Fernandes J, Sokolov D, Yakovlev A.
Elastic Bundles: Modelling and Synthesis of Asynchronous Circuits with Granular Rigidity.
In: 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC). 2017, San Diego, California, USA: IEEE.

Copyright:

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

DOI link to article:

https://doi.org/10.1109/ASYNC.2017.14

Date deposited:

08/12/2017

Elastic Bundles: Modelling and Synthesis of Asynchronous Circuits with Granular Rigidity

Johnson Fernandes, Danil Sokolov, Alex Yakovlev
School of Electrical and Electronic Engineering, Newcastle University, UK
{johnson.fernandes, danil.sokolov, alex.yakovlev}@ncl.ac.uk

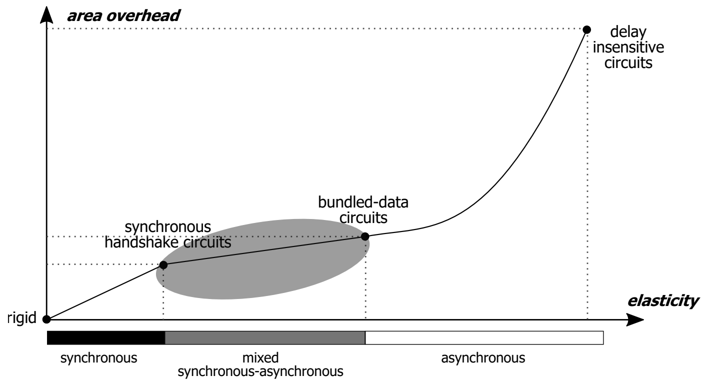
***Abstract*—Elastic circuit design is a revolutionary step in VLSI design paving the way for commercial adoption of asynchronous design techniques. With a growing trend of synchronous-asynchronous CAD tool flow integration, this paradigm shows promise to survive market forces of the semiconductor industry mainly due to scope for reuse of synchronous functional blocks and IP cores, and co-existence of synchronous and asynchronous design styles in a common EDA framework. In this paper, we introduce 'Elastic Bundles', a novel class of elastic circuits, and propose a method for modelling, designing and synthesising these circuits. Starting with a high-level dataflow model of a system, which is natively asynchronous, the key idea is to introduce rigidity of chosen granularity levels without changing functional behaviour. The resulting model is then partitioned into functional blocks of fine-grained and coarse-grained asynchronous elements that would finally be transformed to equivalent circuit descriptions for system logic synthesis using standard EDA tools. The methodology is illustrated using a case study of a 16-point FFT circuit design, which clearly demonstrates a spectrum of solutions that can be achieved in different levels of bundling granularity.**

I. Introduction

Elastic circuit design can be viewed as a revolutionary step in VLSI design paving the way for commercial adoption of asynchronous design techniques in digital circuits. With a growing trend of synchronous-asynchronous CAD tool flow integration [1], [2], [3], this paradigm is showing promise to survive the market forces of the semiconductor industry mainly due to the scope for reuse of synchronous functional blocks and IP cores, and the co-existence of synchronous and asynchronous design styles in a common EDA framework.

We classify the elastic circuits paradigm to encapsulate a range of design practices starting from synchronous handshake circuits [4], which are synchronous circuits embracing principles of asynchronous elasticity, to bundled-data circuits [5], which are asynchronous circuits embracing principles of synchronous rigidity, all the way to delay-insensitive circuits [5], which are the most elastic and robust class of asynchronous circuits. Figure 1, an extension to the illustration of 'Elastic Circuits' classification made in [3], clearly depicts this. As observed, each design technique has distinct consequences to system behaviour and design implementation.

In this paper, our focus is on elastic circuits that provide elasticity with small overhead, specifically the group of circuits encapsulating bundled-data circuits and synchronous handshake circuits, as highlighted in the shaded area of Figure 1. Bundled-data circuits are a class of asynchronous circuits that resemble synchronous behaviour due to its nature of fine-grained local clocking scheme. They can be implemented

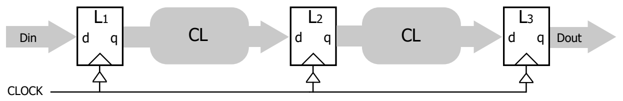

area overhead
delay insensitive circuits
bundled-data circuits
synchronous handshake circuits
rigid
elasticity
synchronous
mixed synchronous-asynchronous
asynchronous
Figure 1: Scope of elastic digital circuits

with the same data path used in synchronous circuits and only differ in the implementation of clock. Several authors have proposed schemes to implement bundled-data circuits using the clocked CAD flow such as [6], [7]. Synchronous handshake circuits fall in the class of synchronous elastic circuits where asynchronous-like elasticity is implemented in a globally clocked domain. Synchronous handshake circuits have been formalised and investigated by several authors in [4], [8], [9], [10].

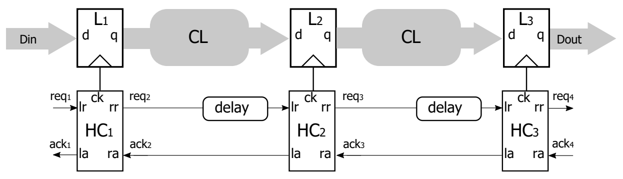
This work introduces 'Elastic Bundles', a novel class of asynchronous circuits with varying levels of rigidity. Elastic-bundle circuits exhibit granular rigidity through a combination of coarse-grained locally clocked elements and fine-grained locally clocked elements. Modelling these circuits starts from a high-level dataflow model of a system which is natively asynchronous. The key idea is to introduce rigidity of chosen granularity levels across the model, without changing functional behaviour, for achieving the design simplicity of synchronous principles. In this manner, the system is partitioned into functional blocks of fine-grained and coarse-grained asynchronous elements based on functional criteria.

The main contributions of this paper are as follows:

- Design flow for development of asynchronous circuits with varying levels of rigidity.
- Novel method for high-level modelling and partitioning of digital systems using Petri nets (PNs). Systems are represented as PN models by adopting a *control data flow graph* (CDFG) format. Partitioning is introduced in these models based on a theory of step persistent bundling.
- Direct mapping of PN models to *register-transfer*



(a) Synchronous pipeline



(b) Bundled-data asynchronous pipeline

Figure 2: Synchronous versus asynchronous pipelines

level (RTL) circuit specifications. PN-based circuit models are translated to RTL circuit descriptions that can be synthesised using EDA tool flows.

- Evaluation of the modelling method and synthesis flow by case study of a 16-point FFT digital architecture. A spectrum of solutions achieved with different levels of bundling granularity is demonstrated in this study.

II. Background

Bundled-data circuits emerged as a design simplification practice for asynchronous circuits by bundling control logic with datapath. In this research, such circuits are optimised further by identifying groups of circuit elements that can be bundled in succession without changing functional behaviour. Such bundling, named as *Elastic Bundles*, is implemented by performing coarse-grain local clocking on bundled-data circuit models. We chose bundled-data circuits as a starting point, as this circuit classification displays the finest granularity of elasticity in the class of elastic circuits that have low overhead. Furthermore, bundled-data asynchronous circuits can share the same data path used for synchronous design as opposed to other asynchronous circuit classes [11]. This is a very strong feature because it enables reuse of existing synchronous functional blocks which improves designer productivity. A bundled-data circuit can be synthesised using traditional clocked CAD flows by replacing the global clock of the synchronous design flow with multiple number of local handshake clocks from the asynchronous control logic [12]. Elastic-bundle circuits can be synthesised in a similar manner using such methods of synchronous-asynchronous CAD tool flow integration. In this paper, we have adopted the *relative timing* (RT) based CAD flow [13] for the synthesis of elastic circuits. Our method, however, is not limited to this tool flow; any other synchronous-asynchronous CAD tool flows can be applied to synthesise elastic-bundle circuits.

A. Bundled-data Asynchronous Pipelines

Pipelining is a principle element of high-performance digital design catering both synchronous as well as asynchronous systems. The fundamental difference between synchronous and asynchronous pipelines lies in the communication channel that enables data items to move from one pipeline stage to the next. Figure 2 illustrates this with a simple example showing a traditional synchronous linear pipeline and an asynchronous linear pipeline. Traditional synchronous systems feature a rigid communication scheme where all pipeline stages operate at a fixed clock frequency. In contrast, a bundled-data asynchronous pipeline achieves communication between pipeline stages by *handshake control* (*HC*) logic blocks which implement fine-grained local locking at registers.

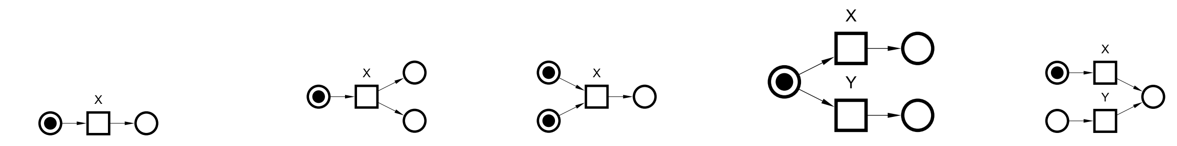
B. RT-based CAD tool flow

Relative timing is a method of modelling and controlling the firing order of two events based on logic path delays. This method can accurately capture, model and validate heterogeneous timing behaviour of synchronous as well as asynchronous circuits [12]. Path-based RT constraints are enforced during circuit synthesis to ensure correct circuit timing and filter out hazardous states. For example, taking the case of the bundled-data linear pipeline in Figure 2b, the following RT constraint: $req_1 \uparrow \mapsto L_2/d + margin \prec L_2/ck \uparrow$ enforces the order that latch L_2 is clocked only after new data is stable at pin d of L_2. This constraint is also responsible for sizing the delay element between stages L_1 and L_2.

The RT-based design flow enables rapid development of asynchronous designs by providing a set of characterised asynchronous templates that can be easily integrated with synchronous CAD tools [12]. These templates can be inserted in designs with supporting clocked CAD tool constraints for synthesis, place and route, timing driven sizing, optimisation and validation. The tool flow enables the adoption of bundled-data asynchronous circuits in the traditional synchronous design flow with little expertise in asynchronous design. First, the bundled-data design is partitioned into data path logic and control logic. The data path is synthesisable using normal synchronous CAD synthesis procedures. Next, handshake clocking is implemented by replacing the global clock with sets of *HC* logic blocks. Characterised asynchronous design elements of the *HC* block, provided by the RT tool flow, are used to implement the control logic. Finally, RT constraints are specified to guarantee correct circuit timing. This process is done either by hand [12] or using the automated tool flow [14]. The constraints are integrated into clocked CAD tool flow by specifying them as design constraints that are supported by commercial tools. For example, RT constraints expressed by *set_max_delay* and *set_min_delay* commands are used to perform timing driven synthesis. The RT constrained architecture is now ready to pass through usual clocked CAD tools and flows.

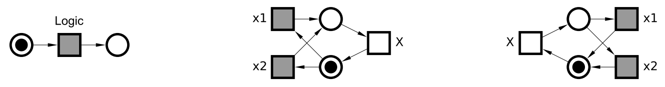
III. Modelling Digital Systems

In our design flow, the *Petri net* (PN) modelling tool is the chosen mathematical language to describe digital systems. The language offered us a convenient tool to capture essential properties of a system from theory formulation to synthesis of control circuits. The Workcraft framework [15] has been used in this research for graphical interpretation and simulation of Petri nets, and for verification of important properties of Petri nets.



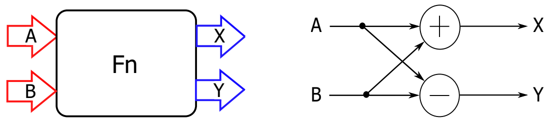
(a) Enabled Transition (b) Fork (c) Join (d) Choice (e) Merge

Figure 3: Basic PN elements



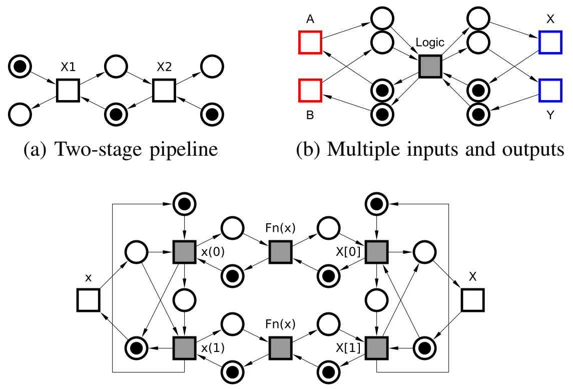
(a) Computation (b) Multiplexer (c) Demultiplexer

Figure 4: PN building blocks for digital systems



(a) System specification (b) Data flow graph

Figure 6: Conceptual design



(a) Two-stage pipeline (b) Multiple inputs and outputs

(c) Wagging [17]

Figure 5: Example of PN model designs

A. *PN building blocks*

Petri nets can capture system behaviour by means of token game semantics where flow of tokens describe net behaviour according to enabling and firing rules. In this paper, we employ the *1-safe labelled PN* class of Petri nets. Assuming their familiarity within the asynchronous community, we invite the reader to refer to [16] for the notations, definitions and basic properties of PNs. Figure 3 shows some of the fundamental elements of PNs that are widely used in modelling systems.

With regards to a sufficient abstract model to describe digital systems, we wanted to use the most compact form of PNs for both asynchronous and synchronous implementations. Digital systems were hence modelled to represent system behaviour on a high-level abstraction of dataflows and flow control, rather than RTL description. Data structures and timing information are excluded in the language description. Such information is labelled on the graph for designer reference. Based on this, we have extracted a set of elements shown in Figure 4 that form the building blocks for digital systems in our framework. The basic PN block shown in Figure 3a is used to describe a buffer which would represent data storage and channel decoupling. Computation blocks are used to denote digital logic from basic gates to complex calculations. The multiplexer and demultiplexer provide capability to route multiple data streams and computations to a single stream, or split a single stream to multiple streams. Since firing of transitions in PNs is instant and atomic, we have annotated logic blocks in grey so that differentiation can be made during design optimisation. We also annotate input places and transitions with colour *red* and output places and transitions with colour *blue*. Figure 5 provides a feel of how these building blocks can be used to describe a range of digital architectures and interaction patterns using PNs [18].

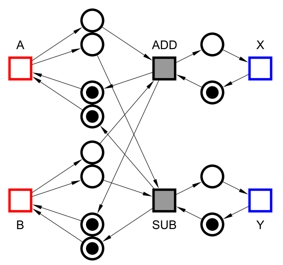
B. *Modelling a Conceptual Design*

In this section, we move on to a specific design example to explain the research concepts of this paper. Let us try to represent a digital system designed to executed a function Fn depicted in Figure 6a. The system processes two input signals, A and B, to produce outputs X and Y that are defined as: $X = (A + B)\,, Y = (A - B)$

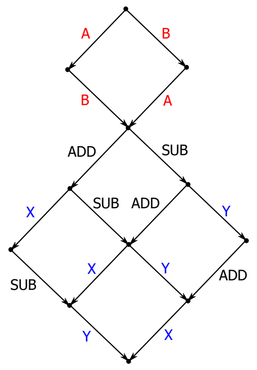
The design is first modelled as a *data flow graph* (DFG) and subsequently, as a PN to illustrate the significance of modelling flow control. DFGs [19] have been widely used to describe digital systems but they are limited as they can not model control. A basic DFG of the system is shown in Figure 6b. PN models represent what is known as a *control data flow graph* (CDFG). Figure 7a shows the CDFG of the conceptual design modelled as a PN. The functional behaviour of the design is depicted in Figure 7b based on scoping of reachable states. It can be visualised that the PN model describes the digital system in its native form featuring highest level of concurrency. Hence, we also treat this model as an elastic or asynchronous description of the design.

C. *Partitioning with Bundles*

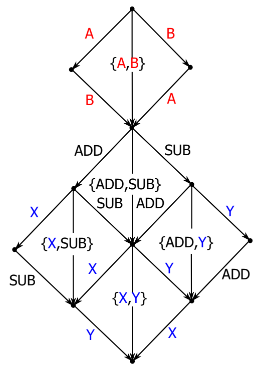
In this section, we implement the principles of bundles with *step persistence* [20] satisfaction to partition the conceptual design based on functional criteria whilst ensuring hazard-free step execution circuit behaviour. Bundles are identified by pruning a concurrent reachability graph (CRG) of a system into a set of persistent steps [20]. The CRG of the design example is shown in Figure 7c. By observing the CRG, it



(a) CDFG

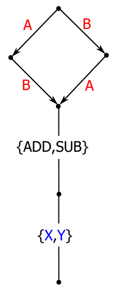


(b) Reachability graph

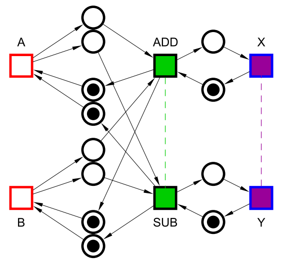


(c) Concurrent reachability graph

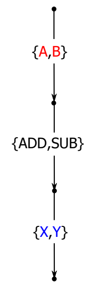
Figure 7: PN models of conceptual design



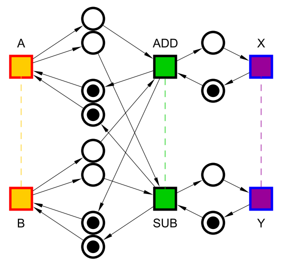
(a) Bundle 1



(b) Policy net for Bundle 1



(c) Bundle 2



(d) Policy net for Bundle 2

Figure 8: Modelling bundles by pruning reachability graphs

makes intuitive sense that the steps *ADD* and *SUB* should be bundled because 1) they are enabled together and 2) their concurrent execution can be restricted to parallel execution because they can be synchronised to complete at a worst-case delay without reducing circuit performance. The choice of bundling the inputs and outputs depends purely on where the signals arrive from and go to. For instance, if the inputs arrive from different timing domains, it is not possible to bundle the inputs. Similarly, bundling of outputs depends on the nature of block receiving them. We have considered two pruning behaviours as shown in Figure 8a and Figure 8c to study the impact of bundling on digital circuit realisation. Step persistence property was checked to avoid hazardous bundling. This verification is essential in filtering out incompatible bundling that would result in undesirable circuit switching activity. Furthermore, step persistent bundling also ensures that system functional behaviour is not changed after bundling by pruning within the limits of the CRG. The implication of the two variations of bundle sets on the PN model is shown in Figure 8b and Figure 8d. Different partitioning scenarios of the design example can be visualised now. By bundling, the transitions of a partition would only fire in a step now, *i.e.*, they would only execute together in a lock-step manner. In this paper, we have used a novel extension to PNs, called *Policy nets,* to describe partitioning and step firing behaviour. Policy nets are, basically, Petri nets with step firing policies. This net classification has been introduced lately in the WORKCRAFT framework.

IV. Digital Circuit Synthesis from PN Models

In this section, we describe our method for synthesising elastic digital circuits from PN models. The key idea here is to introduce synchronous-like rigidity into the asynchronous PN models by functional partitioning with bundles without changing functional behaviour. Granularity of these partitions can be varied according to level of elasticity and rigidity required by the designer. By starting with a pure asynchronous model of a design, we can enforce varying levels of granular rigidity which could eventually transcend to a pure synchronous specification.

A. Model Transformation to Asynchronous Pipeline Models

The first step in circuit synthesis is to transform the PN specification into asynchronous pipeline models. We intentionally did not include asynchronous handshaking in the PN models in Section III so that a designer can focus on functional elements and not worry about the aspects of circuit timing implementation. In this section, we introduce *hardware description language* (HDL) elements into the PN model such as registers, combinatorial logic, control logic and clocking information.

The first step in transformation is to introduce pipelining into the PN models. Pipelining is conducted without changing the behaviour of the system based on the principles of slack elasticity [21]. Handshaking happens between neighbouring pipeline stages and so, distinction between registers and combinatorial logic is made. The bundled-data asynchronous control elements of handshake control and matched delay are incorporated next. Handshake control signals signifying the *HC* block are indicated in the model. Finally, the forks and join in the CDFG are now mapped to control paths. In this manner,

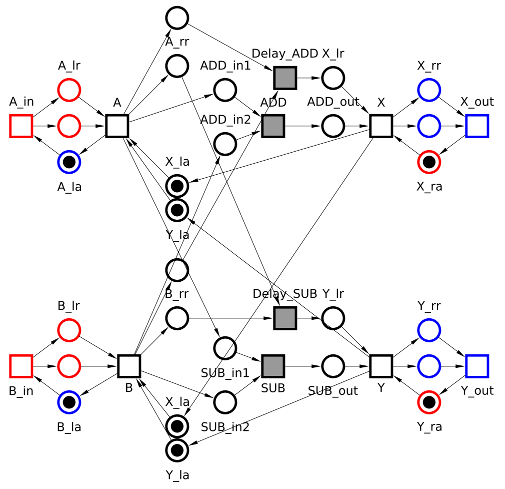


Figure 9: Transformation to bundled-data pipeline

a PN-based CDFG model of design can be transformed into a PN-based HDL-like description of a digital.

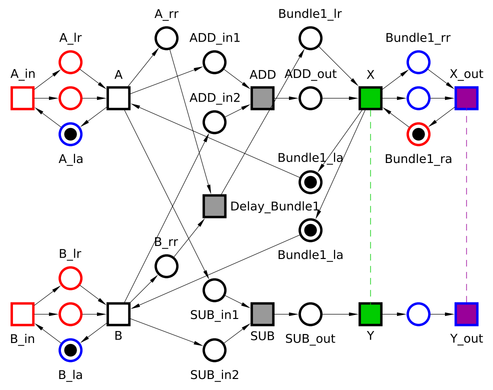
Figure 9 depicts the transformation of the conceptual design from a CDFG to its HDL specification. Pipeline registers A, B, X and Y have been introduced. *HC* control signals manage the handshake clocking of these registers. Distinction between data path and control logic can now be visualised more clearly. Here, the transformation and design verification is done manually at this moment. This would be automated in the future for design productivity.

B. Partitioning into Elastic Bundles

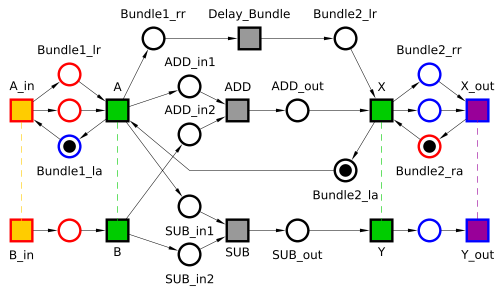
In this section, the implication of bundle transformation to digital circuits is shown. The transformations of bundle set partitions 1 and 2 are shown in Figures 10a and 10b, respectively. It can be seen that partitioning into bundles results in lower control overhead. Registers can be controlled by fewer handshake signals by bundling the requests and acknowledge signals. The transformed PN model can thus be viewed as a mixture of coarse-grained locally clocked elements and fine-grained locally clocked elements. Our outlook is that the coarse-grained locally clocked elements are synchronous to a degree of rigidity by nature of sharing a common clock signal, and the fine-grained locally clocked elements are asynchronous and elastic. The transformed circuits exhibit mixed synchronous-asynchronous nature but still remain asynchronous. We introduce such sets of re-partitioned bundles exhibiting granular rigidity in bundled-data pipelines as ***Elastic Bundles*** (EB).

C. From PN Models to Digital Circuits

In this section, we discuss the method employed to synthesise the PN-based HDL description into digital circuits using standard clocked CAD tools. The RT-based synchronous-asynchronous EDA tool flow, summarised in Section II-B, is used for synthesising the elastic-bundle circuit.



(a) Elastic-bundle 1



(b) Elastic-bundle 2

Figure 10: Elastic-bundle pipeline transformation

The first step is to introduce the asynchronous control elements, specifically the *HC* blocks and fork/join elements. Pre-built *HC* blocks borrowed from the RT tool flow is utilised. Fork/join elements are required for implementing the control of non-linear pipelines. These elements are introduced with the rule that every fork in the data path is associated with a join, and every join in a data path be associated with a fork [12]. The join elements employ Muller's C-elements [22] for synchronising request signals as well as synchronising acknowledgement signals. Next, the PN-based HDL model is direct-mapped to a behavioural HDL language such as Verilog. Active high latches are used for register implementation. In the case of our conceptual design, Figure 9 was thus manually mapped into the following behavioural Verilog description:

```
module Fn (A_in, A_lr, A_la, B_in, B_lr, B_la,
   X_out, X_rr, X_ra, Y_out, Y_rr, Y_ra, rst);
   input [31:0] A_in, B_in;
   output [31:0] X_out, Y_out;
   input A_lr, B_lr, X_ra, Y_ra, rst;
   output A_la, B_la, X_rr, Y_rr;
   wire [31:0] A, B, X, Y, ADD1_in1, ADD1_in2, SUB1_in1, SUB1_in2;

   // Datapath Logic
   reg32_async R1 (.D(A_in), .Q(A), .ck(A_ck), .rst(rst));
   reg32_async R2 (.D(B_in), .Q(B), .ck(B_ck), .rst(rst));
   assign ADD1_in1 = A;    assign ADD1_in2 = B;
   assign SUB1_in1 = A;    assign SUB1_in2 = B;
   assign X = ADD1_in1 + ADD1_in2;
   assign Y = SUB1_in1 - SUB1_in2;
   reg32_async R3 (.D(X), .Q(X_out), .ck(X_ck), .rst(rst));
   reg32_async R4 (.D(Y), .Q(Y_out), .ck(Y_ck), .rst(rst));

   // Control Logic
   handshake_ctl HC1 (.lr(A_lr), .la(A_la), .rr(lr1), .ra(la1), .ck(A_ck), .rst(~rst));
   handshake_ctl HC2 (.lr(B_lr), .la(B_la), .rr(lr2), .ra(la2), .ck(B_ck), .rst(~rst));
   // Control Stage 1 (Fork - Join)
   assign r11 = lr1;    assign r12 = lr1;
   assign r21 = lr2;    assign r22 = lr2;
```

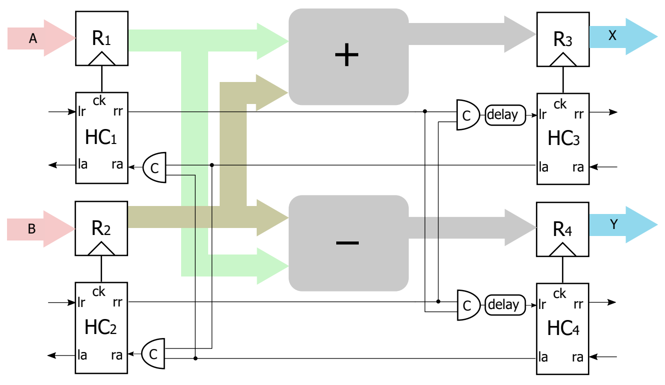


Figure 11: Bundled-data circuit of conceptual design

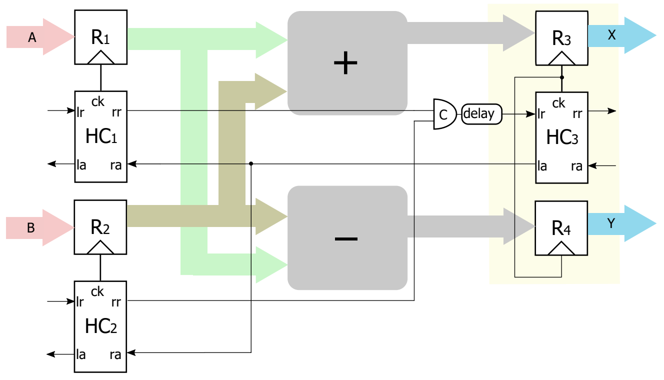
```
    Celement j1 (.in1(a11), .in2(a21), .out(la1));
    Celement j2 (.in1(a12), .in2(a22), .out(la2));
    // Control Stage 2 (Join - Fork)
    Celement j1 (.in1(r11), .in2(r21), .out(lr3_pre));
    Celement j2 (.in1(r12), .in2(r22), .out(lr4_pre));
    assign a11 = la3;    assign a12 = la3;
    assign a21 = la4;    assign a22 = la4;
    // Delay elements for 32 bit adder pipeline
    DelayElement d1 (.in(lr3_pre), .out(lr3));
    DelayElement d2 (.in(lr4_pre), .out(lr4));

    handshake_ctl HC3 (.lr(lr3), .la(la3), .rr(X_rr), .ra(X_ra), .ck(X_ck), .rst(~rst));
    handshake_ctl HC4 (.lr(lr4), .la(la4), .rr(Y_rr), .ra(Y_ra), .ck(Y_ck), .rst(~rst));
endmodule
```

This design can now be synthesised using the RT-based EDA tool flow which would result in the circuit shown in Figure 11.

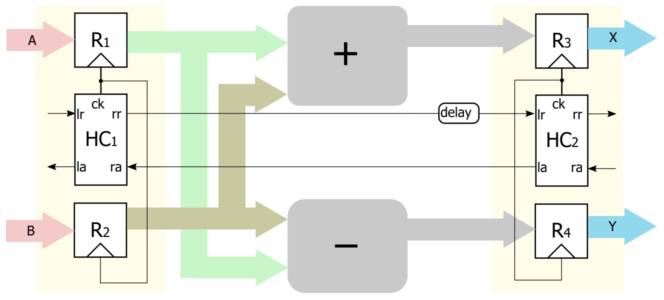
Similarly, Elastic Bundle sets 1 and 2 modelled in Section IV-B would result in circuits depicted in Figures 12a and 12b respectively, after synthesis. Figure 12a is partitioned into three asynchronous domains with HC_3 block introducing a level of rigidity on registers R_3 and R_4. The partitioning in Figure 12b results in two asynchronous domains both exhibiting the same degree of granular rigidity. This particular implementation behaves in the manner of a linear pipeline.

V. 16-POINT FFT CASE STUDY

In this section, we consider an asynchronous 16-point FFT architecture as a case study to implement and test the methodology presented in this paper. Detailed description of the architecture and implementation are outside the scope of this paper. The FFT architecture is based on the design presented by the authors in [23], [13]. In this architecture, the FFT algorithm is described in a multirate format which is highly concurrent and heterogeneous by nature. This very nature of the architecture proved to be an ideal case study for us because it allowed for modelling of the algorithm in its native asynchronous/elastic form. Figure 13a provides a snapshot of the 16-point FFT architecture described in PNs. Four-way wagging structure captures the behaviour of high frequency input stream being decimated to lower frequency data streams and then expanded back to high frequency output stream. Distributed pipelining manages parallel operation of data streams at different frequencies. Furthermore, the 16-point FFT architecture is hierarchically decomposed of eight



(a) Bundle set 1



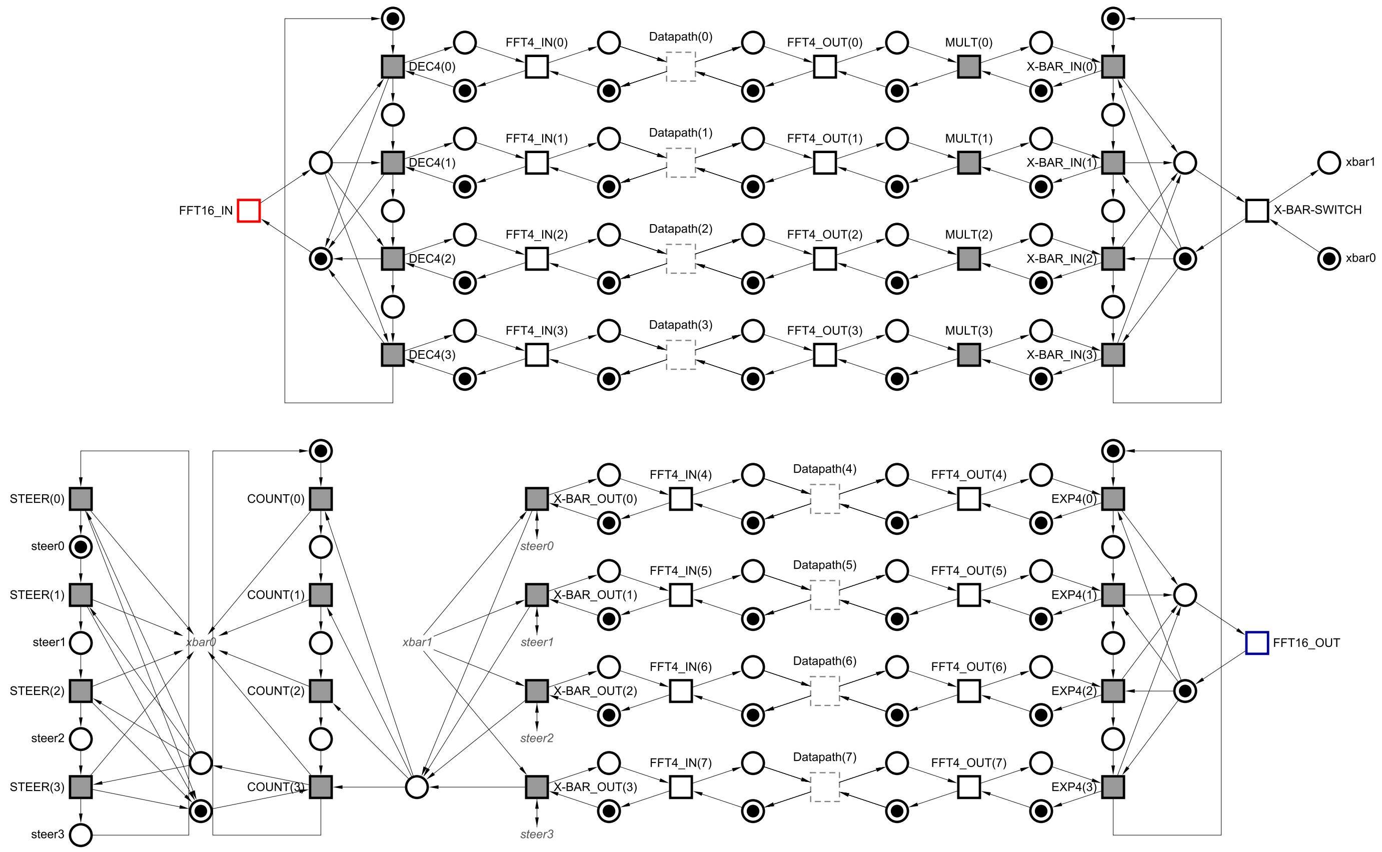
(b) Bundle set 2

Figure 12: Elastic-bundle circuits

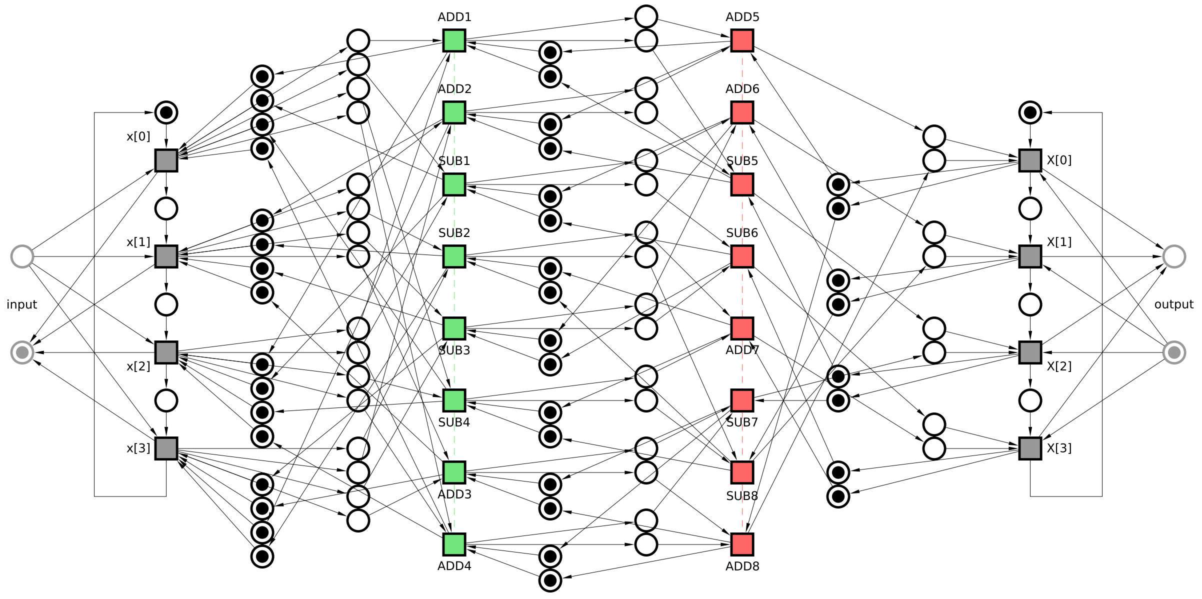
identical 4-point FFT blocks, denoted as dotted boxes in the top-level PN model. The PN model for the 4-point FFT datapath is shown in Figure 13b.

The case study was subjected to the proposed design flow, starting from PN description to EB functional partitioning, and HDL direct mapping to circuit synthesis using the RT design flow. For comparative analysis, five implementations of the 16-point FFT architecture were conducted: one bundled-data asynchronous and four EB implementations. The circuits were described in Verilog and synthesised using Design Compiler in the 90nm Faraday library. The circuits were tested using pre-defined input stream of 1024 random numbers. Circuit simulations were conducted using VCS simulator. The output data stream was verified for each partitioning scheme by comparing with MATLAB 16-point FFT computation. The SAIF (Switching Activity Interchange Format) file from the VCS simulations was used to calculate the power for each design by PrimeTime-PX. The EB implementations are compared against the bundled-data 16-point implementation. Table I summarises the pre-layout synthesis results of the case study.

The *bundled-data* implementation represents a fully elastic or asynchronous implementation of the 16-point FFT architecture. *EB Temporal* is the first elastic-bundle implementation of the architecture. Here, the adders and subtractors within the 4-point FFT datapath blocks are bundled according to their natural order of arrival of data tokens. In this case, the circuit area overhead of asynchronous control got reduced by nearly 60% compared to the bundled-data implementation, with a 34% improvement in control power consumption. *Opt EB Temporal* implementation optimised the previous implementation by restricting concurrency further whilst maintaining the natural



(a) Top-level PN model



(b) PN model for Datapath block

Figure 13: PN model for 16-point FFT

Table I: Synthesis results for several 16-point FFT designs

Design	Total Area (gates)	Control Area (gates)	Control Logic Power (mW)	Energy/Point (pJ)	Energy Benefit	Control Area Benefit
EB Temporal	58,533	3,015	1.34	17.107	1.069	2.45
Opt EB Temporal	58,185	2,667	1.28	17.089	1.071	2.77
EB Maximal	57,921	2,403	1.24	17.105	1.069	3.08
EB Reuse	58,753	3,235	1.38	17.194	1.064	2.29
Bundled Data	62,993	7,399	2.03	18.294	1.000	1.00

order of data token arrival. *EB Maximal* implementation restricted concurrency further with less regard to natural order of data arrival and more focus on reducing control area overhead. This case is described in Figure 13 where a maximal case of bundling adders and subtractors with reduced concurrency can be visualised. Finally, the *Reuse* implementations focussed on reducing datapath computation logic by exploiting natural order of data arrival tokens to reuse adders and subtractors. The savings in area, as evident from the results, were due to reduction of 50% of adders and subtractor logic. The range of circuit area overheads demonstrated in this case study is clearly in line with the illustration of low-overhead elastic circuits presented earlier in Figure 1 (shadowed). Amongst the implementations, *EB Maximal* proves the most optimum demonstrating lowest control logic power consumption and an improvement of $3.08\times$ in terms of control area over its bundled-data counterpart. All of the EB implementations demonstrate ~7% improvement in energy per data point. Considering that all FFT implementations share the same datapath, the energy improvement was due to reduction in power of switching activity in the EB control logic.

VI. Conclusion

In this paper, we proposed a novel method for synthesising asynchronous circuits with varying levels of rigidity. The hypothesis was that *bundles* would reduce the area overheads of asynchronous design by relaxing granularity of handshake control. A PN-based dataflow modelling technique was developed to model digital systems on a higher level of abstraction than RTL. Functional partitioning was then introduced in these dataflow models by identifying sets of *bundles* that could restrict elasticity whilst retaining functional behaviour. Taking the case of asynchronous bundled-data circuits, these sets of *bundles* were extended to a novel notion of *Elastic Bundles* which basically re-partitioned the design into coarse-grained locally clocked elements and fine-grained locally clocked elements. This net transformation enabled synthesis of the elastic-bundle circuits under standard EDA tool flow. The method was tested on a 16-point FFT algorithm. The elastic-bundle FFT designs reduced control area overhead by a margin $> 2\times$ whilst demonstrating ~30% reduction in control power consumption when compared against the bundled-data design.

Acknowledgements

The authors are grateful to the anonymous reviewers for their critiques and valuable suggestions on how to further develop this work. This research was supported by EPSRC grant EP/I038551/1 Globally Asynchronous Elastic Logic Synthesis (GAELS).

References

[1] A. Kondratyev and K. Lwin, "Design of asynchronous circuits by synchronous CAD tools," in *Proc. Design Automation Conference (DAC)*, 2002, pp. 411–414.

[2] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1904–1921, 2006.

[3] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1437–1455, 2009.

[4] A. Peeters and K. Van Berkel, "Synchronous handshake circuits," in *Int. Symp. on Asynchronus Circuits and Systems (ASYNC)*, 2001, pp. 86–95.

[5] J. Sparso and S. Furber, *Principles of asynchronous circuit design: a systems perspective*. Kluwer Academic Publishers, 2001.

[6] I. Blunno and L. Lavagno, "Automated synthesis of micro-pipelines from behavioral verilog hdl," in *Int. Symp. on Asynchronus Circuits and Systems (ASYNC)*, 2000, pp. 84–92.

[7] F. Te Beest, A. Peeters, K. Van Berkel, and H. Kerkhoff, "Synchronous full-scan for asynchronous handshake circuits," *Journal of Electronic Testing*, vol. 19, no. 4, pp. 397–406, 2003.

[8] H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers, "Synchronous interlocked pipelines," in *Int. Symp. on Asynchronus Circuits and Systems (ASYNC)*, 2002, pp. 3–12.

[9] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," in *The Best of ICCAD*, 2003, pp. 143–158.

[10] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. Design Automation Conference (DAC)*, 2006, pp. 657–662.

[11] S. M. Nowick and M. Singh, "High-performance asynchronous pipelines: an overview," *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 8–22, 2011.

[12] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of asynchronous templates for integration into clocked CAD flows," in *Proc. Asynchronous Circuits and Systems (ASYNC)*, 2009, pp. 151–161.

[13] W. Lee, V. Vij *et al.*, "Design of Low Energy, High Performance Synchronous and Asynchronous 64-Point FFT," *Proc. Design, Automation & Test in Europe (DATE)*, pp. 242–247, 2013.

[14] Y. Xu and K. S. Stevens, "Automatic synthesis of computation interference constraints for relative timing verification," in *IEEE Int. Conf. on Computer Design*, 2009, pp. 16–22.

[15] "Workcraft homepage, URL: http://www.workcraft.org."

[16] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[17] C. Brej, "Wagging logic: Implicit parallelism extraction using asynchronous methodologies," in *Int. Conf. on Application of Concurrency to System Design (ACSD)*, 2010, pp. 35–44.

[18] D. Sokolov and A. Yakovlev, "GALS Partitioning by Behavioural Decoupling Expressed in Petri Nets," *Proc. Asynchronous Circuits and Systems (ASYNC)*, pp. 17–26, 2014.

[19] D. Culler, "Dataflow architectures," *Annual Review of Computer Science*, vol. 1, no. 1, pp. 225–253, 1986.

[20] J. Fernandes, M. Koutny, L. Mikulski, M. Pietkiewicz-Koutny, D. Sokolov, and A. Yakovlev, "Persistent and nonviolent steps and the design of gals systems," *Fundamenta Informaticae*, vol. 137, pp. 143–170, 2015.

[21] R. Manohar and A. J. Martin, "Slack elasticity in concurrent computing," in *Mathematics of Program Construction*, 1998, pp. 272–285.

[22] D. Muller and W. Bartky, "A theory of asynchronous circuits," *Proc. Int. Symp. on the Theory of Switching*, pp. 204–243, 1959.

[23] D. J. Barnhart, "An improved asynchronous implementation of a fast fourier transform architecture for space applications," Air Force Institute of Technology, United States Air Force, Tech. Rep., 1999.