ELE00026C - Digital Systems Coursework

Y3903727

December 4, 2023

Abstract

This report focuses on interrupts. The report starts with a definition of what interrupts are (in theory), then looks at practical uses of interrupts in a modern computer system. From there, this report shows an example of an interrupt controller (IC) used in the STMicroelectronics STM32 Nucleo-64 MCU Development Board, a popular microcontroller development board. Finally, this report explains how to design a simple example IC for use with a previously designed 4-bit CPU, ending with a conclusion and references section.

Contents

1 Interrupts: An Introduction

1.1 Interrupts

Interrupts are signals from a device to the CPU to interrupt execution of the current instructions. This signal is called an **Interrupt Request** (IRQ), and is represented by either one or multiple hardware lines to the CPU. When the CPU receives an IRQ (usually checked after the current instruction has finished [1]), the current state of the CPU is stored on the stack (program counter (PC), processor mode, current register values) for later restoration. The CPU will then branch to a predetermined location specified in the **Interrupt Vector Table** (IVT). The IVT specifies all of the addresses for each of the **Interrupt Service Routines** (ISR). An ISR is a block of code which describes what to do when a given IRQ is received. Once the ISR has finished[2], the CPU state is restored from the stack and execution of the next instruction begins.

Interrupts can also be called from software or by the processor. To distinguish between hardware interrupts and software interrupts we call software interrupts exceptions[3]. Interrupts are asynchronous events whereas exceptions are synchronous events. Both are handled the same within the processor.

Interrupts may have different priorities. If two priorities are received at the same time, the interrupt with the highest priority will be executed first. Interupt priority is set according to the order of the IVT. Also, depending on the implementation, nested priorities may be allowed. With nested priorities, if a higher priority interrupt is received while a lower priority interrupt is being handled, the lower priority's state is saved and pushed to the stack and the higher interrupt is handled.

Interrupts can be selectively 'masked' by the processor. The processor stores a mask of which interrupts are enabled. Some interrupts are not able to be masked. Interrupts which are able to be masked are called **maskable interrupts**, whereas interrupts which cannot be masked are called **non-maskable interrupts** (NMI). Non-maskable interrupts are high-priority events which should never be interrupted, such as reporting non-recoverable hardware errors or forcing a system restart.

1.2 Uses of Interrupts

Interrupts have a myriad of uses within a computer system. One such use is with peripherals, where new data may not always be available such as with a keyboard Most major PS/2 driver implementations make use of interrupts. [4] Interrupts cut down on wasted cycles spent polling the keyboard when no data is being sent (i.e small breaks

[1] Marilyn Wolf writes that 'the CPU checks for interrupts at every instruction'[1], therefore interrupts cannot happen mid-instruction.

[2] In the x86 ISA, the ISR ending is denoted with the `iret` opcode, which denotes the process of pushing the previous state from the stack and resuming execution of the interrupted program [2]

[3] Exceptions can be split further into faults, traps, and aborts depending on how they are handled and whether they are recoverable or not, according to the IA-32 architecture[2]

[4] For example, the linux kernel uses an interrupt based driver for PS/2 and AT keyboards.[3] The PS/2 driver used in Windows also uses an interrupt based system. [4]. Apple has never directly supported PS/2.

when no-one is typing). When a keypress occurs, an IRQ is sent to the CPU. The CPU then branches to the IVT where a pointer to a user-defined ISR is located, which then handles the keyboard input (usually by storing the inputs into some form of buffer). Other peripherals could use a similar mechanism, such as a PS/2 mouse which only sends interrupts when a change of state occurs (i.e. mouse button clicked or mouse moved).

Another use of interrupts would be with hardware timers. Hardware timers would allow interrupts to be sent periodically. One such use of this would be to keep track of time, such as creating a program that increments a counter every second to display the time. One pertinent use of periodic interrupts is with scheduling the running of multiple different processes/resources in a timely fashion (for example by giving each process an equal portion of time to run before switching to the next process[5]).

2 The STM32 Nucleo-64 MCU Development Board

2.1 How the STM32 Nucleo-64 Handles Interrupts

The STM32 Nucleo-64 MCU F303RE (shortened to the STM32 in this report) uses an ARM Cortex-M4 microcontroller. The ARM Cortex-M4 has multiple exception types:

Reset Async NMI sent on power up or reset. Highest priority (-3)

NMI Async NMI sent by either peripheral or by software. Second highest priority (-2)

Hard Fault NMI sent if an exception occurs when handling an exception or if an exception cannot be managed (i.e. if no position in IVT). Third highest priority (-1)

MemManage Exception sent for a memory protection related fault. Configurable priority.

Usage Fault Exception sent for faults related to instruction execution. Configurable priority.

SVCall Exception triggered by the OS to access kernel functions and device drivers. Configurable priority.

PendSV Exception used by the OS for context switching. Configurable priority.

SysTick Exception generated by the system timer when it reaches 0 (see subsection 1.2 for uses). Configurable priority.

IRQ General interrupt called by peripherals or a general exception called by software. Configurable priority.

Priorities are handled from the lowest numbered priority (-3) to the highest numbered priority (configurable). All interrupts with negative priority are NMI, the rest are maskable[5]

[5]This is a description of the Round-robin scheduler, a simple to implement process scheduler

The Interrupt Controller (IC) in the STM32 is referred to as the NVIC (nested vectored interrupt controller). As the name states, the NVIC supports nested interrupts. The NVIC is able to handle up to 73 maskable interrupt channels and 16 priority levels[6]. The NVIC automatically pushes the state of the CPU on the stack on entry, and pops the state off the stack on exit with no instruction overhead.

2.2 Interrupt Sequence on the STM32

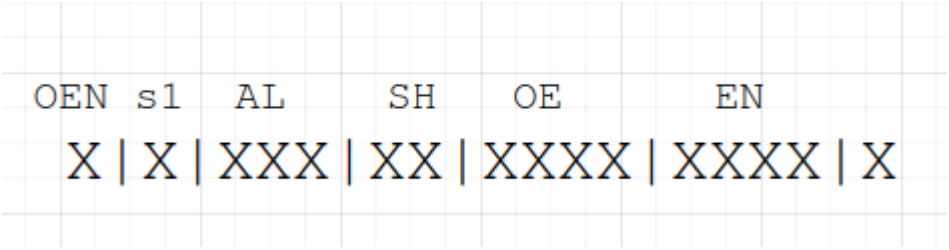
An IRQ event is received by the NVIC on one of its input lines. The NVIC then checks the **interrupt set-enable** and **interrupt clear-enable** registers. These registers show which interrupts are enabled or disabled[6]. If the interrupt is disabled, ignore the interrupt (unless it is NMI). Since some interrupts can have configurable priority (see subsection 2.1), the the **interrupt priority registers** (IPR) are checked. Each IPR can store 4 priorities for 4 interrupts (referred to as a priority field [5]) ranging from 0-255. The interrupt with the highest priority (if there are multiple priorities waiting) is sent to the CPU, where the NVIC will automatically stack the current state of the CPU. The sequence carries on as described in subsection 1.1.

3 Interrupt Controller Design

3.1 Specifications

The specifications of the IC are described in the assignment brief. The IC should be designed as a modification to the 4-bit Control Unit developed in the lab sessions. The IC will only be able to respond to one IRQ at a time (i.e. no nested interrupts). There will only be one IRQ. Upon receiving an IRQ, the current PC value should be incremented and stored, along with forcing the PC to the value 16 (10000b). At the end of the ISR, the PC should be restored. Any pending IRQs should be inhibited until the end of the ISR.

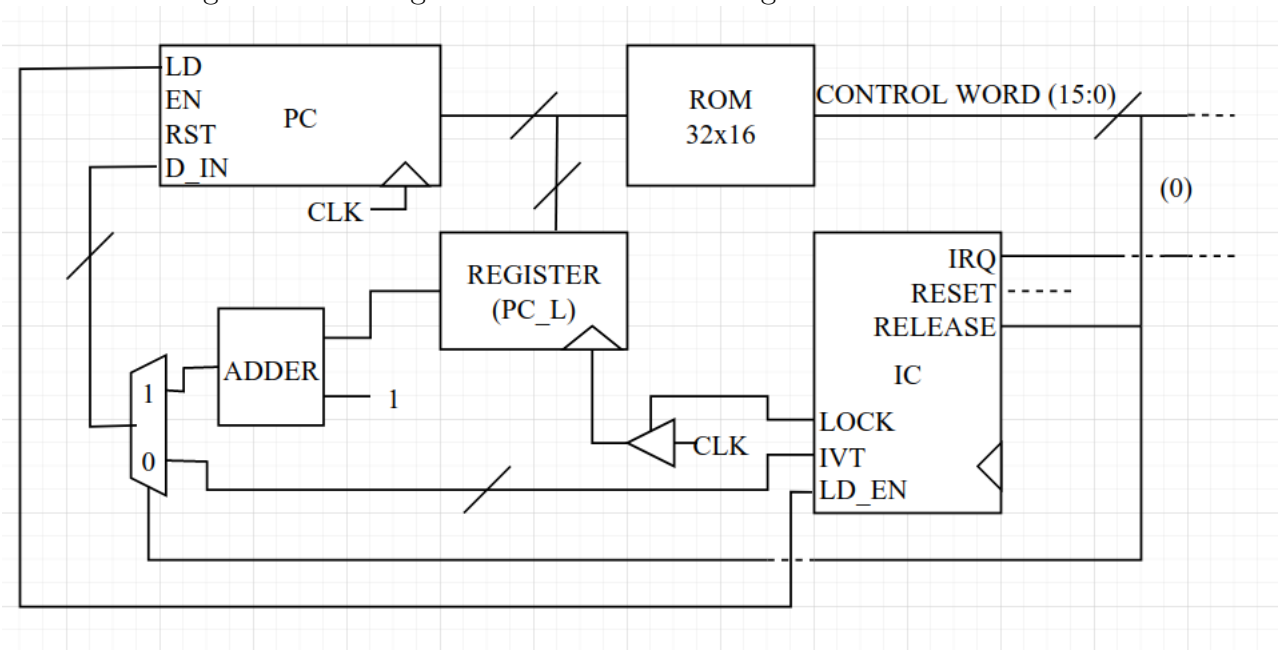
The control word used is shown below.



[6]The NVIC has two sets of registers to determine what is enabled and disabled in order to prevent race conditions

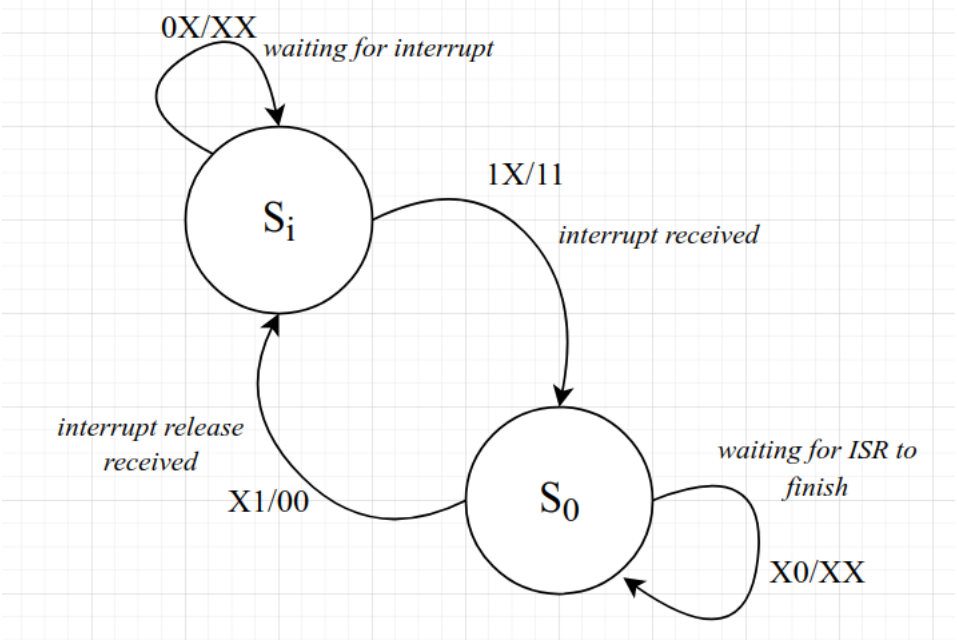
3.2 Block Diagram

The following is a block diagram of the relevant changes to the Control unit:



3.3 State Transition Diagram

The following is a Mealy FSM model of the interrupt controller:

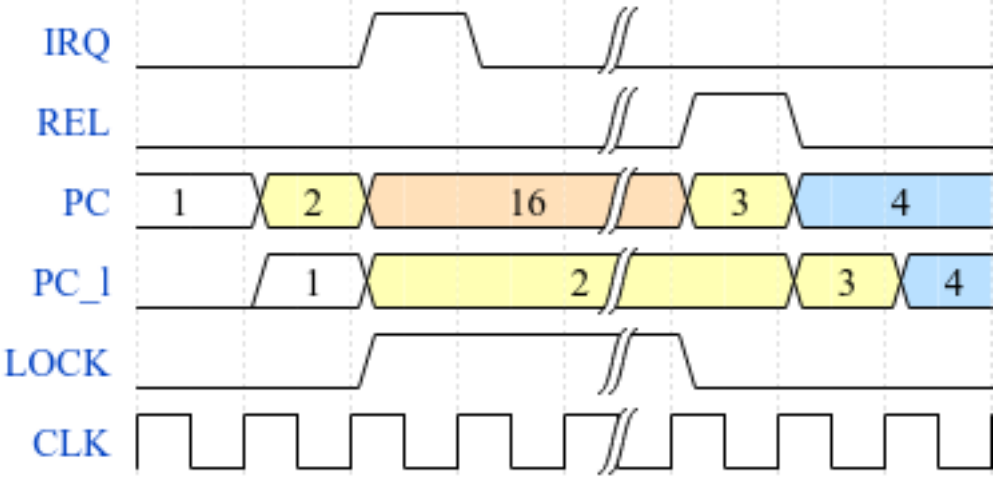


Starts at S_i, key is IRQ, RELEASE / LOCK, LD_EN

3.4 Further Information

The control word (see subsection 3.1) has been modified to create the opcode for leaving the ISR. The least significant bit has been wired to the IC's RELEASE pin (see subsection 3.2). Therefore, the control word to leave the ISR would be `0001000000000001`.

The following is a timing diagram of the entry into the ISR and release from the ISR:



The left side shows entry into the ISR, where the IRQ is set high, causing LOCK to be high and locking the clock to PC_L (so it can no longer update its value). The PC is set to 16 (10000b) and the ISR starts from there. The right side shows the exit from the ISR, where REL is set high, causing LOCK to be set low and unlocking PC_L's clock. PC_L's value is then incremented (see subsection 3.2) and sent to PC, where the program returns as normal from the interrupt.

4 Conclusion

This report has shown an example interrupt controller that complies with the given specification, along with adequately describing what interrupts are, and a real-word example of an IC. Further improvements to the IC would be by adding multiple different IRQ's, or allowing nested interrupts.

5 References

[1] M. Wolf, "Computers as components : Principles of embedded computing system design," in Elsevier, 2017, ch. 3.

[2] Intel, *Intel® 64 and ia-32 architectures software developer's manual*, 2022, ch. 6. [Online]. Available: `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`.

[3] V. Pavlik. "Atkbd.c - at and ps/2 keyboard driver." (2022), [Online]. Available: `https://github.com/torvalds/linux/blob/master/drivers/input/keyboard/atkbd.c`. (accessed: 02.05.2022).

[4] Microsoft. "Ps/2 (i8042prt) driver." (2021), [Online]. Available: `https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/ps-2--i8042prt--driver`. (accessed: 02.05.2022).

[5] ARM, *Cortex-m4 devices - generic user guide*, 2010. [Online]. Available: `https://documentation-service.arm.com/static/5f2ac4ab60a93e65927bbdbf?token=`.

[6] STMicroelectronics, *Stm32f303xd stm32f303xe datasheet*, 2016. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f303re.pdf.