AURIX 2G iLLD Overview

IFCN ATV SMD GC SAE MC

infineon Low Level Drivers – iLLD

- iLLD come from tests and application used by several teams at infineon ATV
- Low level drivers for use and demonstration for almost every module
- All drivers have the same code styling → common look and feel
- Already tested in pre-silicone with virtual prototyping and in RTL-simulations
- Each derivate and step (TC3YXV) has its own set of drivers
- Same driver for the same module of different derivates

infineon Low Level Drivers – iLLD cont'd

- No dependency between the peripheral drivers
- The iLLD coding guidelines allow layering of drivers for multi-dimensional system scenarios
- Compiler versions given in release note

iLLD - Content

- Driver files
- Register files
- Basic interface software
  - E. g. for a HL-PWM
- Intrinsics to use hardware instructions

IfxCpu_Intrinsics.h	20.12.2015 03:14	H File	5 KB
IfxCpu_IntrinsicsDcc.h	20.12.2015 03:14	H File	29 KB
IfxCpu_IntrinsicsGnuc.h	20.12.2015 03:14	H File	39 KB
IfxCpu_IntrinsicsTasking.h	20.12.2015 03:14	H File	10 KB

- Release Notes

iLLD – Documentation

- With every revision, a new documentation is generated
- The documentation is generated automatically via doxygen
- The documentation is provided with every iLLD-package
- All drivers are under source control, so every change is recorded
- Every new low level driver is automatically written in a way that can be documented by doxygen

iLLD – Documentation

› Documentation is provided as html-file

iLLD_TC39x 1.0

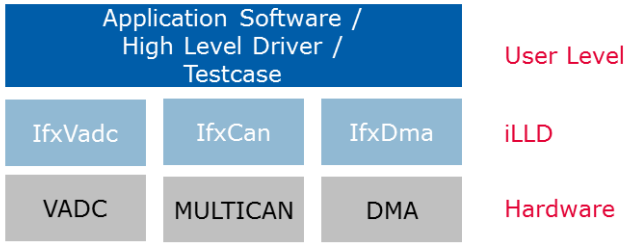
General Structure of iLLDs

We can basically differ between two types of iLLDs:

Unifunctional Peripheral Drivers

Unifunctional peripheral drivers are simple modules where the target peripheral is able to carry out one type of functionality.

Example: VADC peripheral for analog-to-digital conversion. Multican for CAN protocol communication, DMA for DMA functionality, etc.

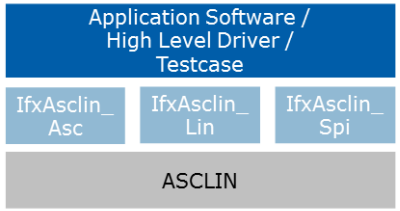


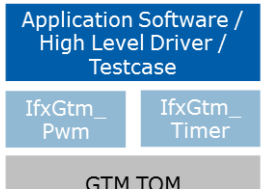
Unifunctional Peripheral Driver

Multifunctional Peripheral Drivers

These kind of peripheral drivers are complex to handle, and here the target peripheral can carry out multiple functionalities ("use cases").

Example: AscLin peripheral can be configured as a common UART, but also as a LIN or even a SPI.
GTM TOM/ATOM can be used as PWM generator, complex waveform generator and any generic timer, etc.

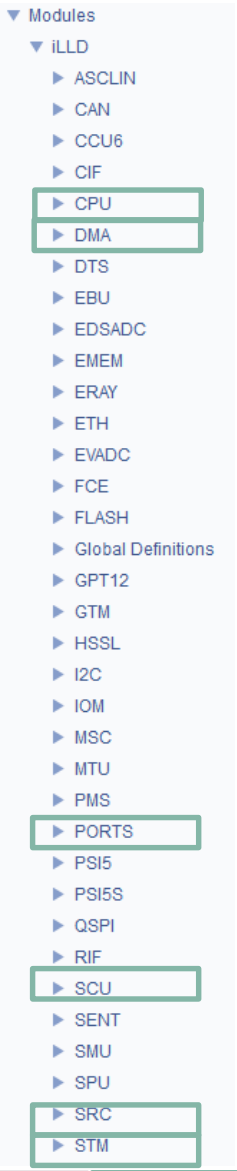




iLLD – Modules

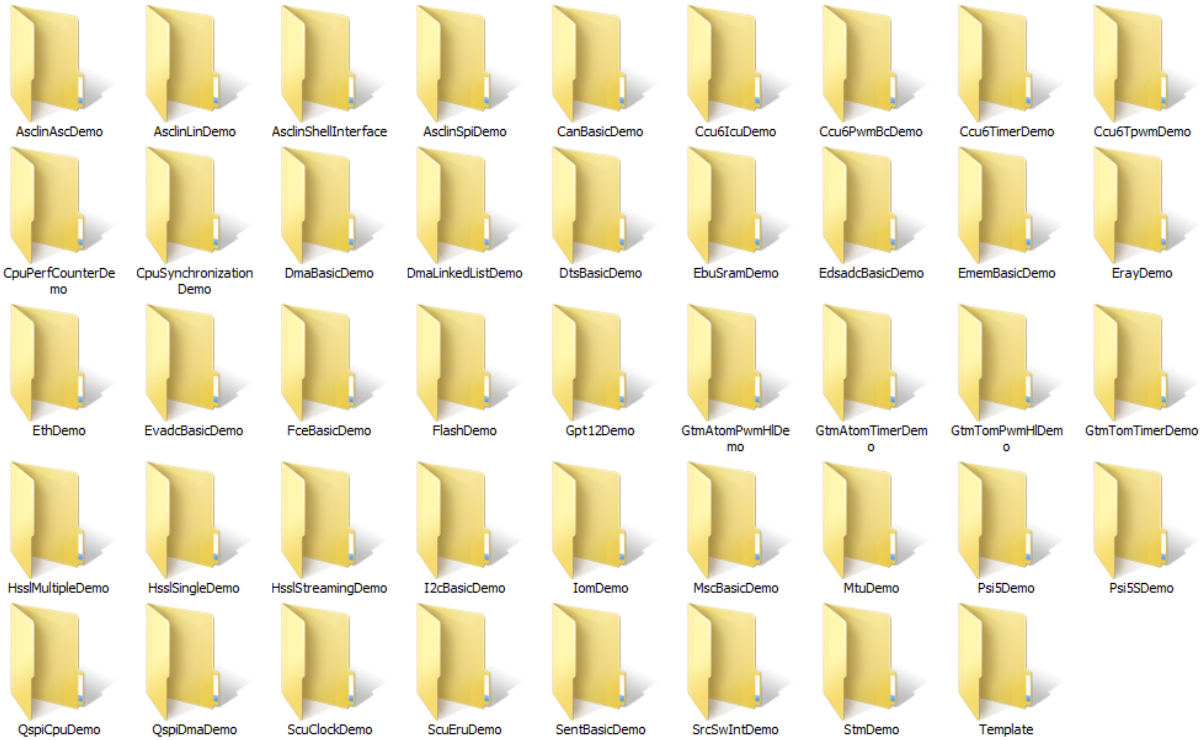
- Almost every module has an own low level driver (33)
- Also a minimal set of driver sources can be used, consisting of:
  - CPU driver
  - DMA driver
  - Ports driver
  - SCU driver
  - Src driver
  - Stm driver

  Always part of the SW_Framework
- To add new drivers, copy the driver files to:
  - 0_Src\4_McHal\<driver>
    and 0_Src\4_McHal_Impl\<driver>_cfg.*
    and 0_Src\4_McHal_PinMap\<driver>_PinMap.*



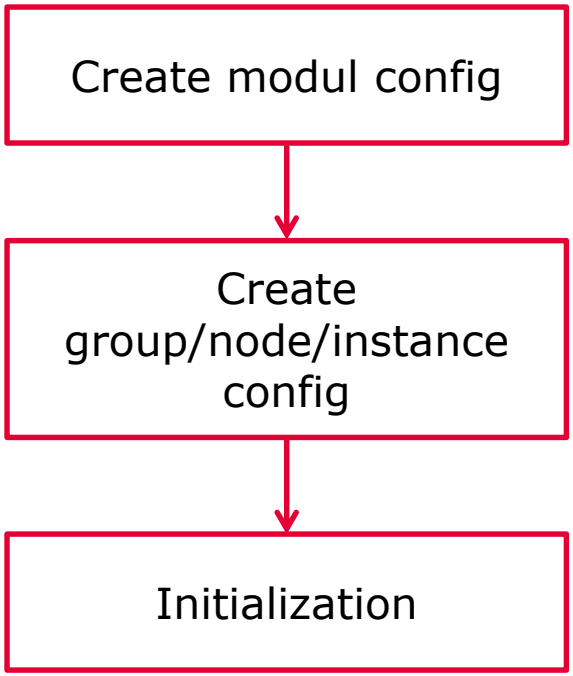
iLLD Examples

- For almost every driver a demo/example is provided



- Easy to import → Copy&Paste in the SW_Framework (only remove printf-instructions)

iLLD – Driver Setup Flow



e. g. Clock

e. g. Pins, trigger sources, interrupt bits

› Interrupt declaration is not in the driver!

iLLD – Config Structs

› All configurations for the initialization of the drivers are in structures so they can easily be expanded

```
/** \brief Module configuration structure
 */
typedef struct
{
    SpiIf_Config                        base;                          /**< \brief SPI interface configuration structure */
    Ifx_QSPI                           *qspi;                          /**< \brief Pointer to QSPI module registers */
    boolean                             allowSleepMode;                /**< \brief Specifies module sleep mode */
    boolean                             pauseOnBaudrateSpikeErrors;    /**< \brief Specifies module pause on baudrate or spike errors */
    IfxQspi_PauseRunTransition          pauseRunTransition;            /**< \brief Specifies module run or pause mode */
    IfxQspi_TxFifoInt                   txFifoThreshold;               /**< \brief Specifies the TXFIFO interrupt threshold */
    IfxQspi_RxFifoInt                   rxFifoThreshold;               /**< \brief Specifies the RXFIFO interrupt threshold */
    IfxQspi_SpiMaster_EnabledInterrupts enabledInterrupts;             /**< \brief Interrupt enables structure */
    const IfxQspi_SpiMaster_Pins       *pins;                          /**< \brief structure for QSPI Master pins */
    IfxQspi_SpiMaster_DmaConfig         dma;                           /**< \brief Dma configuration */
} IfxQspi_SpiMaster_Config;
```

iLLD – Driver Setup

› The configs have dummys for almost every possible config of the module

```
void IfxAsclin_Asc_initModuleConfig(IfxAsclin_Asc_Config *config, Ifx_ASCLIN *asclin)
{
    config->asclin = asclin;

    /* loop back disabled */
    config->loopBack = FALSE;                                          /* no loop back*/

    /* Default values for baudrate */
    config->clockSource           = IfxAsclin_ClockSource_kernelClock;  /* kernel clock, fclc*/
    config->baudrate.prescaler    = 1;                                  /* default prescaler*/
    config->baudrate.baudrate     = 115200;                             /* default baudrate (the fractional divid
    config->baudrate.oversampling = IfxAsclin_OversamplingFactor_4;     /* default oversampling factor*/

    /* Default Values for Bit Timings */
    config->bitTiming.medianFilter        = IfxAsclin_SamplesPerBit_one;       /* one sample per bit*/
    config->bitTiming.samplePointPosition = IfxAsclin_SamplePointPosition_3;   /* sample point position at 3*/
    /* Default Values for Frame Control */
    config->frame.idleDelay               = IfxAsclin_IdleDelay_0;             /* no idle delay*/
    config->frame.stopBit                 = IfxAsclin_StopBit_1;               /* one stop bit*/
    config->frame.frameMode               = IfxAsclin_FrameMode_asc;           /* ASC mode*/
    config->frame.shiftDir                = IfxAsclin_ShiftDirection_lsbFirst; /* shift diection LSB first*/
    config->frame.parityBit               = FALSE;                             /* disable parity*/
    config->frame.parityType              = IfxAsclin_ParityType_even;         /* even parity (if parity enabled)*/
    config->frame.dataLength              = IfxAsclin_DataLength_8;            /* number of bits per transfer 8*/

    /* Default Values for Fifo Control */
    config->fifo.inWidth              = IfxAsclin_TxFifoInletWidth_1;          /* 8-bit wide write*/
    config->fifo.outWidth             = IfxAsclin_RxFifoOutletWidth_1;         /* 8-bit wide read*/
    config->fifo.txFifoInterruptLevel = IfxAsclin_TxFifoInterruptLevel_15;
    config->fifo.rxFifoInterruptLevel = IfxAsclin_RxFifoInterruptLevel_1;
    config->fifo.buffMode             = IfxAsclin_ReceiveBufferMode_rxFifo;    /* RxFIFO*/

    /* Default Values for Interrupt Config */
    config->interrupt.rxPriority    = 0;                                   /* receive interrupt priority 0*/
    config->interrupt.txPriority    = 0;                                   /* transmit interrupt priority 0*/
    config->interrupt.erPriority    = 0;                                   /* error interrupt priority 0*/
```

iLLD – DemoCode Example

› DMA-Linked List

```
void IfxDmaLinkedListDemo_init(void)
{
    uint32                  i;

    /* create module config*/
    IfxDma_Dma_Config       dmaConfig;
    IfxDma_Dma_initModuleConfig(&dmaConfig, &MODULE_DMA);

    /* initialize module */
    IfxDma_Dma              dma;
    IfxDma_Dma_initModule(&dma, &dmaConfig);

    /* initial channel configuration */
    IfxDma_Dma_ChannelConfig cfg;
    IfxDma_Dma_initChannelConfig(&cfg, &dma);

    /* following settings are used by all transactions */
    cfg.transferCount = NUM_TRANSFERED_WORDS;
    cfg.requestMode   = IfxDma_ChannelRequestMode_completeTransactionPerRequest;
    cfg.moveSize      = IfxDma_ChannelMoveSize_32bit;
    cfg.shadowControl = IfxDma_ChannelShadow_linkedList;

    /* generate linked list items */
    for (i = 0; i < NUM_LINKED_LIST_ITEMS; ++i)
    {
        cfg.sourceAddress      =
            IFXCPU_GLB_ADDR_DSPR(IfxCpu_getCoreId(), g_DmaLinkedList.dmaBuffer.source[i]);
        cfg.destinationAddress =
            IFXCPU_GLB_ADDR_DSPR(IfxCpu_getCoreId(), g_DmaLinkedList.dmaBuffer.destination[i]);

        /* address to next transaction set */
        cfg.shadowAddress =
            IFXCPU_GLB_ADDR_DSPR(IfxCpu_getCoreId(), (uint32)&g_DmaLinkedList.drivers.linkedList[(i + 1) % NUM_LINKED_LIST_ITEMS]);

        /* transfer first transaction set into DMA channel */
        if (i == 0)
        {
            IfxDma_Dma_initChannel(&g_DmaLinkedList.drivers.chn, &cfg);
        }

        /* transfer into linked list storage */
        IfxDma_Dma_initLinkedListEntry((void *)&g_DmaLinkedList.drivers.linkedList[i], &cfg);

        if (i == 0)
        {
            /* - trigger channel interrupt once the first transaction set has been loaded (again) into DMA channel */
            g_DmaLinkedList.drivers.linkedList[i].CHCSR.B.SIT = 1;
        }
        else
        {
            /* - activate SCH (transaction request) for each entry, expect for the first one (linked list terminated here) */
            g_DmaLinkedList.drivers.linkedList[i].CHCSR.B.SCH = 1;
        }
    }
}
```

› The demo code **doesn't** show the full functionality

iLLD – Best Way Of Implementation

- Create an handler for every module/HW-instance
- Don‘t execute the handler/driver in parallel on different CPUs
- Take care that all iLLD-files are from the same release

iLLD – General Information

- Coding guidelines for the iLLD are also included as c-files
- For modules that can be connected directly to the input and output ports, there are map-files included that declare the possible connections (Ifx<module>_PinMap.c and .h)
  - Implemented always for the biggest package of the derivate

Part of your life. Part of tomorrow.