Design Automation for Adiabatic Circuits

Alwin Zulehner[1] Michael P. Frank[2] Robert Wille[1]
[1]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[2]Center for Computing Research, Sandia National Laboratories, Albuquerque, USA
alwin.zulehner@jku.at mpfrank@sandia.gov robert.wille@jku.at

Abstract—Adiabatic circuits are heavily investigated since they allow for computations with an asymptotically close to zero energy dissipation per operation—serving as an alternative technology for many scenarios where energy efficiency is preferred over fast execution. Their concepts are motivated by the fact that the information lost from conventional circuits results in an entropy increase which causes energy dissipation. To overcome this issue, computations are performed in a (conditionally) reversible fashion which, additionally, have to satisfy switching rules that are different from conventional circuitry—crying out for dedicated design automation solutions. While previous approaches either focus on their electrical realization (resulting in small, hand-crafted circuits only) or on designing fully reversible building blocks (an unnecessary overhead), this work aims for providing an automatic *and* dedicated design scheme that explicitly takes the recent findings in this domain into account. To this end, we review the theoretical and technical background of adiabatic circuits and present automated methods that dedicatedly realize the desired function as an adiabatic circuit. The resulting methods are further optimized—leading to an automatic and efficient design automation for this promising technology. Evaluations confirm the benefits and applicability of the proposed solution.

I. Introduction

As we approach the end of the semiconductor roadmap [1], we are entering a regime in which fundamental thermodynamic considerations limit the sub-threshold slope, practical switching voltages, and gate energies—implying that further downscaling of device sizes and gate capacitances will soon no longer yield improvements in energy efficiency for conventional logic. Industry's shift towards 3D geometries [1] will somewhat reduce parasitic energy losses in circuit structures, but once that line of improvements is played out, the only remaining approach to further increase energy efficiency will be to begin applying techniques of energy recovery. In this regard, resonant circuit techniques to recycle and reuse logic signal energies, rather than dissipating the entire $\frac{1}{2}CV^2$ circuit node energy on each logic-level transition, are promising. Unlike all other options, no fundamental theoretical limits on the ultimate energy efficiency of energy recovery are known at present—thus, offering a path towards future growth of computing performance within any given energy dissipation constraints.

But apparently the ideal of 100% energy recovery implies that all switching activity of a device must be carried out in a manner that is asymptotically adiabatic—avoiding any abrupt loss of signal energy to heat. This motivated the consideration of *adiabatic circuits* which allow for computations with an asymptotically close to zero energy dissipation (at the expense of a slower execution). Due to Landauer's limit [11], this in turn implies that the computational function of the switching circuit must be *logically reversible*, in the appropriately generalized sense discussed in [6]. Otherwise, the information lost from a conventional circuit leads to an entropy increase and, therefore, to an irreducible energy dissipation. This was recently also advocated to a larger community in [7] stating that the future of computing depends on reversible computations.

While these concepts have already been around for a while—general techniques for designing fully-adiabatic and reversible circuits have been introduced in the 1990's and resulted in a large body of literature (see e.g. [9, 8, 13, 17])—most of the adiabatic design families that have been proposed contain flaws preventing them from being truly adiabatic [5]. In this regard, *two-level adiabatic logic* (2LAL as proposed in [2]) represents a very promising, fully-adiabatic transmission-gate logic family that relies on simple but rather efficient building blocks. However, to realize correct adiabatic and reversible circuit designs that could truly approach arbitrarily low levels of energy dissipation requires to satisfy certain *switching rules* which differ from the design of conventional circuitry—crying out for automated approaches for the design of such adiabatic circuits. Heading into this direction recently also gained relevance in industry—triggered e.g. by investments of funding agencies and national departments [7]. Accordingly, researchers started to work towards such solutions.

However, previously proposed approaches either focus on their electrical realization (see e.g. [17, 2]) or on designing purely reversible building blocks like Toffoli gates (see e.g. [15, 16]). While the former approaches are restricted to small and hand-crafted circuits only, relying on purely reversible building blocks results in an unnecessarily large overhead. Instead, recent findings (summarized in [6]) show that conditional reversibility is sufficient for adiabatic circuits. But thus far, no design automation approach for adiabatic circuits exists which exploits that in an automatic fashion.

In this work, we overcome this issue by combining expertise from both adiabatic circuits and design automation. More precisely, we review the theoretical and technical background of adiabatic circuits and, based on that, propose an automatic *and* dedicated design flow for this promising technology. Two complementary design styles (namely retractile and fully-pipelined) are thereby considered which allow for the generation of adiabatic circuits either focusing on reducing the number of gates or keeping the number of so-called power clocks small. Furthermore, optimizations for both design styles are proposed which utilize application-specific properties and, by this, allow e.g. for a reduction in the number of gates by approx. 37% and 30% on average for the retractile and fully-pipelined design styles, respectively. Evaluations confirm the benefits and applicability of the proposed solution.

M. Frank was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories and by the Advanced Simulation and Computing program under the U.S. Department of Energys National Nuclear Security Administration (NNSA). Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for NNSA under contract de-na0003525. Approved for public release, SAND2018-9936 O.

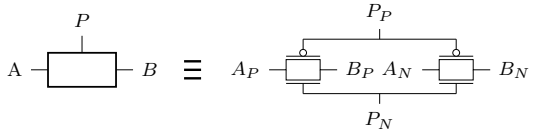
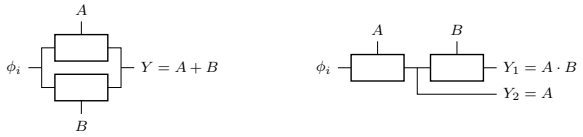


Fig. 1. Transmission gate for dual-rail signals



(a) OR gate (b) AND gate

Fig. 2. Adiabatic gates

II. Adiabatic Circuits

In this work, we consider design automation for adiabatic circuits according to the *two-level adiabatic logic* (2LAL, [2]) circuit family. This type of adiabatic circuit uses only two different voltage levels and heavily relies—on transmission gates. Furthermore, a dual-rail encoding is used for the signals of the circuit, i.e. each signal occurs in uncomplemented as well as in complemented form.[1]

Fig. 1 provides the notation for transmission gates: If the signal P is 1 (the gate is *turned on*), A and B are connected.[2] Otherwise (the gate is *turned off*), A and B are not connected. Since A and B are both encoded in a dual-rail fashion and, thus, have an uncomplemented as well as complemented form, two transmission gates as shown in the right-hand side of Fig. 1 are required.[3] The general *switching rules* for transistors in adiabatic circuits (e.g. outlined in [2, 5]) imply that a transmission gate shall never be turned on if A and B have different values.

Besides that, so-called *power clocks* (denoted ϕ_i) are additionally utilized to realize typical functions such as OR or AND. More precisely, the inputs of the gate control a network of transmission gates which connect the output Y of the gate to one of the power clocks ϕ_i in case the function to be realized evaluates to 1. To obey the switching rules, the output Y of the gate as well as the power clock ϕ_i are assumed to be 0 initially. By transitioning the power clock to 1, the output of the gate is set to the desired value. Moreover, when resetting all inputs of a gate to 0 (and, thus, disconnecting ϕ_i and Y) while ϕ_i is still 1, the output preserves its value (even if resetting ϕ_i to 0 afterwards). This allows for an inherent *latching* of an output value to be used by following gates. An example illustrates the idea:

Example 1 *Fig. 2 shows the 2LAL realization of an OR gate and an AND gate. The OR gate is composed of two parallel transmission gates whose outputs are connected. In case $A = 1$ ($B = 1$), the upper (the lower) transmission gate is turned on and connects the power clock ϕ_i to the output Y. Consequently, Y is connected to ϕ_i if $A + B = 1$. Transitioning ϕ_i to 1 sets Y to the desired value. If we now reset the inputs A and B to 0, the output Y is latched—its value is preserved even when setting ϕ_i back to 0 afterwards. The AND gate is realized similarly as a sequence of two transmission gates. Note that a second output $Y_2 = A$ is required in this case to operate the gate in an adiabatic fashion in case $A = 1$ and $B = 0$ when used in a fully-pipelined circuit (cf. Section V).*

[1] The uncomplemented form of a signal is labeled with a subscript N and the complemented form is labeled with subscript P.

[2] Note that logic 1 (i.e. $X = 1$) is realized by $X_N = 1$ and $X_P = 0$ since a dual-rail encoding is employed.

[3] For sake of simplicity, we abstract the two transmission gates in the following illustrations and use the more compact form as shown in the left-hand side of Fig. 1 instead.

Once the output of a gate is not needed anymore (e.g. by a following gate), an essential step for adiabatic circuits is the ability to decompute it—feeding charge back to the power clocks. In case that the output was not latched (i.e. the output is still connected to the power clock), it is decomputable by simply resetting the power clock to 0 (as discussed above). If the output is latched (i.e. it was disconnected from the power clock by setting the inputs back to 0), the power clock has to be transitioned to 1 as well, before the inputs are applied in order to obey the switching rules. Then, the output is decomputable by transitioning the power clock back to 0.

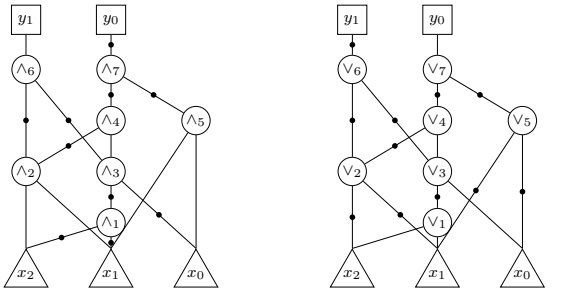
Example 2 *Consider again the 2LAL realization of an OR gate (cf. Fig. 2a). Assume that the output $Y = A+B$ of the gate is latched and that all other signals are set to 0. To unlatch the output Y, we first have to set the power clock ϕ_i to 1. By this, ϕ_i and Y have the same value if they get connected by resetting the inputs to their original value. Then, Y is decomputable by changing the power clock ϕ_i back to 0—the charge representing $Y = 1$ is fed back to the power supply.*

Following this main principle allows for conducting operations with an asymptotically close to zero energy dissipation (at the expense of a slower execution since more steps have to be conducted). In fact, in contrast to conventional circuits in which energy is frequently "grounded", adiabatic circuits allow for feeding energy back to the clocks providing the power supply.

However, this concept of feeding back charge to the power clocks by decomputing signals demands for a logical reversibility of the underlying computations. This is because, in order to not violate the switching rules, the original input assignments have to be applied so that signals with different values are never connected (cf. Example 2). While in the past, a pure reversible scheme has been assumed (see e.g. [15, 16]), findings recently summarized in [6] showed that conditional reversibility is actually sufficient for adiabatic circuits. Again, this is illustrated by means of an example:

Example 3 *Consider again the OR gate shown in Fig. 2a. Considering the state of the signals A, B, and Y, the gate describes a function $f : \mathbb{B}^3 \to \mathbb{B}^3 = (A, B, Y) \to (A, B, A + B)$. This function is not reversible in general, since the initial value of Y can not be computed from the output values. However, the function is conditionally reversible under the precondition that the value of Y is initially set to 0, i.e. an input combination like e.g. $(1, 0, 1)$ can never occur. Conditional reversibility is a much weaker constraint than unconditional reversibility (as e.g. considered in [15, 16])—allowing to realize adiabatic gates as e.g. shown in Fig. 2.*

Obviously, conducting computations in such a fashion requires the corresponding circuits to be designed in a significantly different fashion than conventional circuitry. Besides the generation of a proper netlist composed of transmission gates, this additionally requires dedicated power clocks which correspondingly trigger the required operations at the correct point in time. Moreover, also the design objectives change. While the number of required (transmission) gates is still a factor (e.g. to approximate the required area), their impact on energy consumption is smaller than for conventional circuits. This is because energy is never grounded in adiabatic circuits but frequently fed back to the power supply as described above. In contrast, the number of power clocks is much more crucial as they are the entities which actually require energy and whose waveform might be hard to generate. Besides that, more clocks usually also require longer execution times.



(a) AND-Inverter graph (b) OR-Inverter graph

Fig. 3. Graph representations for Boolean functions

III. Proposed Design Flow

As discussed above, previous design methods for designing adiabatic circuits (e.g. [15, 16]) assumed the requirement of full reversibility. As recently discussed in [6], this leads to a significant overhead and is not necessarily needed. In fact, conditional reversibility as reviewed above is sufficient and constitutes a much weaker constraint. However, thus far, no design automation for this kind of adiabatic circuits exists. Also, solely employing conventional design solutions is not an option since, despite the pure functionality, a dedicated mapping and clocking scheme is required. In this work, we present different methods which address these issues. All of them employ thereby a two-stage process. The first step is similar to the design of conventional circuits: We realize the function to be synthesized with respect to a certain logic gate library. Afterwards, the resulting netlist is mapped to an adiabatic circuit which respectively satisfies and optimizes the rules and objectives reviewed in Section II.

For the first part, we utilize a solution based on *AND-Inverter Graphs* (AIGs [10]) which realize the function to be synthesized in terms of NAND gates.[4] AIGs allow for a graph-based representation of Boolean functions. The graph has one root node for each output of the function. The inputs of the function are provided as terminals. The intermediate nodes of an AIG represent an AND operation and, thus, have two successors each. To gain universality, the inputs of the AND operation can be inverted. This is denoted by black circles on the respective edges. Equal nodes occur frequently and can be shared—allowing for a compact representation of the function to be realized.

Example 4 *Fig. 3a shows the AIG of a 3-input 2-output Boolean function with inputs x_2, x_1, and x_0 as well as outputs y_1 and y_0 which represent $y_1 = \overline{x}_2\overline{x}_1 + \overline{x}_2 x_0 + \overline{x}_1 x_0$ and $y_0 = \overline{x}_2 x_1 + x_1 x_0 + x_2\overline{x}_1\overline{x}_0$ in terms of an AIG and, hence, NAND operations.*

How to determine and optimize an AIG (e.g. minimizing its number of nodes/gates) has intensely been considered in the literature (see e.g. [14]) and, hence, is not covered further in the following. Instead, we focus on the second step, i.e. how to map the resulting NAND netlist to an adiabatic circuit, i.e. a network of transmission gates and the corresponding power clocks. To this end, we translate the AIG into an *OR-Inverter graph* (OIG) so that a NOR gate netlist results. An OIG can easily be derived from an AIG by simply applying De Morgan's laws, i.e. by relabeling the inner nodes from AND to OR and inverting the polarity of the edges to the terminals and the edges to the root nodes (cf. 3b).

[4]Note that the design methods proposed in this work can correspondingly be adjusted to any other synthesis solution and, hence, logic gate library as well.

Now, the nodes of an OIG can directly be mapped to the adiabatic OR gates introduced before in Fig 2a. However, it remains open and non-trivial how to connect these gates to the power clocks and how to generate a corresponding waveform of these clocks (again, following the switching rules and optimization objectives reviewed in Section II). To this end, two (complementary) design styles are considered: *retractile* circuits (cf. [8]) as well as *fully-pipelined* circuits (cf. [17, 2, 6]). Note that for both design styles the conditional reversibility is inherently satisfied by preserving the inputs of the signals throughout the whole computation and by assuming that all additional (intermediate) signals are initially set to 0. In the following sections, we discuss advantages and disadvantages of both design styles and present according (automatic) mapping schemes. More precisely, for each design style we first describe a straightforward mapping scheme (conveying the main idea of the design style) followed by an advanced mapping scheme (which results in a significantly smaller number of gates as well as, in case of retractile circuits, to a smaller number of power clocks). These considerations eventually motivate the implementation of different methods for design automation of adiabatic circuits whose performance is eventually discussed in Section VI.

IV. Retractile Circuits

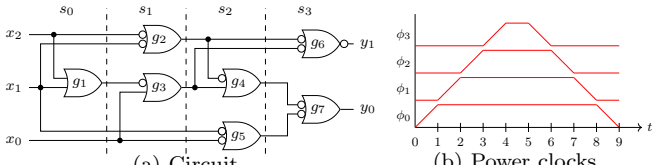
A. Straightforward Solution

The straightforward mapping for retractile circuits is similar to conventional circuitry, where an AIG or OIG is directly mapped to the target technology. In fact, we can realize each node of the OIG with an OR gate and negations with inverters. Moreover, in case of adiabatic circuits, the inverters come "for free" since we are operating on dual-rail signals and, hence, an inverted input can easily be realized with no further hardware by swapping the rails of the signal.

Example 5 *Consider again the OIG depicted in Fig. 3b. Mapping the OIG to conventional gates results in the circuit shown in Fig. 4a. Doing this mapping for adiabatic circuits following the retractile design style, each OR-gate is realized with two transmission gates as discussed in Section II.*

To operate the circuit in an adiabatic fashion, all intermediate signals are first initialized with 0. Furthermore, each stage s_i $(0 \leq i < N)$ of the circuit with depth N has an associated dual-rail encoded clock ϕ_i—allowing to compute the individual stages sequentially. Then, the computations are started by transitioning the 0^{th} clock from 0 to 1—triggering the desired operations of the first stage. Once stable, the operations of the next stages are sequentially triggered. To allow for decomputing the intermediate results, the clocks transition back to 0 in reverse order, i.e. first the $N-1^{th}$ clock is set back to 0, then the other ones. This way, all intermediate results are decomputed and restored back to 0. Overall, this requires $2N+1$ time steps for a single computation (assuming one additional time step is required to process the outputs of the circuit). During these time steps, the inputs have to remain constant—yielding a rather low throughput.

Example 5 (continued) *Since the resulting circuit has four stages (the OIG has a depth of 4), we need four different clocks (eight if we take the dual-rail encoding into account). The waveforms of these clocks are shown in Fig. 4b. Overall, this causes that a single computation of this circuit requires 9 timesteps.*



(a) Circuit (b) Power clocks

Fig. 4. Synthesized retractile circuit

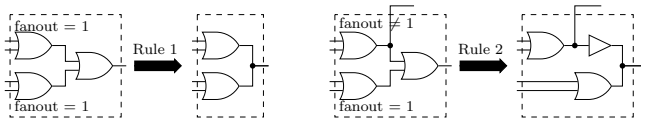


Fig. 5. Rules for optimization

B. Advanced Solution

The straightforward mapping described above can significantly be optimized to reduce the number of required transmission gates and power clocks. The optimized mapping scheme is motivated by an analysis of the realization of an OR gate, which is composed of two parallel buffers (i.e. a transmission gate), whose outputs are connected (cf. Fig. 2a). Consequently, an OR gate with multiple inputs can be generated by adding further buffers in parallel. This way, each OIG node, whose children both have a fanout of 1 (and, thus, represents a 4-inputs OR gate) can be realized in a single stage of the circuit composed of two 2-input OR gates whose outputs are connected. A similar optimization can be performed for OIG nodes, where only one of the children has a fanout of 1. Here, one buffer is required for the child which has a fanout larger than one (in order to avoid sneak-paths). Additionally, the gate representing the child with fanout 1 has to be lifted to the next stage of the circuit since both, the buffer as well as the gate, have to be operated by the same power clock to allow for an adiabatic computation. The optimization rules are shown in Fig. 5 and denoted *Rule 1* and *Rule 2* in the following.

Note that one has to be careful when applying the rules if the corresponding input of the gate is inverted. In this case, the inversion has to be pushed towards the inputs. This is possible by applying De Morgan's law ($\overline{a+b} = \overline{a} \cdot \overline{b}$). Consequently, we have to invert the inputs on this level and exchange the OR gate with an AND gate.[5]

Example 6 *Consider again the circuit shown in Fig. 4. The children of gate g_7 (i.e. g_4 and g_5) both have a fanout of 1. Consequently, we can apply* Rule 1 *to remove g_7. Furthermore, one child of gate g_3 has a fanout of 1 (i.e. the input x_0). Consequently, we can apply* Rule 2 *for gate g_3. The resulting (optimized) circuit is shown in Fig. 6. Since both inputs of g_7 and one input of g_3 are inverted, we have to apply De Morgan's law. Consequently, the gates g_1, g_4, and g_5 are transformed into an AND-gate. The resulting circuit only requires 11 transmission gates and has only two stages (and, thus, suddenly requires only two different dual-rail encoded clocks).*

V. Fully-Pipelined Circuits

The main disadvantages of the retractile circuits considered in Section IV are that many different power clocks are required (one for each stage) and that a computation can be conducted only every $2N+1$ time steps—resulting in a rather low throughput. These issues can be avoided by using fully-pipelined circuits. In conventional design, this would require a register after each stage of the circuit. For the adiabatic circuits considered here, however, this is not necessary, because the gates inherently allow for latching their output (cf. Section II). In fact, we only have to compute the outputs of a stage s_i while decomputing the signals of stage s_{i-1} (i.e. resetting them back to 0). This way, only two different power clocks (four if we take the dual-rail encoding into account) are required (independent from the circuit depth) and computations can be conducted in a pipelined fashion (leading to a much higher throughput).

[5]Note that this is also possible if there are several subsequent nodes for which the rules can be applied.

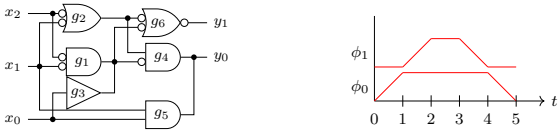
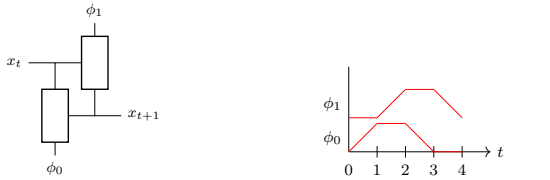


Fig. 6. Optimized retractile circuit



(a) Transmission gates (b) Clocks

Fig. 7. Buffer element for fully-pipelined circuits

To realize this, however, the functions computed in the individual stages have to be (conditionally) reversible. This can easily be achieved by forwarding all the input signals of stage s_{i-1} to the stage s_i by using buffers. The following example illustrates the idea of such buffers.

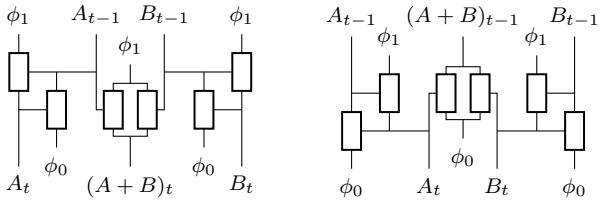
Example 7 *Fig. 7a shows the structure of a buffer that sets $x_{t+1} = x_t$ while decomputing x_t (i.e. while resetting x_t back to 0). Initially, both clocks ϕ_0 and ϕ_1 as well as x_{t+1} are set to 0. If $x_t = 1$, the transmission gate on the right connects ϕ_1 with x_{t+1}. In the first time step, ϕ_0 transitions to 1 (c.f. Fig. 7b). Afterwards, ϕ_1 transitions to 1, setting $x_{t+1} = x_t$. If $x_{t+1} = x_t = 1$, the transmission gate on the left hand side in Fig. 7a connects ϕ_0 with x_t. This does not violate the switching rules discussed in Section II since ϕ_0 is also 1. In the next time step, ϕ_0 transitions back to 0—decomputing x_t and, thus, disconnecting ϕ_i and x_{t+1}. Consequently, the output x_{t+1} remains at its voltage level when eventually transitioning ϕ_1 back to 0—the output is latched.*

To allow for inverted inputs of gates, a quad-rail encoding is required for the signals to properly decompute the inputs [2]. Here, each signal X is represented by two dual-rail signals (one for $X = 1$ and one for $X = 0$). Initially, both dual-rail signals are set to 0. This again allows to realize inverters without any transmission gates—just swapping the two dual-rail signals $X = 1$ and $X = 0$. In the following we again abstract this fact when illustrating the required transmission gates.

A. Straightforward Solution

As for retractile circuits, we again map the OIG nodes to an adiabatic realizations of an OR gate. As mentioned above, this requires to realize each OR gate as shown in Fig. 8a.[6] This way, the signals from stage s_{t-1} (e.g. A_{t-1} and B_{t-1}) serve as input to compute $(A+B)_t = A_{t-1} + B_{t-1}$. Since $(A+B)_t$ is driven by clock ϕ_1, its value is inherently latched. In fact, the input signals A_{t-1} and B_{t-1} are reset to 0 by the according buffers (disconnecting ϕ_1 and $(A+B)_t$), before the clock ϕ_1 is transitioned back to 0.

[6]Signals with fanout do not have to be buffered multiple times.



(a) Computing OR (b) Decomputing OR

Fig. 8. OR gate for fully-pipelined circuits

Now, in contrast to retractile circuits, new hardware is required to decompute the result (after e.g. copying it elsewhere) since the stages of the pipeline already contain the values of the next computation. The (conditionally) reversible function calculated by the pipeline is $F = f_{N-1} \circ f_{N-2} \circ \cdots \circ f_0$, where f_i is the conditionally reversible function computed by stage s_i.[7] Since the function f_i computed by each stage is conditionally reversible, the inverse of F (i.e. F^{-1}) exists and is determined by $F^{-1} = f_0^{-1} \circ f_1^{-1} \circ \cdots \circ f_{N-1}^{-1}$. The inverse f_i^{-1} of the function f_i computed by stage s_i can be easily realized by duplicating the hardware for stage s_i and reconnecting the power clocks ϕ_0 and ϕ_1 (as shown for an OR gate in Fig. 8b). Consequently, decomputing the results requires to double the depth of the pipeline and, thus, doubles the number of required transmission gates.

Example 8 *Consider again the circuit shown in Fig. 4. The first stage contains a single OR gate. Additionally, three buffers are required to forward the inputs x_2, x_1, and x_0 to stage s_1 (while decomputing them in stage s_0). Consequently, $(1+3) \cdot 2 = 8$ transmission gates are required. The second stage has four input signals and requires two OR gates. Therefore, $(4+2) \cdot 2 = 48$ transmission gates are required to realize stage s_1. The third stage has then 6 inputs and requires 16 transmission gates. Finally, the last stage has 8 inputs and requires 20 transmission gates. Overall, this sums up to 56 transmission gates. The reverse cascade of the stages again requires 56 transmission gates. Consequently, a total of 112 transmission gates are required (448 if we take the quad-rail encoding into account) to realize the function in a fully-pipelined fashion—a huge overhead compared to the retractile design methodology. However, the circuit has a higher throughput and only requires two different clocks to be operated (four if we take into account that their complement is also needed due to a dual-rail encoding).*

B. Advanced Solution

The mapping scheme discussed above yields circuits with a huge overhead since many signals are pushed through the whole pipeline—even though they are not required as outputs or to obtain reversibility of a stage. Hence, we propose to decompute such unnecessary signals as soon as possible. As shown in Fig. 8b, the inputs of a gate have to be present until its output is decomputed. This means, the signals resulting from the gates in the next-to-last stage can be decomputed while computing the outputs of the function to be realized. Afterwards, the signals generated in the stage before can be decomputed—eventually resulting in the mapping scheme discussed in the previous subsection—hence, no signal can be decomputed before the final outputs of the function to be realized are determined.

[7] Note that $\circ$ denotes functional composition, i.e. $g(x) \circ f(x) = g(f(x))$.

However, we can easily circumvent this problem by choosing some signals that shall not be decomputed.[8] To this end, we mark the corresponding OIG nodes that generate these signals. This allows to decompute several other signals earlier—while continuing to compute the outputs of the function. Consequently, fewer signals are pushed through the pipeline—reducing the number of required transmission gates.

Recall, that each node v of the OIG is translated to an OR gate on a certain stage of the circuit. To determine when the signal resulting from v can be decomputed we traverse all parents (denoted p_j in the following). For each parent node p_j we determine the stage in which the signal generated by v can be decomputed at the earliest. Then, we take the stage with the largest index, since the constraints for all parents have to be satisfied. If p_j is a node that is marked, we can immediately decompute the signal generated by v in the same stage (since the signal computed by p_j is not decomputed afterwards). If p_j is not marked, we can decompute the signal generated by v at the earliest one stage after the signal generated by p_j can be decomputed (because the signal generated by v is required to decompute the signal generated by p_j).

Example 9 *Consider again the OIG shown in Fig. 3b (as well as the corresponding circuit shown in Fig. 4). Assume that we marked the nodes labeled $\vee_2$ and $\vee_5$ (the nodes labeled $\vee_6$ and $\vee_7$ are inherently marked since they are directly connected to an output). Consequently, we want to decompute the signals generated by the OIG nodes labeled $\vee_1$, $\vee_4$, and $\vee_5$ as soon as possible. In the second stage (i.e. s_1) of the circuit, we compute the result of the nodes labeled $\vee_2$ and $\vee_3$. Since the signal generated by node $\vee_1$ is not required anymore (its single parent labeled $\vee_3$ is marked), it can be decomputed in the second stage as well. Consequently, we can save the buffers for this signal in the third and fourth stage of the circuit. Furthermore, the signals generated by nodes labeled $\vee_4$ and $\vee_5$ can be decomputed while computing the outputs of the function (in stage s_3). Since this is the last stage of the circuit, no buffers can be saved. However, fewer output signals result. Considering the fact that each pipeline stage has to be duplicated, a reduction of four buffers (i.e. 8 transmission gates) can be obtained.*

This leads to the question how to determine a suitable marking scheme for the nodes, i.e. a marking scheme that results in a circuit with a smaller number of transmission gates. A very simple but also effective marking scheme is to mark all nodes of the OIG with a depth that is a multiple of a constant $k \in \mathbb{N}$. For $k = 2$, this means to mark all nodes with an even depth (as done in Example 9). The experimental evaluations summarized in Section VI show that significant improvements can be obtained by using this marking scheme.

VI. Evaluation

In this section, we summarize and discuss the results obtained by our evaluations of the proposed design methods for adiabatic circuits. To this end, we implemented the approaches discussed in Section IV and Section V in C++ and used the tool ABC [3] to generate the initially required AIGs/OIGs (to reduce the number of AIG nodes, we used the synthesis command *dc2*). Afterwards, we evaluated the resulting methods using benchmarks taken from the ISCAS [4] and the IWLS benchmark suite [12].

[8] Note that, in the end, all signals are decomputed since each stage is duplicated as discussed in Section V.A.

TABLE I
EVALUATION

			Retractile (Sec. IV)				Fully-pipelined (Sec. V)					
			Str.-forw. (Sec. IV.A)		Advanced (Sec. IV.B)		Str.-forw. (Sec. V.A)			Advanced (Sec. V.B)		
Name	PI	PO	$\|\phi\|$	$\|tg\|$	$\|\phi\|$	$\|tg\|$	$\|\phi\|$	$\|tg\|$	k	$\|\phi\|$	$\|tg\|$	
apex5	117	88	26	1 826	24	1 181	4	150 544	2	4	112 208	
ex4p	128	28	28	2 020	20	1 260	4	205 968	3	4	136 080	
o64	130	1	16	258	2	130	4	30 928	2	4	23 936	
i3	132	6	12	252	2	132	4	23 024	2	4	18 528	
i5	133	66	36	264	12	160	4	60 096	4	4	50 384	
i8	133	81	30	1 804	24	1 079	4	147 104	3	4	117 504	
apex6	135	99	26	1 210	20	793	4	123 104	2	4	87 216	
x3	135	99	26	1 230	18	773	4	131 120	4	4	88 112	
rot	135	107	50	1 110	34	773	4	221 936	2	4	161 504	
i6	138	67	12	766	8	461	4	36 320	4	4	29 984	
frg2	143	139	24	1 508	20	958	4	116 496	3	4	92 288	
pair	173	137	44	2 608	40	1 693	4	373 552	3	4	245 584	
c5315	178	123	56	2 754	38	1 722	4	559 472	3	4	293 152	
i4	192	6	28	372	2	192	4	75 200	3	4	59 776	
i7	199	67	12	1 012	8	586	4	51 392	4	4	42 384	
i2	201	1	26	416	6	210	4	79 056	3	4	58 864	
c7552	207	108	76	2 940	62	2 085	4	868 272	4	4	474 912	
c2670	233	140	44	1 076	20	663	4	232 496	3	4	152 032	
des	256	245	36	6 784	26	4 495	4	752 016	5	4	499 184	
i10	257	224	66	3 440	48	2 292	4	776 400	3	4	503 184	

$|\phi|$: #required clocks $|tg|$: no. transmission gates
k: parameter discussed in Sec. V.B

Table I summarizes the obtained results. The first columns show the name of the benchmark as well as the number of primary inputs PI and primary outputs PO. Then, we list the results obtained for retractile and fully-pipelined adiabatic circuits. For each design style, we list the number of required transmission gates (denoted $|tg|$) and the number of required power clocks (denoted $|\phi|$) of the straightforward solution as well as the advanced solution (columns denoted *Str.-forw.* and *Advanced*, respectively). Having a dual-rail (for retractile circuits) or quad-rail encoding (for fully-pipelined circuits) is taken into account in the numbers listed for the required transmission gates, as well as the fact that each power clock has to be supplied in two polarities (i.e. a power clock is dual-rail encoded for both types of circuits). For sake of completeness, we also list the parameter k used in the solution discussed in Section V.B. The runtime is not listed in Table I since all methods are capable to produce these results in negligible runtime (i.e. a fraction of a second).

First, the results nicely show the impact of the respectively chosen design style. Retractile circuits are clearly the better choice when it comes to reducing the number of gates, while pipelined circuits are efficient with respect to the number of power clocks and, following that, also the throughput. At a first glance, it might look that the costs of having fewer power clocks in pipelined circuits is not acceptable (in fact, magnitudes more gates are required). However, if area is not an issue, this might still acceptable since, as discussed in Section II, gates in adiabatic circuits do not affect the energy consumption as much as they do in conventional circuits. Hence, each design style has its own advantages and disadvantages and, eventually, the user is presented with complementary solutions out of which the best suitable can be chosen.

Besides that, the results clearly show the improvement of the advanced schemes. On average an improvement of approx. 42% in the number of required power clocks, as well as an average improvement of approx. 37% with respect to the number of required transmission gates is obtained for retractile circuits. For the fully-pipelined circuits, we observe a reduction in the number of transmission gates of approx. 30% on average. Overall, these results clearly confirm the benefit and applicability of the proposed design automation techniques for this kind of circuits. While previously considered circuits were either handcrafted (following approaches e.g. proposed in [17, 2]) or relied on fully reversible realizations which led to an unnecessarily large overhead (as conducted in [15, 16] and discussed in [6]), the proposed design flow allows for generating the desired adiabatic circuits in an automatic fashion while, at the same time, satisfying the switching rules by conditional reversibility only. The improvements obtained by the advanced schemes additionally show the further potential that can be exploited following this direction.

VII. Conclusions

In this work, we proposed an automatic and dedicated design flow for adiabatic circuits which explicitly takes recent findings in this domain (namely that conditional reversibility is sufficient for adiabatic circuits) into account. The proposed flow first realizes the desired functionality in terms of an AIG/OIG and, afterwards, dedicatedly maps the resulting structure to an adiabatic description. For the latter step, two complementary schemes (namely retractile or fully-pipelined) are considered which allow the designer to either focus on reducing the number of gates or keeping the number of power clocks small. Furthermore, optimizations are proposed which allow for a reduction in the number of gates by approx. 37% and 30%, respectively, for both design styles on average. By this, expertise from both, adiabatic circuits and design automation, is combined yielding an automatic *and* dedicated design scheme for this promising technology. This eventually provides the basis for further studies including, besides others, more sophisticated optimizations, the design and use of larger building blocks, as well as the application of the proposed design flow in the physical implementation of adiabatic circuits.

References

[1] S. I. A. 2.0. International technology road-map for semiconductors. https://www.semiconductors.org/main/2015_international_technology_roadmap_for_ 2015.

[2] V. Anantharam, M. He, K. Natarajan, H. Xie, and M. P. Frank. Driving fully-adiabatic logic circuits using custom high-q mems resonators. In *ESA/VLSI*, pages 5–11, 2004.

[3] R. K. Brayton and A. Mishchenko. ABC: an academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40, 2010.

[4] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[5] M. P. Frank. Common mistakes in adiabatic logic design and how to avoid them. In *Int'l Conf. on Embedded Systems and Applications*, pages 216–222, 2003.

[6] M. P. Frank. Foundations of generalized reversible computing. In *Int'l Conf. on Reversible Computation*, 2017.

[7] M. P. Frank. Throwing computing into reverse. *IEEE Spectrum September 2017*, 2017.

[8] J. S. Hall. An electroid switching model for reversible computer architectures. In *Proceedings of Physics of Computation Workshop, Dallas Texas*, 1992.

[9] J. G. Koller and W. C. Athas. Adiabatic switching, low energy computing, and the physics of storing and erasing information. In *Physics of Computation Workshop*, pages 267–270, 1992.

[10] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[11] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961.

[12] K. McElvain. IWLS'93 benchmark set: Version 4.0. In *Int'l Workshop on Logic Synth.*, 1993.

[13] R. C. Merkle. Towards practical reversible logic. In *Physics and Computation*, pages 227–228, 1992.

[14] A. Mishchenko, S. Chatterjee, and R. K. Brayton. Dag-aware AIG rewriting a fresh look at combinational logic synthesis. In *Design Automation Conf.*, 2006.

[15] M. Morrison and N. Ranganathan. Synthesis of dual-rail adiabatic logic for low power security applications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 33(7):975–988, 2014.

[16] A. Rauchenecker, T. Ostermann, and R. Wille. Exploiting reversible logic design for implementing adiabatic circuits. In *Mixed Design of Integrated Circuits and Systems*, pages 264–270, 2017.

[17] S. G. Younis and T. F. Knight Jr. Practical implementation of charge recovering asymptotically zero power cmos. In *Proceedings of the 1993 symposium on Research on integrated systems*, pages 234–250. MIT Press, 1993.