Real Time Embedded System Lab 1 Report
Interrupt & OS switching timing measurements

Olivér Facklam
Ju Wu

April 2021

Introduction

The goal of this lab is to quantitatively evaluate the timings of hardware interrupts on the Nios II processor as well as OS switching in the μC/OS-II real-time operating system. This evaluation is done under various memory and cache configurations.

Hardware design

Specific counter

In this section, we realize the customized specific counter with the following features: 32-bit Avalon slave, 32-bit counter, 1 wait cycle for read access (synchronous read), increment the counter at the system's clock speed (50 MHz), command to reset the counter, command to start the counter, command to stop the counter, the counter value must be readable at all times, transfer the counter value at the start of the read cycle.

Register map of the interface

Register number	Write	Read
0	N/A	$iCounter$
1	Reset Counter	N/A
2	Start Run	N/A
3	Stop Run	N/A
4	$iIRQEn$	$iIRQEn$
5	$iClrEOT$	$iEOT, iEn$
6	N/A	N/A
7	N/A	N/A

Table 1: Register map for the specific counter

Realize and simulate the interface

The specific counter is implemented as a VHDL entity in the file `hw/hdl/special_counter.vhd`

Realize a Qsys component from this interface

We realized a Qsys component for the specific counter, with clock and reset inputs, an interrupt sender, as well as the Avalon slave interface.

Custom parallel port

In this section, we realize our custom parallel port with the following features: a standard 32-bit Avalon slave with 1-wait read and 0-wait write, parametrizable IO width, configurable direction, and interrupt generation on both IO port edges.

Register map of the interface

Register number	Write	Read
0	P_{out}	P_{out}
1	P_{set}	0
2	P_{clear}	0
3	Dir	Dir
4	N/A	P_{in}
5	IEn	IEn
6	Int_{clear}	Int

Table 2: Register map for the parallel port

Realize and simulate the interface

The custom parallel port is implemented as a VHDL entity in the file `hw/hdl/parallel_port/ParallelPort.vhd`

Realize a Qsys component from this interface

We realized a component for the parallel port, with clock and reset inputs, an IO conduit, an interrupt sender, and the Avalon slave interface.

System architecture

The following figure shows the overall architecture of the system we will use for the tests.

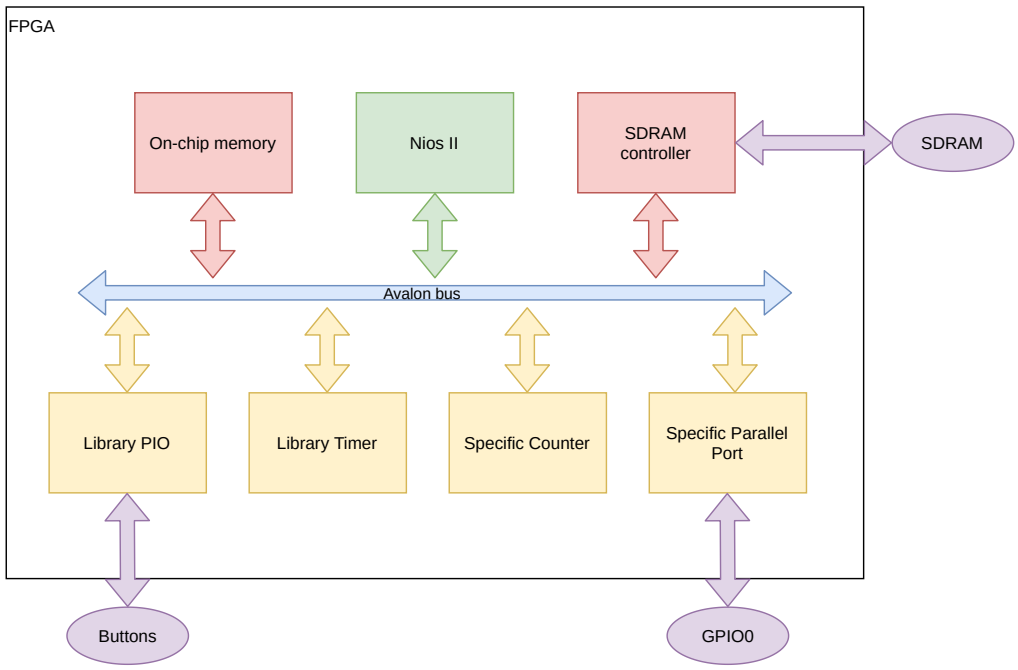


Figure 1: System architecture

Software design

In this section, we employ different kinds of measurement methods, i.e., the normal timer, our designed specific timer, and parallel port & logic analyzer to determine the response and recovery time of an interrupt. For the implementation of this lab, we have referred to the instructions of interrupt times measurement by software and manuals of configuration and reference of μC/OS-II provided on the Moodle page of this course.

Interrupt response time test

In this part, we employ the normal timer from library, as well as PIO & logic analyzer to measure the interrupt response time. The measurement results are listed in Table 4.

In the normal timer method, we snapshot the timer value in the ISR function to determine the interrupt response time. Indeed, after the interrupt is triggered by the timer hitting 0, the timer continues to count down from the initial value. The response time is thus given by the difference between the initial value and the snapshot from the ISR.

In the PIO & logic analyzer method, we trigger the interrupt by setting a bit on the parallel port in the main code, which is then cleared in the ISR. We measure the duration of the signal's 1-level in a period of square wave, which represents the interrupt response time.

Interrupt recovery time test

In this part, we employ the specific counter & library timer, as well as PIO & logic analyzer to measure the interrupt recovery time. The measurement results are listed in Table 5.

In the specific counter & timer method, we snapshot in the main function the value of the specific counter triggered at the end of ISR sourced by the timer IRQ to determine the interrupt recovery time. The first non-zero value sampled is the first value after the recovery from an ISR, and thus represents interrupt recovery time.

In PIO & logic analyzer method, we measure the duration of the signal's 0-level in a period of square wave to determine interrupt recovery time.

Interruptions test based on logic analyzer measurements

In this section, we combine the custom parallel port and the logic analyzer to measure the interrupt response time, recovery time and latency time. The frequency of logic analyzer is 24MHz and system clock frequency is 50MHz. With the logic analyzer and its software, and with our own designed parallel port initialized as output, we do the following tasks:

- We do the operations of *Set bit 0, Clr bit 0, Set bit 0, Clr bit 0* in an infinite loop and observe the signals with the logic analyzer to make timing measurements. The results of pulse frequency and the clock cycles that correspond to a period of the PIO output are demonstrated in Table 3 below, and the value in the bracket of each cell represents standard deviation of 10 measurement samples.
- We carry out interrupt measurements with PIO and logic analyzer under different processors-memories configurations (6 configurations concerning the memory and cache types) and demonstrate the results in Tables 4&5 below, and the value in the bracket of each cell represents standard deviation of 10 measurement samples.

Memory component type	Cache memory configuration		Measurements of 4 instructions loop overhead	
	Data cache	Instruction cache	Pulse frequency (MHz)	Cycles in a PIO period
On-chip memory	Disabled	Disabled	1.1158(0.0786)	45(3)
	Disabled	Enabled	4.6400(0.8044)	11(2)
	Enabled	Enabled	4.6000(0.6325)	11(1)
SDRAM memory	Disabled	Disabled	0.4479(0.0293)	112(8)
	Disabled	Enabled	4.6000(0.6325)	11(1)
	Enabled	Enabled	4.6400(0.8044)	11(2)

Table 3: Results of measurements of parallel port cycle time with different methods and configurations

Result analysis and discussion

In this part, we hereby build the following table to demonstrate our results of measurements to facilitate further comparison analysis and discussion.

We do more than 10 experiments for each configuration of the Table. 4-5 below, and compute the mean value and deviation value (bracketed in each cell) of 10 measurement samples for each configuration.

First, we observe that all interrupt timings are faster with the on-chip memory than with the SDRAM. This is to be expected due to the relative speed and location of those memories. Second, we can see that enabling caches significantly improves performance in every case. It is also clear that since the response time includes the interrupt latency, the response time measurements are systematically larger that the latency measurements. Finally, we can note that the interrupt response time is generally larger than the recovery time.

For a given configuration, the measurements obtained using the timer method versus the PIO method are usually comparable, although some differences can be noted. The response times measured with SDRAM memory show particularly large discrepancies between the 2 methods, while the recovery time measurements are particularly consistent.

In the process of experiment, we find out that there exist (nearly periodic) abnormal measurements of response and recovery time in most configurations. For instance, in the SDRAM with enabled instruction cache configuration, most interrupt response times we measure are around 1691 cycles while abnormal measurements, which are 20 to 30 times the normal ones, appear approximately every 10 to 20 samples.

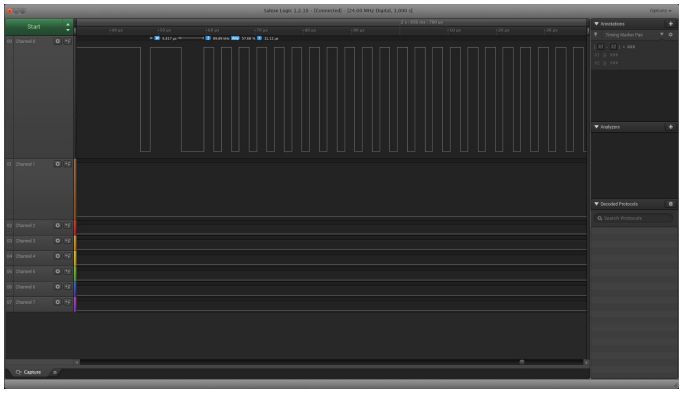
Memory component type	Cache memory configuration		Response time (cycle)		Latency time (cycle)
	Data cache	Instruction cache	Normal timer	PIO+logic analyzer	PIO+logic analyzer
On-chip memory	Disabled	Disabled	497(4)	633(1)	223(0)
	Disabled	Enabled	308(0)	290(1)	84(1)
	Enabled	Enabled	229(11)	161(1)	45(1)
SDRAM memory	Disabled	Disabled	1691(10)	1092(5)	573(4)
	Disabled	Enabled	799(40)	283(1)	116(1)
	Enabled	Enabled	555(139)	104(0)	45(1)

Table 4: Results of measurements of interrupt response and latency time with different methods and configurations

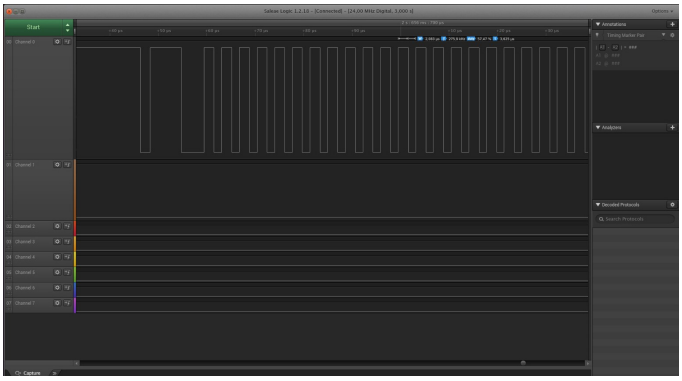
Memory component type	Cache memory configuration		Recovery time (cycle)	
	Data cache	Instruction cache	Timer+specific timer	PIO+logic analyzer
On-chip memory	Disabled	Disabled	319(11)	339(1)
	Disabled	Enabled	199(6)	165(1)
	Enabled	Enabled	93(4)	78(1)
SDRAM memory	Disabled	Disabled	862(25)	929(6)
	Disabled	Enabled	466(42)	369(1)
	Enabled	Enabled	159(5)	78(1)

Table 5: Results of measurements of interrupt recovery time with different methods and configurations

As an additional note, the following figures show snapshots of PIO output measured by logic analyzer demonstrating the interrupt response and recovery time difference between the initial wave samples and stable wave samples. From the Table. 4-5, we assume that these differences are due to the usage of data cache and instruction cache, that is the data and instruction cache are not properly employed at the start of run until the middle of the programme. This was an important note for us, since initially we did measurements directly after system reset, which were not representative of the real timings during continuous operation.



Initial waves samples of PIO output



Stable waves samples of PIO output

μC/OS-II operating system measurements

In this part we use the μC/OS-II operating system and measure the overhead of the various synchronization primitives provided by the OS kernel. The buttons in our design are employed as interrupt sources to measure the time used by μC/OS - II synchronization primitives in the SDRAM with enabled caches configuration. We consider the following synchronization primitives. The elapsed time measured with our specific timer and captured by the logic analyzer are compared respectively for each case.

Semaphore

The main task is made to *wait* on a semaphore. In the ISR for the falling edge of the buttons, we *signal* the semaphore to let the main task continue and measure the overhead of the semaphore.

Flags

The main task is made to *wait* on flags with a Boolean condition. In the ISR, we *signal* the flags to let the main task continue and measure the overhead of the flags. The speed of the AND/OR conditions is tested as follows:

- OR: wait on one of the buttons' falling edge.
- AND: wait until the four buttons have been activated.

Mailbox

The mailbox in μC/OS-II is employed to transfer messages between tasks and it contains a 32-bit value as payload. The message that fits in the 32-bit field can be sent directly through mailbox, while a pointer needs to be put to the data structure in the mailbox if more than 32 bits of data is sent.

In the buttons' ISR, a message as the pointer to a data structure is sent through a mailbox. And the data includes three members, i.e., the button number (0 to 3) pressed, a Boolean value that determines if the interrupt was caused due to a rising or a falling edge of the buttons and the time of the event (when button pressed) on 32 bits with a resolution of 1 us.

Queue

We apply the same setup as for the mailbox, except using a queue to pass messages.

Result analysis and discussion

In this part, we hereby build the following table to demonstrate our results of measurements of overhead of the various synchronization primitives provided by the OS kernel to facilitate further comparison analysis and discussion. We do multiple experiments for each OS primitive of the Table 6 below.

The overhead timings are fairly similar between the various synchronization mechanisms, ranging from 1200 to 1500 cycles. The measurements are also quite consistent between the counter method and the PIO method. We can note that that mailbox and the queue seem to be the fastest synchronization mechanisms, although they also transport the most data.

Memory component type and cache configuration	Synchronization primitives of the OS kernel	Measurement methods	
		Specific counter	PIO+logic analyzer
SDRAM memory with enabled data cache and enabled instruction cache	Semaphore	1412(58)	1332(58)
	Flags(OR)	1363(16)	1266(17)
	Flags(AND)	1487(26)	1392(26)
	Mailbox	1250(23)	1253(25)
	Queue	1246(30)	1150(30)

Table 6: Results of measurements of overhead time with synchronization primitives of the OS kernel and different methods

Conclusion

In this report, we design the specific counter and parallel ports that can trigger interrupt requests and design software to do measurements on interrupt response, recovery and latency times with normal timer, specific counter and PIO & logic analyzer under different configurations of memory component and cache. In virtue of our specific counter and PIO & logic analyzer, we measure the overhead time of the various synchronization primitives provided by μC/OS-II operating system under SDRAM with enabled data cache and enabled instruction cache.