Improving the Security of Dual-Rail Circuits

Danil Sokolov, Julian Murphy, Alex Bystrov, and Alex Yakovlev

School of Electrical, Electronic and Computer Engineering
University of Newcastle, Newcastle upon Tyne, UK
{danil.sokolov,j.p.murphy,a.bystrov,alex.yakovlev}@ncl.ac.uk

Abstract Dual-rail encoding, return-to-spacer protocol and hazard-free logic can be used to resist differential power analysis attacks by making the power consumption independent of processed data. Standard dual-rail logic uses a protocol with a single spacer, e.g. all-zeroes, which gives rise to power balancing problems. We address these problems by incorporating two spacers; the spacers alternate between adjacent clock cycles. This guarantees that all gates switch in each clock cycle regardless of the transmitted data values. To generate these dual-rail circuits an automated tool has been developed. It is capable of converting synchronous netlists into dual-rail circuits and it is interfaced to industry CAD tools. Dual-rail and single-rail benchmarks based upon the Advanced Encryption Standard (AES) have been simulated and compared in order to evaluate the method.

1 Introduction

Secure applications such as smart cards require measures to resist Differential Power Analysis (DPA). Dual-rail encoding provides a method to enhance the security properties of a system making DPA more difficult. As an example, in the design described in [1] the processor can execute special secure instructions. These instructions are implemented as dual-rail circuits, whose switching activity is meant to be independent from data. Whilst alternatives exist at the software level to balance power, the need at the hardware level is also mandatory. Special types of CMOS logic elements have been proposed in [2], but this low-level approach requires changing gate libraries and hence is costly for a standard cell or FPGA user. As a solution, using balanced data encoding such as dual-rail or together with self-timed design techniques has been proposed in [3,4].

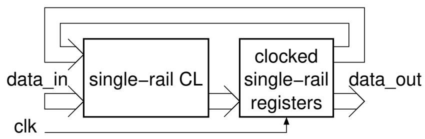
The clock signal is typically used as a reference in power analysis techniques. System "desynchronisation" as in [3,5] can help hide the clock signal. To mask the operation of a block of logic is a much more complex task which could demand very expensive changes to the entire design flow. A cheaper desynchronisation method to rebuild individual blocks within the same synchronous infrastructure so, that their power signatures become independent from the mode of operation and from the data processed. This method is used in [5], where synchronous pipelines are transformed into asynchronous circuits using dual-rail coding.

These desynchronisation methods represent a combination of two aspects of security: reference signal hiding and balancing of data encoding w.r.t. switching activity. In this paper we separate these aspects, concentrating on data encoding only.

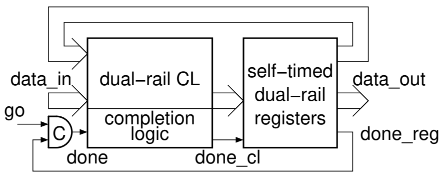
Our idea is to replace blocks in existing architectures dominated by synchronous single-threaded CPU cores and their slow buses, having no pipelining or concurrency, with secure and hazard free dual-rail circuits. Using the standard dual-rail protocol with a single spacer still has certain balancing problems due to the asymmetry between logic gates within a dual-rail gate. In this paper we address and solve these problems by using a new protocol with two spacers alternating in time; leading to all gates switching within every clock cycle. This is the first contribution of the paper.

The other idea is to stay as close to the standard industry design flow as possible. Our method is applied via an automated tool to a clocked single-rail netlist obtained by standard RTL synthesis tools from a behavioural specification. Such circuits have an architecture depicted in Figure 1(a). The result is also a netlist which can be simulated and passed to the back-end design tools. Furthermore, all DFT (Design For Testability) features incorporated at the logic synthesis stage are preserved in our approach.

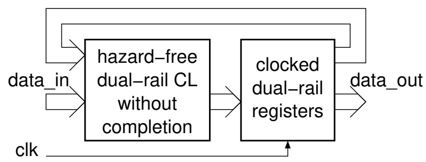
The resultant dual-rail circuit can be built in either of two architectures: *self-timed dual-rail* or *clocked dual-rail*, Figure 1(b, c) respectively.



(a) Single-rail architecture (standard RTL design)



(b) Self-timed dual-rail architecture



(c) Clocked dual-rail architecture

Figure 1. Design architectures

Self-timed dual-rail circuits do not have a clock and their registers are controlled by a completion signal formed in the completion detection logic. Being asynchronous, these circuits should exhibit better throughput, but they suffer from a significant size overhead due to additional logic from completion detection.

Clocked dual-rail circuits do not have completion detection logic and rely on the assumption that the hazard-free dual-rail combinational logic switches by the end of the clock period. In our method this assumption is easy to meet, because the delay characteristics of the dual-rail circuit are inherited from single-rail prototype.

While the method and the tool support both dual-rail architectures, in this paper we concentrate on the latter one. The security aspects of system level, memory elements, buses, etc. also do not belong to the focus of the paper. We are looking at security of logic circuits only.

The rest of the paper is organised as follows. Firstly the theory of applying dual-rail coding to synchronous circuits using a single spacer and two spacers is described, then the operation of the tool is discussed. The AES benchmark results and potential improvements follow and finally the conclusions are presented.

2 Method

2.1 Single spacer dual-rail

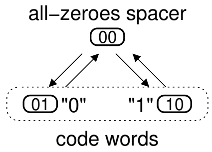
Dual-rail code uses two rails with only two valid signal combinations $\{01, 10\}$, which encode values 0 and 1 respectively. Dual-rail code is widely used to represent data in self-timed circuits [6,7], where a specific protocol of switching helps to avoid hazards. The protocol allows only transitions from all-zeroes $\{00\}$, which is a non-code word, to a *code word* and back to all-zeroes as shown in Figure 2(a); this means the switching is monotonic. The all-zeroes state is used to indicate the absence of data, which separates one code word from another. Such a state is often called a *spacer*.

An approach for automatic converting single-rail circuits to dual-rail, using the above signalling protocol, that is easy to incorporate in the standard RTL-based design flow has been described in [5]. Within this approach, called Null-Convention Logic [8] one can follow one of two major implementation strategies for logic: one is with full completion detection through the dual-rail signals (NCL-D) and the other with separate completion detection (NCL-X). The former one is more conservative with respect to delay dependence while the latter one is less delay-insensitive but more area and speed efficient. For example, an AND gate is implemented in NCL-D and NCL-X as shown in Figure 2(b,c) respectively. NCL methods of circuit construction exploit the fact that the negation operation in dual-rail corresponds to swapping the rails. Such dual-rail circuits do not have negative gates (internal negative gates, for example in XOR elements, are also converted into positive gates), hence they are race-free under any single transition.

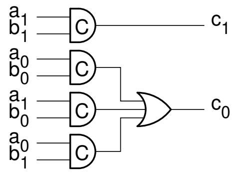
If the design objective is only power balancing (as in our case), one can abandon the completion detection channels, relying on timing assumptions as in standard synchronous designs; thus saving a considerable amount of area and power. This approach was followed in [9], considering the circuit in a clocked environment, where such timing assumptions were deemed quite reasonable to avoid any hazards in the combinational logic. Hence, in the clocked environment the dual-rail logic for an AND gate is simply a pair of AND and OR gates as shown in Figure 2(d).

The above implementation techniques certainly help to balance switching activity at the level of dual-rail nodes. Assuming that the power consumed by one rail in a pair is the same as in the other rail, the overall power consumption is invariant to the data bits propagating through the dual-rail circuit. However, the physical realisation of the rails at the gate level is not symmetric, and experiments with these dual-rail implementations show that power source current leaks the data values.

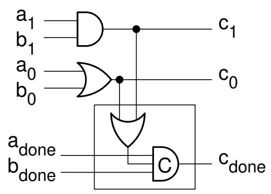
For example, in the structure in Figure 2(d) we compare the gate switching profiles when computing two different binary sequences of values c for corresponding input sequences on a and b. The first sequence is $a = 0000$, $b = 1111$, $c = 0000$, and the second sequence is $a = 1111$, $b = 1111$, $c = 1111$. The switching profile of these sequences at the level of gates is different: in the first sequence there are eight firings of OR gate and in the second there eight firings of AND (note that we counted both *spacer*→*code word* and *code word*→*spacer* phases).



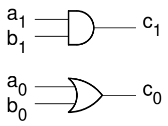
(a) Dual-rail protocol



(b) NCL-D AND gate



(c) NCL-X AND gate



(d) NCL in clocked environment

Figure 2. Single spacer dual-rail

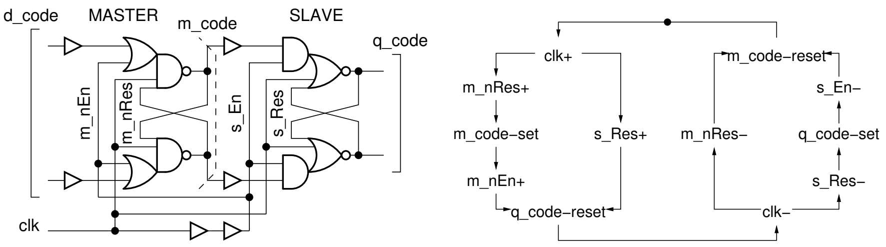
While there could be ways of balancing power consumption between individual gates in dual-rail pairs by means of modifications at the transistor level, adjusting loads and changing transistor sizes, etc., all such measures are costly. The standard logic library requires finding a more economic solution. We do not consider randomisation techniques in this paper as they can be applied independently, and possibly in conjunction with our method.

Synchronous flip-flops are built to be power efficient, so if they switch to the same value (data input remains the same within several clocks) then nothing changes at the output. The absence of the output transition saves power, but in the same time it makes the power consumption data dependent. In order to avoid this, we make flip-flops operate in the return-to-spacer protocol as in Figure 2(a). The solution in Figure 3(a) uses the master-slave scheme, writing to the master is controlled by the positive edge of the clock and writing to the slave is controlled by the negative edge. At the same time the high value of the clock enforces slave outputs into zero (output spacer as in Figure 2(a)) and the low clock value enforces master outputs into one (a similar spacer for the logic with active zero).

This circuit operates as explained in Figure 3(b). Both master and slave latches have their respective reset and enable inputs (active zero for the master). The delay between removing the reset signal and disabling writing for each latch (hold time) is formed by the couple of buffers in the clock circuit. Buffers between master and slave are needed to delay *m_code-set* value until *s_En-*. The advantage of this implementation is the use of a single cross-coupled latch in each stage for a couple of input data signals.

2.2 Dual spacer dual-rail

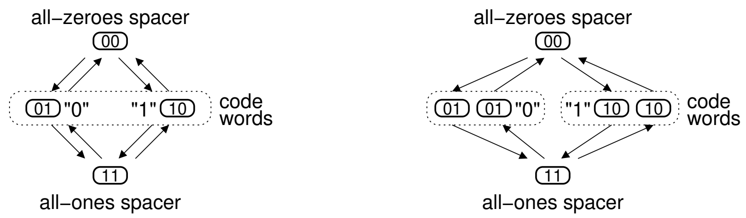
In order to balance the power signature we propose to use two spacers (i.e. two spacer states, $\{00\}$ for *all-zeroes spacer* and $\{11\}$ for *all-ones spacer*), resulting in



(a) Schematic (b) Signal transition graph

Figure 3. Single spacer dual-rail flip-flop

a dual spacer protocol as shown in Figure 4. It defines the switching as follows: *spacer→code word→spacer→code word.* The polarity of the spacer can be arbitrary and possibly random as in Figure 4(a). A possible refinement for this protocol is the *alternating spacer protocol* shown in Figure 4(b). The advantage of the latter is that all bits are switched in each cycle of operation, thus opening a possibility for perfect energy balancing between cycles of operation.

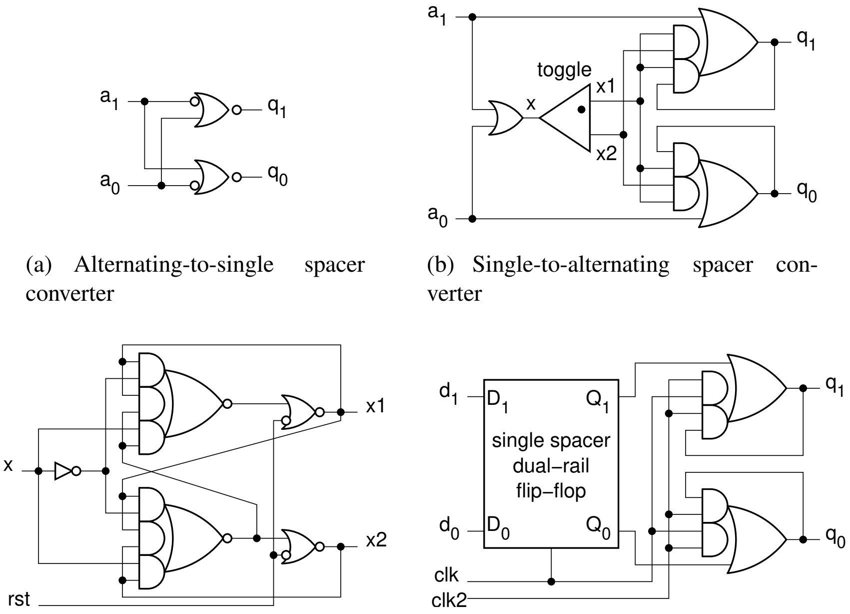


(a) Arbitrary order of spacers (b) Alternating spacers

Figure 4. Dual-spacer dual-rail protocol

As opposed to single spacer dual-rail, where in each cycle a particular rail is switched up and down (i.e. the same gate always switches), in the alternating spacer protocol both rails are switched from *all-zeroes spacer* to *all-ones spacer* and back. The intermediate states in this switching are *code words*. In the scope of the entire logic circuit, this means that for every computation cycle we always fire all gates forming the dual-rail pairs. In [10] we introduce two security characteristics of a circuit w.r.t. DPA attacks: *imbalance* and *exposure time*. By imbalance we mean the variation in power consumption when processing different data values. Exposure time is the time during which the imbalance is exhibited. Our experiments show that the worst case imbalance in a dual-rail circuits under a realistic load is 2.1%. The worst case exposure time depends on the spacer protocol. It is up to the whole time of circuit operation if the single spacer protocol is used. By using of the alternating spacer protocol the exposure time is reduced to less than one clock cycle, which makes the circuit more resistant to DPA.

The new alternating spacer discipline cannot be directly applied to the implementation techniques shown in Figure 2(b,c). Those, both in the logic rails as well as in completion detection assume the fact that for each pair of rails, the $\{11\}$ combination never occurs. In fact the use of *all-ones spacer* would upset the speed-independent implementation in Figure 2(b), because the outputs of the second layer elements would not be acknowledged during *code word→all-ones spacer* transition. The completion detection for those gates can of course be ensured by using an additional three-input C-element, but this extra overhead would make this implementation technique much less elegant because of the additional acknowledgement signal channel. In the single spacer structure, due to the principle of orthogonality (one-hot) between min-terms $a_0 \cdot b_0$, $a_1 \cdot b_0$ and $a_0 \cdot b_1$, only one C-element in the rail c_0 fires per cycle.



(a) Alternating-to-single spacer converter

(b) Single-to-alternating spacer converter

(c) Toggle

(d) Alternating spacer flip-flop

Figure 5. Alternating spacer converters

If some parts of a dual-rail circuit operate using the single spacer and the others the alternating spacer protocol, then spacer converters should be used. The alternating-to-single spacer converter shown in Figure 5(a) is transparent to *code words* and enforces *all-zeroes spacer* on the output if the input is all-ones or all-zeroes.

The implementation of a single-to-alternating spacer converter, Figure 5(b), uses a toggle to decide which spacer to inject all-ones or all-zeroes. The toggle can be constructed out of two latches as shown in Figure 5(c). It operates in the following way: $x+ \to x1+ \to x- \to x2+ \to x+ \to x1- \to x- \to x2-$, i.e. $x1$ changes on positive edge of x, and $x2$ switches on its negative edge. The frequency of $x1$ and $x2$ is half the frequency of x.

The alternation of spacers in time is enforced by flip-flops. The alternating spacer flip-flop can be built combining a single spacer dual-rail flip-flop with a single spacer to alternating spacer converter. The power consumption of the single spacer dual-rail flip-flop is data independent due to the symmetry of its rails. The rails of the spacer converter are also symmetric, which makes the power consumption of the resultant alternating spacer flip-flop data independent. The optimised version of such a flip-flop (toggle is moved outside) is depicted in Figure 5(d). This implementation uses $clk2$ signal to decide which spacer to inject on the positive phase of clk. The signal $clk2$ changes on the negative edge of the clock and is formed by a toggle (one for the whole circuit) whose input is clk. The timing assumption for $clk2$ is that it changes after the output of single spacer flip-flop. Both, the slave latch of the single spacer flip-flop and the toggle which generates $clk2$ signal, are triggered by the negative edge of clk. The depth of logic in the toggle is greater than in the slave latch of the flip-flop. At the same time $clk2$ goes to all flip-flops of the circuit and requires buffering, which also delays it. This justifies our timing assumption.

It should be mentioned that the inputs of the dual-rail circuit must also support the alternating spacer protocol. Moreover, the same spacer should appear each cycle on the inputs of a dual-rail gate. That means the spacer protocol on the circuit inputs and flip-flop outputs must be synchronised in the reset phase.

2.3 Negative gate optimisation

In CMOS a positive gate is usually constructed out of a negative gate and an inverter. That is why the total area overhead in dual-rail logic is more than twofold comparing to single-rail. Use of positive gates is not only a disadvantage for the size of dual-rail circuit, but also for the length of the critical path. Our method for negative gate optimisation [9] is described in this section.

If the *all-zeroes spacer* of the dual-rail code is applied to a layer of negative gates (NAND, NOR, AND-NOR, OR-NAND), then the output will be *all-ones spacer*. The opposite is also true: *all-ones spacer* is converted into *all-zeroes spacer*. The polarity of signals within *code words* remains the same if the output rails are swapped.

The spacer alternation between odd and even layers of combinational logic can be used for *negative gate optimisation* of dual-rail circuits. The optimised circuit uses either *all-ones spacer* or *all-zeroes spacer* in different stages (the spacer changes between the layers of logic) as captured in Figure 6.

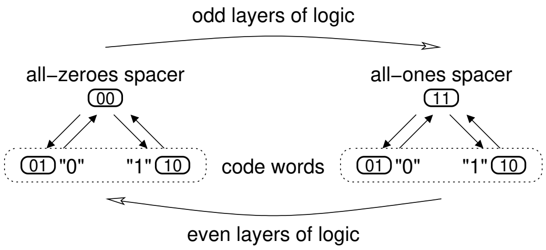
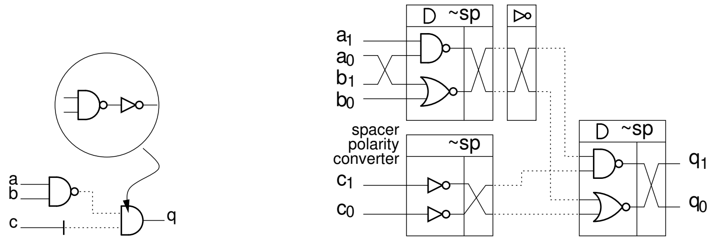


Figure 6. Spacer polarity after logic optimisation

In order to optimise a dual-rail circuit for negative gates the following transformations should be applied. First, all gates of positive dual-rail logic are replaced by negative gates. Then, the output rails of those gates are swapped. Finally, *spacer polar-*

ity converters are placed at the wires that connect the layers of logic of the same parity (odd-to-odd or even-to-even).



(a) Single-rail circuit (b) Negative gate dual-rail circuit

Figure 7. Constructing negative gate dual-rail circuit

Consider negative gate optimisation using a simple example shown in Figure 7(a). Dotted lines in the single-rail circuit indicate signals which will be mapped into the dual-rail with the *all-ones spacer*. The bar on the wire is the location of a spacer polarity converter. The circuit in Figure 7(b) is obtained by replacing gates by their dual-rail versions. These gates are built from traditional positive dual-rail gates by adding signal inversion to their outputs and swapping the output rails (the latter is needed to preserve the polarity of signals in the output code words). The operation of negation is implemented by a rail swapping and does not require any logic gates. The spacer polarity converter is implemented as a pair of inverters having their outputs crossed in order to preserve the polarity of signals in the output code words. It is possible to combine such an optimisation with the alternation of spacers in time.

This section presented the application of dual-rail coding to the building of secure circuits, whose power consumption is independent of the data they process. An extension of the dual-rail protocol was presented, namely *dual spacer in time* (alternating spacer protocol). This aims at power balancing by switching all gates in each cycle of circuit operation. The negative gate optimisation was applied to the circuits implementing such a protocol.

3 Tool Description

The described conversion procedure of single-rail into dual-rail circuit has been implemented as a software tool named the "Verimap design kit". It successfully interfaces to the Cadence CAD tools. It takes as input a structural Verilog netlist file, created by Cadence Ambit (or another logic synthesis tool), and converts it into dual-rail netlist. The resulting netlist can then be processed by Cadence or other EDA tools.

The structure of our Verimap design kit is displayed in Figure 8. The main parts are the tool itself and two libraries. The *library of gate prototypes* contains the description of gates used in the input netlist. It facilitates the structural analysis of the input netlist. The *library of transformation rules* defines: complementary gates needed for construction of the dual-rail logic, the polarity of gate inputs and outputs and specifies if the corresponding dual-rail gate requires completion signal (for asynchronous design only) and if it inverts the spacer. If a predefined dual-rail implementation of a gate is found in the library the tool uses it, otherwise an implementation is built automatically using the rules.

The main function of the tool is conversion of single-rail RTL netlist into dual-rail netlist of either of two architectures: self-timed and clocked, Figure 1(b, c) respectively. It is done in four stages. First, a single-rail circuit is converted into positive logic dual-rail. Second, the positive dual-rail gates are replaced by negative dual-rail gates and the spacer polarity inverters are inserted. Then, the completion signal is generated (asynchronous design only). Finally, a wrapper module connecting the dual-rail circuit to the single-rail environment is added (optional).

Apart from generating netlists, Verimap tool reports statistics for the original and resultant circuits: estimated area of combinational logic and flip-flops, number of negative gates and transistors, number of wires.

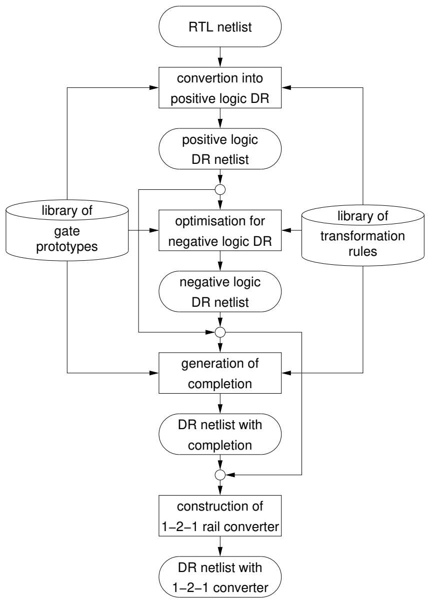


Figure 8. Verimap design kit

The tool also generates a behavioural Verilog file assisting the power analysis of the original and resultant circuits. Being included into simulation testbench these Verilog counts the number of switching events in each wire of the circuits.

4 Benchmark Results and Future Improvements

This section summarises the experiments performed to characterise the proposed method in terms of security, size and power consumption. Two AES designs were used: *Open core AES* and *AES with computable sbox* (for details see Appendix A). For each design a single-rail AES circuit was synthesised from RTL specification by using Cadence Ambit v4.0 tool and AMS-0.35μ library. Our Verimap tool was applied to the netlist generated by Ambit and the dual-rail netlist was produced. The dual-rail circuits were optimised for negative gates and used alternating spacer dual-rail protocol. Both, single-rail and dual-rail designs were analysed for static delays (SDF delay annotation) and simulated in Verilog-XL v3.10. By keeping to the RTL design flow the netlists can be directly used in the back-end design tools of Cadence.

The statistics for the parts of AES, namely *ciphers* and *sboxes,* are shown in Table 1, Table 2 and Table 3.

The purpose of the first experiment was to evaluate the correlation between data and switching activity of the circuits. Switching activity is the number of switching events in the circuit within one clock cycle. Table 1 presents the minimum, average and maximum switching activity for the sboxes and ciphers. These values were obtained by

benchmark name		switching activity (hazards)		
		min	avg	max
sbox (open core)	single-rail	0 (0)	162 (33)	277 (124)
	dual-rail	1,180	1,180	1,180
	overhead	∞	628%	326%
sbox (comp.)	single-rail	0 (0)	525 (345)	936 (746)
	dual-rail	868	868	868
	overhead	∞	65%	-17%
cipher (open core)	single-rail	0	9,147	13,236
	dual-rail	41,285	41,285	41,285
	overhead	∞	351%	211%
cipher (comp.)	single-rail	0 (0)	3,810 (2,013)	6,140 (3,682)
	dual-rail	13,055	13,055	13,055
	overhead	∞	242%	112%

Table 1. Switching activity

simulating the circuits with a number of input vectors: 10,000 random input vectors for sboxes and 284 standard AES testbench vectors for ciphers.

The experiment shows a significant difference between the min/average/max switching activity values for the single-rail sbox benchmarks. The minimum value is zero, and the maximum values are up to 48% higher than the average values. At the same time, switching activity for the dual-rail circuits is constant. In the single-rail switching activity varies significantly depending on data and clearly they exhibit zero switching activity if the input data does not change. In addition many switching events in single-rail circuits are caused by hazards and the single-rail sbox benchmarks are no exception. Here the hazards caused up to 80% of data dependent switching events. The number of switching events in dual-rail combinational logic is constant for any input data and is equal to the number of wires. It shows that all gates are switching in each clock cycle, thus making power consumption data independent (described in detail in Section 2.2).

The single and dual-rail implementations of the AES ciphers were simulated and compared against combinational logic blocks (sboxes). Switching activity in the open core dual-rail cipher is 351% higher than in the single-rail cipher and 255% higher for the AES design with computed sboxes. These values are greater than the results for the corresponding sboxes. The increased difference can be explained by the nature of computations in complex circuits. They execute in bursts defined by their algorithms. Under a burst the switching is similar to our experiments with combinational circuits. However, between the bursts the situation is significantly different: a single-rail circuit is inactive and a dual-rail circuit continues to ‘burn power’ by switching between code words and spacers.

A possible way to address this issue is to implement *clock gating*. This, however, should be different from the conventional clock gating technique. It is important to make it data independent. At this stage we do not see a feasible way of implementing this at the netlist level. Most likely it will require analysis of behavioural specifications. We view this idea as a subject of future work.

benchmark name		switching activity	
		single spacer	alternating spacer
cipher (encryption)	rail_1	8,388	6,505
	rail_0	4,622	6,505
	disbalance	29%	0%
cipher (decryption)	rail_1	8,572	6,505
	rail_0	4,438	6,505
	disbalance	32%	0%

Table 2. Switching activity in dual-rail rails

In order to compare the security features of single spacer and alternating spacer circuits, the AES design with computable sboxes was also converted into single spacer dual-rail. Both, single spacer and alternating spacer dual-rail implementations were simulated with 284 input vectors from the standard AES testbench in the encryption and decryption modes. The switching activities of "1" and "0" rails were recorded separately. Table 2 shows the worst case difference in switching activity between "1" and "0" rails. The imbalance between the number of switching events in the rail_1 and rail_0 is calculated as $disbalance = \frac{|rail\_1 - rail\_0|}{rail\_1 + rail\_0} \cdot 100\%$. While the total switching activity is the same in both implementations, the single spacer implementation exhibits significant differences in the number of switching events on the complementary rails. As the complementary gates within a dual-rail gate have different power consumptions, the power signature of the single spacer dual-rail circuit becomes dependent on the processed data. Alternating spacer dual-rail circuits do not suffer from this leakage because all gates are switching in every clock cycle.

benchmark name		negative gate count (comb. logic)	transistor count (comb. logic)	wire count	estimated area	
					comb. logic	flip-flops
sbox (open core)	single-rail	655	3,180	482	44,593	0
	dual-rail	1,523	6,672	1,180	101,364	0
	overhead	133%	110%	145%	127%	0
sbox (comp.)	single-rail	634	2,362	400	32,975	0
	dual-rail	1,164	4,628	868	68,603	0
	overhead	84%	96%	117%	108%	0
cipher (open core)	single-rail	12,752	68,184	9,980	873,175	142,370
	dual-rail	26,396	139,828	24,367	1,925,190	466,870
	overhead	107%	105%	144%	120%	228%
cipher (comp.)	single-rail	10,372	50,344	5,936	580,046	118,678
	dual-rail	19,510	95,066	13,055	1,237,260	462,021
	overhead	88%	89%	120%	113%	289%

Table 3. Circuit size

The cost of improved security features is the increase in the number of gates, wires and area, see Table 3.

The benchmarks indicate only 84-88% overhead in gate numbers (a positive gate is counted as a pair of a negative gate and an inverter) for AES design with computable sboxes. This is less than 100% due to the negative gate optimisation. For Open core design the overhead is more than 100% due to the structure of its sbox module. During the negative logic optimisation of Open core sbox more inverters were inserted into not-critical path (as components of spacer inverters) than removed from the critical path.

The number of wires is increased by 117-145%. Wires are duplicated in a dual-rail circuit and then spacer converters are added, further increasing the number of wires.

The estimated area of the benchmarks combinational logic indicates a 102%-127% overhead. A significant area increase for flip-flops (228%-289%) can be explained by using dual-rail flip-flops constructed out of standard logic gates. This can be improved by transistor level optimisation of the flip-flops.

It is clear that in the AES designs there are opportunities to minimise power consumption as not all logic is necessarily being used all the time. Industry synthesis tools can identify sleep mode logic and use this information to annotate places in the netlist which could be committed to sleep mode logic later in the design flow. This low power optimisation could be utilised in our dual-rail circuitry, one approach would be to put a spacer on the input to the identified sleep mode logic and holding this there for the clock cycles whilst it is not used. By doing so the switching is now zero, thus saving power. This technique would not reveal data as the sleep mode logic is in a “meaningless” spacer state. By using the synthesis tool to identify the sleep mode logic we are adhering to the RTL design flow and our conversion tool could use the annotated netlist to apply the optimisation to dual-rail circuits; note the committal stage of the sleep mode logic would need to be different to what the synthesis tools would do (simple AND gates using a control signal). Presently this has not been implemented in the tool but investigated using schematic entry with simple examples which gave promising results. This needs to be investigated further together with the clock gating idea.

5 Conclusions

We have presented a technique for improving resistance to DPA attacks at the hardware level by power balancing in a deterministic way. The power consumption within each cycle of operation is constant. Our technique uses two spacers alternating in time within the dual-rail logic framework. It is very cheap yet effective and is supported by software tools that interface to standard RTL design flow tools used by most ASIC designers. The idea of using two spacers is deemed particularly efficient for dual-rail logic, where the Hamming distance between each spacer and a valid combination is the same. While it can still be used without too much overhead in optimally balanced k-of-n codes (e.g. 3-of-6) it would be much less efficient in other popular codes such as 1-of-4 [11].

The AES benchmarks indicate that we have fully eliminated the dependency which existed between data and switching activity in the dual-rail circuits. The price to pay for the improved security features is the increased average switching activity and area overheads.

Acknowledgements: We are grateful to A.Koelmans, A.Kondratyev, S.Moore, A.Taubin for useful comments. EPSRC supports this work via GR/R16754 (BESST), GR/S81421 (SCREEN)

References

1. H.Saputra, N.Vijaykrishnan, M.Kandemir, M.J.Irwin, R.Brooks, S.Kim, W.Zhang: "Masking the energy behaviour of DES Encryption". Proc. DATE'03, Munich, Germany, March 2003.
2. K.Tiri, M.Akmal, I.Verbauwhede: "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards". Proc. ESSCIRC 2002.
3. S.Moore, R.Anderson, P.Cunningham, R.Mullins, G.Taylor: "Improving smart card security using self-timed circuits". ASYNC'02, 2002, pp. 211–218.
4. Z.Yu, S.Furber, L.Plana: "An investigation into the security of self-timed circuits". Proc. of ASYNC'03, Vancouver, May 2003, IEEE CS Press, pp. 206–215.
5. A.Kondratyev, K.Lwin: "Design of asynchronous circuits using synchronous CAD tools". Proc. DAC'02, New Orleans, USA, 2002, pp. 107–117.
6. V.Varshavsky (editor): "Self-timed control of concurrent processes" Kluwer, 1990 (Russian edition 1986).
7. I.David, R.Ginosar, M.Yoeli: "An efficient implementation of boolean functions as self-timed circuits". *IEEE Trans. on Computers*, 1992, 41(1), pp. 2–11.
8. K.Fant, S.Brandt: "Null Convention Logic: a complete and consistent logic for asynchronous digital circuit synthesis". Proc. Int. Conf. Application-Specific Systems, Architectures and Processors (ASAP'96), IEEE CS Press, Los Alamos, Calif., 1996, pp. 261–273.
9. A.Bystrov, D.Sokolov, A.Yakovlev, A.Koelmans: "Balancing Power Signature in Secure Systems". 14th UK Asynchronous Forum, Newcastle, June 2003.
10. D.Sokolov, J.Murphy, A.Bystrov, A.Yakovlev: "Improving the security of dual-rail circuits", Technical report, Microelectronic System Design Group, School of EECE, University of Newcastle upon Tyne, April 2004, http://www.staff.ncl.ac.uk/i.g.clark/async/tech-reports/NCL-EECE-MSD-TR-2004-101.pdf
11. W.Bainbridge, S.Furber: "Delay insensitive system-on-chip interconnect using 1-of-4 data encoding". In Proc. ASYNC'01, March 2001.
12. S.Mangard, M.Aigner, S.Dominikus: "A Highly Regular and Scalable AES Hardware Architecture". IEEE Trans. On Computers, 2003, 52(4), pp. 483–491
13. J.Daemen, V. Rijmen: "The Design of Rijndael". Springer-Verlag, 2002
14. National Institute Of Standards and Technology: "Federal Information Processing Standard 197, The Advanced Encryption Standard (AES)". http://csrc.nist.gov/publications/fips/fips197/fips197.pdf, 2001.
15. R.Usselmann: "Advanced Encryption Standard / Rijndael IP Core". http://www.asic.ws/.
16. J.Wolkerstorfer, E.Oswald, M.Lamberger: "An ASIC implementation of AES S-Boxes", Topics in Cryptology RSA'02, Proc. RSA Conf. 2002, Feb 2002.

A Advanced Encryption Standard

The symmetric block cipher Rijndael [13] was standardised by NIST as the Advanced Encryption Standard (AES) [14] in November 2001 as the successor to DES. The algorithm is a block cipher that encrypts/decrypts blocks of 128, 192, or 256 bits, and uses symmetric keys of 128, 192 or 256 bits. It consists of a sequence of four primitive functions, SubBytes, ShiftRows, MixColumns and AddRoundKey called a round. A round is executed 10, 12 or 14 times depending on the key and plain text lengths. Before the rounds are executed the AddRoundKey function is applied for initialisation plus the last round omits the MixColumns operation. A new key is derived for each round from the previous key.

For decryption the procedure is reversed and inverse versions of the aforementioned functions are applied, excluding AddRoundKey, this has no inverse.

A detailed explanation of each function can be found in [12,13]. For clarity the SubBytes function performs a non linear transformation using byte substitution tables (Sboxes), each Sbox is a multiplicative inversion in GF(256) followed by an affine transformation.

Both designs were synthesised from a RTL Verilog specification using the Cadence Ambit v4.0 tool and AMS-0.35μ library. A brief description of the two architectures follows.

A.1 Open core AES architecture

This design operates on 128 bits and has two separate 'sub-cores' one for encryption and the other for decryption; they share the same type of key generation module [15] and initial permutation module, however separate instances exist inside each sub-core. The core is shown in Figure 9; each sub core has 16 inverse/S-boxes inside the round permutation module. The initial permutation modules simply perform the AddRoundKey function and the round permutation modules loops internally to perform the 10 rounds and the final permutation module performs the last round. For this yields a complete encryption in 12 clock cycles. The decryption core consists of 16 inverse S-boxes these differ from the S-boxes used for encryption. The key reversal buffer stores keys for all the rounds and these are presented to the round permutation module each round in reverse order. Using this principle a complete decryption can be performed in 12 clock cycles. It must be highlighted that since the keys are used in reverse order - the initial key must be first expanded 10 times to get the last key, taking 10 extra clock cycles. In this design the Inv/SubBytes transformations (sboxes) are hardwired instead of being computed on the fly or stored in a ROM. This can be seen as simply a large decoder. The sub-cores both have 128 pins for plain/cipher text and 128 pins for the key and miscellaneous control pins and logic.

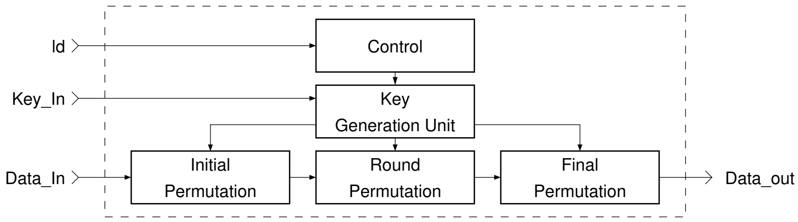


Figure 9. Open core AES

A.2 AES with computed sboxes architecture

This architecture combines encryption and decryption into one core working on 128 bits. The designs' basis is taken from [12], it was chosen due to its structure namely: it is highly regular (this keeps the layout small), it has short balanced combinational paths, hardware reuse for encryption and decryption which yields a small area and finally it has a 32 pin interface for the data (128 pin for the key) and shared computed sboxes.

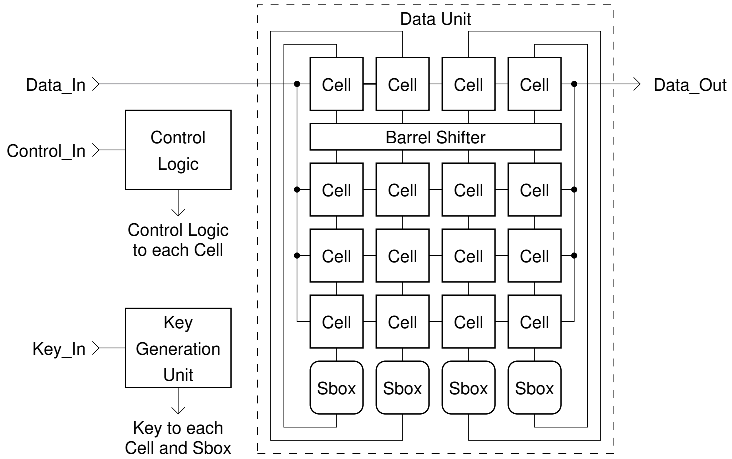


Figure 10. AES with computed sboxes

The design consists of a key generation unit, control logic and a data unit incorporating 16 data cells, 4 sboxes (these perform the sbox and inverse sbox unlike the open core design) and a barrel shifter. The core is displayed in Figure 10, since the core does both encryption and decryption the diagram summarises both.

The data unit can perform any of the AES round functions and uses the key provided by the key unit for the AddRoundKey function. A single data cell comprises of a register, a GF(256) multiplier [12], a bank of XOR gates, and an input selection multiplexer. Additional multiplexers are included to enable the required function to be selected.

The Sboxes are able to perform either the Sbox transformation or the inverse Sbox transformation taking two clock cycles to compute a result due to a two-stage pipeline. Whilst the Sboxes are not used by the data unit (the MixColumns operation) the key generation unit takes advantage of this to generate the next key. The Sbox is computed by reducing the computation to GF(16) and GF(16) arithmetic and then applying the affine transformation as illustrated in [16].

Since the design has a 32 pin interface for the data, four clock cycles are required to clock the plain text, or cipher text into the data unit, and the same number to retrieve the data. After loading, the round functions are selected by the control logic. In total 60 clock cycles are needed for a complete encryption or decryption. As with the other design the input key needs to be expanded to the last key value before any rounds can take place; this takes an extra 20 clock cycles due to the pipelined Sbox. The total number of cycles for encryption or decryption could be reduced to 30 by using 16 sboxes at the expense of more area.