Dynamic global security-aware synthesis using SystemC

F. Burns, J. Murphy, D. Shang, A. Koelmans and A. Yakorlev

Abstract: A dynamic global security-aware synthesis flow using the SystemC language is presented. SystemC security models are first specified at the system or behavioural level using a library of SystemC behavioural descriptions which provide for the reuse and extension of security modules. At the core of the system is incorporated a global security-aware scheduling algorithm which allows for scheduling to a mixture of components of varying security level. The output from the scheduler is translated into annotated nets which are subsequently passed to allocation, optimisation and mapping tools for mapping into circuits. The synthesised circuits incorporate asynchronous secure power-balanced and fault-protected components. Results show that the approach offers robust implementations and efficient security/area trade-offs leading to significant improvements in turnover.

1 Introduction

Cryptographic devices are becoming increasingly ubiquitous and complex, and in order to satisfy the high throughput requirements of many applications, they are often implemented by means of VLSI devices (crypto-accelerators). The high complexity of such implementations raises concerns regarding their reliability. During the last decade, a new class of attacks has arisen against such cryptographic devices including those that are signature-based [1] and those that are fault-based [2]. These side-channel attacks exploit easily accessible information like power consumption, running time and input–output behaviour under malfunctions and can be mounted by anyone using low-cost equipment.

The number of devices to counteract such attacks has increased in the last few years [3]. Common counterattack methods include masking, randomisation and fault detection [4]. Counterattacks based on fault detection assume that attacks based on fault injection can be detected and corrected automatically by fault detection circuits. Other counterattacks are based on power balancing [5] where reduction of data leakage from power signatures is an issue. These counterattacks assume that attacks that are based on power analysis can be prevented by balancing the power signatures generated by the circuit. More recently, the use of asynchronous circuits [6] has been explored. Asynchronous speed-independent (SI) circuits offer some advantages for counteracting such circuit attacks. First, no clock means that clock glitch attacks are removed. Secondly, circuits comprising dual-rail codes, which exhibit two valid signal combinations {01, 10}, can be balanced to reduce data-dependent emissions.

Targetting secure systems to chips normally requires significant manual design effort. Research is therefore needed to develop methodologies and techniques for synthesising robust cryptographic systems efficiently and dynamically. The increasing complexity of today's systems incorporating security demands more elaborate synthesis techniques to close the productivity gap. Recent work towards creating a VLSI design flow for side-channel attack resistant circuits was carried out by Tiri and Verbauwhede [7]. This was primarily applied at the lower level and was targetted towards power-balanced synchronous circuits. Aigner *et al.* [8] investigated side-channel attacks at the lower level and concluded that the best solution to power analysis is to embed countermeasures into logic cells.

The aim of our synthesis approach is to translate specifications efficiently and dynamically from higher level specifications, that is, SystemC specifications, into robust security implementations including asynchronous circuits (single-rail and dual-rail). This entails the use of security-aware scheduling and binding in order to explore the trade-off between security, area and time. Here the synthesised implementations include components that exhibit varying degrees of protection based on the implementation technology.

Recent work on extending the SystemC language to include more specific models of computation has been carried out in [9]. Some work on extending the SystemC language for asynchronous design has been carried out in [10]. Our synthesis flow extends the SystemC framework by adding a high level SystemC library which provides for higher level security constructs (including asynchronous) enabling the construction of behavioural security specifications which are both modular and extensible.

The basic synthesis flow for our technique is shown in Fig. 1. At the top of Fig. 1, the synthesis flow starts out with a SystemC modular behavioural description. The SystemC specifications are first compiled using a high-level security library. After compilation, the synthesis flow enters the stages of partitioning and communication synthesis. At the core of the system is a novel security-aware scheduling and binding algorithm which allows for a trade-off between security, area and time. The global scheduling approach works by scheduling to a range of components which exhibit varying security levels, that is, the algorithm

doi:10.1049/iet-cdt:20060121

Paper first received 16th August 2006 and in revised form 13th February 2007

The authors are with the School of Electrical and Computer Engineering, Merz Court, University of Newcastle Upon Tyne, Newcastle Upon Tyne NE1 7RU, UK

E-mail: f.p.burns@ncl.ac.uk

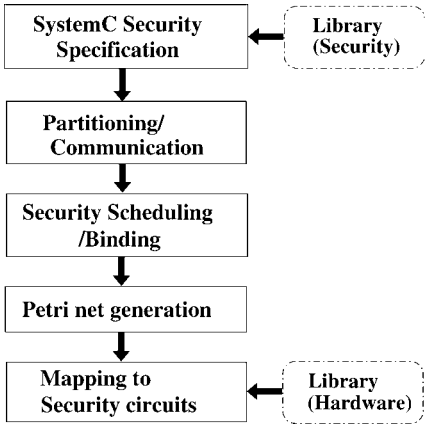

Fig. 1 *Basic synthesis flow*

schedules to components of more than one technology type. The differences in security level of components are based on the different types of technology employed. The algorithm is implemented using simulated-annealing [11] in which security type is included as an additional cost parameter. The energy function for this is taken as a function of the delay, area and percentage security type allocation.

The scheduling and binding output is subsequently translated into an intermediate Petri net format. The intermediate format is used for the mapping of the specification into power-balanced and fault-protected circuits using asynchronous direct mapping methodologies. These are used in conjunction with a dedicated library of hardware components.

2 SystemC security library

The synthesis flow makes use of a SystemC high-level security library. The object-oriented security library itself has been built on top of the core of the SystemC language. The modelling constructs of the SystemC language are provided as a class library. The library has been written using object-oriented techniques such as object references, inheritance and polymorphism.

Fig. 2 gives a block diagram of the structure of the SystemC security library. On the left-hand side of Fig. 2, a set of class definitions exist for each part of the core language such as modules, ports, processes, interfaces and channels. On the right, we have incorporated a separate layer of non-standard user-defined security types alongside the existing SystemC types. The library layer above or on top of this consists of the specialised asynchronous classes that contain security constructs that are considered separate from the SystemC core language. These are split into two

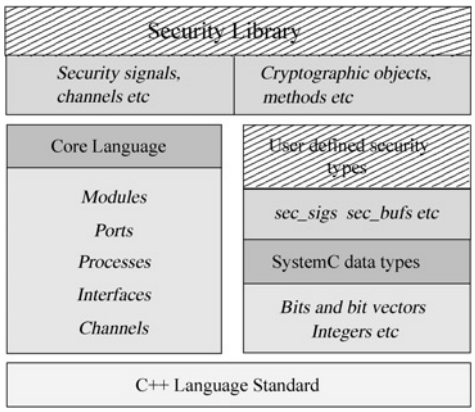

Fig. 2 *SystemC security library*

sections, one consisting of security signals and channels to be used in the system level specification and the other consisting of specific crytographic objects, methods and so on, which may be used in the construction of security subsystems, that is, security cipher blocks.

The security synthesis tool inputs system specifications which consist of a set of behavioural security modules. A specialist set of SystemC behavioural descriptions and classes have been incorporated into the framework. These enable the user to modify the existing specifications before synthesising them into circuits. As well as offering modularity, the SystemC language provides extension by inheritance. This is used in the library for the extension of SystemC behavioural descriptions.

2.1 User-defined asynchronous classes

SystemC behavioural descriptions are represented using a set of user-defined class types derived by inheritance from the existing class set. Classes for security channels are inherited from their primitive classes together with definitions for describing the style of the implementation. For example, the skeleton for a new security class for an asynchronous signal is

```
template <type>
class security_signal: public sc_signal
{
  public:
    //data members
    type asynchronous;
    style dual_rail;
    //constructors
};
```

Here a new class `security_signal` is defined by inheritance from an existing primitive channel `sc_signal`. The derived signal data value is templatised as to its type. The type and style for the security class, asynchronous and dual-rail, is set using constructors. Where necessary, overloading is used for the derived signal. In a similar way, we can specify classes for other styles of signal such as single-rail or multi-rail.

Specifying security concepts using classes enables constructs such as different types of buffer to be captured, for example, single-rail and dual-rail

```
sc_buffer(single_rail);
sc_buffer(dual_rail);
```

These can be specified together with their type and information relating to security metrics. For example, some buffers may be specialised for power balancing. The details of this may be contained in the class.

2.2 Cryptographic objects and methods

The library contains built-in specific objects related to specific encryption cipher classes. For the encryption AES block, cipher-specific objects appear in the library specifically related to the block cipher. For example, for the AES, object-specific methods exist for key expansion

```
aes::keyexpand128();
aes::keyexpand192();
aes::keyexpand256();
```

Other object-specific methods exist for more specialised cryptographic functions such as sbox transformation, column mixing, randomisation and so on.

```
aes::sbox(paritycorr);
aes::mix(srow);
aes::randomize(name,3,on);
```

The library is structured in a modular manner for each class of encryption object.

3 Synthesis of secure circuits

Fig. 3 gives a more detailed block diagram of the synthesis flow exhibited by our technique. The main synthesis flow starts from a SystemC security specification. The specification consists of a network of behavioural modules that are linked by communication channels. The SystemC specifications are first compiled using the high-level security library.

After compilation, the specification is partitioned and the communication channels between modules are established. Channels that have been specified as secure are first passed to the secure channel synthesiser. These are used to form secure transaction level connections between processes. The connections are protected using masking.

Each module contains a single behavioural process. The processes are extracted from the modules and security-aware scheduling and binding is applied to each of them. This is executed using a mix of different technology types with differing levels of security.

After scheduling and binding, the output is subsequently translated into an intermediate Petri net format. The intermediate Petri net format comprises two parts, security datapath nets (SDPNs) and control nets. The SDPNs are used for mapping of the specification into a mixed security datapath based on different technologies using asynchronous direct mapping. The intermediate control nets are mapped directly into control circuits. The direct mapping methodology is used in conjunction with a dedicated library of secure hardware components.

3.1 Security aware scheduling and binding

The behavioural processes themselves are extracted from modules and are scheduled using a global security-aware scheduling and binding algorithm. Scheduling is done so that we allocate different elements of datapath for operations which have different levels of security (e.g. to save power at the cost of less security – because less secure operations need not consume as much power as more secure, more secure may consume more power because of the extra cost incurred by power balancing).

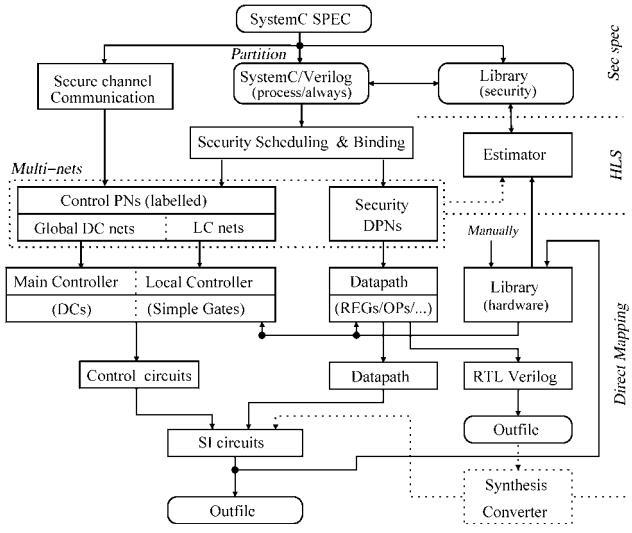

Fig. 3 *Detailed synthesis flow*

For global scheduling and binding, a simulated-annealing algorithm is employed. The algorithm is unique in that it includes security as a cost parameter in addition to area and time. The resource set which is utilised by the algorithm is made up of a mixture of library components where each set of components is based on a different technology.

$$resource\_set = secure\_res\_set\_1 \cup secure\_res\_set\_2$$

Secure components may be selected from the library by the algorithm for power balancing and/or fault protection. The resource set is initially fixed to a maximum limit based on the maximum degree of parallelism derived from the specification.

$$|secure\_res| = |secure\_res\_set\_1| = |secure\_res\_set\_2| = |maxpar\_op|$$

The pseudocode for the global simulated annealing algorithm is shown below.

```
T = initial_temperature;
curr_alloc = INIT_ALLOC(resource_set);
costcurr = SECCOST(curr_alloc);
while !tststop do
  while !endinnerloop do
    new_alloc = STEPFROM(curr_alloc);
    costnew = SECCOST(new_alloc);
    changec = costnew-costcurr;
    x = F(changec,T);
    r = RANDOM(0,1);
    if(r < x) then
      curr_alloc = new_alloc;
      costcurr = costnew;
    endif
  endwhile
  T = UPDATE(T);
endwhile
```

The annealing temperature T (standard cooling temperature) is initially set to a user-defined value by the algorithm. The algorithm then proceeds by assigning a series of initial random bindings using an initial allocation function `(INIT_ALLOC)`. This function selects operations from the specification and randomly binds them to resources from the mixed resource set `(resource_set)`. The energies for the initial set are then calculated using a security-weighted cost function `(SECCOST)`. The cost function is calculated as a weighted sum of the delay, area and percentage of security allocation as follows

$$costcurr = delayw \times delay + areaw \times area + diffareaw \times diff\_percent\_tech$$

where

$$diff\_percent\_tech = abs(area\_tech\_1 - area\_tech\_2) \times 100/totalarea$$

The different weights in the cost function may be adjusted by the user to alter for different levels of security, area and time. The range of the weights is problem dependent and is therefore adjusted based on the relative values of the associated variables.

The main loop and then the inner loop of the algorithm is entered. On each iteration of the inner loop, rebinding steps are taken in the algorithm using a stepping function

(STEPFROM) which randomly rebind or switch one operation at a time to a different resource from the mixed secure/non-secure resource set. The new cost of the rebinding is then calculated, the cost function is called and the difference between the new and old cost (costnew–costcurr) is worked out. This is used to determine whether the new solution is an improvement over the old one.

The simulated annealing offers a very similar strategy to iterative improvement, with one major difference: the process of annealing allows perturbations to move uphill in a controlled fashion. This works by evaluating the energy after each random move. If the energy is reduced, the new configuration has lower energy and is accepted as the starting point for the next move. However, if the energy is increased, the move may still happen. The decision is based on a change in the energy and a random value in the range (0, 1), that is, acceptance is based on a probability determined by the change in c and T, using the function $x = F(changec, T) = exp(-c/k \times T)$, and a random number $r = RANDOM(0, 1)$.

When the fixed number of iterations (inner loop) is complete, the best solution or probable solution that reduces the total energy is taken and the next iteration in the main loop occurs. The stopping criterion for the algorithm is satisified when the temperature approaches zero.

3.2 Heuristics

Heuristics are required for steering or guiding the algorithm to the solution more accurately based on the chosen weights. This is achieved by inserting non-random functionality into the stepping function (function STEPFROM). This works by steering the steps in the stepping function directly rather than randomly after a fixed number of iterations. This is based on the relative value of the energies associated with the weights. If the value of one of these exceeds a relative threshold compared with the other weights, then a move is made directly to reduce the cost associated with that particular weight. For example, if the energy component related to the percentage difference becomes larger than that of the energy component related directly to the area, after a fixed number of iterations, a forced move is made to reduce the former energy component. The effect of the steering of the stepping process is to guide the rebinding in a more direct manner to the new solution more quickly.

Individual resources may be assigned a more refined system of weights to steer the algorithm towards providing an improved secure implementation based on the level of priority for individual components, for example, for a block cipher, the following system of weights may be used

$$costsec = sboxw \times sboxarea + sroww \times srowarea + mixw \times mixarea + xorw \times xorarea$$

Here each crytpographic resource has its own associated weight. Assigning weights inividually to resources ensures that security is targeted towards the least secure components first.

3.3 Net translation

After scheduling, the processes are translated into an intermediate Petri net format comprising SDPNs and control nets. The SDPNs that are generated are derived from a subset of Colored Petri Nets (CPNs) [12]. CPNs are a higher level kind of Petri net where places are associated with types (colour sets). The set of colour sets determines

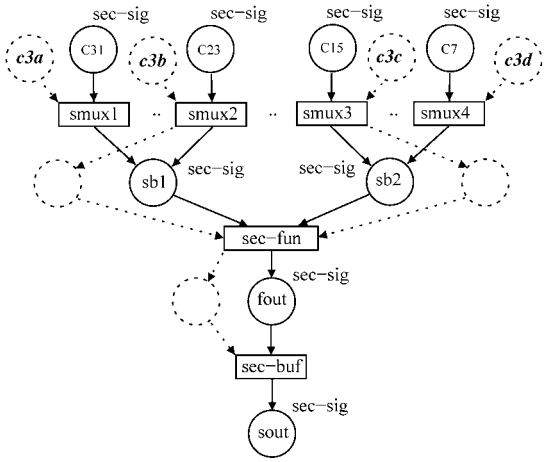

Fig. 4 *Security DPN fragment*

the types, operations and functions that can be used in the net inscriptions.

An example of a security annotated SDPN fragment with localised control flow is shown in Fig. 4 where the SDPN appears flowing downwards. The SDPN uses coloured places, transitions and arcs containing details about the security, and the localised control is shown using dotted places and arcs. The SDPN places are annotated with colour sets containing information about the security type of signal (e.g. sec_sig) and the SDPN transitions are annotated with security labels (e.g. tsec_buf).

The annotations in the SDPNs are derived directly from the class types and style of security signals. The SDPN details are used as a subsequent reference for the lower level mapping procedures.

3.4 Mapping technique

Asynchronous syntax-directed mapping techniques are used for both datapath and control. Because of exploration of the design space, the syntax-directed mapping methodology that is used exploits an implicit, semantical execution-flow link from the specification to circuits.

The intermediate net format contains all the details relevant for mapping. The SDPNs are mapped directly into secure datapaths. The control nets are subsequently mapped directly into an asynchronous network of David Cells [13].

For direct mapping of the SDPN, a hardware library of specialised secure elements is provided. The net fragments

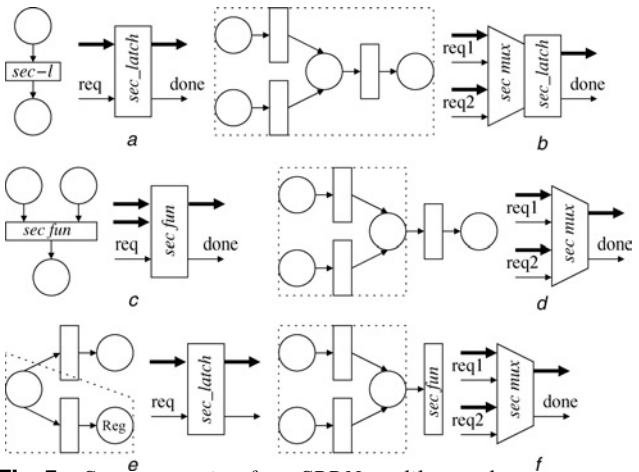

Fig. 5 *Secure mapping from SPDNs to library elements*

which are derived from the SDPN contain details of how the mapping is made to the various secure elements in the library.

Fig. 5 shows how various Petri net fragments are translated into various secure data-path elements. Fig. 5*a* shows the mapping for a secure latch, Fig. 5*b* shows the mapping for a secure multiplexor-latch, Fig. 5*c* shows the mapping for a secure function, Fig. 5*d* shows the mapping for a multiplexor leading to another multiplexor, operator and so on.

3.5 Basic elements in the library

A library of basic specialised SI secure components has been developed. For each function in the SystemC security library, a dedicated IP block exists in the hardware library. Each IP block is implemented using SI circuits. The library consists of all ordinary secure datapath operations, such as secure ALU, add, subtract, compare and multiplex. In addition, secure asynchronous components have been developed that realise basic encryption components such as secure S-boxes, mix columns and key generation.

SI circuits typically use a redundant encoding scheme such as dual-rail. These properties are exploited in the design of power-balanced secure asynchronous components [6]. Standard dual-rail gates are asymmetric, which means they are not good for balancing the power consumption of circuits. Therefore non-standard dual-rail asynchronous AND, OR and XOR gates are used.

Asynchronous memory circuits benefit from their dual-rail properties for code balancing. A set of secure memory elements exist in the library that are specially designed for power balancing. These make use of a return to zero RTZ protocol to ensure that power balancing is attained. Alternative specialised memory elements have also been developed, which use more than one spacer in their protocol. The spacers are specially configured to balance the switching activity and therefore the power signature. Other secure memory elements exist in the library which are more specifically designed to provide resistance against fault attack.

Power-balanced components are implemented using different technologies in the library. For example, components in the libary may be implemented using a differential power-balancing logic based on path balancing [14]. There are various forms of differential technology that are used for this. These are shown in Table 1. Table 1 shows the logic for different technologies together with their physical-level data and timing. This includes things such as implementation style, NAND transistor count and timing characteristics and also power bias data including peak current and time width. Tiri and Verbauwhede [15] proposed using power-balanced dynamic logic, based upon Sense Amplifier Logic (SABL). The key advantage of this technology is that when it is activated the same number of switching events occur and memory effects are mitigated. This is achieved naturally at the logical level as the logic is differential but importantly also at the physical level which makes it more robust.

DIMS technology is based on the following concept. A differential function or gate can be formed by implementing explicitly the Boolean function using a C-element for each min-term and OR gates for ORing of the min-terms. Various types of DIMS technology exist based on their area and power-balancing characteristics. Some examples of DIMS NAND gates are shown in Fig. 6. A logic example of a DIMS_1 NAND gate is shown in Fig. 6*a*. DIMS_1 technology operates at a weaker level of security because the paths leading to the outputs may not perfectly balanced. In order to achieve better balancing, more logic must be added to better balance the output paths. This is referred to as DIMS_2 technology. An example for this is shown in Fig. 6*b*. A more efficient version of this can be derived by shorting a few of the inputs to ground. This is referred to as DIMS_3 technology. An example for this is shown in Fig. 6*c*. Components in the library are implemented using logic based on the technologies outlined above.

3.6 Low level refinements

Protection against coded wire imbalances must also be provided for as these may produce power leakage. Low-level refinements such as balancing wire lengths are carried out using lower level design tools [15].

4 Example

The specifications that we have synthesised using our secure synthesiser incorporate cryptographic modules which include block ciphers such as the AES and DES. For our main example, we focus on the synthesis of a system comprising the standard AES block cipher. The symmetric block cipher Rijndael was standardised by the National Institute of Standards and Technology in November 2001. It is defined for a block size of 128 bits and key lengths of 128, 192 and 256 bits. A block diagram of the AES subsystem is shown in Fig. 7.

The AES is a block cipher which takes a variable size data block and a variable size key and performs four different standard transformations, which are SubBytes, ShiftRows, MixColumns and AddRoundKeys. For more details of the functionality of the AES the reader is referred to [16].

Below we show a skeleton SystemC specification for the AES which has been extracted from the main specification during partitioning by the synthesis system. The chosen block length is 128.

At the top of the SystemC, specification, signals are first defined for the text and key using `sec_sig` security signals which are inherited from their primitive class. Below this, buffers are specified for the key and text and then the main behavioural procedure for the cipher is

Table 1: Logic power-balancing countermeasures

Logic style	Logical level	Physical level				Power bias spike	
Technology	Temporal timing	Implementation	NAND transistor count	Memory effects	Temporal timing	Peak, mA	Width, ns
DIMS_1	yes	standard cell	46	yes	yes	0.17	0.50
DIMS_2	no	standard cell	72	yes	yes	0.09	0.27
DIMS_3	no	standard cell	56	yes	yes	0.08	0.30
DIMS_4	no	full custom	102	no	no	–	–
SABL	yes	full custom	16	no	yes	0.05	0.60

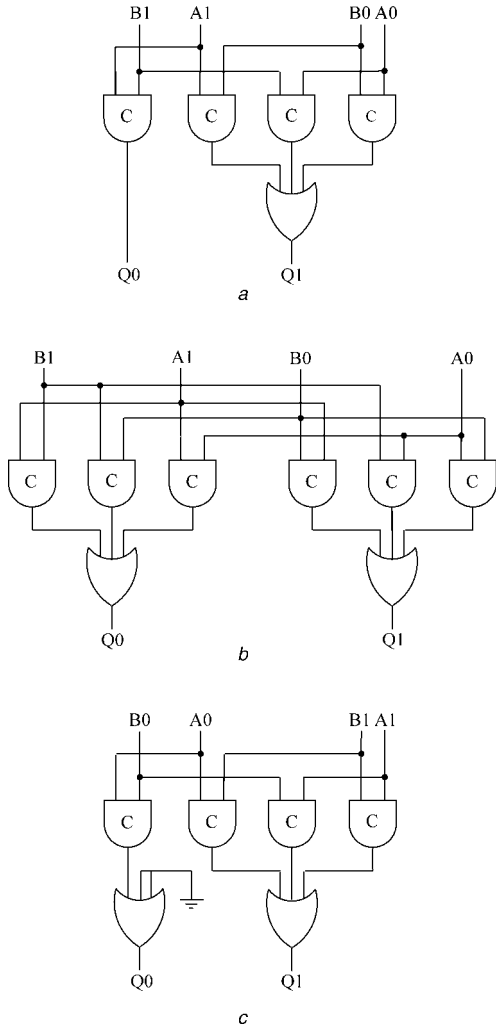

Fig. 6 *DIMS logic gates*
a DIMS_1 NAND
b DIMS_2 NAND
c DIMS_3 NAND

called, that is, `ciph_top` which inherits its own definitions from its behavioral class set.

```
SC_MODULE(aes){
  signal<bool> start;
  sec_sig<sc_int<128>> txtin(dr);
  sec_sig<sc_int<128>> keyin(dr);
  ..
  sec_buf<sc_int<128>> txt(dr);
  sec_buf<sc_int<128>> key(dr);
  void ciph_top();
  void key_gen();
  SC_CTOR(aes) {
  ..
  }
};
void aes::ciph_top() {
  ;
  //definitions
  ..
  if(start) {
    main body {
    n = 0;
    while(10>=n) {
      //sbox instance;
      sbout = aes::sbox(c0,c1,c2,c3);
      //srow instance;
      srout = aes::srow(sbout);
      //etc
      //mix, exor etc
      //etc
      }
    }
  }
};
```

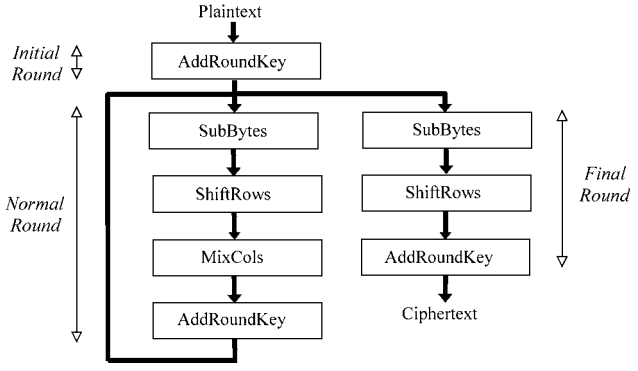

Fig. 7 *AES subsystem*

The specification was compiled. The weights were set in the security-aware scheduling algorithm and the AES was scheduled. The nets were subsequently synthesised.

A diagram of a synthesised SDPN for the AES is shown in Fig. 8. On the left-hand side of Fig. 8, the SPDN is seen to be predominantly annotated using security level_1 (level_1 labels), whereas on the right-hand side, it is predominantly annotated using security level_2 (level_2 labels). In this case, level_1 is used to represent DIMS_1 technology and level_2 is used to represent DIMS_2 technology as described earlier. Five multiplexer nets are shown in the diagram, two before the S-boxes, one before the ShiftRow and two before the AddRoundKey. Four security buffers have been synthesised using security level_2 including the key.

After net generation, the nets were directly mapped into a secure implementation. A diagram of the corresponding synthesised circuit is shown in Fig. 9. The lightly shaded regions in Fig. 9, for example, show the parts of the datapath that have been synthesised using DIMS_1 power-balancing technology, whereas the heavily shaded parts show the parts that have been synthesised using DIMS_2 power-balancing technology. The DIMS_1 functions are synthesised with level_1 storage, whereas DIMS_2 functions are synthesised using level_2 storage. The key generation unit has been completely synthesised using security mapping directly from the specification and all of the subcomponents contained within it are synthesised using the higher level_2 security.

We specified the AES similarly, and by varying the weights in the simulated annealing algorithm, we synthesised for different combinations of components using varying levels of security. We observed the trade-offs in area, time and level of safety. The level of power balancing is measured in each case in terms of the type of power balancing.

A basic level of power balancing is provided in terms of power balancing for the implementation based on the particular technology used. The-power-balancing equation for the implementation is given in terms of the ratio of the power-balanced level area of a particular level to the total circuit area

$$area\,power\,balancing\,level\_n = \frac{powerbalancedarealevel\_n}{totalcircuitarea} \cdot 100\%$$

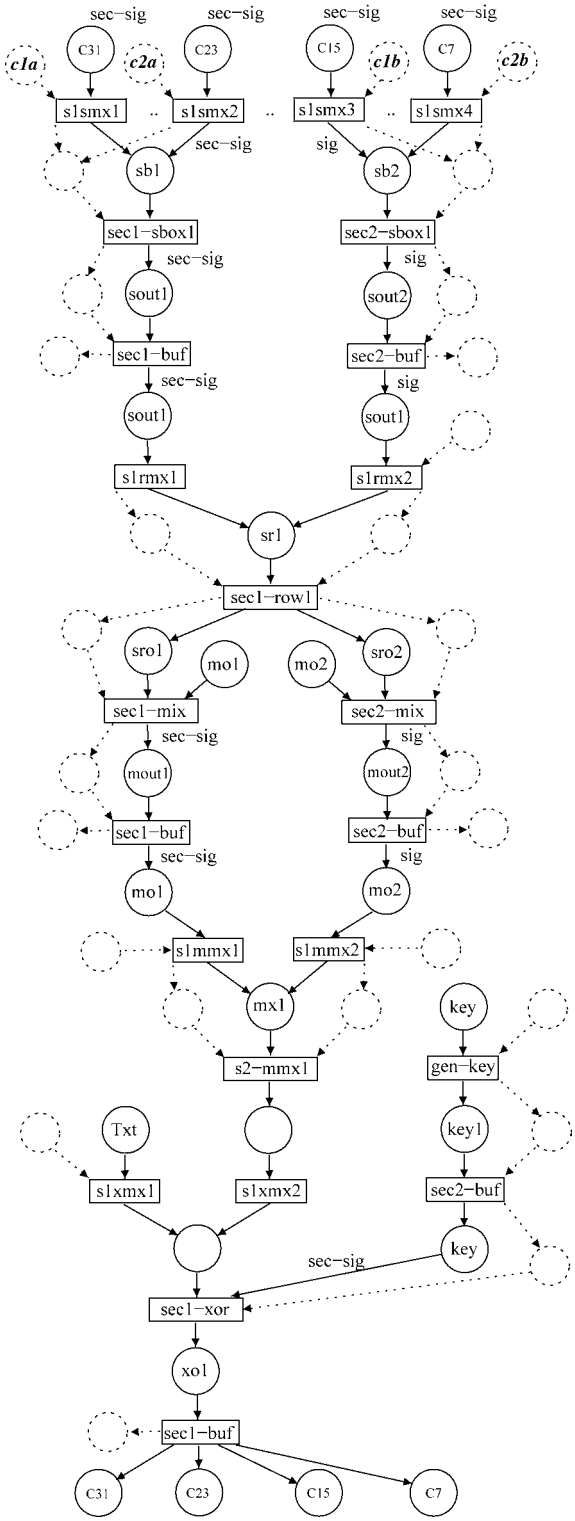

Fig. 8 *AES SDPN*

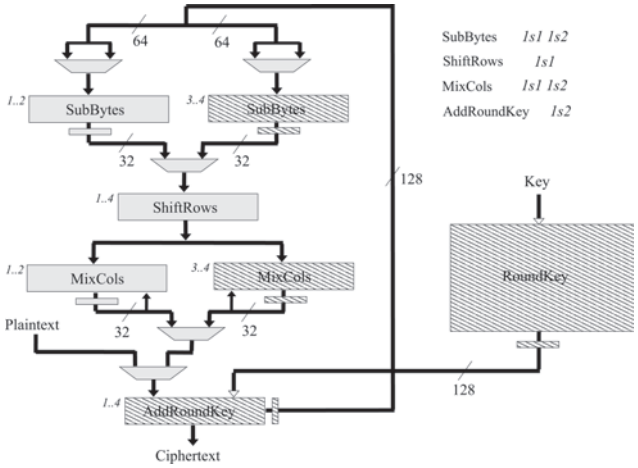

Fig. 9 *AES implementation*

Table 2 shows a comparison of the areas, time and levels of security (in terms of the power-balancing equation) for a number of different synthesised circuits. The first few columns in Table 2 show the simulated annealing weights and the percentage slit that is required between the different technologies. The final two columns in Table 2 show the actual percentage for power balancing using DIMS_1 technology against the percentage for power balancing using DIMS_2 technology. The top half of Table 2 shows the trade-off attained by fixing the value of the weight for the delay at 50.0, keeping the value of the weight for area fixed at 25.0 and varying the required percentage split between the two technologies in the range (30–90). The corresponding value of the weight for Diffareaw was varied in the weight range (10.0–70.0). In the upper half of Table 2, the total area is shown to increase for a general decrease in percentage of DIMS_1 technology and an increase in percentage of DIMS_2 technology. The percentage of DIMS_2 technology closely matches the percentage required as shown in the percentage split column with the largest differential being 7.7% in row 1. The experimental results shown in the upper half of Table 2 was repeated as shown in the lower half of Table 2 but this time the weight for the delay was increased to a higher level in order to improve the time while keeping all the other weights similar to that in the upper half of Table 2 including the percentage split required. In the lower half of Table 2, each row shows an improvement in time against the corresponding row in the upper half. This is at the expense of a larger average difference between the required percentage split (percentage required for DIMS_2) and the actual percentage split. The area measurements in the table are shown normalised and time is given in terms of the loop iteration time (single loop). Estimates were based on an AMS 0.35 μm cell library.

For a second set of results, the DIMS_2 technology was replaced by the DIMS_3 technology to attain a different set of values for power balancing as shown in Table 3. As in the upper half of Table 2, an initial trade-off was made by fixing the value of the weight for the delay at 50.0 while keeping the value of the weight for the area fixed at 25.0 and varying the required percentage split in the range (30–90). The corresponding value of the weight for security difference was varied in the weight range (10.0–50.0). The upper half of Table 3 shows the results of the areas and time against the percentage of power-balancing technology. The results here show a similar pattern to Table 2, that is, the total area is shown to increase for a general decrease in percentage of DIMS_1 technology and an increase in DIMS_3 technology. This trade-off occurs for an average decrease in the total area against Table 2 because of the more efficient DIMS_3 technology used. There is also an overall implied average increase in the power balancing level over Table 2 because of the more secure type of DIMS_3 technology used. In the lower half of Table 3, the weight for the delay was also increased in order to show an improvement in time. As in Table 2, the times on average show an improvement over the upper half of Table 3 at the expense of a larger average difference in required

Table 2: AES results 1

Delayw	Areaw	Diffareaw	Percentage split	Total area	Time, ns	Percentage DIMS_1, %	Percentage DIMS_2, %
50.0	25.0	10.0	30	8 971	115	62.3	37.7
50.0	25.0	27.0	50	10 453	83	53.5	46.5
50.0	25.0	44.0	70	14 102	108	30.1	69.9
50.0	25.0	70.0	90	20 551	89	11.9	88.1
70.0	25.0	10.0	30	8 914	81	70.9	29.1
80.0	25.0	27.0	50	10 148	82	55.5	44.5
60.0	25.0	44.0	70	13 579	77	40.5	59.5
400.0	25.0	70.0	90	19 789	61	27.6	72.4

Table 3: AES results 2

Delayw	Areaw	Diffareaw	Percentage split	Total area	Time, ns	Percentage DIMS_1, %	Percentage DIMS_3, %
50.0	25.0	10.0	30	8 213	113	68.1	31.9
50.0	25.0	30.0	50	10 892	77	51.7	48.3
50.0	25.0	40.0	70	11 543	109	33.6	66.4
50.0	25.0	50.0	90	16 160	81	11.2	88.8
60.0	25.0	10.0	30	8 820	83	63.4	36.6
80.0	25.0	30.0	50	10 847	77	51.5	48.5
60.0	25.0	40.0	70	12 272	77	44.8	55.2
300.0	25.0	50.0	90	16 592	61	32.9	67.1

percentage split (percentage required for DIMS_3) and the percentage split achieved.

5 Conclusions

A new security-directed synthesis approach has been described for efficiently synthesising systems which incorporate secure asynchronous crypto-accelerators from SystemC using an asynchronous direct mapping technique. The use is made of system and functional level modelling to achieve this.

The synthesis system makes use of a stuctured library of SystemC behavioural descriptions. This enables the creation of specifications which are both modular and extensible. The new synthesis approach has been incorporated into a more generalised asynchronous synthesis direct mapping flow to enable more efficient generation of robust secure SI implementations.

At the core of the system, we have introduced a global scheduling and binding algorithm. This is implemented using a global simulated-annealing algorithm in which security is included as an additional cost parameter. The approach we have adopted allows three parameters; security, area and time, to be traded off. This allows us to generate secure asynchronous circuits quickly, while at the same time allowing the potential to dynamically explore the design space using a mixed library of asynchronous components with varying levels of security. Trade-offs are made between different technologies which allow for an improvement in overall level of protection at the expense of area. Results show that we can efficiently trade off security, area and time.

Other work on transformation and optimisation to exploit the design space is also being explored including power consumption. We are also interested in refining the heuristics in order to search the design space more efficiently. We would also like to look at trade-offs using lower level power-balancing technologies at both the logical and physical level.

6 Acknowledgments

Many thanks go to the referees for their helpful and constructive comments. This work has been supported by EPSRC grant GR/S81421 (project SCREEN).

7 References

1 Kocher, P., Jaffe, J., and Jun, B.: 'Differential power analysis'. Proc. Advances in Cryptology (CRYPTO 1999), LCNS, 1999, pp. 388–397
2 Biham, E., and Shamir, A.: 'Differential fault analysis of secret key cryptosystems'. Advances in Cryptology – Proc. CRYPTO '97, 1997, pp. 513–525
3 Bertoni, G., Breveglieri, L., Kohen, I., Maistri, P., and Piuri, V.: 'Error analysis and detection procedures for a hardware implementation of the advanced encryption standard', *IEEE Trans. Comput.*, 2003, **52**, (4), pp. 492–505
4 Yen, C., and Wu, B.: 'Simple error detection methods for hardware implementation of the advanced encryption standard', *IEEE Trans. Comput.*, 2006, **55**, (6), pp. 720–731
5 Benini, L., Macii, A., Macii, E., Omerbegovic, E., Poncino, M., and Pro, F.: 'Energy-aware design techniques for differential power analysis protection'. Proc. Design Automation Conf. (DAC), 2003, pp. 36–41
6 Sokolov, D., Murphy, J., Bystrov, A., and Yakovlev, A.: 'Design and analysis of dual-rail circuits for security applications', *IEEE Trans. Comput.*, 2005, **54**, (4), pp. 449–460

7 Tiri, K., and Verbauwhede, I.: 'A VLSI design flow for secure side-channel attack resistant ICs'. Proc. Design Automation and Test in Europe (DATE), 2005, pp. 58–63
8 Aigner, M., Mangard, S., Menichelli, F., Menicocci, R., Olivieri, M., *et al*.: 'Side channel analysis resistant design flow'. Proc. ISCAS, 2006, pp. 2909–2912
9 Schulz-Key, A., Winterholer, C., Schweizer, M., Kuhn, T., and Rosenstiel, W.: 'Object-oriented modelling and synthesis of SystemC specifications'. Proc. Asia South Pacific Design Automation Conf. (ASPDAC'04), Yokohama, Japan, 2004, pp. 238–243
10 Bjerregaard, T., Mahaderan, S., and Sparso, J.: 'A channel library for asynchronous circuit design supporting mixed-mode modelling'. Proc. Int. Workshop on Power and Time Modelling, Optimization and Simulation (PATMOS), 2004, Springer, pp. 301–310
11 Devadas, S., and Newton, A.: 'Algorithms for hardware allocation in data path synthesis', *IEEE Trans. CAD*, 1989, **8**, pp. 768–781
12 Jensen, K.: 'Coloured Petri nets. Basic concepts, analysis methods and practical use', Volume 1, Basic concepts. EATCS Monographs in theoretical computer science (Springer-Verlag, Berlin, Heidelberg, New York, 1997)
13 Shang, D., Xia, F., and Yakovlev, A.: 'Asynchronous circuit synthesis via direct translation'. Proc. ISCAS, Scottsdale, Arizona, 2002, pp. 369–372
14 Moore, S., Anderson, R., Cunningham, P., Mullins, R., and Taylor, G.: 'Improving smart card security using self-timed circuits'. ASYNC'02, 2002, pp. 114–125
15 Tiri, K., and Verbauwhede, I.: 'Design method for constant power consumption of differential logic circuits'. Proc. Design Automation and Test in Europe (DATE), 2005, pp. 628–633
16 Managard, S., Aigner, M., and Dominikus, S.: 'A highly regular and scalable AES hardware architecture', *IEEE Trans. Comput.*, 2003, **52**, (4), pp. 483–491