GPIO

```
typedef struct {
    __IO uint8_t FIODIR[4];        /**< FIO direction register in byte-align */
       uint32_t RESERVED0[3];    /**< Reserved */
    __IO uint8_t FIOMASK[4];     /**< FIO mask register in byte-align */
    __IO uint8_t FIOPIN[4];        /**< FIO pin register in byte align */
    __IO uint8_t FIOSET[4];        /**< FIO set register in byte-align */
    __O  uint8_t FIOCLR[4];        /**< FIO clear register in byte-align */
} GPIO_Byte_TypeDef;

typedef struct {
    __IO uint16_t FIODIRL;        /**< FIO direction register lower halfword part */
    __IO uint16_t FIODIRU;        /**< FIO direction register upper halfword part */
       uint32_t RESERVED0[3];    /**< Reserved */
    __IO uint16_t FIOMASKL;        /**< FIO mask register lower halfword part */
    __IO uint16_t FIOMASKU;        /**< FIO mask register upper halfword part */
    __IO uint16_t FIOPINL;        /**< FIO pin register lower halfword part */
    __IO uint16_t FIOPINU;        /**< FIO pin register upper halfword part */
    __IO uint16_t FIOSETL;        /**< FIO set register lower halfword part */
    __IO uint16_t FIOSETU;        /**< FIO set register upper halfword part */
    __O  uint16_t FIOCLRL;        /**< FIO clear register lower halfword part */
    __O  uint16_t FIOCLRU;        /**< FIO clear register upper halfword part */
} GPIO_HalfWord_TypeDef;

/* GPIO style ------------------------------- */
void GPIO_SetDir(uint8_t portNum, uint32_t bitValue, uint8_t dir);
void GPIO_SetValue(uint8_t portNum, uint32_t bitValue);
void GPIO_ClearValue(uint8_t portNum, uint32_t bitValue);
uint32_t GPIO_ReadValue(uint8_t portNum);
void GPIO_IntCmd(uint8_t portNum, uint32_t bitValue, uint8_t edgeState);
FunctionalState GPIO_GetIntStatus(uint8_t portNum, uint32_t pinNum, uint8_t edgeState);
void GPIO_ClearInt(uint8_t portNum, uint32_t bitValue);

/* FIO (word-accessible) style ------------------------------- */
void FIO_SetDir(uint8_t portNum, uint32_t bitValue, uint8_t dir);
void FIO_SetValue(uint8_t portNum, uint32_t bitValue);
void FIO_ClearValue(uint8_t portNum, uint32_t bitValue);
uint32_t FIO_ReadValue(uint8_t portNum);
void FIO_SetMask(uint8_t portNum, uint32_t bitValue, uint8_t maskValue);
void FIO_IntCmd(uint8_t portNum, uint32_t bitValue, uint8_t edgeState);
FunctionalState FIO_GetIntStatus(uint8_t portNum, uint32_t pinNum, uint8_t edgeState);
void FIO_ClearInt(uint8_t portNum, uint32_t pinNum);

/* FIO (halfword-accessible) style ------------------------------- */
void FIO_HalfWordSetDir(uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue, uint8_t dir);
void FIO_HalfWordSetMask(uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue, uint8_t maskValue);
void FIO_HalfWordSetValue(uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue);
void FIO_HalfWordClearValue(uint8_t portNum, uint8_t halfwordNum, uint16_t bitValue);
uint16_t FIO_HalfWordReadValue(uint8_t portNum, uint8_t halfwordNum);

/* FIO (byte-accessible) style ------------------------------- */
void FIO_ByteSetDir(uint8_t portNum, uint8_t byteNum, uint8_t bitValue, uint8_t dir);
void FIO_ByteSetMask(uint8_t portNum, uint8_t byteNum, uint8_t bitValue, uint8_t maskValue);
void FIO_ByteSetValue(uint8_t portNum, uint8_t byteNum, uint8_t bitValue);
void FIO_ByteClearValue(uint8_t portNum, uint8_t byteNum, uint8_t bitValue);
uint8_t FIO_ByteReadValue(uint8_t portNum, uint8_t byteNum);
```

PINSEL - SYSTICK - EXTI

```c
typedef struct {
    uint8_t Portnum;    /**< Port Number, should be PINSEL_PORT_x,
                        where x should be in range from 0 to 4 */
    uint8_t Pinnum;        /**< Pin Number, should be PINSEL_PIN_x,
                        where x should be in range from 0 to 31 */
    uint8_t Funcnum;    /**< Function Number, should be PINSEL_FUNC_x,
                        where x should be in range from 0 to 3 */
    uint8_t Pinmode;    /**< Pin Mode, should be:
                        - PINSEL_PINMODE_PULLUP: Internal pull-up resistor
                        - PINSEL_PINMODE_TRISTATE: Tri-state
                        - PINSEL_PINMODE_PULLDOWN: Internal pull-down resistor */
    uint8_t OpenDrain;    /**< OpenDrain mode, should be:
                        - PINSEL_PINMODE_NORMAL: Pin is in the normal (not open drain) mode
                        - PINSEL_PINMODE_OPENDRAIN: Pin is in the open drain mode */
} PINSEL_CFG_Type;

void PINSEL_ConfigPin(PINSEL_CFG_Type *PinCfg);
void PINSEL_ConfigTraceFunc (FunctionalState NewState);
void PINSEL_SetI2C0Pins(uint8_t i2cPinMode, FunctionalState filterSlewRateEnable);

/**********************************************************************/
void SYSTICK_InternalInit(uint32_t time);
void SYSTICK_ExternalInit(uint32_t freq, uint32_t time);

void SYSTICK_Cmd(FunctionalState NewState);
void SYSTICK_IntCmd(FunctionalState NewState);
uint32_t SYSTICK_GetCurrentValue(void);
void SYSTICK_ClearCounterFlag(void);

/**********************************************************************/
typedef enum {
    EXTI_EINT0, /*!<  External interrupt 0, P2.10 */
    EXTI_EINT1, /*!<  External interrupt 1, P2.11 */
    EXTI_EINT2, /*!<  External interrupt 2, P2.12 */
    EXTI_EINT3  /*!<  External interrupt 3, P2.13 */
} EXTI_LINE_ENUM;

typedef enum {
    EXTI_MODE_LEVEL_SENSITIVE,  /*!< Level sensitivity is selected */
    EXTI_MODE_EDGE_SENSITIVE    /*!< Edge sensitivity is selected */
} EXTI_MODE_ENUM;

typedef enum {
    EXTI_POLARITY_LOW_ACTIVE_OR_FALLING_EDGE,
    EXTI_POLARITY_HIGH_ACTIVE_OR_RISING_EDGE
} EXTI_POLARITY_ENUM;

typedef struct {
    EXTI_LINE_ENUM EXTI_Line;
    EXTI_MODE_ENUM EXTI_Mode;
    EXTI_POLARITY_ENUM EXTI_polarity;
}EXTI_InitTypeDef;

void EXTI_Init(void);
void EXTI_DeInit(void);

void EXTI_Config(EXTI_InitTypeDef *EXTICfg);
void EXTI_SetMode(EXTI_LINE_ENUM EXTILine, EXTI_MODE_ENUM mode);
void EXTI_SetPolarity(EXTI_LINE_ENUM EXTILine, EXTI_POLARITY_ENUM polarity);
void EXTI_ClearEXTIFlag(EXTI_LINE_ENUM EXTILine);
```

TIMER

```
typedef enum {
    TIM_MR0_INT =0, /*!< interrupt for Match channel 0*/
    TIM_MR1_INT =1, /*!< interrupt for Match channel 1*/
    TIM_MR2_INT =2, /*!< interrupt for Match channel 2*/
    TIM_MR3_INT =3, /*!< interrupt for Match channel 3*/
    TIM_CR0_INT =4, /*!< interrupt for Capture channel 0*/
    TIM_CR1_INT =5, /*!< interrupt for Capture channel 1*/
}TIM_INT_TYPE;

typedef enum {
    TIM_TIMER_MODE = 0,             /*!< Timer mode */
    TIM_COUNTER_RISING_MODE,        /*!< Counter rising mode */
    TIM_COUNTER_FALLING_MODE,       /*!< Counter falling mode */
    TIM_COUNTER_ANY_MODE            /*!< Counter on both edges */
} TIM_MODE_OPT;

typedef enum {
    TIM_PRESCALE_TICKVAL = 0,       /*!< Prescale in absolute value */
    TIM_PRESCALE_USVAL              /*!< Prescale in microsecond value */
} TIM_PRESCALE_OPT;

typedef enum {
    TIM_COUNTER_INCAP0 = 0,         /*!< CAPn.0 input pin for TIMERn */
    TIM_COUNTER_INCAP1,             /*!< CAPn.1 input pin for TIMERn */
} TIM_COUNTER_INPUT_OPT;

typedef enum {
    TIM_EXTMATCH_NOTHING = 0,       /*!< Do nothing for external output pin if match */
    TIM_EXTMATCH_LOW,               /*!< Force external output pin to low if match */
    TIM_EXTMATCH_HIGH,              /*!< Force external output pin to high if match */
    TIM_EXTMATCH_TOGGLE             /*!< Toggle external output pin if match */
}TIM_EXTMATCH_OPT;

typedef enum {
    TIM_CAPTURE_NONE = 0,   /*!< No Capture */
    TIM_CAPTURE_RISING,     /*!< Rising capture mode */
    TIM_CAPTURE_FALLING,    /*!< Falling capture mode */
    TIM_CAPTURE_ANY         /*!< On both edges */
} TIM_CAP_MODE_OPT;

typedef struct
{
    uint8_t PrescaleOption;     /**< Timer Prescale option, should be:
                                    - TIM_PRESCALE_TICKVAL: Prescale in absolute value
                                    - TIM_PRESCALE_USVAL: Prescale in microsecond value
                                    */
    uint8_t Reserved[3];        /**< Reserved */
    uint32_t PrescaleValue;     /**< Prescale value */
} TIM_TIMERCFG_Type;

typedef struct {
    uint8_t CounterOption;      /**< Counter Option, should be:
                                - TIM_COUNTER_INCAP0: CAPn.0 input pin for TIMERn
                                - TIM_COUNTER_INCAP1: CAPn.1 input pin for TIMERn
                                */
    uint8_t CountInputSelect;
    uint8_t Reserved[2];
} TIM_COUNTERCFG_Type;

typedef struct {
    uint8_t MatchChannel;   /**< Match channel, should be in range
```

```
                            from 0..3 */
    uint8_t IntOnMatch;     /**< Interrupt On match, should be:
                            - ENABLE: Enable this function.
                            - DISABLE: Disable this function.
                            */
    uint8_t StopOnMatch;    /**< Stop On match, should be:
                            - ENABLE: Enable this function.
                            - DISABLE: Disable this function.
                            */
    uint8_t ResetOnMatch;   /**< Reset On match, should be:
                            - ENABLE: Enable this function.
                            - DISABLE: Disable this function.
                            */

    uint8_t ExtMatchOutputType; /**< External Match Output type, should be:
                             -   TIM_EXTMATCH_NOTHING:  Do nothing for external output pin if match
                             -   TIM_EXTMATCH_LOW:  Force external output pin to low if match
                             -   TIM_EXTMATCH_HIGH: Force external output pin to high if match
                             -   TIM_EXTMATCH_TOGGLE: Toggle external output pin if match.
                            */
    uint8_t Reserved[3];    /** Reserved */
    uint32_t MatchValue;    /** Match value */
} TIM_MATCHCFG_Type;

typedef struct {
    uint8_t CaptureChannel; /**< Capture channel, should be in range
                            from 0..1 */
    uint8_t RisingEdge;     /**< caption rising edge, should be:
                            - ENABLE: Enable rising edge.
                            - DISABLE: Disable this function.
                            */
    uint8_t FallingEdge;        /**< caption falling edge, should be:
                            - ENABLE: Enable falling edge.
                            - DISABLE: Disable this function.
                                */
    uint8_t IntOnCaption;   /**< Interrupt On caption, should be:
                            - ENABLE: Enable interrupt function.
                            - DISABLE: Disable this function.
                            */
} TIM_CAPTURECFG_Type;


/* Init/DeInit TIM functions -----------*/
void TIM_Init(LPC_TIM_TypeDef *TIMx, TIM_MODE_OPT TimerCounterMode, void *TIM_ConfigStruct);
void TIM_DeInit(LPC_TIM_TypeDef *TIMx);

/* TIM interrupt functions -------------*/
void TIM_ClearIntPending(LPC_TIM_TypeDef *TIMx, TIM_INT_TYPE IntFlag);
void TIM_ClearIntCapturePending(LPC_TIM_TypeDef *TIMx, TIM_INT_TYPE IntFlag);
FlagStatus TIM_GetIntStatus(LPC_TIM_TypeDef *TIMx, TIM_INT_TYPE IntFlag);
FlagStatus TIM_GetIntCaptureStatus(LPC_TIM_TypeDef *TIMx, TIM_INT_TYPE IntFlag);

/* TIM configuration functions --------*/
void TIM_ConfigStructInit(TIM_MODE_OPT TimerCounterMode, void *TIM_ConfigStruct);
void TIM_ConfigMatch(LPC_TIM_TypeDef *TIMx, TIM_MATCHCFG_Type *TIM_MatchConfigStruct);
void TIM_UpdateMatchValue(LPC_TIM_TypeDef *TIMx,uint8_t MatchChannel, uint32_t MatchValue);
void TIM_SetMatchExt(LPC_TIM_TypeDef *TIMx,TIM_EXTMATCH_OPT ext_match );
void TIM_ConfigCapture(LPC_TIM_TypeDef *TIMx, TIM_CAPTURECFG_Type *TIM_CaptureConfigStruct);
void TIM_Cmd(LPC_TIM_TypeDef *TIMx, FunctionalState NewState);

uint32_t TIM_GetCaptureValue(LPC_TIM_TypeDef *TIMx, TIM_COUNTER_INPUT_OPT CaptureChannel);
void TIM_ResetCounter(LPC_TIM_TypeDef *TIMx);
```

ADC - DAC

```
typedef enum {
    ADC_CHANNEL_0  = 0, /*!<  Channel 0 */
    ADC_CHANNEL_1,       /*!<  Channel 1 */
    ADC_CHANNEL_2,       /*!<  Channel 2 */
    ADC_CHANNEL_3,       /*!<  Channel 3 */
    ADC_CHANNEL_4,       /*!<  Channel 4 */
    ADC_CHANNEL_5,       /*!<  Channel 5 */
    ADC_CHANNEL_6,       /*!<  Channel 6 */
    ADC_CHANNEL_7        /*!<  Channel 7 */
}ADC_CHANNEL_SELECTION;

typedef enum {
    ADC_START_CONTINUOUS =0,    /*!< Continuous mode */
    ADC_START_NOW,                /*!< Start conversion now */
    ADC_START_ON_EINT0,            /*!< Start conversion when the edge selected
                                 * by bit 27 occurs on P2.10/EINT0 */
    ADC_START_ON_CAP01,            /*!< Start conversion when the edge selected
                                 * by bit 27 occurs on P1.27/CAP0.1 */
    ADC_START_ON_MAT01,            /*!< Start conversion when the edge selected
                                 * by bit 27 occurs on MAT0.1 */
    ADC_START_ON_MAT03,            /*!< Start conversion when the edge selected
                                 * by bit 27 occurs on MAT0.3 */
    ADC_START_ON_MAT10,            /*!< Start conversion when the edge selected
                                  * by bit 27 occurs on MAT1.0 */
    ADC_START_ON_MAT11            /*!< Start conversion when the edge selected
                                  * by bit 27 occurs on MAT1.1 */
} ADC_START_OPT;

typedef enum {
    ADC_START_ON_RISING = 0,    /*!< Start conversion on a rising edge
                                *on the selected CAP/MAT signal */
    ADC_START_ON_FALLING        /*!< Start conversion on a falling edge
                                *on the selected CAP/MAT signal */
} ADC_START_ON_EDGE_OPT;

typedef enum {
    ADC_ADINTEN0 = 0,        /*!< Interrupt channel 0 */
    ADC_ADINTEN1,            /*!< Interrupt channel 1 */
    ADC_ADINTEN2,            /*!< Interrupt channel 2 */
    ADC_ADINTEN3,            /*!< Interrupt channel 3 */
    ADC_ADINTEN4,            /*!< Interrupt channel 4 */
    ADC_ADINTEN5,            /*!< Interrupt channel 5 */
    ADC_ADINTEN6,            /*!< Interrupt channel 6 */
    ADC_ADINTEN7,            /*!< Interrupt channel 7 */
    ADC_ADGINTEN            /*!< Individual channel/global flag done generate an interrupt */
}ADC_TYPE_INT_OPT;

typedef enum {
    ADC_DATA_BURST = 0,        /*Burst bit*/
    ADC_DATA_DONE         /*Done bit*/
}ADC_DATA_STATUS;

/* Init/DeInit ADC peripheral ----------------*/
void ADC_Init(LPC_ADC_TypeDef *ADCx, uint32_t rate);
void ADC_DeInit(LPC_ADC_TypeDef *ADCx);

/* Enable/Disable ADC functions --------------*/
void ADC_BurstCmd(LPC_ADC_TypeDef *ADCx, FunctionalState NewState);
void ADC_PowerdownCmd(LPC_ADC_TypeDef *ADCx, FunctionalState NewState);
void ADC_StartCmd(LPC_ADC_TypeDef *ADCx, uint8_t start_mode);
void ADC_ChannelCmd (LPC_ADC_TypeDef *ADCx, uint8_t Channel, FunctionalState NewState);
```

```
/* Configure ADC functions -------------------*/
void ADC_EdgeStartConfig(LPC_ADC_TypeDef *ADCx, uint8_t EdgeOption);
void ADC_IntConfig (LPC_ADC_TypeDef *ADCx, ADC_TYPE_INT_OPT IntType, FunctionalState NewState);

/* Get ADC information functions -------------------*/
uint16_t ADC_ChannelGetData(LPC_ADC_TypeDef *ADCx, uint8_t channel);
FlagStatus ADC_ChannelGetStatus(LPC_ADC_TypeDef *ADCx, uint8_t channel, uint32_t StatusType);
uint32_t ADC_GlobalGetData(LPC_ADC_TypeDef *ADCx);
FlagStatus    ADC_GlobalGetStatus(LPC_ADC_TypeDef *ADCx, uint32_t StatusType);


/**********************************************************************/

typedef enum {
    DAC_MAX_CURRENT_700uA = 0,  /*!< The settling time of the DAC is 1 us max,
                                and the maximum current is 700 uA */
    DAC_MAX_CURRENT_350uA       /*!< The settling time of the DAC is 2.5 us
                                and the maximum current is 350 uA */
} DAC_CURRENT_OPT;

typedef struct {
    uint8_t  DBLBUF_ENA;    /**<
                             -0: Disable DACR double buffering
                             -1: when bit CNT_ENA, enable DACR double buffering feature
                             */
    uint8_t  CNT_ENA;       /*!<
                             -0: Time out counter is disable
                             -1: Time out conter is enable
                             */
    uint8_t  DMA_ENA;       /*!<
                                 -0: DMA access is disable
                                 -1: DMA burst request
                            */
    uint8_t RESERVED;
} DAC_CONVERTER_CFG_Type;


void    DAC_Init(LPC_DAC_TypeDef *DACx);
void    DAC_UpdateValue (LPC_DAC_TypeDef *DACx, uint32_t dac_value);
void    DAC_SetBias (LPC_DAC_TypeDef *DACx,uint32_t bias);
void    DAC_ConfigDAConverterControl (LPC_DAC_TypeDef *DACx,DAC_CONVERTER_CFG_Type *DAC_ConverterConfigStruct);
void    DAC_SetDMATimeOut(LPC_DAC_TypeDef *DACx,uint32_t time_out);
```

GPDMA

```
typedef enum {
    GPDMA_STAT_INT,         /**< GPDMA Interrupt Status */
    GPDMA_STAT_INTTC,       /**< GPDMA Interrupt Terminal Count Request Status */
    GPDMA_STAT_INTERR,      /**< GPDMA Interrupt Error Status */
    GPDMA_STAT_RAWINTTC,    /**< GPDMA Raw Interrupt Terminal Count Status */
    GPDMA_STAT_RAWINTERR,   /**< GPDMA Raw Error Interrupt Status */
    GPDMA_STAT_ENABLED_CH   /**< GPDMA Enabled Channel Status */
} GPDMA_Status_Type;

typedef enum{
    GPDMA_STATCLR_INTTC,    /**< GPDMA Interrupt Terminal Count Request Clear */
    GPDMA_STATCLR_INTERR    /**< GPDMA Interrupt Error Clear */
}GPDMA_StateClear_Type;

typedef struct {
    uint32_t ChannelNum;    /**< DMA channel number, should be in
                                range from 0 to 7.
                                Note: DMA channel 0 has the highest priority
                                and DMA channel 7 the lowest priority.
                                */
    uint32_t TransferSize;  /**< Length/Size of transfer */
    uint32_t TransferWidth; /**< Transfer width - used for TransferType is GPDMA_TRANSFERTYPE_M2M only */
    uint32_t SrcMemAddr;    /**< Physical Source Address, used in case TransferType is chosen as
                                 GPDMA_TRANSFERTYPE_M2M or GPDMA_TRANSFERTYPE_M2P */
    uint32_t DstMemAddr;    /**< Physical Destination Address, used in case TransferType is chosen as
                                 GPDMA_TRANSFERTYPE_M2M or GPDMA_TRANSFERTYPE_P2M */
    uint32_t TransferType;  /**< Transfer Type, should be one of the following:
                            - GPDMA_TRANSFERTYPE_M2M: Memory to memory - DMA control
                            - GPDMA_TRANSFERTYPE_M2P: Memory to peripheral - DMA control
                            - GPDMA_TRANSFERTYPE_P2M: Peripheral to memory - DMA control
                            - GPDMA_TRANSFERTYPE_P2P: Source peripheral to destination peripheral - DMA control
                            */
    uint32_t SrcConn;       /**< Peripheral Source Connection type, used in case TransferType is chosen as
                            GPDMA_TRANSFERTYPE_P2M or GPDMA_TRANSFERTYPE_P2P, should be one of
                            following:
                             - GPDMA_CONN_SSP0_Tx: SSP0, Tx
                             - GPDMA_CONN_SSP0_Rx: SSP0, Rx
                             - GPDMA_CONN_SSP1_Tx: SSP1, Tx
                             - GPDMA_CONN_SSP1_Rx: SSP1, Rx
                             - GPDMA_CONN_ADC: ADC
                             - GPDMA_CONN_I2S_Channel_0: I2S Channel 0
                             - GPDMA_CONN_I2S_Channel_1: I2S Channel 1
                             - GPDMA_CONN_DAC: DAC
                             - GPDMA_CONN_UART0_Tx_MAT0_0: UART0 Tx / MAT0.0
                             - GPDMA_CONN_UART0_Rx_MAT0_1: UART0 Rx / MAT0.1
                             - GPDMA_CONN_UART1_Tx_MAT1_0: UART1 Tx / MAT1.0
                             - GPDMA_CONN_UART1_Rx_MAT1_1: UART1 Rx / MAT1.1
                             - GPDMA_CONN_UART2_Tx_MAT2_0: UART2 Tx / MAT2.0
                             - GPDMA_CONN_UART2_Rx_MAT2_1: UART2 Rx / MAT2.1
                             - GPDMA_CONN_UART3_Tx_MAT3_0: UART3 Tx / MAT3.0
                             - GPDMA_CONN_UART3_Rx_MAT3_1: UART3 Rx / MAT3.1
                             */
    uint32_t DstConn;       /**< Peripheral Destination Connection type, used in case TransferType is chosen as
                            GPDMA_TRANSFERTYPE_M2P or GPDMA_TRANSFERTYPE_P2P, should be one of
                            following:
                             - GPDMA_CONN_SSP0_Tx: SSP0, Tx
                             - GPDMA_CONN_SSP0_Rx: SSP0, Rx
                             - GPDMA_CONN_SSP1_Tx: SSP1, Tx
                             - GPDMA_CONN_SSP1_Rx: SSP1, Rx
                             - GPDMA_CONN_ADC: ADC
                             - GPDMA_CONN_I2S_Channel_0: I2S Channel 0
                             - GPDMA_CONN_I2S_Channel_1: I2S Channel 1
                             - GPDMA_CONN_DAC: DAC
                             - GPDMA_CONN_UART0_Tx_MAT0_0: UART0 Tx / MAT0.0
                             - GPDMA_CONN_UART0_Rx_MAT0_1: UART0 Rx / MAT0.1
                             - GPDMA_CONN_UART1_Tx_MAT1_0: UART1 Tx / MAT1.0
                             - GPDMA_CONN_UART1_Rx_MAT1_1: UART1 Rx / MAT1.1
                             - GPDMA_CONN_UART2_Tx_MAT2_0: UART2 Tx / MAT2.0
                             - GPDMA_CONN_UART2_Rx_MAT2_1: UART2 Rx / MAT2.1
                             - GPDMA_CONN_UART3_Tx_MAT3_0: UART3 Tx / MAT3.0
                             - GPDMA_CONN_UART3_Rx_MAT3_1: UART3 Rx / MAT3.1
                             */
    uint32_t DMALLI;        /**< Linker List Item structure data address
                            if there's no Linker List, set as '0'
                            */
} GPDMA_Channel_CFG_Type;

typedef struct {
    uint32_t SrcAddr;   /**< Source Address */
    uint32_t DstAddr;   /**< Destination address */
    uint32_t NextLLI;   /**< Next LLI address, otherwise set to '0' */
    uint32_t Control;   /**< GPDMA Control of this LLI */
} GPDMA_LLI_Type;


void GPDMA_Init(void);
//Status GPDMA_Setup(GPDMA_Channel_CFG_Type *GPDMAChannelConfig, fnGPDMACbs_Type *pfnGPDMACbs);
Status GPDMA_Setup(GPDMA_Channel_CFG_Type *GPDMAChannelConfig);
IntStatus GPDMA_IntGetStatus(GPDMA_Status_Type type, uint8_t channel);
void GPDMA_ClearIntPending(GPDMA_StateClear_Type type, uint8_t channel);
void GPDMA_ChannelCmd(uint8_t channelNum, FunctionalState NewState);
//void GPDMA_IntHandler(void);
```