my MPLAB X IDE doesnot want to skip into functions while debugging

---

If MPLAB X IDE doesn't step into functions while debugging, it could be due to a few reasons. Here are some steps to troubleshoot the issue:

1. Ensure debugging symbols are enabled: Check if the project is compiled with debugging symbols. Go to Project Properties -> XC8/XC16/XC32 (whichever compiler you're using) -> Option categories: Debugging, and make sure the "Debug" level is not set to "None". It should be set to a higher level, like "Level 1" or "Level 2".
2. Proper breakpoints: Ensure that you've set breakpoints at the right locations in your code. Double-check that the breakpoint is set on a line with executable code, and not on a comment or blank line.
3. Source code and object code correlation: Make sure the source code you're debugging matches the object code being executed. If you've made changes to your code since the last build, you need to rebuild the project to ensure the debugging information is updated.
4. Update MPLAB X IDE and compiler: Ensure you're using the latest version of MPLAB X IDE and the appropriate compiler (XC8, XC16, or XC32). Updates may include bug fixes and improvements to the debugging functionality.
5. Reset the debugger: Sometimes, restarting the debugging session can help. Stop the current debugging session, power cycle the target device if needed, and start a new debugging session.
6. Debugger hardware: If you're using an external debugger, like a PICkit or ICD, ensure that it's properly connected to the target device and functioning correctly. Check the connections, pins, and the target device's power supply.
7. Check optimization level: If the optimization level is too high, it may prevent the debugger from stepping into certain functions. In the Project Properties, go to the XC8/XC16/XC32 compiler options and adjust the optimization level to a lower value, like "Level 1" or "Level 2".
8. Debugger limitations: Some debuggers may not support all debugging features, such as stepping into functions, for all devices. Consult the debugger's documentation to see if there are any known limitations.

If you've tried all these steps and the issue persists, it's possible that you've encountered a bug or limitation in the IDE or compiler. You may want to reach out to Microchip support or visit their forums to seek additional assistance.

is MPLAB X single IDE in which I can work with say PIC32MX ?

Yes, MPLAB X IDE is a single integrated development environment that supports a wide range of Microchip devices, including the PIC32MX family of microcontrollers. MPLAB X IDE provides a unified platform for programming, debugging, and simulating Microchip's PIC, dsPIC, and AVR microcontrollers, as well as their 32-bit microcontrollers like the PIC32MX family.

To work with PIC32MX microcontrollers, you'll also need to use the appropriate compiler, which in this case is the MPLAB XC32 C/C++ Compiler. You can download and install the compiler from the Microchip website. Once the compiler is installed, you can create new projects targeting PIC32MX devices or open existing ones in MPLAB X IDE.

Additionally, you can use MPLAB X IDE with various hardware debugging tools like the PICkit, ICD, or REAL ICE. These tools help you program, debug, and test your PIC32MX applications on actual hardware.

I just cant refresh debugging in MPLAB X to work... please former atmel now "microchip studio" is really only for AVR devices ? cant I do in it also say SAMD21G1 ... I am using microchip Curiosity Nano modules

Yes, Microchip Studio, formerly known as Atmel Studio, primarily supports AVR and SAM (ARM-based) microcontrollers, including the SAMD21 family, which you mentioned. Microchip Studio is separate from MPLAB X IDE and provides support for different microcontroller families.

To work with SAMD21G1 microcontrollers, you can use Microchip Studio as your development environment. The Curiosity Nano Development Boards, like the SAMD21 Curiosity Nano, are also compatible with Microchip Studio. You can program, debug, and test your SAMD21-based applications using this IDE and the compatible hardware.

To get started with Microchip Studio and your Curiosity Nano module, follow these steps:

1. Download and install Microchip Studio from the Microchip website: **https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices**
2. Install any required device packs or updates from the Microchip Studio's "Device Pack Manager."
3. Connect your SAMD21 Curiosity Nano board to your computer using a USB cable. The board should be recognized by Microchip Studio.
4. In Microchip Studio, create a new project targeting the SAMD21G1 microcontroller, or open an existing project.
5. Develop your application code, build the project, and upload the compiled code to the Curiosity Nano board using Microchip Studio's programming and debugging tools.

For more information on how to use Microchip Studio and work with Curiosity Nano boards, consult the official Microchip documentation and resources available on their website.

---

while curiosity nano SAMD21 baord shows as kit in window of Microchip Studio, I was unable to select it to project instead of AVR, it is required to create new?

To create a new project for the SAMD21 Curiosity Nano board in Microchip Studio, follow these steps:

1. Open Microchip Studio.
2. Click on "File" in the menu bar, and then select "New" -> "Project."
3. In the "New Project" window, expand the "C/C++" category and select "GCC C Executable Project" or "GCC C++ Executable Project" depending on your preferred language. Click "Next."

4. In the "New GCC Executable Project" window, choose a project name and location. In the "Device Selection" section, search for your target device, which is the SAMD21G1 microcontroller. Click on the appropriate device (e.g., ATSAMD21G18A) and click "Next."
5. In the "Project Configuration" window, you can set the compiler toolchain, which should be "ARM/GNU C/C++ Compiler" for the SAMD21 family. You can also configure other project settings as needed. Click "Finish" to create the project.

Now you should have a new project targeting the SAMD21 microcontroller in Microchip Studio. Make sure your Curiosity Nano board is connected to the computer. You can write your application code, build the project, and upload the compiled code to the board using the programming and debugging tools in Microchip Studio.

If your SAMD21 Curiosity Nano board is connected to your computer, Microchip Studio should recognize the board as a kit. The kit will appear in the "Available Tools" section of the "Start Without Debugging" or "Start Debugging and Break" windows when you're ready to program or debug your application.

MPLAB X debugging doesn't work anymore against Curiosity Nanos ....

If you're experiencing issues with debugging Curiosity Nano boards using MPLAB X IDE, it could be due to a few reasons. Here are some steps you can try to resolve the issue:

1. Check compatibility: Ensure that you're using a compatible version of MPLAB X IDE and the MPLAB X IPE (Integrated Programming Environment) that supports your Curiosity Nano board. If you are using a SAMD21 Curiosity Nano board, note that MPLAB X IDE primarily supports PIC, dsPIC, and AVR devices. For SAMD21, using Microchip Studio is recommended instead.
2. Update MPLAB X IDE and MPLAB X IPE: Make sure you're using the latest version of MPLAB X IDE and MPLAB X IPE, as updates often include bug fixes and support for newer hardware. You can download the latest versions from the Microchip website.
3. Install necessary device packs: Ensure that the required device packs and drivers are installed for your target microcontroller. Go to "Tools" > "Embedded" > "Device Pack

Manager" in MPLAB X IDE and install the necessary device packs for your microcontroller.

4. Check connections: Confirm that your Curiosity Nano board is properly connected to your computer via a USB cable. Make sure the USB cable is functional and connected to the correct USB port on the board.
5. Power cycle the board: Unplug the Curiosity Nano board from the USB port, wait for a few seconds, and then reconnect it. This can help reset the board and resolve any temporary issues.
6. Restart MPLAB X IDE: Close MPLAB X IDE and restart it. This can help clear any temporary glitches that may be affecting the debugging process.
7. Rebuild the project: Rebuild your project to make sure the latest changes are applied and the debugging symbols are up-to-date.

If you continue to experience issues with debugging Curiosity Nano boards using MPLAB X IDE, consider using Microchip Studio (formerly Atmel Studio) as the primary development environment for SAM devices like the SAMD21. Microchip Studio offers better support for these microcontrollers, and you should be able to debug your Curiosity Nano boards without any issues.

---

when launching debug, on breakpoints MPLAB notes me that the breakpoint was not allocated to current main project (I have loaded more projects, for 10 devices, all sharing source code of VMEX) but even after I agree, it cant stop at breakpoints

---

It seems like the issue is related to breakpoints not working correctly in MPLAB X IDE. Here are some possible reasons and solutions to fix the issue:

1. Ensure the active project: Make sure that the correct project is set as the main project in MPLAB X IDE. To set a project as the main project, right-click on the desired project in the "Projects" window and select "Set as Main Project." The main project should appear in bold text.
2. Check the code: Ensure that breakpoints are set on lines containing executable code. Breakpoints set on blank lines or comments may not trigger during debugging.
3. Rebuild the project: Sometimes, breakpoints may not work if the project's compiled code is out of sync with the source code. Rebuild the project by right-clicking on the main

project in the "Projects" window and selecting "Clean and Build." This will ensure that the compiled code and debugging information are up-to-date.

4. Debugger limitations: Some debuggers have limitations on the number of breakpoints that can be set simultaneously. If you have many breakpoints set, try removing some to see if that resolves the issue. Alternatively, you can use the "Run to Cursor" feature in MPLAB X IDE to pause execution at a specific line without setting a breakpoint.
5. Check optimization level: High optimization levels can sometimes interfere with debugging and breakpoints. In the Project Properties, go to the XC8/XC16/XC32 compiler options and adjust the optimization level to a lower value, like "Level 1" or "Level 2."
6. Update MPLAB X IDE and compiler: Ensure you're using the latest version of MPLAB X IDE and the appropriate compiler (XC8, XC16, or XC32). Updates may include bug fixes and improvements to the debugging functionality.
7. Reset the debugger: Sometimes, restarting the debugging session can help. Stop the current debugging session, power cycle the target device if needed, and start a new debugging session.

If you've tried all these steps and the issue persists, it's possible that you've encountered a bug or limitation in the IDE or compiler. You may want to reach out to Microchip support or visit their forums to seek additional assistance.

---

it is stuck in progressbar at statusl line ... Build, Load. ...

If MPLAB X IDE is stuck on the progress bar at the status line during the "Build, Load" process, it could be due to various reasons. Here are a few possible solutions to troubleshoot the issue:

1. Cancel the build process: Click the 'Stop' button or the 'X' button near the progress bar to cancel the build process. This will allow you to regain control over the IDE.
2. Restart MPLAB X IDE: Close MPLAB X IDE and restart it. Sometimes, restarting the application can help resolve temporary glitches.
3. Check for infinite loops or recursion: Review your code for any infinite loops or recursive function calls. These can cause the compiler or debugger to hang during the build

process. Fix any issues you find and try building the project again.

4. Check the project file paths: Ensure that your project files and folders have valid paths and do not contain any special characters or spaces that might be causing issues during the build process. Rename any problematic files or folders and try building the project again.
5. Rebuild the project: Right-click on the main project in the "Projects" window and select "Clean and Build." This will remove any temporary files and build the project from scratch, which may help resolve issues during the build process.
6. Update MPLAB X IDE and compiler: Make sure you're using the latest version of MPLAB X IDE and the appropriate compiler (XC8, XC16, or XC32). Updates may include bug fixes and improvements to the build process.
7. Check your computer's resources: Ensure that your computer has enough resources (CPU, memory, and disk space) to handle the build process. If your system is running low on resources, close other applications and try building the project again.
8. Inspect build output: Review the build output in the "Output" window for any error messages or warnings. These can provide valuable information about what might be causing the build process to hang.

If you've tried all these steps and the issue persists, consider reaching out to Microchip support or visiting their forums to seek additional assistance. There might be a bug or limitation in the IDE or compiler that requires further investigation.

---

everything tried, excuse me ale je to prostě na hovno ....

I understand that you're frustrated with the situation, and I apologize for any inconvenience this has caused. If you've tried all the suggested troubleshooting steps and are still experiencing issues with MPLAB X IDE, it may be helpful to seek additional support from the Microchip community.

You can reach out to Microchip support through their support portal: https://support.microchip.com/ or visit their forums: https://www.microchip.com/forums/ to ask for help from other users who might have encountered similar issues.

Another option is to look for alternative development environments that support your microcontroller. Some users have had success with third-party IDEs, such as Eclipse or Visual Studio Code, using appropriate plugins or extensions. These alternative environments may provide a more stable debugging experience and can sometimes work around issues with the official IDE.

Remember to back up your project files and settings before trying any new development environment, and take note of the configuration settings required for your specific microcontroller and hardware tools.

---

how to add to new project in TI code composer studio (eclipse) reference to source files as links?

In TI Code Composer Studio (CCS), which is based on Eclipse, you can add source files to your project as links, rather than physically copying them into the project folder. This can be useful when you want to maintain a single copy of shared source files across multiple projects.

Here's how to add source files as links in Code Composer Studio:

1. Open Code Composer Studio and either create a new project or open an existing one.
2. In the "Project Explorer" view, right-click on the project folder (or a subfolder where you want to add the source files) and select "New" > "File" or "New" > "Folder" depending on your preference.
3. In the "New File" or "New Folder" dialog, click on the "Advanced" button to expand the advanced settings.
4. Check the option "Link to file in the file system" (for files) or "Link to folder in the file system" (for folders).
5. Use the "Browse" button to locate the source file or folder you want to link to. You can also manually enter the file or folder path in the "Location" text box. Make sure the path is correct.
6. Click "Finish" to create the linked file or folder in your project.

The linked source file(s) will now appear in your project, and any changes made to the original file(s) will be reflected in the linked files within your project.

Keep in mind that if you move the original source files, the links in your project may break, and you'll need to update the links to point to the new location. Additionally, when sharing projects with other developers or using version control systems, make sure that everyone has access to the shared source files, or consider using a relative path for the links, if possible.

---

how to get from CCS the generated code/data size?

In Code Composer Studio (CCS), you can find the generated code and data size of your compiled project by checking the output of the linker in the build log. The linker output provides information about the memory usage of your application.

Here's how to get the generated code/data size from CCS:

1. Build your project by right-clicking on the project name in the "Project Explorer" and selecting "Build Project."
2. After the build is complete, open the "Console" view in CCS. You can find it in the lower part of the IDE by default. If you don't see it, go to "Window" > "Show View" > "Console."
3. In the "Console" view, select the "Build" tab to view the build log.
4. Scroll through the build log until you find the linker output. You should see a section similar to this:

makefile Copy code

```makefile
<Linker Output> Memory region Used Size Region Size %age Used PROGRAM_FLASH: 11228 B 256 KB 4.29% DATA_FLASH: 0 GB 4 KB 0.00% SRAM: 456 B 64 KB 0.70%
```

In this example, the "PROGRAM_FLASH" represents the code size, while the "SRAM" represents the data size. The used size and percentage used are displayed for each memory region. Keep in mind that the names of the memory regions and their sizes may vary depending on your target device and linker configuration.

If you're using a custom linker script or have different memory regions defined, you may need to refer to your script or device documentation to understand the specific memory regions and their usage.

Additionally, you can generate a more detailed memory usage report by enabling the generation of a MAP file in your project settings. This file will contain more information about the memory usage by individual functions and data sections. To enable the generation of a MAP file, go to "Project" > "Properties" > "Build" > "Settings" > "Tool Settings" > "Linker" > "General Options" and check the option "Generate MAP File." Build your project again, and you'll find the MAP file in the project's output folder.

---

well, I dont like eclipse ... where to find how to comment section of code, where are set keybindings for commands?

I understand that Eclipse may not be everyone's preference, but it does offer powerful features and customization options. To comment out a section of code and access keybindings for commands in Eclipse (or Code Composer Studio based on Eclipse), follow these steps:

1. Comment/Uncomment a section of code:
   a. Select the block of code you want to comment or uncomment.
   b. Use the following keyboard shortcuts to comment or uncomment the code:

   markdown   Copy code

   ```markdown
   - For single-line comments (C/C++ style): - Comment: `Ctrl` + `/` -
   Uncomment: `Ctrl` + `/` (same as comment) - For block comments (C/C++ style):
   - Comment: `Ctrl` + `Shift` + `/` - Uncomment: `Ctrl` + `Shift` + `\`
   ```

2. Keybindings for commands:

You can customize the keybindings for commands in Eclipse by accessing the "Keys" preferences:

a. Go to "Window" > "Preferences" (In Code Composer Studio, it's "Edit" > "Preferences").

b. In the "Preferences" dialog, navigate to "General" > "Keys".

c. In the "Keys" preferences, you can search for a specific command using the "Search" box. For example, type "Comment" to find the keybindings for the comment/uncomment commands.

d. To change a keybinding, click on the command you want to modify, and then click on the "Binding" box. Press the new key combination you want to assign to the command. If there's a conflict with an existing keybinding, Eclipse will notify you.

e. Click "Apply" or "Apply and Close" to save your changes.

By following these steps, you can quickly comment or uncomment sections of code and customize the keybindings for various commands in Eclipse or Code Composer Studio.

---

okay, I will immediatelly set keybindings to my standard I have shared here in 5 IDEs... but why the "comment section" isnt anwhere in Edit menu? how I can find such things in menus?

You're right; the "Comment" and "Uncomment" options are not directly available in the main "Edit" menu in Eclipse. However, they are available as toolbar buttons in the default "Source" toolbar or the right-click context menu when working with source code.

Here's how to access these options:

1. Source Toolbar:
   If the "Source" toolbar is visible, you can find the "Comment" and "Uncomment" buttons on this toolbar. If it's not visible, follow these steps to enable it:
   a. Go to "Window" > "Perspective" > "Customize Perspective...".
   b. In the "Customize Perspective" dialog, switch to the "Toolbar Visibility" tab.
   c. Scroll down the list and check the box next to "Source".

   d. Click "OK" to apply the changes.
   Now you should see the "Source" toolbar with "Comment" and "Uncomment" buttons. You can click on these buttons to comment or uncomment the selected lines of code.
2. Context Menu:
   Another way to access the "Comment" and "Uncomment" options is through the context menu:
   a. Right-click on the source code editor where you want to comment or uncomment code.
   b. In the context menu, go to "Source" > "Add Block Comment" or "Remove Block Comment" to comment or uncomment the selected lines of code.

As for finding specific menu items in Eclipse, the easiest way is to use the "Quick Access" search feature:

1. In the top-right corner of the Eclipse window, you'll see a search box labeled "Quick Access".
2. Click on the "Quick Access" search box and type the name of the command or menu item you're looking for. For example, type "Comment" or "Uncomment".
3. The search results will show relevant commands and actions. You can click on an item in the search results to execute the command directly.

Using the "Quick Access" search feature can help you quickly locate and access various menu items and commands in Eclipse.

---

ya, I see it now, thanks

You're welcome! I'm glad I could help. If you have any more questions or need further assistance with Eclipse or any other IDE, feel free to ask. Good luck with your projects!

---

in fact, CCS has nice and detailed Memory Allocation window/pane, but I have still issues with available ram on my device, probably as I introduced/refactored tests to contain its own local array of test code ... its really not needed but z88dk complained here while compiling the same identifier ... I would be glad to reuse this array anyway.... ummm, now confused by memory allocation even if I limited tests to minimum (not have it by defines)

It sounds like you're facing memory constraints on your device and want to optimize the RAM usage. There are a few strategies you can consider to reduce RAM usage and manage memory allocation more efficiently:

1. Use const or PROGMEM for constant data: If your tests use constant data, such as strings or lookup tables, you can declare them as `const` or use the `PROGMEM` attribute (if available on your platform) to store them in program memory (Flash) instead of RAM.
2. Reuse memory for test code: If you have multiple tests that can be executed sequentially, you can reuse the same memory space for the test code. You can implement this by creating a single test buffer or array, and load each test's data into this shared buffer when needed. This way, you'll only allocate memory for one test at a time.
3. Use dynamic memory allocation: Instead of using statically allocated arrays, you can use dynamic memory allocation (e.g., `malloc`, `calloc`, `realloc`, and `free` functions) to allocate memory for your test code only when needed. This allows you to manage memory usage more efficiently but requires careful handling of memory allocation and deallocation to avoid memory leaks or fragmentation.
4. Optimize data structures: Review your data structures and see if you can use more efficient representations or reduce the size of your data. For example, you can use bitfields, packed structures, or smaller data types (e.g., `int8_t` instead of `int32_t`) where possible.
5. Use function calls instead of inlining: If you're using inline functions or macros, consider replacing them with regular function calls. Inlining can lead to code duplication and increased RAM usage, especially if the inlined code is large.
6. Reduce stack usage: Be mindful of the stack usage in your functions. Local variables are typically allocated on the stack, and deep recursion or large local arrays can quickly consume available RAM. Consider using global or static variables, or dynamically allocating memory when needed, to reduce stack usage.
7. Compiler optimization: Make sure your compiler is configured to optimize for size (`-Os` flag). This will instruct the compiler to generate smaller and more efficient code, potentially reducing RAM usage.

By applying these strategies, you can potentially reduce RAM usage and better manage memory allocation in your project. It's essential to carefully review your code and memory usage to determine which techniques are suitable for your specific case.

Additionally, you can use the Memory Allocation window/pane in CCS to analyze the memory usage of your application, identify areas where optimization is needed, and monitor the effects of your changes on memory usage.

---

well, that array is already const, how can I use PROGMEM ?

Using **`PROGMEM`** is a technique for storing constant data in program memory (Flash) instead of RAM on microcontrollers, particularly AVR-based ones. If you're working with an AVR microcontroller, you can use the **`PROGMEM`** keyword to store your constant data in program memory.

Here's how you can use **`PROGMEM`**:

1. Include the required header:

c Copy code

```c
#include <avr/pgmspace.h>
```

I am now testing on MSP430 with 16kB flash and 512byte RAM

For MSP430 microcontrollers, you can store constant data in Flash memory instead of RAM using the **`const`** keyword along with a specific section attribute. The MSP430 doesn't use **`PROGMEM`** like the AVR microcontrollers.

Here's how you can store constant data in Flash memory on MSP430:

1. Declare your constant data array with the **`const`** keyword and specify the section attribute:

c Copy code

```c
const uint8_t myData[] __attribute__((section (".rodata"))) = { /* your data */
};
```

This tells the compiler to place the constant data in the read-only data (**`.rodata`**) section, which is typically stored in Flash memory.

2. Access the constant data as usual in your code:

c Copy code

```c
uint8_t value = myData[index];
```

By placing constant data in Flash memory, you can save RAM space on your MSP430 microcontroller. Note that accessing data in Flash memory may be slower than accessing data in RAM, so consider this trade-off when optimizing your memory usage.

Additionally, you can review the linker script and memory map for your specific MSP430 device to ensure that the **`.rodata`** section is placed in Flash memory. Check the documentation for your device and toolchain for more information on memory layout and linker scripts.

---

well, so I can introduce some READONLY define selectivelly placing such atttribute in code for MSP430 ... or something other elsewhere

---

Yes, you can create a platform-specific macro that sets the appropriate attribute for storing constant data in Flash memory for different microcontrollers. This way, you can make your code more portable and easier to maintain when switching between platforms.

Here's an example of how you can create a **`READONLY`** macro for both MSP430 and AVR microcontrollers:

c Copy code

```c
// Include the required headers #if defined(__MSP430__) #include <msp430.h>
#elif defined(__AVR__) #include <avr/pgmspace.h> #endif // Define the READONLY
macro based on the target platform #if defined(__MSP430__) #define READONLY
__attribute__((section(".rodata"))) #elif defined(__AVR__) #define READONLY
PROGMEM #else #define READONLY #endif // Use the READONLY macro when declaring
constant data const uint8_t myData[] READONLY = { /* your data */ };
```

This code snippet checks which platform you are targeting and defines the **`READONLY`** macro with the appropriate attribute for storing constant data in Flash memory. For MSP430, it uses the **`.rodata`** section attribute, while for AVR, it uses the **`PROGMEM`** keyword. If neither platform is defined, the **`READONLY`** macro will have no effect.

You can use this **`READONLY`** macro throughout your code to ensure that constant data is stored in Flash memory on both MSP430 and AVR microcontrollers. Make sure to adjust your code for accessing the constant data accordingly, particularly for AVR platforms where you may need to use special functions like **`pgm_read_byte()`** to read data from Flash memory.

---

you are smart, too)) thanks

Thank you for the kind words! I'm glad I could help. If you have any more questions or need assistance with your projects, feel free to ask. Good luck with your work!

---

umm "attribute section doesnt apply to automatic variables" ....

I apologize for the confusion earlier. You're right, the `__attribute__((section(".rodata")))` applies to global and static variables but not to local (automatic) variables.

If you want to store constant data in Flash memory, you should define your array as a global or static variable outside of any function. Here's an example:

c

Copy code

```c
// Include the required headers #if defined(__MSP430__) #include <msp430.h>
#elif defined(__AVR__) #include <avr/pgmspace.h> #endif // Define the READONLY
macro based on the target platform #if defined(__MSP430__) #define READONLY
__attribute__((section(".rodata"))) #elif defined(__AVR__) #define READONLY
PROGMEM #else #define READONLY #endif // Use the READONLY macro when declaring
constant data as a global or static variable const uint8_t myData[] READONLY = {
/* your data */ }; void myFunction() { // Access myData as needed uint8_t value
= myData[index]; }
```

In this example, `myData` is declared as a global variable, so the section attribute should work correctly. If you need to use a constant array within a specific function, you can declare it as a `static` variable inside that function, which will also allow the section attribute to be applied:

c

Copy code

```c
void myFunction() { static const uint8_t myData[] READONLY = { /* your data */
}; // Access myData as needed uint8_t value = myData[index]; }
```

By declaring your constant array as a global or static variable, you should be able to use the `READONLY` macro to store the data in Flash memory on both MSP430 and AVR microcontrollers.

---

umm, still CCS complains mostly by this:

Description	Resource	Path	Location	Type
<a href="file:/D:/~app/~tools/~hw-

dev/ti/ccs1210/ccs/tools/compiler/dmed/HTML/10099.html">#10099-D</a> program will not fit into available memory, or the section contains a call site that requires a trampoline that can't be generated for this section. run placement with alignment fails for section ".data" size 0x240. Available memory ranges: lnk_msp430g2553.cmd /VMEX line 91 C/C++ Problem

---

The error you're encountering indicates that the program doesn't fit into the available memory on your MSP430G2553 microcontroller. The linker script (lnk_msp430g2553.cmd) is unable to place the `.data` section within the available memory ranges. The `.data` section typically contains initialized data (global and static variables with an initial value).

Here are a few suggestions to resolve this issue:

1. Review your memory usage: Analyze the memory usage of your program and identify any large data structures or arrays that can be optimized or removed. You can use the Memory Allocation window/pane in CCS or check the output of the linker to find memory usage details.
2. Use uninitialized data section (`.bss`): If you have large arrays or data structures that don't need to be initialized with specific values, you can move them to the `.bss` section, which holds uninitialized data. This can save space in the Flash memory. Declare your variable with the `__attribute__((section(".bss")))` attribute and remove any initial value:

   c Copy code

   ```c
   uint8_t myLargeArray[1024] __attribute__((section(".bss")));
   ```

3. Double-check the `READONLY` macro usage: Ensure that you have applied the `READONLY` macro correctly to all constant arrays and data structures, so they are placed in the `.rodata` section (Flash memory) rather than the `.data` section.
4. Review the linker script: Check the linker script (lnk_msp430g2553.cmd) for any issues or misconfigurations in the memory layout. You can compare it with the default linker script provided by TI for your specific device to see if there are any discrepancies.

If you have already optimized your memory usage and applied the `READONLY` macro correctly, you might be approaching the memory limits of the MSP430G2553 microcontroller. In this case, you might need to consider using a microcontroller with more memory or offloading some data to external storage if possible.

there is enough of flash, I am sure, problem is with RAM, but I already limited everything...

If you're confident that there's enough Flash memory available and the issue is with RAM, you can try the following steps to address the issue:

1. Optimize stack usage: Be mindful of local variables and stack usage in your functions, as they can consume a significant amount of RAM. Reduce the number of local variables and avoid deep recursion. You can also consider using global or static variables or dynamically allocating memory when necessary.
2. Reduce global and static variables: Minimize the number of global and static variables in your code. If some variables are used only within specific functions, consider making them local variables instead.
3. Use smaller data types: Optimize your data structures and variables by using smaller data types (e.g., `int8_t` instead of `int32_t`) where possible.
4. Use function calls instead of inlining: If you're using inline functions or macros, consider replacing them with regular function calls. Inlining can lead to code duplication and increased RAM usage, especially if the inlined code is large.
5. Compiler optimization: Ensure your compiler is configured to optimize for size (`-Os` flag) or optimize for both size and speed (`-O2` flag). This will instruct the compiler to generate smaller and more efficient code, potentially reducing RAM usage.
6. Investigate the linker map file: The linker map file provides detailed information about memory allocation for your project. You can review this file to identify sections or symbols that consume a significant amount of RAM. In CCS, you can enable the generation of the linker map file in the project settings under "C/C++ Build" -> "MSP430 Linker" -> "File Search Path" -> "Generate linker map file."

If you have already optimized your RAM usage as much as possible and still face issues, it might be necessary to consider using a microcontroller with more RAM or offloading some data to external storage if possible. Additionally, ensure that the linker script is correctly configured to allocate all available RAM for your specific device.

well, I would really like to use single declared array and reuse it, but I need to declared it only ONCE (because of z888dk) and then initialize it / fill it with some initializer ... is possible to REinitialize array in C ?

Yes, you can reinitialize an array in C. Since you want to declare the array only once, you can declare it as a global array and then reinitialize it with different values when needed. Here's an example of how you can do this:

c Copy code

```c
#include <stdint.h> #include <string.h> // Declare the global array uint8_t
myArray[10]; // Function to initialize the array with specific values void
initializeArray(const uint8_t *initializer, size_t length) { memcpy(myArray,
initializer, length); } int main() { // Initializer data const uint8_t
initializer1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; const uint8_t initializer2[] =
{10, 9, 8, 7, 6, 5, 4, 3, 2, 1}; // Initialize the array with initializer1
initializeArray(initializer1, sizeof(initializer1)); // Perform some operations
on the array // Reinitialize the array with initializer2
initializeArray(initializer2, sizeof(initializer2)); // Perform some other
operations on the array return 0; }
```

In this example, we declare a global array **`myArray`** and define a function **`initializeArray()`** that takes an initializer and its length as arguments. The function uses **`memcpy()`** to copy the contents of the initializer to the global array. You can call **`initializeArray()`** whenever you need to reinitialize the array with different values.

Remember that this approach might not be the most efficient in terms of performance, especially if the array is large, as it requires copying the initializer's data into the array. However, it does allow you to reuse the same array for different data while declaring it only once.

how to speed up debugging, loading in CCS for MSP430?

Debugging and loading speed in Code Composer Studio (CCS) for MSP430 can be influenced by various factors, such as the size of your program, the speed of your computer, and the debugger configuration. Here are some suggestions to help speed up debugging and loading in CCS:

1. Optimize your code: Reduce the size of your code by optimizing functions, removing unnecessary code, and making use of compiler optimizations. Smaller code will take less time to load and debug.
2. Use compiler optimizations: Ensure that your compiler is configured to optimize for size (use the `-Os` flag) or optimize for both

---

no, I mean communication with the board, some debugging protocol speed

---

I understand now. To speed up the communication between the debugger and the MSP430 board, you can try the following steps:

1. Increase the JTAG clock speed: The JTAG clock speed determines the communication speed between the debugger and the target device. In CCS, you can change the JTAG clock speed by going to "Run" > "Debug Configurations" > "Target" > "Connection Options" > "Advanced". Here, you can increase the JTAG clock speed to a higher value. Keep in mind that some boards may not support very high JTAG clock speeds, so you may need to experiment to find the optimal speed for your setup.
2. Disable the "Verify Download" option: By default, CCS verifies the downloaded code against the target memory to ensure that the data is correct. This process can take additional time. To disable this option, go to "Run" > "Debug Configurations" > "Target" > "Program" and uncheck the "Verify Download" box.
3. Limit the number of breakpoints: Each breakpoint added to your code can slow down the debugging process. Try to use as few breakpoints as possible and remove any unnecessary ones.

4. Use a faster development board or debugger: Some development boards and debuggers may offer faster communication speeds than others. If possible, consider upgrading to a faster board or debugger to improve the debugging speed.
5. Update CCS and debugger firmware: Ensure that you're using the latest version of CCS and that your debugger's firmware is up-to-date. Newer versions may include optimizations and improvements that can help speed up the debugging process.
6. Optimize your PC's performance: Make sure your PC is running smoothly and isn't bogged down by other resource-intensive tasks. Close unnecessary applications and background processes to free up system resources.

Keep in mind that there may be a trade-off between speed and stability when it comes to debugging. Increasing the JTAG clock speed or disabling the "Verify Download" option may improve the communication speed, but it can also lead to potential issues or instability. It's essential to strike a balance between speed and reliability.

---

umm, surprisingly, few tests go well but then suddenly on next test something hangs on "memset" ... and everything is the same

```
case OPI13_XORI:
{
    const TU8 tpOPI13_XORI[] = {
        0x11,0xAA,0x56,0x78, //addi r10 r10 0x5678
        0x13,0xAA,0x0F,0x0F, //xori r10 r10 0x0F0F0 //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        0x13,0xAA,0x59,0x77, //xori r10 r10 0x05977 //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    };
    memset(&core, 0, sizeof(core));
    testresult = vmex(tpOPI13_XORI, sizeof(tpOPI13_XORI));
    printf("%d : %s\n", testresult, testname); if (testresult!=0) return 1;
}
break;
```

It's difficult to determine the root cause of the issue without more information about your code and the `vmex()` function. However, here are some general troubleshooting steps to help you find and resolve the issue:

1. Verify the **`memset()`** function call: Ensure that the arguments passed to **`memset()`** are correct. In this case, you're using **`&core`** and **`sizeof(core)`**. Make sure that the **`core`** variable is defined correctly and that its size is as expected. If possible, step through the code in the debugger to confirm that **`memset()`** is working as intended.
2. Check the input data: Double-check the values in the **`tpOPI13_XORI`** array to ensure they're correct and don't cause any unexpected behavior in the **`vmex()`** function.
3. Debug the **`vmex()`** function: Step through the **`vmex()`** function in the debugger to see if you can identify any issues or inconsistencies in its behavior when processing the **`tpOPI13_XORI`** array. Pay close attention to how it handles the **`memset()`** call and any other memory-related operations.
4. Test with different input data: Try running your tests with different input data to see if the issue persists. This can help you determine if the problem is specific to the **`tpOPI13_XORI`** array or if it's more general in nature.
5. Inspect your memory allocation: Ensure that there are no memory allocation issues, such as buffer overflows or memory leaks, in your code. Use the Memory Allocation window/pane in CCS to monitor memory usage during the execution of your tests.
6. Review your test setup: Ensure that your test setup is consistent across all tests and doesn't introduce any unexpected side effects. If you're using any global variables or shared resources, make sure they're properly initialized and managed throughout your tests.

By carefully analyzing the code and using the debugger, you should be able to identify the cause of the issue and fix it. If you're still having trouble, please provide more context or code snippets to help diagnose the problem further.

---

how to show consumed memory in IAR embedded workbench for msp430?

In IAR Embedded Workbench for MSP430, you can view the consumed memory by examining the linker map file that gets generated during the build process. The linker map file provides detailed information about memory allocation, including the memory used by different sections of your program. Here's how to access and interpret the linker map file in IAR Embedded Workbench:

1. Build your project: Make sure your project is built successfully without any errors.
2. Open the linker map file: After building the project, the linker map file (usually with a `.map` extension) should be generated in your project's output directory. You can find it in the "Debug" or "Release" folder, depending on your build configuration. Open the `.map` file in a text editor or by double-clicking it in the IAR Embedded Workbench Project Explorer.
3. Analyze the memory usage: In the linker map file, you'll find several sections with information about memory usage. Some important sections to look for are:
    "Module Summary": This section provides an overview of the memory used by each module in your project, including code, data, and constant memory.
    "Memory Configuration": This section shows the memory regions available on your target device and their respective sizes.
    "Global Symbols": This section lists global symbols, such as variables and functions, and their memory addresses.
    "Cross Reference": This section provides information about the relationships between symbols, modules, and memory sections.
4. Calculate memory consumption: To determine the total memory used by your project, look for the "Summary" section at the end of the linker map file. This section summarizes the memory used by each memory region (e.g., FLASH, RAM) and shows the total size used, as well as the remaining free memory. Pay attention to the values for "Program" (code) memory and "Data" (RAM) memory.

By analyzing the linker map file, you can get a comprehensive view of the memory usage in your MSP430 project, allowing you to optimize your code and make informed decisions about memory allocation.

---

how differs MSP430G2553 and MSP430FR2476 ? on the G2553 I see lots of test failing... how they differs?

The MSP430G2553 and MSP430FR2476 are both part of Texas Instruments' MSP430 family of microcontrollers, but they have different features and specifications. Here is a comparison of their key differences:

MSP430G2553:

1. Flash memory: 16 KB
2. RAM: 512 bytes
3. Number of I/O pins: 24
4. 16-bit RISC architecture
5. Analog peripherals: 8-channel, 10-bit ADC
6. Digital peripherals: 2 x 16-bit Timer_A, USCI (UART, SPI, I2C)
7. Power supply: 1.8 V to 3.6 V
8. Built using F-RAM (Ferroelectric RAM) technology

MSP430FR2476:

1. Flash memory: 32 KB (FRAM)
2. RAM: 4 KB
3. Number of I/O pins: 40
4. 16-bit RISC architecture
5. Analog peripherals: 12-channel, 12-bit ADC, 1 x Comparator
6. Digital peripherals: 3 x 16-bit Timer_A, eUSCI (UART, SPI, I2C), RTC, AES256 accelerator
7. Power supply: 1.8 V to 3.6 V
8. Built using FRAM (Ferroelectric RAM) technology, which offers faster write speeds, lower power consumption, and virtually unlimited write endurance compared to traditional Flash memory

The main differences between the two microcontrollers are the amount of memory, number of I/O pins, and the peripheral features they offer. The MSP430FR2476 has more Flash memory (FRAM), more RAM, and additional peripherals compared to the MSP430G2553.

The difference in memory types (FRAM vs. Flash) and the available peripherals could be the reason why you see some tests failing on the MSP430G2553. It's essential to make sure your code is compatible with the specific microcontroller you are using, and you may need to make adjustments to account for these differences.

Keep in mind that when porting code between microcontrollers, you need to ensure that you use the appropriate register names, memory addresses, and peripheral configurations for each device. The best approach is to consult the device-specific datasheets and reference manuals to understand the differences between the two microcontrollers and adjust your code accordingly.

---

well, but why the code fails on one of them so much? they share both the same compiler / ISA or not?

Yes, both MSP430G2553 and MSP430FR2476 microcontrollers are part of the MSP430 family and share the same 16-bit RISC architecture, which means they use the same instruction set architecture (ISA) and can use the same compiler. However, even though they have the same ISA, there can still be differences in their peripherals, memory organization, and other hardware features that can cause code to fail on one device while working on another.

Here are some possible reasons why the code may fail on one of the microcontrollers:

1. Different peripheral configurations: The two microcontrollers have different sets of peripherals, and even when they share similar peripherals, there may be differences in their configurations or the way they operate. Ensure that your code is compatible with the peripherals available on the target microcontroller and that you use the correct register names and addresses for that specific device.
2. Memory organization: The MSP430G2553 and MSP430FR2476 have different memory sizes and types (Flash vs. FRAM). The memory organization in terms of addresses and the available memory regions might be different between the two devices. Make sure that your code uses the correct memory addresses and doesn't exceed the available memory limits on the target device.
3. Interrupt vectors and priorities: The two microcontrollers may have different interrupt vectors, priorities, or handling mechanisms. Make sure that your code uses the correct interrupt settings for the target device and doesn't rely on any specific behavior that might not be present on the other microcontroller.
4. Clock configurations: The clock sources, clock speeds, and clock configurations can vary between the two devices. Ensure that your code is compatible with the clock settings on the target microcontroller and that it correctly configures the clock system.
5. Power management features: The two devices may have different power management features or low-power modes. Make sure your code doesn't rely on specific power

management behaviors that might not be present on the other microcontroller.

To ensure compatibility between the two devices, consult their respective datasheets and reference manuals to understand their differences and adjust your code accordingly. Additionally, you might want to use the device-specific header files, libraries, and driver code provided by Texas Instruments for each microcontroller to help with the porting process.