On Dual-Rail Control Logic for Enhanced Circuit Robustness

Andrey Mokhov[†], Victor Khomenko[†], Danil Sokolov[‡], Alex Yakovlev[‡]

[†]School of Computing Science, Newcastle University, UK
[‡]School of Electrical, Electronic and Computer Engineering, Newcastle University, UK

***Abstract*—Ultra low-power design and energy harvesting applications require digital systems to operate under extremely low voltages approaching the point of balance between dynamic and static power consumption which is attained in the sub-threshold operation mode. Delay variations are extremely large in this mode, which calls for the use of asynchronous circuits that are speed-independent or quasi-delay-insensitive. However, even these classes of asynchronous logic become vulnerable because certain timing assumptions commonly accepted under normal operating conditions are no longer valid. In particular, the delay of inverters, often used as the so-called input 'bubbles', can no longer be neglected and they have to be either removed or properly acknowledged to ensure speed-independence.**

This paper presents an automated approach to synthesis of robust controllers for sub-threshold digital systems based on dual-rail implementation of control logic which eliminates inverters completely. This and other important properties are analysed and compared to the standard single-rail solutions. Dual-rail controllers are shown not to have significant overheads in terms of area and power consumption and are even faster in some cases due to the elimination of inverters from critical paths. The presented automated synthesis techniques are very efficient and can be applied to very large controllers as demonstrated in benchmarks.

I. Introduction

Recent research (e.g., [7][17][21][37]) reveals that for the majority of logic and static memory blocks the optimal energy-per-operation voltage lies near or below the threshold voltage of a MOSFET device, where the point of balance between dynamic and static power consumption is found. This mode is commonly known as a sub-threshold mode. A comprehensive analysis, using the EKV model, of the sub-threshold operation of static logic can be found in [36]. The decision at which Vdd level the circuit should operate to meet its optimum in terms of energy efficiency and guarantee the acceptable level of operational robustness requires considering process and environmental variability. Notably, delay variations are extremely large in the sub-threshold mode. This calls for the use of asynchronous circuits that are speed-independent (SI) or quasi-delay-insensitive (QDI) [23][25]. These classes of asynchronous logic operate on the principles of causality and completion detection rather than matched delay and fundamental mode, which makes them inherently robust to variations in the delays of their gates. Additionally, the performance of such circuits is determined by actual, rather than worst case latency. Recent studies in [6] and [2] show the high potential of asynchronous SI and QDI circuits to build energy-efficient circuits. Moreover, for power harvesting applications, where Vdd can be unstable and varying, this is even more the case (cf. [4] and [9]). The design of SI and QDI circuits for deep-submicron can be performed using existing CAD support, provided by tools such as Haste and Balsa [30] or desynchronization methods [35], which use the principle of single-rail logic and bundled-delay for the data-path, but use handshake QDI circuits for control.

Control logic usually determines the robustness of the overall system to variations and transient errors because it forms its operational kernel. While errors in the data-path can be tolerated using conventional (e.g. error-correction codes) approaches, any error in control logic (e.g. unspecified transition or spurious pulse on a request or acknowledgement line) can be fatal for the entire system. It has been demonstrated that power reduction through voltage scaling increases the soft error rate (SER) exponentially [8]. It is therefore important to support design of robust asynchronous controllers for sub-threshold mode with tools for their efficient synthesis from behavioural specifications.

A. Control circuit synthesis techniques

There are two sufficiently mature control synthesis methodologies: one is based on Petri nets [11] and produces SI/QDI controllers, while the other is based on burst-mode finite state machines [27] and produces controllers working under the *fundamental mode* of interaction with environment. Since the paper focuses on robust controllers working under extremely large variations we follow the Petri net synthesis flow to have as few assumptions on environment as possible. The flow accepts event-based specifications in terms of interpreted Petri nets, called Signal Transition Graphs (STGs), and converts them into logic equations for complex gates in the implementation circuit. By construction, this circuit is an SI circuit with respect to the delays associated with the outputs of the complex gates. Designers usually prefer using complex gates, which can be implemented as the so-called generalised (or asymmetric) C-elements [30]. This way many control circuits published to date, such as controllers for pipeline stages [30], NoC routers [13], as well as controllers for latches for sub-threshold logic [1], have been constructed. However, in the sub-threshold mode, even these SI/QDI implementations may become vulnerable to the effects of variability or susceptibility to noise (cross-talk) and transient faults. For example, the excessive delay variations under low Vdd make certain timing assumptions that are commonly accepted under normal Vdd no longer valid. In particular, most of the complex gates in the implementations produced by SI/QDI synthesis contain the so-called 'bubbles'. These are input inverters, whose delay is typically neglected, or at least regarded to be much smaller than the delay of a certain ('racing') path passing through other logic gates. In sub-threshold operation these bubbles can no longer be ignored. Starodoubtsev has developed a method of behavioural refinement for the synthesis of the SI/QDI class of circuits from STGs [31], which produces circuits in simple monotonic gates (free from bubbles). Unfortunately, this technique tackles both problems, obtaining monotonic functions of the gates and decomposition of gates into simple gates, at the same time, which makes it quite complex in practice (see also a discussion of normalcy in Section III). As a result it has not been automated to date and leads to circuits with long acknowledgement paths, thereby increasing their latency.

To facilitate the availability of tools for designing efficient and robust sub-threshold circuits it would be beneficial to maximally use the existing logic synthesis frameworks, such as that of PETRIFY, to generate monotonic SI/QDI circuits. It would be desirable to obtain a relatively simple synthesis flow, similar in its spirit to the one used in NCL-D/NCL-X [16][20] for data-path logic. Fortunately, the way to such an approach originates in the idea of a 'perfect implementation' for a semi-modular circuit from [14]. The theory for this approach is based on deriving separate Boolean covers for the set and reset functions for each signal x in the circuit specification. Those can be obtained from the excitation and quiescent regions [11] for x. This is equivalent to finding next state functions for two separate rails x and $\widehat{x}$ of each binary signal x. The original theory of [14] was developed for the so-called closed circuits, without input choice, and hence is not directly applicable to the types of controllers designed in practice today. Later this approach was extended to work for open circuits in the technique for monotonic cover implementation [19], however the focus of that work was not specifically on finding dual-rail control implementations; besides it was not fully automated to work with specifications of complex controllers.

B. Contribution and organisation of the paper

In this paper, we address the problem of robust control logic synthesis with a specific requirement of finding an SI/QDI implementation which will meet the needs of sub-threshold mode of operation and will be obtained automatically from a specification which may be as complex as hundreds of logic signals. We pursue the approach of dual-rail synthesis, using the theoretical basis of [14] as well as monotonic cover techniques of [19] to support the decomposition of complex dual-rail gates into separate logic for set and reset functions and standard RS latches. This combination offers a range of dual-rail architectures, called here gRS and stdRS, which can be implemented using static logic libraries as well as custom transistor-level circuits. We implement the synthesis algorithms using efficient methods of coping with huge state spaces that are based on Petri net unfoldings (PUNF/MPSAT tool chain [18]). One of the important benefits of these dual-rail implementations of complex controllers is that the control logic can be distributed, i.e. the appropriate gates producing control signals can be placed next to the data-path sections. This distribution may lead to the need to use long wires, but this risk of violating delay-insensitivity due to wire delays will be mitigated by the inherent robustness of dual-rail implementations. Special dual-rail repeaters will be used where necessary, against transient faults and violations of signal integrity. Besides monotonicity (no bubbles), other benefits of these implementations compared to single-rail control include: less gate complexity, fewer isochronic forks; easier testability (RS-latches instead of C-elements). The paper illustrates these advantages in a relatively simple case study and a set of benchmarks whose complexity scales up to large quantities of control gates.

Our main contributions in this paper can be summarised as follows.

- An in-depth study of dual-rail control methodology is carried out:
  - gate- and transistor-level implementations of standard dual-rail control structures are presented;
  - the implementations are analysed in terms of robustness, area and latency;
  - we demonstrate vulnerability of single-rail controllers due to input inverters;
  - a number of single-rail and dual-rail controllers are compared in terms of area, power consumption, as well as required post-synthesis effort in logic decomposition and isochronic fork balancing.
- A scalable synthesis method for dual-rail control, which is based on Petri net unfoldings, is presented and evaluated.

The paper is organised as follows. Section II presents the main foundation behind dual-rail control logic, including its motivation, advantages and possible penalties. Section III describes the basic synthesis process. Section IV presents a detailed case study and Section V covers the experiments on a set of standard asynchronous controllers.

II. DUAL-RAIL CONTROL

Dual-rail code uses a pair of physical wires, x and $\hat{x}$, per logical signal x. There are two valid signal combinations, 01 and 10, which encode values 0 and 1, respectively. This code is employed to represent data in self-timed circuits [12], where a specific protocol of switching helps to avoid hazards. The protocol allows only the monotonic switching from all-zeroes 00, which is a non-code word, to a *code word* and back to all-zeroes as shown in Figure 1. The all-zeroes state, which is called *spacer*, indicates the absence of data and separates one code word from another.

Traditionally the dual-rail switching protocol is used in asynchronous data-path logic due to its robustness and simplicity of circuit construction, as in [20] where the standard

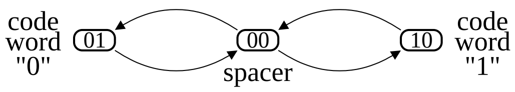


Figure 1: Dual-rail signalling protocol

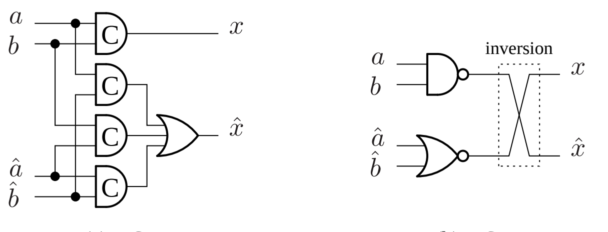


Figure 2: Dual-rail data-path on AND-gate example

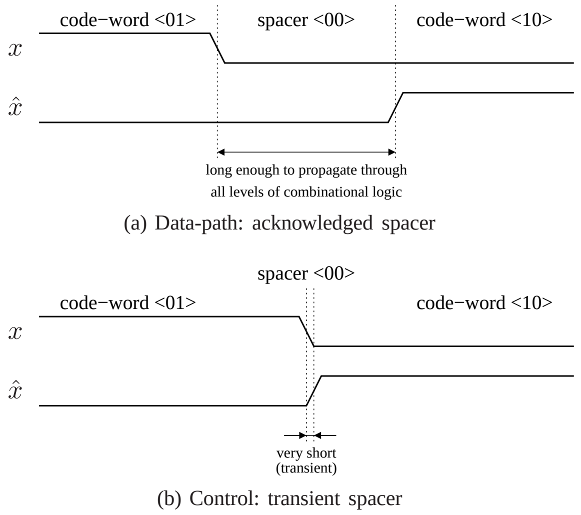

code−word <01>
spacer <00>
code−word <10>
x
$\hat{x}$
long enough to propagate through all levels of combinational logic
(a) Data-path: acknowledged spacer
very short (transient)
(b) Control: transient spacer

Figure 3: Spacer state: dual-rail data-path vs dual-rail control

RTL-based design flow is extended by converting single-rail circuits into dual-rail. Within this approach, called Null-Convention Logic [15] one can follow either of two main implementation styles: NCL-D, which integrates completion detection into the dual-rail logic or NCL-X, which relies on a separate completion detection circuitry and/or on some timing assumptions. The former is more conservative with respect to delay sensitivity while the latter is more area and speed efficient. For example, an AND gate implemented in NCL-D and NCL-X styles is shown in Figure 2.

The inherent property of the dual-rail circuits is that the operation of Boolean negation corresponds to the rail swapping, which allows to achieve race-free operation under any single transition. Another feature of the dual-rail logic is its balanced power consumption which facilitates circuit resistance to the power analysis attacks. Security aspects of the dual-rail circuits have been further improved by introducing a special *alternating spacer* protocol [29] – it guaranties all gates of the circuit switch exactly once in each computation cycle, thus making circuit power consumption invariant to the processed data.

A. Cost of dual-rail control

The major drawback of the *dual-rail data-path* logic is (at least) twofold increase in area and power consumption compared to the single-rail implementation. Since data-path circuits constitute a dominant part of the whole system, this restricts the adoption of dual-rail methodology to the fairly specific domain of security applications and to building truly self-timed systems. However, area and power penalties should not impose significant overhead on the relatively small control circuits being implemented in dual-rail style. A far more important issue for control is latency: switching through the all-zeroes state in dual-rail data-path doubles the computation cycle unless extra logic is inserted to concurrently precharge the combinational logic to spacer [34], see illustration in Figure 3(a). By contrast, *dual-rail control* does not require acknowledgement of the spacer state, so the latter can be *transient* as shown in Figure 3(b), thereby achieving latency of a single-rail implementation.

The asynchronous control is usually dominated by C-elements which perform a generic function of signal synchronisation (a C-element output goes high when all its inputs are high, and goes low when all the inputs are low). While enjoying all the benefits of dual-rail switching protocol, a dual-rail C-element can actually be made comparable in size, power consumption and latency to the single-rail one. For example, a typical single-rail complex-gate C-element shown in Figure 4(a) (as synthesised by MPSAT or PETRIFY tools) requires 12 transistors in static CMOS implementation. An equivalent dual-rail circuit in Figure 4(b) has the same transistor count and is built out of simpler gates, which are more likely to be present in the technology library. Availability of the complex gates is particularly important to avoid hazard-free decomposition of the C-element set/reset functions [19], which is not trivial and often results in significant area, power and latency penalties.

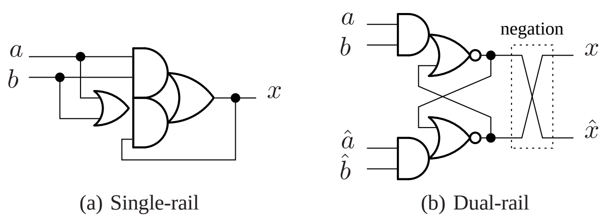
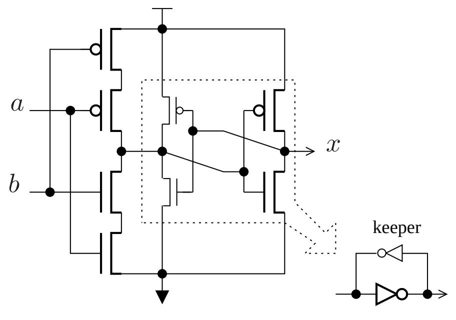
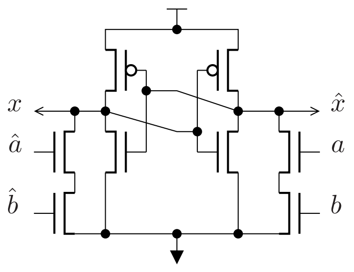


Figure 4: Complex gate implementation of C-element

For custom design the dual-rail C-element is also similar in size to the standard transistor-level single-rail implementation (8 transistors), as illustrated in Figure 5. The state of the C-element is held in a *keeper* – a logic level holding circuit which consists of two inverters connected back to back. Note that for single-rail implementation the feedback inverter has to be weak (made out of small transistors), so that the pull-up and pull-down transistor stacks are able to enforce the keeper state. When $x = 1$ and $a \neq b$ the keeper state is supported

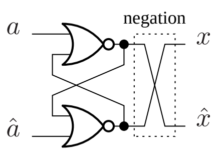


(a) Single-rail

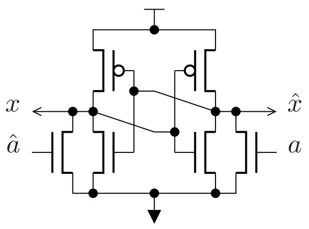


(b) Dual-rail

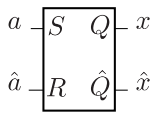
Figure 5: Custom implementation of C-element



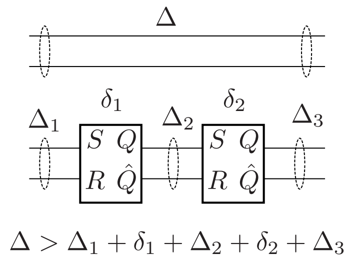
(a) Gate-level



(b) Transistor-level



(c) RS-latch



(d) Repeater insertion

Figure 6: Dual-rail repeaters

by a weak PMOS transistor only and therefore is vulnerable to a *single event upset* (SEU), such as a voltage pulse caused by charge-induced particles or electromagnetic radiation. The corresponding dual-rail solution is based on cross-coupled inverters and pull-down NMOS networks for both set and reset. Under the same conditions this implementation is more robust to SEUs because its state holding element is symmetric and both inverters are strong, hence the critical charge from the particle strike is required to be higher to pull down the middle point sufficiently low [33]. Note that a dual-rail C-element is most exposed to SEUs when neither set nor reset function is evaluated to 1 and its state can be toggled by a particle strike. Therefore, to improve circuit robustness one can explore a tradeoff between the complexity of the set/reset functions and minimisation of the dangerous time interval when both functions evaluate to 0.

The above C-element implementations combine the set/reset functions and the state holding latch. Often it may be necessary to separate the set and/or reset logic from the latch, e.g. to reduce the implementation complexity or to map the latch into a standard library RS-latch. Such decomposition must preserve the hazard-free operation and is achieved by building the set/reset functions satisfying the condition of monotonic cover, as described in Section III. The use of standard RS-latches is advantageous for compatibility with the standard design practice as it helps to avoid combinational loops which often cause problems for EDA tools. Circuit testability can also be addressed by extending the RS-latches with a synchronous scan interface and applying standard testing techniques for *level-sensitive scan design* (LSSD) [28]. With this scan structure the circuit operates asynchronously in mission mode, while it is synchronised with the test clock signals when in test mode.

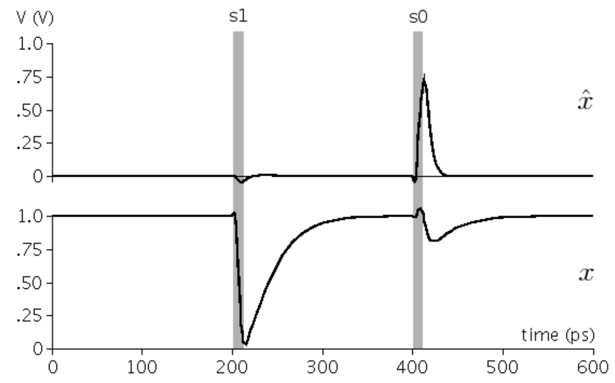
B. Robust dual-rail repeaters

A C-element with trivial set/reset functions, as shown in Figure 6, is called a *repeater* and is employed to maintain signal integrity in long wires. Similarly to single-rail buffers, the dual-rail repeaters can be used to reduce the time delay associated with long wires by inserting them along the switching lines. This technique, known as *repeater insertion*, is well studied [3][26] and can be directly applied for dual-rail control logic as shown in Figure 6(d).

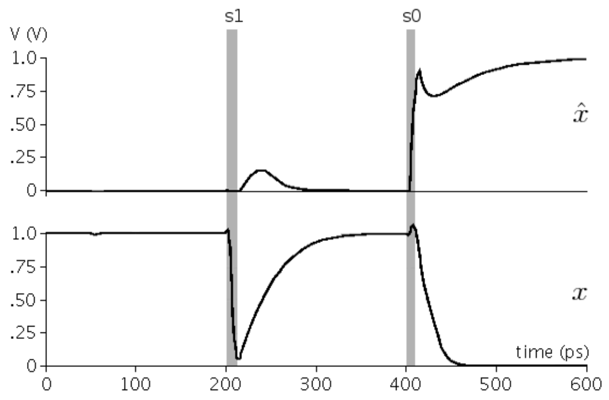
Dual-rail repeaters are very robust to SEUs because their inputs go through the dangerous spacer state only for a short period of time and switch back into a stable code word state immediately – see Figure 3(b). If a single wire distortion occurs while in a code word state the repeater recovers from the error – the information redundancy of dual-rail code words plays its role; a spacer state, on the other hand, does not provide sufficient information for recovery. This is demonstrated by simulation[1] of two SEUs $s0$ and $s1$ on output wires $\hat{x}$ and x, respectively, which is shown in Figure 7(a). SEUs were modelled as 5ps pulses and the full recovery took around 100ps. In the unlikely event of a SEU occurring during the spacer state, the repeater still recovers from $s1$ but cannot recover from $s0$ as illustrated in Figure 7(b).

The notorious penalties of dual-rail data-path, power consumption and cycle time, are irrelevant to the dual-rail control logic. In dual-rail control the switching activity doubles as all the wire pairs go through a spacer state (similar to the dual-rail data-path). However, the load of the wires is roughly halved compared to the corresponding single-rail circuit and therefore the power consumption increase is insignificant (not twofold as in data-path logic). Also the spacer state is transient in dual-

[1]All simulations in this paper have been performed in SPECTRE using Faraday standard gate library based on UMC 90nm technology.



(a) In code word state



(b) In spacer state

Figure 7: Recovery of dual-rail repeater from SEUs

rail control and does not require a dedicated precharge stage as in NCL, therefore the cycle time remains the same as in single-rail control.

To summarise, the penalties associated with the dual-rail data-path circuits do not show in the control logic. In particular, the key building block of asynchronous control, the dual-rail C-element, is similar in size and speed to the standard single-rail implementation, while its operation at sub-threshold voltage is more robust to noise and charge-induced particles. A synthesis method and hazard-free decomposition of set/reset functions is presented in Section III and circuit complexity, size and power consumption is analysed on a set of large benchmarks in Section V.

III. Synthesis

We want to build speed-independent (SI) circuits, assuming that their behaviour is specified using *state graphs* (SGs), which are finite state machines with annotations, cf. Figure 8 (SGs can be constructed from higher-level specifications, such as STGs [11] or HDLs). We further assume that all the states in the SG are reachable from the initial state. With each state s of the SG we associate a vector of binaries $Code(s)$

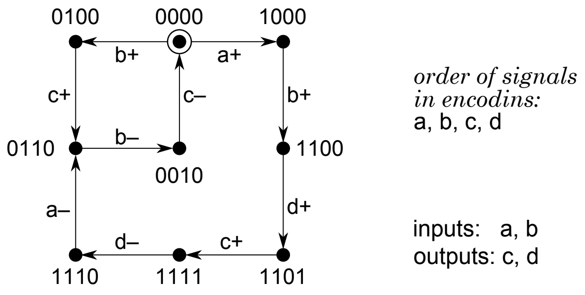


Figure 8: A circuit specification from [5]

representing the values of all the circuit signals at this state; moreover, $Code_z(s)$ will denote the component of $Code(s)$ corresponding to signal z. Each arc of the SG is labelled by z^+ or z^-, where z is a signal. We assume that the specification is *consistent*, i.e. if an arc $(s,\ s')$ is labelled by z^+ (resp. z^-) then $Code_z(s) = 0$ (resp. 1), $Code_z(s') = 1$ (resp. 0), and $Code_{z'}(s) = Code_{z'}(s')$ for all $z' \neq z$. Furthermore, we assume that the SG is *deterministic*, i.e. no two arcs with the same source are labelled by the same signal.

The circuit signals are partitioned into *inputs* and *outputs* (the latter also include internal signals). Input signals are assumed to be generated by the environment, while output signals are produced by the circuit. We assume that the SG is *output-persistent*, i.e. an output cannot be disabled by firing any other transition (i.e. choices are allowed only between inputs).[2]

For each output signal z, the Boolean functions Out_{z^+}, Out_{z^-} and Out_z are defined as follows: $Out_{z^+/z^-/z}(s)$ is 1 if state s enables $z^+/z^-/z^{\pm}$, and 0 otherwise. The Boolean *next-state function* Nxt_z is then defined as $Nxt_z(s) \overset{\text{df}}{=} Code_z(s) \oplus Out_z(s)$, where $\oplus$ is the 'exclusive or' operation. Similarly, the *set* and *reset functions* Set_z and $Reset_z$ are defined as follows:

$$Set_z/Reset_z(s) \overset{\text{df}}{=} \begin{cases} 1 & \text{if } Out_{z^+/z^-}(s) = 1 \\ 0 & \text{if } Nxt_z(s) = 0/1 \\ - & \text{otherwise,} \end{cases}$$

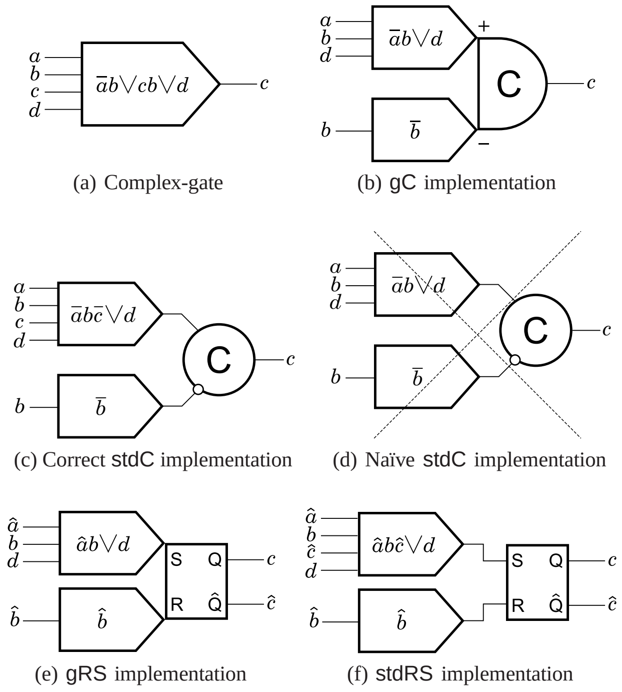
where '$-$' denotes the 'don't care' value (i.e. the value of the function can be chosen arbitrarily, with the view of simplifying the resulting implementation).

Various architectures are used to implement speed-independent circuits; the following are probably the most well-known [11][19] (see Figure 9):

Complex-gate (CG) implementation: Every output is implemented as a single (possibly very complicated) atomic gate [10].

Generalised-C (gC) implementation: Every output is implemented using a pseudo-static latch called *generalised C element (gC element)* which is assumed to be atomic [23]. A

[2]In some applications a choice between outputs is allowed, which can be implemented by a special element called *arbiter* that internally uses some analog circuitry to handle the arising metastability; however, arbiters can be 'factored out' into the environment, so that the remaining part of the specification is output-persistent.



(a) Complex-gate (b) gC implementation

(c) Correct stdC implementation (d) Naïve stdC implementation

(e) gRS implementation (f) stdRS implementation

Figure 9: Implementations of signal c of the SG in Figure 8

gC implementation is specified by the set and reset functions for each output, which are implemented by pull-up and pull-down transistor networks. In the states where both set and reset functions evaluate to 0, a keeper element is used to ensure that the output keeps its current value (it is an error if in some reachable state both functions evaluate to 1 — this can lead to a short circuit). This is similar to the implementation of single-rail C-element from Figure 5(a).

Standard-C (stdC) implementation: Every output is implemented using a C-latch controlled by set and reset signals, which we assume are implemented as complex-gates [5]. This architecture is superficially similar to the previous one, but one should bear in mind that a gC element is assumed to be atomic, while in the stdC implementation the gates controlling a C-latch have delays. Hence a naïve transformation of a gC implementation into an stdC one can result in a hazardous circuit (see below).

Generalised-RS (gRS) and *standard-RS (stdRS) implementations*: These two architectures correspond to gC and stdC ones, in particular the same set and reset functions are used, but an RS-latch is used as the state holding element [19]. Furthermore, the dual-rail representation of each signal is used, and so there are no inverters anywhere in the circuit (except those hidden inside the latch).

For the circuit to be implementable in the CG architecture, the value of Nxt_z must be uniquely determined by the encoding of each reachable state, i.e. it should be a function of $Code(s)$ rather than s: $Nxt_z(s) = F_z(Code(s))$ for some Boolean function F_z, which is eventually implemented as a complex-gate. Similarly, for gC architecture the values of Set_z and $Reset_z$ must be functions of $Code(s)$ rather than s: $Set_z/Reset_z(s) = S_z/R_z(Code(s))$ for some Boolean functions S_z and R_z, which will determine the corresponding gC element. In case of stdC architecture, S_z and R_z must in addition satisfy the *Monotonic Cover condition (MCC)* [5][11], in order to provide a hazard-free circuit. MCC states that *a cover must be entered only via the states enabling the output* z. As MCC reduces the flexibility in choosing S_z and R_z, they can be more complicated than those for gC architecture, cf. Figure 9(b,c).

To illustrate the importance of MCC, consider the implementation shown in Figure 9(d), which does not satisfy it, since the state 0110 (which is covered by the set function $\overline{a}b \vee d$ and does not enable c) can be reached from the state 1110 (which is not covered by this set function and does not enable c). Consider the sequence of states $1111 \xrightarrow{d^-} 1110 \xrightarrow{a^-} 0110 \xrightarrow{b^-} 0010$. The gate computing the set function is high at 1111. Firing of d^- drives its output low, but before it reaches 0, a^- can fire, driving its output high; similarly, before it reaches 1, b^- can fire, driving it low. Hence, this gate can exhibit runt non-digital pulses causing the circuit to malfunction.

It turns out that the notion of implementability of a signal is invariant across the CG/gC/stdC/gRS/stdRS architectures,[3] i.e. if a signal is implementable in one of them, it is implementable in the other architectures as well; moreover, given the mentioned above assumptions on the SG, the implementability of the specification in either architecture is equivalent to the *Complete State Coding (CSC)* property, which states that for every circuit output z, no two states s and s' of the SG satisfy $Code(s) = Code(s')$ and $Out_z(s) \neq Out_z(s')$ [11]. In what follows, we assume that the SG satisfies the CSC property.

Normalcy [32] is a property of SGs, which is a necessary condition for their implementability in the CG architecture using gates without input inversions, i.e. whose characteristic function is either monotonic or a negation of a monotonic function. Normalcy violations can be detected by model checking, and sometimes resolved by insertion of new signals [22]. However, the latter is not always possible, as the sought signal insertions might not exist or cause further normalcy violations, and even if this is possible, the circuit becomes more complicated due to the additional logic needed to implement the new signals.

There are a number of tools that support asynchronous synthesis, e.g. PETRIFY and MPSAT. They both support complex-gate synthesis and derivation of set and reset functions, including monotonic covers, and so can be used to automate any of the described asynchronous architectures. The main problem in synthesis is the *state space explosion:* a relatively small specification can (and often does) yield a huge state graph;

[3]The result is well-known for the former three architectures, and gRS and stdRS implementations use the same set and reset functions as gC and stdC, respectively.

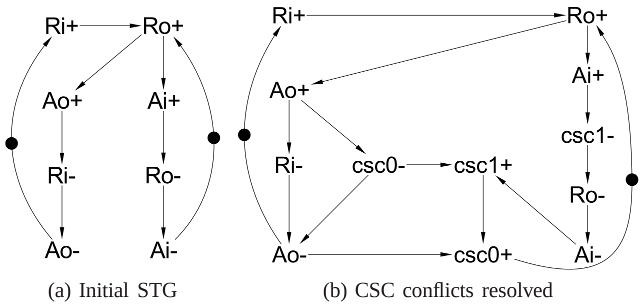


Figure 10: STG specification of the example controller

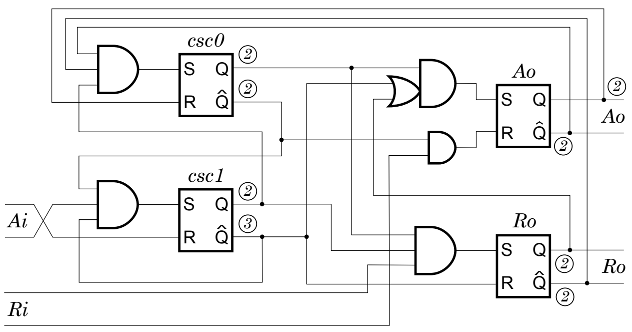


Figure 13: stdRS implementation

this puts a practical limit on the size of circuits that can be synthesised. To alleviate this problem, PETRIFY uses BDDs, and usually can synthesise circuits with up to 20-30 signals. MPSAT avoids generating the state graph altogether, and works on STG unfoldings instead; it usually can synthesise circuits with up to 150-200 signals.

IV. CASE STUDY

Figure 10(a) shows an STG specification of a typical asynchronous pipeline controller from [11] which synchronises two handshakes $(Ri,\ Ao)$ and $(Ro,\ Ai)$ managing adjacent pipeline stages. Request Ri^+ informs the controller about availability of data in the current pipeline stage. In response, the controller immediately prompts the next stage to latch the data (Ro^+) and sends an acknowledgement back to the current stage (Ao^+). Then the handshakes are reset concurrently ($Ri^- \rightarrow Ao^-$ and $Ai^+ \rightarrow Ro^- \rightarrow Ai^-$) for the next data transfer round. In order to satisfy the CSC property it is necessary to introduce two internal signals *csc0* and *csc1* as shown in Figure 10(b); this is done automatically – see details in [11].

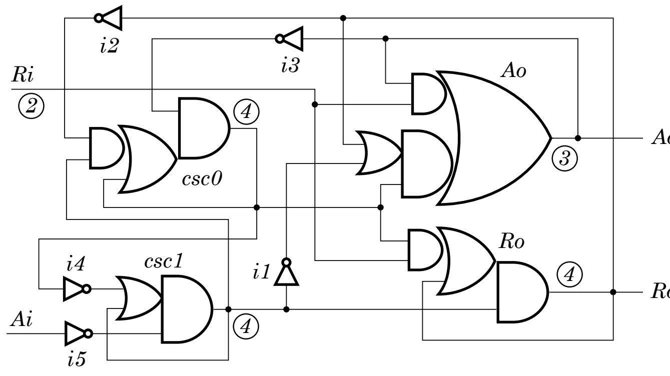


Figure 11: Complex-gates implementation

Now it is possible to use PETRIFY or MPSAT synthesis tool to generate a CG implementation of the STG. The obtained circuit is presented in Figure 11; note that input bubbles of the derived complex-gates are explicitly shown as inverters $i1 - i5$. In a normal operating mode it is commonly assumed that these inverters are faster than any other gate. However, if the controller operates under a sub-threshold voltage the increased delay variations can easily violate this assumption, thus breaking the speed-independence of the circuit. Consider the following sequence of events:

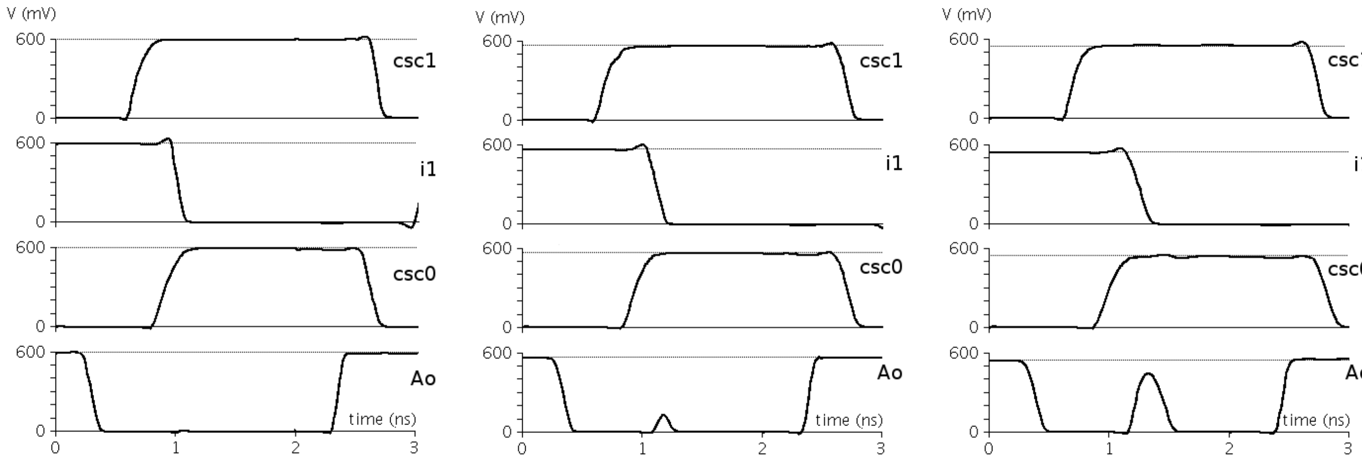
$$\begin{gathered} Ri^+,\ Ro^+,\ Ao^+,\ i2^-,\ i3^-,\ csc0^- \\ i4^+,\ Ai^+,\ i5^-,\ csc1^-,\ i1^+,\ Ro^-,\ i2^+ \\ Ri^-,\ Ao^-,\ i3^+,\ Ai^-,\ i5^+,\ csc1^+ \end{gathered}$$

At this point there is a race between events $i1^-$ and $csc0^+$: if inverter $i1$ happens to be slower than gate $csc0$, there will be an unspecified enabling of Ao^+ which creates a hazard on wire Ao and breaks the environment protocol. At the system level this can easily lead to a global deadlock. Figure 12 shows simulation of the circuit behaviour under different supply voltages. At nominal 1V power supply we get no sign of the hazard. This hazard-free behaviour continues all the way down to 600mV. The hazard becomes visible at 575mV and reaches incorrect voltage levels for output Ao at about 550mV. This is a perfect illustration of how quickly things can go wrong in the sub-threshold domain.

The problem can be solved by applying the dual-rail expansion to all signals, thus removing all the dangerous inverters. The stdRS implementation of the controller is shown in Figure 13; as expected, it contains no inverters. Another important advantage is that it is built of much simpler gates which are very likely to be present in most technology libraries. Large 4- and 5-input gates of the CG implementation will probably require decomposition into smaller gates, potentially introducing new sources of hazards and adding more overheads.

A. Analysis of wire forks

Note that although the number of wires in the dual-rail implementation has doubled, each wire has less load as it has become distributed over two rails. For example, signal *csc0* in the single-rail implementation has 4 gates in its fanout (this fact is denoted as ④ in Figure 11), while both signals $csc0$ and $\widehat{csc0}$ in the dual-rail controller have fanout 2, thus jointly consuming the same amount of energy but switching faster. This also decreases the degree of 'forking': instead of having to balance 4 wire delays to satisfy the isochronic fork assumption during circuit layout, one has to balance only two pairs of wires, which is considerably easier.



(a) Supply voltage = 600mV

(b) Supply voltage = 575mV

(c) Supply voltage = 550mV

Figure 12: Simulation of a hazard in single-rail implementation

Overall, in the CG implementation there are wire forks of the following degrees: 2, 3, 4, 4, 4. The fork degrees of the stdRS implementation are 2, 2, 2, 2, 2, 2, 2, 3. In order to compare the controllers in terms of the effort required for fork balancing we introduce the following measure.

Consider a fork with k branches. There are $\binom{k}{2} = \frac{k(k-1)}{2}$ pairs of wires and each pair, if unbalanced, can lead to a hazard in the controller. Therefore, the overall fork balancing effort will be proportional to $f(C) = \sum_{w \in C} \binom{k_w}{2}$, where C is a given controller and w iterates over all its wires. For the example at hand the comparison measure gives the following result:

$$f(\mathsf{CG}) = \binom{2}{2} + \binom{3}{2} + 3\binom{4}{2} = 22$$

$$f(\mathsf{stdRS}) = 7\binom{2}{2} + \binom{3}{2} = 10$$

Hence, one can conclude that the single-rail version of the controller requires roughly twice as much effort for fork balancing as the dual-rail one. In the next section we will demonstrate that this is a typical situation.

V. Experiments

Table I presents a summary of experimental results. We have taken several standard asynchronous controllers, some of which are scalable, and synthesised their implementations using Punf and Mpsat synthesis tools. Despite the large state spaces (up to 10^{13} states) the processing times were in the order of several seconds.

Each benchmark controller is described with three parameters: $|I|$, $|O|$ and $|S|$ being counts of the circuit inputs, outputs and states, respectively. Columns ‘Inv.’ report numbers of input inverters in CG and stdC single-rail implementations; dual-rail stdRS and gRS implementations are ‘bubble-free’. Area of a particular implementation is estimated as its size in terms of literals (a gate bubble is also counted as a literal because it has to be implemented separately from the gate, hence occupying additional area). Power consumption is estimated similarly, with the exception that a bubble is given smaller weight of 0.5, because an input inverter drives only a single wire and thus consumes less power than an average circuit gate. Power and area estimates are normalised over the CG implementation for easier comparison; average values across all benchmarks are given in the bottom row.

L_{max} represents complexity of the largest gate in an implementation in terms of literals (note that L_{max} for stdC and stdRS implementations are the same). Larger values of L_{max} correspond to circuits which are more difficult for technology mapping. Roughly speaking any gate containing more than 6 literals is very unlikely to be present in a technology library; some libraries are even limited to 3-literal gates only. One can see that dual-rail implementations tend to have simpler gates, therefore being easier for decomposition. Another relevant parameter L_{avg} (average gate complexity) is not shown in the table due to lack of space, but we observed that in general L_{avg} is twice larger for single-rail implementations.

Figure 14 shows a comparison of CG, gRS and stdRS implementations in terms of the fork balancing effort. One can see that single-rail controllers are consistently more expensive in this respect: on average, the effort of balancing forks in gRS (resp. stdRS) implementation is only 50% (resp. 54%) of that of the CG implementation. This also means that the average load on a single-rail wire is the double of that on a dual-rail wire. Intuitively, this is because a single-rail wire is used by both positive and negative value ‘consumers’.

Overall we can conclude that dual-rail gRS implementation has no penalty in terms of power and only 15% overhead in terms of area in comparison to CG single-rail implementation. Dual-rail is more robust though as there are no potentially dangerous inverters and wires have less load and forks. Moreover

Benchmark circuit			Single-rail implementations								Dual-rail implementations				
			CG				stdC				stdRS		gRS		
Name	|I|/|O|	|S|	Inv.	L_{max}	Power	Area	Inv.	L_{max}	Power	Area	Power	Area	L_{max}	Power	Area
LazyRing	5/7	187	10	6	38	33	15	5	223.7%	234.8%	165.8%	190.9%	5	121.1%	139.4%
Ring	11/18	16320	56	14	196	168	72	10	164.3%	170.2%	118.4%	138.1%	10	92.9%	108.3%
Dup4phCsc	12/15	171	50	13	178	153	67	10	169.7%	175.5%	123.6%	143.8%	10	98.9%	115.0%
Dup4phMtrCsc	10/16	149	49	10	165	141	64	10	181.2%	190.0%	132.7%	155.9%	10	105.5%	123.8%
DupMtrModCsc	10/17	321	65	14	186	154	68	13	160.2%	172.0%	114.5%	138.8%	12	89.2%	108.1%
CfAsymCscA	8/26	147684	57	9	194	166	85	7	189.2%	196.1%	132.0%	154.7%	7	105.2%	123.3%
CfAsymCscB	8/24	147684	64	9	205	173	88	7	177.6%	185.0%	122.9%	145.7%	7	99.5%	117.9%
CfSymCscA	8/14	6672	62	21	210	179	86	17	158.1%	161.5%	110.5%	129.6%	17	91.4%	107.3%
CfSymCscB	8/8	690	14	8	56	49	22	5	189.3%	193.9%	135.7%	155.1%	4	103.6%	118.4%
CfSymCscC	8/10	2416	38	11	114	95	46	7	159.6%	167.4%	110.5%	132.6%	7	91.2%	109.5%
CfSymCscD	4/10	414	8	5	34	30	16	5	300.0%	313.3%	223.5%	253.3%	5	158.8%	180.0%
PpWkCsc(2,3)	0/7	128	10	5	37	32	12	2	191.9%	203.1%	140.5%	162.5%	2	102.7%	118.8%
PpWkCsc(2,6)	0/13	8192	22	5	79	68	24	2	173.4%	183.8%	126.6%	147.1%	2	93.7%	108.8%
PpWkCsc(2,9)	0/19	$> 5 \cdot 10^5$	34	5	121	104	36	2	167.8%	177.9%	122.3%	142.3%	2	90.9%	105.8%
PpWkCsc(2,12)	0/25	$> 3 \cdot 10^7$	44	5	161	139	48	2	167.1%	176.3%	121.7%	141.0%	2	90.7%	105.0%
PpWkCsc(3,3)	0/10	1024	15	7	55	48	18	3	189.1%	200.0%	138.2%	160.0%	3	101.8%	117.9%
PpWkCsc(3,6)	0/19	$> 5 \cdot 10^5$	33	7	118	102	36	3	172.0%	182.3%	125.4%	145.8%	3	93.2%	108.4%
PpWkCsc(3,9)	0/28	$> 2 \cdot 10^8$	51	7	181	156	54	3	166.9%	176.8%	121.5%	141.5%	3	90.6%	105.5%
PpWkCsc(3,12)	0/37	$> 10^{11}$	66	7	241	208	72	3	166.4%	175.5%	121.2%	140.4%	3	90.5%	104.8%
PpArbCsc(2,3)	2/13	3312	22	8	82	71	30	6	186.6%	194.4%	134.1%	154.9%	6	100.0%	115.5%
PpArbCsc(2,6)	2/19	$> 2 \cdot 10^5$	34	8	124	107	42	6	176.6%	185.0%	127.4%	147.7%	6	95.2%	110.3%
PpArbCsc(2,9)	2/25	$> 10^7$	44	8	164	142	54	6	173.8%	181.7%	125.6%	145.1%	6	93.9%	108.5%
PpArbCsc(2,12)	2/31	$> 8 \cdot 10^8$	56	8	206	178	66	6	170.4%	178.7%	123.3%	142.7%	6	92.2%	106.7%
PpArbCsc(3,3)	3/19	77032	33	12	129	113	51	9	185.3%	189.8%	131.0%	150.2%	9	99.2%	113.8%
PpArbCsc(3,6)	3/28	$> 3 \cdot 10^7$	51	12	192	167	69	9	176.0%	182.3%	125.5%	144.7%	9	94.8%	109.3%
PpArbCsc(3,9)	3/37	$> 10^{10}$	66	12	252	219	87	9	173.4%	179.7%	124.2%	142.9%	9	93.7%	107.8%
PpArbCsc(3,12)	3/46	$> 10^{13}$	84	12	315	273	105	9	170.2%	177.1%	122.2%	141.0%	9	92.1%	106.2%
Average					**100%**	**100%**			**180.7%**	**189.0%**	**130.4%**	**151.4%**		**99.0%**	**115.0%**

Table I: Summary of experimental results

dual-rail controllers contain simpler gates and are easier for hazard-free technology mapping. stdRS implementation gives even simpler gates (as they are separated from RS-latches), but has larger overheads: 30% and 51% in terms of power consumption and area, respectively. Corresponding single-rail controllers (stdC) have the largest overheads (81% and 89%, respectively).

VI. Conclusions

This paper presented an automated approach to synthesis of robust controllers for sub-threshold digital systems. The approach is based on dual-rail implementation of control logic which eliminates inverters, reduces forks and wire load without introducing significant overheads in terms of area, latency and power consumption.

Future work includes optimisation of set/reset functions for robustness (to minimise the period in which both functions are zero), and further in-depth analysis of testability of RS-latch based design.

Acknowledgement

This work was supported by EPSRC grants EP/G037809/1 (VERDAD), EP/J008133/1 (TrAmS-2) and EP/I038306/1 (GAELS).

References

[1] O.C. Akgun, J. Rodrigues, and J. Sparsoe and. Minimum-energy sub-threshold self-timed circuits: Design methodology and a case study. In *Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2010.

[2] Omer Can Akgun and Yusuf Leblebici. Energy Efficiency Comparison of Asynchronous and Synchronous Circuits Operating in the Sub-Threshold Regime. *Journal of Low Power Electronics*, 3:320–336, 2008.

[3] H. Bakoglu and James Meindl. Optimal interconnection circuits for VLSI. *IEEE Transactions on Electron Devices*, 32(5):903–909, 1985.

[4] A. Baz, D. Shang, F. Xia, and A. Yakovlev. Self-Timed SRAM for energy harvesting applications. In *PATMOS*, 2010.

[5] P.A. Beerel, C.J. Myers, and T.H.-Y. Meng. Covering Conditions and Algorithms for the Synthesis of Speed-Independent Circuits. *IEEE Trans. on CAD*, 1998.

[6] Peter A. Beerel and Marly Roncken. Low power and energy efficient asynchronous design. *J. Low Power Electronics*, 3(3):234–253, 2007.

[7] David Bol, Dina Kamel, Denis Flandre, and Jean-Didier Legat. Nanometer MOSFET effects on the minimum-energy point of 45nm subthreshold logic. In *ISLPED '09: Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 3–8, 2009.

[8] Vikas Chandra and Robert C. Aitken. Impact of voltage scaling on nanoscale SRAM reliability. In *DATE '09: Proceedings of the conference on Design, automation and test in Europe*, pages 387–392, 2009.

[9] J.F. Christmann, E. Beigne and, C. Condemine, N. Leblond, P. Vivet, G. Waltisperger, and J. Willemin. Bringing robustness and power efficiency to autonomous energy harvesting microsystems. In *Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2010.

[10] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, Lab. for Comp. Sci., MIT, 1987.

[11] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic synthesis of asynchronous controllers and interfaces*. Advanced Microelectronics. Springer-Verlag, 2002.

[12] I. David, R. Ginosar, and M. Yoeli. An efficient implementation of boolean functions as self-timed circuits. *IEEE Transactions on Computers*, 41:2–11, 1992.

[13] Rostislav (Reuven) Dobkin, Ran Ginosar, and Avinoam Kolodny. QNoC asynchronous router. *Integr. VLSI J.*, 42(2):103–115, 2009.

[14] Victor I. Varshavsky (Editor). *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, 1990.

[15] K. Fant and S.A. Brandt. Null conventional logic: A complete and

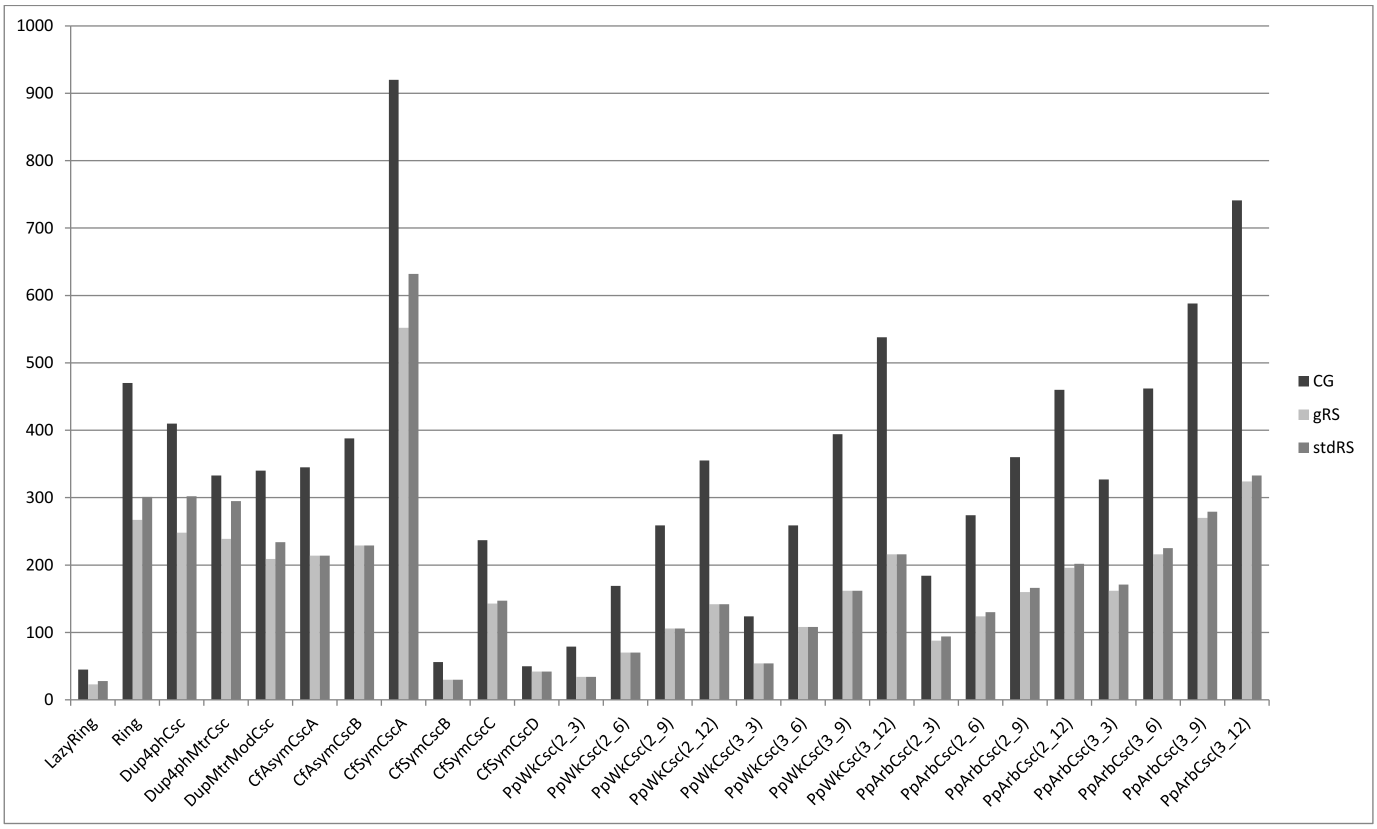


Figure 14: Analysis of fork balancing effort

consistent logic for asynchronous digital circuit synthesis. In *Proc. Int'l Conf. Application-Specific Systems, Architectures, and Processors*, 1996.

[16] C. Jeong and S.M. Nowick. Technology mapping and cell merger for asynchronous threshold networks. *IEEE Trans. on CAD*, 2008.

[17] Sean Keller, Siddarth Bhargav, Chris Moore, and Alain J. Martin. Reliable Minimum Energy CMOS Circuit Design. In *Vari'11: 2nd European Workshop on CMOS Variability*, 2011.

[18] Victor Khomenko, Maciej Koutny, and Alexandre Yakovlev. Logic Synthesis for Asynchronous Circuits Based on STG Unfoldings and Incremental SAT. *Fundamenta Informaticae*, 70(1-2):49–73, 2006.

[19] A. Kondratyev, M. Kishinevsky, and A. Yakovlev. Hazard-free implementation of speed-independent circuits. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 17(9):749–771, sep. 1998.

[20] Alex Kondratyev and Kelvin Lwin. Design of asynchronous circuits using synchronous CAD tools. *IEEE Transactions on Computers*, 2002.

[21] J.P. Kulkarni, K. Kim, and K. Roy. A 160 mV Robust Schmitt Trigger Based Subthreshold SRAM. *IEEE Journal of Solid-State Circuits*, 42(10):2303–2313, 2007.

[22] Agnes Madalinski. *Interactive Synthesis of Asynchronous Systems based on Partial Order Semantics*. PhD thesis, Newcastle University, 2006.

[23] A.J. Martin. Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits. In *Developments in Concurrency and Communication*, UT Year of Prog. Series, pages 1–64, 1990.

[24] A. Mokhov, V. Khomenko, D. Sokolov, and A. Yakovlev. On Dual-Rail Control Logic for Enhanced Circuit Robustness. Technical Report NCL-EECE-MSD-TR-2010-162, Newcastle University, 2010.

[25] D. Muller and W. Bartky. A Theory of Asynchronous Circuits. In *Proc. Int. Symp. of the Theory of Switching*, pages 204–243, 1959.

[26] Ankireddy Nalamalpu and Wayne Burleson. Repeater insertion in deep sub-micron cmos: Ramp-based analytical model and placement sensitivity analysis. In *In IEEE International Symposium on Circuits and Systems*, pages 766–769, 2000.

[27] Steven Nowick. *Automatic Synthesis of Burst-Mode Asynchronous Controllers*. PhD thesis, Stanford University, 1993.

[28] M.-D. Shieh, C.-L. Wey, and P.D. Fisher. A scan design for asynchronous sequential logic circuits using SR-latches. In *Symposium on Circuits and Systems*, volume 2, pages 1300–1303, 1993.

[29] Danil Sokolov, Julian Murphy, Alexander Bystrov, and Alex Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Transactions on Computers*, 54:449 – 460, 2005.

[30] Jens Sparsoe and Steve Furber. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001.

[31] N. Starodoubtsev and S. Bystrov. Behavior and synthesis of two-input-gate asynchronous circuits. In *Symposium on Asynchronous Circuits and Systems (ASYNC'2005)*, pages 190–200, 2005.

[32] N. Starodoubtsev, S. Bystrov, M. Goncharov, I. Klotchkov, and A. Smirnov. Towards synthesis of monotonic asynchronous circuits from signal transition graphs. In *Proceedings of the Second International Conference on Application of Concurrency to System Design*, 2001.

[33] Z.Al Tarawneh, G.Russell, and A.Yakovlev. An analysis of SEU robustness and performance of C-element structures implemented in bulk CMOS and SOI technologies. Technical report, Newcastle University, 2010.

[34] Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10246. IEEE Computer Society, 2004.

[35] E. Tuncer, J. Cortadella, and L. Lavagno. Enabling adaptability through elastic clocks. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 8–10, 2009.

[36] E. Vittoz. *Low Power electronics design, Chapter 16*. CRC Press, 2004.

[37] A. Wang and A. Chandrakasan. A 180mV FFT processor using sub-threshold circuit techniques. In *Solid-State Circuits Conference*, 2004.