IET Computers & Digital Techniques

Research Article

IET Journals
The Institution of Engineering and Technology

Accuracy-aware processor customisation for fixed-point arithmetic

ISSN 1751-8601
Received on 19th October 2014
Revised on 11th March 2015
Accepted on 12th April 2015
doi: 10.1049/iet-cdt.2014.0188
www.ietdl.org

Shervin Vakili ✉, *J.M. Pierre Langlois, Guy Bois*

Department of Computer Engineering, Polytechnique Montréal
✉ *E-mail: shervin.vakili@polymtl.ca*

Abstract: Application-specific customisation of micro-processor architectures has been widely accepted as an effective way to improve the efficiency of processor-based designs. In this work, the authors propose a new processor customisation method based on fixed-point word-length optimisation. Accuracy-aware word-length optimisation (WLO) of fixed-point circuits is an active research area with a large body of literature. For the first time, this work introduces a method to combine the WLO with the processor customisation. The data type word-lengths, the size of register-files and the architecture of the functional units are the main target objectives to be optimised. Accuracy requirements, defined as the worst-case error bound, is the key consideration that must be met by any solution. A custom processor design environment, called PolyCuSP, is used to realise the processor architecture based on the solution found in the proposed optimisation algorithm. The results achieved by evaluating five benchmark show that this method can reduce the number of necessary LUTs and flip-flops by an average of 11.9% and 5.1%, respectively. The latency is also improved by an average of 33.4%. Moreover, the method was further examined through a case study on a JPEG decoder. The results suggest 16.2% and 56.2% reduction in area consumption and latency, respectively.

1 Introduction

Application-specific processor customisation is one of the promising trends to promote the efficiency of processor-based designs. This trend includes various state of the art research areas such as instruction-set customisation in extensible processors [1], micro-architectural customisation in parameterisable processors [2] and application-specific processor design offered in architecture description languages (ADLs) [3]. In this work, we introduce a novel processor customisation approach which explores a new dimension in application-specific micro-architectural optimisation targeting fixed-point applications.

This new dimension is the word-length of the datapath that is normally fixed in microprocessors. In integer computation, the minimum required word-length of the datapath is determined by the maximum range of the data elements in the applications. Customising the word-length of the processor to this value can potentially improve the efficiency of the processor depending on the application.

In fixed-point computation, the problem of word-length allocation is considerably more complex because of the introduction of new factors. Each fixed-point value is comprised of integer and fractional parts. The integer word-length (IWL) of each signal should be long enough to guarantee overflow/underflow avoidance. This lower-bound requirement can be found by range analysis using various existing analytical [4, 5] and simulation-based techniques [6].

Determining the fractional word-length (FWL) is inherently more complex. The FWL of each signal determines the quantisation error which is introduced because of the finite word-length representation of that signal. This quantisation error can propagate through the subsequent levels of the circuit and eventually show up at the outputs as inaccuracy in the computations. Various analytical [7, 8] and simulation-based techniques [9, 10] were introduced in the literature to model the finite-precision error of a circuit based on the word-length allocation of its signals. Reducing the word-length of signals can significantly improve the efficiency of the implementation. The problem of finding the most efficient word-lengths to represent the signals of a given application is widely known as word-length optimisation (WLO). In fixed-point designs, WLO adjusts the IWL and FWL allocated to each signal considering the overflow/underflow hazards and the accuracy requirements. The efficiency can be measured from the hardware area, power consumption, performance or a combination of them based on the design objectives.

There are basically two word-length allocation approaches. The traditional uniform word-length (UWL) allocation approach offers a single word-length for all variables. The multiple word-length (MWL) approach allows different word-lengths for different variables. Fig. 1 represents processor customisation via UWL and MWL approaches. The important customisable elements in the proposed method include the word-length of the register-files, pipeline buffers and functional units and the number of words in each register-file.

WLO has been extensively studied in numerous researches for application-specific integrated circuit designs. However, to the best of our knowledge, no related research work has been considered in custom processor design. The main contribution of this paper is to present the first work of literature that investigates WLO for application-specific customisation of microprocessors by exploring architectural trade-offs. This is illustrated in Figs. 1*a* and *b*.

More precisely, this work proposes a method for accuracy-guaranteed optimisation of the processor word-length for fixed-point applications. This method aims to enhance the efficiency of the processor architecture through application-specific customisation, while meeting the precision requirements.

The architecture of the functional unit is the other target that the proposed method aims to optimise in parallel with the WLO. Complex arithmetic functions such as multiplication commonly have a significant impact on area usage and performance of the processors. There are usually various possible architectures to realise an arithmetic function in hardware. The efficiency of using a specific architecture in a design depends on the application and the word-length allocation. The proposed method customises the number of hardware operators and architecture of each one regarding the word-length allocation solutions.

Finite-precision error modelling is an essential part of any WLO method. Such a model formulates finite precision error at the

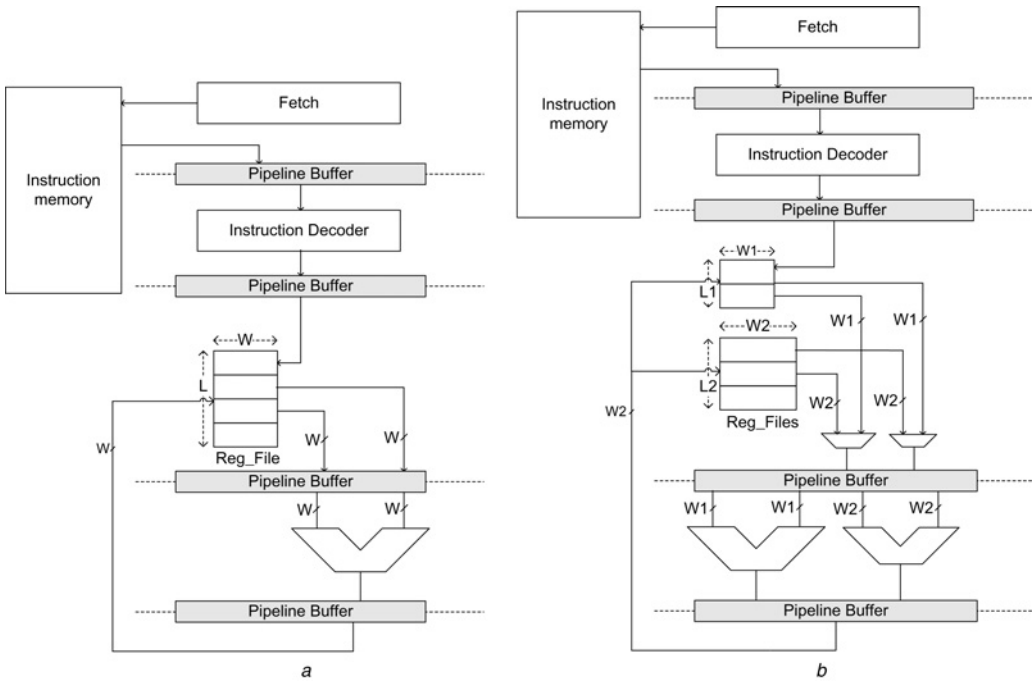

Fig. 1 *Comparing UWL and MWL approaches*

a Processor with customised bitwidth using UWL. In conventional processors, *W* is a fixed power-of-two value, for example, 32
b Processor with customised bitwidth using MWL with two word-lengths

outputs in terms of the FWL of the inputs and the intermediate signals. Given the error model, the UWL can be easily calculated in the UWL approach. However, the MWL optimisation is an NP-hard problem that is normally solved by heuristic search algorithms [11]. The proposed method explores both UWL and MWL approaches in its optimisation algorithm.

The rest of the paper is organised as follows. Section 2 reviews background material and related works. Section 3 describes the proposed methodology. The design flow of the proposed method is described in Section 4. Section 5 presents the optimisation algorithm which is used in this work. Section 6 gives experimental results and comparisons, and Section 7 concludes the paper.

2 Related works

This work is composed of two major parts including an environment for custom processor design and an optimisation algorithm to find the appropriate solution for processor customisation via design space exploration. There is a significant amount of prior work on each of these two parts.

Various trends in custom processor design have been investigated in the literature. Tensilica Xtensa [1], MetaCore [12, 13] and SC build [13] are some examples of partially customisable processor environments in which the main body of the processor is fixed, while a limited number of elements or components are left customisable. ADLs such as PEAS III [14], LISA [15] and EXPRESSION [16] offer designing from scratch, which provides higher flexibility by allowing the designers to define their own instruction-set architecture (ISA) and datapath at the expense of more design complexity.

Yiannacouras *et al.* [17] introduced a soft processor design environment, called soft processor rapid exploration environment, that facilitates design space exploration for both micro-architectural and ISA customisation.

Vakili *et al.* [18] presented a customisable processor design environment, called PolyCuSP (polytechnique customised soft processor), that bridges the gap between ADLs and extensible soft processors. The main characteristic of this environment is to facilitate rapid design space exploration, while preserving a wide range of customisation flexibility. PolyCuSP offers full flexibility in instruction-set description, while limiting the datapath customisation to a predefined set of tunable microarchitectural parameters. This environment is used for implementation and evaluation of the proposed method in this paper.

Range analysis, finite-precision error modelling and FWL selection algorithm are the three main parts of all WLO methods. All these parts have been extensively studied in existing works. Interval arithmetic (IA) is one of the basic methods for analytical range analysis and error modelling [19]. One drawback of IA is that it ignores the correlation among signals [7, 20]. Affine arithmetic is a preferable approach that addresses the correlation problem by taking into account the interdependency of the signals [21].

Le Gal *et al.* [22], Constantinides *et al.* [21] and Menard *et al.* combined the word length optimisation and high-level synthesis (HLS) problems. These works propose new HLS methodologies which take care of data word length in scheduling, allocation and binding processes to aim at optimising the hardware implementation.

Menard *et al.* [23] introduced a grouping algorithm to optimise the resource sharing paradigm for the operations. This process is followed by a WLO algorithm that optimises the word length of each signal group. The algorithm is composed of a greedy and a Tabu search procedures. The optimisation time and the efficiency of the results are the two key measures to evaluate and compare the WLO algorithms.

Sulaimal *et al.* [24] presented a multi-objective genetic algorithm (GA) for real-time optimisation of word-length in a fast Fourier transform (FFT) processor. This is one of the rare works that consider the word length issue in processor architecture. The multi-objective GA is used to find the bit width solution for the FFT coefficients that optimises the precision and power consumption. In that work, the objective is to reduce the power consumption by reducing the number of bits that participate in computations instead of optimising the hardware. Only a single word-length solution is considered in that work.

Finally, some authors have also considered resource sharing capability in their WLO algorithms [21, 23]. This can significantly increase the complexity of the optimisation algorithms. However, till now, WLO has not been studied for a microprocessor domain

that offers highest level of resource sharing. Inherent advantages of microprocessors, particularly their fast and relatively easy design process, make them a popular platform for computations including fixed-point ones. In conventional processors, the word-length is normally a fixed value, equal to a power of two. More advanced processors may support more than one word-length aiming at handling multiple data types effectively. The data type word-lengths are extremely effective on efficiency factors of the processors including their hardware area, power consumption and speed. If the word-length of a processor is wider than the width required by the application, a part of the efficiency is wasted. This problem is more significant in embedded systems where the target functionality, and consequently the application that is to be executed in the processor, are usually fixed.

3 Proposed methodology

In this section, we present our proposed methodology. This methodology aims to generate application-specific customised processors for fixed-point applications. New hardware elements and components are the target of customisation in this work. In the following sections, we will first present the objectives of the proposed method in processor customisation. Then we will illustrate these objectives in detail using an example.

3.1 Methodology objectives

Our proposed method is based on the pursuit of three objectives.

The first objective of the proposed method is to improve the efficiency of the processor architecture by customising the word-length of the data elements and consequently the calculations. The word-length has direct impact on the area cost and speed of various parts of a processor. Any reduction in the word-length of a processor can lead to a significant improvement of overall efficiency. The proposed approach supports the MWL scheme that allows using multiple data types with different word-lengths in the customised processor. Although using multiple word-lengths may increase the complexity of the hardware realisation and the optimisation problem, it also increases the potential of reaching more efficient solutions.

The second objective is to improve the efficiency of the processor architecture by customising the depths of the register-files. One register-file is dedicated to each selected word-length. The minimum depth required for each register-file depends on the application and the word-length allocation. In addition to the area usage, the depths of the register-files also impact the bitwidth required to index the registers in the instruction and consequently the bitwidth of the instructions and related memory units.

The third objective is to improve the efficiency of the design by customising the architecture of the functional units. There are usually many possible architectures to realise a unique operator. The latency, area cost and throughput of the operators may vary for the selected architecture. In this work, one of the issues considered in the optimisation algorithm is to select the best architecture to implement the hardware operator. The number of hardware operators needed to be implemented in the execution stage depends on the number of data types and the operators which are required for each data type. Both of these factors are determined through the design space exploration in the proposed optimisation algorithm. The architecture of each operator can also be optimised based on application requirements and the word-length allocation.

3.2 Illustration of the objectives

In this section, we illustrate how the three objectives are met by our optimisation method using a design example. Fig. 2 illustrates an example circuit that performs the following calculations

$$\begin{aligned} z1 &= b \times a^5 \\ z2 &= c \times d + c \times b \\ z3 &= d \times e + d \times c \end{aligned} \tag{1}$$

The input values are assumed to be in the range of [0,128). Using the approach described in [25], the error model of the example circuit is calculated as follows

$$\begin{aligned} 2^{-\mathrm{FB}_{z1_q}-1} &\geq 5 \times 2^{-\mathrm{FB}_{a_q}+34} + 2^{-\mathrm{FB}_{b_q}+34} + 2^{28} \times \delta_{s1} \\ &\quad + 2^{21} \times \delta_{s2} + 2^{14} \times \delta_{s3} + 2^{7} \times \delta_{s4} \\ 2^{-\mathrm{FB}_{z2_q}-1} &\geq 2^{-\mathrm{FB}_{c_q}+7} + 2^{-\mathrm{FB}_{b_q}+6} + 2^{-\mathrm{FB}_{d_q}+6} + \delta_{s5} + \delta_{s6} \\ 2^{-\mathrm{FB}_{z3_q}-1} &\geq 2^{-\mathrm{FB}_{d_q}+7} + 2^{-\mathrm{FB}_{c_q}+6} + 2^{-\mathrm{FB}_{e_q}+6} + \delta_{s6} + \delta_{s7} \end{aligned} \tag{2}$$

where

$$\delta_{s1} = \begin{cases} 2^{-\mathrm{FB}_{s1_q}-1}\varepsilon_{s1}, & \mathrm{FB}_{s1_q} < \mathrm{FB}_{a_q} + \mathrm{FB}_{b_q} \\ 0, & \text{otherwise} \end{cases}$$

$$\delta_{s2} = \begin{cases} 2^{-\mathrm{FB}_{s2_q}-1}\varepsilon_{s2}, & \mathrm{FB}_{s2_q} < \mathrm{FB}_{a_q} + \mathrm{FB}_{s1_q} \\ 0, & \text{otherwise} \end{cases}$$

$$\delta_{s3} = \begin{cases} 2^{-\mathrm{FB}_{s3_q}-1}\varepsilon_{s3}, & \mathrm{FB}_{s3_q} < \mathrm{FB}_{a_q} + \mathrm{FB}_{s2_q} \\ 0, & \text{otherwise} \end{cases}$$

$$\delta_{s4} = \begin{cases} 2^{-\mathrm{FB}_{s4_q}-1}\varepsilon_{s4}, & \mathrm{FB}_{s4_q} < \mathrm{FB}_{a_q} + \mathrm{FB}_{s3_q} \\ 0, & \text{otherwise} \end{cases}$$

$$\delta_{s5} = \begin{cases} 2^{-\mathrm{FB}_{s5_q}-1}\varepsilon_{s5}, & \mathrm{FB}_{s5_q} < \mathrm{FB}_{b_q} + \mathrm{FB}_{c_q} \\ 0, & \text{otherwise} \end{cases}$$

$$\delta_{s6} = \begin{cases} 2^{-\mathrm{FB}_{s6_q}-1}\varepsilon_{s6}, & \mathrm{FB}_{s6_q} < \mathrm{FB}_{c_q} + \mathrm{FB}_{d_q} \\ 0, & \text{otherwise} \end{cases}$$

$$\delta_{s7} = \begin{cases} 2^{-\mathrm{FB}_{s7_q}-1}\varepsilon_{ds7}, & \mathrm{FB}_{s7_q} < \mathrm{FB}_{d_q} + \mathrm{FB}_{e_q} \\ 0, & \text{otherwise} \end{cases}$$

Objective 1

The value FB_{s_q} represents the FWL allocated to the signal $s$and ε_x is an uncertainty source in signal x.

We start with the word-length allocation which is the first objective of the proposed method. Analysing the error model reveals that the five multiplications that lie within the path of calculating $z1$ must be much wider than the other operations in the application when the same accuracy is requested for all outputs. For instance, assume that 8-bit fractional accuracy is requested for all outputs. Solving the error inequalities in (2) shows that at least 54 bits are required for a, b, $s1$, $s2$, $s3$ and $s4$ to meet the requested accuracy at output $z1$ and to provide the necessary IWL for the signals. This value is obtained by assuming a uniform word length for all signals to simplify the calculations and by solving the first inequality of (2). However, 26 fractional bits are enough for the b, c, d, e, $s5$, $s6$ and $s7$ signals to guarantee 8-bit accuracy at output $z2$ and $z3$. The IWL of the signals is taken into account in these calculations. Fourteen bits are required for the integer part of signals $s1$, $s5$, $s6$ and $s7$ while the integer part of signals $s2$, $s3$ and $s4$ should be at least 21, 28 and 35 bits, respectively.

A single data type processor that can calculate the above example must have a datapath that is at least 54 bits wide. This is, in fact, the realisation of the UWL approach in the processor domain. The register-file, inter-stage signal routes and the hardware operators in the functional unit, including the multiplier, must also be able to handle this bitwidth. However, we know that 26-bits are enough for three of the multiplications and 7 signals that only participate

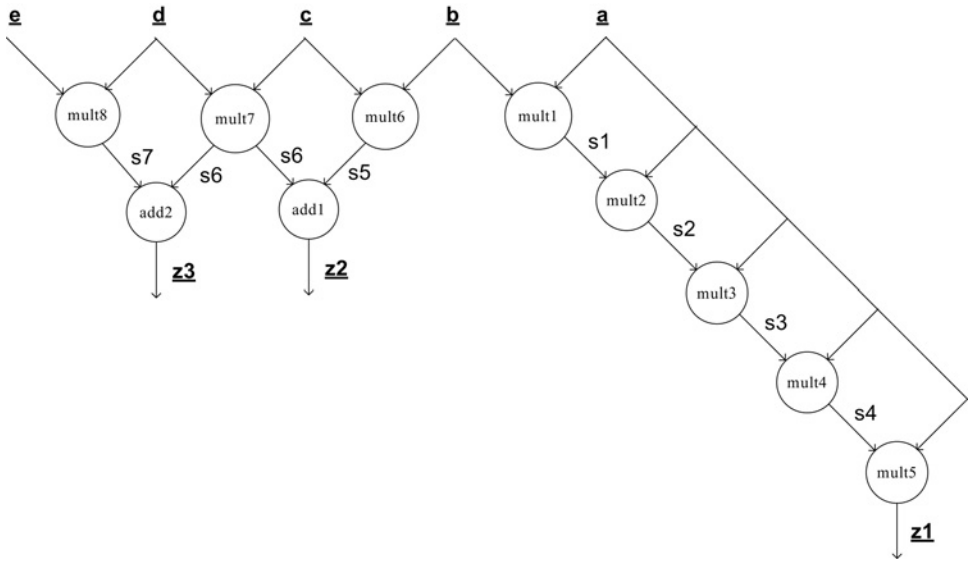

Fig. 2 *Example circuit*

in the calculation of the $z1$ and $z2$ outputs. Converting the processor to a double-word-length architecture with 26- and 54-bit data types can reduce the hardware area of the register-files. Moreover, adding a separate 26-bit multiplier to the processor may enable the design to use more diverse types of multiplier architectures. All these decisions need an effective exploration of a large search space. The mentioned word-length allocation possibilities show the importance of the first objective.

Objective 2

The word-length allocation also has a direct impact on the minimum necessary depth of the register-files as the second objective. In microprocessors, more than one variable can be mapped to the same physical register if there is no time overlap between their living times in the application code. Register allocation algorithms are normally used in compilers to map the registers to the application variables. The register allocation result determines the minimum required depth for the register-files. In the proposed method, variables are divided into different word-lengths in MWL word-length allocation solutions. A change in word-length allocation can change effective factors in the register allocation and can eventually change the minimum required depths of the register-file. For instance, if a 54-bit word-length is selected for variable a and a 26-bit length is selected for variable e, then variable a cannot be mapped to the 26-bit register file. These constraints are considered in the modified register allocation algorithm of the proposed optimisation algorithm described in Section 5. Hence, the register allocation and the required depth of the register-files depend on the word-length allocation. In the proposed algorithm, the word-length allocation is performed through design space exploration.

In the proposed processor architecture, operands of multiple data types can be used by a single instruction. Consequently, increasing the number of data types does not necessitate introduction of new instructions to the ISA as long as new functional units are not added to the architecture. This is realised by using a uniform indexing method for all register-files. The proposed method includes a register allocation algorithm that is responsible for assigning appropriate register indexes to the variables based on data type allocation that is achieved in the first objective.

Objective 3

The third objective focuses on the architecture of complex functions. In this work, we limit our explorations to the multipliers as the most widely used complex function to simplify the presentation. Hardware multipliers are used in most modern embedded processors since multiplication is a basic operation in most DSP and image processing applications. Hence, the proposed method searches for the most efficient multiplier architectures to integrate into the customised processor. However, the method can be easily extended to cover other complex functions such as division and logarithm. A multiplier can be implemented by various architectures with different efficiency characteristics. In this work, we consider three well known architectures: (1) the basic combinational multiplier, (2) the multi-cycle shift-and-add multiplier and (3) the pipelined shift and add multiplier. The multi-cycle and the pipeline architectures divide the multiplier datapath into n fragments, where n is less than or equal to the bitwidth of the operands. Let n be the number of stages in multi-cycle and pipeline architectures, then n is the other configurable parameter that is explored by the optimisation algorithm. The hardware cost of each candidate architecture is measured separately. The results are given to the optimisation algorithm as tables to facilitate the overall cost estimation.

The design space of the multiplier architecture depends on the word-length allocation. Therefore exploration for the best multiplier architecture should be carried out based on a known word-length allocation. To show how different function architectures may be used in a customised processor, we give two possible solutions for the multiplier of the Fig. 2 example. A 2-data type configuration is selected for bitwidth configuration, with signals a, b, $s1$, $s2$, $s3$ and $s4$ assigned to the 54-bit type and the rest to the 26-bit type.

Fig. 3 illustrates these two solutions. The first solution uses a single hardware multiplier unit that is 54 bits wide. Fig. 3*a* represents the time scheduling of the instructions using this solution. In all experiments, the throughput of the execution is supposed to have the highest priority. Therefore the search space of the function architectures is limited to those that do not require extra clock cycles. Different multiplications must be calculated in successive clock cycles as shown in Fig. 3*a*. If a single multiplier is used in the processor, the selected architecture must support a throughput of one multiplication per clock cycle to avoid pipeline stalls. A 54-bit combinational multiplier is large and slow enough to become the critical path in most embedded processors.

The second solution uses a 54-bit multiplier for *mult*1 to *mult*5 and a 26-bit multiplier for *mult*6 to *mult*8. The time schedule of this solution is presented in Fig. 3*b*. Using separate multipliers combined with appropriate ordering of the instructions allows 2-cycle gaps for completion of the calculation in each multiplier. This extra cycle flexibility enables the use of a 2-cycle architecture for the multipliers. Moving from combinational to 2-cycle architecture causes significant reduction of hardware resources and latency. Here, we target single-issue processor architectures that support instruction-level parallelism. The instruction-level parallelism enables utilisation of two multipliers, concurrently.

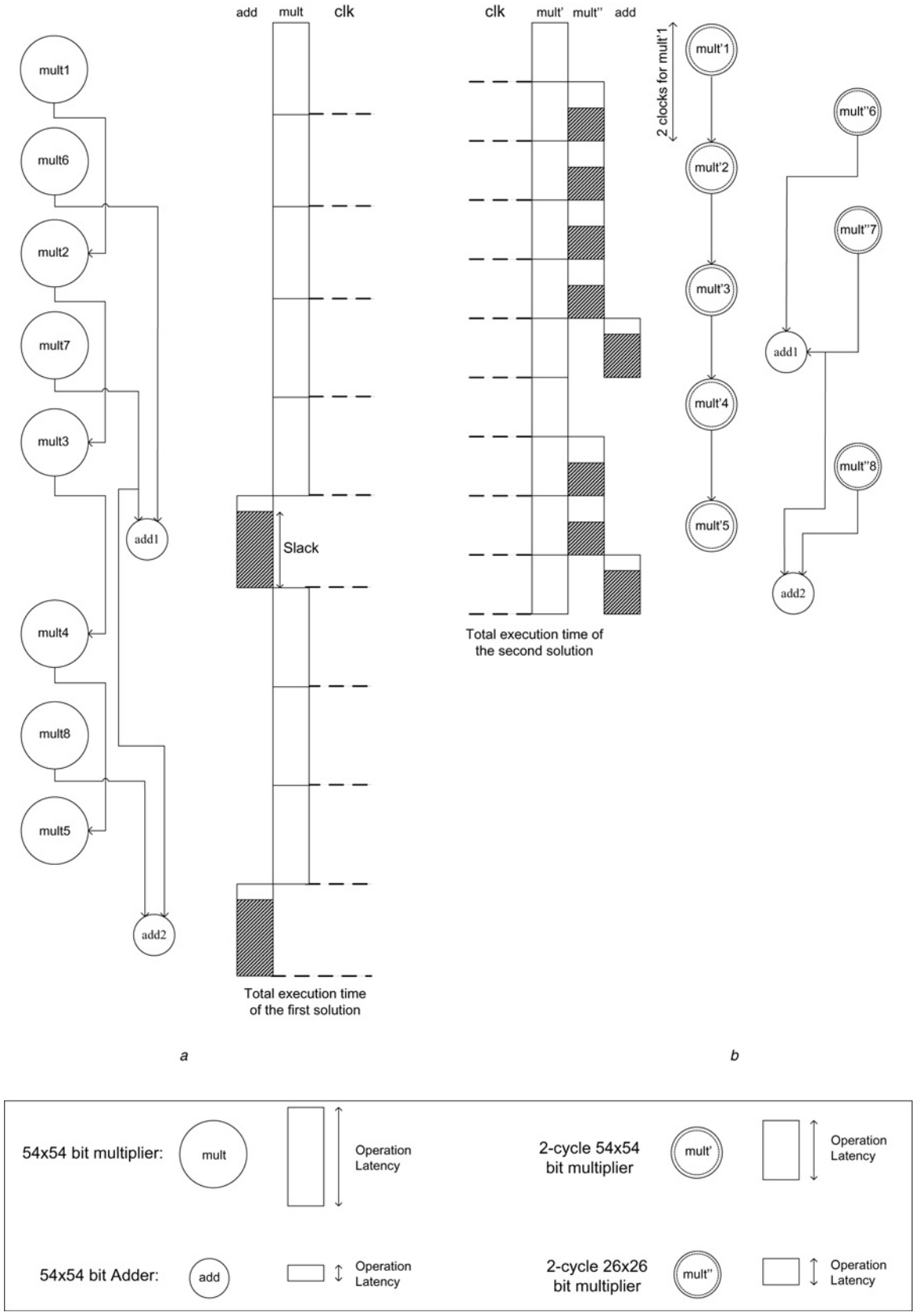

Fig. 3 *Time scheduling diagram of two possible solutions for the Fig. 2 example with*
a Single multiplier
b Double multiplier

Note that the idea of utilising two 2-cycle multipliers can also be used in single data type solution of Fig. 3*a* to improve the clock frequency at the expense of an area overhead. However, in that case, both multipliers must be 54 bits wide that means less efficiency in functional unit hardware compared with the solution of Fig. 3*b*. This example demonstrates how multiple hardware

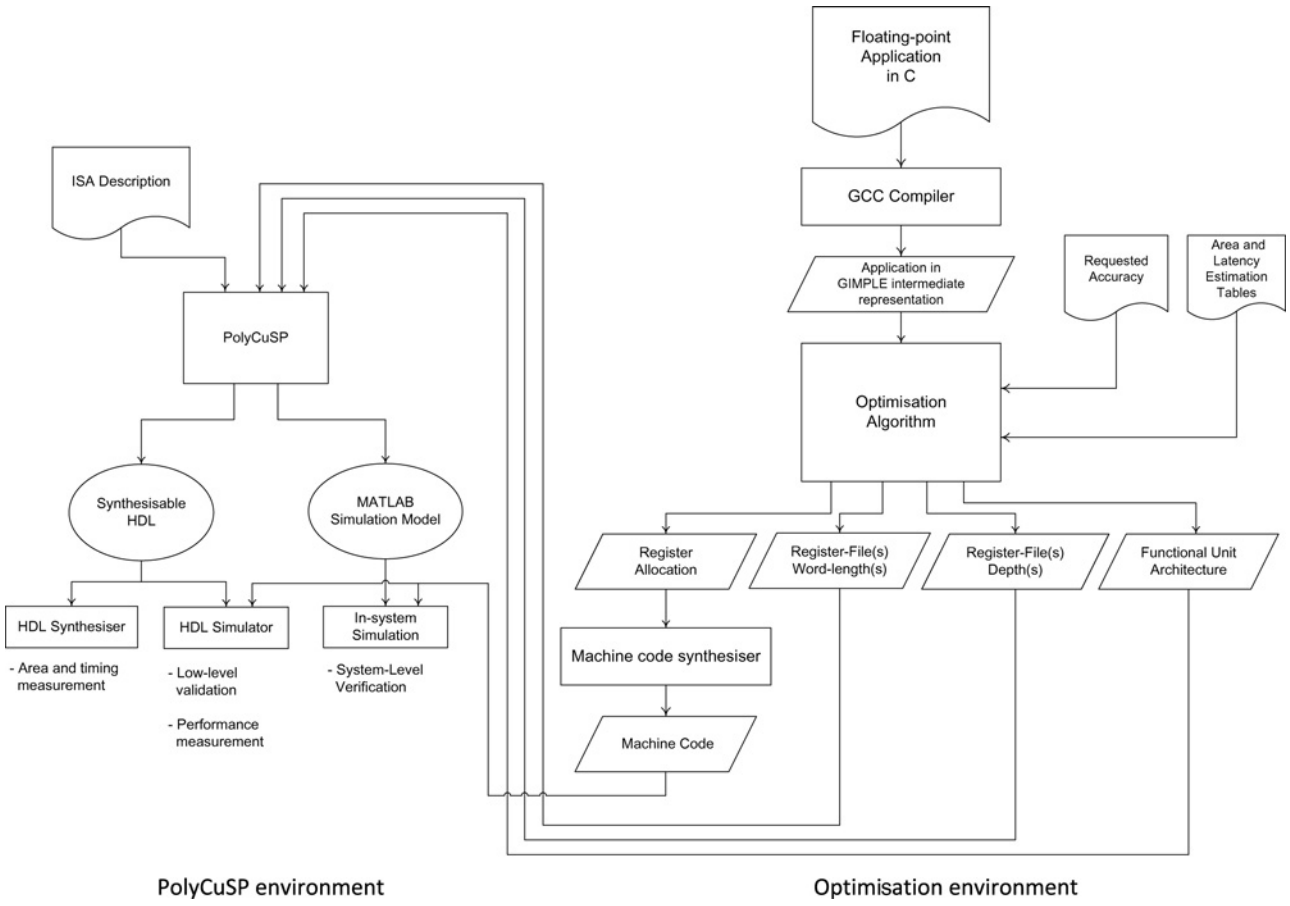

Fig. 4 *Design flow in the proposed method composed of the optimisation and the PolyCuSP environments*

units for a complex operator can improve the efficiency of the design. The optimisation algorithm should explore the possible architectures to find the optimal solution that achieves the highest efficiency.

4 Design flow integration

We propose a complete design flow that incorporates the optimisation and the custom processor generation environments. The optimisation environment is mostly developed in MATLAB. This environment consists of the optimisation algorithm, which will be described in Section 5, and some peripheral units such as cost estimation tables. The processor generation environment creates the customised architecture based on the selected solution in the optimisation algorithm. This section discusses the design flow in more detail.

The design flow is illustrated in Fig. 4. The optimisation process starts with a C application provided by the designer. All fractional variables are represented in floating-point in this input code. In the first step, the input C code is converted into Gimple, which is an intermediate representation of the GCC compiler, similar to a simplified C code. This conversion significantly facilitates code interpretation. The Gimple code is then given to the optimisation algorithm to find a solution to address the customisation objectives listed in Section 3. The selected solution is finally given to an existing processor design environment, called PolyCuSP [18], to generate a corresponding architecture in RT-level VHDL.

The area and latency estimation tables are given to the optimisation algorithm to be used for the fitness evaluation of the candidate solutions. These tables contain the cost estimations of different possible values of the customisable elements. The contents of these tables are obtained from the static synthesis of related components of different sizes. For example, one important table gives the estimated area and maximum achievable frequency for different word-lengths of the datapath regardless of the architecture of the other parts of functional unit. The datapath word-length is equal to the word-length of the longest datapath.

The machine code synthesiser is like a compiler backend that converts the intermediate representation of the application into machine code. The register allocation solution is generated by the optimisation algorithm and given to this unit. The generated machine code is then sent to PolyCuSP to store in the instruction memory of the output processor.

5 Optimisation algorithm

This section presents a detailed description of the proposed optimisation algorithm. The optimisation algorithm contains two nested GA procedures for word-length allocation (Objective 1) and a dedicated search algorithm to find the best architecture for the functional units (Objective 3). Furthermore, the algorithm has a fitness evaluation routine that determines the fitness of the complete candidate solution using cost and performance estimation. Register allocation is a necessary part of the fitness evaluation that also addresses the second objective of the proposed method The register allocation is performed using a graph colouring algorithm just like in previous works. However, since different word-lengths may be assigned to different variables, in the proposed method, the word-lengths should be taken into account. The flow chart of this algorithm is illustrated in Fig. 5. The following definitions facilitate elaboration of each part of the algorithm

N: number of wordlengths or number of data types in the processor.
W_i: the wordlength of the ith data type $0 < i < N$.
S: number of variables.
V_j: the data type used for jth variable $0 < j < S$, $V_j \in \{1...N\}$.

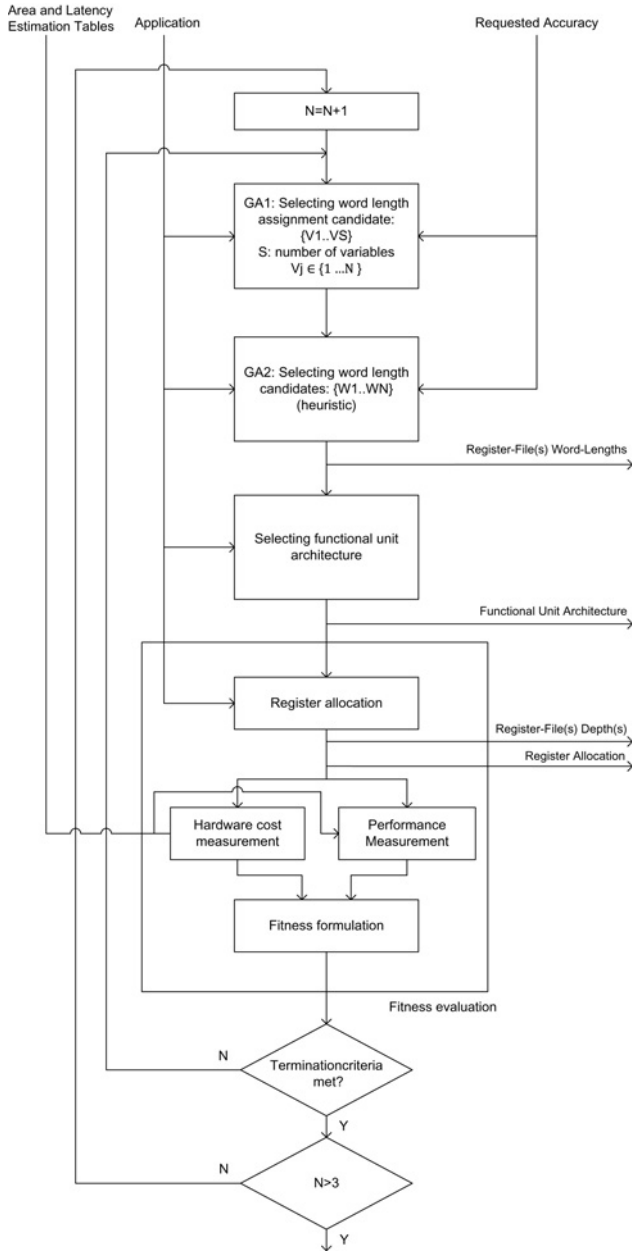

Fig. 5 *Flow chart of the optimisation algorithm*

The different parts of the proposed algorithms are described in the following sections.

5.1 First- and second-level GAs

The number of data types explored in the algorithm, N, is determined manually by the designer. A large value of N increases the chance of finding a more efficient solution. On the other hand, selecting a larger N increases the runtime of the optimisation algorithm. There is no specific rule to determine the best value of N, but successive trials and the experience of the designer can serve as guides. In our experiments, we selected $N = 4$ as the upper-bound, to provide a wide range of flexibility to the optimisation algorithm. In each iteration, the design space is explored for N data types. After completion of the fourth round, the best solution is sent to PolyCuSP to generate the corresponding processor architecture. The solution also includes the register allocation information that is used to generate the machine code of the application from the given GIMPLE code.

The two nested GAs are named GA1 and GA2. On the first level, GA1 searches for the best data type assignment scheme (V_j). Each chromosome is a string composed of S elements, while each element determines the candidate data type for the corresponding variable in that chromosome. The first generation is generated randomly. After generating a population, the second GA procedure starts. On the second-level, GA2 finds the optimal word-length(s) for the data types (W_i) for each chromosome of the given population determined by the first GA. In this phase, each chromosome is composed of $N' \leq N$ word-length values for the N data types. N' is less than N when two or more word-lengths in a chromosome have the same values. To limit the search space, the lower bound and upper bound values of each data type are determined analytically before the start of the GA2.

Only the values between the lower-bound and the upper-bound are searched in the GA2. The lower-bound value of each data type is calculated by assuming infinite word-length for all other data types and solving the output error model with this assumption. The lower-bound values of all data types are calculated first. Then the upper-bound word-length of each data type is calculated by assuming the lower bound word-length for all other data types and the output error model is solved based on this assumption. The upper- and lower-bound word-lengths are highly dependent on the data type assignment scheme. This means that these boundary values may change for each chromosome of the given population determined by GA1. Hence, for each chromosome, the boundary values must be calculated first.

Then, GA2 explores the range between the boundary values of all data types to find the best word-length combination for all data types. Each chromosome in GA2 represents a candidate solution for the word-length of the data types, that is, a candidate W vector. With W and V vectors as the candidates of the first- and second-level chromosomes, the word-length of each variable in the application can be identified as

$$\mathrm{WL}_j = W_{v_j} \tag{3}$$

where WL_j is the word-length assigned to the jth variable. Hence, the combination of one chromosome of GA1 and one chromosome of GA2 gives a complete candidate solution for word-length (or data type) allocation. The register allocation is performed using graph colouring algorithm just like the existing works. However, since different word-lengths may be assigned to different variables, in the proposed method, the word-lengths should be taken into account.

5.2 Architecture selection for functional units

The next step consists of finding the best architecture for each candidate word-length allocation solution. This algorithm can be extended to any complex function. In this work we concentrate on exploring the best architectures for the multipliers.

The developed algorithm, which is presented in Fig. 6, starts by listing the word-lengths of all required multiplications in the given application. For example, in a 2-word-lengths solution with $w1$ and $w2$, at most three multipliers with $w1 \times w1$, $w1 \times w2$ and $w2 \times w2$ widths may be required. This architecture selection algorithm is executed for each candidate of GA2. The aforementioned list of required multiplication widths depends on the word-length allocation and the application.

The narrower multiply operations can always be calculated on a wider multiplier. Therefore this list may differ from the hardware multiplier units that are in the processor. In the next step, the widest required multiplication operation is identified using the application and the word-length allocation information given by the GAs. There should be a multiplier unit that can handle this widest multiplication. Hence, the width of the first multiply unit is known. This multiplier is enough to handle all multiplications in the application but it does not necessarily represent an optimal solution. Adding a shorter multiplier unit may open new areas in the design space of the multipliers. This design space expansion

1. Store all multiplication word-lengths present in the application into vector *M_width*.
2. Find the widest multiplication word-length and add it to the list of candidate word-length for the hardware multipliers in vector *HM_width*
3. **for** all possible combinations of the elements in the *M_width* (excluding the widest one found in Step2)
   {
4. Add the selected combination of the bitwidths to the *HM_width*
5. Find the best architectures for the multipliers with word-lengths that exist in *HM_width* using exhaustive search (each element in the *HM_width* means a hardware multiplier with the same word-length).
6. Delete all elements of the *HM_width* except the widest one (clear the vector for the next candidate)
   }

Fig. 6 *Multiplier selection algorithm*

can enable the use of more efficient architectures just like the example shown in Fig. 3. Therefore the algorithm evaluates addition of shorter multiplications based on the list of required multiplication widths in the application using an exhaustive search.

For each candidate solution, each multiplication operation is assigned to the shortest multiplier that exists in that solution. For example, assume that there are two data types with $w1$ and $w2$ widths while $w1 > w2$. If multiplications with all three possible widths exist in the application, then a multiplier with $w1 \times w1$ width must exist in the processor. If a solution suggests adding a $w1 \times w2$ multiplier, then $w2 \times w2$ multiplications can be assigned to the $w1 \times w2$ multiplier.

Different possible architectures are examined for each multiplier of a candidate solution. The architectures that do not impose further stall cycle in the execution time are evaluated. The total area cost and maximum latency of each solution is estimated. Comparing these estimations, the search algorithm selects the best solution. The designer-defined priorities of area cost and performance are considered in the selection criteria.

5.3 Fitness function

Now, the algorithm needs to measure the fitness of this candidate solution. The fitness of a solution is the efficiency that it achieves when implemented. The GA requires a single fitness value for each chromosome to identify more promising areas in the search space. Therefore the efficiency must be evaluated as a single value that represents a combination of all important efficiency metrics. The weight of different efficiency metrics in the fitness calculation can be adjusted by the designers based on their priorities.

In this work, a formula is developed to calculate a single value to represent the fitness. This formula takes into account four metrics

1. L: Word-length of the largest data type. This parameter defines the bitwidth of the major parts of the datapath.
2. M_{reg}: Amount of on-chip memory resources used for the register-files.
3. A_{mult}: Estimation of the hardware area used for the multipliers.
4. *Freq*: Estimation of the maximum frequency that can be used by the processor.

The fitness formula is as follows

$$\text{fitness} = \frac{C_F \times \text{Freq}}{C_L \times L + C_M \times M_{\text{reg}} + C_A \times A_{\text{mult}}} \quad (4)$$

C_F, C_L, C_M and C_A are weighting factors. For example, increasing the value of C_F will increase the impact of the frequency parameter on the fitness value. The latency of the processor datapath determines the maximum applicable frequency that is the most effective factor of the overall performance of the processor, in this work. The designer can increase the value of C_F to give the higher priority to the performance in optimisation process. This may lead to selection of a faster but more area consuming processor design.

5.4 Termination conditions

The fitness value is returned to GA2, where it is stored for processing. When the fitness evaluation of all chromosomes of one population is completed, the fitness values are compared with identify the best found candidate solutions for GA2, that is, the best W vector.

This process continues until the termination criterion of GA2 is met. Then, the optimisation algorithm terminates and the combination of the best found solutions in the two GA levels (i.e. the best found V vector and the best W vector found for it) is returned as the final solution found by the optimisation algorithm. In this work, the optimisation process terminates when no better result is found for a certain number of iterations.

6 Experimental results

This section presents the results of experimental evaluation of the proposed method. The Virtex V FPGA family was selected as the test-bed platform for implementations and evaluations. ISE 13.2 was used for synthesis and measurements. Modern FPGAs have dedicated DSP and RAM blocks for high performance realisation of arithmetic functions and implementation of local memory. By default, synthesis tools map multipliers into DSP blocks whenever possible. The DSP blocks have fixed word-lengths. As long as the word-length of a supported arithmetic operation, in the design, is less than the fixed word-length of the DSP block, that operation is assigned to one block. For example, each DSP48E1 DSP block of the Xilinx 7 series FPGAs is 48-bits wide and contains a 25×18 multiplier. Hence, in the FPGAs that use these blocks, a 10×10 multiplication and a 18×18 multiplication in the design will both use one DSP block in synthesis, while they are not the same size. This is because, we accept to compromise some computational resources to take advantage of using fast dedicated hardware in DSP blocks. This issue makes the comparisons of the results difficult in our work, because it can hide the impacts of word-length variation on the hardware cost. To solve this problem and to have a unique and illustrative metric for area usage measurement, we avoided utilisation of these dedicated resources in the experiments by adjusting the related settings in the synthesiser.

6.1 Evaluation using five benchmark applications

In the first part of the experiments, fourwell-known benchmark applications along with the Fig. 2 example were used to evaluate the proposed customisation method. The first application is a 126-tap linear-phase low-pass FIR filter with direct form II transposed structure. The second application is an RGB-to-YCrCb converter which is implemented based on the form suggested by the ITU [26]. The third application is an IIR filter of fourth-order [21]. The DCT is an 8-point, one-dimensional decimation in time structure from [27].

We gave higher priority to the area usage in the fitness formulation in our experiments. Regarding (4), this means that C_F was set to a much smaller value compared with C_L, C_M and C_A. The reason is to facilitate demonstration of the effectiveness of the proposed method by focusing on area minimisation as the main optimisation goal.

Table 1 illustrates the hardware cost and the performance results of single-data type and double-data type solutions for the five

Table 1 Hardware cost and performance results of the benchmark applications

Application	# of data types	Type of multipliers	Word-lengths [bits]	# of multi-pliers	Delay of the multipliers	Area		Freq. [MHz] (Imp)
						LUTs (Imp[a])	FFs (Imp)	
FIR	1	combinational	16	1	1 clock cycle	1419 (0%)	630 (0%)	94 (0%)
	2	combinational	13,18	1	1	1288 (9.2%)	585 (7.1)	73 (−22.3%)
	2	pipeline	13,18	1	5	1302 (8.2%)	645 (−2.4%)	141 (50.0%)
	2	multi-cycle	13,18	2	3,1	1213 (14.5%)	619 (1.7%)	122 (29.8%)
RGB-YCrCb	1	combinational	10	1	1	1134 (0%)	266 (0%)	152 (0%)
	2	combinational	9,12	1	1	1092 (3.7%)	252 (5.3%)	142 (6.8%)
	2	pipeline	9,12	2	5,4	1107 (2.3%)	272(−2.6%)	196 (29%)
	2	multi-cycle	9,12	2	2,2	1043 (8.0%)	258 (3.0%)	171 (12.5%)
IIR	1	combinational	12	1	1	1123 (0%)	377 (0%)	142 (0%)
	2	combinational	9,12	1	1	1034 (7.9%)	331 (12.2%)	136 (−4.2%)
	2	pipeline	9,12	2	3,2	1090 (2.9%)	364 (3.4%)	191 (34.5%)
	2	multi-cycle	9,12	2	2,2	977 (13.0%)	352 (6.6%)	177 (24.6%)
DCT	1	combinational	13	1	1	1316 (0%)	565 (0%)	112 (0%)
	2	combinational	10,14	1	1	1192 (9.4%)	511 (11%)	93 (−17%)
	2	pipeline	10,14	2	3,2	1291 (1.9%)	549 (2.8%)	162 (45%)
	2	multi-cycle	10,14	2	2,2	1144 (13.1%)	535 (5.3%)	155 (38%)
Fig. 2 example	1	combinational	54	1	1	2933 (0%)	672 (0%)	16 (0%)
	2	combinational	54,26	1	1	2755 (6.1%)	531 (21.0%)	16 (0%)
	2	pipeline	54,26	2	2,1	2771 (5.5%)	588 (12.5%)	28 (75%)
	2	multi-cycle	54,26	2	2,2	2609 (11.0%)	612 (8.9%)	24 (62.5%)

[a]Imp: % improvement

applications. It also compares the result of employing different multiplier architectures in the MWL approach. The presented area consumption results are for the entire processor core including the register files and all pipeline stages and excluding the instruction and data memories.

The results demonstrate that moving from a single-data type to a 2-data type architecture can significantly improve the area usage and performance of the processor. In these experiments, 8-bit accuracy was requested for the outputs. The results demonstrate how the architecture of complex functions can affect the overall efficiency of the design. We illustrate the best found result by using three different multiplier architectures, separately. New instructions can be fed into the pipeline multiplier in each clock cycle. However, a multi-cycle multiplier does not accept any new input until it completes the last calculation. Therefore the number of stages in pipeline multipliers can be more than the multi-cycle ones. The pipeline multipliers gave better latency results in the experiments. The results include the best found solutions for the data type widths, register-file depths and multiplier architectures. Double word-length solutions with two multi-cycle multipliers give the best area savings in the evaluated benchmarks. Compared with the optimised single data-type processors, these solutions can save LUT and flip-flop resources by an average of 12.9% and 4.9%, respectively, while improving the utilisable clock speed by an average of 41.5%.This is because of the utilisation of the shared resource to calculate different part of the multiplication in successive clock cycles. Note that customised single-data type solutions are used as the reference to measure the improvements. The word-lengths of the processor in these reference designs are adjusted to the value achieved by UWL word-length optimisation. Comparing with the existing processors with fixed power of two word-lengths, the proposed approach can obviously achieve significant improvements.

Fig. 7 shows the run-time progress of the optimisation algorithm for the first three benchmarks. This figure illustrates how the GAs converge towards better solutions. The LUT consumption and the maximum frequency metrics are measured by synthesising the best found candidate solution of each generation with the ISE tool. The results show that the 2-data type solutions achieve the best results for the evaluated benchmarks, while the 3- and 4-data type solutions can reach very close results in most cases. The main reason is that the benchmark applications are not large enough to be able to take advantage of more than two data types.

Increasing the number of data types in a processor can lead to a reduction of the memory usage in the register-files and some other related area resources. It can also increase the area usage in some cases such as the multiplexers that select among the register-files in operand read stage. Hence, there is a trade-off between the savings that more data types can achieve and the overheads that they imply. Fig. 7 also shows that increasing the number of data types results in a longer search time for convergence of the GAs. This is because of the fact that adding a new data type to the processor significantly expands the search space for major parts of the optimisation algorithm.

6.2 Case study of JPEG decoder

In the second part of the experimental results, we conducted a case study on JPEG decoder to measure the effectiveness of the proposed method for more complex multi-module designs. JPEG is a commonly used compression technique for digital images that supports adjustable compression degree. This allows the users to enjoy their preferable trade-off between the size and the quality in image compression.

Fig. 8 represents the block diagram of a JPEG decoder. The variable length decoder) unit decompresses the input image data using a standard Huffman decoding algorithm. During the quantisation a large number of coefficients are rounded to zero. To encode large runs of zeros, JPEG uses run-length encoding [RLE]. To maximise the benefits of RLE, the encoder reorders the elements in the block. In a JPEG decoder, the Zig-Zag scan unit reorders the data elements in the opposite way, such that the elements are placed into their original position. The discrete cosine transform (IDCT) unit transforms data from the frequency domain to the space domain. IDCT is applied on blocks of 8×8 pixels of the image. The DQ unit dequantises the value of the data elements using the information stored in a quantisation table. The Colour Conversion unit is used to convert the colour model of the images from YCbCr to the more commonly used RGB colouring scheme.

The IDCT is the only unit that requires fixed-point calculations. For the experiments, in this section, the requested accuracy for the IDCT calculations complies with the JPEG standard. A public domain JPEG decoder, called PicoJPEG [28], was used as the target application for processor customisation. Unlike the benchmarks of Section 6.1, JPEG decoder is a multi-module design. In such designs, each module has different calculations and therefore may require different data-type word-lengths. The proposed method can benefit from this natural potential of multi-module designs to generate multiple data-type processors.

The fitness function is same as the one used in Section 6.1. Fig. 9 presents the achieved area and performance results. These results

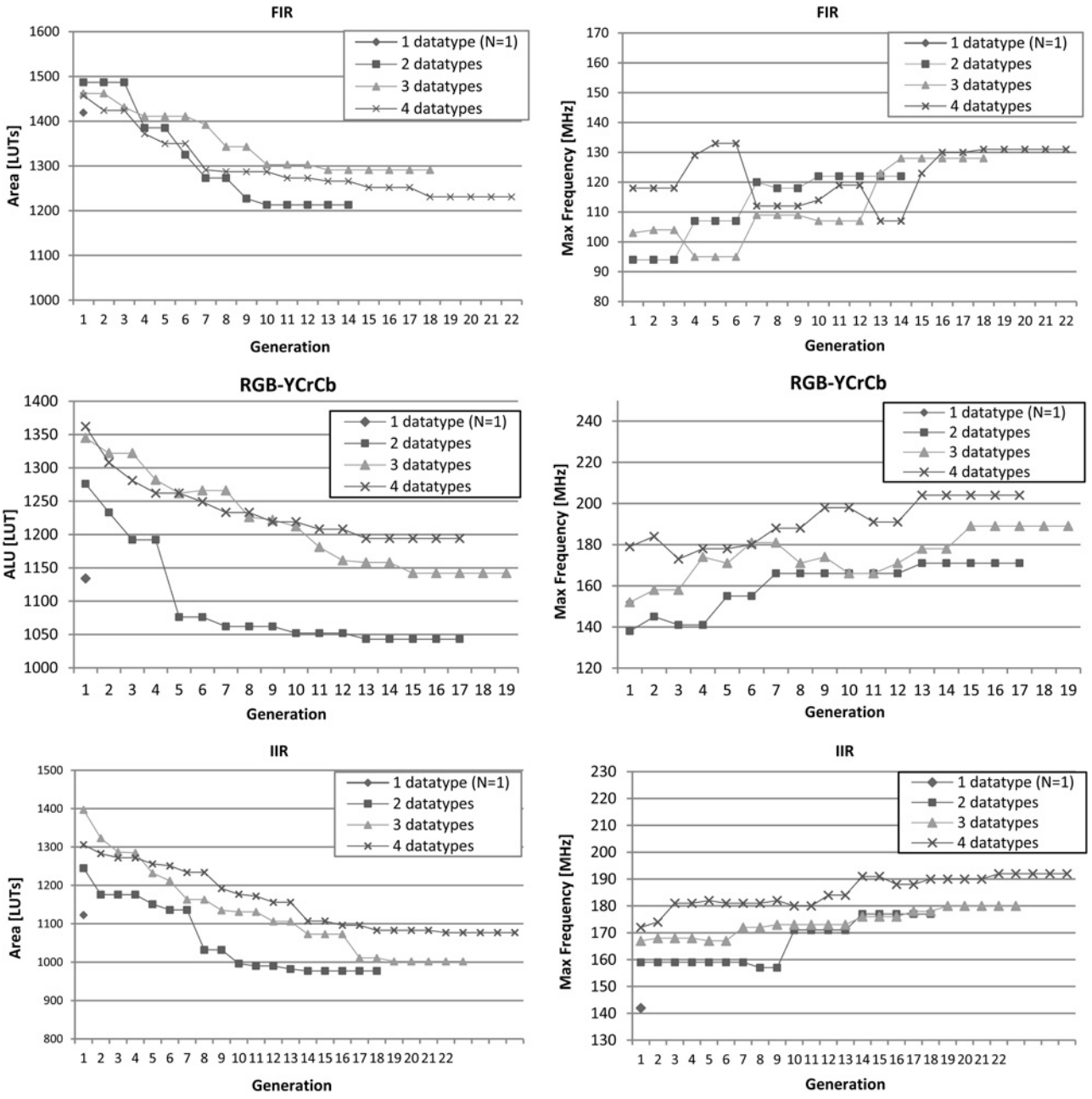

Fig. 7 *Run-time progress in terms of the area usage and the latency of the best found candidate in each generation of the GA1*

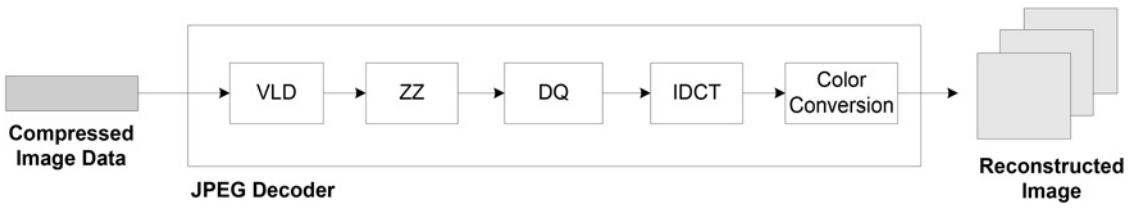

Fig. 8 *JPEG decoder block diagram*

show that the processors with 4 data-types can achieve the best results for this application. The main reason is that distinct units have different word-length requirements. This enables taking advantage of more data-types. The LUT consumptions and the latency of the best found solutions show 16.2% and 56.2% reduction compared with the single data-type configuration, respectively. Comparing the results of this section with those obtained in Section 6.1 demonstrates that increasing the complexity of the application promotes the capabilities of the proposed method to find more efficient processor designs.

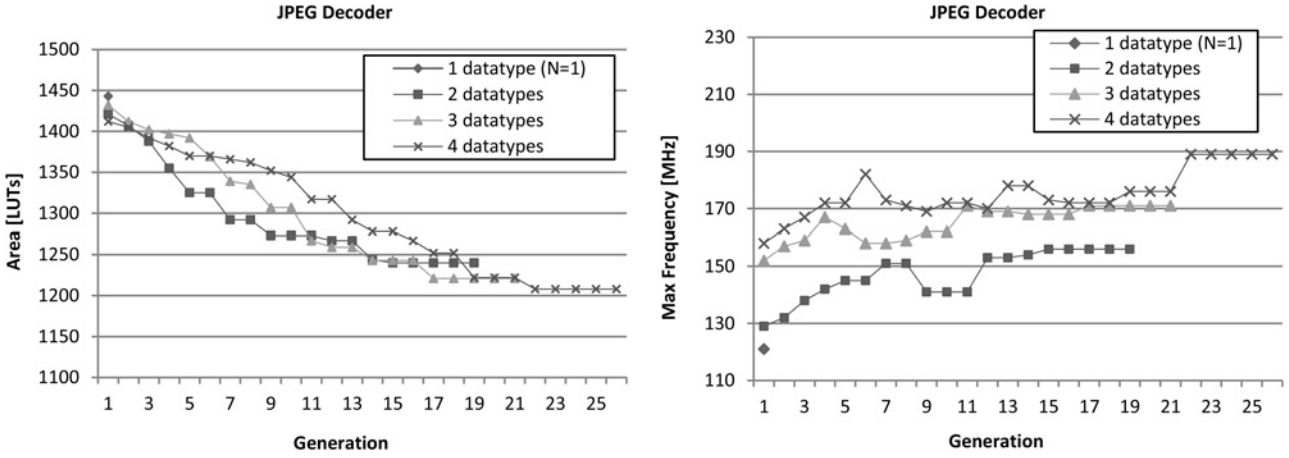

Fig. 9 *Run-time progress for JPEG decoder case study*

7 Conclusion

We proposed a new processor customisation method for fixed-point computations. This method combines the word length optimisation with application-specific processor customisation. The word-lengths of the supported data types, depth of the register-files and the architecture of the functional unit form the customisation targets. A multi-level GA and a dedicated fitness evaluation method were developed for the optimisation algorithm.

Five benchmark applications were used to evaluate the proposed method. The experimental results show that, in the selected FPGA platform, moving from a single word-length processor to a customised double-word-length processor can reduce the area consumption in terms of the number of LUTs and flip-flops by an average of 11.92% and 5.1%, respectively. The results also show an average of 33.4% improvement in the speed of the processor by this customisation. These results demonstrate the effectiveness of the proposed customisation method.

8 References

1 Gonzalez, R.E.: 'Xtensa: a configurable and extensible processor', *IEEE Micro*, 2000, **20**, (2), pp. 60–70
2 Yiannacouras, P., Steffan, J.G., Rose, J.: 'Exploration and customization of FPGA-based soft processors', *IEEE Trans. Comput.-Aided Design Int. Circuits Syst.*, 2007, **26**, (2), pp. 266–277
3 Mishra, P., Dutt, N.: 'Architecture description languages for programmable embedded systems', *IEE Proc. Comput. Digit. Tech.*, 2005, **152**, (3), pp. 285–297
4 Lee, D.U., Gaffar, A.A., Cheung, R.C.C., Mencer, O., Luk, W., Constantinides, G. A.: 'Accuracy-guaranteed bit-width optimization', *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2006, **25**, (10), pp. 1990–2000
5 Yu, P., Radecka, K., Zilic, Z.: 'An efficient method to perform range analysis for DSP circuits'. Int. Conf. on Electronics, Circuits, and Systems (ICECS), December 2010, pp. 855–858
6 Wonyong, S., Ki-Il, K.: 'Simulation-based word-length optimization method for fixed-point digital signal processing systems', *IEEE Trans. Signal Process.*, 1995, **43**, (12), pp. 3087–3090
7 Fang, C.F., Rutenbar, R.A., Chen, T.: 'Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs'. IEEE/ACM Conf. Computer-Aided Design (ICCAD'03), 2003, pp. 275–282
8 Lee, D.U., Gaffar, A.A., Cheung, R.C.C., Mencer, O., Luk, W., Constantinides, G. A.: 'Accuracy-guaranteed bit-width optimization', *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2006, **25**, (10), pp. 1990–2000
9 Lopez, J.A., Carreras, C., Nieto-Taladriz, O.: 'Improved interval-based characterization of fixed-point LTI systems with feedback loops', *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2007, **26**, (11), pp. 1923–1933
10 Caffarena, G., Carreras, C., Lopez, J.A., Fernandez, A.: 'SQNR estimation of fixed-point DSP algorithms', *EURASIP J. Adv. Signal Process*, 2010, **2010**, pp. 1–12
11 Constantinides, G.A., Woeginger, G.J.: 'The complexity of multiple wordlength assignment', *Appl. Math. Lett.*, 2002, **15**, (2), pp. 137–140
12 Yang, J.-H., Kim, B.-W., Nam, S.-J., *et al.*: 'MetaCore: an application specific DSP development system'. Design Automation Conf., April 2000, pp. 800–803
13 Anderson, I.D.L., Khalid, M.A.S.: 'SC Build: a computer-aided design tool for design space exploration of embedded central processing unit cores for field-programmable gate arrays', *IET Comput. Digit. Tech.*, 2009, **3**, (1), pp. 24–32
14 Itoh, M., Higaki, S., Sato, J., *et al.*: 'PEAS-III: an ASIP design environment'. Int. Conf. on Computer Design, 2000, pp. 430–436
15 Chattopadhyay, A., Meyr, H., Leupers, R.: 'LISA: A uniform ADL for embedded processor modelling, implementation and software toolsuite generation', in Mishra, P., Dutt, N. (Eds.): 'Processor description languages' (Morgan Kaufmann, 2008), pp. 95–130
16 Mishra, P., Kejariwal, A., Dutt, N.: 'Synthesis-driven exploration of pipelined embedded processors'. Int. Conf. on VLSI Design, 2004, pp. 921–926
17 Yiannacouras, P., Steffan, J.G., Rose, J.: 'Application-specific customization of soft processor microarchitecture'. Proc. ACM/SIGDA Symp. Field Programmable Gate Arrays, February 2006, pp. 201–210
18 Vakili, S., Langlois, J.M.P., Bois, G.: 'Customised soft processor design: a compromise between architecture description languages and parameterisable processors', *IET Comput. Digit. Tech.*, 2013, **7**, (3), pp. 122–131
19 Moore, R.E., Bierbaum, F.: 'Methods and applications of interval analysis' (SIAM, Philadelphia, 1979)
20 Cong, J., Gururaj, K., Liu, B., *et al.*: 'Evaluation of static analysis techniques for fixed-point precision optimization'. IEEE Symp. on Field Programmable Custom Computing Machines, 2009, pp. 231–234
21 Constantinides, G.A., Cheung, P.Y.K., Luk, W.: 'Optimum and heuristic synthesis of multiple word-length architectures', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2005, **13**, (1), pp. 39–57
22 Le Gal, B., Casseau, E.: 'Word-length aware DSP hardware design flow based on high-level synthesis', *J. Signal Process. Syst.*, 2011, **62**, (3), pp. 341–357
23 Menard, D., Herve, N., Sentieys, O., Nguyen, H.N.: 'High-Level synthesis under fixed-point accuracy constraint', *J. Electr. Comput. Eng.*, 2012, pp. 14
24 Sulaiman, N., Arslan, T.: 'A multi-objective genetic algorithm for on-chip real-time optimisation of word length and power consumption in a pipelined FFT processor targeting a MC-CDMA receiver'. Proc. NASA/DoD Conf. on Evolvable Hardware, July 2005, pp. 154–159
25 Vakili, S., Langlois, J.M.P., Bois, G.: 'Finite-precision error modeling using affine arithmetic'. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), May 2013, pp. 2591–2595
26 Evans, B.L.: 'Raster image processing on the TMS320C7X VLIW DSP'. Available: http://www.ece.utexas.edu/~bevans/hp-dsp-seminar/07_C6xImage2/sld001.htm
27 Parhi, K.: 'VLSI digital signal processing systems' (Wiley, New Yerk, 1999)
28 https://code.google.com/p/picojpeg/, accessed March 2015