Benchmarking I/O response speed of microprocessors

Goncalo Martins
University of Denver
Goncalo.Martins@du.edu

Andrew Stanford-Jason
XMOS Ltd.
andrew@xmos.com

David Lacey
XMOS Ltd.
davel@xmos.com

ABSTRACT

This paper describes a method for comparing the I/O responsiveness of microprocessors with programmable input and output functionality. This addresses the need for benchmarks for real-time systems that measure critical properties for system design that are not currently handled by traditional performance benchmarks.

The benchmarks are tested against three different microprocessor architectures: ARM, PIC and XMOS. The results of the benchmarks are presented and analyzed to show how they can inform real-time system designers.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

benchmarking, real-time, performance

1. INTRODUCTION

Real-time systems [4, 5] are systems whose functionality depend on performing computations within known deadlines. Such systems are common in embedded environments. Often real-time systems are split into *hard* and *soft* real time; hard real-time systems have catastrophic failure if a deadline is not met to whereas soft real-time systems do not have catastrophic failure but do suffer quality loss if deadlines are not met.

The need to adhere to such deadlines makes particular demands of the architecture on which a real-time system is implemented. If the architecture itself is not well suited to this type of programming the developer may have to accommodate this is some way such as over-provisioning (*i.e.* requiring a much faster processor than is needed on average to make the worst case), accepting the failure and implementing special "recovery" methods when things go wrong, or even implementing the most challenging real-time part of the system directly in hardware.

Many existing benchmarking methods are relevant to real-time system designers. The accuracy, throughput performance and reliability of such systems affects the quality of the final application. However, the differentiating feature of a real-time system (that tasks must be completed by certain deadlines) is not particularly well represented by traditional benchmarking techniques.

Complete application benchmarks would be one way to address real-time system benchmarking but complete real-time applications are generally hard to port from one system to another, so pragmatically this method is not well suited to comparing different architectures and systems. Even application kernels can be quite large with complex implementations.

Another option is micro-benchmarking *i.e.* benchmarking the fundamental operations involved in real-time systems. For example, the Thread Metric benchmark suite [3] provides a micro-benchmarking framework for the common operations of real-time operating systems. These provide a useful data point for system designers but are slightly removed from application requirements and only cover systems using a real-time operating system.

This paper takes a different approach to try and characterize a single, but key, property of real-time systems: the speed at which a system can respond to an external stimulus. Furthermore, it tries to characterize how this property varies when the system is also performing many tasks that require responsiveness.

While characterizing a single property will not tell a system designer everything they need to now about a system, it is the authors' belief that this property reflects the potential capability of a system for a range of real-time embedded tasks.

2. METHOD

The key property that the benchmarks in this paper test is the I/O response latency. This can be characterized relatively simply:

The *response latency* of a system is the time it takes the system to change an output signal in response to a change in an input signal.

To measure this is relatively simple. The device under test is connected to a measuring device through two wires: the *input* wire contains the signal to be received by the device under test and the *output* wire contains the signal to be output by the device under test (see Figure 1).

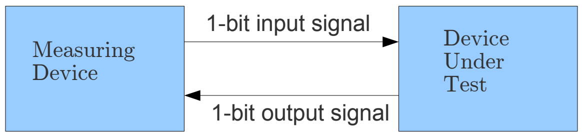

Figure 1: Test setup

The test consists of the measuring device creating a stimulus by driving the input signal from low to high. At this point the device under test should react and drive the output signal from low to high. The measuring device calculates the time between the driving of the input signal and the change in the output signal. The measuring device then drives the input signal low again, the device under test drives the output signal low and the test can be repeated. Figure 2 shows the interaction of the two signals and the measured response time.

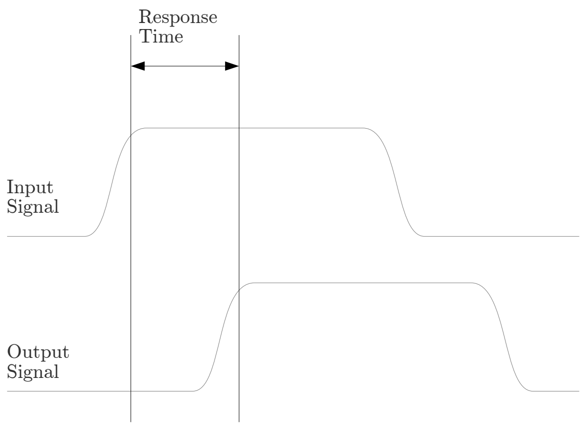

Figure 2: Test signals and response time

The accuracy of the test depends on the accuracy of the measuring device which will have a specific granularity and accuracy of measuring time. For the specific tests reported in this paper this is discussed in Section 2.5.

In addition to measuring response time, we wish to test how the device under test works while doing multiple tasks. This involves adding two extra dimensions to the test:

- The device under test should run a workload task that performs processing in between the responses to the external stimulus.
- The device under test should be tested for multiple responses *i.e.* how it can respond to several input wires in parallel.

When several responses are required, the inputs are all signalled at the same time and all times are measured between this event and each output wire going high.

2.1 Implementation

The implementation on the device under test is required to fulfil two requirements:

- Implement the required response behaviour for the test setup described above (the *characteristic task*) to respond to one or more input signals.
- Run a *workload task* concurrently with the characteristic task.

Details of the workload task can be found in Section 2.2. Due to variations in architecture and programming method it is not always feasible to have the same source code across different platforms. Therefore, in the tests presented here the systems re-implement the same functionality in ways that are natural to the architecture. The implementation should follow these guidelines:

- The implementation cannot be the same across systems so should be implemented using the best method for the architecture so long as it provides the same external behavior.
- The implementation should attempt to be representative of standard real-time embedded programming practices for that device.
- The implementation that responds to the input signal must be part of the generally programmable logic of the system.
- The characteristic tasks must be *concurrent* in the sense that they could respond to the inputs occurring at different times in an asynchronous manner and could be extended to respond in different ways.

The guidelines try to ensure that the implementations are such that the results of the testing can provide meaningful guidance to programmers trying to evaluate how systems are likely to perform in real applications.

The last guideline is to ensure a fair reflection of real application tasks in the implementation even if the test harness works in a specific synchronous manner (*i.e.* signals all inputs at the same time).

2.2 The workload task

The workload tasks model the idea that in a real application the system will be doing many things at once. What the task is and how it is implemented will depend on the system. However, the workload task must represent a “fair” use of all the major types of the systems resources (for example: memory, functional units, I/O).

Overall, the functionality of the task itself is not that important providing the above guidelines are adhered to. Generally a simple loop that performs some calculation and some reads and writes from memory will suffice.

2.3 Performance normalization

The aim of this paper is to use the benchmarking to compare system architectures rather than individual devices' absolute performance. Clearly, a device running at a higher clock rate will generally run faster and is more likely to get a better response latency. For the purpose of comparing architectures, it is necessary to normalize the clock frequency of the device.

There are two approaches to this: run every device under test at the same clock frequency or normalize by dividing the result by the clock frequency. The first approach is probably a more realistic test but is not always possible due to the oscillators and clocking capabilities on specific devices. In the experiments presented here a dual approach was taken: the clock speeds were chosen to be as close as possible to each other and then a normalization factor was applied to the results of each device.

2.4 Response time variation

The response time of a system may vary depending on system state. The *jitter* of the response time is the extent of this variation (*i.e.* the maximum observed response time less the minimum observed response time).

A sensible benchmarking strategy is to run the test multiple times and keep track of the average, minimum and maximum response time. Since the workload tasks do not interact with the outside world the main variation of system state will be done to the relative timing of the input signal against the current state of the implementation. As such it is vital that the signal generator varies the timing of its stimulus creation by random amounts that are not likely to be multiples of any internal clocks of the device under test.

2.5 Setup and measurement

The systems were tested by using an XMOS XK-1A development board as a measurement device. This device measures time intervals with 10ns granularity to an accuracy of within 2.5ns.

Each system was tested by generating a signal change and measuring the response time for 2^{20} repetitions.

2.6 Systems Tested

Three systems were tested with the aim of comparing architectural variation. The ARM and XMOS systems were run with a system clock frequency of 100MHz. The PIC was run with a system clock of 40MHz and the results were scaled up to normalize to 100MHz.

2.7 ARM

The ARM development system had the following properties:

Dev. Board	BlueBoard LPC1768-H
Device	LPC1768FBD100
Core	ARM Cortex-M3
Dev. Tools	Keil uVision 4.21
OS	FreeRTOS 7.0.1

The characteristic task was implemented by mapping the input signals to external interrupts which would interrupt the workload task.

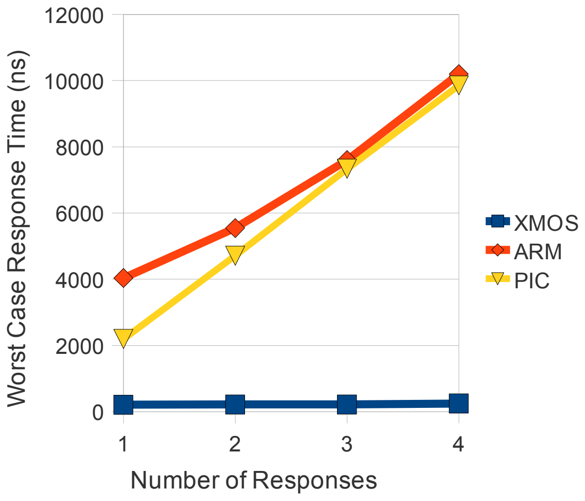

Figure 3: Worst case response times

2.8 PIC

The PIC development system had the following properties:

Dev. Board	dsPICDEM Starter Board V2
Device	dsPIC33FJ256
Dev. Tools	MPLAB v8.80
OS	FreeRTOS 7.0.1

The characteristic task was implemented by mapping the input signals to external interrupts which would interrupt the workload task.

2.9 XMOS

The XMOS development system had the following properties:

Dev. Board	XMOS XC-1A
Device	XS1-G4
Dev. Tools	XMOS Development Tools 11.2.2
OS	None

Both the characteristic and workload tasks were implemented as hardware threads with the input signals reaction implemented as port events using conditional input statements in the XC programming langauge [2].

2.10 Benchmark Availability

All the software required to run the benchmarks is available to download for free under an open source license [1].

3. RESULTS

3.1 Architecture comparisons

The raw data for all the experiments can be found in Appendix A.

Figure 3 shows the normalized maximum response latency of the three architectures benchmarked as the number of real-time tasks increase. This is the most relevant measurement for hard real-time systems.

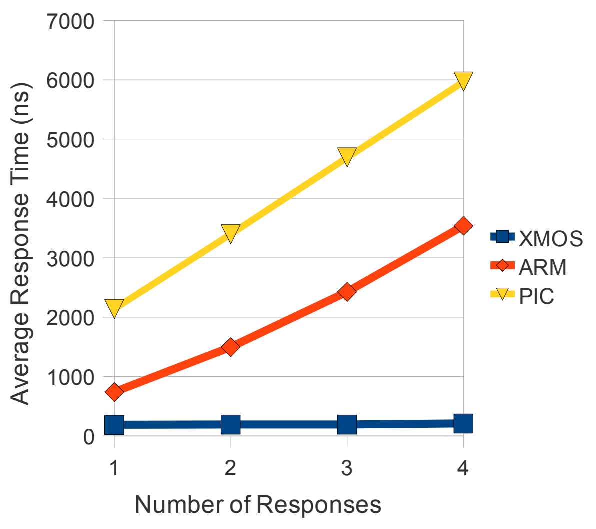

Figure 4: Average response times

The average response latency of the different architectures are shown in Figure 4. This figure may be more relevant to soft real-time systems.

The average response latency should only be viewed taking into account the jitter of that response, that is, the spread between minimum and maximum response time. Figures 5, 6 and 7 show this spread.

3.2 Analysis

Overall it is clear that the XMOS devices exhibit better characteristics in terms of response latency for both soft and hard real-time systems with both the average and, in particular, the worst case latency being significantly less than the other architectures. This is perhaps unsurprising given that the architecture is specifically designed for this type of task. The difference can mainly be attributed to the fact that on the XMOS architecture each response is handled by a separate hardware thread which can respond to the input signal without needing a context switch.

Both the ARM and PIC architectures implement the response via interrupts (the lowest latency method available on those systems) which require a context switch before response. The concatenation of these context switches explain the linear growth in response time given the number of responses needed.

Every attempt has been made to make the comparisons in the previous section represent the best performing setups of the different architectures given realistic assumptions about how the processors would be used for real applications. However, there may be trade-offs designers can make that either negatively affect the response latency or may improve it in certain cases. On both the PIC and ARM chips one of the key factors is the choice of operating system. In fact, these benchmarks could be seen as characterising the combination of both the architecture *and* the OS. Different combinations will likely have different results. In some cases no OS will be used at all and that is likely to improve the results (though at the cost of having to hand craft the context switching, interrupt servicing and task scheduling within the system).

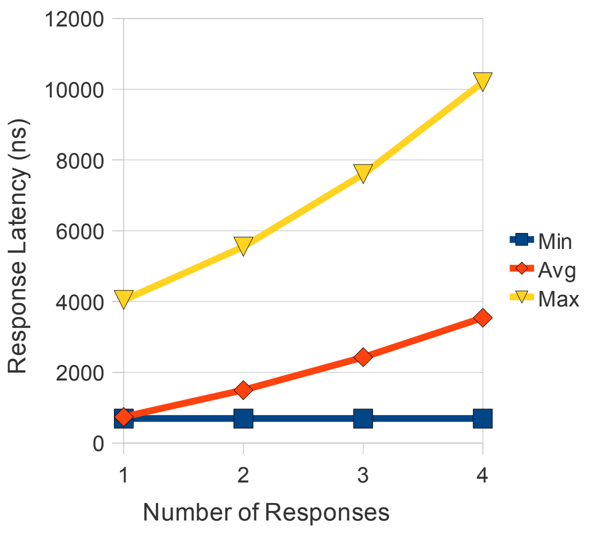

Figure 5: Spread of response times (ARM)

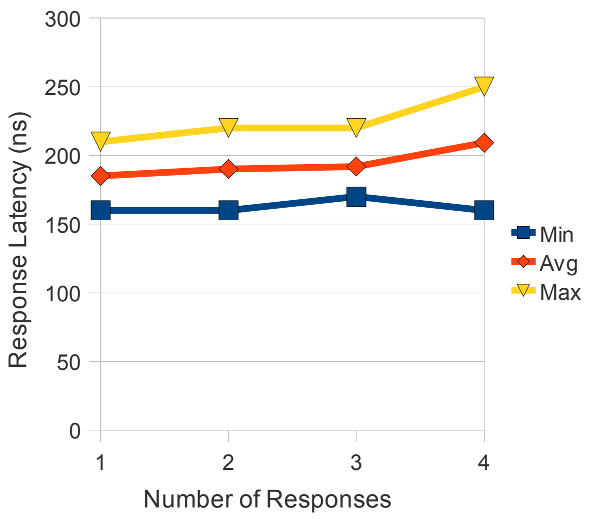

Figure 6: Spread of response times (XMOS)

Even given the variations in results that these design trade-offs can make it is the authors' view that the architecture comparison presented here is representative of what the various architectures can achieve.

4. CONCLUSION

The main contribution of this work is to propose a more suitable metric and related benchmarking method to characterize properties of interest to real-time system developers. The results of the benchmarks show that the response latency of systems can vary significantly between architectures and under different system loads; these results will be useful to designers when initially designing a system. Of course, in all these cases the response latency performances needs to be balanced against other system requirements.

4.1 Future Work

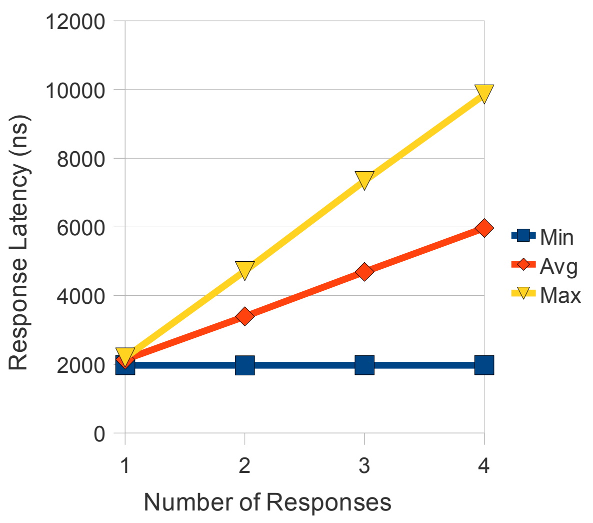

Figure 7: Spread of response times (PIC)

The results presented here only tracked the maximum, minimum and average response times. It would be useful to extend the test harness to produce a distribution of results to see, for example, how likely the maximum is to occur.

The tests were also limited in how many signals are handled by the hardware of the devices tested. However, with more work it would be possible to work around these limitations in software for all the devices under test. This would probably see a dramatic drop in performance past the hardware limit (e.g. four fast interrupts on the ARM device) but the size of that difference and the scaling properties past this limit would be of interest to designers.

5. ACKNOWLEDGMENTS

The authors' would like to thank Jon Ferguson, Mark Lippett, Henk Muller and Douglas Watt for comments and contributions to the work presented here.

6. REFERENCES

[1] I/O benchmark software repository. http://github.com/xcore/sw_io_benchmarks.

[2] Douglas Watt. Programming XC on XMOS Devices. Website, 2009. http://www.xmos.com/published/xcxs1.

[3] Express Logic, Inc. Measuring Real-Time Performance of an RTOS. http://rtos.com/PDFs/MeasuringRTOSPerformance.pdf.

[4] C. M. Krishna. *Real-Time Systems*. John Wiley & Sons, Inc., 2001.

[5] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

APPENDIX

A. TEST DATA

In the following sections all results are in nanoseconds and normalized to a 100MHz system clock.

A.1 ARM Tests

Responses	Best	Average	Worst
1	690	739.47	4040
2	690	1495.40	5550
3	690	2426.56	7600
4	690	3542.61	10260

A.2 PIC Tests

Responses	Best	Average	Worst
1	1976	2137.69	2196
2	1972	3396.78	4712
3	1976	4691.53	7340
4	1976	5967.98	9856

A.3 XMOS Tests

Responses	Best	Average	Worst
1	160	185.01	210
2	160	190.13	220
3	170	191.94	220
4	160	209.21	250