UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
MASTER DEGREE IN COMPUTER SCIENCE - EMBEDDED SYSTEMS

LAURENCE CRESTANI TASCA

Improving Programmable Logic Controller Performance Based on Scan Time Reduction

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Flávio R. Wagner
Coadvisor: Prof. Dr. Edison P. Freitas

Porto Alegre
April 2020

CIP — CATALOGING-IN-PUBLICATION

Laurence Crestani Tasca,

Improving Programmable Logic Controller Performance Based on Scan Time Reduction / Laurence Crestani Tasca. – Porto Alegre: PPGC da UFRGS, 2020.

85 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Master Degree in Computer Science - Embedded Systems, Porto Alegre, BR–RS, 2020. Advisor: Flávio R. Wagner; Coadvisor: Edison P. Freitas.

1. Programmable logic controllers, special architecture, data flow machines, circuit simulation theory, memoization technique, multi-cycle, pipeline, multicore, cycle-accurate simulator, McPAT. I. Wagner, Flávio R.. II. Freitas, Edison P.. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Rui Vicente Oppermann
Vice-Reitora: Prof[a]. Jane Fraga Tutikian
Pró-Reitor de Pós-Graduação: Prof. Celso Giannetti Loureiro Chaves
Diretora do Instituto de Informática: Prof[a]. Carla Maria Dal Sasso Freitas
Coordenadora do PPGC: Prof[a]. Luciana Salete Buriol
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

“The Lord is my shepherd, I lack nothing.
He makes me lie down in green pastures,
he leads me beside quiet waters,
he refreshes my soul.
He guides me along the right paths
for his name’s sake.
Even though I walk
through the darkest valley,
I will fear no evil,
for you are with me;
your rod and your staff,
they comfort me.
You prepare a table before me
in the presence of my enemies.
You anoint my head with oil;
my cup overflows.
Surely your goodness and love will follow me
all the days of my life,
and I will dwell in the house of the Lord
forever.”

— PSALM 23

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application-specific integrated circuit
CAS	Cycle-accurate simulator
DES	Discrete event simulators
HDL	Hardware description language
ISA	Instruction-Set Architecture
PID	Proportional integral derivative
PLC	Programmable logic controller
SE	Search to execute
VE	Verify to execute

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Since their appearance, programmable logic controllers (PLCs) are massively and predominantly used as the central controller in automation systems. Unfortunately, due to the poor performance of the majority of these devices, the typical role of PLCs in automation systems is restricted to a controller, since applications with more sophisticated computational requirements tend to be handled by external processing units along with the PLCs. To solve this issue, this work improves novel architecture proposals based on data flow machines, circuit simulation theory, and memoization technique to achieve a performance boost based on the scan time reduction. Along with the architectural improvements, this dissertation evaluates the impact of different execution units' types and quantities in a cycle-accurate simulator (CAS) that was specially developed to simulate the PLC cores. Furthermore, in order to perform a robust and complete evaluation, the silicon areas of the simulated architectures are calculated using the McPAT framework to establish the performance/area relationship of the simulated cores. Evaluation results show best scan time reductions of up to 68% for cores with single execution units and up to 89% for cores with multiple execution units, as well as a 50% scan time reduction with a minor impact on the silicon area.

Keywords: Programmable logic controllers, special architecture, data flow machines, circuit simulation theory, memoization technique, multi-cycle, pipeline, multicore, cycle-accurate simulator, McPAT.

RESUMO

Desde a sua introdução, os controladores lógicos programáveis (CLPs) são massiva e predominantemente usados como o controlador central em sistemas de automação. Infelizmente, devido ao fraco desempenho da maioria desses dispositivos, o papel típico dos CLPs nos sistemas de automação é restrito a um mero controlador, uma vez que aplicações com requisitos computacionais mais sofisticados tendem a ser tratados por unidades de processamento externas juntamente com os CLPs. Para resolver esse problema, este trabalho aprimora novas propostas de arquitetura baseadas em máquinas data flow, teoria de simulação de circuitos e técnica de memoização para obter um aumento de desempenho com base na redução do tempo de scan. Juntamente com as melhorias arquitetônicas, esta dissertação avalia o impacto de diferentes tipos e quantidades de unidades de execução em um simulador de precisão de ciclo, desenvolvido especialmente para simular os núcleos de CLP. Além disso, para realizar uma avaliação robusta e completa, as áreas de silício das arquiteturas simuladas foram calculadas usando o framework McPAT para estabelecer a relação desempenho/área dos núcleos simulados. Os resultados da avaliação mostram nos melhores casos reduções no tempo de varredura de até 68% para núcleos com unidades de execução única e até 89% para núcleos com várias unidades de execução, além de uma redução de 50% no tempo de varredura com um pequeno impacto na área de silício.

Palavras-chave: Controladores lógicos programáveis, arquiteturas especiais, máquinas data flow, teoria da simulação de circuitos, técnica de memoização, multi-cycle, pipeline, multicore, simulador de precisão de ciclo, McPAT.

1 INTRODUCTION

Since their introduction at the end of the 1960s, programmable logic controllers played an essential role in supporting industries to reach the high demanding levels of productivity of the 21st century. Due to their flexibility, robustness, and reliability, PLCs were massively used as the primary mechanism of control in several sectors in which automation technology is required (BOLTON, 2015).

In short, PLCs are electronic devices that use inputs and outputs to control a process using instructions defined in its programmable logic. Regarding performance, the principal PLCs' characteristic is the scan time, which is composed of the sum of the times required to read the inputs, execute the logics, and write the outputs (WEBB; REIS, 1998). Short scan time is crucial to a reduced response time, which is essential to many deterministic systems, such as motion control, for instance (LEWIS, 1998).

Since the first PLCs replaced the large, expensive, and not reliable relay logic systems, the first programming language adopted and still now popular among automation designers is the ladder diagram (BOLTON, 2015). Ladder diagrams use graphical electrical contacts to represent inputs, relay coils to represent outputs, and wires to represent the logic connections, similarly to the old relay systems schematics, as can be observed in Figure 1.1.

Figure 1.1: A symmetric ladder diagram

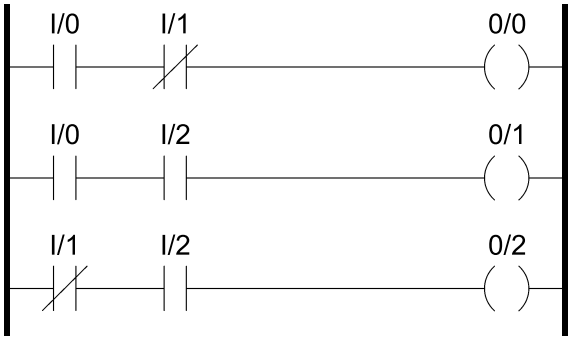


Figure 1.1 presents a ladder diagram example in which each line (also called rung) controls the states of three outputs named as O/0, O/1, and O/2, based on the states of three inputs designated as I/0, I/1, and I/2. Rungs in a ladder diagram may have normally open and normally closed contacts. In the given example, the state of O/0 in the first rung is set to "on" if input I/0 is "on" (normally open contact) and if input I/1 is "off" (normally closed contact). Similarly, in the second rung, the output O/1 is set to "on" state if both I/0 and I/2 inputs are in "on" state. Finally, in the third rung, if input I/1 is "off" and input

I/2 is "on" then the output O/2 is set to the "on" state. Also in the example ladder diagram presented in Figure 1.1, the three inputs interconnected in three independent logical rungs are used to control the three outputs, where the PLC scan time can be measured as the sum of the times required to read the three inputs, execute the three rungs, and update the three outputs.

Controversially, after more than half a century since their introduction, and despite their importance in the automation technology domain, somehow justified by protective PLC patents, research efforts to improve PLC performance are not a recurrent subject in the academic field. Recent and unavoidable trends like the Industry 4.0 and the Internet of Things era will require more computational power from automation controller devices, which will, unavoidably, cover the PLC design space. In this regard, for enhanced applications like motion control, model predictive controllers, cryptography, among others, where a more complex computation is required, the current PLCs' performance becomes an issue. This problem tends to drive solutions towards adaptations with the addition of external dedicated controlling/processing units, therefore pushing PLCs to a merely complementary function as commander units, instead of their original role as central processing units, ending up in an increase of the overall costs of the automation solutions as a consequence.

As a response to this demand, to improve PLCs performance by reducing the scan time, this dissertation uses two novel PLC architecture enhancements, namely Verify to Execute (VE) and Search to Execute (SE). The VE improvement was inspired in data flow machines (DENNIS, 1980) and circuit simulation theory (NAJM, 2010) and reduces the scan time duration by only executing the rungs' logic in which a variation occurs in its inputs, in a similar approach as Chimel and co-workers (CHMIEL; HRYNKIEWICZ, 2009) (CHMIEL et al., 2016a) (CHMIEL; MOCHA; LECH, 2018). Along with VE, the SE improvement searches for a cached memoized result before executing a rung with a given inputs' mask, shortening scan time in case of a cache memory hit by reusing the result and skipping execution, in a similar line of work as Suresh et al. (SURESH et al., 2015).

It is necessary to note that besides normally open/close contact inputs and regular output blocks, several other block types are present in the ladder diagram, as well as other PLC programming languages available and also many PLCs based on microprocessors or microcontrollers. However, these alternatives were not considered in the scope of this work, to the complexity in addressing all possibilities. Even within this limited scope, the

proposed improvements still have their importance in improving PLC performance based on scan time reductions.

Further to VE and SE architecture proposals, the present work evaluates the impact of PLC performance based on the scan time metric using different execution units of multi-cycle and pipeline types, along with multiple execution units quantities, to establish the boundaries of the impact that each of these characteristics may have on the overall PLC performance.

To evaluate the proposed improvements, a cycle-accurate simulator (CAS) was developed and used to simulate the PLC core architectures. Namely, the standard, VE, SE, and VE+SE cores were simulated with both multi-cycle and pipeline execution units along with several quantities of execution units, thus composing a detailed picture of the PLC architectural design space and performance improvement limits. Additionally, the silicon areas of all simulated architectures were calculated using the McPAT framework (LI et al., 2009) also to analyze the correlation of performance and silicon usage.

The experimental batch results show scan time reductions of up to 68% for cores with single execution units and up to 89% for cores with multiple execution units. Regarding the silicon area usage, in some cases, the proposed improvements produced an impressive 50% of reduction in scan time when compared to the standard PLC core, with a minor acceptable increase in the silicon area. Finnaly, the impressive results also justify the utilization and importance of the proposed improvements to enhance PLC performance, instead of a simple increase in the number of execution units or operation frequency of a standard PLC core.

1.1 Objectives

The main objective of this work is to propose improvements and enhancements in the PLC architecture to increase performance through reduction in the scan time duration. To reach this primary goal, an architecture was proposed in the context of this work, which was designed in a tailor-made cycle-accurate simulator to evaluate the architecture performance. Moreover, the specific objectives of this dissertation are:

- Propose and evaluate a novel PLC architecture improvements to achieve a performance boost
- Evaluate the impact that different execution units types and quantities have in the

PLC performance

- Analyze the relationship between performance and area by crossing the evaluation results of these two aspects

1.2 Structure of the text

The remaining of this text is organized as follows. Chapter 2 discusses relevant related works utilized during the development of this dissertation. Chapters 3 and 4 detail the improvements proposed and the architecture developed in the context of this work, respectively. In Chapter 5, the evaluation methodology and results are described, detailed, and discussed. Finally, Chapter 6 provides conclusions and future directions of this dissertation.

2 RELATED WORK

As a sad fact of history, even though programmable logic controllers are still massively used in industries as was when they were introduced, improvements in PLC performance is not a very explored scientific area despite the importance of PLCs in industry automation technology during the 20th and 21st centuries. Nevertheless, essential contributions in this area can be found in the technical literature, and the most relevant ones are compiled in this Chapter.

As relevant structural concepts concerning theory, usage, and historical facts, several researches (BOLTON, 2015) (WEBB; REIS, 1998) (LEWIS, 1998) were consulted as the literature bases of PLC devices. The material that composes those volumes constitutes the primary source of PLC information, which provides a substantial fundamental basis to the background required by this dissertation.

Similarly to one of the improvements presented in this current work, namely to avoid the execution of PLC's rungs whose inputs values do not have a variation, Chmiel et al. (CHMIEL; HRYNKIEWICZ, 2009) (CHMIEL et al., 2016a) (CHMIEL; MOCHA; LECH, 2018) developed related proposals.

In their first proposal (CHMIEL; HRYNKIEWICZ, 2009), the edges of the inputs are verified using the PLC's logic itself without any hardware development. In more recent efforts (CHMIEL et al., 2016a) (CHMIEL; MOCHA; LECH, 2018), a dual-processor 1bit/32bit PLC architecture was implemented by means of an FPGA where the 1bit part checks the inputs' edges for variations detections while the 32bit part executes the rungs' logic appropriately, similarly to a standard PLC.

Regarding the evaluation presented in their work (CHMIEL; HRYNKIEWICZ, 2009), it uses a simple example composing a restricted design space in a standard PLC to prove the benefits of the technique that is also explored in this study. Other works from the same group (CHMIEL et al., 2016a) (CHMIEL; MOCHA; LECH, 2018) use incomplete and somehow inaccurate commercial PLCs' information to compare to the proposed design at the instruction level, which is not appropriate since the architectures are different both in datapath and operation frequency.

Despite adopting improvements that are similar to the ones proposed by Chmiel et al., the present work also aims at other execution improvements, introducing different types and quantities of execution units to evaluate the performance of the PLC's enhancement in different ways. As already mentioned, the Chmiel and coworkers' related studies

use data from commercial PLCs as a basis for a performance comparison with their architecture, which is not the most appropriate approach due to the heterogeneity of the architectures. Therefore, in this work, the improved cores are compared to a standard PLC architecture defined following the literature patterns (BOLTON, 2015) (WEBB; REIS, 1998) (LEWIS, 1998) as a fair and unbiased comparison baseline.

Notwithstanding, as the proposal of data flow machines (DENNIS, 1980) (PELL et al., 2013) as well as one of the principles of circuit simulation theory (NAJM, 2010) (ZHUANG et al., 2016), the idea of avoiding the execution of unnecessary logic is not new. Hence, using this paradigm to avoid rungs' execution in the PLC design space is a novel and promising approach which was developed and evaluated by this research.

Another improvement proposed by this dissertation is to avoid executing rungs by storing and reusing previous execution results, a similar procedure to the one Suresh et al. (SURESH et al., 2015) named memoization in their proposal, which was used to improve the execution of transcendental functions like sine, cosine, tangent, etc.

The study of Suresh et al. obtained significant reductions in execution time by benefiting from the static behavior of the transcendental functions. Likewise, the present work uses a similar memoization procedure to store rungs' results and reuse them to avoid the unnecessary re-execution of rungs which aided in boosting the PLCs performance during evaluations.

In the same line of research with this dissertation, several PLC architectures and frameworks were proposed based on FPGAs (CHMIEL et al., 2016b) (HAJDUK; TRYBUS; SADOLEWSKI, 2015) (MILIK, 2016) (DU; XU; YAMAZAKI, 2010).

The proposals presented by Chmiel et al. (CHMIEL et al., 2016b) and Hajduk et al. (HAJDUK; TRYBUS; SADOLEWSKI, 2015) establish novel FPGA architectures to boost PLC's performance by improving the hardware co-design, which is a different approach from this work. The main focus of this research is in skipping useless executions by adopting improvements in PLC cores and evaluating the performance of different types and quantities of execution units.

Moreover, the current work also approaches the PLCs' performance enhancement problem by a different perspective from the ones presented by Milik (MILIK, 2016) and Du et al. (DU; XU; YAMAZAKI, 2010), which directly convert PLC programs to a hardware description language (HDL). Therefore, the results of those works are also not comparable to this research.

Nevertheless, despite the impossibility of direct comparison of the above works

(CHMIEL et al., 2016b) (HAJDUK; TRYBUS; SADOLEWSKI, 2015) (MILIK, 2016) (DU; XU; YAMAZAKI, 2010) with the present dissertation, all those proposals are related to the current project in terms of limitations and challenges of designing an original PLC architecture, thus providing relevant subsidies to the architectures proposed here.

As this work proposes a novel PLC architecture using multi-cycle and pipeline execution units and takes advantage of the multiple execution units paradigm, which are classical concepts, relevant references for this study areas may be found in several classic books and papers from the scientific community (PATTERSON; HENNESSY, 2013) (BLAKE; DRESLINSKI; MUDGE, 2009) (PALACHARLA; JOUPPI; SMITH, 1997) (OLUKOTUN et al., 1996). Those references compose relevant sources of information that have helped the development of these execution units types in the multiple arrays required by the context of this dissertation.

The work by Ikbal et al. (IQBAL; KHAN; KHAN, 2013) presents an innovative evaluation method for PLCs. Despite the formal definition of a methodology to benchmark PLCs, it is a contribution that influenced the evaluation methodology of the current work. Unfortunately, the Ikbal proposal does not consider the usage of third-party benchmarks, like the proposal of the PLCOpen committee (PLCOPEN, 2019), which may provide a neutral evaluation method for PLCs.

Differently from standard computer architectures, for which there are extensive and standardized benchmark sets, there is a lack of reliable third-party benchmark definitions for PLCs, except for a unique proposal of the PLCopen committee (PLCOPEN, 2019). Unfortunately, the TC3 certification benchmark (WAL, 2009) initiative of the PLCopen committee, initially proposed in 2006, which was a step towards a standard and neutral manner to evaluate PLC's performance instead of an ad-hoc set of test cases, was recently removed from the PLCopen official reference site for unknown reasons. This fact also highlights how PLCs' performance evaluation is neglected by the scientific research community, despite their importance in several automation applications, which the Industry 4.0 revolution and the Internet of Things era will demand.

Also, due to the importance of PLC architecture's design space exploration and despite the lack of benchmarks for PLCs' evaluation, several application proposals show how significant contributions in PLC performance improvement, like the one here presented, are essential for the automation technology field. Process control applications like proportional integral derivative (PID) controllers (MILIK, 2016), model predictive controllers (MOKHTARNAME et al., 2015) as well as SIMON and SPECK cryptography

algorithms (DUKA; GENGE, 2017) still require optimization or extra hardware adaptations like detailed in these proposals to be appropriately executed along with commercial PLCs due to the lack of performance of the currently available devices.

Lastly, Table 2.1 summarizes the contributions of the related work and in which way the present work is compared to them. The table shows how contributions from related works address four different themes: PLC architecture, PLC benchmarks, VE improvements, and SE improvements. As shown in Table 2.1, the present work is the only one in which all themes are covered.

Table 2.1: Related works summary

	PLC Architecture	**VE Improvement**	**SE Improvement**	**PLC Benchmark**
CHMIEL et al., 2016a	x	x		
CHMIEL et al., 2016b	x			
CHMIEL; HRYNKIEWICZ, 2009		x		
CHMIEL; MOCHA; LECH, 2018	x	x		
DENNIS, 1980		x		
DU; XU; YAMAZAKI, 2010	x			
HAJDUK; TRYBUS; SADOLEWSKI, 2015	x			
IQBAL; KHAN; KHAN, 2013				x
MILIK, 2016	x			
NAJM, 2010		x		
PELL et al., 2013		x		
PLCOPEN, 2019				x
SURESH et al., 2015			x	
WAL, 2009				x
ZHUANG et al., 2016		x		
PRESENT WORK	x	x	x	x

3 PROPOSED IMPROVEMENTS

To reach the goal of PLC scan time reduction, this work presents two new architectural improvements, namely the Verify to Execute (VE) and Search to Execute (SE), which are detailed in this chapter. Aside from these improvements, the performance was also evaluated using multi-cycle and pipeline execution units types, organized in single and multiple execution units arrays, which are described along with other architectural details in Chapter 4.

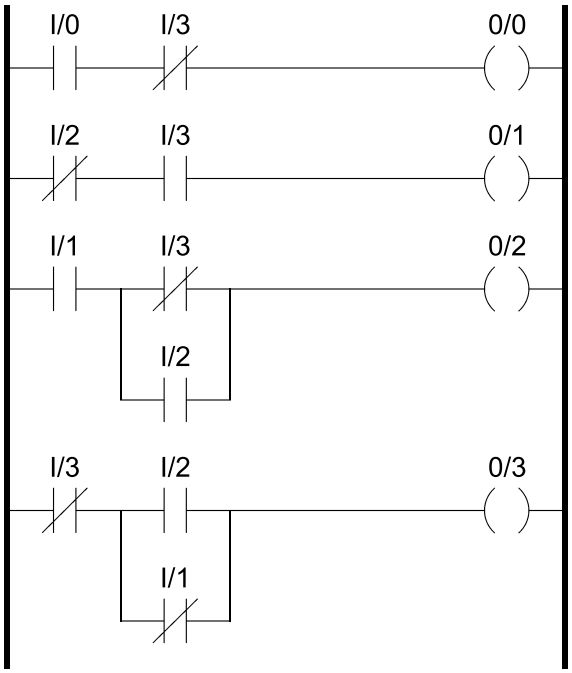
The Verify to Execute improvement verifies which inputs have value transitions and only schedules for execution the rungs which had a change in its inputs from on to off or vice-versa. The VE improvement is mainly inspired by data flow machines (DENNIS, 1980) and circuit simulation theory (NAJM, 2010). However, the introduction of this approach is completely innovative in the PLC domain.

For instance, by executing the ladder diagram program presented in Figure 1.1 with the VE improvement, an edge detected at input I/0 would schedule the first two rungs for execution. Similarly, a variation in input I/1 would program the first and third rungs for execution, while the last two rungs would be scheduled for execution if input I/2 has an edge transition.

Differently from the ladder diagram in Figure 1.1, where the inputs' verification order does not matter due to the symmetry of the diagram, the ladder diagram of Figure 3.1 may delay the first rung execution depending on the order of verification of inputs' edges. As observed in Figure 3.1, a border in input I/3 would cause the execution of all four rungs and a variation in input I/0 would schedule only the first rung for execution. Therefore, a suitable verification order is essential to start the execution of rungs as soon as possible.

The first problem in adopting the VE improvement is its dependency from the ladder diagram format. For symmetric diagrams like the one presented in Figure 1.1, the verification order of the inputs is not so relevant since all rungs have the same logic size, hence the only factor that may delay execution for a specific verification order is the inputs' edge probability, which cannot be measured in pre-execution time. Conversely, in an asymmetric ladder diagram, like the one presented in Figure 3.1, some possible verification orders may certainly delay the first rung execution, once the diagram contains rungs with several inputs and inputs that belong to several rungs. Thus, as already mentioned, a smart approach to select the best input verification order is advised.

Figure 3.1: An asymmetric ladder diagram



In the context of the VE improvement, the time to detect the need and dispatch the first rung to execution is here defined as execution trigger. Hence, the more time the VE improvement takes to detect need and execute the first rung, the worse is the execution trigger. Thus, to address the possible delay in the execution trigger detection, an algorithm based on *impact* and *probability* criteria were adopted for defining the verification order of the inputs. The *impact* criterion of an input is the ratio between the number of occurrences of this input in all rungs and the total quantity of rungs, while the *probability* criterion of an input is the ratio between the number of appearances of this input in all rungs and the total number of all inputs appearances in all rungs. Using these criteria, the verification order of the inputs of a ladder diagram is defined as the decreasing order of the sum of the values of *impact* and *probability* criteria for each diagram's input.

As an example, the verification order criteria for the diagram presented in Figure 3.1 can be observed in Table 3.1. Analyzing the data presented in the table, input I/3 has impact 1.0 (4/4), since it appears in all rungs, while input I/0 has impact 0.25 (1/4) since it is present in a single rung. In turn, input I/3 has probability 0.4 (4/10), since it has four occurrences over the total of ten input occurrences in the whole diagram and the input I/0 has probability 0.1 (1/10), since it has only one appearance over the same ten input occurrences. As a consequence, the verification order for the inputs of the ladder diagram in Figure 3.1, according to proposed verification order algorithm, is I/3 (sum is 1.4), I/2 (sum is 1.05), I/1 (sum is 0.7), and I/0 (sum is 0.35).

Table 3.1: VE verification order for ladder diagram of Figure 3.1

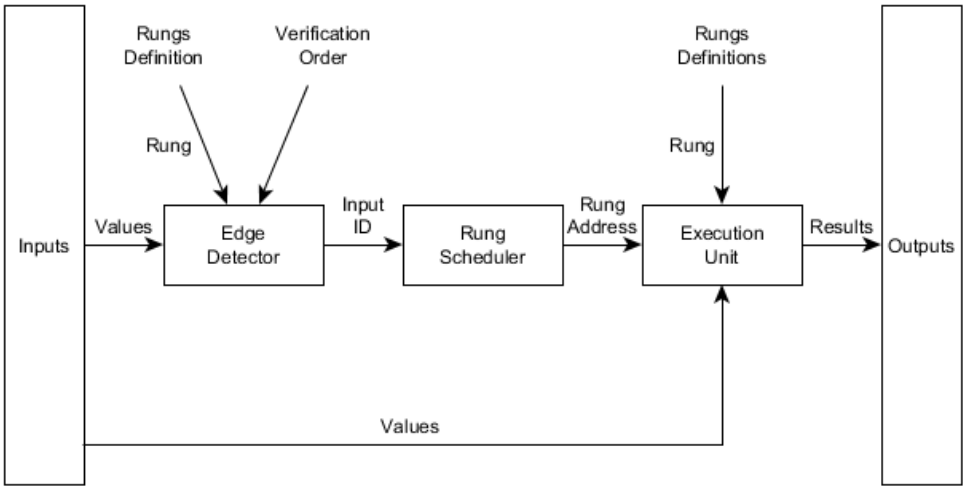
Input	Impact	Probability	Total
I/0	1/4	1/10	0.35
I/1	2/4	2/10	0.70
I/2	3/4	3/10	1.05
I/3	4/4	4/10	1.40

It is essential to clarify that besides the proposed heuristic based on these criteria of impact and probability, several other heuristics could have been proposed to determine the verification order of the VE improvement. Nevertheless, looking for an optimal procedure to determine the best verification order is out of the scope of this work, and so the proposed criteria were adopted as an acceptable solution to reach the main objective, which is to evaluate the proposed VE improvement.

Moreover, also in the example presented in Figure 3.1, once an edge occurs in I/3 in a given scan cycle, all rungs of the ladder diagram will be scheduled for execution following this detection. As a consequence, the verification of edges in the other three inputs is no longer necessary for that particular scan cycle. Applying this procedure not only shortens the delay in executing the first rung (execution trigger) but also decreases the execution time of the edges' verification procedure. The reduction happens because once all rungs of a particular input are scheduled for execution, the edge verification for that input is not necessary anymore.

The block diagram of the VE improved execution core is shown in Figure 3.2, where it is shown that it contains a sequence of three components. The first component of the chain is the edge detector, which uses the inputs' verification order coming from the

Figure 3.2: VE improved execution core



defined pre-execution algorithm and the rung definitions to check inputs edges. Once the edge detector identifies an input value edge, it dispatches this input's identification to the rung scheduler, which acts as a FIFO but also retrieves the rung address corresponding to the input identification and sends it to the execution unit. As the last element in the chain, the execution unit executes the received rungs' addresses in a similar form as a standard execution core. It is imperative to note that the modules named Execution Unit in all architecture figures may be of the multi-cycle or pipeline type and may come in an array with multiple units. More details regarding the VE improvement architecture, along with other ones, are further discussed in Chapter 4.

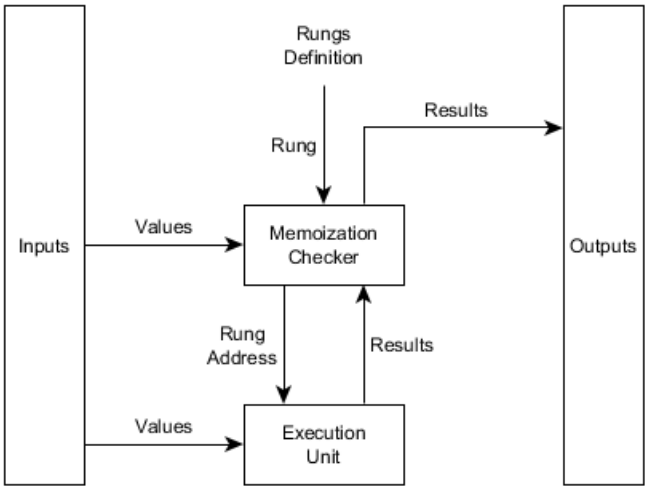
As the performance speedup introduced by the VE improvement results from avoiding the execution of rungs that do not have value transitions in their inputs, its usage is recommended for monotonous systems in which input edges occur only between long periods of idle time. The VE improvement application in systems that execute monotonously may also provide a more prompt response in time-critical applications, like inertial motion detection control, for instance.

From a negative perspective, the VE improvement may result in an increase of the scan time in systems with a large number of occurrences of input variations in the worst case of the verification order, due to the overhead of the verification of the inputs' edges. This drawback is further discussed in Chapter 5, in which it is demonstrated that the VE improvement performance boost compensates this disadvantage in some cases. Moreover, because of the algorithm that defines the verification order, the VE rungs' execution order may differ from a standard core. As a consequence, the VE improved core may present an unexpected execution at not well-implemented ladder diagrams, where the logic of different rungs overlaps each other.

The second architectural improvement adopted in this work is the Search to Execute, named so since it searches for previously-stored rungs' execution results before any execution. Hence, it dispatches for execution only the rungs whose previous execution results for the same input values are not cached. Otherwise, the stored rung result value is used, and the rung execution is entirely skipped. The approach explored by the SE improvement is similar to the memoization technique of Suresh et al. (SURESH et al., 2015). Nevertheless, using this technique in the PLC design space is a totally innovative and novel approach.

As presented in Figure 3.3, the memoization checker is the central component of the SE improved execution core block diagram. Once running, the memoization checker

Figure 3.3: SE improved execution core



verifies current input values and rung definitions for a match in its indexed memoization memory. In the case of a hit, the cached value is used, and the rung execution is disregarded. Otherwise, in the case of a miss, the rung address is dispatched to the execution unit. Similarly to the VE core, the SE core execution unit is equivalent to a standard PLC core, with the exception that the execution results are written back to the memoization checker to keep it updated for subsequent cached results verifications.

Due to its versatility, the block of components that compose the SE improvement can be attached before any PLCs' execution unit, thus eliminating the need for re-execution of cached execution results. However, in a situation in which the SE improvement's memoization memory is not large enough to store all possible results in a system with a large number of edges occurrences in inputs, and therefore a high number of different results, the search procedure plus the execution of rungs that repeatedly do not have the results cached may increase the scan time duration.

Regardless of this potential drawback, the focus here is to analyze the impact of the SE improvement along with other architectural characteristics. Therefore, the memoization memory size impact was not considered in the context of this dissertation and should be investigated as future work due to the complexity and importance of this subject. Nonetheless, indeed, the memoization memory size is a significant factor in the SE improvement performance. However, since the evaluation process in this work considers all probabilities of memoization hits and misses to obtain an overall performance metric, the size of the memoization memory becomes somehow a secondary matter. Nonetheless, respecting its importance, this subject is discussed again in more depth in Chapter 5.

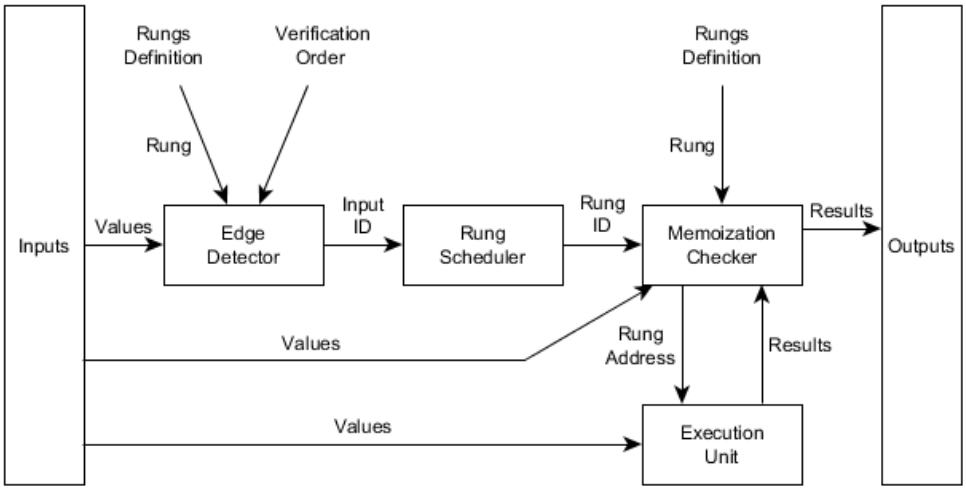
Notwithstanding, when the memoization memory size fits the requirements of the

system, the SE improvement provides superior performance for rungs with recurrent combinations of input values, even in applications with a high incidence of input edges. As also further discussed in Chapter 5, it is demonstrated that the significant SE speedup factor somehow compensates its drawback of higher usage of silicon area.

Lastly, Figure 3.4 details the block diagram of the VE+SE execution core, which includes both the VE and SE improvements. The VE+SE improved execution core reuses the same components of the standalone versions of VE and SE cores but in a different organization. Hence, the only noticeable difference between the VE+SE improved core and the standalone VE and SE cores is the SE memoization checker inserted between the rung scheduler and the execution unit. With this minor modification in the datapath, the rung scheduler delivers a rung identification to the memoization checker instead of the rung address.

In the best case, the execution time of a standard architecture executing the ladder diagram presented in Figure 3.1 will be the sum of the execution times for all the four rungs. For a design with the VE improvement, the best-case occurs when there are no edges, where the system only spends the verification time for all the inputs and does not need to execute any rung. The best scenario for the SE improved architecture occurs when the execution contexts of the four rungs are already cached at the memoization memory, where the system spends only the search and result reuse time for all four rungs stored in the memoization memory. Finally, in an implementation with both improvements VE and SE, the shortest execution time is equal to the design with the VE improvement alone, which consists of the verification time for all the four inputs when there are no input edges occurrences.

Figure 3.4: VE+SE improved execution core



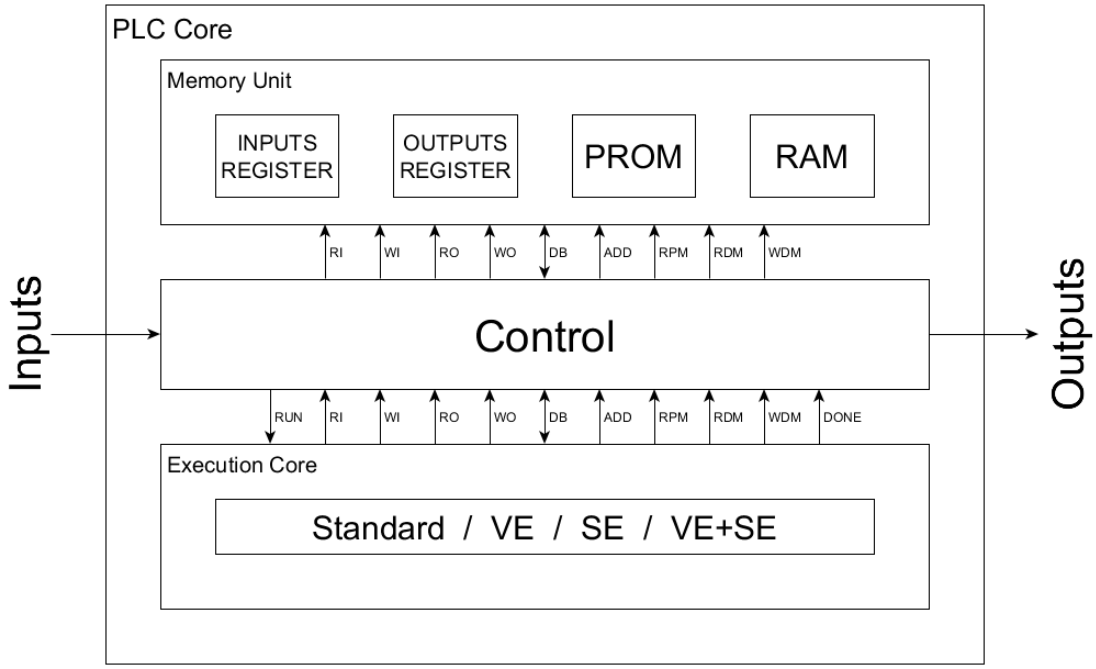
As well for the ladder diagram presented in Figure 3.1, the worst-case for a standard architecture is the same as the best case: the sum of the execution times for all the four rungs. For a design with the VE improvement, the worst scenario occurs when there are edges at I/2 and I/0, when the system needs to spend the verification time for I/3, I/2, and I/0, plus the execution time for all the four rungs. In an architecture with the SE improvement, the most extended execution time occurs when there is no memoization stored for any of the rungs, and the system needs to spend the search time for all the four rungs plus the execution time also for all the four rungs. At last, in architecture with both VE and SE improvements, the worst-case scenario occurs when there is a variation at inputs I/2 and I/0 with no stored memoization. This situation composes the worst-case execution time, with the verification time for I/3, I/2, and I/0 plus the memoization search time for all the four rungs and the execution time for all the four rungs.

Ultimately, since the proposed improvements do not modify the execution unit of the standard architecture, to mitigate the overhead of the worst cases discussed above, the edges' verification and the memoization search operations must occur in parallel by separate dedicated hardware. This design option is explained along with architecture details in the next Chapter 4.

4 ARCHITECTURE DETAILS

To explore the proposed improvements detailed in Chapter 3, a novel PLC architecture was developed to evaluate the performance of the proposed enhancements. Since the developed PLC architecture has several organizations to address the various combinations of Standard, VE and/or SE improvements, along with different types and quantities of execution units, the top-level architecture was planned to be the same independently of the kind of the execution core, as presented in Figure 4.1. This design option not only simplifies the experimental evaluation since the whole batch of tests is entered the same form but also ensures that all organizations are tested in the same external components environment.

Figure 4.1: Proposed PLC architecture

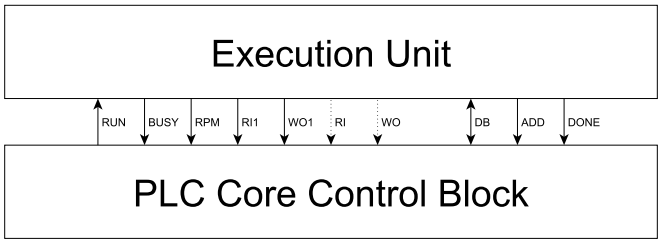


The main component of the architecture is a control block that uses several flag signals and buses to coordinate a memory unit and an execution core. The memory unit contains all data related components: the registers to store input and output states, a PROM for the rungs' logic data, and a RAM for general data storage. As can be observed in Figure 4.1, the architecture's execution core may be implemented as a standard core, a VE improved core, a SE improved core, or a VE+SE core. This characteristic allows four different execution cores implementations to be compared in similar conditions since scan cycle phases for reading the inputs and writing the outputs are executed the same way independently of the execution core.

4.1 Standard Execution Core Architecture

The standard execution core, detailed in Figure 4.2, behaves like a traditional PLC execution core. The read input values are used to execute all the rungs and update the outputs according to the results generated by the execution unit. It is composed of an execution unit, which, as already explained, might be of the multi-cycle or pipeline type and come in an array with multiple units, that directly interfaces with PLC core control block signals. A remarkable difference compared to other types of execution cores is that the standard execution core does not use the *RI* (read inputs) and *WO* (write outputs) signals (which are represented as dashed lines in Figure 4.2) since it is not an improved organization and then its execution unit merely executes all rungs along.

Figure 4.2: Standard execution core architecture

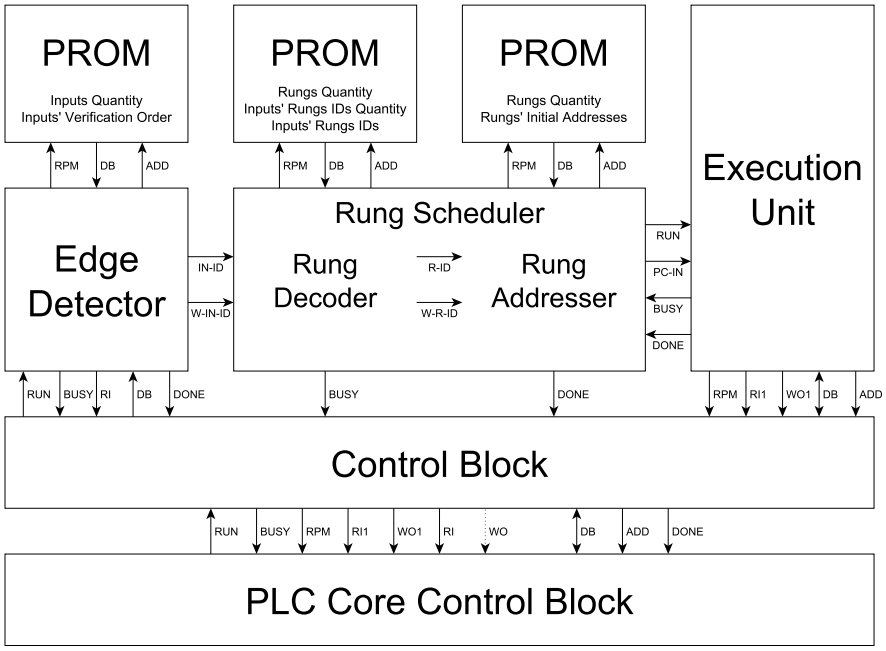


4.2 VE Execution Core Architecture

Following its definition in Figure 3.2, the VE execution core implements three execution phases, each one executed by one of its main components, respectively the edge detector, the rung scheduler, and the execution unit, each of these requiring some cycles to dispatch the execution to next element in the execution chain. The most noticeable characteristic of the VE execution core shown in Figure 4.3 is that it contains three dedicated PROM components, each one storing part of the VE execution core's additional program data. This design option was necessary to achieve the required parallel speedup factor required by an improved execution core by avoiding shared PROM memories buses.

In the first cycle, once it receives the *RUN* command, the VE execution core's edge detector loads the value of the inputs using the *RI* (read inputs) and *DB* (data bus) interfaces. In the second cycle, a variation in all the input values is checked, and if no global edge is detected, any further execution is skipped with the edge detector *DONE* signal is set to high. Otherwise, during the third cycle, the first of the inputs in the execution

Figure 4.3: VE execution core architecture



order priority list is loaded from the edge detector's PROM using its *RPM* (read program memory), *DB* (data bus) and *ADD* (address) signals. In the fourth cycle, in the case of current input's edge detection, the input is dispatched to the rung scheduler through the *IN-ID* (input identification) and *W-IN-ID* (write input identification) ports. Along with the fourth cycle, it is verified if all inputs of the verification list were checked. If so, the edge detector *DONE* signal is set to high. Otherwise, the third and fourth cycles are repeated until this condition is reached.

As observed in Figure 4.3, the VE execution core's rung scheduler contains two PROMs because it is internally divided into two other components. The first component is named rung decoder, and the second one is the rung addresser, each of these responsible, respectively, for decoding which rungs belongs to each input and for retrieving rungs' initial addresses.

Once the rung decoder receives an input identification through *IN-ID* and *W-IN-ID* signals, it uses the first cycle to retrieve from its PROM the number of rungs in which this input identification is present, using the *RPM*, *DB* and *ADD* ports. It then uses as many cycles as the number of rungs to dispatch the rungs to the rung addresser via the *R-ID* (rung identification) and *W-R-ID* (write rung identification) signals.

From its side, once the rung addresser receives a rung identification through the *R-ID* and *W-R-ID* signals, it requests the rung's initial address, which is read from its PROM using the *RPM*, *DB*, and *ADD* signals, to finally dispatch the retrieved address to

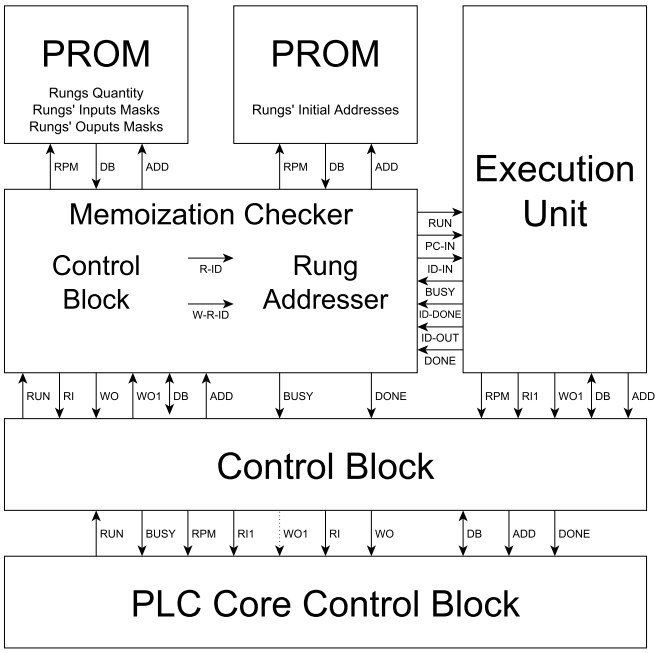
the execution unit using *RUN* and *PC-IN* (program counter input) interfaces.

At last, once the rung decoder, the rung addresser, and the execution unit of the VE execution core finish their executions, the VE core's control block outputs a value high at the *DONE* signal to notify to the PLC core control block that the execution phase has reached the end.

4.3 SE Execution Core Architecture

Following the SE core improved guidelines defined in Figure 3.3, the SE execution core main component is the memoization checker, which is composed of its internal control block and a rung addresser. Similarly to the rung scheduler in the VE execution core, these two components explain the two dedicated PROMs that are present in the SE execution core architecture shown in Figure 4.4.

Figure 4.4: SE execution core architecture



In the SE execution core's first cycle, once it receives the *RUN* command, the memoization checker control block reads the total quantity of rungs from its PROM, using the *RPM*, *DB*, and *ADD* ports. In the first run of the second cycle, all rungs are queued to verify if their execution is needed, and the first rung is set for verification in the memoization checker's memoization memory. In the third cycle, the memoization verification result is checked, and, in case of a memoization hit, the memoized value is updated in the outputs register, and the rung execution is discarded. Otherwise, the rung identification is

dispatched to the rung addresser via *R-ID* and *W-R-ID* ports. In any case, after cycle three is completed, cycles two and three are repeated until all queued rungs were checked.

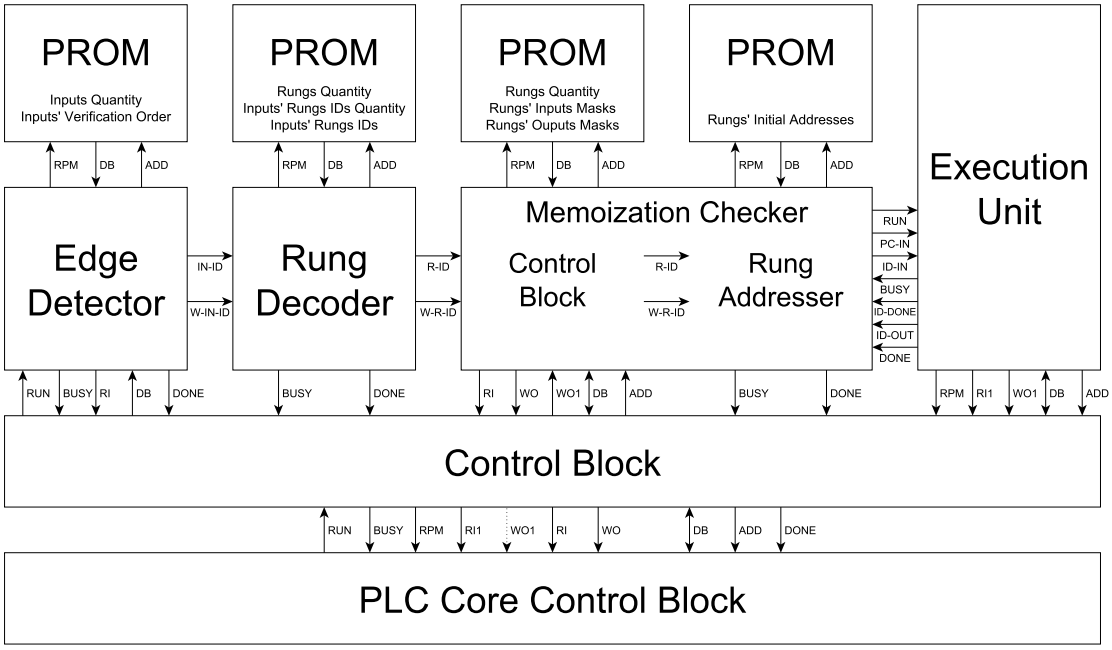
The behavior of the rung addresser present in the SE execution core is the same as explained for the VE execution core, with a minor difference. In the SE execution core, the rung identification is passed by the rung addresser to the execution unit using the *ID-IN* (identification input) signal along with the other *RUN* and *PC-IN* signals. Once the execution unit finishes the rung execution, the rung identification is notified back to the memoization checker control block through the *ID-DONE* (identification done) and *ID-OUT* (identification output) interfaces. This back and forth procedure with the rung identification is necessary to allow the memoization checker's control block to store the execution results in the appropriate memoization memory address.

Lastly, once the memoization checker, rung addresser, and execution unit finish their executions, the SE core's control block notifies the PLC core control block using the *DONE* flag.

4.4 VE+SE Execution Core Architecture

Finally, the VE+SE execution core presented in Figure 4.5 contains the same components and contains the same behavior as described for the VE and SE execution cores, but in a different organization with minor modifications in the datapath. The most notable

Figure 4.5: VE+SE execution core architecture



and crucial difference is that the memoization checker receives the rung identification from the rung decoder through the *R-ID* and *W-R-ID* signals. With this modification, not all rungs are queued for verification as in the SE execution core since the memoization checker's *RUN* signal does not receive any stimulus.

4.5 Execution Unit Architecture and ISA

Regarding the execution units' architecture, it is supported by a register bank, and a small control block is responsible for fetching the instructions from the program memory, decoding, and executing them according to the decoded value. For this purpose, a specific ISA (instruction set architecture) was developed for the execution unit to meet the requirements of the PLC programs.

As can be observed in Table 4.1, each instruction is characterized by a type and format. Instructions of register type (R) define operations in the register bank, a memory type instruction (M) executes a data transfer from/to the memory unit, while a control type instruction (C) characterizes an instruction that deviates the execution unit from its standard behavior. Subsequently, the instruction format defines how each instruction is organized. It is subdivided into an instruction code (represented by a mnemonic) and zero up to three arguments.

Table 4.1: Instructions of the developed ISA

Instruction	**Type**	**Format**			
And 1 bit	R	AND-1	RD	RS1	RS2
Or 1 bit	R	OR-1	RD	RS1	RS2
Not 1 bit	R	NOT-1	RD	RS	
Load single input	M	LI-1	RD	INPUT ID	
Store single output	M	SO-1	RS	OUTPUT ID	
Done	C	DONE			

The AND-1, OR-1, and NOT-1 register type instructions execute respectively a boolean AND, OR, and NOT operation at the register bank. Each of these instructions has a destination register (RD) and one or two source registers (RS). The memory type LI-1 instruction loads the value of an input of the inputs register located in memory with a given identification (INPUT ID) into a destination register (RD) while the SO-1 instruction stores the value of a source register (RS) to the outputs register located in memory with the given output identification (OUTPUT ID). The only control type instruction defined in the ISA is the DONE instruction which performs an essential role in the execution unit,

by notifying the execution core control block that the execution of a rung or the entire program has finished, depending on the execution core organization.

Still regard the DONE instruction, when a program is developed for a standard PLC architecture, this instruction must be added as the last instruction to notify the control block about the execution end. Therefore, to adapt the program to the proposed improved execution cores, a minor modification must be made, which consists of adding a DONE instruction at the end of each rung's instructions. This adjustment is required since, in the proposed cores, not all rungs will be executed in each scan cycle. In other words, the DONE instruction represents the end of the execution step of the scan cycle in the standard architecture and the end of a single rung execution for other execution cores.

4.6 Program Memories Structures

Besides adding additional DONE instructions, the proposed VE, SE, and VE+SE execution cores require extra data in the program memory to achieve a performance boost, differently from the program memory structure of the standard execution core, which needs the instructions only.

As can be observed in Figure 4.6, the VE execution core requires additional program memory fields: the quantity of inputs and rungs, which determine the addresses and size of the other fields; the preprocessed inputs' verification order in which the edge detector unit should check for edges; the identification of the rungs associated to each input, which are used by the rung scheduler to identify which rungs should be executed when the edge detector detected an input edge; the initial addresses of the program fragments corresponding to each rung, which are passed from the rung scheduler to the execution unit; and the program instructions themselves in the developed ISA.

As detailed in Figure 4.7, the program memory structure of the SE execution core contains the following fields: the quantity of rungs, which determines the addresses and size of the other fields; the rungs' inputs masks, to isolate inputs that should be checked by the memoization checker during the verification of each rung; the rungs' outputs masks, to modify only the outputs related to a specific rung; the initial addresses of the program fragments corresponding to each rung, which are passed from the memoization checker to the execution unit once it detects a memoization miss; and the program instructions themselves in the developed ISA.

Finally, the VE+SE improved core program memory structure is a merge of the

Figure 4.6: Program memory structure of the VE core

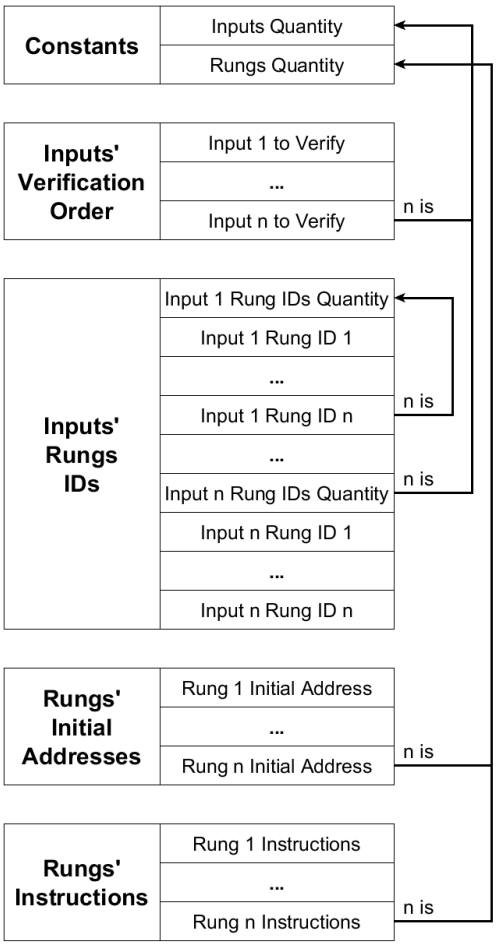
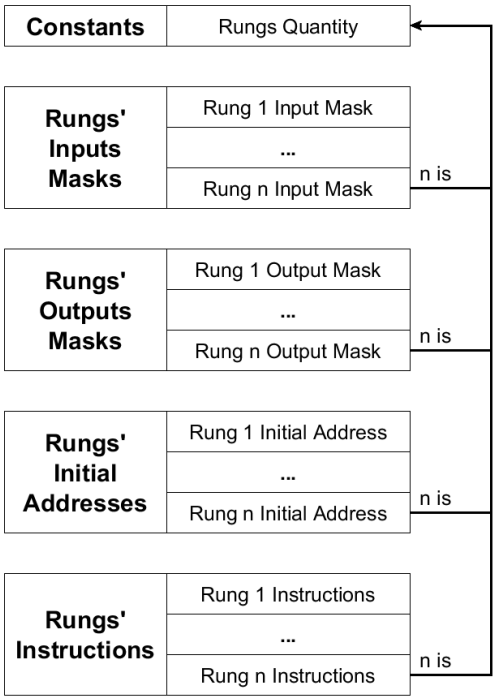


Figure 4.7: Program memory structure of the SE core



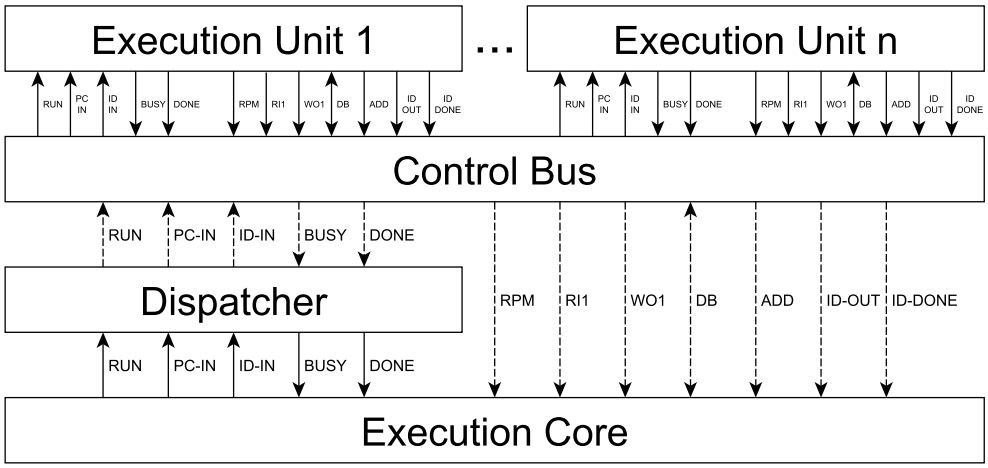
presented VE and SE structures, since this execution core shares the same units of the VE and SE improved execution cores in a different organization. The only difference is that the rung scheduler takes three cycles instead of four to dispatch the rung identification to the memoization checker, since, as may be observed in Figure 3.4, the rung scheduler does not need to retrieve the initial address as it does in VE execution core.

4.7 Multiple Execution Units Architecture

Besides the proposed improvements, another main goal of the present research is to analyze the impact of the type of the execution unit on the PLC architecture's performance, by using multi-cycle and pipeline execution units, in schemes with single and multiple execution units. To achieve this goal, the proposed architecture was developed in a way that the execution units contain the same basic interfaces observed in Figures 4.2, 4.3, 4.4 and 4.5, independently of the type and/or quantity of execution units used. This design choice for a single execution unit is straightforward, but for multiple execution units, it requires modifications in the execution unit's organization and signals.

In Figure 4.8, the architecture with multiple execution units is detailed, in which several execution units, which can be of multi-cycle or pipeline type, are represented along with the dispatcher. The dispatcher is responsible for selecting which of the available execution units will receive a rung to be executed.

Figure 4.8: Multiple execution units architecture



Regarding the multiple execution units interface, although the dispatcher guarantees the same execution unit's trigger interface with other components of the architecture, the multiple execution units require enlarging the buses for coupling them to the memory

unit, a detail that is marked as dashed lines in Figure 4.8. These modifications have a minor, but an essential impact in the coupling with other architecture components and, very importantly, with the requirements for the development of a multi-bus memory unit.

5 EXPERIMENTAL EVALUATION

For experimental evaluation of the present work, the primary choice to develop the proposed PLC architectures would be a real-world FPGA environment. However, this option is hard to deploy due not only to the development of specialized hardware but also due to the effort to build a peripheral rig necessary to produce several stimuli for the simulation batches. In the second place, an FPGA simulation would be an option if the available tools for academic propose could satisfactorily input data from an extensive simulation batch data into them. Finally, traditional widely known cycle-accurate simulators (CAS) and discrete event simulators (DES), like Gem5 (BINKERT et al., 2011) and SystemC (LIAO; TJIANG; GUPTA, 1997), respectively, would be excellent simulation options if those tools were not too sophisticated to map new architectures from scratch.

Since available simulation tools do not fully satisfy the simulation requirements for assessment of this study, a tailor-made CAS software was developed. The designed CAS was carefully adapted to all the complex requirements of a PLC architecture's simulation with its extensive parameters list:

- Inputs count
- Rungs count
- Instructions per rung
- Edges and memoization occurrences
- PLC core type (standard, VE, SE or VE+SE)
- Execution unit type (multi-cycle or pipeline)
- Number of execution units

To speed up the CAS development, the Java framework was selected due to its robustness, versatility, and all the advantages that an object-oriented language has in saving development time, benefiting from the use of hierarchical classes to model PLC architecture's components. In this regard, a complete set of class diagrams developed are demonstrated in Appendix A, which are related to the architecture specified in Chapter 4.

Furthermore, independently of the preference for the Java framework, the adoption of the CAS model, in which the behavior is accurate at cycle level, is adequate for a straightforward future work in porting of the architectural model to any current or future hardware description language (HDL) in a commercial FPGA or even to an application-specific integrated circuit (ASIC) environment. Therefore, this future language porting

would automatically inherit all the characteristics of the proposed improvements as modeled in the CAS.

Furthermore, one of the main reasons to select simulation as the method chosen for the validation procedure is that it can be easily modified to execute batches of experimental runs as well as generate data organized according to the needs of the validation process, as further discussed later on. Lastly, the CAS uses cycle counts as the metric for performance, without considering the frequency of maximum critical path in each stage of the pipeline. Despite the importance of this matter, this evaluation was considered out of the scope of this work, and so it must be addressed as a future work due to its complexity.

5.1 Single Execution Unit and Type Evaluation

There is a lack of official or widely known PLCs' benchmarks, except for the initiative TC3 of the PLCopen committee (WAL, 2009), which, unfortunately, was unavailable at the PLCopen official site (PLCOPEN, 2019) at the time this document was written. Therefore, as an initial evaluation procedure in the developed CAS, the results from the simulations of the VE, SE, and VE+SE execution cores with a single execution unit of the multi-cycle type were compared against each other using the standard execution core following the literature patterns (BOLTON, 2015) (WEBB; REIS, 1998) (LEWIS, 1998) as a fair and unbiased comparison baseline. The total duration of the execution cycle of a program was calculated as the arithmetic mean of the number of execution cycles obtained for all the possible combinations of input edges and/or rungs' memoization.

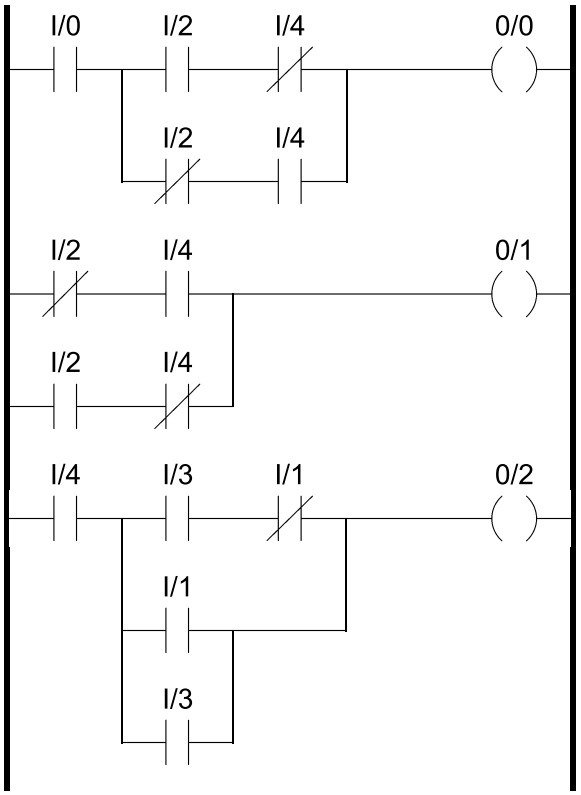
For instance, in example 1 presented in Figure 1.1, there are eight possibilities of edges' occurrences in its three inputs, so the duration of the execution cycle for the VE execution core is calculated as the arithmetic mean of the eight simulation results obtained for all the eight possible combinations of input edges. Similarly, the duration of the SE execution cycle is calculated as the arithmetic mean of the simulation results corresponding to the eight possibilities of hit or miss in the memoization memory for each of the three rungs. Finally, the duration of the execution cycle of the VE+SE execution core is calculated as the arithmetic mean of the simulation results obtained for the forty-five possible combinations considering both input edges and memoization occurrences possibilities.

Although this research utilizes the arithmetic mean as a standardization parameter, it is noteworthy that not all input edges and/or memoization occurrences have the same

probability of happening. This measurement choice was taken to get the overall performance of the results without introducing a bias that may arise by selecting a best or worst case scenario in the simulation executions.

As didactic ladder diagram test cases, along with the example 1 presented in Figure 1.1, which represents a balanced example with rungs of same sizes, and example 2 presented in Figure 3.1, which contains unbalanced rung sizes, a third test case, referenced as example 3 at Figure 5.1, containing rungs with larger sizes, was also used for the initial evaluation purposes.

Figure 5.1: A large rungs ladder diagram



The results regarding the arithmetic mean of the *execution cycles* and *speedup* for the selected didactic test cases are summarized in Tables 5.1 and 5.2. These results show that a substantial speedup is achieved even for example 1, which is the simpler test case diagram. These initial results also seem to show that the higher the rungs' complexity is, the larger the achieved speedup since the best results are achieved with example 3, which has the largest and more complex rungs. This trend is further evaluated in additional experiments, shown later in this chapter.

Tables 5.3, 5.4 and 5.5 present the results for the *speedup probability*, *average positive speedup* and *average negative speedup*, respectively. The *speedup probability* represents the percentage of all possible executions that have a positive speedup when com-

pared to the standard core execution. The *average positive/negative speedup* represents the arithmetic mean of the speedups of all executions that result in a positive/negative speedup when compared to the standard core execution. By analyzing the data in these tables, it is noteworthy that, except for VE, all proposed execution cores have a high probability of positive speedup, although, even for VE, that has the highest probability of negative speedup, values of those speedups are always low.

Table 5.1: Mean execution cycles of didactic test cases

	Example 1	**Example 2**	**Example 3**
Standard	69.00	114.00	109.00
VE	59.25	102.06	90.75
SE	40.50	65.50	60.50
VE+SE	47.53	74.36	64.85

Table 5.2: Mean speedup of didactic test cases

	Example 1	**Example 2**	**Example 3**
VE	1.16	1.12	1.20
SE	1.70	1.74	1.80
VE+SE	1.45	1.53	1.68

Table 5.3: Speedup probability of didactic test cases

	Example 1	**Example 2**	**Example 3**
VE	0.50	0.38	0.53
SE	0.88	0.94	0.88
VE+SE	0.91	0.95	0.91

Table 5.4: Average positive speedup of didactic test cases

	Example 1	**Example 2**	**Example 3**
VE	1.61	1.60	1.59
SE	1.92	1.84	2.05
VE+SE	1.57	1.60	1.84

Table 5.5: Average negative speedup of didactic test cases

	Example 1	**Example 2**	**Example 3**
VE	-0.09	-0.05	-0.06
SE	-0.04	-0.04	-0.03
VE+SE	-0.19	-0.14	-0.13

From the results from these three small didactic examples, it is possible to identify a trend that SE is the best execution core regarding speedup, while VE+SE is the best one for speedup probability. However, it is simplistic to draw firm inferences from

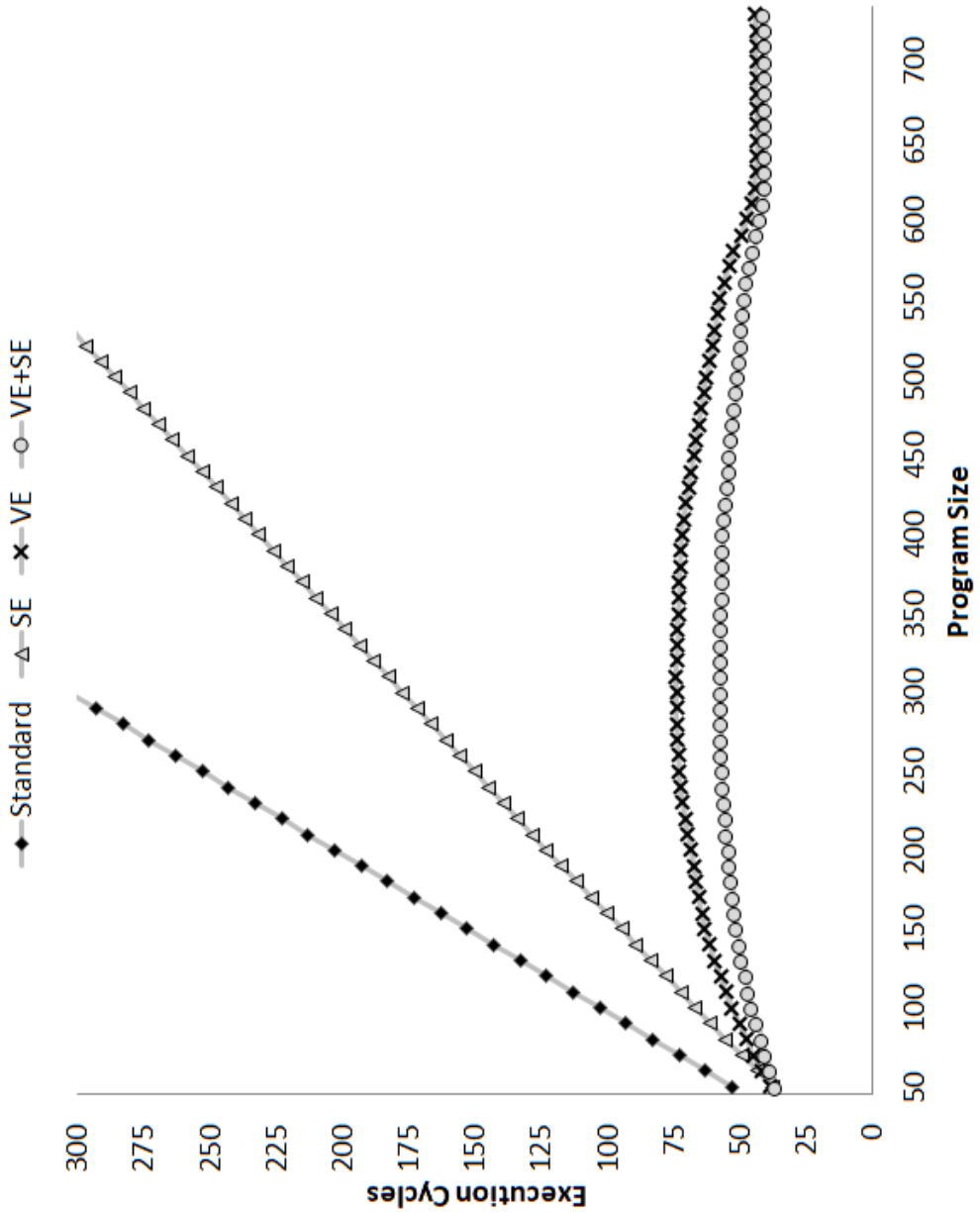
such a small design space. To bring more robust conclusions to the initial evaluation, this study evaluated the proposed execution cores in a larger simulation execution batch, which ranges from 1 up to 8 inputs, 5 up to 20 rungs, and 10 up to 200 cycles to execute each rung. The number of simulated programs was limited to 100,000 programs for each group of those criteria due to the impossibility of simulating, in reasonable computation time, all the 1.532E+74 possibilities from the combinations of numbers of inputs, rungs, rungs' execution cycles, and edges and/or memoization occurrences. Also, programs with less than 5 rungs were not included in the batch, since they have less than 100,000 possibilities of different programs and so could cause a bias in the results.

The batch execution results in the graph presented in Figure 5.2 shows the relationship between program size in cycles of the multi-cycle execution unit and effectively required execution cycles. The graph in Figure 5.2 also confirms the hypothesis proposed when analyzing the results from the didactic test cases: the higher the complexity of the rungs is (large program size in cycles), the more significant is the speedup. The graph also shows that the SE execution core has less speedup scalability than VE and VE+SE since it does not benefit from scenarios with no input edges. Also, despite the very close behavior of VE and VE+SE, it is possible to state that the memoization memory boosts performance in VE+SE by observing that it outperforms VE in programs with less than 600 cycles in program size.

In complement to Figure 5.2, by analyzing scan time reduction in Figure 5.3, it is noticeable the exponential speedup provided by the proposed execution cores. In a superior performance in this aspect, the VE and the VE+SE execution cores reach a top of 95% of reduction when compared to the standard execution core. Moreover, this graph also denotes that SE speedup is not equally scalable as the didactic test cases had indicated. Nevertheless, the SE execution core reaches a top scan time reduction of around 43%.

Program size versus performance (positive/negative speedup) graphs shown in Figures 5.4, 5.5 and 5.6 also provide evidence that the higher the complexity of the rungs is, the better is the achieved positive speedup. Although these graphs reinforce this finding, SE does not show to be the best execution core, differently from the three didactic test cases. SE shows, in fact, a plateau in the speedup around 175%. On the other hand, VE and VE+SE have an exponential speedup growth as a function of program size. At last, similarly to the trends previously identified, when there is a negative speedup, its value is low and tends to zero as program size increases in all proposed improved cores.

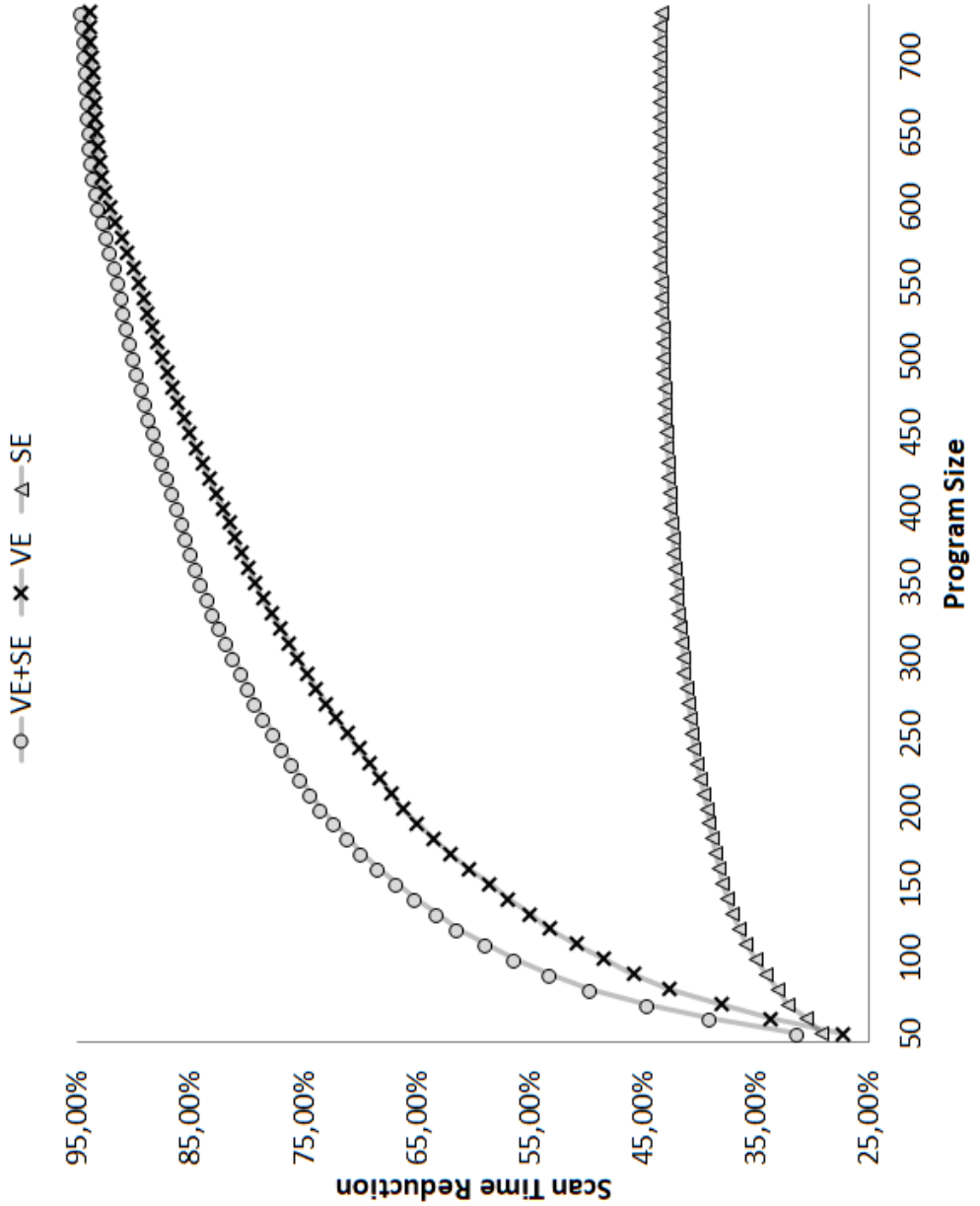
Figure 5.2: Program size vs. performance results



The graphs in Figures 5.7, 5.8 and 5.9 present the relationship between input count and performance. The results show that the performance of VE and VE+SE degrades as the input count grows. However, due to the memoization boost provided by the SE part of the VE+SE execution core, it decreases a bit slower than VE. The almost constant speedup of the SE execution core in Figure 5.8 is justified since this execution core is not affected by edge occurrences, and so, neither by the input count variation.

Figures 5.10, 5.11 and 5.12 show the graphs of the relationship between rung count and performance. The SE speedup value plateau, already observed in Figure 5.5, can also be observed in Figure 5.11. The zigzag pattern in VE and VE+SE results is justified due to the heterogeneity of the programs used in the batch simulation. The limitation of 100,000

Figure 5.3: Program size vs. scan time reduction results



programs causes a larger step in the program simulation count, since the larger the rung count is, the larger the number of different possible programs. A bigger simulation step makes different program forms be simulated for each rung count, thus causing the zigzag pattern. This conclusion was verified by simulating the same batch with a limitation of 10,000 programs, which resulted in a more pronounced zigzag with the same shape of the tendency lines also observed in the presented graphs. Therefore, it is reasonable to conclude that a larger number of tests will reduce the zigzag pattern following the graphs' tendency lines.

Figure 5.4: Program size vs. performance VE results

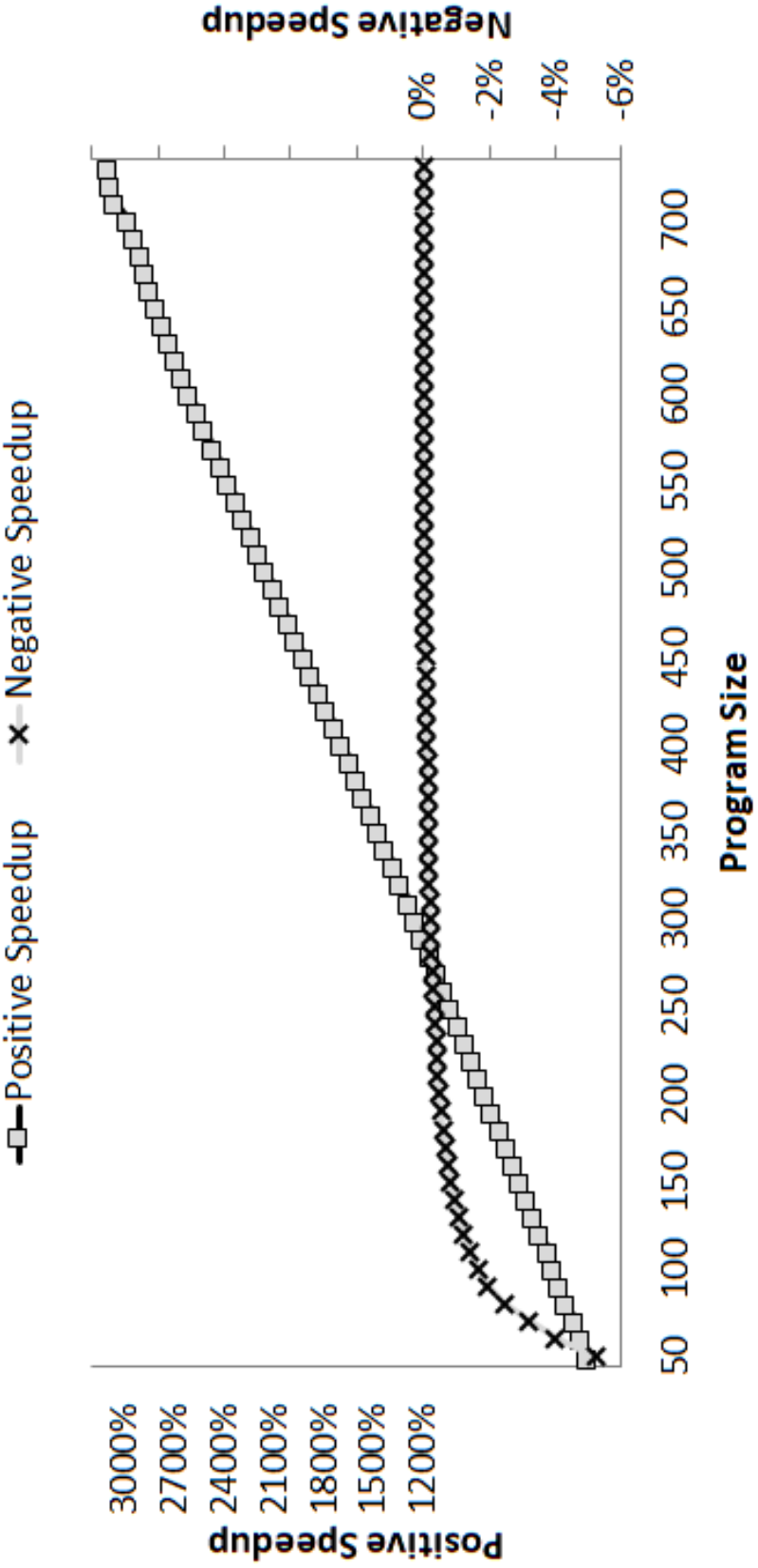


Figure 5.5: Program size vs. performance SE results

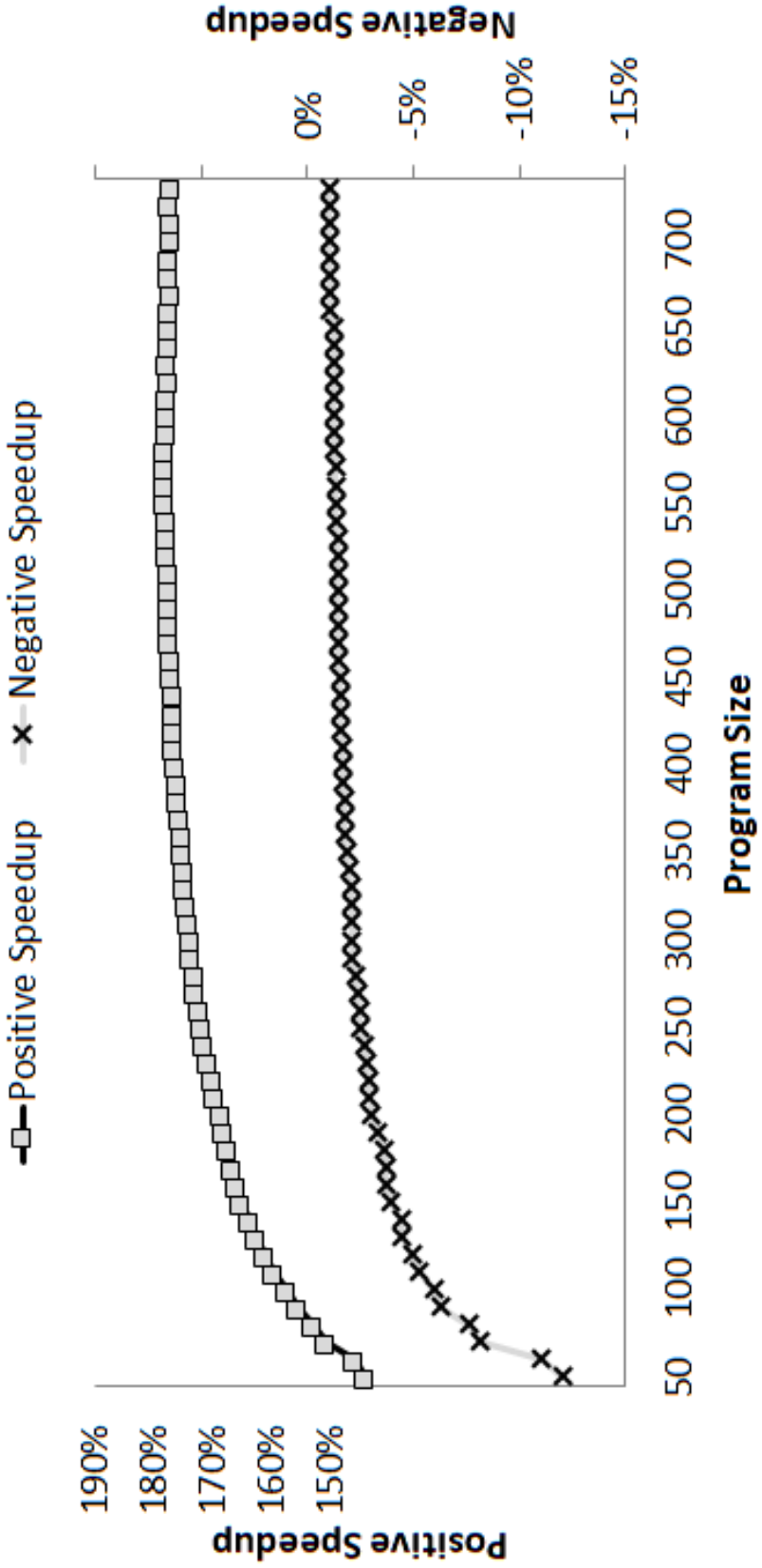


Figure 5.6: Program size vs. performance VE+SE results

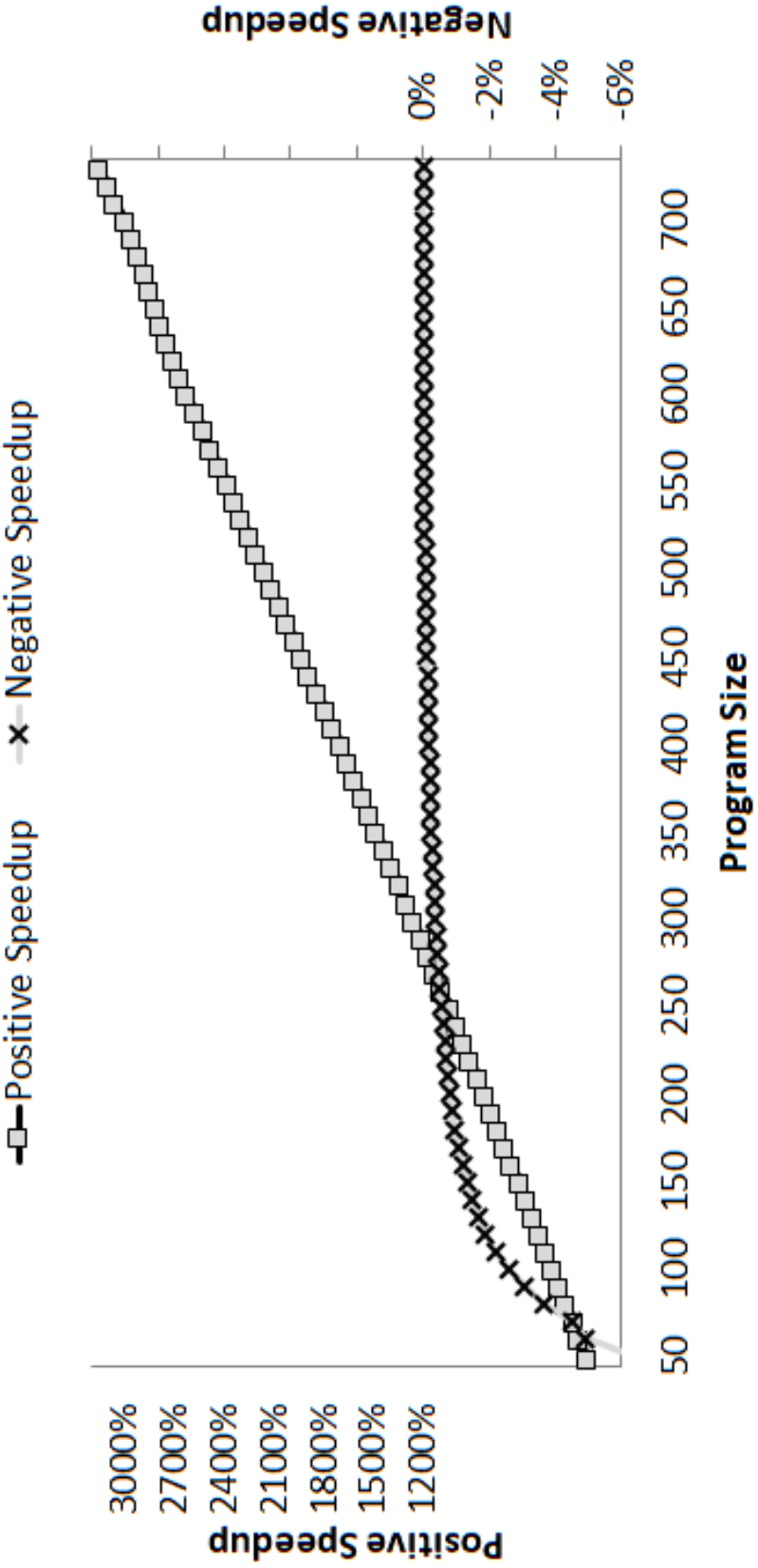


Figure 5.7: Input count vs. performance VE results

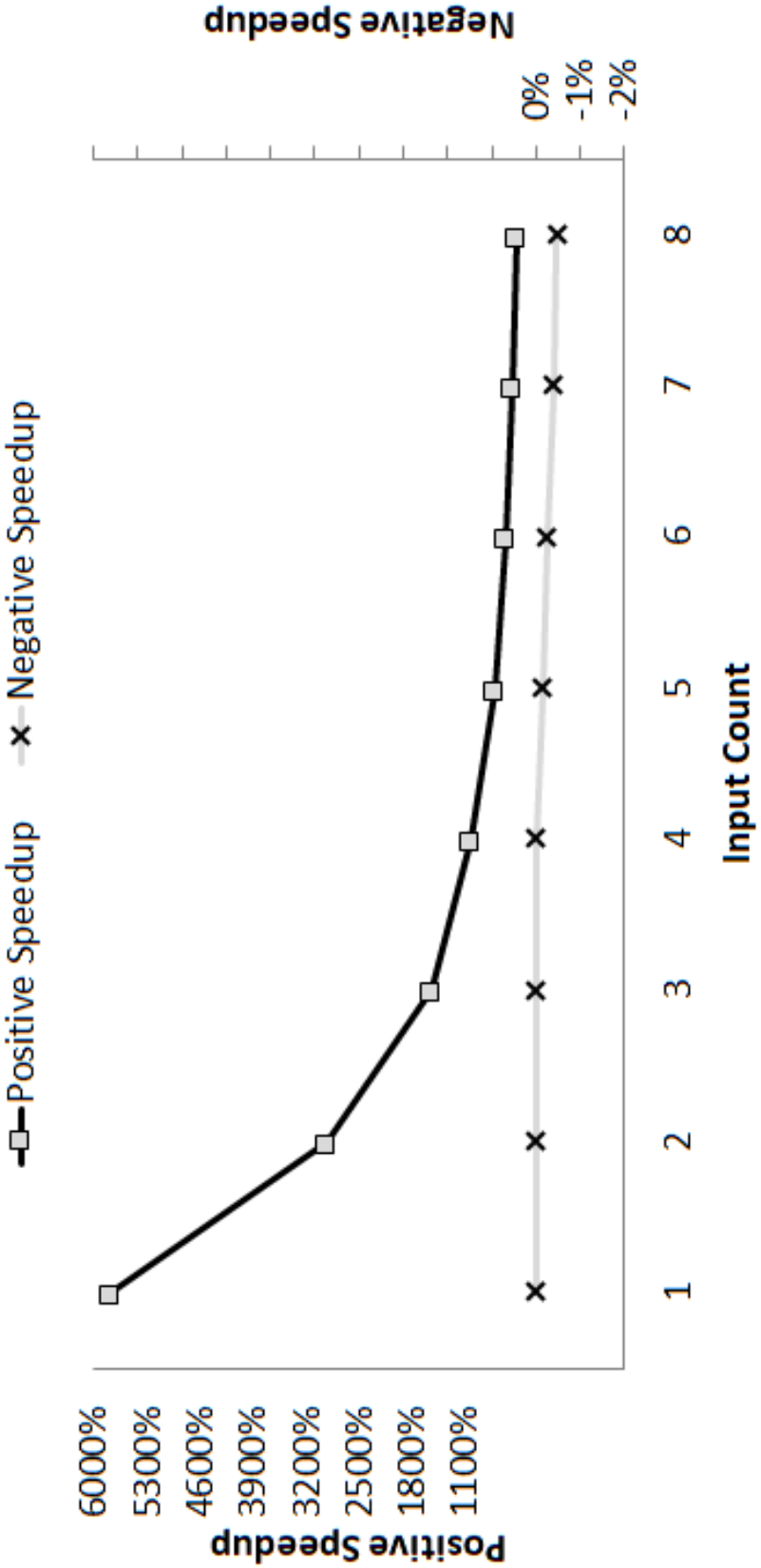


Figure 5.8: Input count vs. performance SE results

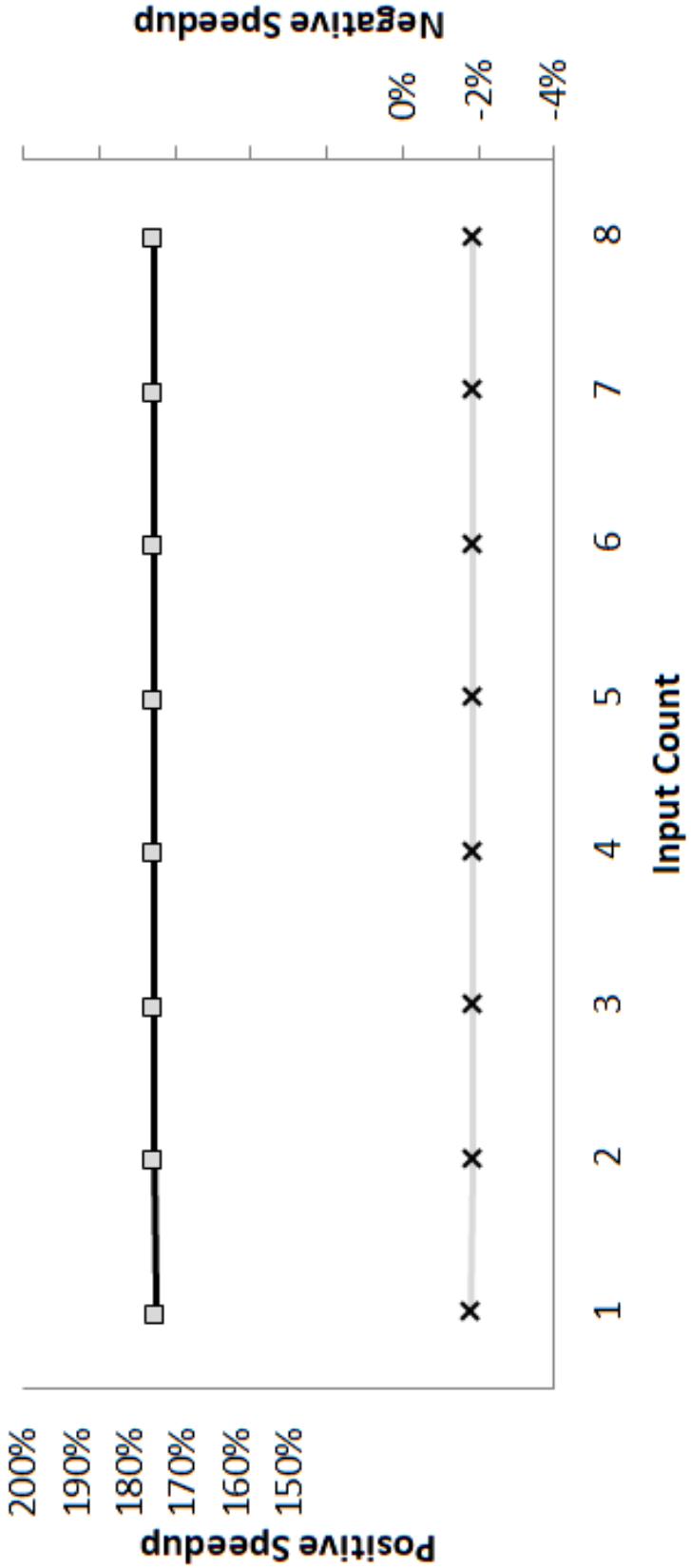


Figure 5.9: Input count vs. performance VE+SE results

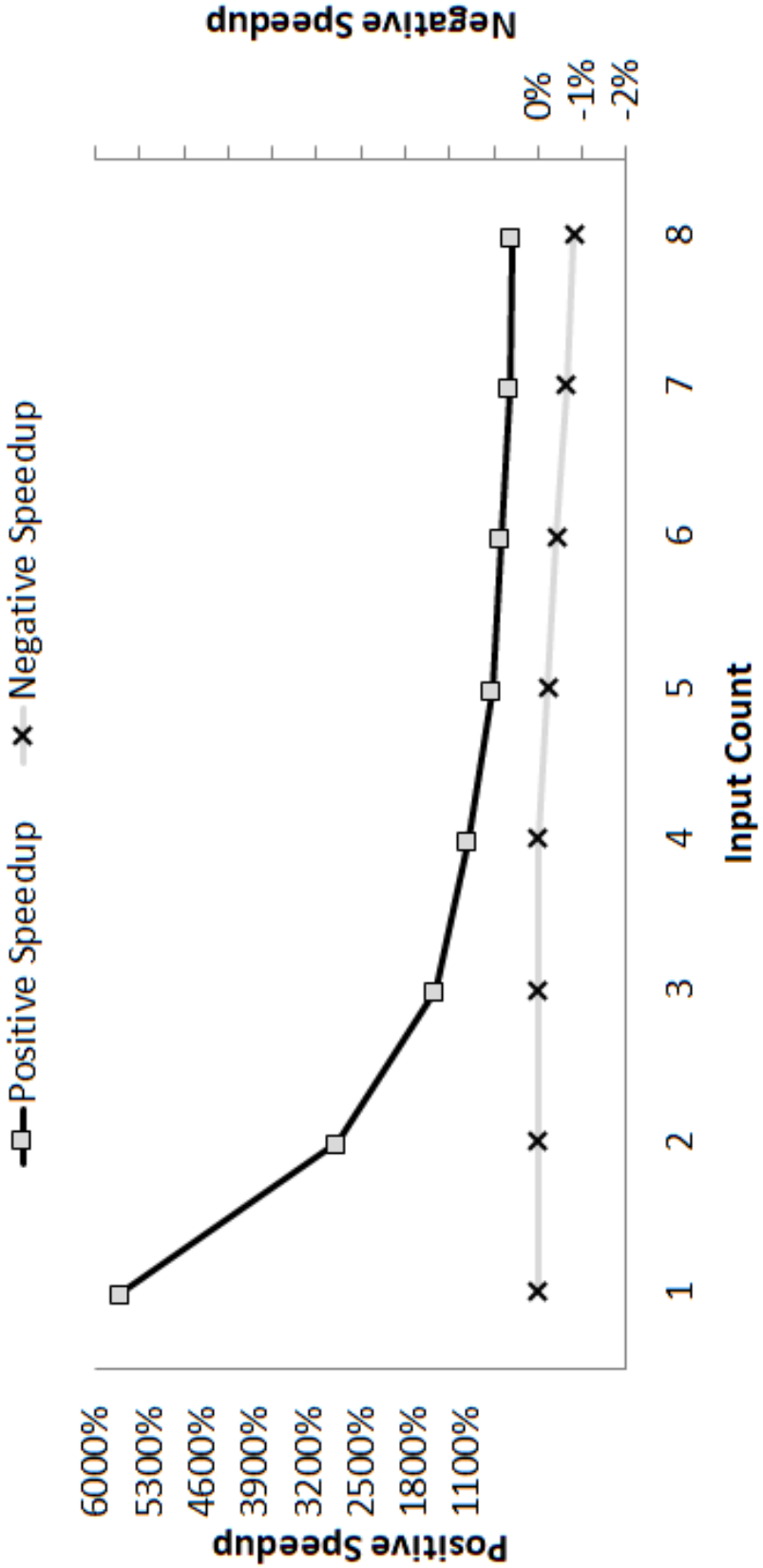


Figure 5.10: Rung count vs. performance VE results

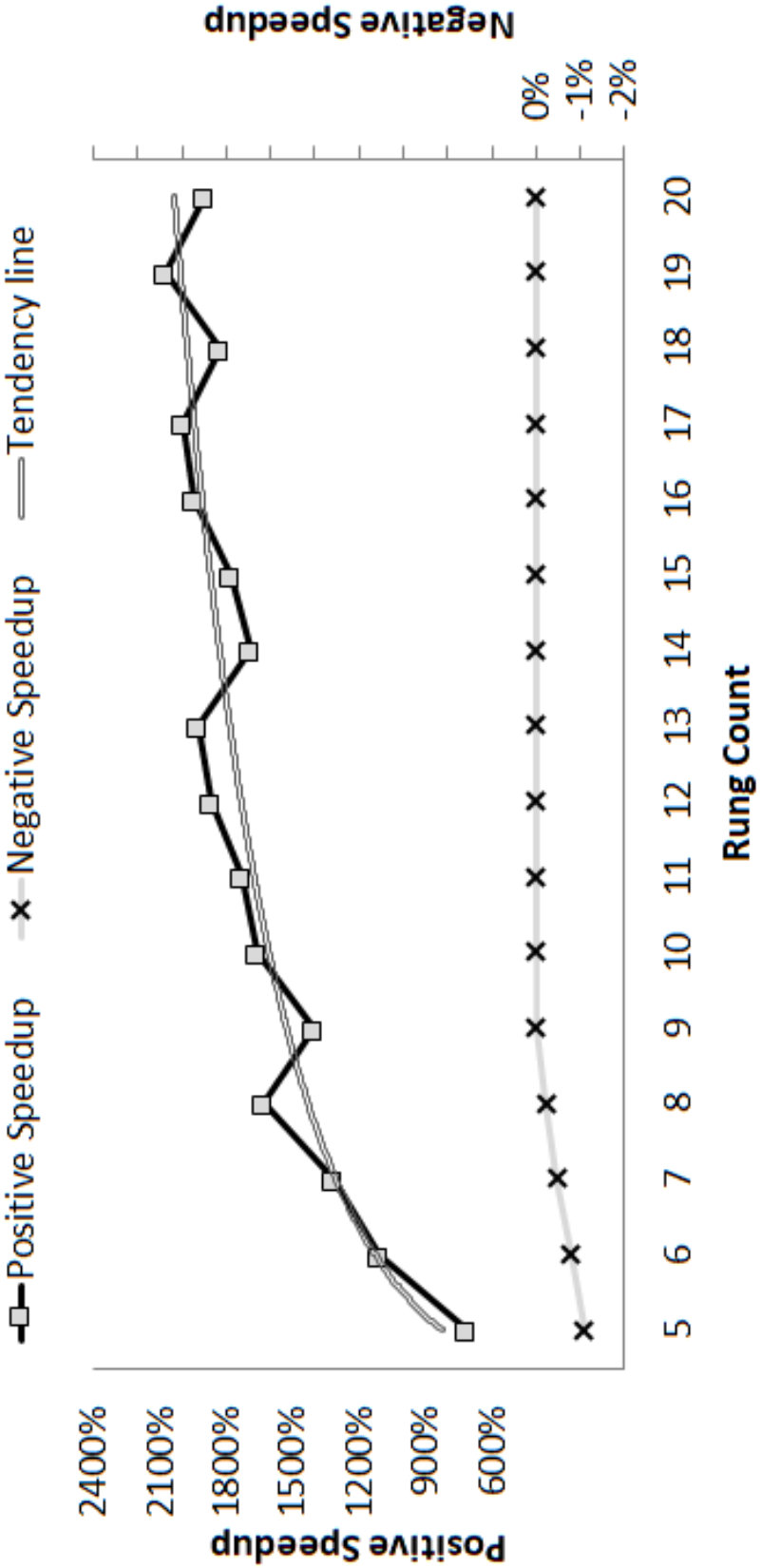


Figure 5.11: Rung count vs. performance SE results

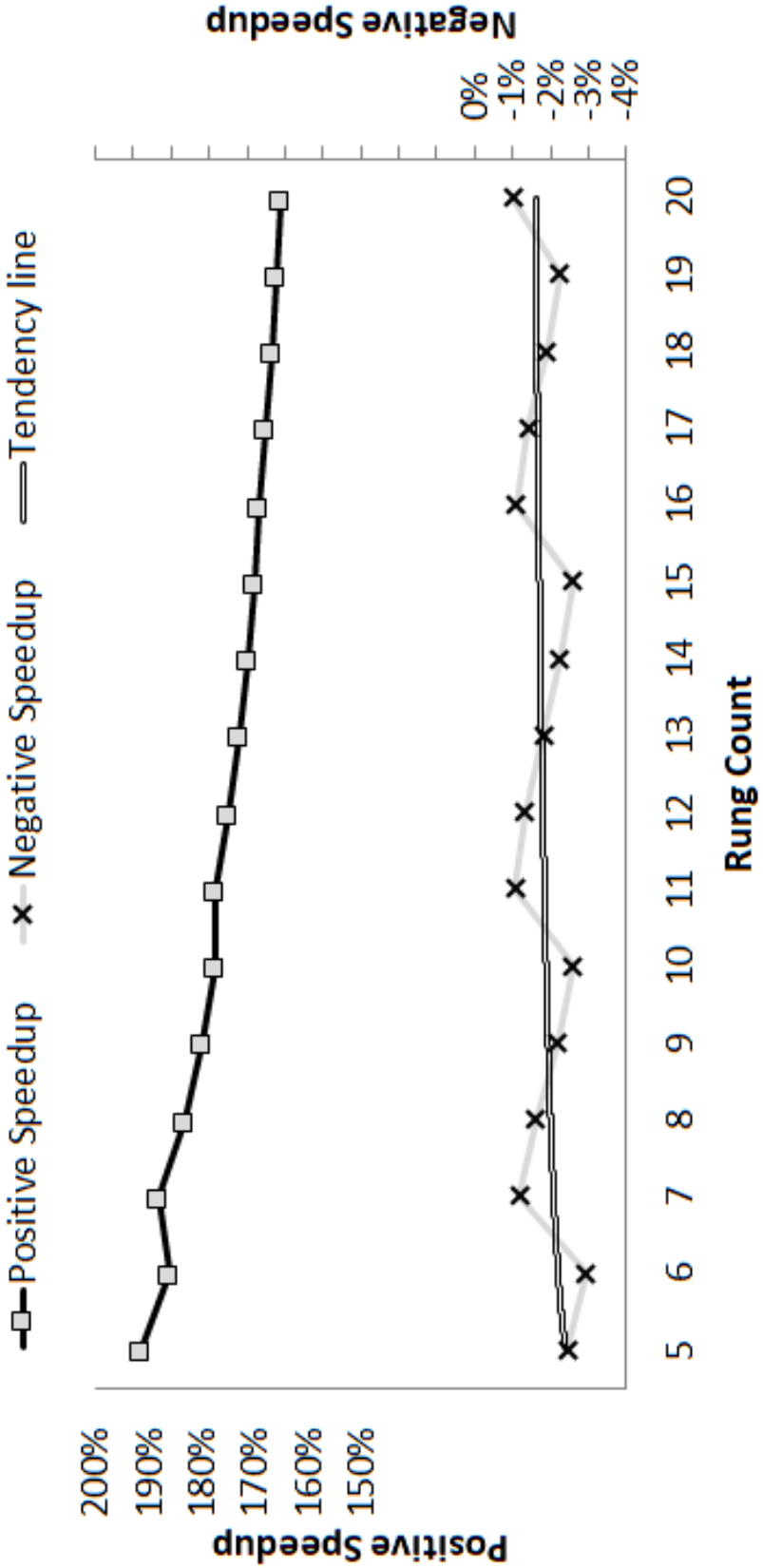
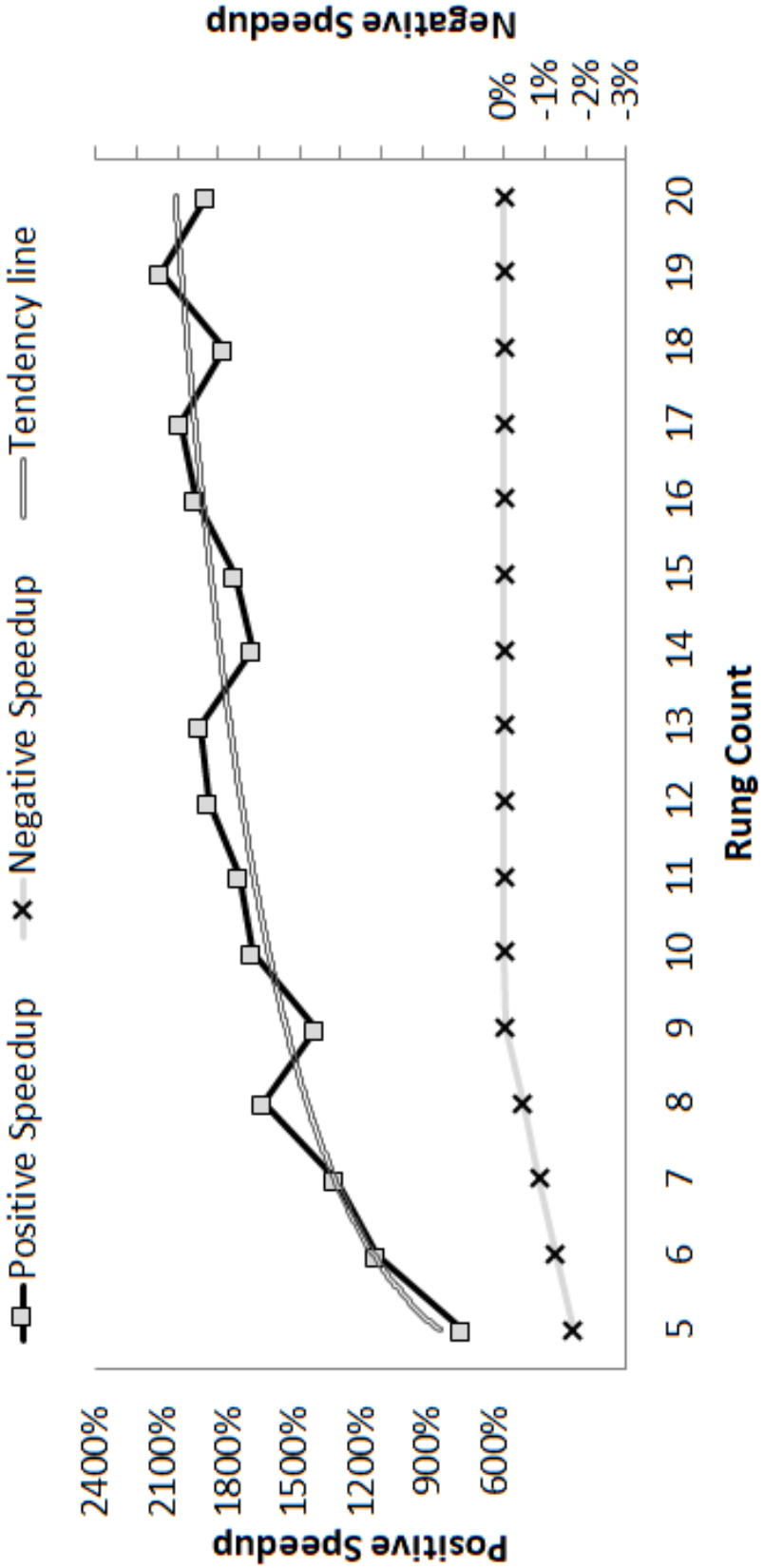


Figure 5.12: Rung count vs. performance VE+SE results



Subsequently, by analyzing the minimum, average, and maximum speedup probability of batch results presented in Table 5.6, it is possible to notice that the SE core is the best one in this attribute, followed by VE+SE, which has the best average speedup probability value. VE execution core was considered the worst execution core also by its minimum speedup value of -11% justified by the overhead in verifying several input edges in the worst scenario. Nevertheless, the VE core has an average performance similar to the other execution cores, benefiting from the widespread scenarios where no input variations occur, as can be observed in their results graphs.

Table 5.6: Speedup probability of results

	Minimum	**Average**	**Maximum**
VE	39.00	96.44	100.00
SE	93.75	98.76	99.87
VE+SE	50.10	99.22	100.00

Finally, since all proposed execution cores have pros and cons, the selection of which improved core to use will be determined by the application requirements, leaving to the developer of the solution the selection of the most appropriate one. In any way, proposed enhanced cores can contribute as a step ahead in enhancing PLC performance by reducing the scan time.

5.2 Multiple Execution Units and Types Evaluation

As already mentioned, there is a lack of official or widely known PLCs' benchmarks, except for the initiative TC3 of the PLCopen committee (WAL, 2009) which is unavailable and so does not apply to this research. Moreover, it is computationally unfeasible to evaluate all the large sets of all possible PLC programs in a continuation of the evaluation method of Section 5.1, but for a larger batch. Then, as a middle-point solution, to cover both extremes of PLC program formats, the second evaluation round considers two types of PLC programs for evaluation: the full balanced and unbalanced ones.

The full balanced PLC program type is the most extensive possible program composed of all the possible inputs of a given set in a given number of rungs. On the contrary, the full unbalanced PLC program type is the smallest possible program with a minimum of only one input of a provided set per each rung of a given number of rungs. By opting for those two PLC program types as benchmark limits, the overall performance can be evaluated since any other PLC program shape will be an intermediary category between

those two extreme types.

Besides adopting the full balanced/unbalanced program types as benchmarks, the second evaluation round also chooses the arithmetic mean of the number of execution cycles obtained for all the possible combinations of inputs' edges and/or rungs' memoization events as a performance evaluation parameter. As already justified, arithmetic mean it is a fair metric to evaluate the overall performance of the proposed improvements which avoids introducing any bias in the evaluation that may emerge by selecting the best or worst case of any simulation scenario.

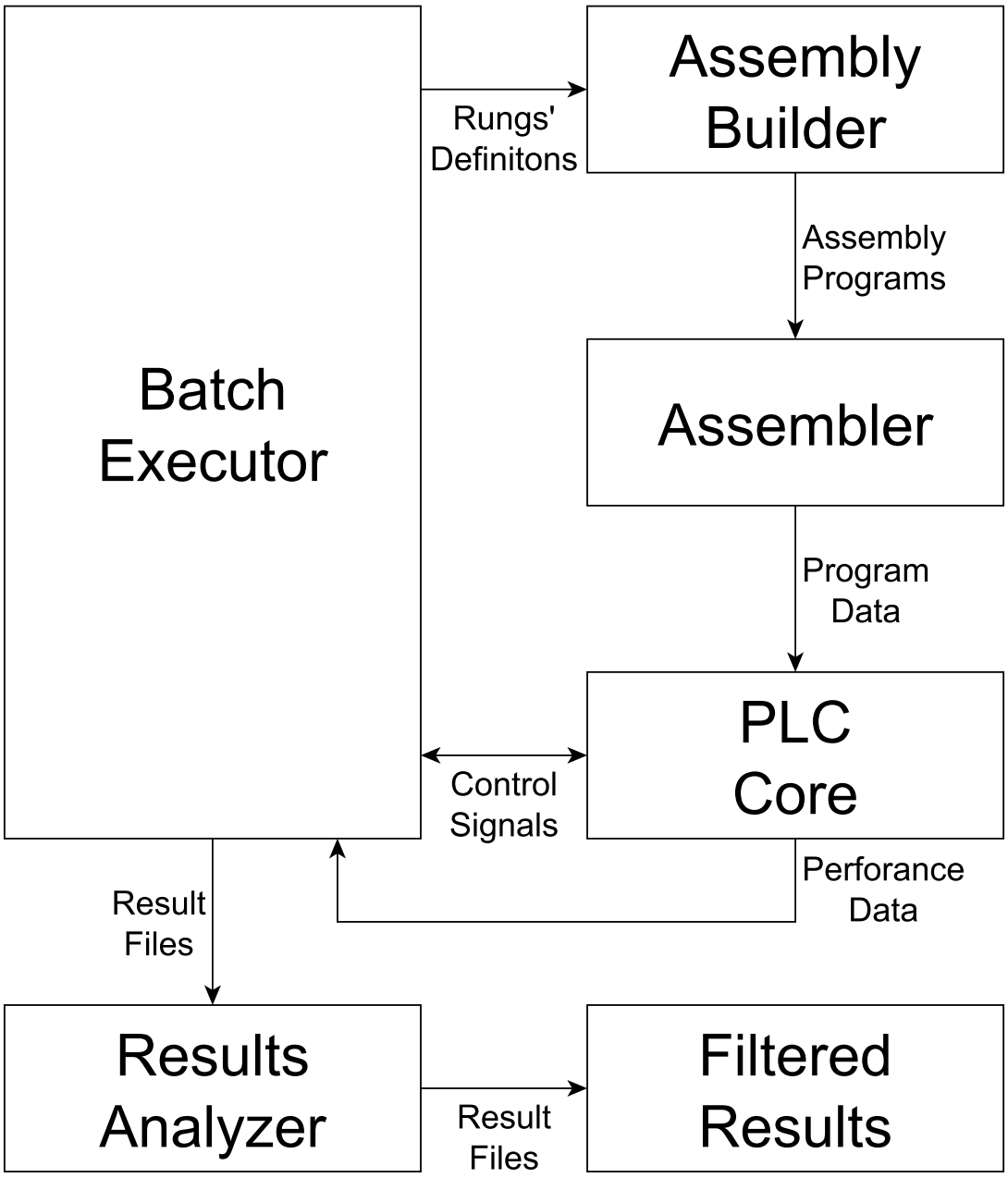
Therefore, the arithmetic mean of execution cycles of the improved architectures obtained from different simulation cases is computed in the CAS and then compared against the standard architecture performance as a baseline following the literature patterns (BOLTON, 2015) (WEBB; REIS, 1998) (LEWIS, 1998). Afterward, the calculated percentage difference in cycles is taken as the scan time reduction for the corresponding core. This calculation is a reasonable assumption since the execution phase of the scan cycle is the only part of the scan cycle that was enhanced in this research (the stages of reading from the inputs and writing to the outputs were not modified), and, consequently, a change in the execution time is the single aspect that affects the scan time duration.

Moreover, as discussed above, the developed CAS requires several input parameters to execute. Then, to cover all those requirements, as an advantage of the custom-made CAS development, and to speed up the generation of necessary simulation data, additional modules of the CAS were coded to streamline the execution of batches by simplifying batch execution parameters to a fewer count. Additionally, improvements in the accuracy of the CAS were made for the simulations of this Section, which result in an average difference of 5% in performance when compared to Section 5.1 results. Therefore, the results shown in this Section were the ones considered for the conclusions of this work.

As shown in the simulation workflow in Figure 5.13, the CAS assembly builder module receives from the batch executor module the rungs' definition values (input count, rung count, and instructions per rung) to generate the assembly programs using all instructions available in the ISA for both full balanced and unbalanced PLC program types. The assembler module takes the assembly program and not only produces machine code for the instructions themselves but also additional program data required by VE and SE improved cores. Then, generated program data are appropriately loaded into the current PLC's core, which may be of multi-cycle or pipeline execution unit type, and for a standard or improved core type with a defined count of execution units. Next, the batch

executor generates the control signals necessary to the PLC core that is currently in simulation and, once finished, collects the performance data to store into result files. Finally, once all possible simulations were completed, the results analyzer module obtains the simulations' result files and generates new filtered result files for graphical presentation.

Figure 5.13: Simulation work flow



The adoption of the simulation workflow of Figure 5.13 allows that any simulation batch may be initiated with a simple set of four parameters: inputs count, rungs count, instructions per rung and execution units count. This small group of parameters is enough for the CAS to generate all required performance results for all the cores, programs, and execution unit types. Hence, the simulation parameters were set starting with initial values of 5 for inputs and rungs, 10 for instructions per rung, and 1 for execution unit count, with increments of 5 for inputs and rungs, 10 for instructions per rung, and 2 times the previous value for execution units count. Then, simulations were executed until performance values reached stagnation in the scan time reduction, which had occurred with 15 inputs, 15 rungs, 100 instructions per rung, and 16 execution units count, thus avoiding the need to execute simulation batches with larger values.

Keeping the same strategy to avoid introducing deviations into the evaluation of the results, the computed scan time reductions were classified into worst and best case values. This categorization approach not only simplifies the presentation of results but also situates within which boundaries the performance boost of the execution cores was established since any other result will be a value between the worst and best case limits.

As can be observed in Figures 5.14 and 5.15, the VE core is quite affected by the PLC program type, since its performance boost comes from avoiding the execution of rungs. With several inputs belonging to each rung in a full balanced PLC program type, every single detected edge results in several rungs to be scheduled to execute. Thus, an extensive verification process culminates in an increase in the scan time. Although this increase in the scan time may be considered as an overhead, the perceived overhead tends to zero with the increase in the number of instructions per rung. Nevertheless, in a full unbalanced PLC program type, the VE core partially outperforms the SE core with substantial reductions already with lower values of instructions per rung, as can be observed in Figures 5.14 versus 5.16 and 5.15 versus 5.17. Hence, with the increase in the number of instructions per rung, the scan time reductions tend to the same plateau for both the VE and SE cores.

Since the SE core is not affected by input edges and, consequently, neither by PLC program type, similar results are observed for both balanced and unbalanced PLC program types, as displayed in Figures 5.16 and 5.17. Notwithstanding, performance deviations observed in Figure 5.16 are justified by the poor multi-cycle execution unit performance, since in the results for the pipeline execution unit those performance differences tend to zero, denoted by the fact that the performance curves overlap each other in

Figure 5.17.

Finally, a stable scan time reduction is observed in Figures 5.18 and 5.19, where the combination of the VE and SE improvements presents a better overall performance than the VE and SE improved cores alone. Only with the pipeline execution unit in a full balanced PLC program type scenario, the VE+SE core presents a worse performance than the SE core alone, which is explained by the poor performance of its VE section. However, the performance difference is not so significant as in the standalone versions of the VE and SE core types. As a performance evaluation conclusion, the VE+SE core can be highlighted as superior to the VE and SE cores alone, especially considering that in real applications the PLC programs tend to be more unbalanced than balanced.

As can be observed in the results for the single execution unit cores, in general, the proposed improvements result in substantial scan time reductions compared to a standard PLC architecture. However, considering the type of the execution unit, the pipelined unit has a lower scan time reduction, since it is an already enhanced execution unit, so its standard baseline has a higher performance value than the multi-cycle execution unit. From another perspective, in the analysis of both full balanced/unbalanced PLC program types, the difference in performance between the VE and SE improvements presented in the results is remarkable.

Likewise, the performance differences between VE and SE cores in single execution unit types, considering the PLC program type, are also observed in the performance of multiple execution units. Notwithstanding, a small scan time reduction is perceived in the VE+SE core with more than one execution unit, where even its SE part is insufficient to keep its performance, as shown in Figure 5.25. Even though, similarly to cases with one execution unit, this result is better than in the case with a single VE core, as can be observed in Figure 5.21.

By analyzing the scan time reductions for single and multiple execution units, it is possible to draw relevant conclusions. The VE+SE core is, in general, a better solution, with better values in the best cases and average numbers in the worst cases of scan time reductions. The SE core is a balanced solution, with similar numbers for both the best and worst scan time reductions. Finally, the VE core may be considered the worst improvement with a substantial execution overhead in full balanced PLC program types, which is not entirely compensated by its partially superior performance in full unbalanced program types with small values of instructions per rung.

Figure 5.14: Single execution unit multi-cycle VE results

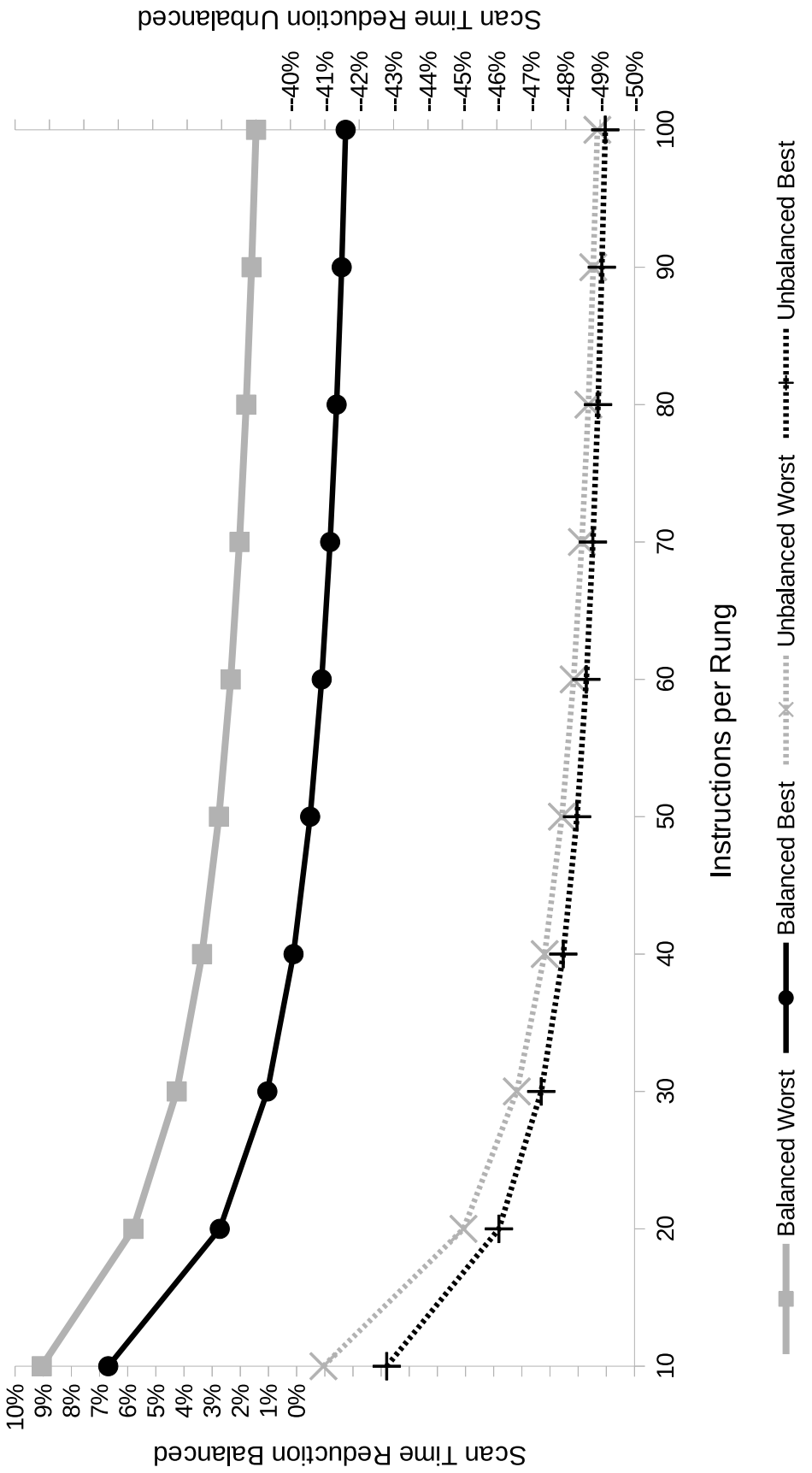


Figure 5.15: Single execution unit pipeline VE results

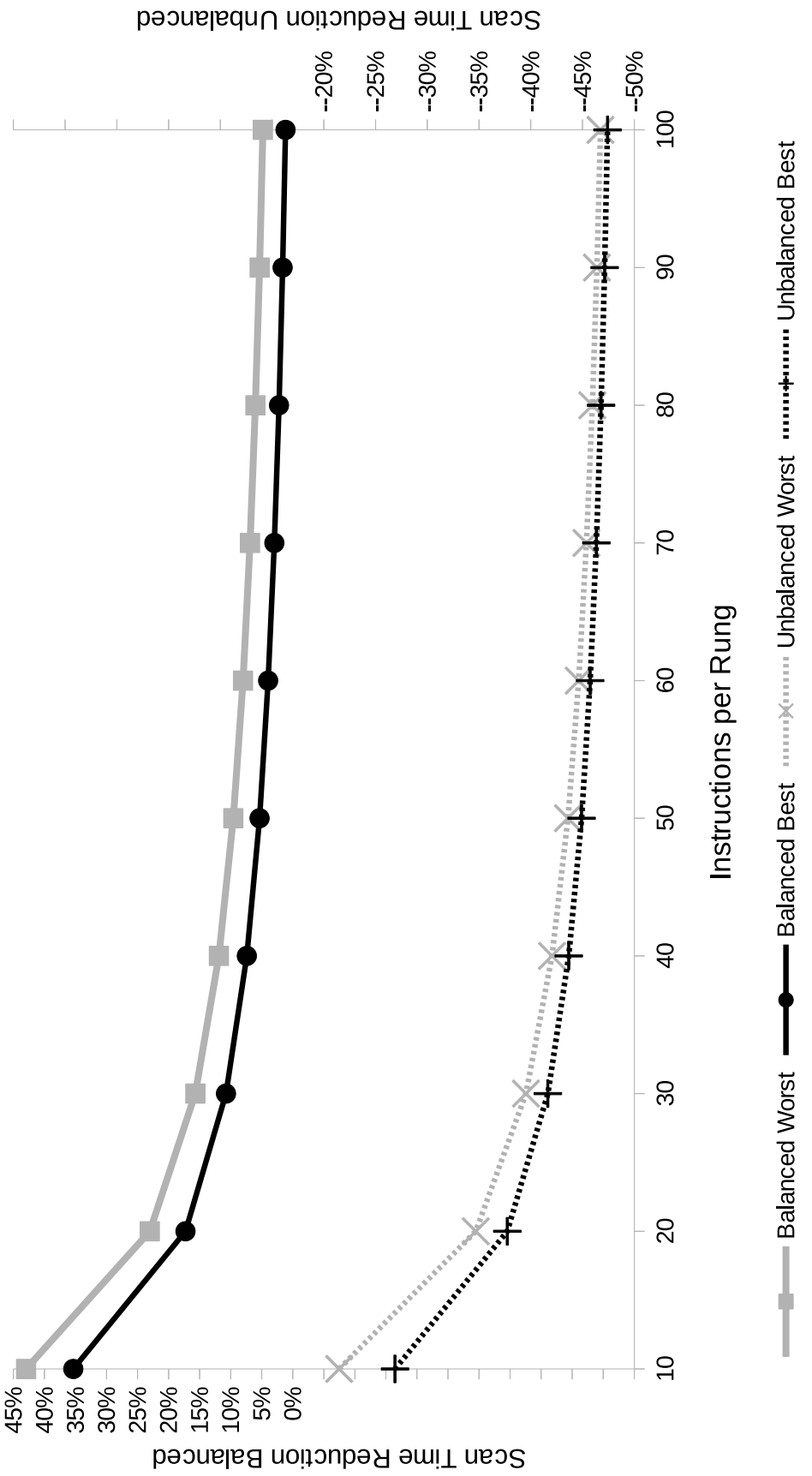


Figure 5.16: Single execution unit multi-cycle SE results

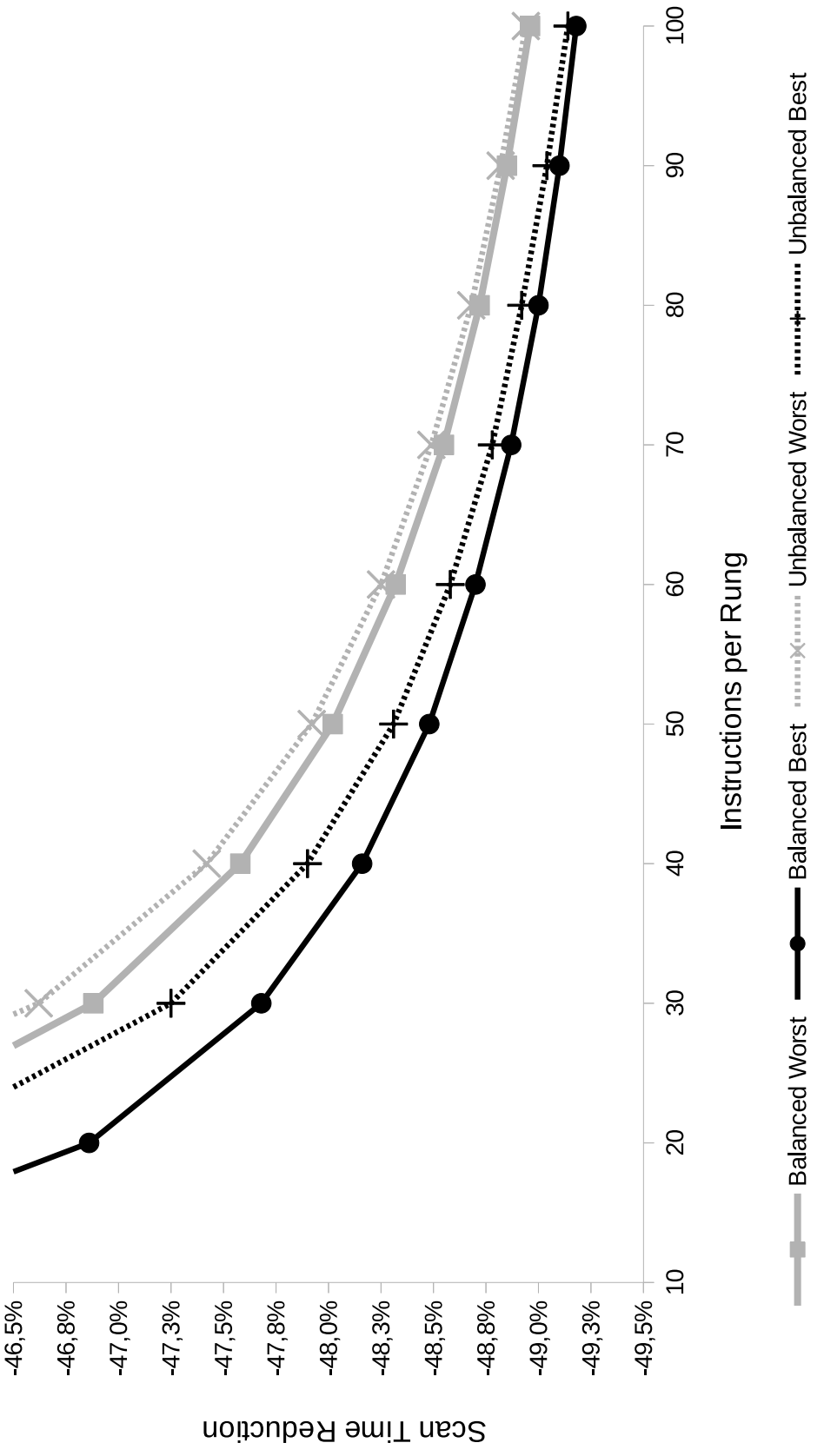


Figure 5.17: Single execution unit pipeline SE results

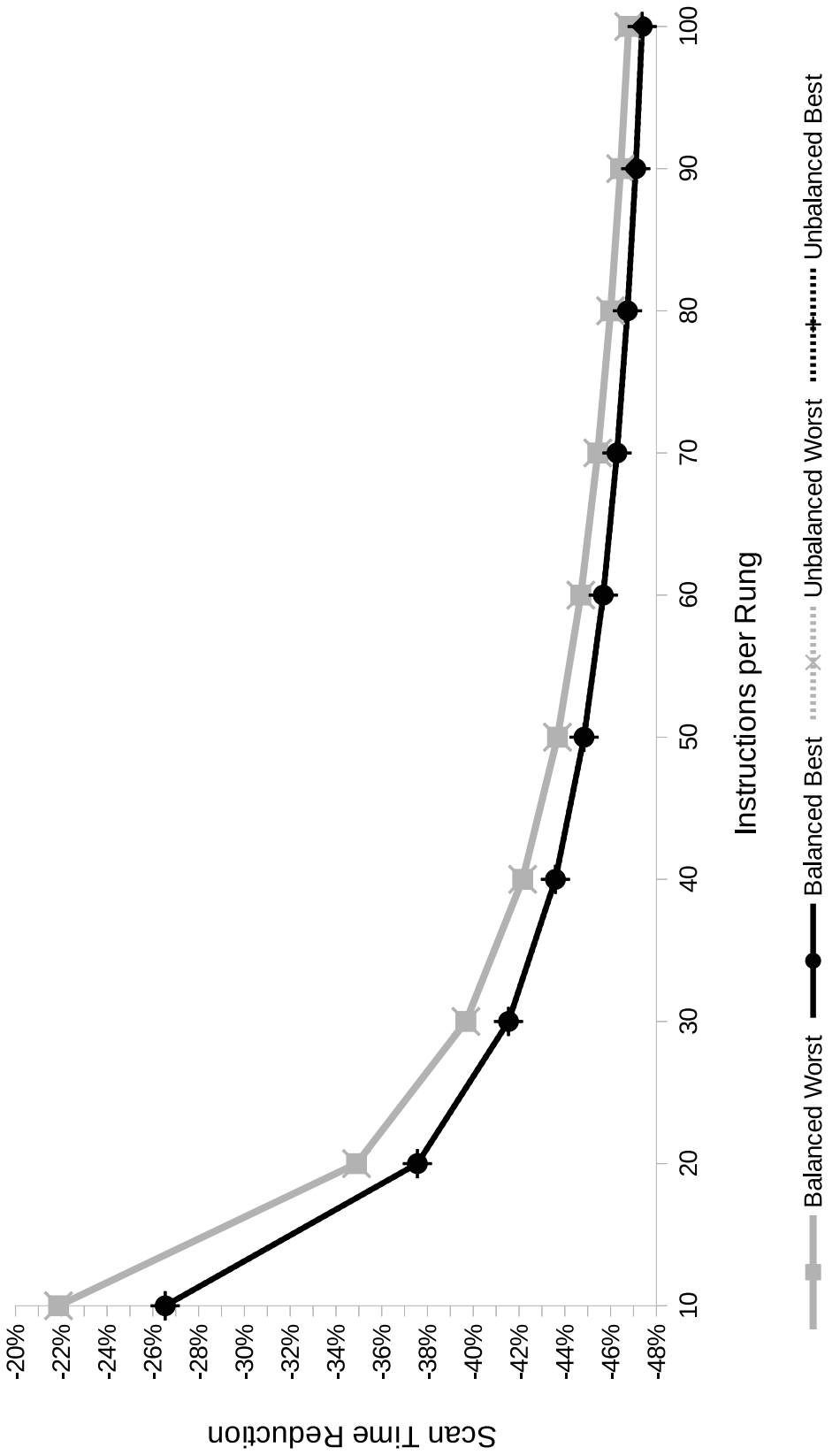


Figure 5.18: Single execution unit multi-cycle VE+SE results

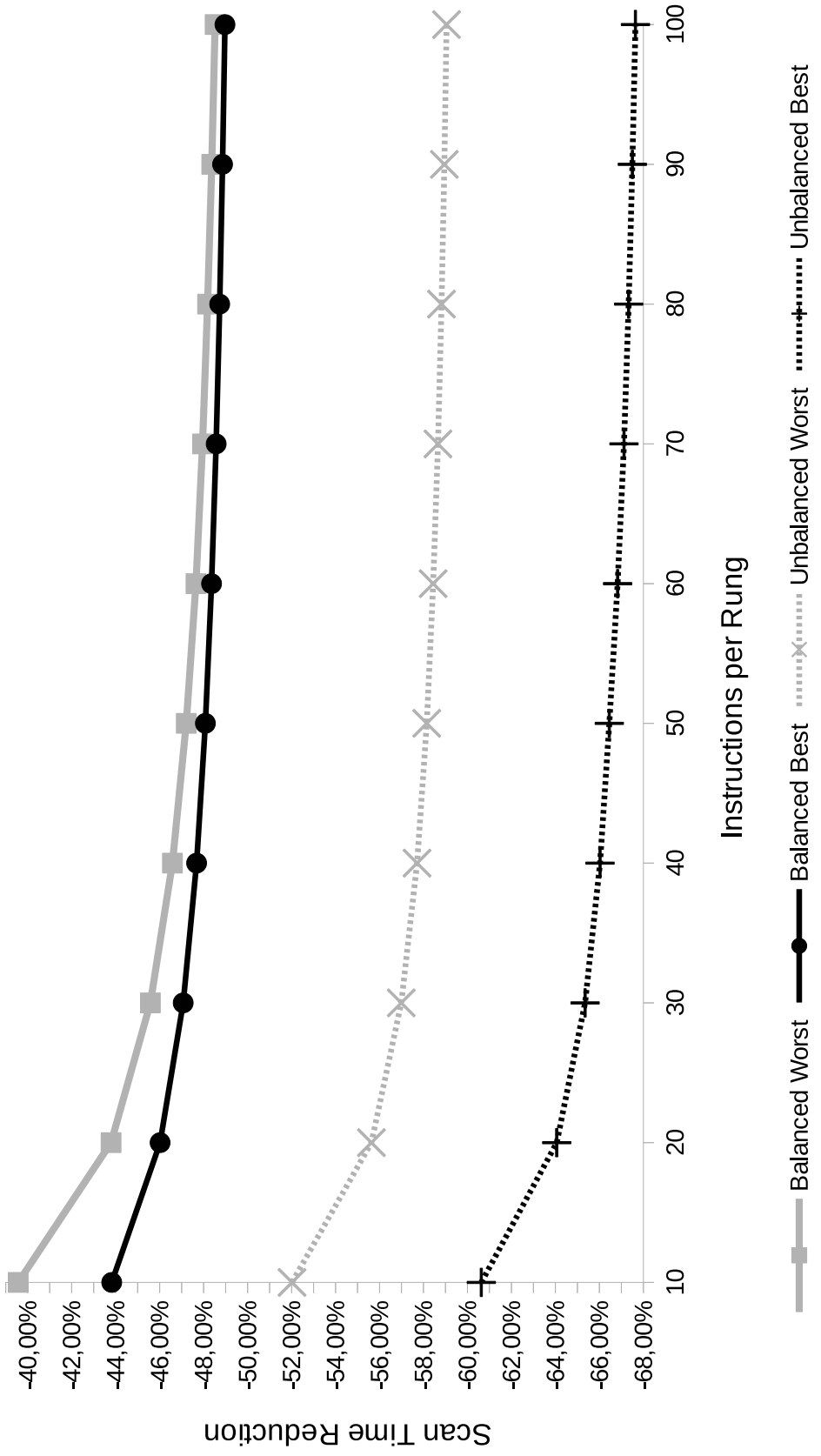


Figure 5.19: Single execution unit pipeline VE+SE results

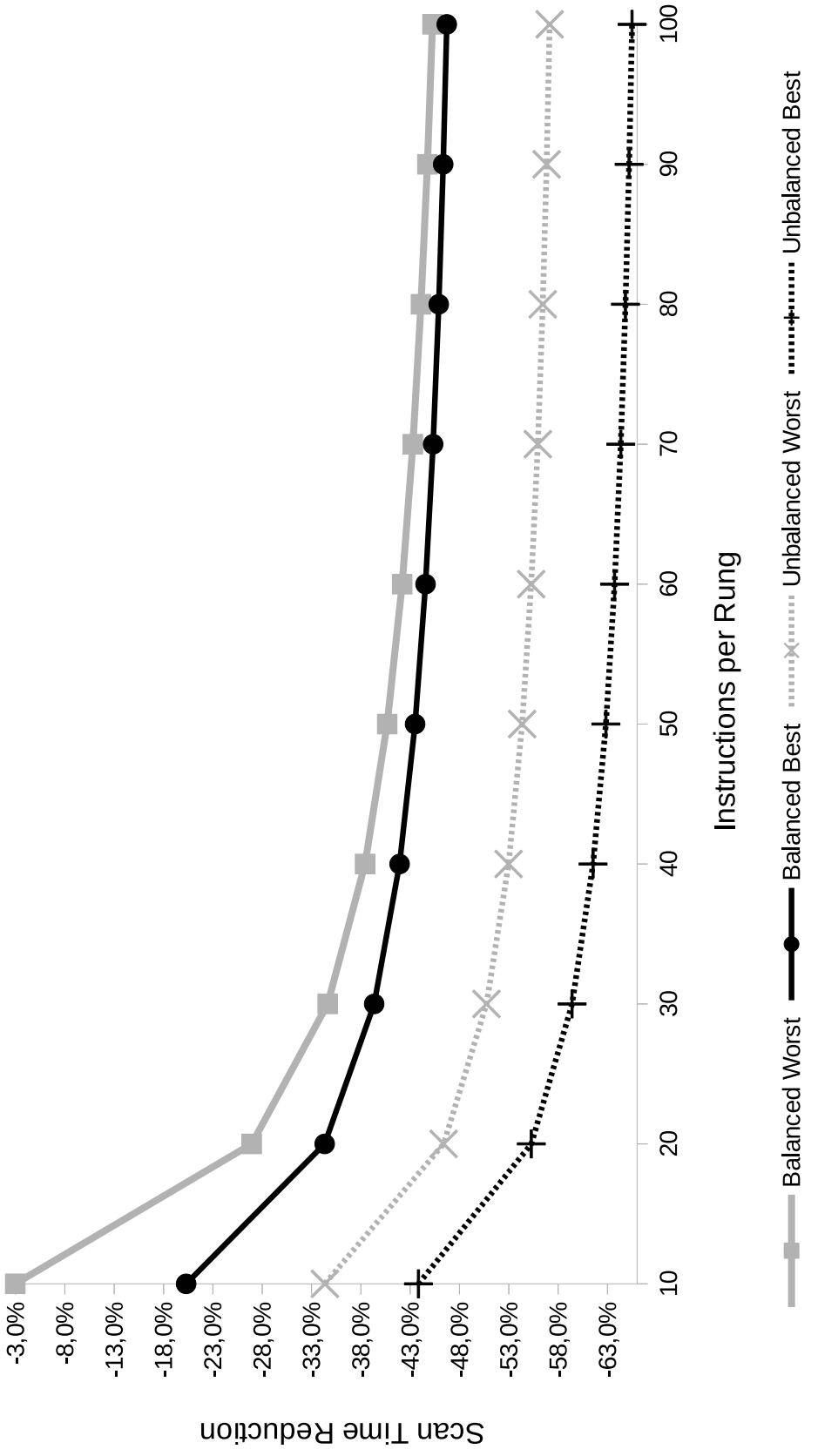


Figure 5.20: Multiple execution units multi-cycle VE results

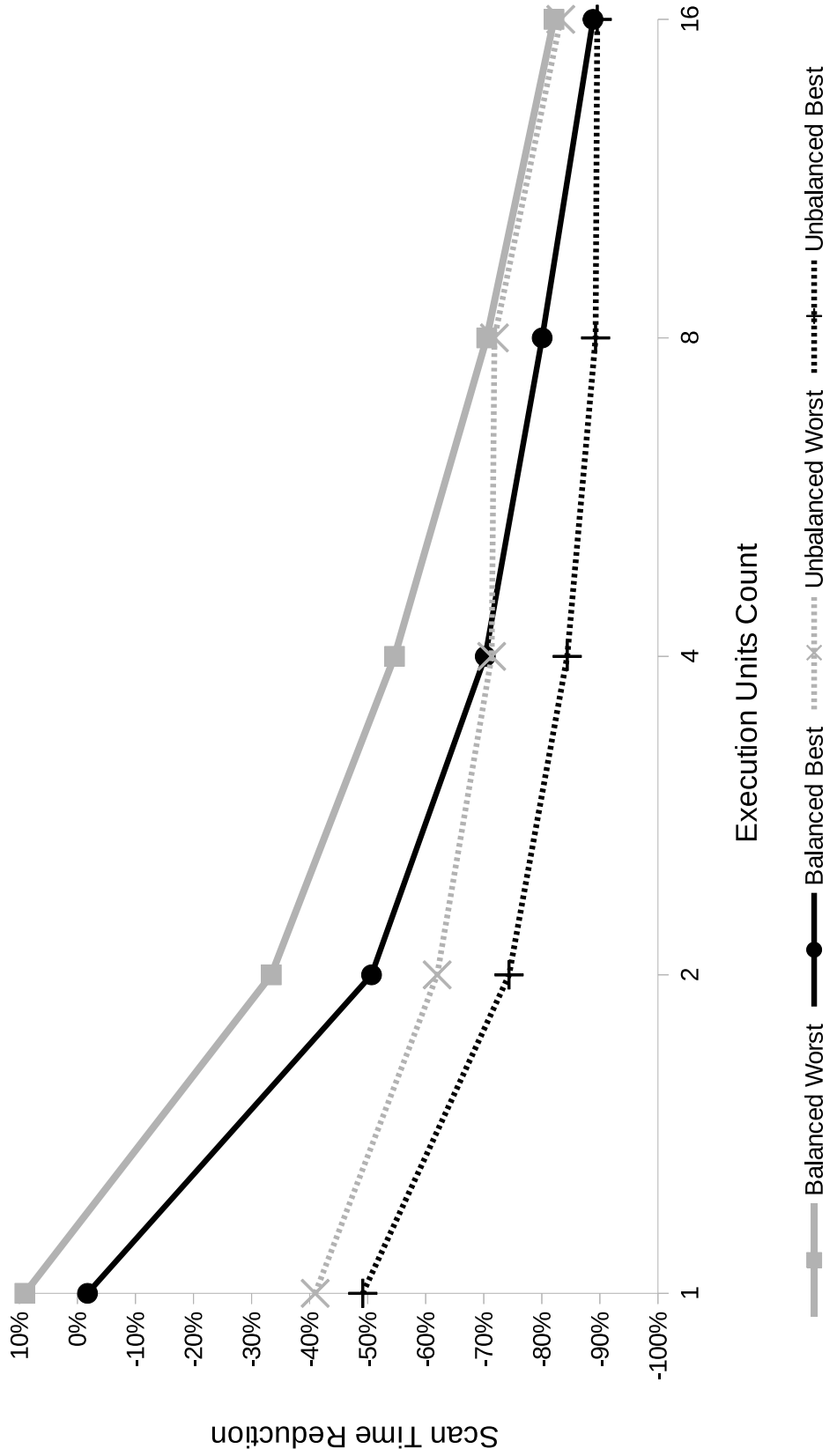


Figure 5.21: Multiple execution units pipeline VE results

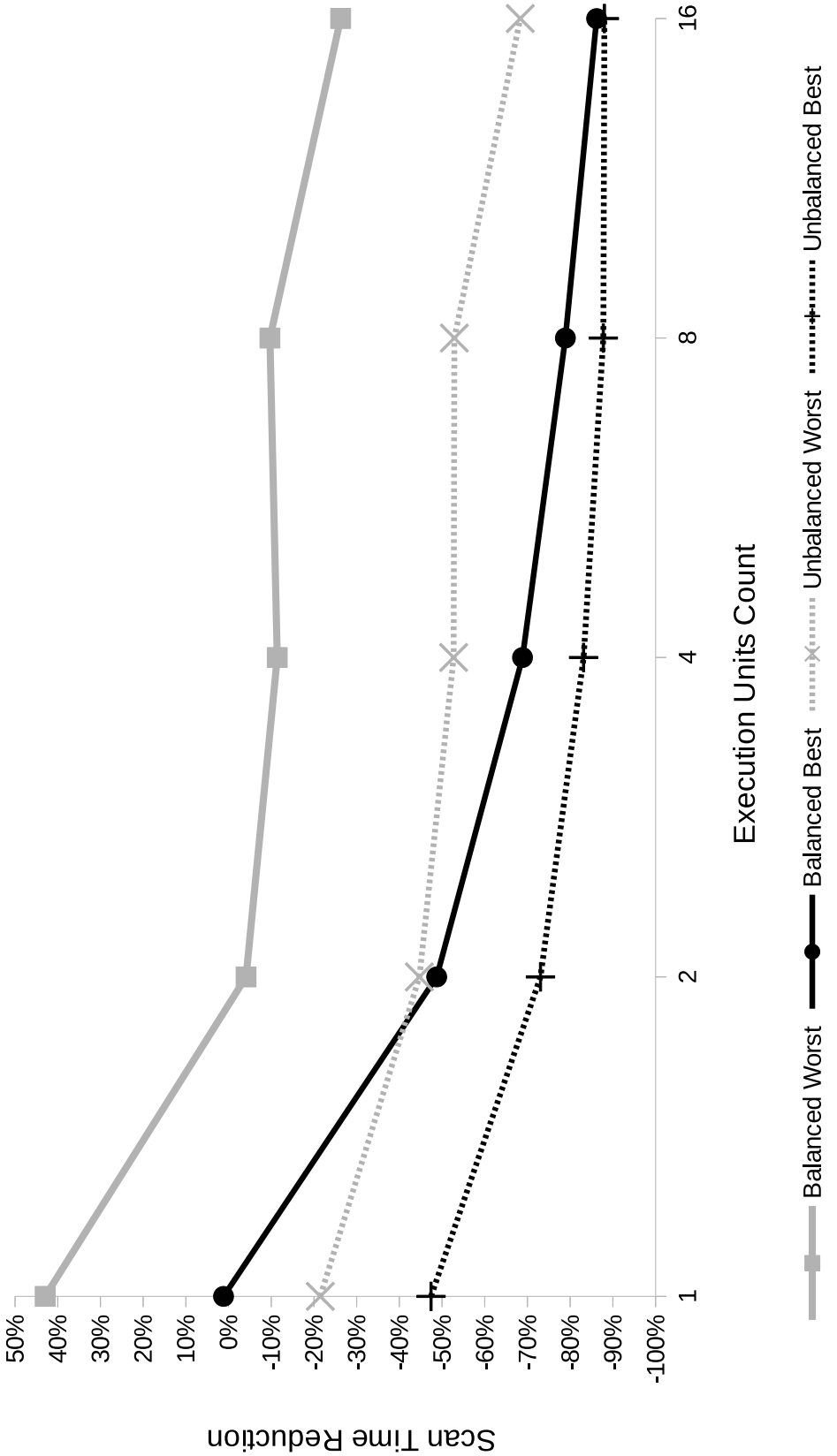


Figure 5.22: Multiple execution units multi-cycle SE results

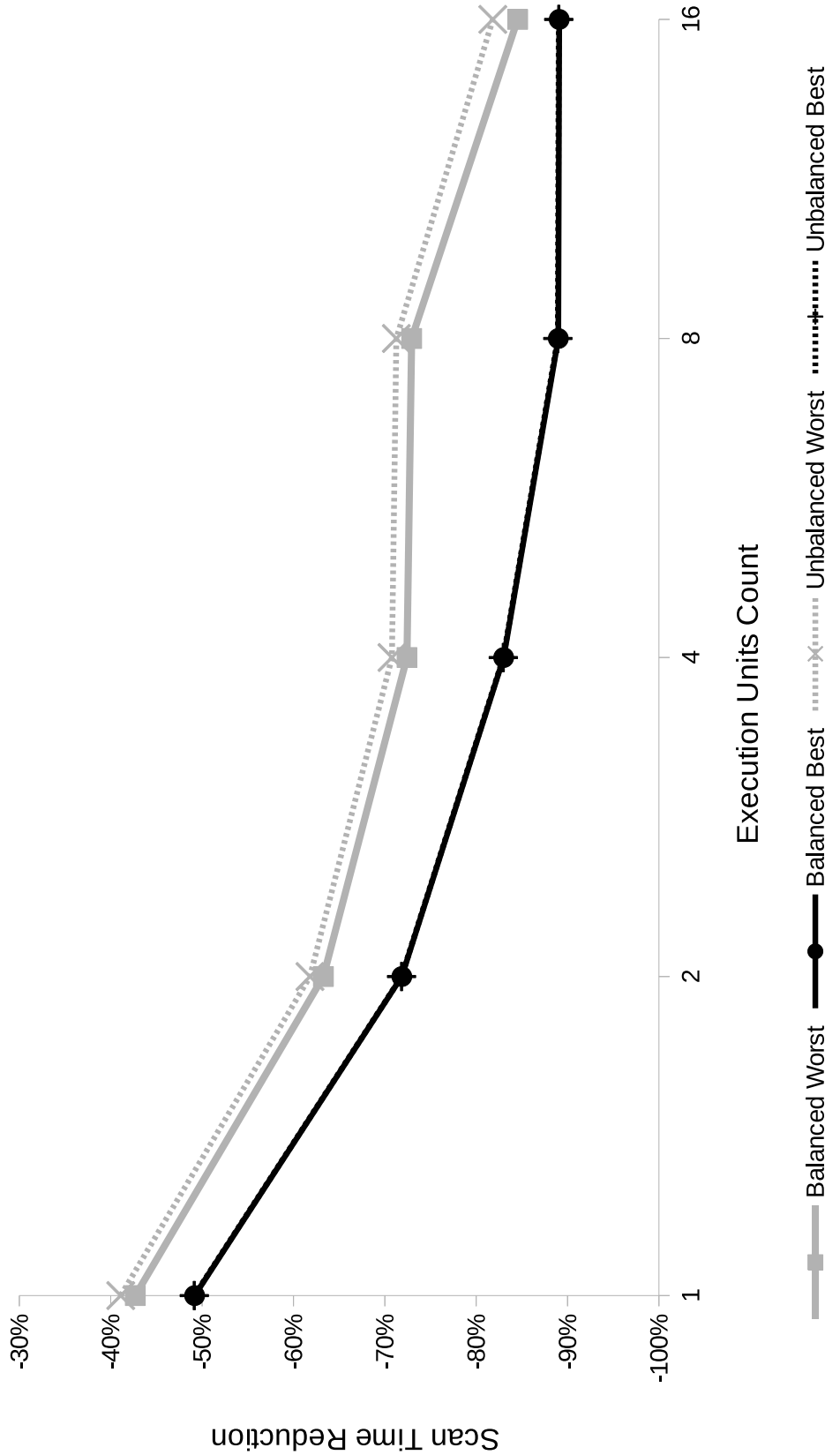


Figure 5.23: Multiple execution units pipeline SE results

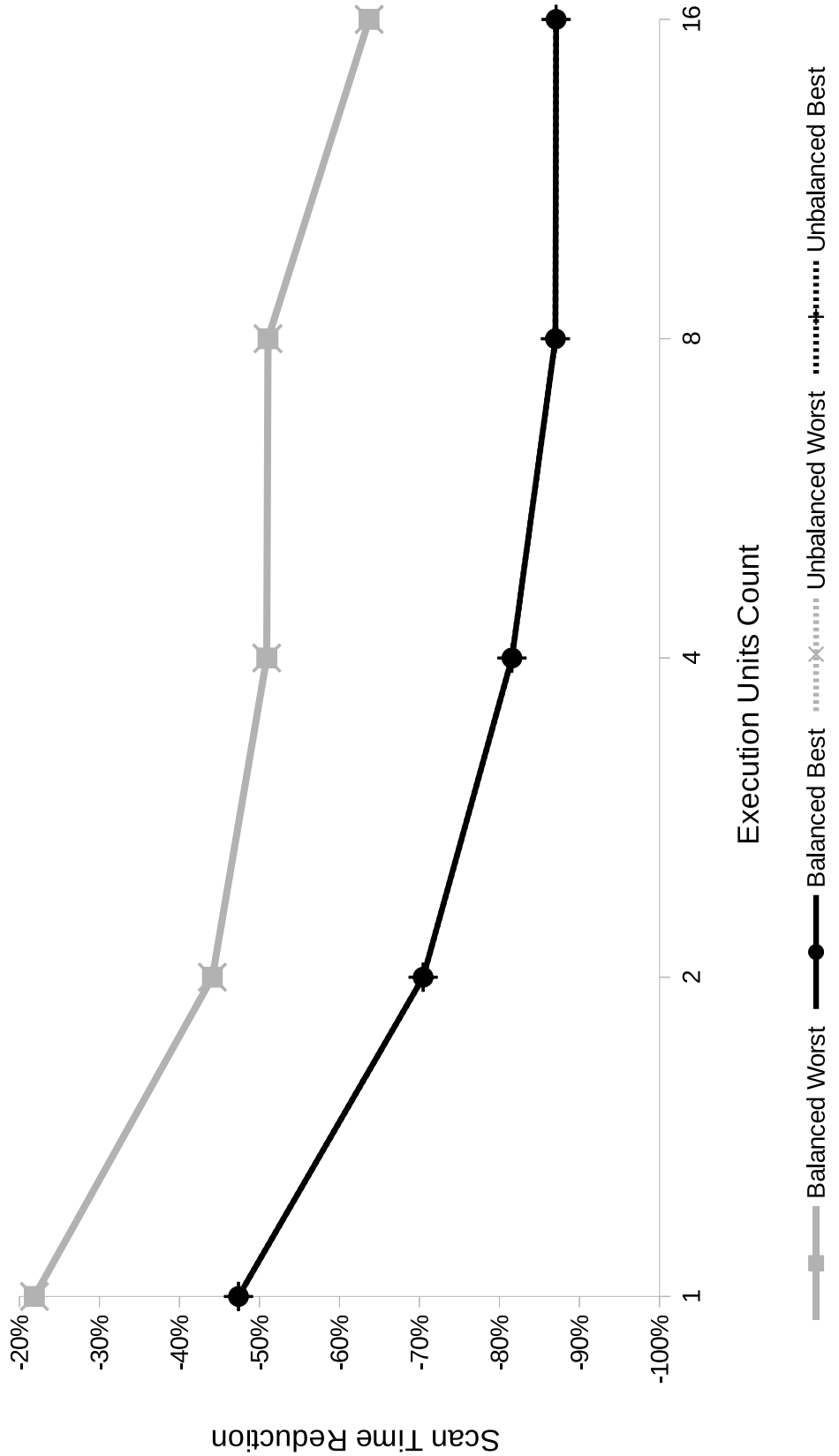


Figure 5.24: Multiple execution units multi-cycle VE+SE results

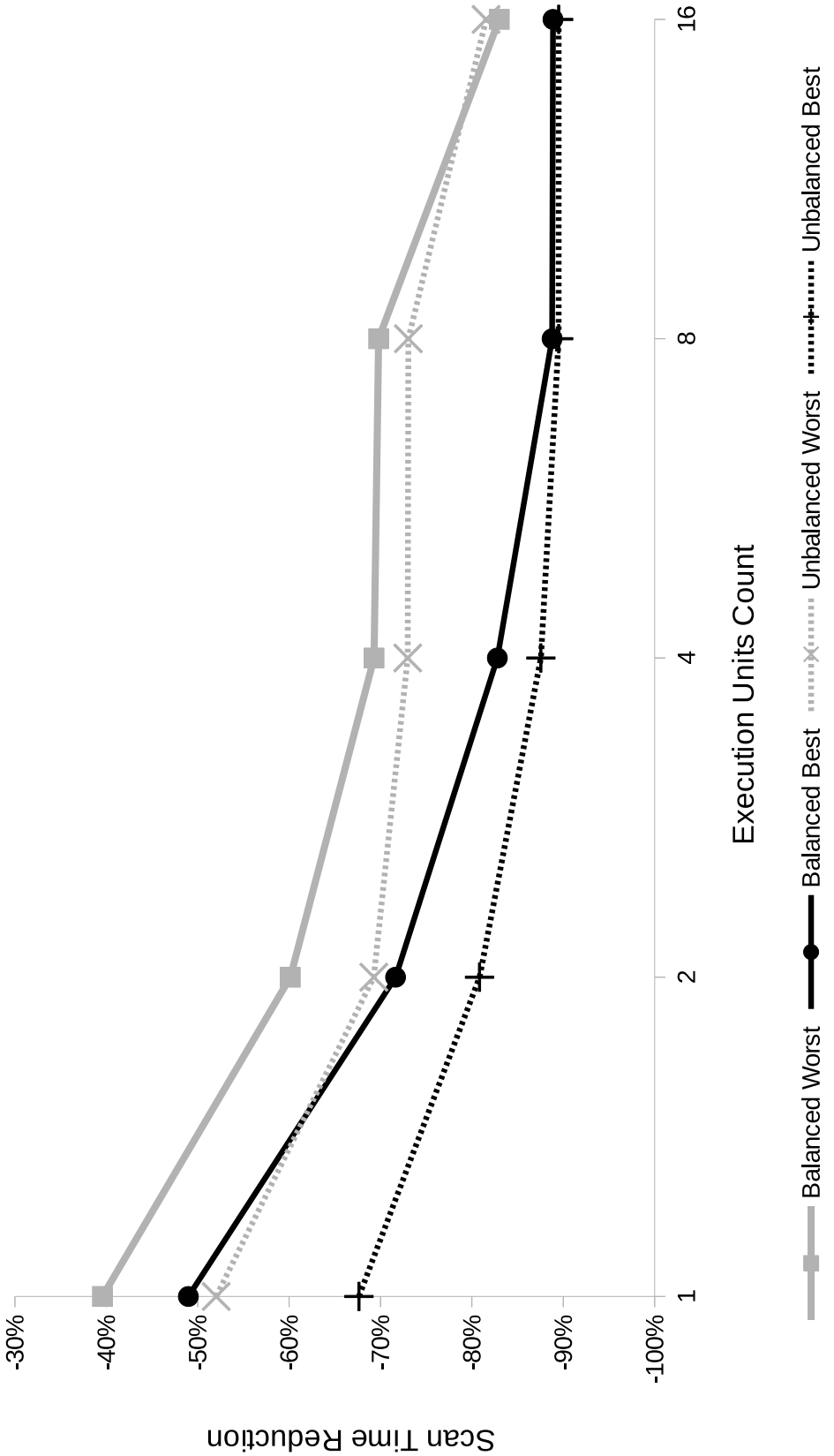
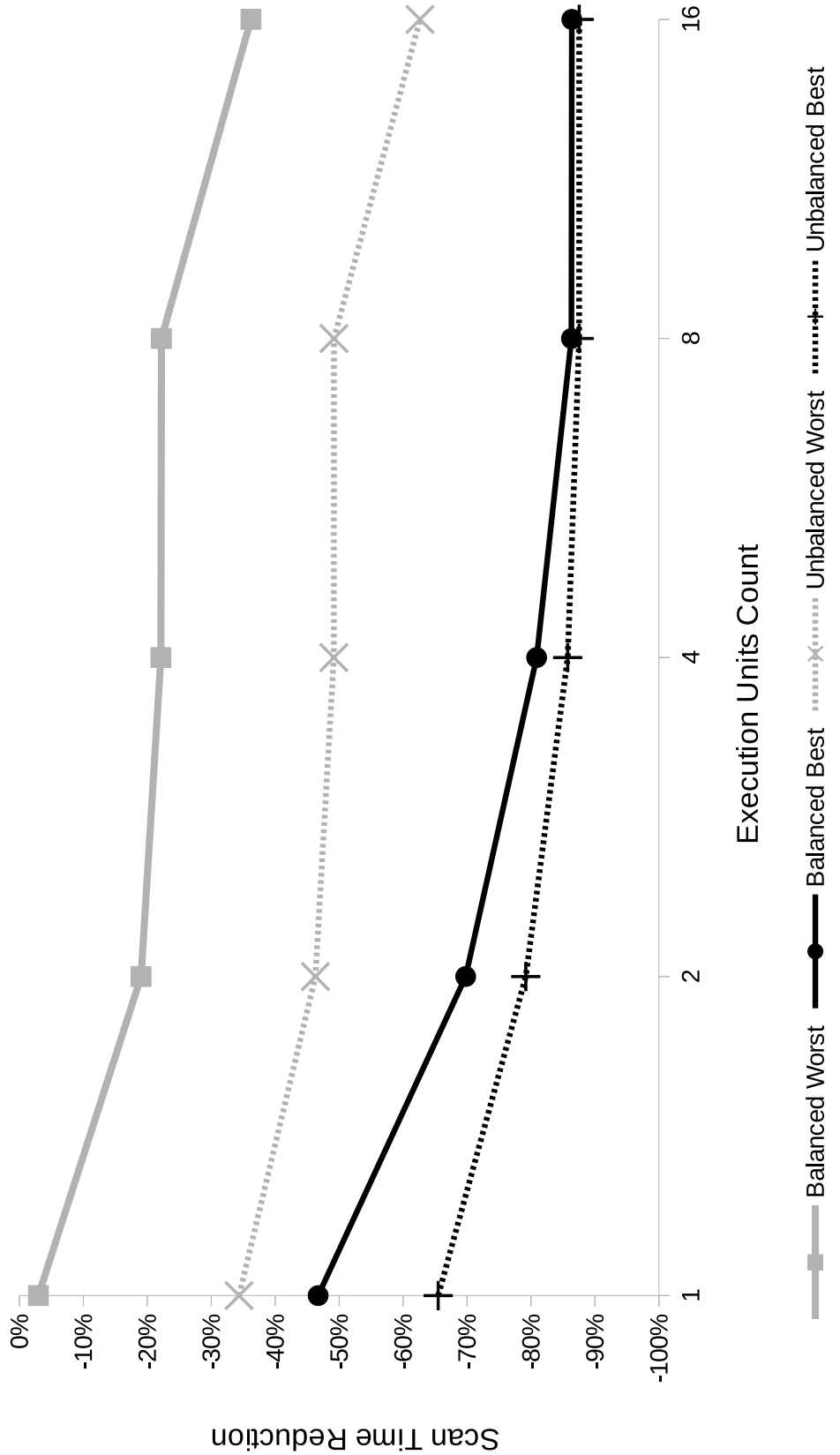


Figure 5.25: Multiple execution units pipeline VE+SE results



5.3 Trade-off analysis of performance and area

Higher execution performance is not the only desired characteristic since it typically demands a large silicon area, so this requirement must also be evaluated to reach a robust inference on which PLC core is the best from a more global perspective. To obtain a silicon area measurement for the various core types, the McPAT framework (LI et al., 2009) was selected due to its simple utilization and substantial accuracy.

To determine the impact in the silicon area of the proposed architecture in McPat, the selected strategy was to outline the differences between the various improved PLC cores. The main differences between proposed cores regard execution unit quantity, register count, number of arithmetic logic units (ALUs), and memoization memory size. Considering these differences, the architectures were mapped into appropriate parameters in McPat, where at the particular case of the memoization memory size, it was considered as L1 cache size. Then the area overheads were calculated based on the variation concerning the corresponding architecture of the same type containing a single execution unit.

As observed in Figures 5.26 and 5.27, the cores with the pipeline execution unit type has a lower impact on the area overhead since this type of execution unit is already an enhanced architecture component, thus requiring a more extensive area in the baseline core. Naturally, for both multi-cycle and pipeline execution unit types, the area overhead increases as the number of execution units increases (the number of execution units is denoted by the small number inside each symbol of the graphs). Figures 5.26 and 5.27 also demonstrate that the superior average performance of the SE core, and consequently also of the VE+SE core, has a high cost in terms of silicon area when compared to the standalone VE core.

Moreover, the results presented indicate that the VE core may be considered the best one in the trade-off between performance and silicon area. The advantage of the VE core is even more pronounced when considering that the memoization memory size of 128KB used for silicon area calculation of the SE and VE+SE cores is sufficient to merely store a whole set of memoization cases of a fully balanced program type of 2 rungs with 10 inputs each since each memoization key has 64 bits. There is an even more significant advantage for the VE core when we consider that the batch performance measurements were based on a simulation option with an unlimited memoization memory size. This simulation option was chosen because determining the optimal size of the memoization

memory is a subject out of the scope of this work. The precise determination would require a more detailed study; thus, no specific value was also adopted to avoid a biased simulation that could emerge depending on the chosen memoization memory size.

Other aspects that must be addressed as future work are the implementation of the function to be used to search for an item in the memoization memory and the number of cycles spent in the memoization checker verification process, with a realistic value for the miss penalty in cycles, which was considered as just one cycle for this experimental study. Similar to precise size determination, search function implementation and determination of verification/miss penalty cycles are also out of the scope of present work. With this simplified assumption, this work indicates a maximum performance that SE improvement can achieve, which will guide the direction of future work.

From another perspective, the memoization memory does not need necessarily to accommodate the full range of possible memoization events to achieve its best performance in a real PLC application, since only the most recurrent memoization events need to be cached. Therefore, the memoization memory size can be reduced to decrease the area overhead, thus moving the values of the cores containing the SE improvement to the left on the X-axis of Figures 5.26 and 5.27.

5.4 Final Considerations from the Experimental Results

Lastly, after analyzing batch results for the proposed improvements, along with the trade-off between area overhead and average performance, it is possible to assess the pros and cons of the proposed VE and SE improvements, execution unit types, number of execution units, and then reach some firm conclusions.

The VE core presents a significant performance boost, particularly in unbalanced PLC program types, thus partially compensating its worst performance in balanced PLC program types. It offers the best trade-off between performance and area overhead among the proposed cores. Hence, the VE core is the first choice to improve PLC architectures, at least for the best cases of the most unbalanced program types, with a minor impact on silicon area overhead.

Despite its stable performance, the SE core has a substantial area overhead correlated with the size of its memoization memory. This drawback could be partly avoided with an enhancement in the memoization memory, with the addition of a ranking mechanism that keeps cached the top requested values, thus allowing to reduce the memoization

memory size. This new enhancement would maintain the performance benefits of the SE core, but certainly with a smaller silicon area usage.

The implementation of an enhanced ranked memoization memory would also be favorable to the VE+SE core, also reducing its bad trade-off between area overhead and performance. Moreover, since the VE+SE core includes all the proposed improvements, it would be perfect for a refined execution model in which the usage of its SE and VE parts could be enabled/disabled in the PLC program development stage or on-the-fly during execution. This dynamic switching could help avoid the disadvantages of both VE and SE cores in specific PLC programs, by turning off architectural components at a particular state of the workload or by PLC program developer decision.

Simulation batch results and calculated silicon area values delimited the pros and cons of the proposed architectural improvements and allowed this research to draw concrete conclusions in this regard. By its superior performance in unbalanced program types, combined with its low silicon area impact, the VE improvement is the first choice for low-end PLC devices, especially for monotonous systems, where a prompt response not found in standard PLCs is required. In middle-end PLC devices, compensating its downside of increment in the silicon area, the SE improvement is ideal to obtain superior performances by keeping reduced scan times even in applications with several input edges, where a standard PLC does not show adequate performance. Lastly, for high-end PLC devices, for which performance and low scan times are mandatory requirements at any cost, the VE+SE improvement has a balanced performance independently of PLC program type, keeping performance at superior overall levels when compared to standard PLC cores.

Finally, since all proposed improved cores have, in general, superior performance over standard PLC architecture cores, they can be used to achieve scan time reduction even if based on execution units with weak performance, like the multi-cycle one, avoiding more complex execution units or even multiple execution units. Consequently, with a fraction of the area of more complex solutions, the proposed improved cores can be adopted with advantages in commercial PLC devices.

Figure 5.26: Multiple execution units multi-cycle area vs performance results

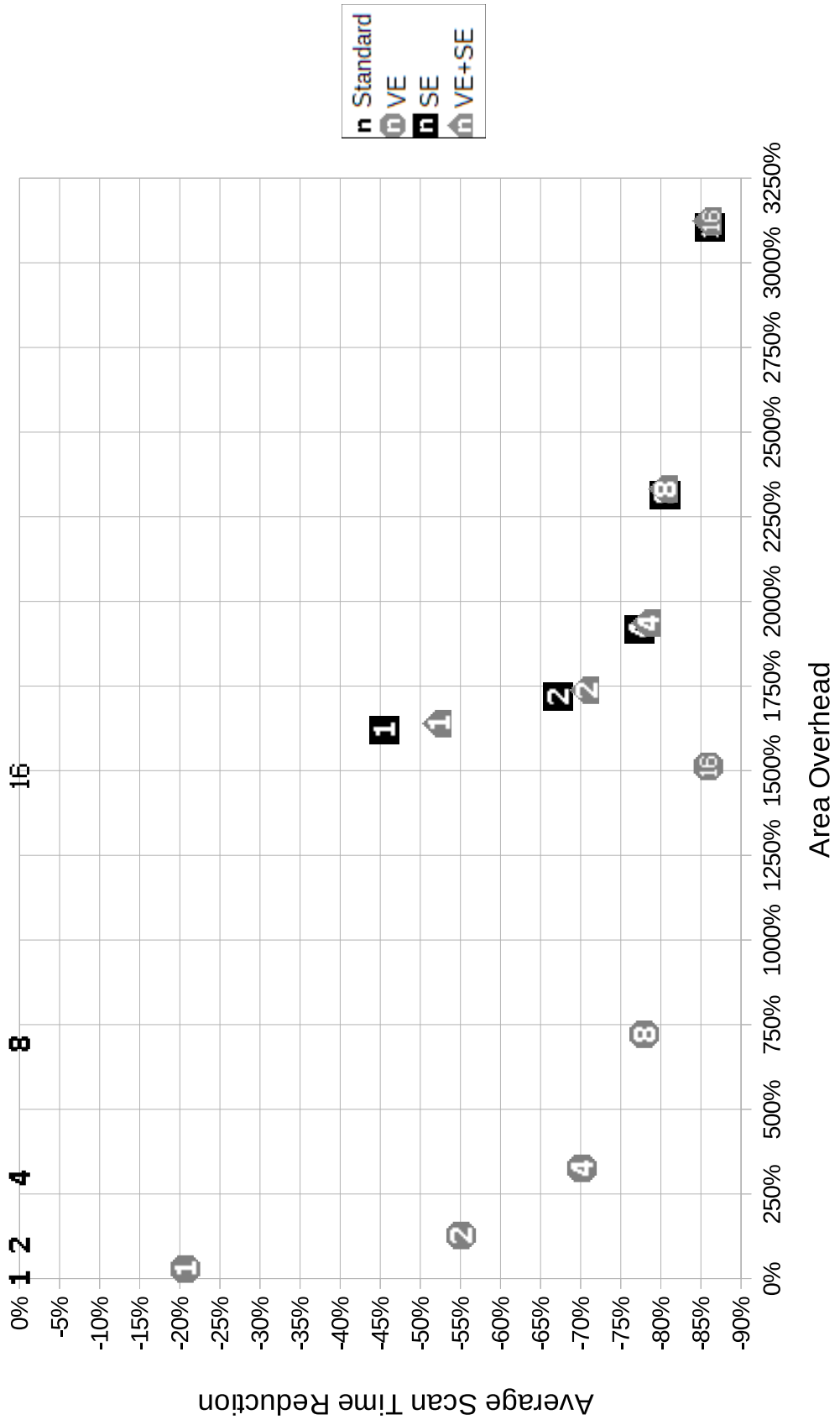
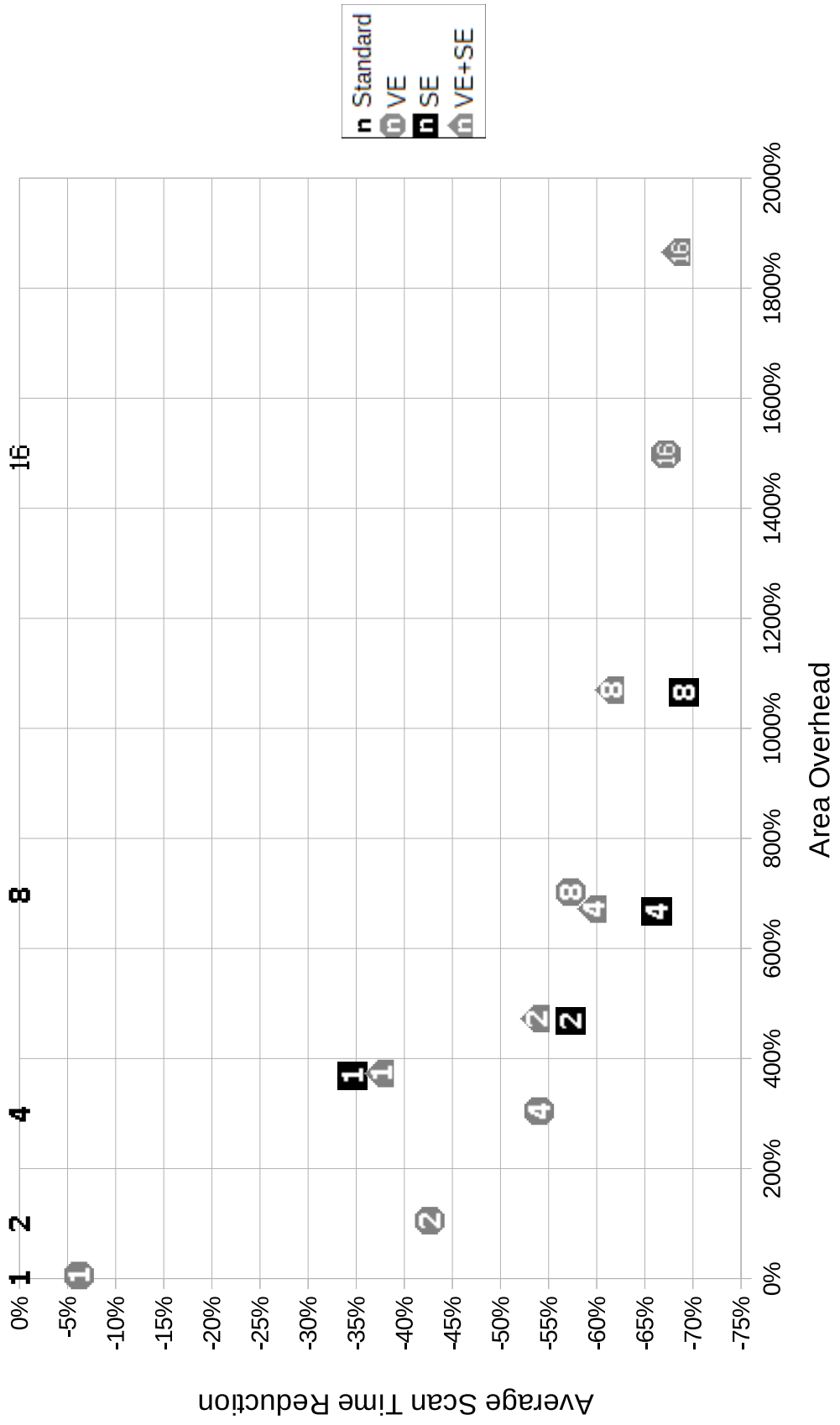


Figure 5.27: Multiple execution units pipeline area vs performance results



6 CONCLUSIONS AND FUTURE WORK

Despite their importance in the industrial automation design space, the performance of PLCs is not a recurrent or widespread subject in current scientific research. Contrary to general-purpose processors, whose performance exponentially grew according to Moore's law in the last decades, the absence of improvements in PLCs' computational power has reduced them to mere controllers, instead of central processing units in automation systems. In an attempt to increase its performance, this work used enhancements for PLC architectures, namely VE and SE, along with different types and quantities of execution units, to boost PLC execution performance based on scan time reduction.

The proposed improvements have different pros and cons, so the applications for them are different. The VE improvement is advised for low-end PLC devices due to its low silicon area impact combined with substantial performance improvement in unbalanced diagrams and monotonous systems. By its superior performance in any diagram but with a high silicon area cost, the SE improvement is recommended for middle-end PLC devices. Lastly, high-end PLC devices should use the VE+SE improvement for its superior performance, justifying its more substantial silicon area cost.

Moreover, independently of the adopted type of execution unit, all proposed improvements in the PLC architecture show superior performance boosts when compared to standard architectures, even using multiple execution units. This experimental conclusion justifies the usage and importance of these improvements to enhance PLC performance, instead of a simple increase in the number of execution units of a standard PLC core.

Besides a study to determine the best SE memoization memory check/miss cycles and size, another future work for this research is the development of an enhanced ranking mechanism for the memoization memory to keep cached the most requested values, thus allowing a reduction of its size and, consequently, of its silicon area usage. Another extension of the present work would be adding the functionality to enable/disable each part of the improved VE+SE core, thus allowing it to benefit from the standalone VE or SE characteristics, according to the developer's option or even on the fly depending on execution workload. Finally, the translation of the developed CAS simulator to real-world hardware in an FPGA, or even as an ASIC, is a future priority task of this research as a step ahead to bring the PLC devices back to their role as central processing units in automation technology contexts.

REFERENCES

BINKERT, N. et al. The gem5 simulator. **ACM SIGARCH Computer Architecture News**, ACM, v. 39, n. 2, p. 1–7, 2011.

BLAKE, G.; DRESLINSKI, R. G.; MUDGE, T. A survey of multicore processors. **IEEE Signal Processing Magazine**, IEEE, v. 26, n. 6, p. 26–37, 2009.

BOLTON, W. **Programmable logic controllers**. [S.l.]: Newnes, 2015.

CHMIEL, M.; HRYNKIEWICZ, E. An idea of event-driven program tasks execution. **IFAC Proceedings Volumes**, Elsevier, v. 42, n. 1, p. 17–22, 2009.

CHMIEL, M. et al. Fpga-based two-processor cpu for plc. In: IEEE. **Signals and Electronic Systems (ICSES), 2016 International Conference on**. [S.l.], 2016. p. 247–252.

CHMIEL, M. et al. An iec 61131-3-based plc implemented by means of an fpga. **Microprocessors and Microsystems**, Elsevier, v. 44, p. 28–37, 2016.

CHMIEL, M.; MOCHA, J.; LECH, A. Implementation of a two-processor cpu for a programmable logic controller designed on fpga chip. In: IEEE. **2018 International Conference on Signals and Electronic Systems (ICSES)**. [S.l.], 2018. p. 13–18.

DENNIS, J. B. Data flow supercomputers. **Computer**, IEEE, n. 11, p. 48–56, 1980.

DU, D.; XU, X.; YAMAZAKI, K. A study on the generation of silicon-based hardware plc by means of the direct conversion of the ladder diagram to circuit design language. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 49, n. 5, p. 615–626, 2010.

DUKA, A.-V.; GENGE, B. Implementation of simon and speck lightweight block ciphers on programmable logic controllers. In: IEEE. **2017 5th International Symposium on Digital Forensic and Security (ISDFS)**. [S.l.], 2017. p. 1–6.

HAJDUK, Z.; TRYBUS, B.; SADOLEWSKI, J. Architecture of fpga embedded multiprocessor programmable controller. **IEEE Transactions on industrial electronics**, IEEE, v. 62, n. 5, p. 2952–2961, 2015.

IQBAL, S.; KHAN, S. A.; KHAN, Z. A. Benchmarking industrial plc & pac: An approach to cost effective industrial automation. In: IEEE. **Open Source Systems and Technologies (ICOSST), 2013 International Conference on**. [S.l.], 2013. p. 141–146.

LEWIS, R. W. **Programming industrial control systems using IEC 1131-3**. [S.l.]: Iet, 1998.

LI, S. et al. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: ACM. **Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture**. [S.l.], 2009. p. 469–480.

LIAO, S.; TJIANG, S.; GUPTA, R. An efficient implementation of reactivity for modeling hardware in the scenic design environment. In: ACM. **Proceedings of the 34th annual Design Automation Conference**. [S.l.], 1997. p. 70–75.

MILIK, A. On hardware synthesis and implementation of plc programs in fpgas. **Microprocessors and Microsystems**, Elsevier, v. 44, p. 2–16, 2016.

MOKHTARNAME, R. et al. Design and implementation of an industrial generalized predictive controller on multivariable processes via programmable logic controllers. In: IEEE. **2015 10th Asian Control Conference (ASCC)**. [S.l.], 2015. p. 1–6.

NAJM, F. N. **Circuit simulation**. [S.l.]: John Wiley & Sons, 2010.

OLUKOTUN, K. et al. The case for a single-chip multiprocessor. In: ACM. **ACM Sigplan Notices**. [S.l.], 1996. v. 31, n. 9, p. 2–11.

PALACHARLA, S.; JOUPPI, N. P.; SMITH, J. E. **Complexity-effective superscalar processors**. [S.l.]: ACM, 1997.

PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization and Design MIPS Edition: The Hardware/Software Interface**. [S.l.]: Newnes, 2013.

PELL, O. et al. Maximum performance computing with dataflow engines. In: **High-Performance Computing Using FPGAs**. [S.l.]: Springer, 2013. p. 747–774.

PLCOPEN. **PLCopen | for efficiency in automation**. 2019. [Online; accessed 18-May-2019]. Available from Internet: <https://www.plcopen.org/>.

SURESH, A. et al. Intercepting functions for memoization: a case study using transcendental functions. **ACM Transactions on Architecture and Code Optimization (TACO)**, ACM, v. 12, n. 2, p. 18, 2015.

WAL, E. van der. Plcopen. **IEEE Industrial Electronics Magazine**, v. 3, n. 4, p. 25, 2009.

WEBB, J. W.; REIS, R. A. **Programmable logic controllers: principles and applications**. [S.l.]: Prentice Hall PTR, 1998.

ZHUANG, H. et al. From circuit theory, simulation to spice diego: A matrix exponential approach for time-domain analysis of large-scale circuits. **IEEE Circuits and Systems Magazine**, IEEE, v. 16, n. 2, p. 16–34, 2016.

APPENDIX A — CLASS DIAGRAMS OF THE CAS

Figure A.1: CAS high level classes diagram

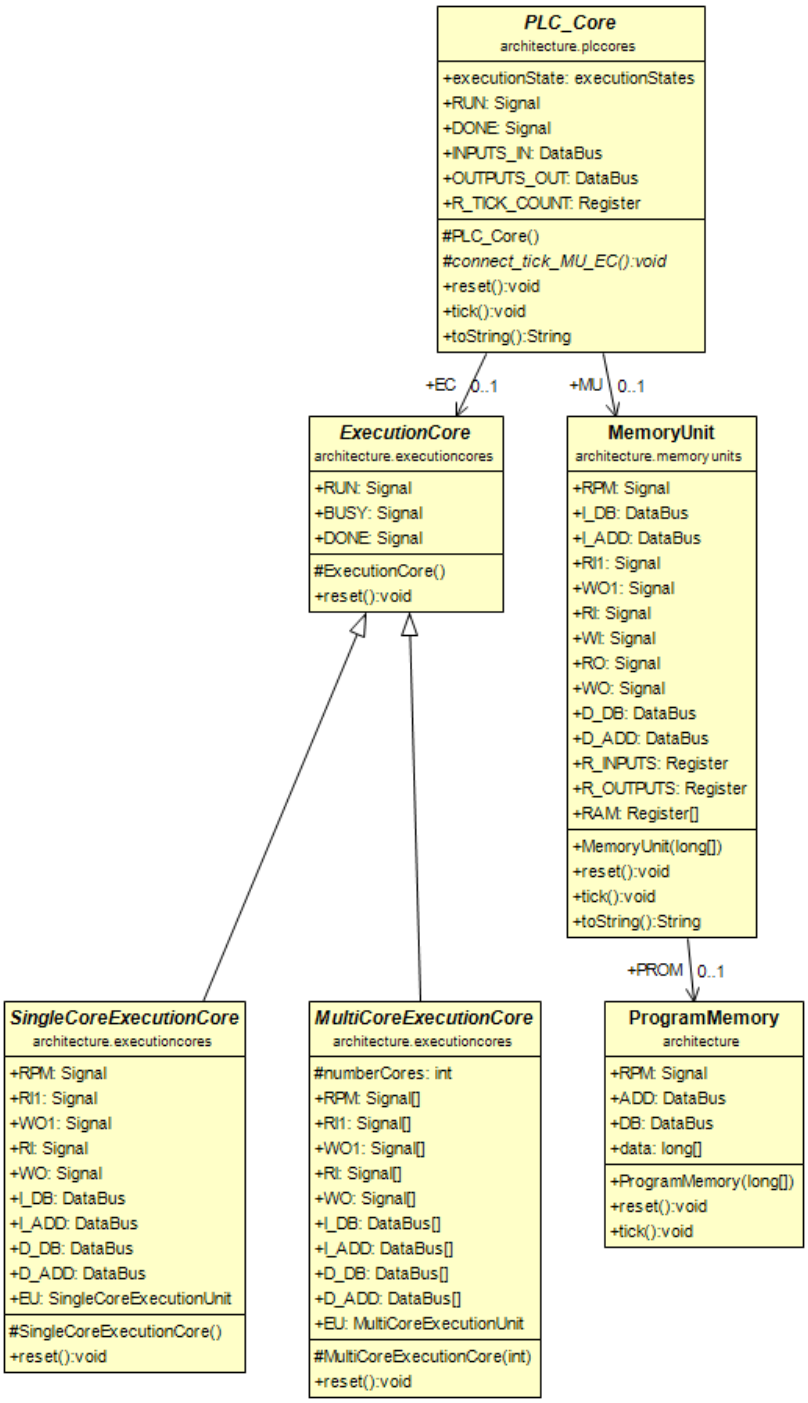


Figure A.2: CAS single execution unit multi-cycle classes diagram

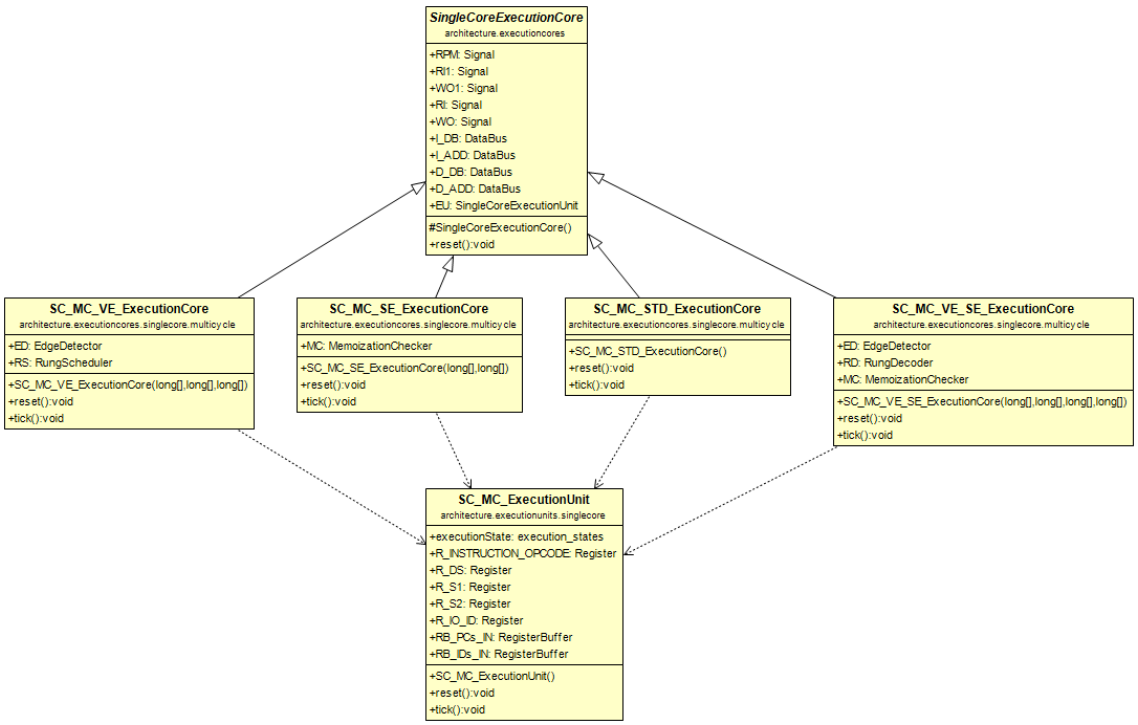


Figure A.3: CAS single execution unit pipeline classes diagram

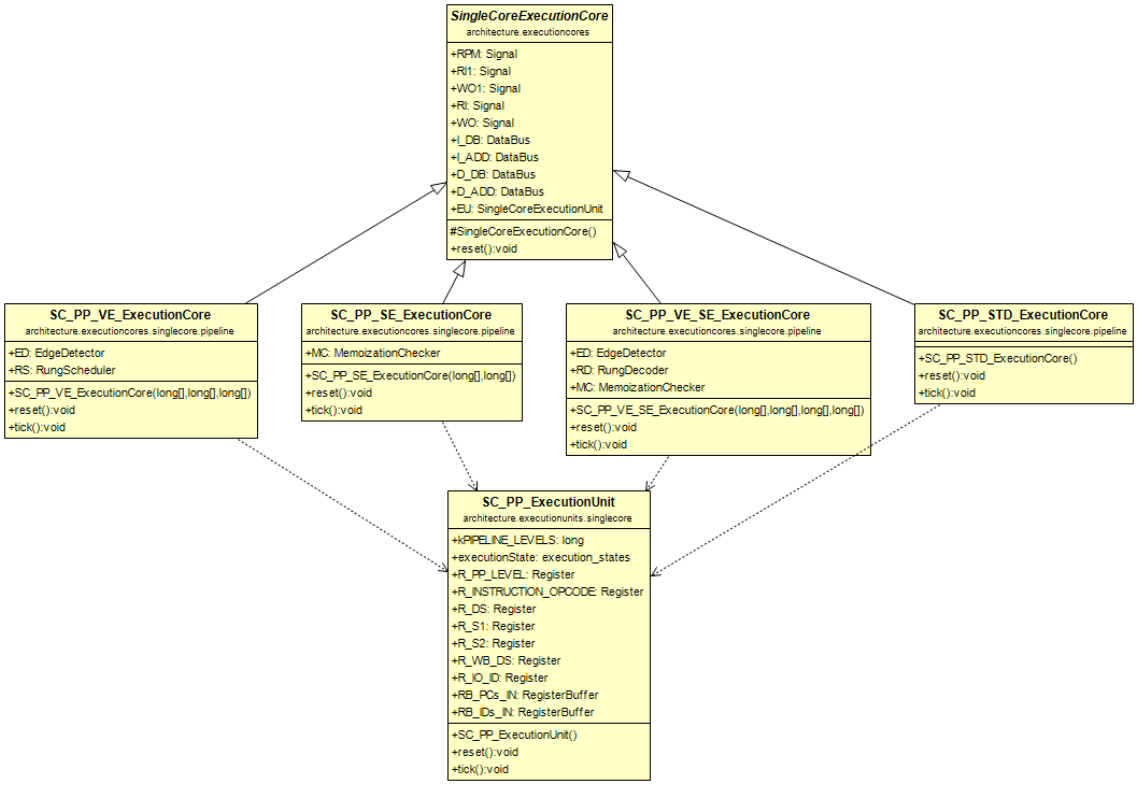


Figure A.4: CAS multiple execution units multi-cycle classes diagram

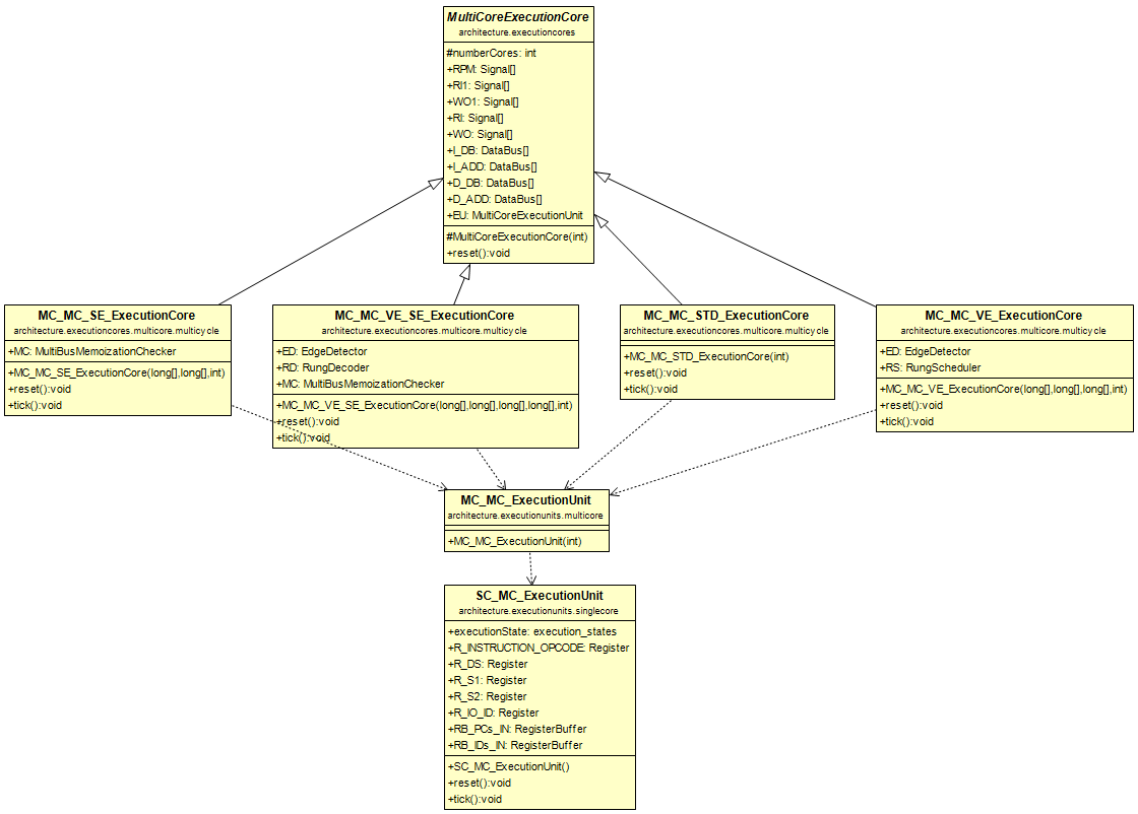


Figure A.5: CAS multiple execution units pipeline classes diagram

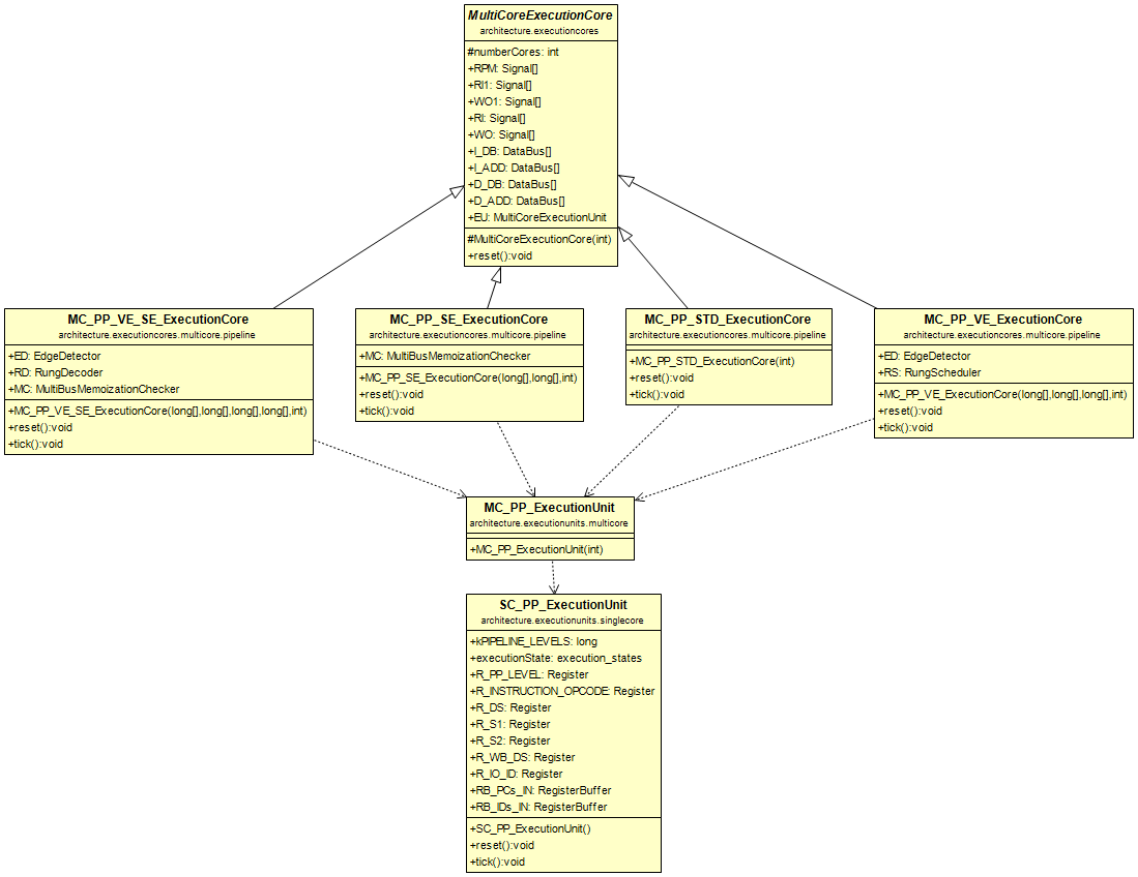


Figure A.6: CAS single execution unit VE improvement classes diagram

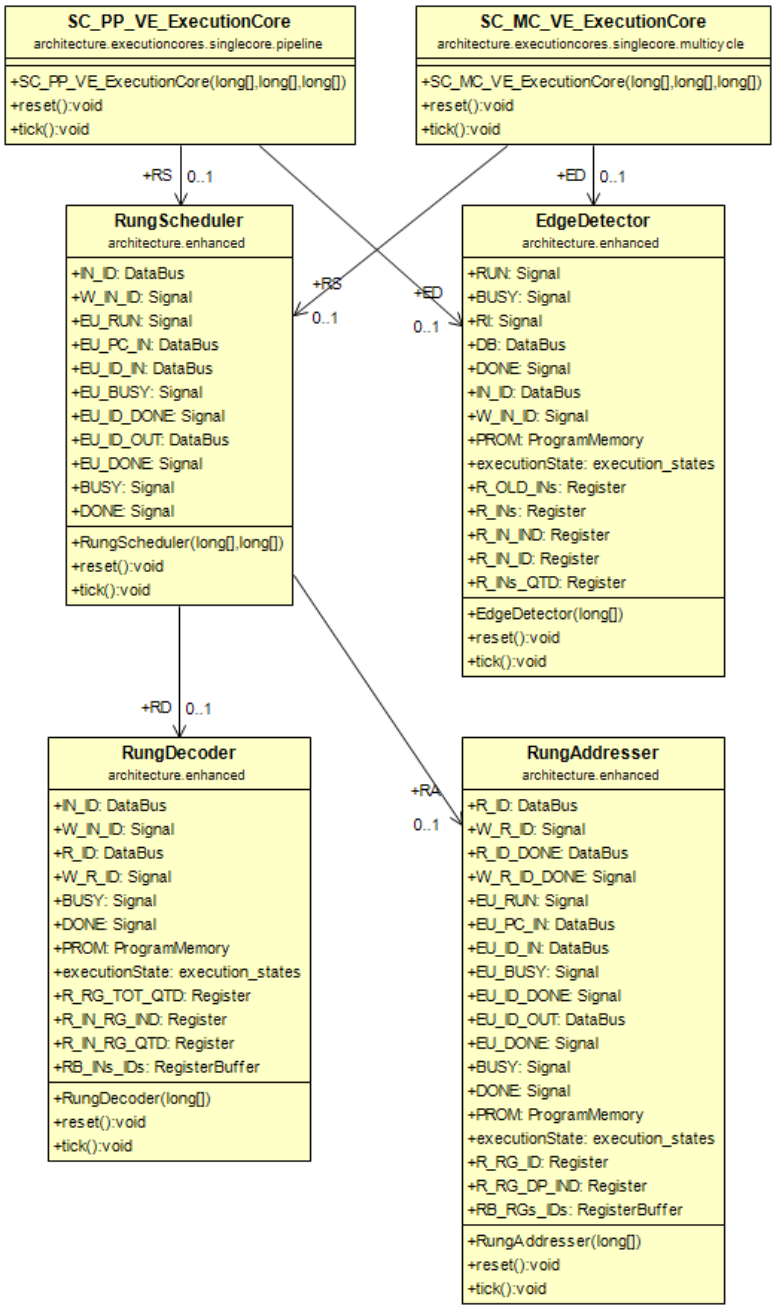


Figure A.7: CAS multiple execution units VE improvement classes diagram

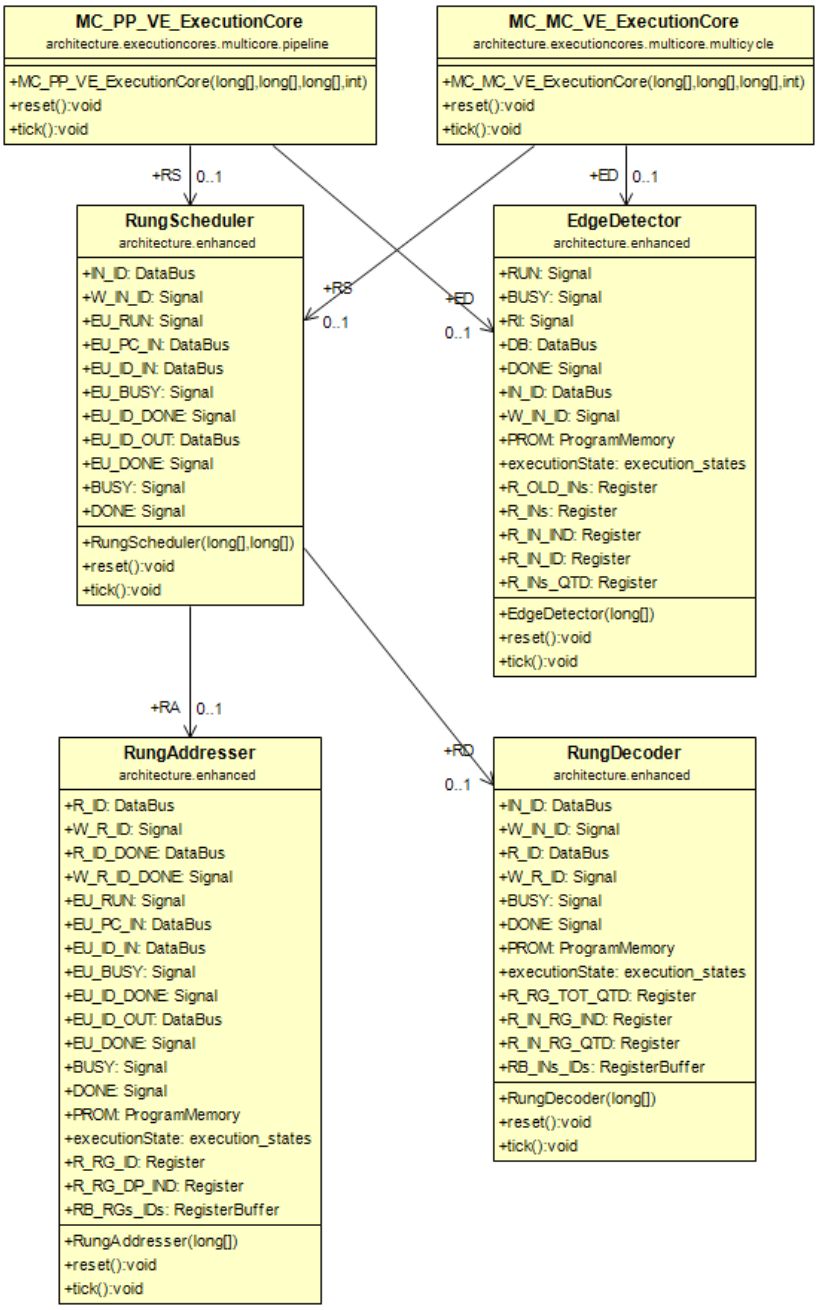


Figure A.8: CAS single execution unit SE improvement classes diagram

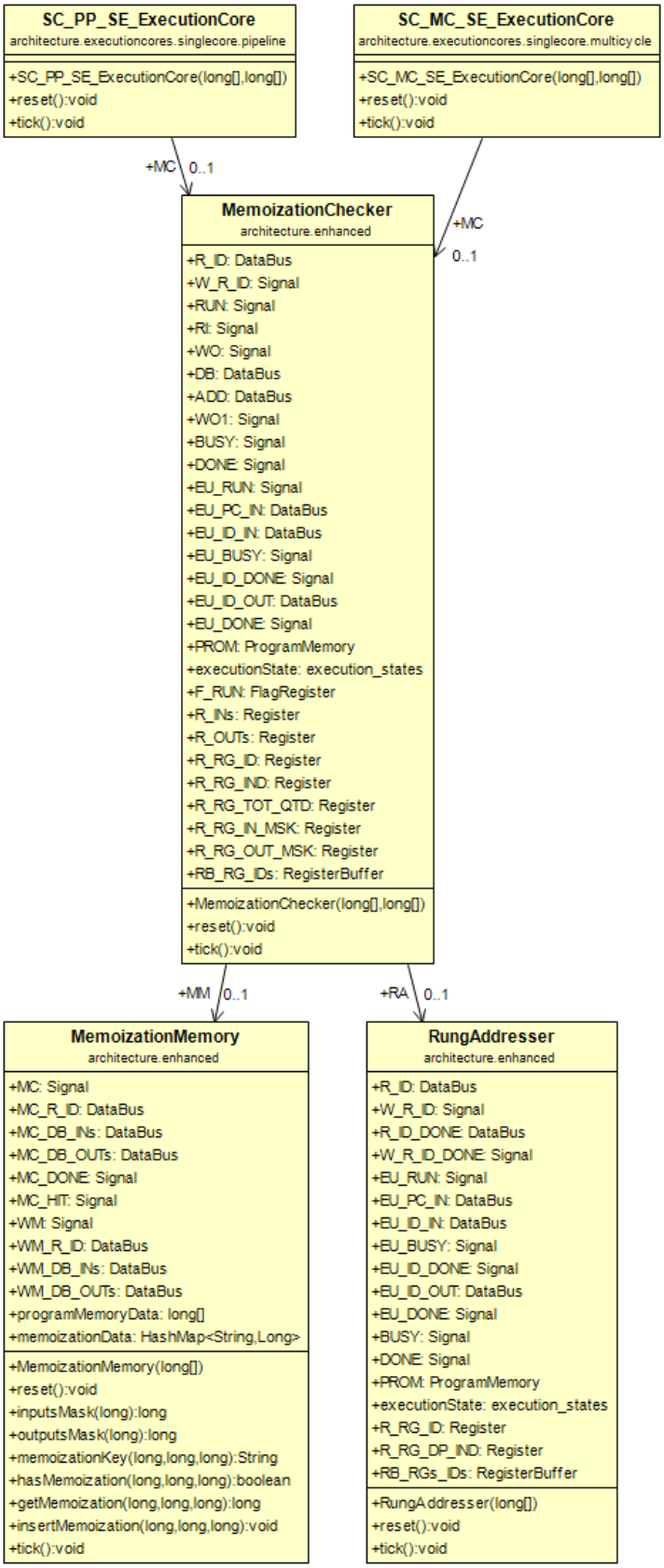


Figure A.9: CAS multiple execution units SE improvement classes diagram

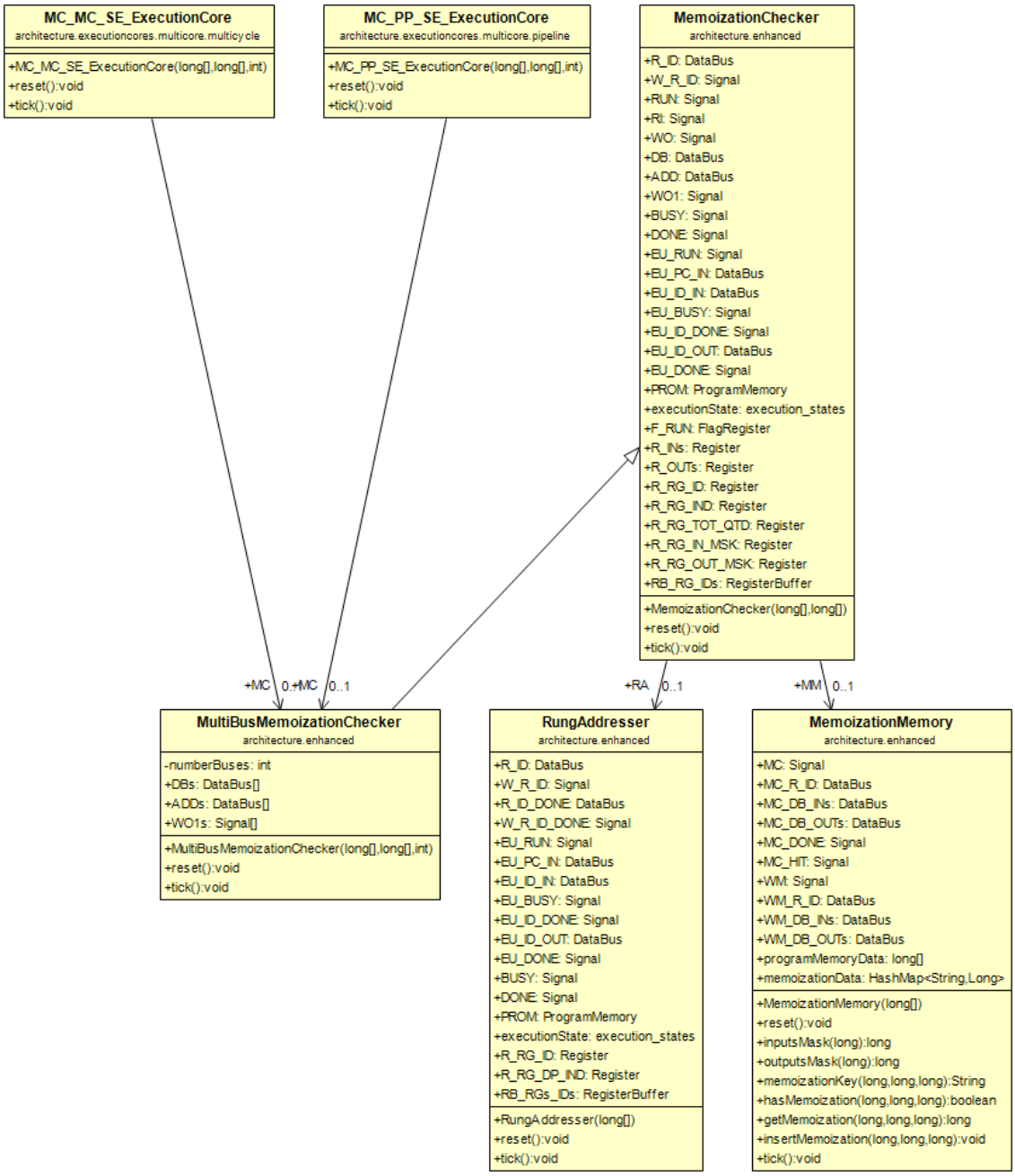


Figure A.10: CAS single execution unit VE+SE improvement classes diagram

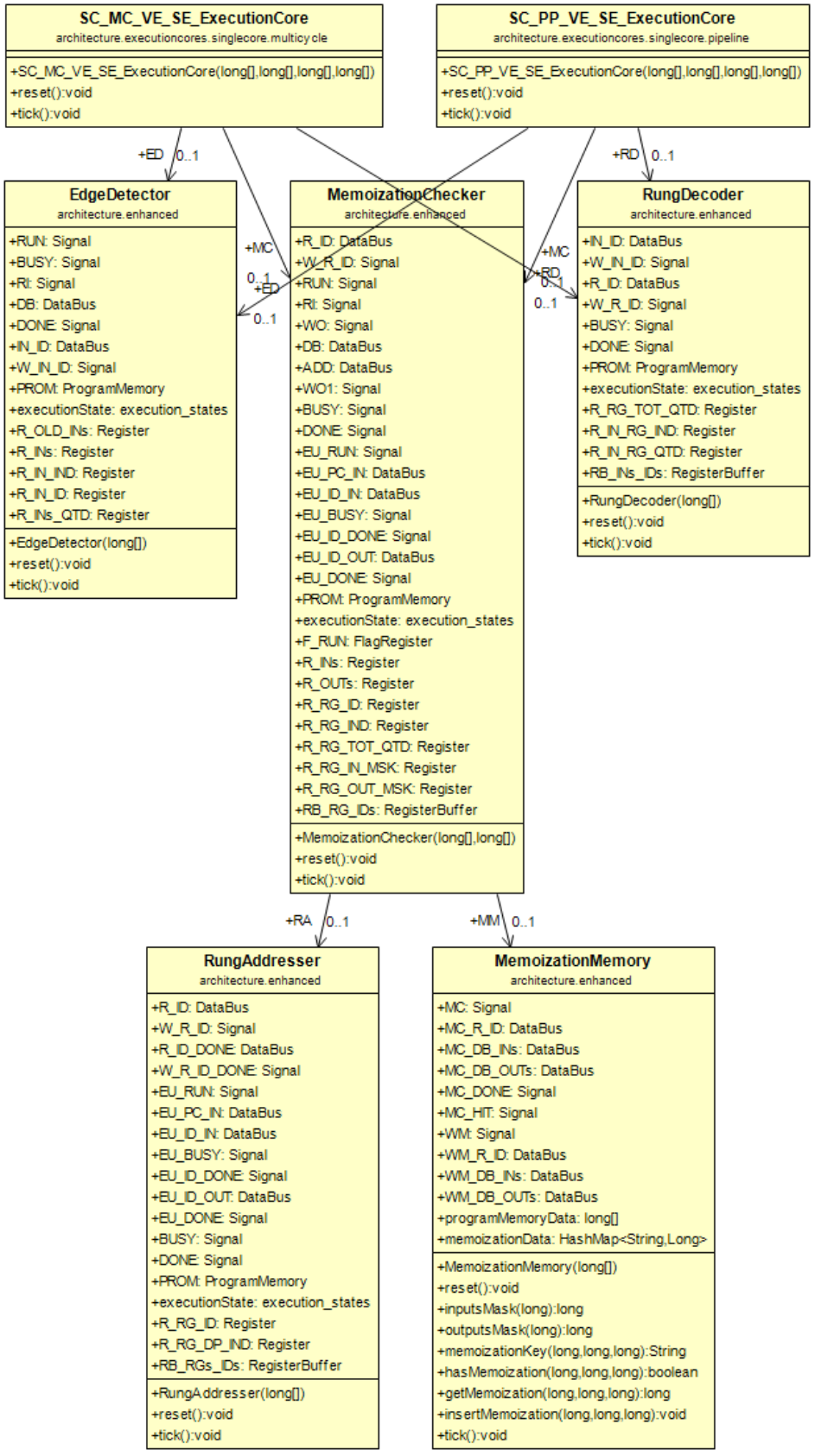


Figure A.11: CAS multiple executions unit VE+SE improvement classes diagram