Architectures for Embedded Systems

Review of C/C++ compilation flow and some topics on variables
Electrical characteristics and absolute maximum ratings
Laboratory assignments

Arnaldo S. R. Oliveira

Academic year 2024/25

Universidade de Aveiro – Dep. de Eletrónica, Telecomunicações e Informática

Outline

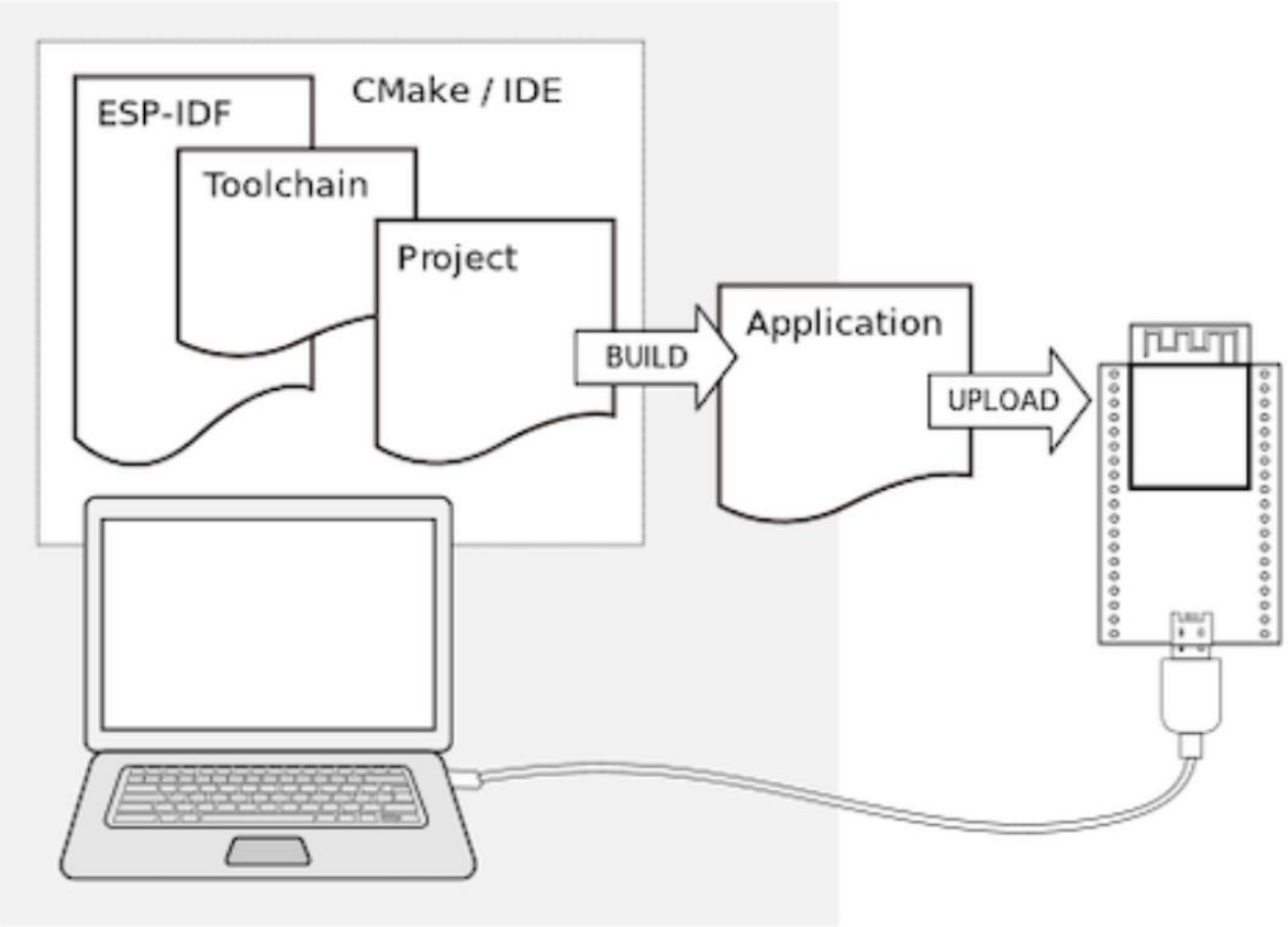
C/C++ compilation flow and tools

C/C++ Variables

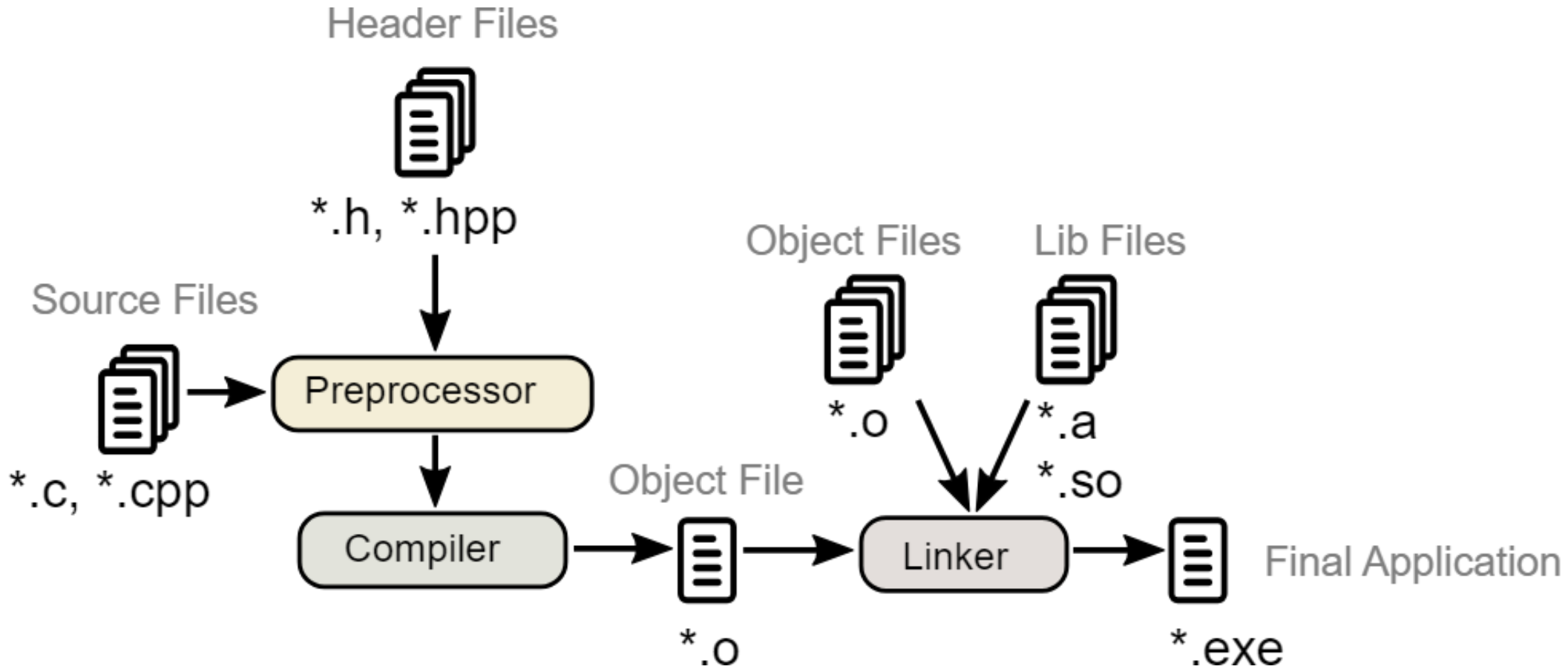
- Types
- Visibility
- Lifetime
- Allocation

Lab assignments

ESP32-C3 Compilation Flow



C/C++ Compilation Flow



What is the purpose of the CMake tool? (1/2)

CMake is an open-source tool designed to manage the build process of software in a platform-independent manner. It is primarily used for generating build files that can be used with different native build systems such as **Makefiles, Ninja, Visual Studio project files, Xcode project files**, and others.

Purpose of CMake:

1. **Cross-Platform Build Configuration**
   - CMake allows developers to write build scripts that work across different operating systems (Windows, macOS, Linux) and compilers (GCC, Clang, MSVC).
2. **Automatic Build System Generation**
   - Instead of manually writing complex Makefiles or project files, CMake generates them based on a simple CMake configuration file (`CMakeLists.txt`).
3. **Dependency Management**
   - It can find and configure external libraries, ensuring that all required dependencies are correctly located and linked.
4. **Out-of-Source Builds**
   - Supports building software in a separate directory from the source code, keeping the source clean and preventing conflicts.
5. **Integration with IDEs**
   - Works with popular Integrated Development Environments (IDEs) like Visual Studio, CLion, and Xcode.
6. **Build Customization**
   - Allows defining custom build options and configurations using `CMakeLists.txt` and `CMakeCache.txt`.
7. **Support for Multiple Build Types**
   - Easily configure different build types such as Debug, Release, RelWithDebInfo, and MinSizeRel.

What is the purpose of the CMake tool? (2/2)

Basic Workflow of CMake

1. **Write a** `CMakeLists.txt` **file** that defines the project, source files, dependencies, and build settings.
2. **Run** `cmake` to generate the build system files:

```
css                                                    Copy  Edit
cmake -S . -B build
```

3. **Use the generated build system** to compile the project:

```
css                                                    Copy  Edit
cmake --build build
```

CMake is widely used in modern C++ projects due to its flexibility, portability, and ease of integration with various tools and compilers.

What is the purpose of the Ninja tool? (1/2)

Purpose of the Ninja Build System

Ninja is a small, fast build system designed to efficiently handle incremental builds. It is primarily used to speed up the compilation of large projects by minimizing unnecessary work and maximizing parallelism.

Key Features and Purpose of Ninja:

1. **Optimized for Speed**
   - Ninja is significantly faster than traditional build systems like **Make** because it avoids unnecessary file checks and only rebuilds what is required.
2. **Incremental Builds**
   - It tracks dependencies efficiently, ensuring that only modified files are recompiled, reducing build times dramatically.
3. **Parallel Execution**
   - Ninja automatically maximizes CPU usage, making it much faster than `make -j` in large projects.
4. **Minimal Overhead**
   - The Ninja build system is lightweight and has fewer features than Make, focusing only on executing build commands as quickly as possible.
5. **Integration with CMake and Other Generators**
   - Unlike Make, Ninja does not have its own build configuration language. Instead, **CMake, GN (Google's build system), or Meson** are often used to generate Ninja build files.
6. **Cross-Platform Support**
   - Works on Windows, Linux, and macOS, making it a portable solution for fast builds.

Ninja vs. Other Build Systems

- **Make:** Ninja is significantly faster and better at handling dependencies.
- **CMake:** CMake is not a build system but a **build system generator**. It can generate Ninja files for fast compilation.
- **Bazel/Meson:** Ninja is simpler but does not include higher-level dependency management.

What is the purpose of the Ninja tool? (2/2)

Basic Usage of Ninja

1. **Generating Ninja Build Files using CMake**

```css
cmake -G Ninja -S . -B build
```

2. **Building the Project with Ninja**

```mathematica
ninja -C build
```

or

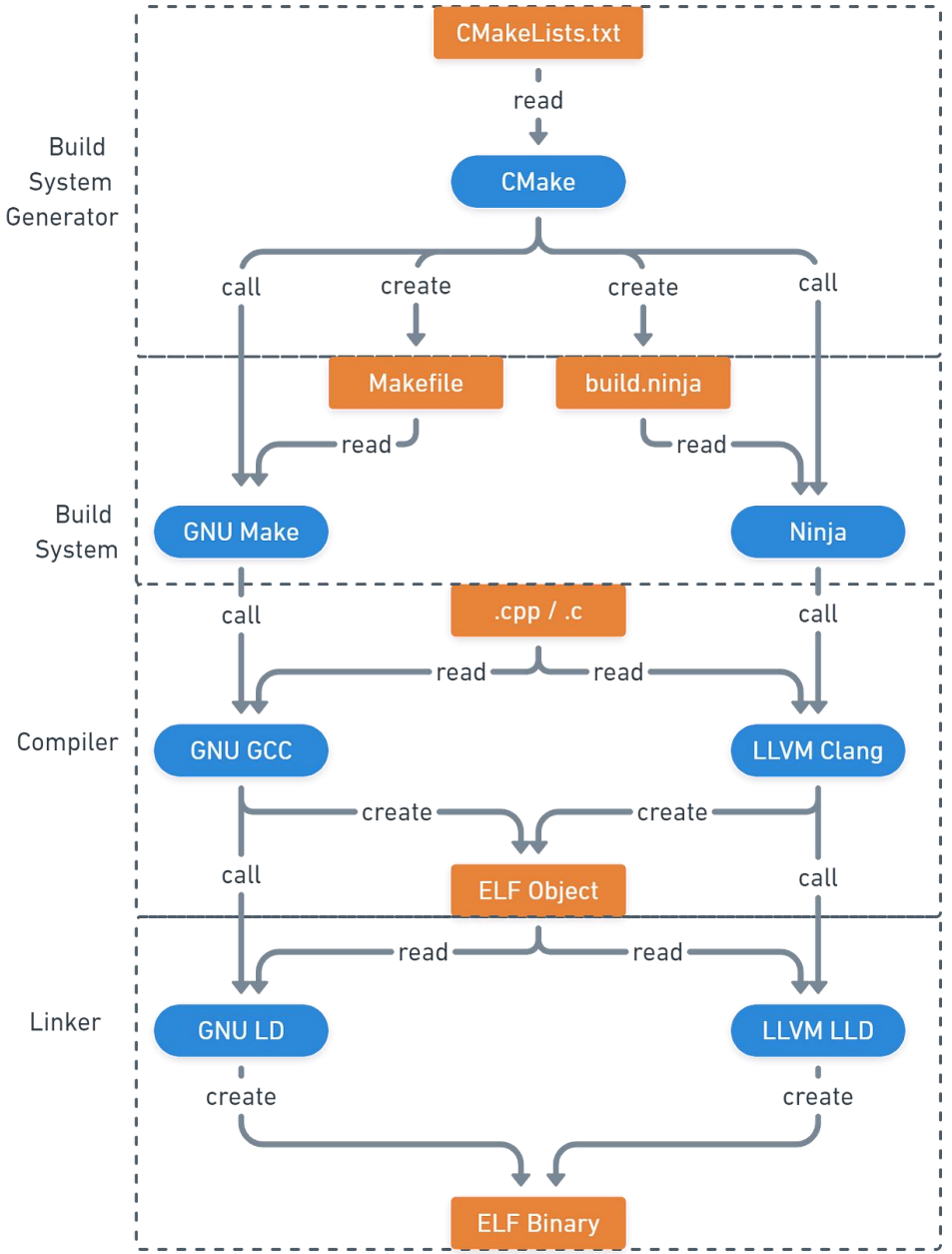
```graphql
cmake --build build -- -j8  # Using CMake with Ninja backend
```

Conclusion

Ninja is widely used in modern development workflows, especially in projects where **fast incremental builds** are essential, such as **Chromium, LLVM, and Android development**. If your project is large and relies on CMake, switching to Ninja can significantly improve build performance. 🚀

Cmake + Ninja Compilation Flow

(versus GNU Make)



C/C++ Primitives Types

Fundamental (Primitive) Types

- **Integer Types:**
  - **Standard:** `int`, `short`, `long`, `long long` (each typically comes in both signed and unsigned versions).
- **Floating-Point Types:**
  - **Standard:** `float`, `double`, and `long double`.
- **Character Types:**
  - **Standard:** `char` (which may be signed or unsigned depending on the implementation).
  - **Additional (C++ and modern C):** `wchar_t`, `char16_t`, and `char32_t` for wider character sets.
- **Void Type:**
  - Represents the absence of type, often used for functions that do not return a value or as a generic pointer type (`void*`).
- **Boolean Type (C++ only):**
  - **Standard:** `bool` (with values `true` and `false`).

C/C++ Derived Types

Derived Types

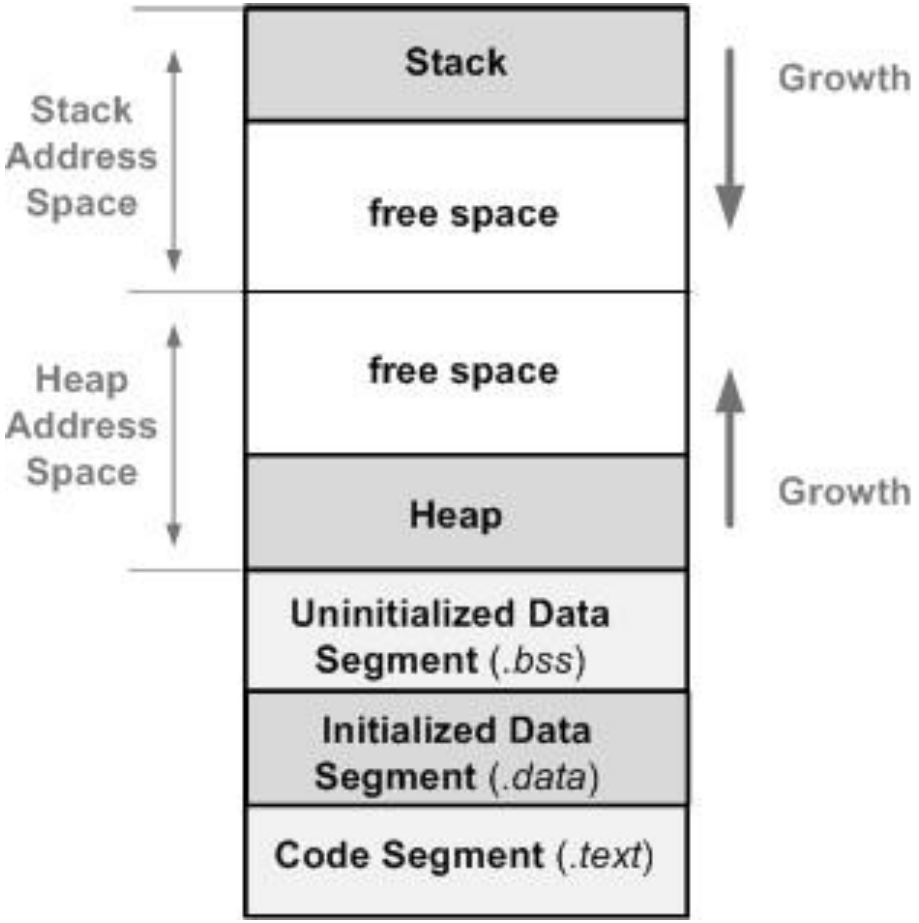
- **Arrays:**
  - Collections of elements of a specified type (e.g., `int arr[10];`).
- **Pointers:**
  - Variables that hold memory addresses (e.g., `int* ptr;`).
- **Functions:**
  - While not a data type in the usual sense, functions can be pointed to, making function pointers a derived type.
- **References (C++ only):**
  - Provide an alternative name for an existing variable (e.g., `int& ref = someInt;`).

C/C++ User-defined Types

User-Defined Types

- **Structures (`struct`):**
  - Custom data types that group together related variables.
- **Unions (`union`):**
  - Similar to structures but with all members sharing the same memory location.
- **Enumerations (`enum`):**
  - Define a type that can have one of a few discrete values, improving code clarity.
- **Classes (C++ only):**
  - The cornerstone of object-oriented programming in C++, classes can encapsulate data and functions together.

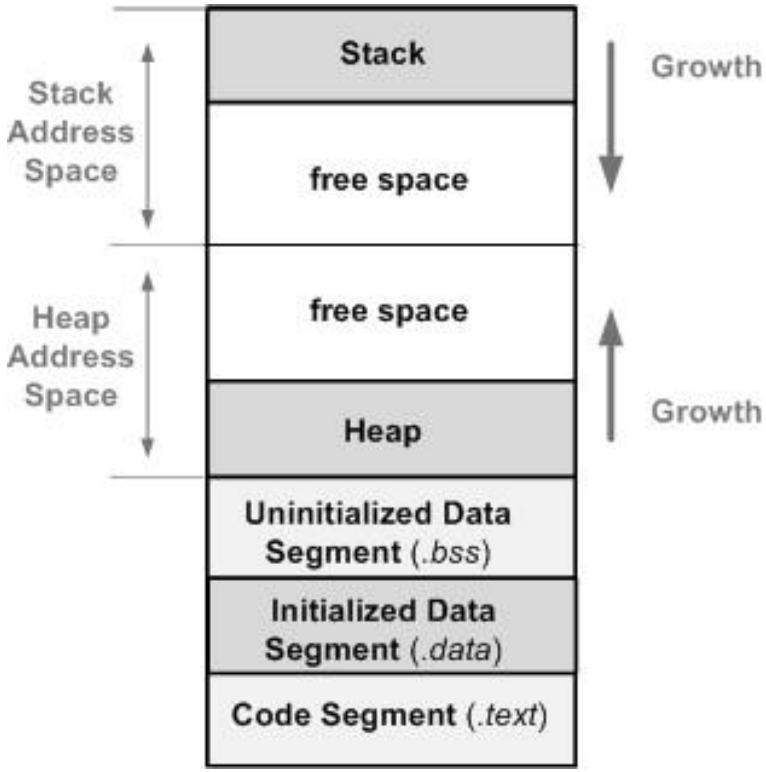
C/C++ Memory Segments



C/C++ Local Variables (automatic vs. static)

```
void f()
{
        int i;
        static int j =  0;
        …
}
```

- Visibility?
- Lifetime?
- Allocation?

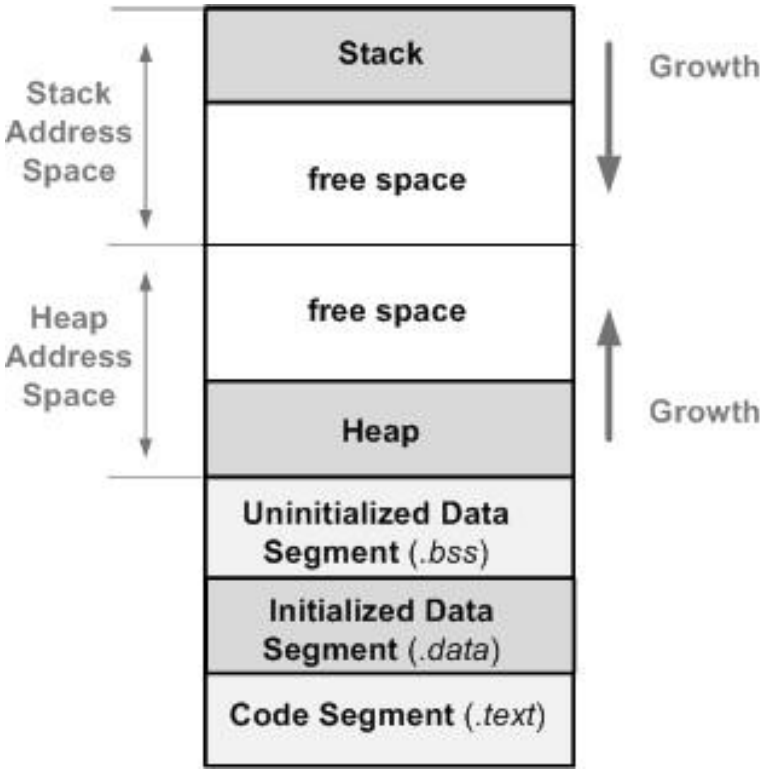


C/C++ Global Variables (program vs. module)

```
int m;
static int n;


void f()
{
        ...
}
```

- Visibility?
- Lifetime?
- Allocation?



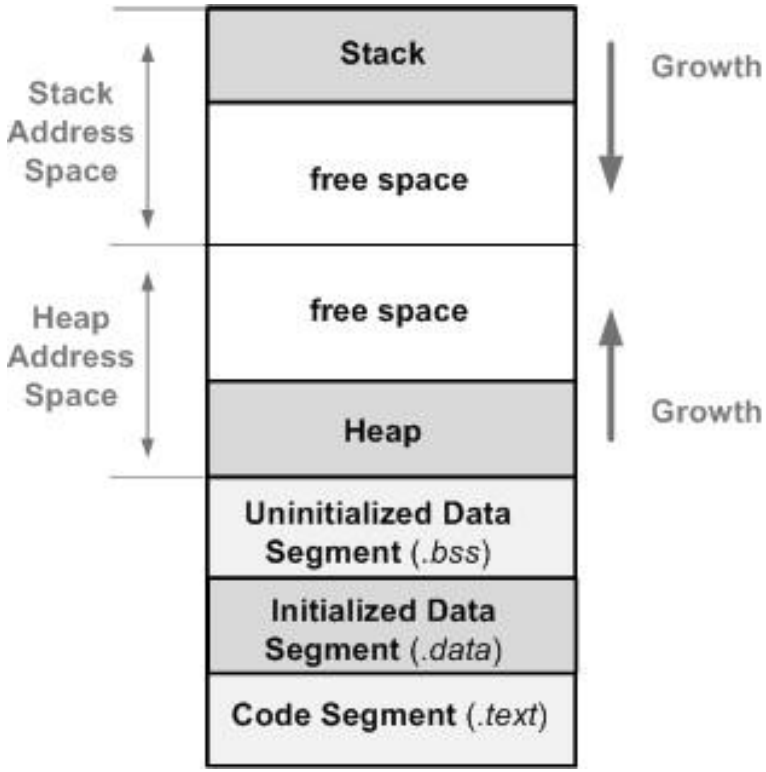
What is the purpose of the “extern” keyword?

C/C++ Heap Allocated Dynamic Variables

```
void f()
{
        int* p;

        p = malloc(N * sizeof(int));
        ...
        free(p);
}
```

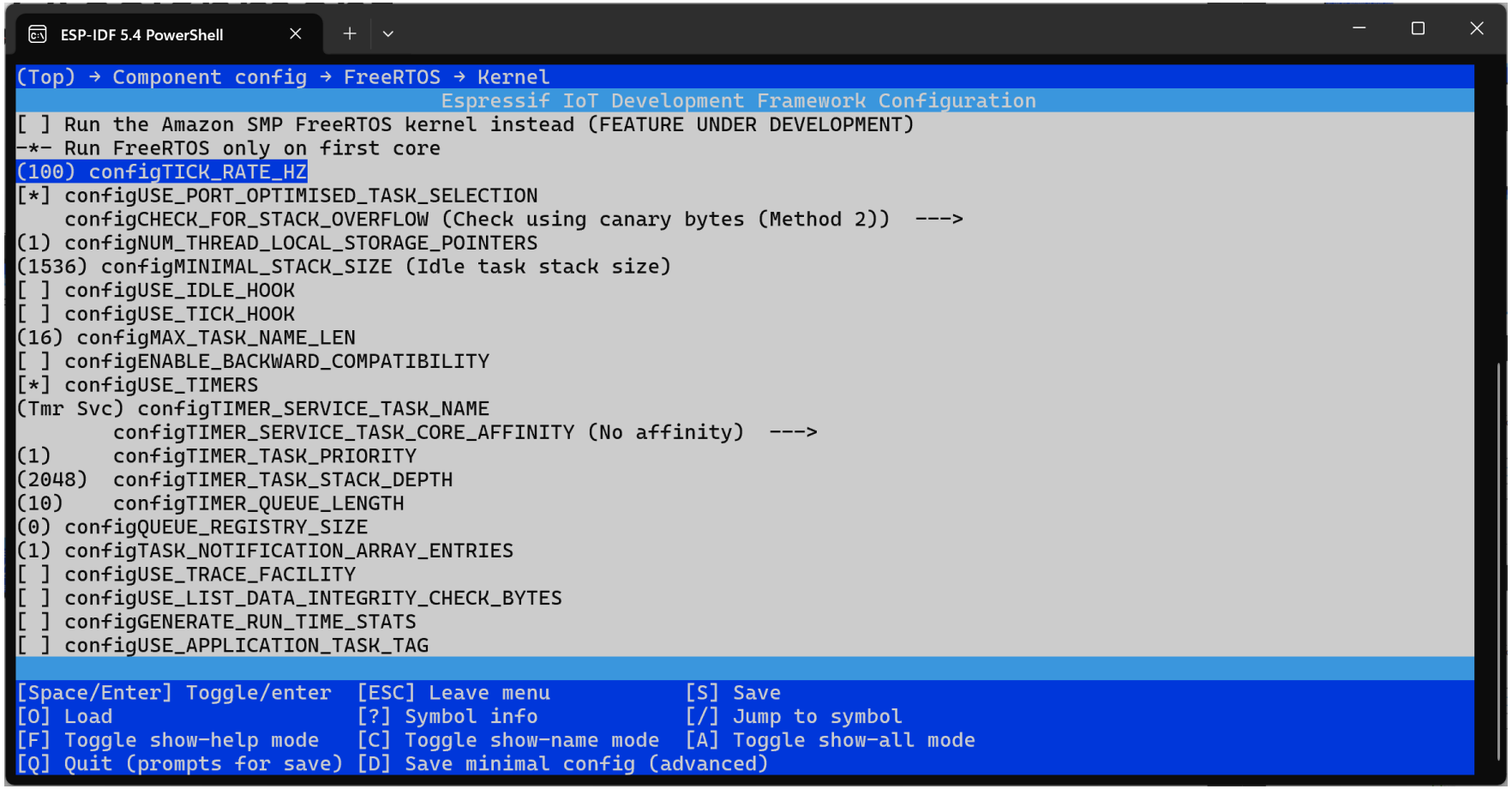
- Where the pointer “p” resides?



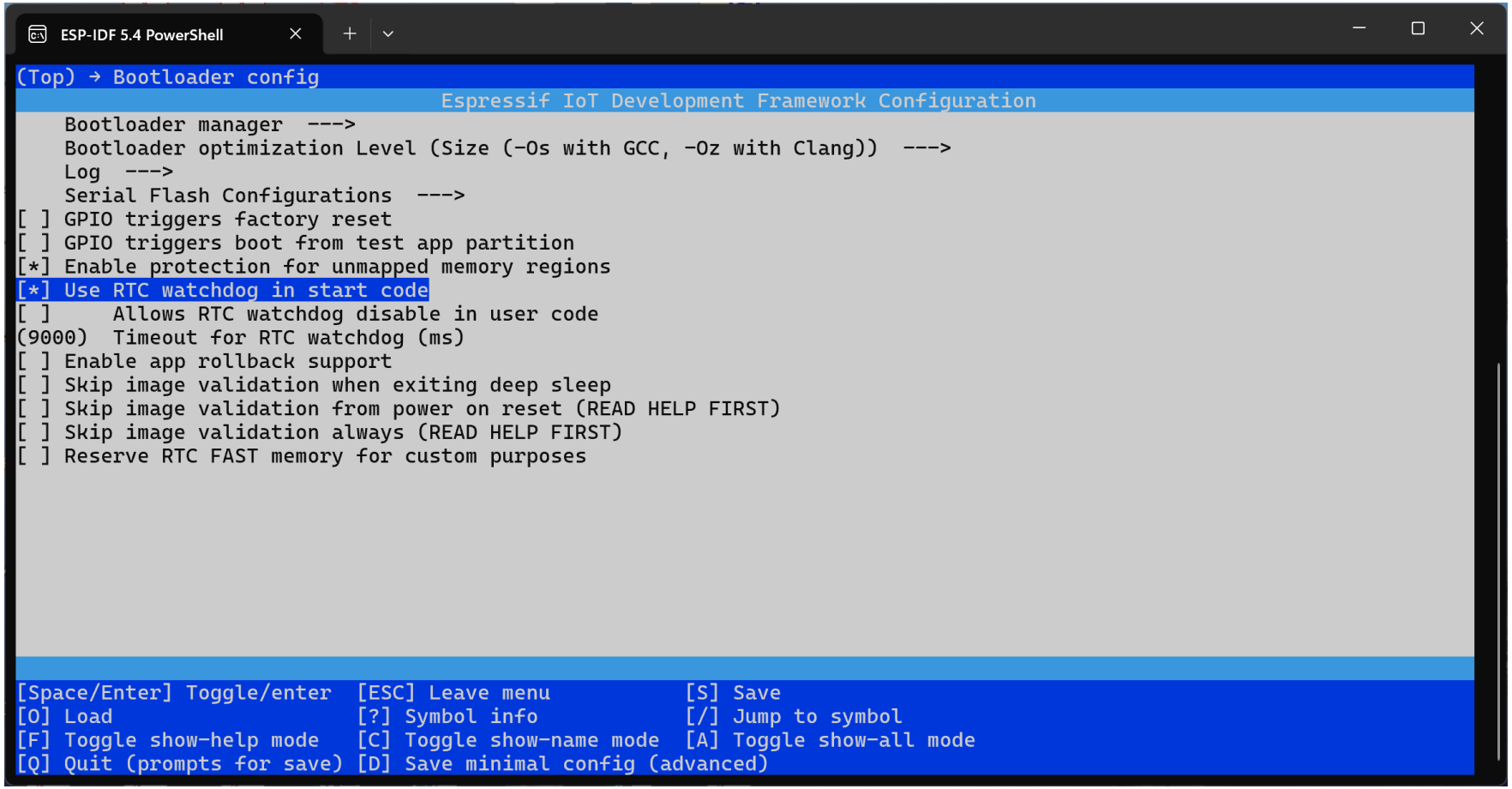
Laboratory Assignment 1 – GPIO Fast Toggling

- Create a copy of the “clean” blink project in your profile folder
- Develop a program that toggles a GPIO as fast as possible (no hardware timers nor PWM generators)
  - Try different approaches and play with the “tick rate” of the FreeRTOS and watchdog options
- Compile and test the project with the help of the oscilloscope

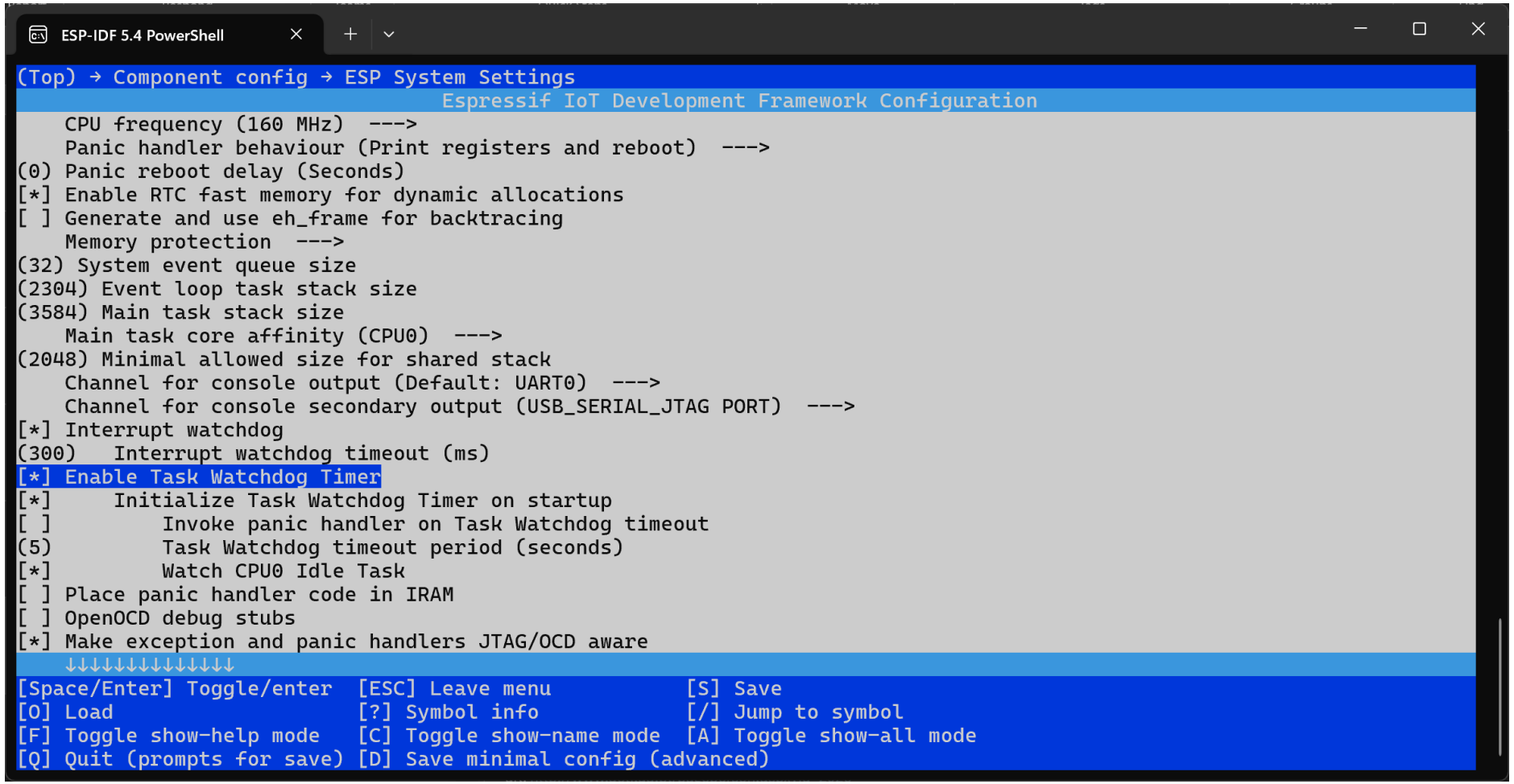
Laboratory Assignment 1 – “menuconfig” Options to Explore - configTICK_RATE_HZ



Laboratory Assignment 1 – “`menuconfig`” Options to Explore – RTC Watchdog



Laboratory Assignment 1 – "`menuconfig`" Options to Explore – Task Watchdog Timer



Laboratory Assignment 2 – Check Compatibility Between ESP32-C3 and TC74

5 Electrical Characteristics

5.1 Absolute Maximum Ratings

Stresses above those listed in Table 5-1 *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only and normal operation of the device at these or any other conditions beyond those indicated in Section 5.2 *Recommended Operating Conditions* is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 5-1. Absolute Maximum Ratings

Parameter	Description	Min	Max	Unit
Input power pins [1]	Allowed input voltage	–0.3	3.6	V
I_{output} [2]	Cumulative IO output current	—	1000	mA
T_{STORE}	Storage temperature	–40	150	°C

[1] For more information on input power pins, see Section 2.5 *Power Supply*.

[2] The product proved to be fully functional after all its IO pins were pulled high while being connected to ground for 24 consecutive hours at ambient temperature of 25 °C.

5.2 Recommended Operating Conditions

For recommended ambient temperature, see Section 1 *ESP32-C3 Series Comparison*.

Table 5-2. Recommended Operating Conditions

Parameter [1]	Description	Min	Typ	Max	Unit
VDDA, VDD3P3, VDD3P3_RTC	Recommended input voltage	3.0	3.3	3.6	V
VDD3P3_CPU [2, 3]	Recommended input voltage	3.0	3.3	3.6	V
VDD_SPI (as input)	—	3.0	3.3	3.6	V
I_{VDD}	Cumulative input current	0.5	—	—	A

[1] See in conjunction with Section 2.5 *Power Supply*.

[2] If writing to eFuses, the voltage on VDD3P3_CPU should not exceed 3.3 V as the circuits responsible for burning eFuses are sensitive to higher voltages.

[3] If VDD3P3_CPU is used to power VDD_SPI (see Section 2.5.2 *Power Scheme*), the voltage drop on R_{SPI} should be accounted for. See also Section 5.3 *VDD_SPI Output Characteristics*.

5.3 VDD_SPI Output Characteristics

Table 5-3. VDD_SPI Internal and Output Characteristics

Parameter	Description [1]	Typ	Unit
R_{SPI}	VDD_SPI powered by VDD3P3_CPU via R_{SPI} for 3.3 V flash_CPU [2]	7.5	Ω

[1] See in conjunction with Section 2.5.2 *Power Scheme*.

[2] VDD3P3_CPU must be more than *VDD_flash_min* + *I_flash_max* * R_{SPI};
where

- *VDD_flash_min* – minimum operating voltage of flash_CPU
- *I_flash_max* – maximum operating current of flash_CPU

5.4 DC Characteristics (3.3 V, 25 °C)

Table 5-4. DC Characteristics (3.3 V, 25 °C)

Parameter	Description	Min	Typ	Max	Unit
C_{IN}	Pin capacitance	—	2	—	pF
V_{IH}	High-level input voltage	0.75 × VDD [1]	—	VDD [1] + 0.3	V
V_{IL}	Low-level input voltage	–0.3	—	0.25 × VDD [1]	V
I_{IH}	High-level input current	—	—	50	nA
I_{IL}	Low-level input current	—	—	50	nA
V_{OH} [2]	High-level output voltage	0.8 × VDD [1]	—	—	V
V_{OL} [2]	Low-level output voltage	—	—	0.1 × VDD [1]	V
I_{OH}	High-level source current (VDD [1] = 3.3 V, V_{OH} >= 2.64 V, PAD_DRIVER = 3)	—	40	—	mA
I_{OL}	Low-level sink current (VDD [1] = 3.3 V, V_{OL} = 0.495 V, PAD_DRIVER = 3)	—	28	—	mA
R_{PU}	Internal weak pull-up resistor	—	45	—	kΩ
R_{PD}	Internal weak pull-down resistor	—	45	—	kΩ
V_{IH_nRST}	Chip reset release voltage CHIP_EN voltage is within the specified range)	0.75 × VDD [1]	—	VDD [1] + 0.3	V
V_{IL_nRST}	Chip reset voltage (CHIP_EN voltage is within the specified range)	–0.3	—	0.25 × VDD [1]	V

[1] VDD – voltage from a power pin of a respective power domain.

[2] V_{OH} and V_{OL} are measured using high-impedance load.

ESP32-C3 Series - Datasheet @ Pages 53-54

Laboratory Assignment 2 – Check Compatibility Between ESP32-C3 and TC74

TC74

1.0 ELECTRICAL CHARACTERISTICS

1.1 Absolute Maximum Ratings†

Supply Voltage (V_{DD}) ..... +6V
Voltage On Any Pin ..... (GND – 0.3V) to (V_{DD} + 0.3V)
Current On Any Pin ..... ±50 mA
Operating Temperature (T_A) ..... -40°C ≤ T_A ≤ +125°C
Storage Temperature (T_{STG}) ..... -65°C to +150°C
Junction Temperature (T_J) ..... +150°C

† Notice: Stresses above those listed under "Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS

Electrical Specifications: Unless otherwise noted, V_{DD} = 3.3V for TC74AX-3.3VXX and V_{DD} = 5.0V for TC74AX-5.0VXX, -40°C ≤ T_A ≤ 125°C. **Note 5**

Parameters	Sym	Min	Typ	Max	Units	Conditions
Power Supply						
Power-on Reset Threshold	V_{POR}	1.2	—	2.2	V	V_{DD} Falling Edge or Rising Edge
Supply Voltage	V_{DD}	2.7	—	5.5	V	**Note 5**
Operating Current	I_{DD}	—	200	350	µA	V_{DD} = 5.5V, **Note 1**
Standby Supply Current	$I_{DD-STANDBY}$	—	5.0	10	µA	V_{DD} = 3.3V Serial Port Inactive, **Note 4**
Temperature-to-Bits Converter						
Temperature Accuracy	T_{ERR}	-2.0 -3.0 —	— — ±2.0	+2.0 +3.0 —	°C	+25°C <T_A < +85°C 0°C < T_A < +125°C -40°C < T_A < 0°C
Conversion Rate	CR	4	8	—	SPS	**Note 2**
Serial Port Interface						
Logic Input High	V_{IH}	0.8 x V_{DD}	—	—	V	
Logic Input Low	V_{IL}	—	—	0.2 x V_{DD}	V	
SDA Output Low	V_{OL}	— —	— —	0.4 0.6	V V	I_{OL} = 3 mA I_{OL} = 6 mA, **Note 3**
Input Capacitance SDA, SCLK	C_{IN}	—	5.0	—	pF	
I/O Leakage	I_{LEAK}	-1.0	0.1	1.0	µA	
Serial Port AC Timing (C_{LOAD} = 80 pF)						
SMBus/I^2C Clock Frequency	f_{SMB}	10	—	100	kHz	
Low Clock Period	t_{LOW}	4.7	—	—	µsec	10% to 10%
High Clock Period	t_{HIGH}	4.0	—	—	µsec	90% to 90%
SMBus/I^2C Rise Time SMBus/I^2C Fall Time	t_R t_F	— —	— —	1000 300	nsec nsec	10% to 90% 90% to10%

Final Remarks

- You must be familiar with the C and C++ programming languages and development tools
- From now on you can use Visual Studio Code with the ESP-IDF extension