[UEFI Practice] SlimBootloader Customization

原创 jiangwei0512

Category Column: UEFI Development Basics Article Tags: slimbootloader uefi edk

摘要 This article details the configuration mechanism of SBL (Slim Bootloader), including the basic configuration defined by YAML files, the overriding configuration of DLT files, and the use of configuration tools. YAML files contain settings such as memory and Silicon, while DLT files are used to customize according to specific boards. The configuration tool ConfigEditor.py provides a graphical interface for loading and editing configurations and generating binary configuration files. Post-build customization allows parameters to be adjusted after the SBL is built.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Overview

One of the features of SBL is Customizable, which is divided into pre-build and post-build customization.

SBL has two types of configuration data, internal configuration data and external configuration data. Internal configuration data is the default data hard-coded in the code, and is used when external data cannot be obtained; external configuration data is generated through predefined files and configuration tools.

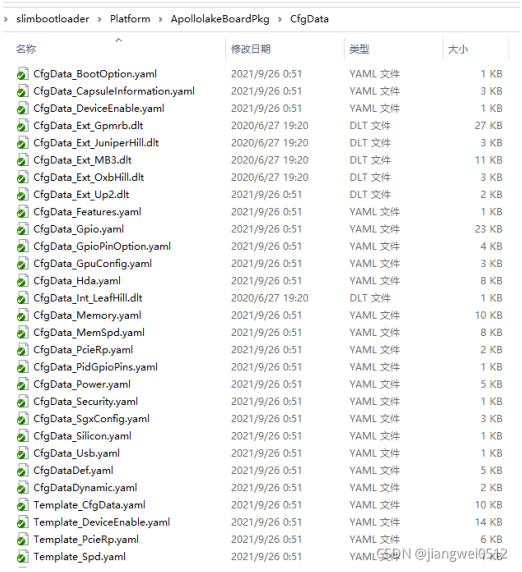
The predefined files and configuration tools in SBL include:

- YAML file: It defines the configuration;
- DLT (delta, which actually means variable data) file: Override data, used to override the configuration in the YAML file;
- Configuration operation tool: usually a Python script.

YAML File

The YAML file includes memory, Silicon, GPIO, startup strategy, security configuration, etc.

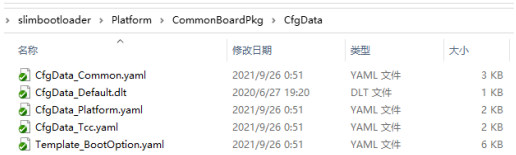
YAML files are usually included in a specific board directory, such as the ApolloLake platform:



› slimbootloader › Platform › ApollolakeBoardPkg › CfgData

名称	修改日期	类型	大小
CfgData_BootOption.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgData_CapsuleInformation.yaml	2021/9/26 0:51	YAML 文件	3 KB
CfgData_DeviceEnable.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgData_Ext_Gpmrb.dlt	2020/6/27 19:20	DLT 文件	27 KB
CfgData_Ext_JuniperHill.dlt	2020/6/27 19:20	DLT 文件	3 KB
CfgData_Ext_MB3.dlt	2020/6/27 19:20	DLT 文件	11 KB
CfgData_Ext_OxbHill.dlt	2020/6/27 19:20	DLT 文件	3 KB
CfgData_Ext_Up2.dlt	2021/9/26 0:51	DLT 文件	2 KB
CfgData_Features.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgData_Gpio.yaml	2021/9/26 0:51	YAML 文件	23 KB
CfgData_GpioPinOption.yaml	2021/9/26 0:51	YAML 文件	4 KB
CfgData_GpuConfig.yaml	2021/9/26 0:51	YAML 文件	3 KB
CfgData_Hda.yaml	2021/9/26 0:51	YAML 文件	8 KB
CfgData_Int_LeafHill.dlt	2020/6/27 19:20	DLT 文件	1 KB
CfgData_Memory.yaml	2021/9/26 0:51	YAML 文件	10 KB
CfgData_MemSpd.yaml	2021/9/26 0:51	YAML 文件	8 KB
CfgData_PcieRp.yaml	2021/9/26 0:51	YAML 文件	2 KB
CfgData_PidGpioPins.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgData_Power.yaml	2021/9/26 0:51	YAML 文件	5 KB
CfgData_Security.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgData_SgxConfig.yaml	2021/9/26 0:51	YAML 文件	3 KB
CfgData_Silicon.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgData_Usb.yaml	2021/9/26 0:51	YAML 文件	1 KB
CfgDataDef.yaml	2021/9/26 0:51	YAML 文件	5 KB
CfgDataDynamic.yaml	2021/9/26 0:51	YAML 文件	2 KB
Template_CfgData.yaml	2021/9/26 0:51	YAML 文件	10 KB
Template_DeviceEnable.yaml	2021/9/26 0:51	YAML 文件	14 KB
Template_PcieRp.yaml	2021/9/26 0:51	YAML 文件	6 KB
Template_Spd.yaml	2021/9/26 0:51	YAML 文件	2 KB

There are also some common YAML files in the Platform\CommonBoardPkg\CfgData directory:



› slimbootloader › Platform › CommonBoardPkg › CfgData

名称	修改日期	类型	大小
CfgData_Common.yaml	2021/9/26 0:51	YAML 文件	3 KB
CfgData_Default.dlt	2020/6/27 19:20	DLT 文件	1 KB
CfgData_Platform.yaml	2021/9/26 0:51	YAML 文件	2 KB
CfgData_Tcc.yaml	2021/9/26 0:51	YAML 文件	2 KB
Template_BootOption.yaml	2021/9/26 0:51	YAML 文件	6 KB

The configuration file entry required by the platform is `CfgDataDef.yaml` , and other configuration files are included as its sub-files `CfgDataDef.yaml` . For example, Platform\ApollolakeBoardPkg\CfgData\CfgDataDef.yaml:

yaml AI generated projects 登录复制

```yaml
## @file
#
#  Slim Bootloader CFGDATA Default File.
#
#  Copyright (c) 2020, Intel Corporation. All rights reserved.<BR>
#  SPDX-License-Identifier: BSD-2-Clause-Patent
#
##

variable:
  COND_GPIO_SKIP                  : ($GPIO_CFG_DATA.$(1)_Half0.GpioSkip == 0)
  COND_GPIO_PID_ENABLE            : ($PID_GPIO_CFG_DATA.$(1).Enable==1) and ($PLATFORMID_CFG_DATA.PlatformId==0)
  COND_PCIE_RP_PWR_PIN_SKIP       : ($PCIE_RP_CFG_DATA.PcieRpPower$(1).Skip == 0)
  COND_PCIE_RP_RST_PIN_SKIP       : ($PCIE_RP_CFG_DATA.PcieRpReset$(1).Skip == 0)
  COND_PCIE_RP_EN                 : ($PCIE_RP_CFG_DATA.PcieRpFeatures$(1).En == 1)
  COND_PCIE_RP_CLK_REQ_SUP        : (($PCIE_RP_CFG_DATA.PcieRpFeatures$(1).ClkReqSup == 1) and ($PCIE_RP_CFG_DATA.PcieRpFeatures$(1).En == 1))
  COND_HDA_EN                     : ($HDA_CFG_DATA.HdaEnable == 1)
  COND_HDA_DSP_EN                 : (($HDA_CFG_DATA.HdaEnable == 1) and ($HDA_CFG_DATA.DspEnable == 1))

template:

  - !include Template_CfgData.yaml

configs:
  - $ACTION      :
      page         : PLT::"Platform", MEM::"Memory Settings", SIL::"Silicon Settings", GEN::"General Settings", GIO::"Gpio Settings", OS::"OS Boot Options"
  - Signature    :

```

```yaml
    length       : 0x04
    value        : {'CFGD'}
  - HeaderLength :
    length       : 0x01
    value        : 0x10
  - Reserved     :
    length       : 0x03
    value        : {0,0,0}
  - UsedLength   :
    length       : 0x04
    value        : _LENGTH_
  - TotalLength  :
    length       : 0x04
    value        : 0x2000

  - !include Platform/CommonBoardPkg/CfgData/CfgData_Platform.yaml

  - $ACTION      :
    page         : IOCUART:PLT:"IOC Uart Settings"
  - $ACTION      :
    page         : IOCUART
  - IOC_UART_CFG_DATA :
    - !expand { CFGHDR_TMPL : [ IOC_UART_CFG_DATA, 0x120, 0, 0 ] }
    - DeviceIndex  :
        name         : Device Index
        type         : Combo
        option       : 0:UART0, 1:UART1, 2:UART2, 3:UART3, 0xF:Disable
        help         : >
                       UART device index for IOC interface (0..3 or Disable)
        length       : 0x01
        value        : 0xF
    - BaudRate     :
        name         : Baud Rate
        type         : Combo
        option       : 0:9600, 1:19200, 2:38400, 3:57600, 4:115200, 5:921600, 6:1.5M
        help         : >
                       UART Baud Rate
        length       : 0x01
        value        : 0
    - Retries      :
        name         : Retries
        type         : EditNum, HEX, (0x00,0xFF)
        help         : >
                       specify retry count
        length       : 0x01
        value        : 0
    - TimeoutInitial :
        name         : TimeoutInitial
        type         : EditNum, HEX, (0x00,0xFF)
        help         : >
                       initial/setup time-out (in milliseconds)
        length       : 0x01
        value        : 0
    - TimeoutXmit  :
        name         : TimeoutXmit
        type         : EditNum, HEX, (0x00,0xFF)
        help         : >
                       transmission time-out
        length       : 0x01
        value        : 0
    - Rsvd         :
        length       : 0x03
        value        : 0

  - $ACTION      :
    page         : PSEL:PLT:"Payload Selection GPIO"
  - $ACTION      :
    page         : PSEL
  - PLATFORM_CFG_DATA :
    - !expand { CFGHDR_TMPL : [ PLATFORM_CFG_DATA, 0x280, 0, 0 ] }
    - PayloadSelGpio :
      - $STRUCT      :
          name         : GPIO pin for switching payload
          struct       : PAYLOAD_SEL_GPIO_PIN
          length       : 0x04
          value        : 0x000000c5
      - PadInfo      :
          name         : Pin Number
          type         : Combo
          option       : !include CfgData_GpioPinOption.yaml
          condition    : ($PLATFORM_CFG_DATA.PayloadSelGpio.Enable > 0)
          help         : >
                         Specify GPIO Pin Number
          length       : 24b
      - Rsvd1        :
          name         : Reserved
          type         : Reserved
          length       : 7b
      - Enable       :
          name         : Payload Selection Pin Enable
          type         : Combo
          option       : $EN_DIS
          help         : >
                         Enable/Disable this pin for payload selection.
          order        : 0000.0000
          length       : 1b

  - !include CfgData_Memory.yaml
  - !include CfgData_Silicon.yaml
  - !include CfgData_Usb.yaml
  - !include CfgData_Gpio.yaml
  - !include Platform/CommonBoardPkg/CfgData/CfgData_Common.yaml
  - !include CfgData_BootOption.yaml
  - !include CfgData_PidGpioPins.yaml
  - !include CfgData_PcieRp.yaml
  - !include CfgData_GpuConfig.yaml
  - !include CfgData_Features.yaml
  - !include CfgData_DeviceEnable.yaml
  - !include CfgData_Hda.yaml
  - !include CfgData_CapsuleInformation.yaml
```

收起 ^

You can see that it contains many `include` commands, specifying YAML files such as memory, Silicon, USB, GPIO, etc.

DLT File

The data in the DLT file is used to override the configuration in the YAML file.

The DLT file contains one `PlatformId`, such as Platform\ApollolakeBoardPkg\CfgData\CfgData_Ext_Up2.dlt:

yaml AI generated projects 登录复制

```yaml
#
# Delta configuration values for platform ID 0x000E
#
PLATFORMID_CFG_DATA.PlatformId           | 0x000E
```

There can be multiple DLT files in a platform, each corresponding to one `PlatformId` and eventually matching a specific board. If `PlatformId` it is equal to 0, it means that it is applicable to all boards, which is equivalent to changing the YAML file itself. These DLT files correspond to BoardConfig.py:

Python AI generated projects 登录复制 run

```python
self._CFGDATA_EXT_FILE = ['CfgData_Ext_Gpmrb.dlt', 'CfgData_Ext_Up2.dlt','CfgData_Ext_OxbHill.dlt','CfgData_Ext_MB3.dlt','CfgData_Ext_JuniperHill.dlt']
```

If you need to add a single board, you usually do not modify the YAML file directly. Instead, you add a DLT file and modify the configuration to overwrite the original configuration, and add the DLT file to the above list.

By the way `PlatformId`, Intel's CRB board specifies several GPIO pins, and specifies a specific board through hardware configuration, so that the code can determine it by reading the GPIO value `PlatformId`. For the ApolloLake platform, the relevant code can be seen in Platform\ApollolakeBoardPkg\Library\Stage1BBoardInitLib\Stage1BBoardInitLib.c:

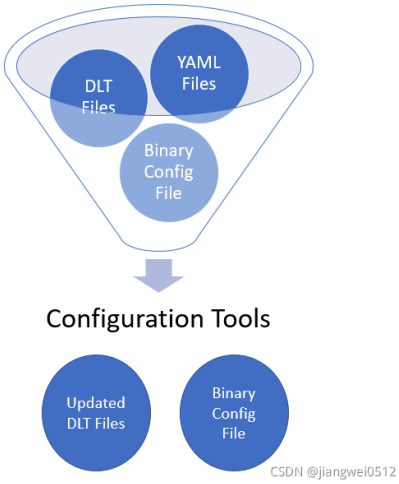
c AI generated projects 登录复制 run

```c
/**
  Detect board and configure PlatformID.

  @retval  EFI_SUCCESS     Configuration data was loaded successfully.
  @retval  Others          Failed to get configuration data blob.
**/
EFI_STATUS
EFIAPI
PlatformIdInitialize (
  IN  VOID
  )
{
  UINT16     PlatformId;

  PlatformId = (UINT16)GetBoardIdFromGpioPins ();

  if (PlatformId != 0xFF) {
    PlatformId += 0x10; // Customer board identified, assign Platform Ids from 16 to 31
  } else {
    PlatformId = (UINT16)GetEmbeddedBoardId ();
    //Platform ID from GPIOs are read as 0 for Juniper hills due to GPIO pins
    //on the board reduced from 4 to 3 (hardware change) hence translating here
    //in the code.
    if (PlatformId == 0){
        DEBUG ((DEBUG_INFO, "GPIO returned platformID 0 translating to 8(JNH)\n"));
        PlatformId = 0x8;
    }
    if ((PlatformId != PLATFORM_ID_OXH) && (PlatformId != PLATFORM_ID_LFH) && (PlatformId != PLATFORM_ID_JNH)) {
      PlatformId = (UINT16)GetIVIBoardId ();
      if (PlatformId != PLATFORM_ID_GPMRB) {
        DEBUG ((DEBUG_ERROR, "BOARD NOT SUPPORTED: 0x%04X\n", PlatformId));
        CpuDeadLoop ();
      }
    }
  }

  SetPlatformId (PlatformId);
  return EFI_SUCCESS;
}
```

收起 ^

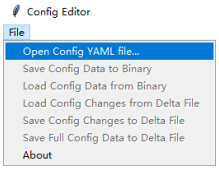
Configuration Tools

The way the configuration tool processes YAML and DLT files looks like this (note that binary files can be used as both input and output):



So this section introduces the configuration tool, which is located in the BootloaderCorePkg/Tools directory. The most important one is ConfigEditor.py, which is a graphical tool. After opening it, it looks like this:

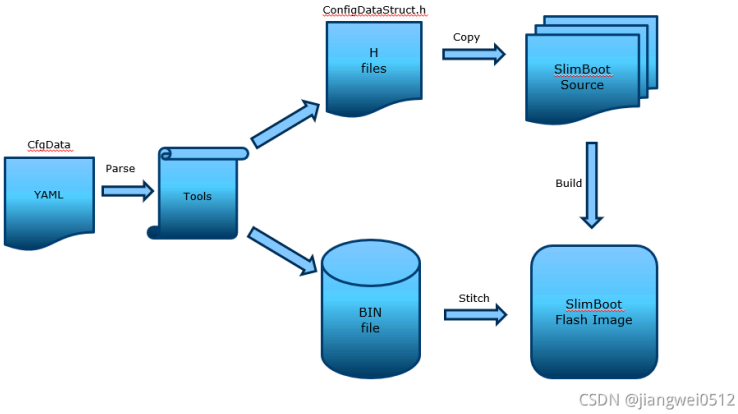
Click "File". For the first time, you can only select "Open Config YAML file...":



Here you can select the CfgDataDef.yaml mentioned above, which is the basic configuration file. After that, you can select "Load Config Changes from Delta File", so that the newly added DLT file will overwrite the original display, and then you can continue to modify and finally save the modified data. Such data can be a DLT file containing only the modified part, or a DLT file that covers all configurations, or directly generate a binary file.

In addition to ConfigEditor.py, there is also a configuration tool used to convert the YAML file into the header file Platform\XXXPkg\Include\ConfigDataStruct.h, which will be included in the SBL code to finally obtain and use the configuration through the code.

The final configuration tool usage process is as shown in the following figure:



In general, the configuration of SBL is the basic configuration defined by the YAML file, and the DLT file modifies the configuration according to the actual board, and finally generates a binary configuration file, which will be placed in the SBL binary for subsequent retrieval. In the SBL code, the current one will be judged `PlatformId` and the required binary configuration file will be loaded to complete the final configuration.

Finally, even after the SBL binary is generated, its parameters can still be modified through tools, which is called post-build customization.