Easy PinMux Tool v2.1 User's Guide

Version: 2.2

Release date: 5 May 2017

Document Revision History

Revision	Date	Description
1.0	1 February 2016	Initial release
1.1	11 March 2016	Provide more description for `.cmp` files.
1.2	26 April 2016	Add demo workspace file (`.dws`) path.
2.0	4 November 2016	EPT version 2.0
2.1	13 January 2017	EPT version 2.1
2.2	5 May 2017	Support more chips in EPT

Table of Contents

Lists of Tables and Figures

1. Overview

MediaTek Easy PinMux Tool (EPT) is a convenient and user-friendly graphical user interface (GUI) to configure pin multiplexor (PinMux) and supported driver settings for MediaTek chipsets, including MT25x3, MT768x. The tool provides modes and options for each PinMux and enables customized settings for input and output (I/O) characteristics according to design requirements.

Once configured, all settings can be saved as a workspace file that can be reloaded to apply the preconfigured tool settings. The results can also be output as C header and source files.

The analog to digital converter (ADC) pins are controlled by the ADC driver, not by the EPT tool. For more information about how to configure the ADC pins, please refer to the ADC module in the Hardware Abstraction Layer (HAL) section of LinkIt for RTOS API Reference Manual.

1.1. Environment

The EPT Tool can be used on Windows (32 or 64 bit editions, Windows XP, Vista, 7, 8 and 10) and Linux (Ubuntu 32 or 64 bit, Ubuntu version 14.04 and higher).

1.2. Configuring your device with the EPT

To use the tool:

1) Launch the executable (`ept.exe`) under EPT tool package folder. Create a new workspace or open an existing one, and edit it according to requirements, see section 1.2.1, “Workspace file options”, for more details.

2) Apply user settings, see section 2, “Driver Settings”, for more details on how to configure the I/O parameters and PinMux settings.

3) In **Gen** menu click **GenCode** to generate a source code of the driver stored under `\output\25x3(768x)` folder of the tool. The main GUI is shown in Figure 1 and Figure 2.

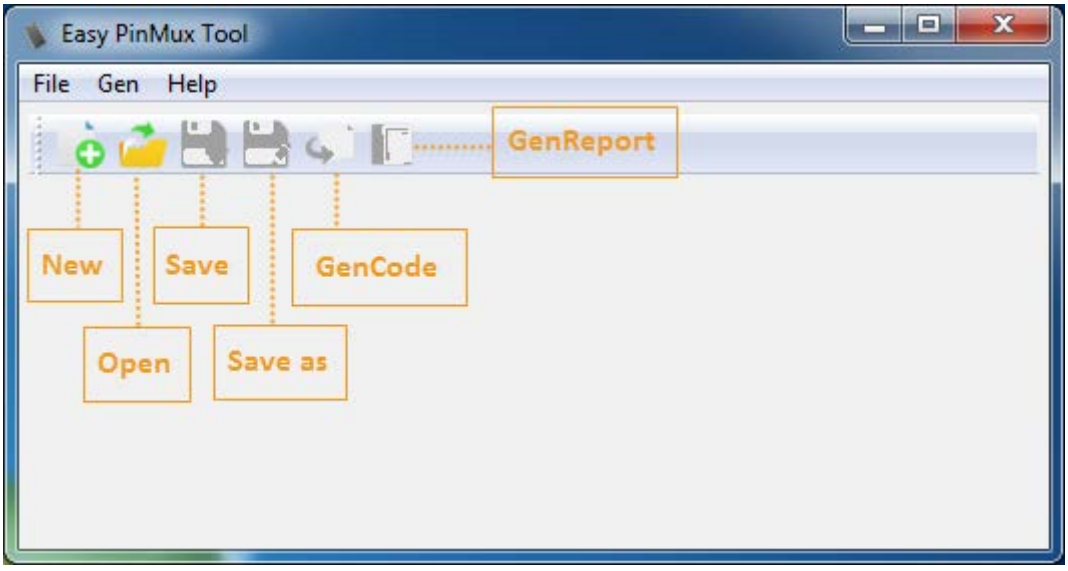

Figure 1. Main UI of the EPT with an empty workspace

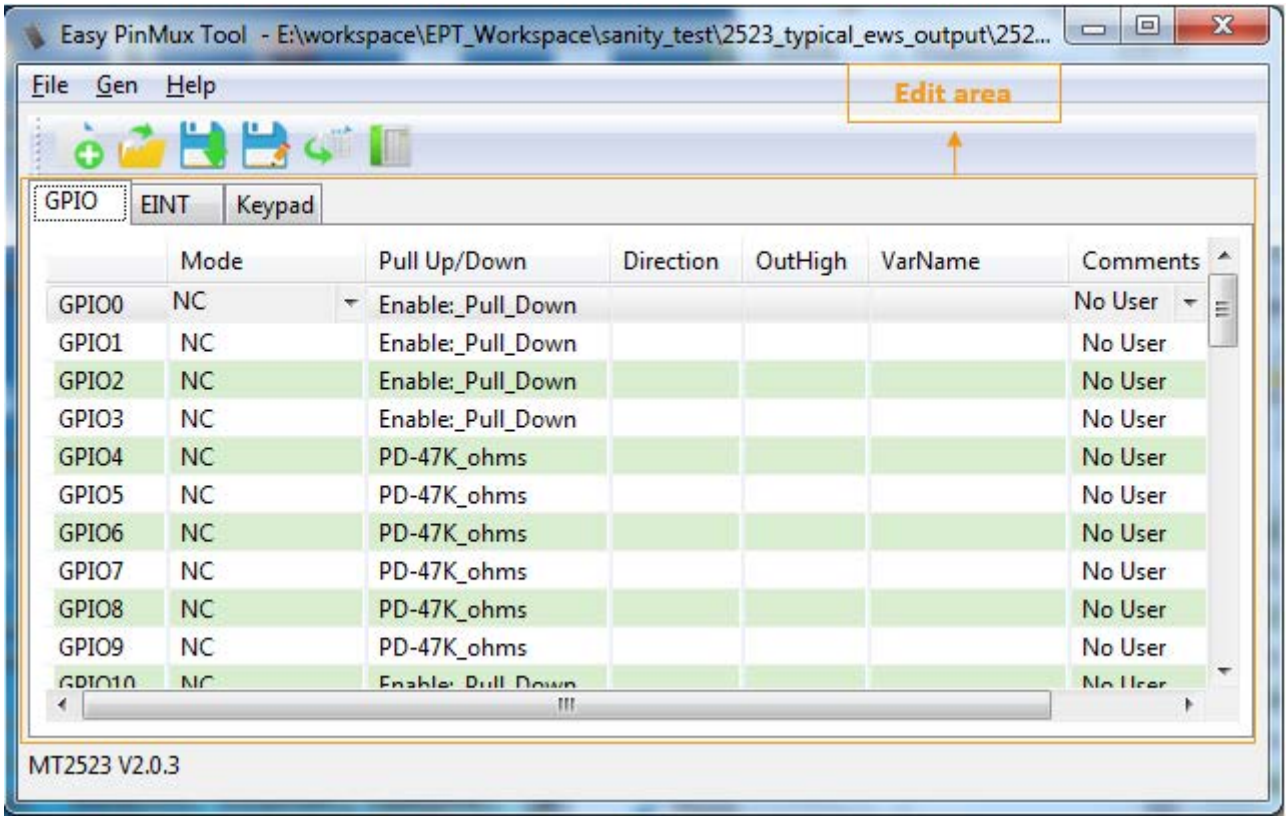

Figure 2. Main UI of the EPT with a workspace

1.2.1. Workspace file options

To create a new workspace:

1) In **File** menu, click **New**, and provide the chipset under **Chip Selection** to create an empty workspace, as shown in Figure 3.

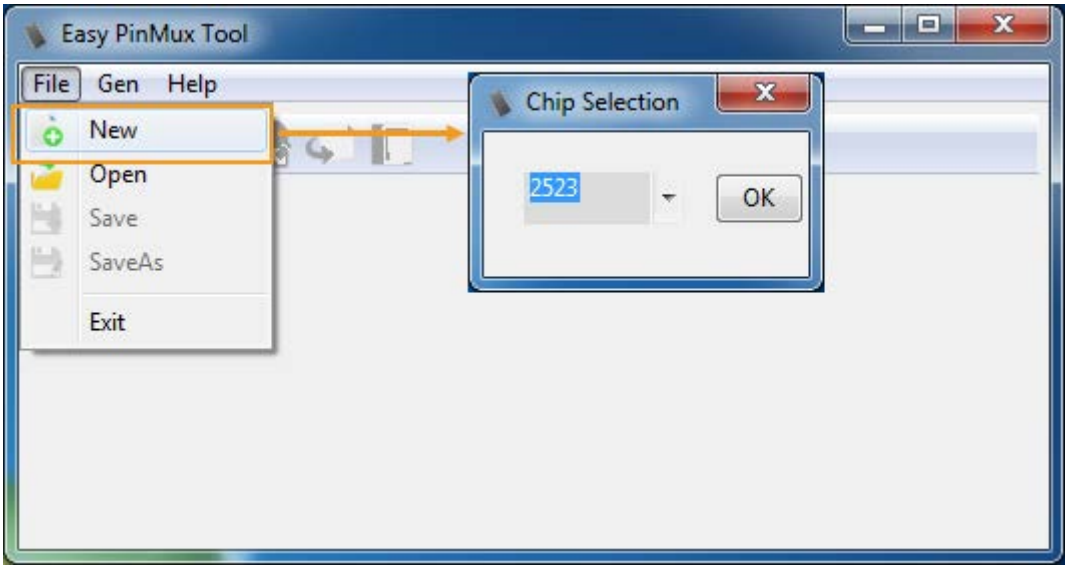

Figure 3. Create a new workspace

2) Save the workspace as a `.ews` file by clicking **Save** or **SaveAs** options in **File** menu.

To open an existing workspace:

1) In **File** menu, click **Open** to open an existing workspace. You can use your own workspace file or a demo workspace file with an extension `.ews` or `.dws` provided by MediaTek. The demo workspace file (`.ews` or `.dws`) is located under:

```
<sdk_root>\tools\config\<board>\ept\
```

To edit the workspace:

1) When an existing workspace is opened or a new workspace is created, you can configure its parameters according to your requirements in the edit area, as shown in Figure 4.

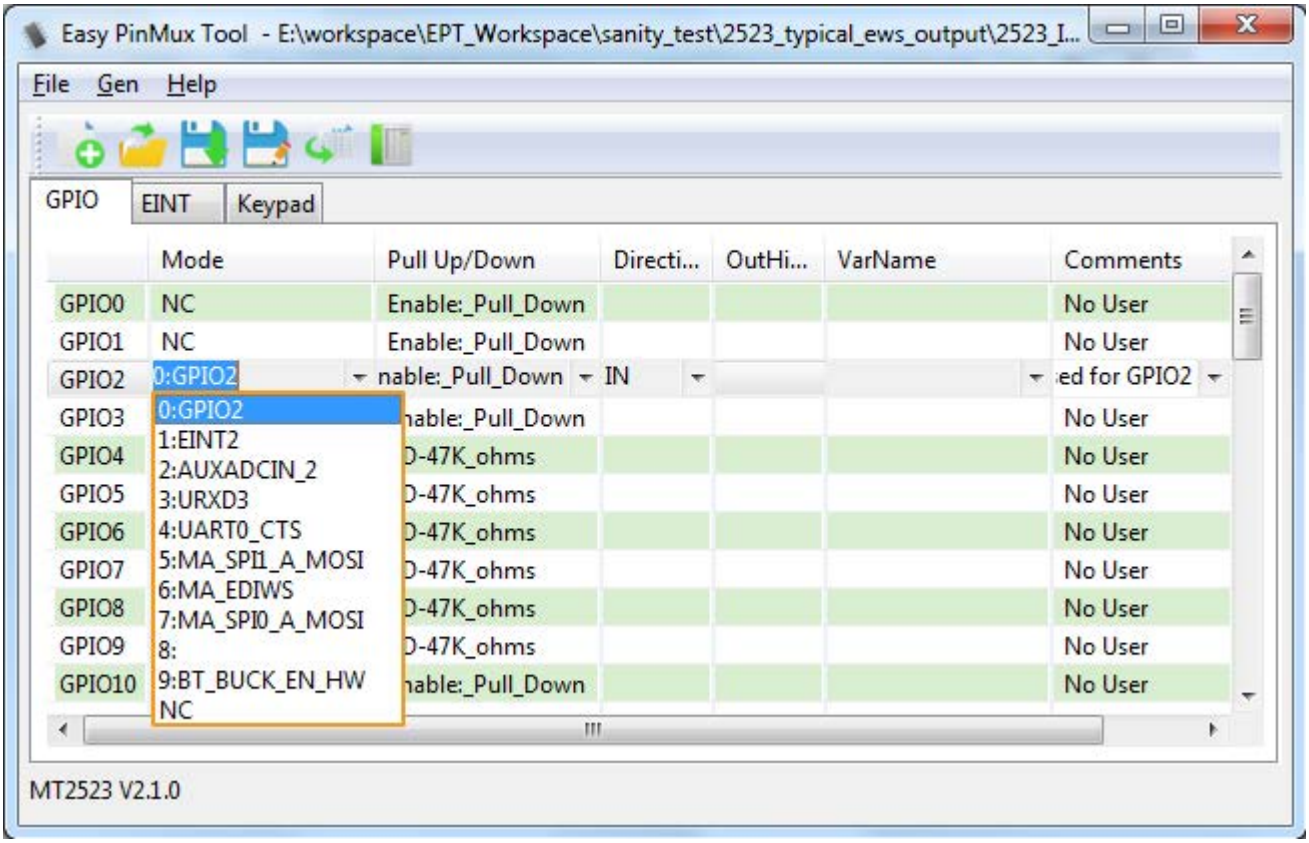

Figure 4 . Edit a workspace

2) Save the workspace as a `.ews` file.

To generate a source code for the driver:

1) In **Gen** menu, click **GenCode** to generate a source code for the driver. All generated files are saved in `\output\25x3(768x)` folder. The header files (`.h` files) are saved in the sub-directory `inc`, while the source files (`.c` files) are saved in the sub-directory `src`. Once the source code is generated successfully, a popup message will prompt the file path, as shown in Figure 5.

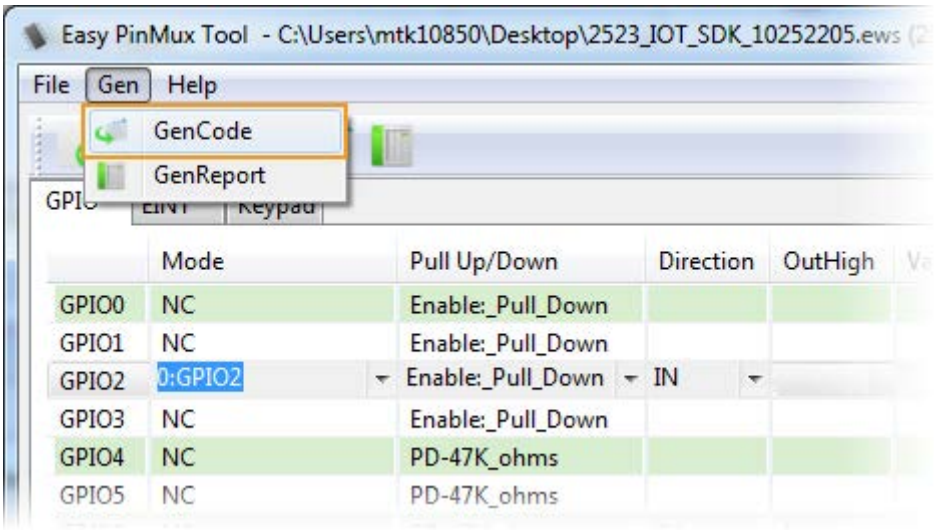

Figure 5. Generate code for the current configuration

A confirmation message will appear, as shown in Figure 6.

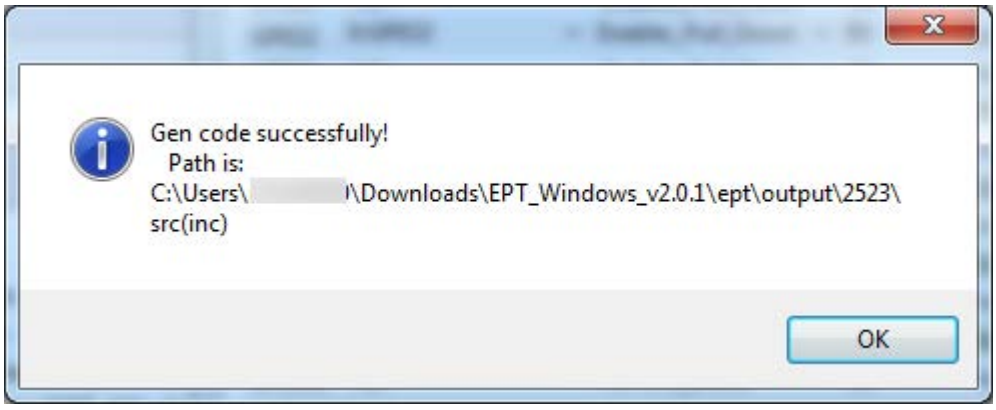

Figure 6. Code is successfully generated

2) Copy the generated source code to destination driver folder of the project. The full path of the driver folder for source and header files is as follows.

```
<sdk_root>\project\<board>\apps\<application>\src
<sdk_root>\project\<board>\apps\<application>\inc
```

Where `<board>` is the name of your board, such as `mt2523_evb`, `mt7687_hdk` and `<application>` is the name of your project, such as `iot_sdk_dev, iot_sdk`.

The GPIO settings configured by the EPT tool take effect only when they are written to GPIO registers. Details about this can be found in the readme file of EPT example code located under the folder:

```
<sdk_root>\project\<board>\apps\<application>\
```

Once the configuration is set, build the load on the target device. More information about building the load can be found at MediaTek LinkIt™ Development Platform for RTOS Get Started Guide.

An example use case to generate the files based on given inputs is shown in Figure 7. The user provides `.ews`, `.chip` and `.conf` files as an input to the EPT tool and the expected outcomes are `.ews`, `.h` and `.c` files.

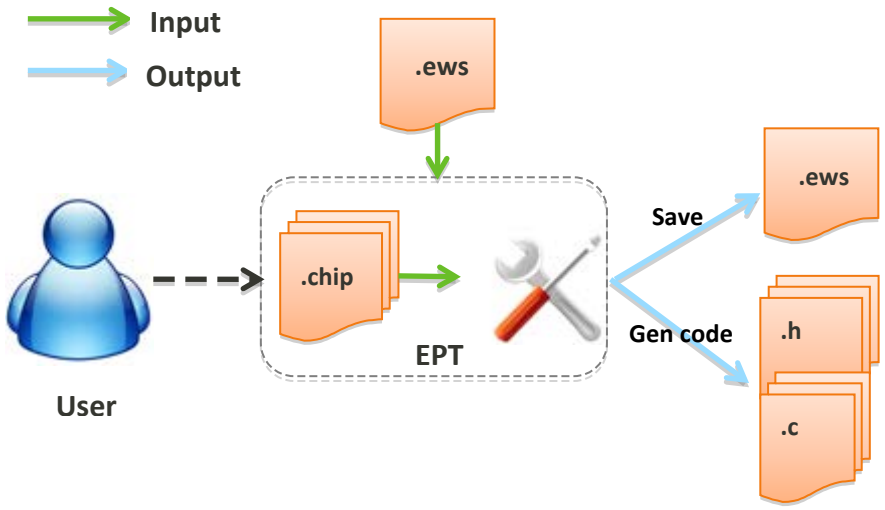

Figure 7. An example to generate configuration and executable files with given input files

1.3. Folder structure of the EPT tool

The EPT tool contains five main folders and two files. The folder structure is shown in Figure 8.

1) `configuration` — This folder stores the configuration files for MediaTek chipsets.
2) `output` — This folder contains output files such as source (`.c`, saved in the `src` folder) and header (`.h`, saved in the `inc` folder) files and the pinout report (`pinout_report.csv`) generated by the EPT tool.
3) `project` — This folder contains the workspace files (`.ews`) saved in this folder by default.
4) `EPT` — This folder contains libraries to run the EPT.
5) `jre1.8_win` — This folder includes the supporting files to run EPT on Windows OS.
6) `generate_script.bat` — Provides a command line mode to generate code.

The `ept` folder also contains the executable file (`ept.exe`) to start the configuration. Launch the program by double-clicking the executable file, no need to install it.

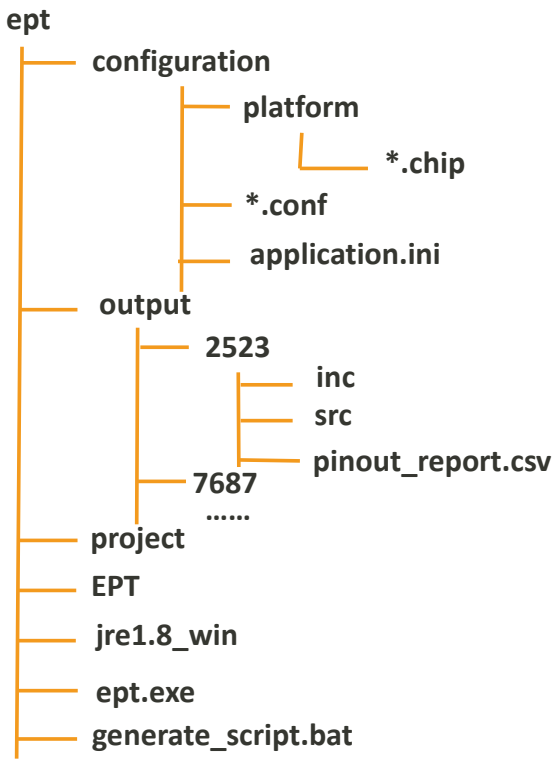

Figure 8. Folder structure

1.3.1. Chip files

Chip files store the chip parameters and options shown on the EPT UI. Every chip has its own chip file. The user must not modify the content of the chip file.

1.3.2. Configuration files

The chipsets share the same configuration (`.conf`) files. Usually there are multiple `*.conf` files for different modules, such as `gpio.conf`, `eint.conf` and `keypad.conf`. An example `.conf` file is shown in Figure 9.

1) `gpio.conf` — contains the header and tail information of .h file for GPIOs.
2) `keypad.conf` — contains the key symbols of the keyboard.
3) `eint.conf` — contains the header and tail information of .h file for EINTs.

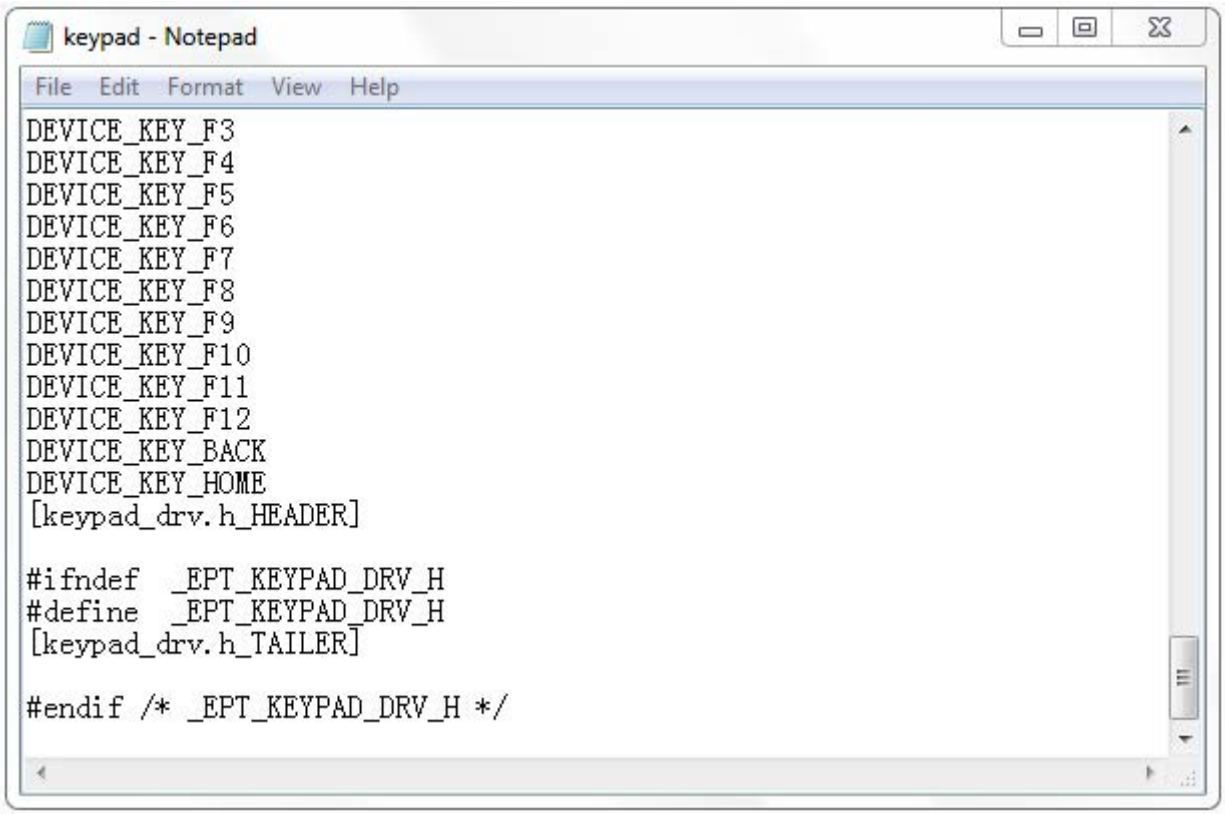

Figure 9. Keypad .conf file

1.3.3. EWS files

EWS (.dws for versions before 2.0.1) files are the EPT workspace files. The file contains customized PinMux and I/O settings, see section 1.2.1, "Workspace file options" to create or modify a workspace file. MediaTek also provides demo workspace files where many PinMux settings are already configured. These demo .dws or .ews files are stored under /project folder. It is recommended to modify and save an existing demo workspace file as a reference for custom design configuration.

1.3.4. Pinout report

To generate a pinout report (`pinout_report.csv`), click **GenReport** in **Gen** menu. The report contains the PinMux information set by the user, including seven columns for **Name**, **Mode**, **Pull Up & Pull Down**, **Direction**, **OutHigh**, **VarName** and **User Information**.

2. Driver Settings

This section describes the driver settings for General Purpose Input Output (GPIO), External Interrupt (EINT) and Keypad modules.

2.1. GPIO

Open the **GPIO** page by open or creating a workspace on the main UI, as shown in Figure 2. It is used to set the GPIO parameters. The GUI enables setting up the **Mode**, **Pull up/down**, **Direction**, **OutHigh**, variable name **VarName** and **User Information** for the GPIO pins.

2.1.1. Setting up the Mode option

Figure 10 shows the relationship between the GPIO **Mode** selected by the user and the code generated in `ept_gpio_drv.h` header file. The **GPIO0** pin has 10 modes. In the example, the user selects **EINT0** corresponding to **Mode 1** for the **GPIO0**.

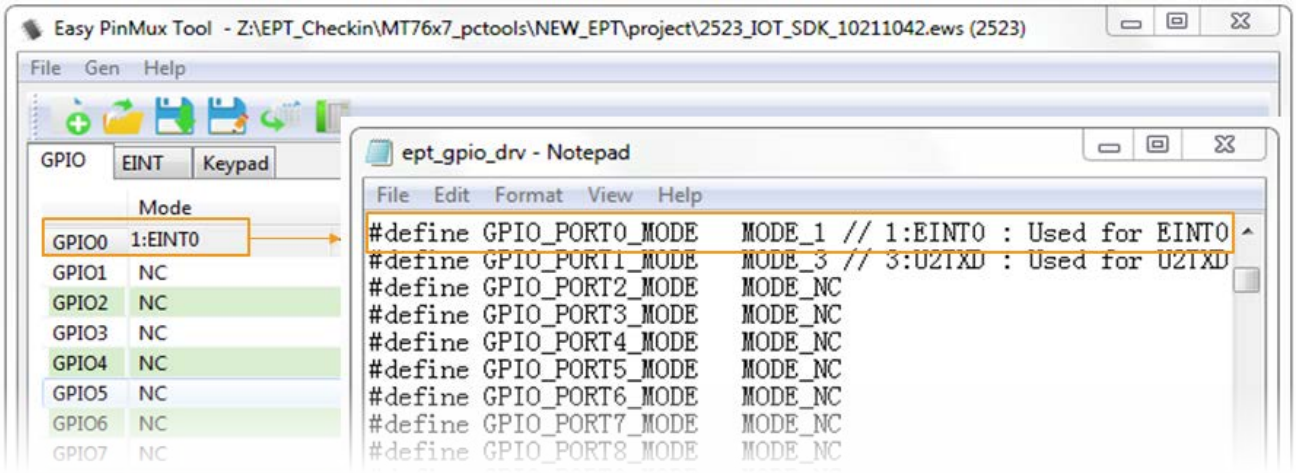

Figure 10. GPIO0 mode and generated code

2.1.2. Setting up the Pull Up/Down, Direction, OutHigh options

The **Pull Up/Down** is available when the **Mode** is selected, if the selected **Mode** is **GPIO**, **Direction** is available too, such as for **GPIO10** shown in Figure 11. Additionally, the **OutHigh** checkbox is invisible until the GPIO pin **Direction** is set to **OUT**. Figure 11 shows the **GPIO10** options and generated code for **Pull Up/Down**, **Direction** and **OutHigh** values are again stored in the `ept_gpio_drv.h` header file.

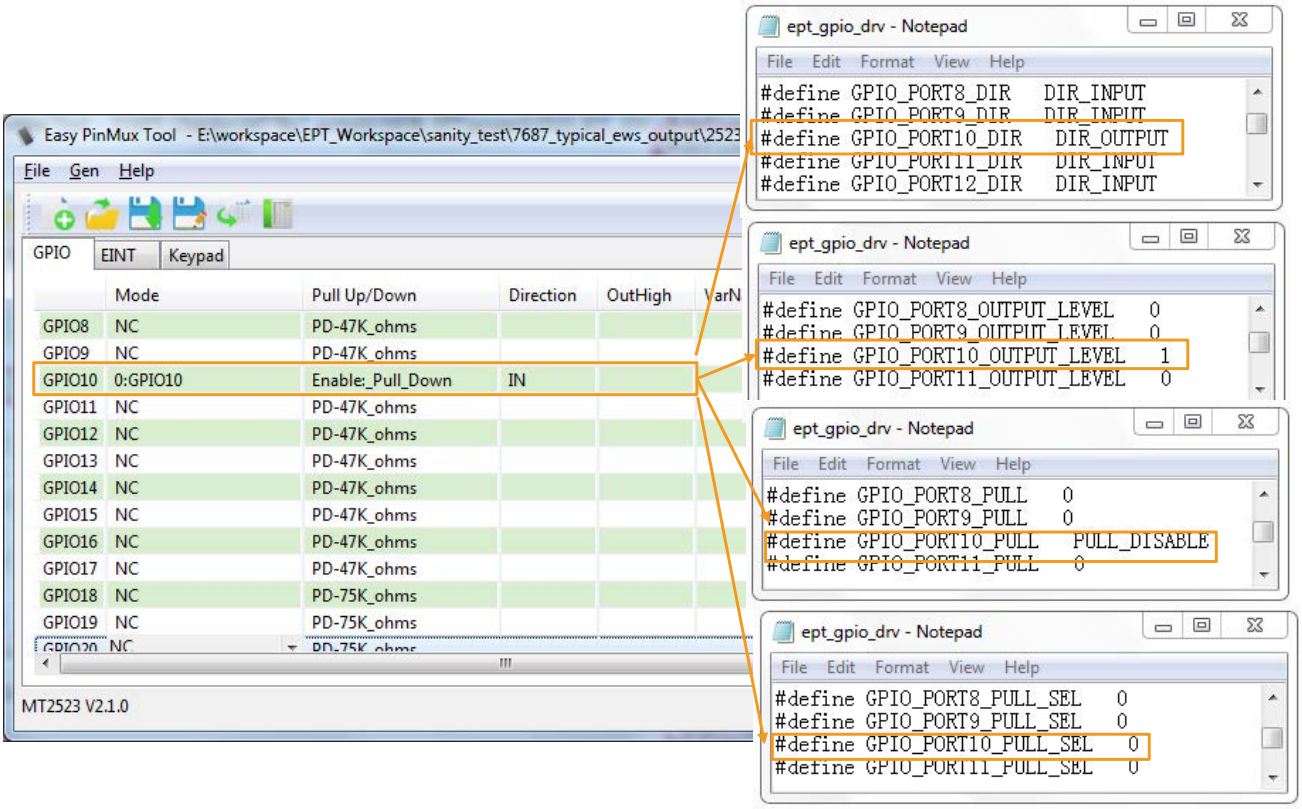

Figure 11. GPIO10 option and generated code in `ept_gpio_drv.h` header file

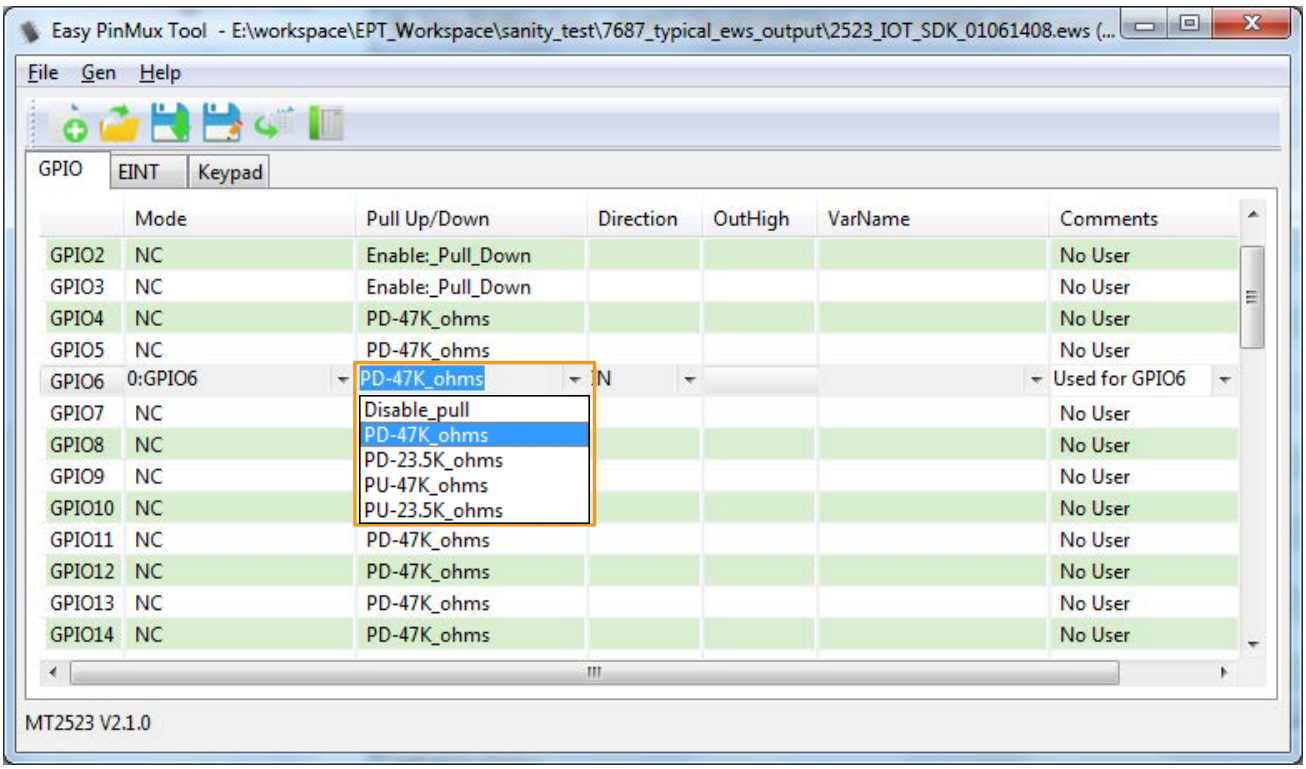

Figure 12. Pull Up/Down options for GPIO6

The option list shown in Figure 12 includes five items. The mapping relationship between user's selections for **Pull Up/Down** and generated code is shown in Figure 13 and Table 1.

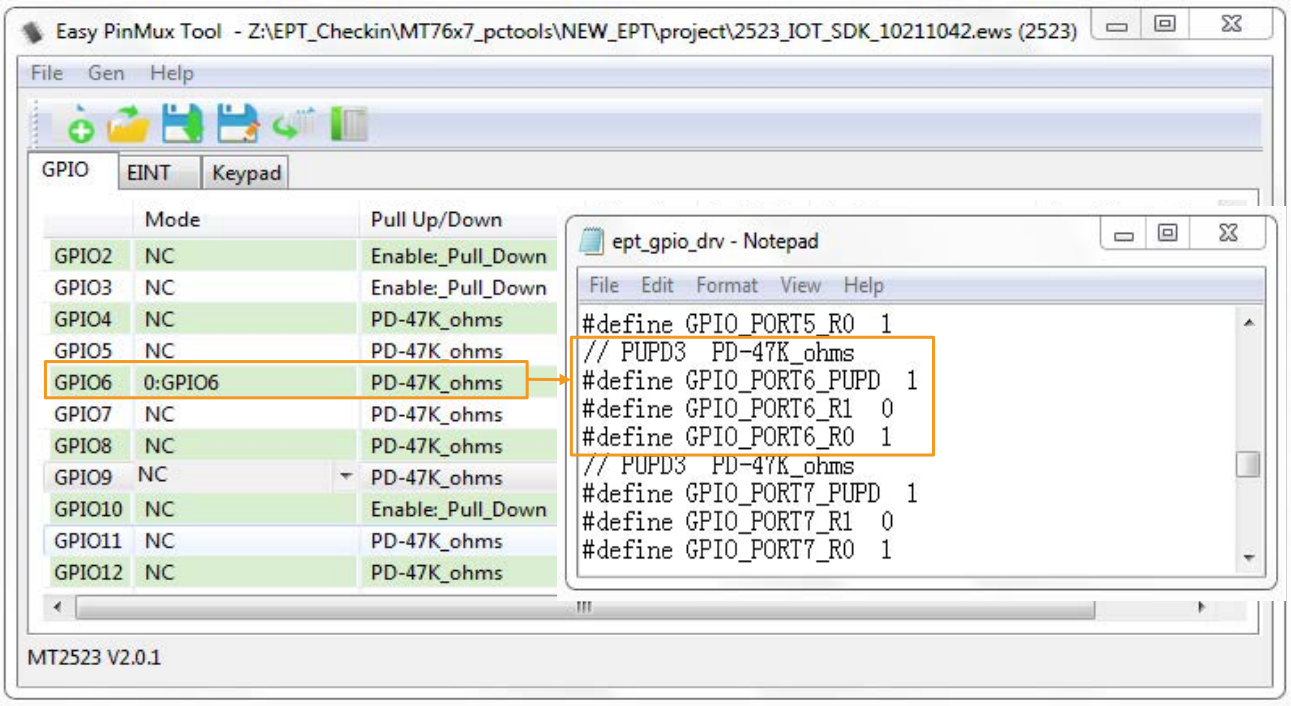

Figure 13. GPIO6 Pull Up/Down state

Table 1. Register and Pull Up/Down mapping options

PUPD	R1	R0	PULL UP/DOWN STATE
0	0	0	Disable both registers
0	0	1	PU-47kΩ
0	1	0	PU-47kΩ
0	1	1	PU-23.5kΩ
1	0	0	Disable both registers
1	0	1	PU-47kΩ
1	1	0	PU-47kΩ
1	1	1	PU-23.5kΩ

2.1.3. Setting up the VarName

Users can also select a variable name for the given GPIO pin. The selected variable name should be unique.

There is a list of options to choose from **VarName** and the default value is set and stored in the `ept_gpio_var.h` file, as shown in Figure 14.

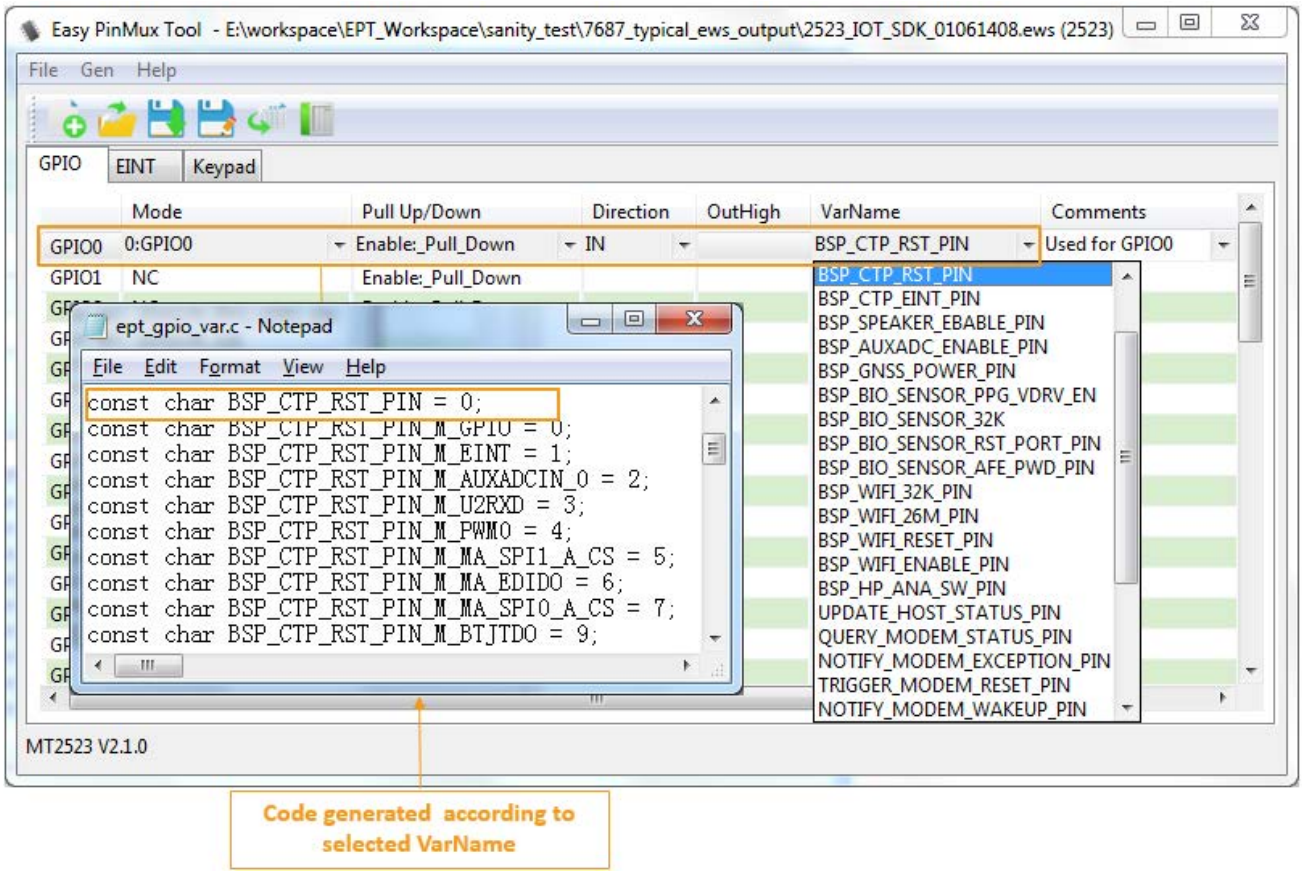

Figure 14. Variable name set for the GPIO0

The final configuration of all pins with their corresponding modes is stored in a file, as shown in Figure 15.

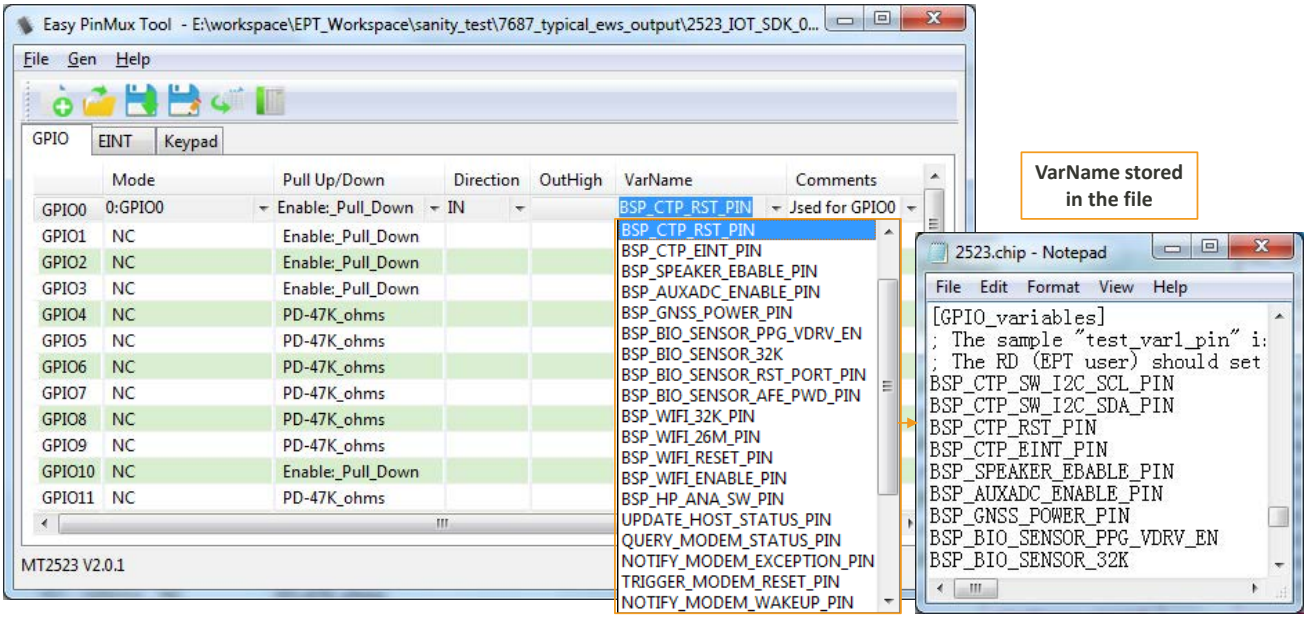

Figure 15. VarName column stored in file

2.1.4. Setting up the Comments

1) The **Comments** defines the use of its corresponding pin, as shown in Figure 16.

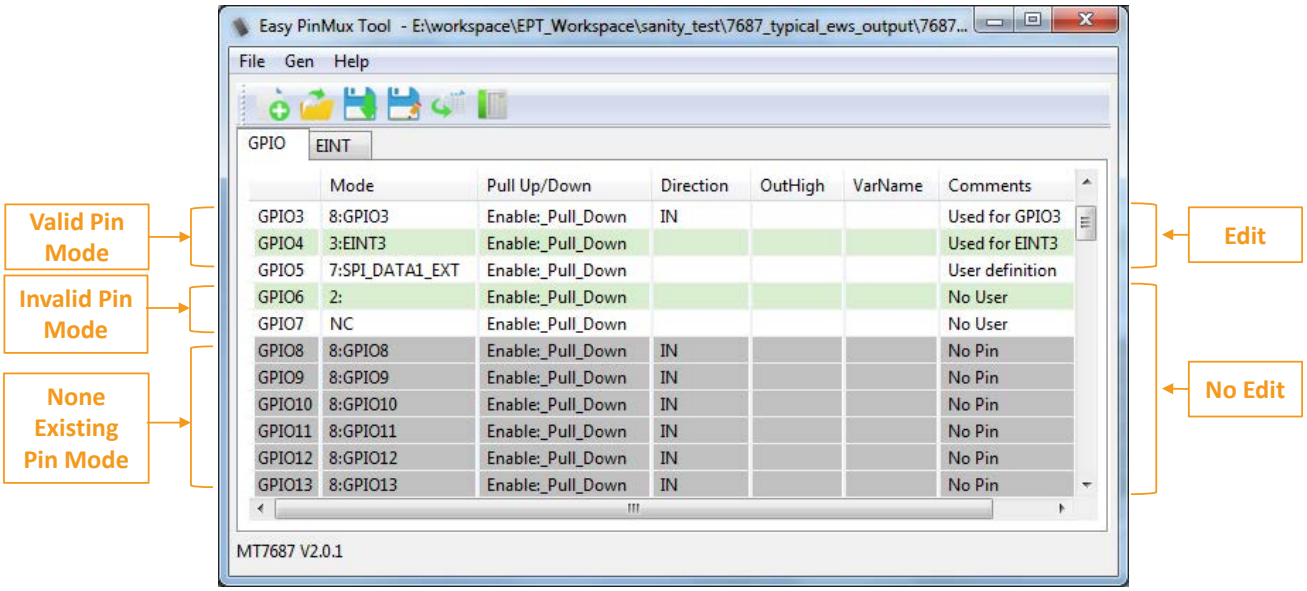

Figure 16. User Information for corresponding GPIO mode

By default, it provides three options to choose from:

1) **Used for "pin name"** — assigned to a valid pin mode selection. You can modify the default option. However, it's recommended not to change it, unless it's necessary.

2) **No User** — assigned to an invalid pin mode selection and cannot be edited.

3) **No Pin** — assigned to the pin mode that's not available for the current chip and cannot be edited.

2.2. EINT

Click the **EINT** tab to set the EINT parameters. You can only set the EINT variable name (**EINT Var**) for the pins that already enabled under the **GPIO** tab. The variable name option list won't appear if you click on a disabled EINT row. Figure 17 shows the relationship between the GPIO page and the EINT.

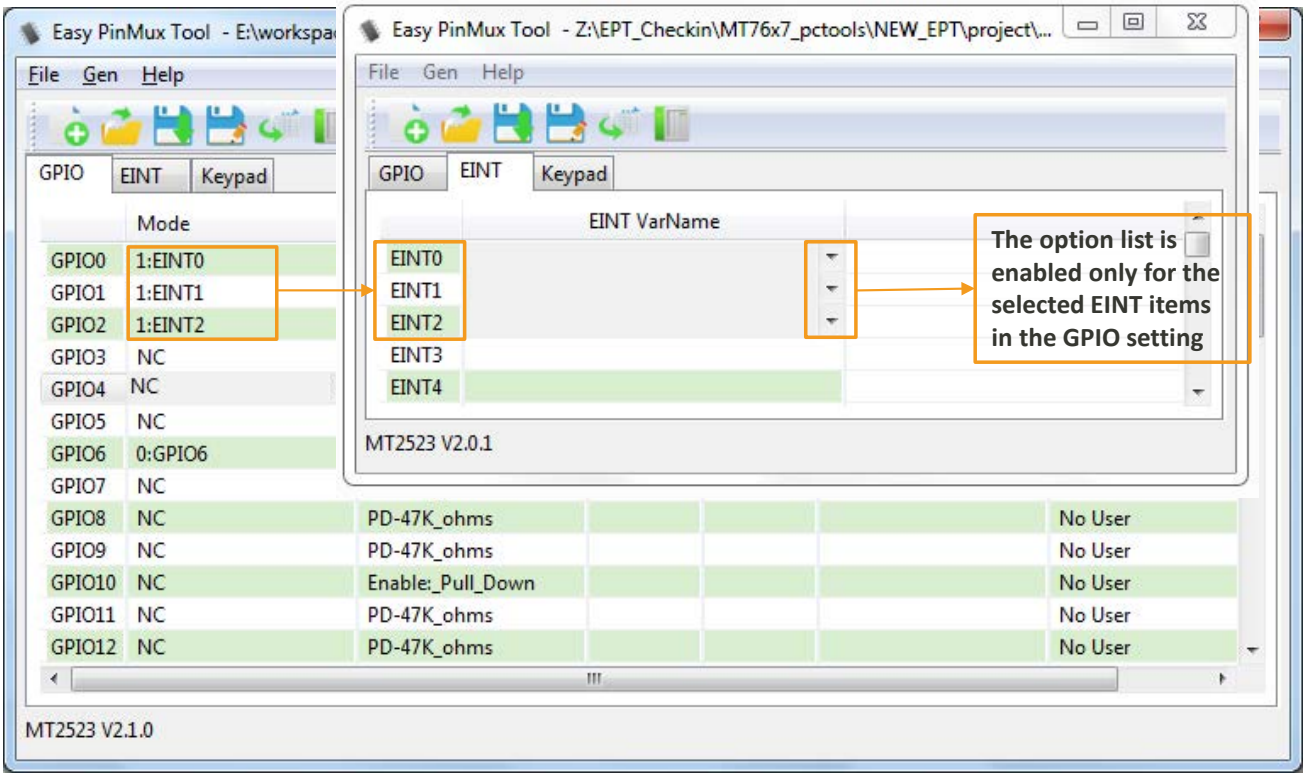

Figure 17. EINT settings corresponding to the GPIO pin settings

By default, the option list for variable names under **EINT VarName** column is mapped from chip file, as shown in Figure 18.

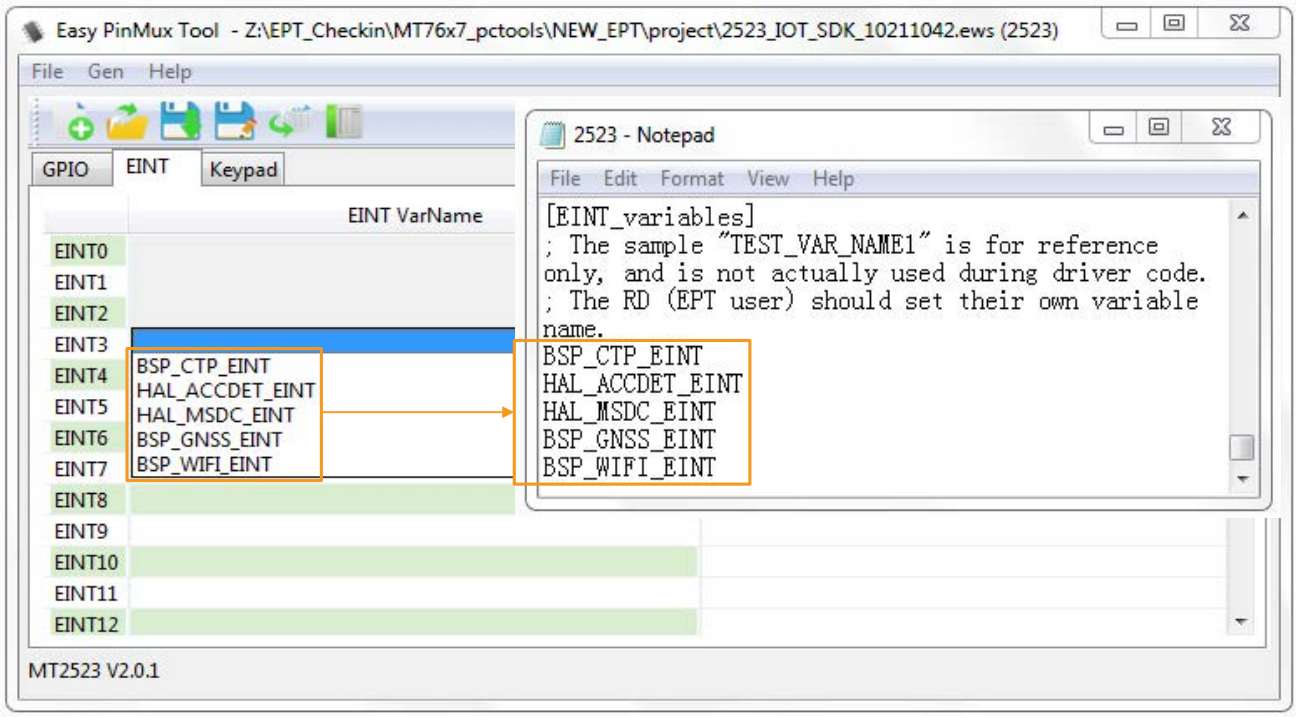

Figure 18. Initialize EINT variable name list

In some cases, several EINT pins are already used by modules defined in section [**EINT_USED**] in `.chip` file. For these EINT pins, you are not allowed to set the variable name. Figure 19 shows that EINTs from `EINT20` to `EINT31` are used by other modules.

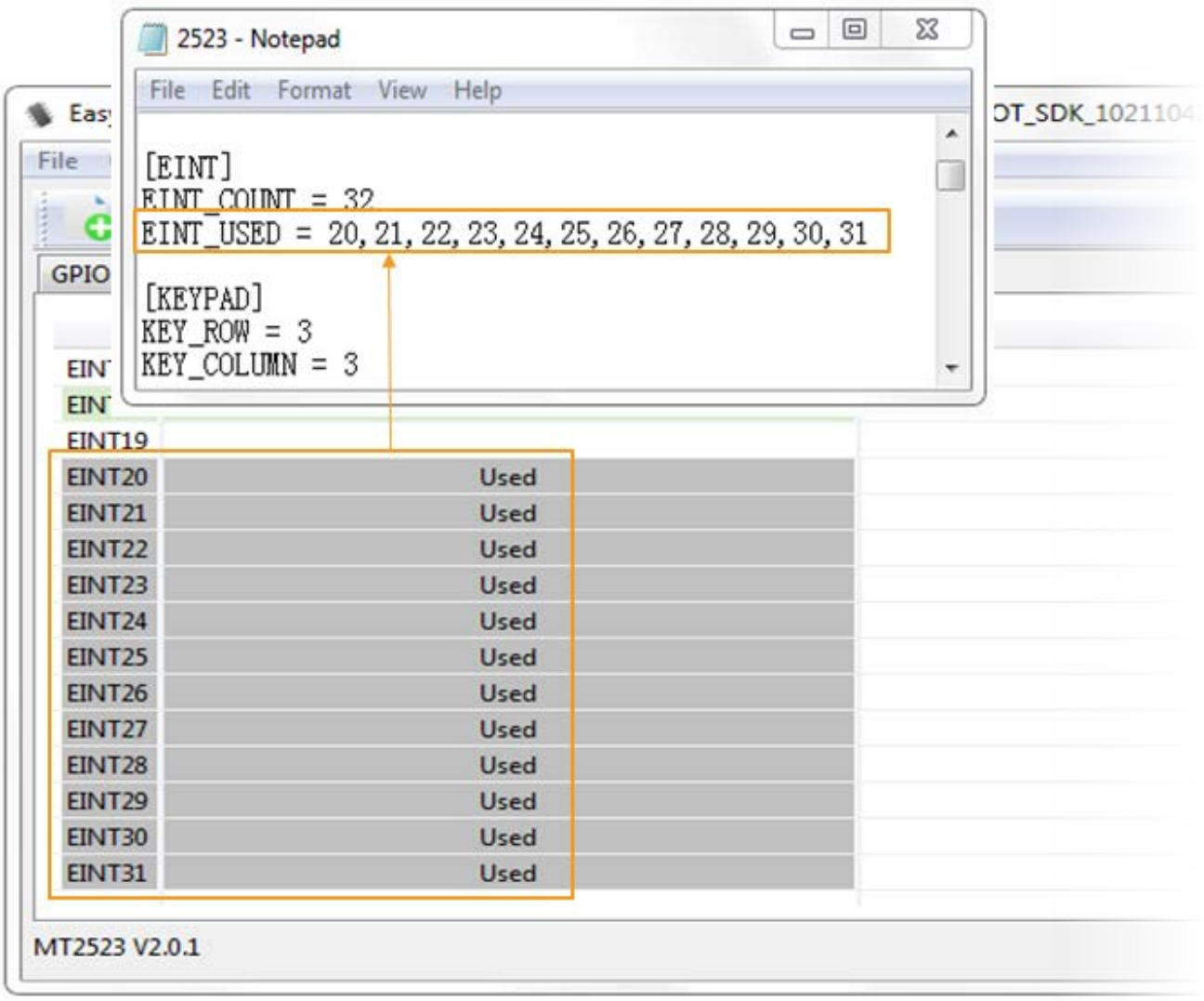

Figure 19. EINT used by modules

2.3. Keypad

Click the **Keypad** tab to set the keypad options. The two parameters that could be configured are **Power Key** and **Key Type**.

2.3.1. Setting up the Key Type

The **Key Type** could be defined as **SINGLE_KEYPAD** (Figure 20) or **DOUBLE_KEYPAD** (Figure 21). When the user chooses **SINGLE_KEYPAD** type, the **Keypad** configuration contains three rows and three columns. When the user selects the **DOUBLE_KEYPAD** type, the **Keypad** configuration contains three rows and six columns.

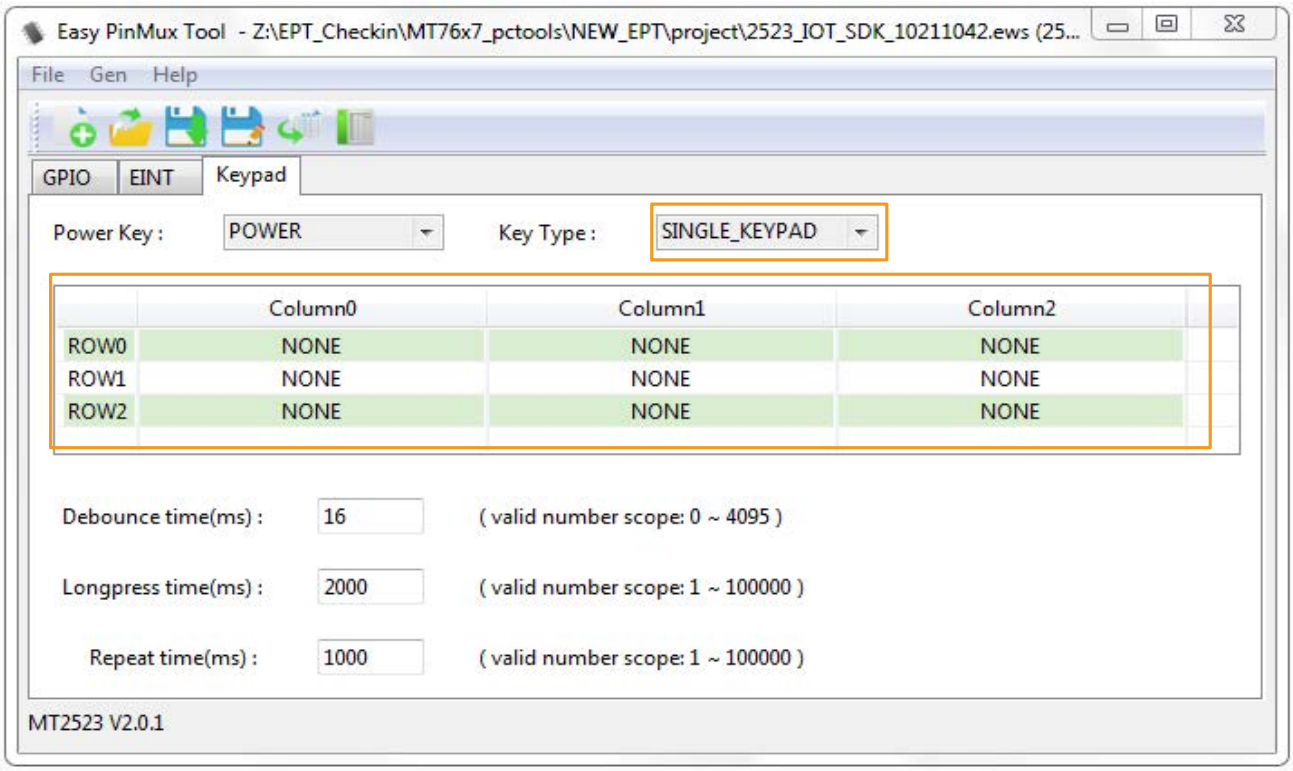

Figure 20. Keypad configuration, SINGLE_KEYPAD TYPE

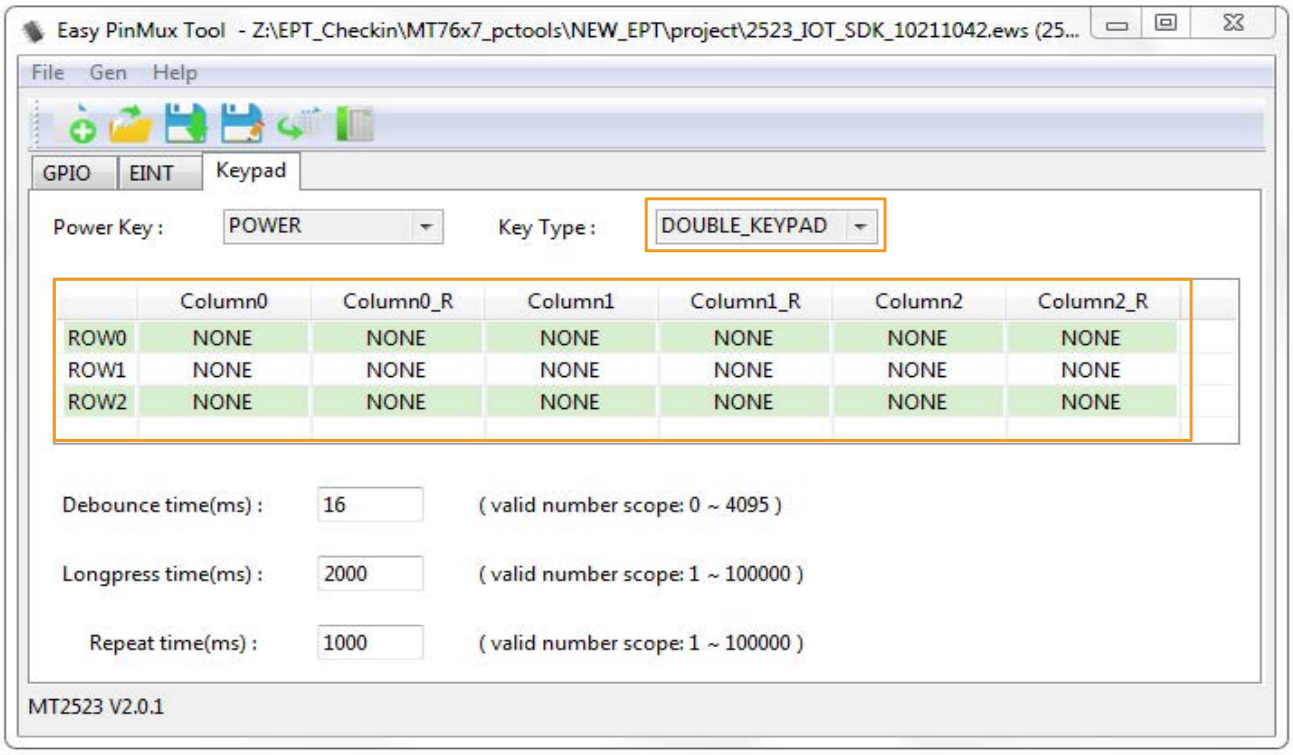

Figure 21. Keypad configuration, DOUBLE_KEYPAD TYPE

There are three parameters: **Debounce time**, **Longpress time** and **Repeat time** for keypad.

- **Debounce time** — defines the waiting period before key press or release events are considered stable. If the de-bounce setting is too small, the keypad will be too sensitive and detect too many unexpected key presses. The suitable de-bounce time setting must be adjusted according to the user's habits.
- **Longpress time** — the event time setting of longpress action. If the key is pressed and held for longer than **Longpress time**, the keypad driver will report a HAL_KEYPAD_KEY_LONG_PRESS event.
- **Repeat time** — the interval time of key event report while a key is pressed and hold. If the

HAL_KEYPAD_KEY_LONG_PRESS is reported, and the key is still on hold, the keypad driver will report the HAL_KEYPAD_KEY_REPEAT event every **Repeat time** interval.

Each row item corresponds to a key that is configured in `keypad.conf` file under **[Key_definition]** section, as shown in Figure 22.

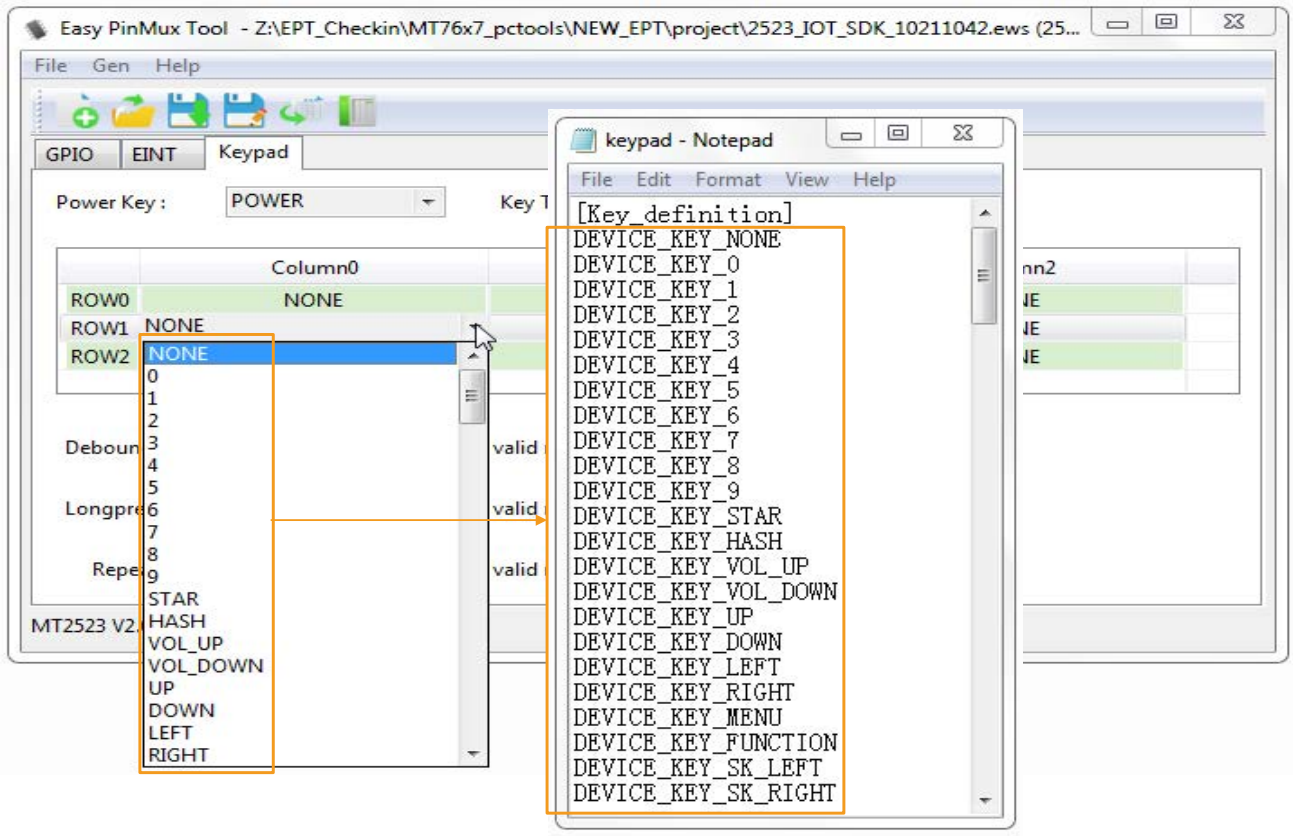

Figure 22. Choose a key

2.3.2. Setting up the Power Key

An example of **Power Key** configuration is shown in Figure 23, where it is set to **DOWN**. After generating the output code, the `ept_keypad_drv.h` file defines the **POWERKEY_POSITION** as **DEVICE_KEY_DOWN** (see Figure 23).

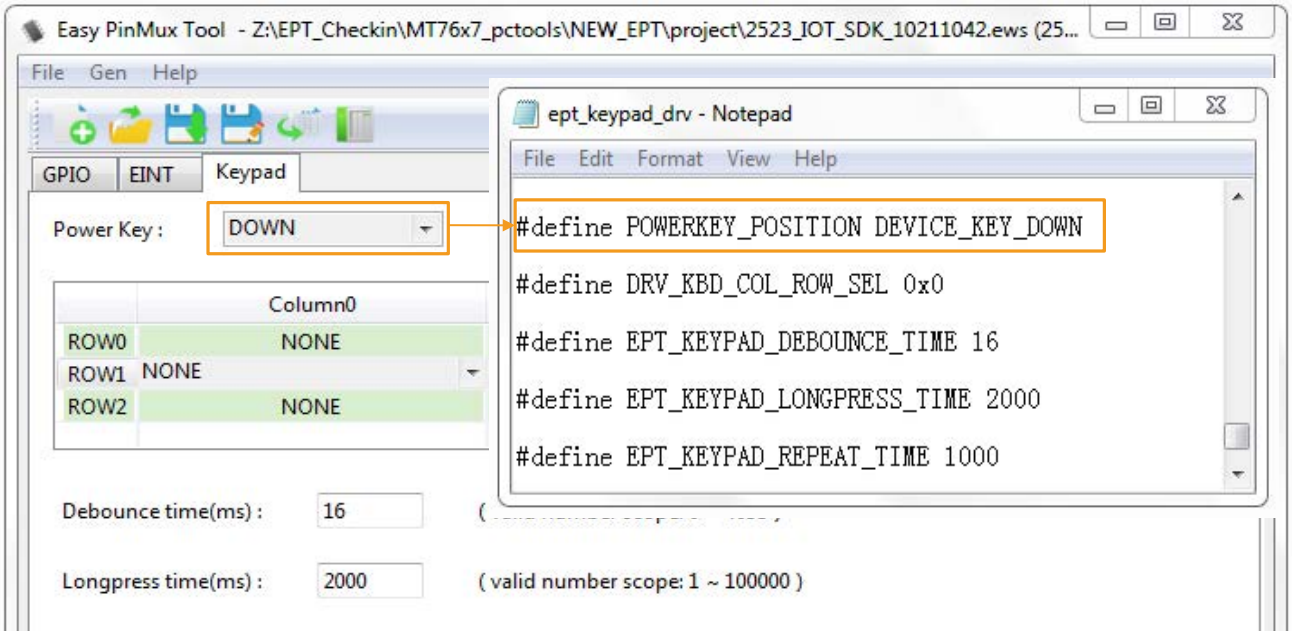

Figure 23. The Power Key configuration