

How to assemble and use the AutoDevKit adaptive front lighting kits

Description

The **AEKD-AFL001** represents the complete logic and driving hardware for an adaptive front lighting system for prototyping, testing and development purposes. It consists of several AutoDevKit boards designed for motor vehicle application development featuring ST automotive-grade components.

The set includes two stepper motor control boards, a four-channel LED driver board, a control board with MCU, a connector board with a FAN switch board and another connector board for wiring configuration.

The **STSW-AFL001** adaptive front lighting system firmware runs on the control board automotive-grade SPC5 Chorus MCU and allows independent control of all the function boards and their respective loads. The package also includes sample applications to help users familiarize themselves with the code more quickly.

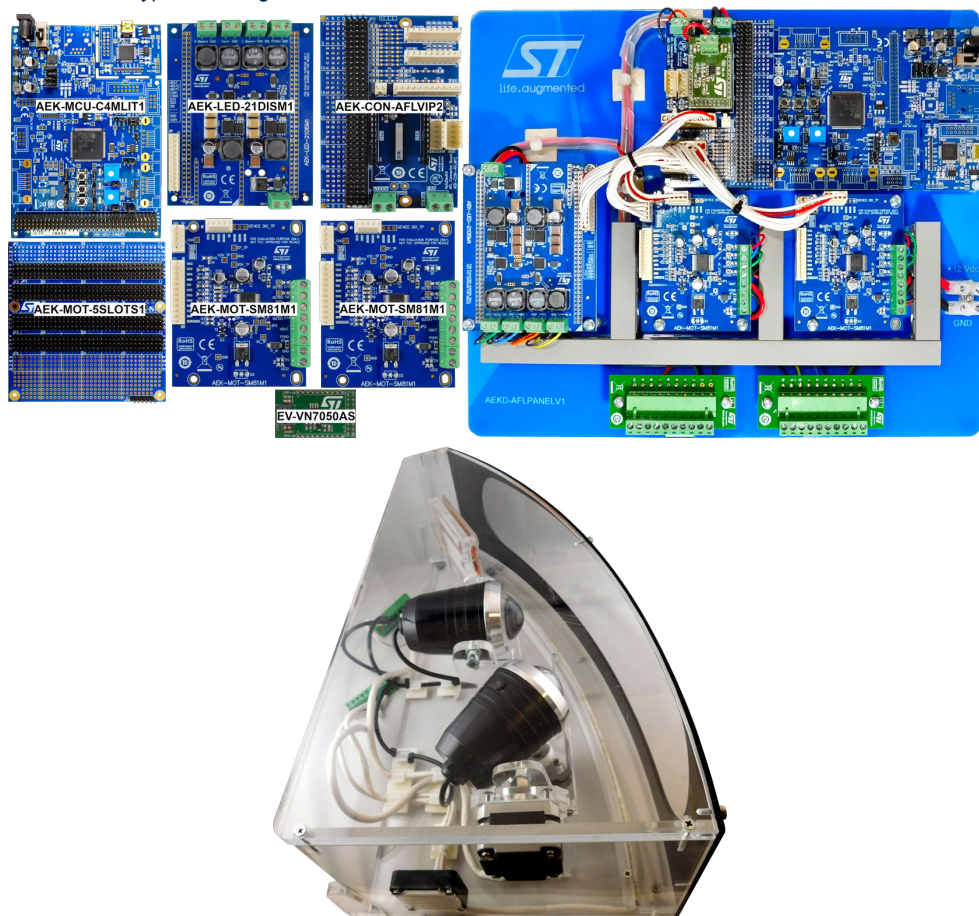
You can also order our fully compatible demo motor vehicle front lighting assembly (**AEKD-AFLIGHT1**) with LED lights, stepper motors and a fan to provide a complete adaptive front lighting tool for application and solution development purposes.

Figure 1. AutoDevKit adaptive front lighting development kits

Left: AEKD-AFL001 kit of individual boards

Right: AEKD-AFLPANEL1 kit of assembled and connected boards

Bottom: AEKD-AFLIGHT1 kit of typical headlight loads



1 Overview of Adaptive front lighting demonstration kits

The [AEKD-AFL001](#) kit consists of an MCU development board, a LED driver board and two motor control boards from the AutoDevKit range designed for automotive application development, as well as a high side driver evaluation board ideal for cooling fan control.

The kit also includes two adapter boards which help manage the various connections between the MCU board and driver boards, and make integrating other boards or components easier with mirrored connectors and breadboard areas.

The [AEKD-AFLPANEL1](#) kit is a convenient alternative to [AEKD-AFL001](#), with the same boards professionally mounted and organized on a Perspex panel, ready for almost immediate use.

Note: All the boards in the [AEKD-AFL001](#) adaptive front lighting kit can be ordered separately from the ST website or from authorized ST distributors.

Table 1. ST development and function boards for AFL applications

Kit RPN	Board RPN	Q.ty	Description	Core product
AEKD-AFL001 / AEKD-AFLPANEL1	AEK-MCU-C4MLIT1	1	MCU discovery board for SPC58 Chorus automotive microcontroller with CAN transceivers	SPC58EC80 32-bit Power Architecture MCU for Automotive General Purpose Applications - Chorus family
	AEK-LED-21DISM1	1	Digitally controlled LED driver board for automotive lighting applications	L99LD21 high power LED driver for automotive applications
	AutoDevKit stepper motor driver board for automotive applications	2	Stepper Motor driver for automotive applications	L99SM81V programmable stepper motor driver for automotive applications
	EV-VN7050AS	1	Driver board ideal for driving AFL cooling fans	VN7050AS high-side driver with MultiSense analog feedback for automotive applications
	AEK-CON-AFLVIP2	1	Adaptive Front Lighting dedicated connector board with VIPower board slot	-
	AEK-CON-5SLOTS1 ⁽¹⁾	1	5 Slot AutoDevKit connector board and breadboard facilitating pin reassignment	-

1. [AEKD-AFL001](#) only

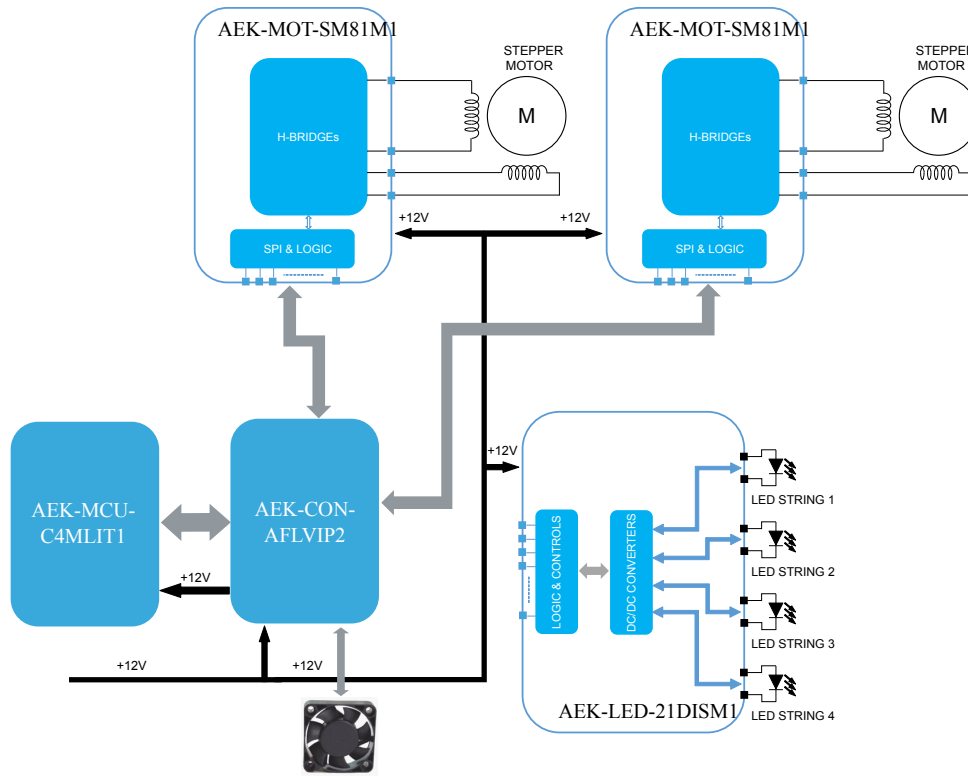
1.1 AFL demonstration kit architecture

The [AEK-MCU-C4MLIT1](#) control board holds the central microcontroller that monitors and controls overall AFL functionality via SPI communication and GPIOs.

The [AEK-CON-AFLVIP2](#) connector board interfaces the control board with the following headlight function boards:

- an [AEK-LED-21DISM1](#) LED driver board, which drives the following independent LED strings for an automotive headlight: high beam, low beam, daytime running light (DRL) and blinker.
- two [AEK-MOT-SM81M1](#) motor control boards to drive independent stepper motors for position adjustment on respective X and Y axes.
- an [EV-VN7050AS](#) cooling fan control board with simple high-side switch.

Note: ST supplies a family of dedicated [EV-VNH7xxx](#) H-bridge boards that provide even greater functionality and control than the board supplied in the kit.

Figure 2. Adaptive Front Lighting development system


All of the boards are 12 V compatible as would be expected in an automotive application powered from a motor vehicle battery. The AEK-AFLPANEL1 kit, in which all the boards are mounted and connected on a Perspex panel, uses common power cabling and a single connector to supply the entire system.

1.2 AFL software development and solutions

The hardware is fully supported by a software ecosystem that includes a development environment ([SPC5-STUDIO](#)) plus plugin ([STSW-AUTODEVKIT](#)) for automotive applications, adaptive front lighting firmware ([STSW-AFL001](#)) complete with sample application code, as well as debugging software ([SPC5-UDESTK-SW](#)).

1.3 Other hardware

ST also supplies an [AEKD-AFLLIGHT1](#) headlight simulation kit with the typical loads found in AFL headlights: two stepper motors, four LED lights and a cooling fan mounted in a Perspex headlight assembly.

Other hardware required for AFL applications using ST boards or kits include:

- A mini-B USB to USB Type-A cable to connect the MCU programmer
- AC adapter 12V 4A
- Jumper wires (if using the [AEKD-AFL001](#))

2 The AutoDevKit AFL development platform

The basic elements of an adaptive front lighting system are the headlight assembly with LED lights and directional positioning motors, and the control and driver logic to manage the system.

The ST AutoDevKit initiative supplies these elements in two highly convenient preassembled kits. The first is the [AEKD-AFLPANEL1](#) kit with all the necessary control and function boards and preloaded firmware, and the second is the [AEKD-AFLLIGHT1](#) headlight assembly, which is connected to the panel through two simple connectors.

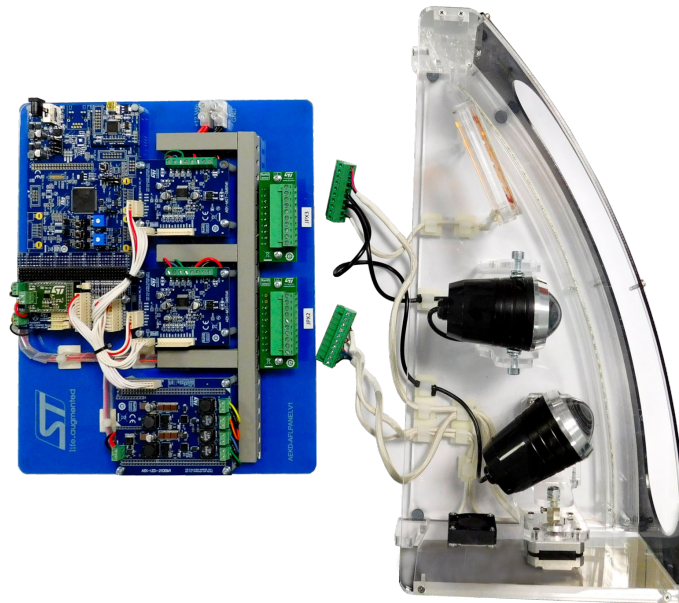
You can also assemble a control unit from the boards in the [AEKD-AFL001](#) kit. This will require wiring the appropriate power and signal lines and interface connectors, and loading the necessary microcontroller firmware before you proceed.

2.1 How to connect the AFL driver kit with the headlight kit

This procedure assumes you are using the [AEKD-AFLPANEL1](#) kit as the control unit and the [AEKD-AFLLIGHT1](#) headlight assembly with typical loads for an adaptive front lighting system.

- Step 1.** Attach the connectors on the headlight assembly to their counterparts on the control board panel.
- Step 2.** Supply appropriate DC power at the dedicated connector.
Connect a 12 V, 4 A power supply adapter.

Figure 3. AFL driver kit and headlight connection



As soon as appropriate power is supplied, the AFL demonstration code pre-loaded on the [AEKD-AFLPANEL1](#) central microcontroller (SPC58 MCU) starts turning the headlights on and off and moving them around in a continuous sequence.

RELATED LINKS

[Watch our Adaptive Front Lighting video](#)

[Appendix A Steering wheel turn simulation with a potentiometer on page 48](#)

3 AFL source code overview: Automatic mode

The `main()` function in the AFL source code consists of the following blocks:

1. The AFL-Demo Setup, which runs an initialization sequence after a power up or reset event.
2. The AFL-Demo main, which runs a sequence of headlight actions in a continuous loop.

3.1 AFL-Demo setup

The AFL-Demo Setup consists of the following functions:

1. Init components
2. Enable Isr
3. Set up AFL environment

The first two instructions manage the initialization of the components related to MCU standard functions and the low-level drivers for the MCU peripherals, such as Clock, External Interrupt Request Queue (EIRQ), Port Configurations, etc. These two functions are automatically generated by SPC5Studio from the configuration information provided during the creation of the SPC5Studio project.

The third function manages the initialization of all AutoDevKit components and MCU functionality implemented by the AFL-Demo engine.

```
/**
 * @brief Setup AFL Environment .
 *
 */
void SetupAflEnv()
{
    /* Turn off the second led of the main Board.*/
    pal_lld_setpad(PORT_F, PF_LED2);
    /* Initialize CAN manager */
    CAN_Manager_init(&CAND2, &can_config_mcanconf);
    /* Initialize the AEK-LED-21DISM1 device */
    ClearAndTrigger(AEK_LED_21DISM1_DEV0);
    /* Initialize the AEK-MOTSM81M1 - Motor x axis */
    init_AEK_MOT_SM81M1(AEK_MOT_SM81M1_DEV0);
    /* Initialize the AEK-MOTSM81M1 - Motor y axis */
    init_AEK_MOT_SM81M1(AEK_MOT_SM81M1_DEV1);
    /* Activate DRL and Low Beam */
    ActivateDRL_LowBeam();
    saradc_lld_start(sarAdc, &saradc_config_saradcconf);
    /* Configure and start WKPU Low Level Driver.*/
    wkpu_lld_start(&WKPUD1, &wkpu_config_configuration_name);
    /* Adjust Low Beam to home position */
    Calibration();
    /* Turn on the cooling fan */
    ActiveIN(EV_VNx7xxx_DEV0);
    /* Start the PIT used for triggering the CAN Message send event. */
    pit_lld_channel_start(&PITD1, PIT0_CHANNEL_CH3);
}
```

The `SetupAflEnv()` function source code shows an important feature of SPC5-STUDIO with the AutoDevKit extension: except for the `ActivateDRL_LowBeam()` and `Calibration()` functions, all the others are part of the APIs made available:

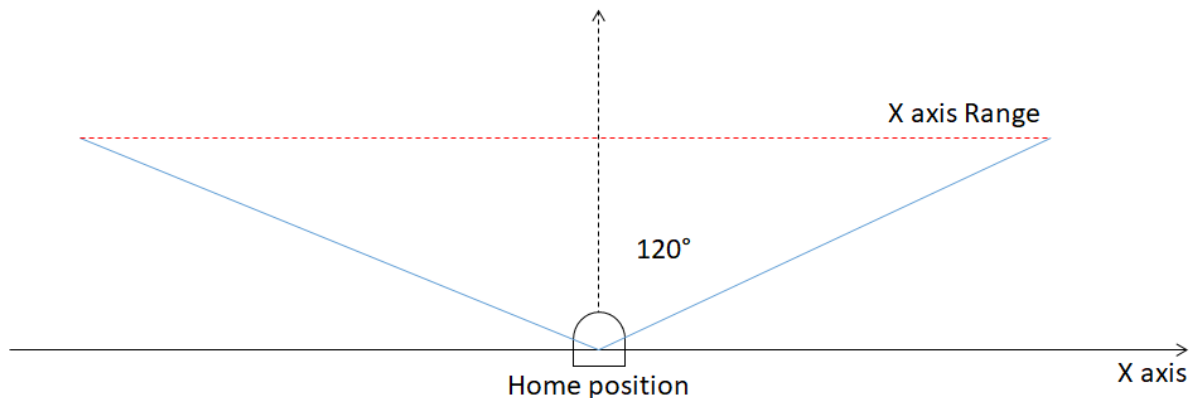
- Low Level Drivers for CAN, ADC, Wake-up, Timer
- `AEK_LED_21DISM1_component_rla`
- `AEK_MOT_SM81M_component_rla`
- `EV_VNx7xxx_component_rla`

These APIs enable direct access to the MCU and its peripheral registers, expose methods to manage the LED driver, stepper motor, and VN actuator components.

3.1.1 Calibration() function

The aim of the Calibration function after power-on is to ensure that the low beam is in its home position. The low beam is moved by the X-axis stepper motor across its full movement range. During this phase, the stepper motor might trip against the X rail end position several times before completing the procedure and returning to center position.

Figure 4. X-axis movement during calibration



The `Calibration()` function source code shows how this functionality is achieved through the functions in the APIs (i.e., `AEK_MOT_SM81M_component_rla`) provided by SPC5-STUDIO and AutoDevKit.

```

/**
 * @brief Calibration .
 *
 */
void Calibration()
{
    TurnRight(AEK_MOT_SM81M1_DEV0,12, 10);
    TurnLeft(AEK_MOT_SM81M1_DEV0,11, 10);
    TurnRight(AEK_MOT_SM81M1_DEV0,5, 1);
}

```

The parameters for the `TurnRight()` and `TurnLeft()` functions belonging to the stepper motor APIs are:

- The device (stepper motor) to be driven by the MCU.
- The number of motor steps to perform either on the right or on the left side.
- The delay time in ms between one motor step and the following one.

3.1.2 ActivateDRL_LowBeam() function

The `ActivateDRL_LowBeam()` function implements a similar API-based approach.

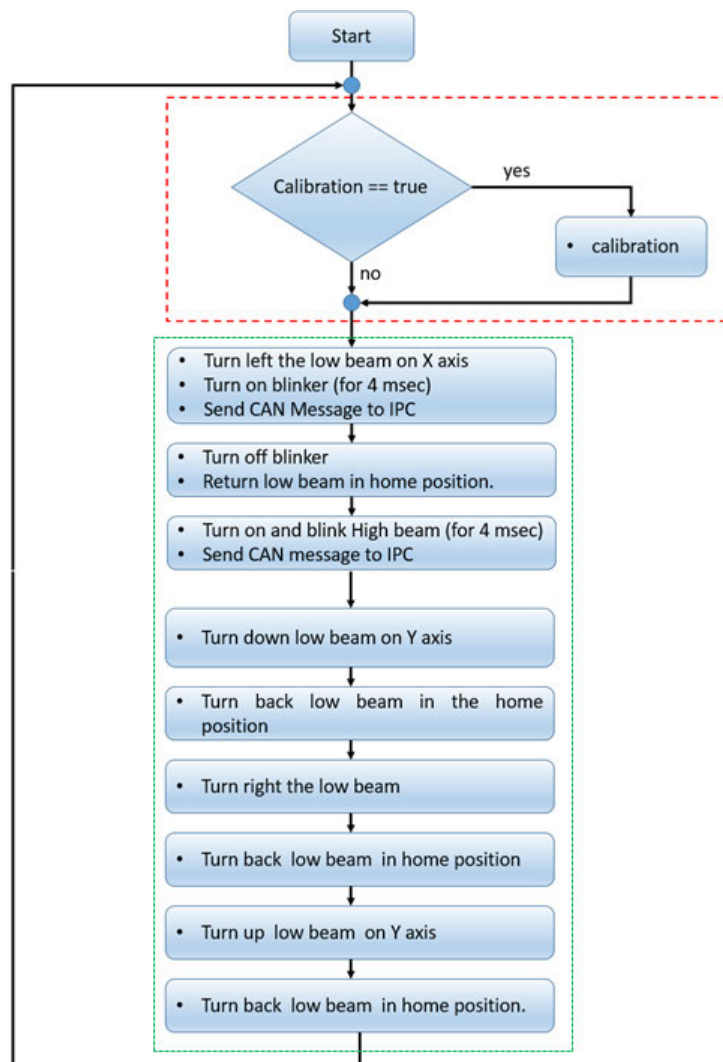

```

void ActivateDRL_LowBeam()
{
    AEK_LED_21DISM1drv_DINActiveBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV1, BUCK2);
    osalThreadDelayMilliseconds(3000);
    AEK_LED_21DISM1drv_DINActiveBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV1, BUCK1);
    CAN_Manager_Activation_LowBeam();
}
    
```

3.2 AFL-Demo main

The AFL-Demo main engine is implemented by the `Afl()` function. The flow chart below shows an initial block involving calibration, followed by a longer block involving several actions. The calibration block is ignored in Automatic Mode because there is no steering wheel input in this mode.

Figure 5. AFL-Demo main flowchart



The source code below implements several functions.

```
// Turn On Blinker
pit_lld_channel_start (&PITD1, PIT0_CHANNEL_CH2);

//Turn left Motor axis x
TurnLeft (AEK_MOT_SM81M1_DEVO, 5, 1);
```

- Turn the low beam left on x-axis.
- Turn on the blinker (for 4 ms).
- Send CAN message to IPC.

3.2.1 TurnLeft()

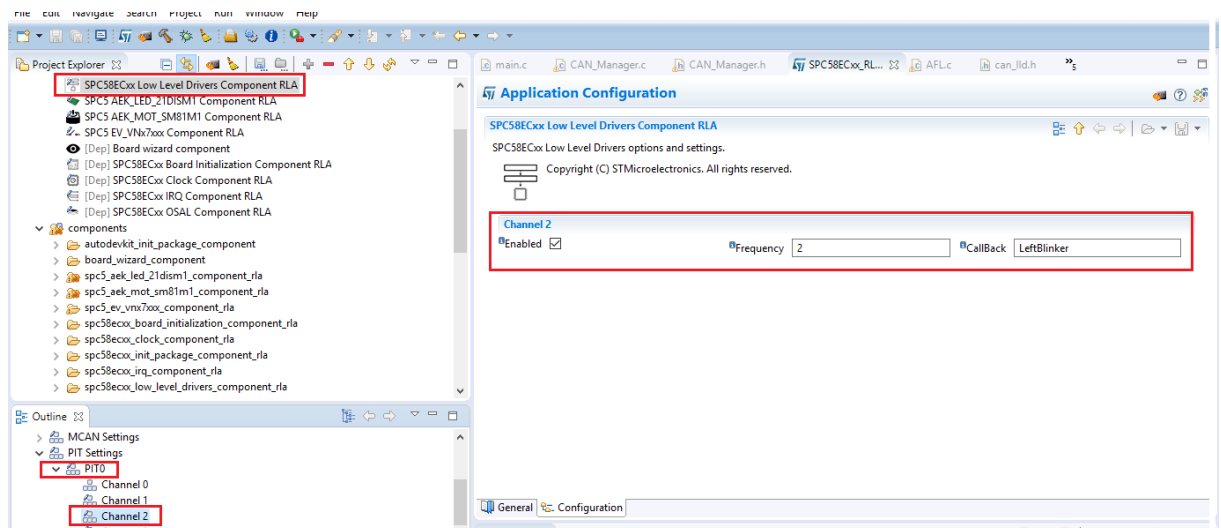
When the `TurnLeft()` function is executed, the MCU sends a driving command via SPI to the x-axis stepper motor to turn the low beam left. This function belongs to the collection of APIs related to `AEK_MOT_SM81M_component_rla` from the AutoDevKit plugin.

3.2.2 pit_lld_channel_start()

The `pit_lld_channel_start()` function configures a programmable interrupt timer (PIT) channel to control the blinking frequency of the turn indicator light. The configuration requires timing information expressed as a frequency and the name of a callback function to be executed at the end of each PIT time lapse. The `pit_lld_channel_start` belongs to the low level driver APIs provided by SPC5Studio.

The `pit_lld_channel_start()` function starts a programmable interrupt timer that raises an event at the set frequency to interrupt the MCU and force it to run the `LeftBlinker(void)` callback function which specifically manages this event.

Figure 6. Component configuration for left turn indicator blinking



RELATED LINKS

[7.4 Component configuration in SPC5-STUDIO on page 33](#)

3.2.3 LeftBlinker()

The `LeftBlinker()` function switches the turning light on or off according to the `on_off` variable, which produces the typical blinking effect on a car turn indicator light.


```

/**
 * @brief Callback called by the PIT (programmable interrupt timer).
 * This function according to value of the variable "on_off"
 * turn on or off the blinker
 */
void LeftBlinker(void)
{
    osalExitCriticalFromISR();
    if (on_off)
    {
        AEK_LED_21DISM1drv_DINActiveBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV2, BUCK2);
        CAN_Manager_DeActivation_BlinkerL();
        on_off = false;
    }
    else
    {
        AEK_LED_21DISM1drv_DINOffBuckSPC(AEK_LED_21DISM1_DEV0, REG_CR3, rxbuf, DEV2, BUCK2);
        CAN_Manager_Activation_BlinkerL();
        on_off = true;
    }
    osalEnterCriticalFromISR();
}

```

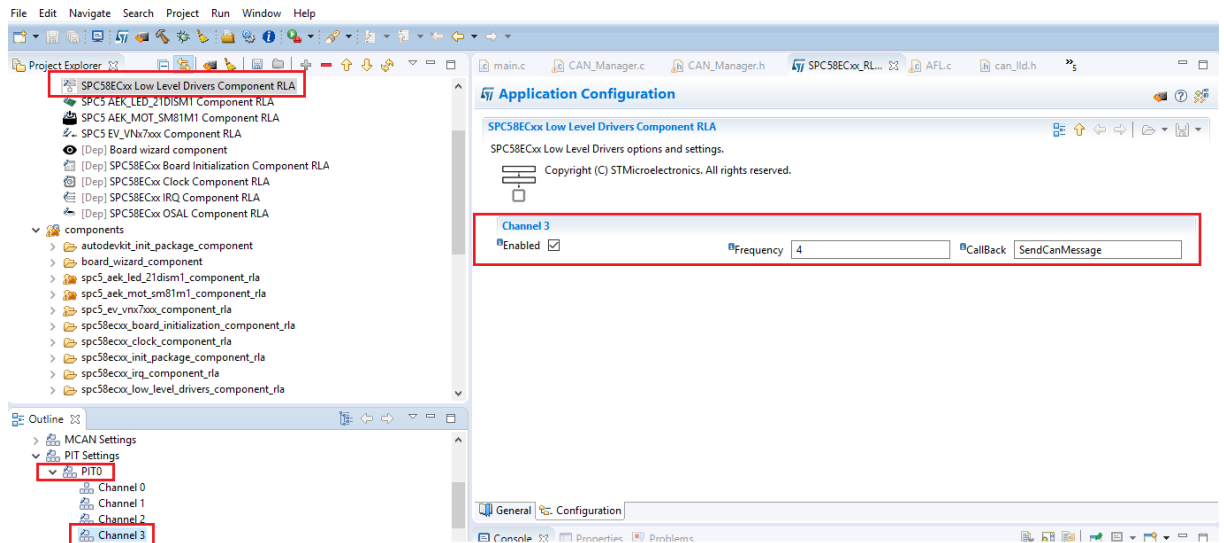
The `LeftBlinker()` source code involves the following API functions:

- `AEK_LED_21DISM1drv_DINActiveBuckSPC()`
- `AEK_LED_21DISM1drv_DINOffBuckSPC()`
- `CAN_Manager_DeActivation_BlinkerL()`
- `CAN_Manager_Activation_BlinkerL()`

The first two APIs send control signals via SPI to activate and deactivate the buck circuit in the LED driver board in order to switch the indicator light on and off.

The other two functions set a hexadecimal code in a CAN message that is transmitted to an instrumental panel control (IPC) to actuate the turn indicator LEDs in the car dashboard. The interrupt that actually sends the CAN message is another dedicated PIT which periodically executes the `CAN_Manager_TransmitMessage()` callback.

Figure 7. Component configuration for dashboard indicator blinking



```
void CAN Manager TransmitMessage(CANDriver *driver)
{
    CANTxFrame tmp;
    int n_signal = 0;
    tmp.ID = can_mess.id;
    tmp.DLC = can_mess.dlc;
    tmp.OPERATION = CAN_OP_NORMAL;
    tmp.TYPE = can_mess.type;
    tmp.data32[0] = can_mess.data32[0];
    tmp.data32[1] = can_mess.data32[1];
    can_ild_transmit(driver, CAN_ANY_TXBUFFER, &tmp);
}
```

RELATED LINKS

[7.4 Component configuration in SPC5-STUDIO on page 33](#)

3.2.4 osalThreadDelayMilliseconds()

The `osalThreadDelayMilliseconds()` function is invoked several times in the source code to implement delays, which are fundamental in real-time asynchronous environments without an operating system. This function is part of the operating system abstraction (OSAL) low level driver provided by SPC5-STUDIO.

3.2.5 Other functions

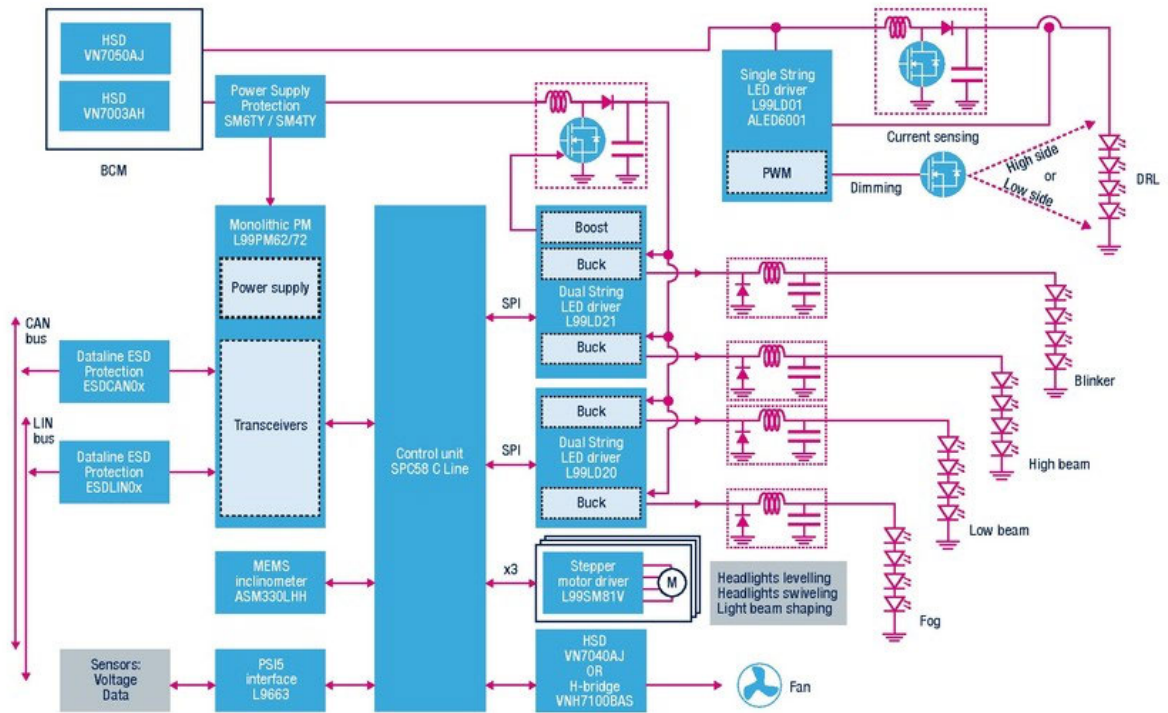
- Low level driver functions:
 - `pal_ild_setpad`: function used to set the logical state of an output port
 - `can_ild_start`: function used to configure and to activate the CAN peripheral
 - `saradc_ild_start`: function used to configure and to activate SAR ADC peripheral
 - `wkpu_ild_start`: function used to configure and to activate the WKPU signal
- `AEK_LED_21DISM1_component_rla`:
 - `AEK_LED_21DISM1drv_DINActiveBuckSPC`: activates the Buck of the LED driver device
 - `AEK_LED_21DISM1drv_DINOffBuckSPC`: deactivates the Buck of the LED driver device
 - `AEK_LED_21DISM1drv_IntensityBuckSPC`: sets the current to define the light intensity
 - `AEK_LED_21DISM1drv_ClearStausRegister`: clears the status LED driver the control register

- AEK_MOT_SM81M_component_ria
 - AEK_MOT_SM81M1drv_WriteSPIRegisterMOV: writes the appropriate register to set the step resolution
 - init_AEK_MOT_SM81M1: initializes the register the stepper motor driver registers

4 Hardware

The image below shows the extent of the range of ST products related to adaptive front lighting applications, including the SPC58 Chorus 4M automotive microcontroller with CAN transceivers, which is able to control the entire system.

Figure 8. AFL block diagram with ST application devices



In the [AEKD-AFL001](#) and [AEKD-AFLPANEL1](#) kits, the [AEK-MCU-C4MLIT1](#) control board MCU monitors and controls the following function boards:

- an [AEK-LED-21DISM1](#) evaluation board with two [L99LD21](#) LED drivers to drive headlight lamps and turning indicators.
- two [AEK-MOT-SM81M1](#) evaluation boards with [L99SM81V](#) stepper motor driver for low beam horizontal swivel adjustment and vertical tilt adjustment.
- [EV-VN7050AS](#) evaluation board with VN7050SA high-side driver for fan control.

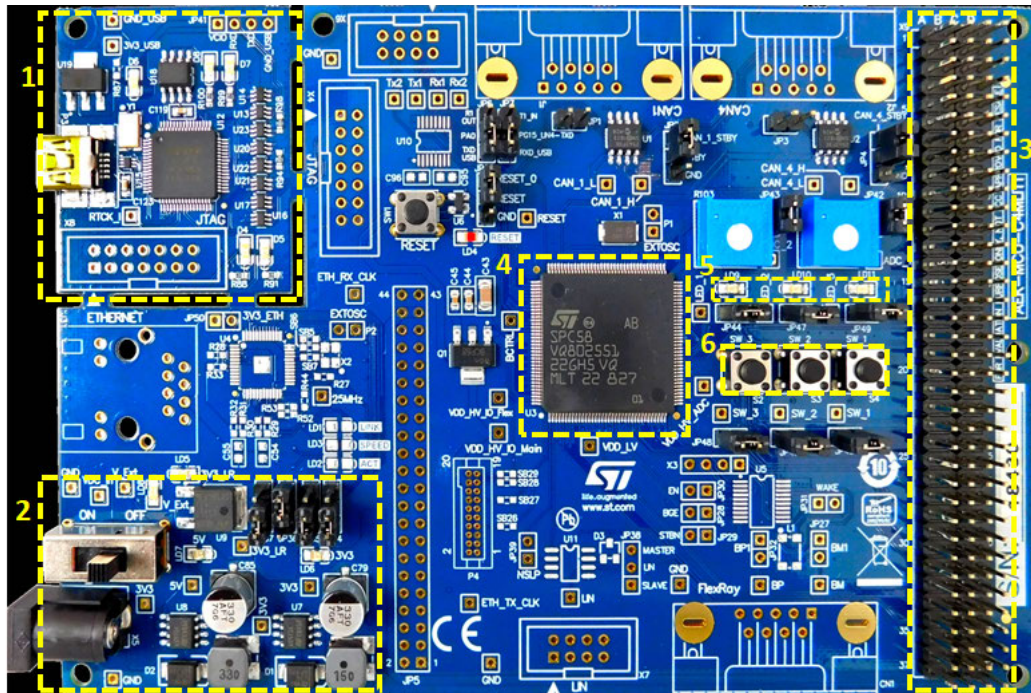
4.1 AEK-MCU-C4MLIT1 system control board

This board provides overall AFL system control via SPI and GPIOs. It is Automotive Safety Integrity Level B (ASIL B) qualified and features a hardware security module (HSM) with security software that is compliant with the EVITA Medium version.

The board has an integrated programmer debugger (supported by PLS) for MCU debugging and programming and a USB Virtual COM port for serial output to a PC running a standard terminal program.

Figure 9. AEK-MCU-C4MLIT1 control board functional blocks

1. PLS debugger/programmer
2. 12V input power and DC-DC conversion to 5V and 3.3V
3. Connector (4x37 pins) for function boards
4. MCU [SPC58EC80E5](#) 32-bit
5. User defined LEDs
6. User defined buttons



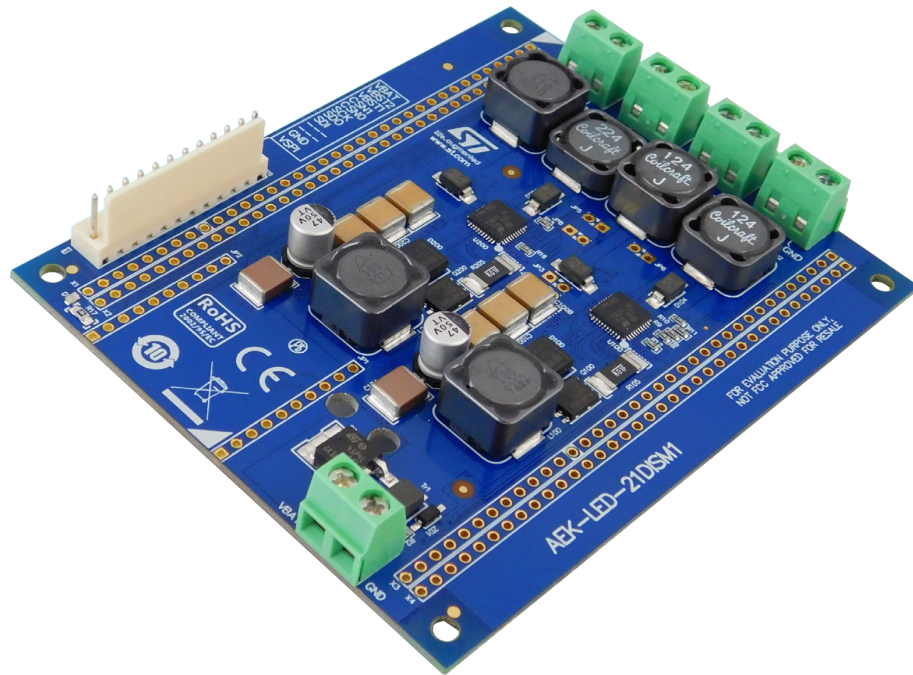
The 32-bit [SPC58EC80E5](#) microcontroller has the following features:

- 2 main dual-issue 32-bit CPUs built on Power Architecture technology, operating up to 180MHz.
- 4224 KB (4096 KB code Flash + 128 KB data Flash) on-chip flash memory.
- 384 KB on-chip general-purpose SRAM (+ 128 KB local data RAM: 64 KB included in each CPU).
- 8 DSPIs available: 3 DSPIs are allocated in AFL-Demo to connect the LED driver board and the two stepper motor driver boards.
- 8 (Programmable Interrupt Timer) PIT channels available: 2 PIT channels are allocated for the AFL-Demo.
- 2 x 32 eMIOS channels available: 1 channel allocated for the AFL-Demo.
- Up to 96 channels SARADC available: 1 channel allocated for steering wheel position.
- 8 MCAN available: 1 MCAN used to connect and control an instrumental dashboard.

4.2 AEK-LED-21DISM1 LED driver board

The [AEK-LED-21DISM1](#) evaluation board is designed to drive four LED strings. In the AFL-Demo, it is used to power the daytime running light (DRL), low beam, high beam and blinker. The board hosts two [L99LD21](#) LED driver ICs, each of which is able to manage a single boost and two buck circuits.

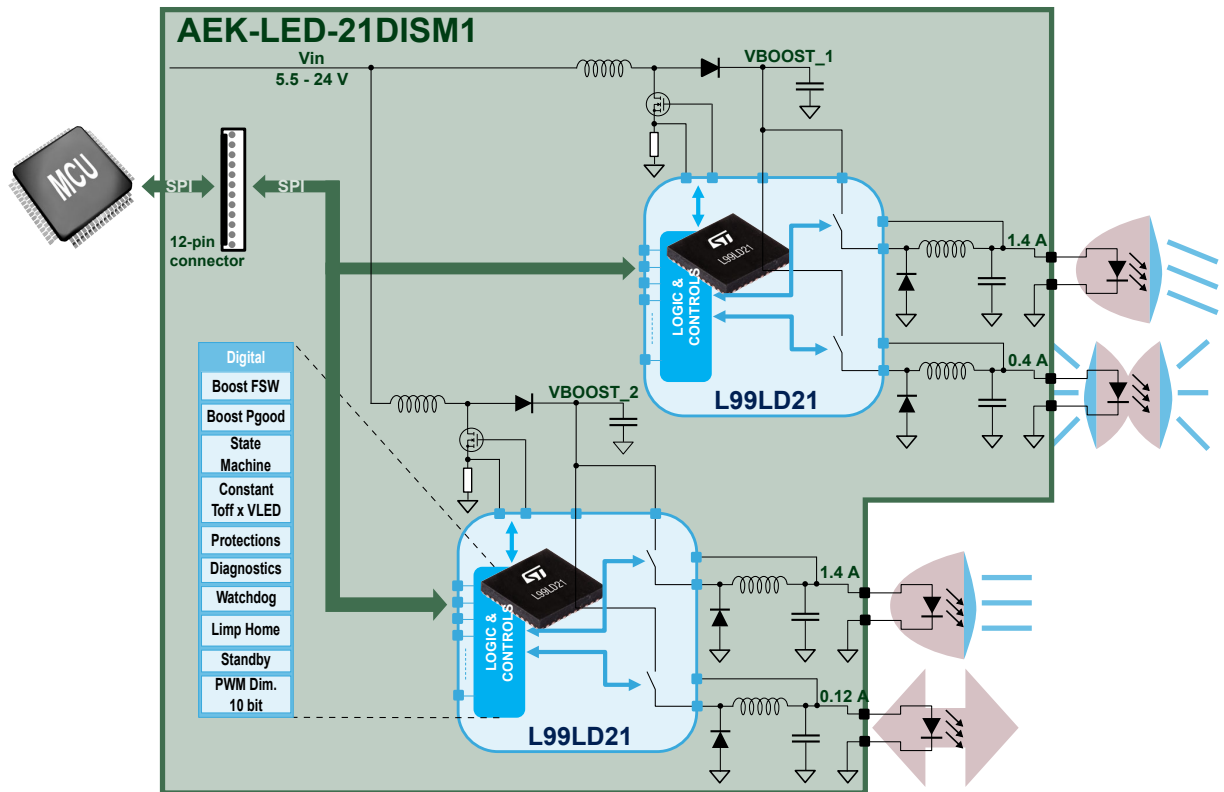
Figure 10. AEK-LED-21DISM1 LED driver board



The boost controller section in each [L99LD21](#) LED driver integrates a high current gate driver for an external N-channel MOSFET and delivers a constant output voltage to two integrated buck converters. The boost controller of the two devices can be stacked to allow dual phase operation for high power applications with an interleaving pattern for improved input current ripple and current load sharing. Each [L99LD21](#) buck converter integrates an N-channel MOSFET driven by a bootstrap circuit.

The overall behavior of the board [AEK-LED-21DISM1](#) is monitored and controlled through [L99LD21](#) dedicated registers, which can be read or set by an external microcontroller through the fast SPI interface via a 12-pin male connector. Finally, a special Limp Home Mode implemented in the [L99LD21](#) driver ensures that active communication with the MCU is regularly monitored.

Figure 11. AEK-LED-21DISM1 four LED load driving block diagram



The AEK-LED-21DISM1 board has the following features:

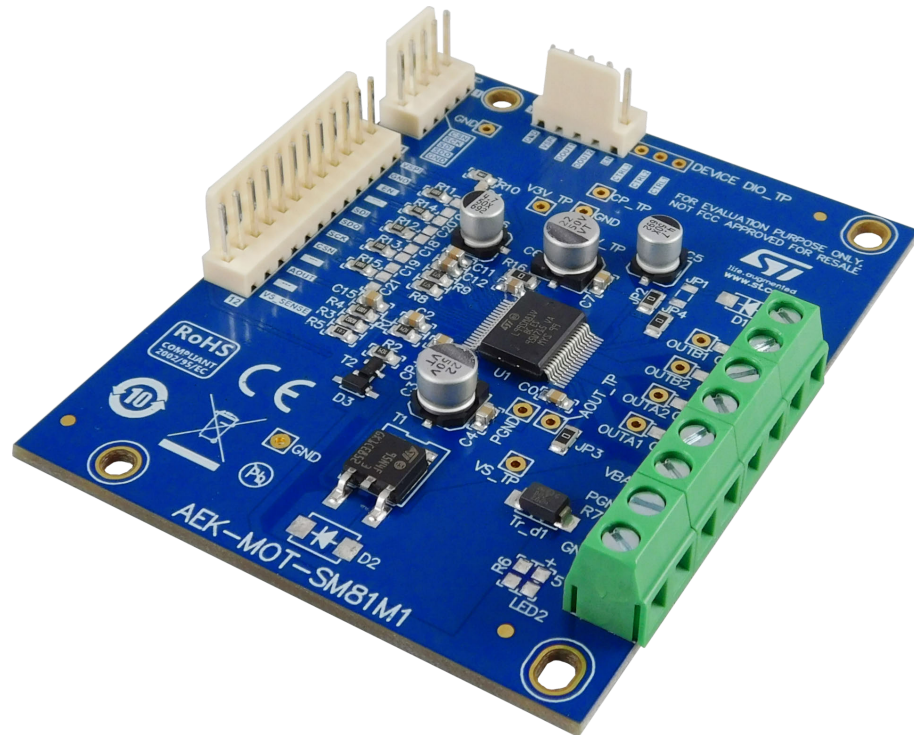
- Two embedded L99LD21 high power LED drivers able to supply four independent channels
- Output current up to 1.69 A each channel
- Input operating voltage range from 5.5 V to 24 V
- Programmable via SPI, allowing accurate LED current setting
- Protections and diagnostics for output short-circuit and open-load, overtemperature, and battery undervoltage.
- Automotive Limp-Home Mode support
- Board size: 81 x 100 mm. Maximum components height: 11 mm
- WEEE and RoHS compliant
- All ST components are qualified Automotive grade
- Board designed according to the AutoDevKit initiative

4.3 AEK-MOT-SM81M1 stepper motor driver board

The AEK-MOT-SM81M1 evaluation board targets automotive stepper motor applications. It is designed to drive one bipolar stepper motor with stepping resolution ranging from 4 full steps to 64 micro steps and with coil voltage monitoring for stall detection. The embedded L99SM81V stepper motor driver manages timers, counters, a reference table and status registers that are manipulated by an external MCU via SPI.

The AEKD-AFL001 and AEKD-AFLPANEL1 kits include two boards to drive an x-axis and a y-axis stepper motor.

Figure 12. AEK-MOT-SM81M1 motor driver board



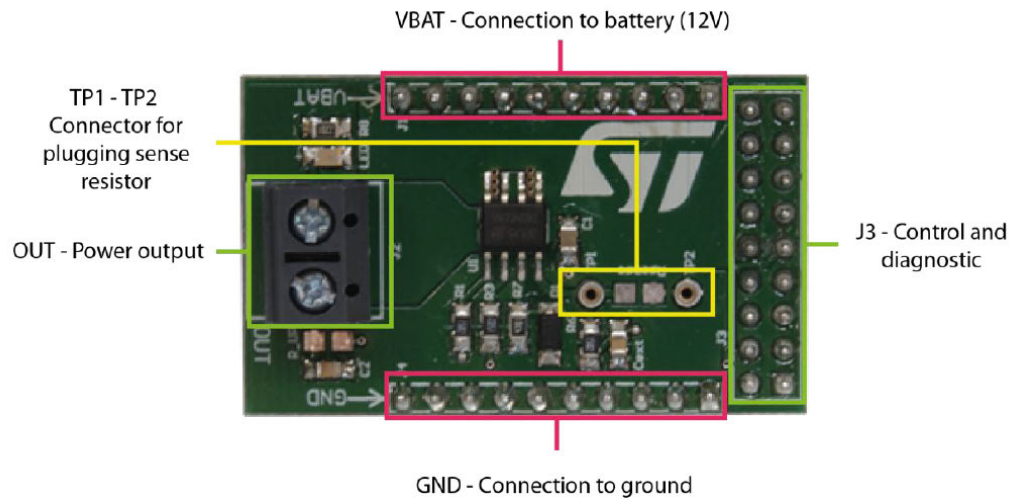
The **AEK-MOT-SM81M1** board has the following features:

- Board functionality based on **L99SM81V** programmable stepper motor driver for automotive applications:
 - with micro-stepping and hold functions
 - BEMF monitoring for stall detection
 - programmable configuration via SPI
 - 5 V internal linear voltage regulator (output available on board connector)
- Board reverse battery protection with **STD95N4F3** MOSFET, which can be substituted with two optionally mounted diodes and a jumper
- Input operating voltage range from 6 V to 28 V
- Output current up to 1.35 A
- Board size: 65 mm length x 81 mm width x 11 mm maximum component height
- WEEE and RoHS compliant
- All ST components are qualified Automotive grade
- Part of the AutoDevKit initiative
- Applications: automotive bipolar stepper motor

4.4 EV-VN7050AS fan control board

The **EV-VN7050AS** evaluation board in the AFL-Demo is used to drive a cooling fan for the LED headlight. The board is based on the **VN7050AS** single channel high-side driver featuring ST's VIPower technology. The device is normally powered at 12 V, but has an operating range of 4-28 V to accommodate battery voltage spikes and crank values.

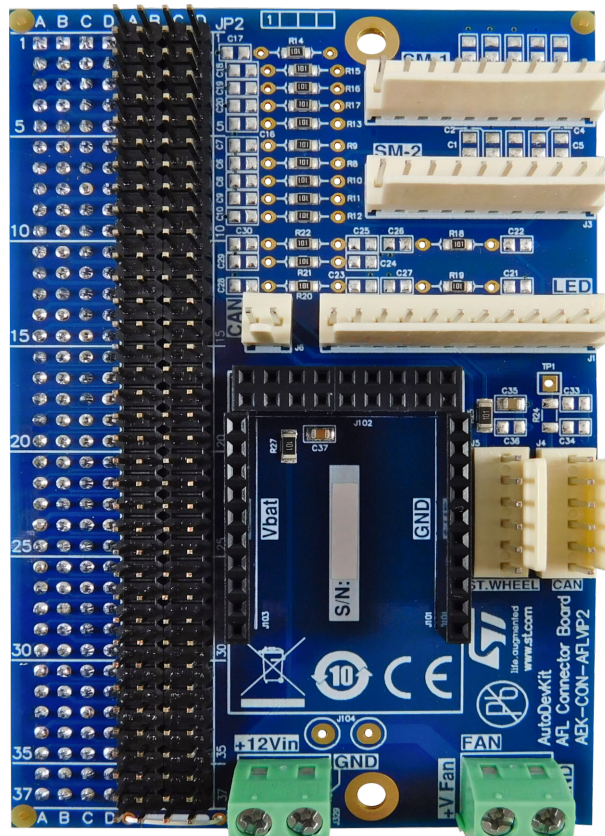
The **VN7050AS** also integrates advanced protection functions such as load current limitation, overload active management through power limitation and over-temperature shutdown. The device current limitation value is 30 A (typical) and has very low standby power. The board can accommodate a sense resistor for load current sensing.

Figure 13. EV-VN7050AS high-side driver board components


4.5 AEK-CON-AFLVIP2 connector board

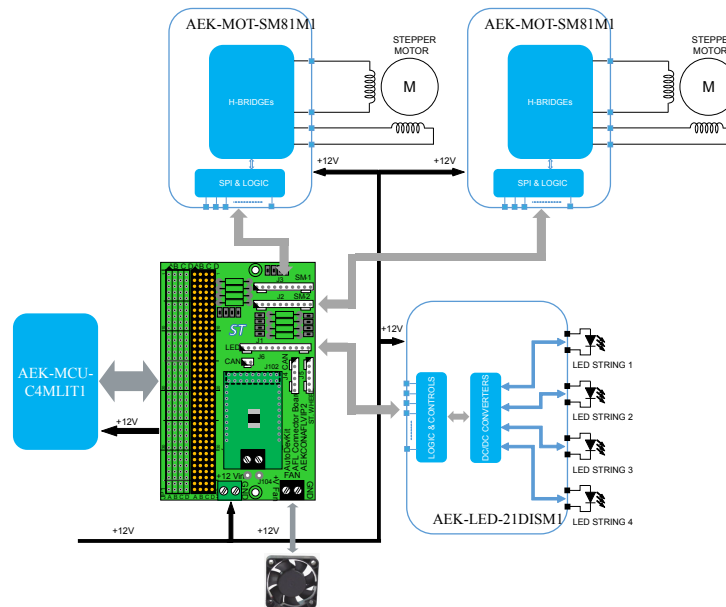
The **AEK-CON-AFLVIP2** evaluation board is designed to facilitate the connection between the boards in the Adaptive Front Light solution by ST: an MCU control board like the **AEK-MCU-C4MLIT1**, two **AEK-MOT-SM81M1** stepper motor boards, and an **AEK-LED-21DISM1** LED driver board, as well as a dedicated slot for EV-VNx boards like the **EV-VN7050AS** for cooling fan control.

Note: The female connector for the MCU control board is located on the rear side of the connector board. The 12 V supply input for the entire system can also be supplied through this board.

Figure 14. AEK-CON-AFLVIP2 connector board


The connector board is designed to allow the addition of extra functions to those already included in the AFL-Demo through the 4x37 male pin connector, which has the same electrical connections as those on the MCU board.

Figure 15. AEK-CON-AFLVIP2 block diagram



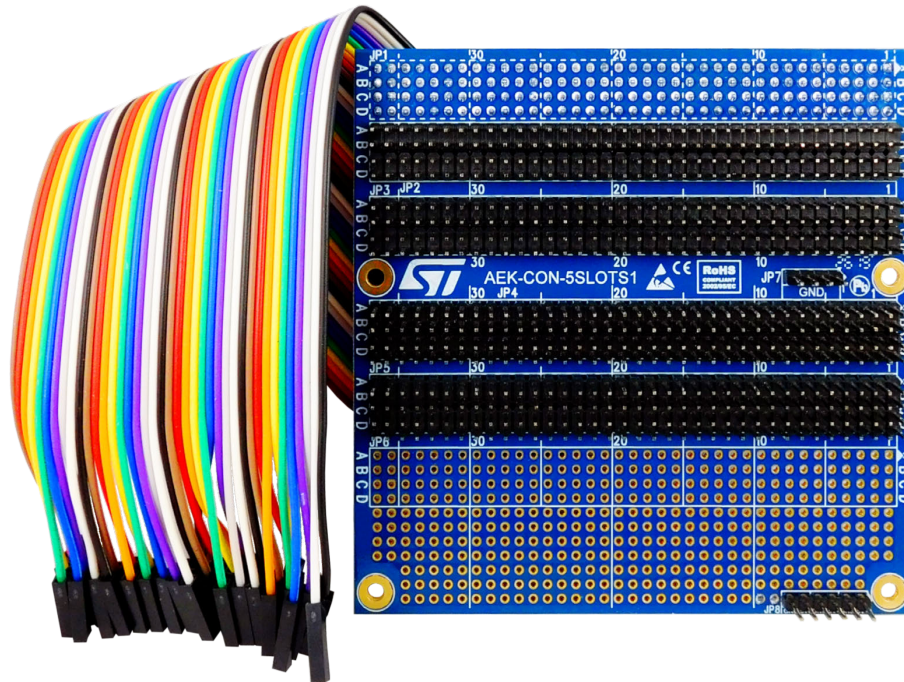
The **AEK-CON-AFLVIP2** board has the following features:

- Designed to connect the following elements for a complete automatic front lighting (AFL) adjustment solution for a LED vehicle headlight:
 - An MCU board such as the AEK-MCU-C4MLIT1 with 4x37 connector
 - Two AEK-MOT-SM81M1 stepper motor boards
 - An AEK-LED-21DISM1 LED driver control board with two L99LD21 LED drivers and providing four independent channels
 - An EV-VNx actuator board such as the EV-VN7050AS to switch a headlight colling fan on and off
- Connected boards are controlled via three separate SPIs
- Input voltage from 8 V_{DC} to 15 V_{DC}
- Board size: 65 x 100 mm. Maximum component height: 27 mm
- Cables included for two stepper motor boards and one LED driver board
- WEEE and RoHS compliant
- All ST components are qualified Automotive grade
- Part of the AutoDevKit initiative

4.6 AEK-CON-5SLOTS1 connector board

The **AEK-CON-5SLOTS1** connector board is designed to facilitate connections and reduce development time while prototyping automotive applications with an SPC5x Discovery board. The set of 4x37 extension connectors allow developers to reconfigure pin assignments for ADCs, Timers, GPIOs, DSPIs and other peripherals on SPC5 Discovery boards and generate different pin assignment patterns on the extension connectors.

Figure 16. AEK-CON-5SLOTS1 connector board

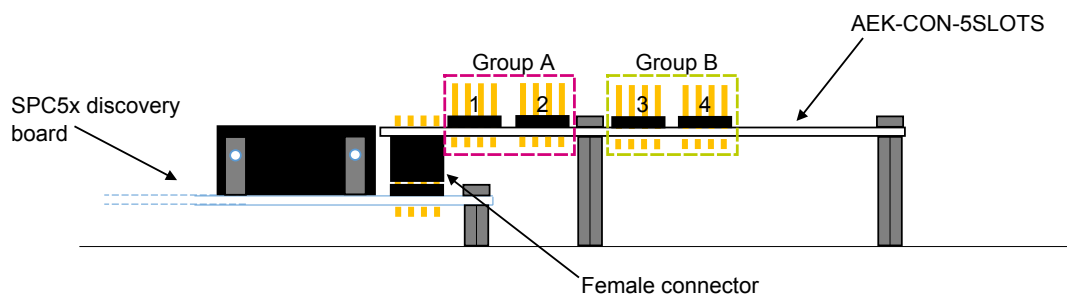


The **AEK-CON-5SLOTS1** board has the following features:

- Fast and easy extension of 4x37 connector on SPC5x discovery boards
- Single female connector to plug the board onto an SPC5x 4x37 connector
- Two male connectors with same outputs as 4x37 connector
- Two configurable male connectors 4x37 whose pins can be reassigned to connect different functional boards
- Additional ground and 5V pins
- Bare section available on the board for specific application purposes
- Female-to-female jump wire set included
- Part of the AutoDevKit initiative

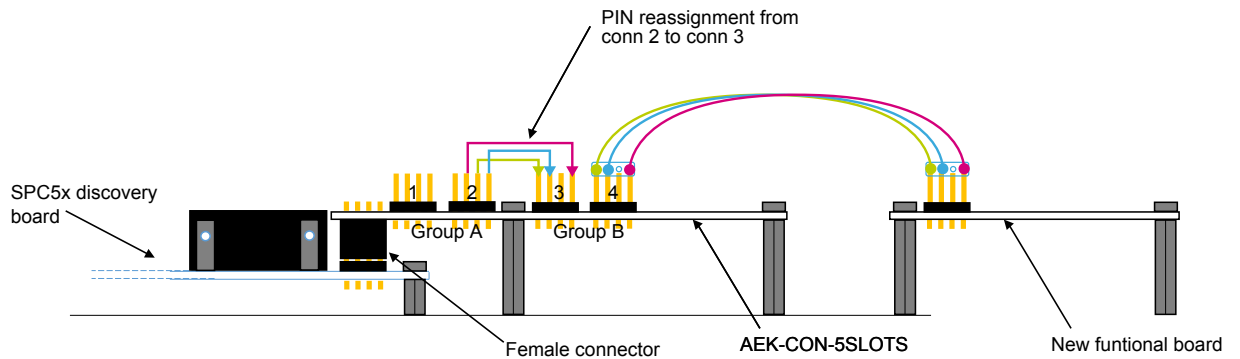
Regarding the following image showing the connector Groups, the two Group A connectors, 1 and 2, share a direct connection, as do the two Group B connectors, 3 and 4; the **AEK-CON-5SLOTS1** is plugged directly on the SPC5x Discovery board via the 4x37 female connector on the bottom side of the board.

Figure 17. AEK-CON-5SLOTS1 connector groups



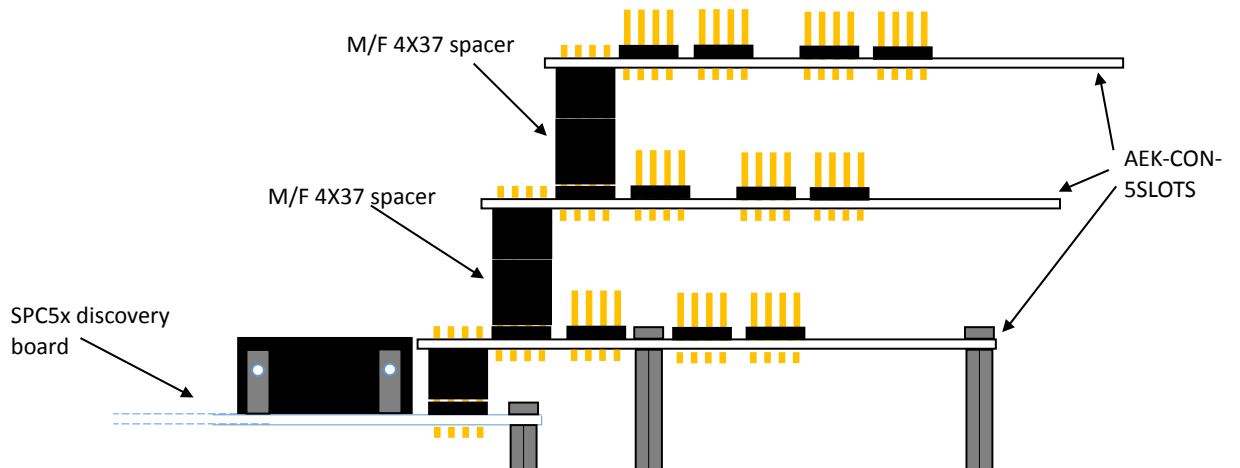
The configuration of the MCU pins is transferred to the 4x37 male connectors 1 and 2 of the **AEK-CON-5SLOTS1** board. The pins on connector 2 can then be reassigned using jumper wires to new pins on connector 3. As connectors 3 and 4 are electrically connected, an external functional board compatible with the new pin assignment can be plugged to connector 4.

Figure 18. AEK-CON-5SLOTS1 pin reassignment for external function boards



The unused connector 1 in group A offers the possibility to stack more extension connector boards on top of each other.

Figure 19. Multiple AEK-CON-5SLOTS1 extension boards mounted on SPC5x Discovery board



The AEK-CON-5SLOTS1 board also has a breadboard area to add simple circuits necessary to complete the prototype application.

5 AFL-Demo software package

The [AFL-Demo](#) software package consists of the following components:

- The [STSW-AFL001](#) firmware with the AFL-Demo code to be loaded on the [SPC58EC80](#) microcontroller.
- SPC58Cxx_RLA Adaptive Front Lighting(AFL) project and driver files, including:
 - SPC5 AEK_LED21DISIM1 Component RLA
 - SPC5 AEK_MOT_SM81M1 Component RLA
 - SPC5 EV_VNx7xxx Component RLA
 - SPC58Cxx Low Level Driver

RELATED LINKS

The latest version of the software is available from the [STSW-AFL001](#) page on the ST website

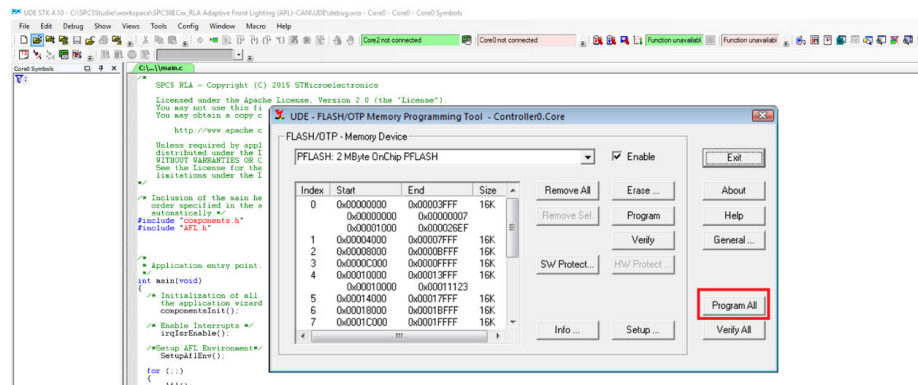
Discover the full range of available components from the [AutoDevKit](#) page on the ST website

6 How to program the SPC58EC80 microcontroller

Follow this procedure to load the latest version of the [STSW-AFL001](#) firmware on the [AEK-MCU-C4MLIT1](#) discovery board. You may skip this procedure if you have the [AEKD-AFLPANEL1](#) kit, as the software is already loaded on the discovery board included in the kit.

- Step 1.** Download and install [SPC5-UDESTK-SW](#) debugger from [www.st.com](#).
- Step 2.** Connect the mini-B USB cable between your PC and the [AEK-MCU-C4MLIT1](#) discovery board.
- Step 3.** Turn-on the discovery board.
- Step 4.** Run the UDE application on your PC.
- Step 5.** In the UDE program select, **[File]>[Open]** 'afl.wsx'.
afl.wsx is found in the UDE directory of the code zip; it is a hook file for the Universal Debug Engine (UDE) to burn the [AFL-Demo](#) code in the SPC58 microcontroller Flash memory.
As soon as the file is loaded a new window appears.
- Step 6.** Click **[Program All]** in the new window.
A confirmation message appears in the same window then the operation is completed.
The new firmware is now flashed in the microcontroller.
- Step 7.** Exit the window.
- Step 8.** Reset the [AEK-MCU-C4MLIT1](#) discovery board to run the updated firmware.

Figure 20. Flashing the AFL firmware using UDE STK



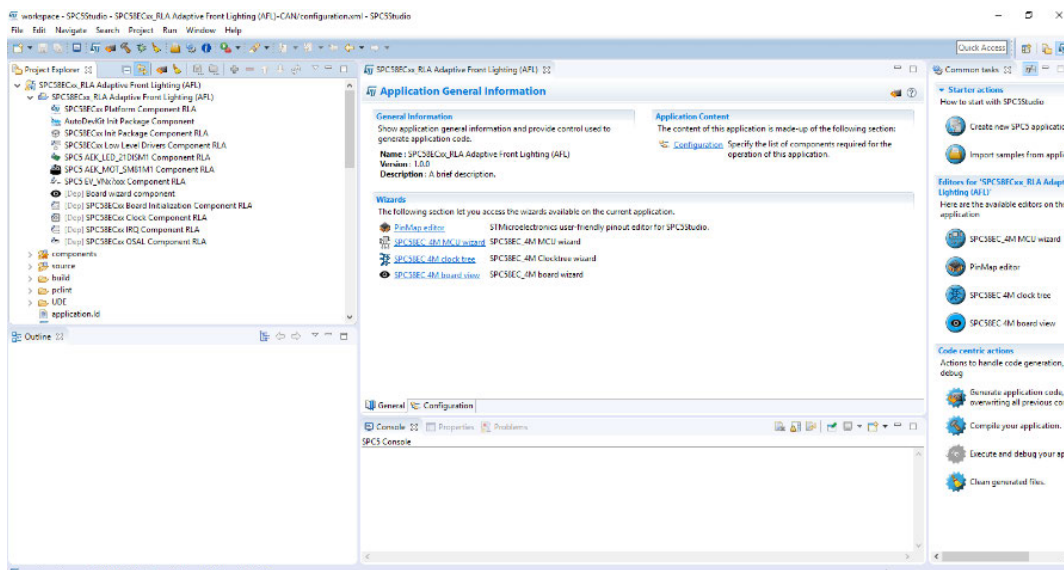
7 SPC5-STUDIO overview

SPC5-STUDIO is an integrated development environment (IDE) based on Eclipse. It contains a standard workspace and an extensible plug-in system environment customization.

The aim of SPC5-STUDIO is to maximize developer productivity of embedded applications based on SPC5 Power Architecture 32-bit microcontrollers with a single tool for evaluation, development, design and production.

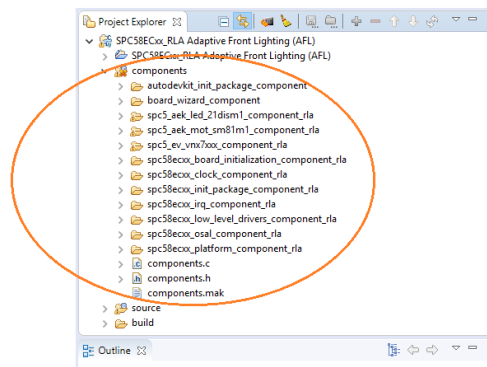
SPC5-STUDIO includes an application wizard to simplify project creation and configuration; it automatically solves component dependencies and generates support files.

Figure 21. SPC5-STUDIO AFL project workspace



The application wizard integrates the initial components into the project, with the key elements employed by SPC5-STUDIO to generate the final application source code. Typical of layered architecture, the services in one component are provided to other components. Furthermore, the configuration of each component is supported by an intuitive GUI.

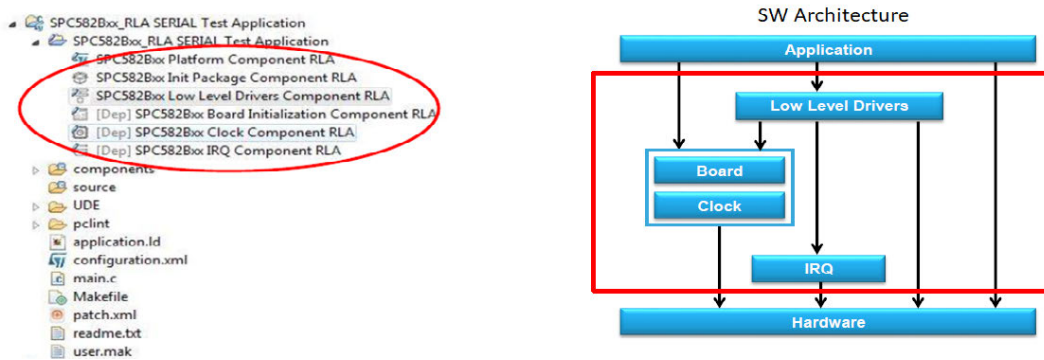
Figure 22. SPC5-STUDIO AFL project components



Register Level Access (RLA) components are low level drivers with direct access to the MCU and peripherals such as CAN, Ethernet, DSPI, ADC, PIT and GTM. The RLA components can be added and configured through a time-saving GUI.

Note: The RLA source code is fully MISRA 2012 compliant.

Figure 23. SPC5-STUDIO AFL RLA drivers



The FreeRTOS open source real time operating system is available on request as a separate component that is compatible with the rest of the environment.

SPC5-STUDIO also contains straightforward software examples for each peripheral in the MCU, which developers can use to become familiar with the specific code involved.

Other advantages of SPC5-STUDIO include:

- the ability to integrate other software products from the Eclipse standard market place
- the availability of a free license GCC GNU C Compiler component
- the possibility to support industry-standard compilers
- support of multi-core microcontrollers
- a PinMap editor to facilitate MCU pin connections

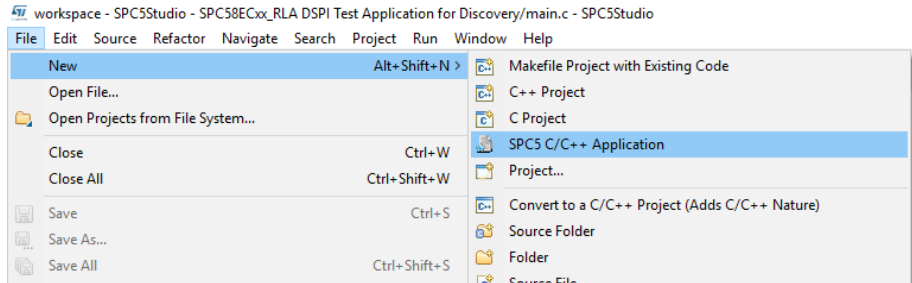
7.1 How to create a new project in SPC5-STUDIO

The following procedure provides step-by-step guide to create a new project in SPC5-STUDIO.

Step 1. Install SPC5-STUDIO (downloadable from www.st.com).

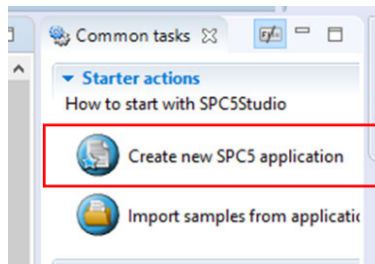
- Step 2.** Create a new SPC5 application:
- Select **[File] [New] [SPC5 C/C++ Application]**

Figure 24. New Project menu



- Or select the icon “Create a new SPC5 application” present in the Starter actions tab.

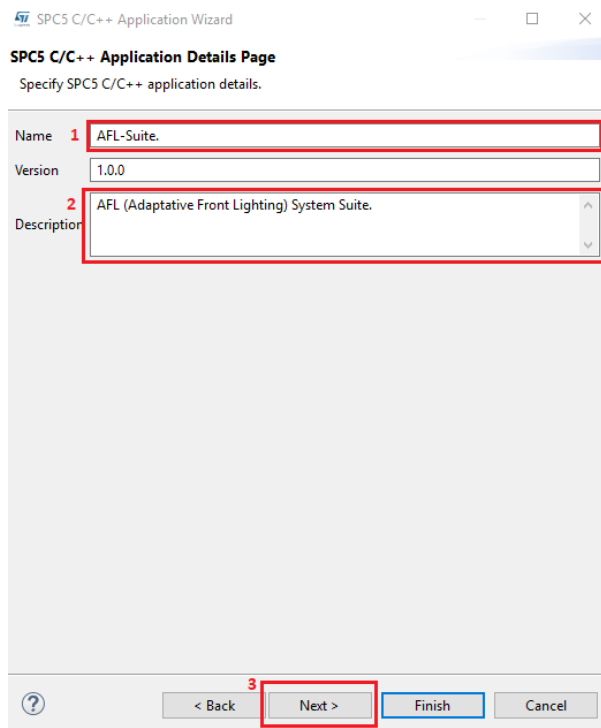
Figure 25. Starter actions tab



A window appears, prompting for application details.

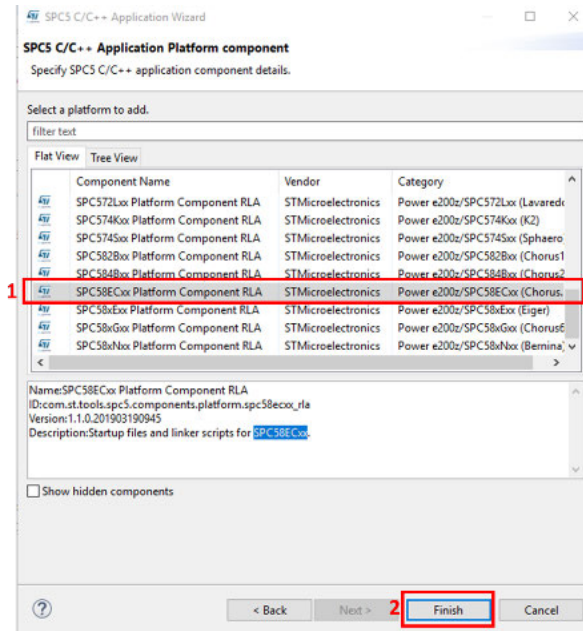
- Step 3.** Fill in the application details and then click on the **[Next]** Button.

Figure 26. New application details page



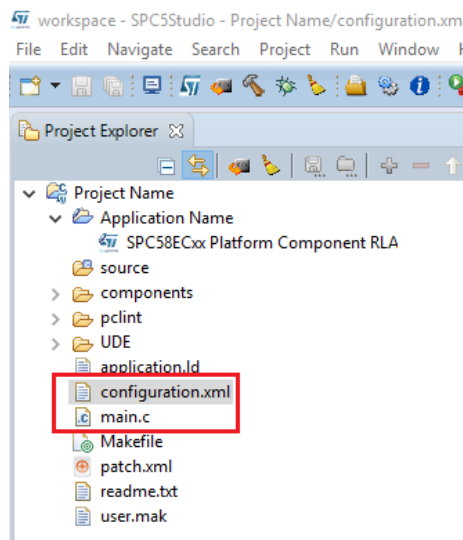
Step 4. Select the SPC58Cxx component and select the **[Finish]** button.

Figure 27. New application MCU platform



Step 5. Click on the **[Generate application code]** button.
A base C project is created in the workspace.

Figure 28. New SPC5 application files in outline view



There are two fundamental files visible in the application tree view:

- configuration.xml with project configuration information that is updated every time the project is changed.
- main.c in which the actual application is implemented. The main.c file only contains a basic initialization section and an infinite loop when it is first created.

Figure 29. main.c application file

Main.c automatically generated

```

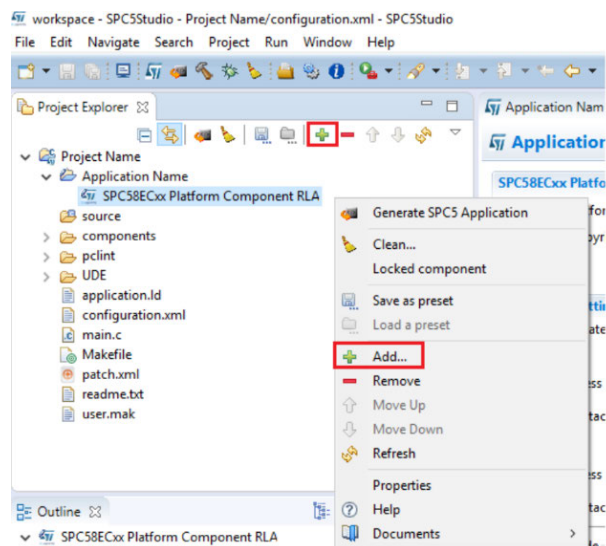
main.c
SPC5 RLA - Copyright (C) 2015 STMicroelectronics
16
17 /* Inclusion of the main header files of all the imported components in the
18 order specified in the application wizard. The file is generated
19 automatically.*/
20 #include "components.h"
21
22 /*
23 * Application entry point.
24 */
25 int main(void) {
26
27 /* Initialization of all the imported components in the order specified in
28 the application wizard. The function is generated automatically.*/
29 componentsInit();
30
31 /* Application main loop.*/
32 for ( ; ; ) {
33
34 }
35 }
36
    
```

7.2 How to add components to an SPC5-STUDIO project

The following procedure shows how to add an available component to a project.

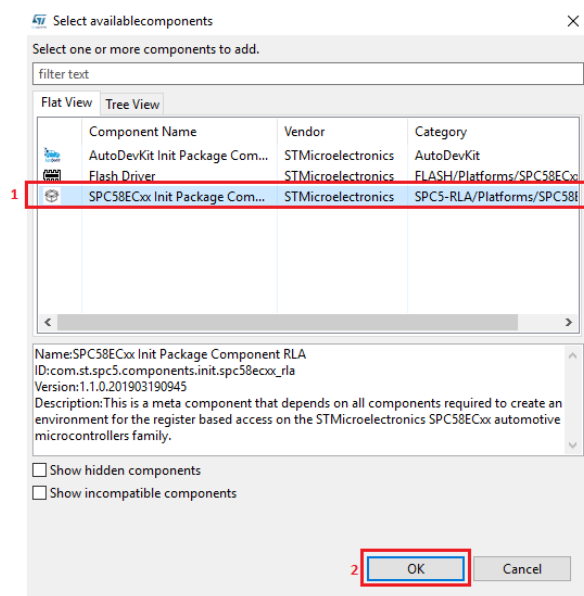
- Step 1.** In the Project Explorer tab, select the SPC58Cxx Platform component RLA.
- Step 2.** View the available components for the chosen platform:
- right click on the mouse and select add, or
 - select the + icon in the project explorer

Figure 30. Open available components list



- Step 3.** Select the components you want to add and click the [OK] button.
 You must at least add the following two components to your SPC5-STUDIO Project:
- SPC58Cxx Init Package Component
 - Low Level Driver

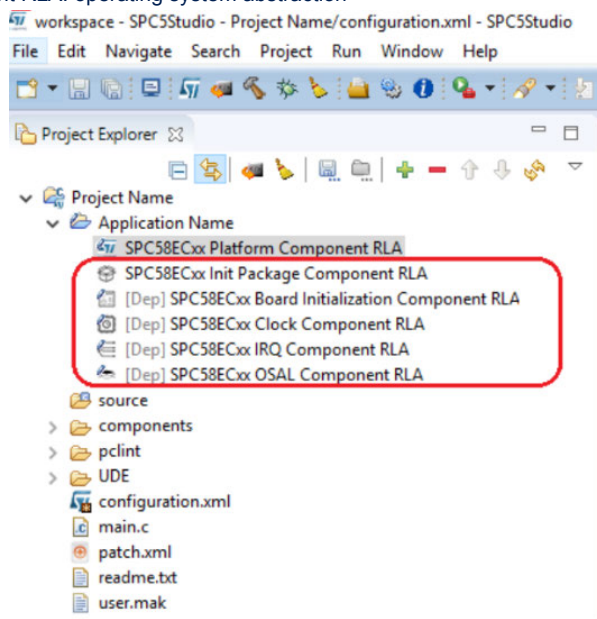
Figure 31. Select available components for platform



The added components appear in the Application Folder:

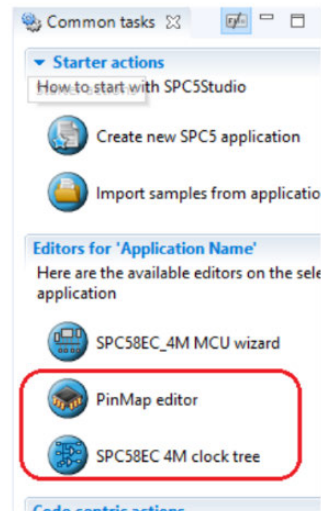
Figure 32. New components visible in project tree

- SPC58Cxx Board Initialization Component RLA: initialization and the configuration of the selected board.
- SPC58Cxx Clock Component RLA: for the configuration of the MCU clock tree
- SPC58Cxx IRQ Component RLA: to set and configure interrupt Request QUEUE
- <SPC58Cxx> OSAL Component RLA: operating system abstraction



The PinMap Editor icon also becomes selectable in the Editors tab:

Figure 33. PinMap editor icon in Editors tab



Repeat the procedure to add other components to your project.

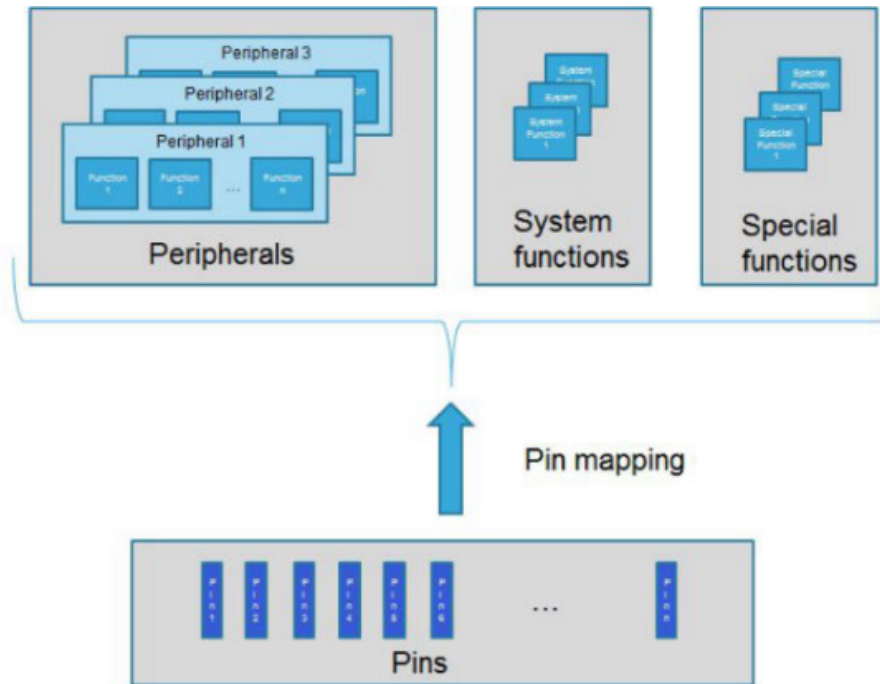
7.3 PinMap Editor

Pins in SPC5 microcontrollers are identified as Ports because they can have different functions depending on the configuration settings in certain MCU registers:

- Peripheral related pins such as CAN, ADC, eMIOS.
- System function pins such as RESET (cannot be modified).
- Special function pins such as supply voltage pins.

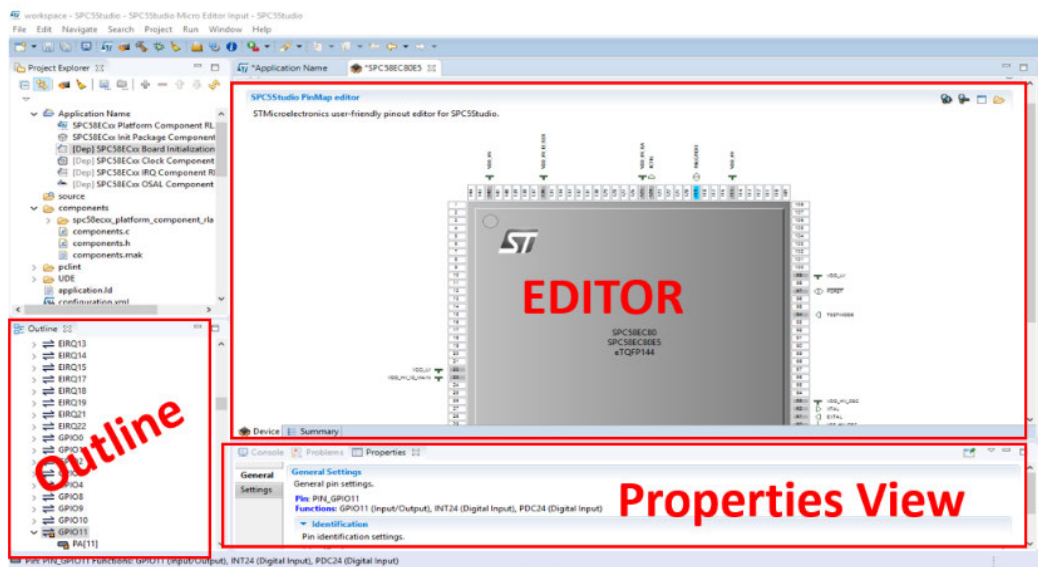
The PinMap editor supports pin functionality selection by providing the possible configurations for each selected pin.

Figure 34. SPC5 MCU pin functions



The PinMap editor features Editor, Outline and Properties windows.

Figure 35. PinMap editor interface windows



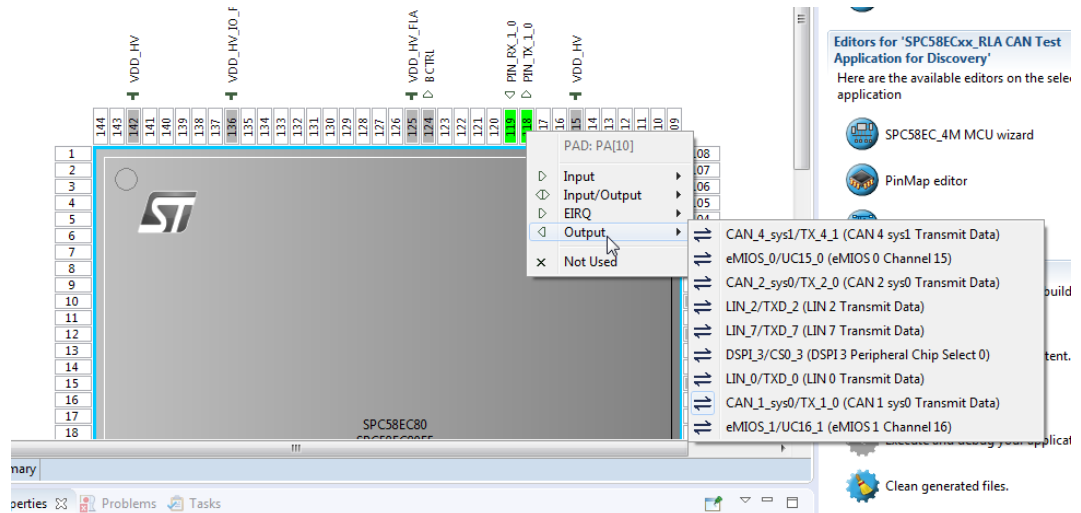
7.3.1 How to use the Editor view

The Editor view is the largest window, where the pin configuration can be performed with the help of a graphical representation of the selected MCU. The following procedure shows how to configure a pin in the Editor view.

Step 1. In the main editor window of the PinMap editor, right click on the pin you wish to edit.

Step 2. Select a signal direction and a corresponding function from the list of pre-loaded configurations that may be associated with the pin.

Figure 36. Pin signal function selection

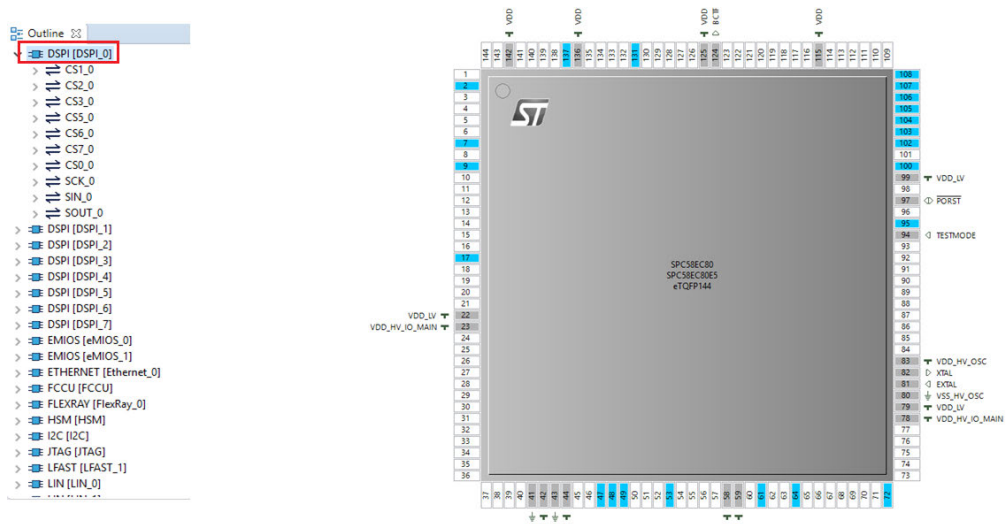


7.3.2 How to use the Outline view

The Outline view is presented in a Device → Function → Pin hierarchy, which lists all the eligible pins for a particular peripheral function.

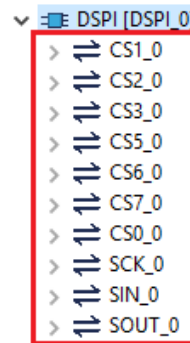
Step 1. In the Outline window select the select the DSPI peripheral.

Figure 37. Selected peripheral in Outline view and eligible pins in Editor view



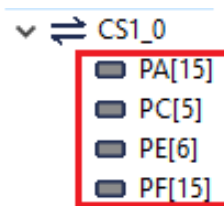
Step 2. Select an appropriate signal for the peripheral.

Figure 38. Signals for DSPI peripheral



Step 3. Select one of the pins available for the chosen signal.
The corresponding MCU ports for each pin are shown in square brackets.

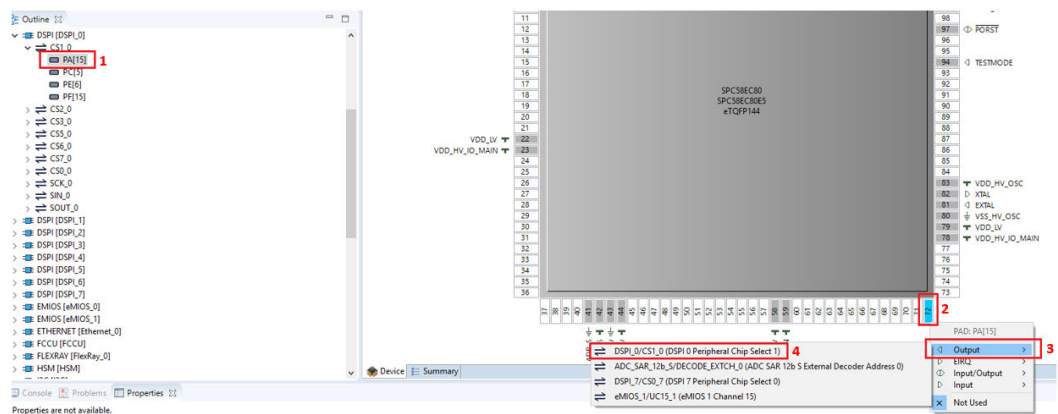
Figure 39. Pins for selected peripheral signal



Step 4. In the Editor window, right-click on the selected pin → select the signal direction → select preloaded configuration.

Step 5.

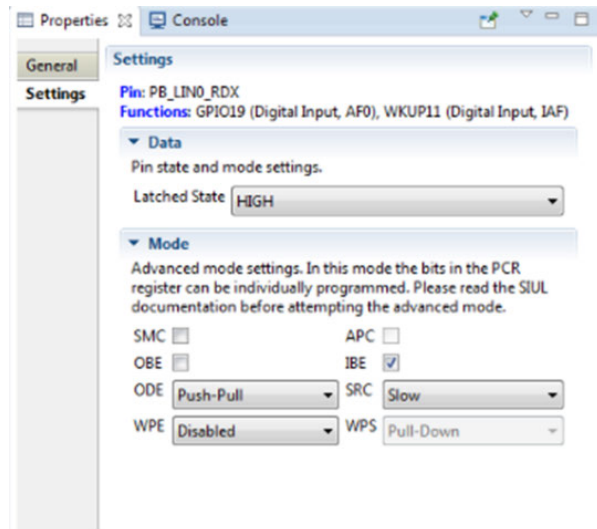
Figure 40. Pin configuration in Editor view



7.3.3

The Properties view

This view provides further information for a selected pin, including name, pin state and mode settings.

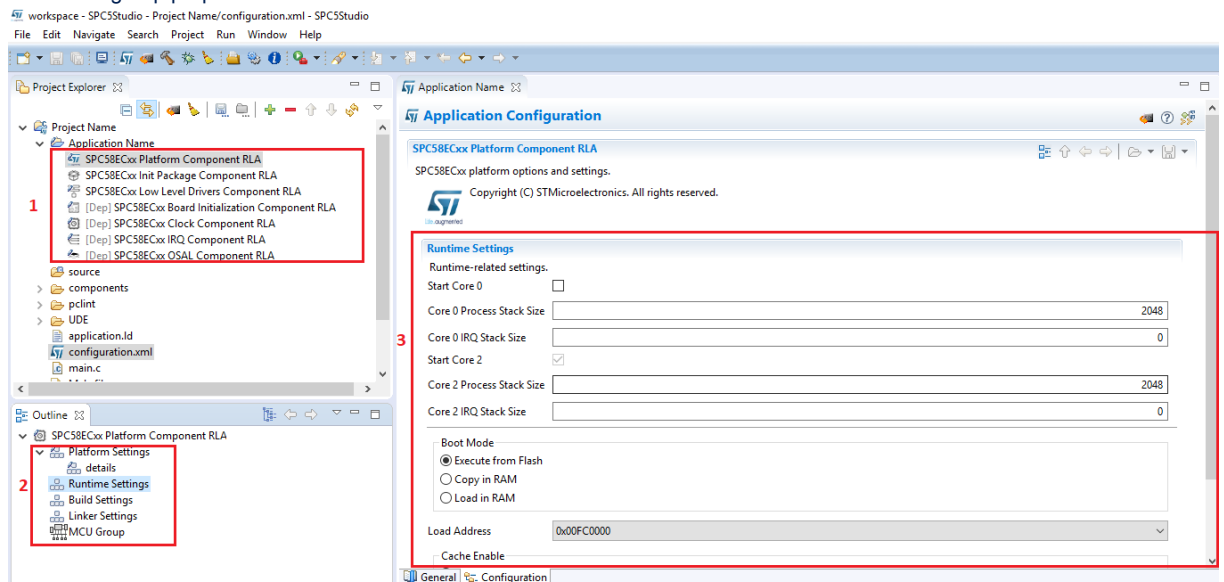
Figure 41. PinMap Properties view


7.4 Component configuration in SPC5-STUDIO

SPC5-STUDIO includes a Configuration viewer with default component configuration values that may or may not be editable.

Figure 42. Component configuration sequence

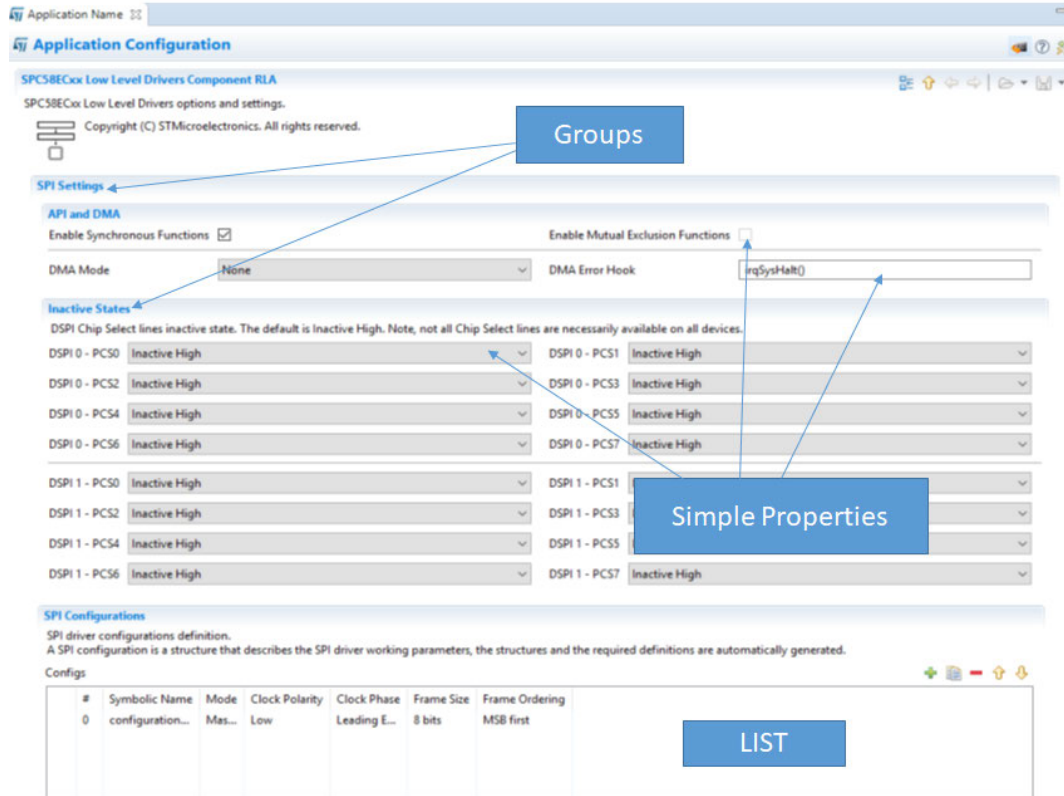
1. Select a component to configure from your project
2. Select a group in the Outline view
3. Edit the group properties



Note: Be sure to save the project whenever you set or change a property in order to keep the changes.

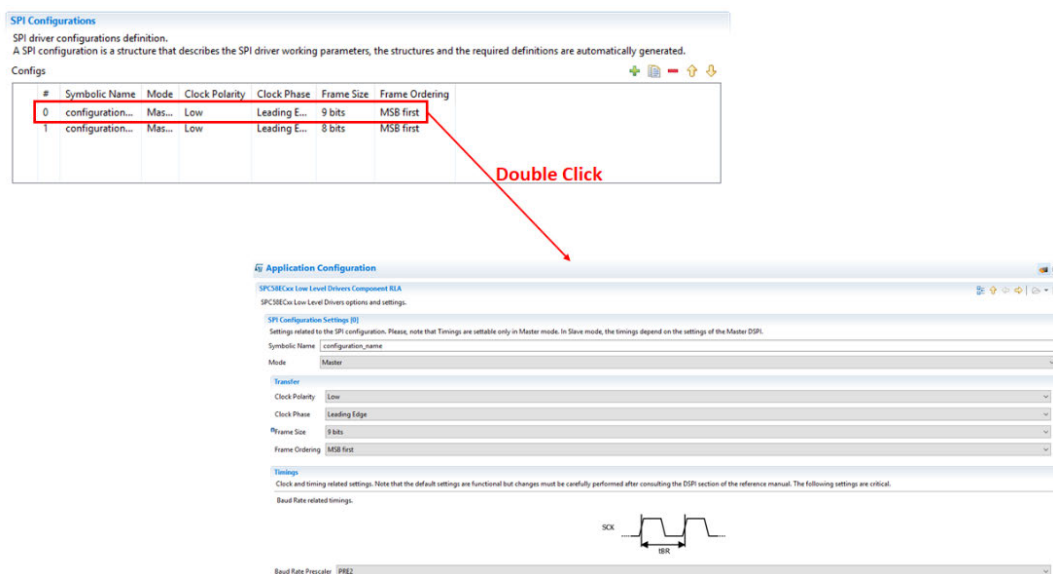
Figure 43. Simple, group and list property fields

Example of property fields



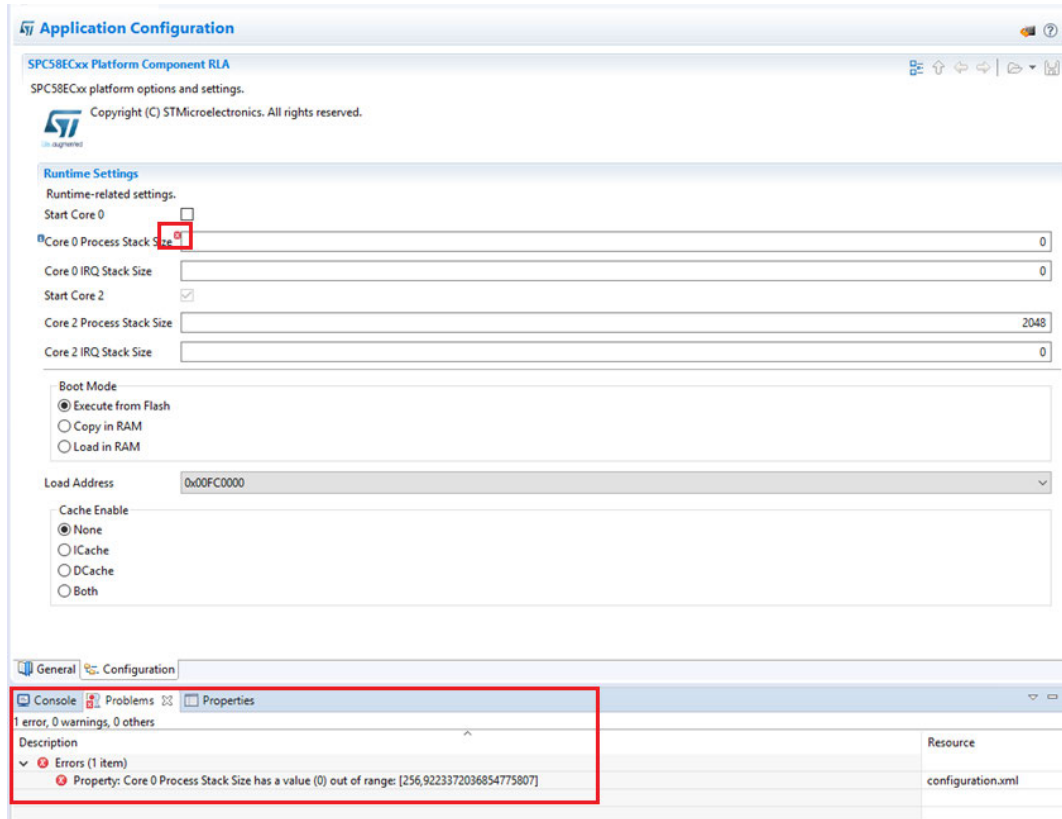
You can edit the content of a list item by double-clicking on it.

Figure 44. Editing list items



Values assigned to a component property are automatically validated and a red “x” appears in the top left corner of the name if an error is detected; a corresponding explanation is provided in the Problems tab.

Figure 45. Validation of property values



When the configuration of the embedded system is completed and there are no errors in the Problem tab, you can select the **[Code generation]** command to output the configuration files.

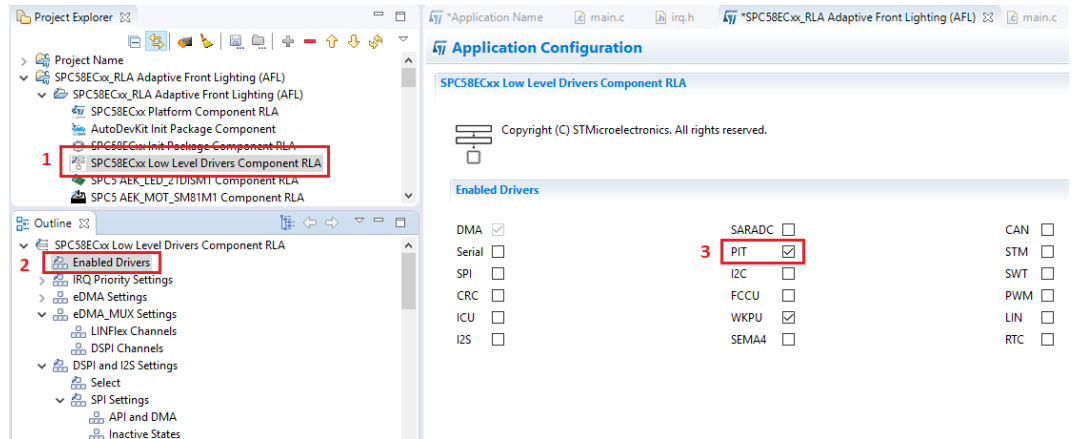
7.5 How to set low level driver IRQ priority settings

The SPC58 Chorus MCU used in the AFL-Demo kit has several interrupts of different types, which can be associated with events linked to peripherals such as DSPI, LIN, PIT, or to other components. These interrupts should be prioritized in order to avoid potential conflicts.

- Step 1.** Select the low level driver component.
- Step 2.** In the Outline tab, select Enabled Drivers.

Step 3. Check the drivers to be enabled.

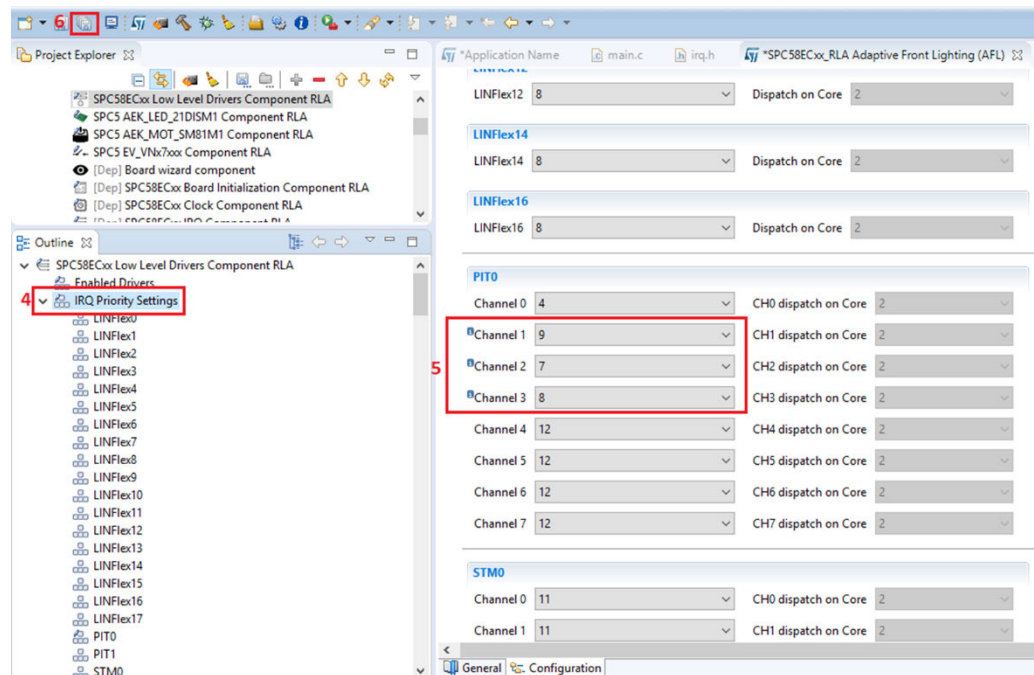
Figure 46. Enabling low level drivers



Step 4. In the Outline tab, select IRQ Property Settings

Step 5. In the configuration Editor, search the group that matches the driver and change the priority according to your needs.

Figure 47. IRQ priority settings



7.6 How to generate and compile application source code

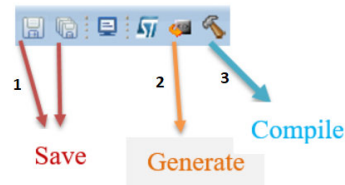
To generate and compile the application source code, follow the procedure below described:

Step 1. Save the project.

Step 2. Click on the **[Generate]** icon to update the configuration files.

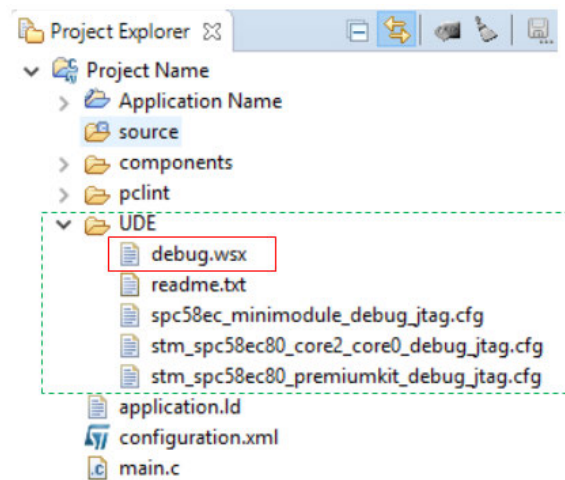
Step 3. Click on the **[Compile]** icon to compile the project source and produce the MCU Flash image.

Figure 48. SPC5-STUDIO save, generate and compile icons



Once the source code is compiled and linked, a file named `debug.wsx` will appear in the UDE folder.

Figure 49. Compiled project files



Note: Users implement their application in the `main()` function contained in the `main.c` file, which is automatically created by SPC5-STUDIO in the project creation phase.

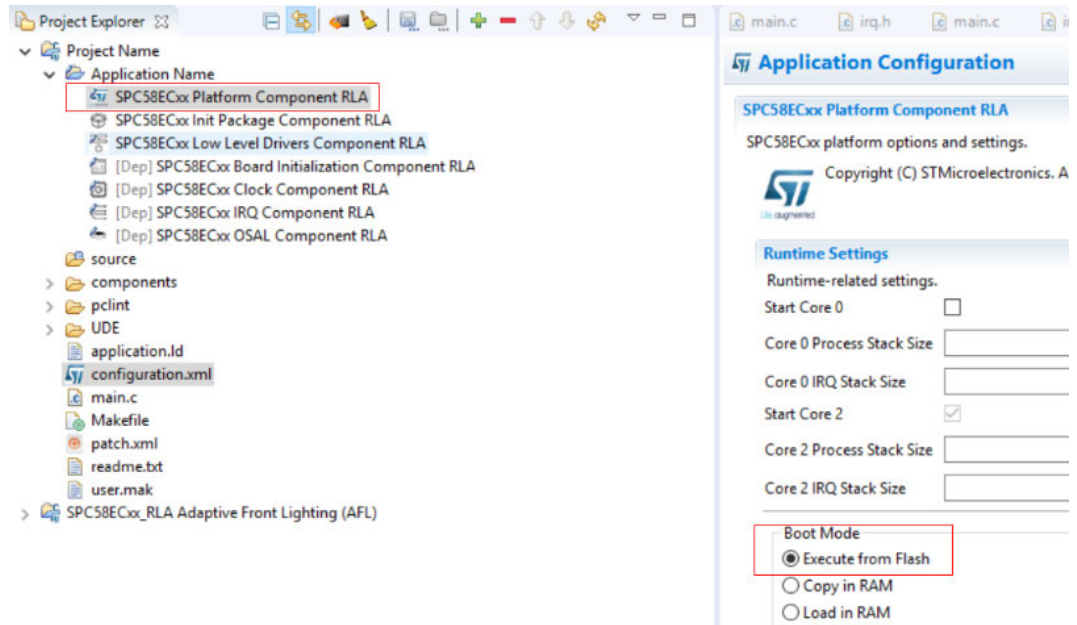
7.7 How to debug an application generated with SPC5-STUDIO

Before you begin debugging, you should install Universal Debug Engine (UDE) tool, which you can download from the [SPC5-UDESTK-SW](#) web page on the ST website.

The procedure below shows how to debug the code generated in SPC5-STUDIO using the (UDE) tool.

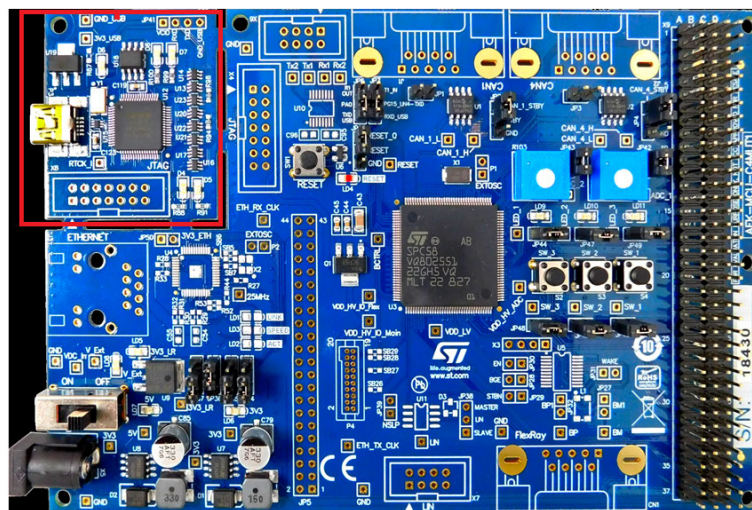
- Step 1.** In **SPC5-STUDIO**, check the **[Execute from Flash]** radio button in the Boot Mode section of the Application Configuration view.

Figure 50. Boot Mode radio button in Application Configuration view



- Step 2.** Compile the project in **SPC5-STUDIO**.
- Step 3.** Connect a USB cable between the mini-USB connector of the PLS on the **AEK-MCU-C4MLIT1** board and the PC.

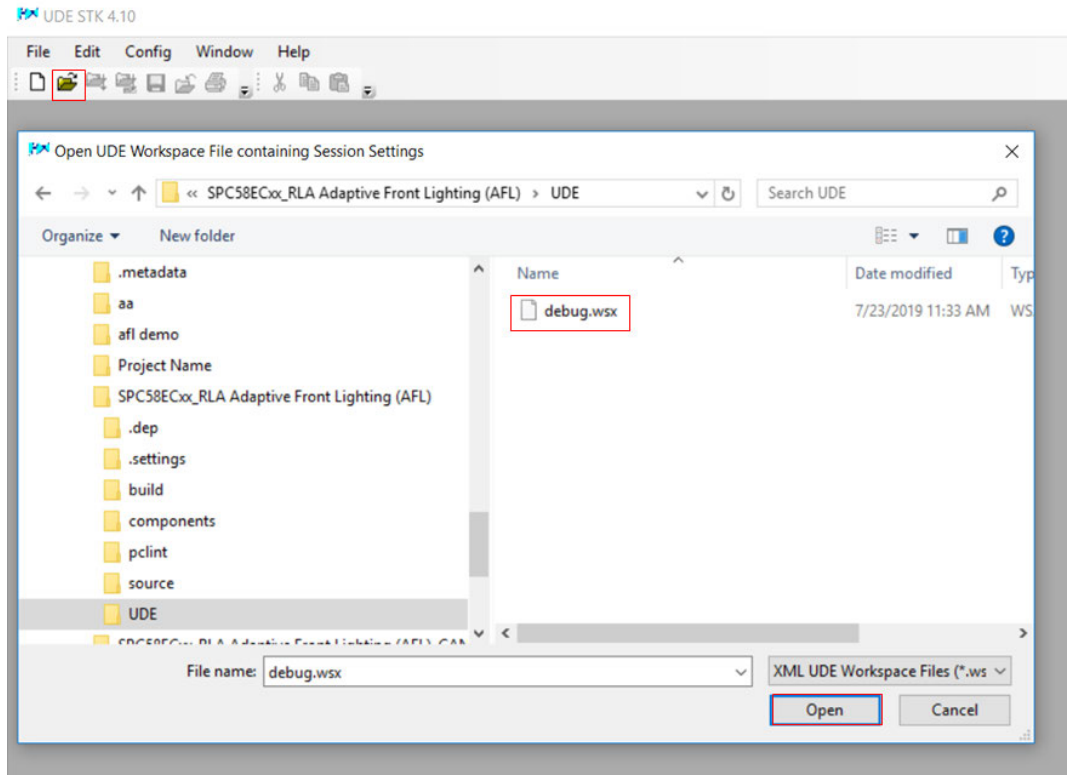
Figure 51. mini-USB port on AEK-MCU-C4MLIT1 board



- Step 4.** Run the UDE STK tool.
- Step 5.** Open the project workspace folder and select debug.wsx from the UDE folder.

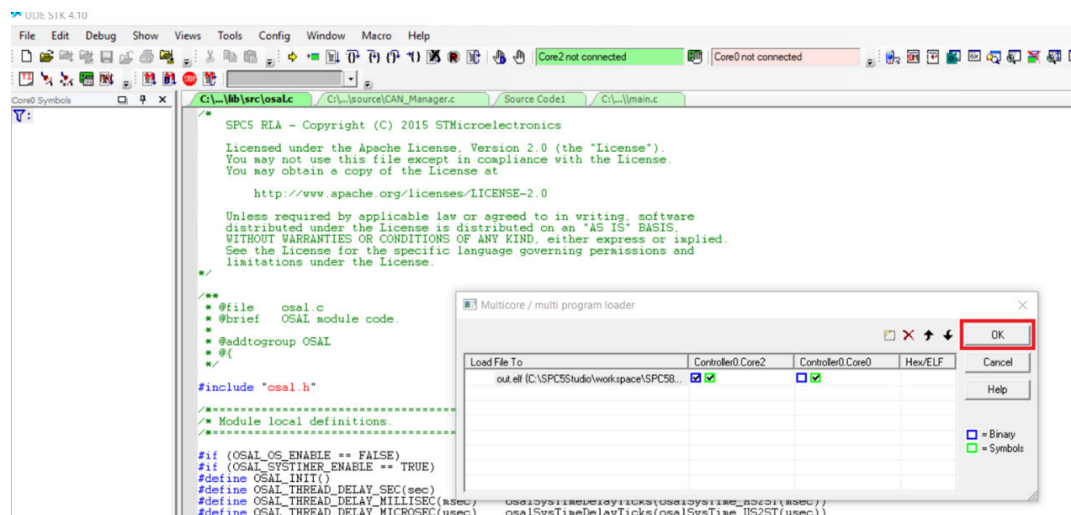
Step 6. Click the [Open] button.

Figure 52. Open debug.wsx in the UDE STK debugging tool



Step 7. Click [OK] in the popup window.

Figure 53. UDE STK program loader



Step 8. Proceed to debug using the command buttons in the Debug task bar or under the Debug menu.

Figure 54. UDE STK debug command buttons



8 AutoDevKit

The AutoDevKit plug-in for Eclipse extends [SPC5-STUDIO](#) into a straightforward, low-cost, time saving tool designed to help automotive application engineers evaluate, prototype, develop and deploy complex embedded systems.

The main advantages of the AutoDevKit initiative are:

- Fast prototyping: with integrated hardware and software components, component compatibility checking and MCU and peripheral configuration tools.
- Flexibility: allowing the creation of new system solutions from existing solutions by adding dedicated boards for further functionality or removing unused boards.
- Hardware abstraction: if, for example, you swap your MCU between an SPC58EC Chorus 4M, an SPC584B Chorus 2M, or an SPC582B Chorus 1M, the tool will automatically re-generate pin allocations accordingly.
- High-level application APIs: AutoDevKit provides a set of function APIs able to control specific functional boards. The APIs contain functions able to exploit the chosen hardware functionality and facilitate communication with the MCU.

AutoDevkit provides a GUI to facilitate the configuration and setup for each supported component, as well as sample code demonstrating typical usage and applications.

Another key feature is the Board View tab, which shows the electrical wiring between microcontroller and board connectors and the connection between the MCU board and other functional boards.

8.1 AutoDevKit installation

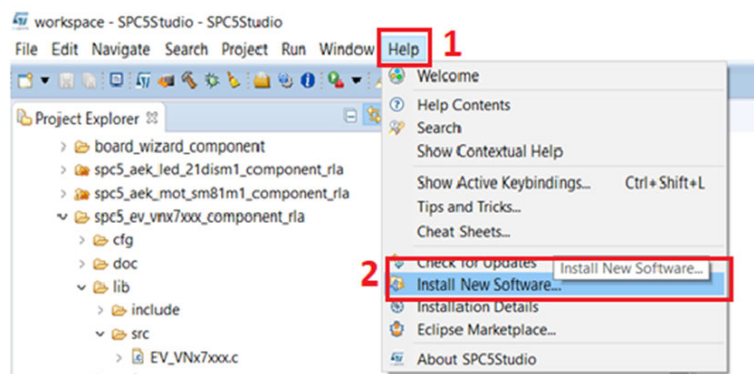
Before you begin, download AutoDevKit from the [STSW-AUTODEVKIT](#) web page on the ST website.

Follow the steps below to install AutoDevKit.

Step 1. Open [SPC5-STUDIO](#).

Step 2. Select [Help]>[Install New Software].

Figure 55. SPC5-STUDIO install new software

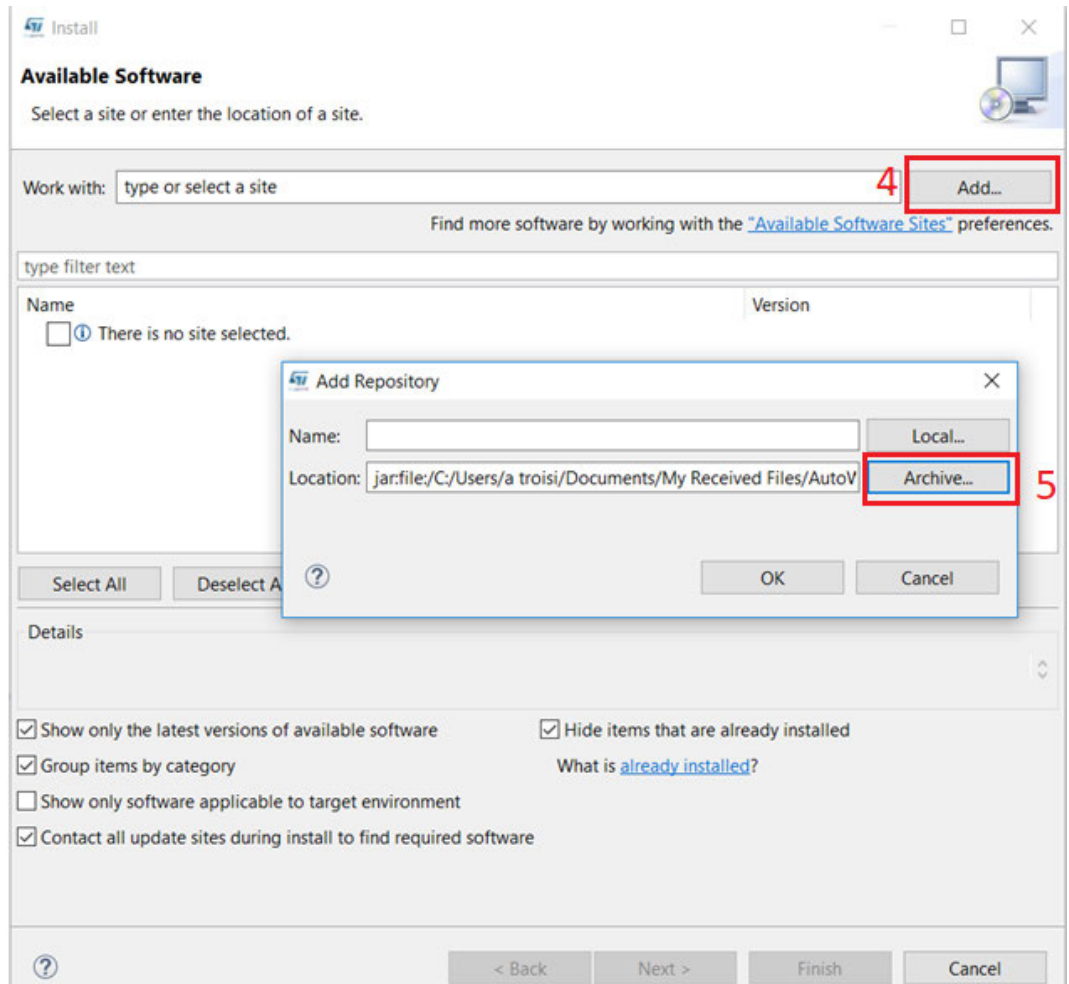


Step 3. Click the [Add] button in the Install window.

- Step 4.** Click the **[Archive]** button, in the Add Repository window and select the path on your pc where the AutoDevKit package is saved.

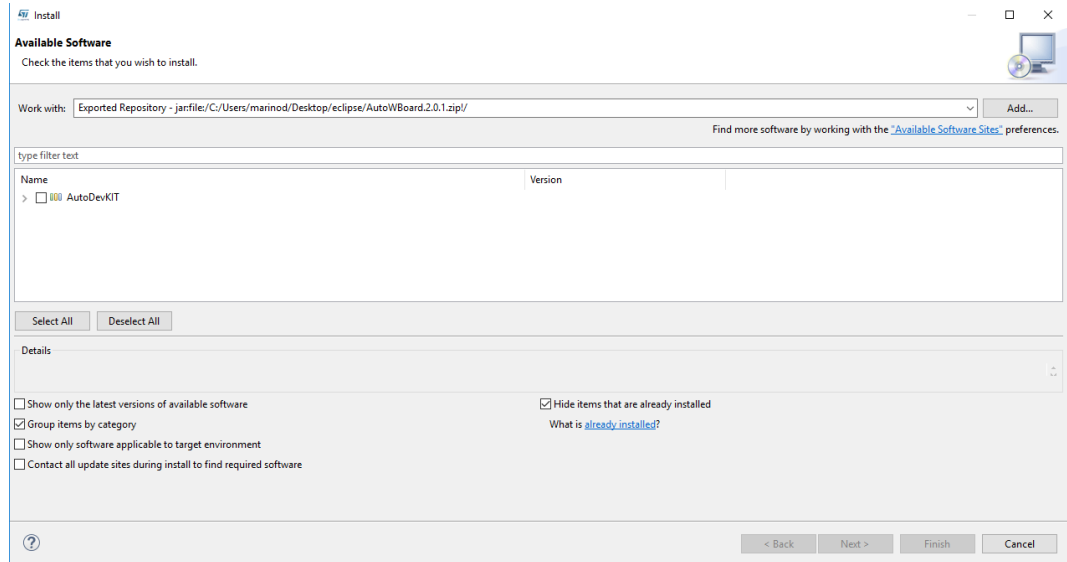
Figure 56. Select software zip location

click Add
select archive



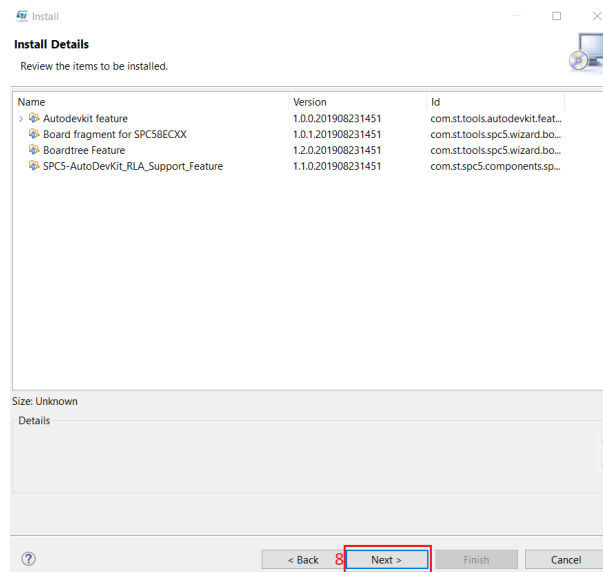
Step 5. Check the AutoDevKit box and click **[Next]**.

Figure 57. Select AutoDevKit for installation



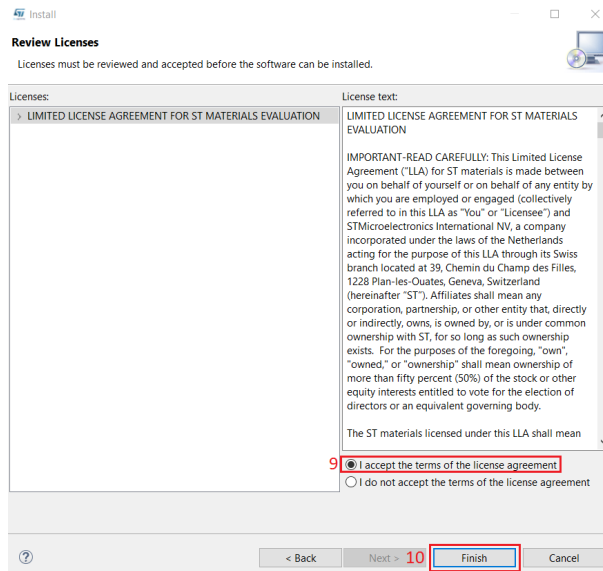
Step 6. Click on the **[Next]** button in the Install Details window.

Figure 58. AutoDevKit installation details



Step 7. Read and accept the license terms if you agree to them, and select **[Finish]**.

Figure 59. AutoDevKit license terms



8.2 Adding AutoDevKit components to an SPC5-STUDIO project

AutoDevKit adds new items to the list of the components that can be used in a **SPC5-STUDIO** project. AutoDevKit components are added in the same way as other components are added to an SPC5 Studio project. Each functional board component only appears in the list of the MCUs able to support it, and there is a Release Note component for all MCU platforms, which provides a summary of the components installed with the AutoDevKit plugin.

RELATED LINKS

- [7.4 Component configuration in SPC5-STUDIO on page 33](#)
 - [Check the AutoDevKit page on the ST website for the latest updates](#)
-

8.3 How to configure an AutoDevKit component

The configuration of an AutoDevKit component is very similar to configuring other components in **SPC5-STUDIO**, the difference being two new features in the Configuration View for Pin Allocation:

- Allocation
- Deallocation

These new features automatically allocate or deallocate available MCU pins to allow the use of the MCU peripherals by newly added components.

To enable these two features, you must first configure the newly added component. The procedure is best illustrated with the following example, in which we add and configure a LED Driver component **AEK-LED-21DISM1** that manages four LED strings.

Step 1. Add the AEK_LED_21DISM1 component to the project.

Figure 60. Add LED driver component to project



Step 2. Follow the steps below to configure the component directly.

This is different to the normal handling of SPC5-STUDIO components where we usually proceed to verify pin allocation in the PinMap Editor.

Step 3. Click on the [+] button available at the top of the AEK_LED_21DISM1 table.

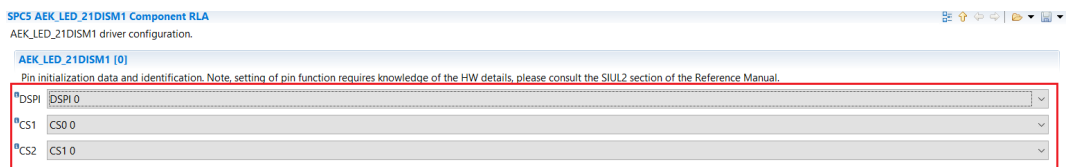
Step 4. Double-click on new DSPI object to open its configuration view.

Figure 61. Add new DSPI object



Step 5. Select a value from the drop-down lists according to your requirements.

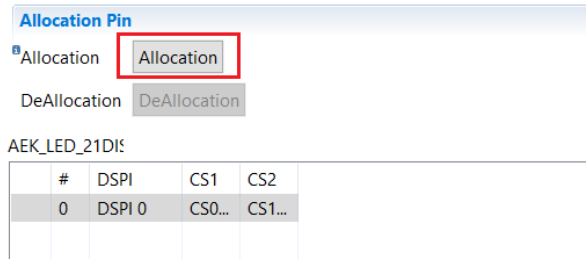
Figure 62. LED driver board pin configuration



Step 6. Select the  button to return to the parent screen.

Step 7. Click the **[Allocation]** button to delegate the automatic pin allocation to AutoDevKit.

Figure 63. Pin allocation button



Step 8. Click **[OK]** in the confirmation window.
You can open the PinMap editor to check that the pins have been allocated correctly.

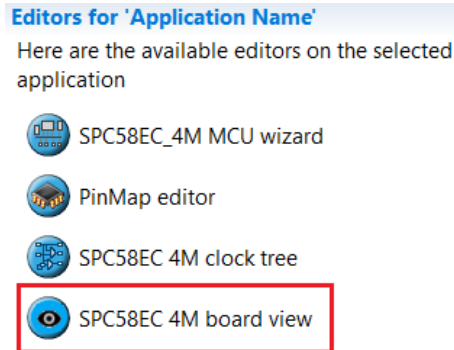
RELATED LINKS

[7.4 Component configuration in SPC5-STUDIO on page 33](#)

8.4 Board View tab

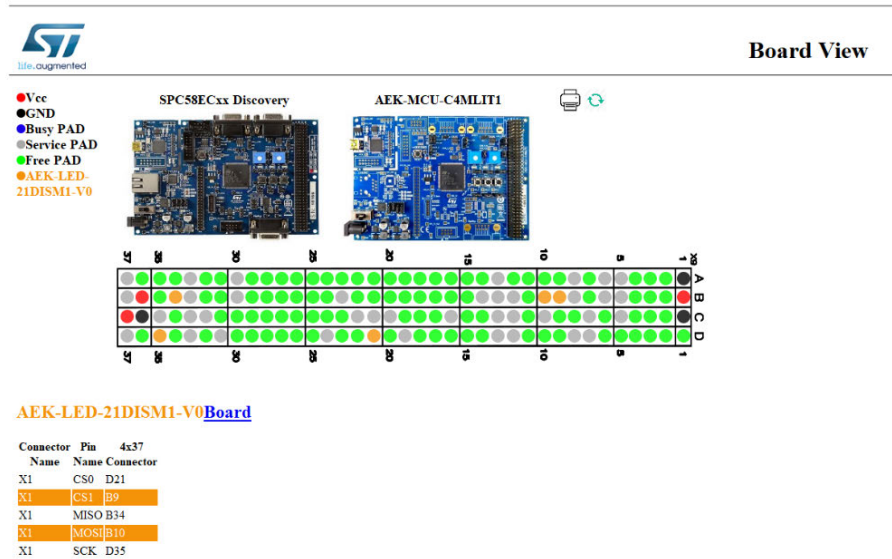
The Board View tab introduced with the AutoDevKit plugin for [SPC5-STUDIO](#) shows how MCU pins are mapped on the available board connectors. The view also provides dedicated tables to represent how the MCU allocated pins are connected to functional boards added from the AutoDevKit component library. Each table shows the mapping between functional board connectors and pins and corresponding MCU board connectors and pins. This information can render the wiring phase less time-consuming and error-prone.

Figure 64. Board View button



Note: You must add the AutoDevKit Init Package Component to your project to render the Board View button visible and to see other AutoDevKit components (functional boards).

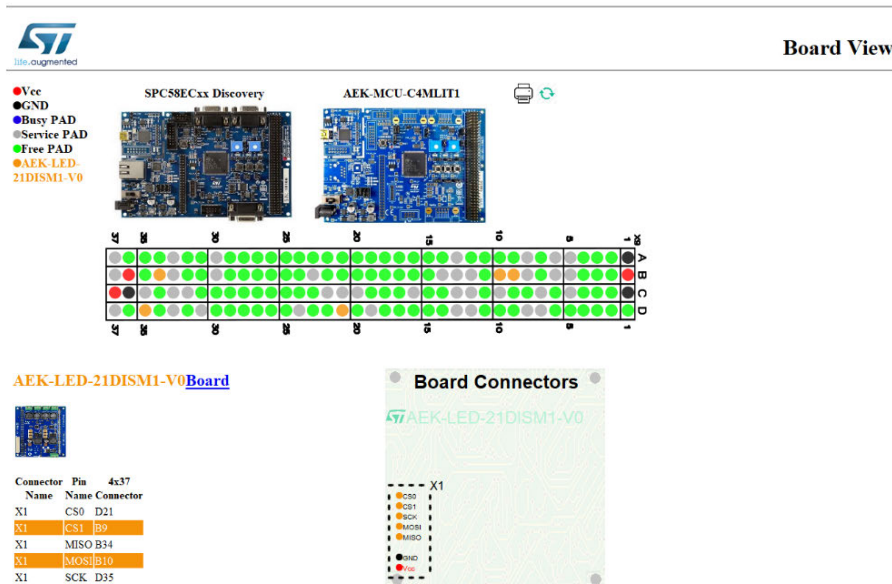
Figure 65. Board View editor window



In the example below, the colored dots in the 4x37 connector image help identify the pins required to wire the functional board to the MCU board. Some pins are required for the AEK-LED-21DISM1 board instance V0, while others are unused or reserved.

Note: *If pins are allocated directly in the PinMap Editor without using the component automatic allocation feature, these pins appear as Busy in the Board View.*
Clicking the 'Board' link next to each table reveals the functional board connector and pin locations.

Figure 66. Board View editor window with board connectors



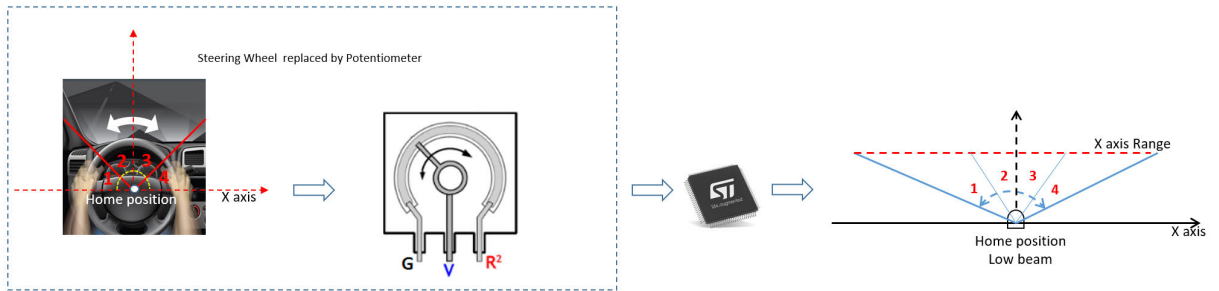
The top rows are frozen, so the MCU board 4x37 connector always remains visible. Use the refresh logo to update the view after project modifications.

Note: To improve the view, you may need to resize the editor window or generate a printable version by clicking the printer icon.

Appendix A Steering wheel turn simulation with a potentiometer

A potentiometer (POT) can be used to simulate steering wheel rotation in an adaptive front lighting platform where the headlight low beam direction is adjusted accordingly.

Figure 67. Representing a steering wheel with a potentiometer in an AFL system



The Potentiometer (POT) acts as a position transducer and its output voltage is used to control the low beam position via the MCU. The POT output voltage signal is fed to the MCU through one of its ADC converter channels, and then manipulated by the MCU firmware to rotate the low-beam headlight accordingly.

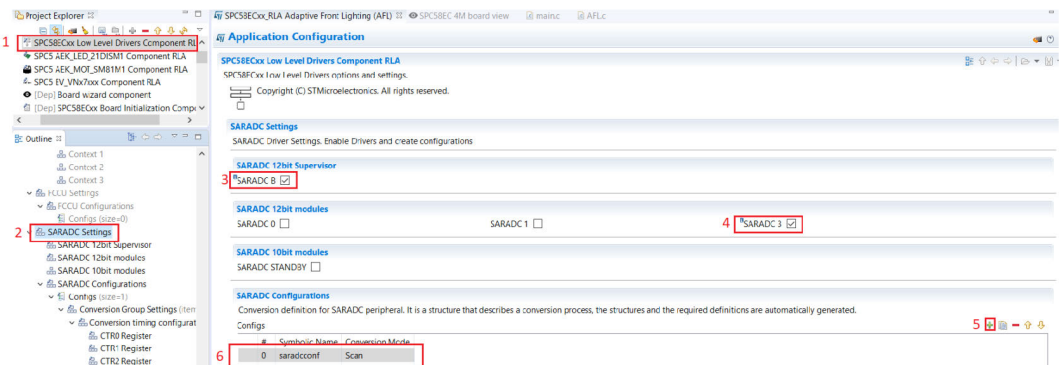
If we define four POT output voltage ranges to represent four low-beam positions, then the motor will have a span of five steps in total. The number of voltage ranges and consequently the number of steps can be increased to raise the sensitivity of the system.

A DC motor can be used to form a continuous system instead of a discrete stepper motor system, as long as appropriate hysteresis is introduced to reduce headlight wobble.

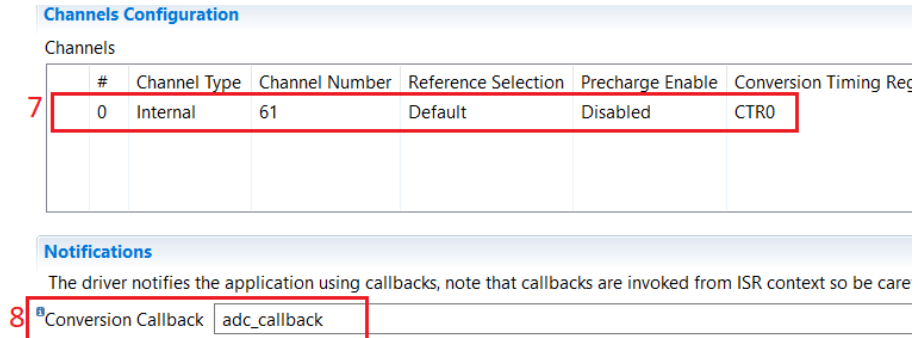
A.1 How to edit the AFL-Demo code to accept potentiometer inputs: Manual Mode

- Step 1.** Load the [AFL-Demo](#) source code and open the PinMap editor.
- Step 2.** Connect the potentiometer output voltage to an MCU ADC channel pin and set it in the PinMap editor. In the AFL-Demo, pin 51 is the designated ADC channel.
- Step 3.** Configure the ADC sampling frequency.

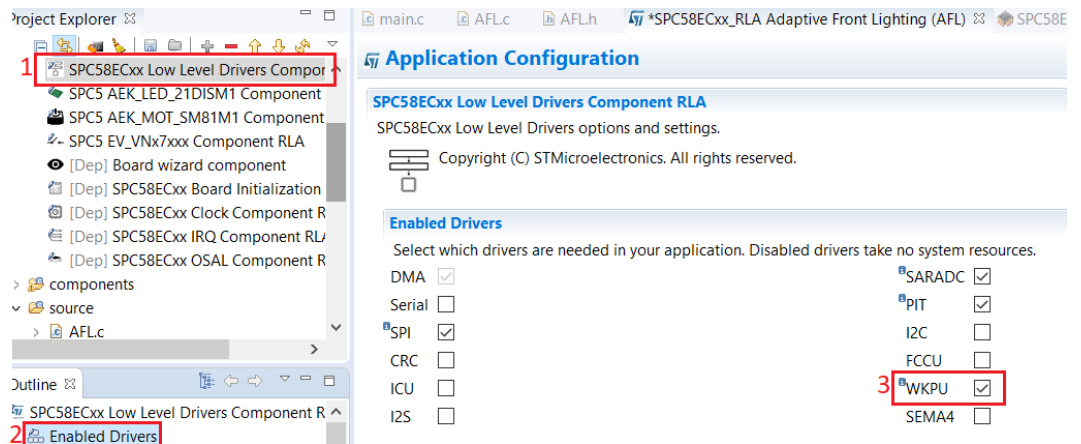
Figure 68. ADC channel configuration



- Step 4.** Double-click the “saradconf” config channel and insert the Callback name to be invoked whenever an ADC value is ready to be acquired.

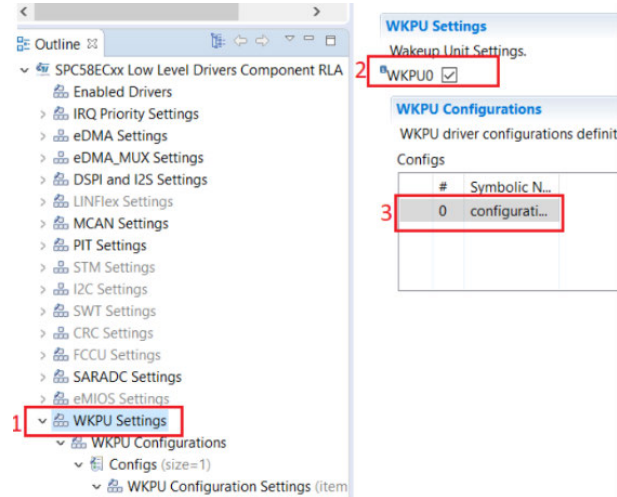
Figure 69. ADC callback nomination


- Step 5.** Use the Board view to connect the potentiometer to pin 5 of the ST. WHEEL connector on the AEK-CON-AFLVIP2 connector board.
- Step 6.** In the PinMap, select and associate a pin to the SW1 button on the MCU board. SW1 will be set up to switch between Manual and Automatic Modes.
- Step 7.** Enable the WKPU driver.

Figure 70. Wakeup Unit (WKPU) driver enable


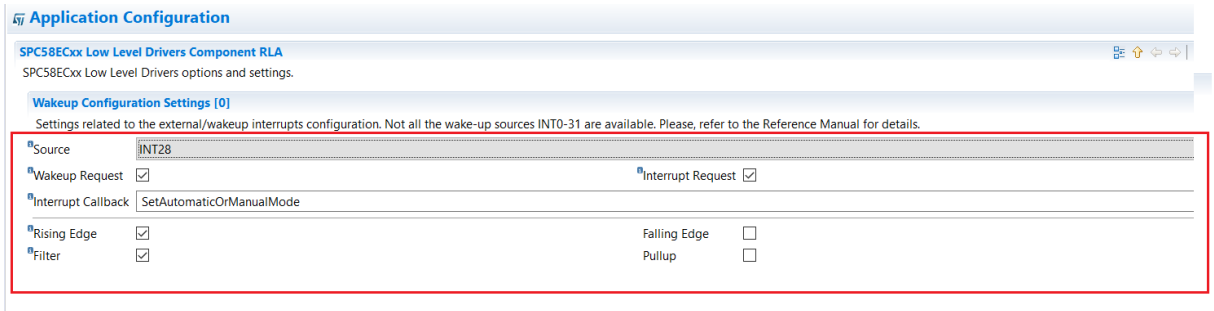
Step 8. Associate and configure the Wakeup signal to the SW1 button.

Figure 71. Wakeup Unit association with SW1



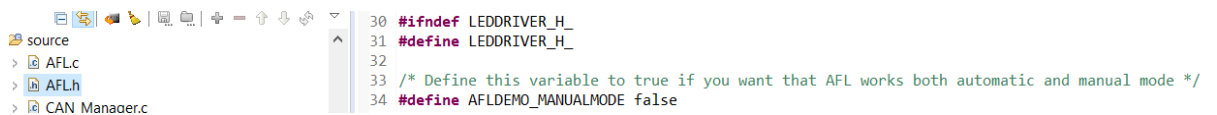
Step 9. Insert 'SetAutomaticOrManualMode' in the Interrupt Callback field.
This function changes the state of a global variable used to switch between Automatic and Manual Mode code execution.

Figure 72. Callback nomination for SW1 interrupt



Step 10. Set the variable "AFLDEMO_MANUALMODE" from false to true in the file AFL.h file.
The code for Manual Mode is already nested in the AFL-Demo project.

Figure 73. Define to enable Manual Mode



Appendix B CAN protocol in automotive applications

The car area network (CAN) protocol is widely used in Automotive Body and Convenience applications where the main body control module (BCM) connects with several electrical control units (ECUs).

The CAN protocol is implemented in the [AFL-Demo](#) software to communicate with an instrument panel control (IPC) so that actions performed by the headlight, such as blinking indicator light, can be reflected in the dashboard.

In-vehicle CAN messages belong to the following categories:

1. Messages triggered by specific events.
2. Message sent cyclically for safety reasons.

The ECU normally manages cyclic messages with a scheduler to ensure appropriate timing of transmission. The [AFL-Demo](#) source code only implements basic communication with the IPC; for example, only one type of message is sent to actuate the blinking effect on a car dashboard, so we have associated a programmable interrupt timer (PIT) to trigger the CAN message send event instead of a scheduler, which would normally be the case.

Revision history

Table 2. Document revision history

Date	Version	Changes
09-Oct-2019	1	Initial release.

Contents

1	Overview of Adaptive front lighting demonstration kits	2
1.1	AFL demonstration kit architecture	2
1.2	AFL software development and solutions	3
1.3	Other hardware	3
2	The AutoDevKit AFL development platform	4
2.1	How to connect the AFL driver kit with the headlight kit	4
3	AFL source code overview: Automatic mode	5
3.1	AFL-Demo setup	5
3.1.1	Calibration() function	6
3.1.2	ActivateDRL_LowBeam() function	6
3.2	AFL-Demo main	7
3.2.1	TurnLeft()	8
3.2.2	pit_ild_channel_start()	8
3.2.3	LeftBlinker()	8
3.2.4	osalThreadDelayMilliseconds()	10
3.2.5	Other functions	10
4	Hardware	12
4.1	AEK-MCU-C4MLIT1 system control board	12
4.2	AEK-LED-21DISM1 LED driver board	13
4.3	AEK-MOT-SM81M1 stepper motor driver board	15
4.4	EV-VN7050AS fan control board	16
4.5	AEK-CON-AFLVIP2 connector board	17
4.6	AEK-CON-5SLOTS1 connector board	18
5	AFL-Demo software package	21
6	How to program the SPC58EC80 microcontroller	22
7	SPC5-STUDIO overview	23
7.1	How to create a new project in SPC5-STUDIO	24
7.2	How to add components to an SPC5-STUDIO project	27
7.3	PinMap Editor	29

7.3.1	How to use the Editor view	30
7.3.2	How to use the Outline view	31
7.3.3	The Properties view	32
7.4	Component configuration in SPC5-STUDIO	33
7.5	How to set low level driver IRQ priority settings	35
7.6	How to generate and compile application source code	36
7.7	How to debug an application generated with SPC5-STUDIO	37
8	AutoDevKit	40
8.1	AutoDevKit installation	40
8.2	Adding AutoDevKit components to an SPC5-STUDIO project	43
8.3	How to configure an AutoDevKit component	43
8.4	Board View tab	45
Appendix A	Steering wheel turn simulation with a potentiometer	48
A.1	How to edit the AFL-Demo code to accept potentiometer inputs: Manual Mode	48
Appendix B	CAN protocol in automotive applications	51
Revision history		52

List of figures

Figure 1.	AutoDevKit adaptive front lighting development kits	1
Figure 2.	Adaptive Front Lighting development system	3
Figure 3.	AFL driver kit and headlight connection	4
Figure 4.	X-axis movement during calibration	6
Figure 5.	AFL-Demo main flowchart	7
Figure 6.	Component configuration for left turn indicator blinking	8
Figure 7.	Component configuration for dashboard indicator blinking	10
Figure 8.	AFL block diagram with ST application devices	12
Figure 9.	AEK-MCU-C4MLIT1 control board functional blocks	13
Figure 10.	AEK-LED-21DISM1 LED driver board	14
Figure 11.	AEK-LED-21DISM1 four LED load driving block diagram	15
Figure 12.	AEK-MOT-SM81M1 motor driver board	16
Figure 13.	EV-VN7050AS high-side driver board components	17
Figure 14.	AEK-CON-AFLVIP2 connector board	17
Figure 15.	AEK-CON-AFLVIP2 block diagram	18
Figure 16.	AEK-CON-5SLOTS1 connector board	19
Figure 17.	AEK-CON-5SLOTS1 connector groups	19
Figure 18.	AEK-CON-5SLOTS1 pin reassignment for external function boards	20
Figure 19.	Multiple AEK-CON-5SLOTS1 extension boards mounted on SPC5x Discovery board	20
Figure 20.	Flashing the AFL firmware using UDE STK	22
Figure 21.	SPC5-STUDIO AFL project workspace	23
Figure 22.	SPC5-STUDIO AFL project components	23
Figure 23.	SPC5-STUDIO AFL RLA drivers	24
Figure 24.	New Project menu	25
Figure 25.	Starter actions tab	25
Figure 26.	New application details page	25
Figure 27.	New application MCU platform	26
Figure 28.	New SPC5 application files in outline view	26
Figure 29.	main.c application file	27
Figure 30.	Open available components list	27
Figure 31.	Select available components for platform	28
Figure 32.	New components visible in project tree	28
Figure 33.	PinMap editor icon in Editors tab	29
Figure 34.	SPC5 MCU pin functions	30
Figure 35.	PinMap editor interface windows	30
Figure 36.	Pin signal function selection	31
Figure 37.	Selected peripheral in Outline view and eligible pins in Editor view	31
Figure 38.	Signals for DSPI peripheral	32
Figure 39.	Pins for selected peripheral signal	32
Figure 40.	Pin configuration in Editor view	32
Figure 41.	PinMap Properties view	33
Figure 42.	Component configuration sequence	33
Figure 43.	Simple, group and list property fields	34
Figure 44.	Editing list items	34
Figure 45.	Validation of property values	35
Figure 46.	Enabling low level drivers	36
Figure 47.	IRQ priority settings	36
Figure 48.	SPC5-STUDIO save, generate and compile icons	37
Figure 49.	Compiled project files	37
Figure 50.	Boot Mode radio button in Application Configuration view	38
Figure 51.	mini-USB port on AEK-MCU-C4MLIT1 board	38
Figure 52.	Open debug.wsx in the UDE STK debugging tool	39

Figure 53.	UDE STK program loader	39
Figure 54.	UDE STK debug command buttons	39
Figure 55.	SPC5-STUDIO install new software	40
Figure 56.	Select software zip location	41
Figure 57.	Select AutoDevKit for installation	42
Figure 58.	AutoDevKit installation details	42
Figure 59.	AutoDevKit license terms	43
Figure 60.	Add LED driver component to project	44
Figure 61.	Add new DSPI object	44
Figure 62.	LED driver board pin configuration	44
Figure 63.	Pin allocation button	45
Figure 64.	Board View button	45
Figure 65.	Board View editor window	46
Figure 66.	Board View editor window with board connectors	46
Figure 67.	Representing a steering wheel with a potentiometer in an AFL system	48
Figure 68.	ADC channel configuration	48
Figure 69.	ADC callback nomination	49
Figure 70.	Wakeup Unit (WKPU) driver enable	49
Figure 71.	Wakeup Unit association with SW1	50
Figure 72.	Callback nomination for SW1 interrupt	50
Figure 73.	Define to enable Manual Mode	50

List of tables

Table 1.	ST development and function boards for AFL applications	2
Table 2.	Document revision history	52

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved