# StarterWare 02.01.01.XX Board and ChipDB Details Update Guide

## Overview

The document explains How to add Pin Mux Data,Device Data and ChipDB details of SoC. Pin Muxing is required for Multiplexing internal peripheral signals like I2C_SCL , UART_RXD, GPMC_AD0 etc. to external Pinouts SoC. Pin Mux can be done either with the help of Pin Mux Tool or can be done manually. Device data need to be added if any new device need to be added on the board like LEDs, Flash etc. ChipDB details include No of Modules like UART, I2C etc present on Board, No of istances of each Module, Base address and Interrupt numbers of each istance.

## Updaing Pin Mux Data

### Updating Pin Mux data with Pin Mux Tool

For Updating the Pin Mux Data with automated Pin Mux Tool follow the procedure as in Wiki link [Ti Pin Mux Tool http://processors.wiki.ti.com/index.php/TI_PinMux_Tool]. Incase of conflicts while using the Pin Mux tool update the data manually.

### Updating Pin Mux Data Manually

Refer to Board Schematics and Pin assingment data sheet for knowing which signal is assigned to a SoC Pins and follow SoC's technical reference manual for knowing the bit fields and supported configurations like Pull up, Pull Down, Rx Enable etc for updating the 32 bit Pin Mux Configuration register.

The file to be referred for modifying Pin Mux configurations is *Starterware_Root/board/SOCNAME/SOCNAME_BOARDNAME_pinmuxdata.c*

BOARDNAME is gpevm, idkevm, beaglebone etc.

SOCNAME is am437x or am335x.

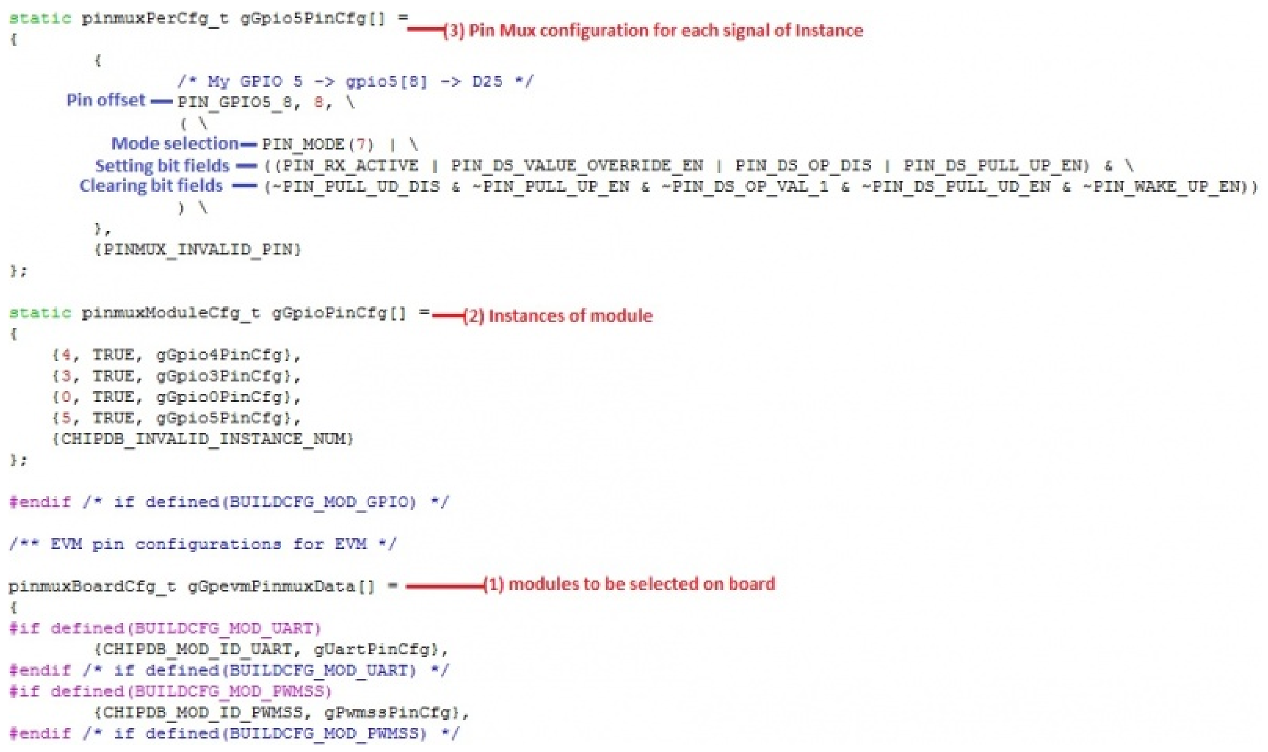
Pin Mux configuration structures are in the following hierarchical order.

(1) Modules on the board like UART, USB, GPMC,I2C, GPIO etc.

(2) Module Instances

(3) Pin Mux configuration for the signals of each module instances

Snap shot of code is provided for quick reference.


(3) Pin Mux configuration for each signal of Instance
Pin offset
Mode selection
Setting bit fields
Clearing bit fields
(2) Instances of module
(1) modules to be selected on board

```
static pinmuxPerCfg_t gGpio5PinCfg[] =
{
         {
                  /* My GPIO 5 -> gpio5[8] -> D25 */
                  PIN_GPIO5_8, 8, \
                  ( \
                  PIN_MODE(7) | \
                  ((PIN_RX_ACTIVE | PIN_DS_VALUE_OVERRIDE_EN | PIN_DS_OP_DIS | PIN_DS_PULL_UP_EN) & \
                  (~PIN_PULL_UD_DIS & ~PIN_PULL_UP_EN & ~PIN_DS_OP_VAL_1 & ~PIN_DS_PULL_UD_EN & ~PIN_WAKE_UP_EN))
                  ) \
         },
         {PINMUX_INVALID_PIN}
};

static pinmuxModuleCfg_t gGpioPinCfg[] =
{
    {4, TRUE, gGpio4PinCfg},
    {3, TRUE, gGpio3PinCfg},
    {0, TRUE, gGpio0PinCfg},
    {5, TRUE, gGpio5PinCfg},
    {CHIPDB_INVALID_INSTANCE_NUM}
};

#endif /* if defined(BUILDCFG_MOD_GPIO) */

/** EVM pin configurations for EVM */

pinmuxBoardCfg_t gGpevmPinmuxData[] =
{
#if defined(BUILDCFG_MOD_UART)
        {CHIPDB_MOD_ID_UART, gUartPinCfg},
#endif /* if defined(BUILDCFG_MOD_UART) */
#if defined(BUILDCFG_MOD_PWMSS)
        {CHIPDB_MOD_ID_PWMSS, gPwmssPinCfg},
#endif /* if defined(BUILDCFG_MOD_PWMSS) */
```

## Updating Device Data

Follow the steps to add a new device to the board data.

- To add new device to the board device data add a Macro in *Starterware_Root/include/device.h*
- Add the information of the device in *Starterware_Root/board/SOCNAME/SOCNAME_BOARDNAME.c*

Example:

```
Provide Macros for the devices to be added in include/device.h as
follows

    #define DEVICE_ID_EEPROM     (0x0U)
    #define DEVICE_ID_TMP275     (0x1U)
    #define DEVICE_ID_LED        (0x2U)

If the LED is being controlled by GPIO, The information related to the
GPIO Module instance need to be known to control the LEDs.
Device Data to control the LED through GPIO is provided for
convenience.
Device data is in board/SOCNAME/SOCNAME_BOARDNAME.c
SOCNAME am335x, am437x etc.
BOARDNAME beaglebone, gpevm, idkevm etc.

const boardDeviceData_t gBoardAm335xBeagleboneDevData[] =
{
   {       /* LED */
      DEVICE_ID_LED,               /* devId */      --> LED ID
      0U,                     /* devInstNum */ --> LED Instance ; If
more LEDs are present they will be added with devInstNum 1,2,3..
      CHIPDB_MOD_ID_GPIO,        /* ctrlModId */  --> Control Module
is GPIO
      1U,                   /* ctrlModInstNum */ --> GPIO Instance 1
 is used for controlling
      23U,                   /* GPIO pin number */ /* ctrlInfo */
--> Pin 23 of GPIO Instance 1 controls the LED
      CHIPDB_MOD_ID_INVALID,       /* dataModId */
      INVALID_INST_NUM,       /* dataModInstNum */
      NULL,                   /* pFnSelectDev */
      NULL,                   /* pFnResetDev */
      NULL                   /* pFnPowerOnDev */
  },
  {     /* EEPROM */
        ..,
  }
};

API declarations for accessing the board data are present in
include/device.h
Example for Usage of API:

chipdbModuleID_t BOARDGetDeviceCtrlModId(uint32_t devId, uint32_t
devInstNum);
devId       = DEVICE_ID_LED
devInstNum  = 0U
API returns CHIPDB_MOD_ID_GPIO which is the control module for LED.
```

## ChipDB details

ChipDB is used to abstract the SoC details across all the platforms through APIs and Headers. ChipDB contains the following details

- Number of IP Instances. Ex: 2 Istances of OCMCRAM
- Presence of IP.
- Base Address of IP
- Interrupt Number

### Usage of ChipDB

- All the Modules are assigned with unique number as in *chipdbModuleID_t* present in *Starterware_Root****/include/SOCNAME/chipdb_defs.h***

SOCNAME is am335x or am43xx etc.

```
typedef enum
{
    CHIPDB_MOD_ID_UART    = 0U,
    CHIPDB_MOD_ID_I2C     = 1U,
    CHIPDB_MOD_ID_MMCSD   = 2U,
    CHIPDB_MOD_ID_USB     = 3U,
    CHIPDB_MOD_ID_EDMA    = 4U,
    CHIPDB_MOD_ID_DMTIMER = 5U,
    CHIPDB_MOD_ID_COUNT   = 6U,
    CHIPDB_MOD_ID_INVALID = UINT32_MAX
} chipdbModuleID_t;
```

- No of Instances of each module can be evaluated from *gChipDBResourceIDMap[CHIPDB_MOD_ID_COUNT + 1]* from the file ***soc/SOCNAME/hw_SOCNAME_chipdb.c***

In the example CHIPDB_MOD_ID_COUNT = 6U.

```
const uint32_t gChipDBResourceIDMap[CHIPDB_MOD_ID_COUNT + 1] =
{
    0U,     /* UART */     /* UART has 3 istances , So following number
 should be 0U +3U = 3U */
    3U,     /* I2C */      /* I2C has 2 istances, So the following
number should be 3U+2U = 5U */
    5U,     /* MMCSD */    /* MMC has 2 istances, Following number
should be 5U+2U = 7U */
    7U,     /* USB */      /* 3 Istances of USB */
    10U,    /* EDMA */     /* 1 Istance of EDMA */
    11U,    /* DMTIMER */  /* 11 instances of DMTIMER */
    22U,    /* CHIPDB_RESOURCE_COUNT */
};
```

**No of Instaces of each module from ChipDB**

can be evaluated as follows.

```
No of Instance of MMCSD = gChipDBResourceIDMap[CHIPDB_MOD_ID_MMCSD +1]
- gChipDBResourceIDMap[CHIPDB_MOD_ID_MMCSD];
                        = gChipDBResourceIDMap[3] -
gChipDBResourceIDMap[2];
                        = 7U - 5U
                        = 2U
No of Instances of MMCSD is 2.
```

**Get Presence of each module from ChipDB**

Presence of each module can be known using Resource table which is encoded table to indicate the presence of module. The bit map is indexed using the Resouce ID.

```
gChipDBResourceTable[ ][ ] = { 0x35A5A5};
                         = {0b 11 0101 1010 0101 1'''0'''10 0101}
 To know the presence of MMCSD istance 2 get the value from Resouce
ID map.
 Resource ID is 6 for MMCSD Instance 2. 5 for MMCSD Instance 1.
 6th encoded bit of Resouce Table is 0
 0 ---> Instance not present
 1 ---> Instance is present

So, MMCSD Instance 2 is not present.
```

```
API to return the Presence of Module:
uint32_t CHIPDBIsResourcePresent(chipdbModuleID_t moduleID, uint32_t
instance);
```

**Get Base Address from ChipDB**

Base address can be fetched from the array ***gChipDBBaseAddrTable[ ]*** from the file ***soc/SOCNAME/hw_SOCNAME_chipdb_baseaddr.c***

```
const uint32_t gChipDBBaseAddrTable[ CHIPDB_RESOURCE_COUNT ] =
{
    BASE_ADDR_UART1,    /* Base address for 1st Instance of UART */
    BASE_ADDR_UART2,
    BASE_ADDR_UART3,
    BASE_ADDR_I2C1,
    BASE_ADDR_I2C2,
    BASE_ADDR_MMCSD1,
    BASE_ADDR_MMCSD2,
    BASE_ADDR_USB1,
    BASE_ADDR_USB2,
    ..
    ..
    ..
};
Total Number of Entries will be equal to the CHIPDB_RESOURCE_COUNT of
gChipDBResourceIDMap which is 22 in the example.

API to get the base address:
uint32_t CHIPDBBaseAddress(const chipdbModuleID_t moduleID,const
uint32_t instance);
```

**Get Interrupt Number from ChipDB**

Interrupt numbers for each module can be fetched from **gChipDBInterruptTables[ ]** from the file **soc/SOCNAME/hw_SOCNAME_chipdb_interrupt.c**

```
const uint16_t * gChipDBInterruptTables[ CHIPDB_RESOURCE_COUNT ] =
{

       gChipDBUART1InterruptMap,
       gChipDBUART2InterruptMap,
       ..,
       ..,
       ..,
};

Getting Interrupt numbers is similar like getting the base addresses.

API to get the interrupt Number:
uint32_t ChipDBInterruptNum(const chipdbModuleID_t moduleID, const
uint32_t instance, const uint32_t intSignal);
```

# Article Sources and Contributors

**StarterWare 02.01.01.XX Board and ChipDB Details Update Guide** *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=204765 *Contributors*: X0222293

# Image Sources, Licenses and Contributors

**Image:Pinmux code.jpg** *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Pinmux_code.jpg *License*: unknown *Contributors*: X0222293