ECE1396H Assignment 5

Zonghao Li #1003843148

A. 6-bit binary-weighter current-mode DAC

Required by the question, the reference voltage is V_{ref} = 256 mV and the full-scale output V_{FS} is also 256 mV. Given a 6-bit code word, the voltage for LSB V_{LSB} is calculated by

$$V_{LSB} = \frac{V_{FS}}{2^6} = 4mV \tag{1}$$

Therefore, for a R = 50 Ω output impedance, the unit current is computed as

$$I = \frac{V_{LSB}}{R} = 80\mu A \tag{2}$$

Therefore we can further calculate the reference resistor R_{ref} can be calculated as

$$R_{ref} = \frac{V_{ref}}{I} = 3.2k\Omega \tag{3}$$

The transistor sizing in this question is difficult since to provide a nicely binary weighted current sources for individual bit, due to the reason that the transistor's width is not constantly proportional to the current, we need to size the transistor for each current source individually. One more concern is the transistor used to steer the current in each current steering block (such as Q1 in the assignment Fig.1). This transistor needs to steer all the current to the ground when the PMOS current source biased by V_{bias} (such as Q2 in the assignment Fig.1) just turns off. During the switch on-off procedures, the voltage swing at the drain of the cascoded PMOS (Q3 in the assignment Fig.1) need to be as small as possible.

We need to associate current, from LSB to MSB, as I, 2I, 4I, 8I, 16I, and 32I. The output voltage is

$$V_{out} = V_{ref}(b_1 2^{-1} + ... + b_6 2^{-6}) \tag{4}$$

Table I shows the dimension for all components used in this design, given the fact the transistor length L for all Q3 from LSB to MSB and Q4 in the current biasing circuit is 0.18 μm, which is the maximum allowed channel length in 65 nm technology, and all Q1 and Q2 plus the current source in the biasing network has a channel length of 0.06 μm. The maximum current density is around (80 μA)/(4 μm) = 20 μA/μm, smaller than the requirement. Do not include in the table, but all transistors in the biasing network use only one finger for each.

The unit R_{unit} is 400 Ω. R_{ref} will be 8 R_{unit} in series, and load impedance R_{load} 50 Ω will be 8 R_{unit} in parallel. This way the process variation and device mismatch can be mitigated.

Fig. 1 shows the output current from each bit and the output voltage given a code word "111111". It can be seen that the output voltage is simulated around 263.3 mV and the current from each bit are binary weighed, from 80 μA given by b6 and 2.56 mA given by b1.

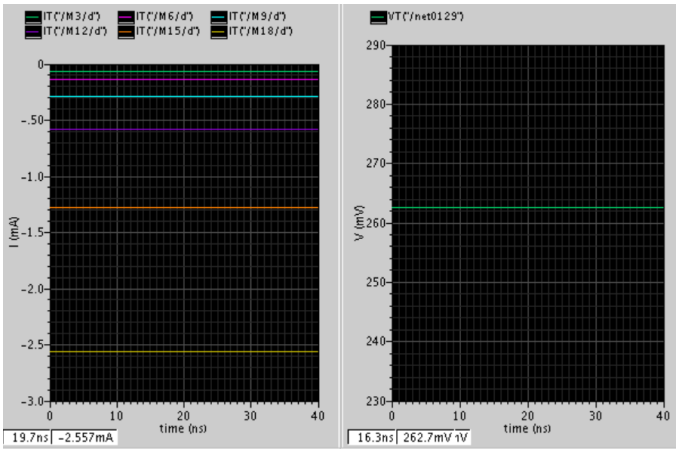

Fig. 1: Output current from each bit (left) and output voltage for "111111".

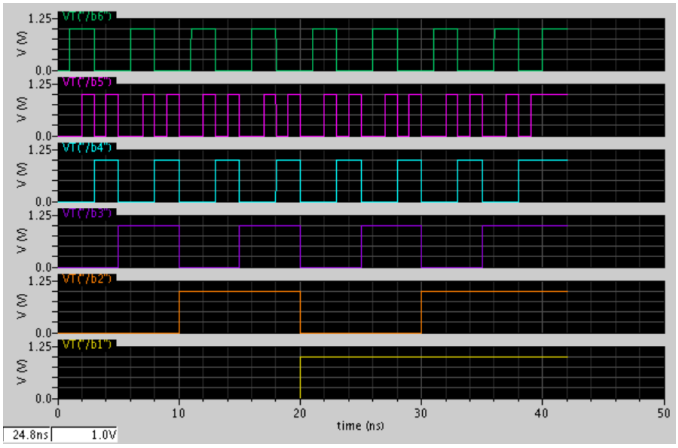

Fig. 2: Output from ideal 6-bit ADC given a ramp voltage input. From top to bottom: LSB to MSB

To test the performance of this DAC, the following testbench is constructed. An full-scale input ramp voltage will be fed to an ideal 6-bit ADC, and the output from the ADC will be fed to the designed DAC, and the output will be further processed. Fig. 2 shows the output from the 6-bit ideal ADC given a full-scale input ramp voltage (40 nS ramping time and 10 Gs/s sampling rate). It can be seen that the ideal ADC indeed convert an full scale analog input to a full-scale digital output, from "000000" to "1111111". Therefore, we manage to generate all code words needed for testing the performance of the DAC.

Fig. 3 shows the output from the DAC after feeding the generate output code words from the ideal ADC. It can be

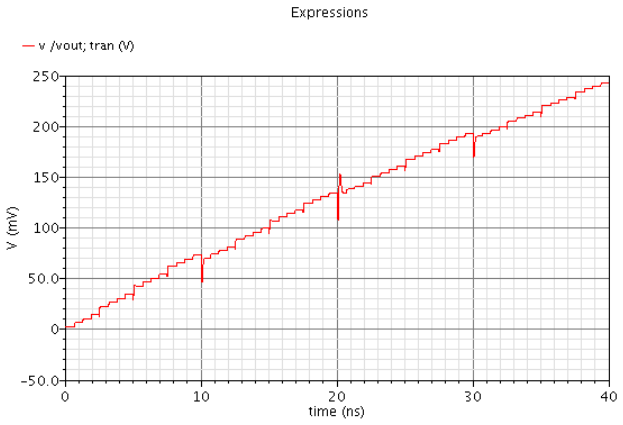

Fig. 3: Output from the designed DAC.

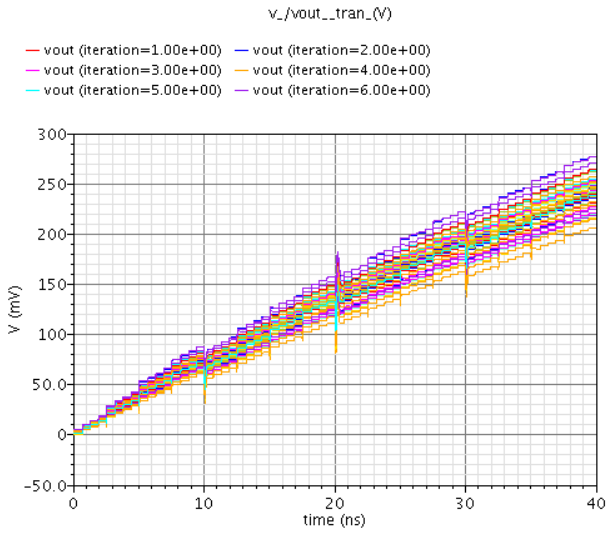

Fig. 4: Monte-Carlo simulation and its output from the designed DAC.

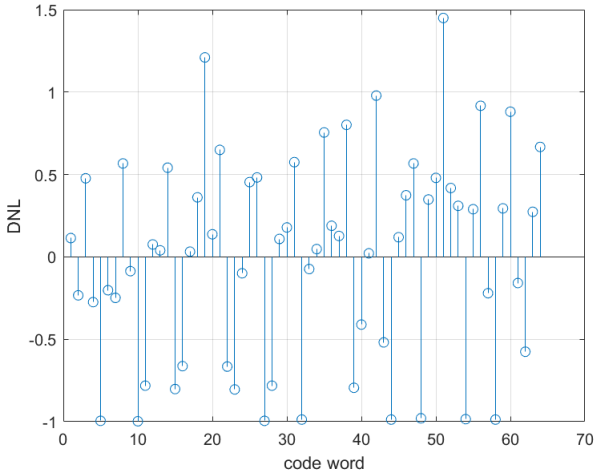

Fig. 5: DNL of the binary-weighted current mode DAC.

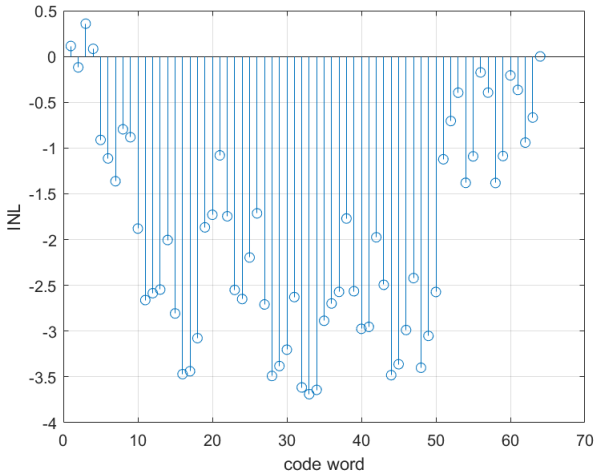

Fig. 6: INL of the binary-weighted current mode DAC.

seen that due to the imperfect transistor sizing, the current from each current source are not perfectly binary weighed, causing the distortion of the transfer characteristic. Another observation is the glitches at output voltage when the steering circuits are switching between on and off states. Especially at the mid-code "100000" (around 128 mV), which is the stage where the most current steering circuits are switching from on to off stages, contributing to a very big glitch at the output. The output voltage range is in between 0 and 250 mV

To test how the process variation and component mismatch impact the circuit transfer characteristic, Monte-Carlo simulation at 65 ^oC is conducted. Fig. 4 shows the simulation result. One can see an about maximum 50 mV variation that corresponds to about 3 LSB bits is simulated. Fig. 5 and 6 the DNL and INL plot after taking the average of 50 Monte-Carlo simulation results, respectively. It can be seen that some code words a maximum DNL above 1 LSB and many code words have a maximum INL larger than 0.5 LSB. Therefore there will be missing codes.

Fig. 7 and 8 shows the SDR and ENOB of an sinusoidal input with various input frequency and sampling frequency, respectively. ENOB for sinusoidal input is defined as

$$ENOB = \frac{SDR - 1.76dB}{6.02} \tag{5}$$

As one can see, the ENOB at low frequency is around 2 bits that does not even close to the 6-bit specification. At about 2.1 GHz there is a 3 dB drops at SDR for both sampling frequency. If this DAC is only targeted for 2-bit, then the effective bandwidth is 2.1 GHz. To further improve the performance of this circuit, more careful and iterative tweaking of each current steering circuit is required, but can be time consuming.

B. Segmented 6-bit current mode DAC

In this designed, we will have a segmented 6-bit current mode DAC whose 3 MSB bits will be rendered by thermometer code based DAC, and 3 LSB bits based on R-2R DAC topology. The circuit is implemented in the differential topology. Shown in Fig. 9 and 10, are the circuit for thermometer code based DAC and R-2R DAC, respectively. For this design, we will res-use the current biasing network. Also, in order to see how segmented DAC improve the result

TABLE I: Transistor finger amount (N) and resistor W and L for 6-bit binary-weighted current-mode DAC

	b1	b2	b3	b4	b5	b6	W (μm)	L (μm)
Q1 (1 finger is 1 μm)	N=80	N=60	N=30	N=20	N=10	N=10	-	-
Q2 (1 finger is 1 μm)	N=256	N=128	N=64	N=32	N=8	N=4	-	-
Q3 (1 finger is 4 μm)	N=6	N=3	N=2	N=1	N=1	N=1	-	-
R_{unit} (400 Ω)	-	-	-	-	-	-	1	0.63
R_{ref} (8 R_{unit} in series = 3.2 K)	-	-	-	-	-	-	-	-
R_{load} (8 R_{unit} in parallel = 50 Ω)	-	-	-	-	-	-	-	-

TABLE II: Transistor finger amount (N) and resistor W and L for 7-bit thermometer-code current-mode DAC

	t1	t2	t3	t4	t5	t6	t7	W (μm)	L (μm)
Q1 (1 finger is 1 μm)	N=1	N=1	N=1	N=1	N=1	N=1	N=1	-	-
Q2 (1 finger is 1 μm)	N=1	N=1	N=1	N=1	N=1	N=1	N=1	-	-
Q3 (1 finger is 4 μm)	N=1	N=1	N=1	N=1	N=1	N=1	N=1	-	-
R_{unit} (400 Ω)	-	-	-	-	-	-	-	1	0.63
R_{ref} (8 R_{unit} in series = 3.2 K)	-	-	-	-	-	-	-	-	-
$R_{load,ther}$ (6 R_{unit} in parallel = 71.38 Ω)	-	-	-	-	-	-	-	-	-

TABLE III: Transistor finger amount (N) and resistor W and L for 3-bit R-2R current-mode DAC

	b4	b5	b6	W (μm)	L (μm)
Q1 (1 finger is 1 μm)	N=1	N=1	N=1	-	-
Q2 (1 finger is 1 μm)	N=1	N=1	N=1	-	-
Q3 (1 finger is 4 μm)	N=1	N=1	N=1	-	-
R_{unit} (400 Ω)	-	-	-	1	0.63
R_{ref} (8 R_{unit} in series = 3.2 K)	-	-	-	-	-
$R_{load,R2R}$ (12 R_{unit} in parallel = 35.69 Ω)	-	-	-	-	-
$R_{div,R2R}$ (4 R_{unit} in series = 1.6 kΩ)	-	-	-	-	-

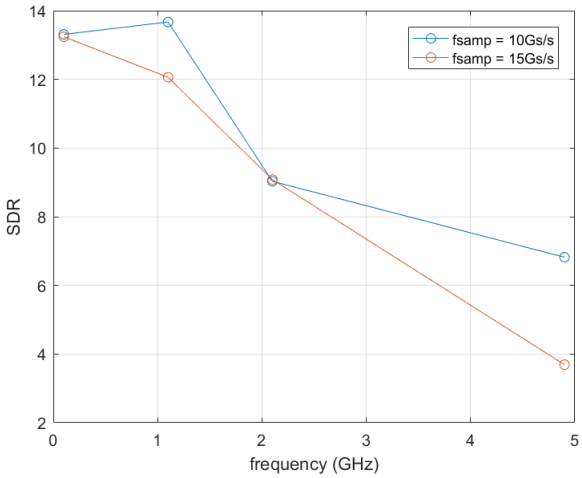

Fig. 7: SDR versus input frequency at different sampling speed.

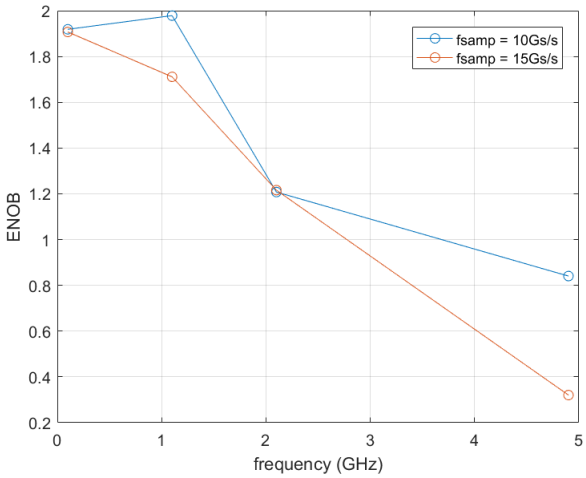

Fig. 8: ENOB versus input frequency at different sampling speed.

compared to binary weighted DAC, we still stick to 256 mV full-scale output.

Since we need to generate 7-bit thermometer code, we need 7 current steering circuit. To first calculate how much current each current steering circuit needs to yield in thermometer code DAC, since the 7-bit thermometer code DAC is essentially the first 3 MSB in the targeted 6-bit code word, we have

$$V_{out_ther} = V_{ref}(b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3}) \quad (6)$$

So the full-scale voltage output from the thermometer-code DAC, given the same V_{ref} as last question, V_{out_ther} = 7/8V_{ref}. Therefore, the product of the current generate by each steering current, with the output impedance, should be 1/8V_{ref}. Since in the thermometer-code DAC each bit has the same current output, to calculate the ideal load impedance that should contribute 1/8V_{ref} = 32 mV, giving the same unit current from

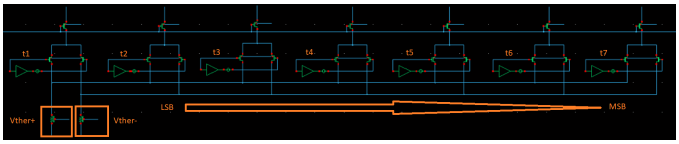

Fig. 9: 7-bit thermometer code DAC.

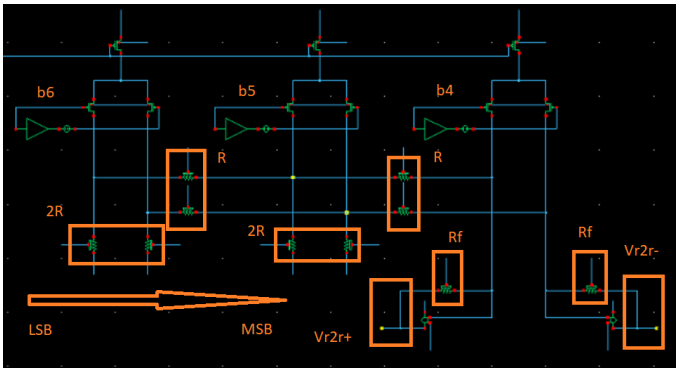

Fig. 10: 3-bit R-2R DAC.

each current source and same R_{ref} as the last question (I = 80 μA) and , we have

$$\frac{V_{ref}}{2R_{ref}}(7)R_{load,ther} = V_{ref}(2^{-1}+2^{-2}+2^{-3})$$
$$R_{load,ther} = \frac{R_{ref}}{4} \tag{7}$$

We also can roughly characterize the R-2R 3-bit DAC. Similar to binary weighted DAC, the product of the current generated by the last LSB current steering circuit, should be equal to V_{LSB} = 4 mV. Therefore, to calculate the load impedance of R-2R DAC

$$2^{-2}IR_{load,R2R} = V_{LSB}$$
$$R_{load,R2R} = \frac{V_{LSB}}{2^{-2}I} \tag{8}$$

Table II summarize all component parameters after some tweaking and optimizing for thermometer-code DAC. $R_{load,ther}$ is optimized to 71.38 Ω by using 6 400 Ω unit resistor in parallel. Do not include in the table, but all transistors in the biasing network use only one finger for each. The channel length L for the transistors and naming conventions are the same as the last question.

Table III summarize all component parameters after some tweaking and optimizing for R-2R DAC. $R_{load,R2R}$ is optimized to 35.69 Ω by using 12 400 Ω unit resistor in parallel. The value for $R_{div,R2R}$ used for current divider (R-2R topology) is 1.6 KΩ (as long as the current get divided by 2 from MSB to LSB). Do not include in the table, but all transistors in the biasing network use only one finger for each. The channel length L for the transistors and naming conventions are the same as the last question.

To test the whole segmented DAC, similar testbench as the previous question is used, excepted the first 3 MSB are passed from the ideal ADC first to a binary-to-thermometer code decoder, then fed to the thermometer-code DAC. Fig. 11 shows the output from the whole DAC, demonstrating high

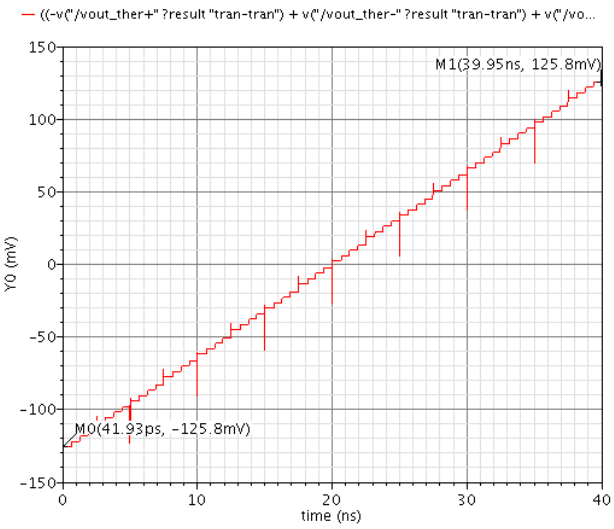

Fig. 11: Output from the 6-bit segmented DAC given a linear ramp input.

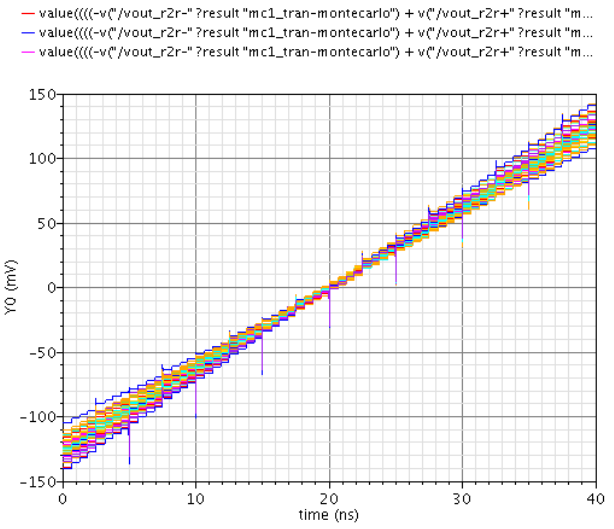

Fig. 12: Monte Carlo simulation of the segmented DAC.

linearity. Noticing that compared to binary weighted DAC, due to the application of thermometer-code DAC, glitches are hugely mitigated. Additionally, in both segmented and R-2R topology, all current sources generate the same amount of current, making the design procedure easier and more robust with process variation and device mismatch. Fig. 12 shows the Monte Carlo simulation result of the segmented DAC. Due to the differential topology, we can see that even the maximum voltage deviation can be up to 50 mV, the average overall full-scale output voltage is still around 256 mV. Despite the impact from process variation and device mismatch, the output characteristic maintains high linearity.

Fig. 13 and 14 show the DNL and INL of the designed segmented DAC. It can be seen that all code words have DNL less than 1 LSB that guarantees the monotonicity, and very few code words have INL larger than 0.5 LSB. Therefore the there

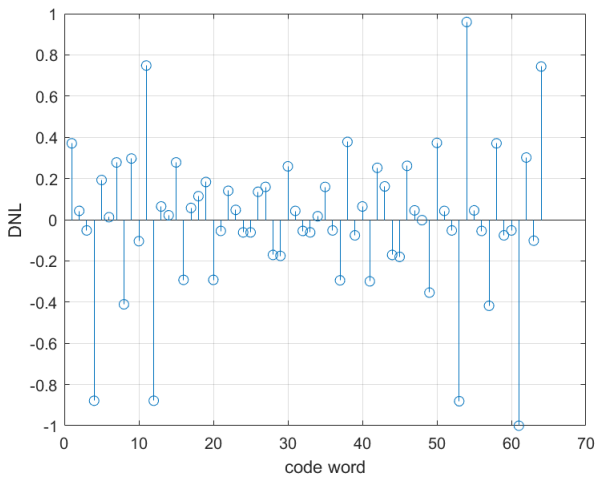

Fig. 13: DNL of the 6-bit segmented DAC.

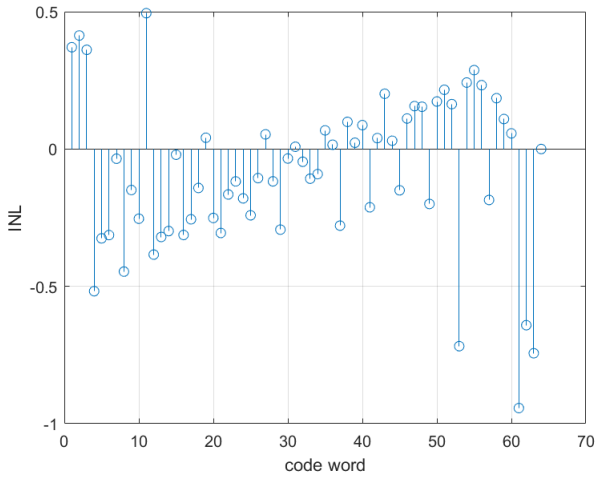

Fig. 14: INL of the 6-bit segmented DAC.

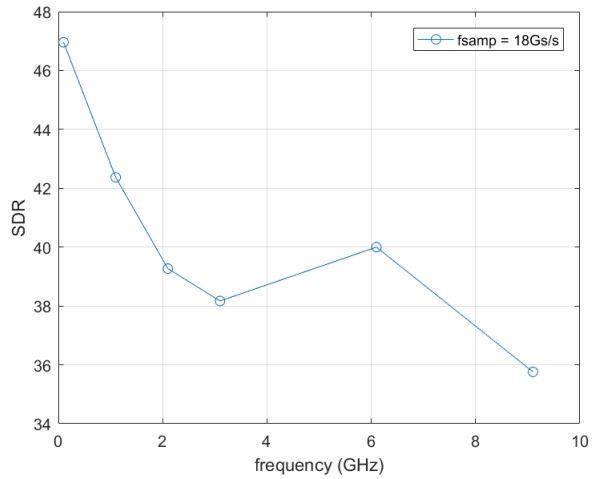

Fig. 15: SDR of the 6-bit segmented DAC with asked input frequency and sampling rate.

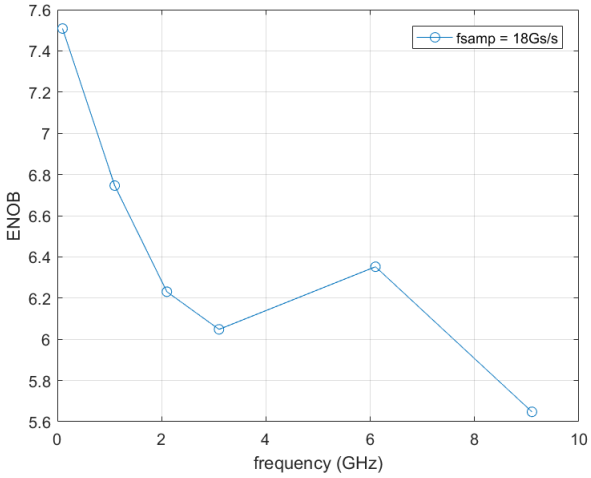

Fig. 16: ENOB of the 6-bit segmented DAC with asked input frequency and sampling rate.

will be very few missing codes. Fig. 15 and 16 shows the SDR and ENOB of the segmented DAC given a sinusoidal input with various asked input frequency, all sampled at 18 Gs/s. With the help of differential topology, noticeable improvement has achieved for SDR and ENOB due to the cancellation of even harmonics. It can be seen that after about 1.1 GHz, the SDR drops 3dB (from 47 dB to 43 dB), but the ENOBis still above 6 bits. At around 9.1 GHz the ENOB is around 5.6 GHz. So from this perspective, the effective bandwidth can be up more than 9.1 GHz.

```
// VerilogA for ece1396h, bin2ther, veriloga

`include "constants.vams"
`include "disciplines.vams"
`include "discipline.h"
`include "constants.h"

// $Date: 1997/08/28 05:54:32 $
// $Revision: 1.1 $
//
//
// Based on the OVI Verilog-A Language Reference Manual, version 1.0 1996
//
//



//--------------------
// bin2ther_ideal
//
// -  Ideal binary to thermometer code decoder
//
// vin:        [V,A]
// vclk:       [V,A]
// vd0..vd6:   data output terminals  [V,A]
//
// INSTANCE parameters
//    tdel, trise, tfall = {usual} [s]
//    vlogic_high = [V]
//    vlogic_low  = [V]
//    vtrans_clk  = clk high to low transition voltage [V]
//    vref        = voltage that voltage is done with respect to [V]
//
// MODEL parameters
//    {none}
//
// This model is ideal in the sense that there is no mismatch modeled.
//

module bin2ther(vd6, vd5, vd4, vd3, vd2, vd1, vd0, vin2, vin1, vin0);
electrical vd6, vd5, vd4, vd3, vd2, vd1, vd0, vin2, vin1, vin0;
parameter real trise = 0 from [0:inf);
parameter real tfall = 0 from [0:inf);
parameter real tdel = 0 from [0:inf);
parameter real vlogic_high = 5;
parameter real vlogic_low  = 0;
//parameter real vtrans_clk     = 2.5;
//parameter real vref        = 1.0;

real vd[0:6];

   analog begin

                vd[6] = 0;
                if (V(vin2) == vlogic_low && V(vin1) == vlogic_low && V(vin0) == vlogic_low) begin
                     vd[6] = vlogic_low;
                     vd[5] = vlogic_low;
                     vd[4] = vlogic_low;
                     vd[3] = vlogic_low;
                     vd[2] = vlogic_low;
                     vd[1] = vlogic_low;
                     vd[0] = vlogic_low;
                end
                else if (V(vin2) == vlogic_low && V(vin1) == vlogic_low && V(vin0) == vlogic_high) begin
                     vd[6] = vlogic_low;
                     vd[5] = vlogic_low;
                     vd[4] = vlogic_low;
                     vd[3] = vlogic_low;
                     vd[2] = vlogic_low;
                     vd[1] = vlogic_low;
                     vd[0] = vlogic_high;
                end
                else if (V(vin2) == vlogic_low && V(vin1) == vlogic_high && V(vin0) == vlogic_low) begin
                     vd[6] = vlogic_low;
                     vd[5] = vlogic_low;
                     vd[4] = vlogic_low;
                     vd[3] = vlogic_low;
                     vd[2] = vlogic_low;
                     vd[1] = vlogic_high;
```

```
                  vd[0] = vlogic_high;
            end
            else if (V(vin2) == vlogic_low && V(vin1) == vlogic_high && V(vin0) == vlogic_high) begin
                  vd[6] = vlogic_low;
                  vd[5] = vlogic_low;
                  vd[4] = vlogic_low;
                  vd[3] = vlogic_low;
                  vd[2] = vlogic_high;
                  vd[1] = vlogic_high;
                  vd[0] = vlogic_high;
            end
            else if (V(vin2) == vlogic_high && V(vin1) == vlogic_low && V(vin0) == vlogic_low) begin
                  vd[6] = vlogic_low;
                  vd[5] = vlogic_low;
                  vd[4] = vlogic_low;
                  vd[3] = vlogic_high;
                  vd[2] = vlogic_high;
                  vd[1] = vlogic_high;
                  vd[0] = vlogic_high;
            end
            else if (V(vin2) == vlogic_high && V(vin1) == vlogic_low && V(vin0) == vlogic_high) begin
                  vd[6] = vlogic_low;
                  vd[5] = vlogic_low;
                  vd[4] = vlogic_high;
                  vd[3] = vlogic_high;
                  vd[2] = vlogic_high;
                  vd[1] = vlogic_high;
                  vd[0] = vlogic_high;
            end
            else if (V(vin2) == vlogic_high && V(vin1) == vlogic_high && V(vin0) == vlogic_low) begin
                  vd[6] = vlogic_low;
                  vd[5] = vlogic_high;
                  vd[4] = vlogic_high;
                  vd[3] = vlogic_high;
                  vd[2] = vlogic_high;
                  vd[1] = vlogic_high;
                  vd[0] = vlogic_high;
            end
            else if (V(vin2) == vlogic_high && V(vin1) == vlogic_high && V(vin0) == vlogic_high) begin
                  vd[6] = vlogic_high;
                  vd[5] = vlogic_high;
                  vd[4] = vlogic_high;
                  vd[3] = vlogic_high;
                  vd[2] = vlogic_high;
                  vd[1] = vlogic_high;
                  vd[0] = vlogic_high;
            end
            else begin
                  vd[6] = vd[6];
                  vd[5] = vd[5];
                  vd[4] = vd[4];
                  vd[3] = vd[3];
                  vd[2] = vd[2];
                  vd[1] = vd[1];
                  vd[0] = vd[0];
            end

      //
      // assign the outputs
      //

      V(vd6) <+ transition( vd[6], tdel, trise, tfall );
      V(vd5) <+ transition( vd[5], tdel, trise, tfall );
      V(vd4) <+ transition( vd[4], tdel, trise, tfall );
      V(vd3) <+ transition( vd[3], tdel, trise, tfall );
      V(vd2) <+ transition( vd[2], tdel, trise, tfall );
      V(vd1) <+ transition( vd[1], tdel, trise, tfall );
      V(vd0) <+ transition( vd[0], tdel, trise, tfall );

   end
endmodule
```