CSIE ADIC Lab 2023

Lab06H. Design a Low-Voltage All-Digital Duty-Cycle Corrector

Exercise1: Interpolator-based Fine-tuning Delay Line

Description

The digitally-controlled delay line (DCDL) provides the delay time between the input clock and the output clock. The architecture of the DCDL is composed of a coarse-tuning delay line (CDL) and a fine-tuning delay line (FDL), as shown in Fig. 1. In CDL, each coarse-tuning delay unit consists of three NAND gates and a dummy NAND gate. The dummy cells are added to balance the capacitance loading of the NAND gates. The CDL consists of 15 coarse delay units. Moreover, the coarse-tuning resolution of the CDL is equal to the delay time of two NAND gates. Two signals (CA_OUT, CB_OUT) which the timing difference is also the delay time of two NAND gates are sent to the FDL. Subsequently, the FDL can enhance the resolution of the delay line to 1/31 of the coarse-tuning resolution (i.e. delay time of two NAND gates).

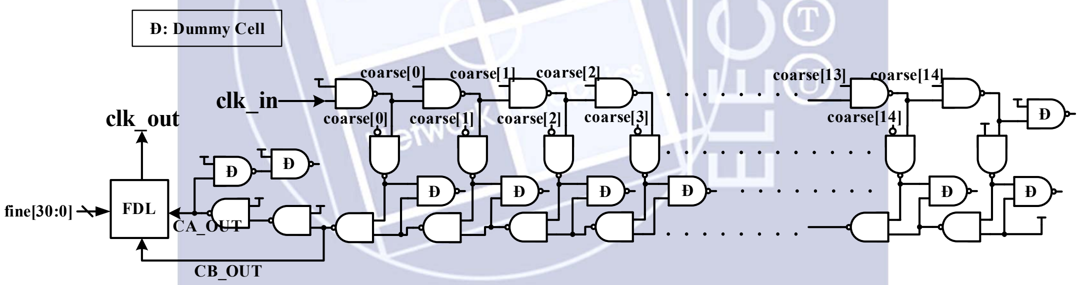


Figure 1: Architecture of the DCDL.

The architecture of the FDL is shown in Fig. 2. In FDL, a digitally controlled phase interpolator is applied to enhance the fine-tuning resolution of the delay line and guarantee the controllable delay range of the fine delay line (FDL) can cover one coarse-tuning resolution with process, voltage, and temperature (PVT) variations. The delay time of FDL is adjusted by the driving strength of two parallel connected tri-state buffer arrays. The rising edge of the output clock (OUT) will be phase aligned with CA_OUT when the fine-tuning control code (code [30:0]) is fully opened (i.e. 31’h7FFF_FFFF). In contrast, the output clock (OUT) will be phase aligned with

CB_OUT when the fine-tuning control code (code [30:0]) is 31'h0. With adjusting the number of turned-on tri-state buffers, the resolution of the delay line can be enhanced to be 1/31 coarse-tuning resolution by the FDL.

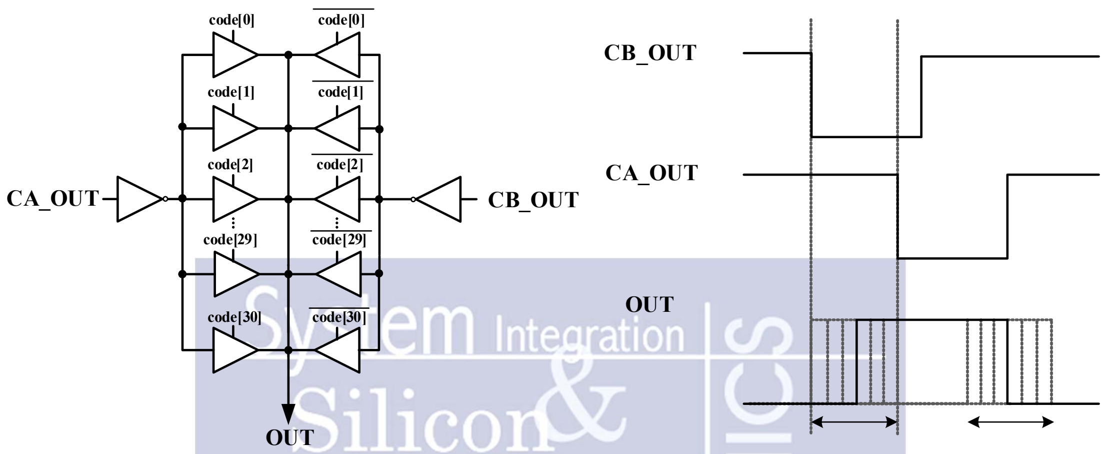


Figure 2: The fine-tuning delay line (FDL).

Specification:

CDL			
Signal Name	Direction	Bit Width	Description
clk_in	input	1	CDL's input signal
clk_a	output	1	CDL's output signal (CA_OUT)
clk_b	output	1	CDL's output signal (CB_OUT)
code0~code14	input	15	CDL's control bits

FDL			
Signal Name	Direction	Bit Width	Description
clk_a	input	1	CDL's output signal (CA_OUT)

clk_b	input	1	CDL's output signal (CB_OUT)
code0~code30	input	31	FDL's control bits
out	output	1	FDL's output signal

1. Data Preparation

1. Extract LAB data from TA's directory:
   % tar xzvf ~adicta/lab06h.tar.gz
2. The extracted LAB directory (**lab06h**) contains:
   a. **00_library/** : HSPICE Models and Cell circuits
   b. **Exercise1/** : Digital controlled delay line (DCDL)
   c. **Exercise2/** : Low-voltage all-digital DCC design

2. Run DCDL Simulation

1. Change to directory: **Exercise1**
2. Open and reading CORE.v
   % gedit CORE.v &
   In the CORE module, the DCDL shown in Fig. 1 is designed. It is composed of a coarse-tuning delay line and a fine-tuning delay line. The control code for the coarse-tuning delay line is coarse_code[14:0], and the control code for the fine-tuning delay line is fine_code[30:0].
3. Open and reading CDL.v
   % gedit CDL.v &
   In the CDL module, the coarse-tuning delay line shown in Fig. 1 is designed with standard cells using gate-level descriptions of Verilog language. The input clk_in signal will be delayed and then output clk_a and clk_b signals for the fine-tuning delay line.
4. Open and reading FDL.v
   % gedit FDL.v &
   In the FDL module, the fine-tuning delay line shown in Fig. 2 is designed with standard cells using gate-level descriptions of Verilog language. The input clk_a and clk_b signals will be interpolated and then output the out signal.
5. Open and reading chip.sp
   % gedit chip.sp &
   In chip.sp, ".vlog_include" is used to include the gate-level Verilog code:

CORE.v into UltraSim simulation. In addition, measurement commands are added in “**measure.inc**” file to calculate the delay time of the delay line with different control codes. Moreover, although the nominal supply voltage for 0.18μm process is **1.8V**, we use **1.0V** to simulate the all-digital DCC at low supply voltage.

6. Open and editing the CORE.vec

   % gedit CORE.vec &

   In CORE.vec, coarse_code[14:0] and fine_code[30:0] are the control codes for coarse-tuning delay line and fine-tuning delay line, respectively. The value of the control code is in hexadecimal. In this exercise, when coarse_code[14:0] is 15’h7FF, we increase the fine_code[30:0] from the minimum value (31’h0) to the maximum value (31’h7FFF_FFFF). Then the coarse_code[14:0] is switched to the next value 15’hFFF, and the fine_code[30:0] returns to its minimum value (31’h0). Subsequently, we increase the fine_code[30:0] from the minimum value (31’h0) to the maximum value (31’h7FFF_FFFF) again. When the delay controllable range of the fine-tuning delay line covers one coarse-tuning delay line, the delay time of the delay line will be monotonically increasing according to the input control codes.

7. Execute Ultrasim to perform SPICE circuit simulation of chip.sp.

   % ./01_run_ultrasim

8. Open and reading the transient simulation measurement results

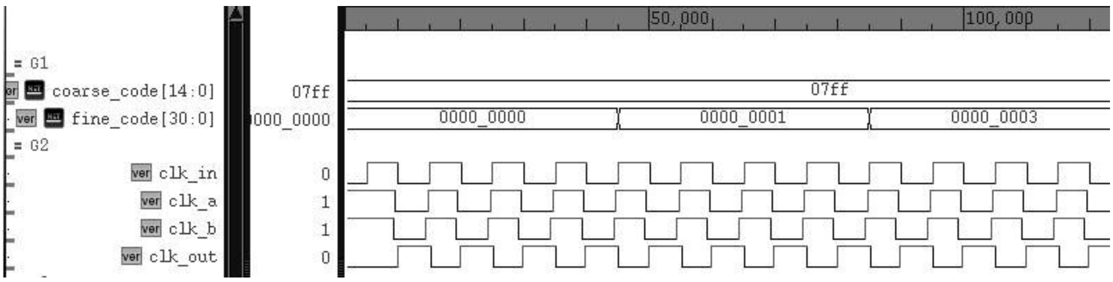
   % gedit chip.meas0 &

   In the measurement results, you can find the delay time of the delay line with the input control codes.

9. After Ultrasim is done, start Verdi to debug the output waveform:

   % nWave &

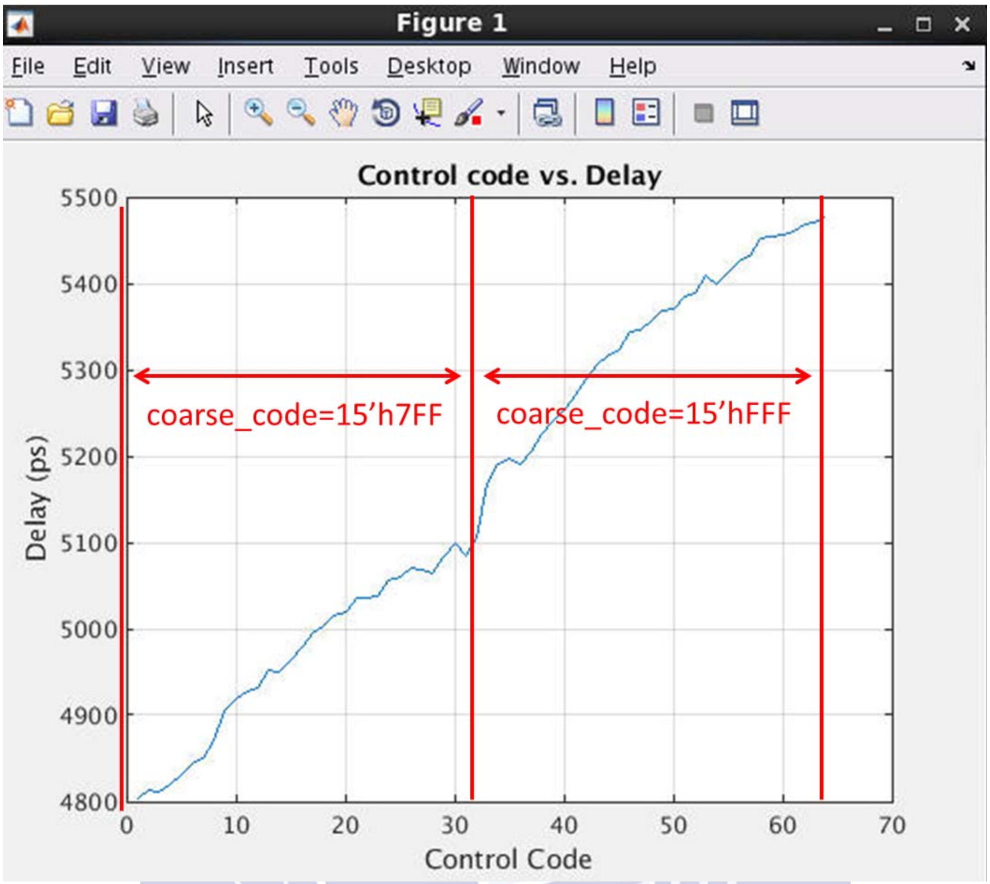
   In nWave, open **chip.fsdb**, and restore the saved signal file **chip.rc**. Then you can see the simulation results.



10. Use MATLAB to plot the delay time versus delay line control code from transient simulation measurement results (**chip.mt0**)

    % ./02_run_matlab

In the following figure, it shows that when the delay controllable range of the fine-tuning delay line covers one coarse-tuning delay line, the delay time of the delay line will be monotonically increasing according to the input control codes.



Exercise2: All-Digital Duty-Cycle Corrector

Description

In Exercise2, the delay line shown in Exercise1 is applied to the design of the all-digital duty-cycle corrector (ADDCC) as the half-cycle time delay line (HCDL). The architecture of the ADDCC is shown in Fig. 3. The HCDL is composed of a 4-bit CDL and a 5-bit FDL.

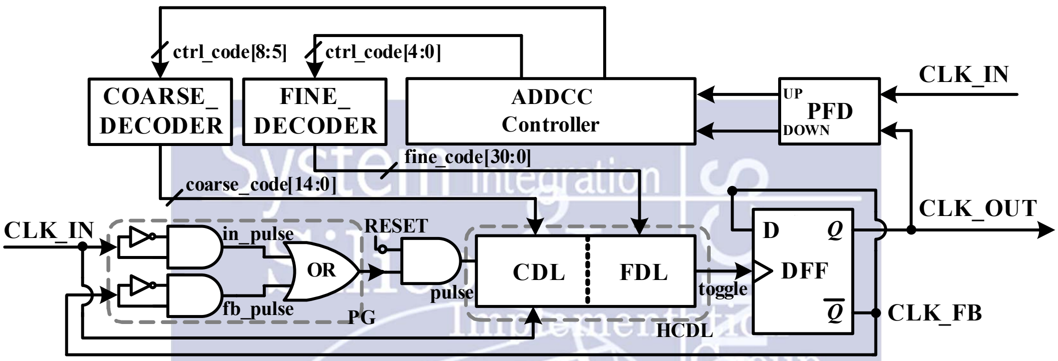


Figure 3: The architecture of the ADDCC.

The PG transforms the input clock (CLK_IN) and the feedback clock (CLK_FB) into narrow pulses (in_pulse and fb_pulse). The AND gate before the “pulse” signal will be pulled down at zero to avoid unnecessary pulses triggering the DFF until the reset signal (RESET) is pulled down. Once reset signal (RESET) is pulled down, the AND gate before the “pulse” signal allows the short pulses propagate through the digitally controlled HCDL.

The ADDCC controller adjusts the delay line control code (ctrl_code[8:0]) by the PFD’s outputs. Subsequently, the COARSE_DECODER and the FINE_DECODER will decode the decimal control code into the thermometer code (coarse_code[14:0] and fine_code[30:0]) to control the HCDL. When ADDCC is locked, the frequency of the “pulse” and “toggle” signals will be two times of the reference clock (CLK_IN) frequency. Finally, the DFF divides the “toggle” signal by two and outputs an exact 50% duty-cycle clock (CLK_OUT). Consequently, the output clock (CLK_OUT) frequency is the same as the reference clock. Moreover, the output clock (CLK_OUT) will also phase align with the reference clock (CLK_IN).

Specification

ADDCC			
Signal Name	Direction	Bit Width	Description
reset	input	1	Active high reset signal
clk_in	input	1	Input clock with duty-cycle error
clk_out	output	1	Output 50% duty-cycle clock
lock	output	1	ADDCC's lock signal

3. Run All-Digital DCC Simulation

1. Change to directory: **Exercise2**
2. Open and reading CORE.v

   % gedit CORE.v &

   The top module of **CORE.v** is CORE, the architecture of ADDCC is shown in Fig. 3, and the meanings of I/O pins are described in the above specification table. The ADDCC controller is designed with behavior-level Verilog language, and then it is synthesized and stored in "**CONTROLLER_syn.v**". The COARSE_DECODER (**COARSE_DECODER_syn.v**) and the FINE_DECODER (**FINE_DECODER_syn.v**) will decode the decimal control code into the thermometer code (coarse_code[14:0] and fine_code[30:0]) to control the HCDL. The phase and frequency detector (**PFD.v**) is designed with standard cells using gate-level descriptions of Verilog language. Moreover, the pulse generator (**PG.v**) is also designed with standard cells using gate-level descriptions of Verilog language.
3. Open and reading chip.sp

   % gedit chip.sp &

   In chip.sp, ".vlog_include" is used to include the gate-level Verilog code:

CORE.v into UltraSim simulation. In addition, measurement commands are added in "**measure.inc**" file to calculate the duty-cycle of the output clock during the operation of the ADDCC. Moreover, although the nominal supply voltage for 0.18μm process is **1.8V**, we use **1.0V** to simulate the all-digital DCC at low supply voltage.

4. Open and editing the CORE.vec

   % gedit CORE.vec &

   In **CORE.vec**, a 150 MHz clock signal (clk_in) with duty-cycle error is input to the ADDCC. The ADDCC will correct the input clock and output a clock with 50% duty-cycle.

5. Execute Ultrasim to perform SPICE circuit simulation of chip.sp.

   % ./01_run_ultrasim

6. Open and reading the transient simulation measurement results

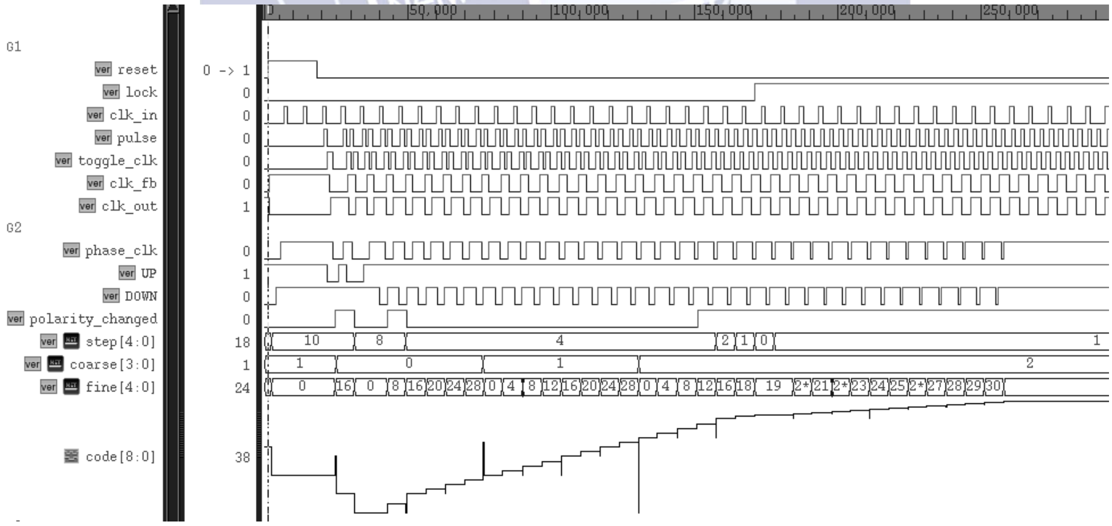
   % gedit chip.meas0 &

   In the measurement results, you can find the pulse width at logic high, the period of the output clock, and the duty-cycle of the output clock in each clock cycle. We measure these results for 50 clock cycles.

7. After Ultrasim is done, start Verdi to debug the output waveform:

   % nWave &

   In nWave, open **chip.fsdb**, and restore the saved signal file **chip.rc**. Then you can see the simulation results. After the ADDCC is locked, the duty-cycle of the output clock (clk_out) will be 50%.



8. Use MATLAB to plot the duty-cycle over clock cycles from the transient simulation measurement results (**chip.mt0**)

   % ./02_run_matlab

   In the following figure, it shows that the duty-cycle of the output clock is changed from 72% to 50% after the ADDCC is locked.

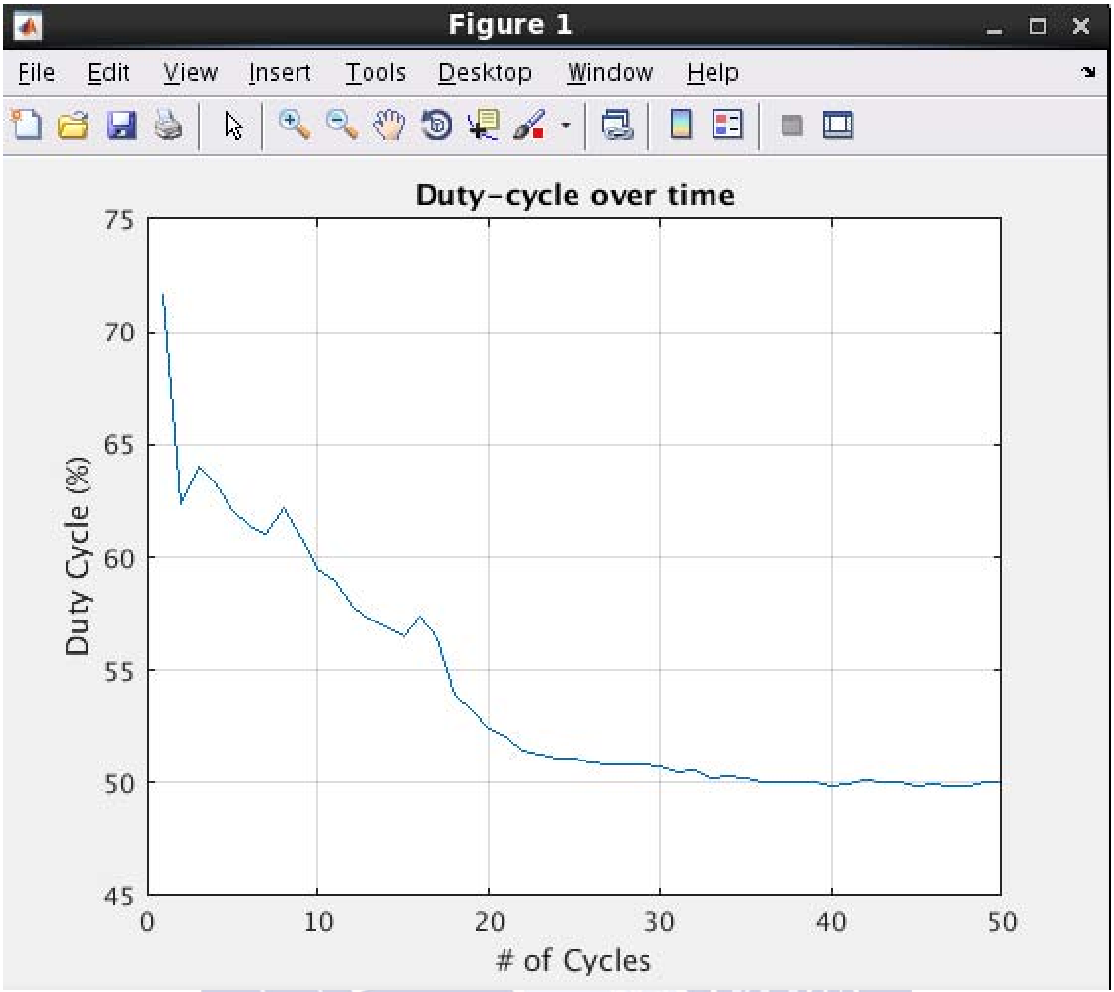


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Duty-cycle over time
Duty Cycle (%)
# of Cycles
75
70
65
60
55
50
45
0
10
20
30
40
50