```c
/*
 * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#if defined(UART_INSTANCE_COUNT)
#include "fsl_uart_hal.h"
#endif
#if defined(LPUART_INSTANCE_COUNT)
#include "fsl_lpuart_hal.h"
#endif
#if defined(UART0_INSTANCE_COUNT)
#include "fsl_lpsci_hal.h"
#endif
#include "fsl_clock_manager.h"
#include "fsl_os_abstraction.h"
#include "print_scan.h"

#if (defined(USB_INSTANCE_COUNT) &&  (defined(BOARD_USE_VIRTUALCOM)))
  #include "usb_device_config.h"
  #include "usb.h"
  #include "usb_device_stack_interface.h"
  #include "usb_descriptor.h"
  #include "virtual_com.h"
#endif

extern uint32_t g_app_handle;
#if __ICCARM__
#include <yfuns.h>
#endif

static int debug_putc(int ch, void* stream);

/*************************************************************************
 * Definitions
 *************************************************************************/
```

```
64
65  /*! @brief Operation functions definiations for debug console. */
66  typedef struct DebugConsoleOperationFunctions {
67      union {
68          void (* Send)(void *base, const uint8_t *buf, uint32_t count);
69  #if defined(UART_INSTANCE_COUNT)
70          void (* UART_Send)(UART_Type *base, const uint8_t *buf, uint32_t count);
71  #endif
72  #if defined(LPUART_INSTANCE_COUNT)
73          void (* LPUART_Send)(LPUART_Type* base, const uint8_t *buf, uint32_t count);
74  #endif
75  #if defined(UART0_INSTANCE_COUNT)
76          void (* UART0_Send)(UART0_Type* base, const uint8_t *buf, uint32_t count);
77  #endif
78  #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
79          void (* USB_Send)(uint32_t base, const uint8_t *buf, uint32_t count);
80  #endif
81      } tx_union;
82      union{
83          void (* Receive)(void *base, uint8_t *buf, uint32_t count);
84  #if defined(UART_INSTANCE_COUNT)
85          uart_status_t (* UART_Receive)(UART_Type *base, uint8_t *buf, uint32_t count);
86  #endif
87  #if defined(LPUART_INSTANCE_COUNT)
88          lpuart_status_t (* LPUART_Receive)(LPUART_Type* base, uint8_t *buf, uint32_t count);
89  #endif
90  #if defined(UART0_INSTANCE_COUNT)
91          lpsci_status_t (* UART0_Receive)(UART0_Type* base, uint8_t *buf, uint32_t count);
92  #endif
93  #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
94          usb_status_t (* USB_Receive)(uint32_t base, uint8_t *buf, uint32_t count);
95  #endif
96
97      } rx_union;
98  } debug_console_ops_t;
99
100 /*! @brief State structure storing debug console. */
101 typedef struct DebugConsoleState {
102     debug_console_device_type_t type;/*<! Indicator telling whether the debug console is inited. */
103     uint8_t instance;              /*<! Instance number indicator. */
104     void*   base;                 /*<! Base of the IP register. */
105     debug_console_ops_t ops;       /*<! Operation function pointers for debug uart operations. */
106 } debug_console_state_t;
107
108 /*******************************************************************************
109  * Variables
110  ******************************************************************************/
111 /*! @brief Debug UART state information.*/
112 static debug_console_state_t s_debugConsole;
113
114 /*******************************************************************************
115  * Code
116  ******************************************************************************/
117 /* See fsl_debug_console.h for documentation of this function.*/
118 debug_console_status_t DbgConsole_Init(
119         uint32_t uartInstance, uint32_t baudRate, debug_console_device_type_t device)
120 {
121     if (s_debugConsole.type != kDebugConsoleNone)
122     {
123         return kStatus_DEBUGCONSOLE_Failed;
124     }
125
126     /* Set debug console to initialized to avoid duplicated init operation.*/
127     s_debugConsole.type = device;
128     s_debugConsole.instance = uartInstance;
129
130
```

```
    /* Switch between different device. */
    switch (device)
    {
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))  /*&& defined()*/
        case kDebugConsoleUSBCDC:
        {
            VirtualCom_Init();
            s_debugConsole.base = (void*)g_app_handle;
            s_debugConsole.ops.tx_union.USB_Send = VirtualCom_SendDataBlocking;
            s_debugConsole.ops.rx_union.USB_Receive = VirtualCom_ReceiveDataBlocking;
        }
        break;
#endif
#if defined(UART_INSTANCE_COUNT)
        case kDebugConsoleUART:
          {
            UART_Type * g_Base[UART_INSTANCE_COUNT] = UART_BASE_PTRS;
            UART_Type * base = g_Base[uartInstance];
            uint32_t uartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableUartClock(uartInstance);

            /* UART clock source is either system or bus clock depending on instance */
            uartSourceClock = CLOCK_SYS_GetUartFreq(uartInstance);

            /* Initialize UART baud rate, bit count, parity and stop bit. */
            UART_HAL_SetBaudRate(base, uartSourceClock, baudRate);
            UART_HAL_SetBitCountPerChar(base, kUart8BitsPerChar);
            UART_HAL_SetParityMode(base, kUartParityDisabled);
#if FSL_FEATURE_UART_HAS_STOP_BIT_CONFIG_SUPPORT
            UART_HAL_SetStopBitCount(base, kUartOneStopBit);
#endif

            /* Finally, enable the UART transmitter and receiver*/
            UART_HAL_EnableTransmitter(base);
            UART_HAL_EnableReceiver(base);

            /* Set the funciton pointer for send and receive for this kind of device. */
            s_debugConsole.ops.tx_union.UART_Send = UART_HAL_SendDataPolling;
            s_debugConsole.ops.rx_union.UART_Receive = UART_HAL_ReceiveDataPolling;
          }
          break;
#endif
#if defined(UART0_INSTANCE_COUNT)
        case kDebugConsoleLPSCI:
          {
            /* Declare config sturcuture to initialize a uart instance. */
            UART0_Type * g_Base[UART0_INSTANCE_COUNT] = UART0_BASE_PTRS;
            UART0_Type * base = g_Base[uartInstance];
            uint32_t uartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableLpsciClock(uartInstance);

            uartSourceClock = CLOCK_SYS_GetLpsciFreq(uartInstance);

            /* Initialize LPSCI baud rate, bit count, parity and stop bit. */
            LPSCI_HAL_SetBaudRate(base, uartSourceClock, baudRate);
            LPSCI_HAL_SetBitCountPerChar(base, kLpsci8BitsPerChar);
            LPSCI_HAL_SetParityMode(base, kLpsciParityDisabled);
#if FSL_FEATURE_LPSCI_HAS_STOP_BIT_CONFIG_SUPPORT
            LPSCI_HAL_SetStopBitCount(base, kLpsciOneStopBit);
#endif

            /* Finally, enable the LPSCI transmitter and receiver*/
            LPSCI_HAL_EnableTransmitter(base);
```

```
198             LPSCI_HAL_EnableReceiver(base);
199
200             /* Set the funciton pointer for send and receive for this kind of device. */
201             s_debugConsole.ops.tx_union.UART0_Send = LPSCI_HAL_SendDataPolling;
202             s_debugConsole.ops.rx_union.UART0_Receive = LPSCI_HAL_ReceiveDataPolling;
203         }
204         break;
205 #endif
206 #if defined(LPUART_INSTANCE_COUNT)
207       case kDebugConsoleLPUART:
208         {
209             LPUART_Type* g_Base[LPUART_INSTANCE_COUNT] = LPUART_BASE_PTRS;
210             LPUART_Type* base = g_Base[uartInstance];
211             uint32_t lpuartSourceClock;
212
213             s_debugConsole.base = base;
214             CLOCK_SYS_EnableLpuartClock(uartInstance);
215
216             /* LPUART clock source is either system or bus clock depending on instance */
217             lpuartSourceClock = CLOCK_SYS_GetLpuartFreq(uartInstance);
218
219             /* initialize the parameters of the LPUART config structure with desired data */
220             LPUART_HAL_SetBaudRate(base, lpuartSourceClock, baudRate);
221             LPUART_HAL_SetBitCountPerChar(base, kLpuart8BitsPerChar);
222             LPUART_HAL_SetParityMode(base, kLpuartParityDisabled);
223             LPUART_HAL_SetStopBitCount(base, kLpuartOneStopBit);
224
225             /* finally, enable the LPUART transmitter and receiver */
226             LPUART_HAL_SetTransmitterCmd(base, true);
227             LPUART_HAL_SetReceiverCmd(base, true);
228
229             /* Set the funciton pointer for send and receive for this kind of device. */
230             s_debugConsole.ops.tx_union.LPUART_Send = LPUART_HAL_SendDataPolling;
231             s_debugConsole.ops.rx_union.LPUART_Receive = LPUART_HAL_ReceiveDataPolling;
232
233         }
234         break;
235 #endif
236       /* If new device is requried as the low level device for debug console,
237        * Add the case branch and add the preprocessor macro to judge whether
238        * this kind of device exist in this SOC. */
239       default:
240         /* Device identified is invalid, return invalid device error code. */
241         return kStatus_DEBUGCONSOLE_InvalidDevice;
242   }
243
244   /* Configure the s_debugConsole structure only when the inti operation is successful. */
245   s_debugConsole.instance = uartInstance;
246
247   return kStatus_DEBUGCONSOLE_Success;
248 }
249
250 /* See fsl_debug_console.h for documentation of this function.*/
251 debug_console_status_t DbgConsole_DeInit(void)
252 {
253   if (s_debugConsole.type == kDebugConsoleNone)
254   {
255     return kStatus_DEBUGCONSOLE_Success;
256   }
257
258   switch(s_debugConsole.type)
259   {
260 #if defined(UART_INSTANCE_COUNT)
261       case kDebugConsoleUART:
262         CLOCK_SYS_DisableUartClock(s_debugConsole.instance);
263         break;
264
```

```
#endif
#if defined(UART0_INSTANCE_COUNT)
        case kDebugConsoleLPSCI:
            CLOCK_SYS_DisableLpsciClock(s_debugConsole.instance);
            break;
#endif
#if defined(LPUART_INSTANCE_COUNT)
        case kDebugConsoleLPUART:
             CLOCK_SYS_DisableLpuartClock(s_debugConsole.instance);
            break;
#endif
        default:
            return kStatus_DEBUGCONSOLE_InvalidDevice;
    }

    s_debugConsole.type = kDebugConsoleNone;

    return kStatus_DEBUGCONSOLE_Success;
}

#if (defined(__KSDK_STDLIB__))
int _WRITE(int fd, const void *buf, size_t nbytes)
{
    if (buf == 0)
    {
        /* This means that we should flush internal buffers.  Since we*/
        /* don't we just return.  (Remember, "handle" == -1 means that all*/
        /* handles should be flushed.)*/
        return 0;
    }


    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buf, nbytes);
    return nbytes;

}

int _READ(int fd, void *buf, size_t nbytes)
{

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buf, nbytes);
    return nbytes;
}
#elif __ICCARM__

#pragma weak __write
size_t __write(int handle, const unsigned char * buffer, size_t size)
{
    if (buffer == 0)
    {
        /* This means that we should flush internal buffers.  Since we*/
        /* don't we just return.  (Remember, "handle" == -1 means that all*/
```

```
        /* handles should be flushed.)*/
        return 0;
    }

    /* This function only writes to "standard out" and "standard err",*/
    /* for all other file handles it returns failure.*/
    if ((handle != _LLIO_STDOUT) && (handle != _LLIO_STDERR))
    {
        return _LLIO_ERROR;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return _LLIO_ERROR;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buffer, size);
    return size;
}

#pragma weak __read
size_t __read(int handle, unsigned char * buffer, size_t size)
{
    /* This function only reads from "standard in", for all other file*/
    /* handles it returns failure.*/
    if (handle != _LLIO_STDIN)
    {
        return _LLIO_ERROR;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return _LLIO_ERROR;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buffer, size);

    return size;
}

#elif (defined(__GNUC__))
#pragma weak _write
int _write (int handle, char *buffer, int size)
{
    if (buffer == 0)
    {
        /* return -1 if error */
        return -1;
    }

    /* This function only writes to "standard out" and "standard err",*/
    /* for all other file handles it returns failure.*/
    if ((handle != 1) && (handle != 2))
    {
        return -1;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
```

```
399
400     /* Send data.*/
401     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t *)buffer, size);
402     return size;
403 }
404
405 #pragma weak _read
406 int _read(int handle, char *buffer, int size)
407 {
408     /* This function only reads from "standard in", for all other file*/
409     /* handles it returns failure.*/
410     if (handle != 0)
411     {
412         return -1;
413     }
414
415     /* Do nothing if the debug uart is not initialized.*/
416     if (s_debugConsole.type == kDebugConsoleNone)
417     {
418         return -1;
419     }
420
421     /* Receive data.*/
422     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, (uint8_t *)buffer, size);
423     return size;
424 }
425 #elif defined(__CC_ARM) && !defined(MQX_STDIO)
426 struct __FILE
427 {
428     int handle;
429     /* Whatever you require here. If the only file you are using is */
430     /* standard output using printf() for debugging, no file handling */
431     /* is required. */
432 };
433
434 /* FILE is typedef in stdio.h. */
435 #pragma weak __stdout
436 FILE __stdout;
437 FILE __stdin;
438
439 #pragma weak fputc
440 int fputc(int ch, FILE *f)
441 {
442     /* Do nothing if the debug uart is not initialized.*/
443     if (s_debugConsole.type == kDebugConsoleNone)
444     {
445         return -1;
446     }
447
448     /* Send data.*/
449     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (const uint8_t*)&ch, 1);
450     return 1;
451 }
452
453 #pragma weak fgetc
454 int fgetc(FILE *f)
455 {
456     uint8_t temp;
457     /* Do nothing if the debug uart is not initialized.*/
458     if (s_debugConsole.type == kDebugConsoleNone)
459     {
460         return -1;
461     }
462
463     /* Receive data.*/
464     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &temp, 1);
465
```

```
    return temp;
}

#endif

/*************Code for debug_printf/scanf/assert*******************************/
int debug_printf(const char  *fmt_s, ...)
{
    va_list  ap;
    int  result;
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    va_start(ap, fmt_s);
    result = _doprint(NULL, debug_putc, -1, (char *)fmt_s, ap);
    va_end(ap);

    return result;
}

static int debug_putc(int ch, void* stream)
{
    const unsigned char c = (unsigned char) ch;
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, &c, 1);

    return 0;

}

int debug_putchar(int ch)
{
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    debug_putc(ch, NULL);

    return 1;
}

int debug_scanf(const char  *fmt_ptr, ...)
{
    char    temp_buf[IO_MAXLINE];
    va_list ap;
    uint32_t i;
    char result;

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    va_start(ap, fmt_ptr);
    temp_buf[0] = '\0';

    for (i = 0; i < IO_MAXLINE; i++)
    {
        temp_buf[i] = result = debug_getchar();
```

```
533
534         if ((result == '\r') || (result == '\n'))
535         {
536             /* End of Line */
537             if (i == 0)
538             {
539                 i = (uint32_t)-1;
540             }
541             else
542             {
543                 break;
544             }
545         }
546
547         temp_buf[i + 1] = '\0';
548     }
549
550     result = scan_prv(temp_buf, (char *)fmt_ptr, ap);
551     va_end(ap);
552
553   return result;
554 }
555
556 int debug_getchar(void)
557 {
558     unsigned char c;
559
560     /* Do nothing if the debug uart is not initialized.*/
561     if (s_debugConsole.type == kDebugConsoleNone)
562     {
563         return -1;
564     }
565     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &c, 1);
566
567     return c;
568 }
    /*******************************************************************************
     * EOF
     ******************************************************************************/
```