ECE385 Spring 2023 – Lab 2 Report

Ziyuan Chen, Weijie Liang | ziyuanc3, weijiel4

Specific colors are used to highlight the Register, Computation, Routing, Control units,
the D[3:0], F[2:0], R[1:0], LA, LB, E, RST, *and* CLK *signals throughout the document.*

Introduction

This lab's circuit is a serial logic processor that

- 🟥 takes two 4-bit numbers A and B,
- 🟧 performs one of the (AND, OR, XOR, CONST1, NAND, NOR, XNOR, CONST0) operations
- on *one* pair of bits 🟥 serially shifted out from the A and B registers,
- and 🟩 routes the result back by one of the schemes among (A|B, A|F, F|B, B|A),

with 🟦 switches for parallel loading the registers and executing the operation.

Operating the Processor

To load a specific 4-bit number into one of the A and B registers, the user should

- flip the "Data" switches D[3:0] to specify the desired number, and
- flip the LoadA or LoadB switches to load into the corresponding registers.

To initiate a specific computing and routing operation, the user should

- flip the "Function" switches F[2:0] to specify the desired operation,
- flip the "Routing" switches R[1:0] to specify the desired routing scheme, and
- flip the Execute switch ON and OFF *exactly once* to initiate the operation.

Encoding of the operations and routing schemes are shown in the tables below.

Function	AND	OR	XOR	CONST1	NAND	NOR	XNOR	CONST0
F[2:0]	000	001	010	011	100	101	110	111

Routing Scheme	A|B	A|F	F|B	B|A
R[1:0]	00	01	10	11

High-Level Structure

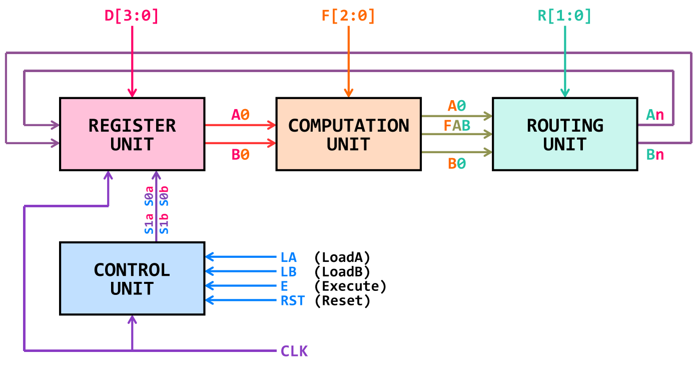
The **register unit (uReg)** consists of two 74195 shift registers storing `A` and `B`. Since we choose the right-shifting mode, two Q_Ds are fed into uComp, and the outputs from uRoute are wired back as serial input. It also incorporates four "Data" bits `D[3:0]` for parallel loading aside from the clock and signals from uCtrl.

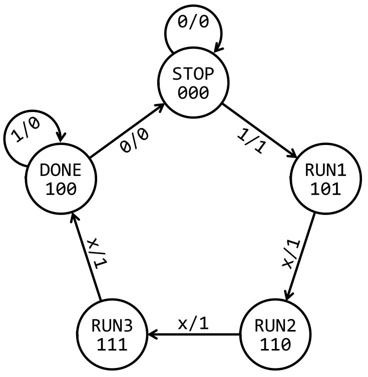
The **computation unit (uComp)** is a standard SOP circuit including one 7400 (`NAND2`), one 7404 (`NOT`), two 7410 (`NAND3`), and one 7486 (`XOR`) chip. It accepts two bits from uReg, performs the computation selected by another three "Function" bits `F[2:0]`, and outputs the calculated logic value *along with the original two bits* to uRoute. (Factually, it was not until the physical circuit was built that we realized there was an easier and clearer implementation based on an 8:1 MUX that uses only four chips in total.)

The **routing unit (uRoute)** is built from a single 74153 Dual 4:1 MUX, which accepts three bits from uComp – the computation result and a copy of the two operands – and routes them back to uReg according to another two "Routing Scheme" bits `R[1:0]`.

The **control unit (uCtrl)** is in essence a standalone 5-state *Mealy* FSM, including two 7474 flip-flops that stores the state and one 7400 (`NAND2`), one 7402 (`NOR2`), one 7404 (`NOT`), and one 7410 (`NAND`) chip that implements the next-state and output logic. It accepts `LoadA`, `LoadB`, `Execute`, and `Reset` as inputs and outputs two "operating mode" signals to each of the two shift registers in uReg. It also shares the `CLK` signal with uReg.

The block diagram below illustrates the inputs, outputs, and high-level unit interconnections in the logic processor. The FSM diagram below illustrates how the states are transitioned in uCtrl. Refer to consequent chapters for fine-grained structures.



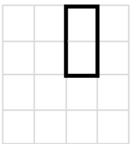


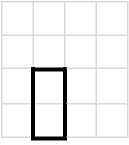
Design Process

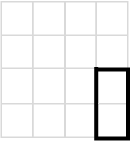
Designs of **uReg** and **uRoute** are straightforward. Each 74195 register accepts four parallel inputs, one serial input from uRoute, and two mode signals plus a clock from uCtrl. One of its four parallel outputs goes into uComp. The 74153 MUX accepts a pair of “select” signals, a total of eight data inputs from uComp, and outputs the routed signals to uReg. STROBEs are kept low, otherwise the MUXes are deactivated.

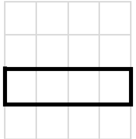
uComp could also have been easier to design using an 8:1 MUX, but we unintentionally detoured and took the SOP approach. Observing that F2 flips the result and is thus supposed to be XORed with f(F1, F0, A0, B0), we solve the following K-map:

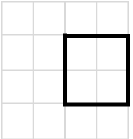
		A0B0			
		00	01	11	10
F1F0	00	0	0	1	0
	01	0	1	1	1
	11	1	1	1	1
	10	0	1	0	1

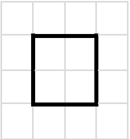

F1'A0B0


F1A0'B0


F1A0B0'


F0F1


F0A0


F0B0

FAB = F2 ^ (F1'A0B0 + F1A0'B0 + F1A0B0' + F0F1 + F0A0 + F0B0)

In the circuit, a 7404 (NOT) prepares F1', A0', and B0'. A 7410 (NAND3) calculates the former three terms while another 7400 (NAND2) calculates the latter three. The two groups are then respectively NANDed together by the second 7410, since no NAND6 is readily available. We then need an additional OR2, which can be transformed into two NOTs (2 slots in the 7404) and one NAND (1 slot in the *second* 7410). There is a final 7486 (XOR).

We once considered merging the latter three terms, which gives F0(F1 + A0 + B0). But not only did this introduce a dedicated 7427 (NOR3), but the latter three terms also experienced *two more gate delays* than the former three! This may amplify the static hazard.

For **uCtrl**, we follow the standard steps of designing an FSM. Compared to its Moore equivalent, a Mealy machine has one fewer state – not that it saves us a state bit, but it looks clearer. The state transition diagram in the former page gives the following truth table:

Q	C1	C0	E	Qn	C1n	C0n	S
0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	1

Here ‘n’ denotes next-state values. Unlisted input combinations produce *don’t cares*.

Again, we solve for the expressions of `Qn`, `C1n`, `C0n`, and `S` with K-maps: (Rows are `EQ` and columns are `C1C0`. Numberings are `00 01 11 10`. Circles are labelled on spot.)

0	X	X	X
0	1	1	1
1	1	1	1
1	X	X	X

`Qn = E + C1 + C0`

0	X	X	X
0	1	0	1
0	1	0	1
0	X	X	X

`C1n = C1C0' + C1'C0`

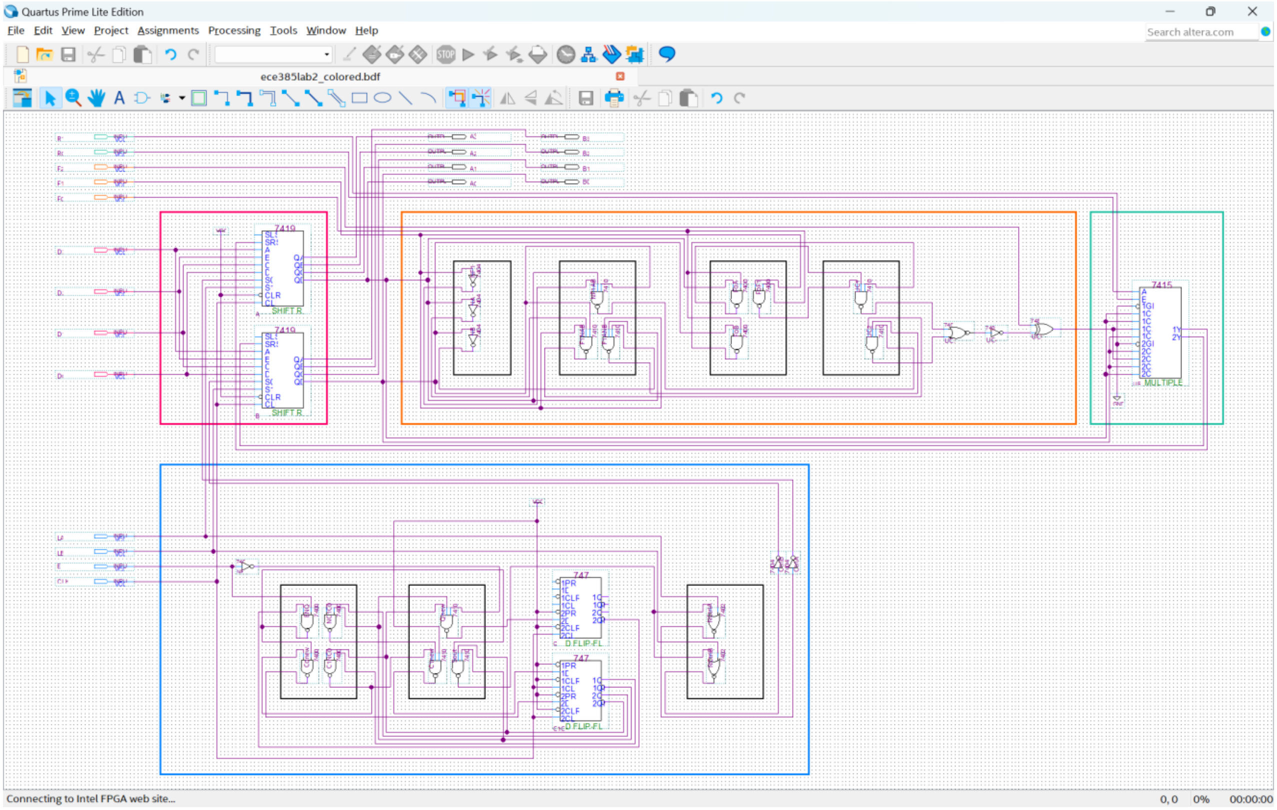
0	X	X	X
0	0	0	1
0	0	0	1
1	X	X	X

`C0n = EQ' + C1C0'`

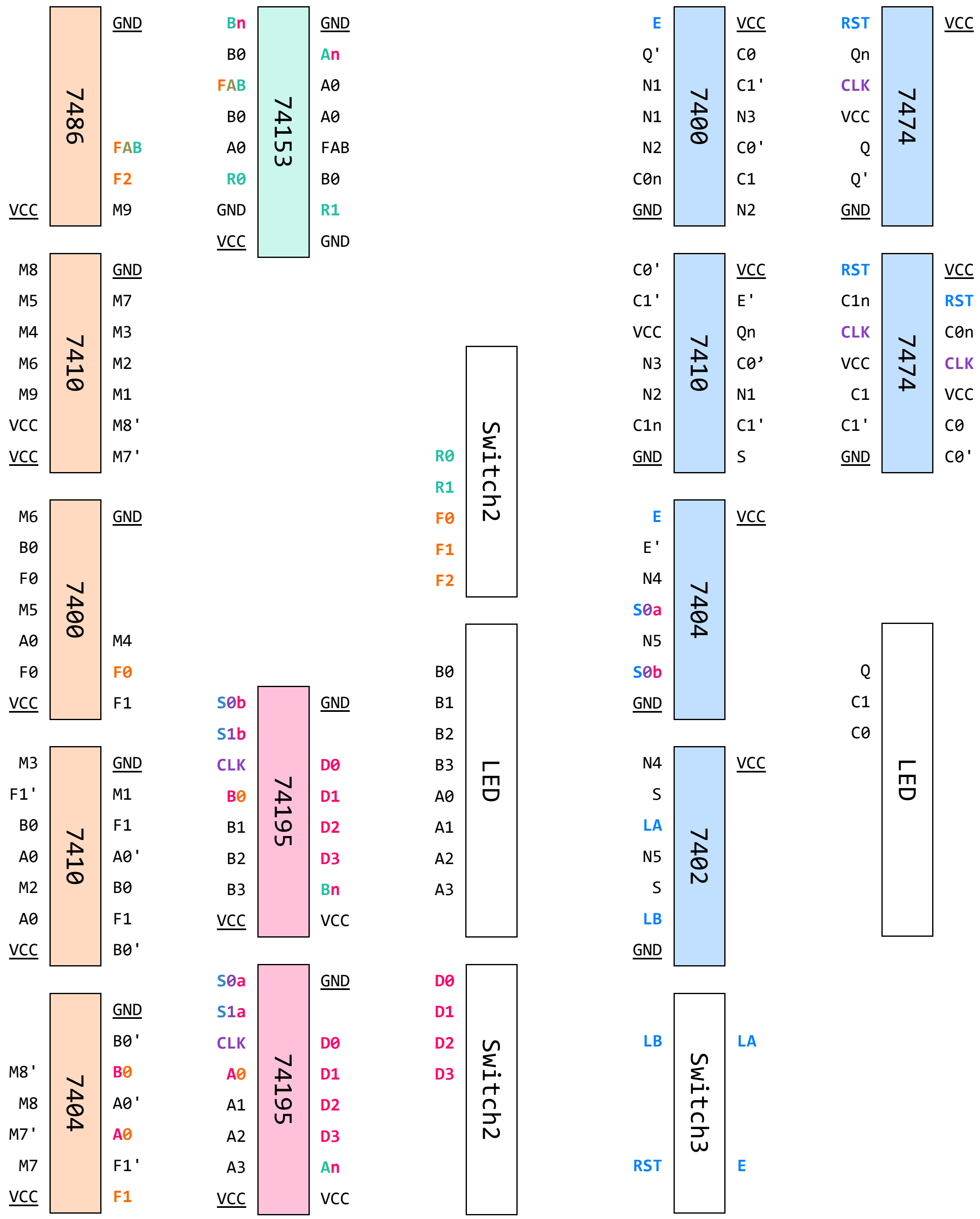
0	X	X	X
0	1	1	1
0	1	1	1
1	X	X	X

`S = EQ' + C1 + C0`

Observe that intermediate terms like `EQ'` and `C1'C0` can be reused. They are described in the layout sheet to be followed. Below is a complete circuit diagram covering all four units. Gates are given meaningful names and organized into chips.


Quartus Prime Lite Edition
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
ece385lab2_colored.bdf
7410
SHIFT R
7415
MULTIPLE
747
D FLIP-FL
Connecting to Intel FPGA web site...
0, 0
0%
00:00:00

Layout Sheet (Chips in the left two columns are installed *upside down*)



Notes on the intermediate variables (M1~M9, N1~N5):

```
FAB = F2 ^ M9
    = F2 ^ (M7'M8')'
    = F2 ^ (M7 + M8)
    = F2 ^ ((M1M2M3)' + (M4M5M6)')
    = F2 ^ ((M1' + M2' + M3') + (M4' + M5' + M6'))
    = F2 ^ (F1A0'B0 + F1A0B0' + F1'A0B0 + F0F1 + F0A0 + F0B0)
```

```
S = (N1C1'C0')' = EQ' + C1 + C0
Qn = (E'C1'C0')' = E + C1 + C0
C1n = (N2N3)' = C1C0' + C1'C0
C0n = (N1N2)' = EQ' + C1C0'
S1a = LA, S0a = N4' = LA + S
S1b = LB, S0b = N5' = LB + S
```

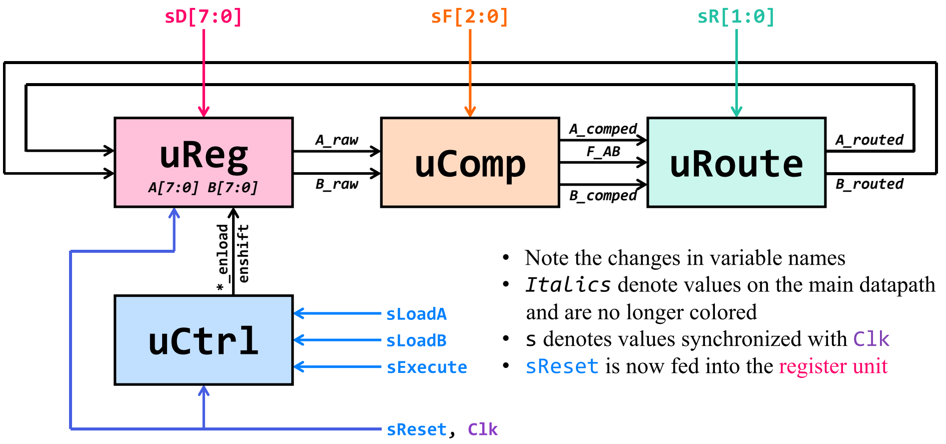
8-bit Extension on FPGA

It should be noted that we extensively rewritten the provided source code, formatting and renaming files and variables in a more informative manner. (‘*’ denotes A or B.)

New filename	Old filename
ureg.sv	Reg_4.sv Register_unit.sv
ucomp.sv	compute.sv
uroute.sv	Router.sv
uctrl.sv	Control.sv
processor.sv	Processor.sv
periphery.sv	HexDriver.sv Synchronizers.sv

New Variable	Old Variable
val	Data_out
load_in	D
enload	Load
*_enload	Ld_*
enshift	Shift_En
*_raw	op*
*_comped	bit*
*_routed	new*

The block diagram below is redrawn using variable names in SystemVerilog and differs from the diagram on Page 2 in subtle ways. Specifically, note the *italics* and color scheme.



- Note the changes in variable names
- Italics denote values on the main datapath and are no longer colored
- s denotes values synchronized with Clk
- sReset is now fed into the register unit

ureg.sv

Inputs: *_routed, sD[7:0], *_enload, enshift, sReset, Clk

Outputs: *_raw, *[7:0]

Description: At enshift, uReg shifts two *_raws as operands to uComp and accepts two *_routeds from uRoute to update the registers at the positive edges of Clk. At *_enload, uReg *synchronously* loads sD[7:0] into A and B. At sReset, uReg *asynchronously* clears the two registers.

Purpose: uReg initializes A and B, keeps track of the operands, and stores the result based on operation and routing scheme.

`ucomp.sv`

Inputs: `*_raw`, `sF[2:0]`

Outputs: `*_comped`, `F_AB`

Description: uComp takes two `*_raw`s, performs the logical operation selected by `sF[2:0]`, and sends the result `F_AB` along with the *original* operands `*_comped` to uRoute.

Purpose: uComp is the core logical unit that performs computations.

`uroute.sv`

Inputs: `*_comped`, `F_AB`, `sR[1:0]`

Outputs: `*_routed`

Description: uRoute takes the result and operands from uComp and routes them to `*_routed` based on scheme selected by `sR[1:0]`.

Purpose: uRoute determines the value to be updated into uReg.

`uctrl.sv`

Inputs: `sLoad*`, `sExecute`, `sReset`, `Clk`

Outputs: `*_enload`, `enshift`

Description: uCtrl sets `*_enload` at the corresponding `sLoad*` and sets `enshift` at `sExecute`. The outputs are supplied to uReg together with `sReset` and `Clk`. *(Note that* `*_enload` *corresponds to Mode* `11` *and* `enshift` *to* `01` *in the TTL circuit.)*

Purpose: uCtrl controls the timing to load the registers with initial values and to start the shifting and computation.

`processor.sv`

Inputs: `D[7:0]`, `F[2:0]`, `R[1:0]`, `Load*`, `Execute`, `Reset`, `Clk`

Outputs: `*val[7:0]`, `*hexU[6:0]`, `*hexL[6:0]`, `LED[3:0]`

Description: The processor takes data and signals and performs operation. `*hex*` inspects *A* and *B* while `LED` inspects control signals.

Purpose: This is the top-level design entity that connects all units.

`periphery[1]`
(HexDriver)

Inputs: `*[7:0]`

Outputs: `*hexU[6:0]`, `*hexL[6:0]`

Description: Four hexadecimal LED drivers each takes 4 bits from *A* or *B* and translate the hex value into a 7-digit LED pattern.

Purpose: The driver displays *A* and *B* for visual inspection.

periphery[2] **Inputs:** D[7:0], F[2:0], R[1:0], Load*, Execute, Reset, Clk

(Synchronizer) **Outputs:** sD[7:0], sF[2:0], sR[1:0], sLoad*, sExecute, sReset

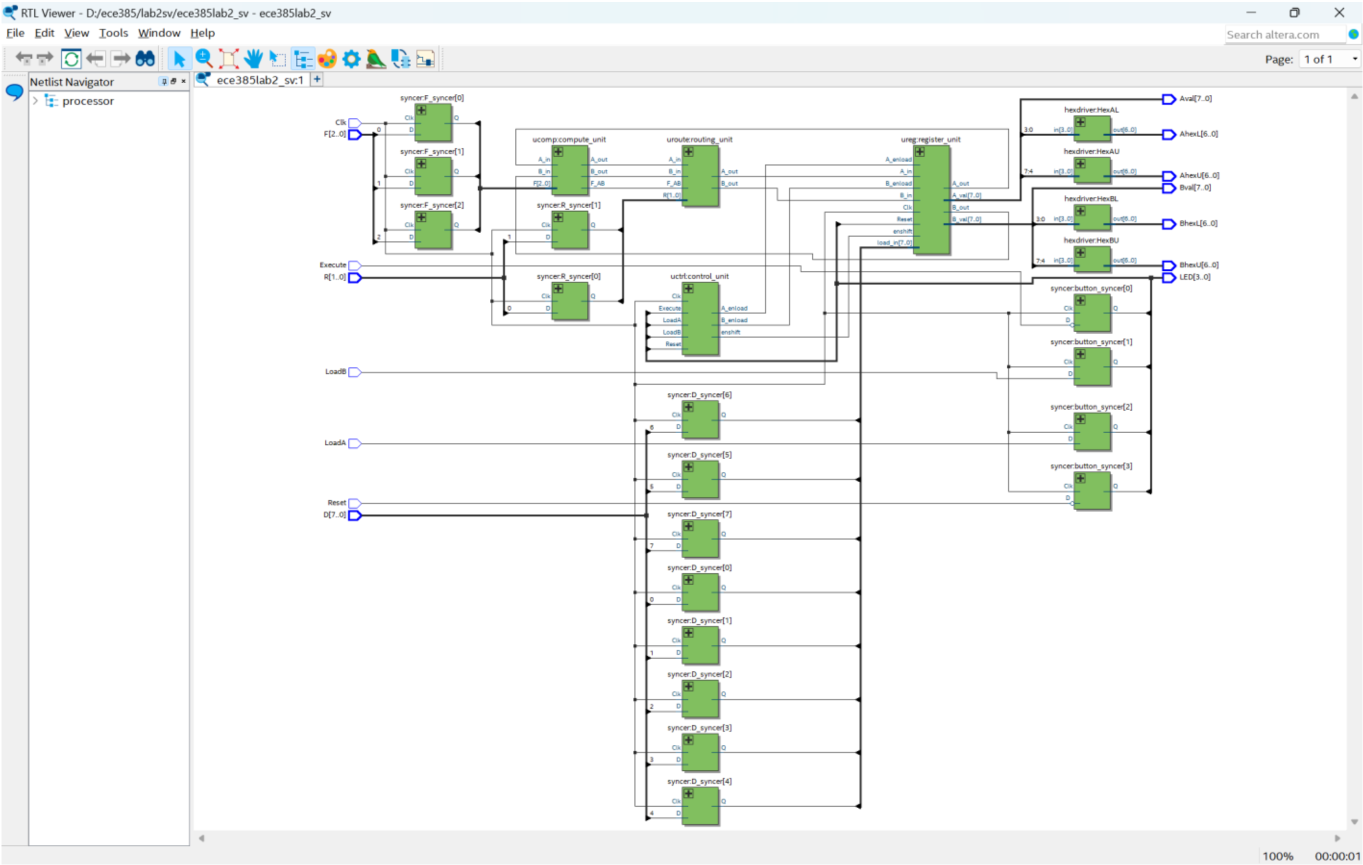
Description: Four flip-flop-based syncers make sure that the control signals only changes at the positive edges of Clk.

Purpose: The syncers align the control signals with Clk so that the whole design remains synchronous.

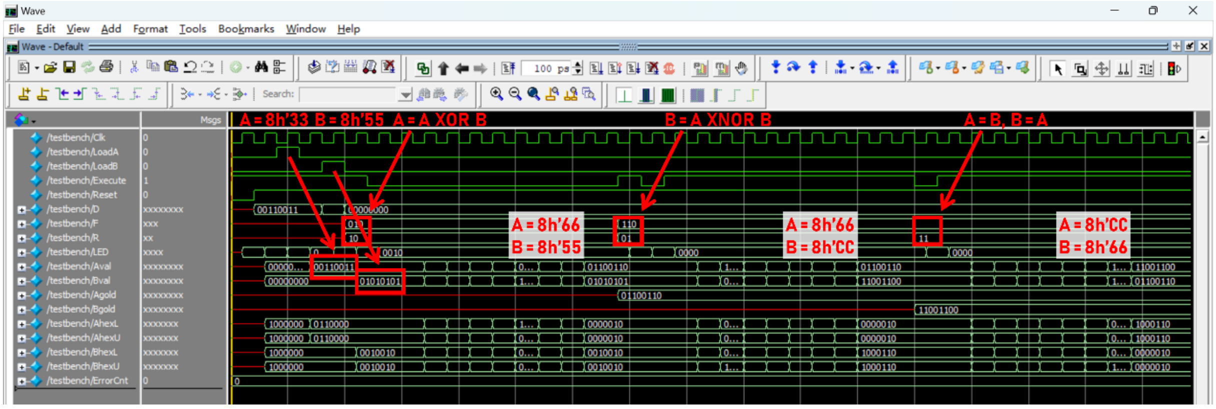
The following table summarizes the changes made to extend the processor to 8 bits.

Filename	Variables or Submodules	Changes
processor.sv	D, sD, *A*, *B*, D_syncer	Extended from 4 to 8 bits ([7:0])
processor.sv	HDAU, HDBU (hexdrivers)	Fed with higher bits ([7:4]) of *A* and *B*
ureg.sv	load_in, *A_val*, *B_val*	Extended from 4 to 8 bits ([7:0])
ureg.sv	val (register value)	Updated with {shift_in, val[7:1]}
uctrl.sv	cur, next (enumed states)	Extended with RUN5~RUN8 (shift 8 times) 10 states are represented by 4 bits ([3:0])

The following diagrams and waveforms are generated by Quartus Prime 20.1.1 Lite.



RTL diagram of the extended 8-bit logic processor



ModelSim waveform of three subsequent operations

The SignalTap ILA simulation trace can be generated by the following steps:

- In “Setup,” add nodes for `reg*{val[7:1], shift_out}`.

  Note that `shift_out` *denotes the LSB since* `val[0]` *is not likely to appear in the list of nodes.*

- Add a node for `Execute` and set trigger condition to “Rising Edge, Basic AND.”

- Set the Clock to `Clk` in “Signal Configuration.”

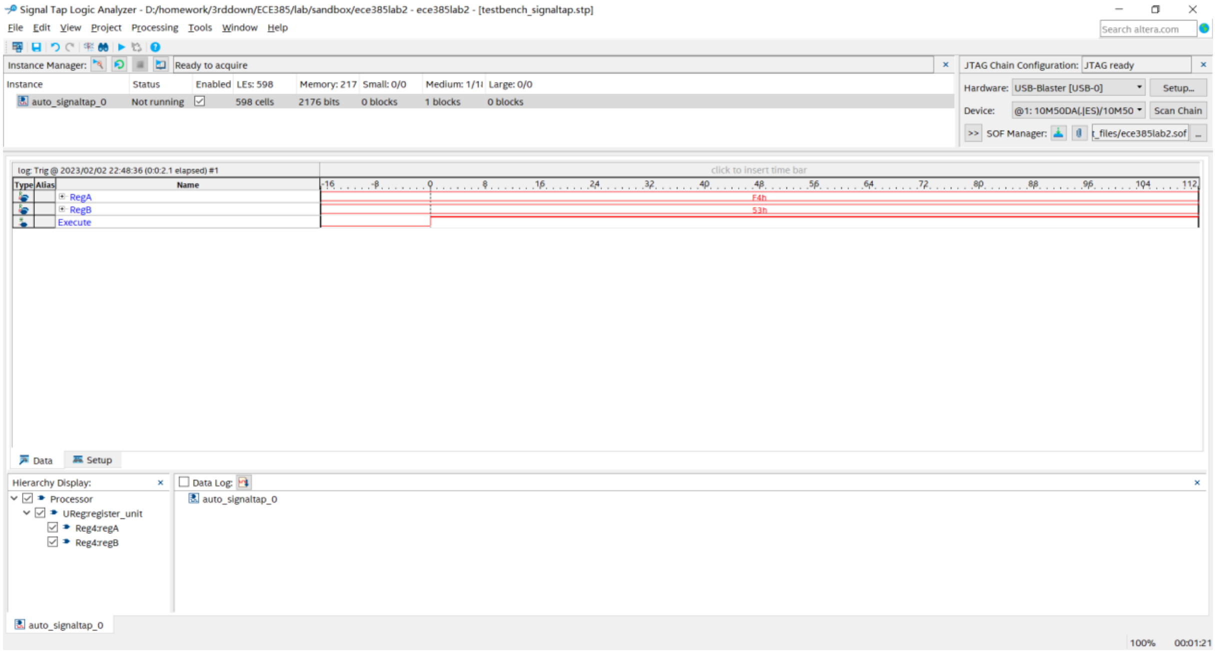
- Set the Hardware and Device to 10M50DAF484C7G in “JTAG Chain Config.”

- Re-compile the project using the SignalTap shortcut.

- Set programming file to `output_files/ece385lab2.sof`.

- Program the device and run analysis. The instance should wait on trigger.

- Manually load the registers `A` and `B` using physical switches on the FPGA board and press `Execute`. Sampling is triggered upon button release.


Signal Tap Logic Analyzer - D:/homework/3rddown/ECE385/lab/sandbox/ece385lab2 - ece385lab2 - [testbench_signaltap.stp]

Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 598	Memory: 217	Small: 0/0	Medium: 1/1	Large: 0/0
auto_signaltap_0	Not running	☑	598 cells	2176 bits	0 blocks	1 blocks	0 blocks

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: 10M50DA(.|ES)/10M50

SOF Manager: t_files/ece385lab2.sof

log: Trig @ 2023/02/02 22:48:36 (0:0:2.1 elapsed) #1

SignalTap ILA trace of the operation “A = 8h’A7, B = 8h’53, A = A XOR B”

Post-Lab Questions

- The Simplest Signal Inverter

  It is an XOR gate. An input is inverted if and only if the other input is 1. Compared to NAND gates, XOR gates require fewer chips (only one). There is also an XNOR variant.

A	B	A XOR B	A XNOR B
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

- Modular Design and Testability

  Modular design enables us to test the modules individually. Each module has relatively fewer chips and is easier to debug. Moreover, we can generally expect the top-level design to work smoothly if modules are thoroughly tested before being combined.

- Tradeoffs in FSM Designing

  A Mealy machine's output is determined by both the current state and the input, while a Moore machine's output only depends on the current state. The Moore machine often requires more states (and sometimes more registers) than the equivalent Mealy machine. But the Mealy machine's output may be unstable when the input is unstable (especially without a `syncer`).

- ModelSim vs. SignalTap

  ModelSim performs software simulation on the computer and SignalTap forms actual circuits on the FPGA board before monitoring the signals. ModelSim allows us to customize testbenches and is useful in the *development* phase where a faulty and hazardous prototype may harm the FPGA, while SignalTap is useful in the *verification* of an iterated design.

Conclusion

In Experiment 2, we use two 4-bit shift registers to store data and perform logic computation using shifted bits on the breadboard. To ensure that the registers shift out 4 bits for each button press, we use a finite state machine to control the circuit. The output of the computation unit can be routed back to the registers. We extend our registers to 8 bits on FPGA, tested the design with ModelSim, and monitored the circuit behavior with SignalTap.