Using a Polymorphic VLIW Processor to Improve Schedulability and Performance for Mixed-Criticality Systems

Joost Hoozemans, Jeroen van Straten, and Stephan Wong

Computer Engineering Lab, Delft University of Technology, The Netherlands
Email:{j.j.hoozemans, j.vanstraten-1, j.s.s.m.wong}@tudelft.nl

***Abstract*—As embedded systems are faced with ever more demanding workloads and more tasks are being consolidated onto a smaller number of microcontrollers, system designers are faced with opposing requirements of increasing performance while retaining real-time analyzability. For example, one can think of the following performance-enhancing techniques: caches, branch prediction, out-of-order (OoO) superscalar processing, simultaneous multi-threading (SMT). Clearly, there is a need for a platform that can deliver high performance for non-critical tasks and full analyzability and predictability for critical tasks (mixed-criticality systems).**

In this paper, we demonstrate how a polymorphic VLIW processor can satisfy these seemingly contradicting goals by allowing a schedule to dynamically, i.e., at run-time, distribute the computing resources to one or multiple threads. The core provides full performance isolation between threads and can keep multiple task contexts in hardware (virtual processors, similar to SMT) between which it can switch with minimal penalty (a pipeline flush). In this work, we show that this dynamic platform can improve performance over current predictable processors (by a factor of 5 on average using the highest performing configuration), and provides schedulability that is on par with an earlier study that explored the concept of creating a dynamic processor based on a superscalar architecture. Furthermore, we measured a 15% improvement in schedulability over a heterogeneous multi-core platform with an equal number of datapaths. Finally, our VHDL design and tools (including compiler, simulator, libraries etc.) are available for download for the academic community.

I. Introduction

In numerous real-time application domains such as automotive computing, emerging restrictions such as power limitations, cost, size, and maintainability push increasingly more tasks onto a single microcontroller. The timing properties of these tasks can vary in the degree of strictness, giving rise to the field of mixed-critical systems [1]. Platforms executing these types of workloads have the following requirements:

- *Full predictability/analyzability* - It should be possible to perform Worst-Case Execution Time (WCET) analysis to extract tight bounds on the required computation time for each task.
- *Timing isolation* - Running tasks should not be affected by external influences, e.g., other tasks contending for resources, to ensure the timing validity of the system.

978-1-5386-1898-1/17/$31.00

However, they also have the following desirable properties that are prone to contradict with above-mentioned requirements:

- *High-performance* - Tasks should not only be able to meet their own deadlines, but also to allow non-critical tasks to achieve a certain quality of service (for example, car entertainment media playback). Many architectural techniques to improve performance impede predictability and subsequently analyzability. Examples are caches, branch prediction, and out-of-order (OoO) processing.
- *Multicore/Multi-threaded* - Multiple tasks should be able to run concurrently, for reasons of performance and power (a single-core that is powerful enough to run all tasks will require more power than multiple cores with lower clock frequency and voltage), and improved schedulability (see Section V). Multicore platforms often under-utilize processor resources, while multi-threaded platforms such as SMT processors fail to provide timing isolation [2].

In order to meet these requirements, numerous processor designs with predictable timing have been introduced. A recent example is FlexPRET [3], which is a fine-grained multi-threaded processor that provides full isolation and timing predictability. It is similar to a barrel processor but allows more flexibility in assigning cycles to threads in order to allow higher single-thread performance at the cost of adding forwarding paths (that need to distinguish the thread ID of instructions) to the design.

In this paper, we propose to use the ρ-VEX polymorphic VLIW processor for mixed-critical and real-time workloads. Its goal is to provide a high level of flexibility by being able to adapt to different workloads. We show that it has good properties in terms of performance and schedulability in this application domain. The runtime reconfigurable (polymorphic) version can choose to target programs with a high level of ILP in a high-performance single core 8-issue VLIW configuration, or multiple threads/programs in a multicore configuration with smaller issue widths. The key behind this runtime-adaptability is being able to split the processor's data paths into separate cores or combining them into a single larger core. When the processor is split, the independent datapaths provide full isolation from each other. The ρ-VEX provides multi-core

processing capabilities without under-utilizing resources. The VLIW architecture, when used without caches, provides a high degree of time-predictability [4] as it offers static branch prediction (the compiler analyzes the most likely control flow and restructures the code accordingly), in-order execution and an exposed pipeline (all instruction latencies are fixed and known to the compiler - no pipeline interlocking or resource contention). Even when using caches, the ρ-VEX provides full performance isolation between tasks as the caches are split in the same fashion as the core, assuming that the backing bus interconnect provides isolation. Naturally, adding caches will reduce predictability and this paper will therefore not evaluate cached setups. The designer can choose to use local memories and/or enable caches using VHDL generics. We evaluate the benefits of using this hardware platform for real-time workloads in terms of schedulability, throughput, and single-thread performance.

The contributions of this work are:

- We propose to use a polymorphic VLIW processor for real-time and mixed-criticality workloads.
- We perform an evaluation of the processor in terms of schedulability and performance using the Mälardalen real-time benchmark suite.
- We show that the number of randomly generated task sets that can be successfully scheduled on the processor is on par with an earlier study on a proposed dynamic superscalar architecture [5].
- We show that the processor can schedule up to 15% more task sets compared to a heterogeneous multicore processor with an equal number of total datapaths.
- We show that the VLIW processor improves performance over a recently introduced time-predictable RISC-V processor by up to $5.5\times$ for the highest performing configuration.

The remainder of this paper is structured as follows. In Section II, the execution platform is introduced and concepts necessary to understand the work are briefly discussed. Section III continues with the scheduling methodology we propose to use that allows us to exploit the dynamic nature of the processor while still guaranteeing timing isolation between Hard Real-Time Tasks (HRTT). Then we will highlight how it can also assign any cycles not needed by HRTTs to Soft Real-Time Tasks (SRTT). Section IV discusses the evaluation setup, Section V shows the results and Section VII concludes the work.

II. Background

This section briefly discusses concepts from earlier work that are needed to understand this work. First, we will introduce the dynamic processing platform used. Subsequently, we discuss the scheduling framework that we will use to schedule workloads for this dynamic processor.

A. Processing platform

The processor used in this paper is the ρ-VEX dynamically reconfigurable (polymorphic) VLIW processor [6]. We briefly introduce here the main concepts of the processor related to this work. The processor works by creating an 8-issue VLIW core and multiplying the full processor state (consisting of the general-purpose register file and a set of control registers such as the program counter) to create by default 4 'virtual cores' called *contexts*. Between the 8 datapaths and the contexts, an interconnect is added that can be configured at run-time. When running in a single 8-issue mode, all datapaths are connected to one of the four contexts, and when running in 4×2-issue mode, each context is attached to a pair of datapaths (the instruction set architecture requires a minimum of two datapaths to support long immediates). A single pair of datapaths or multiples of these pairs can be re-assigned (i.e., reconfigured) to the contexts without the need to save/restore the contexts to/from main memory. Reconfiguring the interconnection can be performed within 9 cycles (4 cycles during which the new configuration is decoded and the core will continue running in the old configuration, 4 cycles to flush the pipeline and 1 cycle to start the new configuration). Datapaths that are unaffected by the reconfiguration command will continue running without stall cycles. A more in-depth discussion regarding the circuit complexity of this design and the benefits in terms of context switching and (reduced) interrupt latency can be found in [7].

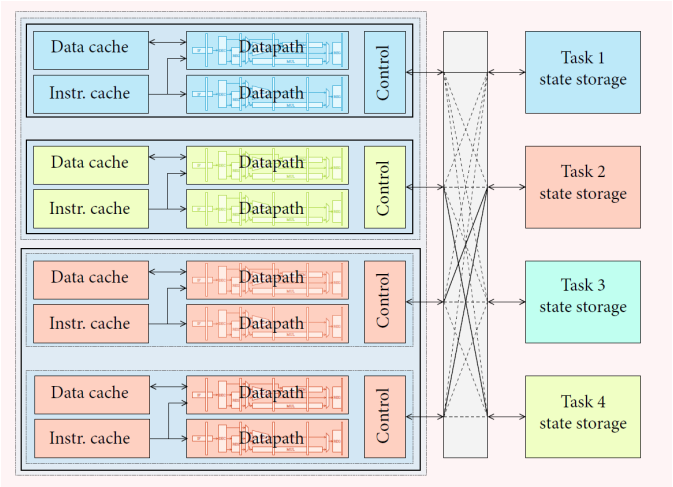


Figure 1. Schematic depiction of the concept behind the ρ-VEX polymorphism: multiple contexts can be connected to the datapaths in different fashions. In this configuration, Task 1 has been assigned a single pair of datapaths (a 2-issue VLIW), task 2 uses 2 pairs (forming a 4-issue VLIW), task 3 is inactive and task 4 has 1 pair of datapaths.

The core has an array of performance counters including cycle, operation, stall, and various cache-related counters. without stall cycles In order to provide high-precision timers, the size of these registers can be configured and are at most 56 bits (resulting in 10 days with single cycle accuracy at 80 MHz before the timer overflows). Using this width, they can be accessed by 2 ordinary load word instructions as the most significant 8 bits of the lower part are identical to the least significant bits of the high part.

The ρ-VEX pipeline has 4 stages by default (see Figure 2) and supports forwarding and variable-length VLIW instruc-

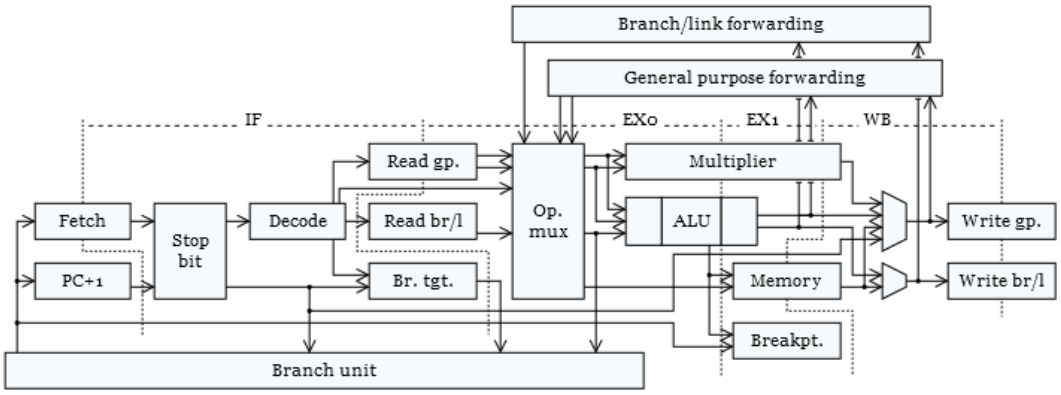


Figure 2. Schematic of a single datapath with default configuration. The VHDL code is highly generic and units can be assigned to a pipeline stage at the designers discretion. The registers between the stages are automatically inserted. [8]

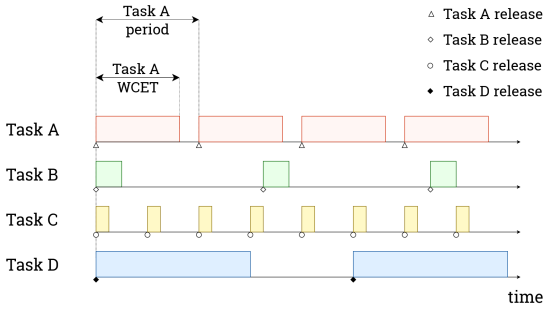
tions. It is also highly configurable at design time using VHDL generics, can be used with or without caches and is synthesizable to ASIC and FPGA targets. It supports up to 8 contexts. Using the default configuration, it can be synthesized at 80 MHz on the Xilinx VC707 evaluation board.

B. Scheduling Methodology for Dynamic Processors

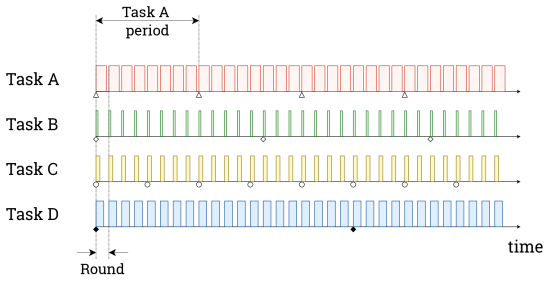
In [5], modifications to an Alpha 21164 processor are proposed to create a dynamically partitionable processor that can run 1 thread in 4-issue in-order superscalar mode, 4 threads in scalar mode, or a combination. The goal for this design is to be able to provide high performance for single threads but also analyzability and timing isolation between threads. Although this work is not directly comparable, being a proposed design without hardware implementation and also targeting the high-performance instead of embedded domain, it does provide us with a very useful scheduling methodology for our dynamic processor. It will be introduced here briefly.

The scheduling framework provides a way to create static schedules for a dynamic processor that supports high frequency reconfigurations. The problem with creating schedules for these processors is that 1) each task in the task set has its own period, so the hyper-period of the task set can become very large, and 2) the processor can be reconfigured at any time, resulting in an infeasible search space when combined with the length of the hyper period. In addition to this (although this problem is not discussed by [5]), a program can have different phases over the course of its execution, in which the Instruction-Level Parallellism (ILP) varies [9]. The impact that changing the issue-width of the processor has on the performance of a program fully depends on the ILP of that program. Therefore, the WCET measurements for different issue widths (see Table II) are only valid if 1) the issue-width stays constant during the entire execution of a program or 2) uniform ILP is assumed (which is not realistic for any non-trivial program).

The main idea behind the scheduling framework in order to mitigate these problems is to divide the schedule into rounds of fixed length and spreading the execution of each task



(a) Task set with WCETs and periods.



(b) Task set spread over rounds.

Figure 3. Figure (a) presents an example task set, Figure (b) shows this set after the tasks have been equally spread over rounds.

evenly over the rounds. To guarantee a single task to meet its deadline, a scheduler must assign CPU time to it equal to $\frac{WCET}{period}$. Similarly, after dividing the period into the fixed-length rounds, our scheduler must guarantee the program that same fraction of each round. At the end of the task's period, it will have executed a number of cycles equal to the WCET, thus guaranteeing the validity of the schedule. In this fashion, we have spread out the executions of all tasks in the task set over their entire period resulting in a common sub-period (the round). Therefore, instead of having to analyze the entire hyper-period, we only have to create a schedule for a round and repeat it indefinitely. In summary, creating a schedule for a workload corresponds to creating a valid schedule for a single

round, resulting in a small search space for reconfigurations. This scheduling method ensures that reconfigurations always occur at the same time within a round and always coincide with a task switch. This way, every task always runs with a constant issue-width during its entire execution. From each task's point of view, it is always running in the same issue width despite of the reconfigurations. Because of this, the WCET for a task using that particular issue width is valid. It is the main point of how the scheduling method allows us to use a dynamic platform to schedule real-time workloads.

III. Approach

In this section, we describe how we use the scheduling framework discussed in Section II-B to create valid static schedules for the dynamic platform discussed in Section II-A. We then proceed to describe how the ρ-VEX can efficiently assign resources not needed by HRTTs to SRTTs in order for them to achieve higher performance.

A. Schedule creation for hard real-time tasks

Creating a valid schedule for a single round involves assigning the required number of cycles and compute resources (datapaths) to each task, drawing from the available pool of cycles and resources. This is equivalent to a 2-dimensional binpacking problem, where the number of cycles is one dimension (time) and the number of datapaths the second (resources). The amount of available cycles, identical to the length of a round, is a designer's choice. In this paper, we have selected 200 cycles, because a number of our benchmarks have runtimes starting from 2000 cycles (see Table II). More realistic scenarios would benefit from longer round lengths to decrease reconfiguration overhead. The amount of available datapaths (issue width) depends on the processor. In case of the ρ-VEX, it is a design-time parameter and can be 2, 4, or 8. The polymorphic core has an issue width of 8 by default. For our evaluations, we will use the polymorphic core and a number of static (fixed) core configurations. The difference from a scheduling point of view, is that fixed cores can only consider WCETs for their particular configuration. The polymorphic core can choose between each of them. This means that if a task has a period that is slightly shorter than the 2-issue WCET, a fixed platform consisting of one or more 2-issue cores cannot meet the deadline. The reconfigurable core can choose to run this task in 8-issue mode, and may therefore be able to schedule this task. Conversely, if a task set has 4 tasks that each have periods slightly longer than the 2-issue WCET, a platform consisting of 4 2-issue cores can meet the deadlines but a single 8-issue core cannot (as the execution time of a task never scales down linearly when increasing the number of issue slots). Again, this task set can be scheduled on the polymorphic core. Because the core can be reconfigured during a round, it is able to schedule task sets that are impossible to schedule on any fixed platform with equal computational resources (see Figure 4 for an example). Conversely, the dynamic platform is able to 'mimic' all possible static setups, so it is able to schedule all task sets that are schedulable on any static setup. In other words, the set of task sets that is schedulable on the dynamic core is a superset of the set that is schedulable on any static platform with equal aggregate resources (datapaths).

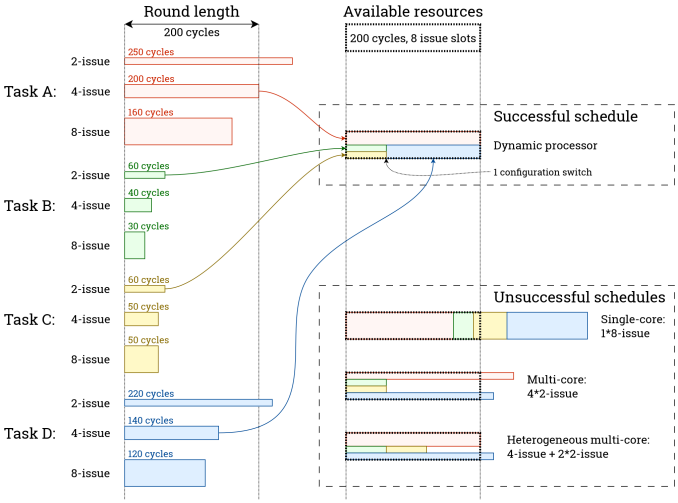


Figure 4. Example of how to create a valid schedule for a dynamic processor.

Activation cycle	Configuration
0	0x2100
60	0x3300

Table I
SCHEDULE TABLE CORRESPONDING TO THE EXAMPLE IN FIGURE 4. THE ρ-VEX CONFIGURATION WORD USES 1 NIBBLE TO ASSIGN A PAIR OF DATAPATHS TO A CONTEXT. IN THE CASE OF 0X2100, CONTEXT 1 AND 2 BOTH RUN ON A 2-ISSUE VLIW AND CONTEXT 0 RUNS ON A 4-ISSUE.

Our solution to the 2-D binpacking problem is implemented as follows. For each task in the task set, a list of two-tuples is created. This list contains an entry for issue widths of 2, 4, and 8; the two-tuples represent the issue width and the corresponding WCET divided by the task period. Only issue widths for which the WCET is smaller than or equal to the task period are included. The tasks are now sorted by area, a common pre-heuristic for binpacking algorithms. The area is defined as follows: $\min_{width}\left(\frac{WCET_{width}}{period}\cdot width\right)$. Before running the binpacking algorithm, a feasibility check is performed by simply testing whether the sum of all the areas is less than or equal to the total computational resources. If a task set is not feasible, it is ignored. The tasks are now packed one by one, by descending area. Packing is first attempted using the narrowest core (in case of the 1x4, 2x2 heterogeneous platform) and using the narrowest issue width compatible with that core. The packing algorithm utilized is bottom-left first (BLF) [10]. If packing fails, wider compatible issue widths are attempted first. If all possible run configurations on the narrowest core in a multi-core system fail, packing is attempted on the next core. If packing fails on all cores in the platform, the task set is considered to not be schedulable on that platform.

A successful schedule consist of a mapping between datapaths and contexts. From the scheduler's point of view, each context runs a task, but it is up to the designer to execute multiple tasks on a single context (he will have to add the WCET of a task save/restore). Depending on the task graph, the mapping can change during a round. This is not only true for the dynamic platform; when there are more tasks than processors, context switches are also required on static platforms. The mappings are stored in a static schedule table, that contains an entry for every change in mapping during the round. Each entry must be activated at a certain cycle during the round. For the ρ-VEX, this can be performed by a state machine that triggers a reconfiguration at certain time values, but it is also possible to program a timer to trigger the runtime scheduler. The runtime scheduler will read the next entry from the table, activate the corresponding configuration and reprogram the timer for the next entry. In this case, the WCET of the scheduler must be added to the schedule for every change in mapping. Table I depicts an example of a schedule table corresponding to the scheduling example of Figure 4.

B. Mixed-criticality: Assigning unallocated cycles to soft real-time tasks

What makes this platform especially suitable for mixed-critical systems, is that any space not consumed by critical tasks can be assigned to non-critical tasks in an efficient way, as will be discussed in this section. To make use of the slack in the schedule, the designer can assign SRTTs to one or more context(s). These context can be added into the static schedule after the scheduling method described in the previous section has guaranteed the required execution time for the HRTTs. However, there will usually be much more free cycles because of the overestimation of the WCET (the actual execution time can be lower than the worst case, sometimes a lot lower if it depends on the input). The ρ-VEX is also able to utilize these cycles as follows. As soon as a HRTT finishes, it will request the scheduler to give its resources to one of the SRTT contexts as depicted in Figure 5. Naturally, the scheduler must ensure that tasks cannot take away resources from other tasks. The scheduler will write this change in the currently active static schedule table. From that moment, the portion of resources that was reserved for that task in each round will be given to the SRTT. When as the original HRTT is triggered, the interrupt routine will restore the original schedule, thereby again guaranteeing the tasks execution time. This means that the HRTT is still fully isolated. Multiple SRTTs can be scheduled on a single context using any classical scheduling algorithm, taking into account that these switches will require a classical save/restore penalty.

Depending on how much resources are left and how many threads are active, the core can run these other tasks in either high-performance 8-issue mode or high-throughput multicore mode, unlike other analyzable processors such as [3] that is limited to scalar execution. This is one of the key points of the ρ-VEX processor - it is able to provide timing isolation,

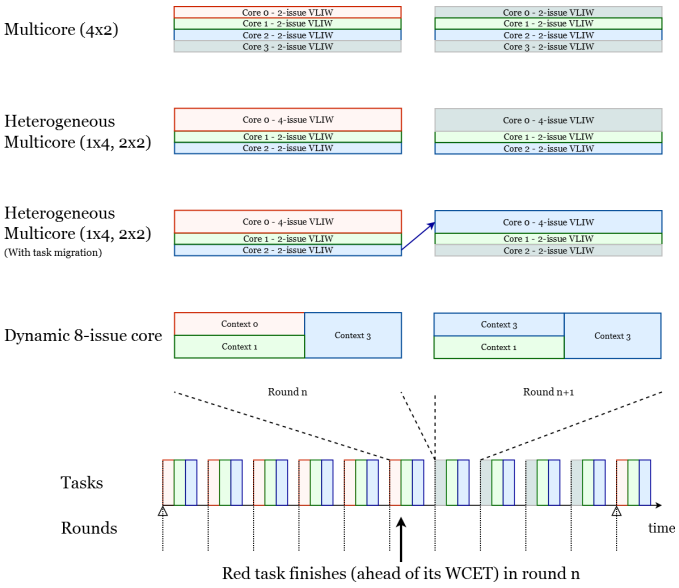


Figure 5. A Hard Real-Time Task (red) giving its resources to a Soft Real-Time Task when it has finished. When the original task is triggered again, it will take its resources back. Static multicore systems are not able to utilize all resources. A heterogeneous multicore can achieve a higher utilization by migrating a task to a larger core when it becomes available. However, this needs state saving and restoring (not needed by the dynamic core).

in addition to exploiting its dynamic properties to adapt to the characteristics of the workload (Instruction-level or Thread-level parallelism as discussed in [8]). When using a heterogeneous multicore system, a task can be migrated from a little core to a big core when it becomes available to increase resource utilization as well (depicted in Figure 5). However, when the HRTT is triggered, the core must first save the state of that other task again before the HRTT can resume. This penalty must be added to the WCET and is not required by the dynamic core.

The ρ-VEX platform can be configured with caches and DDR memory along with memory mapped single-cycle local memories (scratchpads) for both instruction and data, so that critical applications can use the scratchpads and non-critical applications will utilize main memory through the caches. Memory protection (spatial isolation) can be provided using the experimental memory management unit. However, adding this component to the design increases the cycle time considerably, therefore we propose a simpler scheme where accesses to local memories that are not assigned to the corresponding context generate a trap (in the same fashion that the ρ-VEX generates a trap when accessing unmapped addresses). The operating system's trap handler can then try to recover from this error or terminate/restart the offending program, while the other context (that would otherwise be victimized by the access – either because a write changes the program state or because arbitration for the memory port causes a stall cycle) continues unaffected.

IV. Experimental Setup

For the measurements, we are using a cycle-level architectural simulator for flexibility, and base our results on an FPGA prototype of the ρ-VEX processor clocked at 80 MHz running on a Xilinx VC707 development board. We will use 23 programs from the Mälardalen benchmark suite [11]. Execution times, listed in Table II, were measured on a single core for each of the 3 possible configurations, in a platform without caches and using single-cycle memories (implemented by FPGA RAM blocks). These execution times are assumed to be the worst-case execution times, as they are always executed using the same input. However, standard WCET analysis or measurement techniques can be applied [4].

Table II
WCETs of the benchmarks for the possible processor configurations (2-issue, 4-issue, 8-issue).

	Worst-Case Execution Time (cycles)		
Benchmark	2-issue	4-issue	8-issue
adpcm	350009	315107	309206
cnt	3626	2937	2664
compress	5236	4479	4241
cover	2210	2006	1815
crc	17879	14985	14667
edn	23184	18581	17505
expint	10800	9926	9512
fft1	50389	34061	26614
fir	183221	139559	129815
lms	4986376	3372138	2709911
ludcmp	55388	41112	34928
matmult	93211	85882	84649
minver	19289	13406	10844
ndes	31120	26502	24457
ns	5212	4253	4116
nsichneu	6357	6330	6316
prime	25788	23170	23158
qsort-exam	2995	2241	1922
qurt	21211	14494	11591
sqrt	19587	13877	11574
st	3631939	2313149	1731382
ud	11579	10720	10420
whet	29519872	18704428	13694591

The programs were compiled using our port of the Open64 compiler using optimization level 3. Programs were run bare-metal, with our port of the newlib embedded standard C library. Our UART driver was modified so it did not wait when the output buffer is full, in order to remove the influence of serial output. Output is written into a reserved memory region so that it can still be examined after execution.

In order to measure the 'schedulability' (how many task sets can be successfully scheduled) of the processor and compare to [5], we have implemented a program that mimics the task set generator according to their description. We will briefly describe it here for clarity. We will generate task sets consisting of 4 tasks randomly selected from the benchmark set. For each of the tasks, a period is randomly chosen using the following constraints: $WCET_{8issue} \leq period < (1.5 \times n_{tasks}) \times WCET_{2issue}$. These boundaries guarantee any single task to be schedulable on a single 8-issue core (highest performing processor in our design space), and that a sufficient number of schedules are generated that are schedulable on a single 2-issue core (lowest performing processor in our design space).

The task set are divided into 4 bins of varying 'difficulty'; every set is categorized in terms of total 2-issue utilization (the sum of the 2-issue utilization of each task in the set; $\sum_{task} \frac{WCET2_{task}}{period_{task}}$). Bin 0 contains the lightest task set with a total utilization of 0 - 1, and bin 3 has the most difficult task sets with highest utilizations (between 3 and 4). Each task set is randomly generated and assigned to its corresponding bin until each bin has 2500 task sets. In [5], the bins consist of 25 task sets each. However, using that size, we found the results to vary significantly between runs. Therefore, we increased it to get more consistent outcomes.

We will consider a number of platforms in our evaluation. Most of these will have equal aggregate resources (i.e., 8 datapaths in total), to demonstrate the effectiveness of the dynamic processor in utilizing these resources. The exception is the 1x2-issue platform, to match the "scalar processor" in [5].

V. Results

This section presents the measurement results in two evaluation metrics: schedulability and performance.

A. Schedulability

Figure 6 plots the number of task sets that can be successfully scheduled on the different platforms. The four groups on the x-axis each list the results for a certain schedule bin, from 0 (task sets with lowest total utilization) to 3 (task sets with highest total utilization). Every result is relative to the number of feasible schedules in the bin (which is plotted in Figure 7).

A single 2-issue core can, by definition, only schedule tasks from bin 0 (total 2-issue utilization must be < 1). This can be clearly seen in the graph, where all other bins have 0 successful schedules for that platform. As the difficulty increases, the advantage of using the dynamic processor becomes clear; in bin 2 it is able to schedule 97% of the feasible schedules and in bin 3 50%. The homogeneous multi-core platforms can schedule considerably smaller numbers of task sets, with the 4x2-issue being able to accommodate only 10% of the task sets from bin 3. These sets consist of tasks with periods that are close to, but not shorter than, the 2-issue execution time. If there is a single task in the set that requires a larger core to meet the deadline, this platform cannot schedule it. The absence of the 2x4-issue platform is due to the usually small difference between 2-issue and 4-issue execution times (see Table II). For this platform, running the programs in 4-issue mode is the only choice, resulting in more total 'area' utilization for a task, even if it does not need the additional performance. The same applies to the 1x8-issue platform, but the effect is even more pronounced. The heterogeneous 1x4,2x2-issue platform provides a very adequate schedulability, with the dynamic processor beating it by only 15%. This means that, if the higher single-thread performance that the 8-issue dynamic core can deliver is not needed, a heterogeneous platform is a good alternative for designers to consider.

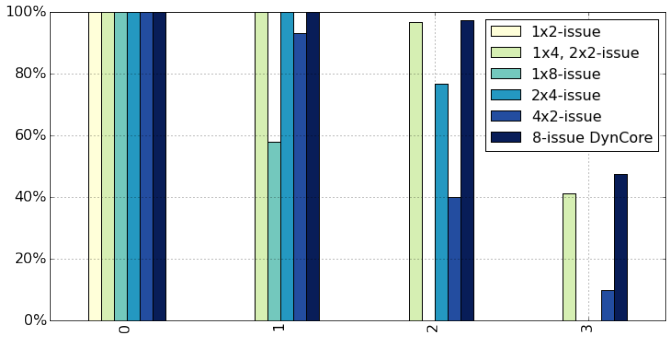


Figure 6. Plot of the number of successful schedulings for each of the evaluated hardware platforms. On the x-axis are the 4 bins with increasing total utilization. Results are relative to the number of feasible task sets in the bins (infeasible task sets are ignored).

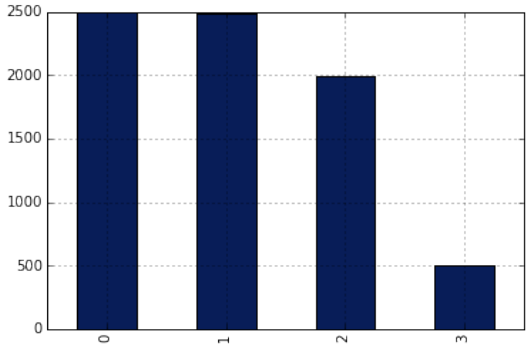


Figure 7. Plot of the number of feasible task sets for each of the task set bins.

The number of feasible task sets per bin is plotted in Figure 7. It drops as the difficulty of the task set bin increases to 80% for bin 2 and 20% for bin 3. Of these 500 feasible task sets in bin 3, the dynamic core is able to schedule around 50%, which could indicate that 1) there is room for improvement in the scheduling framework and binpacking algorithm, or 2) the requirement that every individual task must always run using a constant issue-width (see Section II) could be a limiting factor.

Comparing to the results from [5], plotted in Figure 8, we see a similar curve over the task set bins, but our scheduler performs somewhat better for the dynamic processor. In bin 3, it can schedule almost twice the number of task sets at 50% vs. 28%. An equivalent of the heterogeneous 1x4, 2x2 platform has not been evaluated so we cannot make a comparison. Figure 9 shows a different representation of the schedulability results, plotting the fraction of successful schedulings in relation to the total system utilization. We can see that as the number of tasks increases, the advantage of the dynamic core becomes more pronounced.

B. Performance & Area Utilization

To evaluate the performance of the ρ-VEX processor, a comparison was performed with a 32-bit RISC-V processor on which the FlexPRET time-predictable processor [3] is based.

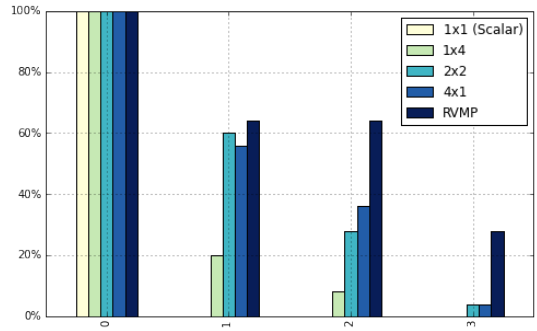


Figure 8. Schedulability results from [5], to be compared with Figure 6. RVMP (Real-time Virtual MultiProcessor) should be compared to dyncore.

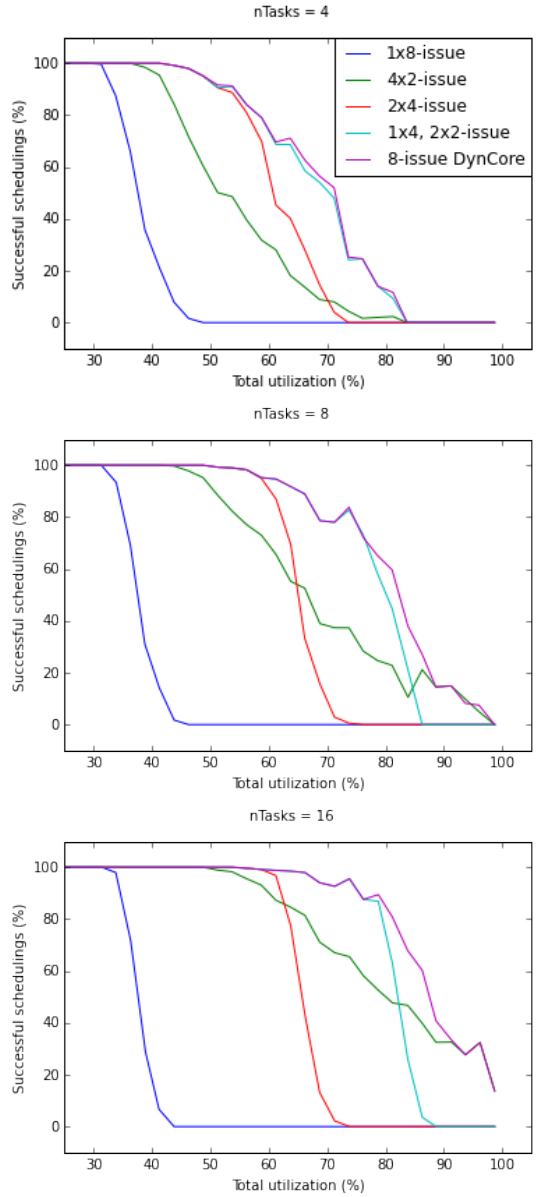


Figure 9. Schedulability plotted in relation to total system utilization.

Table III
PERFORMANCE - RISC-V (RV32I) VS ρ-VEX.

Benchmark	RISC-V	ρ-VEX speedup over RISC-V (factor) 2-issue	4-issue	8-issue
adpcm	1732860	4.95	5.50	5.60
cnt	9554	2.63	3.25	3.59
compress	6917	1.32	1.54	1.63
cover	1808	0.82	0.90	1.00
crc	21633	1.21	1.44	1.47
edn	818203	35.29	44.03	46.74
expint	6726	0.62	0.68	0.71
fir	956526	5.22	6.85	7.37
lms	19926799	4.00	5.91	7.35
ludcmp	194575	3.51	4.73	5.57
matmult	682965	7.33	7.95	8.07
minver	43312	2.25	3.23	3.99
ndes	32785	1.05	1.24	1.34
nsichneu	4260	0.67	0.67	0.67
prime	74616	2.89	3.22	3.22
qsort-exam	10283	3.43	4.59	5.35
qurt	92343	4.35	6.37	7.97
st	15210054	4.19	6.58	8.78
ud	4120	0.36	0.38	0.40
whet	49287174	1.67	2.64	3.60
Weighted avg. speedup	N.A.	2.27	3.52	4.69
Avg. speedup	N.A.	3.95	4.99	5.54

As we are only measuring performance, the timing extensions are not needed. Measurements are based on cycle counts from the spike simulator executing the RV32I instruction set. All benchmarks are compiled using optimization level 3. The speedups are calculated assuming a target clock frequency of 80MHz for both processors. It must be noted, though, that the RISC-V will likely be able to achieve higher clock frequencies. Also note that there are 3 benchmarks that have been removed from the results (fft1, ns and sqrt). This is because the RISC-V compiler completely optimized them away, resulting in a program that only returns the answer.

When weighted to execution time, the highest performing 8-issue ρ-VEX is $4.69\times$ faster than the RISC-V. As this benchmark suite is being dominated by the execution time of *whet*, we also report the non-weighted average of $5.54\times$ speedup. Table IV lists the FPGA utilization for the two processors when prototyped on a FPGA. The utilization of Block RAMs is relatively high in case of the ρ-VEX, because the multi-ported register file implementation requires duplicated storage combined with a Live Value Table [12]. Compared to the FlexPRET, the ρ-VEX utilizes approximately 5 times more resources. As can be seen in Table III, this is similar to the increase in performance. This compares quite favorably as single-thread performance normally does not scale linearly with area utilization.

The size of the ρ-VEX, however, is a factor that designers will need to consider as an 8-issue VLIW will be overkill for many application scenarios. However, for some domains such as media or digital signal processing, VLIWs are known to provide significant performance gains over scalar RISC processors, therefore, in these cases the ρ-VEX is a suitable platform.

Table IV
RESOURCE USAGE OF ρ-VEX VS. THE FLEXPRET TIMING-ANALYZABLE MULTI-THREADED PROCESSOR

	ρ-VEX (4-threads)	FlexPRET (4-threads)	Increase (factor)
Slice Registers	8529	2687	4.72
Slice LUTs	31839	5661	5.6
BRAMs	128	Not reported	N.A.

VI. RELATED WORK

This work touches upon the subjects of (static real-time) schedulability, multi-threaded architectures, and time-predictable processors. In [13] and [14], predictability and schedulability is discussed. Examples of processors with multiple contexts/threads for the purpose of real-time systems are [15] [16]. In [17], performance comparisons are made between increasing the number of cores and increasing the number of register sets. A VLIW architecture with multiple hardware contexts is the Itanium [18]. There, it is used to increase throughput by using SMT. This is not directly comparable to the ρ-VEX used in this work as it targets the high-performance instead of embedded domain, and furthermore it does not provide performance isolation. An SMT architecture with bounded performance interference is proposed in [2]. The multiple contexts of the ρ-VEX are discussed in [7]. In [5], modifications to an Alpha 211164 are proposed that makes use of multiple isolated multi-threaded contexts (virtual processors) to improve static real-time schedulability. This work is the most closely related to ours and is discussed more in-depth in Section II. The contributions we provide over this work is the use of a platform that is implemented in VHDL and available for download for the academic community, and which targets the embedded instead of high-performance domain (Virtex 6 and 7 boards are supported). Although it is not discussed by [5], the scheduling method has some similarities to p-fair scheduling [19]. In particular, p-fair also relies on rounds to assign resources to tasks. However, in p-fair scheduling, a round is atomic regarding resource assignment (a resource can only be assigned to a task for a full round). The scheduling method used in this work divides the resources *within* a round in order to reduce the search space of a dynamic processor that can change at any time during runtime.

In the realm of time-predictable processors, the most notable example is arguably the recently introduced FlexPRET softcore [3] that is also available for download and can be prototyped on an FPGA. Instead of assigning execution lanes to threads, it can assign cycles to threads in a fine-grained multi-threaded fashion. By assigning a larger fraction of the threads, it can also throttle the performance of a single thread. The advantage of the ρ-VEX is that it is a VLIW architecture that can provide high performance over the scalar FlexPRET (see Table III), particularly in certain embedded domains such as signal processing or media applications. See [4] for a study about time-predictability of VLIWs and their compilers.

VII. Conclusions

This paper introduces the ρ-VEX polymorphic processor to the field of real-time and mixed-criticality systems. We showed that it can exploit its dynamic properties to 1) improve schedulability over fixed execution platforms, while still providing execution time guarantees when using a round-based scheduling methodology, and 2) efficiently assign resources to lower-priority threads when high-priority threads finish ahead of their WCET. Due to the 8-issue VLIW architecture, it can also provide significant performance gains compared to scalar RISC architectures such as time-predictable RISC-V processors. The nature of VLIW architectures provides a high degree of predictability as it uses static branch prediction and an exposed pipeline. This makes it possible to establish relatively tight WCET bounds, either using measurements (if it is possible to fabricate an input that activates the longest execution path) or using static analysis techniques. These advantages make it a suitable platform for mixed-criticality systems, especially when the workload contains media and/or signal processing applications. The cost of increased area utilization is a trade-off that designers must make when choosing an execution platform. Keep in mind that, to achieve full predictability in a complete system, a predictable interconnect (and main memory system, if applicable) must be used such as [20]. when using caches instead of local memories, the system still provides performance isolation because the caches are split in the same fashion as the datapaths, but the predictability is severely impacted (one would need to assume that every cache access results in a miss). The ρ-VEX comes with VHDL code, toolchain (consisting of multiple compilers, binutils, newlib, etc.), a fast architectural simulator, extensive debug hardware and interface tools. It can be downloaded for academic use at www.rvex.ewi.tudelft.nl.

Acknowledgements

This work has been supported by the ALMARVI European Artemis project nr. 621439.

References

[1] S. Baruah, H. Li, and L. Stougie, "Towards the Design of Certifiable Mixed-criticality Systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2010, pp. 13–22.

[2] M. Paolieri, J. Mische, S. Metzlaff, M. Gerdes, E. Quiñones, S. Uhrig, T. Ungerer, and F. J. Cazorla, "A Hard Real-time Capable Multi-core SMT Processor," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 3, pp. 79:1–79:26, Apr. 2013. [Online]. Available: http://doi.acm.org/10.1145/2442116.2442129

[3] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A Processor Platform for Mixed-criticality Systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014, pp. 101–110.

[4] J. Yan and W. Zhang, "A Time-predictable VLIW Processor and its Compiler Support," *Real-Time Systems*, vol. 38, no. 1, pp. 67–84, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11241-007-9030-5

[5] A. El-Haj-Mahmoud, A. S. Al-Zawawi, A. Anantaraman, and E. Rotenberg, "Virtual Multiprocessor: an Analyzable, High-performance Architecture for Real-time Computing," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2005, pp. 213–224.

[6] S. Wong and F. Anjam, "The Delft Reconfigurable VLIW Processor," in *Proc. 17th International Conference on Advanced Computing and Communications*, Bangalore, India, December 2009, pp. 244–251.

[7] J. Hoozemans, J. Johansen, J. V. Straten, A. Brandon, and S. Wong, "Multiple Contexts in a Multi-ported VLIW Register File Implementation," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–6.

[8] A. Brandon, J. Hoozemans, J. van Straten, and S. Wong, *Exploring ILP and TLP on a Polymorphic VLIW Processor*. Cham: Springer International Publishing, 2017, pp. 177–189. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-54999-6_14

[9] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and Exploiting Program Phases," *IEEE micro*, vol. 23, no. 6, pp. 84–93, 2003.

[10] B. Chazelle, "The Bottomn-Left Bin-Packing Heuristic: An Efficient Implementation," *IEEE Transactions on Computers*, vol. C-32, no. 8, pp. 697–707, Aug 1983.

[11] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[12] C. LaForest and J. Steffan, "Efficient Multi-ported Memories for FPGAs," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '10. ACM, 2010, pp. 41–50.

[13] J. A. Stankovic and K. Ramamritham, "What is Predictability for Real-time Systems?" *Real-Time Systems*, vol. 2, no. 4, pp. 247–254, 1990. [Online]. Available: http://dx.doi.org/10.1007/BF01995673

[14] G. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011, vol. 24.

[15] U. Brinkschulte, C. Krakowski, J. Kreuzinger, and T. Ungerer, "Interrupt Service Threads-a new Approach to Handle Multiple Hard Real-time Events on a Multithreaded Microcontroller," *RTss WIP sessions, Phoenix*, pp. 11–15, 1999.

[16] A. Oliveira, L. Almeida, and A. de Brito Ferrari, "The ARPA-MT Embedded SMT Processor and Its RTOS Hardware Accelerator," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 3, pp. 890–904, March 2011.

[17] R. Thekkath and S. Eggers, "The Effectiveness of Multiple Hardware Contexts," *SIGOPS Oper. Syst. Rev.*, vol. 28, no. 5, pp. 328–337, Nov. 1994.

[18] R. Riedlinger, R. Bhatia, L. Biro, B. Bowhill, E. Fetzer, P. Gronowski, and T. Grutkowski, "A 32nm 3.1 Billion Transistor 12-wide-issue Itanium Processor for Mission-critical Servers," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, Feb 2011, pp. 84–86.

[19] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996. [Online]. Available: http://dx.doi.org/10.1007/BF01940883

[20] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: a Template for Composable and Predictable Multi-processor System on Chips," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 1, p. 2, 2009.