA
Static Error	-2.756m
Amp Current	4.763µA
Phase Margin	85.62°
PSRR	43.87dB
PSRR Bandwidth	461.9Hz
Load Regulation	96.50µ
Amp Capacitor	3.60pF
Load Capacitor	452pF
Total Gate Area	756.6µm^2

Table 2.3: SCµM 3C auxiliary and digital LDO performance.

2.4 Comparison to Hand-Designed LDOs

SCµM Digital and Auxiliary LDOs

SCµM is generally operated with an auxiliary and digital LDO reference and output voltage of 1V and a battery voltage of 1.5V. Both of these LDOs, which use the same schematic, have an PMOS series device. Table 1.1 lists the three main operating points of SCµM and their total operating current draws. With these numbers, a reasonable nominal current to design for seems to be about 1mA based on the increase in current when the radio is on.

The load capacitance of both the auxiliary and digital LDOs on SCµM contributed by the layout and other circuits is unknown. For this application, PSRR bandwidth is not as important because the regulator loads are entirely digital, so the dominant pole will be placed at the output of the amplifier as it is on SCµM 3C. With this in mind, what is necessary to generate a viable LDO for SCµM is an upper bound on that unknown load capacitance. For this thesis, the assumed load capacitance value will be 100pF. If a better estimate is determined through extraction or measurement, the BAG design script can be rerun with a higher load capacitance by changing a a single line in the input YAML file.

Tables 2.4 through 2.9 and Figures 2.6 through 2.8 show the inputs (in the format the design script would read them) and respective output performances of the LDO design script for various series device and amplifier channel lengths. There is a slight variation between the performance calculated by the script and the simulated performance, but generally the two are fairly close. The biggest issue with these designs is the bias points of the amplifier. The script uses the function `estimate_vth(db,is_nch,lch,vgs,vbs)` to estimate the threshold voltage of a device given a bias point, but is not very accurate. This function uses the the device's V* and V_{gs} to estimate the threshold, assuming a quadratic model for `vgs/lch` $<$ 1V/µm and a linear model otherwise, which varies dramatically as V_{gs} changes. Due to the

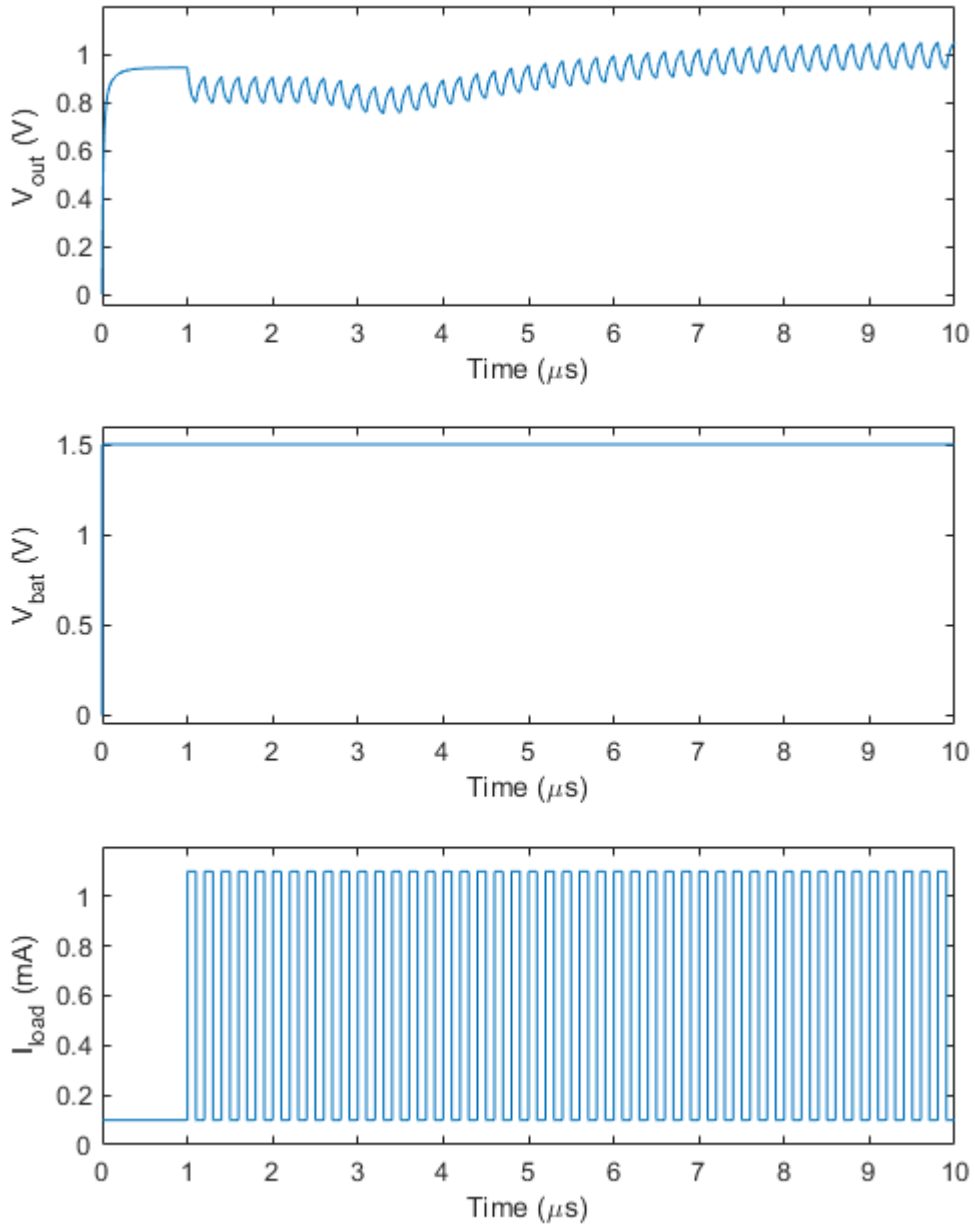


Figure 2.5: SCµM 3C auxiliary and digital LDO transient response at 5MHz.

inaccuracies in the function it is difficult to ensure that all devices are close enough to the threshold to achieve high gain and low current consumption but are still in saturation. Even using the estimated threshold of the amplifier load (which is higher than the series device for the designs in the tables because of the channel length), the PMOS devices generally have a lower V* than the NMOS, so the load devices can be in saturation with the input and tail devices pushed subthreshold. Of the LDOs listed in Tables 2.4 through 2.9, the designs with all devices in saturation are the ones with an amplifier channel length of 5µm. The remaining designs with amplifier channel lengths of 500nm and 1µm have the input and tail devices subthreshold (as determined by simulation), as well as the load devices for the designs with 280nm channel length series devices. On average, the subthreshold devices only had V_{gs} lower than the threshold by about 71mV, with the greatest difference being about 150mV.

The 5µm amplifier channel length designs are each better than the auxiliary and digital LDOs currently on SCµM for each parameter, with the exception of the phase margin, PSRR bandwidth, and peak to peak of the transient plots at 5MHz. Although the phase margin is slightly lower for these designs, they are all still well within the bounds for stability. The PSRR bandwidths are low, especially with larger series devices, but the PSRR bandwidth of SCµM is also only 461.9Hz, less than twice the PSRR bandwidth for the design with a 280nm series device channel length. Given that this script was designed to optimize for power, this design was very successful. Replacing the auxiliary and digital LDOs on SCµM with the 5µm amplifier channel length and 280nm series device channel length design would decrease the amplifier current consumption by 94.8% from 4.763µA to 248.1nA at a load current draw of 1mA. Without even considering the large resistors and capacitors needed for the LDOs presently on SCµM, this design only takes up 10.3% of the device gate area those LDOs need. On top of this, the design script runs much faster than it takes to design an LDO by hand, with a total runtime of about 80 seconds.

If having the amplifier devices slightly below the threshold is not an issue for matching in the particular process all nine designs listed in Tables 2.4 through 2.9 are viable options with significantly lower amplifier current consumption, the highest being 1.432µA with an amplifier channel length of 500nm and device channel length of 320nm. Across every single option, the only specs that are worse than the parameters in Table 2.3 are again the phase margin and PSRR bandwidth, but all options are stable and some even have a higher PSRR bandwidth. The only point of concern is the slightly lower phase margin of the design with an amplifier channel length of 500nm and a series device channel length of 300nm in combination with the unknown parasitic load capacitance on SCµM, but if that value is discovered to be more than 100pF, the PSRR bandwidth is high enough that adding a very small capacitor at the amplifier output will likely resolve the issue of stability without bringing the bandwidth down too far compared to the LDOs on SCµM.

Each transient simulation in Figures 2.6 through 2.8 has a load current that switches between 100µA and 1.1mA at 5MHz, which is an overestimate of how quickly the current will change between two extreme values. At this frequency and magnitude, the output voltage only changes by about 300mVpp for amplifier channel lengths on 500nm and 1µm,

and 200mVpp for an amplifier channel length of 5µm. The digital and auxiliary LDOs on SCµM only have a voltage variation of about 100mVpp, which is due to the larger explicit capacitor at the load.

While this data is only for the auxiliary and digital LDOs, with the right set of parameters the design script can also generate a new schematic for each of the other LDOs on SCµM. There has been discussion of moving SCµM to a different technology process. Given reasonable specs, this script can generate the LDO designs for that project with ease, leaving only the verification simulations and layout to be done manually.

EE290C: 28nm SoC for IoT

For the Spring 2021 iteration of EE290C at UC Berkeley, the students collectively designed and taped out a system-on-a-chip (SoC) for internet-of-things (IoT) applications, with an on-chip CPU, radio, and Bluetooth Low Energy (BLE) and Advanced Encryption Standard (AES) compatibility. This chip design also includes two LDOs for the auxiliary and digital domains both of which use the same amplifier designs and device sizings. These LDOs were designed without the use of BAG. The simulated specs across temperature and process corners are shown in Table 2.10.

These LDOs were designed with 400nm channel length devices for the amplifier devices and a channel length of 150nm for the series device. Due to the nature of the course, these LDOs were designed at the same time as the rest of the chip, so almost no information about the load was known until about halfway through the design process. Even after the current draw from the analog and digital halves of the chip were estimated at 10mA or lower, the load capacitance was still unknown. Because of this, and a desire for a higher frequency power supply noise rejection bandwidth, the dominant pole was placed at the output of the LDO to keep the amplifier output pole and PSRR bandwidth higher and ensure the stability of the feedback loop. Although the phase margin for this design is very low, the simulated performance does not include the additional capacitance from the chip's power grid and circuit blocks attached to it. Because the LDO output is the dominant pole, that additional parasitic capacitance will only increase the phase margin. Through simulation, it also appears that the phase margin increases if the load does not draw the full 10mA allotted to each regulator.

With the same channel lengths for each device as the hand-designed LDO, the results of running the generator script with the parameters listed in Table 2.11 are shown in Table 2.12 (for a series device channel length of 150nm). For this set of inputs, the PSRR and phase margin improved significantly, with a slight decrease in required gate area. The static error has notably gotten worse, but the other parameters have stayed relatively similar. Each generated design of this LDO in Tables 2.11 through 2.14 has all devices in saturation, unlike the designs for SCµM in the 65nm process. Of the options, targeting low power and higher PSRR bandwidth, the 1µm amplifier channel length and 150nm series device channel length

[1]Text in YAML file is the corner name instead of "nominal."

Spec	Value
ser_type	p
vdd	1.5
vout	1
iload	1e-3
iref	500e-9
iamp_max	100e-6
cload	100e-12
cdecap	100e-12
rsource	0

Spec	Value
err	1e-3
psrr	50
psrr_fbw	1e3
pm	60
loadreg	1e-3
load_pole	False
v_res	10e-3
sim_env	nominal[1]

Device	Base Finger Width	Channel Type	Device Type
amp_in	500nm	NMOS	Input
amp_load	500nm	PMOS	Input
amp_tail	500nm	NMOS	Input
amp_mir	500nm	NMOS	Input
ser	500nm	PMOS	Input

Table 2.4: SCµM generated LDO input parameters for amplifier channel length of 500nm.

L_{ser}	280nm	280nm		300nm		320nm	
Data Source	SCµM	Script	Sim.	Script	Sim.	Script	Sim.
Static Error	−2.756m	672.8µ	123.9µ	528.9µ	−225.5µ	462.4µ	−266.3µ
Amp Current	4.763µA	465.3nA	297.8nA	878.9nA	727.4nA	1.462µA	1.432µA
Phase Margin	85.62°	61.88°	69.07°	62.94°	60.53°	82.33°	80.06°
PSRR	43.87dB	63.37dB	60.77dB	66.08dB	68.31dB	68.97dB	71.48dB
PSRR Bandwidth	461.9Hz	2.073kHz	3.040kHz	1.602kHz	2.116kHz	1.708kHz	717.1Hz
Load Regulation	96.50µ	170.4µ	132.8µ	162.9µ	10.95µ	10.50µ	11.87µ
Amp Capacitor	3.60pF	0F	0F	0F	0F	625.7fF	625.7fF
Load Capacitor	452pF	0F	0F	0F	0F	0F	0F
Total Gate Area	756.6µm^2	112.3µm^2	112.3µm^2	153.8µm^2	153.8µm^2	137.2µm^2	137.2µm^2

Table 2.5: SCµM generated LDO performance for amplifier channel length of 500nm.

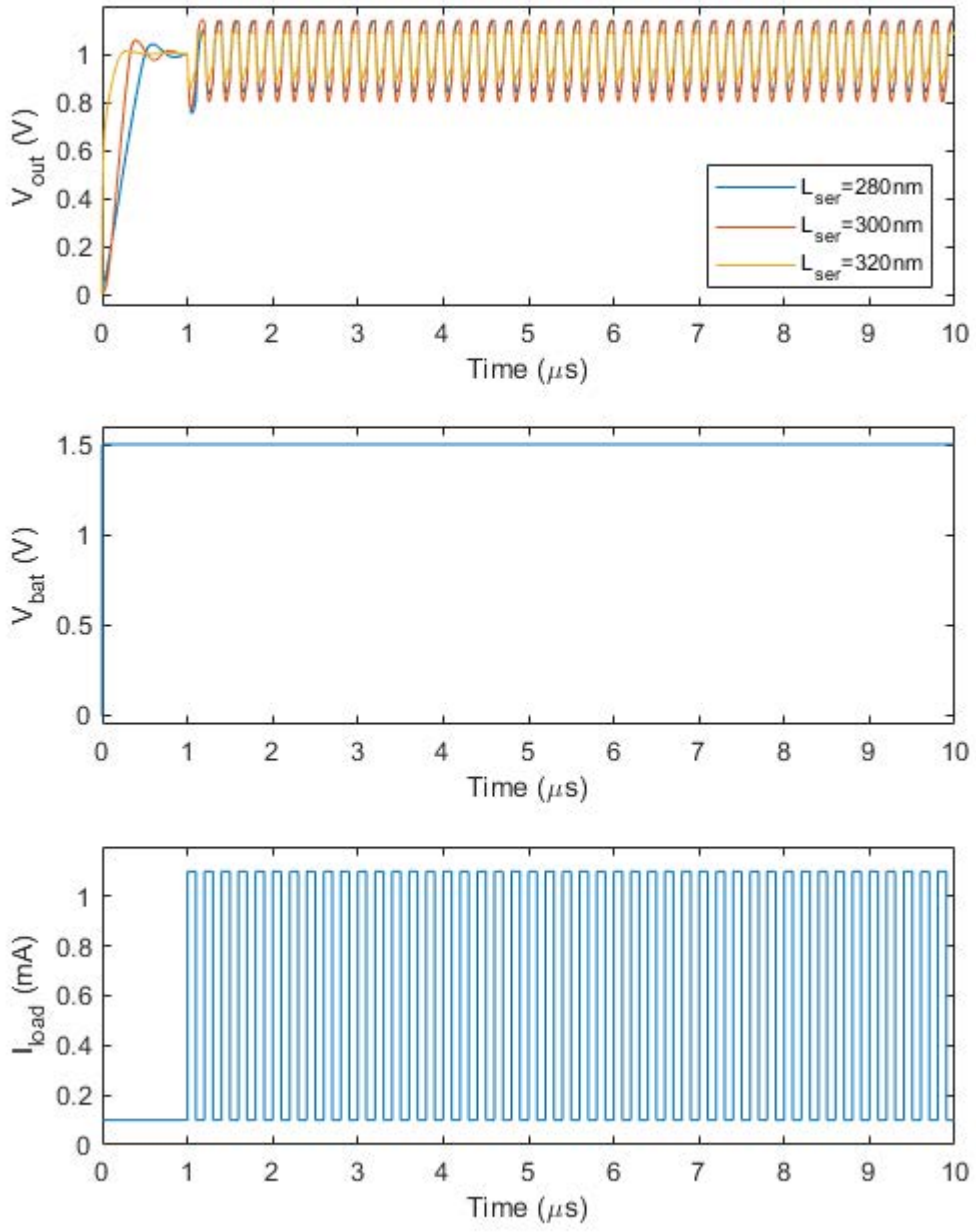


Figure 2.6: SCμM generated LDO transient for amplifier channel length of 500nm at 5MHz.

Spec	Value
ser_type	p
vdd	1.5
vout	1
iload	1e-3
iref	500e-9
iamp_max	100e-6
cload	100e-12
cdecap	100e-12
rsource	0

Spec	Value
err	1e-3
psrr	50
psrr_fbw	100
pm	60
loadreg	1e-3
load_pole	False
v_res	10e-3
sim_env	nominal[1]

Device	Base Finger Width	Channel Type	Device Type
amp_in	500nm	NMOS	Input
amp_load	500nm	PMOS	Input
amp_tail	500nm	NMOS	Input
amp_mir	500nm	NMOS	Input
ser	500nm	PMOS	Input

Table 2.6: SCµM generated LDO input parameters for amplifier channel length of 1µm.

L_{ser}	280nm	280nm		300nm		320nm	
Data Source	SCµM	Script	Sim.	Script	Sim.	Script	Sim.
Static Error	−2.756m	513.2µ	116.8µ	294.8µ	−206.1µ	233.8µ	−274.5µ
Amp Current	4.763µA	310.2nA	198.4nA	338.0nA	327.0nA	499.1nA	488.9nA
Phase Margin	85.62°	70.80°	74.33°	77.64°	74.74°	79.56°	75.7°
PSRR	43.87dB	62.27dB	65.90dB	66.86dB	82.82dB	68.37dB	87.03dB
PSRR Bandwidth	461.9Hz	1.252kHz	1.031kHz	529.2Hz	163.7Hz	404.1Hz	159.7Hz
Load Regulation	96.50µ	130.0µ	5.328µ	74.04µ	5.170µ	60.21µ	4.829µ
Amp Capacitor	3.60pF	0F	0F	0F	0F	0F	0F
Load Capacitor	452pF	0F	0F	0F	0F	0F	0F
Total Gate Area	756.6µm^2	122.9µm^2	122.9µm^2	157.1µm^2	157.1µm^2	233.0µm^2	233.0µm^2

Table 2.7: SCµM generated LDO performance for amplifier channel length of 1µm.

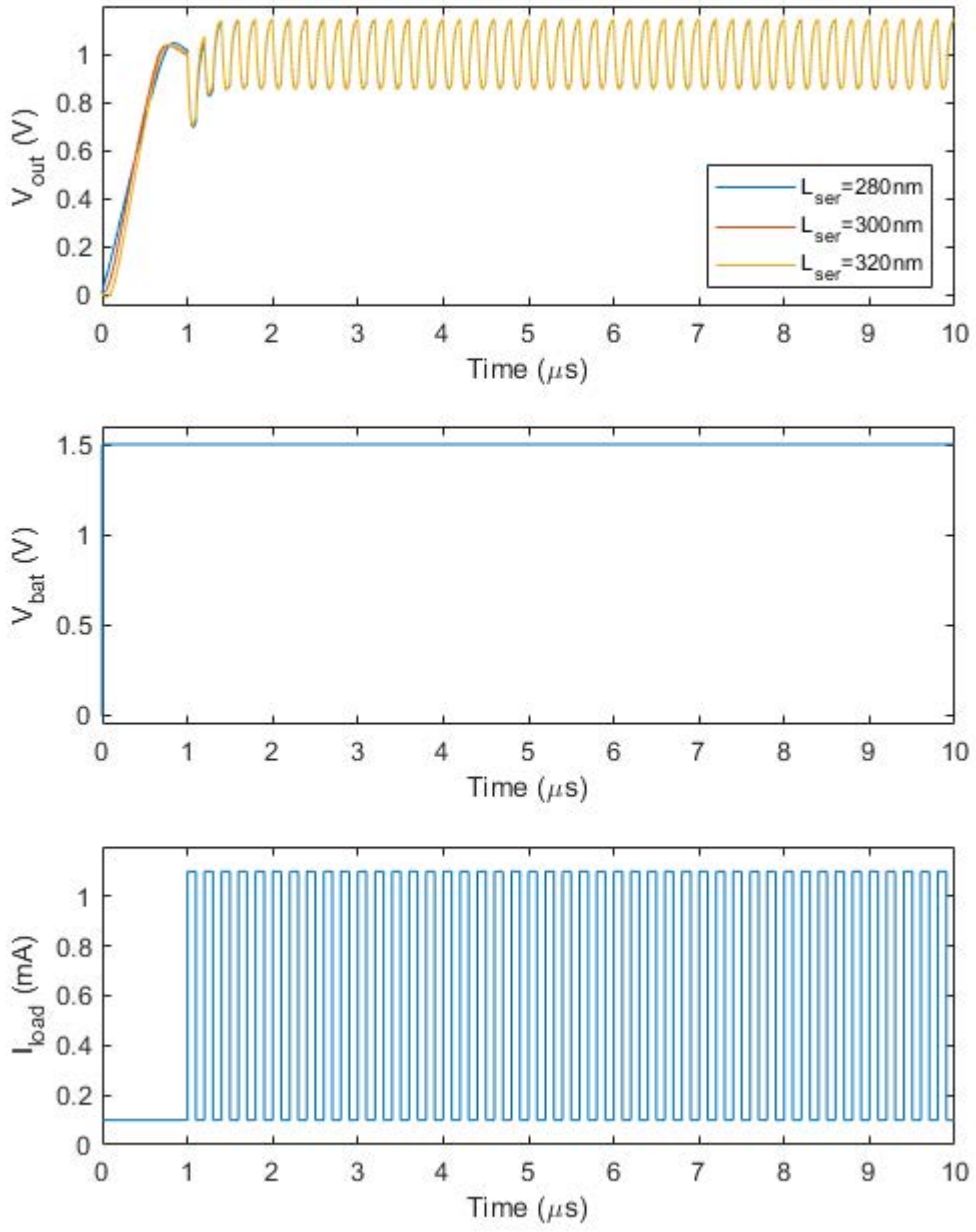


Figure 2.7: SCμM generated LDO transient for amplifier channel length of 1μm at 5MHz.

Spec	Value
ser_type	p
vdd	1.5
vout	1
iload	1e-3
iref	500e-9
iamp_max	100e-6
cload	100e-12
cdecap	100e-12
rsource	0

Spec	Value
err	1e-3
psrr	70
psrr_fbw	100
pm	60
loadreg	1e-3
load_pole	False
v_res	10e-3
sim_env	nominal[1]

Device	Base Finger Width	Channel Type	Device Type
amp_in	500nm	NMOS	Input
amp_load	500nm	PMOS	Input
amp_tail	500nm	NMOS	Input
amp_mir	500nm	NMOS	Input
ser	500nm	PMOS	Input

Table 2.8: SCµM generated LDO input parameters for amplifier channel length of 5µm.

L_{ser}	280nm	280nm		300nm		320nm	
Data Source	SCµM	Script	Sim.	Script	Sim.	Script	Sim.
Static Error	−2.756m	126.7µ	38.46µ	121.3µ	11.21µ	110.6µ	−40.47µ
Amp Current	4.763µA	314.1nA	248.1nA	638.1nA	495.8nA	990.4nA	991.1nA
Phase Margin	85.62°	70.83°	71.38°	70.36°	70.35°	69.31°	66.81°
PSRR	43.87dB	78.83dB	75.08dB	79.36dB	78.92dB	81.42dB	93.80dB
PSRR Bandwidth	461.9Hz	1.589kHz	244.6Hz	4.432kHz	199.2Hz	1.474kHz	52.13Hz
Load Regulation	96.50µ	47.49µ	2.325µ	44.67µ	2.117µ	44.67µ	2.242µ
Amp Capacitor	3.60pF	168.9fF	168.9fF	240.7fF	240.7fF	279.8fF	279.8fF
Load Capacitor	452pF	0F	0F	0F	0F	0F	0F
Total Gate Area	756.6µm^2	130.1µm^2	130.1µm^2	154.2µm^2	154.2µm^2	148.3µm^2	148.3µm^2

Table 2.9: SCµM generated LDO performance for amplifier channel length of 5µm.

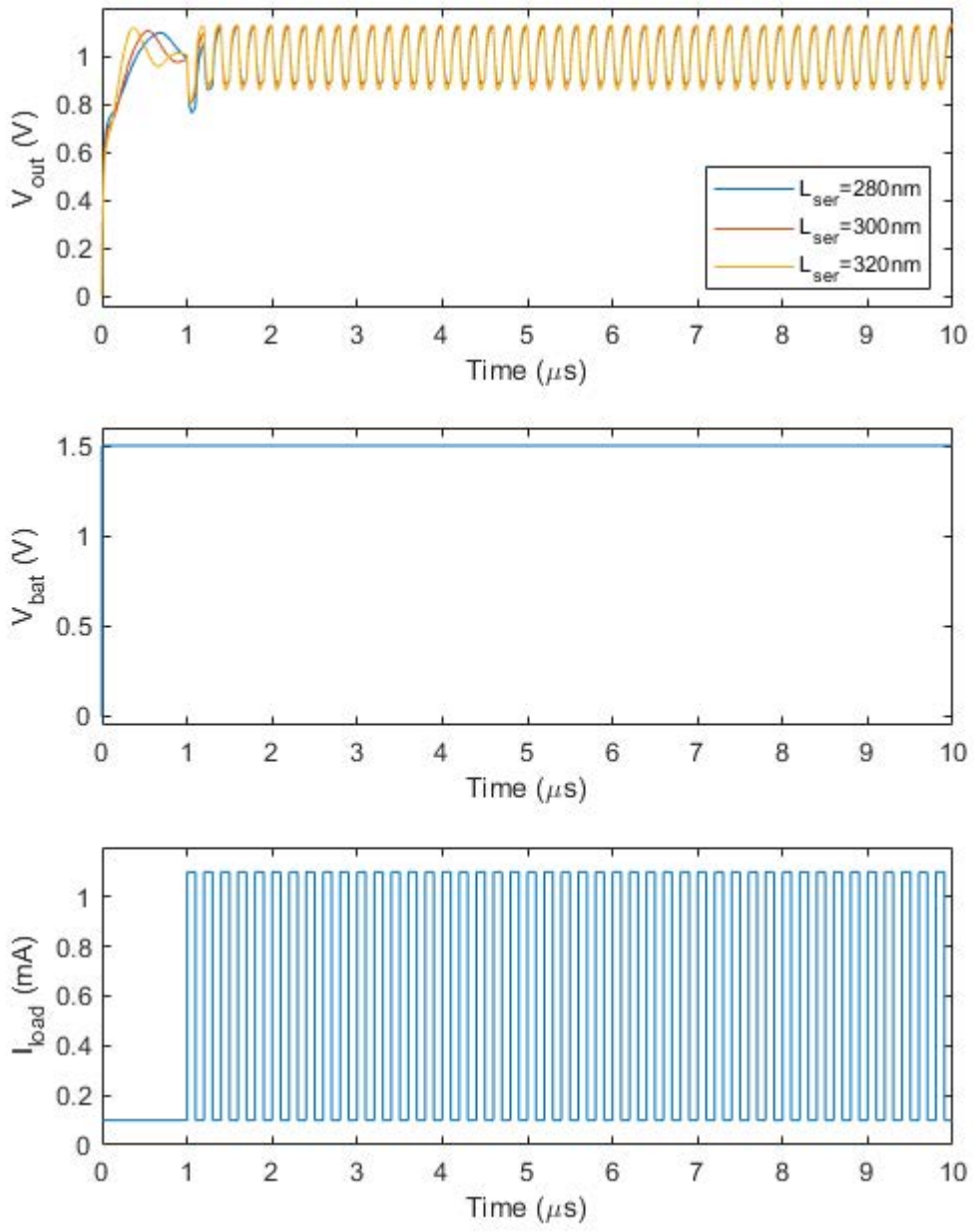


Figure 2.8: SCµM generated LDO transient for amplifier channel length of 5µm at 5MHz.

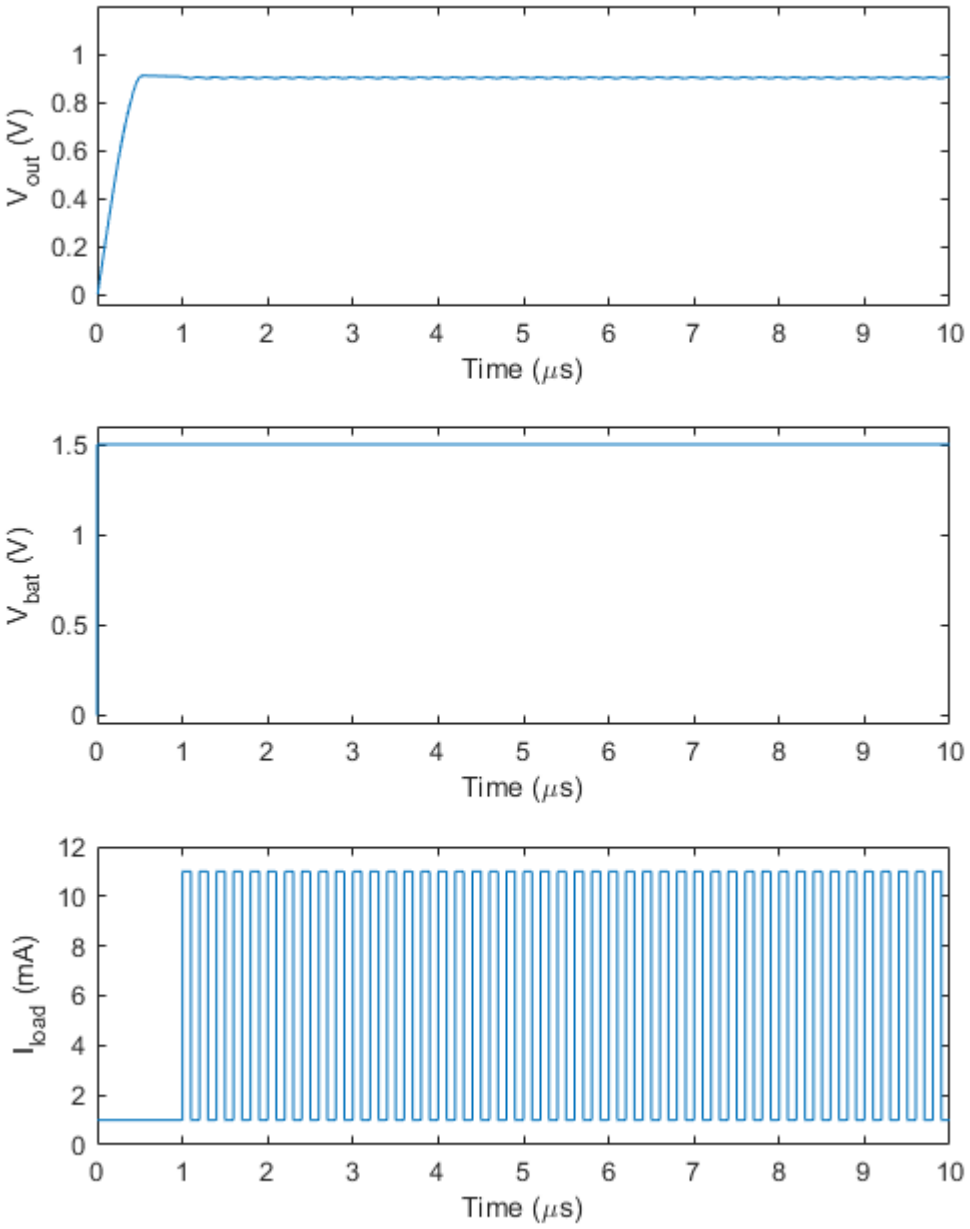


Figure 2.9: EE290C hand-designed LDO transient response at 5MHz.

	Temperature (nominal[1])			Corners ($27°C$)		nominal[1], $27°C$
Parameter	$0°C$	$27°C$	$85°C$	Minimum	Maximum	$R_{sup} = 10$
Static Error	61.93µ	−54.9µ	−184.4µ	−1.076m	923.5µ	260.8µ
Amp Current	192.5µA	196.4µA	203.8µA	176.6µA	217.7µA	192.1µA
Phase Margin	49.57°	50.54°	52.84°	47.22°	55.95°	50.35°
PSRR	52.03dB	52.47dB	62.99dB	50.17dB	59.25dB	49.89dB
PSRR Bandwidth	733.8kHz	663.0kHz	180.9kHz	316kHz	891.5kHz	877.8kHz
Load Regulation	380.5µ	408.1µ	471.5µ	329.0µ	483.1µ	488.6µ
Amp Capacitor	0F	0F	0F	0F	0F	0F
Load Capacitor	100nF	100nF	100nF	100nF	100nF	100nF
Total Gate Area	208.7µm^2	208.7µm^2	208.7µm^2	208.7µm^2	208.7µm^2	208.7µm^2

Table 2.10: EE290C LDO performance with 10mA load current.

design seems to be best. It has comparable power consumption, phase margin, and load regulation to the hand-designed LDO, with significant increases to both PSRR and PSRR bandwidth. The static error is a lot larger, but only amounts to about a 1mV offset from the nominal 900mV. The area is also slightly larger, but would still easily fit within the area bounds of the power block on the chip for EE290C. The transient responses of each generated LDO and the hand-designed LDO are shown in Figures 2.10 and 2.11. The chip for EE290C is designed to run at 100MHz, but the response at this frequency and the plot of the load current were too fast to show at the same scale as the startup transient due to the large output capacitor. The plots at 100MHz and 5MHz are virtually indistinguishable from each other, even between designs with different channel lengths. The main difference is the settling time of the startup transient due to the frequency of the secondary pole at the amplifier output. Although the difference in performance is not as dramatic as it is for the redesigned SCµM LDOs, the speed at which this script is able to generate this designs is still significantly faster than designing without it.

2.5 Approximations and Errors

The main sources of error in this script are likely from the device characterization and simplification of the model the LTICircuit functions use for calculation. The LTICircuit class assumes the canonical small signal model [4, Fig. 2.38]. Inaccuracies with this assumption produce the error seen in the design script outputs.

This script uses is the function `estimate_vth(db,is_nch,lch,vgs,vbs)` to approximate the threshold voltage for the output device, because this information is not included in the device characterization BAG uses due to the variation from linear or quadratic models in real devices. This estimation is very rough and usually inaccurate if the bias points the threshold is estimated at put the device out of saturation. To avoid this, the script estimates

Spec	Value
ser_type	p
vdd	1.5
vout	0.9
iload	10e-3
iref	10e-6
iamp_max	500e-6
cload	0
cdecap	100e-9
rsource	0

Spec	Value
err	0.01
psrr	40
psrr_fbw	500e3
pm	60
loadreg	10e-3
load_pole	True
v_res	10e-3
sim_env	nominal[1]

Device	Base Finger Width	Channel Type	Device Type
amp_in	500nm	NMOS	Input
amp_load	500nm	PMOS	Input
amp_tail	270nm	NMOS	Input
amp_mir	270nm	NMOS	Input
ser	500nm	PMOS	Input

Table 2.11: EE290C generated LDO input parameters for amplifier channel length of 400nm.

L_{ser}	150nm	150nm		180nm	
Data Source	Original	Script	Sim.	Script	Sim.
Static Error	−54.9µ	2.523m	−1.848m	1.862m	−2.123m
Amp Current	196.4µA	251.0µA	258.3µA	393.2µA	405.4µA
Phase Margin	50.54°	65.70°	63.07°	64.48°	62.49°
PSRR	52.47dB	59.56dB	63.81dB	60.85dB	84.72dB
PSRR Bandwidth	663.0kHz	525.8kHz	436.7kHz	508.8kHz	81.15kHz
Load Regulation	408.1µ	774.9µ	432.5µ	694.1µ	400.3µ
Amp Capacitor	0F	0F	0F	0F	0F
Load Capacitor	100nF	100nF	100nF	100nF	100nF
Total Gate Area	208.7µm^2	185.4µm^2	185.4µm^2	295.9µm^2	295.9µm^2

Table 2.12: EE290C generated LDO performance for amplifier channel length of 400nm.

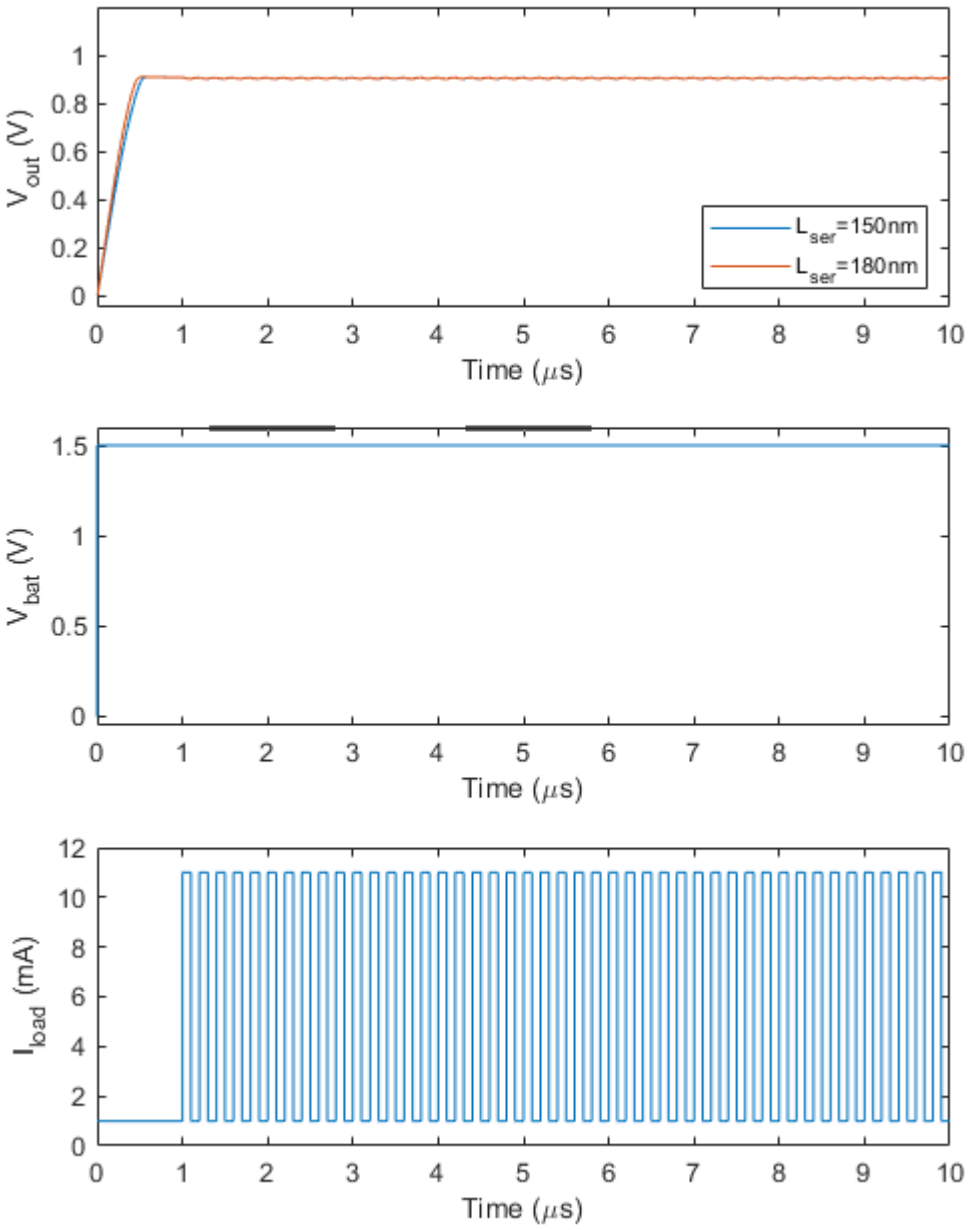


Figure 2.10: EE290C generated LDO transient for amplifier channel length of 400nm at 5MHz.

Spec	Value
ser_type	p
vdd	1.5
vout	0.9
iload	10e-3
iref	10e-6
iamp_max	500e-6
cload	0
cdecap	100e-9
rsource	0

Spec	Value
err	0.01
psrr	60
psrr_fbw	1e6
pm	60
loadreg	10e-3
load_pole	True
v_res	10e-3
sim_env	nominal[1]

Device	Base Finger Width	Channel Type	Device Type
amp_in	500nm	NMOS	Input
amp_load	500nm	PMOS	Input
amp_tail	270nm	NMOS	Input
amp_mir	270nm	NMOS	Input
ser	500nm	PMOS	Input

Table 2.13: EE290C generated LDO input parameters for amplifier channel length of 1µm.

L_{ser}	150nm	150nm		180nm	
Data Source	Original	Script	Sim.	Script	Sim.
Static Error	−54.9µ	2.225m	1.262m	1.539m	489.0µ
Amp Current	196.4µA	192.9µA	205.9µA	281.9µA	294.5uµA
Phase Margin	50.54°	66.36°	54.90°	64.67°	54.24°
PSRR	52.47dB	62.50dB	51.72dB	64.46dB	55.61dB
PSRR Bandwidth	663.0kHz	1.039MHz	1.877MHz	1.080MHz	1.434MHz
Load Regulation	408.1µ	963.9µ	408.8µ	958.4µ	393.6µ
Amp Capacitor	0F	0F	0F	0F	0F
Load Capacitor	100nF	100nF	100nF	100nF	100nF
Total Gate Area	208.7µm^2	235.0µm^2	235.0µm^2	302.7µm^2	302.7µm^2

Table 2.14: EE290C generated LDO performance for amplifier channel length of 1µm.

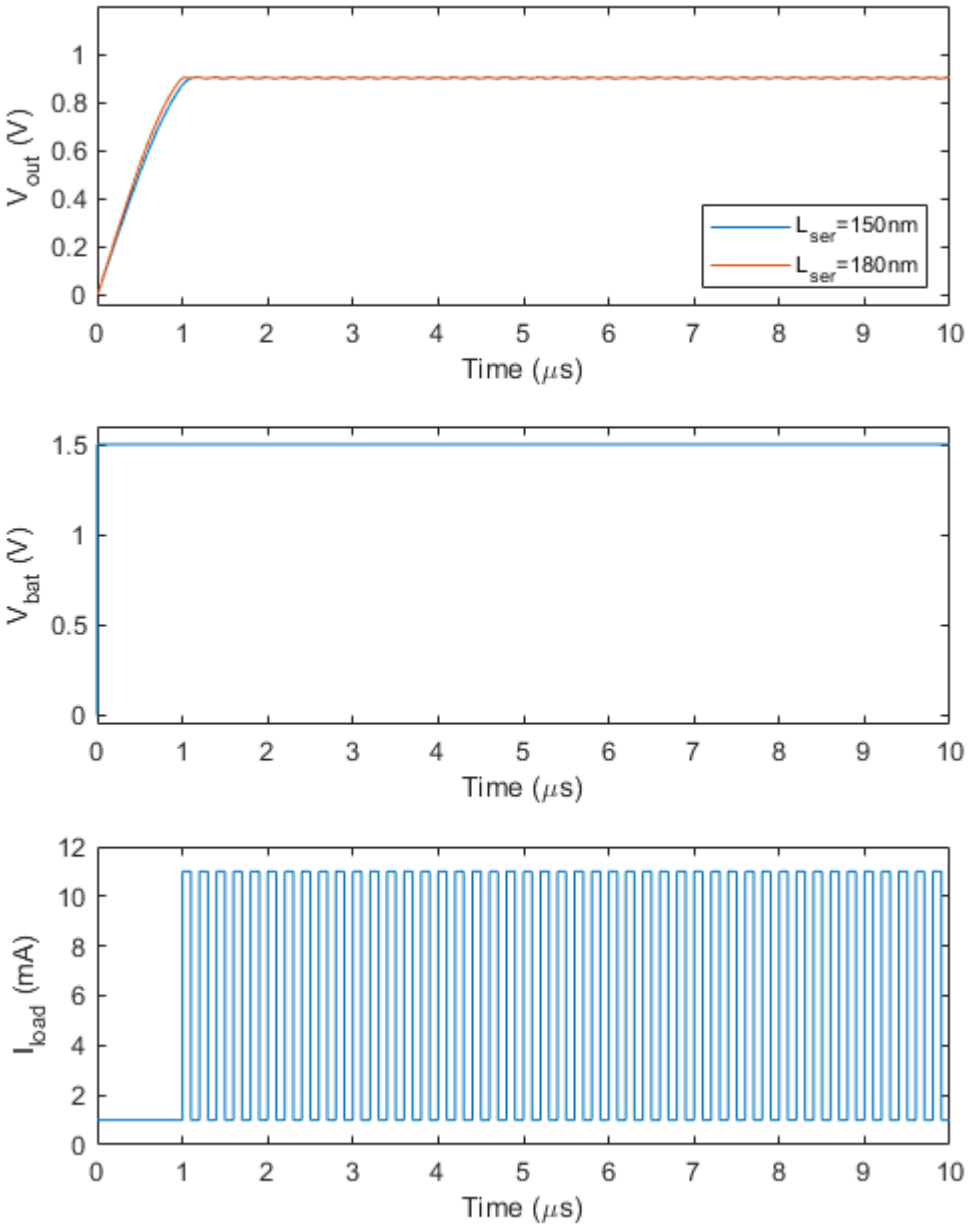


Figure 2.11: EE290C generated LDO transient for amplifier channel length of 1µm at 5MHz.

the threshold of the series device with its defined drain-source voltage (between the supply voltage and the output voltage) with the gate source voltage likely overestimated at half the supply voltage.

The static error is currently measured as the error caused by the non-infinite loop gain as shown in Equation 2.4. As mentioned briefly in Section 2.2, the static error calculated by this design script will likely be dominated by device offsets in the amplifier, which is not taken into account.

The PSRR is also approximated as the inverse of the gain from the power supply to the output rather than the gain from the input to the output divided by the power supply gain as described in Equation 2.5. The PSRR bandwidth would require the convolution of two lists of coefficients for both the numerator and denominator. For the purposes of this script, because the gain from the reference to the output is approximately 1 for a high loop gain, it was omitted from this calculation.

The YAML file that the specs of a completed generation are dumped into appears to have trouble with numbers such as the capacitances or device finger widths that are NumPy data types, but the data is stored properly even though the formatting is not as readable as a basic Python integer or float type is. The results are directly printed to that terminal, and if lost, can easily be generated again in very little time.

Chapter 3

Conclusion

This report details the approach of a BAG design script to generate LDOs with real applications on chips like SCμM. Although there are improvements to be made on this script, as it stands it is still an incredibly efficient and practical tool for LDO design. Using only the most basic LDO topology, design can be done quickly and is easily transferable between processes, allowing for fast estimates of performance and comparisons between designs, optimization goals, and technologies. With known specs for each LDO on SCμM, it is possible to redesign lower power LDOs for each of the eight domains listed in Table 1.2 in under a day, potentially increasing their accuracy and noise reduction as well. The ability to design so many circuits in so little time is invaluable to so many projects, especially in an academic setting, allowing designers to focus more of their energy on the circuits powered by LDOs. With the necessity for power regulation on any chip, this script will undoubtedly see use in the future.

3.1 Future Work

This design script does not check for the amplifier device thresholds, assuming that if the series device is in saturation the amplifier load will be as well, and then the V* of the other amplifier devices will be set to match the load pair keeping them in saturation if they have similar V* characteristics in the different regions of operation. If the channel length for the amplifier devices is significantly larger than the series device length, the amplifier load threshold will exceed the estimate for the series device and put all amplifier device gate voltages subthreshold. If the script did include threshold estimates for the amplifier devices as well using the function `estimate_vth(db,is_nch,lch,vgs,vbs)`, many of the tests run in Section 2.5 may not produce a solution due to the inaccuracy of the estimation function. A near-threshold biasing point improves the gain and power consumption of the amplifier significantly, and the overestimation of the threshold voltage may push devices too far from that point to meet more demanding specs. Each device that was operating subthreshold in Section 2.5 was within 100mV of the actual device threshold. Implementing a more accurate

threshold estimation or even extraction during characterization for BAG would significantly improve the performance of this design script and likely others as well.

Other features this script does not include but could be beneficial for any updated versions of this script are the frequency response of the load regulation, static error caused by device offsets, and the option of increasing the width of the amplifier devices for a pole at the amplifier output if the phase margin spec is met but the PSRR bandwidth spec is not. For the load regulation at higher frequencies, there may be peaking if the load capacitance is small, which depending on the operating frequency of the load circuits, could mean a significantly worse load regulation. This issue occurs most often if the dominant pole of the LDO is at the amplifier output. Adding estimates of the device offset contribution to static error will produce more accurate results in comparison to simulation, particularly at higher loops gains where Equation 2.4 is small. The modification to the amplifier device width for pole placement is only useful if the phase margin of a design is already higher than the spec without additional decap added, and if the PSRR spec is not met. Increasing the amplifier width increases the amplifier pole frequency, bringing the phase margin down and PSRR bandwidth up at the cost of power. It is possible that moving the pole to the output of the LDO will produce a design that meets both specs anyway and uses less power at the same time, but this may be a worthwhile modification if area for capacitors is limited and a higher PSRR bandwidth is necessary.

The final area in which this script could be expanded upon is automating layout. Approximating the area as the device gate area is easy to do by multiplying device widths and lengths. Some scripts do have the infrastructure to generate a layout from the produced design [5], which would be helpful in situations like with EE290C where the area allowed for the LDOs changed multiple times during the design process. Due to lack of setup in the processes used in this report and time, this script does not contain layout functionality, but adding it to this script will unquestionably speed up the design process further.

Appendix D describes additional issues with the power systems on SCµM that have not been resolved to date and possible routes to finding a solution.

Bibliography

[1] Eric Chang et al. "BAG2: A process-portable framework for generator-based AMS circuit design". In: *2018 IEEE Custom Integrated Circuits Conference (CICC)*. 2018, pp. 1–8. DOI: 10.1109/CICC.2018.8357061.

[2] Filip Maksimovic. "Monolithic Wireless Transceiver Design". PhD thesis. EECS Department, University of California, Berkeley, May 2020, pp. 64–65. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-33.html.

[3] Alex Moreno et al. "Single-Chip micro-Mote for Microrobotic Platforms". In: *Government Microcircuit Applications & Critical Technology Conference*. GOMACTech, 2020.

[4] Behzad Razavi. *Design of Analog CMOS Integrated Circuits*. first. McGraw-Hill, 2001, p. 36.

[5] Nicholas Werblun. "Closing the Analog Design Loop with the Berkeley Analog Generator". MS. EECS Department, University of California, Berkeley, May 2019. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-23.html.

Appendix A

BAG Setup

Workspace Repository

To get the files needed to run BAG in the TSMC 65nm LP process, clone the repository at `https://bwrcrepo.eecs.berkeley.edu/tsmc65/tsmc65lp/bag2_tsmc65lp_workspace` by running in a terminal:

```
$ git clone git@bwrcrepo.eecs.berkeley.edu:tsmc65/tsmc65lp/
   bag2_tsmc65lp_workspace
```

To get the files needed to run BAG in the TSMC 28nm process, clone the repository at `https://bwrcrepo.eecs.berkeley.edu/tstech28/bag2_tsmc28_workspace` by running in a terminal:

```
$ git clone git@bwrcrepo.eecs.berkeley.edu:tstech28/
   bag2_tsmc28_workspace
$ git checkout ee194_290c
```

Switching to the `ee194_290c` branch is necessary for TSMC 28nm to avoid an error with submodule paths missing in setup. After cloning the repository, add submodules in the BAG workspace directory with the following commands:

```
$ git submodule add https://github.com/PisterLab/span_ion
   span_ion_proj
$ git submodule add https://github.com/PisterLab/bag2_analog
$ git submodule add https://github.com/PisterLab/bag2_digital
$ git submodule add https://github.com/PisterLab/
   bag2_wrappers
$ git submodule init
$ git submodule update --recursive
```

The `cds.lib` for both technologies will need to be modified so it contains:

```
INCLUDE cds.lib.bag
INCLUDE cds.lib.core
DEFINE BAG_prim $BAG_WORK_DIR/BAG_prim
```

These will provide Cadence Virtuoso a path to the respective technology libraries and BAG frameworks. The final line redefines a library path in `cds.lib.bag` to point to the work directory instead of the technology configuration directory.

For TSMC 28nm, it will also be necessary to copy the directory `scripts_char` from the TSMC 65nm LP workspace, because it does not exist in the 28nm repository. Additionally, in `bag_libs.def`, remove the line:

```
serdes_digitalbase_templates $BAG_WORK_DIR/serdes_digitalbase
   /BagModules
```

It is also necessary to copy `bag_testbenches` from the TSMC 65nm LP workspace into the TSMC 28nm workspace so BAG sees that the correct directory name exists (i.e. without the extension `_ec`). In `cds.lib.bag` add the line:

```
DEFINE bag_testbenches $BAG_WORK_DIR/bag_testbenches/
   bag_testbenches
```

In `bag_libs.def` add the lines:

```
bag_testbenches $BAG_WORK_DIR/bag_testbenches/BagModules
bag2_analog $BAG_WORK_DIR/bag2_analog/BagModules
span_ion $BAG_WORK_DIR/span_ion_proj/BagModules
```

Finally, in `bag_startup.py`, change the line:

```
sys.path.append(os.path.join(os.environ['BAG_WORK_DIR'], '
   bag_testbenches_ec'))
```

to the following:

```
sys.path.append(os.path.join(os.environ['BAG_WORK_DIR'], '
   bag_testbenches'))
```

To get the LDO design script, example input spec file, and another file `dsn_cell.py` necessary to run the design script, also clone `https://github.com/PisterLab/bag2_analog`. Copy the directory `scripts_dsn` from this new into the main BAG workspace folder, and move `dsn_cell.py` into `BAG_framework/run_scripts` inside the workspace. The final hierarchy of these added files should be:

```
bag2_tsmc<process>_workspace
├── BAG_framework
│   └── run_scripts
│       └── dsn_cell.py
└── scripts_dsn
    ├── ldo_params.yaml
    └── regulator_ldo_series.py
```

Create a symbolic link to `scripts_dsn` in the main workspace directory with the command:

```
$ ln -s BAG_framework/run_scripts/dsn_cell.py ./
```

BAG provides a framework for storing and accessing process characterization data for use in design scripts. To run characterization on a transistor, a YAML file such as the example in Appendix B will be necessary. Process-sensitive information has been removed from this example, so in addition to changing any transistor dimensions appropriately, the contents of the `env_list` list will have to be changed from 'nominal' to the appropriate corner names, and any time 'th_type' appears it should be replaced with the name of the intent as described later in the Primitive Setup subsection. To characterize a PMOS device, change every instance of 'nch' to 'pch' and set `is_nmos` to 'False.' This file is read by BAG to

set the testbench parameters for characterization. Once the YAML file and BAG workspace are setup, the process of characterization in the next subsection can begin.

All setup before this point only needs to be run once. The next subsections about characterization and primitive setup only have to be run once for each transistor setting or device type, but must be rerun if any settings change. When changing settings, it is also helpful to set a different filepath for the characterization data to be written to so previously characterized transistor data is still accessible without rerunning characterization.

Process Characterization

To run characterization once the BAG workspace is set up and a YAML file is configured for the appropriate transistor (examples in `specs_mos_char/` in the 65nm workspace), start Virtuoso from a C shell by running:

```
$ source .cshrc
$ virtuoso &
```

If using Bash instead of a C shell, replace `source .cshrc` with:

```
$ source .bashrc
```

Once Virtuoso opens, run the following command from the Virtuoso Log window (not the terminal):

```
load("start_bag.il")
```

To start the BAG session, run the following command from the terminal in the BAG workspace directory Virtuoso was launched in:

```
$ ./start_bag.sh
```

This will begin a terminal-based IPython session. From here, to run characterization:

```
$ run -i scripts_char/nmos_char.py
$ run -i scripts_char/pmos_char.py
```

Each command will run the characterization in the respective YAML files pointed to in `nmos_char.py` and `pmos_char.py`. The `config_file` path in these scripts can be modified to point to the proper characterization input YAML file.

If a testbench for a particular type of transistor already exists in the testbench libraries BAG creates called `AAAFOO_MOSCHAR_NCH` or `AAAFOO_MOSCHAR_PCH`, the characterization script may raise an error that it "cannot load adexl database" if Cadence has locked the ADE XL. To fix this error and run the characterization, delete the existing testbench library, close Virtuoso, and exit the BAG session in the terminal. Restart Virtuoso and BAG using the process described above and run the characterization as before. The characterization should run to completion.

Primitive Setup

If a particular transistor does not have an existing cell in the library `BAG_Prim`, a new cell can be made with drain, source, gate, and body contacts connected to input/output pins labelled `D`, `S`, `G`, and `B` respectively, and transistor length, width, and number of fingers set at `pPar("l")`, `pPar("w")`, and `pPar("nf")` respectively. The name of this cell should be `nmos4_[type]` or `pmos4_[type]` where '`[type]`' is the string in the characterization parameters YAML file used to direct the 'intent' to the correct cell, such as '`lvt`' or '`hvt`.'

Another error that may appear during characterization is "KeyError: 'vgs,'" which occurs if the device width or length is out of the bounds required by the process.

Running a Design Script

To run a design script, from the BAG session run:

```
$ run dsn_cell.py spec_dir/spec.yaml -dump spec_dir/result.
   yaml
```

where `spec_dir/spec.yaml` is the filepath to the YAML file with the design script to use and parameters to run it with such as the example in Appendix C (which is scrubbed of process-sensitive information), and `spec_dir/result.yaml` is the filepath to the YAML file to write the output of the script to.

If changes have been made to any file in the BAG workspace, it may be necessary to exit the BAG session and restart it as described in the Process Characterization subsection above. In some cases, it may be necessary to also close Virtuoso, then rerun all startup commands, also as described in the Process Characterization subsection above.

Appendix B

Example Characterization Input YAML File

```yaml
dut_lib: 'bag_testbenches'
dut_cell: 'mos_analogbase'
layout_package: 'abs_templates.mos_char'
layout_class: 'Transistor'

impl_lib: 'AAAFOO_MOSCHAR_NCH'
dsn_basename: 'NCH'

rcx_params:
  capacitance:
    ground_net: b
view_name: 'schematic'

root_dir: 'data/nch_scrubbed'
summary_fname: 'summary.yaml'

routing_grid:
  layers: [1, 2, 3, 4, 5, 6, 7]
  widths: [0.080, 0.080, 0.080, 0.080, 0.080, 0.080, 0.080]
  spaces: [0.080, 0.080, 0.080, 0.080, 0.080, 0.080, 0.080]
  bot_dir: 'y'

sweep_params:
  intent: ['th_type']

layout_params:
  mos_type: 'nch'
  lch: 1.0e-6
  w: 0.5e-6
  fg: 20
  intent: 'th_type'
  fg_dum: 4
  stack: 1
  ptap_w: 0.5e-6
  ntap_w: 0.5e-6
  tr_w_dict:
    g: 1
    d: 2
    s: 2
  tr_sp_dict:
    gs: 1
    gd: 1
    sb: 1
    db: 1
```

```
45
46 # Used only if we're doing schematic-only simulation
47 schematic_params:
48   mos_type: 'nch'
49   lch: 1.0e-6
50   w: 0.5e-6
51   fg: 20
52   intent: 'th_type'
53   stack: 1
54   dum_info: !!null
55
56 dut_wrappers: []
57
58 env_list: ['nominal']
59
60 measurements:
61   - meas_type: 'mos_ss'
62     meas_package: 'verification.mos.sim'
63     meas_class: 'MOSCharSS'
64     out_fname: 'mos_ss.yaml'
65     is_nmos: True
66     fg: 20
67     testbenches:
68       ibias:
69         tb_package: 'verification.mos.sim'
70         tb_class: 'MOSIdTB'
71         tb_lib: 'bag_testbenches'
72         tb_cell: 'mos_tb_ibias'
73         sch_params: {}
74         wrapper_type: ''
75         vgs_num: 200
76         vgs_max: 1.0
77         ibias_min_fg: 1.0e-9
78         ibias_max_fg: 200.0e-6
79         vgs_resolution: 2.0e-3
80       sp:
81         tb_package: 'verification.mos.sim'
82         tb_class: 'MOSSPTB'
83         tb_lib: 'bag_testbenches'
84         tb_cell: 'mos_tb_sp'
85         sch_params: {}
86         wrapper_type: ''
87         vgs_num: 30
88         vds_num: 20
```

```
89          vds_min: 5.0e-3
90          vds_max: 1.0
91          vbs: [0.0, 0.15, 0.3, 0.45]
92          sp_freq: 1.0e+6
93          cfit_method: 'average'
```

Appendix C

Example Design Script Input YAML File

```yaml
dsn_mod: scripts_dsn.regulator_ldo_series
dsn_cls: bag2_analog__regulator_ldo_series_dsn

params:
  ser_type: p
  vdd: !!float 1.5
  vout: !!float 1
  iload: !!float 1e-3
  iref: !!float 1e-6
  iamp_max: !!float 100e-6
  cload: !!float 1e-9
  cdecap: !!float 1e-9
  rsource: !!float 0
  err: !!float 1e-3
  psrr: !!float 40
  psrr_fbw: !!float 1e3
  pm: !!float 60
  loadreg: !!float 1e-3
  load_pole: False
  v_res: !!float 10e-3

  specfile_dict:
    amp_in: specs_mos_char/nch.yaml
    amp_load: specs_mos_char/pch.yaml
    amp_tail: specs_mos_char/nch.yaml
    amp_mir: specs_mos_char/nch.yaml
    ser: specs_mos_char/pch.yaml
  th_dict:
    amp_in: th_type
    amp_load: th_type
    amp_tail: th_type
    amp_mir: th_type
    ser: th_type
  l_dict:
    amp_in: !!float 500e-9
    amp_load: !!float 500e-9
    amp_tail: !!float 500e-9
    amp_mir: !!float 500e-9
    ser: !!float 500e-9
  sim_env: nominal
```

Appendix D

LDO Design Script

```
# -*- coding: utf-8 -*-

from typing import Mapping, Tuple, Any, List

import os
import pkg_resources
import numpy as np
import warnings
from pprint import pprint

from bag.design.module import Module
from bag.core import BagProject
from span_ion_proj.scripts_dsn import DesignModule, get_mos_db, estimate_vth,
    parallel, verify_ratio, num_den_add, enable_print, disable_print
from bag.data.lti import LTICircuit, get_w_3db, get_stability_margins

# noinspection PyPep8Naming
class bag2_analog__regulator_ldo_series_dsn(DesignModule):
    """Module for library bag2_analog cell regulator_ldo_series
    Fill in high level description here.
    """

    @classmethod
    def get_op_info(cls) -> Mapping[str,str]:
        # type: () -> Dict[str, str]
        """Returns a dictionary from parameter names to descriptions.
        Returns
        -------
        param_info : Optional[Dict[str, str]]
            dictionary from parameter names to descriptions.
        """
        ans = super().get_op_info()
        ans.update(dict(
            specfile_dict = 'Transistor database spec file names for each device'
    ,
            ser_type = 'n or p for type of series device',
            th_dict = 'Transistor flavor dictionary.',
            l_dict = 'Transistor channel length dictionary',
            sim_env = 'Simulation environment',
            vdd = 'Supply voltage in volts.',
            vout = 'Reference voltage to regulate the output to',
            loadreg = 'Maximum absolute change in output voltage given change in
    output current',
            iload = 'Bias current of series device, in amperes.',
```

```
42            iref = 'Reference current for amplifier biasing, in amperes',
43            iamp_max = 'Maximum amplifier current, in amperes',
44            cload = 'Load capacitance from the output of the LDO to ground',
45            cdecap = 'Maximum additional capacitance added to circuit',
46            rsource = 'Resistance from the power supply, in ohms',
47            err = 'Maximum percent static error at output (as decimal)',
48            psrr = 'Minimum power supply rejection ratio (dB, 20*log10(dVdd/dVout
    ))',
49            psrr_fbw = 'Minimum bandwidth for power supply rejection roll-off',
50            pm = 'Minimum phase margin for the large feedback loop, in degrees',
51            load_pole = 'True to ensure dominant pole is at theregulator output',
52            v_res = 'Resolution of voltage bias point sweeps, in volts'
53        ))
54        return ans
55
56    def resize_op(self, op, wm):
57        op_new = dict()
58        for key,value in op.items():
59            if key[0] != 'v':
60                op_new[key] = wm*value
61            else:
62                op_new[key] = value
63        return op_new
64
65    def dsn_fet(self, **params):
66        specfile_dict = params['specfile_dict']
67        ser_type = params['ser_type']
68        th_dict = params['th_dict']
69        sim_env = params['sim_env']
70
71        db_dict = {k:get_mos_db(spec_file=specfile_dict[k],
72                                intent=th_dict[k],
73                                sim_env=sim_env) for k in specfile_dict.keys()}
74
75        vdd = params['vdd']
76        vout = params['vout']
77        vg = params['vg']
78        iload = params['iload']
79
80        vs = vout if ser_type == 'n' else vdd
81        vd = vdd if ser_type == 'n' else vout
82        vb = 0 if ser_type == 'n' else vdd
83
84        ser_op = db_dict['ser'].query(vgs=vg-vs, vds=vd-vs, vbs=vb-vs)
```

```
85          nf = int(round(iload/ser_op['ibias']))
86          m = iload/(2*ser_op['ibias'])
87          wm = (m%1 + 1)
88          nf = 2*int(m)
89          ser_op = self.resize_op(ser_op,wm)
90          return m > 1, dict(nf=nf, wm=wm, op=ser_op)
91
92      def dsn_amp(self, **params):
93          vdd = params['vdd']
94          vincm = params['vout']
95          voutcm = params['voutcm']
96          iload = params['iload']
97          iref = params['iref']
98          iamp_max = params['iamp_max']
99          cload = params['cload']
100         cdecap_max = params['cdecap']
101         rsource = params['rsource']
102         err_max = params['err']
103         psrr_min = params['psrr']
104         psrr_fbw_min = params['psrr_fbw']
105         pm_min = params['pm']
106         loadreg_max = params['loadreg']
107         load_pole = params['load_pole']
108         v_res = params['v_res']
109         ser_type = params['ser_type']
110         ser_info = params['ser_info']
111         db_dict = params['db_dict']
112         amp_in = 'n'
113
114         # Get amplifier load pair parameters
115         op_load = db_dict['amp_load'].query(vgs=-(vdd-voutcm), vds=-(vdd-voutcm),
        vbs=0)
116         Vstar_load = op_load['vstar']
117
118         amp_dsn_info = dict()
119
120         # Choose amp bias voltages
121         Vstar_in_err = float('inf')
122         vtail = 0
123         op_in = dict()
124         for vtail_i in np.arange(0,min(voutcm,vincm),v_res):
125             op_in_i = db_dict['amp_in'].query(vgs=vincm-vtail_i, vds=voutcm-
    vtail_i, vbs=-vtail_i)
126             Vstar_in_i = op_in_i['vstar']
```

```
127             if Vstar_in_err > abs(Vstar_load-Vstar_in_i) and op_in_i['ibias'] >
    0:
128                 Vstar_in_err = abs(Vstar_load-Vstar_in_i)
129                 vtail = vtail_i
130                 op_in = op_in_i
131                 Vstar_in = Vstar_in_i
132
133         Vstar_tail_errsq = float('inf')
134         vgtail = 0
135         op_tail = dict()
136         for vgtail_i in np.arange(0,vdd,v_res):
137             op_tail_i = db_dict['amp_tail'].query(vgs=vgtail_i, vds=vtail, vbs=0)
138             Vstar_tail_i = op_tail_i['vstar']
139             if Vstar_tail_errsq > abs(Vstar_load-Vstar_tail_i)**2+abs(Vstar_in-
    Vstar_tail_i)**2 and op_tail_i['ibias'] > 0:
140                 Vstar_tail_errsq = abs(Vstar_load-Vstar_tail_i)**2+abs(Vstar_in-
    Vstar_tail_i)**2
141                 vgtail = vgtail_i
142                 op_tail = op_tail_i
143                 Vstar_tail = Vstar_tail_i
144
145         # Size reference current mirror
146         op_mir = db_dict['amp_mir'].query(vgs=vgtail, vds=vgtail, vbs=0)
147         m_mir = iref/(2*op_mir['ibias'])
148         wm_mir = (m_mir%1 + 1)
149         nf_mir = 2*int(m_mir)
150         if nf_mir == 0:
151             return False, amp_dsn_info
152
153         # Size amplifier devices and base current
154         Id_tail = wm_mir*op_tail['ibias']
155         wm_tail = wm_mir
156         nf_tail = int(2*max(((2*op_load['ibias'])//Id_tail)+1,((2*op_in['ibias'])
    //Id_tail)+1))
157         if Id_tail*nf_tail > iamp_max:
158             return False, amp_dsn_info
159
160         Id_load = Id_tail*nf_tail/2
161         m_load = Id_load/op_load['ibias']
162         wm_load = (m_load/2)%1 + 1
163         nf_load = 2*int(m_load/2)
164
165         m_in = Id_load/op_in['ibias']
166         wm_in = (m_in/2)%1 + 1
```

```
167         nf_in = 2*int(m_in/2)
168 
169         # Resize op and format parameters
170         op_dict = {'amp_in' : self.resize_op(op_in, wm_in),
171                    'amp_tail' : self.resize_op(op_tail, wm_tail),
172                    'amp_load' : self.resize_op(op_load, wm_tail),
173                    'amp_mir' : self.resize_op(op_mir, wm_mir),
174                    'ser' : ser_info['op']}
175         nf_dict = {'amp_in' : nf_in,
176                    'amp_tail' : nf_tail,
177                    'amp_load' : nf_load,
178                    'amp_mir' : nf_mir,
179                    'ser' : ser_info['nf']}
180         wm_dict = {'amp_in' : wm_in,
181                    'amp_tail' : wm_tail,
182                    'amp_load' : wm_load,
183                    'amp_mir' : wm_mir,
184                    'ser' : ser_info['wm']}
185 
186         A = abs(self._get_loopgain_lti(op_dict, nf_dict, ser_type, amp_in,
    rsource))
187         dc_err = 1/(A+1)
188         loadreg = self._get_loadreg_lti(op_dict, nf_dict, ser_type, amp_in, cload
    , 0, rsource, vincm, iload)
189         psrr, psrr_fbw = self._get_psrr_lti(op_dict, nf_dict, ser_type, amp_in,
    cload, 0, rsource)
190         pm = self._get_stb_lti(op_dict, nf_dict, ser_type, amp_in, cload, 0,
    rsource)
191         if pm > pm_min and psrr > psrr_min and psrr_fbw > psrr_fbw_min and
    loadreg < loadreg_max and dc_err < err_max and Id_tail*nf_tail < iamp_max and
    not load_pole:
192             amp_dsn_info.update(dict(op_dict=op_dict,nf_dict=nf_dict,wm_dict=
    wm_dict))
193             amp_dsn_info.update(cap_dict=dict(cdecap_amp=0, cdecap_load=0))
194             amp_dsn_info.update(dict(loadreg=loadreg, psrr=psrr, psrr_fbw=
    psrr_fbw, pm=pm, err=dc_err, ibias=Id_tail*nf_tail))
195             return True, amp_dsn_info
196         if psrr_fbw > psrr_fbw_min and Id_tail*nf_tail < iamp_max and not
    load_pole:
197             # Find minimum decap necessary with dominant amplifier pole
198             cdecap_min = ser_info['op']['cgg']*ser_info['nf']
199             for cdecap_amp in np.logspace(np.log10(cdecap_min),np.log10(
    cdecap_max),100):
200                 loadreg = self._get_loadreg_lti(op_dict, nf_dict, ser_type,
```

```
    amp_in, cload, cdecap_amp, rsource, vincm, iload)
201             psrr, psrr_fbw = self._get_psrr_lti(op_dict, nf_dict, ser_type,
    amp_in, cload, 0, rsource)
202             pm = self._get_stb_lti(op_dict, nf_dict, ser_type, amp_in, cload,
     cdecap_amp, rsource)
203             if psrr_fbw < psrr_fbw_min:
204                 break
205             if pm > pm_min and psrr > psrr_min and psrr_fbw > psrr_fbw_min
    and loadreg < loadreg_max and dc_err < err_max:
206                 amp_dsn_info.update(dict(op_dict=op_dict,nf_dict=nf_dict,
    wm_dict=wm_dict))
207                 amp_dsn_info.update(cap_dict=dict(cdecap_amp=cdecap_amp,
    cdecap_load=0))
208                 amp_dsn_info.update(dict(loadreg=loadreg, psrr=psrr, psrr_fbw
    =psrr_fbw, pm=pm, err=dc_err, ibias=Id_tail*nf_tail))
209                 return True, amp_dsn_info
210         if psrr > psrr_min and loadreg < loadreg_max:
211             pm = 0
212             psrr_fbw = 0
213             while (pm < pm_min or psrr_fbw < psrr_fbw_min) and Id_tail*nf_tail <
    iamp_max:
214                 # Resize amp parameters
215                 Id_load = Id_tail*nf_tail/2
216 
217                 m_load = Id_load/op_load['ibias']
218                 wm_load = (m_load/2)%1 + 1
219                 nf_load = 2*int(m_load/2)
220 
221                 m_in = Id_load/op_in['ibias']
222                 wm_in = (m_in/2)%1 + 1
223                 nf_in = 2*int(m_in/2)
224 
225                 op_dict.update({'amp_in' : self.resize_op(op_in, wm_in),
226                                 'amp_tail' : self.resize_op(op_tail, wm_tail),
227                                 'amp_load' : self.resize_op(op_load, wm_tail)})
228                 nf_dict.update({'amp_in' : nf_in,
229                                 'amp_tail' : nf_tail,
230                                 'amp_load' : nf_load})
231                 wm_dict.update({'amp_in' : wm_in,
232                                 'amp_tail' : wm_tail,
233                                 'amp_load' : wm_load})
234                 loadreg = self._get_loadreg_lti(op_dict, nf_dict, ser_type,
    amp_in, cload+cdecap_max, 0, rsource, vincm, iload)
235                 psrr, psrr_fbw = self._get_psrr_lti(op_dict, nf_dict, ser_type,
```

```
    amp_in, cload+cdecap_max, 0, rsource)
236             pm = self._get_stb_lti(op_dict, nf_dict, ser_type, amp_in, cload+
    cdecap_max, 0, rsource)
237             nf_tail += 2
238         amp_dsn_info.update(dict(op_dict=op_dict,nf_dict=nf_dict,wm_dict=
    wm_dict))
239         amp_dsn_info.update(cap_dict=dict(cdecap_amp=0, cdecap_load=
    cdecap_max))
240         amp_dsn_info.update(dict(loadreg=loadreg, psrr=psrr, psrr_fbw=
    psrr_fbw, pm=pm, err=dc_err, ibias=Id_tail*nf_dict['amp_tail']))
241         spec_met = pm > pm_min and psrr > psrr_min and psrr_fbw >
    psrr_fbw_min and loadreg < loadreg_max and dc_err < err_max and Id_tail*
    nf_dict['amp_tail'] < iamp_max
242         return spec_met, amp_dsn_info
243     else:
244         return False, amp_dsn_info
245
246
247 def meet_spec(self, **params) -> List[Mapping[str,Any]]:
248     specfile_dict = params['specfile_dict']
249     th_dict = params['th_dict']
250     l_dict = params['l_dict']
251     sim_env = params['sim_env']
252
253     db_dict = {k:get_mos_db(spec_file=specfile_dict[k],
254                             intent=th_dict[k],
255                             sim_env=sim_env) for k in specfile_dict.keys()}
256     params.update(dict(db_dict=db_dict))
257
258     ser_type = params['ser_type']
259     vdd = params['vdd']
260     vout = params['vout']
261     iload = params['iload']
262     iref = params['iref']
263     iamp_max = params['iamp_max']
264     cload = params['cload']
265     cdecap_max = params['cdecap']
266     rsource = params['rsource']
267     err_max = params['err']
268     psrr_min = params['psrr']
269     psrr_fbw_min = params['psrr_fbw']
270     pm_min = params['pm']
271     loadreg_max = params['loadreg']
272     load_pole = params['load_pole']
```

```
273         v_res = params['v_res']
274
275         vth_ser = estimate_vth(db=db_dict['ser'],
276                                is_nch=ser_type=='n',
277                                lch=l_dict['ser'],
278                                vgs=vdd-vout if ser_type=='n' else -vdd/2,
279                                vbs=0-vout if ser_type=='n' else 0)
280
281         # Keep track of best option
282         best_op = dict(ibias=float('inf'),
283                        err=float('inf'),
284                        psrr=0,
285                        psrr_fbw=0,
286                        pm=0,
287                        loadreg=float('inf'))
288
289         type_dict = {'ser' : ser_type,
290                      'amp_load' : 'p',
291                      'amp_in' : 'n',
292                      'amp_tail' : 'n',
293                      'amp_mir' : 'n'}
294
295         w_dict = {k:db.width_list[0] for k,db in db_dict.items()}
296
297         self.other_params = dict(l_dict=l_dict,
298                                  w_dict=w_dict,
299                                  th_dict=th_dict,
300                                  cload=cload,
301                                  ser_type=ser_type)
302
303         # Sweep gate bias voltage of the series device
304         vg_min = vout+vth_ser
305         vg_max = min(vdd+vth_ser, vdd)
306         vg_vec = np.arange(vg_min, vg_max, v_res)
307
308         for vg in vg_vec:
309             print('Designing the series device...')
310             # Size the series device
311             match_ser, ser_info = self.dsn_fet(vg=vg, **params)
312             if not match_ser:
313                 continue
314             print('Done')
315
316             # Design amplifier s.t. output bias = gate voltage
```

```
317             # This is to maintain accuracy in the computational design proces
318             print('Designing the amplifier...')
319 
320             params.update(dict(voutcm=vg,
321                                iamp_max=iamp_max,
322                                ser_info=ser_info))
323             spec_met, amp_dsn_info = self.dsn_amp(**params)
324             print('Done')
325 
326             if not spec_met:
327                 print('Amp specs not met.')
328                 continue
329             else:
330                 print('AMP SPECS MET.')
331 
332             amp_dsn_info.update(dict(w_dict=w_dict, l_dict=l_dict, th_dict=
    th_dict, type_dict=type_dict))
333 
334             best_op.update(self.op_compare(best_op,amp_dsn_info))
335             iamp_max = best_op['ibias']
336         return [best_op]
337 
338     def _get_loopgain_lti(self, op_dict, nf_dict, ser_type, amp_in, rsource) ->
    float:
339         '''
340         Returns:
341             A: DC loop gain
342         '''
343         ckt = LTICircuit()
344 
345         n_ser = ser_type == 'n'
346         n_amp = amp_in == 'n'
347         vdd = 'vdd' if rsource != 0 else 'gnd'
348 
349         # Series device
350         ser_d = vdd if n_ser else 'reg'
351         ser_s = 'reg' if n_ser else vdd
352         ckt.add_transistor(op_dict['ser'], ser_d, 'out', ser_s, fg=nf_dict['ser'
    ], neg_cap=True)
353 
354         # Passives
355         if rsource != 0:
356             ckt.add_res(rsource, 'gnd', 'vdd')
357 
```

```
358         # Amplifier
359         tail_rail = 'gnd' if n_amp else vdd
360         load_rail = vdd if n_amp else 'gnd'
361         inp_conn = 'gnd' if n_ser else 'amp_in'
362         inn_conn = 'gnd' if not n_ser else 'amp_in'
363         ckt.add_transistor(op_dict['amp_in'], 'outx', inp_conn, 'tail', fg=
    nf_dict['amp_in'], neg_cap=False)
364         ckt.add_transistor(op_dict['amp_in'], 'out', inn_conn, 'tail', fg=nf_dict
    ['amp_in'], neg_cap=False)
365         ckt.add_transistor(op_dict['amp_tail'], 'tail', 'gnd', tail_rail, fg=
    nf_dict['amp_tail'], neg_cap=False)
366         ckt.add_transistor(op_dict['amp_load'], 'outx', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
367         ckt.add_transistor(op_dict['amp_load'], 'out', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
368
369         # Calculating stability margins
370         num, den = ckt.get_num_den(in_name='amp_in', out_name='reg', in_type='v')
371         A = num[-1]/den[-1]
372
373         return A
374
375     def _get_psrr_lti(self, op_dict, nf_dict, ser_type, amp_in, cload, cdecap_amp
    , rsource) -> float:
376         '''
377         Returns:
378             psrr: PSRR (dB)
379             fbw: Power supply -> output 3dB bandwidth (Hz)
380         '''
381         n_ser = ser_type == 'n'
382         n_amp = amp_in == 'n'
383
384         # Supply -> output gain
385         ckt_sup = LTICircuit()
386         ser_d = 'vdd' if n_ser else 'reg'
387         ser_s = 'reg' if n_ser else 'vdd'
388         inp_conn = 'gnd' if n_ser else 'reg'
389         inn_conn = 'reg' if n_ser else 'gnd'
390         tail_rail = 'gnd' if n_amp else 'vdd'
391         load_rail = 'vdd' if n_amp else 'gnd'
392
393         # Passives
394         if rsource != 0:
395             ckt_sup.add_res(rsource, 'vbat', 'vdd')
```

```
396         ckt_sup.add_cap(cload, 'reg', 'gnd')
397         ckt_sup.add_cap(cdecap_amp, 'out', 'reg')
398 
399         # Series device
400         ckt_sup.add_transistor(op_dict['ser'], ser_d, 'out', ser_s, fg=nf_dict['
    ser'], neg_cap=False)
401 
402         # Amplifier
403         ckt_sup.add_transistor(op_dict['amp_in'], 'outx', inp_conn, 'tail', fg=
    nf_dict['amp_in'], neg_cap=False)
404         ckt_sup.add_transistor(op_dict['amp_in'], 'out', inn_conn, 'tail', fg=
    nf_dict['amp_in'], neg_cap=False)
405         ckt_sup.add_transistor(op_dict['amp_tail'], 'tail', 'gnd', tail_rail, fg=
    nf_dict['amp_tail'], neg_cap=False)
406         ckt_sup.add_transistor(op_dict['amp_load'], 'outx', 'outx', load_rail, fg
    =nf_dict['amp_load'], neg_cap=False)
407         ckt_sup.add_transistor(op_dict['amp_load'], 'out', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
408 
409         if rsource == 0:
410             num_sup, den_sup = ckt_sup.get_num_den(in_name='vdd', out_name='reg',
     in_type='v')
411         else:
412             num_sup, den_sup = ckt_sup.get_num_den(in_name='vbat', out_name='reg'
    , in_type='v')
413         gain_sup = num_sup[-1]/den_sup[-1]
414         wbw_sup = get_w_3db(den_sup, num_sup)
415 
416         if gain_sup == 0:
417             return float('inf')
418         if wbw_sup == None:
419             wbw_sup = 0
420         fbw_sup = wbw_sup / (2*np.pi)
421 
422         psrr = 10*np.log10((1/gain_sup)**2)
423 
424         return psrr, fbw_sup
425 
426     def _get_stb_lti(self, op_dict, nf_dict, ser_type, amp_in, cload, cdecap_amp,
     rsource) -> float:
427         '''
428         Returns:
429             pm: Phase margin (degrees)
430         '''
```

```
431         ckt = LTICircuit()
432 
433         n_ser = ser_type == 'n'
434         n_amp = amp_in == 'n'
435         vdd = 'vdd' if rsource != 0 else 'gnd'
436 
437         # Series device
438         ser_d = vdd if n_ser else 'reg'
439         ser_s = 'reg' if n_ser else vdd
440         ckt.add_transistor(op_dict['ser'], ser_d, 'out', ser_s, fg=nf_dict['ser'
    ], neg_cap=False)
441 
442         # Passives
443         ckt.add_cap(cload, 'reg', 'gnd')
444         ckt.add_cap(cdecap_amp, 'out', 'reg')
445         if rsource != 0:
446             ckt.add_res(rsource, 'gnd', 'vdd')
447 
448         # Amplifier
449         tail_rail = 'gnd' if n_amp else vdd
450         load_rail = vdd if n_amp else 'gnd'
451         inp_conn = 'gnd' if n_ser else 'amp_in'
452         inn_conn = 'gnd' if not n_ser else 'amp_in'
453         ckt.add_transistor(op_dict['amp_in'], 'outx', inp_conn, 'tail', fg=
    nf_dict['amp_in'], neg_cap=False)
454         ckt.add_transistor(op_dict['amp_in'], 'out', inn_conn, 'tail', fg=nf_dict
    ['amp_in'], neg_cap=False)
455         ckt.add_transistor(op_dict['amp_tail'], 'tail', 'gnd', tail_rail, fg=
    nf_dict['amp_tail'], neg_cap=False)
456         ckt.add_transistor(op_dict['amp_load'], 'outx', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
457         ckt.add_transistor(op_dict['amp_load'], 'out', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
458 
459         # Calculating stability margins
460         num, den = ckt.get_num_den(in_name='amp_in', out_name='reg', in_type='v')
461         pm, _ = get_stability_margins(np.convolve(num, [-1]), den)
462 
463         return pm
464 
465     def _get_loadreg_lti(self, op_dict, nf_dict, ser_type, amp_in, cload,
    cdecap_amp, rsource, vout, iout) -> float:
466         '''
467         Returns:
```

```
468             loadreg: Load regulation for peak-to-peak load current variation of
    20% (V/V)
469         '''
470         n_ser = ser_type == 'n'
471         n_amp = amp_in == 'n'
472         vdd = 'vdd' if rsource != 0 else 'gnd'
473 
474         # Supply -> output gain
475         ckt = LTICircuit()
476         ser_d = vdd if n_ser else 'reg'
477         ser_s = 'reg' if n_ser else vdd
478         inp_conn = 'gnd' if n_ser else 'reg'
479         inn_conn = 'reg' if n_ser else 'gnd'
480         tail_rail = 'gnd' if n_amp else vdd
481         load_rail = vdd if n_amp else 'gnd'
482 
483         # Passives
484         if rsource != 0:
485             ckt.add_res(rsource, 'gnd', 'vdd')
486         ckt.add_cap(cload, 'reg', 'gnd')
487         ckt.add_cap(cdecap_amp, 'out', 'reg')
488 
489         # Series device
490         ckt.add_transistor(op_dict['ser'], ser_d, 'out', ser_s, fg=nf_dict['ser'
    ], neg_cap=False)
491 
492         # Amplifier
493         ckt.add_transistor(op_dict['amp_in'], 'outx', inp_conn, 'tail', fg=
    nf_dict['amp_in'], neg_cap=False)
494         ckt.add_transistor(op_dict['amp_in'], 'out', inn_conn, 'tail', fg=nf_dict
    ['amp_in'], neg_cap=False)
495         ckt.add_transistor(op_dict['amp_tail'], 'tail', 'gnd', tail_rail, fg=
    nf_dict['amp_tail'], neg_cap=False)
496         ckt.add_transistor(op_dict['amp_load'], 'outx', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
497         ckt.add_transistor(op_dict['amp_load'], 'out', 'outx', load_rail, fg=
    nf_dict['amp_load'], neg_cap=False)
498 
499 
500         num, den = ckt.get_num_den(in_name='reg', out_name='reg', in_type='i')
501         transimpedance = num[-1]/den[-1]
502 
503         loadreg = transimpedance*0.2*iout/vout
504 
```

```
505         return loadreg
506
507     def op_compare(self, op1:Mapping[str,Any], op2:Mapping[str,Any]):
508         return op1 if op1['ibias'] < op2['ibias'] else op2
509
510     def get_sch_params(self, op):
511         try:
512             w_dict_new = dict()
513             for key in op['w_dict'].keys():
514                 w_dict_new[key]=op['wm_dict'][key]*op['w_dict'][key]
515             dsn_op = dict(w_dict=w_dict_new,
516                           l_dict=op['l_dict'],
517                           nf_dict=op['nf_dict'],
518                           th_dict=op['th_dict'],
519                           type_dict=op['type_dict'],
520                           cap_dict=op['cap_dict'])
521         except KeyError:
522             dsn_op = 'No solution found within specs'
523         return dsn_op
```

Appendix E

VDDD Tap (SCμM)

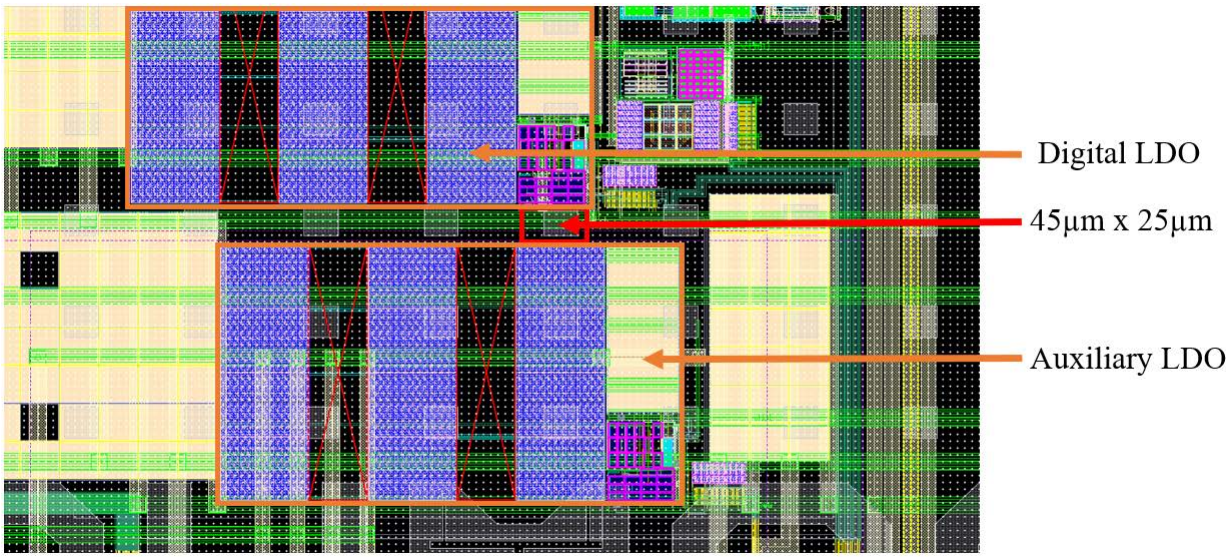


Figure E.1: VDDD tap device proposed layout area.

The biggest unresolved issue with the power systems on SCµM is the "VDDD tap" problem. A large proportion of the fabricated chips will stop executing instructions in the boot sequence unless the output of the digital LDO is briefly shorted to some voltage between the nominal LDO output (1V) and the battery voltage. The amount of time the LDO output must remain shorted is currently unknown. The cause of this problem is also still unknown, but the issue seems to occur after the first SRAM instruction, when the second instruction does not run. One proposed solution is to add a small transistor either in parallel with the series device (PMOS) or between the amplifier output and ground (NMOS) with an inverter at the gate. The proposed area for these added devices is shown in the 45µm by 25µm red box in Figure E.1, and the schematic with an added NMOS is shown in Figure E.2. The gate or inverter could then be connected to a repurposed GPO pin that would default low when power is connected and the boot sequence begins, shorting the series device gate to ground. Once the SRAM instructions continue to run and SCµM continues to boot, the GPO would switch high and the LDO would be allowed to operate normally. If the issue stems from the design of the digital LDO on SCµM, redesigning it may solve this issue on its own, but the auxiliary LDO uses an identical schematic and does not experience this problem. The SRAM on chip is provided by TSMC and was not designed specifically for SCµM, so the interaction between this block and the digital LDO could be another reason for the problem.

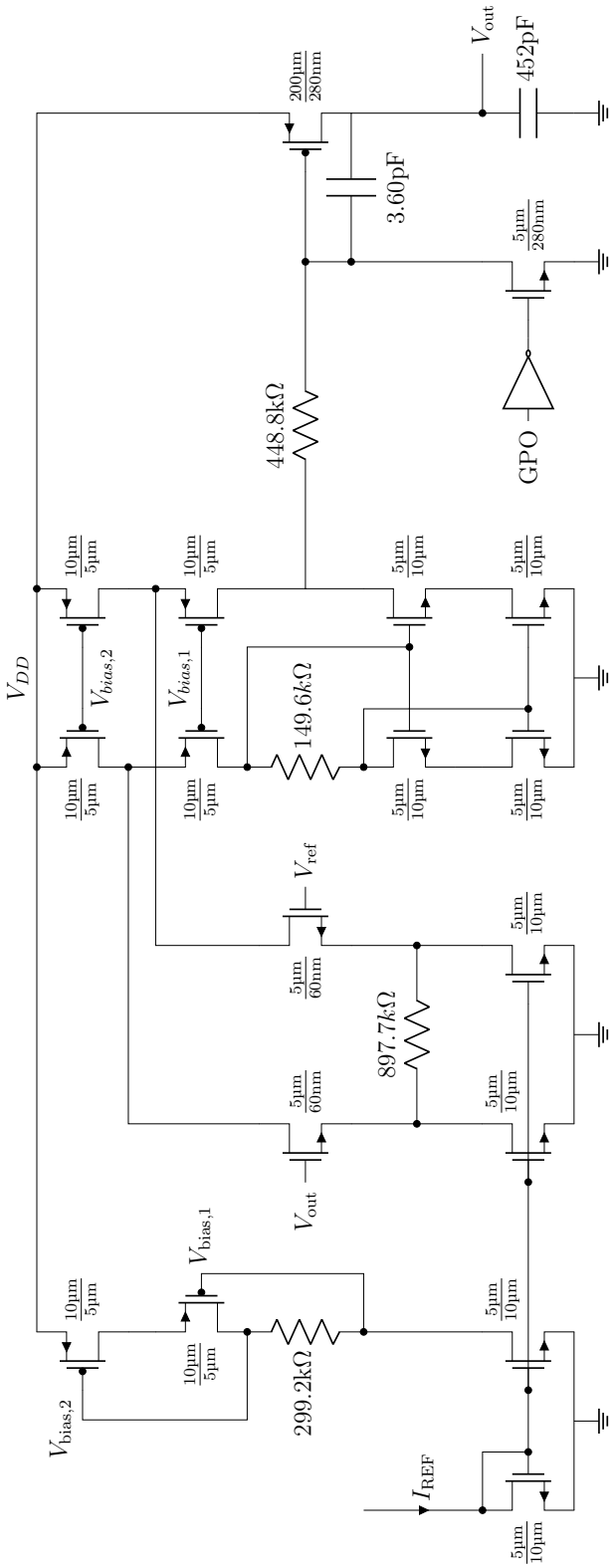


Figure E.2: SCμM Digital LDO schematic with NMOS tap device.