8-2016

# Design and Analysis of an Asynchronous Microcontroller

Michael Hinds
*University of Arkansas, Fayetteville*

Design and Analysis of An Asynchronous Microcontroller

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Engineering

by

Michael Hinds
University of Arkansas
Bachelor of Science in Computer Engineering, 2009
University of Arkansas
Bachelor of Science in Physics, 2009

August 2016
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Dr. Jia Di
Dissertation Director

| | |
|---|---|
| Dr. James P. Parkerson<br>Committee Member | Dr. Dale Thompson<br>Committee Member |

Dr. Jingxian Wu
Committee Member

## ABSTRACT

This dissertation presents the design of the most complex MTNCL circuit to date. A fully functional MTNCL MSP430 microcontroller is designed and benchmarked against an open source synchronous MSP430. The designs are compared in terms of area, active energy, and leakage energy. Techniques to reduce MTNCL pipeline activity and improve MTNCL register file area and power consumption are introduced. The results show the MTNCL design to have superior leakage power characteristics. The area and active energy comparisons highlight the need for better MTNCL logic synthesis techniques.

## ACKNOWLEDGEMENTS

Special thanks to my wife April for her encouragement and eye for detail. Thanks to my advisor Dr. Jia Di for always being available and encouraging and for setting an example of a great work ethic. Thanks to my classmates for great discussions and all their work to tapeout my designs.

TABLE OF CONTENTS

1 INTRODUCTION ........ 1

1.1 Objective ........ 1

1.2 Design Challenges ........ 1

1.3 Organization ........ 3

2 BACKGROUND ........ 3

2.1 Asynchronous Logic ........ 3

2.2 NULL Convention Logic ........ 5

2.3 MTCMOS Power-Gating ........ 10

2.4 Multi-Threshold NULL Convention Logic ........ 11

2.4.1 Overview ........ 11

2.4.2 Previous Work ........ 13

2.5 MSP430 ........ 15

3 DESIGN FLOW ........ 16

3.1 Logic Design ........ 16

3.2 Buffering ........ 18

4 ARCHITECTURE ........ 19

4.1 Datapath ........ 20

4.2 Selective Sleeping ........ 22

4.3 ALU ........ 24

4.4 Register File ........ 26

4.5 Memory Interface and Interrupts ........ 32

4.6 Timer ........ 33

**4.7 Control Unit** ..... **35**

**5 TESTING** ..... **36**

**5.1 Benchmark** ..... **37**

**5.2 Simulation & Results** ..... **38**

5.2.1 Active Energy ..... 39

5.2.2 Area and Leakage Power ..... 41

**5.3 Analysis** ..... **47**

**6 CONCLUSION** ..... **52**

**7 REFERENCES** ..... **53**

## LIST OF FIGURES

Figure 1: NCL Gate Structure ........ 6

Figure 2: NCL Datapath ........ 9

Figure 3: MTCMOS Power Gating ........ 11

Figure 4: MTNCL Gate Structure ........ 12

Figure 5: MTNCL Datapath ........ 13

Figure 6: MTNCL Design Flow ........ 17

Figure 7: openMSP430 Functional Block Diagram ........ 20

Figure 8: MTNCL MSP430 Functional Block Diagram ........ 21

Figure 9: SS Structure ........ 23

Figure 10: ALU Datapath ........ 26

Figure 11: Traditional 3-Ring Register ........ 28

Figure 12: Modified MTNCL Register File Bit Cell ........ 29

Figure 13: MTNCL MSP430 Register File ........ 31

Figure 14: MTNCL MSP430 Memory Interface ........ 32

Figure 15: MTNCL MSP430 Timer Configuration Register ........ 35

Figure 16: MTNCL MSP430 Top-Level Pipeline ........ 36

Figure 17: Transistor Size Comparison between MTNCL MSP430 and openMSP430 (units in meters) ........ 46

Figure 18: Energy per Operation across MSP430 Functional Blocks ........ 48

Figure 19: Energy Consumption of MTNCL MSP430 Handshaking, Sleep Propagation, and Combinational Logic Circuitry ........ 49

Figure 20: Scaled Comparison of Transistor Area between MTNCL MSP430 ALU and Core Compared to openMSP430 ALU and Core ........................................................................ 51

## LIST OF TABLES

Table 1: NCL Dual-Rail Encoding .... 5

Table 2: 27 Fundamental Threshold Gates .... 7

Table 3: ALU Functionality .... 25

Table 4: Transistor Count Comparison between Modified and 3-Ring Register Files .... 30

Table 5: Timer Constraint and Gate Count Comparison .... 37

Table 6: MTNCL MSP430 Energy Consumption .... 40

Table 7: Average Energy Consumption Across Random Operands .... 41

Table 8: Average Energy Consumption across Random Operands for Indexed Addressing Mode Instructions .... 42

Table 9: openMSP430 Average Energy Consumption across Random Operands .... 43

Table 10: openMSP430 Average Energy Consumption across Random Operands for Indexed Addressing Mode Instructions .... 44

Table 11: Area and Leakage Power Comparison .... 45

Table 12: Transistor Width Comparison between Synchronous and MTNCL MSP430 Microcontrollers and AES Encryption/Decryption Cores .... 52

# 1 INTRODUCTION

## 1.1 Objective

The objective of this Ph.D. dissertation is to develop an asynchronous microcontroller using Multi-Threshold NULL Convention Logic (MTCNL) that is instruction-for-instruction compatible with TI's MSP430 microcontroller, and to give an accurate comparison with a synchronous microcontroller with the same instruction set. Key metrics for comparison are speed, energy consumption, and area.

## 1.2 Design Challenges

In the era of mobile and ubiquitous computing power is a key design constraint. Techniques to reduce energy consumption while meeting area and performance requirements are sought at all levels of the design cycle from architecture to processing. These can be quite challenging given the complexity of modern IC designs. As processes shrink and the number of transistors grows, the variation across a design increases. This means that designers must build with ever-greater margins to ensure that their chips meet timing across PVT corners. The performance and power benefits of process scaling are reduced due to the overdesign required in margining.

Another trend with process scaling is that the ratio of leakage to dynamic power consumption is increasing exponentially. In the past, static power consumption could be ignored as it was overshadowed by dynamic power, but as supply and threshold voltages scale down along with transistor lengths static power can exceed 50% of the total power budget [1]. Clearly, standby current can no longer be ignored, and special techniques such as multi-threshold CMOS (MTCMOS) power gating are required to keep leakage power under control.

Traditional synchronous ICs require large, variation-intolerant, and power-hungry clock distribution networks to carry the clock signal across the design. Designers dedicate large amounts of time and circuit resources to distributing a proper clock signal. In order to decrease power consumption new design methods should be explored.

Synchronous design methodologies continue to dominate the VLSI industry, but asynchronous design is gaining traction due to the aforementioned drawbacks of clock distribution and margining. Currently, there is a lack of commercial CAD tool support for asynchronous design flows. However, there have been several academic projects to develop an RTL-to-GDS flow for asynchronous design [2] [3]. Industry still sees the initial cost of development too high compared to the advantages of asynchronous methodologies. Given the trends above, the advantages of asynchronous are beginning to outweigh the initial overhead of developing a commercial asynchronous flow. More companies will adopt asynchronous designs into their ICs according to the International Technology Roadmap for Semiconductors (ITRS).

In order to encourage industry adoption of asynchronous methodologies it must be proven that asynchronous circuits show an advantage compared to their synchronous counterparts. The power, area, scalability, and design effort of asynchronous circuits should be more favorable compared to synchronous if industry is going to move toward asynchronous design. MTNCL is a relatively new style of clockless asynchronous design that has shown power reduction potential in small designs when compared to synchronous benchmarks. Most of the current MTNCL designs are small compared to real-world ICs. Larger and more complex standalone MTNCL designs are needed in order to prove or disprove this particular asynchronous architecture's advantages over the synchronous design methodology. This dissertation presents a full MTNCL MSP430 microcontroller for comparison with a synchronous MSP430 microcontroller.

## 1.3 Organization

This dissertation is organized into 6 chapters. Chapter 2 gives background on the development of MTNCL technology and frames it in relation to other asynchronous techniques. It also includes an introduction to the MSP430 microcontroller and details on why it was selected as the design target. Chapter 3 details the design flow and tools used. Chapter 4 describes the MTNCL microcontroller's structure and design. The synchronous benchmark circuit, testing methodology, and results are discussed in chapter 5. Finally, chapter 6 concludes the dissertation.

# 2 BACKGROUND

## 2.1 Asynchronous Logic

In synchronous logic a clock signal is used to synchronize data flow through a pipeline. Pipeline stage boundaries are typically constructed using level-sensitive latches or edge-triggered flip-flops known as sequential circuit elements. Designers must ensure that the period – rising (falling) edge to rising (falling) edge – of the clock signal is sufficiently long to account for the propagation of data through the longest path delay any pipeline stage. Additionally, sequential elements require extra margin for data setup and hold to ensure that no metastability problems occur.

In contrast, asynchronous logic uses handshaking to control the flow of data through a pipeline. By eliminating the clock tree, energy savings are possible if the handshaking overhead is not too high. There are two main types of asynchronous logic: Delay-Insensitive (DI) and Bounded-Delay (BD).

BD, like synchronous logic, assumes that the delay through a stage can be accurately determined during the design phase. This is usually done through static timing analysis (STA). BD architecture uses a single-rail encoding scheme and adds two control wires for handshaking between stages. The delay of the datapath must be matched in the control path, and delay elements are used to ensure this. Substantial timing analysis is required, and the maximum performance of the pipeline is a function of the worst-case stage delay. The BD designer must take care to prevent glitches and glitch power consumption in the design. Micropipelines are the most common example of BD logic [4].

DI circuits avoid many of the problems associated with BD. They are correct-by-construction, requiring little or no timing analysis. They operate using a handshaking protocol to control the flow of data, and can therefore achieve average case performance. Additionally, DI architectures often use multi-rail encoding, adding additional states to the data for completion detection. It should be noted that the most useful DI architectures are actually Quasi Delay-Insensitive (QDI). They make the timing assumption that when a wire splits both of its endpoints receive the data signal within negligible delay of each other, where negligible generally means less than a gate delay. This assumption is known as the isochronic forks assumption and is applied within basic components such as a full adder. The isochronic forks assumption is necessary to make the DI architecture Turing complete.

There are several types of QDI architectures. Pre-Charge Half Buffer (PCHB), probably the most well-known, uses dynamic logic, and is synthesized at the transistor level [5]. In PCHB registration and logic are integrated into each gate. Phased-Logic is an automated method to transform a synchronous design to asynchronous; however, it cannot match the performance or power dissipation of a customized asynchronous design [6]. Other DI architectures include

Seitz's Method, Anantharaman's Approach, DIMS, Singh's Method, and David's Method. All of these combine Muller C-Elements with Boolean gates to achieve DI [7] [8] [9] [10].

## 2.2 NULL Convention Logic

NULL Convention Logic (NCL), a QDI asynchronous architecture, is designed at the gate level; however, it requires a custom gate library [11]. Standard EDA flows can be modified to work with NCL designs. NCL logic values include three states: DATA1, DATA0, and NULL. To represent three logic values NCL uses dual-rail encoding where each bit requires two wires. The two DATA states are analogous to Boolean logic values of 0 and 1. The third state, NULL, is used for completion detection and acts as a boundary between wavefronts of data. In this way NCL includes values for data processing and data validation. It is said to be symbolically complete.

Table 1: NCL Dual-Rail Encoding

| | **NULL** | **DATA 0** | **DATA 1** | **INVALID** |
|---|---|---|---|---|
| Wire 0 | 0 | 1 | 0 | 1 |
| Wire 1 | 0 | 0 | 1 | 1 |

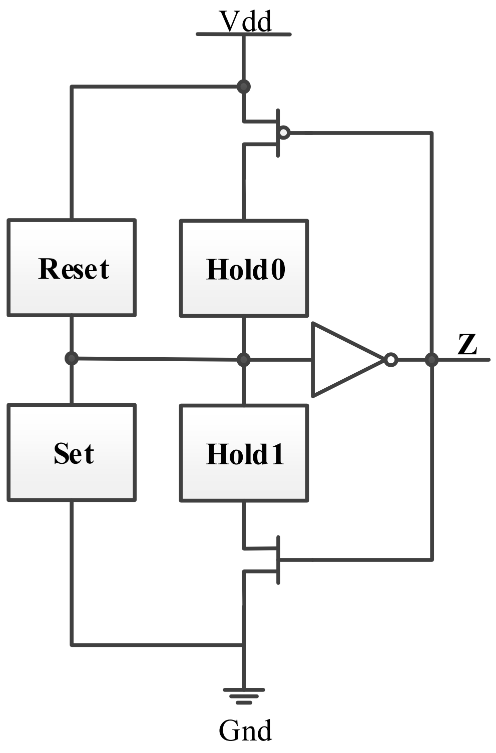

Figure 1: NCL Gate Structure

NCL logic is built from 27 fundamental gates. Each gate is broken into four functional blocks: *Reset, Hold0, Set,* and *Hold1*. Hysteresis is included in every NCL gate. The *Reset* block is responsible for detecting the NULL wavefront – all inputs deasserted – and then deasserting the gate's output. The *Hold1* block is the logical complement of *Reset* and keeps the output asserted until the NULL wavefront is incident on all inputs. *Set* and *Hold0* are also complements of each other. They ensure that the assertion of the output only happens once the gate's logical function is satisfied. Figure 1 shows the structure of the typical NCL gate. It should be noted that for some gate functions transistors are shared between blocks in order to reduce area.

NCL gates are asserted when their threshold number of inputs are asserted, and deasserted when all their inputs are deasserted. They are known as threshold gates and the gate names contain the "*TH*" prefix. There are weighted and unweighted varieties of gates.

Unweighted gates follow the naming convention *THmn* such that $1 \leq m \leq n$, where *n* is the number of inputs to the gate and *m* is the threshold number of asserted inputs required to assert the gate's output. For example, a TH34 gate will be asserted when at least 3 of its 4 inputs are asserted. Weighted gates are those where a certain input can carry greater weight in asserting the output. The naming convention is *THmn*W$x_1x_{2\ldots}x_L$ where $x_L$ determines the weight of the $L^{th}$ input. For example, a TH34w22 applies a weight of 2 to the first and second inputs. In addition to the weighted and unweighted threshold gates there are a few other gates required for completion detection, control flow, and data storage. The complete list of NCL gates along with their functions is given in Table 2.

Table 2: 27 Fundamental Threshold Gates

| **Threshold Gate** | **Boolean Function** |
|---|---|
| TH12 | A + B |
| TH22 | AB |
| TH13 | A + B + C |
| TH23 | AB + AC + BC |
| TH33 | ABC |
| TH23w2 | A + BC |
| TH33w2 | AB + AC |
| TH14 | A + B + C + D |
| TH24 | AB + AC + AD + BC + BD + CD |
| TH34 | ABC + ABD + ACD + BCD |
| TH44 | ABCD |

Table 2: 27 Fundamental Threshold Gates (continued)

| **Threshold Gate** | **Boolean Function** |
|---|---|
| TH24w2 | A + BC + BD + CD |
| TH34w2 | AB + AC + AD + BCD |
| TH44w2 | ABC + ABD + ACD |
| TH34w3 | A + BCD |
| TH44w3 | AB + AC + AD |
| TH24w22 | A + B + CD |
| TH34w22 | AB + AC + AD + BC + BD |
| TH44w22 | AB + ACD + BCD |
| TH54w22 | ABC + ABD |
| TH34w32 | A + BC + BD |
| TH54w32 | AB + ACD |
| TH44w322 | AB + AC + AD + BC |
| TH54w322 | AB + AC + BCD |
| THxor0 | AB + CD |
| THand0 | AB + BC + AD |
| TH24comp | AC + BC + AD + BD |

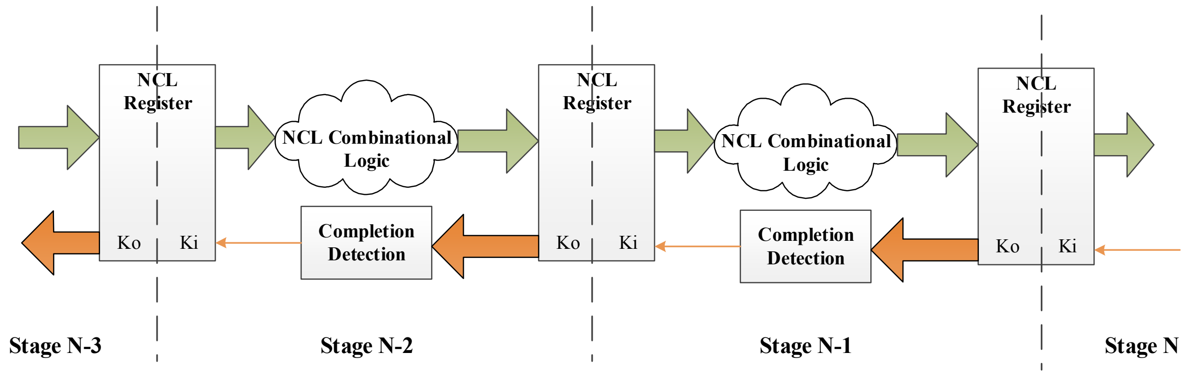

Figure 2: NCL Datapath

NCL's delay insensitivity springs from its handshaking protocol. The datapath of an NCL circuit is inherently pipelined. At the boundary of each pipeline stage is a register and a completion detection circuit that determines when DATA or NULL should be allowed through to the next stage. NCL circuits process data in waves, known as DATA wavefronts, where two adjacent DATA wavefronts are separated by a NULL wavefront. The NULL wavefront acts as a boundary to prevent DATA wavefronts from colliding and overwriting each other. An NCL register stores the previous wavefront until another complete wavefront arrives and is detected by the completion detection circuitry. The completion detection circuitry is also responsible for handshaking with the neighboring stages. For example, once the completion detection circuit determines that 1) the complete DATA wavefront has reached the register and 2) that the next stage is requesting DATA (RFD), it will signal the register to allow the DATA wavefront to pass through the register. The register latches the DATA value so that the next stage's combinational logic can process it. Once the aforementioned conditions are satisfied, the completion detection circuit issues a request for NULL (RFN) for its stage.

## 2.3 MTCMOS Power-Gating

In order to reduce the leakage power consumed in a CMOS circuit, transistors with various threshold voltages (Vt) can be incorporated to maintain performance while reducing leakage power consumption. High-Vt transistors, those that are less leaky but slower, are used to gate current to inactive portions of the circuit as shown in Figure 3. When the circuit is active, these transistors are enabled; but when the circuit becomes inactive as determined by sleep control logic, the transistors are disabled. The high-Vt transistors cut off the leakage path from power to ground. Low-Vt transistors are fast but leaky. They are used along the critical path due to their superior switching speed. Incorporating transistors with multiple Vt's into a circuit in this manner is known as Multi-Threshold CMOS (MTCMOS) power gating.

MTCMOS power gating, while having the ability to significantly reduce leakage, suffers from three major drawbacks [12]:

1. Sizing the power-gating transistors is a difficult tradeoff. If the transistors are too small, then the circuit's performance will be reduced due to a lack of current flow to the active logic. If the transistors are too large, then valuable area is wasted.
2. Data stored in flip-flops is lost when the circuit is gated. Special memory cells are required to retain data during sleep mode.
3. Generating sleep control signals requires extra logic, which adds area and power overhead. In addition, careful timing analysis must be performed to ensure the MTCMOS circuit blocks are slept/woken at the correct times.

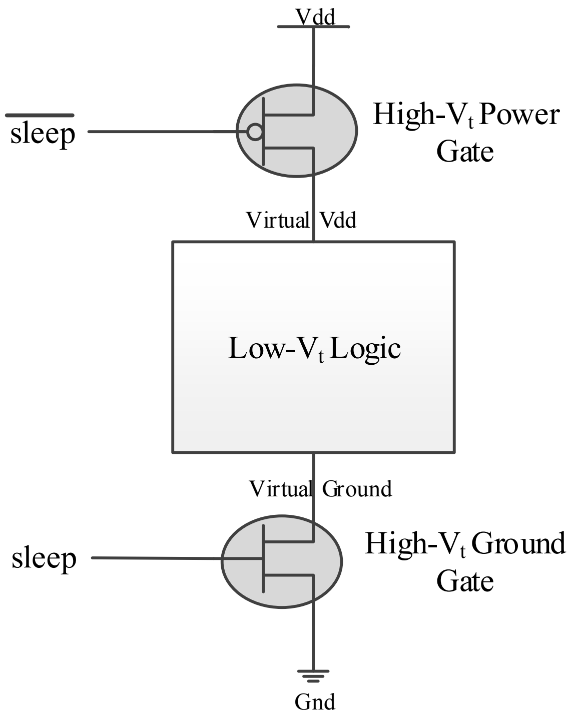

Figure 3: MTCMOS Power Gating

## 2.4 Multi-Threshold NULL Convention Logic (MTNCL)

### 2.4.1 Overview

One of the most promising low power QDI architectures, MTNCL, features a combination of NCL and MTCMOS power gating [13]. Like NCL it is dual-rail, asynchronous, QDI, and operates using threshold gates. MTNCL incorporates MTCMOS power gating into each of the threshold gates. By doing so, it is actually able to shrink the size of the gates. Only two transistors connected to *sleep* are needed to replace the *Reset* and *Hold1* blocks of the NCL gate. Moreover, hysteresis is only required in a small subset of MTNCL logic gates, thereby reducing area and simplifying gate design compared to NCL. Figure 4 gives a diagram of the MTNCL threshold gate structure with high-Vt transistors circled.

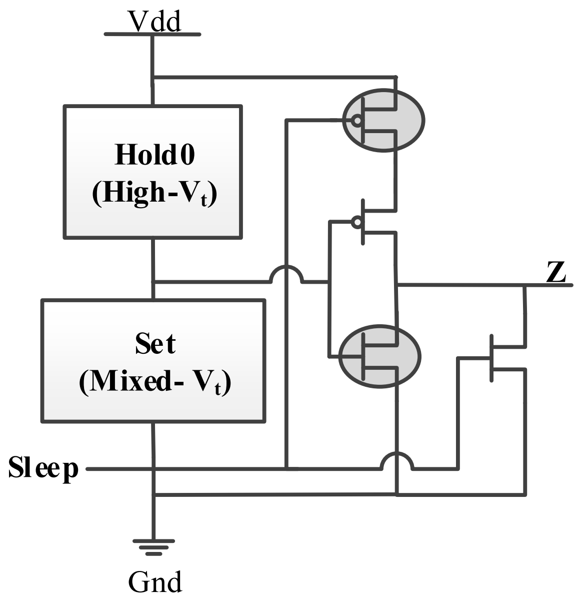

Figure 4: MTNCL Gate Structure

The MTNCL operating protocol is similar to NCL, but with a few modifications. The *sleep* signal connects to each MTNCL gate, and when asserted causes the output to fall to 0, while High-Vt transistors cut off the leakage path to ground. This is used to facilitate DATA/NULL wavefronts. When a NULL wavefront is needed from a stage, the *sleep* signal can be asserted thereby driving all the gates to 0 and presenting a valid NULL pattern to the register. This *sleep*-to-NULL behavior compromises the QDI of the architecture. A partial NULL waveform can be passed between stages, such that when the next DATA pattern arrives it gets mixed with DATA bits left over from the previous stage thereby causing invalid DATA to propagate through the circuit [14]. Fortunately, a technique called Fixed Early Completion Input Incomplete (FECII) can be used to ensure that all bits at the stage output are NULL before the next DATA wave is allowed to pass. FECII eliminates the possibility of passing a partial NULL and ensures QDI for MTNCL.

The MTNCL pipeline is shown in Figure 5. In addition to the combinational logic, both the registers and completion components can be slept when not in use [15]. This leaves very few non-power-gated components, and therefore has low leakage power.

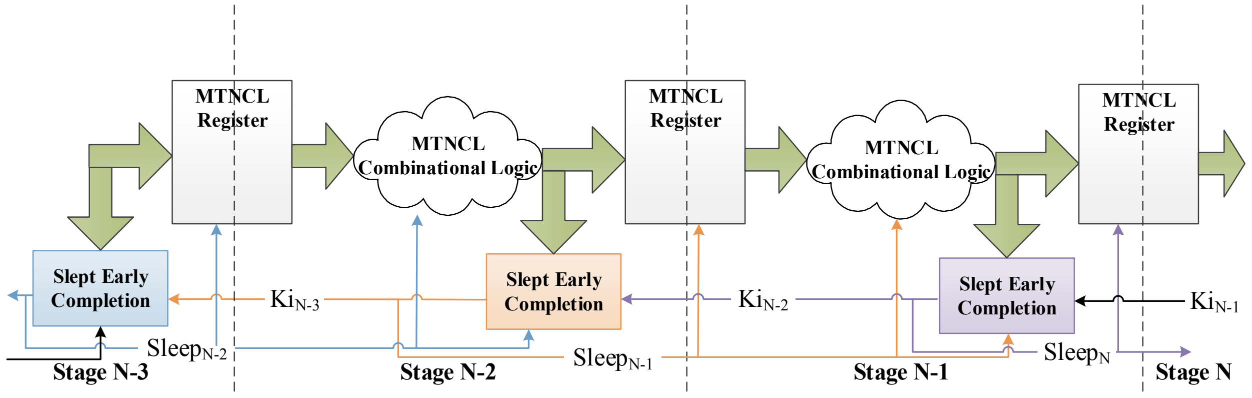

Figure 5: MTNCL Datapath

### 2.4.2 Previous Work

Although MTNCL is a recent invention, there have been a handful of circuits implemented using the technology. In [16] the MTNCL architecture is enhanced through several variations on the gate design, sleeping, and completion detection. A 4×4 array multiplier is implemented in several MTNCL incarnations. The various multipliers are compared amongst themselves and with an NCL implementation of the multiplier.

[15], an expansion of the work in [16], develops several enhancements to the MTNCL architecture. They implement a floating-point coprocessor in NCL, MTNCL, and synchronous MTCMOS architectures. Several variations of the MTNCL coprocessor implementation are tested. The FECII MTNCL architecture with completion and registration slept, called “SMTNCLwith SECRII w/o nsleep” in the paper, was found to be superior. It was compared to NCL, other MTNCL variations, and a synchronous implementation in terms of area, active

energy, and leakage power. In subsequent work, this MTNCL variation is simply referred to as MTNCL since it is the most efficient variation.

In [17], a 128-bit Advanced Encryption Standard (AES) core is developed using the MTNCL style developed in [15]. The AES design is also implemented using a synchronous methodology and compared to the MTNCL AES in terms of area, speed, robustness, and energy efficiency. The MTNCL design is 60% more energy efficient during operation and uses 6× less leakage current compared to the synchronous AES. Furthermore, the MTNCL AES is able to run encryption from a nominal supply of 1.2V down to 0.3V, while the synchronous AES can only operate down to 0.5V. The MTNCL design's throughput scales naturally as supply voltage decreases, but the synchronous AES requires the clock frequency to be tuned for each supply voltage operating point. In terms of area, the MTNCL AES has smaller total transistor width than the synchronous MTNCL.

In [18] MTNCL's resilience to changes in supply voltage is exploited to implement a Dynamic Voltage Scaling (DVS) system. The system uses multiple MTNCL array multiplier cores coupled with parallelism components and a voltage regulator. It is able to dynamically scale supply voltage of parallel cores based on the input data rate. Since MTNCL is asynchronous it runs at its natural frequency given the supply voltage. Thus, the timing analysis of such a system is substantially simplified compared to a synchronous design.

[19] presents an MTNCL Finite Impulse Response (FIR) Filter. The relationship between pipeline granularity, latency, and power is examined across four different pipeline granularities.

While the work above implies MTNCL has potential across a wide variety of applications, it must be demonstrated that MTNCL can offer benefits over state-of-the-art synchronous designs in order for it to be widely adopted in industry. Only two of the above designs, the AES Core and Floating Point Coprocessor, are compared to a synchronous implementation. Both of these are useful components, but neither represents a standalone design. Also, they are both datapath-dominated with small amounts of control logic. Larger more complicated systems need to be designed in order to prove MTNCL as a viable alternative to synchronous methodologies. It needs to be demonstrated that MTNCL can be beneficial in a larger design with less datapath and more control logic focus.

## 2.5 MSP430

The MSP430 is a RISC microcontroller developed by Texas Instruments in the 1990's. It has gone through several iterations and established itself as a leading microcontroller for power constrained embedded designs. It comes in several varieties with different peripherals and amounts of RAM. At its core is a RISC pipeline with 27 instructions and 24 emulated instructions. It incorporates multiple clocks in order to facilitate various low-power modes. Additionally, it has an extensive vectored interrupt capability and supports up to 14 peripherals. In the Internet-of-Things era, the MSP430 is a very important design. It is well suited for low-power data processing and decision-making tasks ubiquitous across connected devices.

An open source version of the MSP430 (openMSP430) was developed in [20]. It implements the entire MSP430 instruction set and is instruction for instruction compatible with TI's microcontroller. In [3] the results from an asynchronous implementation of the MSP430 using the balsa asynchronous design environment are presented. They explore three different implementations of the MSP430: bundled data, dual-rail, and 1-of-4 encoding. However, their design stops at the gate level, and no physical design is explored. The design uses Boolean gates, which add a large power and area overhead to QDI designs.

The MSP430 is ideal for implementation in MTNCL. With its goal of being a low-power, low-cost microcontroller the MSP430 gives the opportunity to demonstrate MTNCL's energy-efficiency against an industry standard for low-power computing. The MSP430 is more complex than any MTNCL design to date, and encompasses a complete system capable of functioning as a standalone chip. The MSP430, being a microcontroller, is a control-dominated rather than datapath-dominated architecture; therefore, it is a good test of MTNCL's merits for other control-dominated designs. Additionally, the synchronous openMSP430 can be implemented in the same technology and serve as a benchmark for comparing the MTNCL microcontroller to a well-proven synchronous design.

# 3 DESIGN FLOW

The MTNCL design flow is immature compared to synchronous methods used in industry. Logic design and buffering are particularly challenging, as no standardized methods exist for the MTNCL design. For each new technology MTNCL standard cell libraries must be gates as standard cells designed and characterized since standard Process Design Kits (PDK) do not include MTNCL.

## 3.1 Logic Design

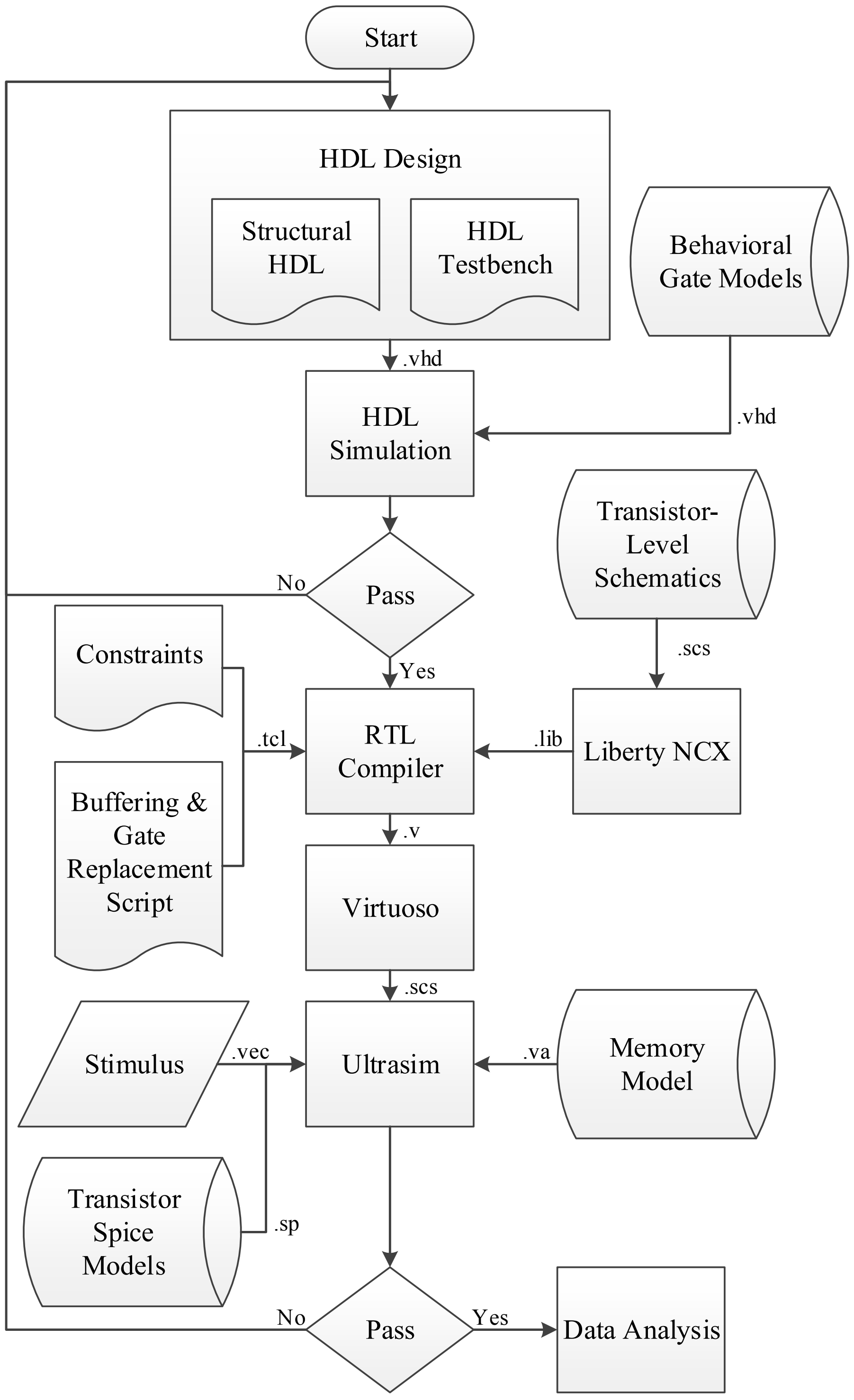

Figure 6: MTNCL Design Flow

At the time of this design, synthesis tools for MTNCL are immature. They accept synchronous RTL HDL as input, perform synthesis to generate a single-rail synchronous netlist, then convert the netlist to dual-rail, map the combinational logic to MTNCL gates, and add MTNCL registration and completion components [2]. This produces an unoptimized MTNCL design. Since the starting point is a synchronous design, the high-level blocks are designed with synchronous instead of MTNCL architecture in mind. To build an optimized MTNCL design, the high-level architecture must be built with MTNCL methodology in mind. In the case of the MTNCL MSP430, the automated synthesis technique is suboptimal in the high-level control, Register File, and Timer.

This MTNCL MSP430 is designed at the structural level using VHDL. Behavioral models for the MTNCL gates act as primitives and are used to construct larger blocks. Additionally, several synchronous logic gates are included in the library. These are used in the Register File and various control components. Mentor Modelsim is used for functional simulation.

## 3.2 Buffering

Although MTNCL requires minimal timing analysis, it is beneficial to set a max transition or max capacitance for each gate. This avoids long delays due to wire or fanout that can violate the relative timing assumptions of MTNCL [2]. It also helps to reduce short circuit power consumption and pipeline bottlenecks. The max capacitance/transition is determined when the MTNCL threshold gate library is characterized. It is based on the transistor size and drive strength of the gate. Commercial buffering tools query the Liberty timing file for the max capacitance/transition values. Then, they size, clone, and insert buffers such that all gates meet their capacitance and transition requirements in the Liberty file. The max capacitance/transition rules along with max fanout fall under the category of design rules (DRV). DRVs are checked in synchronous designs both during synthesis and physical implementation, and this flow is easily adapted to MTNCL designs once the threshold gates have been characterized. In synchronous designs the buffering tool must ensure that various constraints on path delays are met. For example, the longest path delay through a stage must be less than the clock period minus the setup time of the capture flop. These timing constraints are critical to the synchronous circuit's operation and, therefore, supersede DRV requirements. However, MTNCL, being QDI, has no such timing requirement, and the cost function of the buffering tool is changed to only respect DRVs.

Cadence RTL Compiler (RC) is used for buffering the MTNCL MSP430. RC is given the structural VHDL netlist along with Liberty files for each standard cell. Then, RC's cost function is changed to only respect DRVs. The input driving cells and output load cells are defined using SDC constraints. RC produces a buffered and sized netlist that is appropriate for schematic capture and transistor-level simulation. The same buffering and sizing flow can be used in Cadence Encounter during physical implementation where real wireloads are taken into account.

# 4 ARCHITECTURE

Like the openMSP430, the MTNCL MSP430 is instruction-for-instruction compatible with TI's MSP430 microcontroller. The functional division of the MTNCL MSP430 is similar to the openMSP430 as shown in Figure 7 and Figure 8. The following chapter details the design and design considerations of the MTNCL MSP430 architecture, while contrasting it with that of the openMSP430.

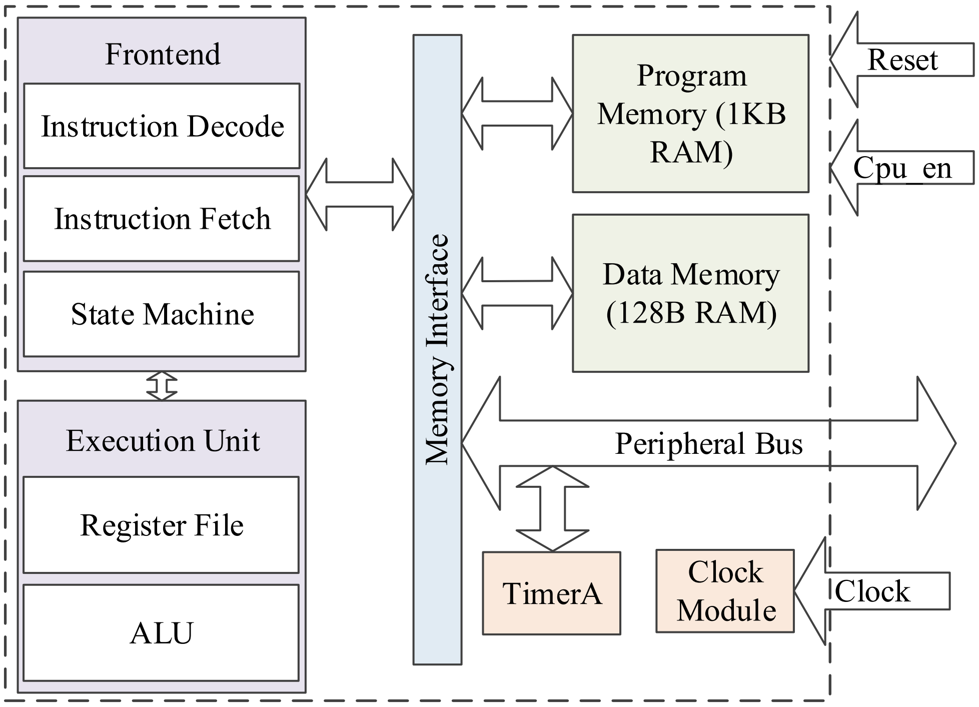

Figure 7: openMSP430 Functional Block Diagram

## 4.1 Datapath

Determining the number of pipeline stages in a synchronous design is a balance between energy efficiency, throughput, and data dependencies. The data dependencies are determined by the instruction set of the processor, which in this case is fixed for both synchronous and MTNCL designs. The MSP430 with its goal of being a low-power and low cost microcontroller uses a RISC instruction set with simple functional blocks. Similar to a basic MIPS architecture, it does not use high-speed, high-area overhead techniques such as out-of-order execution, hierarchical caching, or multiple execution units. In fact, the openMSP430 consists of only a single pipeline stage. This makes the microcontroller extremely compact and energy efficient at the expense of the throughput benefits of extra pipeline stages.

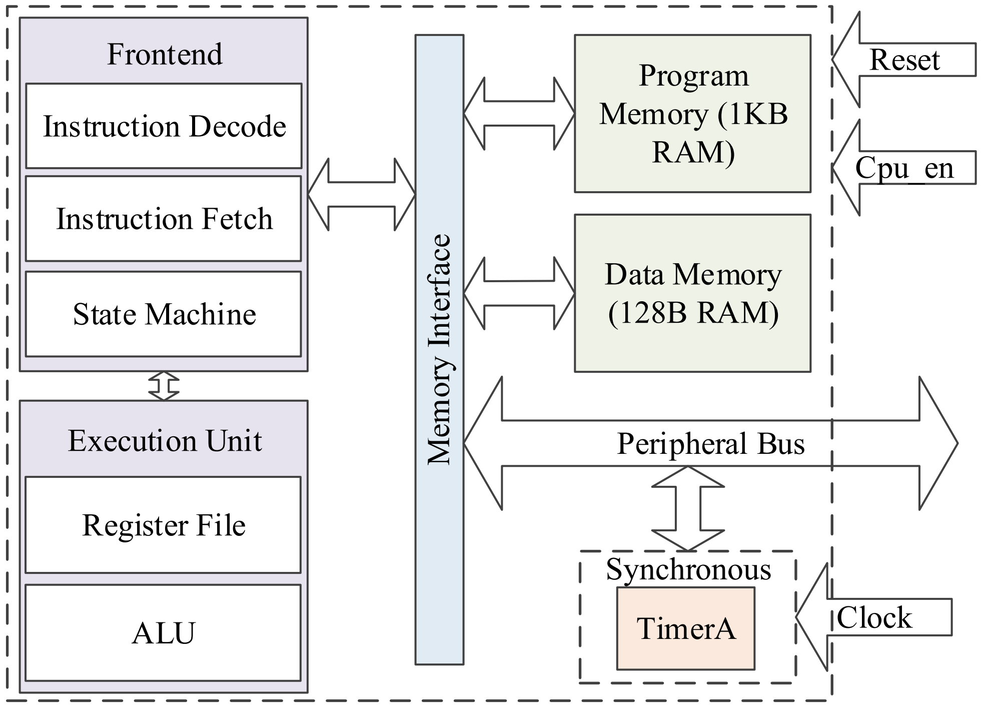

Figure 8: MTNCL MSP430 Functional Block Diagram

Pipelining the MTNCL MSP430 datapath requires additional considerations compared to the openMSP430. The lower bound of the number of pipeline stages is defined by the requirements of the dual-rail DATA/NULL protocol. In order for proper feedback of DATA/NULL wavefronts, a dual-rail encoded design must have at least three pipeline stages [21]. The upper bound of the number of pipeline stages is a similar tradeoff as that of synchronous circuit design: throughput vs. area and energy. Since a primary constraint of the MSP430 is energy, the lower bound of three pipeline stages is used.

After determining the number of stages in the pipeline, the design is partitioned into stages. This is a key step to guaranteeing design efficiency. In order to maximize throughput in the MTNCL MSP430, combinational logic is divided among the three pipeline stages as evenly as possible given the data dependencies in the datapath. Unlike the regular structure of an array multiplier, AES, or other regular circuits, the datapath of the MSP430 microcontroller has a large variance in the number of gates along each feedback loop. Some paths have a large amount combinational logic such as the paths through the ALU. Others have very little such as the register file or operand registers. While some paths, such as instruction decode and control, are limited by data dependencies in the datapath. Their outputs are required at the first register stage.

## 4.2 Selective Sleeping

A technique that can increase flexibility in register location, while at the same time reducing active energy consumption is Selective Sleeping (SS). Using SS a designer divides the combinational logic into smaller functional blocks where each block can operate independently of the others. The registers at the output of these functional blocks are split such that each function becomes a smaller pipeline. Each individual pipeline can be slept separately from others. Therefore, when the circuit is in operation, the pipelines not needed for the current operation can remain slept. The structure of SS is shown in Figure 9.

Once the pipeline has been split, the handshaking signals must be combined such that the unused blocks remain slept. The *sleep* signals for the functional blocks and the input completion components can be determined using combinational logic in the first stage of the pipeline. This logic, however, must always be activated when using any of the functional blocks. The *sleep* generation block reads the input DATA wavefront, and generates the *SEL* vector to select the block or blocks to be activated for the next cycle. Then, *SEL* is inverted and OR'd with the incoming *sleep* signal to produce the *sleep* signal for the given functional block.

$$slp(i) = \overline{SEL(i)} + slp\_in \tag{1}$$

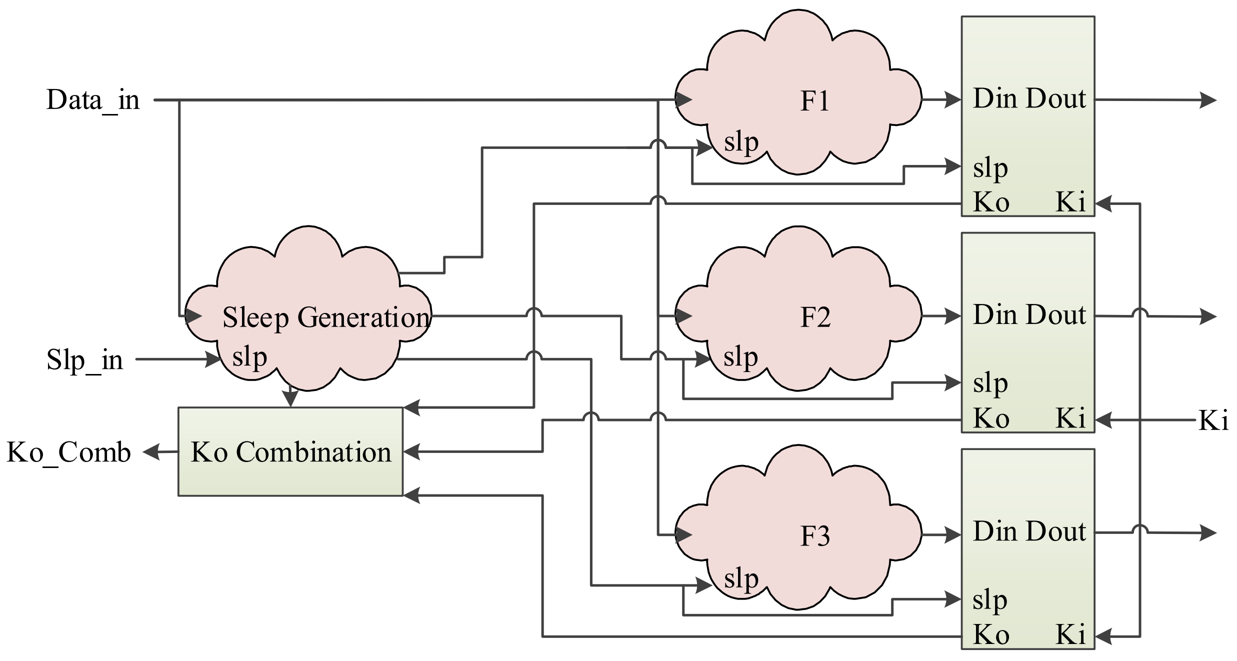

Figure 9: SS Structure

In addition to the sleep signal generation, SS implements special logic for handling the *Ko*'s from the individual pipelines. The *Ko* signals from each pipeline are combined in the *Ko* Combination block to produce a global *Ko* representing all the pipelines. This combinational process can be broken down into two pieces. The first piece generates the RFN signal when the selected pipelines are all RFN. The second piece generates the RFD signal when all pipelines are RFD. (2) is the equation for generating the combined *Ko* signal.

$$Ko\_Comb = \prod Ko(i) + \sum TH22(SEL(i), Ko(i)) \tag{2}$$

The overhead associated with SS is quite small, and a proper implementation can decrease the energy lost to *sleep* net switching and wavefront propagation in much the same way that clock gating can decrease the switching energy of the clock net in a synchronous design. *Ko* combination and *sleep* generation lie along the handshaking signal path, and thus add delay to the pipeline stage. In most designs the function selection logic to generate *SEL* is already present.

This limits the delay overhead of the s*leep* generation block to only a single TH12 gate. Furthermore, the *Ko* product terms in the first part of (2) would be required in a normal MTNCL pipeline. Therefore, the overhead associated with the *Ko* Combination block is an OR-tree for the summation term and a TH22 gate for each functional block. The *Ko* Combination block adds only a TH22 delay and a TH12 delay to the handshaking signal path. Thus, the total delay overhead for the stage from SS is (3) and the area overhead for a stage split into N functions is (4).

$$delay(TH22) + 2 \cdot delay(TH12) \tag{3}$$

$$N \cdot area(TH22) + area(OR\_Tree\_width\_N) + (N+1) \cdot area(TH12) \tag{4}$$

With a low area and delay overhead, SS can be used to reduce power consumption in a wide variety of MTNCL circuits. It eliminates the switching power due to the *sleep* signal transition as well the power due to the propagation of the DATA/NULL wavefronts for the unused function blocks. Thus, the power savings will be circuit specific and based on the activity factor of the function blocks. In the case of the MTNCL MSP430 design SS was used to reduce power consumption of the Control, ALU, and Register File.

## 4.3 ALU

The MSP430 ALU is simple in structure with the two main blocks being an adder and a logical operations block. The ALU supports four single operand operations and eight double operand operations shown in Table 3. A modified version of SS is used in the ALU with the functional division between the logical operations block and the adder block. The functional division is shown in Table 3. (L) indicates the function is contained in the logical operations block, while (A) indicates the function is contained in the adder block. Additionally, there is an always-active logic block to multiplex the ALU outputs and handle the single operand functions.

Table 3: ALU Functionality

| **Single Operand** | **Double Operand** |
|---|---|
| Rotate Right | Addition with/without Carry In (A) |
| Rotate Right Arithmetically | Two’s complement Subtraction with/without Carry In (A) |
| Swap Bytes | Bit Vector Comparison (L) |
| Sign Extension | Decimal Addition (A) |
| | Bit Testing (L) |
| | Bit Clearing (L) |
| | Bitwise XOR (L) |
| | Bitwise AND (L) |

The modified SS operates only on the combinational blocks of the ALU. It does not select registers. Therefore, the ALU must handle NULL outputs from the unselected functional block during DATA propagation. The key is to observe that the functional block outputs are mutually exclusive so that only the selected output is DATA. For example, when the adder is selected, the DATA wavefront will pass through the adder block to the Output MUX, while the logical operations outputs remain NULL. The only signals that are asserted will be from the selected components, the correct outputs. Thus, the Output MUX can be simplified to a simple OR tree for both RAIL1 and RAIL0 signals. This further reduces the area and power consumption of the ALU logic.

The ALU has a depth of three pipeline stages as shown in Figure 10. A previous version of the ALU placed the second register stage between the function blocks and the Output MUX; however, testing revealed this implementation to be faster but less energy efficient than the current pipeline. When the second register is placed before the Output MUX, it must receive the outputs from each function, increasing the width by approximately four times compared to the placement shown in Figure 10. Since energy is the primary constraint for the MSP430, a partitioning scheme was selected to minimize the area overhead and energy consumption of each register by keeping the registers as small as possible.

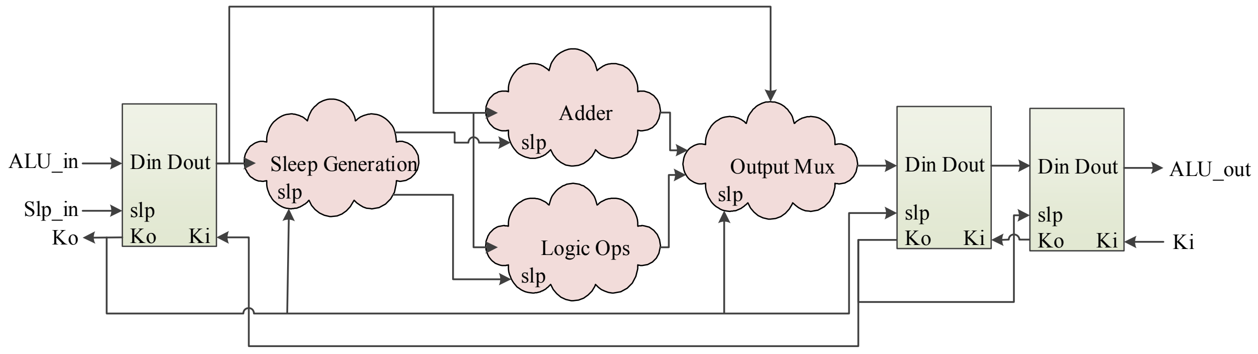

Figure 10: ALU Datapath

## 4.4 Register File

The MSP430 Register File should be distinguished from MTNCL registration. The Register File is a collection of memory elements that retain their values during DATA/NULL cycles. MTNCL registers, on the other hand, act as separators between stages in the MTNCL pipeline. They do not store information between DATA/NULL cycles, but are refreshed each DATA/NULL cycle.

The Register File is a key component of a microcontroller. It is accessed at least once, usually more, for every instruction the microcontroller executes. Optimization of this component in the MTNCL MSP430 is an important part of reaching the overall goal of low power.

In traditional MTNCL and NCL designs, data storage between cycles requires a large area and power overhead. A three-ring register structure with a load/store multiplexer is used to store a data pattern between datapath cycles. The area for this component is given by (8), where R is the registration area, C is the completion component area, and M is the multiplexer area.

$$R = 2 \cdot N \cdot area(TH12(dn)m) \tag{5}$$

$$M = N \cdot area(THAND0m) \tag{6}$$

$$C = [area(TH22n) + 2 \cdot area(INV) + N/2 \cdot area(TH24COMPh1m) + ((N-1)/3) \cdot area(TH44h1m)] \tag{7}$$

$$Traditional\_\mathrm{Re}gister\_File\_Area = 3 \cdot R + M + 3 \cdot C \tag{8}$$

When compared to a synchronous register consisting of a multiplexor and D-flip flops, the MTNCL memory element is quite large. It is also particularly power hungry since the data pattern must continuously loop through the registers every cycle of the main pipeline. To further add to the power burden, the structure cannot be used with SS since a slept pipeline will go to NULL and lose the stored data pattern.

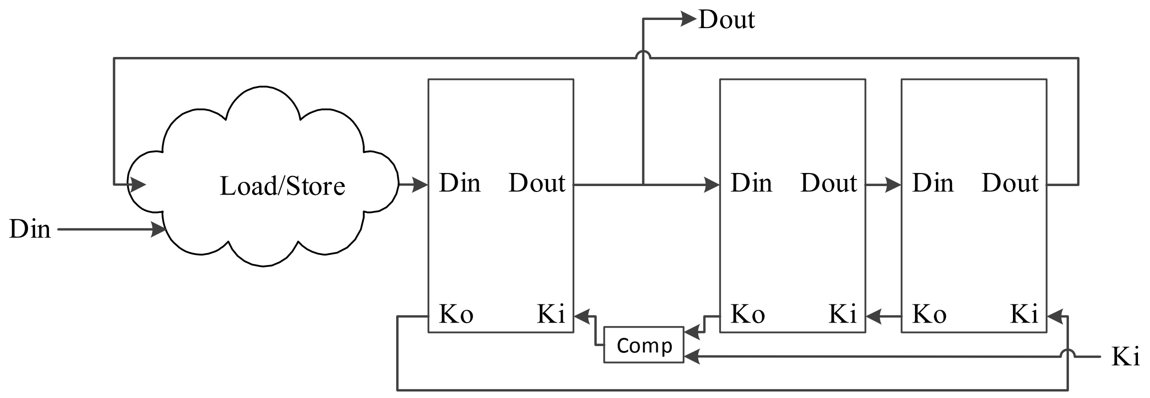

Figure 11: Traditional 3-Ring Register

Optimization of the Register File can be accomplished by incorporating D-flip flops, which are not typically used in asynchronous design due to their setup and hold time requirements. Special logic is required to ensure that the setup and hold times for the D-flip flops are always met despite the asynchronous behavior of the circuit. This is accomplished by making a slight modification to the D-flip flop structure and incorporating completion detection on the inputs and outputs.

The bit cell structure to guarantee setup time is shown Figure 12. A third output is added to the D-flip flop, which connects to the internal node between the two latches. D-flip flops are commonly constructed of two cascaded latches triggered on opposite clock edges. The output of the first latch connects to the input of the second latch. The *Q_int* signal is the output of the first D-flip flop. Since the setup time is defined as the time it takes for the internal node, *Q_int*, to charge/discharge, the clock edge can be safely triggered on the transition of *Q_int*.

The Register File only needs to latch data when the incoming data is different than the data being stored. Thus, the clock is triggered when *Q* and *Q_int* are different, meaning that the incoming data pattern has charged the internal node, *Q_int*, to a value different than the stored value, *Q*. The sleepable XOR gate with input *Q* and *Q_int* triggers the clock edge to occur when the two signals are different. Then, if *load* is asserted, the D-flip flop receives a rising clock edge causing the value on *Q_int* to be latched.

Hold time is guaranteed by the XNOR gate and MTNCL completion. The XNOR gate generates a *Ko* signal for each bit when the output, *Q*, and input, *D*, match. This indicates that the proper value has been latched for the given bit. The XNOR gates replace the TH24comp gates in a standard MTNCL completion component, and ANDing the individual *Ko*'s together gives a completion signal for the entire pipeline stage. Following the MTNCL handshaking protocol the pipeline stage's completion is an input to the previous stage, which causes the previous stage to hold its DATA pattern until all the bit cells of the register have properly stored the DATA pattern.

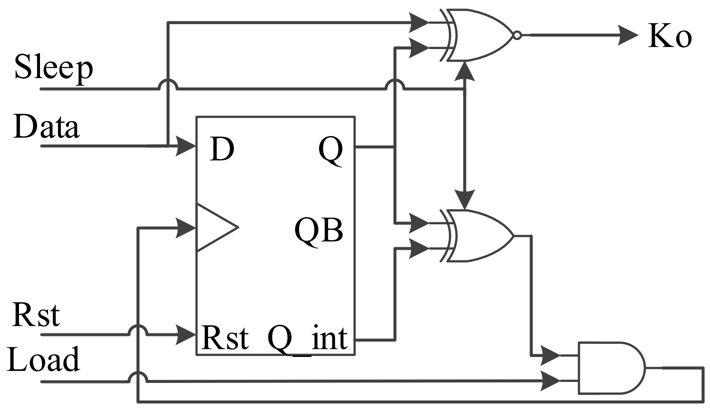

Figure 12: Modified MTNCL Register File Bit Cell

It is important to note that the XOR gate controlling the clock is sensitive to glitches on the *Q_i*nt signal. If *Q_int* has a glitch and Load is asserted, an incorrect value could be latched into the bit cell. To prevent this, the XOR gate has a sleep input controlled by a completion component attached to the inputs of the Register File pipeline stage. The input completion component detects when the DATA pattern arrives and only allows the XOR gate to transition switch once the all the inputs have settled.

The bit cell structure is very efficient in terms of power and area. Switching in the modified bit cell only occurs when a change in the stored value is requested. On the other hand, the traditional MTNCL three-ring register transitions between DATA and NULL continuously as the microcontroller's pipeline cycles. Furthermore, the total area for the modified MTNCL Register File structure is less than the traditional three-ring register structure. Table 4Table 1 gives an area comparison in terms of transistor counts. The modified Register File achieves a 15-17% area savings depending on the number of bits being stored.

Table 4: Transistor Count Comparison between Modified and 3-Ring Register Files

| **Width (bits)** | **Traditional** | **Modified** | **Area Savings** |
|---|---|---|---|
| 4 | 539 | 446 | 17% |
| 8 | 1033 | 864 | 16% |
| 16 | 2021 | 1700 | 16% |
| 32 | 3997 | 3372 | 16% |
| 64 | 7949 | 6716 | 16% |
| 128 | 15853 | 13404 | 15% |
| 256 | 31661 | 26780 | 15% |
| 512 | 63277 | 53532 | 15% |
| 1024 | 126509 | 107036 | 15% |

By combining the modified MTNCL bit cells into the MTNCL MSP430 Register File, a new opportunity for power savings appears. Sleeping the three-ring bit cell causes it to lose all stored data; however, the modified bit cell retains its data while slept. This allows SS to be used in the MSP430 Register File, drastically reducing its power consumption. The structure of the Register File is shown in Figure 13.

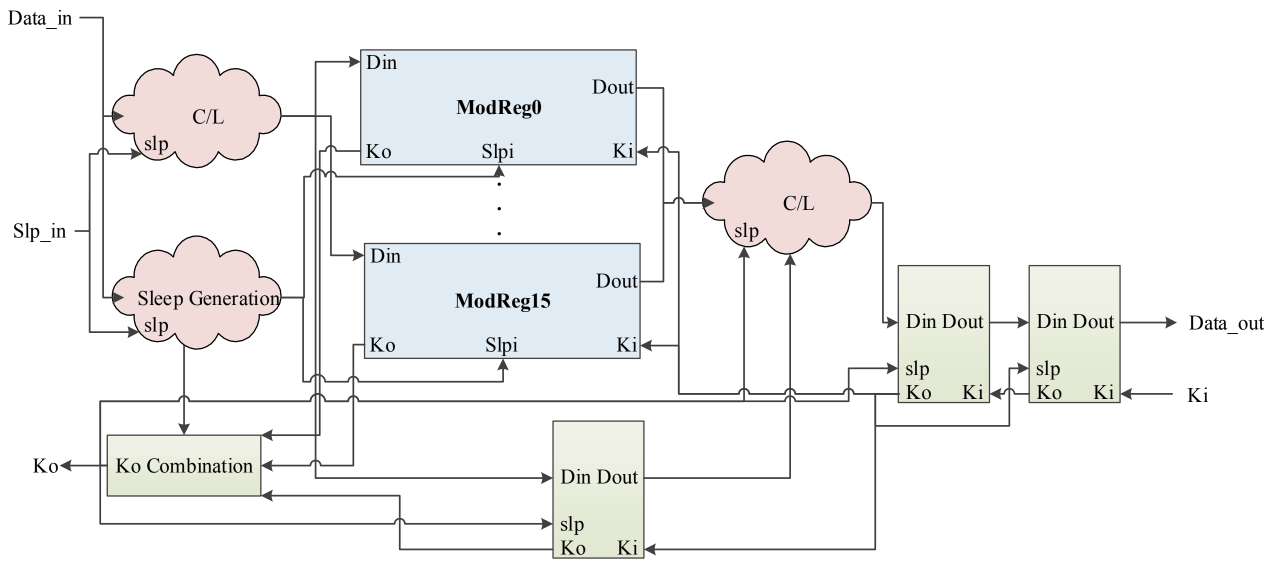

Figure 13: MTNCL MSP430 Register File

The MSP430 Register File consists of sixteen 16-bit registers, an adder, a constant generator, and an output multiplexer. It also contains the SS components shown in the diagram. Each 16-bit register is separately sleepable. In normal operation the Register File will only activate one or two 16-bit registers allowing the majority of the registers to remain slept conserving power. Like the ALU, the design requires an always-active path so that the appropriate *select* signal can be sent to the output multiplexer.

## 4.5 Memory Interface and Interrupts

The MSP430 includes both Program and Data Memory, as well as address space dedicated to peripherals and interrupt vectors. The Memory Interface handles the connection to all of these. It selects the appropriate component based on the address it receives from the microcontroller. The Memory Interface handles connections to both synchronous and asynchronous peripherals.

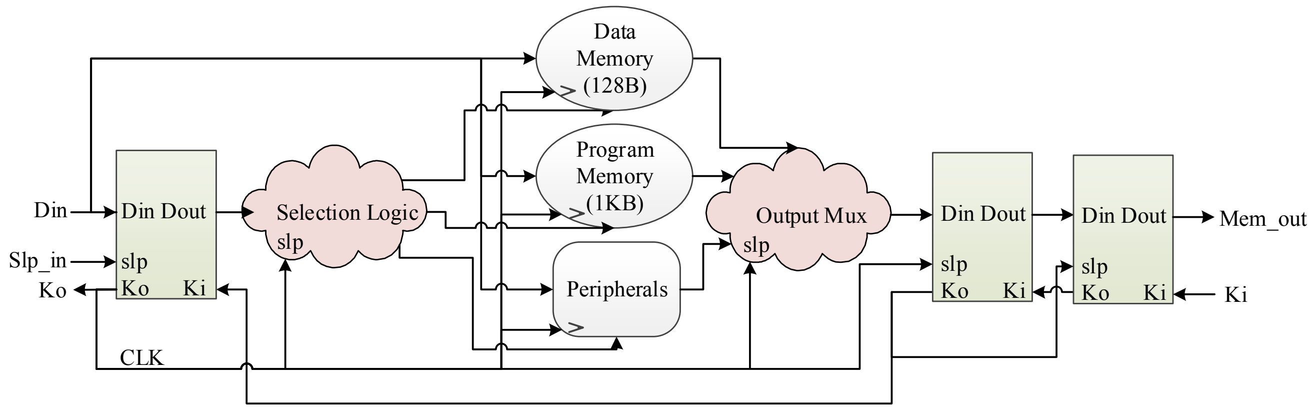

Figure 14: MTNCL MSP430 Memory Interface

Asynchronous peripherals are written and read using MTNCL handshaking. However, for synchronous connections like the memories and the synchronous peripherals, a clock is required to determine when the input data should be latched. The clock can be generated by the *Ko* signal from the completion component on the input to the memory interface. The completion component on the input register detects when the data has arrived by monitoring the dual-rail signals, *Din*. Once it detects the DATA wavefront has arrived and its *Ki* is logic one, it will transition to RFN, logic zero, creating a clock edge. Like all synchronous designs, the timing must be analyzed to ensure that the clock rate of the *Ko* signal is not too fast for the memory or peripheral latches.

$$delay(CC) \geq T_{setup} + delay(wire) \tag{9}$$

In most cases, the propagation delay of the completion component is much longer than the setup time of the input latches, and the relative delay of the completion component guarantees the timing will be met. However, this assumes that there is no single rail logic between *Din* and the latch in the synchronous component. The delay from the single rail logic could easily be longer than the completion component delay. This case will be discussed along with the Timer in the next section.

For a write operation there is no output from the peripherals or memories. The output combinational logic detects if a write has been performed, and generates a DATA wavefront to trigger the output completion component. In the case of a read, the memory or peripheral generates a DATA/NULL pattern on its outputs to signal the output completion component. The output completion component ensures that the microcontroller pipeline will wait until the correct data has arrived from the peripheral or memory before continuing operation. Peripherals have an additional trigger for communication with the microcontroller. Each peripheral is assigned an interrupt line and an interrupt vector. When a peripheral's interrupt line is asserted, the microcontroller will jump to the instruction located at the interrupt vector.

## 4.6 Timer

The TimerA peripheral of the openMSP430 is included in the MTNCL MSP430. Despite a few modifications to ensure delay requirements are met, it remains a synchronous component and has the same functionality as in the openMSP430. Since the MTNCL MSP430 is asynchronous, TimerA requires an external clock to keep accurate time. The timer's accumulator register is clocked by the external clock. Additionally, the timer has another clock domain for its configuration registers. The *Ko* from the Memory Interface controls the clocking of the configuration registers.

Some of TimerA's configuration registers have single rail logic at their inputs as shown in Figure 15. The delay from the single rail logic causes a setup time violation for the clock edge generated by the *Ko*. As warned in the previous section, the delay from the single rail logic is greater than the delay from the completion component causing a timing violation. The edge generated by Memory Interface *Ko* comes too late for the configuration registers to latch the correct data.

To meet the setup time, latches are added to the inputs of TimerA, and the *Ko* clock is inverted. The latches hold DATA at TimerA's inputs through the NULL cycle of the Memory Interface. The inversion of the *Ko* clock then causes the configuration registers to be clocked at the end of the Memory Interface's NULL cycle. The setup time equation becomes:

$$2 \cdot delay(CC) + delay(NULL) \geq T_{setup} + delay(wire) + delay(SR\_\log ic) \tag{10}$$

Reading TimerA's register values can be accomplished through the Memory Interface. The microcontroller sends a read request to an address in the timer's memory space, and the Memory Interface will enable the timer and read the appropriate output data. The output multiplexer for the timer's outputs is required to be dual-rail, in order to ensure the proper data pattern has propagated to the output. Additionally, the timer contains the ability to interrupt the MTNCL MSP430 once the accumulator reaches a certain value.

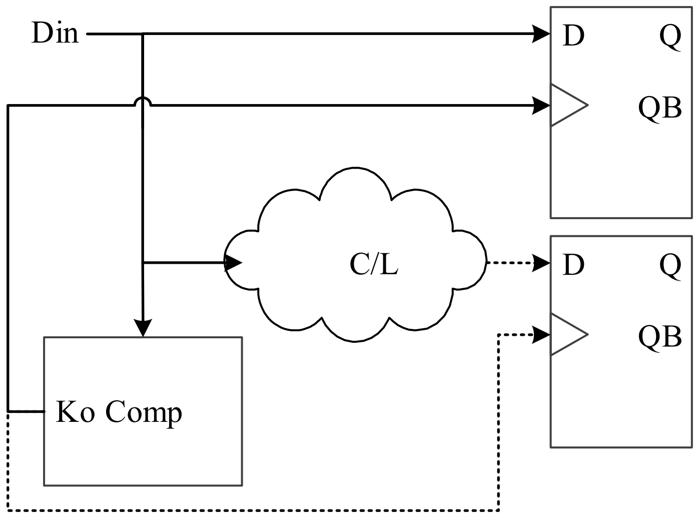

Figure 15: MTNCL MSP430 Timer Configuration Register

## 4.7 Control Unit

At the heart of the microcontroller is the state machine of the Control Unit. The state machine coordinates data flow between all of the microcontrollers components. The Control Unit incorporates SS. In each cycle of the pipeline, the Control Unit selects which components are needed. Those that are not required remain slept.

The top-level pipeline of the MTNCL MSP430 is shown in Figure 16. It consists of several parallel pipelines using the SS technique to keep the unused paths slept. The Control Unit has its own pipeline, which is always active since it must retain the state of the microcontroller.

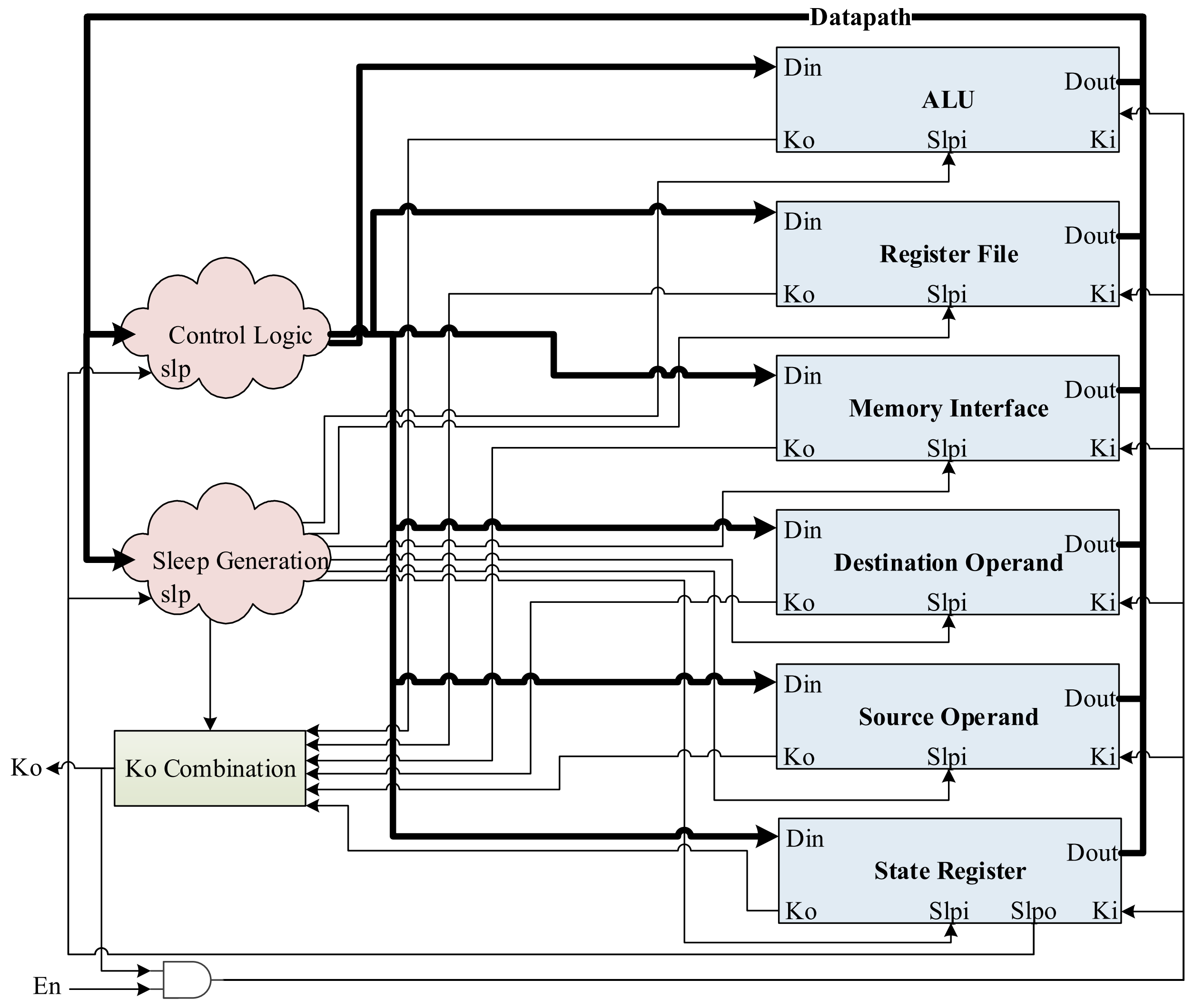

Figure 16: MTNCL MSP430 Top-Level Pipeline

## 5 TESTING

Designing fair comparison data between asynchronous and synchronous microcontrollers is a non-trivial task. Several factors must be taken into account: architecture of the benchmark circuit, test vectors, and the energy measurement time window. If care is not taken, any of these factors has the potential to skew the energy comparison.

## 5.1 Benchmark

The synchronous benchmark circuit is synthesized from the openMSP430 RTL source code [20]. This design is instruction-for-instruction compatible with TI's production MSP430, but allows for customization of included peripherals, operating speed, and process technology. Without these customizations a fair comparison with the MTNCL MSP430 is infeasible. However, by customizing the parameters to match those of the MTNCL MSP430, a fair comparison can be designed.

Table 5: Timer Constraint and Gate Count Comparison

| | **MTNCL MSP430 Timer** | **openMSP430 Timer** |
|---|---|---|
| **Clock Constraint** | 154 MHz | 154 MHz |
| **Gate Count** | 1077 gates | 906 gates |

A peripheral timer is included in the synchronous benchmark to match the timer in the MTNCL MSP430. The timer in the openMSP430 RTL source code is used in the synchronous benchmark, and the same timer design is also used in the MTNCL MSP430 with only minor modifications to handle the asynchronous peripheral interface. The constraints and gate counts between the two timers are given in Table 5. In general, a peripheral connected to the MTNCL MSP430 requires minimal modification compared to peripherals in the openMSP430. Thus, the focus of the comparison is on the core. The timer is added as a demonstration of functionality in the MTNCL MSP430, and as a benchmark for comparison in the synchronous openMSP430.

In addition to the synthesis constraints of the timer, the constraints of the openMSP430 core are based on the nominal operating speed of the MTNCL MSP430. The MTNCL MSP430 is simulated at a nominal temperature of 27°C and Vdd of 1.2V. An average frequency of the pipeline is taken over a wide range of instructions. Then, the openMSP430 clock is constrained to the average frequency of the MTNCL MSP430 pipeline. This gives the benchmark design a similar throughput to the MTNCL MSP430. A fair comparison can be made since the cores are capable of executing instructions in a similar amount of time. This technique for constraining the synchronous benchmark circuit according the average throughput of the asynchronous pipeline was developed in [17].

## 5.2 Simulation & Results

Energy measurement is performed using the UltraSim circuit simulator inside Cadence Analog Design Environment. UltraSim is a versatile simulation environment and has accuracy/speed settings from fast digital simulation down to SPICE circuit solver. The coarse-grained simulation settings listed in the order of least to most accurate are: Digital Extended, Digital Fast, Digital Accurate, Mixed Signal, Memory, Analog, and SPICE. As the accuracy increases so does the run time. Fine-grained accuracy settings can be made by adjusting the speed option to control the relative tolerance for voltage and current calculations or by using mixing accuracy modes across instances of the circuit. For timing and energy measurements of the MTNCL MSP430 and synchronous benchmark, Digital Accurate mode with a speed setting of 5 was selected. This mode employs a nonlinear current and charge model for MOSFETs and their diffusion junctions. With these settings the simulation is bounded to within 5% of SPICE accuracy, while yielding reasonable runtimes for the long simulations required to execute a sufficient amount of instructions on the MSP430.

### 5.2.1 Active Energy

The energy consumption of a circuit is largely dependent on how it is used. The application determines which instructions are executed, and the instructions executed determine which blocks of the circuit are activated. For those blocks that get activated, their frequency of activation, and the data patterns they process are the key factors in determining the total energy consumption. When performing a comparison between asynchronous and synchronous circuits, it is important to average the results across blocks and various input patterns so that an advantage of one circuit for a particular pattern on a particular block does not weigh too heavily in the overall energy results. Without a targeted application, the best method of comparison is an average across all possible patterns. However, that is not feasible for circuits with many inputs and a random subset of input patterns should be used.

In the case of the MSP430 microcontroller, the blocks that get activated in a given cycle and the energy consumption are correlated to the type of instruction being executed. The instruction operand values play a smaller role in the energy consumption of the circuit. The MTNCL MSP430 is simulated with multiple input patterns across several instructions and different addressing modes. The energy results are tabulated in Table 6.

The energy data in Table 6 are based on a variety of register-to-register instructions, but using only a single set of operands for each instruction. Varying the operands can change which gates get activated, and in turn the energy consumption. Table 7 shows the energy consumption averaged across six different random operand combinations. There is a 3% difference in the average energy consumption from the baseline operands.

Table 6: MTNCL MSP430 Energy Consumption

| **Instruction** | **Format** | **Cycles** | **Total Time** | **Avg Power** | **ETop(J)** |
|---|---|---|---|---|---|
| Add R5, R6 | Double | 4 | 2.93E-08 | 4.48E-03 | 1.32E-10 |
| AddC R5,R6 | Double | 4 | 2.92E-08 | 4.28E-03 | 1.26E-10 |
| AND R5, R6 | Double | 4 | 2.93E-08 | 4.28E-03 | 1.26E-10 |
| BIC R5, R6 | Double | 4 | 2.89E-08 | 4.25E-03 | 1.23E-10 |
| BIS R5,R6 | Double | 4 | 2.89E-08 | 4.64E-03 | 1.35E-10 |
| BIT R5,R6 | Double | 4 | 2.81E-08 | 4.36E-03 | 1.23E-10 |
| CMP R5,R6 | Double | 4 | 2.82E-08 | 4.07E-03 | 1.15E-10 |
| DADD R5, R6 | Double | 4 | 2.89E-08 | 4.26E-03 | 1.24E-10 |
| MOV R5,R6 | Double | 3 | 2.22E-08 | 3.65E-03 | 8.13E-11 |
| SUB R5, R6 | Double | 4 | 2.89E-08 | 4.17E-03 | 1.20E-10 |
| SUBC R5, R6 | Double | 4 | 2.89E-08 | 4.56E-03 | 1.32E-10 |
| XOR R5, R6 | Double | 4 | 2.93E-08 | 4.30E-03 | 1.26E-10 |
| PUSH R5 | Single | 3 | 2.51E-08 | 4.90E-03 | 1.23E-10 |
| **Average** | | | **2.83E-08** | **3.34E-03** | **9.42E-11** |

Additionally, the addressing mode of the instruction can have an impact on the energy consumption of a given operation. Table 8 gives the average energy consumption across random operands when the addressing mode is set to indexed. There is a 41% increase in average energy per operation when using indexed mode instructions. This is due to the extra cycle(s) required to fetch operands from memory.

Table 7: Average Energy Consumption Across Random Operands

| **Instruction** | **Energy** |
|---|---|
| ADD | 1.01E-10 |
| ADDC | 1.01E-10 |
| AND | 9.92E-11 |
| BIC | 9.79E-11 |
| BIS | 9.89E-11 |
| BIT | 9.46E-11 |
| CMP | 9.54E-11 |
| DADD | 9.98E-11 |
| MOV | 7.88E-11 |
| SUB | 9.86E-11 |
| SUBC | 9.93E-11 |
| **Average** | **9.67E-11** |

As a reference point the openMSP430 is simulated across the same instruction and input pattern combinations. From Table 9 and Table 10 it is clear that the openMSP430 uses significantly less energy per operation than the MTNCL MSP430. To understand why, a deeper investigation is conducted in the Analysis section of this chapter.

### 5.2.2 Area and Leakage Power

Both the openMSP430 and MTNCL MSP430 are implemented using IBM 8RF 130nm process. The instance count, area, and leakage for both designs are given in Table 11. The

MTNCL MSP430 contains more instances and has a larger area yet its leakage is more than 2× less.

Table 8: Average Energy Consumption across Random Operands for Indexed Addressing Mode Instructions

| **Instruction** | **Energy** |
|---|---|
| ADD | 1.32E-10 |
| ADDC | 1.32E-10 |
| AND | 1.31E-10 |
| BIC | 1.31E-10 |
| BIS | 1.31E-10 |
| BIT | 1.23E-10 |
| CMP | 1.22E-10 |
| DADD | 1.30E-10 |
| MOV | 7.48E-11 |
| SUB | 1.31E-10 |
| SUBC | 3.36E-10 |
| **Average** | **1.43E-10** |

Area and leakage power are important metrics when evaluating the quality of a circuit design. Typically, they are correlated. As area increases so does leakage power; however, there are factors that can lead to an increase in area without affecting leakage. These factors include wiring congestion, layout efficiency, and transistor type. Wiring congestion can cause extra space to be added between standard cells thereby increasing the overall area, but not the transistor width.

Table 9: openMSP430 Average Energy Consumption across Random Operands

| **Instruction** | **Energy** |
|---|---|
| ADD | 1.76E-11 |
| ADDC | 1.60E-11 |
| AND | 1.79E-11 |
| BIC | 1.78E-11 |
| BIS | 1.92E-11 |
| BIT | 1.74E-11 |
| CMP | 1.80E-11 |
| DADD | 1.92E-11 |
| MOV | 1.99E-11 |
| SUB | 1.71E-11 |
| SUBC | 1.78E-11 |
| **Average** | **1.80E-11** |

Congestion is a function of gate fanout as determined by the design's logic synthesis. Layout efficiency is a measure of the density of each standard cell. It quantifies how much of the cell area is used for wiring and how much is used for transistors. A lower efficiency means less leakage per unit area in the standard cell. Layout efficiency is determined by the logic function of the cell, process DRC rules, standard row height, and the layout engineer. Typically, MTNCL gate libraries have lower layout efficiency compared to synchronous standard cells due to more wiring and smaller transistors. MTNCL logic gate layouts have a PMOS to NMOS area ratio of approximately 2-to-1 leaving a large unused area on the NMOS side of the gate. The threshold voltage and carrier type of the transistors can also have a large affect on leakage power without changing the area. Low-Vt transistors have higher leakage current than High-Vt transistors, and NMOS transistors leak 3-10× as much as PMOS.

Table 10: openMSP430 Average Energy Consumption across Random Operands for Indexed Addressing Mode Instructions

| **Instruction** | **Energy** |
|---|---|
| ADD | 1.76E-11 |
| ADDC | 1.60E-11 |
| AND | 1.79E-11 |
| BIC | 1.77E-11 |
| BIS | 1.94E-11 |
| BIT | 1.76E-11 |
| CMP | 1.84E-11 |
| DADD | 1.91E-11 |
| MOV | 2.03E-11 |
| SUB | 1.74E-11 |
| SUBC | 1.79E-11 |
| **Average** | **1.81E-11** |

When analyzing a circuit prior to place and route, an interesting metric to consider is the sum of the width and length of each transistor. For each transistor in the design the width and length of the poly over active is summed.

$$T_{Width} = \sum width(T_i)\, , T_{length} = \sum length(T_i) \tag{11}$$

In the case of the MTNCL MSP430 there are four types of transistors to consider: Low-Vt and Standard-Vt variations of PFET and NFET. The openMSP430 uses only Standard-Vt transistors and thus has just two transistor types in the design. The width and length for each type of transistor are given in Figure 17. While the MTNCL MSP430 uses Low-Vt transistors, they make up a small proportion of the overall design and don't have a large effect on overall leakage.

Table 11: Area and Leakage Power Comparison

| | **MTNCL MSP430** | **openMSP430** |
|---|---:|---:|
| Standard Cell Instances | 11797 Gates | 9546 NAND2 Equivalents |
| P&R Area | 0.7mm$^2$ | 0.2mm$^2$ |
| Leakage Power | 1.06E-05W | 2.18E-05W |

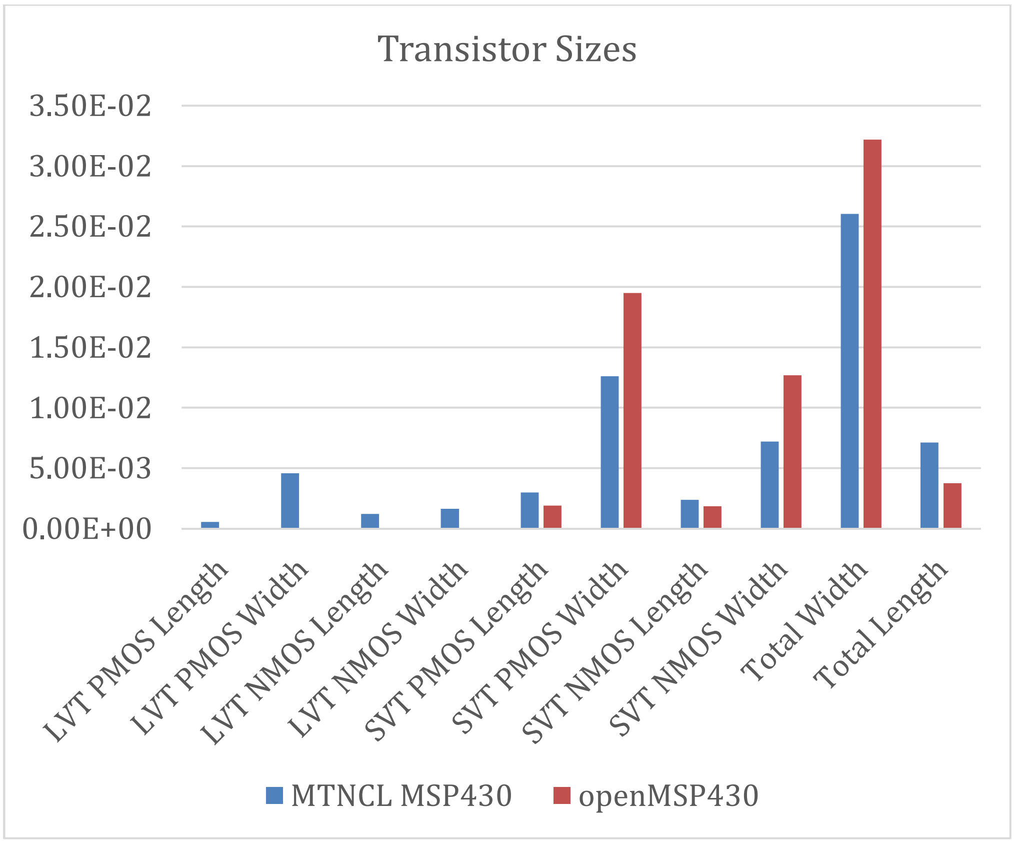

Figure 17: Transistor Size Comparison between MTNCL MSP430 and openMSP430 (units in meters)

The leakage of a transistor is proportional to W/L as shown in Equation X. From Figure 17 it is clear that the total width of the transistors in the MTNCL MSP430 is smaller than the total width of the transistors in the openMSP430, while the total length of the MTNCL MSP430 is larger than that of the openMSP430. The MTNCL MSP430 has an average W/L ratio of 3.66 and the openMSP430 has an average W/L ratio of 8.59. Therefore, despite having a larger area and more standard cell instances, the leakage power of the MTNCL MSP430 is lower than that of the openMSP430.

$$I_{off} = 100 \cdot \frac{W}{L} \cdot e^{-qV_t/\eta kT} \tag{12}$$

## 5.3 Analysis

Given that previous MTNCL designs in [15] and [17] showed a reduction in dynamic power over synchronous comparison circuits, it is unexpected for the dynamic power consumption of the MTNCL MSP430 to be higher than the openMSP430. A more detailed analysis of the MTNCL power consumption is required. The following section examines the MTNCL MSP430 power by functional block to determine the cause for the high dynamic power consumption.

Figure 18 lists the average energy per operation of different functional blocks across the two cores. Since the MTNCL MSP430 and openMSP430 have different partitioning schemes, a finer-grained breakdown of the Control, Execution Unit, and Instruction Decoder is not possible. The "Core" block contains all other blocks except the Register File.

The functional block breakdown shows only two functional blocks where MTNCL MSP430 has lower energy per operation than the openMSP430. These are the ALU and the Timer. The energy per operation improvement in the Timer is not very interesting since it is a clone of the openMSP430's timer with just a handful of extra gates. The reason it has lower energy consumption is due to shorter runtime and less toggling on the clock signal on its input. Thus, the ALU is chosen for further analysis and is compared to the MTNCL MSP430 Core to determine which characteristics lead to lower energy consumption compared to synchronous blocks with the same function.

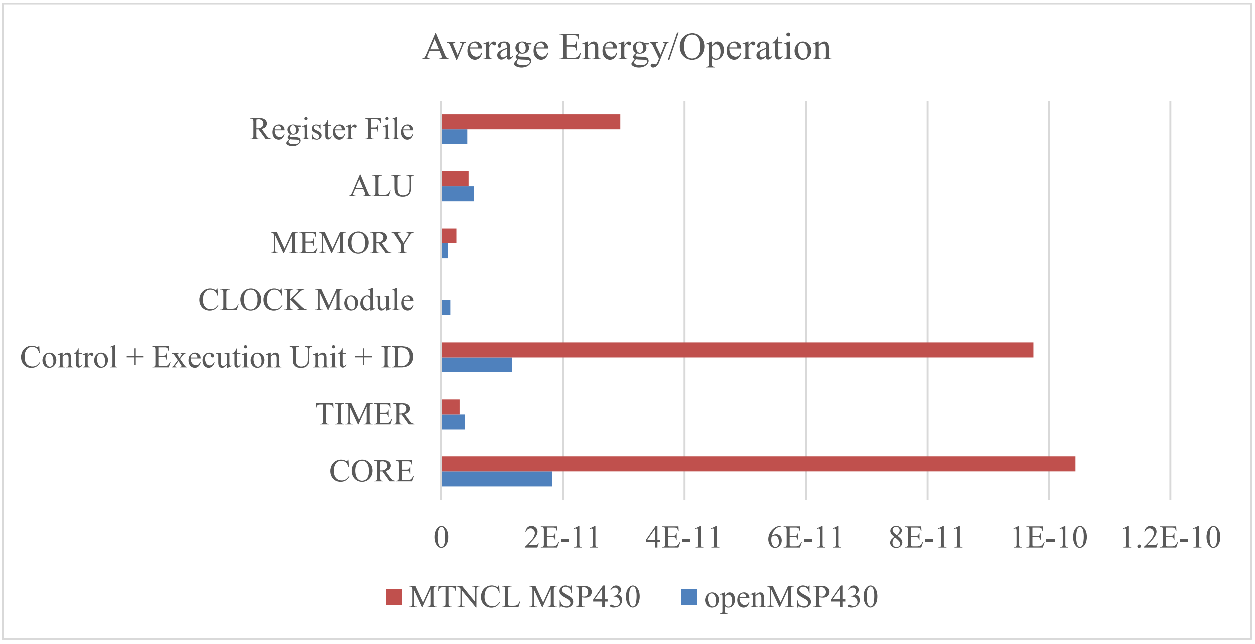

Figure 18: Energy per Operation across MSP430 Functional Blocks

MTNCL has the advantage of not requiring a clock tree to synchronize various logic blocks. It does, however, require *sleep* signal propagation and handshaking circuitry. These can be grouped into three categories: *sleep* propagation, completion detection, and registration. Measuring the energy consumed by this overhead circuitry as a percentage of total circuit energy can reveal inefficiencies in MTNCL designs. Figure 19 gives a comparison of the percentage of energy devoted to completion, registration, and *sleep* propagation for the Core and ALU. Clearly, the ALU's energy advantage over synchronous is not due to a smaller overhead as the Core devotes a larger percentage of energy to combinational logic and less to MTNCL overhead than the ALU. Additionally, the MTNCL MSP430 ALU shows an energy improvement over the synchronous openMSP430 ALU while the MTNCL Core does not.

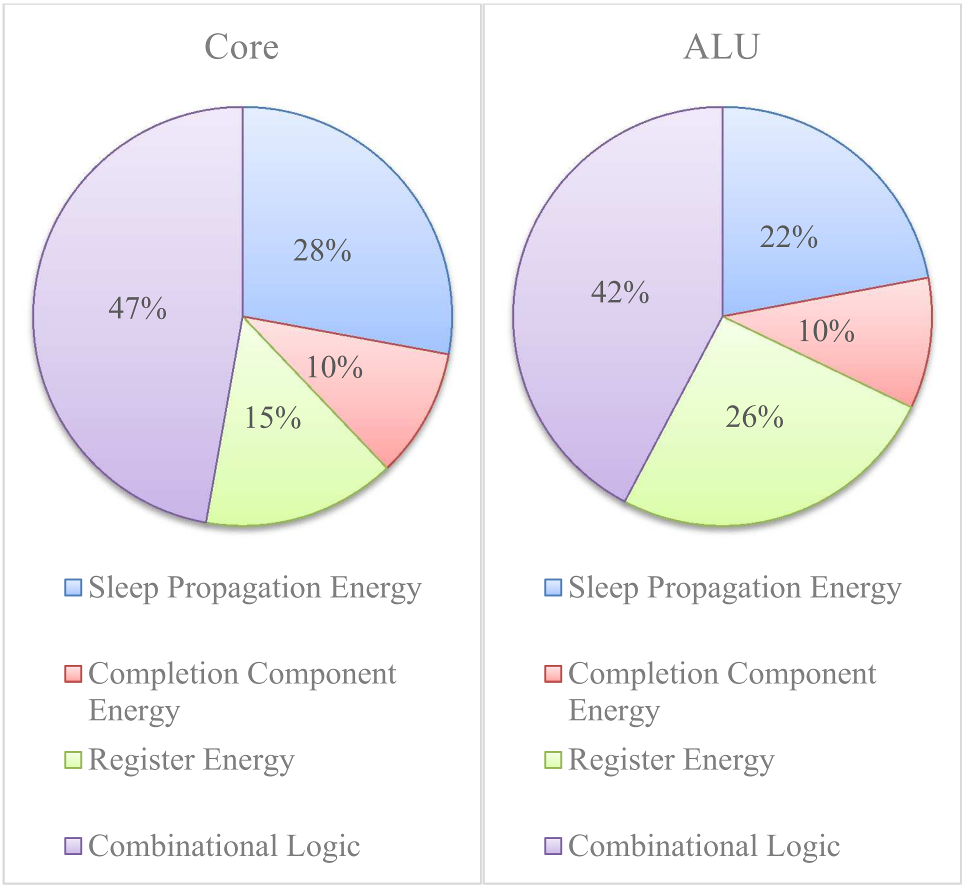

Figure 19: Energy Consumption of MTNCL MSP430 Handshaking, *Sleep* Propagation, and Combinational Logic Circuitry

Aside from MTNCL overhead, another potential cause of the MTNCL MSP430 Core's extra energy consumption is its SS logic. SS has not been used in MTNCL up to this point and should be ruled out as a potential reason for the increase in energy per operation compared to the synchronous reference. SS is facilitated by always-on gates which control the sleeping of the individual pipelines. The overhead of these always-on gates is measured in the core and found to make up only 16% of the total energy. Since the extra power consumption of the core is several times larger compared to the openMSP430, the extra 16% of SS can be ruled out as the cause of the MTNCL MSP430 energy inefficiency.

Finally, analyzing the MTNCL Core and ALU in terms of energy density and total transistor area reveals the cause of the high energy consumption of the MTNCL MSP430 Core to be inefficient logic synthesis. This is revealed by comparing the area of the ALUs and Cores. The MTNCL MSP430 Core is 54% larger than the openMSP430 Core, while the MTNCL MSP430 ALU is 32% smaller than the openMSP430 ALU. In general, comparing MTNCL and synchronous blocks that perform the same function, it is expected that the transistor area ratio would be similar. Under that expectation the MTNCL MSP430 Core should be approximately 30% smaller than the openMSP430 Core. This can be seen in [17] where a MTNCL AES core is compared to a synchronous AES core. Both designs perform the same function, and the transistor area of the MTNCL AES is 24% smaller than the transistor area of the synchronous AES.

Since the MTNCL completion, registration, and sleep propagation logic is not the cause of the Core's extra energy consumption, it must be that the additional overhead is spread out across the whole design. The best explanation for this is an inefficient RTL-to-gate logic synthesis, which is due to manual design of the control logic for the MTNCL MSP430 Core. The openMSP430 synthesis on the other hand is automated. The control logic includes many variables and logic minimization with more than four variables is impractical to do by hand. Thus, control logic in the MTNCL MSP430 is added incrementally rather than optimized as one large piece. This results in redundant gates that could be optimized out with a higher-level automated approach.

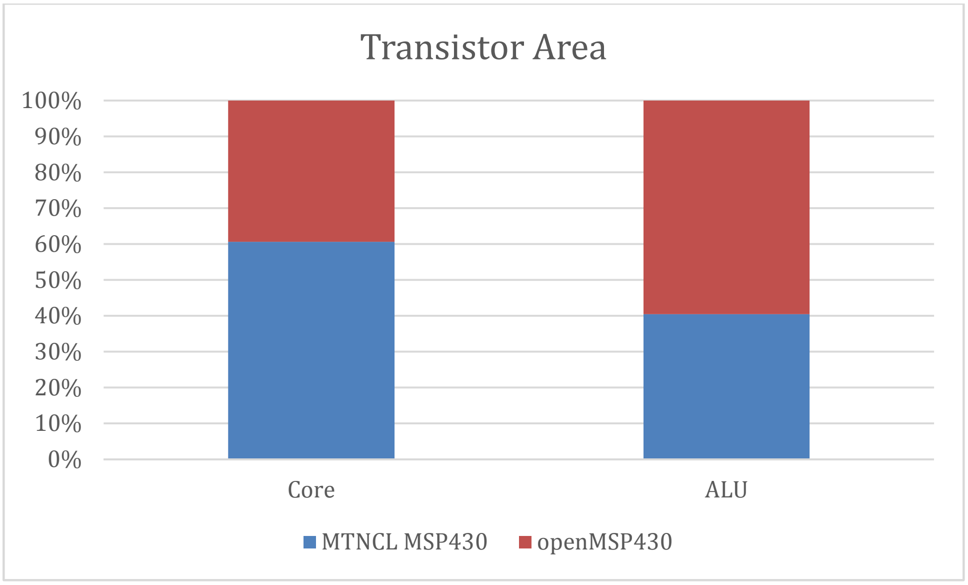

Figure 20: Scaled Comparison of Transistor Area between MTNCL MSP430 ALU and Core Compared to openMSP430 ALU and Core

This theory explains how a simpler block like the MTNCL ALU which contains minimal control logic can be lower power than the synchronous ALU while the more complex blocks like the MTNCL Control and Core are much higher power. The MTNCL logic synthesis for the ALU is more easily optimized by hand since it is made up of mostly well-known arithmetic blocks. As blocks include more complex control circuitry the number of redundant gates from inefficient synthesis increase yielding worse energy per operation.

A comparison with the work in [17] confirms this. The transistor width of the MTNCL AES core is 24% smaller than the synchronous AES benchmark. It should also be noted that the power consumption follows the total transistor width. The MTNCL AES is lower power than the synchronous AES. The AES core is largely arithmetic logic and has much less control logic than the MTNCL MSP430. Thus, there is not as much opportunity for an inefficient manual synthesis to reduce the energy efficiency of the AES.

Table 12: Transistor Width Comparison between Synchronous and MTNCL MSP430 Microcontrollers and AES Encryption/Decryption Cores

| | **Synchronous** | **Asynchronous** |
|---|---:|---:|
| MSP430 | 2.6E-02m | 3.22E-02m |
| AES | 4.14E-02m | 3.33E-02m |

Another example to confirm the theory is in the comparison of a MTNCL floating point coprocessor [15] with a synchronous benchmark. The MTNCL design has 16% fewer transistors and consumes significantly less energy per operation. Again, the design is mainly arithmetic logic with minimal control circuitry.

# 6 CONCLUSION

This dissertation expands on previous work in asynchronous logic circuit design. It focuses on the promising MTNCL asynchronous architecture and develops the most complex MTNCL design to date. The MTNCL MSP430 is a fully functioning low-power microcontroller based on the industry standard design from TI. The MTNCL MSP430 incorporates novel techniques to reduce power consumption such as Selective Sleeping and a low-area and low-power MTNCL delay insensitive register file.

The MTNCL MSP430 is benchmarked against an open source version of the MSP430 called openMSP430. Both designs are simulated to measure energy consumption. The MTNCL MSP430, while incorporating low-Vt transistors and having larger area, still manages lower leakage power than the openMSP430 due to a better W/L ratio.

Dynamic power is measured across a variety of operations. While the MTNCL MSP430 has some functional blocks that consume less energy than those of the openMSP430, overall the MTNCL MSP430 has a higher energy per operation. A breakdown of the energy data reveals the cause to be an inefficient synthesis of the control logic. This result is confirmed by comparing across other MTNCL designs, which up to this point are arithmetic based datapaths. This important finding highlights the need for better automated MTNCL synthesis techniques. For MTNCL to compete with synchronous architectures in larger designs such as a full microcontroller, an improved automated design flow including better synthesis techniques is required.

# 7 REFERENCES

[1] K. Yeo and K. Roy, Low-Power VLSI Subsystems, McGraw-Hill Professional, 2004.

[2] P. Palangpour, "CAD Tools for Synthesis of Sleep Convention Logic," University of Arkansas, Fayetteville, 2013.

[3] S. Kwak, H.-W. Lee, Y. Zafar, M.-H. Oh and D. Har, "Design of Asynchronous MSP430 Microprocessor Using BALSA Back-End Retargeting," in *Programmable Logic*, Sao Carlos, 2009.

[4] I. E. Sutherland, "Micropipelines," *Communications of the ACM,* vol. 32, no. 6, pp. 720-738, June 1989.

[5] A. J. Martin and M. Nystrom, "Asynchronous Techniques for System-on-Chip Design," *Proceedings of the IEEE,* vol. 94, no. 6, pp. 1089-1120, 2006.

[6] D. H. Linder, "Phased Logic: A Design Methodology for Delay-Insensitive, Synchronous Circuitry," Mississippi State University, Starkville, 1995.

[7] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Addison-Wesley, 1980, pp. 218-262.

[8] T. S. Anantharaman, *A Delay Insensitive Regular Expression Recognizer,* 1986.

[9] N. P. Singh, "A Design Methodology for Self-Timed Systems," MIT, 1981.

[10] I. David, R. Ginosar and M. Yoeli, "An Efficient Implementation of Boolean Functions as Self-Timed Circuits," *IEEE Transactions on Computers,* vol. 41, no. 1, pp. 2-10, 1992.

[11] K. M. Fant and S. A. Brandt, "Null Convention Logic: A Complete and Consisten Logic for Asynchronous Digital Circuit Synthesis," *International Conference on Application Specific Systems, Architectures, and Processors,* pp. 261-273, 1996.

[12] J. Kao and A. Chandrakasan, "Dual-Threshold Voltage Techniques for Low-Power Digital Circuits," *IEEE Journal of Solid-State Circuits,* vol. 35, no. 7, pp. 1009-1018, 2000.

[13] J. Di and S. C. Smith, "Ultra-Low Power Multi-Threshold Asynchronous Circuit Design". United States Patent 7,977,972 B2, July 2011.

[14] L. Zhou, "Ultra-Low Power and Radiation Hardened Asynchronous Circuit Design," University of Arkansas, Fayetteville, 2012.

[15] L. Zhou, R. Parameswaran, F. Parsan, S. C. Smitch and J. Di, "Multi-Threshold NULL Convention Logic (MTNCL): An Ultra-Low Power Asynchronous Circuit Design Methodology," *Journal of Low Power Electronics,* pp. 81-100, 2015.

[16] 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, 2010.

[17] M. Hinds, B. Sparkman, J. Di and S. C. Smith, "An Asynchronous Advanced Encryption Standard Core Design for Energy Efficiency," *Journal of Low Power Electronics,* vol. 9, no. 2, pp. 175-188, 2013.

[18] B. Hollosi, "Design and Analysis of an Adaptive Asynchronous System Architecture for Energy Efficiency," University of Arkansas, Fayetteville, 2012.

[19] L. Men and J. Di, "An Asynchronous Finite Impulse Response Filter Design for Digital Signal Processing Circuit," *International Midwest Symposium on Circuits and Systems,* pp. 25-28, 2014.

[20] O. Girard, "openMSP430 Documentation," 2012. [Online].

[21] Designing Asynchronous Circuits using NULL Convention Logic (NCL), Morgan & Claypool Publishers, 2009.

[22] L. Zhou, R. Parameswaran, R. Thian, S. C. Smith and J. Di, "MTNCL: An Ultra-Low Power Asynchronous Circuit Design Methodology," 2010.