Synthesizing IEC 61499 Function Blocks to hardware

Hammond Pearce, Partha Roop
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand
Email: {hammond.pearce, p.roop}@auckland.ac.nz

***Abstract*—IEC 61499 Function Blocks is the emerging standard for distributed control of industrial automation systems. Within this domain, it is common for devices to need robust timing and functional guarantees. Unfortunately, many industry-standard approaches can only provide analysis for functional correctness. This is primarily because IEC 61499 is usually executed upon architectures and systems that are not amenable to static timing analysis, such as those that utilize general purpose processors or embedded operating systems. Hence, designers must rely on timing values obtained by measurement-based approaches, a methodology that will not be able to provide strong timing guarantees. In this paper, we propose a novel methodology for instead compiling Function Blocks to FPGA-based hardware, using synchronous semantics for their execution. To avoid restricting the functionality of these implementations, complex mechanisms such as software updates and distributed networking can be provided via on- or off-chip coprocessing. This overall approach provides for straightforward timing analysis, as well as a methodology for designing reliable, efficient controllers with extremely fast response times.**

I. Introduction

Currently, there is a trend within the industrial automation domain known as "Industry 4.0". This is pushing for "smarter" process lines which feature increased levels of mechanization, networking, and control, with the ultimate goal being the emergence of digital factories [1], [2].

Within this sphere, IEC 61499 Function Blocks (FBs) [3] is emerging as a design standard of choice [4]. It allows for model-driven engineering, where components and their behaviours can be rigidly defined and encapsulated into their FB specification. Then, these FBs can be composed into communicating networks, allowing for complex functionality to be realised while maintaining the re-usability of each individual components.

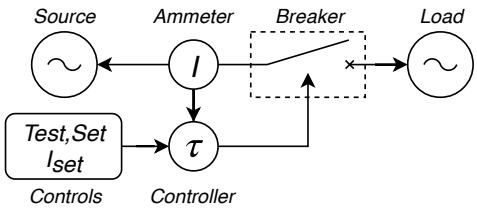


Figure 1: An example overcurrent protection breaker.

However, contemporary implementations of IEC 61499 FBs do not fully synergize with the requirements of Industry 4.0. Take the example of a smart-grid overcurrent breaker, as depicted in Figure 1. Here, the controller τ must open its associated breaker to terminate electrical current flow when an overcurrent event occurs, i.e. an unsafe level of electrical current has been flowing for an unsafe period of time. This is an example of a safety-critical system, where the overall system must adhere to not just strict functional requirements, but also to precise timing deadlines, i.e. its Worst Case Execution Times (WCETs) must be known. These are also known as Precision Timed Industrial Automation (PTIA) systems [5].

Unfortunately, it can be very difficult to obtain timing properties for common implementations of PTIA systems. This is because they are most commonly implemented using modern Programmable Logic Controllers (PLCs) architectures. These are usually based on general-purpose processors designed by ARM, Intel, and Freescale, and can also feature complex runtimes and/or Real-Time Operating Systems (RTOSs). These underlying architectural decisions can increase performance and flexibility compared with old-fashioned bare-metal and analog approaches, but come with a drawback — they greatly complicate static timing analysis and verification. As a result, designers must turn to simple measurement-based approaches to derive their WCET values [6], [7]. Such approaches can never provide robust guarantees, as it is very difficult to ensure that all possible execution paths have been examined during testing.

In this paper, we propose an alternative approach for PTIA. We derive a methodology for FBs to be synthesized directly to FPGA-based hardware. This is done by adopting synchronous semantics, which model software components as if they were concurrently-operating hardware modules, and have already been used to demonstrate time-predictable behaviour of IEC 61499 FBs [7]–[9]. Our generated hardware can thus be analysed to get extremely tight timing bounds, as well as granting extremely rapid and consistent system responses.

The rest of this paper is organised as follows. Section II provides an introduction to the IEC 61499 standard via the overcurrent breaker PTIA example. Section III then provides an overview of related work and its shortcomings. Then, Section IV discusses the methodology of our approach. Section V

then demonstrates the efficacy of our approach via a set of benchmarks including the overcurrent protection breaker. Finally, Section VI provides concluding remarks.

II. IEC 61499 Function Blocks (FBs)

In order to introduce IEC 61499 FBs, we will discuss the operation of the overcurrent breaker from Figure 1. In general, fault detection via such systems is implemented through an Inverse Definite Minimum Time (IDMT) curve, covered by the IEC 60255 standard [10]. The curve is defined by Equation 1.

$$t = \frac{K\beta}{\left(\frac{I}{I_{set}}\right)^{\alpha} - 1} \tag{1}$$

Here, t is the safe overcurrent operating time, K is a time multiplier, I and I_{set} are the measured and nominal currents, and α and β are two tuning parameters. By changing these values, different IDMT curves with different slopes can be generated. They are typically selected as required by the given application. For this case study, a *very inverse* type curve is used, with $\alpha = 1.0$, $\beta = 13.5$, and time-multiplier $K = 0.1$. This curve is depicted in Figure 2.

Example II.1. *Using Equation 1 with our chosen tuning parameters and an overcurrent magnitude of 10, i.e. $I = 10 \times I_{set}$, we get a safe operating time of 0.15 s. However, for the same settings, if the overcurrent magnitude is lower, e.g. 5 times, the safe operating time is longer, 0.34 s.*

We can implement the controller for the overcurrent breaker using the IEC 61499 FBs standard [3]. FBs totally encapsulate the memories and behaviours of the components that they represent, allowing for flexible and re-usable component definitions. FBs can be composed into networks of blocks communicating over event-data interfaces, updating variable data lines when their associated events trigger. There are three major types of FB, introduced through Figure 3.

- Composite Function Blocks (CFBs) contain networks of other FBs. In our case, the controller can be represented by the CFB presented in Figure 3a. This contains within it the network of FBs presented in Figure 3b.
- Basic Function Blocks (BFBs) contain within themselves local copies of I/O variables, as well as their own internal variables. They are managed by an Execution Control Chart (ECC), which is a state machine that can advance based on input events and local data variables. Associated with each ECC state is a set of *actions*, which can be output events and/or data-manipulating algorithms. Our *test/set/unsafe* logic for opening and closing the breaker can be implemented via the ECC in Figure 3c.
- Service Interface Function Blocks (SIFBs) provide implementation-specific functionality allowing for interaction with hardware and peripheral components. In our case, appropriate SIFBs would provide the source/destinations for the I/O connections on the parent CFB in Figure 3a.

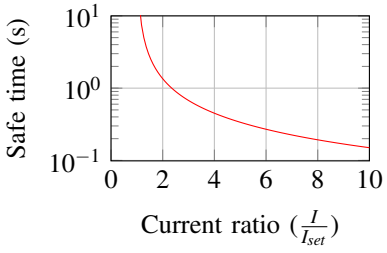
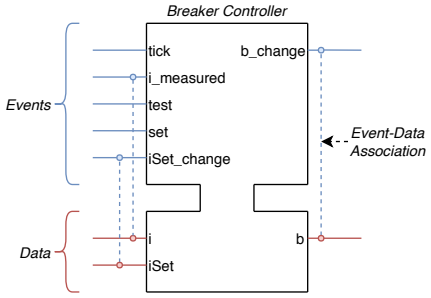
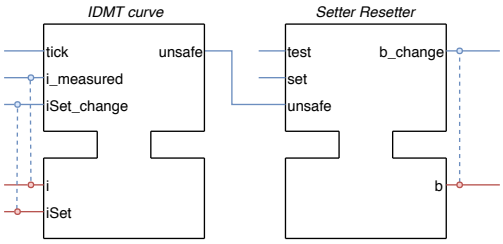


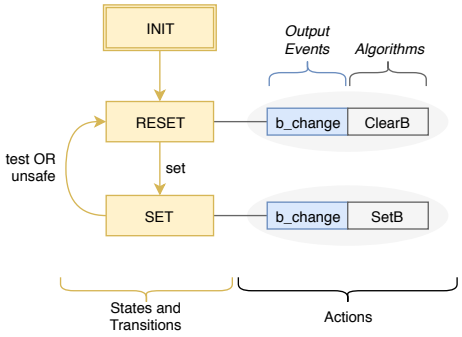
Figure 2: The IDMT curve for this case study.



(a) CFB for the breaker controller τ.



(b) BFB network contained within the breaker CFB in Figure 3a.



(c) ECC contained within "Setter Resetter" BFB in Figure 3b.

Figure 3: Overcurrent breaker sample IEC 61499

III. Related Work

There has been plenty of interest in executing time-predictable IEC 61499 networks in the literature. Primarily, these all focus on the same issue — addressing the unpredictable nature of current implementations, which rely both on general purpose processors as well as complex runtimes and/or RTOS. For instance, neither Forte [11] nor IsaGRAF [12], both popular IEC 61499 execution frameworks, have had worst-case timing analysis performed [7].

In part, this is due to the the standard model of computation for IEC 61499, which is fundamentally event-triggered in nature, and tends to be antithetical to static timing analysis [13]. As a result, the synchronous approach for IEC 61499 is suggested for use when concrete timing values are required [7]. This is because synchronous languages like Esterel [14] and SCADE [15] model concurrently-executing software components as if they were parallel hardware modules evolving discretely along with the progression of a logical tick. This simplifies program flow and analysis, especially in the face of variable data, which can usually only pass between modules on tick boundaries.

Many time-predictable methodologies for IEC 61499 thus focus on using the synchronous approach in conjunction with time-predictable architectures taking the place of general purpose processors [5]. For instance, in [7], the low-performance MicroBlaze [16] architecture was used for sequential execution of code; in [8], the Precision Timed (PRET) architecture FlexPRET [17] was used as a platform for predictable multi-threaded execution; and in [9], the time-predictable multi-core architecture T-CREST [18] had a set of FB tasks distributed over it.

However, even though synchronous languages are emulated as if they were hardware, no effort to compile IEC 61499 FBs to hardware with synchronous semantics has yet been undertaken. Indeed, the only effort to run FBs directly on FPGAs is presented in [19], where O'Sullivan *et al.* propose a custom architecture in VHDL for greatly accelerated execution of IEC 61499. Their approach however has a key drawback: it does not formalise the communication or scheduling between parallel execution of blocks, instead relying on the designer to ensure that no conflicts can occur. Model driven engineering should instead rely on correct by construction methodologies, such as the synchronous approach. Furthermore, they do not present mechanisms for the blocks to communicate over network interfaces, nor discuss how the networks could be maintained or updated.

IV. Methodology

Adopting synchronous semantics for IEC 61499 means that the execution model for concurrently executing FBs is clearly defined. Each FB is an independently executing unit of hardware, which performs a complete update (i.e captures inputs, performs computation, and emits outputs) in every clock cycle. A consequence of this is that FBs must communicate using delayed synchronous semantics — they read values which were emitted in the previous tick. This is common in synchronous languages, such as in Esterel, where this behaviour is described with the '`pre`' keyword [20].

A. Compiling IEC 61499 FBs to Verilog

In this paper, we focus on compiling the three main types of Function Block: BFBs, CFBs, and SIFBs, where the SIFBs can interface with physical or other digital peripherals, such as switches, LEDs, motor drivers, or co-processors. They thus provide connectivity and external functionality to the overall network.

1) Compiling Basic Function Blocks (BFBs): The generalised hardware architecture for BFBs is depicted in Figure 4. This can be broken down into three stages, the input capture (*INP*), the state and data management (*EXE*), and the output emission (*OUT*).

INP is made up of sets of local data registers combined with forwarding multiplexors. Forwarding data is used to allow the rest of the circuit to operate within a single-cycle if the data is needed this cycle. The registers are updated (and data is forwarded) using the associated event line as an enable signal.

EXE is built up from a mealy-style finite state machine derived directly from the ECC within the original BFB. While it appears that a moore-style machine would be a better fit, as outputs are associated with states in IEC 61499 ECCs, they actually only trigger their output events and data algorithms *on entry* to a state (i.e. if no state advancement occurs in a given tick, no outputs are emitted). This is not compatible with a moore-style machine, which assert their outputs continuously while in a given state.

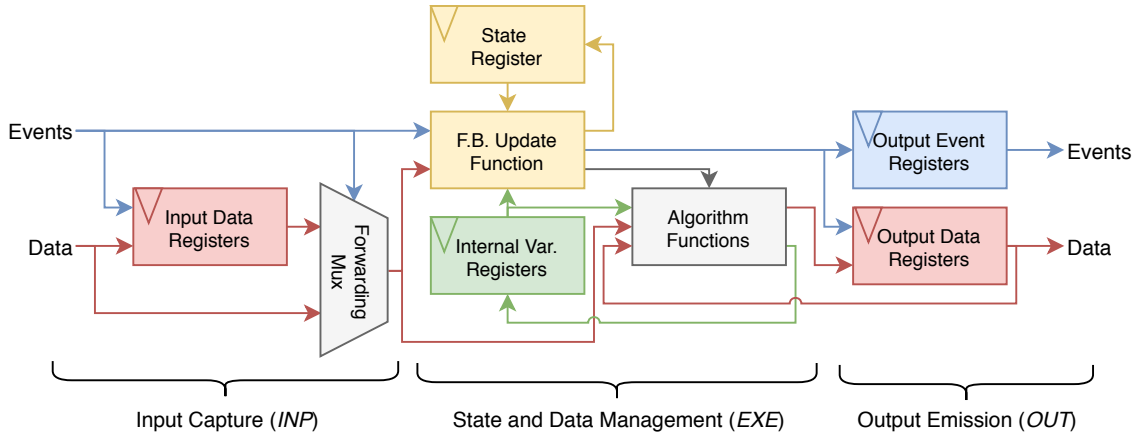


Figure 4: Generalised hardware architecture for IEC 61499 Basic Function Blocks (BFBs).

```
1  module SetterResetter (...); // i/o omitted
2
3  ... // var declaration/assignment omitted
4
5  always@(posedge clk) begin
6    //reset logic omitted
7
8    // no input data variables to capture
9
10   // clear output events
11   b_change = 0;
12
13   // clear algorithm triggers
14   SetB_alg_en = 0;
15   ClearB_alg_en = 0;
16
17   // (FB Update Function)
18   // advance state, set output events, trigger algorithms
19   case(state)
20     `STATE_s_init: begin
21       state = `STATE_s_reset;
22       b_change = 1;
23       ClearB_alg_en = 1;
24     end
25     `STATE_s_reset: begin
26       if(set) begin
27         state = `STATE_s_set;
28         b_change = 1;
29         SetB_alg_en = 1;
30       end
31     end
32     `STATE_s_set: begin
33       if(test || unsafe) begin
34         state = `STATE_s_reset;
35         b_change = 1;
36         ClearB_alg_en = 1;
37       end
38     end
39   endcase
40
41   // algorithm functions
42   if(ClearB_alg_en) begin
43     b = 0;
44   end
45
46   if(SetB_alg_en) begin
47     b = 1;
48   end
49
50   //output data registers
51   if(b_change) begin
52     b_O = b;
53   end
54
55   end
56 endmodule
```

Listing 1: Compiled Verilog for Setter Resetter BFB

Hence, each state in the *ECC* becomes a state in the new machine, with a state's associated actions (event outputs and algorithm invokations) duplicated over its incoming transitions. Therefore, within any given clock cycle, the internal state machine can advance, and the associated outputs will be triggered. This places a restriction on algorithms, which are represented internally by purely combinatorial functions: they must not depend on a clock cycle to function.

Finally, *OUT* is simply a set of output registers for outgoing events, which update every cycle; and data, which update according to their associated events (used as enable signals).

Example IV.1. *The Setter Resetter BFB for the Overcurrent Breaker case study from Section II can be compiled to the Verilog presented in Listing 1. The comments in this listing link to the blocks in Figure 4. As can be seen, each of the ECC states in Figure 3c is placed into the F.B. Update Function in lines 19-29. This block of code is also responsible for outputting events, as well as triggering the execution of algorithms (e.g* `SetB`*,* `ClearB`*).*

2) Compiling Composite Function Blocks (CFBs): In the proposed approach, CFBs are converted to a simple netlist of modules connected with Verilog '`wire`'s. They contain no logic nor registers of their own, which means that they introduce no overhead to the design.

3) Compiling Service Interface Function Blocks (SIFBs): SIFBs are compiled to special modules which pass their I/O up to the top level entity within the Verilog codes, allowing for custom logic to be specified and realised within the design. Then, the SIFB links can be manually interfaced with any user-specified peripheral, such as LEDs or switches.

These SIFBs designs can also provide networking support through interfaces with coprocessors, such as those within an Altera HPS system as present in the Terasic DE10-Standard development board [21]. For instance, a remote-request command delivered over TCP/IP can be exposed to the network as a single environmental event input line which may trigger at any time. This allows any networking process to be simply modelled as general purpose I/O, simplifying local functional and timing verification.

B. Deriving WCETs via Worst Case Reaction Times (WCRTs)

Synchronous languages, as they operate over logical ticks, introduce the concept of a Worst Case Reaction Time (WCRT). This is the maximum execution time taken for any given tick. As a given program may take multiple logical ticks to complete an execution, WCET of a synchronous program can be computed by multiplying WCRT by the maximum number of logical ticks required.

With our compilation approach, BFBs and SIFBs take one clock cycle to execute, and CFBs take no cycles (as they are just connecting wires). Clock cycles map to synchronous logical ticks exactly. Therefore, the WCET of a given program is simply the longest single trace of BFB and SIFB that program flow must pass through in order to produce a given output.

Example IV.2. *In the overcurrent breaker case study, the longest execution trace comes from the* `tick` *event. This comes from the following trace: (1)* `tick` *is created in an external SIFB. (2) IDMT Curve captures* `tick` *and emits* `unsafe`*. (3) Setter Resetter captures* `unsafe` *and emits* `b_change`*. (4) An external SIFB captures* `b_change`*.*

This means that the WCET of the case study will be $4\times$*WCRT.*

It is important to note that the WCET computed via this method does not take into account the time taken for any execution of C code required for networking functionalities. This is because this complexity is abstracted away and considered external to the network (i.e. seen as part of the environment). See Section IV-A3 for further details.

Benchmark Name	IEC 61499 Design			Compiled Hardware				
	Total FB instances	Networking?	Longest chain	ALMs	Registers	Max Frequency (MHz)	WCRT (cycles)	
Overcurrent Breaker	5	Yes	4	82*	35*	88.35	1	
Traffic Lights	10	Yes	8	467*	203*	72.9	1	
Water Pump	3	No	3	41	27	217.2	1	
Benchmarks from [9]								WCRT from [9] (T-CREST cycles @ 50MHz)
VVI Mode Pacemaker	32	No	4	164	213	157.78	1	8706
Bottling Plant	36	No	7	257	264	187.48	1	22505

Table I: Benchmark results

V. Results

To demonstrate the efficacy of the proposed approach, a number of benchmarks were analysed, including the *Overcurrent Breaker* case study from Section IV. All benchmarks were written using IEC 61499, with algorithms written in Structured Text if applicable, before being compiled to Verilog using the process introduced in Section 4. They are presented in Table I. Aside from the case study, two other benchmarks are presented, *Traffic Lights*, which allows for normal timed round-robin R-G-Y progression of a 4-way intersection as well as remote overrides for manual operation; and *Water Pump*, which manages the control logic for the balancing of two industrial water storage tanks. Within the *Overcurrent Breaker*, the IDMT curve was approximated using a pessimistic seven-stage step-function to provide a good tradeoff between accuracy and hardware consumption (as division is quite costly).

A. Comparison Benchmarks

To compare our approach with the time-predictable processor methodology, we also implemented two of the benchmarks in [9], and compare with their published 4-core scratchpad memory mode WCRT (this had the fastest execution time). *Bottling Plant* controls a model of a system which fills bottles with fluid. This contains a number of components, including doors, lasers, conveyors, a bottle filler, and a robotic arm model. *VVI Mode Pacemaker* contains a simplified pacemaker controller model.

B. Experimental Methodology

IEC 61499 FBs were specifed in standard XML-based formats and then compiled to Verilog. The Verilog was synthesized using Intel Quartus 17.0 for a Terasic DE10-Standard development board, targeting the onboard Cyclone V 5CSXFC6D6F31C6. Where networking blocks were necessary, they were provided using custom C compiled to the integrated ARM® Cortex™-A9 dual-core processor running Linux at 800MHz, and GPIO pins were exposed to SIFBs running within the IEC 61499 network via a bridge implemented using Quartus Qsys. This used 4280 Adaptive Logic Modules (ALMs) and 4096 registers, and was consistent across each of the networked benchmarks. So that the consumption of each network can be better judged, in Table I, the resource usage is presented less this overhead where applicable (*).

To calculate the max frequency for the compiled hardware, Quartus's TimeQuest Timing Analysis tool was used with the *slow 1100mv 85C* model.

As the integrated dual-core processor has the ability to update the FPGA hardware, special C code for performing software updates according to the IEC 61499 specification can be loaded into the Linux system with or without the data bridge.

C. Discussion

The results are presented in Table I. As can be seen, the hardware FB are all capable of running at at least 50 MHz, meaning that at these speeds, as they take only a single cycle to execute, their WCRT is just 20 ns. Compare this to the same networks running on T-CREST in [9], where the WCRT for an equivalent 50 MHz clock is 174.12 μs for the *VVI Mode Pacemaker* and 450.1 μs for the *Bottling Plant* — equivalent hardware for these benchmarks can execute at least six orders of magnitude faster!

Our benchmarks that used networking had a lower WCRT than those that did not. This was due to the Quartus Qsys-provided bridge for joining the integrated dual-core processor with the FPGA fabric, which constrained the maximum clock speed. Out of all the benchmarks, the *Water Pump* could execute the fastest, as it was the smallest design and did not feature the data bridge. In general, the hardware usage for each benchmark was extremely minimal, granting very high operational speeds while maintaining correct functionality.

VI. Conclusions

As the needs and scope of Industry 4.0 begin to encompass safety-critical PTIA systems, the requirements for reliable, performant, and formally verifiable controllers become apparent. However, typical approaches for these kinds of systems can involve RTOS and general-purpose processors, which greatly complicate timing analysis. In this paper we presented an alternative approach for achieving this functionality, via the automatic compilation of IEC 61499 Function Blocks (FBs) to FPGA-based hardware. Our approach developed a novel architecture which synchronously executes FBs in a single clock cycle, allowing for very accurate and simple timing analysis. Compared with an alternative approach for executing IEC 61499 synchronously on the time-predictable T-CREST architecture, our hardware can run at least six orders of magnitude faster while preserving identical functionality.

Future work could involve examining ECC algorithms requiring multi-cycle execution, investigating mechanisms for automatic derivation of the longest-length event-data chains, and research into a more automated compilation process for SIFB peripheral and networking function blocks.

Source Access

All examples used in this paper, and the source code for the hardware compiler, can be found online at https://github.com/PRETgroup/goFB.

References

[1] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, May 2014, pp. 1–4.

[2] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[3] *International Standard IEC 61499-1: Function blocks - Part 1: Architecture*, International Electrotechnical Commission Std., April 2013.

[4] V. Vyatkin, "IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, Nov 2011.

[5] M. M. Y. Kuo, S. Andalam, and P. S. Roop, "Precision timed industrial automation systems," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 1024–1025.

[6] M. Lv, N. Guan, Y. Zhang, Q. Deng, G. Yu, and J. Zhang, "A survey of WCET analysis of real-time operating systems," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, May 2009, pp. 65–72.

[7] M. Kuo, L. H. Yoong, S. Andalam, and P. Roop, "Determining the worst-case reaction time of IEC 61499 function blocks," in *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, July 2010, pp. 1104–1109.

[8] H. A. Pearce, M. M. Y. Kuo, P. S. Roop, and M. Biglari-Abhari, "RunSync: A predictable runtime for precision timed automation systems," in *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*, May 2016, pp. 116–123.

[9] H. Pearce, P. Roop, M. Biglari-Abhari, and M. Schoeberl, "Faster function blocks for precision timed industrial automation," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, May 2018, pp. 67–74.

[10] Int. Electrotechnical Commission (IEC), *Standard for Measuring Relays and Protection Equipment*, Std. 60 255, 2008.

[11] 4DIAC. (2015, Dec.) FORTE. [Online]. Available: http://www.eclipse.org/4diac/en_rte.php

[12] I. T. I. Inc. (2015, Dec.) ISaGRAF. [Online]. Available: http://www.isagraf.com/

[13] L. H. Yoong, P. Roop, V. Vyatkin, and Z. Salcic, "A synchronous approach for IEC 61499 function block implementation," *Computers, IEEE Transactions on*, vol. 58, no. 12, pp. 1599–1614, Dec 2009.

[14] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Sci. Comput. Program.*, vol. 19, no. 2, pp. 87–152, Nov. 1992.

[15] T. Le Sergent, "SCADE: A comprehensive framework for critical system and software engineering," in *SDL 2011: Integrating System and Software Modeling*, I. Ober and I. Ober, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 2–3.

[16] Xilinx. (2009) Microblaze processor reference guide. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[17] M. Zimmer, D. Broman, C. Shaver, and E. Lee, "FlexPRET: A processor platform for mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, April 2014, pp. 101–110.

[18] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449 – 471, 2015.

[19] D. O'Sullivan and D. Heffernan, "VHDL architecture for IEC 61499 function blocks," *IET computers & digital techniques*, vol. 4, no. 6, pp. 515–524, 2010.

[20] L. H. Yoong, P. Roop, V. Vyatkin, and Z. Salcic, "Synchronous execution of IEC 61499 function blocks using Esterel," in *2007 5th IEEE International Conference on Industrial Informatics*, vol. 2, June 2007, pp. 1189–1194.

[21] Terasic. (2018, November) DE10-Standard. [Online]. Available: http://de10-standard.terasic.com/