# Workcraft: Ten Years Later

Danil Sokolov, Victor Khomenko, Andrey Mokhov

Newcastle University, Newcastle upon Tyne, UK

**Abstract** A large number of models that are employed in the field of concurrent systems' design, such as Petri nets, Signal Transition Graphs, gate-level circuits, dataflow structures have an underlying static graph structure. Their semantics, however, is defined using additional entities, e.g. tokens or node/arc states, which collectively form the overall state of the system. We jointly refer to such formalisms as *Interpreted Graph Models* (IGMs).
Workcraft is a framework for capturing, simulation, synthesis and analysis of IGMs. It provides an extendible cross-platform plugin-based front-end to a variety of computationally-intensive command-line back-end tools. This paper gives the developers' perspective on Workcraft and overviews its evolution.

## 1 Introduction

We want our research to be used to make the world a better place. However, technology transfer is challenging in practice due to a number of obstacles. One of the primary research outcomes is scientific publications. However, engineers do not have time to read them and do not normally have a necessary background to comprehend and apply that knowledge, or even find the necessary publications.

One of the ways to make this knowledge more accessible is to encapsulate it in software tools. Ideally, these tools should be usable by non-experts. In practice, however, the situation is very different. A research tool is typically developed up to a point when it can produce a table of results for a research paper. The motivation to develop it any further is diminished afterwards – in the current "publish or perish" academic culture it is usually more advantageous to start exploring new topics for writing another paper, and it is also more interesting than polishing old tools. Moreover, research funding is usually granted for a very limited period of time and so the academics, researchers, and PhD students who developed the tool leave or get allocated to different projects. Hence, much of research software has only a command-line interface with cryptic options, poor documentation, limited error handling, requires modifying the source code to adjust the tool to a particular variant of a problem and is not maintained.

Integrating several research tools into a coherent flow presents further challenges. File formats are often invented by the creators of the tools and are non-standard. Furthermore, there are often gaps in the flow that have to be patched to complete it. This requires a large amount of slog with no perspective of publishing its results. Therefore, using research software in a real industrial flow is

often infeasible. The net result of the above is that much knowledge remains buried in publications or "experts-only" tools and is not accessible to practitioners.

Workcraft is unusual in several respects. There have been a series of funded projects related by the common topic of application of Petri nets to circuit design. This allowed to have relatively stable group of developers and sufficient time to make the tool usable. Furthermore, early versions of the tool attracted industrial interest, which motivated putting more effort into user interface development. In addition, several previously developed command-line tools were deployed within Workcraft as back-ends. In turn, the success of Workcraft and the perspectives of industrial exploitation motivated the developers of back-ends to maintain and enhance the functionality of these tools.

As the result, Workcraft opens access to the goodness hidden in research tools. The main enabling factors for this to happen are:

- *Availability* – open-source front-end and plugins, permissive freeware licenses for back-end tools, as well as frequent releases with bug fixes and features requested by users.
- *Usability* – elaborated GUI that was developed with much feedback from the users.
- *Portability* – it runs on Windows, Linux, and Mac OS X operating systems.
- *Extendibility* – the framework is designed to easily include new IGMs and interfaces to back-end tools as plugins.
- *Automation* – several complete design flows have been implemented by bridging the gaps between back-ends and converting file formats.

The focus on availability, usability, portability and extendibility, along with extensive networking, has proven effective – there is a large and diverse user base. For example, in 2015 there were 4.4k downloads from 1.2k unique IPs and 11.1k visits to `http://workcraft.org/`, 4.7k of them from unique IPs. During that year there were 4 releases with 49 bug fixes and 23 new features. We do not know all Workcraft users, but one can identify at least the following categories: developers, industrial users, undergraduate students, academics, researchers, and PhD students.

In this paper we give the developers' perspective on Workcraft. The main principles of Workcraft architecture and its design flow are outlined in Sections 2 and 3. Supported IGMs together with relevant case studies use are presented in Sections 4 and 5. We analyse the categories of Workcraft users in Section 6 and overview the timeline of Workcraft evolution in Section 7.

## 2 Workcraft philosophy

In this section we discuss several basic principles/ideas underlying Workcraft. Some of them are visible to the user and aimed at enhancing the user experience. Others are concerned with the internal organisation and aimed at simplifying the integration of new research tools.

### 2.1 Interpreted Graph Models

A large number of models that are employed in the field of concurrent systems' design, such as Petri nets, Signal Transition Graphs, gate-level circuits, dataflow structures have an underlying static graph structure. Their semantics, however, is defined using additional entities, e.g. tokens or node/arc states, which collectively form the overall state of the system. We jointly refer to such formalisms as *Interpreted Graph Models* (IGMs) [21].

The similarities between the interpreted graph models allow for links between different formalisms to be created, either by means of adapter interfaces or by conversion from one model type into another. This greatly extends the range of applicable modelling and analysis techniques.

Workcraft is designed to provide a flexible common framework for development of interpreted graph models, including visual editing, (co-)simulation and analysis. The latter can be carried out either directly or by mapping a model into a behaviourally equivalent model of a different type (usually a Petri net or Signal Transition Graph). Hence the user can design a system using the most appropriate formalism (or even different formalisms for the subsystems), while still utilising the power of Petri net analysis techniques. In Section 4 there is a summary of the currently supported IGMs.

### 2.2 Front-end vs. back-end

Workcraft provides front-end to a number of command-line back-end tools, such as Petrify [10,2] and UnfoldingTools toolkit [3]. The calls to back-end tools are transparent to the user: Workcraft automatically chooses the correct command-line parameters, parses the output of the tools and presents it to the user in an appropriate graphic form. For example, to check whether a digital circuit conforms to its environment the user needs to click a single menu item. In response the following sequence of actions is performed by the front-end:

1. The circuit is converted to an equivalent STG.
2. The internal signal transitions in the environment STG (it models the contract between the circuit and its environment) are replaced by dummies – this is required for technical reason.
3. The STGs obtained in the previous two steps are composed by calling PComp back-end with appropriate command line parameters.
4. The front-end expresses the conformation property as an expression in Reach language. Parts of this expression are specific to the circuit under test and need to be calculated by the front-end.
5. The composed STG is unfolded by calling Punf back-end.
6. The resulting unfolding prefix and Reach expression are passed to MPSat back-end that performs verification.
7. The verification results are parsed by the front-end. If the property holds then an appropriate message is displayed. Otherwise the violation trace of the composed STG reported by MPSat is projected to the circuit, and the

user can execute it step-by-step to debug the problem. All the capabilities of the front-end simulator are available, e.g. navigation within the trace, branching, etc.

Each of the above steps looks trivial and "boring" (and hence unpublishable) from the research point of view. However, checking conformation is a frequent task during the circuit design. Performing all the above steps manually would have been very tedious and error-prone, discouraging the casual user from applying formal verification. Hence using WORKCRAFT makes it feasible for the user to harness the power of research tools, which helps to catch the bugs early in the design process, and reduces the risk of an incorrect circuit going into production.

### 2.3 Plugin-based architecture

Extendibility is an important part of WORKCRAFT philosophy and this is reflected in its plugin-based architecture [23]. In particular, there is a framework for adding new IGMs and integrating new back-end tools. This framework provides a number of standard services available to the plugins, such as (de-)serialisation of IGMs, common editing features (creation of nodes and connections, undo-redo, copy-paste, etc.) and model visualisation.

To add a new IGM the developer has to implement a small set of Java interfaces for mathematical and visual representation of the IGM. Most of the functionality has default implementations provided by the WORKCRAFT core and only "unusual" features of a new model need to be explicitly implemented.

For integration of a back-end tool the developer needs to provide versions of the tool for the supported operating systems (this is usually not a problem because console applications are relatively easy to port), and implement a simple Java interface that specifies how to run the tool, interpret its output, which IGMs it is applicable to, and which menu to integrate it into.

There is also a possibility to add more complicated plugins that interact with visual representation of the IGMs, e.g. the simulation plugin.

## 3 WORKCRAFT design flow

The WORKCRAFT design flow is modelled by the Petri net in Figure 1. A typical way of designing a circuit is as follows.

1. The STG specification (place specification) is created in the WORKCRAFT editor (transition edit) or perhaps imported from a *.g file (transition import).
2. The user verifies various properties of this specification, such as consistency, deadlock freeness, output persistency, input properness, complete state coding (CSC) and some custom design specific properties (transition verify).
3. The verification report from a back-end tool (place report) is then presented to the user in a convenient form, e.g. a violation trace can be simulated (transition simulate), CSC conflict cores can be visualised as a core map or a core density map (transition visualise). This helps the user to debug the STG.

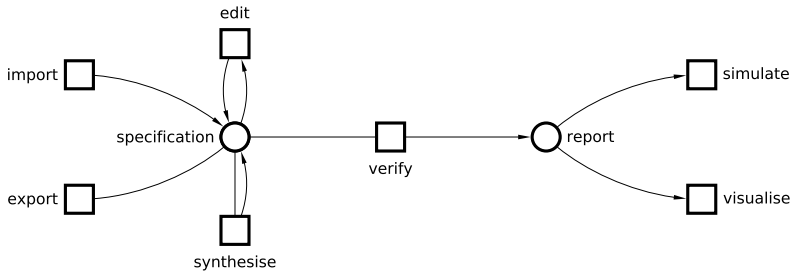



Figure 1: WORKCRAFT design flow.

4. Once a correct specification is obtained, it can be implemented as a circuit. At this point the CSC property may still be violated, and so a new STG where the conflicts are resolved by inserting new internal signals can be automatically created by synthesis back-ends (transition synthesise; hence place specification will contain two tokens representing the original and modified STGs).
5. At this point the user can synthesise a digital circuit (transition synthesise; hence place specification will contain three tokens representing the two STGs and the circuit).
6. The user can manually alter the circuit, e.g. by improving the layout or changing the polarity of some internal signals (transition edit).
7. The circuit must be verified against the initial specification, as synthesis tools are complicated and may have bugs and manual editing is error-prone (transition verify). Verification report is presented to the user in a convenient form.
8. The created models can be exported, e.g. as Verilog netlist for circuits and *.g files for STGs (transition export). In addition models can be exported in a variety of graphic formats for inserting into documentation or research papers.

## 4 WORKCRAFT models

As explained in Section 2, WORKCRAFT supports several IGMs, and new models are added from time to time. Some of the most popular IGMs are described below. A crucial aspect of WORKCRAFT is interaction and synergy between different types of IGMs.

A popular formalism for capturing the behaviour of a concurrent system is *Finite State Machines (FSMs)*. The advantage of FSMs is their relative simplicity compared to the alternative formalisms. However, they represent concurrency by multi-dimensional interleaving 'diamonds' which is unnatural and leads to exponential blow-up in the size of the model [27].

*Petri nets (PNs)* [19] are a well-known 'true concurrency' formalism which is much more convenient for practical modelling. WORKCRAFT supports conversions between FSMs and PNs: one can construct the reachability graph of a

PN or, vice versa, synthesise a PN as a compact representation of a behaviour expressed as an FSM – see Vending Machine case study in Section 5.1 for FSM and PN.

*Signal Transition Graphs (STGs)* [9,24] are a kind of PNs where transitions are labelled by rising and falling edges of signals. STGs are often used to specify the behaviour of speed-independent *Digital Circuits* [18,12], which is another IGM supported by WORKCRAFT. Many kinds of interactions and conversions between these two models are supported. For example, one can synthesise an STG as a circuit using several implementation styles or convert a circuit to an STG – this is necessary for composing it with the environment expressed as an STG for subsequent verification of various standard and custom correctness properties [22]. Section 5.2 presents a case study on designing a speed-independent circuit using STGs.

*Dataflow Structures (DFSs)* [25] and *xMAS Circuits* [8] are high-level models for designing pipelines, in particular data paths of circuits. WORKCRAFT supports the simulation and analysis of these IGMs by converting them to STGs and utilising the established functionality. Some model-specific functionality such as finding bottlenecks, cycle analysis, and performance optimisation using *wagging* [7] are also supported. Section 5.3 showcases DFS functionality using a baseband transmitter pipeline.

*Conditional Partial Order Graphs (CPOGs)* [15] is a formalism for specifying a collection of behavioural scenarios, and combining them into a compact graph representation using the optimal encoding. For example, CPOGs can be used for synthesis of application-specific microprocessor instruction sets, see Section 5.4 for an ARM Cortex-M0 case study.

*Structured Occurrence Nets (SONs)* [14] is a model for capturing and analysis of causality and concurrency in families of execution traces. They can be used to represent the current state of crime or accident investigation – see Section 5.5 for the use of SONs to model Ladbroke Grove rail crash.

The diagram in Figure 2 shows the relationships between the currently supported IGMs. There are several categories of automatic conversions. *Synthesis*, e.g. from STGs to Digital Circuits or from FSMs to PNs, is a computationally intensive procedure whose resulting graph is structurally very different from the input graph. *Translation* is a relatively simple transformation yielding a structurally similar graph. *Lossless* translation, e.g. from PNs to STGs, does not lose information, i.e. the original model can be restored from the result of the conversion. *Lossy* translation, e.g. from STGs to PNs, loses some information, in this case the signal information attached to transitions.

## 5 Case studies

The applications of WORKCRAFT are wide-ranging: from modelling concurrent algorithms and biological systems to designing asynchronous circuits and investigating crimes. In this section we present several examples of how WORKCRAFT can be used.

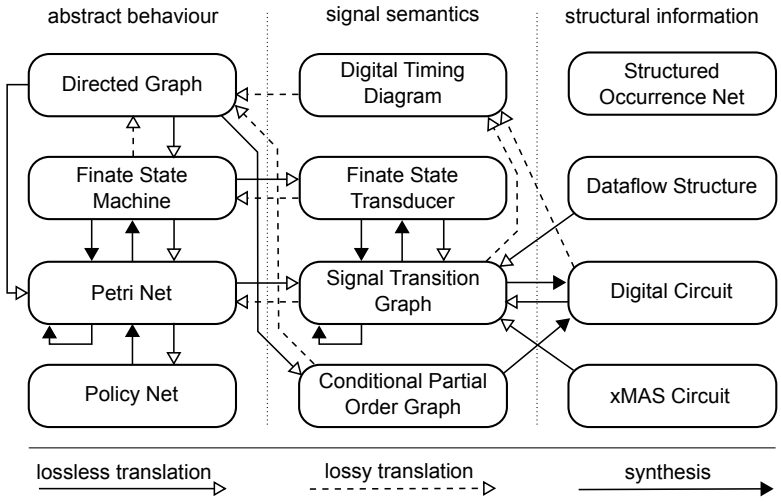
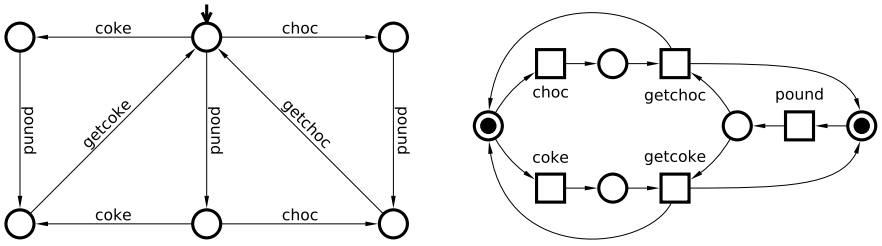



Figure 2: Relationships between WORKCRAFT models.

### 5.1 Modelling Concurrent Systems: Vending Machine

In this case study WORKCRAFT is used to capture the behaviour of a concurrent vending machine as an FSM shown in Figure 3a. It allows the user to insert a £1 coin (action pound) concurrently with making an order (actions coke and choc). Note that the concurrency between actions pound and coke as well as pound and choc is represented by *interleaving* – there are two corresponding diamonds in this FSM, and the layout is chosen so as to highlight them.




(a) FSM specification. (b) Synthesised Petri net.

Figure 3: Concurrent vending machine.

A PN can often be automatically obtained from the initial FSM model by the process called synthesis (`Conversion→Net synthesis [Petrify]` menu item of FSM model). The resultant PN is shown in Figure 3b; one can validate that be reachability graph of this PN coincides with the original FSM. Note that

transitions pound and coke as well as pound and choc are now truly concurrent in the PN.

### 5.2 Design of Asynchronous Circuits: VME Bus Controller

In this case study WORKCRAFT is used to formally specify and derive a speed-independent implementation of VME bus controller. A controller for VME bus provides an interface between a data bus and a slave device, as shown in Figure 4.

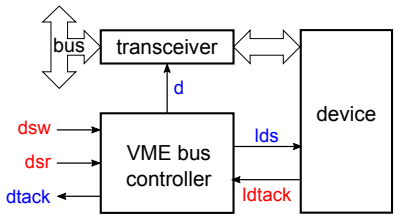



Figure 4: VME bus interface.

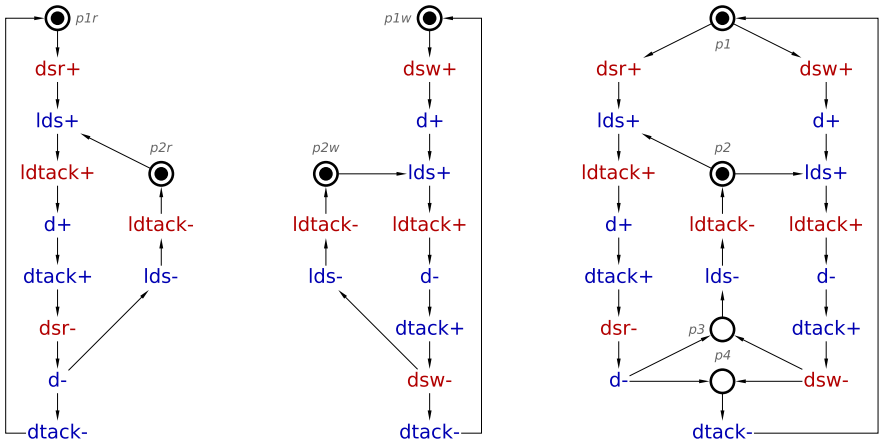
The controller has two modes of operation: reading from the device into the bus (activated by dsr+) and writing from the bus into the device (activated by dsw+). In the reading mode, a request to read data form the device is made through lds+. When the device has the data ready and this is acknowledged by ldtack+, the controller opens the transceiver by d+ and notifies the bus that data is ready for transfer by dtack+. After the read operation is complete, all the signals return to the initial state.

In the writing mode, once the data is stable on the bus, the transceiver is opened by d+, and the write request is made by lds+. When the device acknowledges the receipt of data by ldtack+, the transceiver is closed with d-, thus isolating the device from the bus, and the bus is notified that the write operation is complete by dtack+. After that all the signals return to the initial state.

The read and write modes of VME control are captured in WORKCRAFT by the STGs in Figures 5a and 5b respectively. These two STGs describe the behaviour of the same circuit and need to be combined into one specification by merging their initial states. Note that transitions ldtack-, lds- and dtack- occur in both branches of the choice and can also be merged. For merging places and transitions one can use the corresponding operations in the `Transformations` menu. The complete STG specification of VME bus controller is shown in Figure 5c.

Before proceeding to synthesis the STG needs to satisfy the following soundness properties (`Verification` menu of WORKCRAFT enables checking all these properties with a single click):

– *Deadlock freeness* – every reachable marking enables at least one transition.




(a) Read operation. (b) Write operation. (c) Combined operation.

Figure 5: STG specification of VME bus controller.

- *Consistency* – the '+' and '–' transitions of every signal alternate in every execution, always starting with the same sign.
- *Input properness* – an input cannot be disabled by an output or internal signal, and cannot be triggered by an internal signal.
- *Output persistency* – an enabled output or internal signal cannot be disabled by any other signal.

The STG specification can now be synthesised into an asynchronous circuit. A complex-gate solution is as follows (`csc0` signal was automatically inserted by PETRIFY back-end to resolve a CSC conflict):

```
INORDER = dsr dsw ldtack d dtack lds csc0;
OUTORDER = [d] [dtack] [lds] [csc0];
[d] = dsr ldtack csc0' + dsw (csc0 + ldtack');
[dtack] = d' csc0' (dsr' + dsw) + dsw' d;
[lds] = csc0';
[csc0] = dsr' d' (csc0 + dsw') + ldtack csc0;
```

This solution uses complex gates that do not usually exist in real gate libraries. Such gates need to be decomposed to map them to existing library gates – this is done by logic decomposition that preserves speed-independence of the circuit. The library of available gates can be passed to WORKCRAFT in SIS GENLIB format.

The result of technology mapping into TSMC gate library (with an addition of C-elements) is shown in Figure 6. One can verify (via `Verification` menu) that the circuit implementation is deadlock-free, hazard-free, and conforms to the original STG specification.

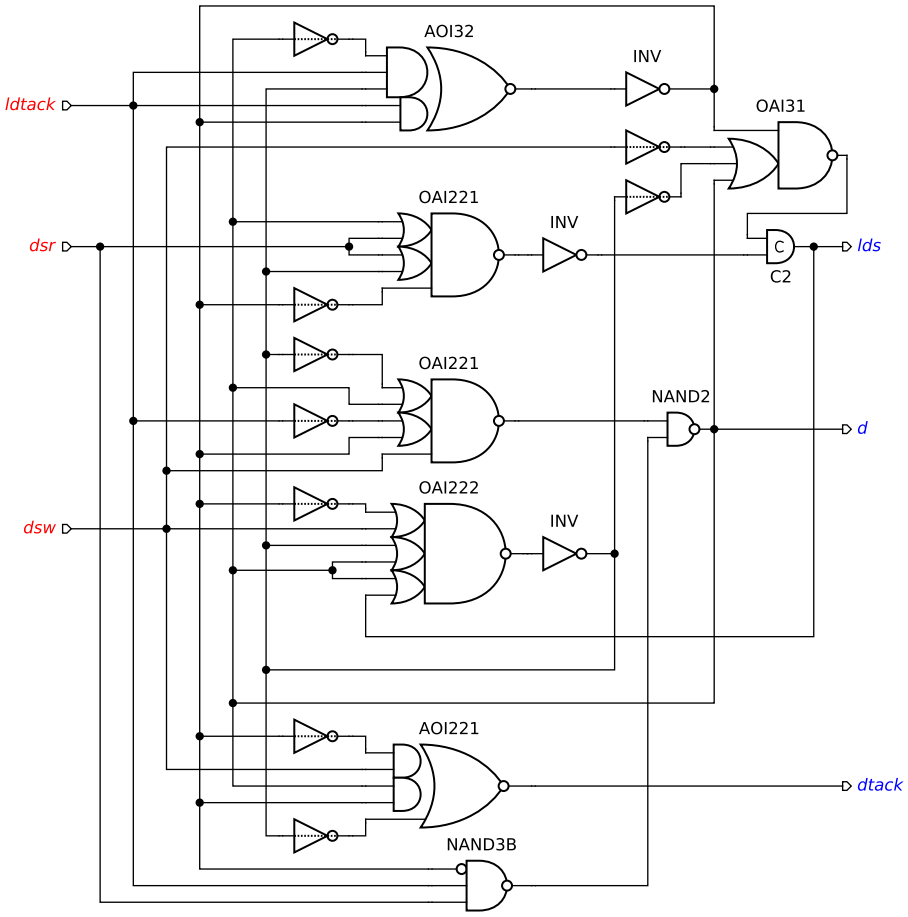



Figure 6: Implementation of VME bus controller. The dotted lines through the inverters express the timing assumption that their delays are smaller than any other gate delay in the circuit.

### 5.3 Analysis of Asynchronous Pipelines: Baseband Transmitter

Pipelines can be modelled in WORKCRAFT using the Dataflow Structure (DFS) formalism. This abstraction separates the structure and the function of the system from the implementation details of its components.

The possibility of formally modelling and reasoning about the system at this architectural level is crucial, as the design decisions made at this level will affect all the subsequent stages of the design. Moreover, optimisations performed at this level are likely to have a much stronger impact than micro-optimisations applied towards the end of the design process.

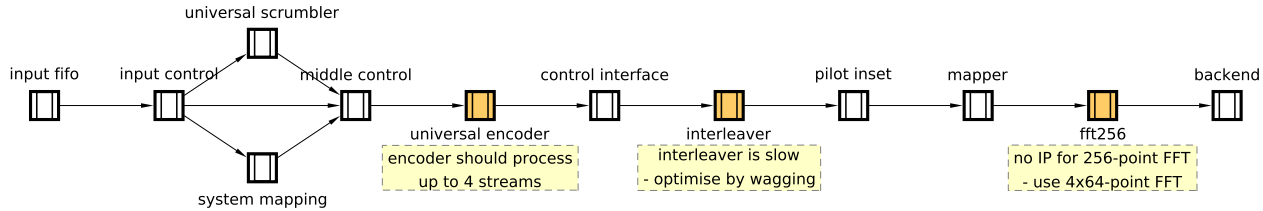
A DFS model in Figure 7a represents pipeline stages of a baseband transmitter at a rather high level of abstraction. Even at this level important design decisions can be made about specific implementation of the pipeline components. For example, information about the maximum number of streams to encode (up to 4 streams), available components for Fast Fourier Transform (maximum 64-points) and the preliminary performance estimates (interleaver is the bottleneck as it is 3 times slower than the other pipeline stages) define the refinement of the DFS model shown in Figure 7b. Note that the refinement of the interleaver stage into three concurrent slices was obtained automatically using the 3-way *wagging* [7] operation in the `Transformations` menu.

### 5.4 Instruction Set Architecture: ARM Cortex-M0+

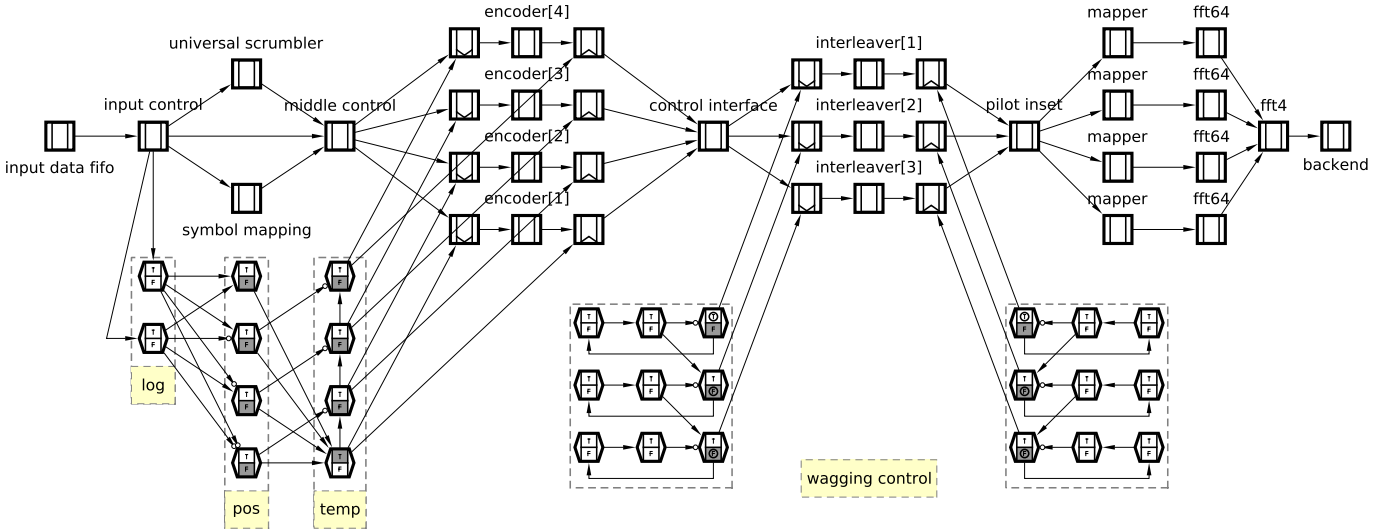
WORKCRAFT can be used to specify and explore processor Instruction Set Architectures (ISAs), and synthesise efficient hardware implementations for their microcontrollers. In this section we use WORKCRAFT to specify a subset of ARM Cortex-M0+ instructions, automatically derive optimal opcodes for them, and synthesise a Verilog netlist for the corresponding microcontroller. The presented approach relies on CPOGs as the modelling formalism [15] that provides the designer with a convenient ISA visualisation notation as well as analysis methods.

Figure 8 shows 11 partial orders corresponding to instruction classes of ARM Cortex-M0+ processor [1][11]. The events in these partial orders correspond to primitive computation steps performed during instruction execution:

- PCIU stands for the Program Counter Increment Unit. It is used to advance the program counter when fetching instruction opcodes and operands from the program memory.
- The Instruction Fetch Unit (IFU) loads instruction opcodes and immediate instruction operands from the program memory into the processor instruction register.
- The data memory can be accessed using the Memory Access Unit (MAU), which transfers data between the processor registers and the data memory.
- The Arithmetic Logic Unit (ALU) is capable of performing basic computations such as addition, multiplication, comparison, bitwise Boolean logic operations, etc.




(a) Abstract DFS model.




(b) Refined DFS model.

Figure 7: Self-timed baseband transmitter.

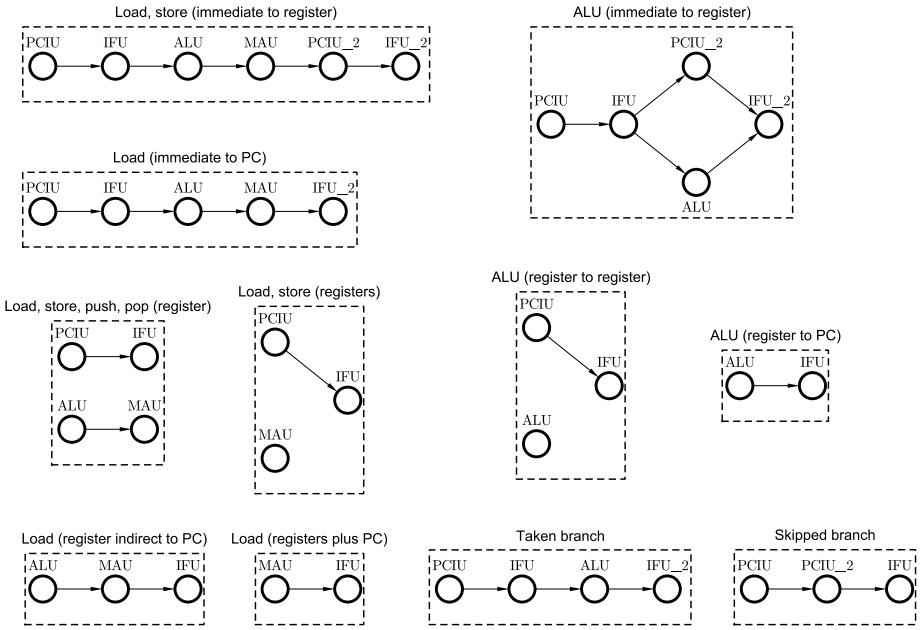



Figure 8: ARM Cortex M0+ instruction classes.

The partial orders can be created in WORKCRAFT either by individually placing and connecting events, or by using the Parameterised Graphs Algebra [17] plugin, which allows one to specify partial orders algebraically (as an example, the partial order *ALU (register to register)* in Figure 8 can be specified by the expression `PCIU -> IFU + ALU`).

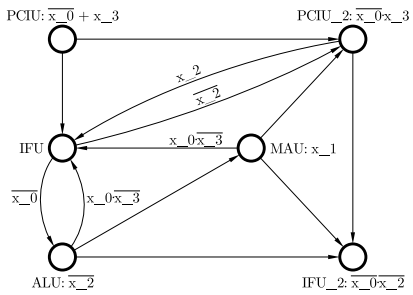



Figure 9: Compact representation of ARM Cortex M0+ instructions.

The CPOG encoding plugin SCENCO [11] can be used to automatically derive instruction opcodes minimising the area of the resulting microcontroller.

The plugin allows one to set a desired opcode length and reserve opcode bits in certain instructions. Several encoding algorithms are supported – see Table 1. Note that choosing good opcodes has a significant impact on the area of the resulting microcontroller. Having computed the optimal opcodes for the given partial orders, it is possible to represent them compactly as a CPOG, see Figure 9. Furthermore, it is possible to synthesise the processor microcontroller that can execute all 11 instruction classes. The microcontroller can be automatically synthesised as a Digital Circuit in WORKCRAFT – see Figure 10, or exported as a Verilog netlist for processing with traditional EDA toolkits.

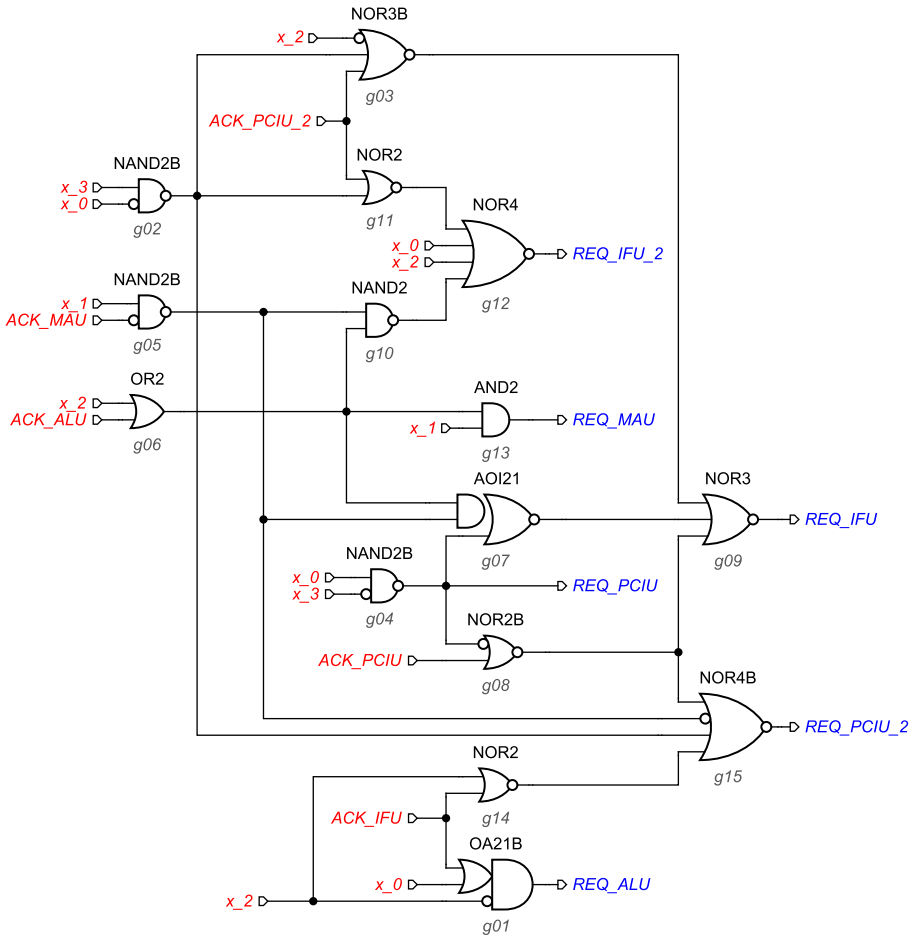



Figure 10: Synthesised microcontroller.

| Encoding algorithm | Microcontroller area, $\mu m^2$ | Gate count |
|---|---|---|
| Sequential assignment | 468 | 28 |
| Random, 10 samples | 436 | 28 |
| Random, 100 samples | 364 | 24 |
| Random, 200 samples | 372 | 24 |
| Heuristic [11], 10 seeds | 312 | 21 |
| Heuristic, 100 seeds | 256 | 16 |
| Heuristic, 200 seeds | **248** | **15** |
| SAT-based [16] | 256 | 16 |

Table 1: Comparison of CPOG encoding algorithms.

### 5.5 Accident Investigation: Ladbroke Grove Rail Crash

In this case study WORKCRAFT is used for modelling Ladbroke Grove rail crash [20]. Ladbroke Grove, London, was the scene of a serious railway accident in October 1999. An outbound diesel train collided with a high speed train at the combined velocity of 130mph, with 31 people killed and over 500 injured. The immediate cause of the disaster was that the diesel train passed signal SN109 at red, although there were many other contributing factors.

The SON model in Figure 11 captures the details of the Ladbroke Grove rail crash. It consists of five occurrence nets that represent separate parties of the accident:

- signals – represents the track signals that diesel train passed by in sequence. The first signal SN43 is green (proceed). The next two signals SN63 and SN87 are both yellow (caution). The last one SN109 is red (stop).
- driver – shows the behaviours of the diesel train driver. It captures the operation of the train speed control with seven speed notches (1-7) and the brake actions.
- diesel train – models the speed of diesel train as a reaction to driver actions.
- control centre – is the behaviours of the signaller who was in charge of monitoring the situation.
- HST – models the high speed train shortly before collision.

WORKCRAFT allows one to verify several kinds of properties of SON models, including behavioural, structural and temporal consistency.

## 6 WORKCRAFT users

User-centred design process is at the heart of WORKCRAFT philosophy. WORKCRAFT is used for different purposes, such as education, research, and industrial circuit design to list a few. Therefore there are several categories of users: taught students, academics and research students, industrial engineers. The developers of WORKCRAFT also use it intensively in their research. It is not trivial to develop a tool that suits all these categories. Hence the developers made a point not

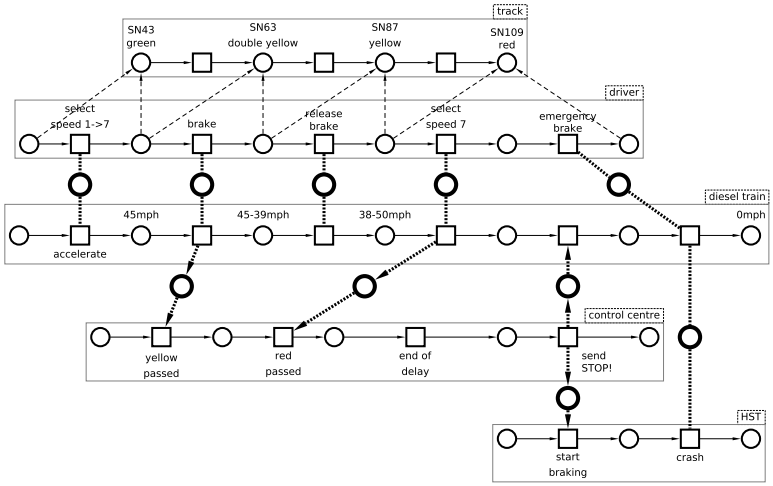



Figure 11: SONs model of the Ladbroke Grove rail crash.

to impose their design decisions on the users, but rather listen to their feedback and try to understand their needs. This resulted in an intuitive GUI. Historically there have been many reports of features as bugs because the users were confused due to lack of experience and understanding. Rather than dismissing these reports, an effort was made to improve the interface and address the sources of confusion.

Plenty of resources to support Workcraft users are available from the `http://workcraft.org/` website:

- binary distributions for Windows, Linux, Mac OS X and a link to the source code at GitHub;
- educational materials in the form of tutorials and case studies;
- user manuals;
- guidelines for developers of new IGMs, plugins, and back-ends;
- news track and announcement of the training events.

### 6.1 Education

Workcraft and the developed educational materials have been deployed in the learning process at Newcastle University as a part of undergraduate module *CSC3324: Understanding Concurrency* for Stage 3 students within the *Research-led Teaching* initiative. Not only this enhanced the learning process for the students, but the developers have also benefited by gathering invaluable data about

- how novice users attempted to install and use Workcraft;
- unexpected use patterns;
- features which students found difficult or confusing;

– bug reports;
– suggestions for improvements;
– feature requests.

This data helped to improve the stability and usability of Workcraft. The developers directly engaged with the students and tried to resolve the issues as soon as possible. In particular, several versions of Workcraft were released in the course of the module, so that the students could see their feedback making real difference.

Workcraft is also used by other universities to support teaching of asynchronous circuit design, in particular by Technical University of Denmark for *02204: Design of Asynchronous Circuits* course and by Southampton University for the *MSc Systems on Chip* course.

### 6.2 Research

Researchers use Workcraft for formal modelling or designing various systems. Accessible simulation, synthesis and verification features enhance this process and automate some routine and tedious parts of it. Researchers occasionally come up with new IGM formalisms. The extendible architecture of Workcraft makes it possible to implement them as plugins, benefiting from the functionality that is already present in the Workcraft core and existing plugins.

An important part of researcher survival is funding applications. Workcraft has been used, and is planned to be used, in several research proposals. There are several ways in which Workcraft is useful for this purpose:

– As a platform for developing solutions for the problems stated in the project proposal.
– As an impact and dissemination channel: the research results are encapsulated in Workcraft to enable non-expert users to reap the benefits.
– Establishing industrial links which strengthen the impact cases. Industry also helped to focus on industrially relevant topics and discover new areas of research (e.g. analogue-to-asynchronous interfaces) generating ideas for research proposals.
– In the REF2014 UK national exercise, one of the Newcastle socio-economic impact case studies was *Worldwide Adoption of Asynchronous Circuits and Improved Business Process Modelling* – it was judged to be world-leading. This case study included the development of techniques and tools for the synthesis of asynchronous systems (these tools are now incorporated into Workcraft as back-ends). This achievement can be leveraged in future funding applications.

On the practical side in research papers related to Petri nets and asynchronous circuits one usually needs a number of diagrams. Using LaTeX primitives or general-purpose graphics editors is tedious and time-consuming. Workcraft export feature allows one to produce high quality diagrams of the supported IGMs in a variety of graphic formats with minimum effort. This also improves the

consistency between the published diagrams and the models that were actually used in the research.

### 6.3 Industry

Interaction with industry is of paramount importance for the researchers living in the ivory tower of academia, and industrial uptake is a crucial criterion of success for research software. We are proud that Workcraft is used by several hardware design companies to develop real-life electronic circuits.

Interaction with industry proved to be of enormous use. It provided real-life case studies, gave insight into problems faced by industry where research is necessary and what kind of solutions can be actually implemented in silicone. We had a number of joint projects and publications with industrial partners and exchanged numerous visits. Several tutorials and teaching materials were developed to address educational needs of engineers. The feedback and feature requests submitted by the engineers helped to improve the usability of Workcraft in industrial context as well as identify and implement missing links in the design flow.

## 7 Workcraft timeline

Workcraft started in 2006 as a PhD research project of Ivan Poliakov to provide a modelling and simulation tool for PNs and STGs. Additional requirements were to make it cross-platform, user-friendly, and easily extendible. The result proved to be a convenient framework for a range of PN-like models (subsequently called IGMs), and other researchers started adding their favourite formalisms to Workcraft.

Figure 12 captures the history of Workcraft as a SON model. The central part of the diagram represents the development of Workcraft versions, highlighting some major features associated with each release. The shaded vertices denote new IGMs introduced in the corresponding release. Developers of the codebase and their contribution to specific features are listed at the top. The main dissemination results in terms of exhibition demonstrations, training sessions, and teaching are at the bottom of the diagram.

The main milestones in Workcraft development are as follows:

- Workcraft 1 series (A New Hope) was implemented in Java with native libraries for OpenGL graphics. It supported PNs, STGs, ACMs and Digital Circuits models. It also worked as a sandpit for trying different flavours of self-timed datapath models with various token game semantics. This lead to formalisation of a folklore static DFS model and its extension with dynamic elements. Figure 13 shows Workcraft v1.0 when simulating a motor control system with asynchronous communication between slow speed controller, fast torque controller, and the slowest adaptive parameter tuner [13].

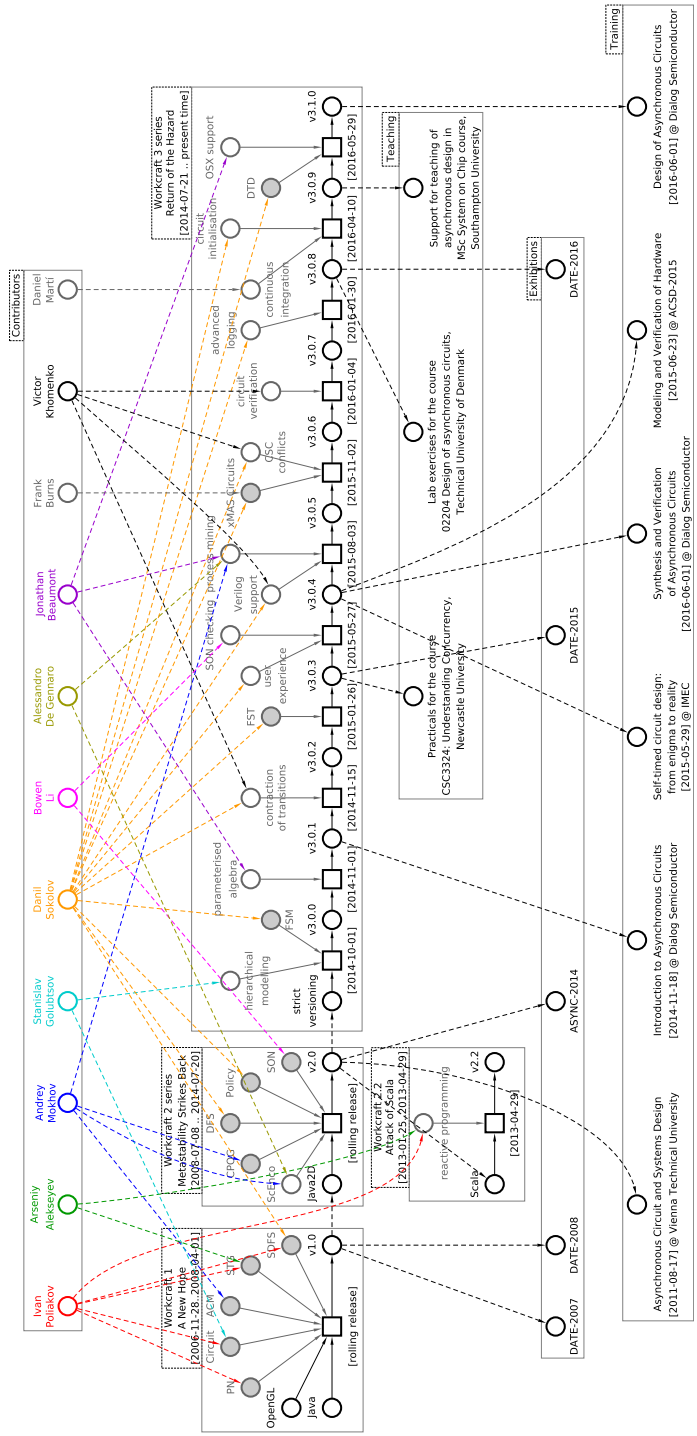



Figure 12: WORKCRAFT timeline expressed in SONs.

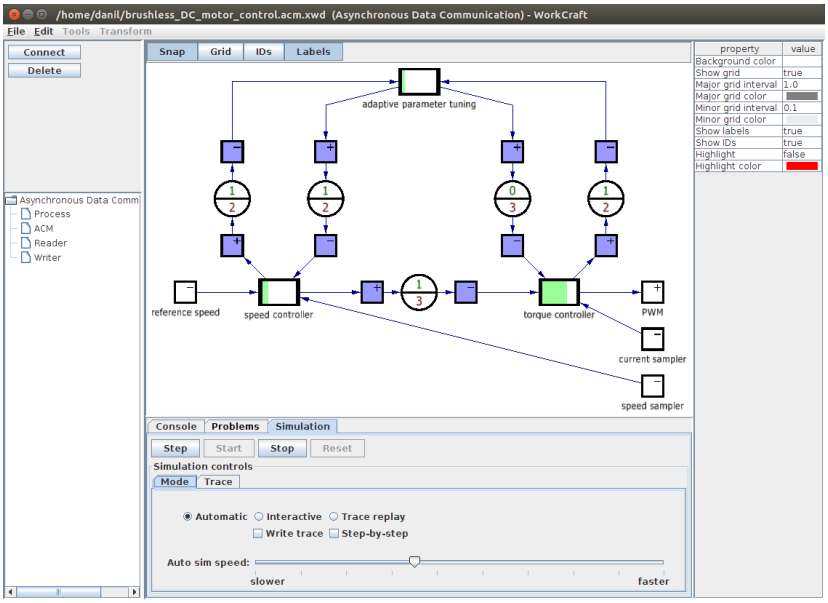



Figure 13: Modelling asynchronous communication in WORKCRAFT v1.0.

– WORKCRAFT 2 series (Metastability Strikes Back) was a major revision of the core functionality, plugin architecture, file exchange format, and rendering engine. Several new IGMs we added in this series, namely CPOGs, SONs and Policy nets. A screenshot in Figure 14 shows a development of Petri net representation of algorithm for generating a pair of public and private keys used in AES encryption in WORKCRAFT v2.0 [26].

– WORKCRAFT 2.2 branch (Attack of Scala) was a heroic attempt to direct the development into reactive programming paradigm using Scala functional programming language. Due to inherent complexity of the design concepts only few researchers could contribute to this development. As a result this branch was abandoned.
– WORKCRAFT 3 series (Return of the Hazard) is currently under active development and follows regular schedule of releases (as opposed to rolling releases in the previous development). The focus of this series is on improving user experience and developing tutorial materials. Several new IGMs were also added, namely FSMs, FSTs and DTDs. Figure 15 shows a screenshot of WORKCRAFT v3.1.0 when designing an asynchronous controller for a basic buck converter [5].

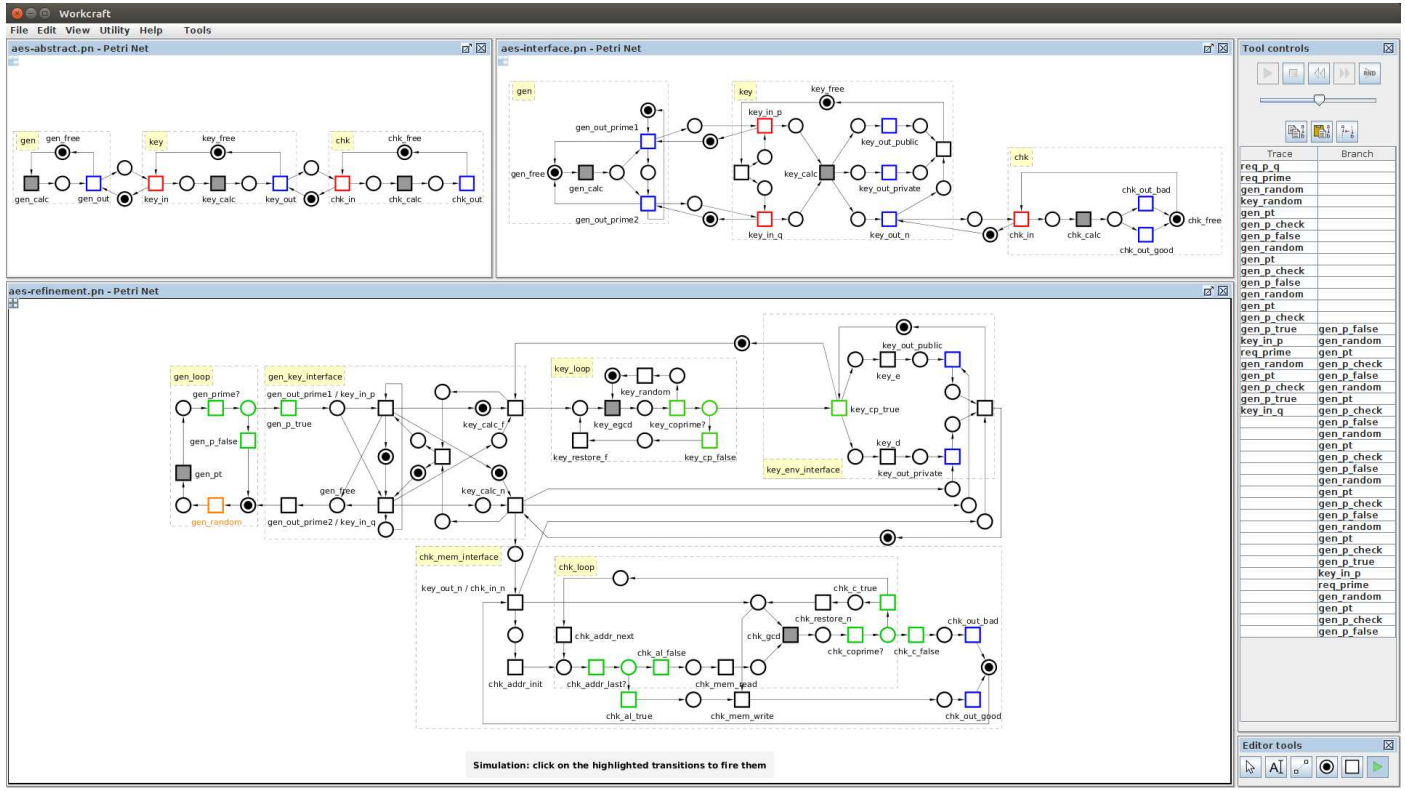



Figure 14: Refinement of algorithm for generation of AES keys using Petri nets in WORKCRAFT v2.0.

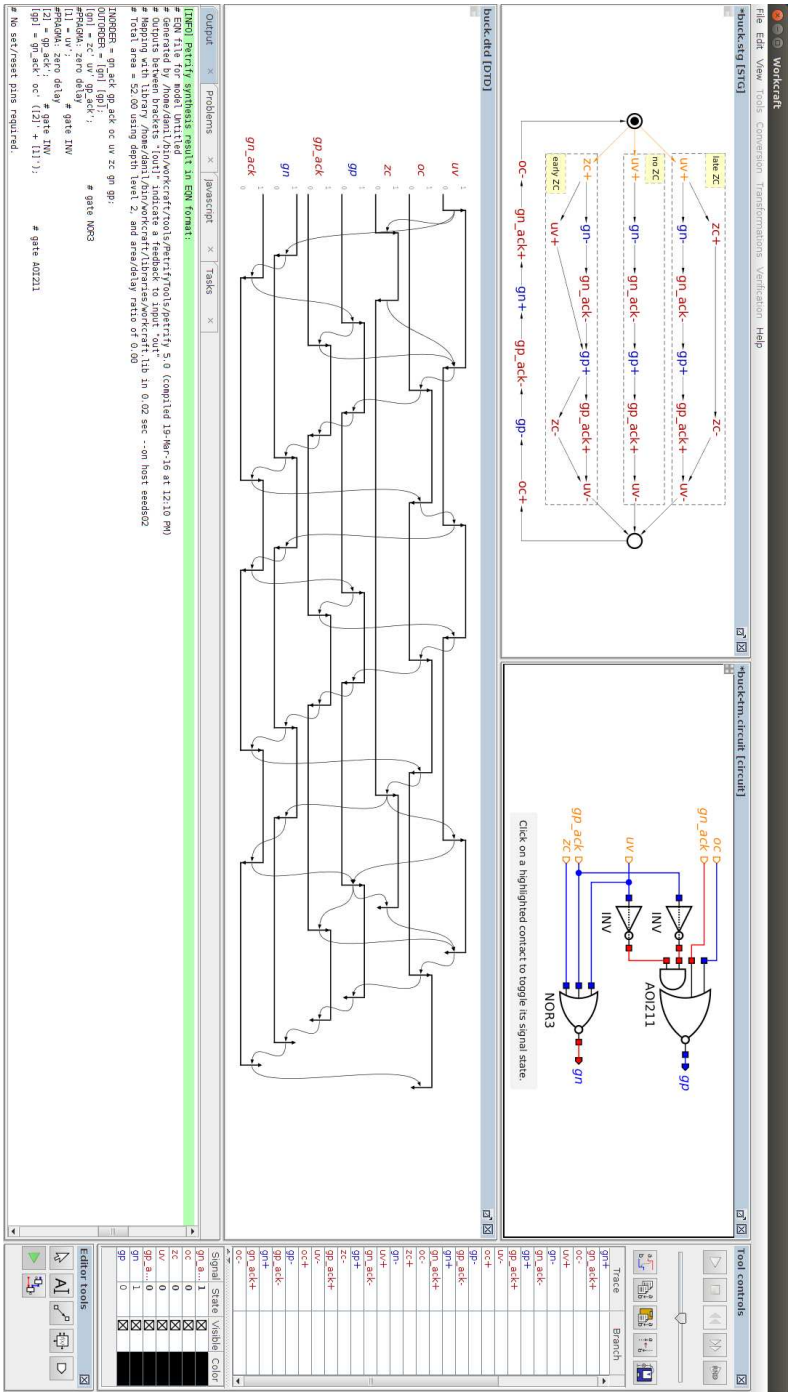



Figure 15: Design of a controller for basic buck converter in WORKCRAFT v3.1.0.

## 8 Conclusions

Workcraft is a versatile framework for capturing, simulation, synthesis and analysis of IGMs. It opens access to all the goodness hidden in command-line academic tools. This has significantly increased the user base, including industrial users, and thus the research impact of this software. The future plans include:

- Promoting Workcraft to an even wider audience. This will be achieved by creating and delivering tutorials to academic and industrial audience, development of the website [4], and improving the user experience based on their feedback.
- Encouraging other researchers to integrate their tools as back-ends.
- Extending Workcraft by adding new IGMs and implementing new features for existing ones. This will often be based on user requests. For example, waveforms model will be added in near future based on a request from an industrial partner.
- Finding other areas of application for the supported methods. For example, techniques based on STGs and speed-independent circuits appear to be well suited for modelling and analysis of Genetic Regulatory Networks [6].

**Acknowledgements**

The work on Workcraft was partially supported by Dialog Semiconductor, Maxeler Technologies, Royal Society (Research Grant "Computation Alive"), and the following EPSRC research grants:

- SElf-timed DATapath synthEsis (SEDATE) – EP/D053064/1
- Self-Timed Event Processor (STEP): EP/E044662/1
- Globally Asynchronous Elastic Logic Synthesis (GAELS) – EP/I038551/1
- Parallel Model Checking – GR/M99293/01
- Design And Verification of Asynchronous Circuits (DAVAC) – EP/C53400X/1
- Asynchronous design for Analogue electronics (A4A) – EP/L025507/1
- UNderstanding COmplex system eVolution through structurEd behaviouRs (UNCOVER) – EP/K001698/1
- Dataflow Computation à la Carte – Newcastle University Impact Acceleration Account EP/K503885/1

## References

1. ARM Cortex-M0+ technical reference manual. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0432c/index.html`.
2. Petrify homepage. `http://www.cs.upc.edu/~jordicf/petrify/`.
3. UnfoldingTools homepage. `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/UnfoldingTools/current/`.
4. Workcraft homepage. `http://workcraft.org/`.

5. Towards asynchronous power management. In *Proc. IEEE Faible Tension Faible Consommation(FTFC)*, 2014.
6. R. Banks, V. Khomenko, and J. Steggles. A case for using signal transition graphs for analysing and refining genetic networks. *Electron. Notes Theor. Comput. Sci.*, 227:3–19, 2009.
7. C. Brej. Wagging logic: Implicit parallelism extraction using asynchronous methodologies. In *Proc. Application of Concurrency to System Design (ACSD)*, pages 35–44. IEEE Computer Society, 2010.
8. F. Burns, D. Sokolov, and A. Yakovlev. GALS synthesis and verification for xMAS models. In *Proc. Design, Automation & Test in Europe Conference & Exhibition*, pages 1419–1424, 2015.
9. T.-A. Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, MIT Laboratory for Computer Science, 1987.
10. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. PETRIFY: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.
11. Alessandro de Gennaro, Paulius Stankaitis, and Andrey Mokhov. A heuristic algorithm for deriving compact models of processor instruction sets. In *International Conference on Application of Concurrency to System Design (ACSD)*, pages 100–109. IEEE, 2015.
12. V. Varshavsky (Editor). *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, 1990.
13. F. Hao. *Asynchronous Data Communication Mechanism Models in Matlab and Their Applications*. PhD thesis, Newcastle University, 2007.
14. M. Koutny and B. Randell. Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundam. Inform.*, 97(1-2):41–91, 2009.
15. A. Mokhov. *Conditional Partial Order Graphs*. PhD thesis, Newcastle University, 2009.
16. A. Mokhov, A. Alekseyev, and A. Yakovlev. Encoding of processor instruction sets with explicit concurrency control. *IET Computers and Digital Techniques*, 5(6):427–439, 2011.
17. A. Mokhov and V. Khomenko. Algebra of parameterised graphs. *ACM Transactions on Embedded Computing*, 13(4s), 2014.
18. D. Muller and W. Bartky. A theory of asynchronous circuits. In *Proc. Int. Symp. of the Theory of Switching*, pages 204–243, 1959.
19. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
20. The Rt Hon Lord Cullen PC. The Ladbroke Grove rail inquiry, 2001.
21. I. Poliakov, V. Khomenko, and A. Yakovlev. WORKCRAFT – a framework for interpreted graph models. In *Proc. Application and Theory of Petri Nets (ATPN)*, volume 5606 of *Lecture Notes in Computer Science*, pages 333–342. Springer-Verlag, 2009.
22. I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov, and A. Yakovlev. Automated verification of asynchronous circuits using circuit Petri nets. In *Proc. Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 161–170, 2008.
23. I. Poliakov, D. Sokolov, and A. Mokhov. WORKCRAFT: a static data flow structure editing, visualisation and analysis tool. In *Proc. Application and Theory of Petri Nets (ATPN)*, pages 505–514, 2007.

24. L. Rosenblum and A. Yakovlev. Signal graphs: from self-timed to timed ones. In *International Workshop on Timed Petri Nets, Torino, Italy, 1985*, 1985.
25. D. Sokolov, I. Poliakov, and A. Yakovlev. Analysis of static data flow structures. *Fundamenta Informaticae*, 88(4):581–610, 2008.
26. D. Sokolov and A. Yakovlev. GALS partitioning by behavioural decoupling expressed in Petri nets. In *Proc. Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 17–26, 2014.
27. A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, pages 429–528. Springer, 1998.