Exam 1 Study

Wednesday, September 25, 2024 8:24 PM

Information taken from https://sites.google.com/vt.edu/introduction-to-embeddedsystem/course-management/syllabus

The midterm consists of:

- 60 minutes (10 minutes added to normal class time)
- We can refer to any course material from the semester and Wikipedia during the exam.

There is a Quizlet that contains the answers to the quizzes we need for this exam:
https://quizlet.com/948226688/flashcards?funnelUUID=43aa7f5b-a277-4c17-9946-64e605402521

We can use this Quizlet to study the topics that we have learned before this first midterm:

1. Introduction
2. Kit Overview
3. Digital I/O
4. API
5. FSM

Overview of Content from external website (see left):

1. Introduction

Section 1.1: Computers, Embedded Systems, and More

A **computer** is a device that can be instructed to carry out an arbitrary set of arithmetic or logical operations automatically.

An **embedded system** is a computer system with a dedicated function within a more extensive mechanical or electrical system.

- Compare it to a gear. It has a specific function.
- It contributes to the operation of a larger system.
- Many technologies use embedded systems, such as cars, buildings, medical devices, defense.
- More than 98% of computer systems are embedded systems.

Low-performance and low-complexity processors are known as **microcontrollers**.
High-performance and high-complexity processors are known as **microprocessors**.

- Embedded systems use variation of complexities in their processing.

Section 1.2: Hardware Introduction

An **abstraction layer** or **abstraction level** is a way of hiding the working details of a subsystem.

- Using abstraction enables efficient design and testing.

Transition from abstraction layer:

1. Atoms
2. Molecules (like SiO2)
3. CMOS transistors
   a. A tiny switch that can be opened or closed
4. Logic gates
5. Combinational logic
   a. Boolean circuits, function of the present input only
6. Sequential logic
   a. Group of gates, including storage gates that implement functions depending on current inputs and history of inputs.
   b. This has memory.
   c. Flip flops to store data.
7. Architectural units
   a. Advanced logic circuits that have a specific role, such as the ALU, register file, decoder.
8. CPU
   a. Central Processing Unit made of architectural units for general-purpose computation.
9. MCU
   a. Microcontroller Unit is a System-On-Chip (SoC) that consists of a CPU and a set of peripherals like ADC and communication units on the same chip.
10. Development board
    a. PCB that houses the MCU and other chips.
    b. Used for design and debugging.
11. Multi-board system:
    a. More boards.

Section 1.3: Software Introduction

To program microprocessors, a programmer can choose different programming languages based on their complexity:

- High-level languages:
  - C/C++,Java, Fortran
  - One line can encompass a complex series of operations.
  - These can be used for different computer architectures.
- Low-level languages
  - Closer to hardware level
  - Machine languages
    - Refers to binary code
    - One line of machine code represents one processor instruction.
- Assembly
  - Language between high-level and machine language
    - This is machine specific.
    - Each line of code encompasses one machine instruction.

A **compiler** is a computer software that transforms computer code in one language to another.

- These typically convert higher level languages to lower level languages (so C to assembly).

Machine code created from programming is called **object code**. This often cannot be directly executed on the machine, and needs to be linked.

- We use a **linker** program to create an **executable**.

An **assembler** translates assembly to machine code. A **disassembler** does the opposite.

The process of transforming a high-level language to an executable is called **building**.

2. Kit Overview

Section 2.1: Launchpad Board

The heart of the Launchpad board is the **MSP432** chip. This is a SoC microcontroller unit that houses the CPU and its peripherals. This is an ARM processor core.

The launchpad board also contains peripherals and other chips:

- XDS110: Large chip comparable in size to the MSP432. This is the debugger chip, so it only exists

on our development board.
    - A completed embedded system (completely debugged and tested), often exists on another board with just the MCU and other peripherals.
- LEDs
- Pushbuttons

Section 2.2: MSP432 Block Diagram

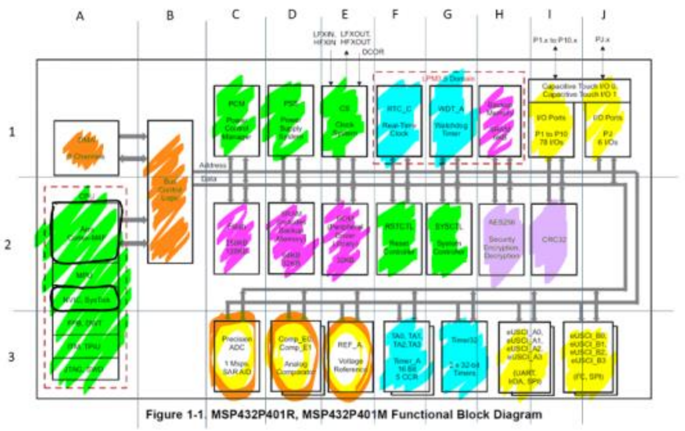


Figure 1-1. MSP432P401R, MSP432P401M Functional Block Diagram

1. The green blocks contain the CPU, Power, reset, and clock.
2. Orange blocks are for internal communication between blocks.
3. Magenta blocks are memory (storing program and data).
4. Cyan blocks are timers.
5. Yellow blocks are modules that help the microcontroller interact with other digital devices (digital I/O, I2C, UART, SPI).
6. Yellow AND orange blocks are interface blocks needed so that the microcontroller can interact between its own digital world and analog outside.
7. Purple blocks are for security and reliability protocols (AES, CRC).

Section 2.3: Guide to Documents
The MCU, dev board, launchpad, and booster pack have their own technical documents on the website.

Section 2.4: Components Summary
The Launchpad contains:
    The MSP432 chip
    The debugger chip
    The microUSB port
    The reset button
    Two push buttons
    One single-color and one tri-color LED
    Connector to the BoosterPack

The BoosterPack contains:
    The display
    The buzzer
    The microphone
    The joystick
    The accelerometer
    The tri-color LED
    Two buttons
    The light sensor
    The temperature sensor
    Connector to the Launchpad

The MSP432 contains:
    The processor core (ARM)
    The UART peripheral
    The I2C peripheral
    The SPI peripheral
    The GPIO
    The Timer_32
    The Timer_A
    The watchdog timer
    The interrupt controller
    The A/D

3. Digital I/O
Section 3.1: Introduction
On its own, a computer cannot do much physically but math with 1s and 0s.

Peripherals are auxiliary devices that are not part of the main computer.
- Connections between the computer and peripheral can always be severed or reconnected.

Section 3.2: Basics
General-purpose input/output (GPIO) can be used for input, output, and functionality depends on peripheral.
- On the MSP432, these GPIO (or digital I/O) connections come from **pins**.

    - Pins are a physical structure that create connections between the microcontroller and a peripheral (wires).

A pin can be either configured as an input or output, but not both at the same time (as it will create a short).

- A memory element like a flip-flop is needed (or a **register**) to configure the pin as:
    - An input or an output (direction).
    - How the pin behaves.

A group of pins is called a **port**. Each port has its own set of registers. Each bit of those registers corresponds to a pin.

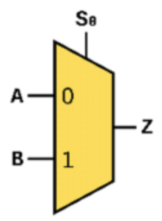
- Each individual port has their own set of pins numbered starting at 0.
- Each pin on a port shares that ports registers, but not the registers of other ports.

Because pins can be configured as inputs or outputs, there exists multiplexers and tri-state buffers to have different configurations.

Section 3.3: Components

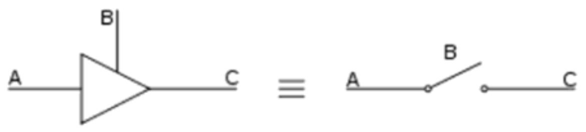
A digital I/O [in has many configurations. It contains:

A few **multiplexers** and logical gates like AND and OR



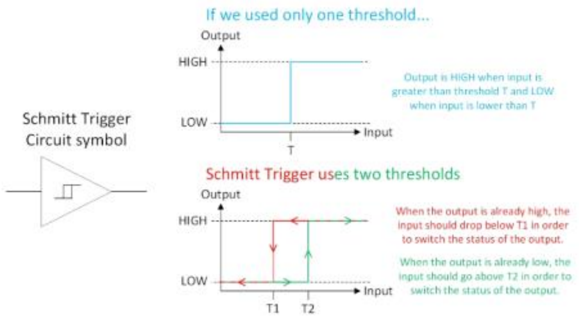
Z = ((! S_0) & A) | (S_0 & B).

Tri-state buffers allow an output port to assume a high impedance state in addition to the 0 and 1 logic levels.



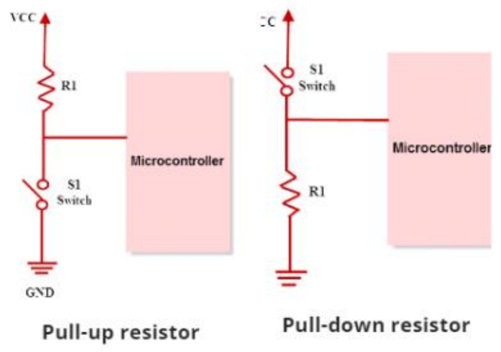
It is not OK to tie the output of two gates as it may cause a short.

Schmitt Trigger (1) is used as a basic ADC to prepare an input signal to be used as a digital signal. This uses hysteresis to create an output, and protects against noise.



Pull-up/down resistors. Since inputs only have two states (0/1), instead of the opposite logic (1 or 0), they are floating, and their voltage is undetermined. To help inputs, GPIO incldues a resistor.

a. If the input needs help to be a 1, it needs a pull-up resistor, which is a resistor between the input and voltage source.
b. If the input needs help to be a 0, it needs a pull-down resistor, which is a resistor between the input and ground.



Section 3.4: Pin Configuration

To designate a GPIO pin as an input or output, the **direction register PyDIR.x** must be used.

- This register sets whether a pin is currently an input or an output. As the pin diagram shows, 0 indicates that a pin is being used as an input, and 1 indicates output.
- This value for **PyDIR.x** is written to a register (flip-flop) and sent to a multiplexer to determine what lines the input pin is.

We are trying to create a path internally to the outside. Setting **PyDIR.x** to 1 causes the tri-state buffer to act as a wire to a multiplexer, which is connected to the output register **PyOUT.x**.

- This is also connected to a pin, which creates a link between the pin and output register.

To set a pin to an input, we set **PyDIR.x** to 0, and the tri-state buffer acts as high impedance. The pin is now connected solely to the Schmitt trigger, which is connected to the input register **PyIN.x**.

- The input register will match the value of the pin.

However, we need to eliminate the risk of producing a short circuit, so we need to configure the **resistor enable register PyREN.x** to create a pull-up/pull-down resistor to prevent from shorts.

- This register is connected to an AND gate, and since one of the AND gate's inputs is 1, the switch that enables the resistor is controlled by the value on the register (0 for closed, 1 for connected).

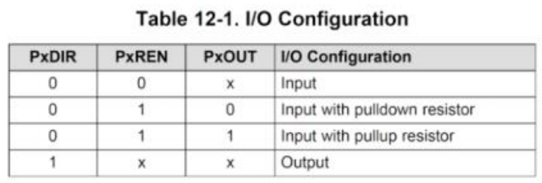


Table 12-1. I/O Configuration

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

(This can be found on Chapter 12 Section 2 of the MCU guide).

Section 3.5: Launchpad Peripherals

The Launchpad has four notable peripherals:

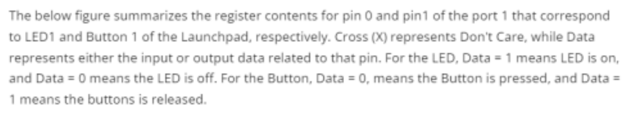
1. A standard LED
2. An RGB LED
3. Two pushbuttons

The two buttons are inputs, whereas the LEDs are outputs.

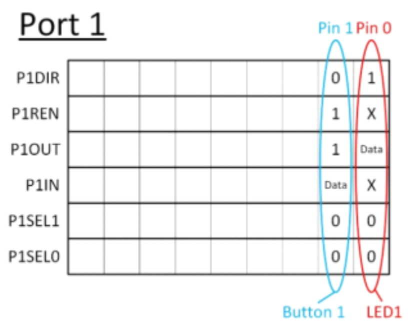
We can see that there are four wires for LEDS, one to enable the standard LED, and three others to RGB LED.

The push buttons are straightforward to enable.

- We must set the direction register to the appropriate value (P1DIR.x 0).
- We have to consider whether the inputs need pull-up/down resistors.
    - The ones on the Launchpad will need pull-up resistors (P1REN.x = 1).
- P1OUT.x needs to be 1.



The below figure summarizes the register contents for pin 0 and pin1 of the port 1 that correspond to LED1 and Button 1 of the Launchpad, respectively. Cross (X) represents Don't Care, while Data represents either the input or output data related to that pin. For the LED, Data = 1 means LED is on, and Data = 0 means the LED is off. For the Button, Data = 0, means the Button is pressed, and Data = 1 means the buttons is released.



Section 3.6: BoosterPack Peripherals

The BoosterPack is connected to the Launchpad through jumper headers. These headers are four clolumns of pins on the Launchpad that connect to four matching sockets underneath the BoosterPack.

- Each jumper pin corresponds to a GPIO port/pin mapping.
- All the BoosterPack peripherals are mapped to a jumper header.
    - Therefore each peripheral is mapped to GPIO.

Reference the BoosterPack pinout diagram in OneNote.

Section 3.7: Programming Digital I/O

See section (HW2/3).

Section 3.8: Memory-Mapped I/O

Digital registers can configure a pin. They also act as storage for any output/input value.

Since the processor should be able to read and write from the registers, and there are a large number of registers, the processor should be able to say which one of these registers it wishes to read/write to.

- This is called **addressing**.

There are two possible solutions for addressing:

1. Registers have private numbering (**address space**).
2. Registers are numbered together with memory locations of the processor (separate address space).

A processor with n-bit wide address bus has 2^n addressable locations. This set of 2^n addressable locations is called address space.

- The peripheral mapping scheme where the peripherals' registers have their own address space is called **port-mapped I/O**.

With larger address spaces, **memory-mapped I/O** became more viable because in cases like 32-bit and 64-bit processors, there exists 2^32 and 2^32 addressable locations, respectively. As such, there is plenty of space.

- The MSP432 uses memory-mapped I/O.

With port-mapped I/O, the processor requires a unique set of instructions to communicate with peripherals.

Memory-mapped I/O uses the same load/store instructions for data memory and the peripherals.

- The processor doesn't distinguish peripherals from data memory.

<u>Note: An n-bit processor is typically a processor whose address bus, data bus, ALU, and registers all have a width of n bits.</u>

4. API

<u>Section 4.1: Introduction</u>
In programming, we strive to build interfaces similar to examples like a restaurant menu (interface from customer to kitchen), TV remote (user and many TVs), email inbox (user to data).

We want a <u>clear interface to define an interaction</u>.

Application Interface (API) is an interface that allows what each side of the communication to understand what they are doing on their own, without needing to know each other.

<u>Section 4.2: API Basics</u>
An **application programming interface (API)** is a set of routines, protocols, and tools that say how different software components should interact with each other.

- Allows for efficient development of the software as building blocks.

API exists sometimes purely as software interactions, or as bridges between software and hardware.

<u>Section 4.3: API Driverlib</u>
A **device driver** is a computer program that allows communication between hardware devices by using software interface.

- This is a special form of API.

We are interested in the interactions between the CPU and the peripherals on MSP432 chip. Since there is a long list, there are many drivers.

- These drivers are stored in DriverLib (**driver library**).

The biggest advantage of using DriverLib is that it consolidates code into a simpler, easier-to-understand form.

These functions also tell the programmer what they do at a glance, forgoing any knowledge required beforehand.

<u>Section 4.4: HAL</u>
A **hardware abstraction layer (HAL)** is a conceptual interface between the user and hardware. The definition sounds similar to API, and that is because <u>HAL is a specific form of API.</u>

Portability refers to how easily you can export code to a different architecture.

HAL strives to keep functions high-level so that another person using the code can understand its basics.

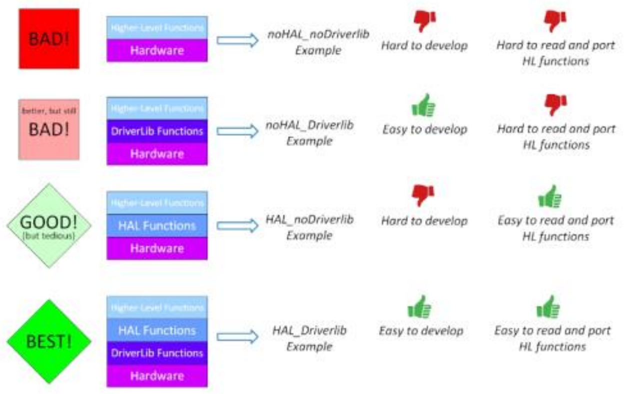
<u>Section 4.5: HAL vs DriverLib</u>
HAL and DriberLib are both forms of API.
Here are the comparisons:

DriverLib does obscure hardware details, but DriverLib are not portable.

- DriberLib functions are specific to Texas Instruments and won't work on Arduinos.

HAL is more of an abstract concept than an actual tool. Programmers must create their own HAL.

- Because HAL functions are so generalized, it's much easier to port code that has that abstraction layer than code that consists purely of DriberLib functions.
- The Arduino example can work, as you'd just have to replace the values for specific hardware configurations.


BAD!
Higher-Level Functions
Hardware
noHAL_noDriverlib Example
Hard to develop
Hard to read and port HL functions
better, but still BAD!
Higher-Level Functions
DriverLib Functions
Hardware
noHAL_Driverlib Example
Easy to develop
Hard to read and port HL functions
GOOD! (but tedious)
Higher-Level Functions
HAL Functions
Hardware
HAL_noDriverlib Example
Hard to develop
Easy to read and port HL functions
BEST!
Higher-Level Functions
HAL Functions
DriverLib Functions
Hardware
HAL_Driverlib Example
Easy to develop
Easy to read and port HL functions