Self-timed, Minimum Latency Circuits for the Internet of Things

Adrian Wheeldon*, Jordan Morris, Danil Sokolov, Alex Yakovlev
μSystems Group, Newcastle University, Newcastle Upon Tyne, United Kingdom

Abstract
This work presents a design flow for asynchronous, self-timed dual-rail circuits which introduces a timing assumption in the return-to-spacer phase. The design flow enables power proportionality and is demonstrated through the design of a 32-bit ripple-carry adder and a 32-bit comparator for internet of things applications. The designs are synthesized to a 65 nm cell library with state-of-the-art transistor sizing for subthreshold. Simulation results show improved performance and energy per computation across operating conditions compared with single-rail equivalents. The design flow allows extension of the power proportional philosophy to a wider range of circuits.

Keywords: dual-rail, internet of things, power proportional, self-timed

1. Introduction

Nowadays with the proliferation of internet of things (IoT) devices, we see the utilization of low power accelerators and signal conditioning circuits in many different usage scenarios: from hand-held battery-powered devices; to small, wide sensor network nodes powered by energy harvesters; and mains-powered smart-home devices which are always on.

An important factor in the IoT is the reduction of the functional supply voltage ($V_{DD,min}$). For systems powered by energy harvesters, maximum power point tracking (MPPT) is commonly implemented to extract what little power may be available [1, 2]. The nature of MPPT can lead to a very low $V_{DD}$ supplied to the circuit. In many cases it is preferable for the circuit to continue operation under this condition, albeit with reduced performance. If the circuit did not function at this $V_{DD}$, the energy might simply be wasted, or stored in an inefficient battery system.

Asynchronous dual-rail circuits give the possibility to detect when the circuit has completed its computation by using a self-timed approach [3]. Using a weakly-indicating circuit (such as one based on the NCL-X design methodology [4]) with completion detection, we can implement a design which is early propagative. Such a design allows flexible performance which is self-adapting to the $V_{DD}$, and therefore the power availability when coupled with an energy harvester. We call this a power proportional design.

A useful side-effect of a power proportional design comes from its ability to adapt to the $V_{DD}$. In systems where we have high power availability, but lower energy availability (such as battery-powered systems), we can artificially lower the $V_{DD}$, simulating a power-sparse condition. Operating over a long period of time in this condition leads to low energy usage overall. In such battery-powered applications, we are interested in operating at the system’s minimum energy point (MEP)—the point at which the system consumes the minimum amount of energy per computation. Operating at the MEP will lead to maximized battery life. The amount of energy per computation at the MEP ($E_{min}$) occurs close to the point where the leakage energy of the circuit is equal to its dynamic energy ($E_{leak} \approx E_{dyn}$). $E_{min}$ can be characterized as a
function of supply voltage. As a result, we define $V_{DD, MEP}$ as the supply voltage for which $E_{min}$ is achieved. We are interested specifically in operation where $V_{DD} < V_{TH}$ since this is where the MEP tends to lie for complex CMOS designs [5, 6].

In the contrary situation where we have high power and high energy availability (such as in a mains-powered device), we can supply a higher $V_{DD}$. The power proportional properties of the system will allow it to operate with higher performance, albeit at the expense of greater energy per operation.

In IoT edge nodes, operands can appear spuriously and far apart in time. This leads to long periods of inactivity in the circuit where it is idle and does no work. In a dual-rail design, we normally must acknowledge both codeword and spacer to avoid hazards. However, if we make an assumption that the idle time is sufficiently long to allow the circuit to reset to spacer, then we no longer need to acknowledge the spacer. This is especially true in single-cycle accelerators and signal conditioning circuits where subsystems are duplicated for performance; in favor of resource sharing. This timing assumption can lead to reduced completion detection circuitry, and therefore reduced energy consumption, while retaining the benefits of early propagation and power proportionality.

Here we have discussed how a power proportional design can operate under three different usage scenarios with no extra design effort: energy harvesting, battery-powered and mains-powered. It is for this reason that we present 32-bit adder and comparator designs based on the power proportionality principle.

The main contributions of this work are as follows:

- a novel sizing strategy is presented for CMOS cells which reduces the effects of process variation on propagation delay in the subthreshold region;
- two types of circuit are identified as candidates for early-propagative designs—namely adders and comparators;
- a design flow is presented for dual-rail IoT circuits where acknowledgment of the spacer can be replaced with a timing assumption.

Section 2 introduces the cell library upon which this work is based. Section 3 demonstrates the dual-rail design flow. Section 4 uses the power proportional adder design as a case study. Section 5 applies the same methodology to a comparator design. Section 6 shows the results of the designs when benchmarked in a simulation environment. Section 7 summarizes the work and results.

Throughout the paper, dual-rail refers to the asynchronous design style which utilizes two wires per data bit [7]. Single-rail refers to a conventional circuit utilizing one wire per data bit.

2. Full Diffusion Cell Library

The cell library in [8] is based on a commercial 65 nm low-power process. The library uses a novel, full diffusion sizing strategy for subthreshold with 100 nm transistor lengths. It features improved delay variability and increased performance in subthreshold at the expense of greater leakage current when compared to libraries sized for superthreshold.

During ion implantation in today’s deep submicron technologies, dopants may penetrate more or less deeply into the channel. Known as random dopant fluctuation (RDF), this causes up to 70% variation in propagation delays during subthreshold operation [9]. RDF is proportional to the inverse square root of the device’s diffusion area, therefore increasing gate area leads to lower RDF. The full diffusion sizing strategy maximizes the gate area therefore reducing delay variability due to RDF, whilst maintaining parallel transistors (contrast Figures 1 and 2). This novel sizing strategy shows a delay variation of up to 30% compared to 200% for minimum and regular diffusion sizing strategies [8]. These attributes are summarized for the standard-drive inverter cells in Table 1. The full diffusion cells with 1, 2, 3 and 4 fingers are equivalent to 210 nm, 420 nm, 630 nm and 840 nm regular cells respectively. Furthermore, performance increases due to fingering, and at a rate proportional with leakage power. This leads to higher performance at a similar energy cost.

Transistor stacks degrade the $I_{on}/I_{off}$ ratio of cells, reducing performance whilst increasing leakage power. In superthreshold the effect is small enough that several transistors can be stacked without a problem. In subthreshold however, the effect is greater to the point where only transistor stacks of two are used. Connecting cells to create more complex logic is favored over more complex cells. The transistor stack limitation leads to a small
number of cells in the library, namely: NAND2, NOR2, INV, AOI22, OAI22.

Each full diffusion cell exists in a one, two, three and four finger variant. Exceptions are the AOI22 and OAI22 cells which do not exist in the four finger variant due to cell dimension limitations. An increased number of fingers results in increased performance and leakage power. These variants give rise to a performance and leakage range for the synthesis tool to work with.

In superthreshold, carrier mobility decreases with increasing temperature and is the main cause of temperature-related variation. However in subthreshold, variation in $V_{TH}$ takes over due to the exponential dependence on the transistor’s current characteristic. $V_{TH}$ has a negative correlation with temperature [10] and therefore cells perform best at high temperatures and fail at low temperatures—opposite to what is observed in superthreshold.

<table>
<thead>
<tr>
<th>Cell Propagation Delay (ns)</th>
<th>Leakage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>Std. Dev.</td>
</tr>
<tr>
<td><strong>Regular Diffusion</strong></td>
<td></td>
</tr>
<tr>
<td>210 nm 140</td>
<td>35.8</td>
</tr>
<tr>
<td>420 nm 155</td>
<td>30.9</td>
</tr>
<tr>
<td>630 nm 159</td>
<td>26.3</td>
</tr>
<tr>
<td>840 nm 164</td>
<td>23.7</td>
</tr>
<tr>
<td><strong>Full Diffusion</strong></td>
<td></td>
</tr>
<tr>
<td>1F 164</td>
<td>23.7</td>
</tr>
<tr>
<td>2F 128</td>
<td>19.0</td>
</tr>
<tr>
<td>3F 89.9</td>
<td>12.8</td>
</tr>
<tr>
<td>4F 68.7</td>
<td>8.50</td>
</tr>
</tbody>
</table>

3. Design Flow

A dual-rail NCL-X design style is chosen for its robustness and early-propagative nature as discussed in Section 1. These attributes allow better average-case performance in exchange for silicon area when compared to a single-rail design. This design style is not as robust as dual-rail NCL-D from a timing perspective, however it benefits from a more tolerable area overhead and better performance [4]. A single-rail bundled-data circuit is self-timed and comes at a much lower area overhead—however it does not offer the same robustness nor can it be early-propagative. From hereon in, all references to dual-rail will refer to the dual-rail NCL-X design style.

In dual-rail encoding, two wires are used to represent a codeword. For a single bit $x$, the dual-rail encoding consists of the positive and negative rails $\{x_1, x_0\}$. $x = 0$ is encoded as $\{0, 1\}$, and $x = 1$ is encoded as $\{1, 0\}$. One of the remaining states—$\{0, 0\}$ or $\{1, 1\}$—is chosen to represent the empty state, referred to as a spacer, which separates codewords temporally so they can be distinguished from each other. Care must be taken to correctly handle spacers in the design, otherwise data hazards could occur where one codeword overtakes another [11]. The remaining state is forbidden and must not be used.

Completion detection circuitry can be added to a dual-rail circuit to make it self-timed. A completion detector generates a true value when the state of a bit is either $\{0, 1\}$ or $\{1, 0\}$, indicating a valid codeword. The output of these completion detec-
tors can be combined to create a done signal for the entire computation. Figure 3 shows how this can be done for a circuit using an all-zeroes spacer (\(\{0,0\}\)). The AND gate denoted with a ‘C’ represents a Muller C-element [12] which acknowledges all-zeroes at its inputs. This is in addition to acknowledging all-ones like a regular AND gate. The C-element ensures that all output signals return to spacer before the done signal is deasserted. The OR gates can be replaced with NAND gates if the circuit uses an all-ones spacer. Although strictly an XOR gate should be used, we design our circuits to guarantee the invalid state will not be entered so these simpler gates can be used. The done signal facilitates inter-block handshaking to self-time the design.

Completion detection allows the circuit to be early-propagative, gives rise to power proportionality, and minimizes latency. In this work we have substituted the C-element for an AND gate. This implementation only acknowledges the all-ones state, and not the all-zeroes state—giving the side-effect that race hazards can occur in the return-to-spacer phase. It therefore makes the assumption that sufficient time is given for all logic to reset to spacer before a new operand is applied to the primary inputs. This assumption is reasonable in IoT applications where activity factor is low and there may be extensive idle-time between operands. Figure 4 illustrates how a spacer hazard may occur in this simplified completion detection. Signals \(w_1, w_0, y_1\) and \(y_0\) are outputs of a dual-rail circuit. Input shows the state of the circuit’s inputs (codeword or spacer). done is generated from simplified completion detection. Signal \(y_1\) did not return to spacer (logic 0) before the second codeword was applied. As a result, done is asserted prematurely and the outputs are invalid. Our assumption is that this spacer time will always be sufficient for all signals to return to spacer, and therefore the hazard will not occur.

Figure 5 gives an overview of the design flow used in this work. The first step to obtaining a dual-rail circuit is the same as a single-rail circuit—we must derive a boolean equation for the output from the specification. After this, we replace any XOR/XNOR gates with alternatives since they are not present in our cell library. At this stage, we can choose to synthesize the equations to obtain a single-rail netlist. To obtain a dual-rail netlist, we instead apply dual-rail expansion as shown in Figure 6. The single-rail inputs and outputs are replaced by a positive-rail notation (eg. \(a\) becomes \(a_1\)) in the process of positive-rail insertion. Next, negative-rail optimization is used to replace inverted literals by their negative-rail equivalent (eg. \(a_1\) is replaced by \(a_0\)). Now we have obtained the final equation for the positive rail, we clone it, exchanging all gates in the equation for their duals (eg. AND changes to OR, OR changes to AND, etc.). We call this gate dualing. This gives the equation for the negative rail. Together these two equations form the dual-rail boolean equations for the original specification.

Once we have the complete set of equations for our dual-rail implementation, we can apply spacer-aware negative-gate optimization. During this optimization, care is taken to ensure that correct spacer behavior is retained, since introduction of negative gates can cause mismatched spacer polarity at the outputs. Completion detection insertion is then used to make the design self-timed. Finally, we synthesize the equations using a standard commercial tool as with any single-rail circuit.

In this work, all circuits were synthesized using Synopsys Design Compiler with a combination of fingered variants from the regular-VTH full diffusion cell library. The tool chose faster cells with more fingers in order to optimize the critical path delay. Fewer-fingered variants were chosen elsewhere in order to minimize power dissipation.
4. 32-bit Ripple-Carry Adder

We illustrate the design flow in Section 3 using a 32-bit adder as it is an essential block in any IoT system. Since leakage power is increased in the target library compared to libraries with standard transistor sizing, it is important to use circuit design techniques for the design to have a competitive \( E_{\text{min}} \). Furthermore, in silicon processes with reduced feature sizes such as the one used in this work, \( E_{\text{leak}} \) is inherently greater, requiring more effort in circuit design to keep the energy per computation low. For these reasons, a ripple-carry architecture is chosen for its small logic footprint, since a smaller logic footprint leads to a lower \( E_{\text{leak}} \) from a circuits perspective.

For the 32-bit ripple-carry adder, in the worst case, the carry signal must travel through all 32 full adder blocks before the output is valid. However, due to the carry-propagate and -generate characteristics, it is possible for the output to be valid much sooner than the worst case—after a delay of \( \log N \) carry stages on average [13]. We take advantage of this by using an early propagative design based on the dual-rail NCL-X design style discussed in Section 3.

In this section, we present a single-rail full adder design from which a dual-rail full adder is derived. These designs are chained to form 32-bit ripple-carry adders.

4.1. Single-rail

The single-rail design is derived from the well-known equations (1a) and (2a). We remove XORs in order to obtain (1b) and (2b) in terms of simple gates.

\[
\text{sum} = a \oplus b \oplus c \\
= (ab + a\overline{b}) \oplus (a\overline{b} + a\overline{b}) c \tag{1a}
\]

\[
\text{cout} = ab + (a \oplus b)c \\
= ab + (a\overline{b} + a\overline{b}) c \tag{2a}
\]

These forms allow us to extract the common term \((ab + a\overline{b})\). We apply DeMorgan’s theorem to this term to obtain it in terms of negative gates. We write (3) using \( z \) as an intermediate term for sim-
Applying DeMorgan’s theorem to (1b) and (2b) we can now use $z$ to give (4) and (5) which lead to the circuit implementation in Figure 7.

$$\text{sum} = (z + c) \cdot (z + c)$$  \hspace{1cm} (4)

$$\text{cout} = (a + b) \cdot (z + c)$$  \hspace{1cm} (5)

Figure 7: The circuit for the single-rail full adder.

4.2. Dual-rail

Equations for the dual-rail design are derived from those of the single-rail design as described in Section 3. We start from (1b) and (2b) since these are in terms of positive gates. This will ensure correct spacer behavior from the outset (since negative gates invert the spacer polarity). All of the inputs and outputs are substituted with their positive dual-rail counterparts and all inverted inputs are replaced by the corresponding negative-rail input. After noting that $ab + a\bar{b} = a\bar{b} + ab$, the result is (6) and (7). Furthermore, the negative rails (8) and (9) are derived.

$$\text{sum}_1 = (a_1b_0 + a_0b_1)c_0 + (a_1b_1 + a_0b_0)c_1$$  \hspace{1cm} (6)

$$\text{cout}_1 = a_1b_1 + (a_1 + b_1)c_1$$  \hspace{1cm} (7)

$$\text{sum}_0 = (a_0b_1 + a_1b_0)c_1 + (a_0b_0 + a_1b_1)c_0$$  \hspace{1cm} (8)

$$\text{cout}_0 = a_0b_0 + (a_0 + b_0)c_0$$  \hspace{1cm} (9)

Figure 8 shows the initial implementation of the dual-rail full adder. The circuit is made up of positive gates as shown by the groupings. These must be implemented as negative gates with a following inverter due to the cells available in the library.

To improve the circuit, we can transform the AO gates into OAI gates with inverted inputs like. The result is shown in Figure 9 where we have removed some of the double inversions and consequently reduced the number of inverters from eight to six. In the new circuit, for an all-zeros spacer at layer 0, layer 1 produces an all-ones spacer due to the inverting logic. We now have spacer inversion in layer 1 and we use the $\{1, 1\}$ state as a spacer and forbid the $\{0, 0\}$ state. Following on, layer 2 has another spacer inversion and therefore the primary outputs produce all-zeros spacers. We term this alternation of spacers between logic layers an alternating spacer protocol.

Figure 10 shows a further optimized design. Here, $\text{sum}_1$ and $\text{sum}_0$ use all-zeros spacers, whereas $\text{cout}_1$ and $\text{cout}_0$ use all-ones spacers. To achieve this, the gates directly preceding the $\text{cout}_1$ and $\text{cout}_0$ primary outputs have been moved into a new layer of logic by the introduction of inverters at their inputs. Consequently, $c_1$ and $c_0$ are moved into layer 1 so that their spacers match $\text{cout}_1$ and $\text{cout}_0$ from the previous block when the full adders are chained. This optimization removes a further two inverters from the design, reducing the amount of logic and therefore lessening the effect of leakage power on the MEP.
5. 32-bit Comparator

A comparator is another widely utilized circuit forming part of an ALU or a discrete component in IoT applications. A full comparator takes two \( n \)-bit operands, \( a \) and \( b \), and asserts \( a > b \) (greater-than), \( a < b \) (less-than), or \( a = b \) (equal).

Figure 11 shows how multiple 1-bit comparators are connected to form an \( n \)-bit comparator. The \( \text{gt} \) and \( \text{lt} \) outputs are combined through OR gates to form the \( n \)-bit outputs, whereas the equal output is derived from the final comparator in the chain. The \( \text{eval} \) input reads the \( \text{eq} \) output of the previous stage. If \( \text{eval} \) is low, the result of the previous comparison was \( \text{not equal} \) and comparison in the current stage need not take place—the outputs should be zero. If \( \text{eval} \) is high, the result of all previous comparison was \( \text{equal} \) and the current bits must be compared. If the final \( \text{eq} \) output is high, all bits of the operands are equal. This architecture can benefit greatly from early propagation as the result is known immediately after one of the 1-bit comparators asserts \( \text{gt} \) or \( \text{lt} \). The longest propagation path is \( \text{equal} \) which must ripple through all comparator stages.

\[
\begin{align*}
\text{gt} &= \overline{\text{eval}} + (a \cdot b) \\
\text{lt} &= \overline{\text{eval}} + (\overline{a} \cdot b) \\
\text{eq} &= \overline{\text{eval}} + (a \cdot b + \overline{a} \cdot \overline{b})
\end{align*}
\]

The inverted literals from these equations can be shared giving the final single-rail implementation in Figure 12.

5.2. Dual-rail

Table 2 shows that the outputs of the single-rail design have a one-hot encoding. One-hot encoding, like dual-rail encoding, is a subset of 1-of-\( n \) codes [14]. Provided a spacer separates codewords,
Table 2: Truth table for the single-rail 1-bit comparator.

<table>
<thead>
<tr>
<th>eval</th>
<th>a</th>
<th>b</th>
<th>gt</th>
<th>eq</th>
<th>lt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The switching of 1-of-n codes is monotonic which leads to delay insensitivity. We take advantage and retain the one-hot encoding at the output, deviating from the design methodology, transforming only the inputs to dual-rail encoding. The one-hot-encoded outputs can be completion detected in a similar way to dual-rail encoding—giving rise to a self-timed design.

Starting from Table 2, applying negative-rail optimization, generating redundant eval terms for each input and applying DeMorgan’s theorem; we arrive at (14) given (13), (15) and (16) follow.

\[ x_n' = \text{eval} \cdot x_n \quad (13) \]
\[ \text{gt} = a_1' + b_0' \quad (14) \]
\[ \text{lt} = a_0' + b_1' \quad (15) \]
\[ \text{eq} = (a_1' + b_1') \cdot (a_0' + b_0') \quad (16) \]

This implementation gates the dual-rail inputs to the circuit, having the advantage that dynamic power dissipation is saved in stages where eval is low. We rename the eval signal to req since it forms half a handshake between two self-timed circuit. The final circuit implementation is shown in Figure 13.

The 1-bit self-timed comparator is cascaded in the same way shown in Figure 11 to construct a 32-bit comparator, the only difference being operands a and b are dual-rail encoded and eval is renamed req. Due to the one-hot encoding, the done signal for this circuit is derived by ORing the greater, equal and less outputs.

Figure 12: The single-rail 1-bit comparator implemented in the target library.

6. Results

The designs were synthesized using SYNOPSYS DESIGN COMPILER for \( V_{DD} = 0.25 \text{V} \). Both single-rail and dual-rail designs use the full diffusion cell library. 100 000 randomized 32-bit integer operands were used as input in a digital simulation environment on the post-synthesis netlist—no layout was performed. The computation times for single operands were measured from the dual-rail designs using the done signal and sorted into bins giving the normalized probability density functions.

To obtain a range of computation time and energy results, the designs were simulated in an analogue environment from \( V_{DD} = 600 \text{mV} \) down to \( V_{DD} = V_{DD,\min} \) (the circuit’s minimum operating voltage). The voltage range was chosen to show both sub- and super-threshold behaviors of the circuits. The temperature range was chosen as an extended commercial range of \(-10^\circ\text{C} \) to \(+85^\circ\text{C} \) which covers the harsh operating conditions expected of IoT nodes. The lower limit was enforced since lower temperatures severely degraded logic levels causing the circuits to fail (refer to the discussion in Section 2). Operating corners available for analogue simulation were TT, FF and SS. The operating conditions were identified as follows:

- Typical: TT, \( 27^\circ\text{C} \).
- Best: FF, \( 85^\circ\text{C} \).
- Worst: SS, \( -10^\circ\text{C} \).

The mean operands were chosen from the binning process and used as input to the analogue simulations. For the single-rail designs, the time taken for the circuit to compute at \( V_{DD,\min} \) for the worst conditions was used as the clock period throughout testing. A 10% margin was applied to account for on-chip variation which is not accounted for by the process corners.
Figure 14 shows the average time for a single computation as a function of \( V_{DD} \) for the 32-bit adder. The 32-bit comparator follows the same pattern. The return-to-spacer time is not included in this figure, since for our applications we assume idle time is used for this, and we are more interested in the computation latency. The single-rail design is shown as if implemented with a three-step dynamic voltage and frequency scaling (DVFS) strategy. The hatched areas illustrate time saved by the self-timed implementation.

Figure 15: Probability distribution for the self-timed adder design.

The remainder of the section discusses the delay distributions and energy results in detail.

6.1. 32-bit Adder Results

From Figure 15 we see the adder exhibits a log-normal distribution, since the average number of full adders the carry must ripple through is \( \log n \) for \( n \)-bit operands [13]. A higher concentration of operands fall towards a faster computation time, illustrating the advantages of early propagation in the adder circuit.

Figures 16 and 17 show the energy per computation as a function of \( V_{DD} \) for the single-rail (SR) and dual-rail (DR) implementations respectively. These figures include the return-to-spacer energy for the self-timed design. For the dual-rail design under typical and best conditions, the MEP lies in the range 150 mV to 200 mV. The energy in the single-rail design is much higher due to leakage, since the clock period is much longer than required. Under worst conditions, the circuit does not function below 250 mV. These are the only conditions for which the single-rail design overtakes the dual-rail in terms of energy. Figure 18 shows this is around 270 mV. Since the single-rail period is optimized for these conditions and has a lesser area, a lesser amount of energy is consumed per operation.

Figure 16: Energy per computation of the dual-rail (DR) adder.

6.2. 32-bit Comparator Results

The fastest propagation path occurs when the MSBs differ. In this case, a decision can be made on the 31st bits when they are either \{0,1\} or \{1,0\}. The remaining bits account for \( 2^{32-1} = 2^{31} \) combinations. Since there are two MSB combinations, this leads to \( 2^{33} \) input combinations which are decided on the MSBs—half of the total input operand
space which is $2^{64}$. Following on, a quarter of the input operand space can be decided on the 30th bits. This decision carries a delay of two comparators. This reasoning leads to the negative exponential probability distribution seen in Figure 19.

Figures 20 and 21 show the energy per operation for the 32-bit comparator designs. In the dual-rail (DR) design, the circuit operating under worst conditions outperforms the typical conditions at the upper part of the voltage range. This can be explained by the reduced leakage power due to the low temperature ($-10\,^\circ\text{C}$). As the worst-case computation time becomes exponentially slower however, the energy grows greater. In contrast to the dual-rail design, the energy per computation of the single-rail (SR) design varies over several orders of magnitude throughout the operating conditions.

The dynamic energy saved by gating the inputs to the self-timed design leads to improved energy per operation across all operating conditions compared to the single-rail design, including the worst conditions shown in Figure 22.

7. Conclusions

In this work we have described a design flow for datapath circuits in applications where average-case performance and energy per computation are the most important factors. We have designed a 32-bit ripple-carry adder and a 32-bit comparator using this flow and shown them to outperform single-rail counterparts in these metrics.
Table 3 summarizes the designs under typical conditions. Both self-timed dual-rail adder and comparator designs save 80% to 95% energy per computation and computation time. The area penalty for the dual-rail adder is double that of the single-rail design due to the duplicated logic and completion detection. In contrast, the dual-rail comparator uses less area than the single-rail design. This is achieved by retaining the one-hot encoding at the output. Additionally, the dual-rail inputs allow some inverters to be saved. The leakage power of the designs scales with the area as could be expected. In order to assess the full energy advantage of the dual-rail over single-rail designs, we should take into account the energy leaked during idle periods. For the dual-rail designs, there is some maximum time between new input operands which allows the design to use less energy (including idle periods) than the single-rail equivalent. We find this by dividing the difference in computation energy by the difference in leakage power. For the adder designs, this is 855 μs—equivalent to an input frequency of 1.2 kHz. In the case of the comparator, the total energy consumed will always be less than the single-rail design since both computation energy and leakage power are lowered.

Table 3: Comparison of single-rail (SR) and self-timed dual-rail (DR) designs under typical conditions with \( V_{DD} = 0.25 \text{ V} \).

<table>
<thead>
<tr>
<th>Design</th>
<th>Average per Computation Energy (fJ)</th>
<th>Area (µm²)</th>
<th>Leakage Power (pW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit SR</td>
<td>436</td>
<td>15.8</td>
<td>592</td>
</tr>
<tr>
<td>Adder DR</td>
<td>61.0 (-81%)</td>
<td>3.01</td>
<td>1350</td>
</tr>
<tr>
<td>32-bit DR</td>
<td>1479 (-94%)</td>
<td>14.4</td>
<td>799</td>
</tr>
<tr>
<td>Comparator DR</td>
<td>88.4 (-79%)</td>
<td>3.00</td>
<td>758</td>
</tr>
</tbody>
</table>

Acknowledgment

This work was supported by EPSRC and ARM.

References


