ATAN0145

Using SAM DA1 in LIN Ultra-low-power Nodes

APPLICATION NOTE

Introduction

This application note provides an example of LIN master and LIN slave communication within LIN networks focused on ultra-low-power consumption using Atmel® SMART SAMDA1 automotive-qualified microcontrollers. SAMDA1 is an ultra-low-power embedded MCU device with very low power consumption in standby sleep mode. To achieve the lowest power consumption while in sleep mode, the device must be configured in standby sleep mode with its SERCOM UART peripheral set to asynchronous wake-up. Because the internal voltage regulator is only able to drive low-current loads while in standby, it is necessary to configure the core and its peripherals so that the device only consumes a very low amount of power and the voltage regulator is not being overdriven. This application note describes the use of peripherals and the operating modes of the LIN slave node in order to achieve ultra-low sleep current while in standby.

Table of Contents

1. Summary

- The Atmel® SAMDA1 must be put into standby sleep mode to achieve the lowest power consumption while in sleep.
- In standby sleep mode the internal voltage regulator is switched to the low-power state. In this state the regulator can only supply limited current to the core and peripherals. So that wake-up is successful, the load current on the internal VREG should not exceed 50µA before entering standby sleep mode.
- Although multiple peripherals can run in standby mode, it is critical to enable only the ones required for generating a wake-up signal. Careful power budgeting is recommended.
- In a LIN application, a wake-up trigger for a LIN slave node can, for example, be derived from the UART asynchronous wake-up trigger by enabling the SFDE (start of frame detection enable) bit or using an asynchronous wake-up via EIC (external interrupt controller) and enabling interrupt on pin change. In both cases, the falling edge on the LIN pin (i.e., entering the break field) triggers a wake-up.
- In order to successfully enter standby mode, all unused peripherals and their clocks must be disabled before entering standby sleep mode (e.g., disable clocks in the Power Manager APBxMASK registers (e.g., APBCMASK.AC1, I2S, PTC, DAC, ADC, SERCOM5-0, etc.).
- When using the PTC module, disable PTC debug pulses to minimize power consumption (i.e., DEF_TOUCH_QDEBUG_ENABLE=0 in touch.h).
- Minimize standby power by clearing the RUNSTDBY bit on used oscillators. This disables the oscillator when the device enters standby sleep mode. Use a separate oscillator that can have the RUNSTDBY bit enabled for peripherals that have to run in standby mode.
- Set RUNSTDBY=1 and ONDEMAND=1 bits to keep the oscillator running only when there is a peripheral requesting its clock while in standby sleep mode.
- To minimize clock tree power consumption, select GCLK1, GCLK2, etc., as a generic clock driver with a GCLK0 exception. GCLK0 is the main clock module and its drivers route output clocks to all peripherals . This increased load increases power consumption when GCLK0 is used.
- When using EIC as a asynchronous wake-up source, the OSCULP32K oscillator can be used as a system clock. This is a very low-power internal RC oscillator which is permanently enabled (even in standby sleep mode) and can be used while in standby mode to further reduce power consumption.

2. SAMDA1-XPRO Resources which Support the LIN Bus

Atmel® SMART SAM DA1 resources used in a LIN application:

- ATSAMDA1J16A-ABT: ARM® Cortex®-M0+ MCU, TQFP64
  - 64KB PM Flash, 2KB RWW Flash, 8KB SRAM
  - Oscillators: OSC8M, OSC32K, OSCULP32K
  - Idle/standby sleep mode
  - SERCOM (UART) – supports LIN communication (LIN master, LIN slave)
  - External Interrupt Controller (EIC) when used as wake-up source from the LIN TRX
  - Peripheral Touch Controller (PTC) – touch interface support
  - Real-Timer Counter (RTC) – Break and interframe timer
  - WDT
  - Serial Wire Debug (SWD) interface for debugging and programming
- LIN transceiver: Atmel ATA663231-FAQW
  - SBC chip, including 3.3V VREG
  - LIN 2.1 transceiver
- Integrated on SAMDA1-XPRO USB embedded debugger (with SWD interface) for MCU programming and debugging. Connects to PC host via USB mini connector.

3. LIN Protocol Overview

The Local Interconnect Network (LIN) bus is based on a serial network protocol used for low-speed communication between a LIN master node and up to 16 LIN slave nodes. It is primarily used in the automotive industry for communication between individual ECUs which support a low baud rate (up to 20Kbaud), one- wire serial, HV (12V) signaling. The LIN protocol supports digital data communication such as with the vehicle sunroof, doors, engine, and seats. LIN relies on a master-slave protocol where the transmitting node provides master data frame timing and transmits the header data including the synchronization and ID header data. The slave transmission constitutes the frame response consisting of individual data fields and the integrity data check field as a checksum.

The main LIN features are:

- Master-slave protocol
- Based on standard UART hardware and data transmission formatting
- Physical layer is based on enhanced ISO 9141 (K-line)
- Baud rate up to 20Kbaud
- Addressing IDs for 16 nodes (e.g., 1 × master, 15 × slaves)
- Support of up to 8bytes of payload/frame

A typical LIN frame consists of a header and the response intervals. The header is transmitted by one LIN master node and the response is transmitted either by one of the LIN save nodes or the LIN master itself. The LIN master normally uses a scheduling table to configure the LIN frame where it addresses individual LIN slaves using device/node IDs. Data is transmitted as 8-bit data bytes with one start, one stop and no parity when sending UART 8-bit formatted data.

The figure below depicts a LIN frame and its individual fields.

Figure 3-1. LIN Message Frame Fields

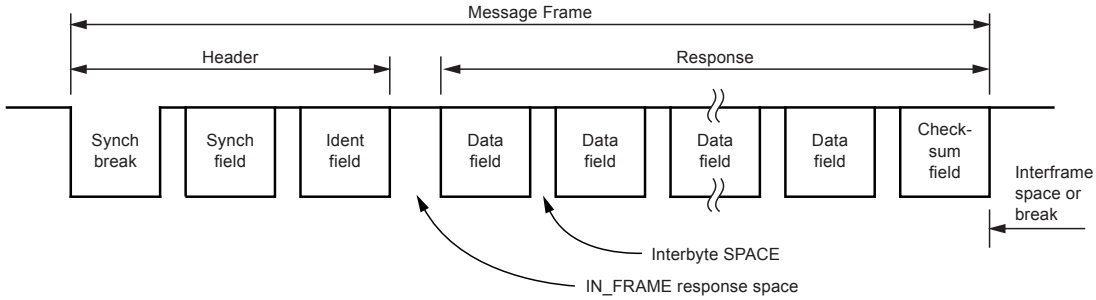
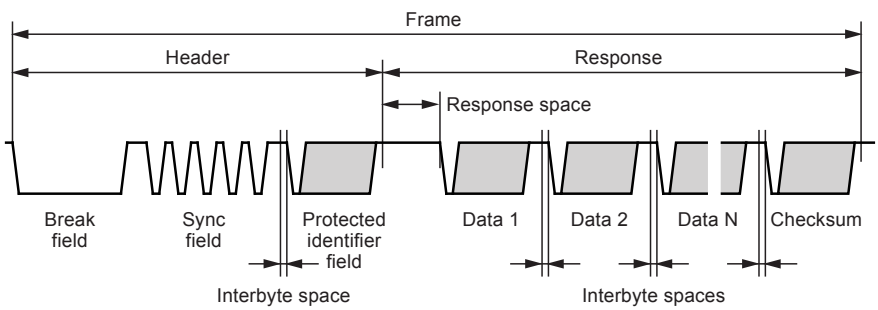


Figure 3-2. LIN Frame Signaling



The header includes::

- Break: This indicates the beginning of the LIN frame and has a threshold of at least 11 × Tbit and can be as long as 13 × Tbit. Its level is set to dominant (logic 0, i.e., GND).

- Sync: This is used for data synchronization of the slaves with the master. It follows directly after the BREAK field and is defined as a typical data byte with its start bit, sync data byte (0X55), and the end bit as well as two parity bits.
- Interbyte space: This is a recessive signal providing a gap for interfield jitter adjustment.
- Identifier: This is where the PID (protected identifier) is sent to the slaves. Only the node with a corresponding PID can respond. The PID consists of two subfields: (1) the frame identifier - ID[5:0]], with the values from 0 to 63, and (2) the parity - P[1:0], calculated on frame ID bits.

The response field includes:

- Data: Any LIN frame carries from 1 to 8 bytes of payload data. All the payload data is transmitted in this data field. LSB is transmitted first (little-endian).
- Checksum: This is an inverted 8-bit sum with carry over all the data bytes and the PID field.

LIN frame types include unconditional, event triggered, sporadic and diagnostic frames.

- Unconditional frames always carry signals and their identifiers from 0 to 59. All subscribers receive and provide the frame to the application.
- Event-triggered frames are frames generated based on the header information. They provide a response only when data values have changed; if there is no change, the rest of the frame is silent.
- Sporadic frames are updated header info frames sent by the master. The publisher always provides the response to the header.
- Diagnostic frames are frames which carry diagnostic information and always provide 8 data bytes; the PID is set to 60 or 61.
- User-defined frames are specific to the user requirements. Their PID = 62 and the data payload is user-defined.
- Reserved frames use PID = 63. They cannot be used in a LIN 2.0 cluster.

1. **Identity Field Formatting**
   The protected identifier field consists of the frame identifier (ID00 to ID05) and the parity for the identifier(P06 and P07).

   Frame Identifier

   Six bits are reserved for the frame identifier in the range from 0 to 63:

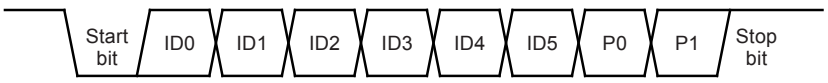
   - 0–59 (0x3B): frame carrying the signal
   - 60 (0x3C) and 61 (0x3D): frames carrying diagnostic and configuration data
   - 62 (0x3E) and 63 (0x3F): frames reserved for future protocol enhancements

   Parity

   The P0 bit is calculated by computing the “OR” function of all the identifier bits (ID0 to ID4). The P1 bit is calculated by computing the “OR” function of the identifier bits (ID1 to ID5) and inserting the result.

   Protected identifier (PID) data is transmitted by sending ID bits in a manner similar to that used for the data field format (ID0-5, LSB first) followed by the parity bits P0-1.

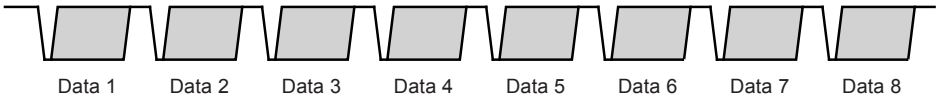
   Figure 3-3. Identifier Bit Order



2. **Data Field Formatting**

Each frame can consist of 8 bytes of payload data and is transmitted in the byte field. The number of bytes is agreed upon by the frame publisher and the frame subscriber at the network deployment stage..

Figure 3-4. Data Field Bit Order

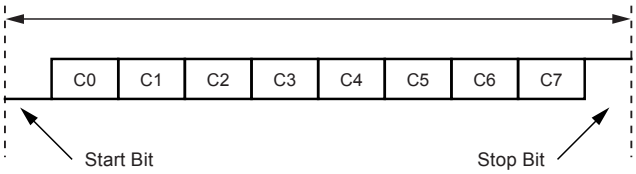


3. **Checksum Field Formatting**

The checksum field consists of a checksum byte which is a modulo-256 sum of all data bytes within the data field (also including the PID data). To compute the checksum, all the data bytes are added in with carry and the resulting sum is inverted.

The checksum is transmitted in the data field and uses data field formatting

Figure 3-5. Checksum Field Bit Order



4. LIN Demo Description

This LIN demo application consists of the LIN master and LIN slave node pair made up of SAMDA1-XPRO and QT1-XPRO PCBs for each node. Both nodes are powered using 12VDC external power supply and connected to the LIN bus via a single wire. The QT1 PCB attached to the LIN slave node is used as the touch sensor interface where a detection of the touch, the slider and the wheel sensor is displayed on QT1 LEDs. Once a touch is detected by the LIN slave node, it is published as a LIN message to the LIN master node, where it is displayed on the LIN master QT-XPRO LEDs. This allows the QT1-XPRO on the LIN slave to be used for detecting the touch and displaying it on its LEDs, with the QT1-XPRO on the LIN master used only for displaying published LIN messages (no touch interface is enabled on the LIN master) consisting of touch activity of the LIN slave node only.

The LIN demo hardware setup:

- 2 × SAMDA1-XPRO evaluation PCBs – LIN master and LIN slave node
- 2 × QT1 Xplained Pro SelfCap touch evaluation PCBs – Qtouch interface boards. Each PCB plugs directly into EXT1 and EXT2 on each of the SAMDA1-XPRO PCBs and consists of 2 × touch buttons, 1 × slider and 1 × wheel (optional)
- 12VDC power supply (VS_LIN) and a single wire connection between the SAMDA1-XPRO boards on the LIN bus pin (LIN)

Table 4-1. LIN Demo SAMDA1-XPRO/QT1 Touch (Master) to SAMDA-XPRO/QT1 Touch (Slave) HW Connections

Description - Signal/Port	SAMDA1-XPRO (Master) Signal/Port	SAMDA1-XPRO (Slave)
Vsupply-LIN (12V)(1)	VS-LIN/HV port	VS-LIN/HV port
Ground	GND/HV port	GND/HV port
LIN signal	LIN/HV port	LIN/HV port
QT1 touch extension PCB - EXT1	SAMDA1-XPRO/EXT1	SAMDA1-XPRO/EXT1
QT1 touch extension PCB - EXT2	SAMDA1-XPRO/EXT2	SAMDA1-XPRO/EXT2
EDBG USB debugger port(2)	SAMDA1-XPRO/USB_EDBG	SAMDA1-XPRO/USB_EDBG
USB target port(3)	SAMDA1-XPRO/USB_TARGET	SAMDA1-XPRO/USB_TARGET

Note:

1. VS_LIN pins on the slave and the master node are connected to a 12VDC power supply.
2. EDBG USB debugger ports are used to download and debug source code to SAMDA1 targets.
3. USB target ports on SAMDA1 can be used to communicate with PC host hyper terminal, for example.

5. Application Software

This LIN demonstration software example is provided for both the LIN master and LIN slave nodes which are both based on the Atmel® SAMDA1J16 MCU. The supporting hardware consists of the SAMDA1 Xplained Pro evaluation board and the optional QT1-XPRO Qtouch board supporting the LIN communication protocol in the following scenarios:

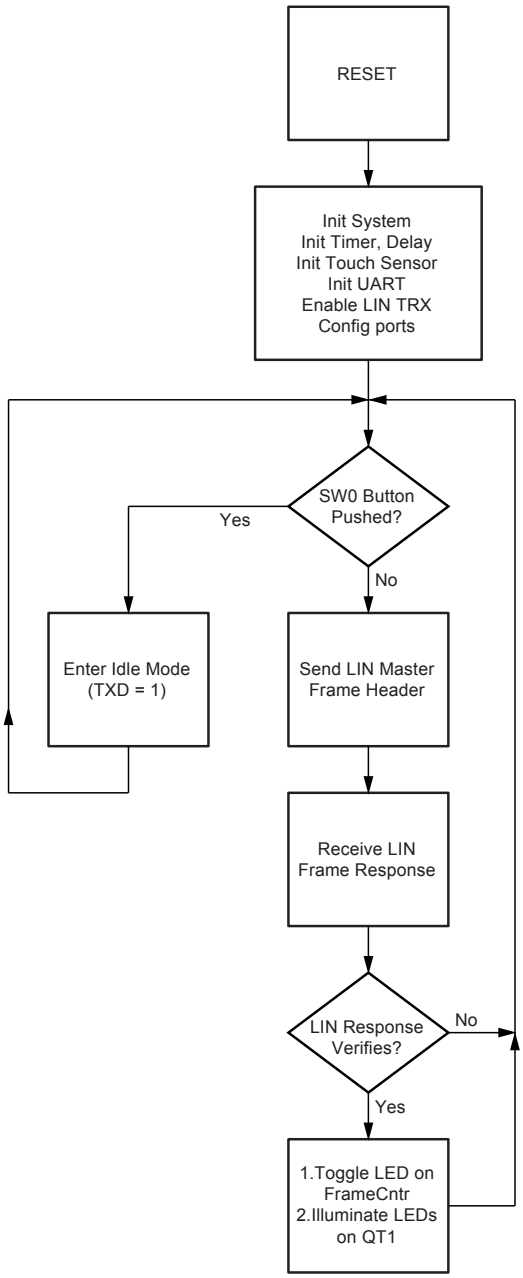
- The SAMDA1-XPRO is used in conjunction with the QT1-XPRO to demonstrate LIN touch functionality while the touch data collected by the sensors on the QT1_XPRO slave node is transmitted to the LIN master.
- The SAMDA1-XPRO is used in stand-alone configurations with LIN functionality and ultra-low-power consumption.

LIN master application software overview:

- The LIN master uses master and slave software tasks to transmit and receive LIN data from the LIN slave node.
- The LIN master node generates LIN subscription frames and sends the LIN frame header (Break, Sync, PID) and receives the response published by the LIN slave consisting of the touch payload data.
- The system clock is based on an OSC8M internal on-chip oscillator.
- The LIN master polls switch SW0 input, and if a push is detected, it transmits an idle LIN signal (recessive level only) on the LIN pin. This ensures no data is transmitted on the LIN pin and enables power measurement while in sleep mode on the LIN slave node.
- TC module generates a timer tick to transmit LIN frame header data at 10msec intervals (i.e., “time_to_schedule_lin” flag is set in main()).
- AtaHw_LinMasterStartFrame() is executed to transmit LIN starting frame break.
- AtaHw_LinMasterPollingTask() sends subsequent LIN header content.
- AtaHw_LinSlavePollingTask() receives published LIN response data. Then main.c parses button1_state, button2_state, slider_position, and rotor_position active states. With QT1- XPRO connected on the EXT1 and EXT2 header, the touch status is displayed for the buttons. The position for the slider and the wheel is indicated on the QT1-XPRO LEDs.
- Once the l_flg_tst_statusupdated() flag is set, user LED0 toggles to indicate reception of data from the slave node.

Note: There are no touch detection functions enabled on the LIN master node.

Figure 5-1. LIN Master Software Flow



LIN Slave Node Overview

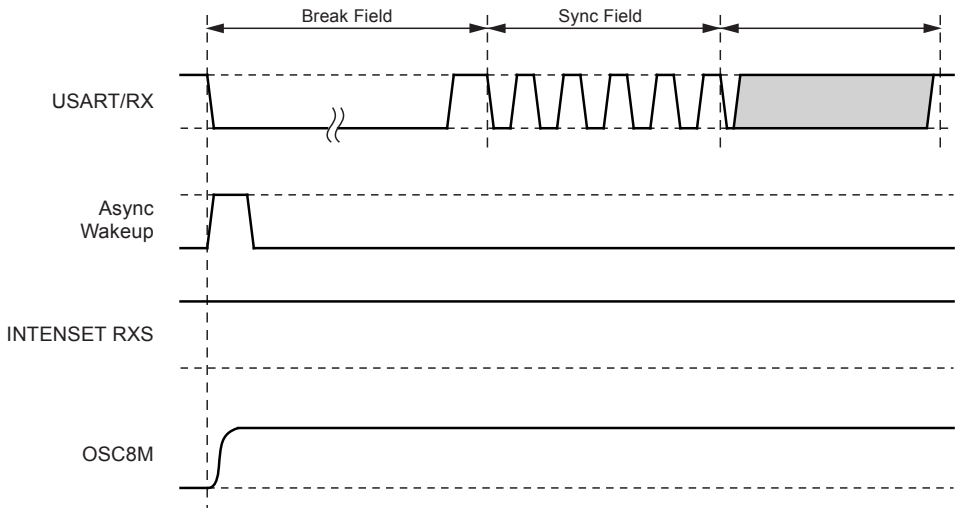
- The LIN slave publishes payload data in the response field (this becomes the publish message for the slave).
- The system clock is integrated within an OSC8M internal on-chip oscillator.
- RTC timing out triggers the PTC module to sense for the touch and update its touch state data on “rtc_overflow_callback” by setting the touch_time.time_to_measure_touch flag variable.
- Touch status is updated once the “measurement_done_touch” flag is set. Touch sensor states are updated. Upon touch detection, the LEDs on the QT-XPRO PCB are illuminated.
- The SAMDA1-XPRO user LED blinks to indicate valid LIN frame data communication.
- The LIN slave enters standby sleep mode each time a wait-for-interrupt instruction is executed (asm("wfi")). This instruction is included in the while (1) loop in main() and it is executed each time

going through the loop. Each time a task is completed, it returns back to while (1) loop for application execution.

- Interrupts are used as sources of wake-up. For data reception – asynchronous Start-of-Frame Detection Trigger wake-up is used as data is detected on the RX pin by the UART peripheral.
- As an alternative, the interrupt-on-pin-change trigger using EIC can also be used as a wake-up source as the frame data is detected on the RX pin. Upon device wake-up (via EIC), Atmel SAMDA1 must be reconfigured to enable SERCOM/USART on the same RX pin as data input pin used for wake-up by the EIC. While in standby, the EIC method consumes standby current at a level to using the UART asynchronous wake-up.
- The LIN slave uses AtaHw_LinSlavePollingTask() to receive and transmit data using interrupts.
- When set to “0,” the <PTC_enable> variable excludes the PTC module to achieve the lowest possible power consumption.
- The LIN driver can use interrupts or is polled by defining LIN_SLAVE_INTERRUPTS in the conf_lin.h file.

The LIN slave node is put into standby mode with all peripherals and their clocks disabled in PM for optimal power consumption. The LIN slave can use asynchronous wake-up by setting the Start of Frame Detection Enable bit (SFDE) in the CTRLB register. Figure 5-2 shows an asynchronous wake-up generation as the break field is detected. With the INTENSET.RXS bit configured, asynchronous wake-up occurs on the falling edge of the RX signal. SAMDA1 reaches its full functionality after the OSC8M wake-up period, when it reaches its operating voltage and frequency. For more information, see Figure 5-2.

Figure 5-2. Atmel SAMDA1 LIN Asynchronous Wake-up



QT1-Xpro: The touch extension board (QT1) can be optionally connected to the EXT1 and EXT2 headers on the SAMDA1-XPRO and is a touch interface using Peripheral Touch Controller (PTC) functionality. One slider, one wheel and two touch buttons are implemented on the QT1 and supported by the Atmel SAMDA1 touch interface. The Atmel SAMDA1 Qtouch library is populated with its API functions as shown in Table 5-1. Set PTC_enable=0 to disable PTC functionality for the best low-power consumption.

Note:

1. Disabling the PTC debug clocks by setting DEF_TOUCH_QDEBUG_ENABLE to 0 improves power consumption performance while the PTC peripheral is enabled. See the touch.h file in the QTouch driver for this parameter.
2. The MCU current consumption can be measured while in sleep mode by setting the LIN signal to idle (recessive level) on the LIN master node and by pushing and holding down the SW0 button.

Figure 5-3. LIN Slave Application Overview

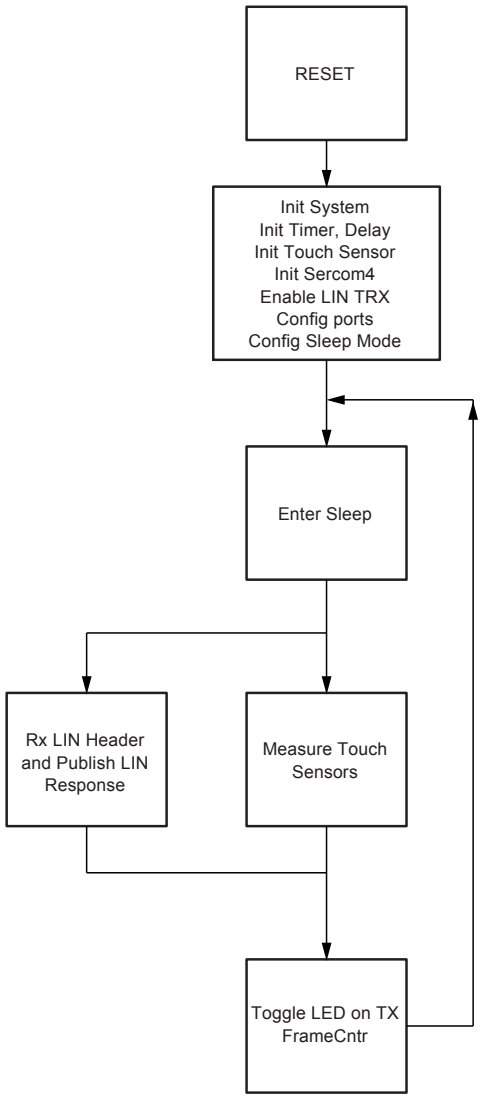


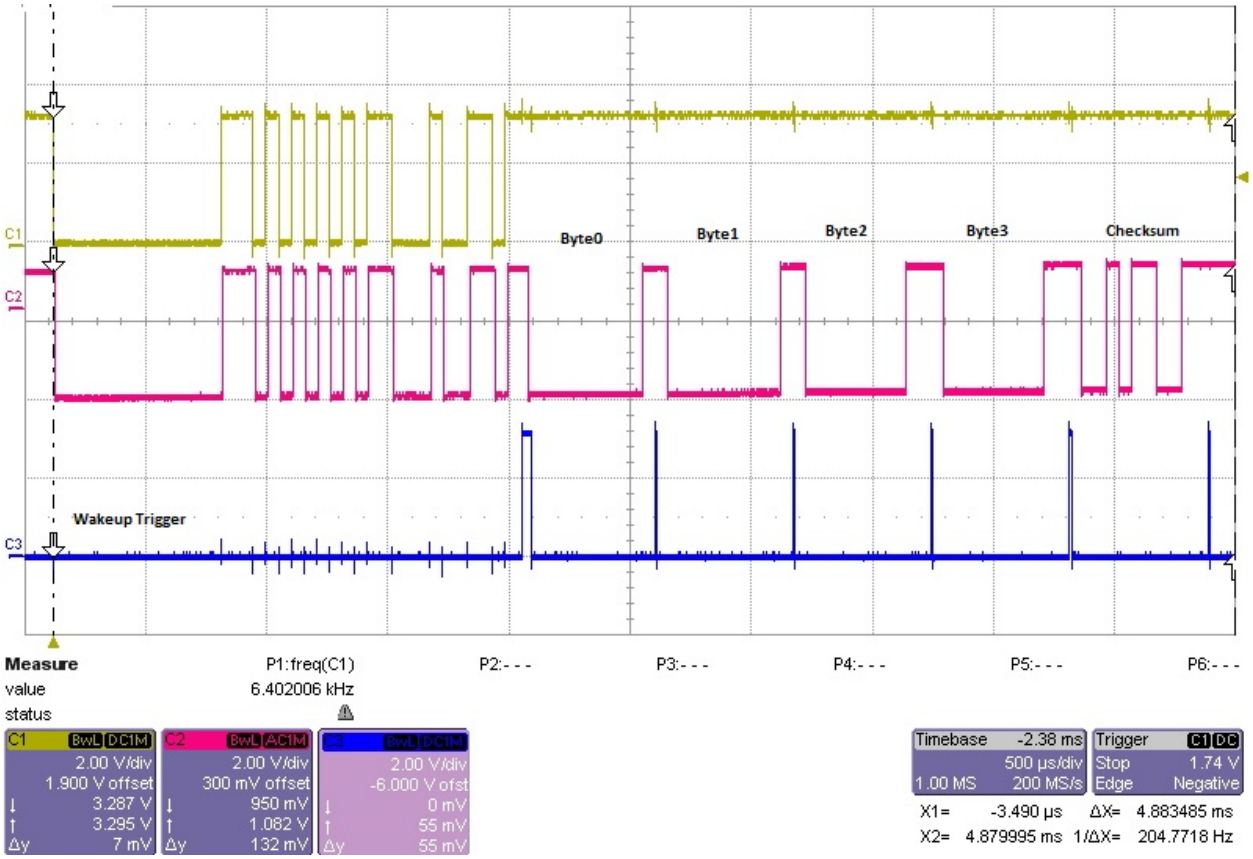
Table 5-1. PTC Functions Used in LIN Slave (PTC_enable=1)

Function	Description
touch_sensors_init()	Initializes the QTouch lib and configures the touch sensors
configure_port_pins()	Configures the QT1 PCB outputs
touch_sensors_measure()	Starts touch sensor measurement
GET_SELFCAP_SENSOR_STATE(n)	Returns touch sensor state for a sensor number “n” as a sensor number from 0 to 3: (1) n=1, button1 and button2 states respectively; (2) n=2, rotor state; (3) n=3, slider state.
GET_SELFCAP_ROTOR_SLIDER_POSITION(1)	Returns slider (n=1) position

The timing diagram below shows LIN frame transmission. The LIN master transmits the LIN header interval and the LIN slave transmits the response. The scope trace show input pins and output pins on the Atmel SAMDA1 device RXD and TXD connections to the LIN TRX:

- Channel_1(C1): UART_TX pin (data transmit)
- Channel_2(C2): UART_RX pin (data receive)
- Channel_3(C3): The GPIO signal is used as a byte strobe for byte reception in the response interval. It is generated by the LIN master node upon byte 0-3 and checksum reception.

Figure 5-4. LIN Subscription Frame Transmission



6. Standby Power Consumption

The Atmel®SAMDA1 includes idle and standby sleep modes. They are both activated by the Wait For Interrupt Instruction (WFI). Idle sleep mode is activated by setting idle bits in the sleep mode register (SLEEP.IDLE) in the power manager. Standby mode is set by setting SLEEPDEEP bit in the system control register of the CPU.

- IDLE mode: While in idle sleep mode, the CPU clock is stopped and, depending on which idle mode is selected, the IDLE0, IDLE1, or IDLE 2 synchronous clock domains are stopped. VREG is enabled and operated at its normal capacity.
- STANDBY mode: The CPU and all clocks are stopped. All clock sources are stopped depending on those selected by the RUNSTDBY bit in the respective oscillator register. The regulator operates in low-power mode which can only support load currents of up to 50µA. In this mode, only a very limited number of clocks and/or oscillators can be enabled so that the regulator is not overloaded.

Table 6-1. SAMDA1 Sleep Modes

Mode	Level	Mode Entry	Wake-up Sources
IDLE	0	SCR.SLEEPDEEP=0 SLEEP.IDLE=Level; WFI	Synchronous (APB, AHB) asynchronous
	1		Synchronous (APB) asynchronous
	2		Asynchronous
STANDBY		SCR.SLEEPDEEP=1	Asynchronous

LIN slave node can conserve power by entering a sleep mode. The device is powered down in standby sleep mode, leaving only the most critical functionality such as USART/EIC for generating a wake-up trigger once the WFI instruction is executed.

Examples for entering Idle_2 and standby sleep mode:

```
//Enable IDLE mode

// Sleepdeep bit in System Control Register in the CPU must be set to '0'
       SCB-> SCR.SLEEPDEEP = 0;
    PM->SLEEP.reg = 0x02;    //Idle[1:0]=0x2 where CPU, AHB and APB domains are stopped after
WFI instr.

while(1)
{
asm("wfi");
}
```

```
//Enable Standby mode

// Sleepdeep bit in System Control Register in the CPU must be set to '1'
    SCB->SCR.SLEEPDEEP = 1;    //Enters Standby mode after WFI instr.

while(1)
{
asm("wfi");
}
```

When using an asynchronous UART wake-up trigger as a wake-up source, the Start of Frame Detection Enable bit (SFDE) in the USART CTRLB register must be set during initialization. When the LIN frame is

detected on the USART/RX input pin, the device triggers a wake-up when the INTENSET.RXS bit is set to “1”.

Similarly, EIC can detect a frame when its interrupt signal input can be assigned to the USART RX pin for wake-up only. Once the device is powered, the RX inputs can be reassigned back to their respective USART peripheral for data detection.

1. Measuring standby power consumption
   - Connect an ammeter to the power measurement header on the SAMDA1-XPRO.
   - Press SW0 on the LIN master node SAMDA1-XPRO PCB and hold down to suspend data transmission. The LIN pin is put into its recessive (high) state. At this time the LIN slave node is put into standby mode. In this state the LIN slave is suspended in the standby state because it is waiting for the Start-of-Frame asynchronous wake-up signal to be generated by the USART module. The power consumption at this time is displayed by the ammeter connected to the power measurement header.

   Once SW0 is released, the LIN master node begins to send LIN frame data and the LIN demo resumes normal operation. Note that the LIN slave still enters standby mode between data transmissions and at any other times a wake-up signal occurs.
2. Measurement Results

Table 6-2 shows a sample of measurements taken using the normal, idle_2 sleep and standby sleep modes.

Table 6-2. Atmel SAMDA1 Power Consumption

Operating Mode	PTC_enable=1	PTC_enable=0
Normal operation (LIN protocol running) - sleep mode=idle - sleep mode=standby	 932µA 500µA	 724µA 300µA
Sleep only: idle mode(1)	870µA	628µA
Sleep only: standby mode(2)	355µA	24µA

Note:

1. LIN slave in idle mode with no RX/TX data functionality.
2. LIN slave is in standby mode with no RX/TX data functionality.
3. PTC power consumption can be further reduced by utilizing only one sense channel used as one Low Power Sensor (see application note: AT12405 Low Power Sensor Design with PTC).
4. Baud rate = 19200bps. LIN frame spacing = 10ms for all measurements.

7. Appendix A - LIN Node Configuration

Node configuration of a LIN node is provided using the LIN driver conf_lin.h and lin_driver.h files. Peripheral mapping is configured using the lin_hw_mapping.h file.

- conf_lin.c, conf_lin.h – sets up data structures, vars and parameters
- lin_driver.c, lin_driver.h – defines LIN driver functions
- lin_hw_mapping.h – includes mapping: SERCOM mapping (0 to 5), LIN TC instance (3 to 7), port IO mapping

Table 7-1. Common LIN Node Configuration Settings

Parameter	Value	Description
LIN_BAUD_BPS	19200	Sets baud rate to 19.2k
LIN_NMBR_FRAMES	1	Defines the LIN number of frames
LIN_SERCOM_INSTANCE	4	Selects SERCOM instance
LIN_SERCOM_GCLK_GENERATOR	GCLK_ GENERATOR_0	Uses gclk0 to drive the UART
LIN_SERCOM_GCLK_FREQ_HZ	8000000	Sets gclk frequency to 8MHz
LIN_TC_INSTANCE	6	Selects TC instance
LIN_TC_GCLK_GENERATOR	GCLK_ GENERATOR_0	Uses gclk0 to drive TC
LIN_TC_GCLK_FREQ_HZ	8000000	Sets gclk frequency to 8MHz
LIN_TX_PIN	PIN_PB10	Defines UART TX pin (output to TX pin on ATA663231 LIN TRX). EXT3-14.
LIN_RX_PIN	PIN_PB11	Defines the UART RX pin (input from TX pin on ATA663231 LIN TRX). EXT3-13.

Parameters shown below are set using #define directives in the conf_lin.h file. A LIN node sets its type by defining the SELECT_LIN_MASTER parameter if it is to be configured as a LIN master (this parameter does not need to be defined for LIN slaves).

Each node can also select the LIN_SLAVE_INTERRUPTS parameter if interrupts are to be used instead of bit polling.

Table 7-2. Individual LIN Node #Define Settings

Parameter	Value	Description
SELECT_LIN_MASTER	defined	Defines the master node. It also enables the LIN timer to be used as a TC peripheral
LIN_SLAVE_INTERRUPTS	defined	Sets the slave node to be interrupt driven

See example configuration for setting LINSERCOM before entering standby mode for the LIN slave node:

```
/* Enable USART gclk in PM */
```

```
PM->APBCMASK.reg |= PM_APBCMASK_LIN_SERCOM;

/* Assign GCLK0 to SERCOM4 and enable the clock */
GCLK->CLKCTRL.reg = (LIN_SERCOM_GCLK_ID_CORE << GCLK_CLKCTRL_ID_Pos) |
(LIN_SERCOM_GCLK_GENERATOR << GCLK_CLKCTRL_GEN_Pos) | GCLK_CLKCTRL_CLKEN;

/* Disable unneeded clocks in PM module by disabling enable bits APBCMASK registers (this
disables ADC gclk) */
PM->APBCMASK.reg &= ~(PM_APBCMASK_ADC);

/* Configure USART peripheral
...disable USART first so BAUD rate can be configured */
LIN_SERCOM->USART.CTRLA.reg = 0x00000000;
while ( LIN_SERCOM->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_MASK ) { /* ...wait for
synchronization */}

/* Set BAUD rate reg */
LIN_SERCOM->USART.BAUD.reg = BAUD_FP | BAUD_INT;
while ( LIN_SERCOM->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_MASK ) { /* ...wait for
synchronization */}

/* Config CTRLB reg.*/
// ... TX enabled, RX enabled (2-stop bit, 8 data bits) - break+autobaud will not operate w/
1 stop bit following sync field...
LIN_SERCOM->USART.CTRLB.reg = SERCOM_USART_CTRLB_SFDE | SERCOM_USART_CTRLB_SBMODE |
SERCOM_USART_CTRLB_TXEN | SERCOM_USART_CTRLB_RXEN;
while ( LIN_SERCOM->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_MASK ) { /* ...wait for
synchronization */}

// ... clear all interrupt flags & error flags
LIN_SERCOM->USART.INTFLAG.reg = SERCOM_USART_INTFLAG_MASK;
LIN_SERCOM->USART.STATUS.reg = SERCOM_USART_STATUS_MASK;

/* Config CTRLA reg */
USART.CTRLA.reg LIN_SERCOM->USART.CTRLA.reg = SERCOM_USART_CTRLA_DORD |
SERCOM_USART_CTRLA_FORM(4) \
                                                    | SERCOM_USART_CTRLA_SAMPA(0) |
SERCOM_USART_CTRLA_RXPO(LIN_RX_RXPO) \
                                                    |
SERCOM_USART_CTRLA_TXPO(LIN_TX_TXPO) | SERCOM_USART_CTRLA_SAMPR(1) \
                                                    |
SERCOM_USART_CTRLA_MODE_USART_INT_CLK | SERCOM_USART_CTRLA_ENABLE ;

while ( LIN_SERCOM->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_MASK ) { /* ...wait for
synchronization */}


/* Enable Interrupts when LIN_SLAVE_INTERRUPTS is defined */
 LIN_SERCOM->USART.INTENSET.reg = SERCOM_USART_INTENSET_RXC | SERCOM_USART_INTENSET_RXBRK |
SERCOM_USART_INTENSET_ERROR;

NVIC_EnableIRQ(LIN_SERCOM_IRQ);

/* Set up OSC8M prescaler */
SYSCTRL->OSC8M.reg = (SYSCTRL->OSC8M.reg & (~SYSCTRL_OSC8M_PRESC_Msk)) |
SYSCTRL_OSC8M_PRESC(3);
```

Enabling IO ports for the USART TX and RX pins

```
/* Configure USART TX port */
       PORT->Group[LIN_TX_PIN/32].OUTSET.reg = 1u << (LIN_TX_PIN % 32);
    PORT->Group[LIN_TX_PIN/32].DIRSET.reg = 1u << (LIN_TX_PIN % 32);
    PORT->Group[LIN_TX_PIN/32].PMUX[(LIN_TX_PIN % 32)/2].reg &= ~(PORT_PMUX_PMUXE_Msk <<
((LIN_TX_PIN % 2) * 4 ));
    PORT->Group[LIN_TX_PIN/32].PMUX[(LIN_TX_PIN % 32)/2].reg |= (LIN_TX_PMUX << ((LIN_TX_PIN
% 2) * 4 ));
    PORT->Group[LIN_TX_PIN/32].PINCFG[LIN_TX_PIN % 32].reg = PORT_PINCFG_PMUXEN;

/* Configure USART RX port */
       PORT->Group[LIN_RX_PIN/32].DIRCLR.reg = 1u << (LIN_RX_PIN % 32);
    PORT->Group[LIN_RX_PIN/32].PMUX[(LIN_RX_PIN % 32)/2].reg &= ~(PORT_PMUX_PMUXE_Msk <<
((LIN_RX_PIN % 2) * 4 ));
    PORT->Group[LIN_RX_PIN/32].PMUX[(LIN_RX_PIN % 32)/2].reg |= (LIN_RX_PMUX << ((LIN_RX_PIN
% 2) * 4 ));
    PORT->Group[LIN_RX_PIN/32].PINCFG[LIN_RX_PIN % 32].reg = PORT_PINCFG_PMUXEN;
```

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**