**main.c**

|  |
| --- |
| /////////////////////////////////////////////////////////////////////////////// |
| // Includes |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| // Standard C Included Files |
| #include <string.h> |
| #include <stdio.h> |
| // SDK Included Files |
|  |
| #include "board.h" |
| #include "pin\_mux.h" |
| #include "fsl\_clock\_manager.h" |
|  |
| #include "fsl\_debug\_console.h" |
| //#include "adc\_hw\_trigger.h" |
| //#include "fsl\_adc16\_driver.h" |
|  |
| #include "Custom\_Main.h" |
| #include "Custom\_Circular\_Buffer.h" |
| #include "Custom\_UART.h" |
| #include "Custom\_ASCII\_Counter.h" |
|  |
| UART0\_Operation\_Type State; |
|  |
| char UART\_print[50]; |
|  |
| DWord Fib\_n, Fib\_1 = 0, Fib\_2 = 1; |
| Byte led = 0; |
|  |
| #ifdef FRDM |
| void hardware\_init(void); |
|  |
| //FGETS using custom UART function |
| void FGETS(char \*array\_to\_write, Byte bytes, FILE \*stream) |
| { |
| //Cleanup and assign FGETS buffer (of fixed length) |
| CBuffer\_Assign(FGETS\_Buffer\_ID); |
|  |
| //Set the proper state for ISR |
| State = FGETS\_Operation; |
|  |
| //Don't proceed until enter is pressed (or length is reached) |
| while(State == FGETS\_Operation); |
|  |
| if(CBuffer\_Instance[FGETS\_Buffer\_ID].Status == Full) Output\_String("\n\rOverwriting"); |
|  |
| //Form a proper string/array |
| char \*tmp; |
| tmp = array\_to\_write; |
| do{ |
| if(CBuffer\_Byte\_Read(FGETS\_Buffer\_ID, array\_to\_write)) break; |
| if(\*array\_to\_write == Enter\_Detected) break; |
| array\_to\_write += 1; |
| }while(array\_to\_write < (tmp + bytes)); |
| } |
| #endif |
|  |
| int main (void) |
| { |
| //Init hardware |
| hardware\_init(); |
|  |
| //Fun |
| PORTB->PCR[18] = PORT\_PCR\_MUX(0x01); |
| PTB->PDDR |= (1 << 18); |
| PTB->PDOR |= (1 << 18); |
|  |
| PORTB->PCR[19] = PORT\_PCR\_MUX(0x01); |
| PTB->PDDR |= (1 << 19); |
| PTB->PDOR |= (1 << 19); |
|  |
| PORTD->PCR[1] = PORT\_PCR\_MUX(0x01); |
| PTD->PDDR |= (1 << 1); |
| PTD->PDOR |= (1 << 1); |
|  |
| //UART and ASCII counter (application) Init |
| Custom\_UART0\_Init(); |
| ASCII\_Counter\_Init(); |
|  |
| //If polling mode then just echo the received characters |
| #ifdef POLLING\_MODE |
| Output\_String("\n\rPolling Mode\n\r"); |
|  |
| while(1) |
| { |
| Custom\_UART0\_Rx\_Byte(&test); |
| Custom\_UART0\_Tx\_Byte(test); |
| } |
| #else |
| Output\_String("\n\rInterrupt Mode\n\r"); |
| #if APPLICATION |
| Output\_String("\n\rApplication Running\n\r"); |
| if(CBuffer\_Init()) Output\_String("\n\rError in Buffer Init\n\r"); |
| else Output\_String("\n\rBuffer Init Success\n\r\n\r"); |
| while(1) |
| { |
| switch(led) |
| { |
| case 0: |
| PTD->PDOR |= (1 << 1); |
| PTB->PDOR |= (1 << 18); |
| PTB->PDOR |= (1 << 19); |
| break; |
| case 1: |
| PTD->PDOR &= ~(1 << 1); |
| PTB->PDOR |= (1 << 18); |
| PTB->PDOR |= (1 << 19); |
| break; |
| case 2: |
| PTD->PDOR |= (1 << 1); |
| PTB->PDOR &= ~(1 << 18); |
| PTB->PDOR |= (1 << 19); |
| break; |
| case 3: |
| PTD->PDOR |= (1 << 1); |
| PTB->PDOR |= (1 << 18); |
| PTB->PDOR &= ~(1 << 19); |
| break; |
| case 4: |
| PTD->PDOR &= ~(1 << 1); |
| PTB->PDOR &= ~(1 << 18); |
| PTB->PDOR |= (1 << 19); |
| break; |
| case 5: |
| PTD->PDOR &= ~(1 << 1); |
| PTB->PDOR |= (1 << 18); |
| PTB->PDOR &= ~(1 << 19); |
| break; |
| case 6: |
| PTD->PDOR |= (1 << 1); |
| PTB->PDOR &= ~(1 << 18); |
| PTB->PDOR &= ~(1 << 19); |
| break; |
| case 7: |
| PTD->PDOR &= ~(1 << 1); |
| PTB->PDOR &= ~(1 << 18); |
| PTB->PDOR &= ~(1 << 19); |
| break; |
| } |
| Fib\_n = Fib\_1 + Fib\_2; |
| while(CBuffer\_Instance[UART0\_Rx\_Buffer\_ID].Status != Empty) ASCII\_Counter(); |
| if(Fib\_n > 3900000000) |
| { |
| Fib\_2 = 0; |
| Fib\_n = 1; |
| } |
| Fib\_1 = Fib\_2; |
| Fib\_2 = Fib\_n; |
| } |
|  |
| #else |
| Output\_String("\n\rToggling LEDs and Echo\n\r"); |
| while(1) |
| { |
| PTD->PTOR |= (1 << 1); |
| PTB->PTOR |= (1 << 18); |
| PTB->PTOR |= (1 << 19); |
| } |
|  |
| #endif |
|  |
| #endif |
| } |

**lptmr\_trigger.c**

|  |
| --- |
| /////////////////////////////////////////////////////////////////////////////// |
| // Includes |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| // SDK Included Files |
| #include "adc\_hw\_trigger.h" |
| #include "fsl\_lptmr\_driver.h" |
| #include "fsl\_sim\_hal.h" |
| #if defined(KM34Z7\_SERIES) |
| #include "fsl\_xbar\_driver.h" |
| #endif |
| /////////////////////////////////////////////////////////////////////////////// |
| // Variables |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| extern SIM\_Type \* gSimBase[]; |
| static lptmr\_state\_t gLPTMRState; |
|  |
| /////////////////////////////////////////////////////////////////////////////// |
| // Code |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| /\*! |
| \* @Brief enable the trigger source of LPTimer |
| \*/ |
| void init\_trigger\_source(uint32\_t adcInstance) |
| { |
| uint32\_t freqUs; |
|  |
| lptmr\_user\_config\_t lptmrUserConfig = |
| { |
| .timerMode = kLptmrTimerModeTimeCounter, |
| .freeRunningEnable = false, |
| .prescalerEnable = false, // bypass perscaler |
| #if (CLOCK\_INIT\_CONFIG == CLOCK\_VLPR) |
| // use MCGIRCCLK, 4M or 32KHz |
| .prescalerClockSource = kClockLptmrSrcMcgIrClk, |
| #else |
| // Use LPO clock 1KHz |
| .prescalerClockSource = kClockLptmrSrcLpoClk, |
| #endif |
| .isInterruptEnabled = false |
| }; |
|  |
| // Init LPTimer driver |
| LPTMR\_DRV\_Init(0, &gLPTMRState, &lptmrUserConfig); |
|  |
| // Set the LPTimer period |
| freqUs = 1000000U/(INPUT\_SIGNAL\_FREQ\*NR\_SAMPLES)\*2; |
| LPTMR\_DRV\_SetTimerPeriodUs(0, freqUs); |
|  |
| // Start the LPTimer |
| LPTMR\_DRV\_Start(0); |
|  |
| // Configure SIM for ADC hw trigger source selection |
| #if defined(KM34Z7\_SERIES) |
| SIM\_HAL\_EnableClock(gSimBase[0], kSimClockGateXbar0); |
| SIM\_HAL\_SetAdcTrgSelMode(gSimBase[0], kSimAdcTrgSelXbar); |
| XBAR\_DRV\_ConfigSignalConnection(kXbaraInputLPTMR0\_Output, kXbaraOutputADC\_TRGA); |
| #else |
| SIM\_HAL\_SetAdcAlternativeTriggerCmd(gSimBase[0], adcInstance, true); |
| SIM\_HAL\_SetAdcPreTriggerMode(gSimBase[0], adcInstance, kSimAdcPretrgselA); |
| SIM\_HAL\_SetAdcTriggerMode(gSimBase[0], adcInstance, kSimAdcTrgSelLptimer); |
| #endif |
| } |
|  |
| /\*! |
| \* @Brief disable the trigger source |
| \*/ |
| void deinit\_trigger\_source(uint32\_t adcInstance) |
| { |
| LPTMR\_DRV\_Stop(0); |
| LPTMR\_DRV\_Deinit(0); |
| } |

**fsl\_adc\_irq.c**

|  |
| --- |
| /////////////////////////////////////////////////////////////////////////////// |
| // Includes |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| // Standard C Included Files |
| #include <stdint.h> |
| #include <stdbool.h> |
| // SDK Included Files |
| #include "fsl\_adc16\_driver.h" |
|  |
| /////////////////////////////////////////////////////////////////////////////// |
| // Variables |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| // Define array to keep run-time callback set by application. |
| void (\* volatile g\_AdcTestCallback[ADC\_INSTANCE\_COUNT][ADC\_SC1\_COUNT])(void); |
| volatile uint16\_t g\_AdcValueInt[ADC\_INSTANCE\_COUNT][ADC\_SC1\_COUNT]; |
|  |
| /////////////////////////////////////////////////////////////////////////////// |
| // Code |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| /\* User-defined function to install callback. \*/ |
| void ADC\_TEST\_InstallCallback(uint32\_t instance, uint32\_t chnGroup, void (\*callbackFunc)(void) ) |
| { |
| g\_AdcTestCallback[instance][chnGroup] = callbackFunc; |
| } |
|  |
| /\* User-defined function to read conversion value in ADC ISR. \*/ |
| uint16\_t ADC\_TEST\_GetConvValueRAWInt(uint32\_t instance, uint32\_t chnGroup) |
| { |
| return g\_AdcValueInt[instance][chnGroup]; |
| } |
|  |
| /\* User-defined ADC ISR. \*/ |
| static void ADC\_TEST\_IRQHandler(uint32\_t instance) |
| { |
| uint32\_t chnGroup; |
| for (chnGroup = 0U; chnGroup < ADC\_SC1\_COUNT; chnGroup++) |
| { |
| if ( ADC16\_DRV\_GetChnFlag(instance, chnGroup, kAdcChnConvCompleteFlag) ) |
| { |
| g\_AdcValueInt[instance][chnGroup] = ADC16\_DRV\_GetConvValueRAW(instance, chnGroup); |
| if ( g\_AdcTestCallback[instance][chnGroup] ) |
| { |
| (void)(\*(g\_AdcTestCallback[instance][chnGroup]))(); |
| } |
| } |
| } |
| } |
|  |
| /////////////////////////////////////////////////////////////////////////////// |
| // IRQ Handlers |
| /////////////////////////////////////////////////////////////////////////////// |
|  |
| /\* ADC IRQ handler that would cover the same name's APIs in startup code \*/ |
| void ADC0\_IRQHandler(void) |
| { |
| // Add user-defined ISR for ADC0 |
| ADC\_TEST\_IRQHandler(0U); |
| } |
|  |
| #if (ADC\_INSTANCE\_COUNT > 1U) |
| void ADC1\_IRQHandler(void) |
| { |
| // Add user-defined ISR for ADC1 |
| ADC\_TEST\_IRQHandler(1U); |
| } |
| #endif |
|  |
| #if (ADC\_INSTANCE\_COUNT > 2U) |
| void ADC2\_IRQHandler(void) |
| { |
| // Add user-defined ISR for ADC2 |
| ADC\_TEST\_IRQHandler(2U); |
| } |
| #endif |
|  |
| #if (ADC\_INSTANCE\_COUNT > 3U) |
| void ADC3\_IRQHandler(void) |
| { |
| // Add user-defined ISR for ADC3 |
| ADC\_TEST\_IRQHandler(3U); |
| } |
| #endif |

**startup.c**

|  |
| --- |
| #include "startup.h" |
| #include "fsl\_device\_registers.h" |
|  |
| #if (defined(\_\_ICCARM\_\_)) |
| #pragma section = ".data" |
| #pragma section = ".data\_init" |
| #pragma section = ".bss" |
| #endif |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Code |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : init\_data\_bss |
| \* Description : Make necessary initializations for RAM. |
| \* - Copy initialized data from ROM to RAM. |
| \* - Clear the zero-initialized data section. |
| \* - Copy the vector table from ROM to RAM. This could be an option. |
| \* |
| \* Tool Chians: |
| \* \_\_GNUC\_\_ : GCC |
| \* \_\_CC\_ARM : KEIL |
| \* \_\_ICCARM\_\_ : IAR |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| void init\_data\_bss(void) |
| { |
| uint32\_t n; |
|  |
| /\* Addresses for VECTOR\_TABLE and VECTOR\_RAM come from the linker file \*/ |
| #if defined(\_\_CC\_ARM) |
| extern uint32\_t Image$$VECTOR\_ROM$$Base[]; |
| extern uint32\_t Image$$VECTOR\_RAM$$Base[]; |
| extern uint32\_t Image$$RW\_m\_data$$Base[]; |
|  |
| #define \_\_VECTOR\_TABLE Image$$VECTOR\_ROM$$Base |
| #define \_\_VECTOR\_RAM Image$$VECTOR\_RAM$$Base |
| #define \_\_RAM\_VECTOR\_TABLE\_SIZE (((uint32\_t)Image$$RW\_m\_data$$Base - (uint32\_t)Image$$VECTOR\_RAM$$Base)) |
| #elif defined(\_\_ICCARM\_\_) |
| extern uint32\_t \_\_RAM\_VECTOR\_TABLE\_SIZE[]; |
| extern uint32\_t \_\_VECTOR\_TABLE[]; |
| extern uint32\_t \_\_VECTOR\_RAM[]; |
| #elif defined(\_\_GNUC\_\_) |
| extern uint32\_t \_\_VECTOR\_TABLE[]; |
| extern uint32\_t \_\_VECTOR\_RAM[]; |
| extern uint32\_t \_\_RAM\_VECTOR\_TABLE\_SIZE\_BYTES[]; |
| uint32\_t \_\_RAM\_VECTOR\_TABLE\_SIZE = (uint32\_t)(\_\_RAM\_VECTOR\_TABLE\_SIZE\_BYTES); |
| #endif |
|  |
| if (\_\_VECTOR\_RAM != \_\_VECTOR\_TABLE) |
| { |
| /\* Copy the vector table from ROM to RAM \*/ |
| for (n = 0; n < ((uint32\_t)\_\_RAM\_VECTOR\_TABLE\_SIZE)/sizeof(uint32\_t); n++) |
| { |
| \_\_VECTOR\_RAM[n] = \_\_VECTOR\_TABLE[n]; |
| } |
| /\* Point the VTOR to the position of vector table \*/ |
| SCB->VTOR = (uint32\_t)\_\_VECTOR\_RAM; |
| } |
| else |
| { |
| /\* Point the VTOR to the position of vector table \*/ |
| SCB->VTOR = (uint32\_t)\_\_VECTOR\_TABLE; |
| } |
|  |
| #if !defined(\_\_CC\_ARM) && !defined(\_\_ICCARM\_\_) |
|  |
| /\* Declare pointers for various data sections. These pointers |
| \* are initialized using values pulled in from the linker file \*/ |
| uint8\_t \* data\_ram, \* data\_rom, \* data\_rom\_end; |
| uint8\_t \* bss\_start, \* bss\_end; |
|  |
| /\* Get the addresses for the .data section (initialized data section) \*/ |
| #if defined(\_\_GNUC\_\_) |
| extern uint32\_t \_\_DATA\_ROM[]; |
| extern uint32\_t \_\_DATA\_RAM[]; |
| extern char \_\_DATA\_END[]; |
| data\_ram = (uint8\_t \*)\_\_DATA\_RAM; |
| data\_rom = (uint8\_t \*)\_\_DATA\_ROM; |
| data\_rom\_end = (uint8\_t \*)\_\_DATA\_END; |
| n = data\_rom\_end - data\_rom; |
| #endif |
|  |
| /\* Copy initialized data from ROM to RAM \*/ |
| while (n--) |
| { |
| \*data\_ram++ = \*data\_rom++; |
| } |
|  |
| /\* Get the addresses for the .bss section (zero-initialized data) \*/ |
| #if defined(\_\_GNUC\_\_) |
| extern char \_\_START\_BSS[]; |
| extern char \_\_END\_BSS[]; |
| bss\_start = (uint8\_t \*)\_\_START\_BSS; |
| bss\_end = (uint8\_t \*)\_\_END\_BSS; |
| #endif |
|  |
| /\* Clear the zero-initialized data section \*/ |
| n = bss\_end - bss\_start; |
| while(n--) |
| { |
| \*bss\_start++ = 0; |
| } |
| #endif /\* !\_\_CC\_ARM && !\_\_ICCARM\_\_\*/ |
| } |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* EOF |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**startup.h**

|  |
| --- |
| #ifndef \_STARTUP\_H\_ |
| #define \_STARTUP\_H\_ |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* API |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| /\*! |
| \* @brief Make necessary initializations for RAM. |
| \* |
| \* - Copy initialized data from ROM to RAM. |
| \* - Clear the zero-initialized data section. |
| \* - Copy the vector table from ROM to RAM. This could be an option. |
| \*/ |
| void init\_data\_bss(void); |
|  |
| #endif /\* \_STARTUP\_H\_\*/ |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* EOF |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**fsl\_debug\_console.c**

|  |
| --- |
| #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 |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| /\*! @brief Operation functions definiations for debug console. \*/ |
| typedef struct DebugConsoleOperationFunctions { |
| union { |
| void (\* Send)(void \*base, const uint8\_t \*buf, uint32\_t count); |
| #if defined(UART\_INSTANCE\_COUNT) |
| void (\* UART\_Send)(UART\_Type \*base, const uint8\_t \*buf, uint32\_t count); |
| #endif |
| #if defined(LPUART\_INSTANCE\_COUNT) |
| void (\* LPUART\_Send)(LPUART\_Type\* base, const uint8\_t \*buf, uint32\_t count); |
| #endif |
| #if defined(UART0\_INSTANCE\_COUNT) |
| void (\* UART0\_Send)(UART0\_Type\* base, const uint8\_t \*buf, uint32\_t count); |
| #endif |
| #if (defined(USB\_INSTANCE\_COUNT) && defined(BOARD\_USE\_VIRTUALCOM)) |
| void (\* USB\_Send)(uint32\_t base, const uint8\_t \*buf, uint32\_t count); |
| #endif |
| } tx\_union; |
| union{ |
| void (\* Receive)(void \*base, uint8\_t \*buf, uint32\_t count); |
| #if defined(UART\_INSTANCE\_COUNT) |
| uart\_status\_t (\* UART\_Receive)(UART\_Type \*base, uint8\_t \*buf, uint32\_t count); |
| #endif |
| #if defined(LPUART\_INSTANCE\_COUNT) |
| lpuart\_status\_t (\* LPUART\_Receive)(LPUART\_Type\* base, uint8\_t \*buf, uint32\_t count); |
| #endif |
| #if defined(UART0\_INSTANCE\_COUNT) |
| lpsci\_status\_t (\* UART0\_Receive)(UART0\_Type\* base, uint8\_t \*buf, uint32\_t count); |
| #endif |
| #if (defined(USB\_INSTANCE\_COUNT) && defined(BOARD\_USE\_VIRTUALCOM)) |
| usb\_status\_t (\* USB\_Receive)(uint32\_t base, uint8\_t \*buf, uint32\_t count); |
| #endif |
|  |
| } rx\_union; |
| } debug\_console\_ops\_t; |
|  |
| /\*! @brief State structure storing debug console. \*/ |
| typedef struct DebugConsoleState { |
| debug\_console\_device\_type\_t type;/\*<! Indicator telling whether the debug console is inited. \*/ |
| uint8\_t instance; /\*<! Instance number indicator. \*/ |
| void\* base; /\*<! Base of the IP register. \*/ |
| debug\_console\_ops\_t ops; /\*<! Operation function pointers for debug uart operations. \*/ |
| } debug\_console\_state\_t; |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Variables |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| /\*! @brief Debug UART state information.\*/ |
| static debug\_console\_state\_t s\_debugConsole; |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Code |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| /\* See fsl\_debug\_console.h for documentation of this function.\*/ |
| debug\_console\_status\_t DbgConsole\_Init( |
| uint32\_t uartInstance, uint32\_t baudRate, debug\_console\_device\_type\_t device) |
| { |
| if (s\_debugConsole.type != kDebugConsoleNone) |
| { |
| return kStatus\_DEBUGCONSOLE\_Failed; |
| } |
|  |
| /\* Set debug console to initialized to avoid duplicated init operation.\*/ |
| s\_debugConsole.type = device; |
| s\_debugConsole.instance = uartInstance; |
|  |
| /\* 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); |
| LPSCI\_HAL\_EnableReceiver(base); |
|  |
| /\* Set the funciton pointer for send and receive for this kind of device. \*/ |
| s\_debugConsole.ops.tx\_union.UART0\_Send = LPSCI\_HAL\_SendDataPolling; |
| s\_debugConsole.ops.rx\_union.UART0\_Receive = LPSCI\_HAL\_ReceiveDataPolling; |
| } |
| break; |
| #endif |
| #if defined(LPUART\_INSTANCE\_COUNT) |
| case kDebugConsoleLPUART: |
| { |
| LPUART\_Type\* g\_Base[LPUART\_INSTANCE\_COUNT] = LPUART\_BASE\_PTRS; |
| LPUART\_Type\* base = g\_Base[uartInstance]; |
| uint32\_t lpuartSourceClock; |
|  |
| s\_debugConsole.base = base; |
| CLOCK\_SYS\_EnableLpuartClock(uartInstance); |
|  |
| /\* LPUART clock source is either system or bus clock depending on instance \*/ |
| lpuartSourceClock = CLOCK\_SYS\_GetLpuartFreq(uartInstance); |
|  |
| /\* initialize the parameters of the LPUART config structure with desired data \*/ |
| LPUART\_HAL\_SetBaudRate(base, lpuartSourceClock, baudRate); |
| LPUART\_HAL\_SetBitCountPerChar(base, kLpuart8BitsPerChar); |
| LPUART\_HAL\_SetParityMode(base, kLpuartParityDisabled); |
| LPUART\_HAL\_SetStopBitCount(base, kLpuartOneStopBit); |
|  |
| /\* finally, enable the LPUART transmitter and receiver \*/ |
| LPUART\_HAL\_SetTransmitterCmd(base, true); |
| LPUART\_HAL\_SetReceiverCmd(base, true); |
|  |
| /\* Set the funciton pointer for send and receive for this kind of device. \*/ |
| s\_debugConsole.ops.tx\_union.LPUART\_Send = LPUART\_HAL\_SendDataPolling; |
| s\_debugConsole.ops.rx\_union.LPUART\_Receive = LPUART\_HAL\_ReceiveDataPolling; |
|  |
| } |
| break; |
| #endif |
| /\* If new device is requried as the low level device for debug console, |
| \* Add the case branch and add the preprocessor macro to judge whether |
| \* this kind of device exist in this SOC. \*/ |
| default: |
| /\* Device identified is invalid, return invalid device error code. \*/ |
| return kStatus\_DEBUGCONSOLE\_InvalidDevice; |
| } |
|  |
| /\* Configure the s\_debugConsole structure only when the inti operation is successful. \*/ |
| s\_debugConsole.instance = uartInstance; |
|  |
| return kStatus\_DEBUGCONSOLE\_Success; |
| } |
|  |
| /\* See fsl\_debug\_console.h for documentation of this function.\*/ |
| debug\_console\_status\_t DbgConsole\_DeInit(void) |
| { |
| if (s\_debugConsole.type == kDebugConsoleNone) |
| { |
| return kStatus\_DEBUGCONSOLE\_Success; |
| } |
|  |
| switch(s\_debugConsole.type) |
| { |
| #if defined(UART\_INSTANCE\_COUNT) |
| case kDebugConsoleUART: |
| CLOCK\_SYS\_DisableUartClock(s\_debugConsole.instance); |
| break; |
| #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 |
| #if (defined(USB\_INSTANCE\_COUNT) && defined(BOARD\_USE\_VIRTUALCOM)) |
| case kDebugConsoleUSBCDC: |
| VirtualCom\_Deinit(); |
| CLOCK\_SYS\_DisableUsbfsClock(0); |
| 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; |
| } |
|  |
| /\* Send data.\*/ |
| s\_debugConsole.ops.tx\_union.Send(s\_debugConsole.base, (uint8\_t \*)buffer, size); |
| return size; |
| } |
|  |
| #pragma weak \_read |
| int \_read(int handle, char \*buffer, int size) |
| { |
| /\* This function only reads from "standard in", for all other file\*/ |
| /\* handles it returns failure.\*/ |
| if (handle != 0) |
| { |
| return -1; |
| } |
|  |
| /\* 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, (uint8\_t \*)buffer, size); |
| return size; |
| } |
| #elif defined(\_\_CC\_ARM) && !defined(MQX\_STDIO) |
| struct \_\_FILE |
| { |
| int handle; |
| /\* Whatever you require here. If the only file you are using is \*/ |
| /\* standard output using printf() for debugging, no file handling \*/ |
| /\* is required. \*/ |
| }; |
|  |
| /\* FILE is typedef in stdio.h. \*/ |
| #pragma weak \_\_stdout |
| FILE \_\_stdout; |
| FILE \_\_stdin; |
|  |
| #pragma weak fputc |
| int fputc(int ch, FILE \*f) |
| { |
| /\* 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, (const uint8\_t\*)&ch, 1); |
| return 1; |
| } |
|  |
| #pragma weak fgetc |
| int fgetc(FILE \*f) |
| { |
| uint8\_t temp; |
| /\* 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, &temp, 1); |
| 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(); |
|  |
| if ((result == '\r') || (result == '\n')) |
| { |
| /\* End of Line \*/ |
| if (i == 0) |
| { |
| i = (uint32\_t)-1; |
| } |
| else |
| { |
| break; |
| } |
| } |
|  |
| temp\_buf[i + 1] = '\0'; |
| } |
|  |
| result = scan\_prv(temp\_buf, (char \*)fmt\_ptr, ap); |
| va\_end(ap); |
|  |
| return result; |
| } |
|  |
| int debug\_getchar(void) |
| { |
| unsigned char c; |
|  |
| /\* Do nothing if the debug uart is not initialized.\*/ |
| if (s\_debugConsole.type == kDebugConsoleNone) |
| { |
| return -1; |
| } |
| s\_debugConsole.ops.rx\_union.Receive(s\_debugConsole.base, &c, 1); |
|  |
| return c; |
| } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* EOF |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**Fsl\_debug\_console.h**

|  |
| --- |
| #if !defined(\_\_FSL\_DEBUG\_CONSOLE\_H\_\_) |
| #define \_\_FSL\_DEBUG\_CONSOLE\_H\_\_ |
|  |
| #include <stdint.h> |
| #include "fsl\_os\_abstraction.h" |
|  |
| /\* |
| \* @addtogroup debug\_console |
| \* @{ |
| \*/ |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Definitions |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| #define IO\_MAXLINE 20 |
|  |
| #if (defined (FSL\_RTOS\_MQX) && (MQX\_COMMON\_CONFIG != MQX\_LITE\_CONFIG)) |
| #define PRINTF printf |
| #define SCANF scanf |
| #define PUTCHAR putchar |
| #define GETCHAR getchar |
| #include <stdio.h> |
| #else |
| /\*Configuration for toolchain's printf/scanf or KSDK version printf/scanf \*/ |
| #define PRINTF debug\_printf |
| //#define PRINTF printf |
| #define SCANF debug\_scanf |
| //#define SCANF scanf |
| #define PUTCHAR debug\_putchar |
| //#define PUTCHAR putchar |
| #define GETCHAR debug\_getchar |
| //#define GETCHAR getchar |
| #endif |
|  |
| /\*! @brief Error code for the debug console driver. \*/ |
| typedef enum \_debug\_console\_status { |
| kStatus\_DEBUGCONSOLE\_Success = 0U, |
| kStatus\_DEBUGCONSOLE\_InvalidDevice, |
| kStatus\_DEBUGCONSOLE\_AllocateMemoryFailed, |
| kStatus\_DEBUGCONSOLE\_Failed |
| } debug\_console\_status\_t; |
|  |
| /\*! @brief Supported debug console hardware device type. \*/ |
| typedef enum \_debug\_console\_device\_type { |
| kDebugConsoleNone = 0U, |
| kDebugConsoleLPSCI = 15U, /\*<! Use strange start number to avoid treating 0 |
| as correct device type. Sometimes user forget |
| to specify the device type but only use the |
| default value '0' as the device type. \*/ |
| kDebugConsoleUART = 16U, |
| kDebugConsoleLPUART = 17U, |
| kDebugConsoleUSBCDC = 18U |
| } debug\_console\_device\_type\_t; |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* API |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| #if defined(\_\_cplusplus) |
| extern "C" { |
| #endif |
|  |
| /\*! @name Initialization\*/ |
| /\*@{\*/ |
|  |
| /\*! |
| \* @brief Init the UART/LPUART used for debug messages. |
| \* |
| \* Call this function to enable debug log messages to be output via the specified UART/LPUART |
| \* base address and at the specified baud rate. Just initializes the UART/LPUART to the given baud |
| \* rate and 8N1. After this function has returned, stdout and stdin will be connected to the |
| \* selected UART/LPUART. The debug\_printf() function also uses this UART/LPUART. |
| \* |
| \* @param uartInstance Which UART/LPUART instance is used to send debug messages. |
| \* @param baudRate The desired baud rate in bits per second. |
| \* @param device Low level device type for the debug console. |
| \* @return Whether initialization was successful or not. |
| \*/ |
| debug\_console\_status\_t DbgConsole\_Init( |
| uint32\_t uartInstance, uint32\_t baudRate, debug\_console\_device\_type\_t device); |
|  |
| /\*! |
| \* @brief Deinit the UART/LPUART used for debug messages. |
| \* |
| \* Call this function to disable debug log messages to be output via the specified UART/LPUART |
| \* base address and at the specified baud rate. |
| \* @return Whether de-initialization was successful or not. |
| \*/ |
| debug\_console\_status\_t DbgConsole\_DeInit(void); |
|  |
| /\*! |
| \* @brief Prints formatted output to the standard output stream. |
| \* |
| \* Call this function to print formatted output to the standard output stream. |
| \* |
| \* @param fmt\_s Format control string. |
| \* @return Returns the number of characters printed, or a negative value if an error occurs. |
| \*/ |
| int debug\_printf(const char \*fmt\_s, ...); |
|  |
| /\*! |
| \* @brief Writes a character to stdout. |
| \* |
| \* Call this function to write a character to stdout. |
| \* |
| \* @param ch Character to be written. |
| \* @return Returns the character written. |
| \*/ |
| int debug\_putchar(int ch); |
|  |
| /\*! |
| \* @brief Reads formatted data from the standard input stream. |
| \* |
| \* Call this function to read formatted data from the standard input stream. |
| \* |
| \* @param fmt\_ptr Format control string. |
| \* @return Returns the number of fields successfully converted and assigned. |
| \*/ |
| int debug\_scanf(const char \*fmt\_ptr, ...); |
|  |
| /\*! |
| \* @brief Reads a character from standard input. |
| \* |
| \* Call this function to read a character from standard input. |
| \* |
| \* @return Returns the character read. |
| \*/ |
| int debug\_getchar(void); |
|  |
| /\*@}\*/ |
|  |
| #if defined(\_\_cplusplus) |
| } |
| #endif |
|  |
| /\*! @}\*/ |
|  |
| #endif /\* \_\_FSL\_DEBUG\_CONSOLE\_H\_\_\*/ |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* EOF |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**fsl\_misc\_utilities.c**

|  |
| --- |
| #include <stdarg.h> |
| #include <stdio.h> |
| #include <stdlib.h> |
| #include "fsl\_misc\_utilities.h" |
| #if defined(\_\_GNUC\_\_) |
| #include <errno.h> |
| #endif |
| #include "fsl\_debug\_console.h" |
|  |
| #if (defined(\_\_CC\_ARM)) |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : \_\_aeabi\_assert |
| \* Description : called by assert in KEIL |
| \* This function is called by the assert function in KEIL. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| void \_\_aeabi\_assert(const char \*expr, const char \*file, int line) |
| { |
| printf("assert failed:%s, file %s:%d\r\n",expr,file,line); |
| } |
|  |
| #endif |
|  |
| #if defined(\_\_GNUC\_\_) |
| caddr\_t |
| \_sbrk (int incr) |
| { |
| extern char end \_\_asm ("end"); |
| extern char heap\_limit \_\_asm ("\_\_HeapLimit"); |
| static char \* heap\_end; |
| char \* prev\_heap\_end; |
|  |
| if (heap\_end == NULL) |
| heap\_end = & end; |
|  |
| prev\_heap\_end = heap\_end; |
|  |
| if (heap\_end + incr > &heap\_limit) |
| { |
| #ifdef NIO\_ENOMEM //TODO: Update NIO error code for MQX |
| errno = NIO\_ENOMEM; |
| #else |
| errno = ENOMEM; |
| #endif |
| return (caddr\_t) -1; |
| } |
|  |
| heap\_end += incr; |
|  |
| return (caddr\_t) prev\_heap\_end; |
| } |
| #endif |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : assert\_func |
| \* Description : Print out failure messages. |
| \* This function is used to print out failure messages. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| void assert\_func(const char \*file, int line, const char \*func, const char \*failedExpr) |
| { |
| PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file , line, func); |
|  |
| for (;;) |
| {} |
|  |
| } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* EOF |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**print\_scan.c**

|  |
| --- |
| #include "print\_scan.h" |
| #include <stdio.h> |
| #include <stdlib.h> |
| #include <ctype.h> |
| #include <stdint.h> |
| #include <stdbool.h> |
| // Keil: suppress ellipsis warning in va\_arg usage below |
| #if defined(\_\_CC\_ARM) |
| #pragma diag\_suppress 1256 |
| #endif |
|  |
| #define FLAGS\_MINUS (0x01) |
| #define FLAGS\_PLUS (0x02) |
| #define FLAGS\_SPACE (0x04) |
| #define FLAGS\_ZERO (0x08) |
| #define FLAGS\_POUND (0x10) |
|  |
| #define IS\_FLAG\_MINUS(a) (a & FLAGS\_MINUS) |
| #define IS\_FLAG\_PLUS(a) (a & FLAGS\_PLUS) |
| #define IS\_FLAG\_SPACE(a) (a & FLAGS\_SPACE) |
| #define IS\_FLAG\_ZERO(a) (a & FLAGS\_ZERO) |
| #define IS\_FLAG\_POUND(a) (a & FLAGS\_POUND) |
|  |
| #define LENMOD\_h (0x01) |
| #define LENMOD\_l (0x02) |
| #define LENMOD\_L (0x04) |
| #define LENMOD\_hh (0x08) |
| #define LENMOD\_ll (0x10) |
|  |
| #define IS\_LENMOD\_h(a) (a & LENMOD\_h) |
| #define IS\_LENMOD\_hh(a) (a & LENMOD\_hh) |
| #define IS\_LENMOD\_l(a) (a & LENMOD\_l) |
| #define IS\_LENMOD\_ll(a) (a & LENMOD\_ll) |
| #define IS\_LENMOD\_L(a) (a & LENMOD\_L) |
|  |
| #define SCAN\_SUPPRESS 0x2 |
|  |
| #define SCAN\_DEST\_MASK 0x7c |
| #define SCAN\_DEST\_CHAR 0x4 |
| #define SCAN\_DEST\_STRING 0x8 |
| #define SCAN\_DEST\_SET 0x10 |
| #define SCAN\_DEST\_INT 0x20 |
| #define SCAN\_DEST\_FLOAT 0x30 |
|  |
| #define SCAN\_LENGTH\_MASK 0x1f00 |
| #define SCAN\_LENGTH\_CHAR 0x100 |
| #define SCAN\_LENGTH\_SHORT\_INT 0x200 |
| #define SCAN\_LENGTH\_LONG\_INT 0x400 |
| #define SCAN\_LENGTH\_LONG\_LONG\_INT 0x800 |
| #define SCAN\_LENGTH\_LONG\_DOUBLE 0x1000 |
|  |
| #define SCAN\_TYPE\_SIGNED 0x2000 |
|  |
| /\*! |
| \* @brief Scanline function which ignores white spaces. |
| \* |
| \* @param[in] s The address of the string pointer to update. |
| \* |
| \* @return String without white spaces. |
| \*/ |
| static uint32\_t scan\_ignore\_white\_space(const char \*\*s); |
|  |
| #if defined(SCANF\_FLOAT\_ENABLE) |
| static double fnum = 0.0; |
| #endif |
|  |
| /\*! |
| \* @brief Converts a radix number to a string and return its length. |
| \* |
| \* @param[in] numstr Converted string of the number. |
| \* @param[in] nump Pointer to the number. |
| \* @param[in] neg Polarity of the number. |
| \* @param[in] radix The radix to be converted to. |
| \* @param[in] use\_caps Used to identify %x/X output format. |
|  |
| \* @return Length of the converted string. |
| \*/ |
| static int32\_t mknumstr (char \*numstr, void \*nump, int32\_t neg, int32\_t radix, bool use\_caps); |
|  |
| #if defined(PRINTF\_FLOAT\_ENABLE) |
| /\*! |
| \* @brief Converts a floating radix number to a string and return its length. |
| \* |
| \* @param[in] numstr Converted string of the number. |
| \* @param[in] nump Pointer to the number. |
| \* @param[in] radix The radix to be converted to. |
| \* @param[in] precision\_width Specify the precision width. |
|  |
| \* @return Length of the converted string. |
| \*/ |
| static int32\_t mkfloatnumstr (char \*numstr, void \*nump, int32\_t radix, uint32\_t precision\_width); |
| #endif |
|  |
|  |
| static void fput\_pad(int32\_t c, int32\_t curlen, int32\_t field\_width, int32\_t \*count, PUTCHAR\_FUNC func\_ptr, void \*farg, int \*max\_count); |
|  |
| double modf(double input\_dbl, double \*intpart\_ptr); |
|  |
| #if !defined(PRINT\_MAX\_COUNT) |
| #define n\_putchar(func, chacter, p, count) func(chacter, p) |
| #else |
| static int n\_putchar(PUTCHAR\_FUNC func\_ptr, int chacter, void \*p, int \*max\_count) |
| { |
| int result = 0; |
| if (\*max\_count) |
| { |
| result = func\_ptr(chacter, p); |
| (\*max\_count)--; |
| } |
| return result; |
| } |
| #endif |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : \_doprint |
| \* Description : This function outputs its parameters according to a |
| \* formatted string. I/O is performed by calling given function pointer |
| \* using following (\*func\_ptr)(c,farg); |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| int \_doprint(void \*farg, PUTCHAR\_FUNC func\_ptr, int max\_count, char \*fmt, va\_list ap) |
| { |
| /\* va\_list ap; \*/ |
| char \*p; |
| int32\_t c; |
|  |
| char vstr[33]; |
| char \*vstrp; |
| int32\_t vlen; |
|  |
| int32\_t done; |
| int32\_t count = 0; |
| int temp\_count = max\_count; |
|  |
|  |
| uint32\_t flags\_used; |
| uint32\_t field\_width; |
|  |
| int32\_t ival; |
| int32\_t schar, dschar; |
| int32\_t \*ivalp; |
| char \*sval; |
| int32\_t cval; |
| uint32\_t uval; |
| bool use\_caps; |
| uint32\_t precision\_width; |
| //uint32\_t length\_modifier = 0; |
| #if defined(PRINTF\_FLOAT\_ENABLE) |
| double fval; |
| #endif |
|  |
| if (max\_count == -1) |
| { |
| max\_count = INT32\_MAX - 1; |
| } |
|  |
| /\* |
| \* Start parsing apart the format string and display appropriate |
| \* formats and data. |
| \*/ |
| for (p = (char \*)fmt; (c = \*p) != 0; p++) |
| { |
| /\* |
| \* All formats begin with a '%' marker. Special chars like |
| \* '\n' or '\t' are normally converted to the appropriate |
| \* character by the \_\_compiler\_\_. Thus, no need for this |
| \* routine to account for the '\' character. |
| \*/ |
| if (c != '%') |
| { |
| n\_putchar(func\_ptr, c, farg, &max\_count); |
|  |
| count++; |
|  |
| /\* |
| \* By using 'continue', the next iteration of the loop |
| \* is used, skipping the code that follows. |
| \*/ |
| continue; |
| } |
|  |
| /\* |
| \* First check for specification modifier flags. |
| \*/ |
| use\_caps = true; |
| flags\_used = 0; |
| done = false; |
| while (!done) |
| { |
| switch (/\* c = \*/ \*++p) |
| { |
| case '-': |
| flags\_used |= FLAGS\_MINUS; |
| break; |
| case '+': |
| flags\_used |= FLAGS\_PLUS; |
| break; |
| case ' ': |
| flags\_used |= FLAGS\_SPACE; |
| break; |
| case '0': |
| flags\_used |= FLAGS\_ZERO; |
| break; |
| case '#': |
| flags\_used |= FLAGS\_POUND; |
| break; |
| default: |
| /\* we've gone one char too far \*/ |
| --p; |
| done = true; |
| break; |
| } |
| } |
|  |
| /\* |
| \* Next check for minimum field width. |
| \*/ |
| field\_width = 0; |
| done = false; |
| while (!done) |
| { |
| switch (c = \*++p) |
| { |
| case '0': |
| case '1': |
| case '2': |
| case '3': |
| case '4': |
| case '5': |
| case '6': |
| case '7': |
| case '8': |
| case '9': |
| field\_width = (field\_width \* 10) + (c - '0'); |
| break; |
| default: |
| /\* we've gone one char too far \*/ |
| --p; |
| done = true; |
| break; |
| } |
| } |
|  |
| /\* |
| \* Next check for the width and precision field separator. |
| \*/ |
| precision\_width = 6; |
| if (/\* (c = \*++p) \*/ \*++p == '.') |
| { |
| /\* precision\_used = true; \*/ |
|  |
| /\* |
| \* Must get precision field width, if present. |
| \*/ |
| precision\_width = 0; |
| done = false; |
| while (!done) |
| { |
| switch (c = \*++p) |
| { |
| case '0': |
| case '1': |
| case '2': |
| case '3': |
| case '4': |
| case '5': |
| case '6': |
| case '7': |
| case '8': |
| case '9': |
| precision\_width = (precision\_width \* 10) + (c - '0'); |
| break; |
| default: |
| /\* we've gone one char too far \*/ |
| --p; |
| done = true; |
| break; |
| } |
| } |
| } |
| else |
| { |
| /\* we've gone one char too far \*/ |
| --p; |
| } |
|  |
| /\* |
| \* Check for the length modifier. |
| \*/ |
| /\* length\_modifier = 0; \*/ |
| switch (/\* c = \*/ \*++p) |
| { |
| case 'h': |
| if (\*++p != 'h') |
| { |
| --p; |
| } |
| /\* length\_modifier |= LENMOD\_h; \*/ |
| break; |
| case 'l': |
| if (\*++p != 'l') |
| { |
| --p; |
| } |
| /\* length\_modifier |= LENMOD\_l; \*/ |
| break; |
| case 'L': |
| /\* length\_modifier |= LENMOD\_L; \*/ |
| break; |
| default: |
| /\* we've gone one char too far \*/ |
| --p; |
| break; |
| } |
|  |
| /\* |
| \* Now we're ready to examine the format. |
| \*/ |
| switch (c = \*++p) |
| { |
| case 'd': |
| case 'i': |
| ival = (int32\_t)va\_arg(ap, int32\_t); |
| vlen = mknumstr(vstr,&ival,true,10,use\_caps); |
| vstrp = &vstr[vlen]; |
|  |
| if (ival < 0) |
| { |
| schar = '-'; |
| ++vlen; |
| } |
| else |
| { |
| if (IS\_FLAG\_PLUS(flags\_used)) |
| { |
| schar = '+'; |
| ++vlen; |
| } |
| else |
| { |
| if (IS\_FLAG\_SPACE(flags\_used)) |
| { |
| schar = ' '; |
| ++vlen; |
| } |
| else |
| { |
| schar = 0; |
| } |
| } |
| } |
| dschar = false; |
|  |
| /\* |
| \* do the ZERO pad. |
| \*/ |
| if (IS\_FLAG\_ZERO(flags\_used)) |
| { |
| if (schar) |
| { |
| n\_putchar(func\_ptr, schar, farg, &max\_count); |
| count++; |
| } |
| dschar = true; |
|  |
| fput\_pad('0', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| vlen = field\_width; |
| } |
| else |
| { |
| if (!IS\_FLAG\_MINUS(flags\_used)) |
| { |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| if (schar) |
| { |
| n\_putchar(func\_ptr, schar, farg, &max\_count); |
| count++; |
| } |
| dschar = true; |
| } |
| } |
|  |
| /\* the string was built in reverse order, now display in \*/ |
| /\* correct order \*/ |
| if ((!dschar) && schar) |
| { |
| n\_putchar(func\_ptr, schar, farg, &max\_count); |
| count++; |
| } |
| goto cont\_xd; |
| #if defined(PRINTF\_FLOAT\_ENABLE) |
| case 'f': |
| case 'F': |
| fval = (double)va\_arg(ap, double); |
| vlen = mkfloatnumstr(vstr,&fval,10, precision\_width); |
| vstrp = &vstr[vlen]; |
|  |
| if (fval < 0) |
| { |
| schar = '-'; |
| ++vlen; |
| } |
| else |
| { |
| if (IS\_FLAG\_PLUS(flags\_used)) |
| { |
| schar = '+'; |
| ++vlen; |
| } |
| else |
| { |
| if (IS\_FLAG\_SPACE(flags\_used)) |
| { |
| schar = ' '; |
| ++vlen; |
| } |
| else |
| { |
| schar = 0; |
| } |
| } |
| } |
| dschar = false; |
| if (IS\_FLAG\_ZERO(flags\_used)) |
| { |
| if (schar) |
| { |
| n\_putchar(func\_ptr, schar, farg, &max\_count); |
| count++; |
| } |
| dschar = true; |
| fput\_pad('0', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| vlen = field\_width; |
| } |
| else |
| { |
| if (!IS\_FLAG\_MINUS(flags\_used)) |
| { |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| if (schar) |
| { |
| n\_putchar(func\_ptr, schar, farg, &max\_count); |
| count++; |
| } |
| dschar = true; |
| } |
| } |
| if (!dschar && schar) |
| { |
| n\_putchar(func\_ptr, schar, farg, &max\_count); |
| count++; |
| } |
| goto cont\_xd; |
| #endif |
| case 'x': |
| use\_caps = false; |
| case 'X': |
| uval = (uint32\_t)va\_arg(ap, uint32\_t); |
| vlen = mknumstr(vstr,&uval,false,16,use\_caps); |
| vstrp = &vstr[vlen]; |
|  |
| dschar = false; |
| if (IS\_FLAG\_ZERO(flags\_used)) |
| { |
| if (IS\_FLAG\_POUND(flags\_used)) |
| { |
| n\_putchar(func\_ptr, '0', farg, &max\_count); |
| n\_putchar(func\_ptr, (use\_caps ? 'X' : 'x'), farg, &max\_count); |
| count += 2; |
| /\*vlen += 2;\*/ |
| dschar = true; |
| } |
| fput\_pad('0', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| vlen = field\_width; |
| } |
| else |
| { |
| if (!IS\_FLAG\_MINUS(flags\_used)) |
| { |
| if (IS\_FLAG\_POUND(flags\_used)) |
| { |
| vlen += 2; |
| } |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| if (IS\_FLAG\_POUND(flags\_used)) |
| { |
| n\_putchar(func\_ptr, '0', farg, &max\_count); |
| n\_putchar(func\_ptr, (use\_caps ? 'X' : 'x'), farg, &max\_count); |
| count += 2; |
|  |
| dschar = true; |
| } |
| } |
| } |
|  |
| if ((IS\_FLAG\_POUND(flags\_used)) && (!dschar)) |
| { |
| n\_putchar(func\_ptr, '0', farg, &max\_count); |
| n\_putchar(func\_ptr, (use\_caps ? 'X' : 'x'), farg, &max\_count); |
| count += 2; |
| vlen += 2; |
| } |
| goto cont\_xd; |
|  |
| case 'o': |
| uval = (uint32\_t)va\_arg(ap, uint32\_t); |
| vlen = mknumstr(vstr,&uval,false,8,use\_caps); |
| goto cont\_u; |
| case 'b': |
| uval = (uint32\_t)va\_arg(ap, uint32\_t); |
| vlen = mknumstr(vstr,&uval,false,2,use\_caps); |
| goto cont\_u; |
| case 'p': |
| uval = (uint32\_t)va\_arg(ap, uint32\_t); |
| uval = (uint32\_t)va\_arg(ap, void \*); |
| vlen = mknumstr(vstr,&uval,false,16,use\_caps); |
| goto cont\_u; |
| case 'u': |
| uval = (uint32\_t)va\_arg(ap, uint32\_t); |
| vlen = mknumstr(vstr,&uval,false,10,use\_caps); |
|  |
| cont\_u: |
| vstrp = &vstr[vlen]; |
|  |
| if (IS\_FLAG\_ZERO(flags\_used)) |
| { |
| fput\_pad('0', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| vlen = field\_width; |
| } |
| else |
| { |
| if (!IS\_FLAG\_MINUS(flags\_used)) |
| { |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| } |
| } |
|  |
| cont\_xd: |
| while (\*vstrp) |
| { |
| n\_putchar(func\_ptr, \*vstrp--, farg, &max\_count); |
| count++; |
| } |
|  |
| if (IS\_FLAG\_MINUS(flags\_used)) |
| { |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| } |
| break; |
|  |
| case 'c': |
| cval = (char)va\_arg(ap, uint32\_t); |
| n\_putchar(func\_ptr, cval, farg, &max\_count); |
| count++; |
| break; |
| case 's': |
| sval = (char \*)va\_arg(ap, char \*); |
| if (sval) |
| { |
| vlen = strlen(sval); |
| if (!IS\_FLAG\_MINUS(flags\_used)) |
| { |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| } |
| while (\*sval) |
| { |
| n\_putchar(func\_ptr, \*sval++, farg, &max\_count); |
| count++; |
| } |
| if (IS\_FLAG\_MINUS(flags\_used)) |
| { |
| fput\_pad(' ', vlen, field\_width, &count, func\_ptr, farg, &max\_count); |
| } |
| } |
| break; |
| case 'n': |
| ivalp = (int32\_t \*)va\_arg(ap, int32\_t \*); |
| \*ivalp = count; |
| break; |
| default: |
| n\_putchar(func\_ptr, c, farg, &max\_count); |
| count++; |
| break; |
| } |
| } |
|  |
| if (max\_count) |
| { |
| return count; |
| } |
| else |
| { |
| return temp\_count; |
| } |
| } |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : \_sputc |
| \* Description : Writes the character into the string located by the string |
| \* pointer and updates the string pointer. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| int \_sputc(int c, void \* input\_string) |
| { |
| char \*\*string\_ptr = (char \*\*)input\_string; |
|  |
| \*(\*string\_ptr)++ = (char)c; |
| return c; |
| } |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : mknumstr |
| \* Description : Converts a radix number to a string and return its length. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| static int32\_t mknumstr (char \*numstr, void \*nump, int32\_t neg, int32\_t radix, bool use\_caps) |
| { |
| int32\_t a,b,c; |
| uint32\_t ua,ub,uc; |
|  |
| int32\_t nlen; |
| char \*nstrp; |
|  |
| nlen = 0; |
| nstrp = numstr; |
| \*nstrp++ = '\0'; |
|  |
| if (neg) |
| { |
| a = \*(int32\_t \*)nump; |
| if (a == 0) |
| { |
| \*nstrp = '0'; |
| ++nlen; |
| goto done; |
| } |
| while (a != 0) |
| { |
| b = (int32\_t)a / (int32\_t)radix; |
| c = (int32\_t)a - ((int32\_t)b \* (int32\_t)radix); |
| if (c < 0) |
| { |
| c = ~c + 1 + '0'; |
| } |
| else |
| { |
| c = c + '0'; |
| } |
| a = b; |
| \*nstrp++ = (char)c; |
| ++nlen; |
| } |
| } |
| else |
| { |
| ua = \*(uint32\_t \*)nump; |
| if (ua == 0) |
| { |
| \*nstrp = '0'; |
| ++nlen; |
| goto done; |
| } |
| while (ua != 0) |
| { |
| ub = (uint32\_t)ua / (uint32\_t)radix; |
| uc = (uint32\_t)ua - ((uint32\_t)ub \* (uint32\_t)radix); |
| if (uc < 10) |
| { |
| uc = uc + '0'; |
| } |
| else |
| { |
| uc = uc - 10 + (use\_caps ? 'A' : 'a'); |
| } |
| ua = ub; |
| \*nstrp++ = (char)uc; |
| ++nlen; |
| } |
| } |
| done: |
| return nlen; |
| } |
|  |
| #if defined(PRINTF\_FLOAT\_ENABLE) |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : mkfloatnumstr |
| \* Description : Converts a floating radix number to a string and return |
| \* its length, user can specify output precision width. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| static int32\_t mkfloatnumstr (char \*numstr, void \*nump, int32\_t radix, uint32\_t precision\_width) |
| { |
| int32\_t a,b,c,i; |
| double fa,fb; |
| double r, fractpart, intpart; |
|  |
| int32\_t nlen; |
| char \*nstrp; |
| nlen = 0; |
| nstrp = numstr; |
| \*nstrp++ = '\0'; |
| r = \*(double \*)nump; |
| if (r == 0) |
| { |
| \*nstrp = '0'; |
| ++nlen; |
| goto done; |
| } |
| fractpart = modf((double)r , (double \*)&intpart); |
| /\* Process fractional part \*/ |
| for (i = 0; i < precision\_width; i++) |
| { |
| fractpart \*= radix; |
| } |
| //a = (int32\_t)floor(fractpart + (double)0.5); |
| if (r >= 0) |
| { |
| fa = fractpart + (double)0.5; |
| } |
| else |
| { |
| fa = fractpart - (double)0.5; |
| } |
| intpart += ((int64\_t)fa - (int64\_t)fractpart); |
| for (i = 0; i < precision\_width; i++) |
| { |
| fb = fa / (int32\_t)radix; |
| c = (int32\_t)(fa - (int64\_t)fb \* (int32\_t)radix); |
| if (c < 0) |
| { |
| c = ~c + 1 + '0'; |
| }else |
| { |
| c = c + '0'; |
| } |
| fa = fb; |
| \*nstrp++ = (char)c; |
| ++nlen; |
| } |
| \*nstrp++ = (char)'.'; |
| ++nlen; |
| a = (int32\_t)intpart; |
| if(a == 0) |
| { |
| \*nstrp++ = '0'; |
| ++nlen; |
| } |
| else |
| { |
| while (a != 0) |
| { |
| b = (int32\_t)a / (int32\_t)radix; |
| c = (int32\_t)a - ((int32\_t)b \* (int32\_t)radix); |
| if (c < 0) |
| { |
| c = ~c + 1 + '0'; |
| }else |
| { |
| c = c + '0'; |
| } |
| a = b; |
| \*nstrp++ = (char)c; |
| ++nlen; |
| } |
| } |
| done: |
| return nlen; |
| } |
| #endif |
|  |
| static void fput\_pad(int32\_t c, int32\_t curlen, int32\_t field\_width, int32\_t \*count, PUTCHAR\_FUNC func\_ptr, void \*farg, int \*max\_count) |
| { |
| int32\_t i; |
|  |
| for (i = curlen; i < field\_width; i++) |
| { |
| func\_ptr((char)c, farg); |
| (\*count)++; |
| } |
| } |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : scan\_prv |
| \* Description : Converts an input line of ASCII characters based upon a |
| \* provided string format. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| int scan\_prv(const char \*line\_ptr, char \*format, va\_list args\_ptr) |
| { |
| uint8\_t base; |
| /\* Identifier for the format string \*/ |
| char \*c = format; |
| const char \*s; |
| char temp; |
| /\* Identifier for the input string \*/ |
| const char \*p = line\_ptr; |
| /\* flag telling the conversion specification \*/ |
| uint32\_t flag = 0 ; |
| /\* filed width for the matching input streams \*/ |
| uint32\_t field\_width; |
| /\* how many arguments are assigned except the suppress \*/ |
| uint32\_t nassigned = 0; |
| /\* how many characters are read from the input streams \*/ |
| uint32\_t n\_decode = 0; |
|  |
| int32\_t val; |
| char \*buf; |
| int8\_t neg; |
|  |
| /\* return EOF error before any convernsion \*/ |
| if (\*p == '\0') |
| { |
| return EOF; |
| } |
|  |
| /\* decode directives \*/ |
| while ((\*c) && (\*p)) |
| { |
| /\* ignore all white-spaces in the format strings \*/ |
| if (scan\_ignore\_white\_space((const char \*\*)&c)) |
| { |
| n\_decode += scan\_ignore\_white\_space(&p); |
| } |
| else if (\*c != '%') |
| { |
| /\* Ordinary characters \*/ |
| c++; |
| ordinary: if (\*p == \*c) |
| { |
| n\_decode++; |
| p++; |
| c++; |
| } |
| else |
| { |
| /\* Match failure. Misalignment with C99, the unmatched |
| \* characters need to be pushed back to stream. HOwever |
| \* , it is deserted now. \*/ |
| break; |
| } |
| } |
| else |
| { |
| /\* convernsion specification \*/ |
| c++; |
| if (\*c == '%') |
| { |
| goto ordinary; |
| } |
|  |
| /\* Reset \*/ |
| flag = 0; |
| field\_width = 0; |
| base = 0; |
|  |
| /\* Loop to get full conversion specification \*/ |
| while ((\*c) && (!(flag & SCAN\_DEST\_MASK))) |
| { |
| switch (\*c) |
| { |
| case '\*': |
| if (flag & SCAN\_SUPPRESS) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_SUPPRESS; |
| c++; |
| break; |
| case 'h': |
| if (flag & SCAN\_LENGTH\_MASK) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_LENGTH\_SHORT\_INT; |
|  |
| if (c[1] == 'h') |
| { |
| flag |= SCAN\_LENGTH\_CHAR; |
| c++; |
| } |
| c++; |
| break; |
| case 'l': |
| if (flag & SCAN\_LENGTH\_MASK) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_LENGTH\_LONG\_INT; |
|  |
| if (c[1] == 'l') |
| { |
| flag |= SCAN\_LENGTH\_LONG\_LONG\_INT; |
| c++; |
| } |
| c++; |
| break; |
| #if defined(ADVANCE) |
| case 'j': |
| if (flag & SCAN\_LENGTH\_MASK) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_LENGTH\_INTMAX; |
| c++ |
| case 'z' |
| if (flag & SCAN\_LENGTH\_MASK) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_LENGTH\_SIZE\_T; |
| c++; |
| break; |
| case 't': |
| if (flag & SCAN\_LENGTH\_MASK) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_LENGTH\_PTRDIFF\_T; |
| c++; |
| break; |
| #endif |
| #if defined(SCANF\_FLOAT\_ENABLE) |
| case 'L': |
| if (flag & SCAN\_LENGTH\_MASK) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| flag |= SCAN\_LENGTH\_LONG\_DOUBLE; |
| c++; |
| break; |
| #endif |
| case '0': |
| case '1': |
| case '2': |
| case '3': |
| case '4': |
| case '5': |
| case '6': |
| case '7': |
| case '8': |
| case '9': |
| if (field\_width) |
| { |
| /\* Match failure\*/ |
| return nassigned; |
| } |
| do { |
| field\_width = field\_width \* 10 + \*c - '0'; |
| c++; |
| } while ((\*c >= '0') && (\*c <= '9')); |
| break; |
| case 'd': |
| flag |= SCAN\_TYPE\_SIGNED; |
| case 'u': |
| base = 10; |
| flag |= SCAN\_DEST\_INT; |
| c++; |
| break; |
| case 'o': |
| base = 8; |
| flag |= SCAN\_DEST\_INT; |
| c++; |
| break; |
| case 'x': |
| case 'X': |
| base = 16; |
| flag |= SCAN\_DEST\_INT; |
| c++; |
| break; |
| case 'i': |
| base = 0; |
| flag |= SCAN\_DEST\_INT; |
| c++; |
| break; |
| #if defined(SCANF\_FLOAT\_ENABLE) |
| case 'a': |
| case 'A': |
| case 'e': |
| case 'E': |
| case 'f': |
| case 'F': |
| case 'g': |
| case 'G': |
| flag |= SCAN\_DEST\_FLOAT; |
| c++; |
| break; |
| #endif |
| case 'c': |
| flag |= SCAN\_DEST\_CHAR; |
| if (!field\_width) |
| { |
| field\_width = 1; |
| } |
| c++; |
| break; |
| case 's': |
| flag |= SCAN\_DEST\_STRING; |
| c++; |
| break; |
| #if defined(ADVANCE) /\* [x]\*/ |
| case '[': |
| flag |= SCAN\_DEST\_SET; |
| /\*Add Set functionality \*/ |
| break; |
| #endif |
| default: |
| #if defined(SCAN\_DEBUG) |
| printf("Unrecognized expression specifier: %c format: %s, number is: %d\r\n", c, format, nassigned); |
| #endif |
| return nassigned; |
| } |
| } |
|  |
| if (!(flag & SCAN\_DEST\_MASK)) |
| { |
| /\* Format strings are exausted \*/ |
| return nassigned; |
| } |
|  |
| if (!field\_width) |
| { |
| /\* Larget then length of a line \*/ |
| field\_width = 99; |
| } |
|  |
| /\* Matching strings in input streams and assign to argument \*/ |
| switch (flag & SCAN\_DEST\_MASK) |
| { |
| case SCAN\_DEST\_CHAR: |
| s = (const char \*)p; |
| buf = va\_arg(args\_ptr, char \*); |
| while ((field\_width--) && (\*p)) |
| { |
| if (!(flag & SCAN\_SUPPRESS)) |
| { |
| \*buf++ = \*p++; |
| } |
| else |
| { |
| p++; |
| } |
| n\_decode++; |
| } |
|  |
| if (((!(flag)) & SCAN\_SUPPRESS) && (s != p)) |
| { |
| nassigned++; |
| } |
| break; |
| case SCAN\_DEST\_STRING: |
| n\_decode += scan\_ignore\_white\_space(&p); |
| s = p; |
| buf = va\_arg(args\_ptr, char \*); |
| while ((field\_width--) && (\*p != '\0') && (\*p != ' ') && |
| (\*p != '\t') && (\*p != '\n') && (\*p != '\r') && (\*p != '\v') && (\*p != '\f')) |
| { |
| if (flag & SCAN\_SUPPRESS) |
| { |
| p++; |
| } |
| else |
| { |
| \*buf++ = \*p++; |
| } |
| n\_decode++; |
| } |
|  |
| if ((!(flag & SCAN\_SUPPRESS)) && (s != p)) |
| { |
| /\* Add NULL to end of string \*/ |
| \*buf = '\0'; |
| nassigned++; |
| } |
| break; |
| case SCAN\_DEST\_INT: |
| n\_decode += scan\_ignore\_white\_space(&p); |
| s = p; |
| val = 0; |
| /\*TODO: scope is not testsed \*/ |
| if ((base == 0) || (base == 16)) |
| { |
| if ((s[0] == '0') && ((s[1] == 'x') || (s[1] == 'X'))) |
| { |
| base = 16; |
| if (field\_width >= 1) |
| { |
| p += 2; |
| n\_decode += 2; |
| field\_width -= 2; |
| } |
| } |
| } |
|  |
| if (base == 0) |
| { |
| if (s[0] == '0') |
| { |
| base = 8; |
| } |
| else |
| { |
| base = 10; |
| } |
| } |
|  |
| neg = 1; |
| switch (\*p) |
| { |
| case '-': |
| neg = -1; |
| n\_decode++; |
| p++; |
| field\_width--; |
| break; |
| case '+': |
| neg = 1; |
| n\_decode++; |
| p++; |
| field\_width--; |
| break; |
| default: |
| break; |
| } |
|  |
| while ((\*p) && (field\_width--)) |
| { |
| if ((\*p <= '9') && (\*p >= '0')) |
| { |
| temp = \*p - '0'; |
| } |
| else if((\*p <= 'f') && (\*p >= 'a')) |
| { |
| temp = \*p - 'a' + 10; |
| } |
| else if((\*p <= 'F') && (\*p >= 'A')) |
| { |
| temp = \*p - 'A' + 10; |
| } |
| else |
| { |
| break; |
| } |
|  |
| if (temp >= base) |
| { |
| break; |
| } |
| else |
| { |
| val = base \* val + temp; |
| } |
| p++; |
| n\_decode++; |
| } |
|  |
| val \*= neg; |
| if (!(flag & SCAN\_SUPPRESS)) |
| { |
| switch (flag & SCAN\_LENGTH\_MASK) |
| { |
| case SCAN\_LENGTH\_CHAR: |
| if (flag & SCAN\_TYPE\_SIGNED) |
| { |
| \*va\_arg(args\_ptr, signed char \*) = (signed char)val; |
| } |
| else |
| { |
| \*va\_arg(args\_ptr, unsigned char \*) = (unsigned char)val; |
| } |
| break; |
| case SCAN\_LENGTH\_SHORT\_INT: |
| if (flag & SCAN\_TYPE\_SIGNED) |
| { |
| \*va\_arg(args\_ptr, signed short \*) = (signed short)val; |
| } |
| else |
| { |
| \*va\_arg(args\_ptr, unsigned short \*) = (unsigned short)val; |
| } |
| break; |
| case SCAN\_LENGTH\_LONG\_INT: |
| if (flag & SCAN\_TYPE\_SIGNED) |
| { |
| \*va\_arg(args\_ptr, signed long int \*) = (signed long int)val; |
| } |
| else |
| { |
| \*va\_arg(args\_ptr, unsigned long int \*) = (unsigned long int)val; |
| } |
| break; |
| case SCAN\_LENGTH\_LONG\_LONG\_INT: |
| if (flag & SCAN\_TYPE\_SIGNED) |
| { |
| \*va\_arg(args\_ptr, signed long long int \*) = (signed long long int)val; |
| } |
| else |
| { |
| \*va\_arg(args\_ptr, unsigned long long int \*) = (unsigned long long int)val; |
| } |
| break; |
| default: |
| /\* The default type is the type int \*/ |
| if (flag & SCAN\_TYPE\_SIGNED) |
| { |
| \*va\_arg(args\_ptr, signed int \*) = (signed int)val; |
| } |
| else |
| { |
| \*va\_arg(args\_ptr, unsigned int \*) = (unsigned int)val; |
| } |
| break; |
| } |
| nassigned++; |
| } |
| break; |
| #if defined(SCANF\_FLOAT\_ENABLE) |
| case SCAN\_DEST\_FLOAT: |
| n\_decode += scan\_ignore\_white\_space(&p); |
| fnum = strtod(p, (char \*\*)&s); |
|  |
| if ((fnum == HUGE\_VAL) || (fnum == -HUGE\_VAL)) |
| { |
| break; |
| } |
|  |
| n\_decode += (int)(s) - (int)(p); |
| p = s; |
| if (!(flag & SCAN\_SUPPRESS)) |
| { |
| if (flag & SCAN\_LENGTH\_LONG\_DOUBLE) |
| { |
| \*va\_arg(args\_ptr, double \*) = fnum; |
| } |
| else |
| { |
| \*va\_arg(args\_ptr, float \*) = (float)fnum; |
| } |
| nassigned++; |
| } |
| break; |
| #endif |
| #if defined(ADVANCE) |
| case SCAN\_DEST\_SET: |
| break; |
| #endif |
| default: |
| #if defined(SCAN\_DEBUG) |
| printf("ERROR: File %s line: %d\r\n", \_\_FILE\_\_, \_\_LINE\_\_); |
| #endif |
| return nassigned; |
| } |
| } |
| } |
| return nassigned; |
| } |
|  |
| /\*FUNCTION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* Function Name : scan\_ignore\_white\_space |
| \* Description : Scanline function which ignores white spaces. |
| \* |
| \*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| static uint32\_t scan\_ignore\_white\_space(const char \*\*s) |
| { |
| uint8\_t count = 0; |
| uint8\_t c; |
|  |
| c = \*\*s; |
| while ((c == ' ') || (c == '\t') || (c == '\n') || (c == '\r') || (c == '\v') || (c == '\f')) |
| { |
| count++; |
| (\*s)++; |
| c = \*\*s; |
| } |
| return count; |
| } |

**print\_scan.h**

|  |
| --- |
| #ifndef \_\_print\_scan\_h\_\_ |
| #define \_\_print\_scan\_h\_\_ |
|  |
| #include <stdio.h> |
| #include <stdarg.h> |
| #include <stdint.h> |
| #include <stdbool.h> |
| #include <string.h> |
|  |
| //#define PRINTF\_FLOAT\_ENABLE 1 |
| //#define PRINT\_MAX\_COUNT 1 |
| //#define SCANF\_FLOAT\_ENABLE 1 |
|  |
| #ifndef HUGE\_VAL |
| #define HUGE\_VAL (99.e99)///wrong value |
| #endif |
|  |
| typedef int (\*PUTCHAR\_FUNC)(int a, void \*b); |
|  |
| /\*! |
| \* @brief This function outputs its parameters according to a formatted string. |
| \* |
| \* @note I/O is performed by calling given function pointer using following |
| \* (\*func\_ptr)(c,farg); |
| \* |
| \* @param[in] farg Argument to func\_ptr. |
| \* @param[in] func\_ptr Function to put character out. |
| \* @param[in] max\_count Maximum character count for snprintf and vsnprintf. |
| \* Default value is 0 (unlimited size). |
| \* @param[in] fmt\_ptr Format string for printf. |
| \* @param[in] args\_ptr Arguments to printf. |
| \* |
| \* @return Number of characters |
| \* @return EOF (End Of File found.) |
| \*/ |
| int \_doprint(void \*farg, PUTCHAR\_FUNC func\_ptr, int max\_count, char \*fmt, va\_list ap); |
|  |
| /\*! |
| \* @brief Writes the character into the string located by the string pointer and |
| \* updates the string pointer. |
| \* |
| \* @param[in] c The character to put into the string. |
| \* @param[in, out] input\_string This is an updated pointer to a string pointer. |
| \* |
| \* @return Character written into string. |
| \*/ |
| int \_sputc(int c, void \* input\_string); |
|  |
| /\*! |
| \* @brief Converts an input line of ASCII characters based on a provided |
| \* string format. |
| \* |
| \* @param[in] line\_ptr The input line of ASCII data. |
| \* @param[in] format Format first points to the format string. |
| \* @param[in] args\_ptr The list of parameters. |
| \* |
| \* @return Number of input items converted and assigned. |
| \* @return IO\_EOF - When line\_ptr is empty string "". |
| \*/ |
| int scan\_prv(const char \*line\_ptr, char \*format, va\_list args\_ptr); |
|  |
| #endif |

**Custom\_ASCII\_Counter.c**

|  |
| --- |
| #include "Custom\_ASCII\_Counter.h" |
| #include "Custom\_UART.h" |
| #include "Custom\_Circular\_Buffer.h" |
|  |
| DWord Fib\_n; |
|  |
| Byte Counter, ASCII\_Char; |
| DWord ASCII\_Array[256], ASCII\_Value; |
| char UART\_print[50]; |
|  |
| //Function to set the array properly |
| void ASCII\_Counter\_Init(void) |
| { |
| //Storing all ascii values in higher 8 bits |
| for(Counter = 0; Counter < 0xFF; Counter ++) ASCII\_Array[Counter] = (Counter << 24); |
| ASCII\_Array[Counter] = (Counter << 24); |
| } |
|  |
| //Actual application |
| void ASCII\_Counter(void) |
| { |
| //Give out warning if overwriting |
| if(CBuffer\_Instance[UART0\_Rx\_Buffer\_ID].Status == Full) Output\_String("\n\rOverwriting"); |
|  |
| //Read one byte |
| CBuffer\_Byte\_Read(UART0\_Rx\_Buffer\_ID, &ASCII\_Char); |
|  |
| //Scan for all valid ascii values |
| for(Counter = 0; Counter < 0xFF; Counter ++) |
| { |
| //If match then increase the count by 1 |
| if((ASCII\_Array[Counter] >> 24) == ASCII\_Char) |
| { |
| ASCII\_Value = ASCII\_Array[Counter] & ASCII\_Counter\_Mask; |
| ASCII\_Value += 1; |
| ASCII\_Value &= ASCII\_Counter\_Mask; |
| ASCII\_Array[Counter] &= ASCII\_Char\_Mask; |
| ASCII\_Array[Counter] |= ASCII\_Value; |
| break; |
| } |
| } |
| if(Counter == 0xFF) |
| { |
| ASCII\_Value = ASCII\_Array[Counter] & ASCII\_Counter\_Mask; |
| ASCII\_Value += 1; |
| ASCII\_Value &= ASCII\_Counter\_Mask; |
| ASCII\_Array[Counter] &= ASCII\_Char\_Mask; |
| ASCII\_Array[Counter] |= ASCII\_Value; |
| } |
|  |
| //Printing fibonacci number |
| sprintf(UART\_print, "n\rFibonacci Number: %lu", Fib\_n); |
| Output\_String(UART\_print); |
|  |
| //Printing report |
| Output\_String("\n\rNew Data\n\r"); |
| for(Counter = 0; Counter < 0xFF; Counter ++) |
| { |
| ASCII\_Char = ASCII\_Array[Counter] >> 24; |
| ASCII\_Value = ASCII\_Array[Counter] & ASCII\_Counter\_Mask; |
| if(ASCII\_Value != 0) |
| { |
| sprintf(UART\_print, "\n\rCharacter: %c\tHex: 0x%02X\tOccurrence: %ld", ASCII\_Char, ASCII\_Char, ASCII\_Value); |
| Output\_String(UART\_print); |
| } |
| } |
| Output\_String("\n\r\n\rEnter Text:\n\r"); |
| } |

**Custom\_Circular\_Buffer.c**

|  |
| --- |
| #include "Custom\_Circular\_Buffer.h" |
|  |
| //Variables |
|  |
| CBuffer CBuffer\_Instance[Maximum\_Buffers]; |
| ptr\_type Location, Continuous\_Read; |
| Byte CBuffer\_Data, Error, No\_of\_CBuffers, cbuffer\_i, cbuffer\_j; |
| char CBuffer\_Input[10]; |
| DWord CBuffer\_Instance\_Length[Maximum\_Buffers], value; |
| Byte return\_value; |
| Byte resize = 0; |
|  |
|  |
| //Function to get values from string - SCANF can be used instead of FGETS |
|  |
| void String\_to\_Decimal(char \*stod\_ptr) |
| { |
| if(stod\_ptr) |
| { |
| char \*stod\_i; |
| stod\_i = stod\_ptr; |
| for(; \*stod\_ptr != 0; stod\_ptr ++) |
| { |
| if(isdigit(\*stod\_ptr) == 0) |
| { |
| Output\_String("\n\r\rNon Integer Value Entered\n\r\r"); |
| Error = 1; |
| value = 0; |
| break; |
| } |
| } |
| if(\*stod\_ptr == 0) |
| { |
| Error = 0; |
| value = atoi(stod\_i); |
| } |
| } |
| else |
| { |
| Output\_String("\n\rNull Pointer\n\r"); |
| Error = 1; |
| value = 0; |
| } |
| } |
|  |
| // Initializing each of the buffers |
|  |
| Byte CBuffer\_Assign(Byte CBuffer\_ID) |
| { |
| CBuffer\_Instance[CBuffer\_ID].Length = CBuffer\_Instance\_Length[CBuffer\_ID]; |
| CBuffer\_Instance[CBuffer\_ID].Elements\_count = 0; |
| CBuffer\_Instance[CBuffer\_ID].Start\_ptr = (Byte \*)malloc(CBuffer\_Instance\_Length[CBuffer\_ID]); |
| if(CBuffer\_Instance[CBuffer\_ID].Start\_ptr == 0) return 1; |
| CBuffer\_Instance[CBuffer\_ID].Head = 0; |
| CBuffer\_Instance[CBuffer\_ID].Tail = 0; |
| CBuffer\_Instance[CBuffer\_ID].Index = 0; |
| CBuffer\_Instance[CBuffer\_ID].Status = Empty; |
| return 0; |
| } |
|  |
| //Input about buffers from user |
|  |
| Byte CBuffer\_Init(void) |
| { |
| No\_of\_CBuffers = 2; |
| CBuffer\_Instance\_Length[FGETS\_Buffer\_ID] = FGETS\_Buffer\_Length; |
| // if(CBuffer\_Assign(FGETS\_Buffer\_ID)) return 1; |
| //Get length of each, store it in array called CBuffer\_Instance\_Length |
| for(cbuffer\_i = 1; cbuffer\_i < No\_of\_CBuffers; cbuffer\_i ++) |
| { |
| Output\_String("\n\rEnter the length of "); |
| Output\_String("Rx Buffer: "); |
| Input\_String(CBuffer\_Input, FGETS\_Buffer\_Length, stdin); |
| cbuffer\_j = 0; |
| while(CBuffer\_Input[cbuffer\_j] != Enter\_Detected) cbuffer\_j ++; |
| CBuffer\_Input[cbuffer\_j] = 0; |
| Output\_String("\n\rLength set: "); |
| String\_to\_Decimal(CBuffer\_Input); |
| if(Error) return 1; |
| CBuffer\_Instance\_Length[cbuffer\_i] = value; |
| sprintf(CBuffer\_Input, "%ld", value); |
| Output\_String(CBuffer\_Input); |
| } |
|  |
| //Initialize each buffer |
| for(cbuffer\_i = 1; cbuffer\_i < No\_of\_CBuffers; cbuffer\_i ++) |
| { |
| if(CBuffer\_Assign(cbuffer\_i)) return 1; |
| } |
| return 0; |
| } |
|  |
| //Function to write 1 byte in given buffer |
|  |
| Byte CBuffer\_Byte\_Write(Byte CBuffer\_ID, Byte data) |
| { |
| //If buffer is non empty and head meets the tail then buffer is full and overwriting |
| if((CBuffer\_Instance[CBuffer\_ID].Status != Empty) && (CBuffer\_Instance[CBuffer\_ID].Head == CBuffer\_Instance[CBuffer\_ID].Tail)) |
| { |
| CBuffer\_Instance[CBuffer\_ID].Status = Full; |
| return\_value = Overwriting; |
| } |
| else |
| { |
| if(return\_value != Overwriting) return\_value = Success; |
| } |
|  |
| //Attempt to not lose data even when buffer is full |
| if(return\_value == Overwriting) |
| { |
| CBuffer\_Instance[CBuffer\_ID].Start\_ptr = (Byte \*)realloc(CBuffer\_Instance[CBuffer\_ID].Start\_ptr, CBuffer\_Instance\_Length[CBuffer\_ID]); |
| CBuffer\_Instance[CBuffer\_ID].Head = CBuffer\_Instance[CBuffer\_ID].Length; |
| CBuffer\_Instance[CBuffer\_ID].Length += CBuffer\_Instance\_Length[CBuffer\_ID]; |
| CBuffer\_Instance[CBuffer\_ID].Status = Success; |
| return\_value = Success; |
| } |
|  |
| //Write data using start pointer + relative address from head |
| \*(CBuffer\_Instance[CBuffer\_ID].Start\_ptr + CBuffer\_Instance[CBuffer\_ID].Head) = data; |
|  |
|  |
| //Joining end to start - that is if head is at the end, instead of incrementing, wrap it back to the start (0) |
| if(CBuffer\_Instance[CBuffer\_ID].Head == (CBuffer\_Instance[CBuffer\_ID].Length - 1)) CBuffer\_Instance[CBuffer\_ID].Head = 0; |
| else CBuffer\_Instance[CBuffer\_ID].Head += 1; |
|  |
| //Raising this flag after actual write is necessary for other functions |
| if(CBuffer\_Instance[CBuffer\_ID].Head == CBuffer\_Instance[CBuffer\_ID].Tail) CBuffer\_Instance[CBuffer\_ID].Status = Full; |
|  |
| //Change status flag if wrote to empty buffer |
| if(CBuffer\_Instance[CBuffer\_ID].Status == Empty) CBuffer\_Instance[CBuffer\_ID].Status = Success; |
|  |
| //Read chain broke down by writing |
| Continuous\_Read = 0; |
|  |
| //Added element, increment count |
| if(CBuffer\_Instance[CBuffer\_ID].Status != Full) CBuffer\_Instance[CBuffer\_ID].Elements\_count += 1; |
| else CBuffer\_Instance[CBuffer\_ID].Elements\_count = CBuffer\_Instance[CBuffer\_ID].Length; |
| return (return\_value); |
| } |
|  |
| Byte CBuffer\_Byte\_Read(Byte CBuffer\_ID, Byte \*address) |
| { |
| if(return\_value == Overwriting) return\_value = Success; |
|  |
| //If read the whole buffer then mark it as empty |
| if(Continuous\_Read == CBuffer\_Instance[CBuffer\_ID].Length) |
| { |
| CBuffer\_Instance[CBuffer\_ID].Tail = CBuffer\_Instance[CBuffer\_ID].Head; |
| CBuffer\_Instance[CBuffer\_ID].Status = Empty; |
| Output\_String("e1"); |
| return (Empty); |
| } |
| else |
| { |
| //Hasn't performed maximum number of reads but, tail and head meet |
| if(CBuffer\_Instance[CBuffer\_ID].Tail == CBuffer\_Instance[CBuffer\_ID].Head) |
| { |
| //If buffer wasn't full then it is empty now |
| if(CBuffer\_Instance[CBuffer\_ID].Status != Full) |
| { |
| CBuffer\_Instance[CBuffer\_ID].Status = Empty; |
| Output\_String("e2"); |
| return (Empty); |
| } |
|  |
| //If the buffer was full then update the index to mark that this will be the new tail starting location after reading everything |
| else |
| { |
| CBuffer\_Instance[CBuffer\_ID].Index = CBuffer\_Instance[CBuffer\_ID].Tail; |
| CBuffer\_Instance[CBuffer\_ID].Status = Success; |
| } |
| } |
| } |
|  |
| //read using pointer |
| \*address = \*(CBuffer\_Instance[CBuffer\_ID].Start\_ptr + CBuffer\_Instance[CBuffer\_ID].Tail); |
|  |
| //joining end to start to make a circular path |
| if(CBuffer\_Instance[CBuffer\_ID].Tail == (CBuffer\_Instance[CBuffer\_ID].Length - 1)) CBuffer\_Instance[CBuffer\_ID].Tail = 0; |
| else CBuffer\_Instance[CBuffer\_ID].Tail += 1; |
|  |
| //increases continuous read variable |
| Continuous\_Read += 1; |
|  |
| if(CBuffer\_Instance[CBuffer\_ID].Tail == CBuffer\_Instance[CBuffer\_ID].Head) |
| { |
| if(CBuffer\_Instance[CBuffer\_ID].Status != Full) |
| { |
| Output\_String("e3"); |
| CBuffer\_Instance[CBuffer\_ID].Status = Empty; |
| } |
| } |
|  |
| //element removed |
| if(CBuffer\_Instance[CBuffer\_ID].Elements\_count > 0) CBuffer\_Instance[CBuffer\_ID].Elements\_count -= 1; |
| return (Success); |
| } |
|  |
| //Function to handle both read and write |
| void CBuffer\_Operation(Byte CBuffer\_ID, Byte type, Byte data, Byte \*address) |
| { |
| if(type == Write) |
| { |
| //the only error is overwriting warning |
| if(CBuffer\_Byte\_Write(CBuffer\_ID, data)) Output\_String("Buffer%d Full, Overwriting\n\r", CBuffer\_ID); |
|  |
| //printing useful information |
| Location = CBuffer\_Instance[CBuffer\_ID].Head; |
| if(Location) Location -= 1; |
| else Location = CBuffer\_Instance[CBuffer\_ID].Length - 1; |
| Output\_String("Wrote %c at %d in buffer%d", data, Location, CBuffer\_ID); |
| } |
| else if(type == Read) |
| { |
| //only error is buffer empty |
| if(CBuffer\_Byte\_Read(CBuffer\_ID, address)) |
| { |
| Output\_String("Buffer%d Empty\n\r", CBuffer\_ID); |
| } |
| else |
| { |
| Location = CBuffer\_Instance[CBuffer\_ID].Tail; |
| if(Location) Location -= 1; |
| else Location = CBuffer\_Instance[CBuffer\_ID].Length - 1; |
| Output\_String("Read return is %c from %d in buffer%d", CBuffer\_Data, Location, CBuffer\_ID); |
| } |
| } |
| } |
|  |
| DWord CBuffer\_Elements(Byte CBuffer\_ID) |
| { |
| return (CBuffer\_Instance[CBuffer\_ID].Elements\_count); |
| } |
|  |
| //resizing existing buffer |
| Byte CBuffer\_Resize(Byte CBuffer\_ID) |
| { |
| Output\_String("\n\rEnter new length of buffer%d: ", CBuffer\_ID); |
| Input\_String(CBuffer\_Input, 10, stdin); |
| cbuffer\_j = 0; |
| while(CBuffer\_Input[cbuffer\_j] != Enter\_Detected) cbuffer\_j ++; |
| CBuffer\_Input[cbuffer\_j] = 0; |
| String\_to\_Decimal(CBuffer\_Input); |
| if(Error) return 1; |
| free(CBuffer\_Instance[CBuffer\_ID].Start\_ptr); |
| CBuffer\_Instance\_Length[CBuffer\_ID] = value; |
| CBuffer\_Assign(CBuffer\_ID); |
| return 0; |
| } |

**Custom\_UART.c**

|  |
| --- |
| #include "Custom\_UART.h" |
| #include "Custom\_Circular\_Buffer.h" |
| volatile Byte UART0\_Byte; |
| UART0\_Operation\_Type State = Normal\_Operation; |
|  |
| volatile Byte isr\_arr[50], isr\_cnt = 0; |
| Byte led; |
|  |
| //UART0 Initialization Function |
| void Custom\_UART0\_Init(void) |
| { |
| //Enabling clock first |
| Enable\_UART0\_Clock(); |
|  |
| //Selecting proper Mux values for UART function |
| Enable\_UART0\_Rx\_Function(); |
| Enable\_UART0\_Tx\_Function(); |
|  |
| //Disabling pins for configuring UART safely |
| Disable\_UART0\_Tx(); |
| Disable\_UART0\_Rx(); |
|  |
| //Selecting and configuring clock source to drive UART |
| Select\_PLL\_Clock\_Divby2(); |
| UART0\_FLL\_PLL\_Clock\_Source(); |
| Set\_BAUD\_Rate\_High\_Register(); |
| Set\_BAUD\_Rate\_Low\_Register(); |
|  |
| //Addition steps for interrupt mode |
| #ifdef INTERRUPT\_MODE |
| Enable\_Rx\_Interrupt(); |
| NVIC\_EnableIRQ(UART0\_IRQn); |
| #endif |
|  |
| //Selecting oversampling value |
| Set\_Oversampling(); |
|  |
| //Enabling pins |
| Enable\_UART0\_Tx(); |
| Enable\_UART0\_Rx(); |
| } |
|  |
| //Polling transmitting byte function |
| void Custom\_UART0\_Tx\_Byte(Byte data) |
| { |
| //Polling flag to check for availability of UART transmitter |
| UART0\_Wait\_for\_Tx\_Data\_Register(); |
|  |
| //Putting byte in buffer/data register |
| UART0\_Tx\_Data(data); |
| } |
|  |
| //Function to transmit strings through UART |
| void Custom\_UART0\_Tx\_String(char \*array) |
| { |
| DWord uart\_i; |
| DWord string\_length = strlen(array); |
|  |
| //For interrupt mode, set a array which is shared between ISR and this function |
| //Cleaning up that array here |
| #ifdef INTERRUPT\_MODE |
| while(isr\_cnt != 0); |
| for(uart\_i = 0; uart\_i < 50; uart\_i ++) isr\_arr[uart\_i] = 0; |
| #endif |
| //Acutally setting up array |
| for(uart\_i = 0; uart\_i < string\_length; uart\_i ++) |
| { |
| //Calling polling transmit byte function repeatedly |
| #ifdef POLLING\_MODE |
| Custom\_UART0\_Tx\_Byte(array[uart\_i]); |
| #else |
| //Filling data in shared array |
| isr\_arr[uart\_i] = array[uart\_i]; |
| #endif |
| } |
| //Enable transmitter buffer empty interrupt |
| //It should be kept disable normally to avoid going into ISR infinitely, continuously, and instantly |
| #ifdef INTERRUPT\_MODE |
| Enable\_TxE\_Interrupt(); |
| #endif |
| } |
|  |
| //Polling function to receive a byte |
| void Custom\_UART0\_Rx\_Byte(volatile Byte \*address) |
| { |
| //Polling flag to see if any data has been received |
| UART0\_Wait\_for\_Rx\_Data\_Register(); |
|  |
| //Storing that byte using the pointer of the variable |
| UART0\_Rx\_Data(address); |
| } |
|  |
| void UART0\_IRQHandler(void) |
| { |
| //Check whether Rx interrupt has caused the code to go in ISR or Tx empty interrupt |
| if(UART0\_Rx\_Interrupt()) |
| { |
| //First store the byte in a variable |
| UART0\_Rx\_Data(&UART0\_Byte); |
|  |
| #if APPLICATION |
|  |
| //To see whether it's FGETS running or normal one |
| if(State == Normal\_Operation) |
| { |
| //Write byte in the UART Rx circular buffer |
| CBuffer\_Byte\_Write(UART0\_Rx\_Buffer\_ID, UART0\_Byte); |
|  |
| //Code for fun :P |
| if(led < 7) led += 1; |
| else led = 0; |
| } |
| else if(State == FGETS\_Operation) |
| { |
| //Echo byte to actually see what's being typed in |
| UART0\_Tx\_Data(UART0\_Byte); |
|  |
| //Write byte in FGETS circular buffer |
| CBuffer\_Byte\_Write(FGETS\_Buffer\_ID, UART0\_Byte); |
|  |
| //Leave FGETS function is the user presses enter or the buffer is filled up |
| if((CBuffer\_Instance[FGETS\_Buffer\_ID].Status == Full) || (Enter\_Detected == UART0\_Byte)) State = Normal\_Operation; |
| } |
|  |
| #else |
| UART0\_Tx\_Data(UART0\_Byte); |
| #endif |
|  |
| } |
| else if(UART0\_TxE\_Interrupt()) |
| { |
| //Transmit one byte if the index in shared array hasn't reached null |
| if(isr\_arr[isr\_cnt] != 0) UART0\_Tx\_Data(isr\_arr[isr\_cnt ++]); |
|  |
| //If the index is pointing to null then reset index, and disable Tx empty interrupt |
| else |
| { |
| Disable\_TxE\_Interrupt(); |
| isr\_cnt = 0; |
| } |
| } |
| } |

**Custom\_ASCII\_Counter.c**

|  |
| --- |
| #ifndef CUSTOM\_INCLUDES\_CUSTOM\_ASCII\_COUNTER\_H\_ |
| #define CUSTOM\_INCLUDES\_CUSTOM\_ASCII\_COUNTER\_H\_ |
|  |
| #include "Custom\_Main.h" |
|  |
| //A single 32bit array is used for both - to store all ascii values and to store counts of them |
| //Each double word will have both. MSB 8 bits holding ascii char and lower 24 bits holding counts |
| #define ASCII\_Char\_Mask 0xFF000000 |
| #define ASCII\_Counter\_Mask 0x00FFFFFF |
|  |
| extern Byte Counter, ASCII\_Char; |
| extern DWord ASCII\_Array[256], ASCII\_Value; |
|  |
| void ASCII\_Counter\_Init(void); |
| void ASCII\_Counter(void); |
|  |
| #endif /\* CUSTOM\_INCLUDES\_CUSTOM\_ASCII\_COUNTER\_H\_ \*/ |

**Custom\_Circular\_Buffer.h**

|  |
| --- |
| #ifndef CUSTOM\_CIRCULAR\_BUFFER\_H\_ |
| #define CUSTOM\_CIRCULAR\_BUFFER\_H\_ |
|  |
| #include "Custom\_Main.h" |
|  |
| #define Success 0 |
| #define Overwriting 1 |
| #define Empty 2 |
| #define Full 3 |
| #define Write 0 |
| #define Read 1 |
|  |
| #define FGETS\_Buffer\_ID 0 |
| #define UART0\_Rx\_Buffer\_ID 1 |
| #define UART0\_Tx\_Buffer\_ID 2 |
|  |
| #define FGETS\_Buffer\_Length 10 |
|  |
|  |
| typedef struct |
| { |
| Byte \*Start\_ptr; |
| Byte Status; |
| DWord Length; |
| DWord Elements\_count; |
| ptr\_type Head; |
| ptr\_type Tail; |
| ptr\_type Index; |
| }CBuffer; |
|  |
| #define Maximum\_Buffers 10 |
|  |
| extern CBuffer CBuffer\_Instance[Maximum\_Buffers]; |
| extern Byte CBuffer\_Data, Error, No\_of\_CBuffers; |
| extern ptr\_type Location, Continuous\_Read; |
| extern char CBuffer\_Input[10]; |
| extern DWord CBuffer\_Instance\_Length[Maximum\_Buffers], value; |
|  |
|  |
| void String\_to\_Decimal(char \*stod\_ptr); |
| Byte CBuffer\_Assign(Byte CBuffer\_ID); |
| Byte CBuffer\_Init(void); |
| Byte CBuffer\_Byte\_Write(Byte CBuffer\_ID, Byte data); |
| Byte CBuffer\_Byte\_Read(Byte CBuffer\_ID, Byte \*address); |
| void CBuffer\_Operation(Byte CBuffer\_ID, Byte type, Byte data, Byte \*address); |
| DWord CBuffer\_Elements(Byte CBuffer\_ID); |
| Byte CBuffer\_Resize(Byte CBuffer\_ID); |
|  |
|  |
|  |
| #endif /\* CUSTOM\_CIRCULAR\_BUFFER\_H\_ \*/ |

**Custom\_Main.h**

|  |
| --- |
| #ifndef CUSTOM\_MAIN\_H\_ |
| #define CUSTOM\_MAIN\_H\_ |
|  |
|  |
| #include<stdint.h> |
| #include<inttypes.h> |
| #include<stdio.h> |
| #include<stdlib.h> |
| #include<string.h> |
|  |
|  |
| #include "Custom\_Sys\_Identifier.h" |
|  |
| #ifdef FRDM |
| //#include "core\_cm0plus.h" |
| #include "MKL25Z4.h" |
| //#include "board.h" |
| #include "fsl\_debug\_console.h" |
| //#include "fsl\_os\_abstraction\_bm.h" |
| #endif |
|  |
| #define Invalid() output\_string("\nInvalid Command\n") |
| #define Null\_Ptr() output\_string("\nNull Pointer. Can't proceed. Abort.\n") |
|  |
| typedef uint8\_t Byte; |
| typedef uint16\_t Word; |
| typedef uint32\_t DWord; |
| typedef volatile uint8\_t vuint8\_t; |
| typedef volatile uint32\_t vuint32\_t; |
|  |
| extern char UART\_print[50]; |
| extern DWord Fib\_n; |
| extern Byte led; |
|  |
|  |
| void FGETS(char \*array\_to\_write, Byte bytes, FILE \*stream); |
|  |
| #endif /\* CUSTOM\_MAIN\_H\_ \*/ |

**Custom\_Sys\_Identifier.h**

|  |
| --- |
| #ifndef \_\_CUSTOM\_SYS\_IDENTIFIER\_H\_\_ |
| #define \_\_CUSTOM\_SYS\_IDENTIFIER\_H\_\_ |
|  |
| /\* |
| \* This header file decides if the device being targeted is a host machine or an embedded system. |
| \* It will check if the target is Linux, Windows, or a Mac. Else, it is an embedded system. |
| \* It will then adjust the code accordingly. |
| \* |
|  |
| \*/ |
|  |
| /\* Compiler flag \*/ |
| #define TARGET\_DEVICE (\_\_linux\_\_ || \_WIN64 || \_\_APPLE\_\_ ) |
| #if TARGET\_DEVICE |
| #define HOST |
| #else |
| #define FRDM |
| #endif |
|  |
| #ifdef HOST |
| #define Output\_String printf |
| #define Input\_String fgets |
| typedef uint64\_t ptr\_type; |
| #define exit\_function system\_exit |
| #define Enter\_Detected '\n' |
| #else |
| #define Output\_String Custom\_UART0\_Tx\_String |
| #define Input\_String FGETS |
| typedef volatile uint32\_t ptr\_type; |
| #define exit\_function embedded\_exit |
| #define Enter\_Detected 0x0D |
| #endif |
|  |
|  |
| #endif /\* CUSTOM\_SYSTEM\_IDENTIFIER\_H\_ \*/ |

**Custom\_UART.h**

|  |
| --- |
| #ifndef CUSTOM\_INCLUDES\_CUSTOM\_UART\_H\_ |
| #define CUSTOM\_INCLUDES\_CUSTOM\_UART\_H\_ |
|  |
| #include "Custom\_Main.h" |
|  |
| //Defines and conditions for switching between polling and interrupt mode |
| #define POLLING 0 |
| #define INTERRUPT 1 |
|  |
| #define UART\_MODE INTERRUPT |
|  |
| #if UART\_MODE |
| #define INTERRUPT\_MODE |
| #define APPLICATION 1 |
| #else |
| #define POLLING\_MODE |
| #endif |
|  |
| //Defines for clocking UART |
| #define Clock\_Gating\_Register\_4 SCGC4 |
| #define System\_Integration\_Module SIM |
| #define UART0\_Clock\_Gate\_Bit 10 |
| #define Enable\_UART0\_Clock() (System\_Integration\_Module->Clock\_Gating\_Register\_4 |= (1 << UART0\_Clock\_Gate\_Bit)) |
|  |
| //Define for function selection |
| #define UART0\_Port PORTA |
| #define Pin\_Control\_Register PCR |
| #define UART0\_Rx\_Pin 1 |
| #define UART0\_Tx\_Pin 2 |
| #define UART0\_Function 0x02 //page 162 in ref manual |
| #define Pin\_Function\_Select(x) PORT\_PCR\_MUX(x) |
|  |
| //Macros for function select |
| #define Enable\_UART0\_Rx\_Function() (UART0\_Port->Pin\_Control\_Register[UART0\_Rx\_Pin] |= \ |
| Pin\_Function\_Select(UART0\_Function)) |
| #define Enable\_UART0\_Tx\_Function() (UART0\_Port->Pin\_Control\_Register[UART0\_Tx\_Pin] |= \ |
| Pin\_Function\_Select(UART0\_Function)) |
|  |
| //Defines for UART Interrupt configuration |
| #define UART0\_TxE\_Interrupt\_Bit 7 |
| #define UART0\_Rx\_Interrupt\_Bit 5 |
| #define UART0\_Transmitter\_Enable\_Bit 3 |
| #define UART0\_Receiver\_Enable\_Bit 2 |
| #define UART0\_Control\_Register\_2 C2 |
| #define UART0\_Register\_Handler UART0 |
|  |
| //Macros for UART Interrupt support |
| #define Disable\_UART0\_Tx() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 &= \ |
| ~(1 << UART0\_Transmitter\_Enable\_Bit)) |
| #define Disable\_UART0\_Rx() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 &= \ |
| ~(1 << UART0\_Receiver\_Enable\_Bit)) |
| #define Enable\_UART0\_Tx() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 |= \ |
| (1 << UART0\_Transmitter\_Enable\_Bit)) |
| #define Enable\_UART0\_Rx() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 |= \ |
| (1 << UART0\_Receiver\_Enable\_Bit)) |
| #define Enable\_Rx\_Interrupt() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 |= \ |
| (1 << UART0\_Rx\_Interrupt\_Bit)) |
| #define Enable\_TxE\_Interrupt() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 |= \ |
| (1 << UART0\_TxE\_Interrupt\_Bit)) |
| #define Disable\_TxE\_Interrupt() (UART0\_Register\_Handler->UART0\_Control\_Register\_2 &= \ |
| ~(1 << UART0\_TxE\_Interrupt\_Bit)) |
|  |
| //Defines and macros for clock source selection and configuration for UART |
| #define System\_Option\_Register\_2 SOPT2 |
| #define PLL\_FLL\_Clock\_Select\_Bit 16 |
| #define Select\_PLL\_Clock\_Divby2() (System\_Integration\_Module->System\_Option\_Register\_2 |= \ |
| (1 << PLL\_FLL\_Clock\_Select\_Bit)) |
| #define Select\_FLL\_Clock() (System\_Integration\_Module->System\_Option\_Register\_2 &= \ |
| ~(1 << PLL\_FLL\_Clock\_Select\_Bit)) |
| #define UART0\_Clock\_Source\_Offset 26 |
| #define UART0\_Clock\_Souce\_FLL\_PLL 1 |
| #define UART0\_FLL\_PLL\_Clock\_Source() (System\_Integration\_Module->System\_Option\_Register\_2 |= \ |
| (UART0\_Clock\_Souce\_FLL\_PLL << UART0\_Clock\_Source\_Offset)) |
|  |
| //Defines and macros for BAUD rate |
| #define BAUD\_Rate 115200UL |
| #define System\_Clock 48000000UL |
| #define Oversampling 16 |
| #define BAUD\_Rate\_Setting\_Value (Word)(System\_Clock / (BAUD\_Rate \* Oversampling)) |
| #define BAUD\_Rate\_High\_Mask 0x1F00 |
| #define BAUD\_Rate\_Low\_Mask 0x00FF |
| #define BAUD\_Rate\_High\_Register BDH |
| #define BAUD\_Rate\_Low\_Register BDL |
| #define Set\_BAUD\_Rate\_High\_Register() (UART0\_Register\_Handler->BAUD\_Rate\_High\_Register = \ |
| (BAUD\_Rate\_Setting\_Value & BAUD\_Rate\_High\_Mask)) |
| #define Set\_BAUD\_Rate\_Low\_Register() (UART0\_Register\_Handler->BAUD\_Rate\_Low\_Register = \ |
| (BAUD\_Rate\_Setting\_Value & BAUD\_Rate\_Low\_Mask)) |
|  |
| //Defines and macros for oversampling |
| #define UART0\_Control\_Register\_4 C4 |
| #define Oversampling\_16 0x0F |
| #define Set\_Oversampling() (UART0\_Register\_Handler->UART0\_Control\_Register\_4 = \ |
| Oversampling\_16) |
|  |
| //Defines and macros for polling UART functions |
| #define UART0\_Status\_Register\_1 S1 |
| #define Tx\_Data\_Register\_Empty\_Flag\_Bit 7 //1 means empty |
| #define Tx\_Data\_Transmission\_Complete\_Flag\_Bit 6 //1 means complete |
| #define UART0\_Tx\_Empty\_Flag\_Status() ((UART0\_Register\_Handler->UART0\_Status\_Register\_1) & \ |
| (1 << Tx\_Data\_Register\_Empty\_Flag\_Bit)) |
| #define UART0\_Wait\_for\_Tx\_Data\_Register() while((!UART0\_Tx\_Empty\_Flag\_Status())) |
|  |
| #define Rx\_Data\_Register\_Full\_Flag\_Bit 5 //1 means full |
| #define UART0\_Wait\_for\_Rx\_Data\_Register() while(!((UART0\_Register\_Handler->UART0\_Status\_Register\_1) & \ |
| (1 << Rx\_Data\_Register\_Full\_Flag\_Bit))) |
|  |
| //Defines and macros for interrupt UART functions |
| #define UART0\_Data\_Register D |
| #define UART0\_Tx\_Data(x) (UART0\_Register\_Handler->UART0\_Data\_Register = x) |
| #define UART0\_Rx\_Data(addr) (\*addr = UART0\_Register\_Handler->UART0\_Data\_Register) |
|  |
| #define UART0\_Rx\_Interrupt() (UART0\_Register\_Handler->UART0\_Status\_Register\_1 & \ |
| (1 << Rx\_Data\_Register\_Full\_Flag\_Bit)) |
|  |
| #define UART0\_TxE\_Interrupt() (UART0\_Register\_Handler->UART0\_Status\_Register\_1 & \ |
| (1 << Tx\_Data\_Register\_Empty\_Flag\_Bit)) |
|  |
| //Currently only 2 modes - can be increased easily in future |
| typedef enum |
| { |
| FGETS\_Operation, |
| Normal\_Operation |
| }UART0\_Operation\_Type; |
|  |
| //Function initializations |
| void Custom\_UART0\_Init(void); |
| void ASCII\_Counter2(void); |
| void Custom\_UART0\_Tx\_Byte(Byte data); |
| void Custom\_UART0\_Tx\_String(char \*array); |
| void Custom\_UART0\_Rx\_Byte(volatile Byte \*address); |
| void UART0\_IRQHandler(void); |
|  |
| //Variables |
| extern UART0\_Operation\_Type State; |
| extern volatile Byte UART0\_Byte; |
|  |
| #endif /\* CUSTOM\_INCLUDES\_CUSTOM\_UART\_H\_ \*/ |

**board.c**

|  |
| --- |
| #include "board.h" |
| #include "fsl\_clock\_manager.h" |
| #include "fsl\_smc\_hal.h" |
| #include "fsl\_debug\_console.h" |
| #include "pin\_mux.h" |
|  |
| /\* Configuration for enter VLPR mode. Core clock = 4MHz. \*/ |
| const clock\_manager\_user\_config\_t g\_defaultClockConfigVlpr = |
| { |
| .mcgConfig = |
| { |
| .mcg\_mode = kMcgModeBLPI, // Work in BLPI mode. |
| .irclkEnable = true, // MCGIRCLK enable. |
| .irclkEnableInStop = false, // MCGIRCLK disable in STOP mode. |
| .ircs = kMcgIrcFast, // Select IRC4M. |
| .fcrdiv = 0U, // FCRDIV is 0. |
|  |
| .frdiv = 0U, |
| .drs = kMcgDcoRangeSelLow, // Low frequency range |
| .dmx32 = kMcgDmx32Default, // DCO has a default range of 25% |
|  |
| .pll0EnableInFllMode = false, // PLL0 disable |
| .pll0EnableInStop = false, // PLL0 disalbe in STOP mode |
| .prdiv0 = 0U, |
| .vdiv0 = 0U, |
| }, |
| .simConfig = |
| { |
| .pllFllSel = kClockPllFllSelFll, // PLLFLLSEL select FLL. |
| .er32kSrc = kClockEr32kSrcLpo, // ERCLK32K selection, use LPO. |
| .outdiv1 = 0U, |
| .outdiv4 = 4U, |
| }, |
| .oscerConfig = |
| { |
| .enable = true, // OSCERCLK enable. |
| .enableInStop = false, // OSCERCLK disable in STOP mode. |
| } |
| }; |
|  |
| /\* Configuration for enter RUN mode. Core clock = 48MHz. \*/ |
| const clock\_manager\_user\_config\_t g\_defaultClockConfigRun = |
| { |
| .mcgConfig = |
| { |
| .mcg\_mode = kMcgModePEE, // Work in PEE mode. |
| .irclkEnable = true, // MCGIRCLK enable. |
| .irclkEnableInStop = false, // MCGIRCLK disable in STOP mode. |
| .ircs = kMcgIrcSlow, // Select IRC32k. |
| .fcrdiv = 0U, // FCRDIV is 0. |
|  |
| .frdiv = 3U, |
| .drs = kMcgDcoRangeSelLow, // Low frequency range |
| .dmx32 = kMcgDmx32Default, // DCO has a default range of 25% |
|  |
| .pll0EnableInFllMode = false, // PLL0 disable |
| .pll0EnableInStop = false, // PLL0 disalbe in STOP mode |
| .prdiv0 = 0x1U, |
| .vdiv0 = 0x0U, |
| }, |
| .simConfig = |
| { |
| .pllFllSel = kClockPllFllSelPll, // PLLFLLSEL select PLL. |
| .er32kSrc = kClockEr32kSrcLpo, // ERCLK32K selection, use LPO. |
| .outdiv1 = 1U, |
| .outdiv4 = 3U, |
| }, |
| .oscerConfig = |
| { |
| .enable = true, // OSCERCLK enable. |
| .enableInStop = false, // OSCERCLK disable in STOP mode. |
| } |
| }; |
|  |
| /\* Function to initialize OSC0 base on board configuration. \*/ |
| void BOARD\_InitOsc0(void) |
| { |
| // OSC0 configuration. |
| osc\_user\_config\_t osc0Config = |
| { |
| .freq = OSC0\_XTAL\_FREQ, |
| .hgo = MCG\_HGO0, |
| .range = MCG\_RANGE0, |
| .erefs = MCG\_EREFS0, |
| .enableCapacitor2p = OSC0\_SC2P\_ENABLE\_CONFIG, |
| .enableCapacitor4p = OSC0\_SC4P\_ENABLE\_CONFIG, |
| .enableCapacitor8p = OSC0\_SC8P\_ENABLE\_CONFIG, |
| .enableCapacitor16p = OSC0\_SC16P\_ENABLE\_CONFIG, |
| }; |
|  |
| CLOCK\_SYS\_OscInit(0U, &osc0Config); |
| } |
|  |
| /\* Function to initialize RTC external clock base on board configuration. \*/ |
| void BOARD\_InitRtcOsc(void) |
| { |
|  |
| } |
|  |
| static void CLOCK\_SetBootConfig(clock\_manager\_user\_config\_t const\* config) |
| { |
| CLOCK\_SYS\_SetSimConfigration(&config->simConfig); |
|  |
| CLOCK\_SYS\_SetOscerConfigration(0, &config->oscerConfig); |
|  |
| #if (CLOCK\_INIT\_CONFIG == CLOCK\_VLPR) |
| CLOCK\_SYS\_BootToBlpi(&config->mcgConfig); |
| #else |
| CLOCK\_SYS\_BootToPee(&config->mcgConfig); |
| #endif |
|  |
| SystemCoreClock = CORE\_CLOCK\_FREQ; |
| } |
|  |
| /\* Initialize clock. \*/ |
| void BOARD\_ClockInit(void) |
| { |
| /\* Set allowed power mode, allow all. \*/ |
| SMC\_HAL\_SetProtection(SMC, kAllowPowerModeAll); |
|  |
| /\* Setup board clock source. \*/ |
| // Setup OSC0 if used. |
| // Configure OSC0 pin mux. |
| PORT\_HAL\_SetMuxMode(EXTAL0\_PORT, EXTAL0\_PIN, EXTAL0\_PINMUX); |
| PORT\_HAL\_SetMuxMode(XTAL0\_PORT, XTAL0\_PIN, XTAL0\_PINMUX); |
| BOARD\_InitOsc0(); |
|  |
| /\* Set system clock configuration. \*/ |
| #if (CLOCK\_INIT\_CONFIG == CLOCK\_VLPR) |
| CLOCK\_SetBootConfig(&g\_defaultClockConfigVlpr); |
| #else |
| CLOCK\_SetBootConfig(&g\_defaultClockConfigRun); |
| #endif |
| } |
|  |
| void dbg\_uart\_init(void) |
| { |
| configure\_lpsci\_pins(BOARD\_DEBUG\_UART\_INSTANCE); |
|  |
| // Select different clock source for LPSCI. \*/ |
| #if (CLOCK\_INIT\_CONFIG == CLOCK\_VLPR) |
| CLOCK\_SYS\_SetLpsciSrc(BOARD\_DEBUG\_UART\_INSTANCE, kClockLpsciSrcMcgIrClk); |
| #else |
| CLOCK\_SYS\_SetLpsciSrc(BOARD\_DEBUG\_UART\_INSTANCE, kClockLpsciSrcPllFllSel); |
| #endif |
|  |
| DbgConsole\_Init(BOARD\_DEBUG\_UART\_INSTANCE, BOARD\_DEBUG\_UART\_BAUD, kDebugConsoleLPSCI); |
| } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* @name usb\_device\_board\_init |
| \* |
| \* @brief This function is to handle board-specified initialization |
| \* |
| \* @param controller\_id: refer to CONTROLLER\_INDEX defined in usb\_misc.h |
| \* "0" stands for USB\_CONTROLLER\_KHCI\_0. |
| \* @return status |
| \* 0 : successful |
| \* 1 : failed |
| \*\* |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| uint8\_t usb\_device\_board\_init(uint8\_t controller\_id) |
| { |
| int8\_t ret = 0; |
|  |
| if (0 == controller\_id) |
| { |
| /\* TO DO \*/ |
| /\*add board initialization code if have\*/ |
| } |
| else |
| { |
| ret = 1; |
| } |
|  |
| return ret; |
|  |
| } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* |
| \* @name usb\_host\_board\_init |
| \* |
| \* @brief This function is to handle board-specified initialization |
| \* |
| \* @param controller\_id: refer to CONTROLLER\_INDEX defined in usb\_misc.h |
| \* "0" stands for USB\_CONTROLLER\_KHCI\_0. |
| \* @return status |
| \* 0 : successful |
| \* 1 : failed |
| \*\* |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| uint8\_t usb\_host\_board\_init(uint8\_t controller\_id) |
| { |
| int8\_t ret = 0; |
| /\*"0" stands for USB\_CONTROLLER\_KHCI\_0 \*/ |
| if (0 == controller\_id) |
| { |
|  |
| } |
| else |
| { |
| ret = 1; |
| } |
|  |
| return ret; |
|  |
|  |
| } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* EOF |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

**board.h**

|  |
| --- |
| #if !defined(\_\_BOARD\_H\_\_) |
| #define \_\_BOARD\_H\_\_ |
|  |
| #include <stdint.h> |
| #include "pin\_mux.h" |
| #include "gpio\_pins.h" |
|  |
| /\* The board name \*/ |
| #define BOARD\_NAME "FRDM-KL25Z" |
|  |
| #define CLOCK\_VLPR 1U |
| #define CLOCK\_RUN 2U |
| #define CLOCK\_NUMBER\_OF\_CONFIGURATIONS 3U |
|  |
| #ifndef CLOCK\_INIT\_CONFIG |
| #define CLOCK\_INIT\_CONFIG CLOCK\_RUN |
| #endif |
|  |
| #if (CLOCK\_INIT\_CONFIG == CLOCK\_RUN) |
| #define CORE\_CLOCK\_FREQ 48000000U |
| #else |
| #define CORE\_CLOCK\_FREQ 4000000U |
| #endif |
|  |
| /\* OSC0 configuration. \*/ |
| #define OSC0\_XTAL\_FREQ 8000000U |
| #define OSC0\_SC2P\_ENABLE\_CONFIG false |
| #define OSC0\_SC4P\_ENABLE\_CONFIG false |
| #define OSC0\_SC8P\_ENABLE\_CONFIG false |
| #define OSC0\_SC16P\_ENABLE\_CONFIG false |
| #define MCG\_HGO0 kOscGainLow |
| #define MCG\_RANGE0 kOscRangeVeryHigh |
| #define MCG\_EREFS0 kOscSrcOsc |
|  |
| /\* EXTAL0 PTA18 \*/ |
| #define EXTAL0\_PORT PORTA |
| #define EXTAL0\_PIN 18 |
| #define EXTAL0\_PINMUX kPortPinDisabled |
|  |
| /\* XTAL0 PTA19 \*/ |
| #define XTAL0\_PORT PORTA |
| #define XTAL0\_PIN 19 |
| #define XTAL0\_PINMUX kPortPinDisabled |
|  |
| /\* The UART to use for debug messages. \*/ |
| #ifndef BOARD\_DEBUG\_UART\_INSTANCE |
| #define BOARD\_DEBUG\_UART\_INSTANCE 0 |
| #define BOARD\_DEBUG\_UART\_BASEADDR UART0 |
| #endif |
| #ifndef BOARD\_DEBUG\_UART\_BAUD |
| #define BOARD\_DEBUG\_UART\_BAUD 115200 |
| #endif |
|  |
| /\* This define to use for power manager demo \*/ |
| #define BOARD\_LOW\_POWER\_UART\_BAUD 9600 |
|  |
| #define BOARD\_USE\_LPSCI |
| #define PM\_DBG\_UART\_IRQ\_HANDLER UART0\_IRQHandler |
| #define PM\_DBG\_UART\_IRQn UART0\_IRQn |
|  |
| /\* Define print statement to inform user which switch to press for |
| \* power\_manager\_hal\_demo and power\_manager\_rtos\_demo |
| \*/ |
| #define PRINT\_LLWU\_SW\_NUM \ |
| PRINTF(" PTD6 J2-17 to VSS J9-14") |
|  |
| /\* Defines the llwu pin number for board switch which is used in power\_manager\_demo. \*/ |
| #define BOARD\_SW\_HAS\_LLWU\_PIN 1 |
| #define BOARD\_SW\_LLWU\_EXT\_PIN 15 |
| /\* Switch port base address and IRQ handler name. Used by power\_manager\_demo \*/ |
| #define BOARD\_SW\_LLWU\_PIN 6 |
| #define BOARD\_SW\_LLWU\_BASE PORTD |
| #define BOARD\_SW\_LLWU\_IRQ\_HANDLER PORTD\_IRQHandler |
| #define BOARD\_SW\_LLWU\_IRQ\_NUM PORTD\_IRQn |
|  |
| #define BOARD\_I2C\_GPIO\_SCL GPIO\_MAKE\_PIN(GPIOE\_IDX, 24) |
| #define BOARD\_I2C\_GPIO\_SDA GPIO\_MAKE\_PIN(GPIOE\_IDX, 25) |
|  |
| #define HWADC\_INSTANCE 0 |
| #define ADC\_IRQ\_N ADC0\_IRQn |
|  |
| /\* The instances of peripherals used for dac\_adc\_demo \*/ |
| #define BOARD\_DAC\_DEMO\_DAC\_INSTANCE 0U |
| #define BOARD\_DAC\_DEMO\_ADC\_INSTANCE 0U |
| #define BOARD\_DAC\_DEMO\_ADC\_CHANNEL 0U |
|  |
| /\* The i2c instance used for i2c DAC demo \*/ |
| #define BOARD\_DAC\_I2C\_INSTANCE 1 |
|  |
| /\* The i2c instance used for i2c connection by default \*/ |
| #define BOARD\_I2C\_INSTANCE 1 |
|  |
| /\* The spi instance used for spi example \*/ |
| #define BOARD\_SPI\_INSTANCE 0 |
|  |
| /\* The TPM instance/channel used for board \*/ |
| #define BOARD\_TPM\_INSTANCE 0 |
| #define BOARD\_TPM\_CHANNEL 1 |
|  |
| /\* The bubble level demo information \*/ |
| #define BOARD\_BUBBLE\_TPM\_INSTANCE 2 |
| #define BOARD\_TPM\_X\_CHANNEL 0 |
| #define BOARD\_TPM\_Y\_CHANNEL 1 |
| #define BOARD\_MMA8451\_ADDR 0x1D |
| #define BOARD\_ACCEL\_ADDR BOARD\_MMA8451\_ADDR |
| #define BOARD\_ACCEL\_BAUDRATE 100 |
| #define BOARD\_ACCEL\_I2C\_INSTANCE 0 |
|  |
| /\* board led color mapping \*/ |
| #define BOARD\_GPIO\_LED\_BLUE kGpioLED3 |
| #define BOARD\_GPIO\_LED\_RED kGpioLED2 |
| #define BOARD\_GPIO\_LED\_GREEN kGpioLED1 |
|  |
| #define BOARD\_TSI\_ELECTRODE\_CNT 2 |
| #define BOARD\_TSI\_ELECTRODE\_1 9 |
| #define BOARD\_TSI\_ELECTRODE\_2 10 |
|  |
| #define LED1\_EN (GPIO\_DRV\_OutputPinInit(&ledPins[0])) /\*!< Enable target LED1 \*/ |
| #define LED2\_EN (GPIO\_DRV\_OutputPinInit(&ledPins[1])) /\*!< Enable target LED2 \*/ |
| #define LED3\_EN (GPIO\_DRV\_OutputPinInit(&ledPins[2])) /\*!< Enable target LED3 \*/ |
|  |
| #define LED1\_DIS (PORT\_HAL\_SetMuxMode(PORTB, 19, kPortMuxAsGpio)) /\*!< Enable target LED1 \*/ |
| #define LED2\_DIS (PORT\_HAL\_SetMuxMode(PORTB, 18, kPortMuxAsGpio)) /\*!< Enable target LED2 \*/ |
| #define LED3\_DIS (PORT\_HAL\_SetMuxMode(PORTD, 1, kPortMuxAsGpio)) /\*!< Enable target LED3 \*/ |
|  |
| #define LED1\_OFF (GPIO\_DRV\_WritePinOutput(ledPins[0].pinName, 1)) /\*!< Turn off target LED1 \*/ |
| #define LED2\_OFF (GPIO\_DRV\_WritePinOutput(ledPins[1].pinName, 1)) /\*!< Turn off target LED2 \*/ |
| #define LED3\_OFF (GPIO\_DRV\_WritePinOutput(ledPins[2].pinName, 1)) /\*!< Turn off target LED3 \*/ |
|  |
| #define LED1\_ON (GPIO\_DRV\_WritePinOutput(ledPins[0].pinName, 0)) /\*!< Turn on target LED1 \*/ |
| #define LED2\_ON (GPIO\_DRV\_WritePinOutput(ledPins[1].pinName, 0)) /\*!< Turn on target LED2 \*/ |
| #define LED3\_ON (GPIO\_DRV\_WritePinOutput(ledPins[2].pinName, 0)) /\*!< Turn on target LED3 \*/ |
|  |
| #define LED1\_TOGGLE (GPIO\_DRV\_TogglePinOutput(ledPins[0].pinName)) /\*!< Toggle on target LED1 \*/ |
| #define LED2\_TOGGLE (GPIO\_DRV\_TogglePinOutput(ledPins[1].pinName)) /\*!< Toggle on target LED2 \*/ |
| #define LED3\_TOGGLE (GPIO\_DRV\_TogglePinOutput(ledPins[2].pinName)) /\*!< Toggle on target LED3 \*/ |
|  |
| #define BOARD\_HAS\_ONLY\_MULTIPLE\_COLOR\_LED |
| #define LED\_RTOS\_EN LED1\_EN |
| #define LED\_RTOS\_TOGGLE LED1\_TOGGLE |
|  |
| /\* The CMP instance used for board. \*/ |
| #define BOARD\_CMP\_INSTANCE 0 |
| /\* The CMP channel used for board. \*/ |
| #define BOARD\_CMP\_CHANNEL 0 |
|  |
| /\* The rtc instance used for rtc\_func \*/ |
| #define BOARD\_RTC\_FUNC\_INSTANCE 0 |
|  |
| #if defined(\_\_cplusplus) |
| extern "C" { |
| #endif /\* \_\_cplusplus \*/ |
|  |
| void hardware\_init(void); |
| void dbg\_uart\_init(void); |
| /\*This function to used for power manager demo\*/ |
| void disable\_unused\_pins(void); |
| void enable\_unused\_pins(void); |
| /\* Function to initialize clock base on board configuration. \*/ |
| void BOARD\_ClockInit(void); |
|  |
| /\* Function to initialize OSC0 base on board configuration. \*/ |
| void BOARD\_InitOsc0(void); |
|  |
| /\* Function to initialize RTC external clock base on board configuration. \*/ |
| void BOARD\_InitRtcOsc(void); |
|  |
| /\*Function to handle board-specified initialization\*/ |
| uint8\_t usb\_device\_board\_init(uint8\_t controller\_id); |
| /\*Function to handle board-specified initialization\*/ |
| uint8\_t usb\_host\_board\_init(uint8\_t controller\_id); |
|  |
| #if defined(\_\_cplusplus) |
| } |
| #endif /\* \_\_cplusplus \*/ |
|  |
| #endif /\* \_\_BOARD\_H\_\_ \*/ |

**gpio\_pins.c**

|  |
| --- |
| #include <stdint.h> |
| #include <stdlib.h> |
| #include "gpio\_pins.h" |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Definitions |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| gpio\_input\_pin\_user\_config\_t switchPins[] = { |
| { |
| .pinName = kGpioSW1, |
| .config.isPullEnable = true, |
| .config.isPassiveFilterEnabled = false, |
| .config.interrupt = kPortIntDisabled, |
| }, |
| { |
| .pinName = GPIO\_PINS\_OUT\_OF\_RANGE, |
| } |
| }; |
|  |
| /\* Declare Output GPIO pins \*/ |
| gpio\_output\_pin\_user\_config\_t ledPins[] = { |
| { |
| .pinName = kGpioLED1, |
| .config.outputLogic = 1, |
| .config.slewRate = kPortSlowSlewRate, |
| .config.driveStrength = kPortLowDriveStrength, |
| }, |
| { |
| .pinName = kGpioLED2, |
| .config.outputLogic = 1, |
| .config.slewRate = kPortSlowSlewRate, |
| .config.driveStrength = kPortLowDriveStrength, |
| }, |
| { |
| .pinName = kGpioLED3, |
| .config.outputLogic = 1, |
| .config.slewRate = kPortSlowSlewRate, |
| .config.driveStrength = kPortLowDriveStrength, |
| }, |
| { |
| .pinName = GPIO\_PINS\_OUT\_OF\_RANGE, |
| } |
| }; |
|  |
|  |
| /\* END gpio\_pins. \*/ |
| /\*! |
| \*\* @} |
| \*/ |
| /\* |
| \*\* ################################################################### |
| \*\* |
| \*\* This file was created by Processor Expert 10.5 [05.21] |
| \*\* for the Freescale Kinetis series of microcontrollers. |
| \*\* |
| \*\* ################################################################### |
| \*/ |

**gpio\_pins.h**

|  |
| --- |
| #ifndef \_\_FSL\_GPIO\_PINS\_H\_\_ |
| #define \_\_FSL\_GPIO\_PINS\_H\_\_ |
|  |
| #include "fsl\_gpio\_driver.h" |
|  |
| /\*! @file \*/ |
| /\*!\*/ |
| /\*! This file contains gpio pin definitions used by gpio peripheral driver.\*/ |
| /\*! The enums in \_gpio\_pins map to the real gpio pin numbers defined in\*/ |
| /\*! gpioPinLookupTable. And this might be different in different board.\*/ |
|  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Definitions |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| /\*! @brief gpio pin names.\*/ |
| /\*!\*/ |
| /\*! This should be defined according to board setting.\*/ |
| enum \_gpio\_pins |
| { |
| kGpioLED1 = GPIO\_MAKE\_PIN(GPIOB\_IDX, 19), /\* FRDM-KL25Z4 Green LED \*/ |
| kGpioLED2 = GPIO\_MAKE\_PIN(GPIOB\_IDX, 18), /\* FRDM-KL25Z4 Red LED \*/ |
| kGpioLED3 = GPIO\_MAKE\_PIN(GPIOD\_IDX, 1), /\* FRDM-KL25Z4 Blue LED \*/ |
| kGpioSW1 = GPIO\_MAKE\_PIN(GPIOD\_IDX, 6), /\* FRDM-KL25Z4 power manager \*/ |
|  |
|  |
| }; |
|  |
| extern gpio\_input\_pin\_user\_config\_t switchPins[]; |
| extern gpio\_output\_pin\_user\_config\_t ledPins[]; |
|  |
| #endif /\* \_\_FSL\_GPIO\_PINS\_H\_\_ \*/ |
|  |
| /\*! |
| \*\* @} |
| \*/ |
| /\* |
| \*\* ################################################################### |
| \*\* |
| \*\* This file was created by Processor Expert 10.5 [05.21] |
| \*\* for the Freescale Kinetis series of microcontrollers. |
| \*\* |
| \*\* ################################################################### |
| \*/ |

**hardware\_init.c**

|  |
| --- |
| #include "board.h" |
| #include "pin\_mux.h" |
| #include "fsl\_clock\_manager.h" |
| #include "fsl\_debug\_console.h" |
|  |
| void hardware\_init(void) { |
|  |
| /\* enable clock for PORTs \*/ |
| CLOCK\_SYS\_EnablePortClock(PORTA\_IDX); |
| CLOCK\_SYS\_EnablePortClock(PORTB\_IDX); |
| CLOCK\_SYS\_EnablePortClock(PORTC\_IDX); |
| CLOCK\_SYS\_EnablePortClock(PORTD\_IDX); |
| CLOCK\_SYS\_EnablePortClock(PORTE\_IDX); |
|  |
| /\* Init board clock \*/ |
| BOARD\_ClockInit(); |
| // dbg\_uart\_init(); |
| // configure\_dac\_pins(0U); |
| } |
|  |
| /\*! |
| \*\* @} |
| \*/ |
| /\* |
| \*\* ################################################################### |
| \*\* |
| \*\* This file was created by Processor Expert 10.4 [05.10] |
| \*\* for the Freescale Kinetis series of microcontrollers. |
| \*\* |
| \*\* ################################################################### |
| \*/ |

**pin\_mux.c**

|  |
| --- |
| #include "fsl\_device\_registers.h" |
| #include "fsl\_port\_hal.h" |
| #include "pin\_mux.h" |
|  |
| void configure\_gpio\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case PORTA\_IDX: /\* PTA \*/ |
| /\* PORTA\_PCR14 MMA8451 - INT1 \*/ |
| PORT\_HAL\_SetMuxMode(PORTA,14u,kPortMuxAsGpio); |
| /\* PORTA\_PCR15 MMA8451 - INT2 \*/ |
| PORT\_HAL\_SetMuxMode(PORTA,15u,kPortMuxAsGpio); |
| break; |
| case PORTB\_IDX: /\* PTB \*/ |
| /\* PORTB\_PCR19 LED1 - Green \*/ |
| PORT\_HAL\_SetMuxMode(PORTB,19u,kPortMuxAsGpio); |
| /\* PORTB\_PCR18 LED2 - Red \*/ |
| PORT\_HAL\_SetMuxMode(PORTB,18u,kPortMuxAsGpio); |
| break; |
| case PORTD\_IDX: /\* PTD \*/ |
| /\* PORTD\_PCR1 LED3 - Blue \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,1u,kPortMuxAsGpio); |
| /\* PORTD\_PCR6 LLWU\_P15 SW1 - Power Manager demo \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,6u,kPortMuxAsGpio); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_i2c\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case I2C0\_IDX: /\* I2C0 \*/ |
| /\* PORTB\_PCR2 \*/ |
| PORT\_HAL\_SetMuxMode(PORTB,2u,kPortMuxAlt2); |
| PORT\_HAL\_SetPullCmd(PORTB,2u,true); |
| PORT\_HAL\_SetPassiveFilterCmd(PORTB,2u,false); |
| /\* PORTB\_PCR3 \*/ |
| PORT\_HAL\_SetMuxMode(PORTB,3u,kPortMuxAlt2); |
| PORT\_HAL\_SetPullCmd(PORTB,3u,true); |
| PORT\_HAL\_SetPassiveFilterCmd(PORTB,3u,false); |
| break; |
| case I2C1\_IDX: /\* I2C1 \*/ |
| /\* PORTC\_PCR1 \*/ |
| PORT\_HAL\_SetMuxMode(PORTC,1u,kPortMuxAlt2); |
| PORT\_HAL\_SetPullCmd(PORTC,1u,true); |
| PORT\_HAL\_SetPassiveFilterCmd(PORTC,1u,false); |
| /\* PORTC\_PCR2 \*/ |
| PORT\_HAL\_SetMuxMode(PORTC,2u,kPortMuxAlt2); |
| PORT\_HAL\_SetPullCmd(PORTC,2u,true); |
| PORT\_HAL\_SetPassiveFilterCmd(PORTC,2u,false); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_rtc\_pins(uint32\_t instance) |
| { |
| /\* PORTE\_PCR0 \*/ |
| PORT\_HAL\_SetMuxMode(PORTE,0u,kPortMuxAlt4); |
| } |
|  |
| void configure\_lpsci\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case UART0\_IDX: /\* LPSCI0 \*/ |
| /\* PORTA\_PCR1 \*/ |
| PORT\_HAL\_SetMuxMode(PORTA,1u,kPortMuxAlt2); |
| /\* PORTA\_PCR2 \*/ |
| PORT\_HAL\_SetMuxMode(PORTA,2u,kPortMuxAlt2); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_uart\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case UART1\_IDX: /\* UART1 \*/ |
| /\* PORTE\_PCR0 \*/ |
| PORT\_HAL\_SetMuxMode(PORTE,0u,kPortMuxAlt3); |
| /\* PORTE\_PCR1 \*/ |
| PORT\_HAL\_SetMuxMode(PORTE,1u,kPortMuxAlt3); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| /\* Set-up TSI pins for on board electrodes \*/ |
| void configure\_tsi\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case TSI0\_IDX: /\* TSI0 \*/ |
| /\* PORTB\_PCR16 \*/ |
| PORT\_HAL\_SetMuxMode(PORTB,16u,kPortPinDisabled); |
| /\* PORTB\_PCR17 \*/ |
| PORT\_HAL\_SetMuxMode(PORTB,17u,kPortPinDisabled); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_spi\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case SPI0\_IDX: /\* SPI0 \*/ |
| /\* PORTC\_PCR6 \*/ |
| PORT\_HAL\_SetMuxMode(PORTC,6u,kPortMuxAlt2); /\* MOSI \*/ |
| /\* PORTC\_PCR7 \*/ |
| PORT\_HAL\_SetMuxMode(PORTC,7u,kPortMuxAlt2); /\* MISO \*/ |
| /\* PORTC\_PCR5 \*/ |
| PORT\_HAL\_SetMuxMode(PORTC,5u,kPortMuxAlt2); /\* SCK \*/ |
| /\* PORTC\_PCR4 \*/ |
| PORT\_HAL\_SetMuxMode(PORTC,4u,kPortMuxAlt2); /\* PCS0 \*/ |
| break; |
| case SPI1\_IDX: /\* SPI1 \*/ |
| /\* PORTD\_PCR6 \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,6u,kPortMuxAlt2); /\* MOSI \*/ |
| /\* PORTD\_PCR7 \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,7u,kPortMuxAlt2); /\* MISO \*/ |
| /\* PORTD\_PCR5 \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,5u,kPortMuxAlt2); /\* SCK \*/ |
| /\* PORTD\_PCR4 \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,4u,kPortMuxAlt2); /\* PCS0 \*/ |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_tpm\_pins(uint32\_t instance) |
| { |
| switch(instance) { |
| case TPM0\_IDX: /\* TPM0 \*/ |
| /\* PTD\_PCR1 TPM0 channel 1 \*/ |
| PORT\_HAL\_SetMuxMode(PORTD,1u,kPortMuxAlt4); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_cmp\_pins(uint32\_t instance) |
| { |
| switch (instance) { |
| case CMP0\_IDX: |
| PORT\_HAL\_SetMuxMode(PORTC,6u,kPortPinDisabled); /\* PTC6 - CMP0\_IN0. \*/ |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| void configure\_dac\_pins(uint32\_t instance) |
| { |
| switch (instance) { |
| case DAC0\_IDX: |
| PORT\_HAL\_SetMuxMode(PORTE,30u,kPortPinDisabled); |
| break; |
| default: |
| break; |
| } |
| } |
|  |
| /\* END pin\_mux. \*/ |
| /\*! |
| \*\* @} |
| \*/ |
| /\* |
| \*\* ################################################################### |
| \*\* |
| \*\* This file was created by Processor Expert 10.5 [05.21] |
| \*\* for the Freescale Kinetis series of microcontrollers. |
| \*\* |
| \*\* ################################################################### |
| \*/ |

**pin\_mux.h**

|  |
| --- |
| #ifndef pin\_mux\_H\_ |
| #define pin\_mux\_H\_ |
|  |
| /\* MODULE pin\_mux. \*/ |
|  |
| /\* |
| \*\* =================================================================== |
| \*\* Method : configure\_gpio\_pins (component PinSettings) |
| \*/ |
| /\*! |
| \*\* @brief |
| \*\* GPIO method sets registers according routing settings. Call |
| \*\* this method code to route desired pins into: |
| \*\* PTA, PTB, PTC, PTD, PTE |
| \*\* peripherals. |
| \*\* @param |
| \*\* uint32\_t instance - GPIO instance number 0..4 |
| \*/ |
| /\* ===================================================================\*/ |
| void configure\_gpio\_pins(uint32\_t instance); |
| /\* |
| \*\* =================================================================== |
| \*\* Method : configure\_i2c\_pins(component PinSettings) |
| \*/ |
| /\*! |
| \*\* @brief |
| \*\* I2C method sets registers according routing settings. Call |
| \*\* this method code to route desired pins into: |
| \*\* I2C0, I2C1, I2C2 |
| \*\* peripherals. |
| \*\* @param |
| \*\* uint32\_t instance - I2C instance number 0..2 |
| \*/ |
| /\* ===================================================================\*/ |
| void configure\_i2c\_pins(uint32\_t instance); |
| /\* |
| \*\* =================================================================== |
| \*\* Method : configure\_rtc\_pins(component PinSettings) |
| \*/ |
| /\*! |
| \*\* @brief |
| \*\* RTC method sets registers according routing settings. Call |
| \*\* this method code to route desired pins into RTC periphery. |
| \*\* @param |
| \*\* uint32\_t instance - RTC instance number (0 is expected) |
| \*/ |
| /\* ===================================================================\*/ |
|  |
| void configure\_rtc\_pins(uint32\_t instance); |
| /\* |
| \*\* =================================================================== |
| \*\* Method : configure\_uart\_pins(component PinSettings) |
| \*/ |
| /\*! |
| \*\* @brief |
| \*\* UART method sets registers according routing settings. Call |
| \*\* this method code to route desired pins into: |
| \*\* UART0, UART1, UART2, UART3, UART4, UART5 |
| \*\* peripherals. |
| \*\* @param |
| \*\* uint32\_t instance - UART instance number 0..5 |
| \*/ |
| /\* ===================================================================\*/ |
| void configure\_uart\_pins(uint32\_t instance); |
| void configure\_lpsci\_pins(uint32\_t instance); |
| void configure\_spi\_pins(uint32\_t instance); |
| void configure\_tpm\_pins(uint32\_t instance); |
| void configure\_cmp\_pins(uint32\_t instance); |
|  |
| /\* |
| \*\* =================================================================== |
| \*\* Method : configure\_tsi\_pins (component PinSettings) |
| \*/ |
| /\*! |
| \*\* @brief |
| \*\* TSI method sets registers according routing settings. Call |
| \*\* this method code to route desired pins into: |
| \*\* TSI0 |
| \*\* peripheral. |
| \*\* @param |
| \*\* uint32\_t instance - TSI instance number 0 |
| \*/ |
| /\* ===================================================================\*/ |
| void configure\_tsi\_pins(uint32\_t instance); |
|  |
| void configure\_dac\_pins(uint32\_t instance); |
| /\* END pin\_mux. \*/ |
| #endif /\* #ifndef \_\_pin\_mux\_H\_ \*/ |
| /\*! |
| \*\* @} |
| \*/ |
| /\* |
| \*\* ################################################################### |
| \*\* |
| \*\* This file was created by Processor Expert 10.5 [05.21] |
| \*\* for the Freescale Kinetis series of microcontrollers. |
| \*\* |
| \*\* ################################################################### |
| \*/ |