Misaligned Address Finder for mikro compilers (16-bit and 32-bit)

Author: VCC
Version: 1.0 (May 2018), Version 1.1 (May 2020)

Misaligned Address Finder is a tool used to generate a report on variables and constants, which are not Word or DWord aligned. Some variables / constants are required to be allocated at even addresses (for 16-bit architectures) and at “multiple by 4” addresses (32-bit architectures), so users can be notified if the compiler doesn’t do that allocation as required. This is to avoid running into address exceptions when working with misaligned pointers. There are variables / constants, which can be allocated at misaligned addresses and do no harm, but the tool can’t figure out which is which. Users will have to decide what identifiers need to be added to the list.

The words “Report” and “Notification” might be used interchangeably throughout this documentation, because they refer to approximately the same content/action. The report is a list of misaligned identifiers (variables / constants), presented as a message dialog and/or saved to file(s). It is called a notification, because users are presented with a report when the application starts and a lst file is received as command line argument.

Features:

- It is started from a mikro compiler as a tool
- Uses the compiler generated lst file as input, to get the list of identifiers (variables / constants)
- Keeps track of multiple lst files (added by user)
- Notifies either about all misaligned identifiers or about a user-defined subset of all identifiers
- Notifies through a message dialog or through two different generated report files
- The report verbosity is controlled by various application settings
- The application settings are stored in a separate ini file than the list of lst files
- The two applications settings files can be passed as command line arguments, if desired
- The application can be part of an automation environment

Requirements / Recommendations:

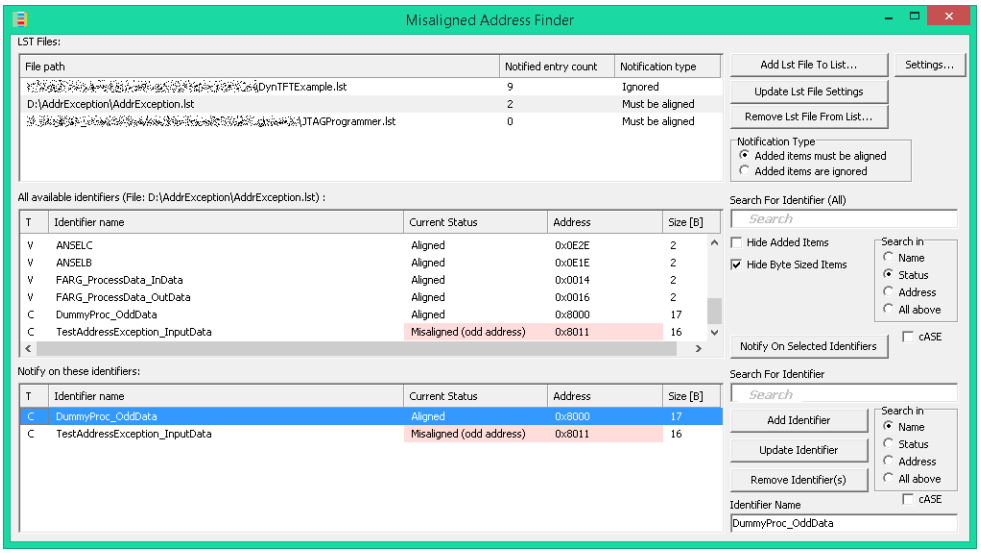
- The application must be run with **"%LIST_FILE_NAME"** parameter from compiler's IDE, including quotes.
- Be careful of what files are passed as command line arguments, because the second and the third arguments are expected to be ini files and they will be overwritten while closing the application.
- An IDE bug deletes the quotes from LIST_FILE_NAME parameter, so there must be at least two parameters, which are correctly saved: **"%LIST_FILE_NAME" dummy** The second one has no quotes and is ignored by the application.

Limitations:

- Neither the list of files nor the lists of identifiers can be sorted.
- No misaligned identifier will be reported by default. Users have to either add them to the list (manually), or to set the application to report on all misaligned identifiers.
- When configured to autoclose, the application will have to be started without command line arguments to allow UI interaction.

Table of contents:

1. General view of the application



The application's main window is split into three sections: a list of lst files – *LST Files*, a list of all identifiers (variables and constants) from the selected lst file – *All available identifiers*, and a list of identifiers, which will be either included or excluded from the generated report – *Notify on these identifiers*. Controlling which identifiers are included and which are excluded, is done by the *Notification Type* radio group box. Only one report can be generated at a time, either by passing the lst file as command line argument to application, or from the *LST Files* list, by right-clicking and selecting *Generate Report*.

Warning: do not pass multiple lst files as subsequent arguments to application's command line, because the second and the third arguments are expected to be custom settings files and they will be overwritten when closing the application.

Lst files can be added to the list in three ways. One is by starting the application with a lst file as the first command line argument, and the application will ask if that file should be added (if not in the list already). To avoid asking, users can pass the keywords `add` or `skipadd` as the fourth command line argument (see Settings chapter about the second and the third arguments). The second is by using the *Add Lst To File List* button. The third way is somehow related to the first, because it allows users to add the file, which they answered with No, when asked if the file should be added. This is done by right-clicking the *All available identifiers* label and selecting *Add this file to list* from the pop-up menu.

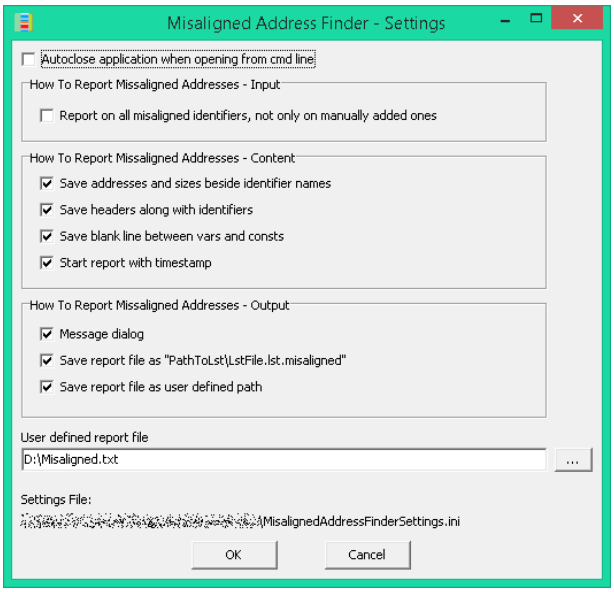
When adding a file, its notification type is decided by the *Notification Type* radio group box. This can be changed later, by selecting the file, setting the desired notification type, then updating from the *Update Lst File Settings* button. Files can be removed from the list, using the *Remove Lst File From List* button. Only one file can be selected at a time.

Notification type can be set to either include the misaligned identifiers in the report (selecting *Added items must be aligned*), or to exclude the misaligned identifiers (selecting Added items are ignored). Every file has its own notification type. Right-click the list of files, for additional file options.

The application allows including all misaligned items in the generated report, or to manually configure a subset of them. This is done from the Settings window (see Settings button). All available identifiers are presented in the second list from the application's main window. This list is refreshed every time a file is selected in the list of files (first list on the window). When the application is started with a lst file as the first command line argument, that file might not be added in the list of files (per user's choice), but its content will be added to the *All available identifiers* list. Clicking a new file in the first list will cause the second list to display its content and discard the initial content. Setting items to be included/excluded from the generated report, is done by selecting them in the *All available identifiers* list and clicking the *Notify On Selected Identifiers* button. When users rename identifiers and recompile, this is reflected in the lst file. When the application reloads a lst file, which contains renamed identifiers, it will display "Not found" on their current status. Identifiers with the "Not found" status will not be reported.

Each of the two lists of identifiers have separate search boxes, which can be configured to look into the name, status or address fields.

2. Settings



Using the *Settings* button, the Settings window will be displayed, allowing users to configure various reporting options. When the *Autoclose application when opening from cmd line* checkbox is checked, the application closes automatically after generating the report. An exception to this is when the application is started without a lst file, sent as command line argument. Also, when the report is generated from the pop-up menu of the list of lst files, the application will not close.

If users don't want to specify which identifiers to be verified, they can check the *Report on all misaligned identifiers, not only on manually added ones* checkbox. If checked, the notification type setting is ignored and all identifiers are verified.

The generated report allows for different content options, in order to facilitate parsing if desired. At a minimum, a report will contain the names of the misaligned identifiers. In addition to these names, the report may also contain a timestamp, three lines of headers for variables and three lines for constants, a blank line as separator between these two categories, and identifier addresses and sizes.

The report itself can be presented as a message dialog and/or saved to files. There can be a different file for each verified lst file, having the same name as the lst file, plus a *.misaligned* extension. This is enabled by checking the *Save report file as "PathToLst\LstFile.lst.misaligned"* checkbox. The second file, which contains the report, has a fixed filename and is user defined. This is enabled from the *Save report file as user defined path* checkbox, and set from the *User defined report file* editbox.

When the application is started with a second command line argument, this is used as a custom settings file and its path is displayed under the *Settings File* label.

3. Settings files

The application uses two separate settings files, both using the ini format. One is for storing the list of lst files and their identifiers (*MisalignedAddressFinder.ini*) and the other is for storing the application settings (*MisalignedAddressFinderSettings.ini*). If the application is started with no command line arguments or with only one argument, the default filenames are used as the settings files, expected to be in the same directory as the executable. When users want to group lst files in different categories with different settings, they can pass custom settings files as command line arguments. The application expects the first argument to be the lst file, the second argument to be the *MisalignedAddressFinderSettings.ini* file, and the third argument to be the *MisalignedAddressFinder.ini* file. If passing paths to non-existent files as the second or third arguments, the application will use default settings files.

The path to the *MisalignedAddressFinderSettings.ini* file is indicated on the Settings window, while the path to *MisalignedAddressFinder.ini* file is indicated as a hint of the list of lst files on the main window.

4. Reported errors

There are few error strings, which may appear in a generated report, if the input file can't be parsed or the report can't be saved. The first two strings are followed by a blank (ASCII 32), then the file name. The third string will be concatenated with the exception message. The strings are:

```
File not added for notifications:
Input file not found:
Exception on saving file:
```

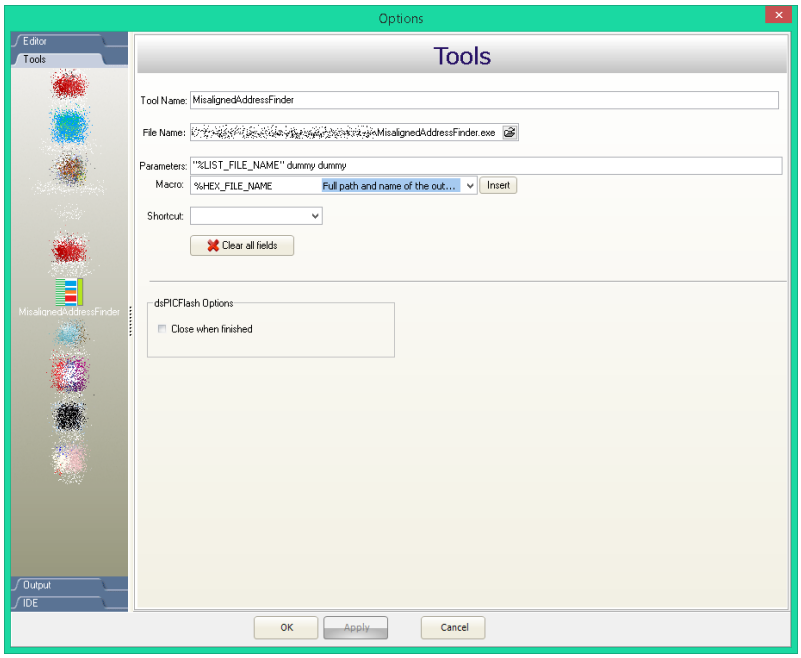
The first one appears when the file, passed as first argument to the application's command line, is not in the list of lst files for some reason. This is more of a debug message than a critical error, because the file can be automatically added if the fourth command line argument is the keyword `add`.

The second error message appears when the lst filename, passed as first argument to the application's command line, is a non-existent file.

The third one appears only on saving report files and is displayed with a message dialog only.

5. Application as a mikroe compiler IDE tool

There are two typical use cases. One is to start the application without any command line arguments, allowing for UI interaction when the autoclose option is enabled, and the other is to pass a lst file, to be notified about misaligned identifiers.



As command line parameters, users should fill in: **"%LIST_FILE_NAME" dummy**
The first parameter must contain quotes, for paths that contain spaces, while the second parameter is used to avoid a bug in the IDE, which makes the quotes disappear. The second parameter is ignored by the application, because it is not a valid file.