

Exploiting Bayesian Belief Network for Adaptive IP-reuse Decision

A. W. Azman^{1,2} A. Bigdeli^{1,2} M. Biglari-Abhari³ Y. M. Mustafah^{1,2} B. C. Lovell^{1,2}

¹ School of ITEE, The University of Queensland, Australia.
 {amelia,yasir,lovell}@itee.uq.edu.au

² National ICT Australia, Queensland Research Laboratory, Australia.
 Abbas.Bigdeli@nicta.com.au

³ Department of Electrical & Computer Engineering, University of Auckland, New Zealand.
 m.abhari@auckland.ac.nz

Abstract—A smart camera processor has to perform substantial amount of processing of data-intensive operations. Hence, it is vital to identify critical segments of the processing load by involving HW/SW codesign in smart camera system design. This paper presents a novel fully automatic hybrid framework that combines heuristic and knowledge-based approaches to partition, allocate and schedule IP modules efficiently. In this work, the concept of Bayesian Belief Network (BBN) is utilised and incorporated into the proposed framework. In the experiment section of this paper, we report a comparison of our proposed framework with three previously published work: A BBN based method proposed by a research group from the University of Arizona, the exhaustive algorithm and finally the with greedy algorithms.

I. INTRODUCTION

It has been established that a more sophisticated systems such as smart camera is required to assist the existing CCTV monitoring systems. This is a pragmatic solution since a smart camera has built-in processing capacity to allow early computations on the camera, hence, providing instant identifications and notifications of any unusual behaviour in the area under surveillance [1]. Figure 1 depicts both low-level as well as high-level image processing operations that could be typically incorporated into a smart camera.

Recently, there has been a trend in the smart camera design as well as other embedded systems to use reconfigurable platforms particularly FPGAs [2]. By unifying a reconfigurable portion, it enables products such as a smart camera to have a broader range of applications [3]. Advancement in reconfigurable computing platforms such as FPGAs has led to a considerable growth in the HW/SW codesign research field. The key feature of this growth is due to the fact that modern FPGAs allow single or multiple embedded processors to be implemented on the same chip alongside other logic resources [4]. Besides that, the flexibility of reconfigurable architecture allows one to adapt the reconfigurable portion with the evolving standards or when a new IP becomes available for a product.

This paper presents a fully automatic framework for adaptive IP-reuse mainly for smart camera systems with recon-

figurably architecture. By using pre-designed and pre-verified IP blocks, the main objective of this framework is to solve an integration problem of new IP blocks onto an existing platform to meet user-defined timing and hardware resources constraints. Figure 2 illustrates an example of new IP blocks to be incorporated onto a smart camera. For this work, it is assumed that the hardware and software implementation of IP blocks are available in SystemC and C respectively for each of the smart camera related modules. One application for this work is to allow the reconfiguration process for the new IP blocks that targets smart camera platform that is either fully or partially programmable. The term virtual processor used in the diagram is to indicate that the software implementation can involve a single or multiple instances of microprocessors.

The latest trend in HW/SW codesign is to explore alternative methods that could replace the conventional method of pure heuristic approach. For example there has been research work to use a hybrid of Genetic Algorithm (GA) with Integer Linear Programming (ILP) [5] or to exploit the use of Artificial Neural Network (ANN) [6] as well as Bayesian Belief Network [7] in codesign. Our work explores and extends the use of Bayesian Belief Network (BBN) to *adapt* the IP blocks and efficiently partition, allocate and schedule the IP blocks into hardware and software components to satisfy the user-defined constraints. In the recent literatures, BBN has attracted much attention to take over the ANN. The reason for this shift of attention can be explained by the fact that although implementing BBN can be more difficult than implementing ANN [8], BBN has explainable learning process with higher degree of flexibility in the topology design. By using BBN, one can also avoid data over-fitting which is usually the case for ANN. Further literature review on BBN has shown that there is a potential of utilising BBN in HW/SW codesign. The reason for this lies in the core of the BBN concepts that stores causal and local evidence of the entire network. Similar to solving uncertainties in Artificial Intelligence (AI), by speculating encoded knowledge together with reasoning, a possible outcome can also be predicted for a codesign problem. For this work in particular, the outcome would be a

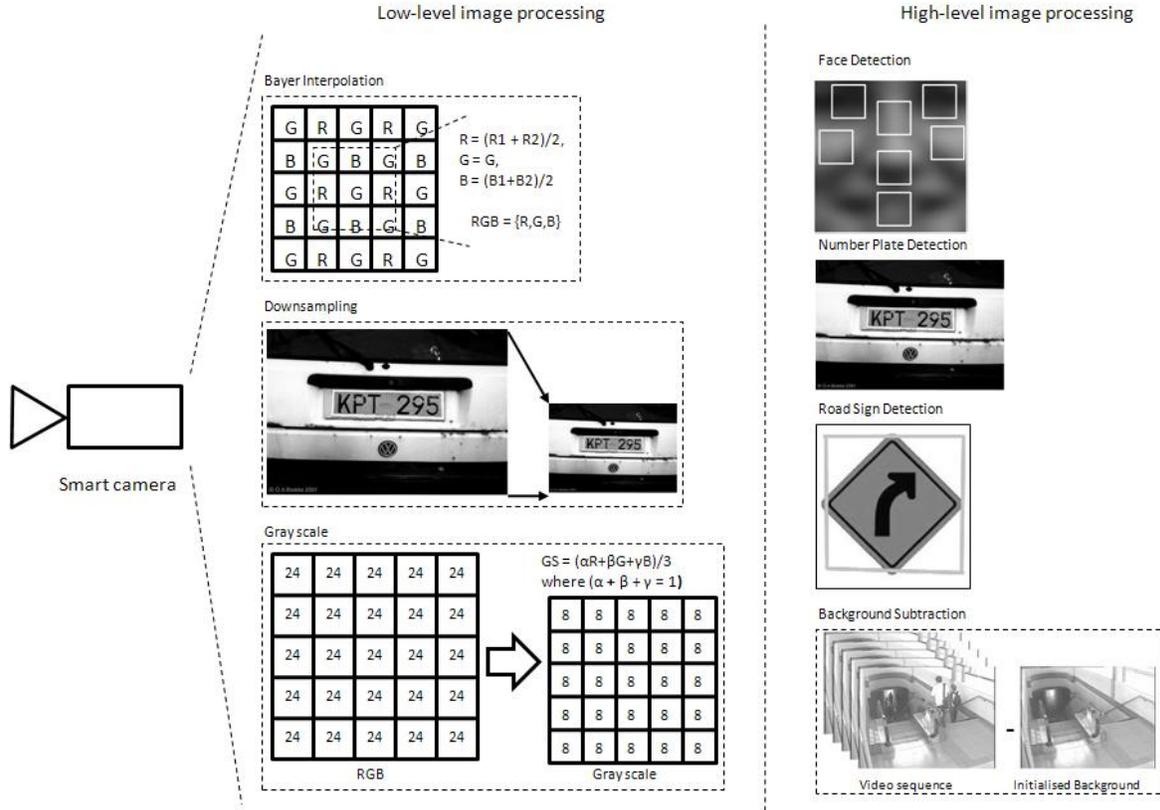


Fig. 1. Examples of the various vision algorithm for a smart camera system

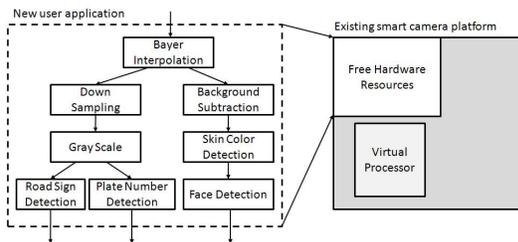


Fig. 2. Incorporating new user application to an existing smart camera reconfigurable platform

partitioning solution for an embedded system functional units into either hardware or software components.

A. Paper's Contribution

The main challenge in implementing BBN is the requirement to deduce from the knowledge of a system into a reliable relationship or influence factor. This relationship factors which are also known as the conditional probability are stored as link matrices in between nodes of a Directed Acyclic Graph (DAG). In the previous work by Olson et al. [7], a link matrix equation has been introduced to produce the conditional probability between two nodes. We tested the equation by using several random case studies and the result shows that

the link matrices calculated using this equation led to data conflicts or inconsistency in the BBN. Data conflict can occur in cliques values during observation process i.e. when evidences are entered. This can be a problem in the domain of HW/SW codesign. For example, from a thorough observation, an evidence, e_1 , is entered at clique A to be implemented as a software. Another subsequent observation was made and clique B is entered with an evidence to make clique B a hardware implementation. When evidence from clique B is propagated throughout the network, the status of clique A converges to hardware implementation which opposes the initial observation. This is known as data conflict.

Besides the link matrices, the other influential factor of a BBN is the observation or the evidence value. In [7], response time are used as the measurement for the evidence. This is given by the vector $e_i = (RT_{HW}/T, RT_{SW}/T)$ where RT_{HW} and RT_{SW} are hardware response time and software response time respectively while T is $T = (RT_{HW} + RT_{SW})$. In order to exploit the full potential of BBN in HW/SW codesign, a different set of observation is required since the execution time for hardware implementation is usually much faster than the software execution time.

In this paper we addressed these two problems and presents the following contributions: 1) a fully automatic hybrid partitioning framework, 2) a new link matrix equation to allow

better convergence and reducing the case of conflict in the BBN, 3) an investigation how timing and area metrics can be encoded in the link matrices and 4) an automatic evidence generator based on solving simple Constraint Satisfaction Problem (CSP). CSP is a structural iterative search method that can be used in solving many temporal and combinatorial problems [9].

B. Paper Structure

The remaining of this paper is organised as the following; Section 2 will explain the motivation behind this framework while section 3 covers the previous works on codesign particularly on HW/SW partitioning approaches. Section 4 will give an introduction to the proposed framework while the methodology behind the framework is discussed in section 5. Section 6 elaborates on the experimental setup for this framework. This is followed by a discussion and conclusion in section 7.

II. MOTIVATION

The advancement in smart camera technology has increased dramatically with the expanding number of research groups targeting to enhance the smart camera systems. In parallel to that, more complex vision IPs are designed to be incorporated into a smart camera. This only means that the standalone processor of a smart camera needs to be able to process complex computations and control signals while managing data transfer efficiently. The HW/SW codesign is one of the available methods that can be integrated into smart camera system design to find an optimum trade-off solution between hardware and software implementation while meeting the system specification. Meanwhile, reconfigurable devices especially FPGAs have emerged as an alternative implementation platforms in embedded system design [10] primarily because of their low cost, flexibility and shorter time-to-market. Besides allowing incorporating microprocessors on the same chip, FPGAs have large arrays of parallel logics and registers which enable designers to implement effective parallel processing. At the same time, to boost the potential of FPGAs, researchers are also keen on dynamically reconfiguring FPGAs in real time. In the case of this research work, the framework is proposing a case where a user is combining new IP blocks onto an existing programmable smart camera architecture with limited hardware resources. Meanwhile, there is a trend to encourage the IP-reuse in system-on-chip (SoC) [11] [12]. The main advantage of having standardized IP is the reusability which will indirectly result in shorter time-to-market and can reduce development time and excessive overhead costs. In a short time, many systems may benefit from the proliferation of IP-reuse. The same idea of IP-reuse can be applied for smart camera systems since different smart cameras can be configured with different sets of IP to be used for various applications.

Similar to solving AI, HW/SW codesign have uncertainties that need to be solved to achieve satisfactory trade-off solution between hardware and software. In particular, the scheduling

and partitioning in HW/SW codesign are well known to be an NP-complete (Nondeterministic Polynomial time) problems. For a reconfigurable architecture, these problems have been proven to be NP-hard [13]. Because of that, optimisation techniques based on heuristic methods are generally incorporated in the search to achieve feasible and near-optimal solutions. There are two commonly used heuristic approaches in HW/SW partitioning; 1) the software-oriented and 2) the hardware-oriented approach. In a software-oriented approach, all tasks are initially scheduled to be in software. Later, part of the systems tasks are scheduled as hardware until a feasible solution is determined. In most cases, this is until all timing constraints are met. The hardware-oriented approach on the other hand, starts the system implementation in hardware and gradually some tasks are migrated to software as long as the timing constraints are not violated while the hardware cost is reduced. During the rescheduling process, most published works have utilised the iterative method. While exhaustive heuristics method might be able to establish a suitable fit for the additional IP blocks on the available hardware resources, this manual approach usually takes longer decision time. The decision time grows exponentially as the number of IP blocks increases. Hence, to establish a more reliable partitioning solution, some designers began to exploit the knowledge-based approach. In this work, the proposed framework is based on the hardware-oriented partitioning approach where a hybrid of heuristic and knowledge-based techniques are incorporated to achieve a faster and more reliable hardware/software partitioning solution.

For an embedded system design such as a smart camera system, the *timing constraint*, *logic area* and *power consumption* are the three main characteristics that need to be assessed carefully in order to achieve a high performance design. It has been established that there are two main timing constraints in embedded system design; the hard and the soft timing constraints. Failure to meet a hard timing constraint would mean fatal fault in the design. One example for this case is a smart camera in a distributed surveillance system that has to produce a statistical output at a given time before passing the processed data to subsequent camera in the network in order to establish a correct surveillance activity. Meanwhile, power and energy consumption are becoming important especially for portable devices. A low-energy design is required to increase the longevity of the battery life. The power issue becomes increasingly important with the use of FPGAs. FPGA power consumption is highly dependent on its design implementation. Recently, FPGA manufacturers have begun to provide some optimised low power solutions at both architecture as well as at device levels¹.

III. PREVIOUS WORK

One of the earliest research works that explores hardware-oriented partitioning approach is by Gupta et al. [14]. The

¹<http://www.altera.com/products/devices/cyclone3/overview/power/cy3-power.html>

group presented a partitioning approach called Vulcan. They explained how hardware blocks written in HardwareC were iteratively moved to software to reduce the hardware cost. Another work that also utilises a hardware-oriented based approach is by Nieman as reported in his book [15]. The book structured an elaborate Integer Linear Programming (ILP) model for both partitioning and scheduling problem for a system specification written in VHDL. The ILP has proven to provide a near optimal solution. Arato et al. in [5] then took the idea and combined ILP with Genetic Algorithm (GA) to perform the HW/SW partitioning. In this work however, the constraints are limited to timing only. It did not look into design space exploration since it was assumed that enough resources were available for the system implementation. In fact, one of the main research work that introduced GA in partitioning is by Srinivasan et al. [16] which introduced researchers to explore GA in HW/SW codesign.

Although a GA based approach may be able to find very good solutions for a variety of applications, the amount of time required for large computations and iterations is enormous. Hence, software implementation of GAs for increasingly complex applications can cause unacceptable delays [17]. Another limitation of this method is that the optimisation depends greatly on the fitness of the chromosome of the GA since GA does not have the ability to calculate changes of part of the chromosome. This issue was raised by Olson et al. in [7] where they have introduced BBN to solve the problem of measuring the effects of a choice made by an independent module. This is in fact the main research work that contributes to the idea of utilising BBN in adapting IP in our framework. In their work, BBN is used to predict the implementation of hardware and software components for their system-based model. The work combines qualitative values of complexity, bandwidth and frequency metrics in deciding the partitioning. In addition to that, the weight for each of the metric is manually provided depending on designers' preferences. At the same time, there is no constraint that needs to be met in the proposed system.

IV. OVERVIEW OF THE PROPOSED FRAMEWORK

This new proposed framework employs the idea of BBN and integrates it with solving CSP. The aim is to provide a reliable partitioning methodology as well as to accelerate the partitioning decision by exploiting the intelligence of the BBN. In many areas, CSP has been used widely in resource allocation and scheduling applications². For this framework, CSP will be used to automatically generate evidence in the BBN. In this work, the *timing* and the *hardware resource* constraints are the two metrics that will be used in formulating the link matrices. This section will elaborate on the idea of how BBN are exploited in the framework. In this work the BBN is used to provide a prediction on which IP blocks (i.e. nodes in the directed acyclic graph) can be moved to either software or hardware based on the direct causal and evidence influences. At the same time, by propagating this knowledge

²<http://cswww.essex.ac.uk/Research/CSP/edward/FCS.html>

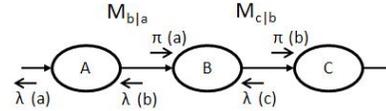


Fig. 3. An example of directed acyclic graph of an embedded system

in the BBN, the framework will see how a change in a node would affect the whole system. By taking this approach, the framework will avoid randomly moving modules into software such as in the conventional heuristic approach, hence, reducing the total computation time in the partitioning process.

Figure 3 is an example of a graph representation of a given system. Each node in the diagram represents a module. For each node, there are three vital information that will be exploited. They are the causal information (π), the evidence information (λ) and the belief, BEL . Equation 1 and equation 2 show how new λ and π are calculated respectively. While λ is evidence that updates the network from bottom-up propagation, π updates the network by propagating new message from parent to child. The belief on each node will indicate the likelihood of it becoming either hardware or software, (HW, SW). The belief of a node is updated by equation 3 where the term α is the normalisation constant rendering $\sum_x BEL(x) = 1$. In between nodes, there are the conditional probability matrix, M , also known as the link matrix. The link matrix is given by the equation 4 where the values are the conditional probability of B becoming either hardware or software given that A has occurred. In other words, $P(HWb|SWa)$ gives the probability of module of node B becoming a hardware given that the module of node A is software. In this framework, the final partitioning decision will be based on the maximum likelihood criterion which is the final belief values. A module will become a hardware if the value of HW is greater than SW .

$$\lambda(b) = M_{c|b} \cdot \lambda(c) \quad (1)$$

$$\pi(b) = \pi(a) \cdot M_{b|a} \quad (2)$$

$$BEL(b) = (HW, SW) = \alpha \lambda(b) \pi(b) \quad (3)$$

$$M_{b|a} = \begin{vmatrix} P(HWb|HWa) & P(SWb|HWa) \\ P(HWb|SWa) & P(SWb|SWa) \end{vmatrix} \quad (4)$$

In addition to the Bayes' theorem, to avoid updating belief by counting any evidence twice, the framework will also obey Jeffrey's rule during the propagation process as described in [18].

V. FRAMEWORK METHODOLOGY

Figure 4 illustrates the proposed HW/SW partitioning framework. The three shaded blocks in the diagram are the three stages where BBN computations are involved. In this work we have interfaced our framework with Hugin Lite, a shareware version provided by the Hugin System³ that will

³<http://www.hugin.com/>

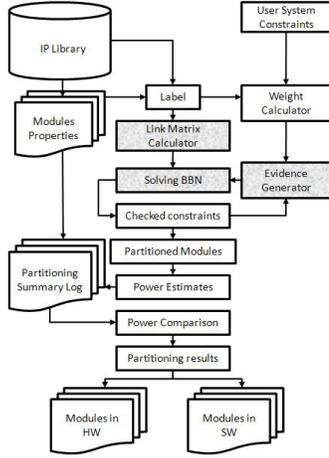


Fig. 4. Proposed partitioning framework

perform BBN related computations. It should be pointed out that at this stage of our research work, all states for power optimisation shown in Fig. 4 has not been integrated into the framework. From the IP library, the hardware metrics i.e. execution time, $T = (t_1, t_2, \dots, t_n)$, and the resources utilisation, $A = (a_1, a_2, \dots, a_n)$, information for each module will be obtained. These two information are fed into the “Link Matrix Calculator” stage where all the link matrices are calculated.

At the same time, the control/data flow graph (CDFG) of the embedded system is obtained by using a high-level synthesis tool. The CDFG is then transformed into a Directed Acyclic Graph (DAG). From the DAG, the given system is now represented by $S = \{E, V\}$, where E is all edges representing data flow while V is all vertices or the nodes which are the functional modules. The module of a system is given by $M = \{m_1, m_2, \dots, m_n\}$. Besides that, the CDFG graph will be used to locate all dependencies especially the case of shared resources (specified in coding) among the functional modules. The graph is later inputted into the “Solving BBN” stage. Together with the link matrices that are calculated, all vertices and the edges will be initialised.

At the same time, the weight for both timing delay and resources are calculated at the “Weight Calculator” stage. The weight is calculated using the equation of $TW_{m,n} = T_{m,n}/MAXT$ ($TW = \{tw_1, tw_2, \dots, tw_n\}$) and $AW_{m,n} = A_{m,n}/MAXA$ ($AW = \{aw_1, aw_2, \dots, aw_n\}$) where $MAXT$ and $MAXA$ is the user-defined timing and area constraints respectively. These weights will be labeled according to each individual module. Besides the weights that are used in solving problem constraint equations, the weight value for each node will also be used as the evidence value. The evidence value to either hardware or software for a node is given by its average node weight plus 0.5.

From the output of BBN, a new total timing weight for the new configuration are calculated for both the time, $\sum tw_i$, where $\forall tw_i \in TW$, and the area, $\sum aw_i$, where $\forall aw_i$

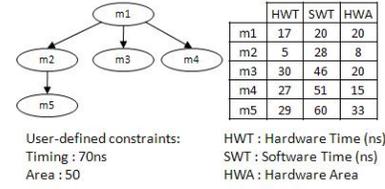


Fig. 5. Example of a tree-structured graph

```

generate evidence;
WHILE (constraints not met)
generate HW/SW configuration from belief;
calculate total area weight;
calculate highest timing weight;
IF (timing and area violated)
  IF (timing violation is worse)
    introduce evidence at modules with
    highest software timing;
  ELSE IF (area violation is worse)
    introduce evidence at module with
    highest hardware area;
  END IF;
ELSE IF (timing violated)
  introduce evidence at modules with
  highest software timing;
ELSE IF (area violated)
  introduce evidence at module with
  highest hardware area;
END IF;
END WHILE;

```

Fig. 6. Pseudo-code on the main function in the framework

$\in AW$ in the “Checked Constraints” stage. Together with this new information obtained from “Checked Constraints”, evidence will be generated. In our framework, we are proposing a framework that will have an automatically generated evidence based on the weight of each module as well as the given user constraints. For a parallel tree-structured such as shown in figure 5, there will be an independent analysis of timing constraints at each branch of the tree to acquire the longest execution time of the system. For example, in the case where m1, m2, m3 and m5 are hardware whereas m4 is a software, although the longest route is from m1 to m2 to m5 ($17 + 5 + 29 = 51ns$), the execution time to complete from m1 to m4 is longer ($17 + 51 = 68ns$). Hence, the value $68ns$ will be tested against the user-defined timing constraint.

The next step is to estimate the total power consumption. If there are more than one acceptable partitioned solutions generated by the framework, the one with the minimum power among the generated solutions will be output from the framework as the final proposed HW/SW partitioning solution. In this work, the power consumption will be estimated using the power estimator tool by Xilinx. Xilinx Power Estimator (XPE) tool estimates power based on design characteristic such as the design resources usage and the clock rate. The XPE tool is useful in this work since it can estimate power consumption as early as prior to the implementation stages. The values used

in the XPE tool such as the number of LUTs and FFs are the same values obtained from the synthesis summary. Together with the synthesis summary and the user-defined data (system frequency, toggle rate and etc), the power consumption will be estimated. At this stage, the modules provided by the user have been divided into sets of modules that are either as hardware ($HW = \{m_x, m_y, \dots, m_z\} \in m, n$) or software ($SW = \{m_i, m_j, \dots, m_k\} \in m, n$). While efficient HW/SW partitioning has proven to be one of the solutions to achieve a low-power design, the final output of this framework might have to trade between power and performance. The BBN will continue to compute a solution as long as there is new evidence to be introduced.

VI. EXPERIMENTAL PROCEDURE

A. Experimental setup

To test the efficiency of our proposed method, we have setup an experiment to compare our framework with the exhaustive search algorithm, greedy algorithm and the previous method given in [7]. We have developed an algorithm that generates synthetic systems of ten modules that have random hardware time, software time and number of slices with different tree-structured configurations. The discussion in this section will be based on the system shown in Figure 7. As shown in the figure, the user-defined timing and area constraints is $60ns$ and 700 slices respectively. While the exhaustive search will propagate through the whole combinations of $2^{10} = 1024$ in finding an optimal solution, the greedy algorithm makes a decision based on the ‘best’ choice using information collected locally. In this experiment, the optimal solution is defined as to have the smallest total value of the maximum time and number of slices utilises while meeting the user-defined timing and area constraints. As for our proposed method and the method presented in [7], besides finding a possible solution, the other aim of the experiment is to see if a BBN could converge to a solution that satisfy the user given constraints with respect to the evidence introduced. This is because, the BBN used in [7] have no timing and area constraints that need to be met. The other aim to be achieved in this experiment is to validate the new proposed techniques in generating evidence automatically based on the user-defined constraints as well as the IPs information.

In this experiment, the link matrices given in [7] will be based on the relative equation given in that report. The *time* as well as *area* data will replace the complexity, frequency and bandwidth metrics. For this experiment, we have set the designer preference metrics weight to have the value 1 because all the metric elements (i.e. *time* and *area*) are equally important. Hence, the relative value for *time* and *area* are:

$$rel_{time}(a, b) = \frac{1}{\log(\max(\frac{time_a}{time_b}, \frac{time_b}{time_a})) + 0.1} \quad (5)$$

$$rel_{area}(a, b) = \frac{1}{\log(\max(\frac{area_a}{area_b}, \frac{area_b}{area_a})) + 0.1} \quad (6)$$

Now by taking the total of equation 5 and 6, the relative value between two nodes is:

$$rel = rel_{time}(a, b) + rel_{area}(a, b) \quad (7)$$

Hence, the link matrix of node a connected to node b is given by equation 8 where N is the normalised variable.

$$LinkMatrix(a, b) = N \begin{vmatrix} rel & \frac{1}{rel} \\ \frac{1}{rel} & rel \end{vmatrix} \quad (8)$$

In our framework, we have introduced the software weight to equation 8 to achieve better convergence. The software weight is calculated from the software time. Hence, our final link matrix is given by equation 9:

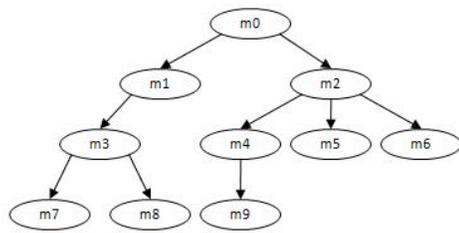
$$Prop_LinkMatrix(a, b) = N \begin{vmatrix} rel & \frac{sw_b}{rel} \\ \frac{sw_a}{rel} & sw_{ab}rel \end{vmatrix} \quad (9)$$

On top of that, for the test of link matrix given in [7] as well as our proposed link matrix equation, the framework is set to take only one evidence at each iteration. At the same time, once a node has been introduced with an evidence, no new evidence can be entered at that particular node.

B. Results

For the system shown in figure 7, the exhaustive method finds the optimal solution at the 703th iteration. The IP configurations that meets both timing and hardware resources constraints are given by $\{m0 \dots m9\} = \{SW, HW, SW, HW, SW, SW, SW, SW, SW, HW\}$. The maximum execution time for this configuration is $69ns$ and utilises 696 slices. For this example, the greedy algorithm manages to converge to the same configuration but at a faster rate. The solution was found at the 47th iteration. Our framework also manages to find a solution to meet user-defined constraints. For this example, the configuration found by our framework is also the optimal solution. Table II shows the changes in the *Belief* values before converging to the solution at the fourth iteration. The *Initial* row shows the *Belief* values for each node once the network is initialised. Each remaining row correspond to an iteration count. Because we have introduced software weight, hence the initial *Belief* values calculated using our proposed link matrix equation will not start at 50 as shown in Table II. All the *Belief* value below 50 indicates a software implementation whereas value of equal and above 50 is hardware implementation. Unfortunately, for the link matrix equation presented in [7], our framework failed to converge to any possible solution as shown in Table III.

From Table III, we could see unsolved data conflict in the *Belief* values which caused our framework to fail from producing a valid solution. With the introduction of evidence to node $m8$ to become a software implementation, node $m3$ which was previously set to become hardware converges to software. This data conflict continue to occur even after an evidence of hardware implementation was entered to node $m1$. However, after node $m9$ was correctly selected to become hardware, node $m3$ changes state to become hardware again. In



	HWT	SWT	HWA
m0	15	25	511.00
m1	7	17	162.00
m2	6	9	199.00
m3	3	17	421.00
m4	11	20	400.00
m5	6	19	233.00
m6	13	25	511.00
m7	10	21	123.00
m8	10	24	521.00
m9	1	15	113.00
	82.00	192.00	3194.00

User-defined constraints:
Timing : 60ns
Area : 700 slices

Fig. 7. An experimental setup of a system with ten modules flow diagram

TABLE I
EVIDENCE VALUE FOR FRAMEWORK

	m0	m1	m2	m3	m4	m5	m6	m7	m8	m9
average weight	0.25	0.13	0.10	0.13	0.19	0.14	0.24	0.16	0.21	0.07
evidence value	0.75	0.63	0.60	0.63	0.69	0.64	0.74	0.66	0.71	0.57

TABLE II
EXPERIMENT USING PROPOSED LINK MATRIX EQUATION

Node	m0	m1	m2	m3	m4	m5	m6	m7	m8	m9
Initial	50.00	45.52	45.14	42.80	40.83	42.17	42.33	43.67	39.75	45.68
m3, HW	53.12	51.96	46.68	55.71	41.73	43.56	43.11	48.73	48.24	45.91
m0, SW	27.07	38.89	34.12	49.27	34.39	32.31	36.71	46.20	44.00	43.99
m1, HW	32.51	51.96	36.73	55.71	35.92	34.65	38.04	48.73	48.24	44.39
m9, HW	32.97	52.18	37.74	55.81	37.63	35.55	38.56	48.77	48.31	51.41

TABLE III
EXPERIMENT USING THE LINK MATRIX EQUATION AS PRESENTED IN [7]

Node	m0	m1	m2	m3	m4	m5	m6	m7	m8	m9
Initial	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00
m5, SW	41.22	44.21	36.77	46.09	39.77	36.32	40.84	47.82	46.85	45.88
m3, HW	46.81	52.83	40.32	58.97	42.52	39.74	43.30	55.01	57.23	47.00
m8, SW	39.07	40.89	35.40	41.13	38.71	35.00	40.00	45.05	34.86	45.45
m1, HW	47.60	54.04	40.82	49.96	42.90	40.22	43.64	49.98	41.46	47.14
m9, HW	49.37	57.59	41.41	53.29	44.06	40.70	44.55	51.43	39.56	60.44
m7, SW	46.96	53.36	40.04	46.23	43.04	39.39	43.68	35.08	35.25	56.21

the next iteration however, node $m7$ was selected to be made into software which again affected node $m3$ to return back to a false state of software. At this point the total slice utilisation was 275 slices while the maximum timing is 73ns which means that some nodes must be made as hardware. And since that node $m3$ has been introduced with evidence as hardware, this would mean a false state will happened in the next iterations. As for our proposed method, although data conflict occurs at node $m3$ after node $m0$ is selected as software as shown in Table II, the data conflict was rectified when node $m1$ was selected to become hardware implementation. This shows the strong influence information in the link matrix between node $m1$ and $m3$.

It is important to highlight it here that the method to generate evidence for the link matrices used in [7] is the same as has been done for the proposed link matrix equation. Because the candidate that was selected by the framework are made based on the *Belief* values at each iteration, hence, the

sequence of which candidate to be introduced with evidence as well as to whether the node is made hardware or software will differ between our approach and the link matrix given in [7].

VII. DISCUSSION AND CONCLUSION

This paper presents a fully automatic BBN-based framework for adaptive IP-reuse. In this proposed framework, we looked into two fundamental requirements of a real-time embedded system characteristic - *timing* and *resources constraints*. The main challenge of utilising BBN in this framework is to establish reliable equations to represent the link matrices. The link matrices in the bayesian network are important since they store the knowledge and the influence information between nodes. The work presented in [7] introduce an equation to formulate the link matrices values. It is based on how "similar" are the two functional units. This is known as the relative equation as shown in equation 8. To improve the influence factor, we have introduced software weight to the

relative equation. As described in section VI-B, our method manage to avoid data conflict that could lead to random or false states. From the experiment, we have shown that 1) BBN can converge to a solution that meets user-defined time and resource constraints, 2) software weight in link matrix equation can improve convergence in BBN and 3) evidence for the BBN can be generated automatically by satisfying the problem constraint equations. While the experimental result is promising, there are much work that needs to be done to improve the robustness of the framework. In our future work, we will incorporate the power estimation states to select a solution with optimal power consumption. At the same time, we would like to extend the framework to be able to solve the mapping process i.e. to be able to predict or decide which platform is the best architecture for a particular IP block. This is viable since new “information” can be incorporated into a BBN-based framework.

Although the smart camera application is main concept that propel this research work, we believe that the framework can also be used for other generic embedded system designs.

ACKNOWLEDGMENT

NICTA is funded by the Australian Government via the Department of Broadband, Communications and the Digital Economy, as well as the Australian Research Council through the ICT Centre of Excellence program.

REFERENCES

- [1] P.-M. Jodoin, J. Konrad, and V. Saligrama, “Modeling background activity for behavior subtraction,” 2008, pp. 1–10.
- [2] A. Azman, A. Bigdeli, Y. Mustafah, and B. Lovell, “Optimizing resources of an fpga-based smart camera architecture,” in *Digital Image Computing Techniques and Applications (DICTA'07)*, 2007, pp. 600–606.
- [3] A. La Rosa, L. Lavagno, and C. Passerone, “Hardware/software design space exploration for a reconfigurable processor,” in *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, 2003, p. 10570.
- [4] G. Michell and R. Gupta, “Hardware/software co-design,” in *Proceeding of the IEEE*, 1997, pp. 349–365.
- [5] P. Arato, S. Juhasz, Z. Mann, A. Orban, and D. Papp, “Hardware-software partitioning in embedded system design,” *IEEE International Symposium on Intelligent Signal Processing*, vol. 2003, pp. 197 – 202, 2003.
- [6] T.-Y. Ma, Z.-Q. Li, and J. Yang, “A novel neural network search for energy-efficient hardware-software partitioning,” in *International Conference on Machine Learning and Cybernetics*, 2006, pp. 3053–3058.
- [7] J. Olson, J. Rozenblit, C. Talarico, and W. Jacak, “Hardware/software partitioning using bayesian belief networks,” in *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 2007, pp. 655–668.
- [8] N. Isaac, *Handbook of Medical Imaging: Processing and Analysis*. San Diego, CA: Academic Press, 2000.
- [9] R. Dechter, *Constraint Processing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.
- [10] Y. Mustafah, A. Azman, A. Bigdeli, and B. Lovell, “An automated face recognition system for intelligence surveillance: Smart camera recognizing faces in the crowd,” in *International Conference on Distributed Smart Cameras (ICDSC'07), ACM/IEEE*, 2007, pp. 147–152.
- [11] D. Gajski, A.-H. Wu, V. Chaiyakul, S. Mori, T. Nukiyama, and P. Bricaud, “Essential issues for ip reuse,” in *Proceedings of the ASP-DAC Design Automation Conference*, 2000, pp. 37–42.
- [12] M. Jacome and H. Peixoto, “A survey of digital design reuse,” *IEEE Design and Test of Computers*, vol. 18, pp. 98–107, 2001.
- [13] T. Wiangtong, P. Cheung, and W. Luk, “Comparing three heuristic search methods for functional partitioning in hardware-software codesign,” *Journal Design Automation for Embedded Systems*, vol. 6, pp. 425–449, 2002.
- [14] R. Gupta and G. D. Micheli, “Hardware-software cosynthesis for digital systems,” *IEEE Design and Test of Computers*, vol. 10, no. 4, pp. 29–41, 1993.
- [15] R. Nieman, *Hardware-Software Co-Design for Data Flow Dominated Embedded Systems*. United States: Kluwer Academic Publisher, 1998.
- [16] S. V. Srinivasan, S. Radhakrishnan, and R. Vemuri, “Hardware software partitioning with integrated hardware design space exploration,” in *Design, Automation and Test in Europe (DATE'98)*, 1998, pp. 28–35.
- [17] S. Narayanan and C. Purdy, “Hardware implementation of genetic algorithm modules for intelligent systems,” vol. 2, Aug. 2005, pp. 1733–1736.
- [18] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference*. San Mateo, CA: Morgan Kaufmann Publisher, 1988.