DSP/BIOS UART Device Driver

USER'S GUIDE

Document Revision History

Rev No	Author(s)	Revision History	Date	Approval(s)
0.6	Chandan Nath	Updated for adding compiler switches in build options	May 20, 2008	Updating
0.5	Nagarjuna K	Updated for DM6437/C6424 and DM648/C6453	November 14, 2007	Updating
0.4	Nagarjuna K	Correcting build structure	July 10, 2007	Corrections
0.3	Nagarjuna K	Corrected version numbers fro tools used	June 15, 2007	Corrections
0.2	Nagarjuna K	Added to new formatting	May 23, 2007	Initial Draft
0.1	Vichu	Formatted to new template	May 9, 2007	Draft

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

This document is the reference guide for the uart driver and it explains how to configure and use the driver.

DSP/BIOS applications use the driver typically through UART APIs to perform read/write operations on the UART peripheral. UART is implemented as a simple wrapper on top of the GIO class driver and provides an application-specific interface. For more information on the DSP/BIOS device driver model and the GIO class driver, refer to the References section of this document.

1.1 Terms & Abbreviations

Term	Description
⚐	This bullet indicates important information. Please read such text carefully.
❑	This bullet indicates additional information.
API	Application Programming Interface
CSLR	Chip Support Library for Registers
DDC	Device Driver Core
IOM	Input/Output Mini driver (Device Driver Adapter)
ISR	Interrupt Service Routine
OS	Operating System
ROM	Read Only Memory
SOC	System On Chip

1.2 References

1.	Uart_fs.pdf	UART functional specification document
2.	SPRU-404g.pdf	DSP/BIOS Driver Guide

1.3 S/W Support

This UART device driver has been developed for the DSP/BIOS operating system using the TI supplied Chip Support Library. For more details on the version numbers refer to the release notes in the root of the installation.

1.4 Driver Components

The UART driver is constituted of following sub components:

UART IOM – Application facing, OS Specific Adaptation of UART Device Driver
UART DDC –OS Independent part of UART Driver Core
UART CSLR–The low-level UART h/w register overlay

System components:

PALOS – DSP/BIOS Abstraction

Below Figure shows UART driver architecture.

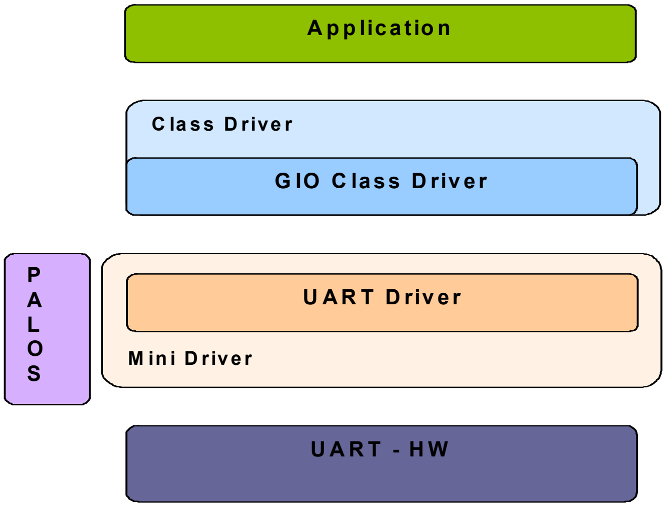


Figure 1. UART driver architecture

1.5 Default Driver Configuration

1.1. Default Driver Configuration

By default, the UART driver is configured as follows:

UART module clock frequency (Hz)	(Uint32) 117000000
UART internal FIFO enable	(Bool) TRUE
Driver operating mode	Interrupt
Loop back enable/disable	(Bool) FALSE
Baud rate	115.2Kbps
Number of stop bits	1
Parity	None
Character Length	8
Flow control	None

1.6 Driver Capabilities

The DSP/BIOS UART Device Driver supports multi-instances (But in DM648/C6452 there is only one instance of the UART) and it is re-entrant safe driver. In addition, it provides synchronous IO. The driver operates in the following modes: Polled and Interrupts modes. The driver has built-in software transmit internal buffer for improved synchronous output response times.

The significant driver features are:

- Isolates H/W and OS Accesses.
- Easy to maintain & re-target to new platforms.
- Supports Multiple Instances.
- Provides synchronous and asynchronous mode support to the application. Asynchronous mode support can be enabled/disabled by defining the preprocessor macro PSP_UART_ASYNC_MODE_SUPPORT in the driver.

1.7 Driver Limitations

- Baud-rates more than 115200 are not supported in UART

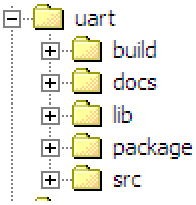
1.8 System Requirements

- Refer to release notes for the details on Target environment, BIOS version, XDC version…

2 Installation Guide

2.1 Component Folder

Upon installing the UART driver the following directory structure is found in the driver's directory.



- uart
  - build
  - docs
  - lib
  - package
  - src

Figure 2. UART Driver Directory Structure

This top level uart folder contains uart driver psp header file and XDC package files (package.bld, package.xdc and package.xs)

- **build:** This folder contains uart driver library project file. The generated driver library shall be included in the application where UART driver have to be used.
- **docs:** This folder contains architecture document, datasheet, release notes, user guide and doxugen compiled api reference document.

  Architecture document contains the driver details which can be helpful for the developers as well as consumers to understand the driver design.

  Datasheet gives the idea about the memory consumption by the driver and description of the top level APIs.

  Release Note gives the details about system requirements, steps to Install/Uninstall the package.This document list the known issues of the driver.

  User Guide provides information about how to use the driver. It contains description of sample applications which guide the end user to make their applications using this driver.

  API reference document gives the details about the API's used in UART driver.
- **Lib:** This folder contains libraries generated in all the configuration modes(debug, idebug, irelease and release)
- **Package:** This folder contains files generated by XDC tool.
- **src:** This folder contains uart driver source files. It also contains header files that are used by the driver.

2.2 Build

This section describes for each supported target environment, the applicable build options, supported configurations and how selected, the featured capabilities and how enabled, the allowed user customizations for the software to be installed and how the same can be realized.

The component might be delivered to user in different formats:

- Source-less i.e. binary executables and object libraries only.
- Source-inclusive i.e.The entire source code is used to implement the driver is included in the delivered product.
- Source-selective ie. Only a part of the overall source is included. This delivery mechanism might be required either because ;certain parts of the driver require soruce level extensions and/or customization at the user's end or because,specific parts of the driver is exposed to user at the source level to insure user's software development.

When source is included as part of the product delivery, the CCS project file is provided as part of the package. When object format is distributed, the driver header files are part of the "src" folder and the driver library is provided in *"\pspdrivers\lib"* folder.

2.3 Build Options

To compile driver, change build options as mentioned below:

The build folder contains a CCS project file that builds the driver into a library for debug, idebug, irelease, release mode.

Following compiler switches are used to compile for different options.

- **_DEBUG**
  This is used as a flag to compiler whether to include the debug statement inserted in the code into the final image. This flag helps to build DEBUG image of the program. For RELEASE images this is not passed to the compiler.
- **CHIP_XXXX**
  The CSL layer is written in a common file for all the variants of a SOC. This flag differentiates the variant we are compiling for, for e.g. - CHIP_DM648, and the CSL definitions for that variant appropriately gets defined for registers base addresses, num of ports of a peripheral etc.
- **UART_INSTRUMENTATION_ENABLED**
  This flag is passed to the compiler to include the instrumentation code parts into the final image/lib of the program. This helps build the iRelease/iDebug versions of the image/lib with a common code base
- **PSP_UART_ASYNC_MODE_SUPPORT**
  This option is used to support asynchronous mode of operation.
- **PSP_UART_EDMA_SUPPORT**
  This option is passed to the compiler for EDMA mode.
- **PSP_UART_BUFFERING_ENABLED**
  This option is passed to the compiler when Buffering mode is enabled.
- **UART_DEBUG_PRINTF**
  They are the macros used to print debug messages. These expand a valid print statement.

3 DSP/BIOS UART DRIVER Structures

This section discusses about the initialization details and initialization structures used in the uart driver. Please note that for some structure member information/details the DM648/C6452 uart peripheral API reference document might need to be referred.

3.1.1 Initialization details

To use UART device driver, a device entry must be added and configured in the DSP/BIOS configuration tool.

To have UART device driver included in the application, corresponding TCI file have to be included in BIOS TCF (i.e. *"dm648_uart.tci" for DM648)* must be included in BIOS TCF file of the application for using UART instance of the driver.

The following are the device configuration settings required to use the uart driver.
Note: This has to be done for all of the required driver instances.

TCI Configuration Parameters	**Description**
initFxn - Init Function	Pointer to the application function used to prepare UART module and initialize init time configuration data structures. This function typically does enabling UART module in LPSC, PIN muxing and setting uart driver configurations.
fxnTable - Function Table Pointer	*UART_Fxns*. This is a global variable, which points to the UART driver APIs.
fxnTableType - Function Table Type	*IOM_Fxns*
deviceId - Device Id	Specify which UART to use. For example to use UART 0 this should be given as 0.
params – Pointer to Port parameter	A pointer to an object of type *uart0DevParams* as defined in the header file *psp_uart.h*. This pointer will point to a device parameter structure. In BIOS TCI files, this structure object is passed as an argument. Application should declare and initialize the structure object properly.
Device Global Data Pointer	N/A, not used by this driver

Final tci file should contain the following details which were explained above:

```
bios.UDEV.create("UART0");
bios.UDEV.instance("UART0").fxnTableType = "IOM_Fxns";
bios.UDEV.instance("UART0").initFxn = prog.extern("uart0_dev_init");
bios.UDEV.instance("UART0").params = prog.extern("uart0DevParams");
bios.UDEV.instance("UART0").fxnTable = prog.extern("UARTMD_FXNS");
```

3.1.2 UART device params (*PSP_uartDevParams*)

"psp_uart.h" file contains *PSP_uartDevParams* data structure that is passed while mdBindDev call which is defined with UDEV UART parameters in *.tcf file of application. The members of this structure are explained below:

Structure Members	Description
Uint32	inputFreq
Bool	fifoEnable
PSP_ OpMode	opMode
Bool	loopbackEnabled
PSP_UartConfigParams	xferConfigParams

3.1.2.1 PSP_UartConfigParams

Following table describes the parameters contained in PSP_UartConfigParams (xferConfigParams) data structure available in "psp_uart.h" file:

Structure Members	Description
PSP_UartBaudRate	baudrate
PSP_UartNumStopBits	stopbits
PSP_UartCharLen	charlen
PSP_UartParity	parity
PSP_UartRxTrigLvl	rxThreshold
PSP_UartFlowControl	Fc

3.1.2.2 PSP_UartBaudRate

Following table describes the parameters contained in PSP_UartConfigParams (baudrate) enum, which is a parameter of xferConfigParams and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_BAUD_RATE_2_4K	Baud Rate of 2400 kbps
PSP_UART_BAUD_RATE_4_8K	Baud Rate of 4800 kbps
PSP_UART_BAUD_RATE_9_6K	Baud Rate of 9600 kbps
PSP_UART_BAUD_RATE_19_2K	Baud Rate of 19200 kbps
PSP_UART_BAUD_RATE_38_4K	Baud Rate of 38400 kbps
PSP_UART_BAUD_RATE_57_6K	Baud Rate of 57600 kbps
PSP_UART_BAUD_RATE_115_2K	Baud Rate of 115200 kbps

3.1.2.3 PSP_UartNumStopBits

Following table describes the parameters contained in *PSP_UartNumStopBits* (stopbits) enum, which is a parameter xferConfigParams and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_NUMSTOP_1	Number of Stop bits = 1
PSP_UART_NUMSTOP_1_5	Number of Stop bits = 1.5
PSP_UART_NUMSTOP_2	Number of Stop bits = 2

3.1.2.4 PSP_UartCharLen

Following table describes the parameters contained in *PSP_UartCharLen* (charlen) enum, which is a parameter xferconfigParams and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_CHARLEN_5	Character length of 5
PSP_UART_CHARLEN_6	Character length of 6
PSP_UART_CHARLEN_7	Character length of 7
PSP_UART_CHARLEN_8	Character length of 8

3.1.2.5 PSP_UartParity

Following table describes the parameters contained in *PSP_UartParity* (parity) enum, which is a parameter xferConfigParams and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_PARITY_ODD	Odd Parity
PSP_UART_PARITY_EVEN	Even Parity
PSP_UART_PARITY_NONE	No Parity

3.1.2.6 PSP_UartRxTrigLvl

Following table describes the parameters contained in *PSP_UartRxTrigLvl* (*FC*) enum, which is a parameter xferConfigParams and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_TRIGGER_LEVEL_1	Trigger level for 1 Byte
PSP_UART_TRIGGER_LEVEL_4	Trigger level for 4 Byte
PSP_UART_TRIGGER_LEVEL_8	Trigger level for 8 Byte
PSP_UART_TRIGGER_LEVEL_14	Trigger level for 14 Byte

3.1.2.7 PSP_UartFlowControl

Following table describes the parameters contained in *PSP_UartFlowControl* (*rxThreshold*) data structure, which is a parameter xferConfigParams and details of the same is available in "psp_uart.h" file:

Structure Members	**Description**
PSP_UartFcType	fcType
PSP_UartFcParam	fcParam

3.1.2.8 PSP_UartFcType

Following table describes the parameters contained in *PSP_UartFcType* (*FC type*) enum, which is a parameter PSP_UartFlowControl and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_FLOWCONTROL_NONE	Neither H/W nor S/W flow Control
PSP_UART_FLOWCONTROL_SW	Software Flow Control
PSP_UART_FLOWCONTROL_HW	Hardware Flow Control

3.1.2.9 PSP_UartFcParam

Following table describes the parameters contained in *PSP_UartFcParam* (*FC param*) enum, which is a parameter PSP_UartFlowControl and details of the same is available in "psp_uart.h" file:

Structure Members	Description
PSP_UART_FC_NONE	No Flow Control
PSP_UART_FC_XONXOFF_1	Receiver/Transmitter compares XON1,XOFF1
PSP_UART_FC_XONXOFF_2	Receiver/Transmitter compares XON2,XOFF2
PSP_UART_FC_XONXOFF_12	Receiver/Transmitter compares XON1,XON2,XOFF1,XOFF2

PSP_UART_FC_AUTO_RTS_CTS	HW FC - Auto RTS CTS
PSP_UART_FC_AUTO_CTS_ONLY	HW FC - Auto CTS only

3.1.3 UART channel params (PSP_uartIoParams)

The file psp_uart.h has the **PSP_uartIoParams** data structure that is passed to GIO_create. The parameters are explained below:

Structure Members	Description
PSP_Handle hEdma	Handle to EDMA object

UART API's

This chapter describes the functions, data structures, enumerations and macros for the UART driver module.

The following API functions are defined by the GIO module:

GIO_create	Allocate and initialize an UART channel object
GIO_delete	De-allocate an UART channel object
GIO_control	Send a control command to the mini-driver
GIO_submit	API used to transfer the data with slaves

3.2 Structures Passed to GIO APIs

3.2.1 Structure for data parameter passed to GIO_create

The file psp_uart.h has the **PSP_uartIoParams** data structure that is passed to GIO_create The parameters are explained below:

Structure Members	Description
PSP_Handle hEdma	Handle to EDMA (Note : This should be always NULL since the driver wont support DMA mode)

3.2.2 Structure for data parameter passed to GIO_submit

The file psp_uart.h has the **PSP_uartDataParam** data structure that is passed to GIO_submit. The parameters are explained below:

Structure Members	Description
addr	IO Buffer address
Timeout	Timeout value for this IO packet

3.2.3 Enum for IOCTL

Following are the enumerations passed as command argument while GIO_control call.

Structure Members	Description
PSP_UART_IOCTL_SET_BAUD	To set the Baud Rate
PSP_UART_IOCTL_SET_STOPBITS	To set the number of stopbits
PSP_UART_IOCTL_SET_DATABITS	To set the data bits
PSP_UART_IOCTL_SET_PARITY	To set the Parity
PSP_UART_IOCTL_SET_FLOWCONTROL	To set the Flow control if enabled
PSP_UART_IOCTL_SET_TRIGGER_LEVEL	To set the trigger level if the FIFO is enabled
PSP_UART_IOCTL_RESET_RX_FIFO	To reset the receiver FIFO
PSP_UART_IOCTL_RESET_TX_FIFO	To reset the transmitter FIFO
PSP_UART_IOCTL_CANCEL_CURRENT_IO	To Cancel the io requests
PSP_UART_IOCTL_GET_STATS	To get the DDC Stats
PSP_UART_IOCTL_CLEAR_STATS	To clear the DDC Stats
PSP_UART_IOCTL_MAX_IOCTL	Number of max. IOCTLS (House

3.3 API Definition

3.3.1 GIO_ create

Syntax

```
GIO_Handle GIO_create (
                String name,
                Int mode,
                Int *status,
                Ptr chanParams,
                GIO_Attrs *attrs
                );
```

Parameters

name

The name argument is the name specified for the device when it was created in the configuration or at runtime. It is used to find a matching name in the device table.
Note: strings are case sensitive.

For UART drivers the string contains one token '/' followed by the name.

- UART driver or port instance
  This identifies the UART driver or port instance and this will be typically *"/UART0", "/UART1"* and so on, where suffix to UART denotes instance ID. This string depends on the device registration string given in BIOS driver TCI file.

mode

The mode argument specifies the mode in which the device is to be opened. This will be either IOM_INPUT or IOM_OUTPUT.

status

The status argument is an output parameter that this function fills with a pointer to the status that was returned by the mini-driver.

chanParams

GIO_Create for uart requires PSP_uartIoParams_t as its channel parameter and details of the same is explained in section 3.2.1.

attrs

The attrs parameter is a pointer to a structure of type *GIO_Attrs*. This is not supported and NULL should be passed.

Return Value

It returns the handle of type *GIO_Handle* on successful opening of a device. It returns NULL if the device could not be opened.

Description

An application calls *GIO_create* to create and initialize a uart driver channel to the driver.

Constraints

This function can only be called after the device has been loaded and initialized.

Example

The example below shows creation of Channel for UART

```
GIO_Handle chanHandle;
GIO_Attrs gioAttrs = GIO_ATTRS;
PSP_uartIoParams_t chanParams;

chanParams.hEdma = NULL;

chanHandle = GIO_create("/UART0", IOM_INPUT,
                             NULL, &chanParams, &gioAttrs);
if (NULL == chanHandle)
{
       printf(" Failed create UART0 channel \r\n");
       return;
}
```

3.3.2 GIO_delete

Syntax

int GIO_delete(GIO_Handle gioChan);

Parameters

Handle of the uart driver channel that was created with a call to GIO_create.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

This function call will close the logical channel associated with GIO_create. It will also free the buffers allocated by driver.

Constraints

This function can only be called after the device has been loaded, initialized and created.

Example

The example below shows creation and deletion of Channel for UART

```
GIO_Handle chanHandle;
chanHandle = GIO_create("/UART0", IOM_INOUT,
                    NULL, NULL, NULL);
if (NULL == chanHandle)
{
       printf(" Failed creating channel \r\n");
}
Status = GIO_delete(chanHandle);
If (status != IOM_COMPLETED)
{
       printf(" Failed deleting channel \r\n");
}
```

3.3.3 GIO_control

Syntax

status = GIO_control (gioChan, cmd, args);

Parameters

gioChan

Handle of the UART driver channel that was created with a call to *GIO_create*.

cmd

The cmd argument specifies the control command

args

The args argument is a pointer to the argument or structure of arguments that are specific to the command being passed.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

An application calls *GIO_control* to send device-specific control commands to the mini-driver.

IOCTL commands available for UART driver are available in section 3.2.3

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle chanHandle;


/* channel creation should be done here  using GIO_create.*/

/* Invoking IOCTL using GIO_control */
status = GIO_control(chanHandle, PSP_UART_IOCTL_SET_BAUD, NULL);

if (IOM_COMPLETED != status)
{
        printf(" Failed to set baud rate \r\n");
        return;
}
```

3.3.4 GIO_Submit

Syntax

```
status = GIO_submit (        GIO_Handle gioChan,
                             Uns cmd,
                             ptr bufP,
                             Uns* Psize,
                             GIO_AppCallBack* appCallback
                        )
```

Parameters

gioChan

Handle of the UART driver channel that was created with a call to *GIO_create*.

cmd

The cmd argument specifies the control command, which can be any of the following

- IOM_READ /* for read operation */
- IOM_WRITE /* for write operation */
- IOM_FLUSH /* for flushing the packets */
- IOM_ABORT /* for aborting the packets */

bufP

The bufP argument is a pointer to the buffer which need to be written or data to read, and this value should be used only when cmd is used as IOM_READ or IOM_WRITE. For other two commands it is NULL.

Psize

The Psize argument is a pointer to the argument of data type size_t, and this value represents the number of bytes of data to be written or to read.

appCallback

The Psize argument is a function pointer, which will be called once the submit operation is completed. Since UART works in synchronous mode of operation this parameter will be NULL.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

This function is called by the application to perform the read/write/flush/abort operation.

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle gioChan;
size_t size = 4;
Uint8 wBuffer[4] = {'T','E','S','T'};
PSP_uartDataParam     buf;

/* channel creation and allocBuffer should be done here */
       buf.addr      = wBuffer;
       buf.timeout = 1000;
status = GIO_submit (gioChan,IOM_WRITE,&rBuffer, &size, NULL) ;
```

3.3.5 GIO_read/GIO_write

Syntax

```
status = GIO_read/GIO_write (      GIO_Handle gioChan,
                                ptr bufP,
                                Uns* Psize,
                        )
```

Parameters

gioChan

Handle of the Uart driver channel that was created with a call to *GIO_create*.

bufP

The bufP argument is a pointer to the buffer, which need to be written, or data to read.

Psize

The Psize argument is a pointer to the argument of data type size_t, and this value represents the number of bytes of data to be written or to read.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code

Description

- Similar to GIO_submit with cmd wither IOM_READ or IOM_WRITE with appCallBack value is NULL.

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle gioChan;
size_t size = 100;
Uint8 rBuffer [100];
 PSP_uartDataParam     buf;

/* channel creation and allocBuffer should be done here */
        buf.addr            = rBuffer;
        buf.timeout = 1000;

      Status = GIO_read (gioChan, &buf, &size);
```

3.3.6 GIO_flush/GIO_abort

Syntax

Status = GIO_flush/GIO_abort (GIO_Handle gioChan)

Parameters

gioChan

Handle of the UART driver channel that was created with a call to *GIO_create*.

Return Value

On success driver returns IOM_COMPLETED, on failure/error condition IOM error code. In the current implementation of UART driver this is not implemented so it returns IOM_ENOTIMPLEMENTED error.

Description

Similar to GIO_submit with cmd either IOM_FLUSH or IOM_ABORT with appCallBack value is NULL.

Constraints

This function can only be called after the device has been loaded, initialized and created. The handle supplied as an argument to this function should have been obtained with a previous call to *GIO_create*.

Example

```
GIO_Handle gioChan;

/* channel creation and allocBuffer should be done here */

        status = GIO_flush (gioChan);
```

4 Example Applications

This section describes the example applications that are included in the package. These sample application can be run as is for quick demonstration, but the user will benefit most by using these samples as sample source code in developing new applications.

5.1 Writing Applications for UART

This section provides guidance to user for writing their own application for UART drivers.

5.1.1. *File Inclusion*

To write sample application user has to include following header files in the application:

1. **std.h**

   This file contains standard data types, macros and structures.

2. **gio.h**

   This file contains GIO layer macros and structures. These macros are wrapper macros to form a wrapper above GIO.

3. **tsk.h**

   This file contains all task module details.

4. **psp_uart.h**

   This file contains uart parameters which are passed to driver at the time of UART driver registration with BIOS.

5.2 Sample Applications

5.2.1. *Introduction*

The sample application is a representative test program. Initialization of UART driver is done by calling initialization function from BIOS.

5.2.2. *Building the application*

Please follow below steps to build sample application:

- Open CCS 3.3 setup. Import proper CCS configuration file. Set the proper CCS Gel file (Refer PSP_Release_Notes.pdf for details). Click on “Save & Quit” button and exit the setup.
- Open sample application as mentioned in *“<root>\pspdrivers\system\<soc>\bios\<evmNAME>\uart\build\<uartsample pjt>”*.

- For example for C6424, its *"<root>\pspdrivers\system\c6424\bios\evm6424\uart\build\c6424_evm_uart_st_sample.pjt"*.
- Compile this project using Project->Build
- Note: Following Components needs to be linked for successful build and functionality of the application.
    - I2C
    - PAL_OS
    - SoC specific PAL_SYS

5.2.3. *Loading and running the application*

The sample application is loaded and executed via Code composer studio. It is good idea to reset the board before loading Code Composer. The application will print out the status messages and type of functionality the driver performs on the message log.

5.2.4. *Sample UART application*

- **Sample directory structure**

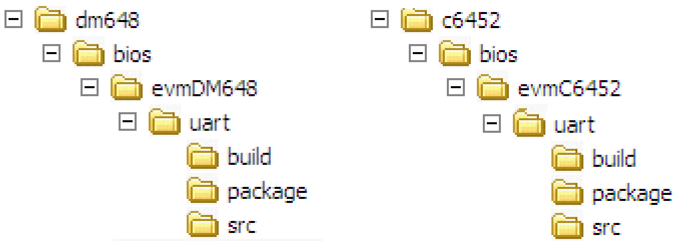


Figure 3. UART sample directory structure for DM648 and C6452

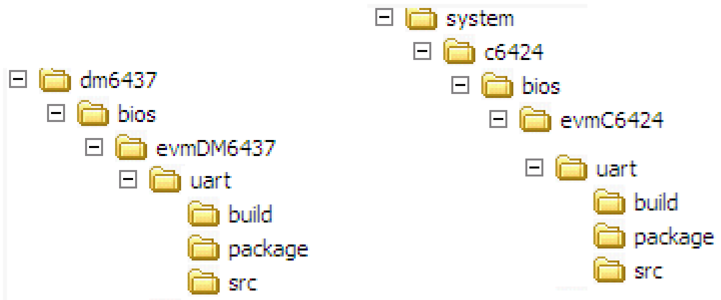


Figure 4. UART sample directory structure for DM6437 and C6424

Top level folder shown in the above figure contains header and tci files required specifically for sample application along with XDC packaging files(package.bld and package.xdc)

build: This subfolder in the sample folder contains project for sample appliaction.

src: This folder contains uart sample application source files. It also contains header files related to uart driver that are used by the sample application if any.

Package: This folder contains files generated by XDC tool

The sample application performs read/write operation to Terminal connected to COM port.

❖ UART Configuration Parameters

UART_devParams used during UART driver registration with BIOS using TCI files. UART_devParams is configured as follows. Also refer section **"3.1.1 Initialization details"** for configuring TCI file.

```
PSP_uartDevParams Uart_DevParams =
            {
              /* 11700000 (clock frequency) */
               PSP_UART_MODULE_CLOCK,
               /* FIFO enabled */
              TRUE,
               /* Interrupt Mode of Operation */
              PSP_OPMODE_INTERRUPT,
               /* No loop back mode*/
              FALSE,
               /* Baud rate -  115.2 kbps*/
              PSP_UART_BAUD_RATE_115_2K,
               /* 1 stop bits */
              PSP_UART_NUM_STOP_BITS_1,
               /* 8 character length*/
              PSP_UART_CHARLEN_8,
               /* No parity*/
              PSP_UART_PARITY_NONE,
               /* 14 – trigger level*/
              PSP_UART_RX_TRIG14,
               /* No flow control type*/
              PSP_UART_FC_TYPE_NONE,
               /* No Flow control*/
              PSP_UART_FC_NONE
            }
```

❖ UART Data Param Configuration Parameters

UART data param structure (already explained at section 3.2.2). So following are the configurations to be done for performing transfer.

```
  PSP_uartDataParam buf;
   Int status   =  0;
   size_t len   =  0;

   /* Input buffer */
   buf.addr = TestStringStart;
   /*Time out valueb*/
   buf.timeout = SYS_FOREVER;
   /*Length of data*/
   len = SIZEOF_UART_STRING_START;
   status = GIO_write(hUart_OUT, &buf, &len);
```

❖ Driver naming convention used for Channel creation

Application calls GIO_create to create and initialize a UART driver channel for both input and output channel.

Edma handle is the channel parameter, whose details are already mentioned in section 4.1.1.

❖ Pragma directives used in the sample application

The uartTestStringStart and Uart_TxBuffer are used by the application and need to be cache aligned at 128 bytes.

5.2.5.Switch Settings:

5.2.5.1. Settings for DM6437/C6424:

None

5.2.5.2. Settings for DM648/C6452:

DIP-switch settings are to be made properly in order to see the output on tera-term / hyper terminal / some other software used to see the data coming from COM port.
Details of above mentioned settings can be seen in DM648/C6424 evm specification document.