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 resulted 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. [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.
When used in a fully-pipelined circuit (cf. Section V), the output of the gate 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 as well, before the outputs are computed 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 $\phi_1$ to 1. By this, $\phi_1$ 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 $\phi_1$ 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 $f : \mathbb{B}^3 \rightarrow \mathbb{B}^3 = (A, B, Y) \rightarrow (A, B, A + B)$. This function is not reversible in general because 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 [17,18])—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.
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 [10], 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) which realize the function to be synthesized in terms of NAND gates. As AIGs allow for a graph-based representation of Boolean functions, the graphs have 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 $y_1 = \overline{x_2} x_1 + \overline{x_2} x_0 + x_2 \overline{x_0}$ and $y_0 = \overline{x_2} x_1 + x_1 x_0 + x_2 \overline{x_0}$ 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. [11]) and, hence, is not covered further in the literature. 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. [16]).

Now, the nodes of an OIG can directly be mapped to the adiabatic OR gates introduced 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.

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. 3a. 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 VII.

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 $\phi_i$—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 computing the intermediate results, the clocks transition back to 0 in reverse order, i.e. first the $N - 1^{st}$ 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.

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.
For the adiabatic circuits considered here, however, this is not necessary, because the gates inherently allow for latching their output (cf. Section III). 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).

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 \( x_{i+1} = x_i \) while decomputing \( x_i \) (i.e. while resetting \( x_i \) back to 0). Initially, both clocks \( \phi_0 \) and \( \phi_1 \) as well as \( x_{i+1} \) are set to 0. If \( x_i = 1 \), the transmission gate on the right connects \( \phi_1 \) with \( x_{i+1} \). In the first time step, \( \phi_0 \) transitions to 1 (c.f. Fig. 7b). Afterwards, \( \phi_1 \) transitions to 1, setting \( x_{i+1} = x_i \). If \( x_{i+1} = x_i = 1 \), the transmission gate on the left hand side in Fig. 7a connects \( \phi_0 \) with \( x_i \). This does not violate the switching rules discussed in Section III since \( \phi_0 \) is also 1. In the next time step, \( \phi_0 \) transitions back to 0 —decomputing \( x_i \) and, thus, disconnecting \( \phi_1 \) and \( x_{i+1} \). Consequently, the output \( x_{i+1} \) remains at its voltage level when eventually transitioning \( \phi_1 \) 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. 3a. This way, the signals from stage \( s_{i-1} \) (e.g. \( A_{i-1} \) and \( B_{i-1} \)) serve as input to compute \( (A+B)_i \). Since \( (A+B)_i \) is driven by clock \( \phi_1 \), its value is inherently latched. In fact, the input signals \( A_{i-1} \) and \( B_{i-1} \) are reset to 0 by the according buffers (disconnecting \( \phi_1 \) and \( (A+B)_i \)) before the clock \( \phi_1 \) is transitioned back to 0.
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 \( s_i \). 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 third stage has then \( 3 \) input signals and requires two OR gates. Therefore, \( (4 + 2) \cdot 2 = 48 \) 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).

**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 the signals \( s_1 \). 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. 5, 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.

However, we can easily circumvent this problem by choosing some signals that shall not be decomputed\(^7\) 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 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. 4 (as well as the corresponding circuit shown in Fig. 4).

Assume that we marked the nodes labeled \( \lor_2 \) and \( \lor_3 \) (the nodes labeled \( \lor_6 \) and \( \lor_7 \) are inherently marked since they are directly connected to an output). Consequently, we want to decompute the signals generated by the OIG nodes labeled \( \lor_3, \lor_4, \) and \( \lor_5 \) as soon as possible. In the second stage (i.e. \( s_1 \)) of the circuit, we compute the result of the nodes labeled \( \lor_2 \) and \( \lor_3 \). Since the signal generated by node \( \lor_1 \) is not required anymore (its single parent labeled \( \lor_3 \) 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 \( \lor_4 \) and \( \lor_5 \) 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 \(^12\).

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

\(^8\)Note that, in the end, all signals are decomputed since each stage is duplicated as discussed in Section VII.
TABLE I

<table>
<thead>
<tr>
<th>Name</th>
<th>Retractile (Sec. V)</th>
<th>Fully-pipelined (Sec. V)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Str.-forw.</td>
<td>Advance.</td>
</tr>
<tr>
<td>apex5</td>
<td>117 88</td>
<td>26</td>
</tr>
<tr>
<td>exdp1</td>
<td>128 28</td>
<td>28</td>
</tr>
<tr>
<td>c64</td>
<td>130 16</td>
<td>16</td>
</tr>
<tr>
<td>t3</td>
<td>132 62</td>
<td>252</td>
</tr>
<tr>
<td>i5</td>
<td>133 66</td>
<td>36</td>
</tr>
<tr>
<td>i8</td>
<td>138 81</td>
<td>30</td>
</tr>
<tr>
<td>apex6</td>
<td>138 20</td>
<td>12</td>
</tr>
<tr>
<td>c3</td>
<td>139 99</td>
<td>26</td>
</tr>
<tr>
<td>rot</td>
<td>137 27</td>
<td>59</td>
</tr>
<tr>
<td>i6</td>
<td>137 67</td>
<td>26</td>
</tr>
<tr>
<td>frg2</td>
<td>143 139</td>
<td>24</td>
</tr>
<tr>
<td>pair</td>
<td>173 137</td>
<td>44</td>
</tr>
<tr>
<td>c5115</td>
<td>178 123</td>
<td>56</td>
</tr>
<tr>
<td>i4</td>
<td>192 62</td>
<td>26</td>
</tr>
<tr>
<td>i7</td>
<td>199 67</td>
<td>12</td>
</tr>
<tr>
<td>t2</td>
<td>201 1</td>
<td>26</td>
</tr>
<tr>
<td>c7552</td>
<td>207 108</td>
<td>76</td>
</tr>
<tr>
<td>c2070</td>
<td>213 140</td>
<td>44</td>
</tr>
<tr>
<td>des</td>
<td>256 245</td>
<td>36</td>
</tr>
<tr>
<td>i10</td>
<td>257 226</td>
<td>44</td>
</tr>
</tbody>
</table>

| \(\phi\): required clocks | \(\#\): transmission gates |

k: parameter discussed in Sec. V.B

Table I summarizes the obtained results. The first column shows the name of the benchmark as well as the number of primary inputs \(P\) and primary outputs \(O\). 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 \(\#\)) 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 \#required clocks 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 VII. 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 required power clocks (showing that the throughput). At 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 (in fact, magnitudes more gates are required). Besides 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 handcrafed (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 presented flow first realizes the desired functional-structure gates of approx. 30% on average. Overall, these 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