

Electrical Engineering M216A

Mini-Project

Due date Midnight, Wednesday, November, 26 2003

In this project, you (a group of 2 students) will be building a simple digital gain+high-pass+low-pass filter. This will give you the experience of the entire design process. The filter function and coefficient will be provided. All inputs and outputs will be two's-complement 24-bit fixed-point numbers (integers). You will be responsible for the entire design using each of the CAD tools you have learned in the course. Your goals are to first minimize the area and secondly the cycle time. A well-written report will also be favorably graded. The technology will be the 0.25- μm CMOS that you have been using for your homeworks. A static-CMOS standard cell library is provided for the design of the datapath and controller.

Filter Specification

A digital filter has been designed by a DSP engineer. The filter has three settings: $h=2$ for direct input to output with some multiplication factor (determined by α , where α is a 3-bit input), $h=1$ for high-pass filtering, and $h=0$ for low-pass filtering. The filter can be formulated in the z -domain as follows:

$$h=2: Y=\alpha X; \alpha=-4, -3, -2, -1, 0, 1, 2, \text{ or } 3$$

$$h=1: \frac{Y}{X} = \frac{(8 - 7z^{-1})}{(2 - z^{-1})}$$

$$h=0: \frac{Y}{X} = \frac{(2 - z^{-1})}{(8 - 7z^{-1})}$$

The filter is intended to operate at audio frequencies where $f_s=44.1\text{kHz}$. The low-pass filter setting ($h=0$) filters frequencies above 1kHz, and the high-pass filter setting ($h=1$) will accentuate the frequencies above 1kHz. The DSP architect learned in EE216A that one good way to minimize power is to design the circuit for the highest possible sampling rate, and then drop the supply voltage until the sampling rate is the desired 44.1kHz. Your target as the VLSI designer is to design the circuit for the maximum operating frequency. The DSP architect also chose very *nice* filter coefficients that are all powers of 2's (not so easy to do.) The coefficients and the symmetry in the paths can potentially be leveraged by the VLSI designer. A C-code of the filter is made available for reference.

Furthermore, the architect decided that the filter will use a clock that is at twice the rate of the arriving input samples. A good VLSI designer can use the extra cycle to schedule functional block/register to reduce the area. This will be at a slight cost in speed. Note that there are MANY possible implementations, try to start with a good micro-architecture.

Your grade on the project will depend on both speed and area.

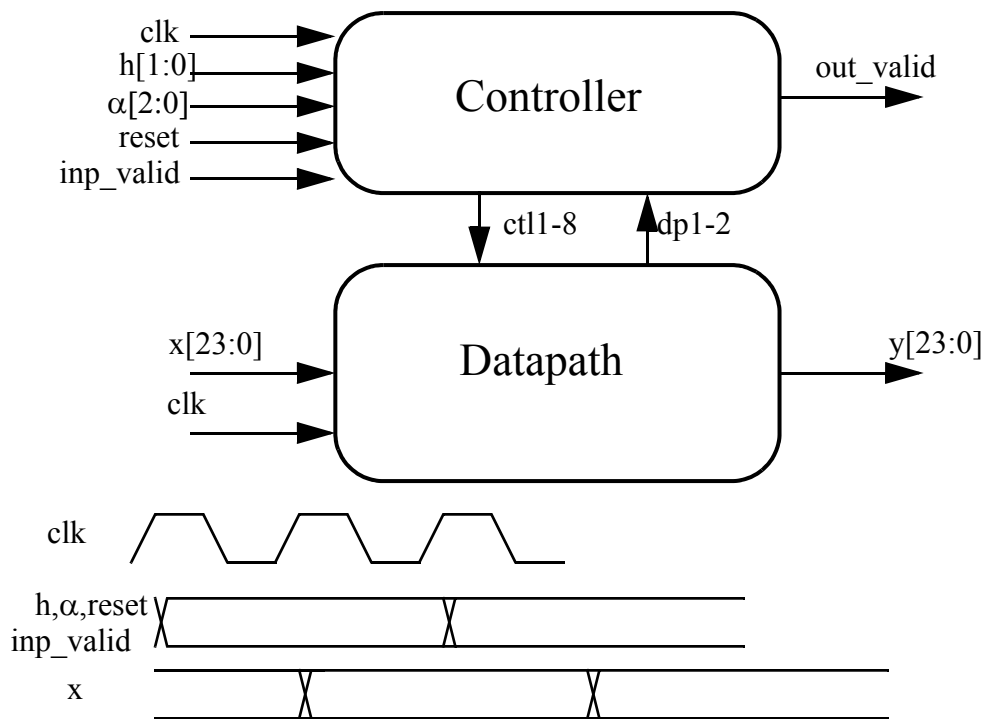
Inputs

- *clock* - the system clock is used in both the controller and the datapath. This clock is at twice the rate of the input. Note that the clock is NOT running at the sampling rate. You may assume that the clock is strongly driven (no need for buffering unless you plan to qualify the

clock with a control signal). The period of *clock* determines the speed performance of your design.

- *x* - a 24-bit data word. The MSB is the sign bit. The input arrives at half the rate of the system clock (you have 2 cycles to generate the data.)
- *h* - a 2-bit control signal that indicates if the circuit is configured as an all-pass (with gain), high-pass or low-pass filter. This signal is an input to the controller and arrives with the data. It is a control signal asserted for 2 cycles.
- α - a 3-bit 2's complement control signal that indicates the multiplication factor.
- *reset* - a 1-bit control signal that resets the controller FSM. The reset is not to be directly used in your datapath. However, during reset, the controller must reset all registers in the datapath to 0. Reset can occur at any point and will remain asserted for 2 cycles.
- *inp_valid* - a 1-bit control signal that indicates whether the input word is valid. The input is applied only to the controller FSM. When the input is invalid, the internal registers must maintain their state so that the digital filter resumes operation when the input is again valid. The signal is also asserted for 2 cycles.

Note that all inputs except *clock* are asserted/deasserted for 2 clock cycles. All control inputs transition 1 cycle before the data transitions. You may assume that the inputs are synchronous to clock and changes immediately after the rising edge of clock. Inputs, *h*, α , *reset*, and *inp_valid* are only available to the controller and not to the datapath. You are also **required** to build a Moore state machine.



Outputs

- *y* - a 24-bit data word that stores the result of the digital filter. The output word need only be valid for 1 cycle (instead of 2).
- *out_valid* - a 1-bit control output (from the controller) that indicates the cycle which the output word is valid. The signal is asserted (HIGH) one cycle before the output word that you

want to validate.

Internal Signals

- *ctl1, ctl2, ctl3, ctl4, ctl5, ctl6, ctl7, ctl8* - control signals from the controller to the datapath. Not all eight signals need to be used.
- *dp1, dp2* - signals from the datapath to the controller. You may not need these at all but it depends on your implementation.

Even though you may not need all these control signals, do not delete them from the ports list.

This will break scripts that we run to verify your design. If you find that you need more than what is provided, contact the teaching staff, we will require you to submit your `filt_top.v` file.

Input/Output

- Datapath inputs, $x[23:0]$, are driven by INVX1 and is timed immediately after the rising edge of the clock (assume that the data arrives from a DFF)
- Controller inputs, $h[1:0]$, $\alpha[2:0]$, *inp_valid*, and *reset*, are also driven by INVX1.
- We will assume that clock is driven *very* strongly (by a large inverter, INVX20).
- Datapath outputs, $y[23:0]$, should drive an additional equivalent gate load of a large inverter INVX4.
- All other signals are considered internal signals.

Hand Calculations

To improve the choice of the cells in the datapath, it may help to do some hand-calculations to estimate the fanout (to try to get the effective fanout to be approximately 4 equivalent inverters). You can either use logical effort or actual delay calculation. For the latter, assume $C_{S/D}$ (for both P and NMOS) = $0.45\text{fF}/\lambda$, C_G (for both P and NMOS) = $0.6\text{fF}/\lambda$, $R_{\text{PMOS-pullup}} = 40\text{k}\Omega - \lambda$ and $R_{\text{NMOS-pulldown}} = 20\text{k}\Omega - \lambda$ (PMOS pulling down and NMOS pulling up have resistances that are double of their respective values.)

Project Tasks

1. Draw an ASM (Moore) of the function and split the algorithm into controller and datapath.
2. Write a behavioral Verilog code that describes the controller and datapath. You should allocate resources wisely to minimize area with little delay penalty. Simulate and verify the functionality of the behavioral Verilog code. Skeletons of the *dpath_beh.v* (behavioral datapath) and *ctrlr.v* (controller) are provided.
3. Use Synopsys to map *ctrlr.v* into gates provided in the *ee216a.db* standard cell library.
4. Custom implement a structural *dpath_str.v* using Verilog. Do not use synopsys to synthesize your structural. The intent is for you to think about how to build a datapath even if it is using the cells in the library. Parts of the structural datapath (the input devices and the load devices to the outputs) has been provided inside the file.
5. Check the functionality of the structural verilog code you wrote by hand (*dpath*) and generated using Synopsys (*ctrlr*). This should function roughly the same as your behavioral code. You will need to include the verilog library in your simulation. At this point, we will not annotate delay so the delay of the gates are unit delay.
6. Use Silicon Ensemble to place and route your structural Verilog (both controller and datapath). Consider constraining your datapath so that the height (assuming that the 24-bit datapath

runs horizontally) is between 24 to 36 rows of cells. This helps keep the datapath slightly more regular.

7. Generate the SDF for the Silicon Ensemble output of the *filt* design and run the back annotated simulation.
8. Determine the total area, and speed of your design. Try to test your design with a realistic sequence to check the speed.

Files, Scripts, and Directories

- *filter.c* and *filter* - a c-program that gives you an idea of what the filter does.
 - *DOT.cshrc* - you need to source this in your *.cshrc*.
- proj/verilog* - a directory containing the Verilog files that you'll use and need in this design.
- *source_beh* and *source_str* - contains the file list that you will be running, *verilog -x -f source_beh/str*, for the behavioral and structural datapaths.
 - *ctrlr.v*, *dpath_beh.v*, *dpath_str.v* - skeletons of the Verilog code you will be writing.
 - *filt.v* - a module containing both controller and datapath modules.
 - *filt_top.v* - This is to test your algorithm much like the c-code. You can alter portions of this file as you wish. You will notice that even though the inputs can resemble the c-code, the outputs will NOT be the same. Be sure to explain why in your report. This is especially true when we toggle *h* and α values.

proj/synopsys - a directory containing the synopsys library.

- *DOT.synopsys_dc.setup* - this file should already be a symbolic link to *.synopsys_dc.setup*
- *ctrlr.scr* - Synopsys script. You may have to change parts of this file to run your design.
- *ee216a.db* - the synopsys synthesis library inside the directory, *synopsys*.

proj/se - a directory that contains the library for the place and route tool, Silicon Ensemble. Note that *cds.lib*, *mylib*, *ee216a.lef*, *ee216a.ctlf*, and *ee216a.gcf* are the files/directories needed to run place and route.

- *setup.mac* - file to run once before doing P&R to import *ee216a.lef* and *ee216a.gcf*
- *ee216aSetColor.mac* - *filt.mac* runs this script when opening SE in order to see higher level metals.
- *filt.mac* - script that runs *sedsm -m=96* for the controller and datapath. The file generates the *filt.sdf*, and *filt.def*.
- *filt.ioc* - scripts that places the I/O pins for the design.
- *dpath_str.v*, *filt.v*, and *filt_top.v* - should be linked in order to run Verilog after P&R for back-annotated simulation.
- *source_sdf* (*source_normal*) - files to use when running Verilog to include (or not include) back-annotation.

/proj/test - a directory that contains sim test vectors that we find useful.

Suggested Progress

- Monday November 10,
Completed behavior Verilog for the controller and the datapath. The behavior datapath is intended for you to verify your implementation of the datapath in structural Verilog. You should have an ASM completed.

- Monday November 17,
Completed structural Verilog for both controller and datapath. This includes you having completed a floorplan for the datapath and the Synopsys for the logic optimization of the controller.

- Wednesday, November 26,

You will electronically submit your design. Your files should work with the *filt_top.v* provided. You are encouraged to test your design with other input to verify the functionality. You will submit an archive file (using the command: *tar -cvf filt.tar <files>*). The tar file should contain: (a) Verilog files: *ctrlr.v ctrlr_gate.v dpath_beh.v dpath_str.v filt.v*, (b) Silicon Ensemble outputs: *filt.sdf, filt.def*.

To submit electronically, you will need to use your **seasnet** account with the following command:
submit eem216a filt.tar

Final Report

Please type the text. For the figures, hand-drawn is fine but I prefer an electronic format. Please do not submit the report electronically, since I want to read it and make marks. The goal is for you to describe the decisions you made during the design and the salient points of the design.

- Performance (1-page max)

Indicate the area and speed of your design. That is all I want from the first page.

- Design Architecture and Issues (5-page max)

Explain your architecture and design in clear and concise wording. Be sure to focus on any considerations that improved the area or performance of the design.

- Simulation Results (2-page max)

If your design works, then this section should be quite short. For the Verilog, do NOT include gazillion pages of Verilog outputs just indication of functionality and speed.

- Layout Floorplan (2-page max)

Submit several pictures. You should also print out a picture of your Silicon Ensemble result of the entire design. Indicate how the metal layers are used. Be sure to indicate the total size of the final design from Silicon Ensemble.

EEM216A Mini-Project
A CMOS Digital Highpass/Lowpass Filter

Huiyu Luo
Wenjiang Shen

Professor C.K. Yang

November 28, 2003

Performance

Speed:

Based on the SE back annotation, the delay of critical path is $2 \times 980\text{ps} = 1960\text{ps}$. If we add some overhead for the clock cycle. We can achieve the clock frequency of about 450Mhz

Design Architecture and issues

Controller

The finite state machine of the controller has two states: **CHECK** and **FILTER**. Shown in Fig. 1 is the ASM for the controller. The structure of the datapath is shown on the following pages.

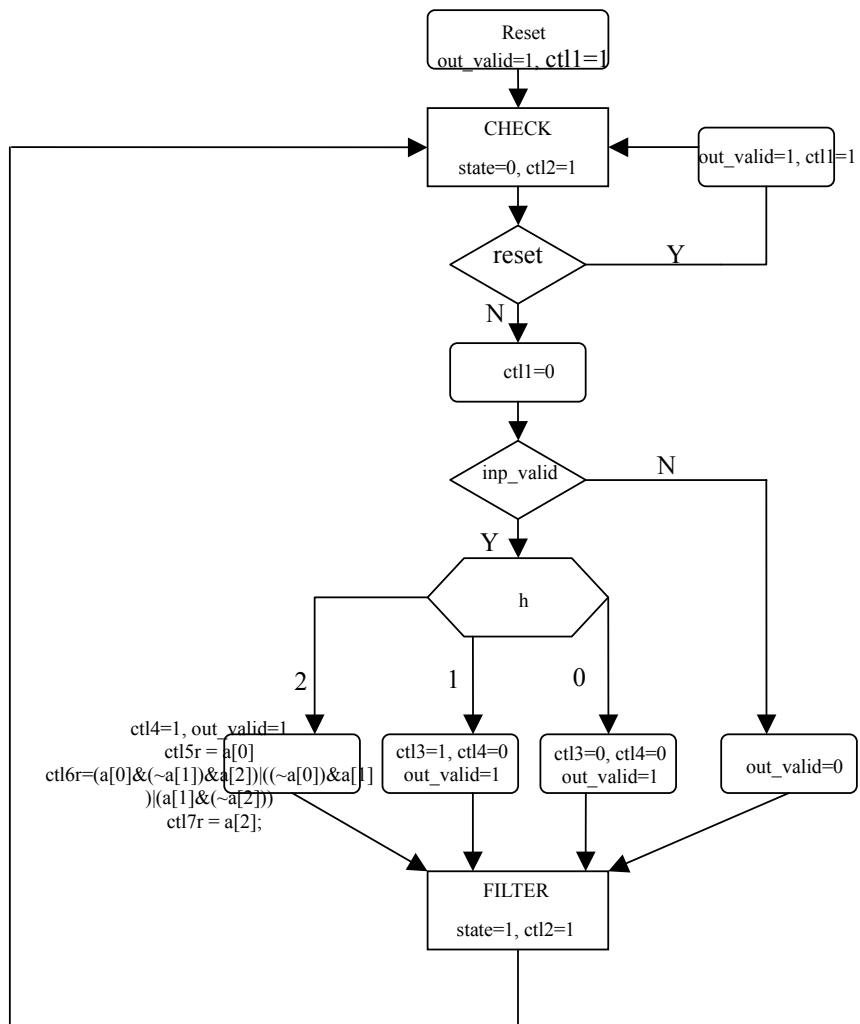


Figure 1 controller's ASM

CHECK State:

The controller checks input signals: reset, input_valid, h[1:0], and $\alpha[2:0]$, then outputs appropriate control signals (ctl1 ~ ctl8) to the datapath and out_valid.

ctl1 indicates reset;

ctl2 indicates inp_valid and which cycle the datapath is at (more explanations later);

ctl3 and ctl4 indicates lowpass, highpass, and allpass filters;

Ctl5 ~ ctl8 provide $\alpha[2:0]$ when h=2.

FILTER State

Change the state back to CHECK.

CONTROLLER DESIGN

All inputs except clock are asserted/deserted for 2 clock cycles. As a result, all control inputs transition 1 cycle before the data transitions and remain for two cycles. When the first clock cycle comes, the controller checks the input signals, and determines the control signals. At second clock cycle, all the control signals except ctl2 (will explain later) remain the same.

When h=2, α only takes values from a limited numbers: -4, -3, -2, -1, 0, 1, 2, and 3. Instead of using a full multiplier, we decided to use shifters and adders to perform the multiplication. To resolve the problem of multiplying with twos-complement numbers, we set ctl7=0 for positive number and ct7=1 for negative number, then, the left two bits, we only consider the absolute value of α , and set ctl5, ctl6 for these two bits. Table 1 lists the control signals corresponding to $\alpha[2:0]$:

α	ctl7	ctl6	ctl5
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
-4	1	0	0
-3	1	1	1
-2	1	1	0
-1	1	0	1

Table 1: control signals for α .

Because some control signals drives a lot of multiplexers, we set their loads to appropriate values in the synopsys script so that larger driving strengths are assigned to the control signals. In addition, to help synopsys carry out the optimization, whenever the control signals are irrelevant, their values are set to x.

Datapath

Datapath basically deals with the three types of filters ($h[1:0]$). We use $ctl4$ to decide if it is $h=2$ case or not. And when $ctl4=0$, we use $ctl3$ to assign low pass or high pass.

$$ctl4 = 1 \rightarrow h = 2 :$$

$$y = \alpha x$$

$$ctl4 = 0, ctl3 = 1 \rightarrow h = 1 :$$

$$y = \frac{(x - x_d) \times 7 + x + y_d}{2}$$

$$ctl4 = 0, ctl3 = 0 \rightarrow h = 0 :$$

$$y = \frac{7 \times y_d + x + (x - x_d)}{8}$$

When $h=2$, $ctl5$, $ctl6$, $ctl7$ signals are used to decide the value of α . As mentioned before, we used 2 shifters (left shift 1 bit and left shift 2 bits) and one adder instead of multiplier to carry out the multiplication. The idea is $\times 2 =$ left shift 1 bit, $\times 3 =$ (left shift 1 bit + original number), $\times 4 =$ left shift 2 bits.

However, all the combinations for $ctl5$, $ctl6$ and $ctl7$ are correct except for $\alpha=-4$ case. In this case, $ctl7=1$, $ctl6=ctl5=0$, if follows the same calculation, it will give a result of $y=0$. so in the controller, we generate another control signal especially for $\alpha=-4$ case. That is $ctl8=ctl5'ctl6'ctl7$. when $ctl8=1$ ($\alpha=-4$), we first calculate $4x$ by a 2 bits left shifter, then get its 2's complements for the final result y .

When $h=0, 1$, from the formula above, there is a multiplier. We use the same idea as in the $h=2$ case, $\times 7 =$ (left shift 3 bits - original number). $\times \frac{1}{2} =$ right shift 1 bit, and $\times \frac{1}{8} =$ right shift 3 bits. This way, again we avoided using the multiplier. Moreover, this method also brings us a chance to share some adders by using multiplexers in the datapath design.

Design for high performance:

We use carry lookahead adder for faster speed. Three levels of abstractions are used. The first level consists of six 4-bit adders, the second level has two blocks each of which has three 4-bit adders, and the highest level computes the carryin ($c12$ and $c24$). This design avoids the carryin ripples through the whole 24 bit before it reaches the last bit at the expense of a little more hardware.

By taking advantage of the fact that two clock cycles are available for computation, we can reduce the number of adders to **two**. First from controller, we generate a

control signal (ctl2), $ctl2 = (inp_valid) \& (\sim state)$. So when input is valid, the ctl2 signal varies in 2 clock cycle, which means we generate another “clock” signal (ctl2) whose period is twice of the clock cycle. This ctl2 signal is used to sequence the adds, so that they can perform different computations in two clock cycles. As a result, each adder is use twice for each input x at lowpass and highpass filter states. After the first clock cycle, the temporary results are stored in two registers (rega and regb), and the second clock cycle use these temporary results to finish the whole computation. The following example is for low pass case (ctl4 = 0, ctl3 = 0), this example shows how this idea works.

At the first clock cycle, because the ctl2 signal has delay, so at clock raising edge, $ctl2=0$, and the first clock cycle result in $rega = x-xd$, $regb=8yd+x$, $yr=(8yd+x)/8$, this value can not pass through ctl2 and clock multiplexes. So this intermediate value will not show at the output. The values in two registers will be used in the second clock cycle. The second clock cycle, all the control signals remain the same except $ctl2=1$, and in this cycle, $rega= x-xd-yd$, $regb=8yd+x+x=xd-yd$, so $yr=regb/8=(7yd+x+x-xd)/8$. and this result will pass through the ctl2 and clock multiplexes to get output y

Fig. 2 is the datapath diagram. Besides the normal output y, the datapath also generates an input to the controller through dp1. This signal asserts if the computation has overflowed. A register can sample the value of dp1 at each clock cycle and output appropriate signal. This part is not required in the project description, but we consider it is essential to any practical applications.

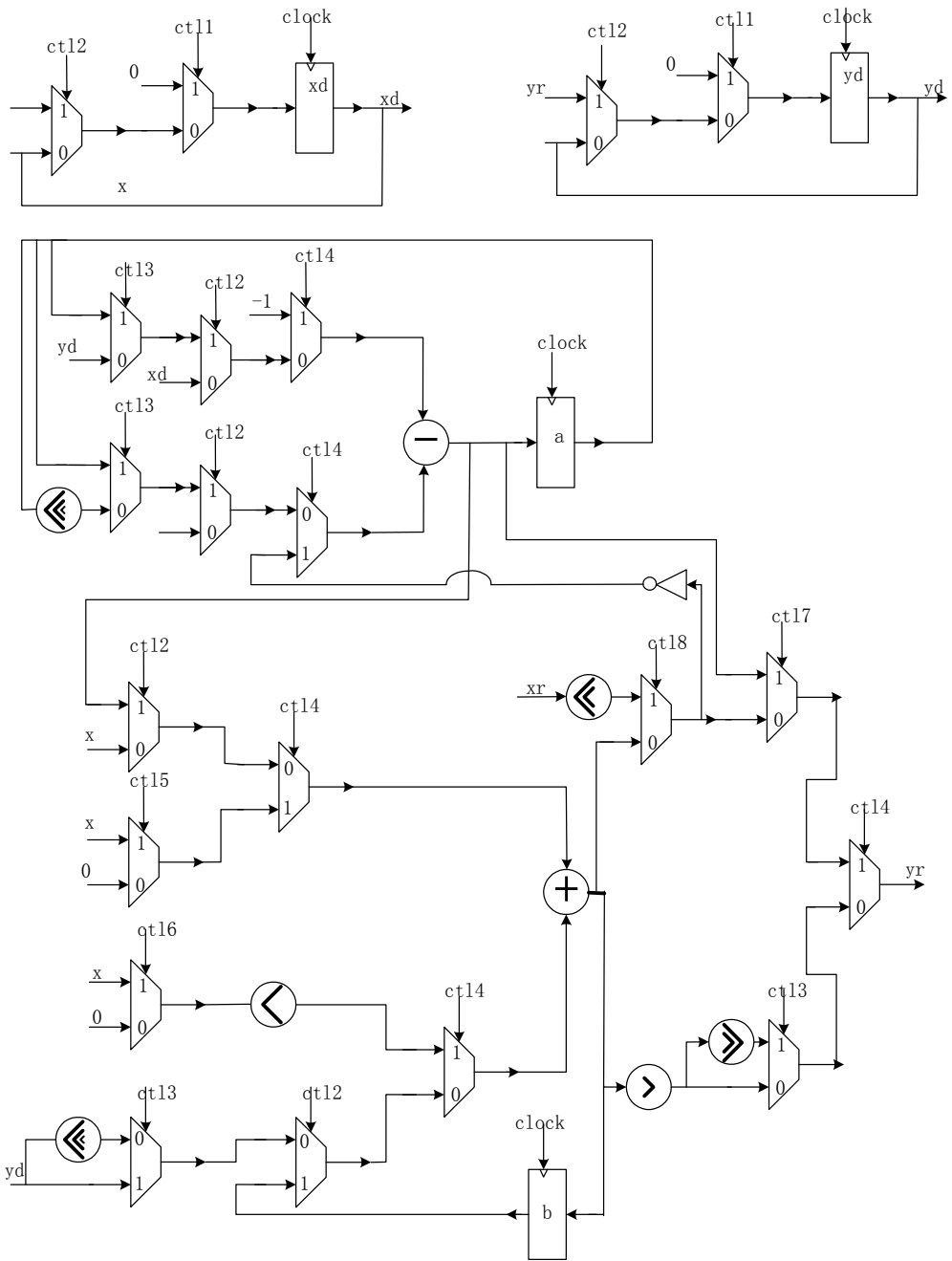


Figure 2 data flow in datapath

Simulation Results

We tested the behavior code, and the simulation turned out to be very successful.

Layout Floorplan

Total size of the floorplan is 380*340

Total metal layers: metal5

Picture 3 shows the overview of SE output

