**GURU MEDITATION DOCUMENTATION.**

**TITLE: GURU MEDITATION ERROR.**

The targeted audience for this documentation is anyone interested in knowing about Guru Meditation Error in microcontrollers and the solutions to this error.

**EXECUTIVE SUMMARY.**

The Document is intended to explain on the causes of Guru Meditation and the means of solving it.

**SCOPE OF THE DOCUMENTATION.**

The scope of this research and documentation is mainly restricted to Espressif microcontrollers, that is the ESP32 and the ESP 8266.

**LITERATURE REVIEW.**

Below are the causes of Guru Meditation error in Espressif Microcontrollers

* Illegal Instruction
* Instruction Fetch Prohibited
* Store Prohibited, Load Prohibited.
* Integer divide by Zero
* Load Store Alignment
* Load Store error
* Unhandled Debug Exception
* Interrupt Watchdog Timeout on CPU
* Cache disabled but Cache memory region accessed

1. **Illegal Instruction.**

This CPU exception indicates that the instruction which was executed was not a valid instruction. Most common reasons for this error include:

* Free RTOS task function has returned. In Free RTOS, if a task function needs to terminate, it should call ***vTaskDelete()*** and delete itself, instead of returning.
* Failure to read next instruction from SPI flash. This usually happens if:
* Application has reconfigured the SPI flash pins as some other function (GPIO, UART, etc.). Consult the Hardware Design Guidelines and the datasheet for the chip or module for details about the SPI flash pins.
* Some external device has accidentally been connected to the SPI flash pins, and has interfered with communication between ESP32 and SPI flash.

In C++ code, exiting from a non-void function without returning a value is considered to be an undefined behavior. When optimizations are enabled, the compiler will often omit the epilogue in such functions. This most often results in an Illegal Instruction exception. By default, ESP-IDF build system enables ***-Werror=return-type*** which means that missing return statements are treated as compile time errors. However, if the application project disables compiler warnings, this issue might go undetected and the Illegal Instruction exception will occur at run time.

1. **Instruction Fetch Prohibited.**

This CPU exception indicates that the CPU could not read an instruction because the address of the instruction does not belong to a valid region in instruction RAM or ROM.

Usually, this means an attempt to call a function pointer, which does not point to valid code. PC (Program Counter) register can be used as an indicator: it will be zero or will contain a garbage value (not 0x4xxxxxxx).

1. **Load Prohibited, Store Prohibited**

These CPU exceptions happen when an application attempts to read from or write to an invalid memory location. The address which has been written/read is found in the EXCVADDR register in the register dump.

If this address is zero, it usually means that the application has attempted to dereference a NULL pointer.

If this address is close to zero, it usually means that the application has attempted to access a member of a structure, but the pointer to the structure is NULL.

If this address is something else (garbage value, not in 0x3fxxxxxx - 0x6xxxxxxx range), it likely means that the pointer used to access the data is either not initialized or has been corrupted.

1. **Integer Divide By Zero.**

* The application has attempted to do an integer division by zero.

1. **Load Store Alignment.**

This occurs when the application has attempted to read or write a memory location, and the address alignment does not match the load/store size. For example, a 32-bit read can only be done from a 4-byte aligned address, and a 16-bit write can only be done to a 2-byte aligned address.

1. **Load Store Error**

This exception may happen in the following cases:

* If the application has attempted to do an 8- or 16- bit read to, or write from, a memory region which only supports 32-bit reads/writes. For example, dereferencing a *char\** pointer to instruction memory (IRAM, IROM) will result in such an error.
* If the application has attempted to write to a read-only memory region, such as IROM or DROM.

1. **Unhandled Debug Exception.**

The error is accompanied by the below message.

***Debug exception reason: Stack canary watchpoint triggered (task\_name)***

This error indicates that the application has written past the end of the stack of the task with name task\_name. Note that not every stack overflow is guaranteed to trigger this error. It is possible that the task writes to memory beyond the stack canary location, in which case the watchpoint will not be triggered.

1. **Interrupt Watchdog Timeout on CPU.**

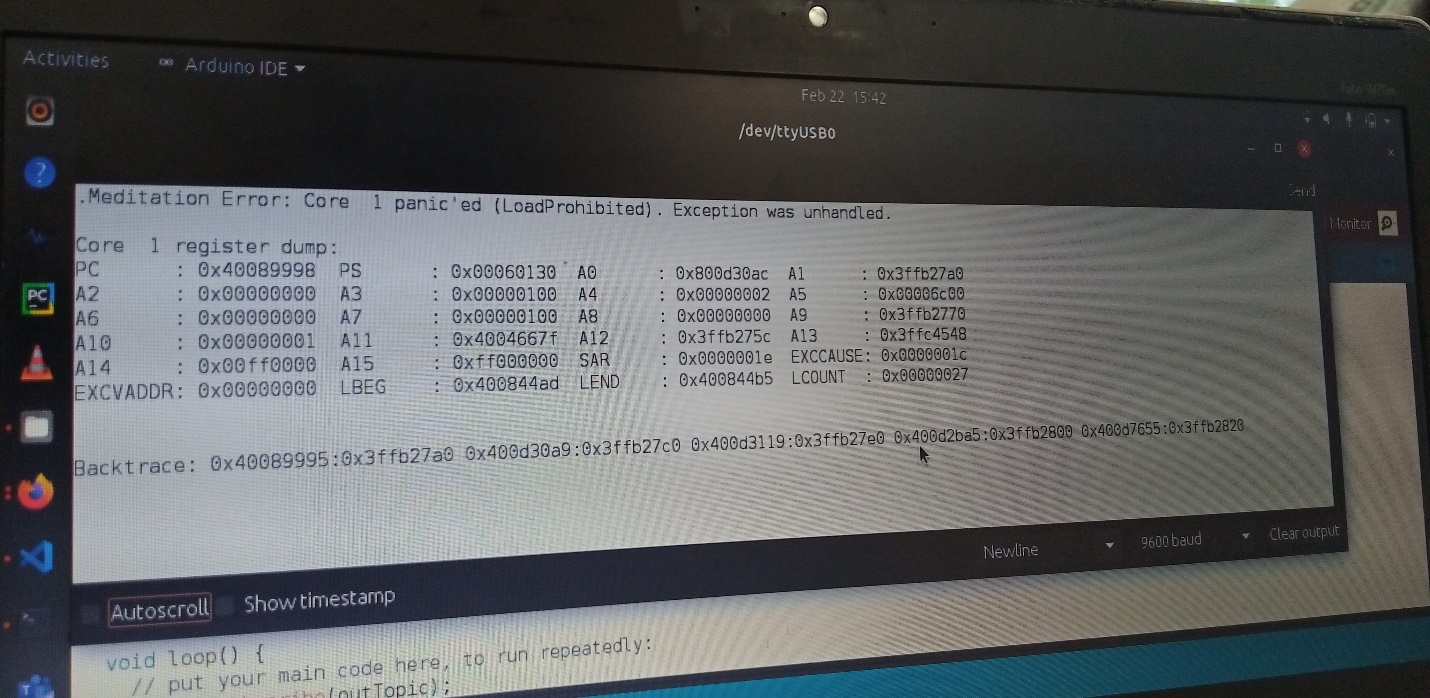
The watchdog timer (WDT) is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur. Generally speaking, a watchdog timer is based on a counter that counts down from some initial value to zero.

The embedded software selects the counter’s initial value and periodically restarts it. If the counter ever reaches zero before the software restarts it, the software is presumed to be malfunctioning and the processor’s reset signal is asserted. The processor (and the embedded software it is running) will be restarted.

As shown below, the WDT is a chip external to the processor. However, it could be included within the same chip as the CPU. This is done in many microcontrollers. In any case, the output of the WDT is tied directly to the Processor’s reset signal.

**Findings and Discussion.**

The error for Guru Meditation was displayed at the Serial Monitor by the following message.



From this error message, the type shown is of “Load Prohibited”.

In our case, looking at the value of the EXCVADDR register address is 0. This means that our code/application attempted to dereference a NULL pointer.

In this case, the NULL pointer is the outTopic variable that is passed to the call back function which then fetches the data from that topic. This caused an error since the outTopic had no value to be read and was NULL, thus the Guru Meditation error arose.

Solution to this problem was to change the parameters of the callback, that is the topic expected, from outTopic to just the variable Topic as is default.

**Conclusion.**

The Guru Meditation error arose from attempt of the application to dereference a NULL pointer. In any setting, to avoid this, use the callback function with its default parameters and do not change the value of the first parameter, which is the Topic.