Notes for ECE 36200 - Microprocessor Systems and Interfacing

Shubham Saluja Kumar Agarwal

August 28, 2024

These are lecture notes for spring 2024 ECE 36200 at Purdue as taught by Professor Younghyun Kim. Modify, use, and distribute as you please.

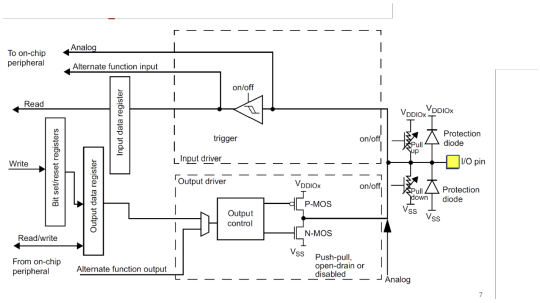
Contents

General-Purpose IO (GPIO)

GPIO allows us to interface a microcontroller with the rest of the world.
These microcontrollers have a number of pins which can be written to and read from using software.
Some functions of microcontroller pins are:

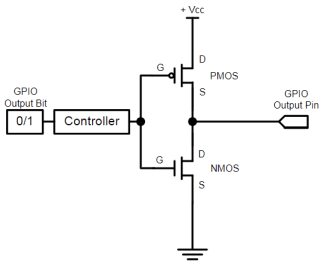
- Read to check if external voltages are applied.
- Apply a voltage to a pin.
- It can also perform Digital-to-Analog and Analog-to-Digital conversions.
- Several other functions are also available.

A GPIO has the following corresponding circuitry:

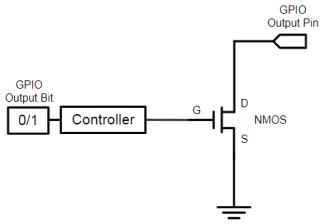


Output

The output is controlled by the following (push/pull configuration):



When the GPIO bit is 0, the output pin will have a 1, and vice versa.
That is, it connects to the output with a NOT gate.
However, there are other methods as well, such as the following open drain circuit:



In this configuration, when the GPIO out is 0, the output will be 0 as well. However, if it is 1, the output will float. This thus requires a pull-up resistor to function.

Control Speed

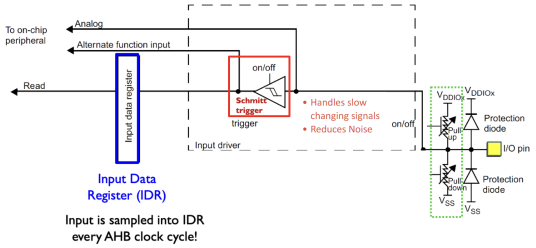
On a slight side note, we will analyze speed control. The tradeoffs of speed control are the following:

- More GPIO speed $\implies$ more noise and more power consumption

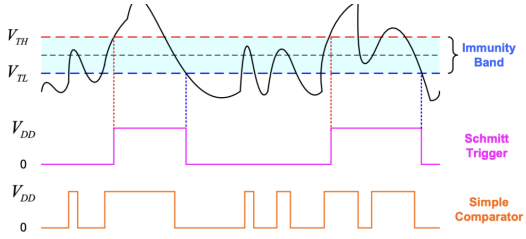
Thus, the speed must be configured based on peripheral speed. We can then relate the speed with the slew rate of the signal: $slewrate = \max\left(\frac{\Delta V}{\Delta t}\right)$

Input

The following is the input control/read circuit:



The Schmitt trigger in the above circuit allows for noise control on the input. A Schmitt trigger has two thresholds, restricting changes from 0 to 1 and from 1 to 0:



However, it should be noted that there exist quite a few electrical non-idealities, complicating input read procedures.

One solution for some of these problems is the use of the famous pull-up/pull-down resistors.

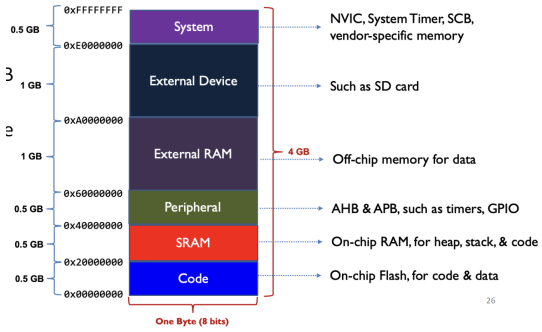
The resistance values must be calculated. If it were too low, it would act as a short circuit, increasing power consumption. If it were too high, however, it would be very slow.

Control

There are two main options for IO control.:

1. Special instructions in the processor.
   This is known as port-mapped IO, where each device is assigned a unique port number.

2. Memory-mapped IO
   Each one is assigned a unique memory address.
   IMPORTANT: For example, a 32-bit memory space has 1B long addresses.
   This means the microcontrollers that are used in class, which are 32-bit have limited read/write memory (2^{32}B$\approx$ 4GB).
   The memory can be allocated as follows:



Memory Types

- SRAM (Static Random Access Memory)
  - Volatile
  - Variables
  - Stack
  - Buffers
  - Test Code
- Flash Memory
  - Non-Volatile

- App Code
- Static Data
- Vectors (resets and interrupts)

Microprocessor Interfacing

A CPU reads from RAM to read/write variables.
It reads from ROM which contains read-only data.
It uses a bus to drive the address.
Another bus is used to send or receive data.
All GPIO peripherals have their own addresses, and can thus be accessed similar to the memory.
GPIO control needs to control both the Direction Register and the Data Register.
Note: memory is accessed at a minimum granularity of 1B, but more commonly 4B.

Conclusion

GPIO can do pretty much anything, given enough pins.
It uses bus protocols to implement these processes.

Interrupts

An interrupt is generated by hardware or software to inform that a condition has been fulfilled.
Interrupts have an advantage over forever loops in that the conditions are not checked in every loop, making the whole program far more efficient.
An interrupt is a type of exception.
Exception: hardware invoked subroutine

Program Counter

The code defined by your program is stored in memory as a series of binary values.
To keep track of location in code, called Program Counter, or PC.
PC is an address that points to the next command to be run.

Basics

Peripherals can raise interrupts.
The CPU can be interrupted by more than one thing, that is, more than one kind of interrupt.

The interrupts are linked to functions (Interrupt Service Routines) stored in a vector table.

These interrupts can be enabled and disabled, permitting or preventing peripheral from raising interrupts.
Interrupts have priorites, that is, a higher priority interrupt can interrupt a lower priority interrupt, but not the other way around.
When interrupts are raised, they are not immediately handled, but are instead placed into a sort of pending queue.
The following is the flow of an interrupt:

1. Peripheral raises an interrupt
2. CPU checks if this is enabled
3. CPU marks it as pending
4. Checks priority against current priority level (CPL), and runs if highest
5. If the priority is the highest:
   - Update CPL to this priority
   - Push CPU to stack
   - Push this entry into vector table on PC
6. The ISR should now be running

Tables like the ISR vector table are stored in the hardware, and modified or accessed using memory locations through direct access.
Only vector table lookup and stack saving require the memory to be accessed.
When interrupted, the CPU stores the current state (to return to later) into the SRAM. This information is retrieved once the interrupt is fulfilled.
Interrupts are usually handled at the end of a line of instruction. The code later continues at the line right after it.
If a fault happen sin the middle of an interrupt, the resume is at the same interrupt.
Interrupts are expected to be quick.
Interrupts also require specialized hardware to effectively interrupt the CPU.

Basic Timers, Debouncing, Multiplexing

Timers are a piece of hardware that allow us to execute tasks periodically.
Any useful microcontroller has a rich hardware system.

A System Ticker is a "special" timer used for system specific tasks.
A timer causes the CPU to execute a task, after every interval T.

SysTick Interrupt Time Period = (SysTick Load Period + 1) $\times$