what is difference between 8051 dlib and clib ?

The 8051 microcontroller family has different C library implementations for IAR Embedded Workbench. The two main libraries are the DLIB and CLIB. Here are the key differences between them:

1. DLIB (Dedicated Library):
   This is the default library provided by IAR Embedded Workbench for 8051.
   It is highly optimized for size and speed and is tailored for embedded systems.
   DLIB is a more efficient and compact library, which results in faster execution and smaller code size.
   The drawback of using DLIB is that it may not be compatible with all standard C library functions, and some functions may behave differently compared to their standard C library counterparts.
2. CLIB (Compact Library):
   CLIB is an alternative library provided by IAR Embedded Workbench for 8051.
   It is designed to be compatible with the ANSI/ISO C standard library.
   CLIB is easier to use if you're migrating code from another platform or if you require full standard C library functionality.
   However, CLIB might not be as optimized for size and speed as DLIB, which could result in larger code size and slower execution.

In summary, DLIB is a more optimized library for embedded systems, while CLIB offers better compatibility with the standard C library. You should choose the library that best fits your project requirements and constraints.

may be that stdint is not in clib ?

I apologize for any confusion. Both DLIB and CLIB in IAR Embedded Workbench for the 8051 microcontroller family should support the stdint.h header file. The stdint.h header file is part of the C99 standard library, which defines a set of fixed-width integer types.

Both DLIB and CLIB aim to be compatible with the C standard, and stdint.h is an essential part of that. If you are experiencing any issues with including stdint.h in your project, make sure that the header file is accessible in the include path and that your project is configured correctly to use the desired library.

---

is possible to somehow replace stdint by custom code?

Yes, it is possible to replace stdint.h with custom code if you have specific requirements that the standard header does not fulfill. To do this, you will need to follow these steps:

1. Create a new header file (e.g., custom_stdint.h) in your project's source or include directory.
2. In the custom_stdint.h file, define the fixed-width integer types and their corresponding limits as per your needs. You can refer to the original stdint.h file for guidance or implement your own custom definitions. For example:

c

Copy code

```c
#ifndef CUSTOM_STDINT_H #define CUSTOM_STDINT_H typedef signed char int8_t; typedef unsigned char uint8_t; typedef signed short int16_t; typedef unsigned short uint16_t; typedef signed int int32_t; typedef unsigned int uint32_t; typedef signed long long int64_t; typedef unsigned long long uint64_t; #define INT8_MIN (-128) #define INT8_MAX 127 #define UINT8_MAX 255 // Add other limits as required #endif // CUSTOM_STDINT_H
```

3. Replace all occurrences of `#include <stdint.h>` with `#include "custom_stdint.h"` in your project's source files. This will ensure that your custom header file is used instead of the standard one.
4. If you need any additional functionality provided by the stdint.h header, such as macros for handling constant expressions of the appropriate width (e.g., `INT16_C`, `UINT32_C`), you will need to implement those in your custom_stdint.h file as well.

Please note that by replacing the standard stdint.h with custom code, you might introduce compatibility issues with third-party libraries or code that depend on the standard definitions. It is recommended to use the standard stdint.h header file unless you have a specific reason to replace it with custom code.

---

what are sizes of short int and long in 8bit C compilers?

In 8-bit C compilers, such as those targeting the 8051 microcontroller family, the sizes of `short`, `int`, and `long` data types typically follow these conventions:

1. `short` (short int): 16 bits (2 bytes)
2. `int`: 16 bits (2 bytes)
3. `long` (long int): 32 bits (4 bytes)

Please note that these sizes are not mandated by the C standard itself, but rather, are common choices made by 8-bit C compiler developers. The C standard only defines minimum size requirements for these data types, stating that `short` must be at least 16 bits, `int` must be at least as large as `short`, and `long` must be at least 32 bits and at least as large as `int`.

To verify the sizes of these data types in your specific compiler, you can use the `sizeof()` operator:

c

Copy code

```c
#include <stdio.h> int main() { printf("Size of short: %zu bytes\n",
sizeof(short)); printf("Size of int: %zu bytes\n", sizeof(int)); printf("Size of
long: %zu bytes\n", sizeof(long)); return 0; }
```

This code will print the sizes of `short`, `int`, and `long` in bytes for your specific compiler and target platform.