```c
/*
 * dma_driver.c
 *
 *  Created on: Dec 11, 2018
 *      Author: Dominic Doty
 */

/* HEADER */
#include "dma_driver.h"


/* STATIC FUNCTION DECLARATIONS */
static bool dma_bad_addr(volatile void* addr);
static bool dma_null_ptrs(dma_init_config* config);
static bool dma_mux_null_ptrs(dma_mux_config* config);


/* FUNCTION DEFINITIONS */

// DMA Initialization
dma_error dma_init(dma_init_config* config)
{
    // Initialize
    dma_error ret = DMA_ERROR_SUCCESS;

    if(dma_null_ptrs(config))
    {
        ret = DMA_ERROR_NULL_PTR;
    }
    else if(dma_bad_addr(config->src_addr) |
            dma_bad_addr(config->dest_addr))
    {
        ret = DMA_ERROR_BAD_ADDR;
    }
    else if(config->dma->DMA[config->channel].DSR_BCR & DMA_DSR_BCR_BSY_MASK)
    {
        ret = DMA_ERROR_BUSY;
    }
    else if(config->byte_count & ~0xFFFFF)
    {
        ret = DMA_ERROR_BYTE_COUNT;
    }
    else if(config->dma != DMA0)
    {
        ret = DMA_ERROR_UNKNOWN_DMA;
    }
    else
    {
        // Easy Read Address
        DMA_Type* dma = config->dma;

        // Clock Enable
        CLOCK_EnableClock(kCLOCK_Dma0);

        // Source Address
        dma->DMA[config->channel].SAR = (uint32_t)config->src_addr;

        // Destination Address
        dma->DMA[config->channel].DAR = (uint32_t)config->dest_addr;

        // Byte Count Register
        dma->DMA[config->channel].DSR_BCR = DMA_DSR_BCR_BCR(config->byte_count);

        // DMA Control Register
        dma->DMA[config->channel].DCR = (DMA_DCR_EINT(config->interrupt))    |
                                        (DMA_DCR_ERQ(config->peripheral_en))       |
                                        (DMA_DCR_CS(config->steal_cycles))           |
                                        (DMA_DCR_AA(config->auto_align))           |
                                        (DMA_DCR_EADREQ(config->async_en))           |
```

```
70                                  (DMA_DCR_SINC(config->src_inc))                |
71                                  (DMA_DCR_SSIZE(config->src_size))             |
72                                  (DMA_DCR_DINC(config->dest_inc))             |
73                                  (DMA_DCR_DSIZE(config->dest_size))            |
74                                  (DMA_DCR_START(config->start))                |
75                                  (DMA_DCR_SMOD(config->src_mod))               |
76                                  (DMA_DCR_DMOD(config->dest_mod))             |
77                                  (DMA_DCR_D_REQ(config->auto_disable_req))    |
78                                  (DMA_DCR_LINKCC(config->link_mode))           |
79                                  (DMA_DCR_LCH1(config->link_chan_1))           |
80                                  (DMA_DCR_LCH2(config->link_chan_2))    ;
81
82          if(config->interrupt)
83          {
84              NVIC_EnableIRQ(DMA0_IRQn);
85          }
86      }
87      return ret;
88  }
89
90  // Used to restart a DMA transfer on an already configured DMA Channel (resets peripheral_en)
91  void dma_transfer_restart(DMA_Type* dma, dma_channel channel, volatile void* buffer_ptr, uint32_t byte_count)
92  {
93      dma->DMA[channel].DAR = (uint32_t)buffer_ptr;
94      dma->DMA[channel].DSR_BCR |= DMA_DSR_BCR_BCR(byte_count);
95      dma->DMA[channel].DCR |= DMA_DCR_ERQ(true);
96  }
97
98  dma_error dma_mux_init(dma_mux_config* config)
99  {
100     // Initialize
101     dma_error ret = DMA_ERROR_SUCCESS;
102
103     if(dma_mux_null_ptrs(config))
104     {
105         ret = DMA_ERROR_NULL_PTR;
106     }
107     else if(config->dma_mux != DMAMUX0)
108     {
109         ret = DMA_ERROR_UNKNOWN_DMA;
110     }
111     else
112     {
113         // Enable Clock
114         CLOCK_EnableClock(kCLOCK_Dmamux0);
115
116         // Configure
117         config->dma_mux->CHCFG[config->channel] =     DMAMUX_CHCFG_ENBL(config->channel_enable)    |
118                                                     DMAMUX_CHCFG_TRIG(config->trigger_mode)        |
119                                                     DMAMUX_CHCFG_SOURCE(config->slot)            ;
120     }
121
122     return ret;
123 }
124
125 // Enable or Disable a Mux Channel
126 void dma_mux_channel_enable(DMAMUX_Type* dma_mux, dma_channel channel, bool enable)
127 {
128     if(enable)
129     {
130         dma_mux->CHCFG[channel] |= DMAMUX_CHCFG_ENBL_MASK;
131     }
132     else
133     {
134         dma_mux->CHCFG[channel] &= DMAMUX_CHCFG_ENBL_MASK;
135     }
136 }
137
138 /* STATIC FUNCTION DEFINITIONS */
139
```

```
// Check for NULL Pointers
static bool dma_null_ptrs(dma_init_config* config)
{
    bool ret = false;

    if(    (config == NULL)            |
        (config->dma == NULL)        |
        (config->src_addr == NULL)    |
        (config->dest_addr == NULL)    )
    {
        ret = true;
    }

    return ret;
}

// Determine if a supplied address is legit
static bool dma_bad_addr(volatile void* addr)
{
    bool ret = false;

    uint32_t masked_addr = (uint32_t)addr & 0xFFF00000;

    switch(masked_addr)
    {
    case 0x00000000:
    case 0x1FF00000:
    case 0x20000000:
    case 0x40000000:
        break;
    default:
        ret = true;
        break;
    }

    return ret;
}

// Check for NULL Pointers
static bool dma_mux_null_ptrs(dma_mux_config* config)
{
    bool ret = false;

    if(    (config == NULL)             |
        (config->dma_mux == NULL)    )
    {
        ret = true;
    }

    return ret;
}
```