Exploiting Reversible Logic Design for Implementing Adiabatic Circuits

Andreas Rauchenecker, Timm Ostermann, Robert Wille
Institute for Integrated Circuits
Johannes Kepler University, Austria
Email: andreas.rauchenecker@jku.at, timm.ostermann@jku.at, robert.wille@jku.at

***Abstract*—Today, energy saving is a major design target, since the number of mobile- or power-independent devices is increasing. These devices should operate as long as possible with one battery charge. Especially for designs where speed is of secondary importance, adiabatic circuits are a promising alternative. These kind of circuits are rather slow but extremely energy efficient. However, the full potential of adiabatic circuits can only be fully exploited if computations are conducted in a reversible fashion. This is not the case for conventional circuits which are usually composed of non-reversible gates such as AND, OR, etc. At the same time, design methods for so-called reversible circuits received significant interest – mainly motivated by emerging technologies such as quantum computation or encoder design. In this work, we exploit the accomplishments of these complementary areas and, realize fully reversible adiabatic circuits. Experimental evaluations show that this yields circuits which have a substantially smaller power consumption than conventional circuit technologies.**

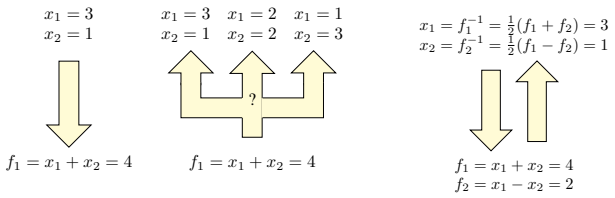
***Keywords*—reversible logic, adiabatic circuits, Toffoli gate, synthesis**

I. Introduction

In the modern world, computation devices are found everywhere. Most visible might be desktop and laptop computers, but the vast majority of computation devices is actually embedded in everything from children's toys to smartphones. In particular for the latter applications, power consumption becomes a major issue as it is directly related to usability and convenience in power-limited contexts. For example, battery-powered devices often need to be charged (e.g. smartphones almost daily) or require a change of batteries (e.g. hearing aids every week).

If conventional CMOS technologies reach their limits in further decreasing the power consumption, *adiabatic circuits* provide a promising alternative. Although these circuits are rather slow and require significantly more area than conventional circuits, they allow for executing computations in a power-efficient fashion. Hence, particularly for systems which (1) rely on a battery or other low-power connections, (2) are designed for a specialized application rather than for general computation, and (3) operate with little-to-no external input for long periods of time, adiabatic circuits provide an ideal trade-off. These findings led to many realizations of adiabatic circuits such as *Efficient Charge Recovery Logic* (ECRL, [1], [2]) or *Split-level Charge Recovery Logic* (SCRL, [3], [4]). However, the full potential of adiabatic circuits cannot be exploited by these solutions since they still consider the realization of conventional functions composed of gates such as AND, OR, etc. This is a significant drawback since energy recovery in adiabatic circuits relies, besides others, on reverting (i.e. uncomputing) the functionality of single gates. In fact, *reversible* computations allow to explicitly control a so-called discharge path and, by this, to re-use energy which has been loaded to capacitors. Unfortunately, only very few works exists which explicitly investigated the realization of purely reversible functions in the domain of adiabatic circuits (probably due to the fact that most circuits available thus far are inherently non-reversible). Among those are the works proposed in [5], [6], which however realized the reversible circuits again using conventional design methods (loosing the the benefit for adiabatic circuits). In [7], reversible functions have been realized in a quasi-adiabatic logic style – but again, without explicitly controlling the discharge path in order to save energy. Only the concept of *Reversible Energy Recovery Logic* (RERL, [8]) makes explicit use of this characteristic and employs a reversible gate in order to fully exploit the adiabatic computation and save energy. The main problem with this solution is, however, that very limited functionality has been realized with this concept thus far. Until today, no synthesis and technology mapping flow is available which lifts adiabatic circuits fully exploiting their potential to a level that allow for the realization of complex functionality. In this work, we propose such a methodology for circuits designed with methods for reversible circuits [9], [10], [11], [12], [13], [14], [15]. A technology scheme is proposed which realizes the logical circuits obtained by these solutions into corresponding transistor netlists – while at the same time satisfying all "rules" to be considered in order to fully exploit the benefits of adiabatic computation. Based on these contributions, a synthesis flow results which, for the first time, allows for the realization of fully reversible adiabatic circuits. Experimental evaluations show that the resulting circuits indeed outperform conventional solutions with respect to energy consumption.

The remainder of this paper is structured as follows: Section II reviews the basics on both (complementary) research areas, reversible circuit design and adiabatic circuits. This provides the motivation of our work (outlined in Section III) and leads to our main contribution: the implementation of fully reversible adiabatic circuits. Technical details on the corresponding implementation are afterwards covered in Section IV, while the resulting synthesis flow is described in Section V. The performance of the obtained reversible adiabatic circuits is finally evaluated and compared to conventional realizations summarized in Section VI. Section VII concludes the paper.



(a) Non-reversible logic (b) Reversible logic

Fig. 1. Computation paradigms

II. Background

This work deals with the exploitation of reversible circuits for implementing adiabatic circuits. Hence, we first review the corresponding background on both issues in this section.

A. Reversible Logic

In the past, researchers and engineers narrowed the investigation of computation machines down to a preponderantly *non-reversible* logic paradigm. A simple standard operation like the logical AND already illustrates that: Although, it is possible to obtain the input values of an AND gate if the output is set to 1 (then, both inputs must be set to 1 as well), it is not possible to determine the input values if the AND outputs 0.

In contrast, *reversible* computation is an alternative computation paradigm which only allows for bijective operations, i.e. reversible n-input n-output functions that map each possible input vector to a unique output vector. The underlying idea of reversible computation is exemplary illustrated in Fig. 1a by means of a simple addition. Performing solely the addition leads to an information loss and makes it impossible to undo the calculation without knowing the original inputs. Instead, if the addition is realized as shown in Fig. 1b computations can be performed in a reversible fashion, i.e. from the inputs to the outputs and vice versa.

Albeit not so well established yet, this computation paradigm is of significant interest for many emerging technologies such as quantum computation [16] as well as new ways for low-power computation [17], and additionally found interest e.g. in the design of encoders [18], on-chip interconnects [19], [20], or verification [21]. Consequently, a broad variety of different design solutions have been proposed for reversible logic – including e.g. exact methods that guarantee minimality with respect to the number of gates (e.g. [9], [10]) and heuristic approaches (e.g. [11], [12], [13], [14], [15]) as well as dedicated hardware description languages [22] which allow for the design of complex functionality. All these methods rely thereby on a circuit model which inherently is reversible.

More precisely, a *reversible circuit* realizes reversible functions, i.e. logic functions $f : \mathbb{B}^m \rightarrow \mathbb{B}^{m'}$ over inputs $X = \{x_0, \ldots, x_{m-1}\}$, where

- the number of inputs is equal to the number of outputs (i.e. $m = m'$) and
- each input pattern maps to a unique output pattern.

Since non-reversible gates such as AND, OR, etc. do not satisfy these characteristics (neither do they work on the same number of inputs/outputs nor do they realize a unique input/output mapping), reversible circuits are composed of *reversible gates* only.

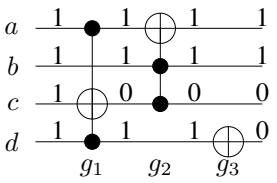


Fig. 2. Reversible circuit

The most frequently occurring reversible gate is the so-called *Toffoli gate* $T(C, t)$ which is composed of a set of *control lines* $C = \{x_{i_0}, x_{i_1}, \ldots, x_{i_{k-1}}\}$ with $C \subset X$ and a single *target line* $x_j \in X$ with $x_j \notin C$. This gate maps $(x_0, x_1, \ldots, x_j, \ldots, x_{m-1})$ to $(x_0, x_1, \ldots, (x_{i_0} x_{i_1} \ldots x_{i_{k-1}}) \oplus x_j, \ldots, x_{m-1})$, i.e. the target line is inverted if all control lines are set to 1; otherwise the value of the target line is passed through unchanged.

Fig. 2 shows a reversible circuit composed of $m = 4$ circuit lines and $d = 3$ Toffoli gates. Control lines are indicated by black circles, while a target line is indicated by $\oplus$. The gates are given by $g_1 = T(\{a, d\}, c), g_2 = T(\{b, c\}, a)$, and $g_3 = T(\emptyset, d)$. This circuit maps e.g. the input pattern 1111 to the output pattern 1100 (as shown in Fig. 2). Inherently, every computation can be performed in both directions (i.e. computations towards the outputs *and* towards the inputs can be performed).

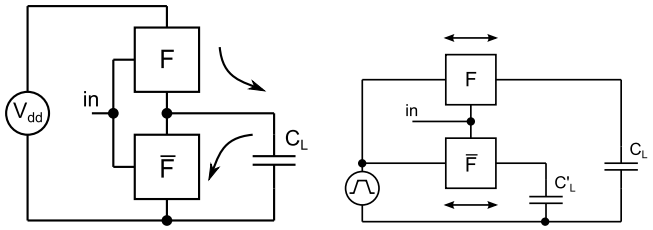
Reversible circuits as introduced above are universal, i.e. each desired function can be realized as a cascade of Toffoli gates (in order to realize non-reversible functionality, a so-called *embedding* step has to additionally be performed; see e.g. [23]). Motivated by several (emerging) technologies, also corresponding design methods are already available for this purpose (as already mentioned above; see e.g. [9], [10], [11], [12], [13], [14], [15]).

B. Adiabatic Circuits

In conventional CMOS technologies, the respectively desired result of a logic function (e.g. a gate) is realized by establishing a connection either to a supply voltage (*Vdd*) or to the ground. This is illustrated by means of Fig. 3a. More precisely, if an output of a gate is supposed to evaluate to 1, the load C_L of the gate[1] gets charged to the *Vdd* level (in Fig. 3a, realized by a block F and illustrated by the top arrow). Otherwise (if an output of a gate is supposed to evaluate to 0), the load C_L gets discharged to the ground (in Fig. 3a, realized by a block $\overline{F}$ and illustrated by the bottom arrow). Obviously, the latter case yields a significant loss of energy.

Adiabatic circuits have been introduced as an alternative to avoid this energy loss by avoiding discharges to the ground whenever possible. Here, the static power supply (as shown in Fig. 3a) is replaced by a pulsing power supply which frequently charges and discharges the loads C_L and C'_L as

[1]The load of a gate (here represented by C_L usually consists of the input capacitance of the following gate and the wire capacitance.



(a) CMOS-based Circuit (b) Adiabatic Circuit

Fig. 3. CMOS-based vs. Adiabatic Circuit

shown in Fig. 3b. To this end, adiabatic circuits are typically realized in a dual-rail fashion, thus there are two capacitances in the figure. By this, the respectively provided energy can be recovered during the low phase of the power supply. More precisely, if the block F evaluates to 1, then the load C_L gets charged if the power supply additionally rises to the 1-level. Otherwise (the block F evaluates to 0), load C_L is not charged. Since F is still valid when the power supply drops back to the ground level, the load C_L gets discharged through the same path as it has been charged. The same analogously happens for block $\overline{F}$. This way, the charge is recovered back to the power supply (in contrast to conventional CMOS-based circuits, where the energy used for charging the load is lost).

However, charging the load as described above can be seen as charging a capacitance through a resistor. This causes an energy dissipation E whose amount can be determined by

$$E = \frac{RC}{T} * CV^2, \tag{1}$$

where R, C, T, and V denotes the resistance, the capacity, the time needed for charging, and the voltage, respectively. But in contrast to conventional CMOS-based circuits, this energy loss can be controlled: In fact, the energy E is decreased when the time T is increased. Hence, instead of directly providing the full supply voltage (as in conventional gates), adiabatic circuits increase the time T by slowly ramping up the supply voltage. Although this yields a slower circuit, this constitutes a suitable trade-off for many energy-dependent applications.

Moreover, the time T can further be increased by ensuring that transistors in the charging/discharging path are only turned on if there is no potential difference between their drain and source contacts. Aside of that, dissipation increases when a transistor is turned off while current is flowing through it. Since the current is not cut off immediately, but over a certain period of time during which the transistor is in an intermediate state where the voltage drop over drain and source is increased, a higher power consumption results. Overall, this leads to two *rules* that allow for energy efficient adiabatic circuits:

1) Never turn on a transistor if there is a voltage difference between drain and source.
2) Never turn off a transistor if there is a current through it.

Because of the pulsing power supply, this requires a dedicated timing. In its simplest form, the process described above can be distinguished into three phases: a charging phase, an evaluation phase, and a discharging phase. In order to efficiently realize these phases, a pipelining structure composed of multiple gates as shown in Fig. 4. can be employed. Here, the result of a gate is handed over to the next stage during an evaluation phase. The power supply of the following stage should not ramp up until its inputs are stable, i.e. until the charging of the first stage is completed and the evaluation phase started for that gate. Therefore, the rising ramp of the power supply needs to be shifted by one phase. A disadvantage is that the stages get charged in ascending order and need to be discharged in descending order – resulting in highly impractical circuits. The solution is to use explicit discharge paths rather than charging and discharging through the same path [24]. While this increases the area overhead, it provides the basis for energy efficient circuit realizations.

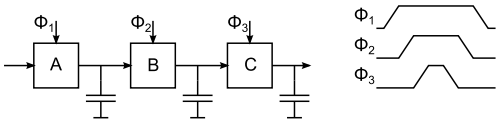


Fig. 4. Pipeline structure

III. Motivation and General Idea

While adiabatic circuits as reviewed above indeed provide a promising alternative to conventional CMOS-based solutions, a problem occurs when controlling the discharge path: If a gate evaluates to a logic 0, the load is never connected to the power supply and, thus, gets not charged. Accordingly, there is no need to discharge this load (moreover, the load is even not allowed to get discharged), because when the load is at potential 0 and the discharge path should be established, a potential difference between drain and source exists (since the power supply is still at Vdd level). This constitutes a violation of the first rule reviewed in Section II.B. In order to handle this problem, the discharge path has to be controlled. To this end, it is necessary to know whether the output is at 1 or at 0-level. This issue is illustrated using the example circuit in Fig. 5. Both gates, B1 and B2, represent a simple adiabatic buffer with a separate discharge path FB1 and FB2. Assuming the input is held constant at 1, B1 gets charged with Φ_1 and B2 with Φ_2, i.e. a pipeline structure is established. Since Q1 (the output of B1) acts as the input to B2, discharging of B1 must not happen until B2 evaluated its output Q2 (that is one phase delayed to B1). Hence the output Q2 is one phase delayed to Q1 and would be ideal to control the discharge path FB1 –at least from a timing point of view.

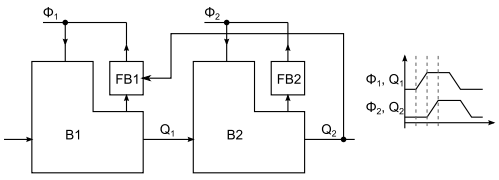


Fig. 5. Discharge control

If B2 is a more complex gate where the output Q2 not only depends on Q1, Q2 cannot directly control the discharge path FB1. Instead, Q1 must be reconstructed from Q2. This is only possible if the function of B2 is reversible. Fig. 6 sketches this situation: F is the function of B1 and G the function of B2.

The discharge path of B1 is marked with G^{-1} representing the inverted function of G to reconstruct Q1 from Q2.

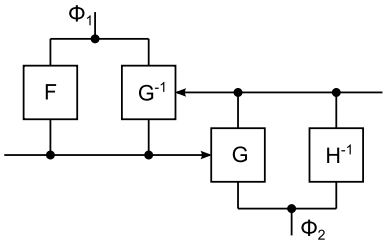


Fig. 6. Inverted feedback

Because of this, the full potential of adiabatic circuits can only be exploited if functions are realized whose results can be reverted. This holds for reversible circuits as reviewed in Section II.A. However, no real exploitation of reversible circuit design for the realization of adiabatic circuits has been proposed yet. To the best of our knowledge, only [5], [6], [7], [8] investigated this relation yet. But [5], [6] only realized standard functions such as adder and multiplier which, on top of that, realize the reversible circuits following the conventional computation paradigm (and, hence, loosing the benefit for adiabatic circuits). The same holds for [7], where reversible functions have been realized in a quasi-adiabatic logic style that, however, does not utilize the reversibility of the circuit. Finally, the concept of *Reversible Energy Recovery Logic* [8] represents the most promising exploitation of reversibility thus far. But here only single gates or very limited functionality has actually been realized.

In this work, we are aiming for overcoming these limitations and propose a methodology which is capable of realizing adiabatic circuits enhanced by reversibility for large functionality. To this end, the accomplishments in the design of reversible circuit design – in particular, the synthesis methodologies [9], [10], [11], [12], [13] – is exploited. They already provide the means to realize complex functionality in a logic fashion. Using that as a basis, corresponding adiabatic circuits can be created by:

- mapping the respective Toffoli gates (used in the reversible circuits obtained by these design methods) into corresponding technology cells,
- realizing the discharge path to support the resulting cells while, at the same time, satisfying the rules of adiabatic circuits, and
- adjusting clocking so that all phases are still conducted accordingly.

In the next section, how to accomplish these steps is described in detail. This leads to a synthesis flow (described in Section V) which allows to realize complex functionality while exploiting the full potential of adiabatic circuits.

IV. Implementation

A. Mapping Toffoli Gates to Technology Cells

The synthesis methods to be applied within the concept proposed above determine reversible circuits which are composed of Toffoli gates as their only gate type (see Section II-A). Hence, a technology cell for this gate type is required. Fig. 7 shows a corresponding proposal.

The PMOS transistors T_1 to T_6 realize the logic function, whereas T_7 separates the input from the output after the charging phase (so that the input can change to the next value). T_1 to T_7 represent the charging path used to energize the gate. Since the path is only used to charge and is only conducting if the output should evaluate to 1, it is sufficient to use PMOS transistors since the "strong 0" of the NMOS is not needed. Other adiabatic circuits (even the ones proposed in [8]) used full transmission gates thus far. The NMOS transistors T_8 to T_9 represent the discharging path that is controlled by the output of the following gate (T_8). Note that this discharge path is not part of the cell but is a cell by itself (highlighted by a dashed rectangle in Fig. 7 and discussed later in Section IV.B). T_9 is used to separate the output from the supply voltage (like T_7). Since the load should be discharged down to the ground level, NMOS transistors have been used for the discharge path. Due to switching activity and leakage effects, the output gets slightly charged even if it should result to 0. Unfortunately, this charge is not unloaded during the discharge phase, since the discharge path is only conducting if the output is 1. The charge accumulates over multiple 0-periods leading to glitches and a wrong interpretation of this output. A first idea to avoid this is to use a rail clamp transistor to ground. Unfortunately, the charge is then lost which increases the power consumption. Hence, instead of dissipating the charge to ground, we removed the connection to the ground and added the rail clamp transistor T_{10} to the inverted pulsing supply voltage (in our case, $\overline{\Phi}$ is inverted to Φ). By this, the unwanted charge can be drained from the output and is fed back to $\overline{\Phi}$. This results in an increased power consumption for this one cell alone since the charge is not fed back to the actual supply Φ. For the remaining circuit with multiple stages, the power consumption is reduced due to the fact that there will also be a cell driven by $\overline{\Phi}$ where the charge is fed back to Φ.

In Addition to the described circuit, the cell has a complementary circuit so that the cell evaluates both, Q and not Q. This is important since the inverted signals are needed for controlling the following stages. Furthermore, note that the described cell realizes a Toffoli gate with two control lines. Toffoli gates with more (or less) control lines can however be designed in a similar fashion.

B. Realizing the Discharge Path

Next, the discharge path for a cell F has to be realized. As discussed before by means of Fig. 6, this is conducted by realizing the inverted function G^{-1} of a cell G which follows F. Using the Toffoli cell as proposed in the previous sub-section, this can be realized (since Toffoli gates are self-inverse, cells for both, G and G^{-1}, are identical). In order to additionally satisfy the rule "Never turn on a transistor if there is a voltage difference between drain and source", the resulting realization is adjusted so that the output is to be discharged only if there is a charge to unload. Note that the realization of this discharge path using G^{-1} is not considered to be a part of cell F, but an own cell by itself (highlighted by a rectangle

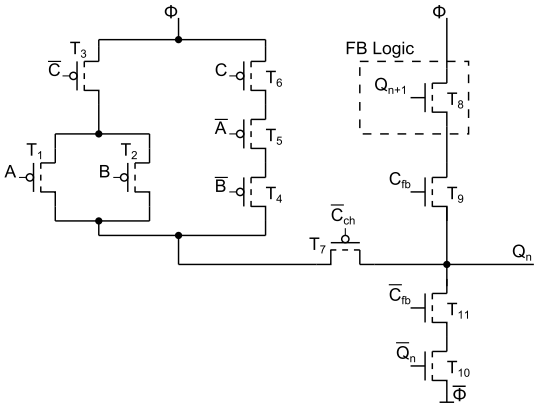


Fig. 7. Toffoli gate schematic

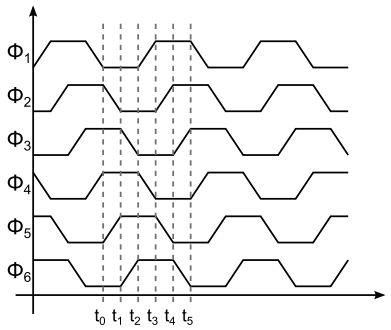


Fig. 8. Clocking

in Fig. 7). This allows to independently realize F and G^{-1} and to combine them when the actual synthesis is applied.

C. Adjusting the Clock

The above introduced cell operates in six phases and, therefore, six different clocks are required – each clock with a phase shifted 60 degrees compared to the previous clock. This yields a clocking scheme as sketched in Fig. 8.

These clocks are not only used as supply voltage but also as control signals in order to turn on and turn off the charge and discharge paths. In the following, the different phases of operations are explained by means of the circuit from Fig. 7 and the clocking scheme from Fig. 8. Let Φ be Φ_1 and its inverted counterpart Φ_4, then the inputs should change in phase with Φ_6 (one phases before Φ_1 rises to Vdd). T_7 should turn on two phases before Φ_1 rises, i.e. in phase with Φ_5. At timestep t_0, the charge path is activated – in phase with Φ_5. Since T_7 is a PMOS transistor, we need the inverted control signal Φ_2. The input signals will be coming from one stage before and, therefore, are arriving with clock Φ_6 at timestep t_1. At timestep t_2, Φ_1 charges the circuit and, at timestep t_3, the evaluation phase starts and the charge path deactivates since T_7 turns off. The fed back output signal from the following stage arrives at timestep t_3 (in phase with Φ_2). The discharge path is still separated from the output. Only at timestep t_4, T_9 gets activated with Φ_3 so that the discharge path is now established. At timestep t_5, Φ_1 drops back to the 0-level and discharges the circuit. This eventually leads to the following mapping for one cell:

- Φ_1: pulsed power supply
- Φ_2: $\overline{C}_{ch}$
- Φ_3: C_{fb}
- Φ_4: inverted pulsed power supply
- Φ_5: is not used
- Φ_6: $\overline{C}_{fb}$

V. Resulting Synthesis Flow

Using the solutions proposed above, an automated synthesis flow can be compiled which realizes the desired functionality as an adiabatic circuit which, due to the exploited reversibility, fully unleashes its potential. This section briefly describes the resulting flow and its step.

First, the desired functionality is realized in terms of a reversible logic circuit using existing synthesis methods such as [9], [10], [11], [12], [13]. This yields e.g. a gate netlist as shown in Fig. 9 realizing a full adder.

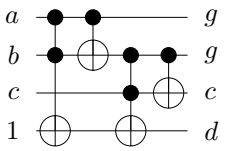


Fig. 9. Logic representation of full adder

Each gate in this circuit can be seen as a stage in the adiabatic pipeline. Because the feedback path of each cell depends on the following cell, the proposed synthesis flow iterates the netlist in reverse order from right to left. The next step is to insert output flipflops to every lane. This is necessary to buffer the output and to be able to control the feedback path of the last stage. The added flipflops are triggered by Φ_1.

Fig. 10 shows the resulting schematic for the full adder from Fig. 9. On the right hand-side marked with "step 1", the flipflops can be seen. Next, the previous entry of the netlist gets mapped; in this case, the Toffoli gate $T(\{b\}, c)$, whose corresponding cell is is highlighted with "step 2" in Fig. 10. Since the algorithm knows the following cell for this lane is a flipflop, a feedback cell with one input is added ("step 3"). Since the Toffoli cell only consists of the operation on the second input, the first input needs a buffer to be added ("step 4"). And also for this buffer, the corresponding feedback cell is added ("step 5"). These elements are one stage before the flipflops and, thus, are supplied with Φ_6.

In a similar fashion, all remaining gates of the circuit from Fig. 9 are realized. Note that, in the case a lane is not used in a stage, no buffer is inserted. Buffer insertion follows not until the lane is required later by another gate. This is e.g. the case for the first lane in Fig. 10 as the buffers a_1 and a_2 need to be inserted for stage Φ_5 and Φ_6 so that the output signal reaches the flipflop at the same phase as the other lanes do. For inserting these buffers two strategies are possible: On the one hand, buffers can be inserted for every stage in which the lane is not needed from the input of the design up to the output. Resulting in a full pipeline structure, this allows a throughput of signals with the same frequency as the pulsing power supplies Φ. This leads to a higher overhead due to

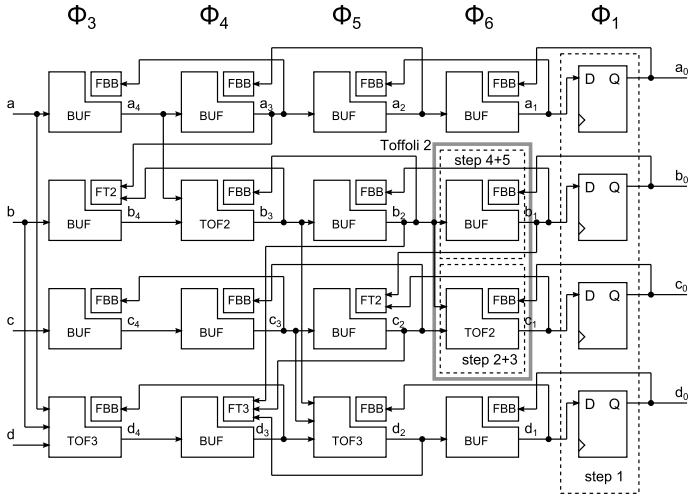


Fig. 10. Netlist of a full adder

the necessary buffers and, thus, a bigger design with higher power consumption. Alternatively, it is possible to only insert a number of buffers so that the actual stage gets the desired Φ (as depicted in Fig. 11). This results in less overhead, i.e. smaller area and less power consumption. On the downside, the input of the design needs to be stable during the entire signal flow through all stages – resulting in a throughput of $\frac{s}{6} \cdot p$, where s denotes the number of stages and p the period of Φ. However, this is only relevant for more complex designs with a big logic depth. While for the full adder example, no difference between the two variants exists. The experiments summarized in the next section assumed the second variant due to our goal of minimum power consumption.

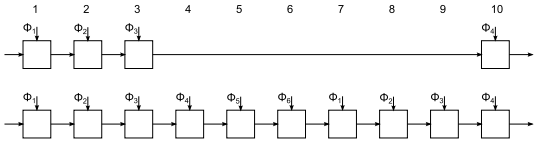


Fig. 11. Buffer insertion

VI. Experimental Evaluation

In order to evaluate the proposed solution, we implemented resulting synthesis flow including all steps in Pyhton. The correspondingly required Toffoli cells have been implemented using 180nm technology[2].

Since the simulation of complex designs with thousands of gates on the transistor level is rather inefficient, we decided to use an approach similar to digital design: The performance of each gate has been simulated solely and, afterwards, the resulting numbers are correspondingly multiplied with their occurrences in the design. But since the behavior of one gate depends on the surrounding gates, its load, and its drivers, one single simulation of a gate is not sufficient. In order to address this, all possible combinations of Toffoli cells, feedback paths, ascending cell preceding cell, etc. as well as all possible input combinations have been simulated using *Cadence Spectre Simulator* and accordingly incorporated in a power library file. After these simulations, a Python script has been used to iterate over the verilog netlist. For each cell in the netlist, the corresponding entry in the power library file is determined by analyzing its surroundings. From that, the respectively required energy consumption is determined. The resulting numbers with respect to power consumption (in [mW]), delay (in [ns]), and transistor count are provided in Table I. As benchmarks, we considered reversible circuits taken from *RevLib* [25] which have been realized using the reversible circuit design methods mentioned e.g. in Section II.A. In order to compare the obtained results to conventional solutions, the same functionality has additionally been realized using the *Synopsys Design Compiler* and a standard CMOS technology. Corresponding numbers are provided in Table I as well.

As can be seen from the table, the proposed design flow is superior in power consumption except for very small designs. This can be explained by the fact that it is not possible to regain all energy, since some energy is not fed back to the actual Φ but to the inverted one (as also discussed above). If the number of stages is small (below 6) and/or not a multiple of 6, the realization of a discharge path cannot be guaranteed. If the designs get larger, this effect becomes negligible. Besides that, it can clearly be seen that the delay and the transistor count significantly increases. This, however, is the expected trade-off for adiabatic circuits in general (as discussed in Section I and Section II) and of the proposed solution fully exploiting reversible logic as well as the full pipeline structure (as discussed in Section III and Section IV). This trade-off eventually results in the significant improvements in the power consumption – an eventual "killer-argument" for power-dependent devices. At this point it should again be mentioned that, with filling all stages with buffers, the drawback concerning timing can be countered. The design becomes a pipeline strcture with a certain delay but with considerable fast throughput. The throughput can happen at the frequency of the power clocks.

VII. Conclusions

In this work, we combined the accomplishments of two different research areas: the design of reversible logic as well as the design of adiabatic circuits. This was motivated by the fact that each adiabatic circuit can only fully exploit its potential, if the respectively realized gates are reversible. Although this has been known for a while, existing approaches to utilized this were not applicable to complex functionality. We proposed an adiabatic implementation of reversible Toffoli gates and, based on that, a new synthesis flow which tackles this problem. For the first time, complex functionality has been realized in a fully reversible and adiabatic fashion. Experiments demonstrated that this lead to circuits with significantly less power consumption than circuits realized with state-of-the-art conventional methods.

Acknowledgment

This work has partially been supported by the European Union through the COST Action IC1405.

[2]Note that the results obtained by this technology will remain valid with decreasing feature sizes as well [24].

TABLE I
EXPERIMENTAL RESULTS

Name	Power		Delay		Tr count	
	Rev	Std	Rev	Std	Rev	Std
4gt11_82	0.0060	**0.0017**	26.00	**0.12**	736	**5**
4gt11_83	0.0120	**0.0017**	18.00	**0.12**	448	**5**
4gt11_84	0.0197	**0.0017**	8.00	**0.12**	202	**5**
add16_174	**0.1656**	0.2861	20.00	**4.65**	5964	**386**
add16_175	**0.1257**	0.2861	6.00	**4.65**	1464	**386**
add32_183	**0.3784**	0.5792	20.00	**9.17**	11584	**770**
add32_185	**0.2500**	0.5792	**6.00**	9.17	2904	**770**
add64_184	**0.7610**	1.1654	20.00	**18.20**	23276	**1538**
add64_186	**0.4985**	1.1654	**6.00**	18.20	5784	**1538**
add8_172	**0.0807**	0.1407	20.00	**2.42**	2928	**194**
fredkin_6	**0.0234**	0.0234	8.00	**0.92**	158	**37**
gray6_47	**0.0103**	0.0196	12.00	**0.16**	366	**44**
gray6_48	**0.0103**	0.0196	12.00	**0.16**	366	**44**
ham3_102	**0.0106**	0.0175	12.00	**0.53**	238	**27**
ham3_103	**0.0144**	0.0175	8.00	**0.53**	142	**27**
ham7_106	**0.0089**	0.1236	38.00	**2.59**	1854	**165**
hwb4_52	**0.0182**	0.0757	22.00	**2.02**	630	**104**
miller_11	0.0194	**0.0166**	12.00	**0.65**	258	**33**
mod5d1	**0.0189**	0.0244	12.00	**0.42**	368	**43**
peres_9	**0.0099**	0.0117	6.00	**0.27**	94	**23**
rd32_272	0.0283	**0.0140**	12.00	**0.41**	386	**24**
rd53_138	**0.0101**	0.0444	16.00	**0.99**	824	**57**
rd73_140	**0.0281**	0.0863	24.00	**1.23**	1328	**92**
rd84_142	**0.0128**	0.1294	28.00	**1.59**	2188	**128**
sym9_146	**0.0253**	0.1327	30.00	**2.04**	1736	**138**
sym9_192	**0.0253**	0.1327	30.00	**2.04**	1736	**138**
toffoli_2	**0.0038**	0.0101	4.00	**0.27**	46	**19**
tof_db_4	**0.0169**	0.0175	6.00	**0.51**	126	**34**
urf1_149	**2.5129**	163.9984	17016.00	**2220.43**	1102728	**101949**

REFERENCES

[1] Y. Moon and D.-K. Jeong, "Efficient charge recovery logic," in *Symposium on VLSI Circuits, Digest of Technical Papers*, June 1995, pp. 129–130.

[2] M. L. Keote and P. T. Karule, "Design and implementation of energy efficient adiabatic ECRL and basic gates," in *International Conference on Soft Computing Techniques and Implementations*, Oct 2015, pp. 87–91.

[3] S. Younis and T. Knight, "Asymptotically zero energy computing using split-level charge recovery logic," *Technical Report AITR-1500, MIT AI Laboratory*, 1994.

[4] S. D. Kumar and S. K. N. Mahammad, "A novel adiabatic SRAM cell implementation using split level charge recovery logic," in *International Symposium on VLSI Design and Test*, June 2015, pp. 1–2.

[5] K. Babulu, M. Kamaraju, P. Bujjibabu, and K. Pradeep, "Design and implementation of H/W efficient multiplier: Reversible logic gate approach," in *International Conference on Communications and Signal Processing*, April 2015, pp. 1660–1664.

[6] H. Thapliyal and A. P. Vinod, "Transistor realization of reversible TSG gate and reversible adder architectures," in *Asia Pacific Conference on Circuits and Systems*, Dec 2006, pp. 418–421.

[7] G. Kumar and T. N. Sasamal, "Design and analysis of Toffoli gate using adiabatic technique," in *International Conference on Computing, Communication Automation*, May 2015, pp. 1344–1348.

[8] Y. Ye and K. Roy, "Energy recovery circuits using reversible and partially reversible logic," *Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 43, no. 9, pp. 769–778, Sep 1996.

[9] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 22, no. 6, pp. 710–722, 2003.

[10] D. Große, R. Wille, G. W. Dueck, and R. Drechsler, "Exact multiple control Toffoli network synthesis with SAT techniques," *IEEE Trans. on CAD*, vol. 28, no. 5, pp. 703–715, 2009.

[11] D. M. Miller, D. Maslov, and G. W. Dueck, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conf.*, 2003, pp. 318–323.

[12] P. Gupta, A. Agrawal, and N. K. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Trans. on CAD*, vol. 25, no. 11, pp. 2317–2330, 2006.

[13] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *Design Automation Conf.*, 2009, pp. 270–275.

[14] A. Zulehner and R. Wille, "Improving synthesis of reversible circuits: Exploiting redundancies in paths and nodes of QMDDs," in *Conf. on Reversible Computation*, 2017.

[15] R. Drechsler and R. Wille, "From truth tables to programming languages: Progress in the design of reversible circuits," in *International Symp. on Multi-Valued Logic*, 2011, pp. 78–85.

[16] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[17] A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, "Experimental verification of Landauer's principle linking information and thermodynamics," *Nature*, vol. 483, pp. 187–189, 2012.

[18] A. Zulehner and R. Wille, "Taking one-to-one mappings for granted: Advanced logic design of encoder circuits," in *Design, Automation and Test in Europe*, 2017, pp. 818–823.

[19] R. Wille, R. Drechsler, C. Osewold, and A. Garcia-Ortiz, "Automatic design of low-power encoders using reversible circuit synthesis," in *Design, Automation and Test in Europe*, 2012, pp. 1036–1041.

[20] R. Wille, O. Keszocze, S. Hillmich, M. Walter, and A. G. Ortiz, "Synthesis of approximate coders for on-chip interconnects using reversible logic," in *Design, Automation and Test in Europe*, 2016, pp. 1140–1143.

[21] L. G. Amarù, P. Gaillardon, R. Wille, and G. D. Micheli, "Exploiting inherent characteristics of reversible circuits for faster combinational equivalence checking," in *Design, Automation and Test in Europe*, 2016, pp. 175–180.

[22] R. Wille, E. Schönborn, M. Soeken, and R. Drechsler, "SyReC: A hardware description language for the specification and synthesis of reversible circuits," *Integration*, vol. 53, pp. 39–53, 2016.

[23] A. Zulehner and R. Wille, "Make it reversible: Efficient embedding of non-reversible functions," in *Design, Automation and Test in Europe*, 2017, pp. 458–463.

[24] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[25] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: an online resource for reversible functions and reversible circuits," in *International Symp. on Multi-Valued Logic*, 2008, pp. 220–225, RevLib is available at http://www.revlib.org.