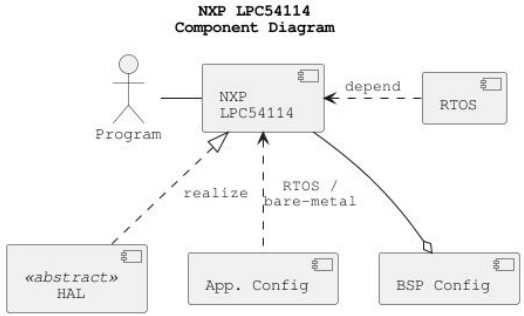
NXP LPC54114

The BSP development is made with a NXP LPCXpresso54114 development board. The BSP features a CMake and GCC build system and built based on **Hardware Abstraction Layer (HAL)**. It requires an application configuration file, which allows the user to specify the CPU clock frequency, enable or disable RTOS, and further define project-level I/O and settings.

This microcontroller offers a range of key features, including support for RTCC (Real Time Clock and Calendar) and numerous communication channels for I2C, UART, and SPI (8 channels each).

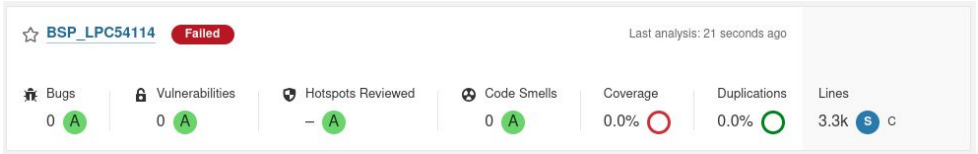
Some peripherals provide a resource lock interface for thread-safe operation, including I2C, PWM, SPI, and UART.

Version: 0.6.0



Available Peripherals

- ❏ ADC x12
- ❏ GPIO x48
- ❏ I2C x8
- ❏ PWM x8
- ❏ RTCC x1
- ❏ SPI x8
- ❏ TIMER x5
- ❏ UART x8
- ❏ WATCHDOG



th muda engineering

Hardware Abstraction Layer (HAL)

GPIO APIs

```
gpio_result_t hal_gpio_init( void )
gpio_result_t hal_gpio_set_mode( gpio_pin_t, gpio_mode_t )
gpio_result_t hal_gpio_set_interrupt( gpio_pin_t, gpio_int_mode_t,
                                      hal_gpio_callback )
gpio_state_t hal_gpio_read( gpio_pin_t )
gpio_result_t hal_gpio_write( gpio_pin_t, gpio_state_t )
```

Delay APIs

```
void hal_delay_sec( uint32_t )
void hal_delay_ms( uint32_t )
void hal_delay_us( uint32_t )
```

ADC APIs

```
adc_result_t hal_adc_init( adc_channel_attr_t )
adc_result_t hal_adc_calibrate( float )
uint32_t hal_adc_read( gpio_pin_t )
float hal_adc_read_volt( gpio_pin_t )
```

DAC APIs

```
dac_result_t hal_dac_init( dac_channel_attr_t )
dac_result_t hal_dac_calibrate( float )
dac_result_t hal_dac_write( uint16_t, uint32_t )
dac_result_t hal_dac_write_volt( gpio_pin_t, float )
dac_result_t hal_dac_mutex_take( gpio_pin_t, uint32_t )
dac_result_t hal_dac_mutex_give( gpio_pin_t )
```

PWM APIs

```
pwm_result_t hal_pwm_init( pwm_channel_attr_t )
pwm_result_t hal_pwm_start( gpio_pin_t )
pwm_result_t hal_pwm_update( gpio_pin_t, uint8_t )
pwm_result_t hal_pwm_stop( gpio_pin_t )
pwm_result_t hal_pwm_mutex_take( gpio_pin_t, uint32_t )
pwm_result_t hal_pwm_mutex_give( gpio_pin_t )
```

System APIs

```
system_result_t hal_system_clock_init( system_clock_attr_t )
system_result_t hal_system_reset( void )
system_result_t hal_system_set_power_mode( system_power_mode_t )
uint32_t hal_system_get_tick( void )
system_result_t hal_system_get_serial_number( uint32_t*, size_t )
system_result_t hal_system_get_temp_celsius( float* )
```

NVM APIs

```
nvm_result_t hal_nvm_init( void )
nvm_result_t hal_nvm_read( uint32_t, uint32_t*, size_t )
nvm_result_t hal_nvm_write( uint32_t, uint32_t*, size_t, nvm_mode_t )
nvm_result_t hal_nvm_erase( uint32_t, nvm_mode_t )
nvm_status_t hal_nvm_get_status( void )
nvm_result_t hal_nvm_mutex_take( uint8_t, uint32_t )
nvm_result_t hal_nvm_mutex_give( uint8_t )
```

Timer APIs

```
timer_result_t hal_timer_init( timer_handle_t*, timer_attr_t,
                               hal_timer_callback )
timer_result_t hal_timer_start( timer_handle_t* )
timer_result_t hal_timer_stop( timer_handle_t* )
```

Watchdog APIs

```
wdt_result_t hal_watchdog_init( wdt_attr_t )
wdt_result_t hal_watchdog_feed( void )
wdt_result_t hal_watchdog_enable( void )
wdt_result_t hal_watchdog_disable( void )
```

Quadrature Encoder Interface (QEI) APIs

```
qei_result_t hal_qei_init( qei_handle_t*, qei_channel_attr_t )
int32_t hal_qei_get_counter( qei_handle_t* )
qei_result_t hal_qei_set_counter( qei_handle_t*, int32_t )
qei_result_t hal_qei_reset_counter( qei_handle_t* )
qei_result_t hal_qei_start( qei_handle_t* )
qei_result_t hal_qei_stop( qei_handle_t* )
```

Real-time Clock (RTC) APIs

```
rtc_result_t hal_rtc_init( rtc_handle_t*, rtc_channel_attr_t )
rtc_result_t hal_rtc_start( rtc_handle_t* )
rtc_result_t hal_rtc_get_datetime( rtc_handle_t*, rtc_datetime_attr_t* )
rtc_result_t hal_rtc_set_alarm( rtc_handle_t*, rtc_datetime_attr_t,
                                hal_rtc_callback )
rtc_result_t hal_rtc_stop( rtc_handle_t* )
```

Hardware Abstraction Layer (HAL)

UART APIs

```
uart_result_t hal_uart_init( uart_handle_t*, uart_channel_attr_t )
uart_result_t hal_uart_write( uart_handle_t*, uint8_t*, size_t )
uart_result_t hal_uart_read( uart_handle_t*, uint8_t*, size_t )
uart_result_t hal_uart_read_until( uart_handle_t*, uint8_t, uint8_t*, size_t )
uart_result_t hal_uart_stop( uart_handle_t* )
uart_bus_status_t hal_uart_get_status( uart_handle_t* )
uart_result_t hal_uart_mutex_take( uint8_t, uint32_t )
uart_result_t hal_uart_mutex_give( uint8_t )
```

I2C APIs

```
i2c_result_t hal_i2c_init( i2c_handle_t*, i2c_channel_attr_t )
i2c_result_t hal_i2c_write( i2c_handle_t*, uint16_t, uint32_t, size_t, uint8_t*, size_t )
i2c_result_t hal_i2c_read( i2c_handle_t*, uint16_t, uint32_t, size_t, uint8_t*, size_t )
i2c_result_t hal_i2c_stop( i2c_handle_t* )
i2c_result_t hal_i2c_mutex_take( uint8_t, uint32_t )
i2c_result_t hal_i2c_mutex_give( uint8_t )
```

SPI APIs

```
spi_result_t hal_spi_init( spi_handle_t*, spi_channel_attr_t )
spi_result_t hal_spi_write( spi_handle_t*, gpio_pin_t, uint8_t*, size_t )
spi_result_t hal_spi_read( spi_handle_t*, gpio_pin_t, uint8_t*, size_t )
spi_result_t hal_spi_read_write( spi_handle_t*, gpio_pin_t, uint8_t*, uint8_t*, size_t )
spi_result_t hal_spi_stop( spi_handle_t* )
spi_result_t hal_spi_mutex_take( uint8_t, uint32_t )
spi_result_t hal_spi_mutex_give( uint8_t )
```

CAN APIs

```
can_result_t hal_can_init( can_handle_t*, can_channel_attr_t )
can_result_t hal_can_write( can_handle_t*, uint32_t, can_frame_t, uint8_t*, size_t )
can_result_t hal_can_read( can_handle_t*, uint32_t*, can_frame_t*, uint8_t*, size_t )
can_result_t hal_can_stop( can_handle_t* )
can_result_t hal_can_sleep( can_handle_t* )
can_result_t hal_can_mutex_take( uint8_t, uint32_t )
can_result_t hal_can_mutex_give( uint8_t )
```

Modbus APIs

```
modbus_result_t hal_modbus_init( modbus_handle_t*, modbus_channel_attr_t )
modbus_result_t hal_modbus_write( modbus_handle_t*, modbus_comm_attr_t*, uint8_t*, size_t )
modbus_result_t hal_modbus_read( modbus_handle_t*, modbus_comm_attr_t*, uint8_t*, size_t )
modbus_result_t hal_modbus_stop( modbus_handle_t* )
modbus_status_t hal_modbus_get_status( modbus_handle_t* )
modbus_result_t hal_modbus_mutex_take( uint8_t, uint32_t )
modbus_result_t hal_modbus_mutex_give( uint8_t )
```