8-2012

Comparison of Various Pipelined and Non-Pipelined SCl 8051 ALUs

Jingyi Zhao
University of Arkansas, Fayetteville

COMPARISON OF VARIOUS PIPELINED AND NON-PIPELINED SCL 8051 ALUs

COMPARISON OF VARIOUS PIPELINED AND NON-PIPELINED SCL 8051 ALUs

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

By

Jingyi Zhao
University of Qingdao Technological, P.R.China
Bachelor of Engineering, 2009

August 2012

University of Arkansas

ABSTRACT

This paper describes the development of an 8-bit SCL 8051 ALU with two versions: SCL 8051 ALU with *nsleep* and *sleep* signals and SCL 8051 ALU without *nsleep*. Both versions have combinational logic (C/L), registers, and completion components, which all utilize slept gates. Both three-stage pipelined and non-pipelined designs were examined for both versions. The four designs were compared in terms of area, speed, leakage power, average power and energy per operation. The SCL 8051 ALU without *nsleep* is smaller and faster, but it has greater leakage power. It also has lower average power, and less energy consumption than the SCL 8051 ALU with both *nsleep* and *sleep* signals. The pipelined SCL 8051 ALU is bigger, slower, and has larger leakage power, average power and energy consumption than the non-pipelined SCL 8051 ALU.

This thesis is approved for recommendation to the Graduate Council.

Thesis Director:

_____

Dr. Scott C. Smith

Thesis Committee:

_____

Dr. Jia Di

_____

Dr. Randy Brown

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed _____

Jingyi Zhao

Refused _____

Jingyi Zhao

ACKNOWLEDGMENTS

Special thanks are due to Dr. Scott Smith for all of his help with my thesis. It would be impossible to make it through the semester without his help.

Also, special thanks go out to Dr. Jia Di and Dr. Randy Brown. Thank you for reviewing my thesis and giving me such a lot of good suggestions.

TABLE OF CONTENT

TABLE OF FIGURES

TABLE OF TABLES

I. INTRODUCTION

An Arithmetic Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the Central Processing Unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs. Therefore, to improve the efficiency and decrease the area of ALUs is important.

Currently, most digital circuits are designed using a synchronous approach, which results in chips requiring precise timing of their components. However, as clock rates have significantly increased while feature size has decreased, clock skew has become a major problem. High performance chips must dedicate increasingly larger portions of their area for clock drivers to achieve acceptable skew, causing these chips to dissipate increasingly higher power, especially at the clock edge, when switching is most prevalent. Asynchronous, clockless circuits require less power, generate less noise, and produce less electro-magnetic interference (EMI), compared to the synchronous counterparts, without degrading performance.

Therefore, a variation of NULL Convention Logic (NCL) [1], called Sleep Convention Logic (SCL) was chosen as the design logic for this ALU, as it is a delay-insensitive asynchronous logic that effectively eliminates such timing dependencies. NCL is a delay-insensitive (DI) asynchronous (i.e., clockless) paradigm, which means that NCL circuits will operate correctly regardless of when the circuit inputs become available; therefore, NCL circuits are said to be correct-by-construction (i.e., no timing analysis is necessary for correct operation).

SCL combines the Multi-Threshold CMOS (MTCMOS) [5] technique with NCL to *sleep* the circuit during idle mode, in place of the NULL cycle, to yield a fast ultra-low power asynchronous circuit design methodology, which requires less area and substantially reduced energy usage compared to the original NCL circuit.

In this paper, an SCL 8051 ALU is designed with two versions: one with C/L, registers, and completion all slept using the gates with both *sleep* and *nsleep*; and the other using the new version of the gates without *nsleep*. The two versions are then compared in terms of area, leakage power, and energy per operation for non-pipelined and 3-stage pipelined designs.

II. BACKGROUND

A. Introduction to NCL

NCL is a delay-insensitive (DI) asynchronous (i.e., clockless) paradigm, which means that NCL circuits will operate correctly regardless of when circuit inputs become available; therefore, NCL circuits are said to be correct-by-construction (i.e., no timing analysis is necessary for correct operation).

NCL circuits utilize multi-rail logic, such as dual-rail, to achieve delay-insensitivity. A dual-rail signal, *D*, consists of two wires or rails, D^0 and D^1, which may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state ($D^0 = 1$, $D^1 = 0$) corresponds to a Boolean logic 0, the DATA1 state ($D^0 = 0$, $D^1 = 1$) corresponds to a Boolean logic 1, and the NULL state ($D^0 = 0$, $D^1 = 0$) corresponds to the empty set meaning that the value of *D* is not yet available. The two rails are mutually exclusive, such that both rails can never be asserted simultaneously. This state is defined as an illegal state.

NCL circuits are comprised of 27 fundamental gates. These 27 gates constitute the set of all functions consisting of four or fewer variables. The primary type of threshold gate, shown in Fig.1, is the *THmn* gate, where $1 \leq m \leq n$. *THmn* gates have *n* inputs. At least *m* of the *n* inputs must be asserted before the output will become asserted. NCL threshold gates are designed with hysteresis state-holding capability such that all asserted inputs must be de-asserted before the output will be de-asserted, as shown in Fig 2. NCL threshold gates may also include a reset input to initialize the output. These resettable gates are used in the design of DI registers.

The NCL gate static implementation has four blocks: *reset*, *set*, *hold0*, and *hold1* as shown in Fig.2 (a). Block *set* is the gate's Boolean function that determines when the gate output is

asserted. *reset* determines when the gate output is deasserted, which occurs only when all inputs are logic0. *hold1* is the complement of *reset*, which holds the output at logic1 once it is asserted as long as any input remains asserted. *hold0* is the complement of set, which holds the output at logic0 once it is deasserted, until the *set* logic becomes true to assert the output. Fig.2 (b) shows the *TH23* gate static implementation as an example. The Boolean function of *TH23* is *AB* + *AC* + *BC*. The Boolean functions of the four blocks are shown as follows:

set = $AB + AC + BC$; hold0 = $\overline{A}\overline{B} + \overline{A}\overline{C} + \overline{B}\overline{C}$;

reset = $\overline{A} + \overline{B} + \overline{C}$; hold1 = $A + B + C$;

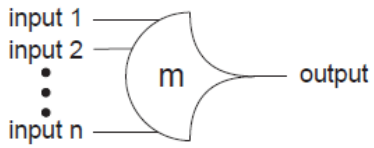

Figure 1. *THmn* threshold gate

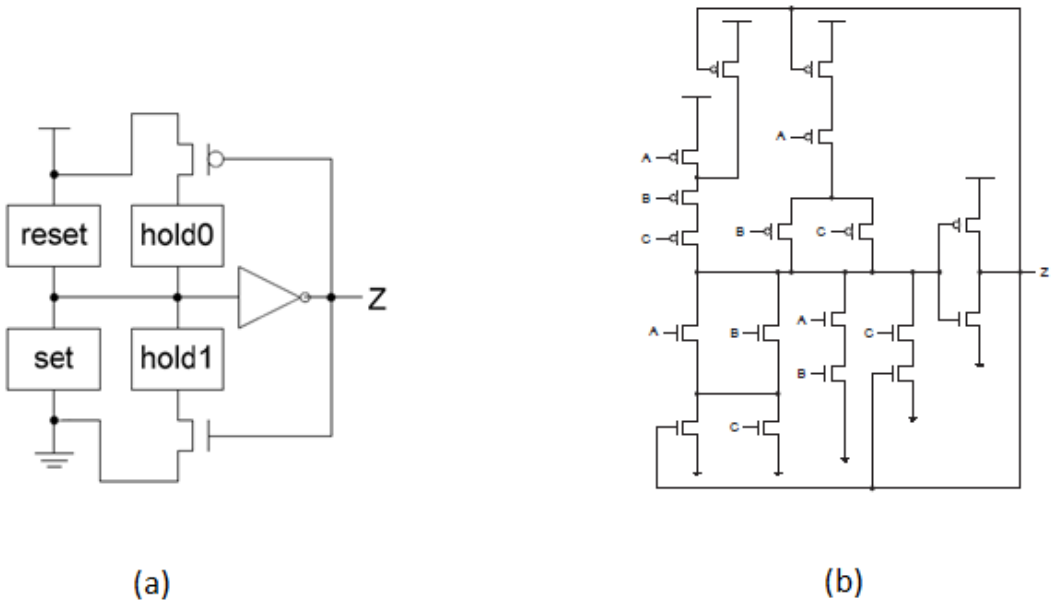

Figure 2. (a) NCL gate static implementation; (b) TH23 gate static implementation

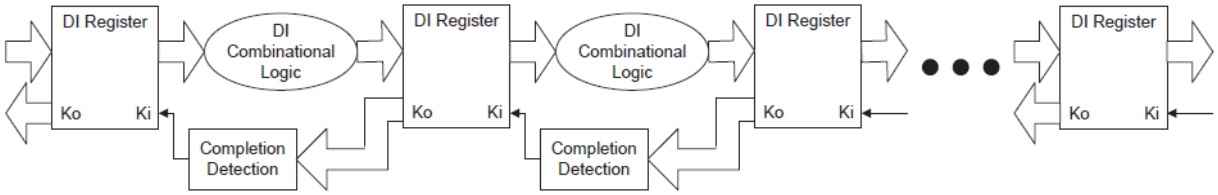

Figure 3. NCL system framework

NCL systems contain at least two delay-insensitive (DI) registers, one at both the input and at the output, and can be finely pipelined by inserting additional registers, as shown in Fig 3. Two adjacent register stages interact through their request and acknowledge signals, K_i and K_o, respectively, to prevent the current DATA wavefront from overwriting the previous DATA wavefront, by ensuring that the two DATA wavefronts are always separated by a NULL wavefront. The acknowledge signals are combined in the Completion Detection circuitry to produce the request signal to the previous register stage, utilizing either the full-word or bit-wise completion strategy.

To ensure delay-insensitivity, NCL circuits must adhere to the following criteria: Input-Completeness and Observability. Input-Completeness requires that all outputs of a combinational circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and that all outputs of a combinational circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL. Observability requires that no *orphans* may propagate through a gate. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Therefore, any gate that transitions must contribute to at least one output transitioning.

B. Introduction to SCL

Multi-Threshold CMOS (MTCMOS) [5] incorporates transistors with two or more different threshold voltages (V_t) in a circuit. Low-V_t transistors offer fast speed but have high leakage, whereas high-V_t ones have reduced speed but far less leakage current. MTCMOS combines these two types of transistors by utilizing low-V_t transistors for circuit switching to preserve performance and high-V_t transistors to gate the circuit power supply to significantly decrease sub-threshold leakage.

SCL uses modified static SCL threshold gates without hold1 to implement the Combinational Logic, static SCL threshold gates with hold1 (SCL1) for the Completion Detection circuitry, and buffers for large fanout signals (e.g., *Sleep/Ki*).

The SCL gate with both *sleep* and *nsleep* is illustrated in Fig. 4, where the high-*Vt* transistors are circled [4]. During active mode, the *Sleep* signal is logic 0 and $\overline{sleep}$ is logic1, such that the gate functions as normal. During *sleep* mode, *Sleep* is logic 1 and $\overline{sleep}$ is logic0, such that the output low-*Vt* pull-down transistor is turned on quickly to pull the output to logic0, while the high-*Vt* NMOS gating transistor is turned off to reduce leakage. The set block, the PMOS transistor of the output inverter, and the output pull-down transistor are low-*Vt*, since they are on the critical path. The other transistors are high-*Vt* to reduce leakage.

The SCL gate without *nsleep* is illustrated in Fig. 5 [2], where the high-*Vt* transistors are circled. It works similar to the SCL gate with both *sleep* and *nsleep*. During active mode, the *Sleep* signal is logic 0, such that the gate functions as normal. During *sleep* mode, *Sleep* is logic1, such that the output low-*Vt* pull-down transistor is turned on quickly to pull the output to logic0, while the high-*Vt* NMOS gating transistor is turned off to reduce leakage. The PMOS transistor

of the output inverter and the output pull-down transistor are low-*Vt*, since they are on the critical path. The transistors located on the bottom of the *set* block are high-*Vt* to reduce leakage, while the rest of the *set* block transistors are low-*Vt* to preserve speed. The other transistors are high-*Vt* to reduce leakage.

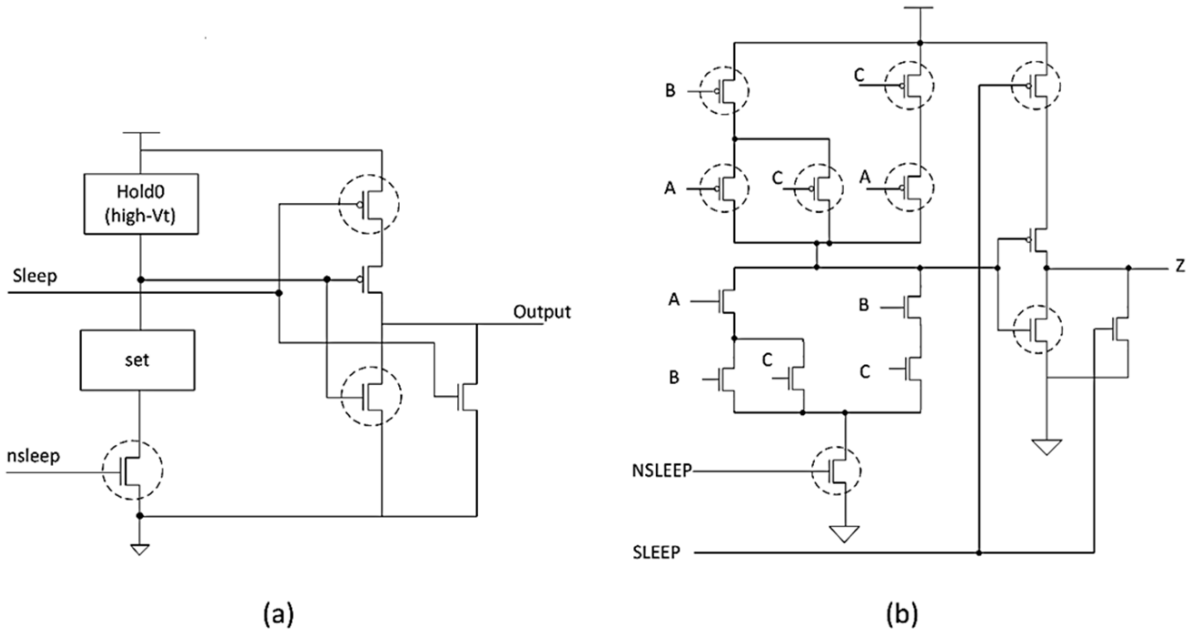

Figure 4. (a) SCL gate structure with *sleep* and *nsleep*, and (b) TH23 implementation

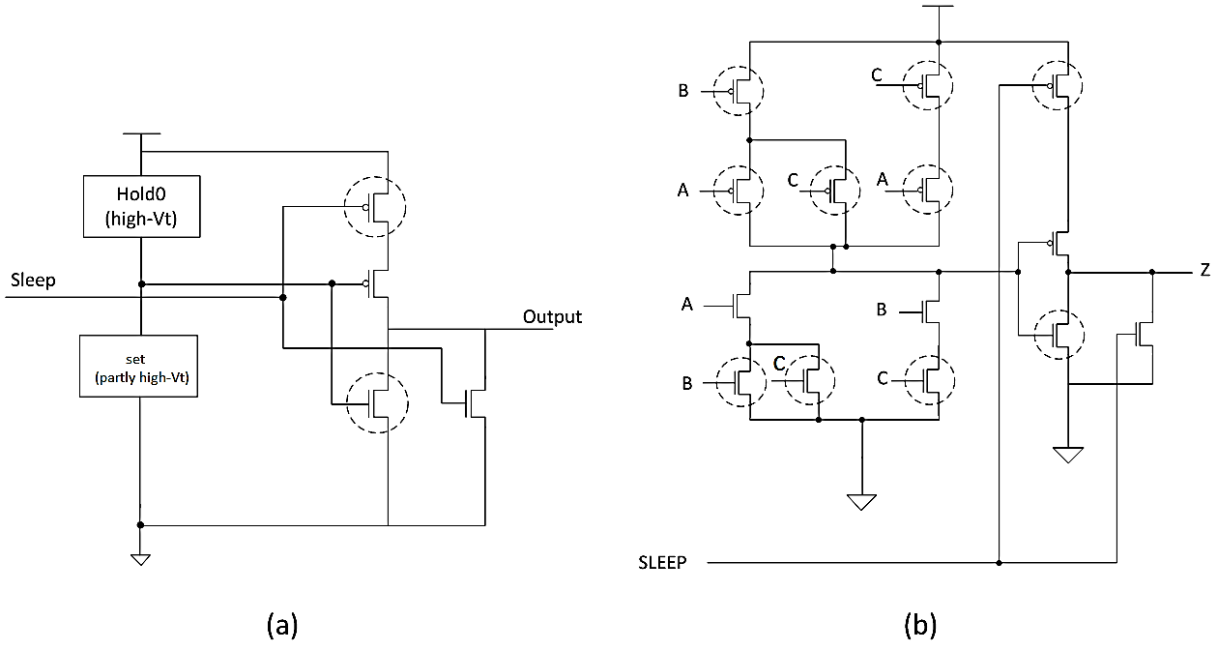

Figure 5. (a) SCL gate structure without *nsleep*, and (b) TH23 implementation

The Early Completion Input-Incomplete SCL architecture, denoted as *ECII*, ensures input-completeness through the *sleep* mechanism, such that input-incomplete logic functions can be used to design the circuit, which decreases area, and power, and increases speed. In SCL, Early Completion [1] is utilized instead of regular completion, as shown in Fig 6, where each completion signal is used as the *sleep* signal for all threshold gates in the subsequent pipeline stage. Early Completion utilizes the inputs of register$_{i-1}$ along with the *Ki* request to register$_{i-1}$, instead of just the outputs of register$_{i-1}$ as in regular completion, to generate the request signal to register$_{i-2}$, Ko_{i-1}.

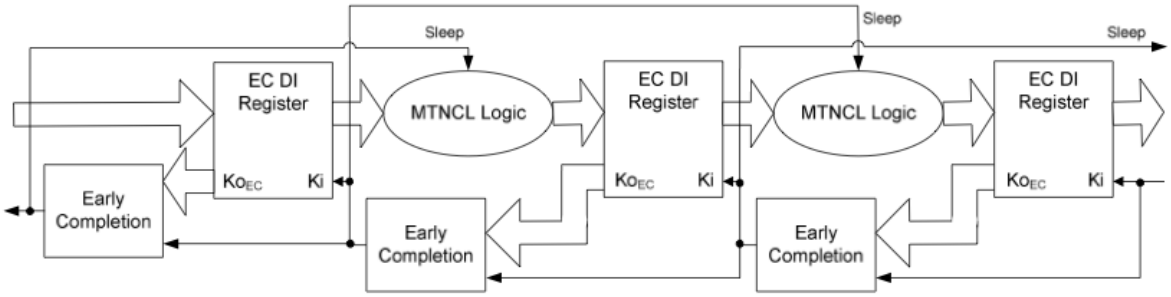

Figure 6. ECII SCL architecture

C. Introduction to ALU

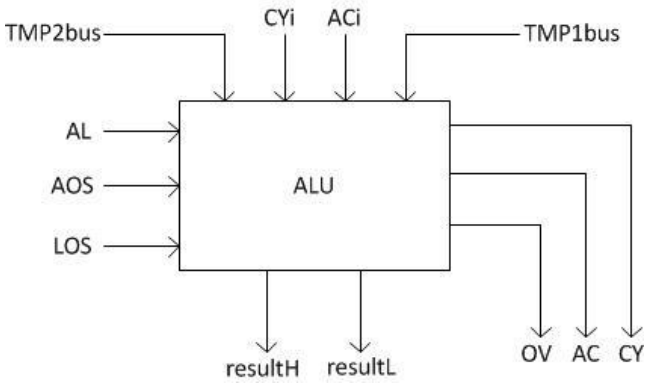

Figure 7. ALU block diagram

An ALU is a necessary component of any general-purpose microcontroller/microprocessor. Fig.7 shows the inputs and outputs of an ALU block. Currently, most digital circuits are designed using a synchronous approach, which results in chips requiring precise timing of their components. Functional problems arise when such circuitry is placed in extreme environments, due to the variances in semiconductor device behaviors. SCL was chosen as the design logic for this ALU, as it is a delay-insensitive asynchronous logic that effectively eliminates such timing dependencies, allowing the design to match the specification while operating under extreme environments, while utilizing substantially less power.

The ALU is capable of executing the following arithmetic operations: add, addc, subb, inc, dec, inc DPTR, mul, div, and DA. It is also capable of executing the following logical operations: and, or, xor, CPL, RL, RR, RLC, RRC, and swap.

The 8051 ALU has three inputs (*AL*, *AOS*, and *LOS*) to choose the instruction to be executed. "*AL*" stands for Arithmetic/ Logic; "*AOS*" stands for Arithmetic Operation Select and "*LOS*" stands for Logic Operation Select. The following table shows the instructions that are chosen by the three inputs [3].

Table 1: Arithmetic instructions

instruction	AL	AOS	LOS	input involved	output involved
add	0	0	x	*TMP1, TMP2*	*resultL, CY, AC, OV*
addc	0	1	x	*TMP1, TMP2, CY*	*resultL, CY, AC, OV*
subb	0	2	x	*TMP1, TMP2, CY*	*resultL, CY, AC, OV*
inc	0	3	x	*TMP1*	*resultL*
dec	0	4	x	*TMP1*	*resultL*
incDPTR	0	5	0	*TMP1, TMP2*	*resultL, resultH*
MUL	0	5	1	*TMP1, TMP2*	*resultL, resultH, OV*
DA	0	5	2	*TMP1, CY, AC*	*resultL, CY*
DIV	0	5	3	*TMP1, TMP2*	*resultL, resultH, OV*

Table 2: Logic instructions

instruction	AL	LOS	AOS	input involved	output involved
and	1	0	x	*TMP1, TMP2*	*resultL*
or	1	1	x	*TMP1, TMP2*	*resultL*
xor	1	2	x	*TMP1, TMP2*	*resultL*

CPL	1	3	0	*TMP1*	*resultL*
RL	1	3	1	*TMP1*	*resultL*
RR	1	3	2	*TMP1*	*resultL*
RLC	1	3	3	*TMP1, CY*	*resultL, CY*
RRC	1	3	4	*TMP1, CY*	*resultL, CY*
SWAP	1	3	5	*TMP1*	*resultL*

According to the values of *AL, AOS, LOS*, we set *AL* as dual rail logic, *LOS* as quad rail logic, and *AOS* as a 6 rail MEAG (Mutually Exclusive Assertion Group). Fig.8 shows the original design of the NCL 8051 ALU.

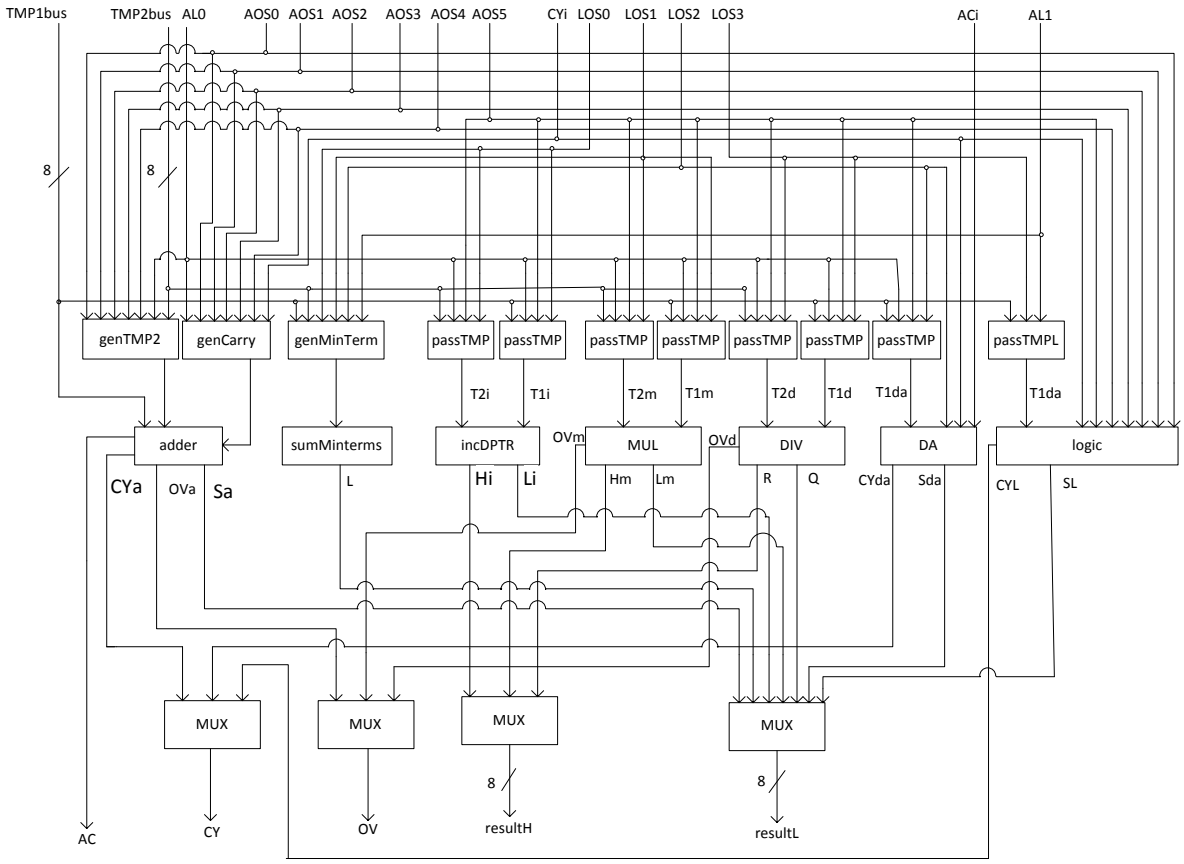

Figure 8. Original NCL 8051 ALU design

III. PREVIOUS WORK

NCL Non-pipelined 8051 ALU Design

The 8051 ALU has 18 different instructions. Based on the type of instruction executed, different inputs and outputs would be DATA and the rest of the inputs and outputs will remain NULL. For example, for CPL, RL, and RR instructions, only the *TMP1bus* input is used and *resultL* will hold the output. On the other hand, for MUL, DIV, and incDPTR, both *TMP1bus* and *TMP2bus* are used and both *resultH* and *resultL* will hold the output.

Because the size of registers cannot be dynamic, a special registration method is used. The idea is based on the fact that by reading the operation determination signals (*AL*, *AOS*, *LOS*), we can determine which outputs will be utilized and therefore which registers must be used to produce the completion signal. The Completion Extension block and Completion block were designed to produce the completion signal. The connection is shown in Fig.9.

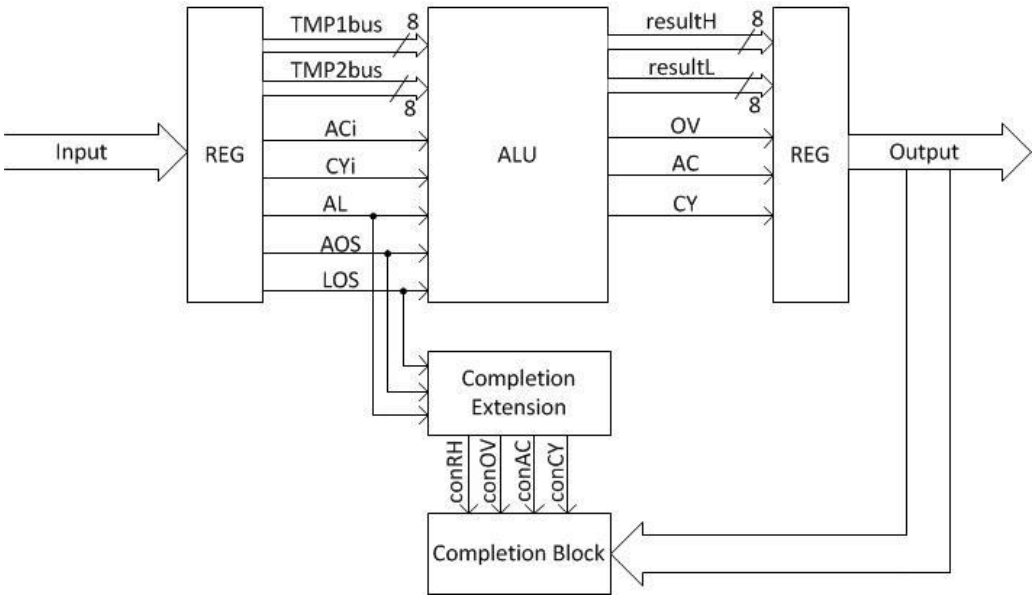

Figure 9. Non-pipelined NCL 8051 ALU block diagram

1. Completion Block design

The NCL 8051 ALU has five outputs: *resultH, resultL*, *OV, AC,* and *CY*. For some operations, not all of the outputs will be used. In this case, *Ko* won't be deasserted when the operation is finished. Therefore, instead of using one 19-bit register for all the outputs, each output has its own register and *Ko* signal. And a specialized completion block is designed to choose which outputs are involved to produce the final *Ko*.

Four control signals (*conRH, conOV, conAC, conCY*) were defined in the design to choose the outputs that are involved in one operation. These control signals will only be asserted if their corresponding output is not involved in the ALU output for the current operation, otherwise they stay low. The design of the output completion block is shown in Fig.10.

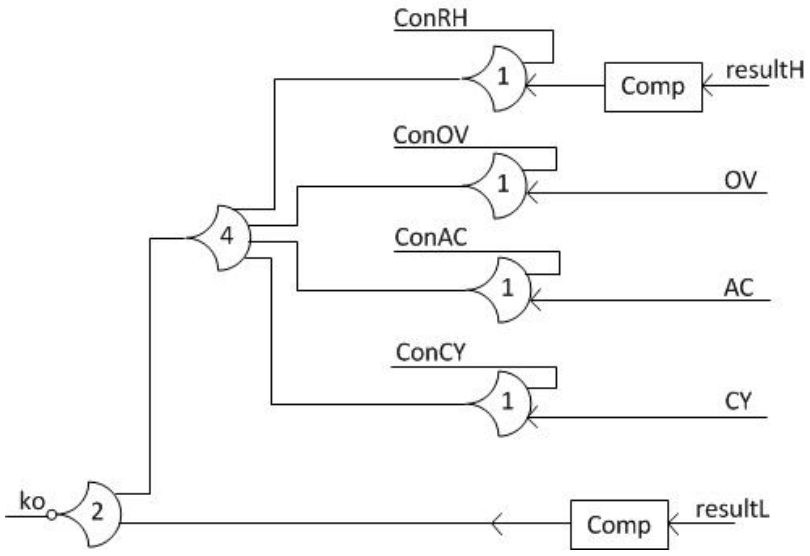

Figure 10. Output Completion Block Design

In this design, a revised register with an uninverted *Ko* is used, as shown in Fig.11 (b).

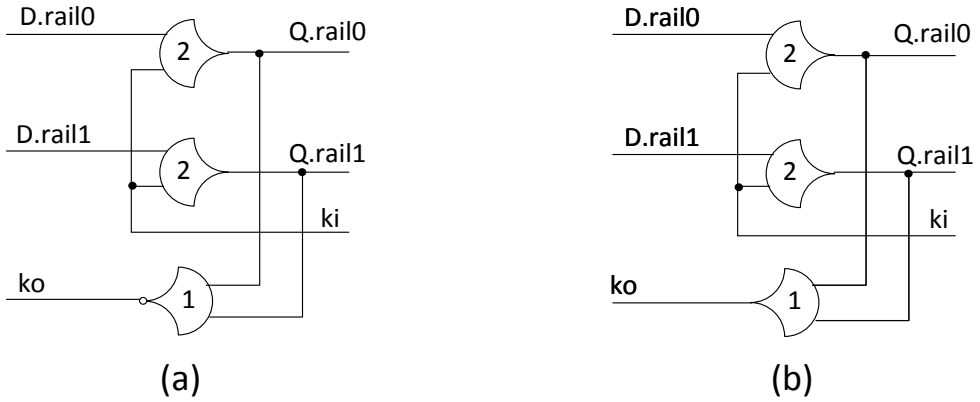

Figure 11. (a) normal register diagram; (b) revised register diagram

2. Completion Extension design

The completion extension block is designed to generate the control signals that choose which inputs will be involved in a given operation.

Table 1 and 2 shows the inputs and outputs that are involved for each instruction.

In the NCL 8051 ALU design, *AL* is set as dual rail logic, *AOS* as six-rail MEAG, and *LOS* as quad rail logic. For example when "mul" instruction is chosen, *AL* = 0, *AOS* = 5, *LOS* = 1, which equals to *AL.rail0* = 1, *AOS.rail5* = 1, *LOS.rail1* = 1.

Based on the inputs and outputs that are involved for each instruction, the formula of each control signal is as follows:

$conAC = AL^1 + AL^0 * (AOS^3 + AOS^4 + AOS^5);$

$conCY = AL^0 * (AOS^3 + AOS^4) + AL^0 * AOS^5 * LOS^0 + AL^0 * AOS^5 * (LOS^1 + LOS^3) + AL^1 * (LOS^0 + LOS^1 + LOS^2) + AL^1 * LOS^3 * (AOS^0 + AOS^1 + AOS^2 + AOS^5);$

$conOV = AL^1 + AL^0 * AOS^5 * (LOS^0 + LOS^2) + AL^0 * (AOS^3 + AOS^4);$

$$conRH = AL^0 * (AOS^0 + AOS^1 + AOS^2) + AL^0 * (AOS^3 + AOS^4) + AL^0 * AOS^5 * LOS^2;$$

IV. APPROACH AND IMPLEMENTION

A. NCL Pipelined 8051 ALU Design

An instruction pipeline is a technique used to increase instruction throughput (the number of instructions that can be executed in a unit of time). Pipelining does not reduce the time to complete an instruction, but increases the number of instructions that can be processed at once.

Each instruction is split into a sequence of dependent steps. The first step is always to fetch the instruction from memory; the final step is usually writing the results of the instruction to processor registers or to memory. Pipelining seeks to let the processor work on as many instructions as there are dependent steps, just as an assembly line builds many vehicles at once, rather than waiting until one vehicle has passed through the line before admitting the next one.

As the goal of the assembly line is to keep each assembler productive at all times, pipelining seeks to keep every portion of the processor busy with some instruction. Pipelining ideally lets one instruction complete in every cycle.

In each stage, data is divided into several groups depending on the components that are involved for each operation. Each stage can be divided into 10 parts: Adder, sumMinterms, incDPTR, MUL, DIV, DA, logic, *AL*, *AOS*, and *LOS*. For each part, one control signal was assigned to decide if the component is involved in the operation.

The stage1 control signals were assigned as follows.

$$Con1\_adder = AL^1 + AL^0 * AOS^5;$$

$$Con1\_genMinterm = AL^0 + AL^1 * LOS^3;$$

$Con1\_DPTR = AL^1 + AL^0 * (AOS^0 + AOS^1 + AOS^2 + AOS^3 + AOS^4) + AL^0 * AOS^5 * (LOS^1 + LOS^2 + LOS^3);$

$Con1\_mul = AL^1 + AL^0 * AOS^5 * (LOS^0 + LOS^2 + LOS^3) + AL^0 * (AOS^0 + AOS^1 + AOS^2 + AOS^3 + AOS^4);$

$Con1\_DA = AL^1 + AL^0 * AOS^5 * (LOS^0 + LOS^1 + LOS^3) + AL^0 * (AOS^0 + AOS^1 + AOS^2 + AOS^3 + AOS^4);$

$Con1\_div = AL^1 + AL^0 * AOS^5 * (LOS^0 + LOS^1 + LOS^2) + AL^0 * (AOS^0 + AOS^1 + AOS^2 + AOS^3 + AOS^4);$

$Con1\_logic = AL^0 + AL^1 * (LOS^0 + LOS^1 + LOS^2);$

The final *Ko1* is generated by the completion block in Fig.12.

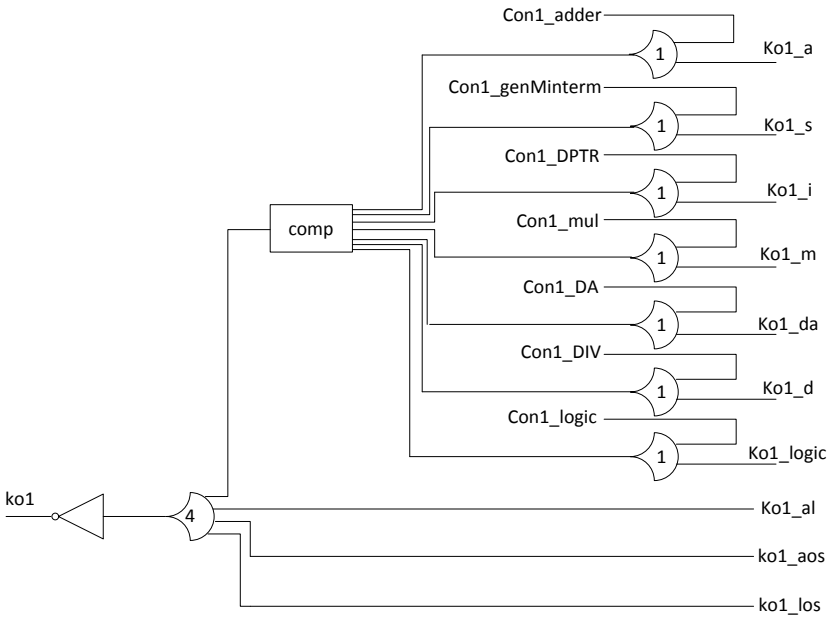

Figure 12. Stage 1 completion block design

Stage 2 is a little different from Stage 1 because for each component, some outputs are not involved in the result dependent on which operation is being executed. For “adder”, *OVa*, *AC*, or

CYa may not be used; for MUL, *OVm* may not be used, for DIV, *OVd* may not be used. Hence, a more detailed extension completion design is required for Stage 2.

There are four outputs in the "Adder" component: *AC, CYa, OVa,* and *Sa*. *AC, CYa, OVa* may not be used in an operation. *Con2_AC, Con2_CYa, Con2_OVa* are assigned to decide if any of the three outputs are used or not.

$Con2\_AC = AL^1 + AL^0 * (AOS^3 + AOS^4 + AOS^5);$

$Con2\_CYa = AL^1 + AL^0 * (AOS^3 + AOS^4 + AOS^5);$

$Con2\_OVa = AL^1 + AL^0 * (AOS^3 + AOS^4 + AOS^5);$

There are two outputs in the "logic" component: CY_L and S_L. CY_L may not be used in one operation. *Con2_CYL* is assigned to decide if CY_L is used or not.

$Con2\_CY_L = AL^0 + AL^1 * [LOS^0 + LOS^1 + LOS^2 + LOS^3 * (AOS^0 + AOS^1 + AOS^2 + AOS^4 + AOS^5)];$

The completion block design of Stage 2 is shown in Fig.13.

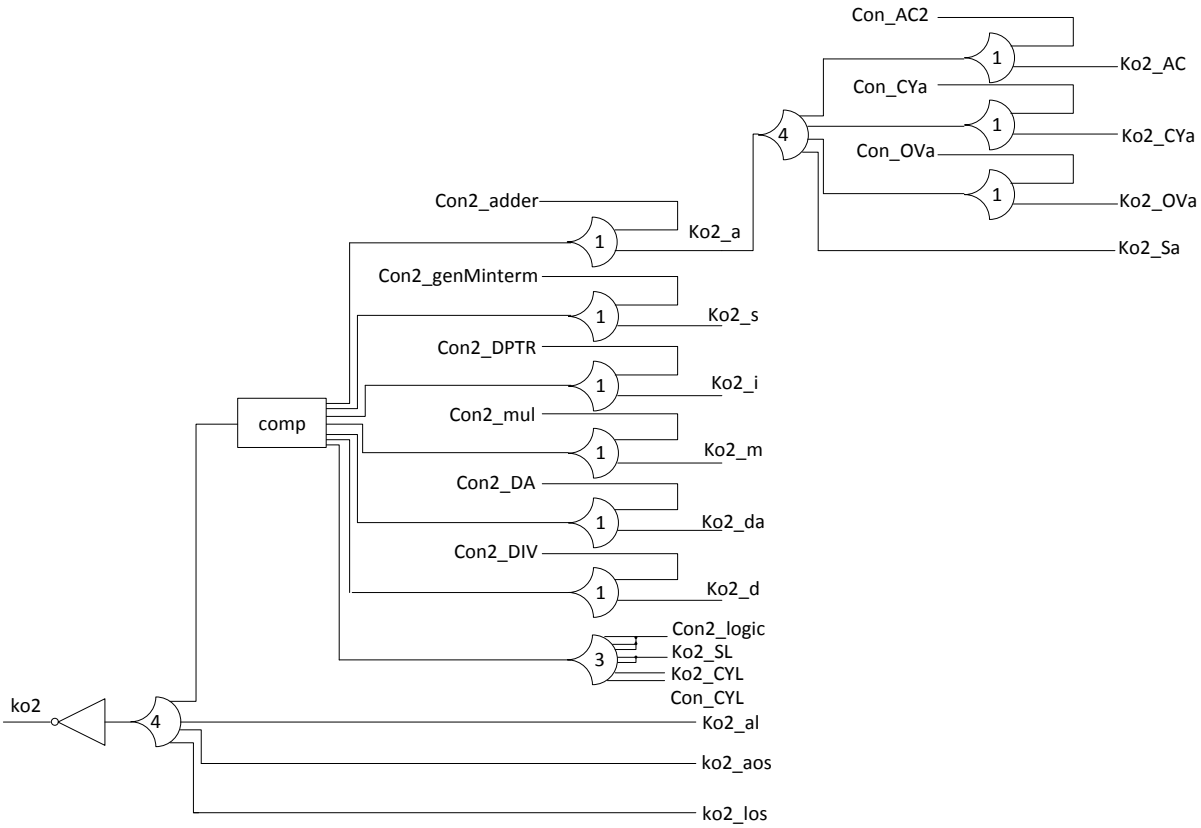

Figure 13. Completion Block for Stage 2

B. SECRII Architecture

In this thesis, an SCL 8051 ALU with *sleep* and *nsleep* and an SCL 8051 ALU without *nsleep* are designed and compared. Each version has two designs: non-pipelined and pipelined. The pipelined architecture that is used in this thesis is called SECRII (Slept Early Completion and Registration Input-Incomplete) [4], as shown in Fig.14.

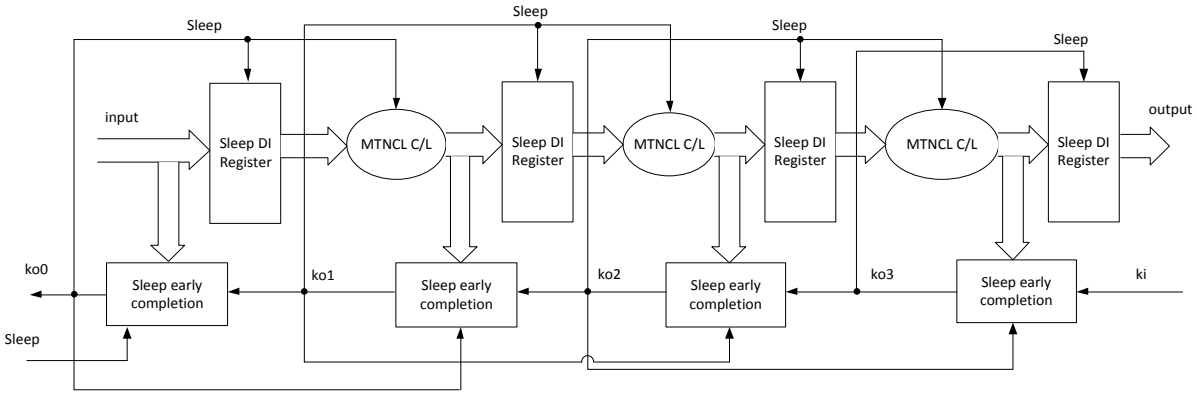

Figure 14. SECRII architecture

The SECRII architecture has the ability to *sleep* both completion components and registers at the same time. It has all the benefits of the SECII [4] in addition to saving more energy and area because of *sleep*ing the registers.

Since the registers are now controlled by the *sleep* signals rather than *Ki* signals, the registers are different in SECRII. In order to save area, the two gates required for passing a dual rail signal share some *sleep* transistors. These gates use the SCL1 style [4] for the completion component, and SCL gates for combinational logic (because the new architecture is similar to FECII, which doesn't let partial NULL propagate).

C. SCL 8051 ALU with Both *nsleep* and *sleep* Signals

The SCL 8051 ALU with both *sleep* and *nsleep* pipeline structure is shown in Fig.15, which utilizes the SECRII architecture with the SCL gates that have both *nsleep* and *sleep* inputs.

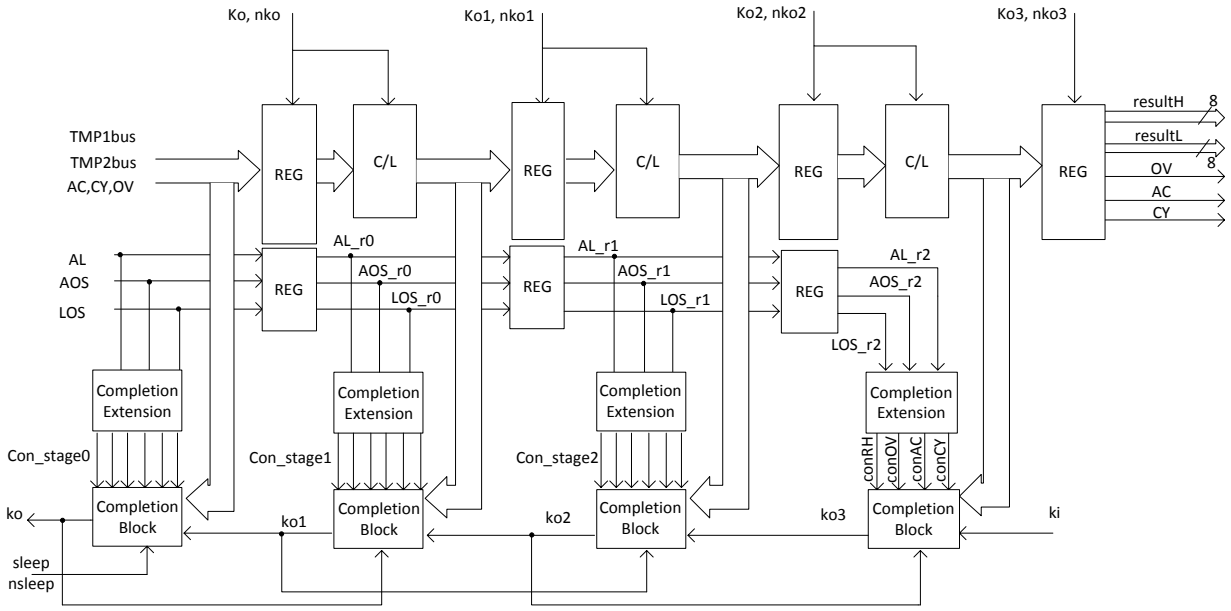

Figure 15. SCL 8051 ALU with *nsleep* and *sleep* pipeline structure

D. SCL 8051 ALU Without *nsleep* Design

The SCL 8051 ALU without *nsleep* pipeline structure is shown in Fig. 16, which utilizes the SECRII architecture with the SCL gates that only have a *sleep* input.

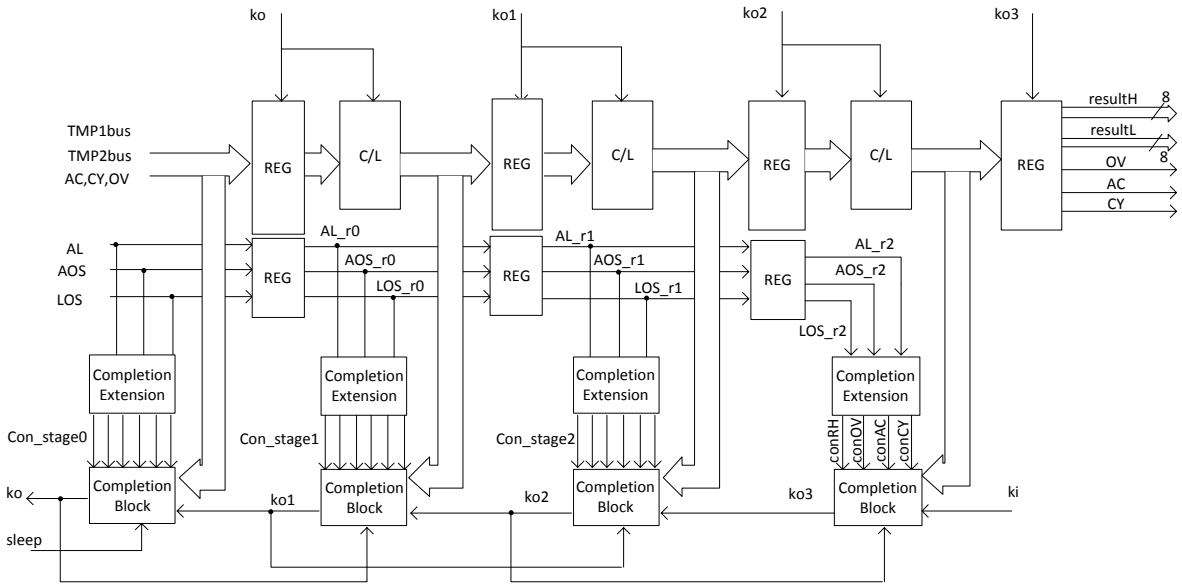

Figure 16. SCL 8051 ALU pipeline structure without *nsleep*

V. SIMULATION RESULTS AND EXPLANATIONS

The simulation was run in Mixed Signal simulation mode in UltraSim simulator. Four implementations of an 8051 ALU utilizing SCL gates were simulated at the transistor level, after inserting buffers, using the 1.2V IBM 8RF-LM 130nm CMOS process, and compared in terms of area, leakage power, and energy per operation, average power and average cycle time (TDD). The SCL 8051 ALU with *nsleep* and *sleep* has lower leakage power, but it is slower than SCL 8051 ALU without *nsleep*. Comparing the SCL 8051 ALU with *nsleep* and *sleep* to the SCL 8051ALU without *nsleep*, the SCL 8051 ALU without *nsleep* is faster, smaller, and requires less energy per operation, but bigger average power than the version without *nsleep*.

A. Area

1. Non-pipelined comparison

Table 3: Non-pipelined SCL 8051 ALU Area Comparison (without buffer)

	gates	NFETs	PFETs	Transistors	Area (um^2)
SCL (without *nsleep*)	1330	7973	7408	15381	301
SCL (*sleep* & *nsleep*)	1336	9276	7922	17198	347

Table 4: Non-pipelined SMTNCL 8051 ALU Area Comparison (with buffer)

	gates	NFETs	PFETs	Transistors	Area (um^2)
SMTNCL (without nsleep)	2079	8740	8175	16915	356
SMTNCL (sleep & nsleep)	1685	9677	8323	18000	399

2. Pipelined comparison

Table 5: Pipelined SCL 8051 ALU Area Comparison (without buffer)

	gates	NFETs	PFETs	Transistors	Area (um^2)
SCL (without *nsleep*)	1775	10962	10199	21161	412
SCL (*sleep* & *nsleep*)	1819	12639	10670	23309	468

Table 6: Pipelined SMTNCL 8051 ALU Area Comparison (with buffer)

	gates	NFETs	PFETs	Transistors	Area (um^2)
SMTNCL (without nsleep)	2756	11961	11198	23159	483
SMTNCL (sleep & nsleep)	2273	13208	11239	24447	535

The SCL 8051 ALU (*nsleep* & *sleep*) is bigger than the SCL 8051 ALU without *nsleep*. Compare the gate structures shown in Fig. 4(a) and Fig. 5(a), there is one more NFET in SCL gates with *nsleep* and *sleep* than SCL gates without *nsleep*. It reduces the area of SCL ALU without *nsleep*.

Before being buffered, the SCL 8051 ALU w/o *nsleep* has fewer gates than the one with both *sleep* and *nsleep*. The reason is the SCL 8051 ALU with both *nsleep* and *sleep* needs many "*invx0"* gates to generate "*nko*" for each stage.

After being buffered, the SCL ALU w/o *nsleep* has more gates than the version with both *sleep* and *nsleep*. For each node, the gates w/o *nsleep* have a little higher load capacitance than the gates with both *nsleep* and *sleep*. The buffer script added several small size buffers "*inverter_a*" for the nodes in the SCL ALU w/o *nsleep.* Buffer "*inverter_a*" is the minimum sized buffer, so it didn't add much area to the overall ALU.

B. Leakage Power

Table 7: SCL 8051 ALU Leakage Power Comparison

	non-pipelined(nW)	3-stages pipelined(nW)
SCL (*nsleep* & *sleep*)	149.7	156.2
SCL (w/o *nsleep*)	160.1	183.6

The leakage power of the version with both *sleep* & *nsleep* is slightly less compared to the version w/o *sleep*.

$P_{leakage} = V_{DD} I_{leakage}$, and leakage current is influenced by transistor width, supply voltage, and transistor threshold voltages. For SCL gates without *nsleep*, they utilize high-*Vt* for the NFETs at the bottom of "*set*" block (Fig.17. b). Compared to the SCL gates with both *nsleep and sleep* (Fig.17.a), the high-*Vt* transistor width has increased. Because of the bigger high-*Vt* transistor width, the leakage current increases a little more than the gates with both nsleep and sleep.

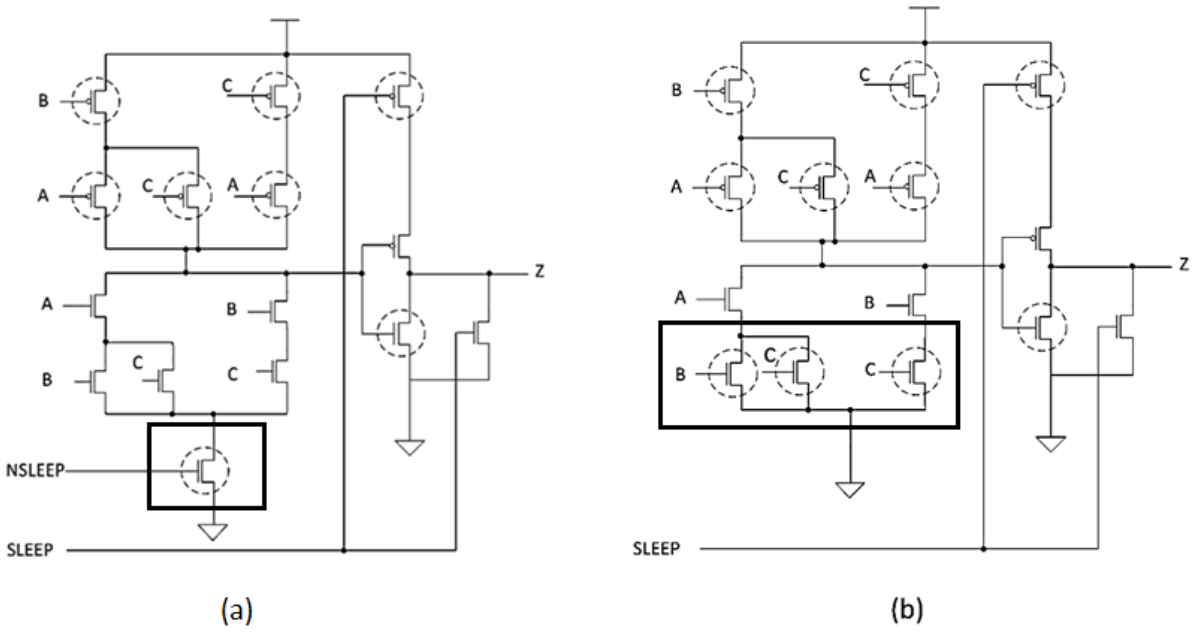

Figure 17. (a) TH23 with *nsleep & sleep*; (b) TH23 w/o *nsleep*

C. Simulation Time and TDD

1. Non-pipelined

Table 8: Non-pipelined SCL 8051 ALU Simulation Time and TDD Comparison

instruction	SCL(w/o *nsleep*)		SCL(*nsleep* & *sleep*)	
	simulation time (ns)	TDD(ns)	simulation time (ns)	TDD(ns)
add	61.6	6.2	85.7	8.6
addc	63.7	6.4	83.0	8.3
subb	63.6	6.4	84.5	8.5
inc	63.2	6.3	84.3	8.4
dec	63.2	6.3	84.7	8.5
incDPTR	58.2	5.8	81.4	8.1
MUL	68.5	6.9	98.9	9.9
DA	56.5	5.7	81.8	8.2
DIV	90.5	9.0	120.8	12.8
and	56.5	5.7	78.5	7.9
or	56.6	5.7	79.9	8.0
xor	56.6	5.7	79.9	8.0
CPL	56.7	5.7	77.9	7.8
RL	56.7	5.7	77.8	7.8
RR	56.6	5.7	77.7	7.8
RLC	56.6	5.7	77.5	7.8
RRC	56.6	5.7	77.4	7.7
SWAP	56.5	5.7	77.5	7.8

2. Pipelined

Table 9: Pipelined SCL 8051 ALU Simulation Time and TDD Comparison

instruction	SCL(w/o *nsleep*)		SCL(*nsleep* & *sleep*)	
	simulation time (ns)	TDD (ns)	simulation time(ns)	TDD (ns)
add	69.8	7.0	117.2	11.7
addc	76.3	7.6	122.8	12.3
subb	78.1	7.8	124.7	12.5
inc	77.5	7.8	124.3	12.4
dec	77.7	7.8	124.8	12.5
incDPTR	74.7	7.5	121	12.1
MUL	86.4	8.6	136.4	13.6
DA	74.0	7.4	121.2	12.1
DIV	97.5	9.8	161	16.1
and	67.5	6.8	120.5	12.0
or	63.2	6.3	116.3	11.6
xor	63.3	6.3	116.3	11.6
CPL	66.4	6.6	113.4	11.3
RL	66.7	6.7	112.9	11.3
RR	66.5	6.7	112.9	11.3
RLC	66.6	6.7	112.8	11.3
RRC	66.6	6.7	112.7	11.3
SWAP	66.6	6.7	112.7	11.3

The w/o *nsleep* version has a shorter TDD. Because the SCL w/o *nsleep* gates have one less NFET than SCL *nsleep* & *sleep* gates, the rise time for SCL w/o *nsleep* gates is shorter than SCL *nsleep* & *sleep* gates.

Unexpectedly, after pipelining the SCL 8051 ALU, the TDD is even longer than the non-pipelined ALU. This is because the added completion extension block needed to choose which signals are involved in each operation.

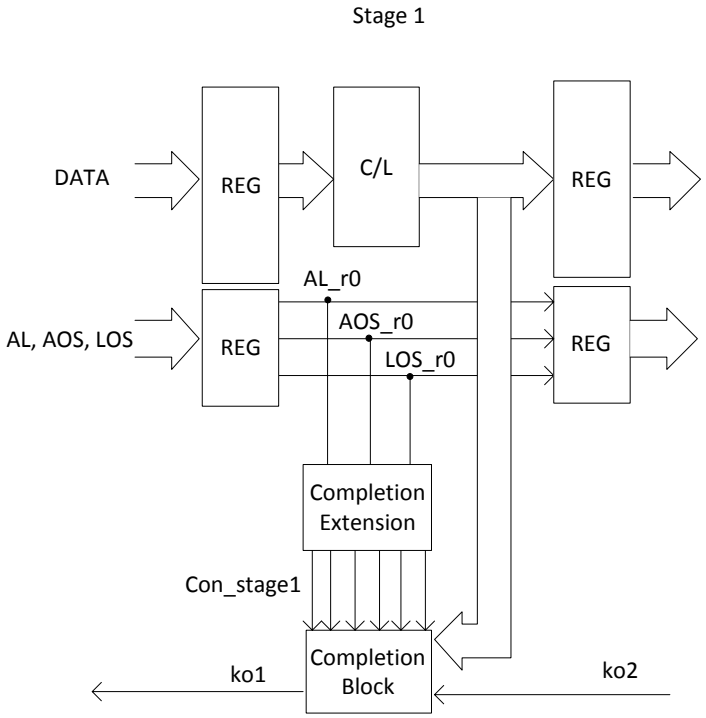

Figure 18. Pipeline stage 1 architecture of SCL 8051 ALU

Take operation “add” as an example, in the first stage of operation “add”, shown in Fig. 18, the worst case delay of the C/L is 553.51 ps while the worst case delay of stage 1 completion extension is 1046.1 ps. The design of “completion extension” extends the simulation time for each stage, in this case the whole simulation time is also extended which explains why the simulation time and TDD for the pipelined ALU is longer than the non-pipelined ALU.

D. Energy for Each Instruction

1. Non-pipelined

Table 10: Non-pipelined SCL 8051 ALU Energy Consumption Comparison

instruction	SCL(w/o *nsleep*) (pJ)	SCL (*nsleep* & *sleep*) (pJ)
add	52.3	87.5
addc	58.1	91.5
subb	57.8	91.9
inc	57.8	90.4
dec	56.8	91.1
incDPTR	60.3	94.1
MUL	71.7	102.8
DA	58.9	91.8
DIV	71.3	104.5
and	56.9	90.1
or	54.6	90.2
xor	54.6	90.7
CPL	53.5	90.0
RL	53.0	89.6
RR	53.6	89.2
RLC	53.2	89.1
RRC	53.2	88.1
SWAP	54.0	90.1

2. Pipelined

Table 11: Pipelined SCL 8051 ALU Energy Consumption Comparison

instruction	SCL(w/o *nsleep*) (pJ)	SCL (*nsleep* & *sleep*) (pJ)
add	59.3	90.1
addc	63.0	100.2
subb	63.3	99.9
inc	62.2	99.5
dec	61.8	98.9
incDPTR	63.9	101.6
MUL	72.2	113.3
DA	63.4	100.4
DIV	73.9	111.9
and	60.8	100.1
or	60.2	98.1
xor	60.1	97.5
CPL	59.6	96.1
RL	59.7	96.4
RR	59	97.7
RLC	59.8	97.4
RRC	59.3	96.9
SWAP	60.4	96.7

From the table above, the SCL version without *nsleep* consumes less energy than the version with both *nsleep* and *sleep*.

$$P = \alpha C_L V_{DD}{}^2 f + \alpha t_{sc} V_{DD} I_{peak} f + V_{DD} I_{leakage},$$

Where $P_{dynamic} = \alpha C_L V_{DD}{}^2 f$.

SCL gates w/o *nsleep* remove the NFET which is driven by *nsleep.* This reduces the load capacitance, which decreases both energy and dynamic power. Also, the total number of transistors in the SCL 8051 ALU with both nsleep and sleep is more than the SCL 8051 ALU w/o *nsleep*, which also decreased the total energy consumption for the version w/o *nsleep*.

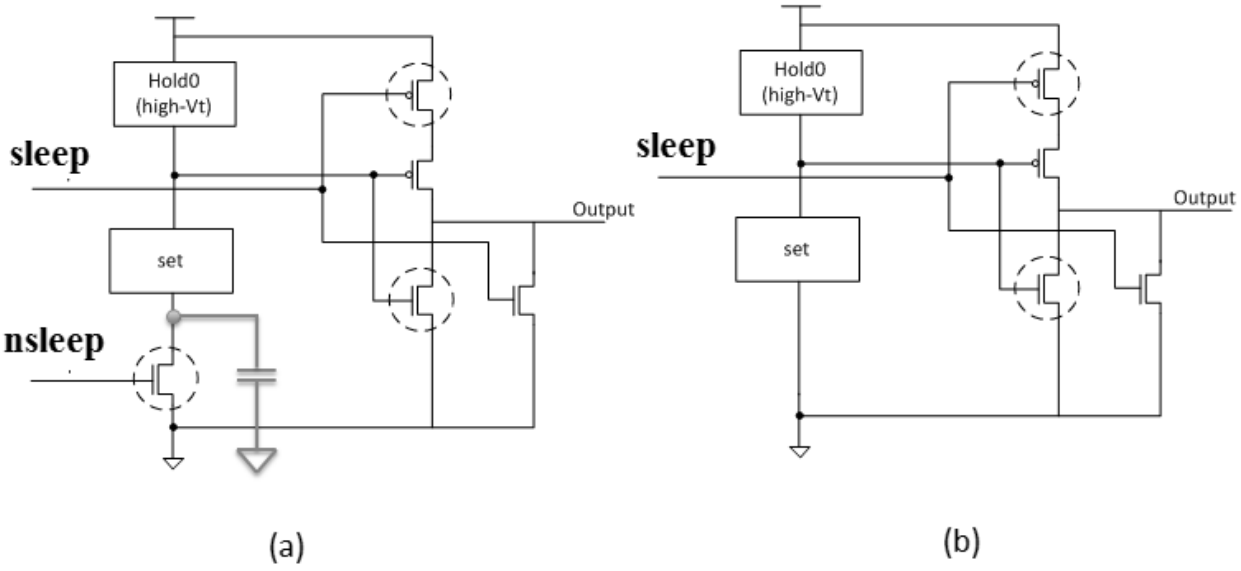

Figure 19. (a) SCL gates with sleep & nsleep; (b) SCL gates w/o nsleep

E. Average power

1. Non-pipelined

Table 12: Non-pipelined SCL 8051 ALU Average Power Comparison

instruction	SCL(w/o *nsleep*) (mW)	SCL (*nsleep* & *sleep*) (mW)
add	0.74	0.77
addc	0.74	0.81
subb	0.75	0.80
inc	0.73	0.80
dec	0.72	0.79
incDPTR	0.79	0.84
MUL	0.82	0.93
DA	0.78	0.83
DIV	0.86	0.96
and	0.83	0.85
or	0.85	0.85
xor	0.85	0.87
CPL	0.79	0.85
RL	0.78	0.85
RR	0.79	0.86
RLC	0.79	0.86
RRC	0.79	0.86
SWAP	0.80	0.86

2. Pipelined

Table 13: Pipelined SCL 8051 ALU Average Power Comparison

instruction	SCL (w/o *nsleep*) (mW)	SCL (*nsleep* & *sleep*) (mW)
add	0.96	1.02
addc	0.99	1.10
subb	0.99	1.09
inc	0.98	1.07
dec	0.98	1.07
incDPTR	1.09	1.15
MUL	1.27	1.35
DA	1.12	1.25
DIV	1.13	1.27
and	1.08	1.14
or	1.06	1.13
xor	1.06	1.13
CPL	1.05	1.15
RL	1.05	1.15
RR	1.04	1.15
RLC	1.06	1.15
RRC	1.05	1.14
SWAP	1.07	1.16

The SCL 8051 ALU w/o *nsleep* has a lower average power than the SCL 8051 ALU with both *nsleep* and *sleep* inputs, even though the version w/o nsleep has a higher leakage power. However, compared to the dynamic power, the leakage power is small and can be ignored. The reason that the SCL 8051 ALU with both *sleep & nsleep* has a higher dynamic power has already been explained in Section D.

The pipelined SCL 8051 ALU has a higher average power than the non-pipelined SCL 8051 ALU because the pipelined version has more gates than the non-pipelined version.

VI. CONCLUSION

This thesis first pipelined a previous design of an NCL 8051 ALU, and then designed both pipelined and non-pipelined SCL 8051 ALU versions, using gates with both *nsleep* and *sleep* signals, and using gates with just a *sleep* signal. The SCL 8051 ALU reduced the leakage power and total energy consumption compared to the NCL 8051 ALU. The SCL 8051 ALU without *nsleep* version provides many advantages such as smaller, faster and lower energy consumption, but has a higher leakage power.

REFERENCES

[1] S. C. Smith and J. Di, Designing Asynchronous Circuits using NULL Convention Logic (NCL), Synthesis Lectures on Digital Circuits and Systems, Vol. 4/1, July 2009, Morgan & Claypool Publishers (doi: 10.2200/S00202ED1V01Y200907DCS023);

[2] L. Zhou, S. C. Smith, and J. Di, "Bit-Wise MTNCL: An Ultra-Low Power Bit-Wise Pipelined Asynchronous Circuit Design Methodology," *IEEE Midwest Symposium on Circuits and Systems*, pp. 217-220, August 2010.

[3] Atmel 8051 Microcontrollers Hardware Manual;

[4] Jia Di, S. C. Smith, “Ultra-low Power Multi-Threshold Asynchronous Circuit Design”, United States Patent, Patent No.: US 8,207,758 B2, Jun 26, 2012;

[5] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J.Yamada, “1-V Power Supply High-Speed Digital Circuit Technology with Multithreshold-Voltage CMOS,” *IEEE Journal of Solid-State Circuits*, Vol. 30/8, pp. 847-854, August 1995.