

# Delay-Insensitive Asynchronous Circuits for Operating under Extreme Temperatures

**Brent Hollosi<sup>1</sup>, Jia Di<sup>1</sup>, Scott C. Smith<sup>2</sup>, H. Alan Mantooth<sup>2</sup>**

<sup>1</sup>Computer Science and Computer Engineering Department, <sup>2</sup>Electrical Engineering Department  
University of Arkansas, {bhollor, jdi, smithsco, mantooth}@uark.edu

**Abstract:** *Space electronics are exposed to ultra-wide temperature swings. Synchronous ICs exposed to such swings generally require additional overhead, such as heat shields or warm boxes, to counter potential timing violations. This adds to the weight, size, and power consumption of the electronic system. Due to its flexible timing requirements, delay-insensitive (DI) asynchronous logic has the natural ability to operate across an ultra-wide temperature range, making it an ideal choice for space electronics. This paper presents the testing results of a DI 8031 microcontroller from 125°C to -180°C.*

**Keywords:** asynchronous; delay-insensitive; extreme temperature; NULL Convention Logic

## Introduction

Electronics on spacecrafts are exposed to ultra-wide temperatures as high as +180°C and as low as -230°C. In such environments, MOS transistors exhibit significant variations in their switching speed, which, in synchronous integrated circuits (ICs), can lead to crippling timing violations. In order to prevent this, synchronous ICs are protected via onboard heat shields or warm boxes to maintain ambient temperatures in a much smaller range. This additional onboard equipment not only adds to the weight, size, and power of the electronic systems, but also degrades system reliability. Therefore, ultra-wide temperature electronics, which are capable of operate properly across large temperature swings while not requiring additional heating/cooling subsystems, are highly desirable.

Published research and experimental results for wide temperature devices are promising. Several passive components and standard Si-based devices operate satisfactorily down to approximately -200 °C despite some performance degradation. However, most of the devices tested are not capable of performing complex computation/control operations, which are prevalingly handled by synchronous circuits.

Synchronous circuits, controlled by clocks, generally have strict timing requirements which limit the range of temperatures these circuits can naturally operate in. Furthermore, research has shown that not only does MOS transistor variation scale inversely with feature size, but also smaller MOS transistors are more significantly affected by temperature variations than larger ones. It follows that synchronous systems utilizing MOS transistors will require increasingly more precise and cost-intensive

protective mechanisms if space electronics are to be migrated to smaller process sizes. Hence, new technologies need to be identified for designing ultra-wide temperature electronics.

Asynchronous logic, specifically delay-insensitive (DI) asynchronous logic, is one such paradigm that shows great promise in constructing ultra-wide temperature reliable circuits. Its utilization of handshaking protocols rather than global clock structures provides many benefits such as no clock skew, flexible timing requirements, improved power efficiency, and low noise/emission generation. Furthermore, due to their clockless nature, DI asynchronous circuits generally require very little, if any, timing analysis. In fact, as long as the transistors can switch properly, DI asynchronous circuits should always function correctly; in other words, they are correct-by-construction. Hence, DI asynchronous circuits' ability to maintain correct operation regardless of ultra-wide temperature swings makes them an ideal candidate for space electronics applications.

## NCL

DI asynchronous methodologies like NULL Convention Logic (NCL) do not use clocks for data validation; instead, they use handshaking protocols, which eliminate the need for timing analysis. NCL circuits utilize multi-rail signals to achieve delay-insensitivity. The most prevalent multi-rail encoding in NCL circuits is the dual-rail scheme. A dual-rail signal,  $S$ , consists of two wires,  $S^0$  and  $S^1$ . There are three valid states for these two wires, the DATA0 state, the DATA1 state, and the NULL state. The DATA0 state ( $S^0 = 1, S^1 = 0$ ) corresponds to a Boolean logic 0, the DATA1 state ( $S^0 = 0, S^1 = 1$ ) corresponds to a Boolean logic 1, and the NULL state ( $S^0 = 0, S^1 = 0$ ) corresponds to the empty set meaning that  $S$  is not valid data. The invalid state ( $S^0 = 1, S^1 = 1$ ) never occurs in normal NCL circuit operation; hence, the two rails are typically thought of as being mutually exclusive.

NCL circuits are designed using combinations of the 27 fundamental gates, called threshold gates. The standard threshold gate is the TH $m$ n gate, where  $1 \leq m \leq n$ . TH $m$ n gates have  $n$  inputs; at least  $m$  of the  $n$  inputs must be asserted before the output will become asserted. Once a gate's output is asserted, all  $n$  inputs must be deasserted for the output to be deasserted. The gate holds its current state through *hysteresis*, a feedback function that ensures complete transitions to either the asserted or deasserted state [2].

## NCL 8031 Design

### Overview

It is first important to note that this 8031 was designed with the intention to be pin-to-pin compatible with synchronous 8031s. As such, some elements of the design were required to be synchronous. These elements include the UART, interrupt detection mechanisms in Interrupt Control, and all memory access state machines in Timing and Control (TAC). Other elements like the SRAM and Timer/Counters are by nature synchronous. As such, a clock signal is used to control data flow of these elements. Furthermore, the term *frequency*, when used in the following sections, describes the frequency of the clock supplied to these synchronous elements. All other elements of the design are DI asynchronous elements and do not require a clock. Figure 1 shows broadly how this 8031 is organized.

In general, it is easier to conceptually grasp the 8031's design if one divides it into three parts: source elements, destination elements, and the TAC. Source elements are composed of DI registers and/or DI logic that are responsible for generating DATA/NULL wavefronts. Destination elements are composed only of DI registers. Their sole purpose is to store DATA/NULL wavefronts and indicate successful storage. The TAC, composed of a 3-ring register and some logic, initiates DATA/NULL wavefront generation from source elements while enabling a destination element to receive the wavefront. It also monitors the destination element's completion signal to determine successful reception of said wavefront. Thus, the TAC is responsible for controlling the successful communication of data between source and destination elements.

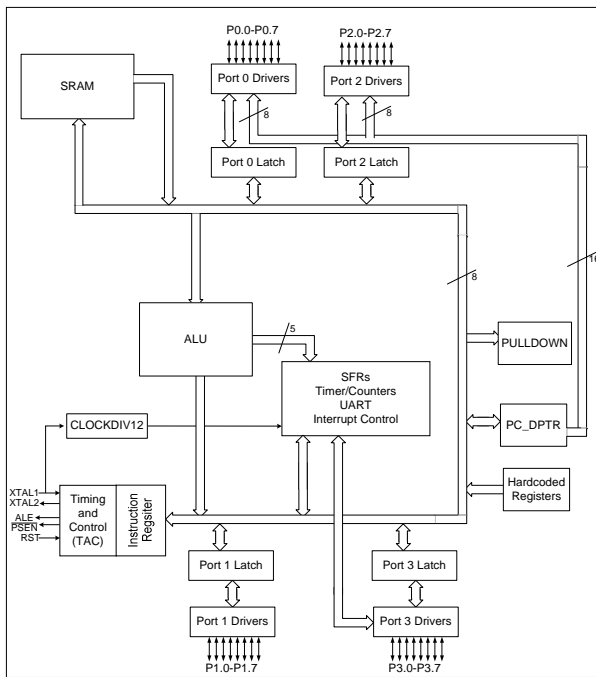


Figure 1: 8031 Top Level Diagram

To understand this process more precisely in terms of the DATA/NULL cycle, one must first know that every source and destination element is controlled by the TAC via a  $K_i$  signal, and every destination element indicates successful storage via a  $K_o$  signal sent back to the TAC. During a reset, the TAC is in its NULL phase; hence, all  $K_i$  signals will be low, and all  $K_o$  signals will be high. Directly after reset, the TAC will transition to its DATA phase. During this phase, some  $K_i$  signals are asserted by the TAC. These control signals activate one, and only one, source and destination element between which a DATA wavefront is communicated. The successful passing of a DATA wavefront from a source to a destination, and hence fall in that destination's  $K_o$  signal, indicates the completion of the TAC's DATA phase. When the low  $K_o$  signal is detected by the TAC, it transitions into its NULL phase. During the NULL phase, all  $K_i$  signals are deasserted allowing a NULL wavefront to be stored by the destination element. Upon successful storage of the NULL wavefront, the destination's  $K_o$  signal will rise, indicating successful completion of the TAC's NULL phase. In terms of instruction execution, the DATA/NULL cycling of the TAC is referred to as a state.

### Instruction Execution

Each of the 110 unique instruction types requires a unique state sequence to complete the instruction; however, each one must start with the same instruction fetch procedure. The 8031's fetch procedure is similar to a synchronous 8031's fetch procedure in that it is controlled via the clock so that the memory access control signals may adhere to their timing specifications. However, the main dissimilarity is that the fetch procedure is initiated asynchronously by the TAC. Hence, rather than fetch continuously, as is done in a synchronous 8031, the fetch procedure is only performed when it is needed.

After a successful instruction fetch, the PC is incremented and the state sequence for the fetched instruction is completed. As an example, what follows is the state sequence for the Increment A instruction:

1. S0 – fetch,  $wr\_ir$
2. S1 –  $inc\_pc$
3. S2 –  $rd\_acc$ ,  $wr\_tmp1$
4. S3 –  $inc$
5. S4 –  $rd\_alu\_low$ ,  $wr\_acc$

For S0, the *fetch* control signal activates the ASM responsible for fetching the instruction from external memory. This ASM generates the control signals necessary for the fetch procedure, and, at the proper time, enables Port 0 to pass through the instruction onto the main bus. The  $wr\_ir$  signal enables the instruction to be stored in the Instruction Register. For S1,  $inc\_pc$  directs the Program Counter's combinational logic to increment its

value by 1. For S2, *rd\_acc* allows the value of the Accumulator to pass through onto the main bus, while *wr\_tmp1* allows the TMP1 register of the ALU to store that value. For S3, *inc* directs the ALU to increment the value of TMP1 by 1 and store it in the ALU's low byte output register. For S4, *rd\_alu\_low* allows the value of the ALU's low byte output register to pass onto the main bus, while *wr\_acc* allows the Accumulator to store the value. All instructions are implemented in this way using some differing combination of states.

After the final state in any instruction's state sequence, the interrupt state sequence begins being completed. This sequence is responsible for polling any potentially set interrupts, and, in the case of an interrupt, branching accordingly.

### Interrupts

Due to the asynchronous nature of the TAC, it is impossible to synchronously poll the interrupt flags as is done in a synchronous 8031; however, the interrupt detection mechanism can be the same. In this way, the timing constraints on generating an interrupt are the same, however the time required to poll and handle the interrupt is dependent on the current instruction which is variable due to the asynchronous nature of the 8031's operation. Because of this asynchronous handling of interrupts, synchronously generated interrupts may not be handled as usual; however, some instructions' execution time does match closely with their synchronous counterparts. Such instructions could be used to facilitate rhythmic asynchronous operation.

### Fabrication

The 8031 was physically designed using IBM SiGe 5AM process and was fabricated at MOSIS.

## 8031 Testing

### Setup

The test setup utilized an Altera DE2 FPGA board, a logic analyzer (TLA5024B), a pulse generator, and a test chamber with temperature control to test the 8031's instruction execution at -180°C, 25°C, 45°C, 85°C, and 125°C. This setup is shown in Figure 2.

### Procedure

For both cryogenic and thermogenic tests, the Altera DE2 FPGA board was used as an external program and data memory. Small programs were written for each instruction except the following: MOVX A, @Ri; MOVX A, @DPTR; and MOVX @Ri, A. Each program contained roughly 20 executions of a particular instruction as well as a data validation routine ending in a final MOVX @DPTR, A to verify correct program execution. The entire program's execution was captured via the logic analyzer in 2ns intervals. Captured data was parsed by custom scripts which ascertained the *fetch time* (the time spent utilizing

the fetch procedure), *computation time* (the time spent doing anything other than fetching), and *execution time*

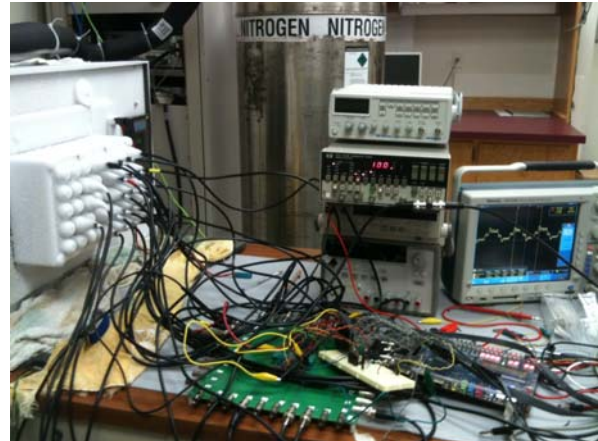


Figure 2: Test Setup

(the summation of fetch time and computation time). Each instruction was tested and captured this way at five temperature points -180°C, 25°C, 45°C, 85°C, and 125°C.

### Results

At each of the five temperature points, all instructions executed properly with a clock frequency of 12MHz or higher. The exception to this was the SJMP instruction which failed to work at -180°C with a clock frequency faster than 6MHz. Given a certain frequency, the fetch time remained relatively constant at all temperature points. Computation time scaled directly with temperature; however, maximum operating frequency did not. The maximum operating frequency scaled with temperature from 25°C to 125°C, but appeared to be inversely correlated at -180°C.

As there is too much data to adequately show all instructions' results, the fetch, computation, and execution time trends for the Add A, Rn instruction are shown in Figure 3. The full list of extracted data for the Add A, Rn instruction is also provided as an example. This data can be found in Table 1.

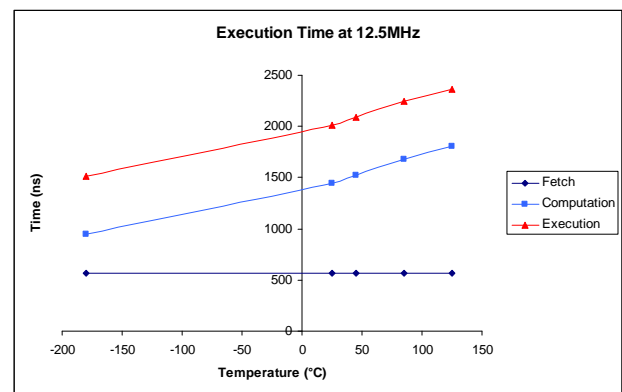


Figure 3: Add A, Rn Execution Time at 12.5MHz over Temperature

Temperature (°C)	Frequency (MHz)	Fetch Time (ns)	Computation Time (ns)	Execution Time (ns)
125	17.86	408	1806	2214
125	16.67	424	1811	2235
125	12.5	564	1804	2368
125	10	703	1807	2510
85	19.61	404	1683	2087
85	16.67	423	1682	2105
85	12.5	563	1683	2246
85	10	702	1645	2347
45	20.41	390	1528	1918
45	16.67	423	1510	1933
45	12.5	563	1528	2091
45	10	702	1505	2207
25	23.26	482	1453	1935
25	16.67	517	1446	1963
25	12.5	566	1449	2015
25	9.52	736	1388	2124
-180	15.62	444	911	1355
-180	12.5	562	950	1512
-180	10	701	901	1602

**Table 1:** Add A, Rn Timing Results

### Conclusion

In general, the data obtained from the over-temperature tests correlated with expected results. All instructions except the SJMP instruction worked at higher than nominal frequencies at all temperature points. Theoretically, these instructions should work across an even wider temperature range; however, the capabilities of the test chamber limited the temperature range to between -180°C and 125°C.

As expected, fetch time remained relatively constant over temperature; however, the maximum operating frequency did not scale with temperature. This means that timing

violations were more easily incurred as the temperature approached outlying temperature extremes.

Also, as expected, the computation time scaled directly with temperature, indicating that the asynchronous DI elements were directly influenced by ambient temperature. In fact, operation at -180°C yielded a significant increase in performance for most, if not all, instructions.

These hot and cold temperature tests have shown that DI asynchronous circuits performing computation/control operations function correctly and function well over an ultra-wide temperature range. Their ability to naturally sustain operation across such temperatures proves its advantages in extreme environments like that of space. Future work in this area includes testing the 8031 at even more extreme temperatures and exploring the robustness of DI asynchronous circuits at processes beyond 0.5µm.

### References

1. *MCS<sup>®</sup> 51 Microcontroller Family User's Manual*, February 1994, Publication number 121517, Intel Corporation
2. K. M. Fant and S. A. Brandt, "NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," *International Conference on Application Specific Systems, Architectures, and Processors*, pp. 261-273, 1996.
3. Ward, R.R., W. Dawson, L. Zhu, R. Kirschman, G. Niu, R. Nelms, O. Mueller, M. Hennessy, E. Mueller, R. Patterson, J. Dickman, and A. Hammoud. "SiGe Semiconductor Devices for Cryogenic Power Electronics - IV." *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE* (2006).
4. Davis, A., S. M. Nowick. "An Introduction to Asynchronous Circuit Design." <http://www1.cs.columbia.edu/async/publications/davis-nowick-intro-tr.pdf>
5. S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," *Elsevier's Integration, The VLSI Journal*, Vol. 37/3, pp. 135-165, August 2004.