Advance Software Framework 4 (ASF4) Overview

Quang Hai Nguyen
20.01.2020

Table of Contents

What is ASF4 and its structure?

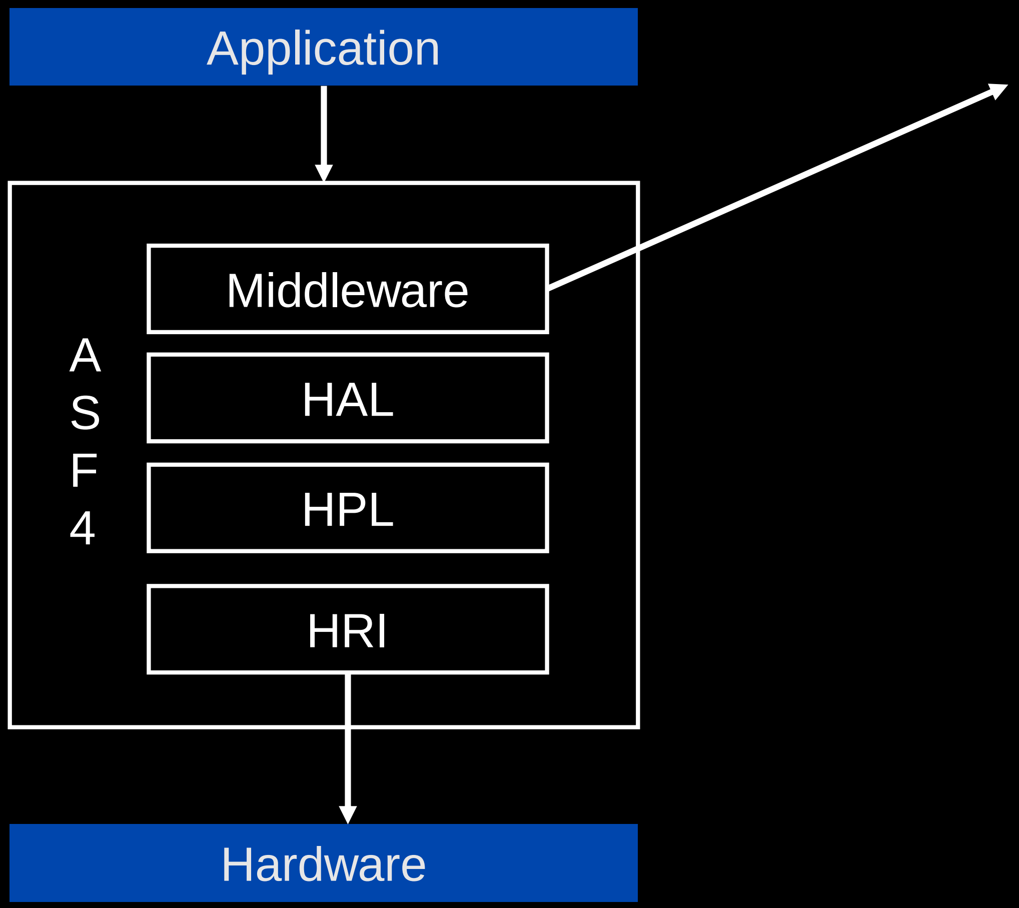
What is ASF4 and its structure?

ASF4

- Collection of software components, e.g. peripheral drivers, middleware, and software application
- Support Microchip SAM controllers
- Designed to work with Atmel Start
  - Atmel Start is web-based interface for peripherals configuration and code generation
- Features:
  - Common set of software interfaces across different devices
  - Smaller code size (compare to previous version)
  - Easier to use

What is ASF4 and its structure?

Software Architecture

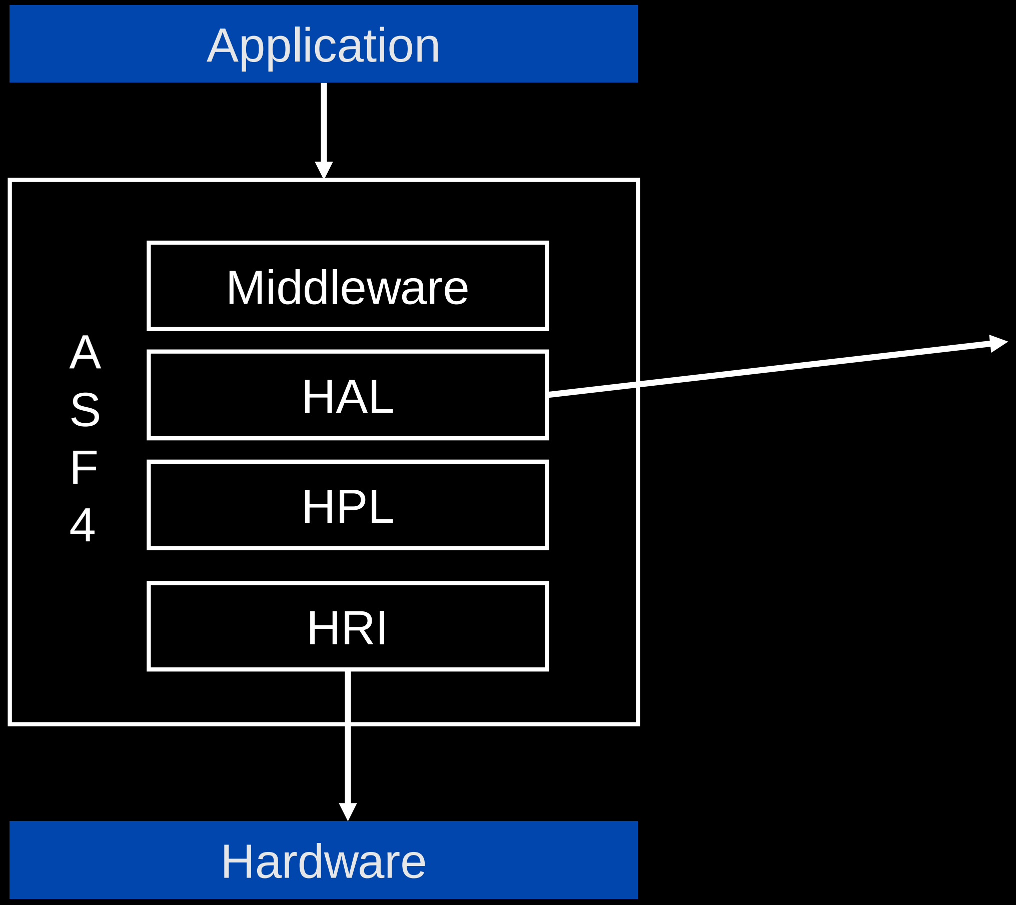


Middleware:

- Communication stack
- RTOS
- Network stack
- Graphic stack

What is ASF4 and its structure?

Software Architecture

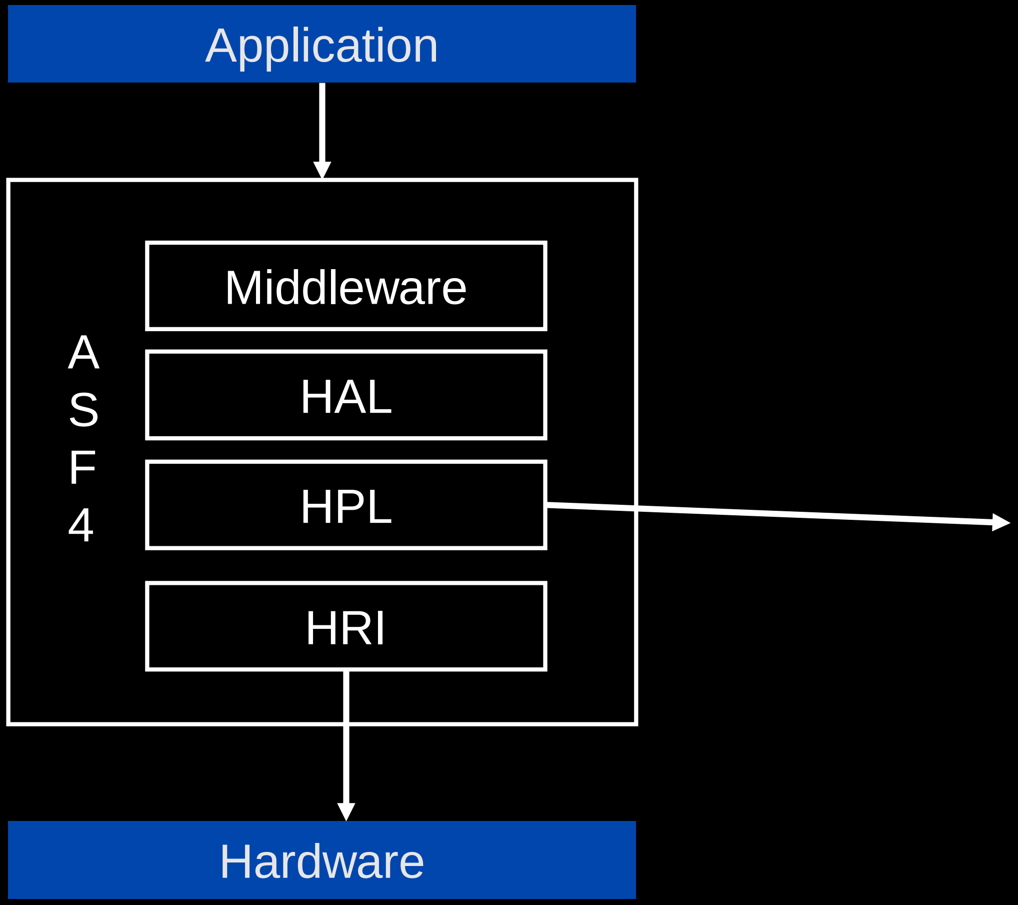


Hardware Abstraction Layer (HAL):

- Supposed to be mostly used by the user
- Hardware dependency hiding – provides the same APIs to developer regardless of the hardware
- General functionalities:
  - Initialize process
  - Start/Stop process
  - Transceiving data
  - Registering callback function

What is ASF4 and its structure?

Software Architecture



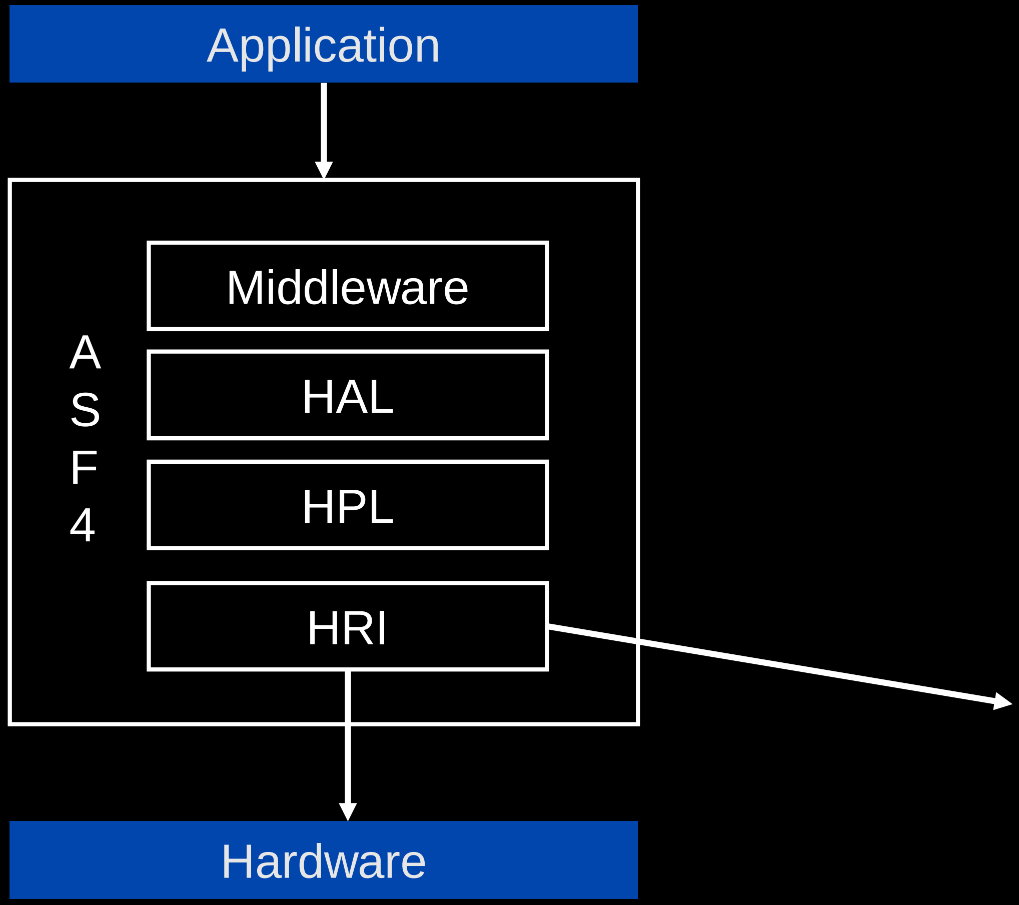
Hardware Proxy Layer (HPL):

> Hardware dependent code is implemented

> Must be ported to support new hardware

What is ASF4 and its structure?

Software Architecture

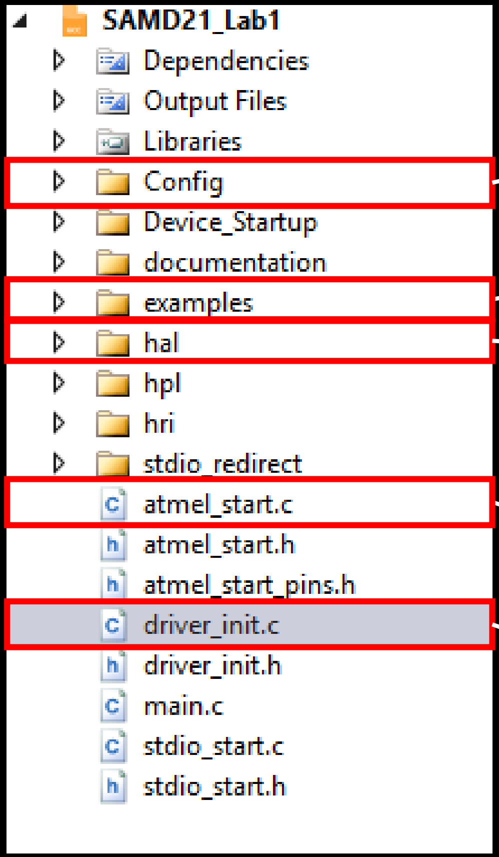


Hardware Register Interface (HRI):
> Register manipulation

ASF4 Project Structure

ASF4 Project Structure

Project structure



SAMD21_Lab1
- Dependencies
- Output Files
- Libraries
- Config
- Device_Startup
- documentation
- examples
- hal
- hpl
- hri
- stdio_redirect
- atmel_start.c
- atmel_start.h
- atmel_start_pins.h
- driver_init.c
- driver_init.h
- main.c
- stdio_start.c
- stdio_start.h

Config → Configuration files

examples → Quick example on how to use initialized software components

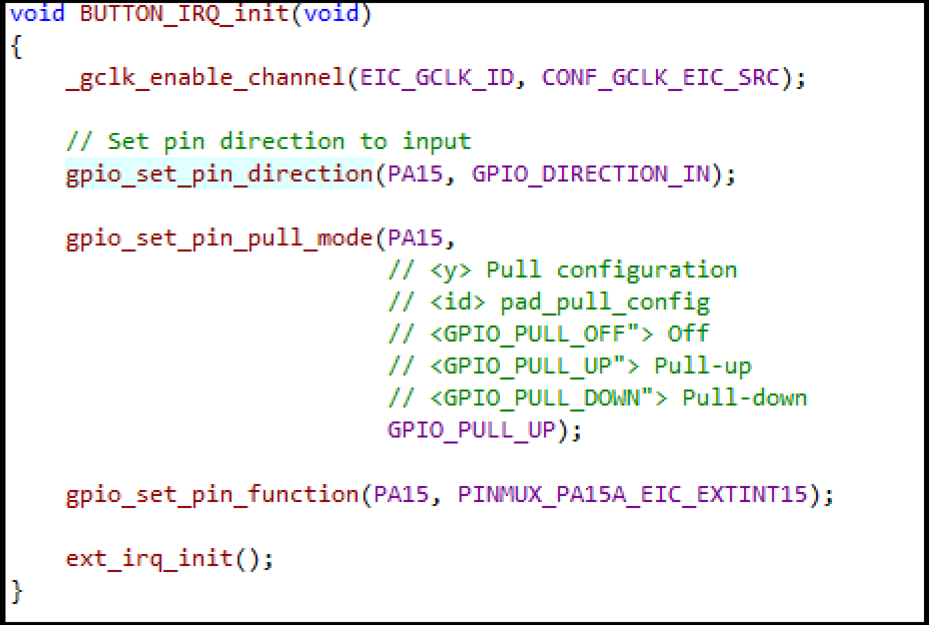
hal → HAL files

atmel_start.c → System and Middleware initialization

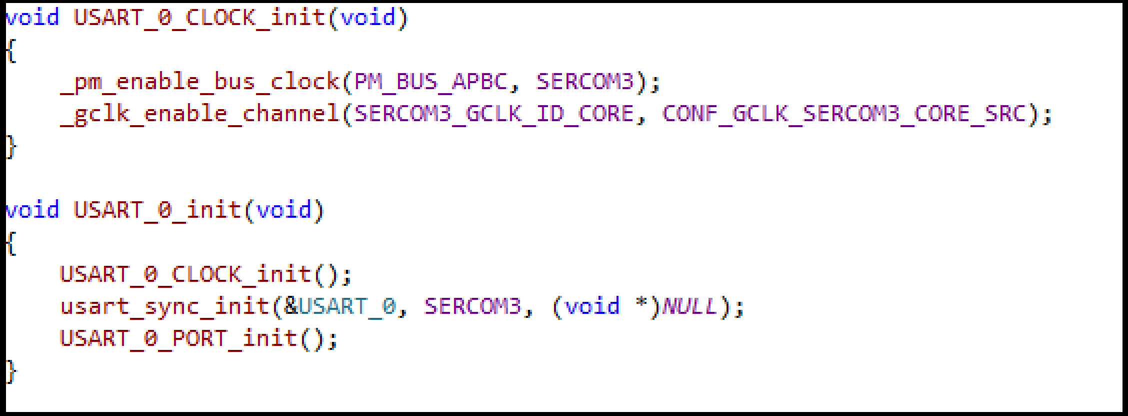
driver_init.c → Peripherals driver initialization

ASF4 Project Structure

driver_init.c



```
void BUTTON_IRQ_init(void)
{
    _gclk_enable_channel(EIC_GCLK_ID, CONF_GCLK_EIC_SRC);

    // Set pin direction to input
    gpio_set_pin_direction(PA15, GPIO_DIRECTION_IN);

    gpio_set_pin_pull_mode(PA15,
                           // <y> Pull configuration
                           // <id> pad_pull_config
                           // <GPIO_PULL_OFF"> Off
                           // <GPIO_PULL_UP"> Pull-up
                           // <GPIO_PULL_DOWN"> Pull-down
                           GPIO_PULL_UP);

    gpio_set_pin_function(PA15, PINMUX_PA15A_EIC_EXTINT15);

    ext_irq_init();
}
```

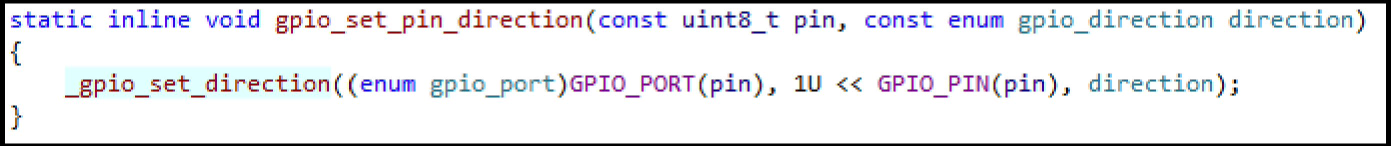


```
void USART_0_CLOCK_init(void)
{
    _pm_enable_bus_clock(PM_BUS_APBC, SERCOM3);
    _gclk_enable_channel(SERCOM3_GCLK_ID_CORE, CONF_GCLK_SERCOM3_CORE_SRC);
}

void USART_0_init(void)
{
    USART_0_CLOCK_init();
    usart_sync_init(&USART_0, SERCOM3, (void *)NULL);
    USART_0_PORT_init();
}
```

The peripherals are initialized according to Atmel Start and using HAL APIs

ASF4 Project Structure

hal_peripherals.c/.h file

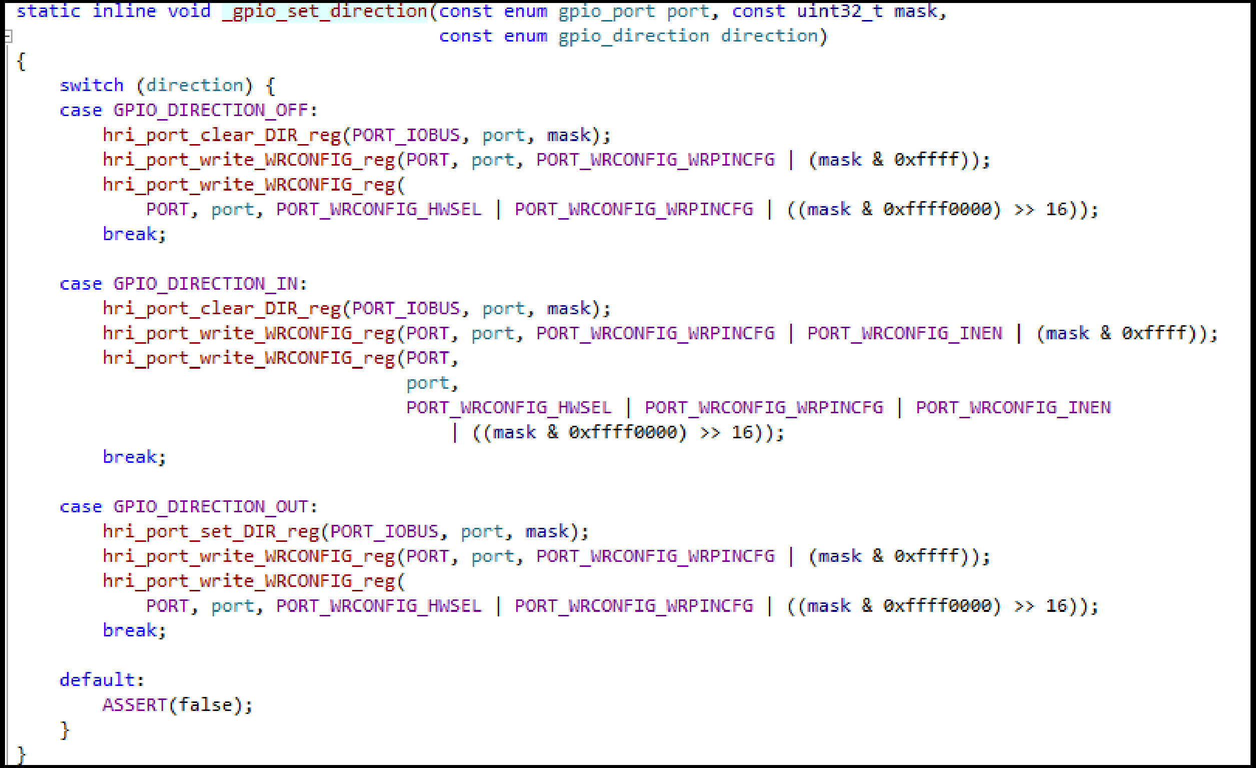


```
static inline void gpio_set_pin_direction(const uint8_t pin, const enum gpio_direction direction)
{
    _gpio_set_direction((enum gpio_port)GPIO_PORT(pin), 1U << GPIO_PIN(pin), direction);
}
```

> Calling function from HPL library

ASF4 Project Structure

hpl_peripherals.h file



```c
static inline void _gpio_set_direction(const enum gpio_port port, const uint32_t mask,
                                       const enum gpio_direction direction)
{
    switch (direction) {
    case GPIO_DIRECTION_OFF:
        hri_port_clear_DIR_reg(PORT_IOBUS, port, mask);
        hri_port_write_WRCONFIG_reg(PORT, port, PORT_WRCONFIG_WRPINCFG | (mask & 0xffff));
        hri_port_write_WRCONFIG_reg(
            PORT, port, PORT_WRCONFIG_HWSEL | PORT_WRCONFIG_WRPINCFG | ((mask & 0xffff0000) >> 16));
        break;

    case GPIO_DIRECTION_IN:
        hri_port_clear_DIR_reg(PORT_IOBUS, port, mask);
        hri_port_write_WRCONFIG_reg(PORT, port, PORT_WRCONFIG_WRPINCFG | PORT_WRCONFIG_INEN | (mask & 0xffff));
        hri_port_write_WRCONFIG_reg(PORT,
                                    port,
                                    PORT_WRCONFIG_HWSEL | PORT_WRCONFIG_WRPINCFG | PORT_WRCONFIG_INEN
                                        | ((mask & 0xffff0000) >> 16));
        break;

    case GPIO_DIRECTION_OUT:
        hri_port_set_DIR_reg(PORT_IOBUS, port, mask);
        hri_port_write_WRCONFIG_reg(PORT, port, PORT_WRCONFIG_WRPINCFG | (mask & 0xffff));
        hri_port_write_WRCONFIG_reg(
            PORT, port, PORT_WRCONFIG_HWSEL | PORT_WRCONFIG_WRPINCFG | ((mask & 0xffff0000) >> 16));
        break;

    default:
        ASSERT(false);
    }
}
```

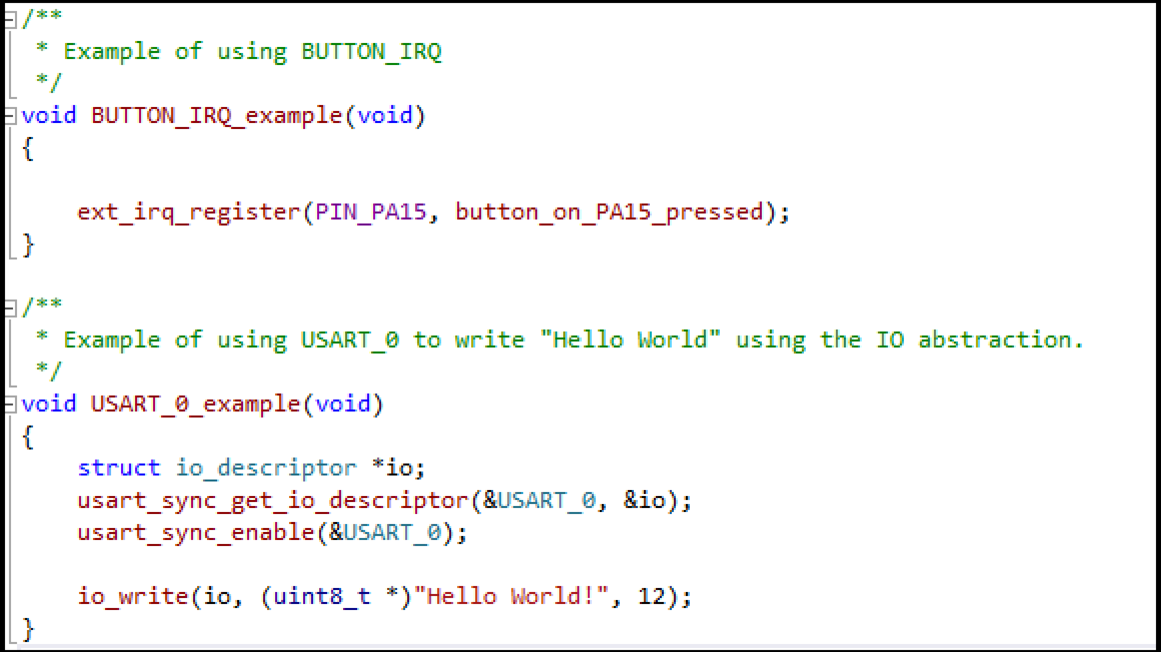
> Calling function from HRI library

ASF4 Project Structure

driver_example.c file

> Showing quick ways to implement a peripherals

> Generated and updated according to Atmel Start



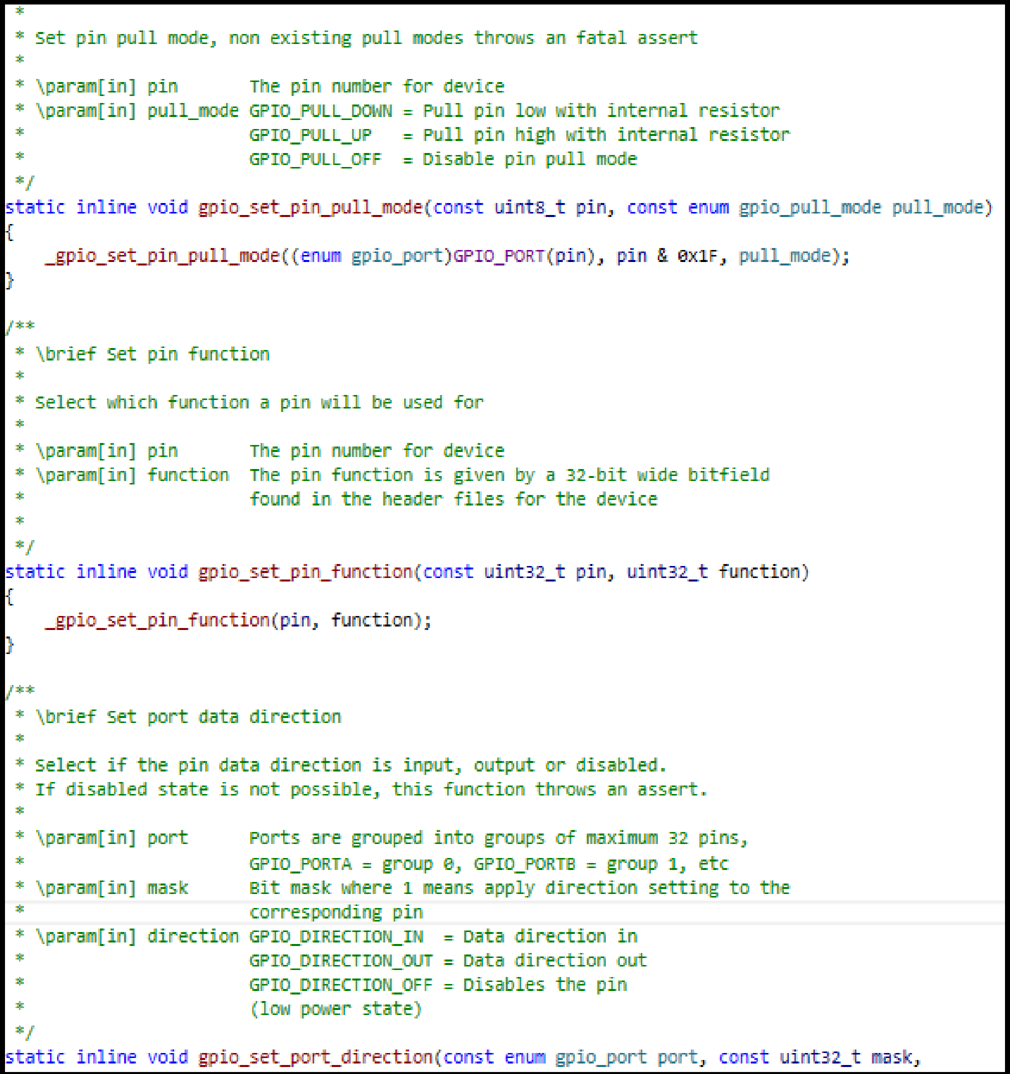
```
/**
 * Example of using BUTTON_IRQ
 */
void BUTTON_IRQ_example(void)
{

    ext_irq_register(PIN_PA15, button_on_PA15_pressed);
}

/**
 * Example of using USART_0 to write "Hello World" using the IO abstraction.
 */
void USART_0_example(void)
{
    struct io_descriptor *io;
    usart_sync_get_io_descriptor(&USART_0, &io);
    usart_sync_enable(&USART_0);

    io_write(io, (uint8_t *)"Hello World!", 12);
}
```

ASF4 Project Structure

hal/include/hal_peripherals.h file

> Available APIs for a particular peripherals

> API description, inputs, outputs and parameters



```
 *
 * Set pin pull mode, non existing pull modes throws an fatal assert
 *
 * \param[in] pin       The pin number for device
 * \param[in] pull_mode GPIO_PULL_DOWN = Pull pin low with internal resistor
 *                      GPIO_PULL_UP   = Pull pin high with internal resistor
 *                      GPIO_PULL_OFF  = Disable pin pull mode
 */
static inline void gpio_set_pin_pull_mode(const uint8_t pin, const enum gpio_pull_mode pull_mode)
{
    _gpio_set_pin_pull_mode((enum gpio_port)GPIO_PORT(pin), pin & 0x1F, pull_mode);
}

/**
 * \brief Set pin function
 *
 * Select which function a pin will be used for
 *
 * \param[in] pin       The pin number for device
 * \param[in] function  The pin function is given by a 32-bit wide bitfield
 *                      found in the header files for the device
 *
 */
static inline void gpio_set_pin_function(const uint32_t pin, uint32_t function)
{
    _gpio_set_pin_function(pin, function);
}

/**
 * \brief Set port data direction
 *
 * Select if the pin data direction is input, output or disabled.
 * If disabled state is not possible, this function throws an assert.
 *
 * \param[in] port      Ports are grouped into groups of maximum 32 pins,
 *                      GPIO_PORTA = group 0, GPIO_PORTB = group 1, etc
 * \param[in] mask      Bit mask where 1 means apply direction setting to the
 *                      corresponding pin
 * \param[in] direction GPIO_DIRECTION_IN  = Data direction in
 *                      GPIO_DIRECTION_OUT = Data direction out
 *                      GPIO_DIRECTION_OFF = Disables the pin
 *                      (low power state)
 */
static inline void gpio_set_port_direction(const enum gpio_port port, const uint32_t mask,
```

Thank You

Quang Hai Nguyen
Field Application Engineer

M +49 1511 6242003
quanghai.nguyen@arrow.com

Arrow Central Europe GmbH
Frankfurter Straße, 211
63263 Neu-Isenburg