Implementation of AUTOSAR Communication Stack with SCI-UART

Prathiba M. Patil
Dept. of Electronics and Communication Engineering
Bangalore Institute of Technology
Bengaluru, Karnataka, India
prathibapatil70@gmail.com
Contact No: 9663887402

Abstract— AUTOSAR (Automotive Open System Architecture) is an open and standardized software architecture used in automotive industry for vehicle software. AUTOSAR architecture is layered and its layers are: ASW, RTE and BSW. BSW is grouped functionally into functions or stacks. In this paper, AUTOSAR Communication stack is implemented with SCI-UART and I/O driver module for asynchronous serial communication. Serial communication output is visualized on Hyper Terminal.

Index Terms— AUTOSAR (Automotive Open System Architecture), ASW, BSW, MCAL (MicroController Abstraction Layer), RTE (Runtime Environment) , SCI (Serial Communication Interface), UART.

1 INTRODUCTION

AUTOSAR – AUTomotive Open System ARchitecture is a software platform developed as a solution for the software demands in automotive embedded systems. AUTOSAR is an open and standardized software architecture for vehicle software. It is developed jointly by automobile manufacturers (OEM's), suppliers and tool vendors working in coordination.

AUTOSAR is completely dedicated to Electronic control units (ECUs). AUTOSAR software components is used to satisfy the functions of ECU systems. This standard uses the component based software design model with defined interfaces. AUTOSAR uses a layered architecture shown in Fig 1, having three software layers which runs on a Microcontroller: Application Layer (ASW), Runtime Environment (RTE), Basic software (BSW) . Application Layer (Software components) is developed independent of the Base Software and Hardware [1].

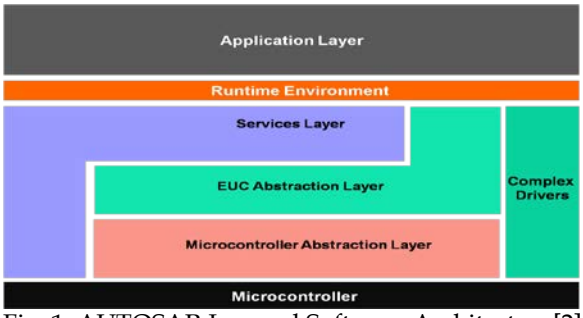


Fig. 1. AUTOSAR Layered Software Architecture[2]

Basic software does not have functionality, but it provides hardware dependent and hardware independent services to the above layer (RTE). This layer is realized through the use of APIs (Application Programming Interfaces). Basic software (BSW) is further divided into Services layer, ECU Abstraction Layer, Microcontroller Abstraction Layer (MCAL) and complex drivers. Fig. 1. shows the layered software architecture of AUTOSAR.

This paper presents implementation of AUTOSAR Communication stack with SCI-UART driver to ECU for asynchronous serial communication. BSW is further grouped vertically as functional groups or stacks. Communication Stack within BSW includes Communication drivers, Communication Hardware abstraction and Communication services. Fig. 2. shows AUTOSAR communication stack within BSW.

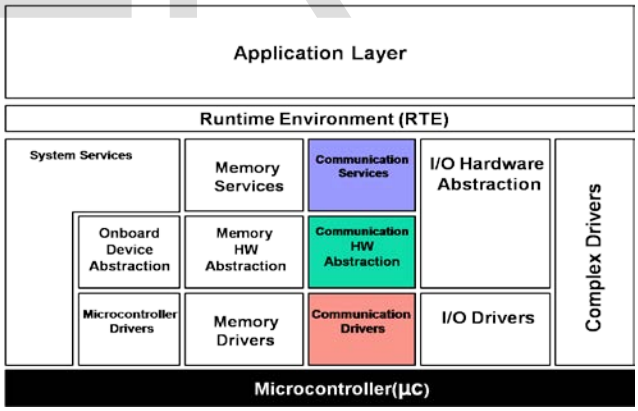


Fig. 2. Functional groups of AUTOSAR Architecture [2]

2 AUTOSAR LAYERED ARCHITECTURE

2.1 Application Software Layer

AUTOSAR Software is an application software that exists above RTE, and it contains Software components which realize the functionality of ECU. Application Layer contains Application software components, Sensor and actuator software components. These AUTOSAR software components (SWCs) are interconnected and communication with each other takes place over two kinds of ports: Client/Server and Sender/Receiver ports in synchronous or asynchronous environment. This communication between SWCs can be either ECU

local communication or network based communication.

2.2 AUTOSAR Interfaces

An AUTOSAR interface defines the ports of software components and BSW modules through which communication between SWCs and BSW modules takes place. Types of interfaces are:

- AUTOSAR Interface: It is provided by RTE and serves as interface between SWCs or between SWCs and ECU. With these interfaces SWCs can read input values and write an output value. [2]
- Standardized Interface: This interface is predefined as API and is used between BSW modules, between RTE and OS, between RTE and communication layer.

2.3 Runtime Environment (RTE)

RTE is the layer between Application layer and BSW layer. RTE realizes two software functions: Communication and Scheduling. RTE helps in information exchange between Application SWCs and connects to the right ECU. It separates the ASW components from the hardware. Main task of RTE is to make the above layer hardware independent. However, RTE is specifically generated for different ECUs and hence it is ECU and application dependent.

2.4 ECU Abstraction Layer

ECU abstraction layer is on top of MCAL layer. It responds to functions of application software and connects to MCAL. It defines APIs to access the peripherals, microcontroller and drivers of external devices connected. It makes above layers independent of ECU hardware layout.

2.5 Microcontroller Abstraction Layer

Microcontroller Abstraction layer is the lowest layer in BSW. It contains internal device drivers with direct access to microcontroller and peripherals. MCAL is hardware dependent and can access memory and registers mapped to ECU.

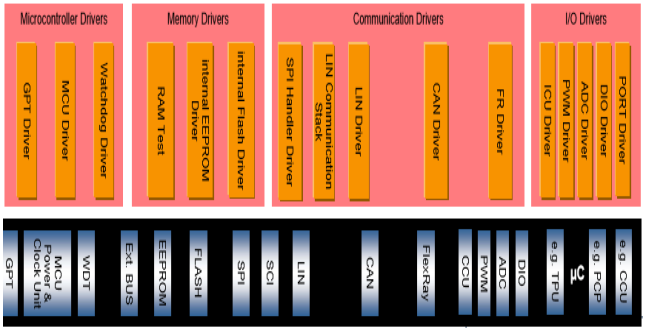


Fig. 3. AUTOSAR MCAL Layer [2]

MCAL consists of four types of modules. Fig. 3. shows MCAL modules:

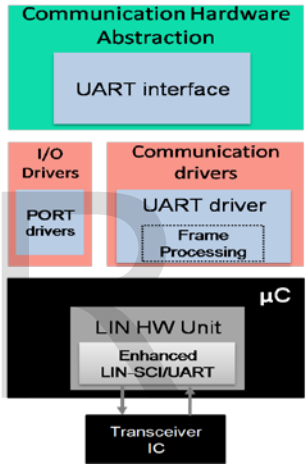
- Microcontroller Drivers (General Purpose Timer, MCU and Watchdog drivers)
- Memory Drivers (internal flash, internal EEPROM) and external memory mapped drivers (External flash).
- Communication Drivers (CAN, Flexray, LIN, Ethernet)
- Input/Output Drivers (ADC, DIO, PWM drivers.)

In this paper, LIN driver operating in UART mode is implemented and PORT driver is used to configure port for serial communication.

3 AUTOSAR COMMUNICATION STACK

Layers of BSW are divided further into functional groups. One such functional group is Communication stack. The Communication stack consists of Communication drivers, Communication hardware abstraction and Communication services from microcontroller to RTE and SWCs. [4] Communication stack is implemented with the specifications of the communication protocol used. Universal Asynchronous Receiver/Transmitter is the Communication protocol used. Fig. 4. shows AUTOSAR communication stack implemented with UART software.

Fig. 4. AUTOSAR communication stack with UART software layering [4]



3.1 Architectural Overview

The UART driver is a communication driver and is a part of Microcontroller abstraction Layer (MCAL), that access the hardware and provides a hardware independent API to the above layer. The only upper layer that has access to the UART driver, is the UART interface which is a part of Communication Hardware Abstraction Layer and uses UART driver to get access to Communication controller. UART interface module provides generally specified interface to the communication system for the upper layers. PORT driver is an Input/ Output driver which configures the port pins as transmit pin and receive pin used for serial communication [6].

3.2 Classification of LIN Hardware Unit

The LIN hardware unit combines one or more LIN channels. Classification of different LIN hardware types connected to LIN physical channels is shown in Fig. 5. LIN hardware is configured to use as Serial communication interface (SCI)-UART. Each UART channel is connected to single LIN cluster through Transceiver IC.

3.3 Functional Overview

LIN/UART hardware unit is a LIN communication controller. LIN/UART interface is provided with different modes. According to the application, appropriate mode has to be used: LIN master, LIN slave or UART. UART can be operated in two modes:

- LIN reset mode
- UART mode

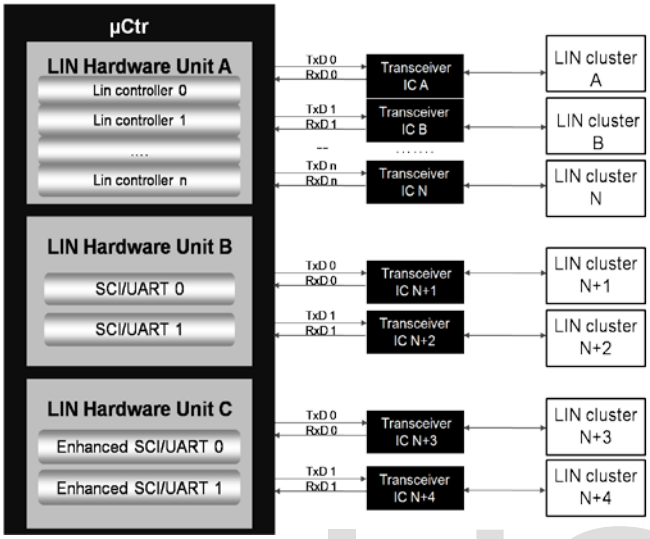


Fig. 5. Classification of LIN hardware unit [6]

3.4 UART driver dependencies on other modules

UART driver depends on the following modules:

- MCU module: The LIN communication clock/ LIN peripheral clock of the internal LIN hardware unit depends on the system clock, Prescalar division and PLL. UART bit timings depends on the clock settings of the MCU module.[6]
- PORT module: Port driver configures the port pins used for UART driver as input or output and selects the alternate functionality of the port pins to be used.
- Operating System (OS): UART driver uses interrupts for transmission complete, reception complete and error detection. Hence there is a dependency on the OS, which configures interrupt sources and handles interrupts on its occurrence.

4 Implementation Of SCI-UART Communication Stack

4.1 Implementation of UART driver

LIN hardware unit to be configured to operate as UART requires the unit to be in LIN reset mode. LIN unit can change its mode only from LIN reset mode to other mode. Changes between any other modes are not supported.

Steps for UART initialization:

- Set LIN hardware unit to LIN reset mode
- Configure UART related registers: write the values to the registers only after LIN reset.
- Set the Baud rate with value calculated using the formula shown in equation (1).
- Set the data field configuration with specified data format.
- Set Interrupt generation timing
- Set mode of operation to UART mode
- Cancel or wake up from LIN reset mode.
- Enable transmission and reception.

Transmission function is implemented for single byte transmission. Reception function reads the receive buffer which has received byte.

1. Calculation of Baud rate:

UART Baud rate is calculated using the following formula:

$$Baud\_Rate = \frac{\text{LIN Peripheral clock frequency} \times \text{Prescalar clock select}}{\text{Baud rate Prescalar} \times \text{Bit Sampling count}} \quad (1)$$

LIN peripheral clock frequency is derived from System clock and PLL. Prescalar clock selects the frequency division ratio. LIN clock is divided by this Prescalar. In UART mode, number of Bit sampling varies from 6 to 16 samplings. Bit sampling count gives number of bit samples in one Tbit (1/Baud rate).

4.2 Configuration of MCAL

The application requires PORT pins to be configured for serial data transmission and reception [5].

1. Configuration of PORT driver

PORT Pin configuration for TransmissionSelection.

- Pin initial mode: GPIO
- Pin direction: Output
- Configure pin present mode: Alternative function (LIN Transmission)

PORT Pin configuration for Reception

- Pin initial mode: GPIO
- Pin direction: Input
- Configure pin present mode: Alternative function (LIN Reception)

2. Configuration of UART driver

UART driver is designed with following configurations:

Baud rate	: 115200bps
Parity	: none
Data bits	: 8
Number of start bits	: 1
Number of stop bits	: 1
Flow control	: none
Data Flow	: LSB first
Space configuration between successive bytes (1 Tbit = 1/baud rate)	: no space or 0Tbits
Reception complete	: Interrupt driven

UART uses 1 start bit, 8 bits for data, no parity and 1 stop bit. Thus, it takes 10 bits to transmit or receive a byte of data.

4.3 UART Interface

UART interface access the UART driver to provide services to upper layers. It calls the right API to access the hardware for transmission and reception. UART errors are handled and

necessary actions are taken care in this layer. UART errors include :

- Framing Error: This error occurs during reception due to Baud rate mismatch. If transmitter and receiver's baud rates are not matched, then during reception start bit is not detected leading to framing error. Error handling requires reconfiguring Baud rates to match for transmission and reception.
- Over run error: This error occurs when the next byte reception is completed before the previous byte received is read. To exit from this error mode, discard the byte received and clear the error status register.
- Bit Error: Bit error is seen during transmission. Transmitted data and the data monitored at the receive pin does not match due to sampling point difference. To correct this error, reconfiguring LIN module to proper bit sampling point is required.

When any or combination of these UART errors occur, an Error detection Interrupt is generated. Type of UART error can be known in this ISR and necessary action is done in UART interface.

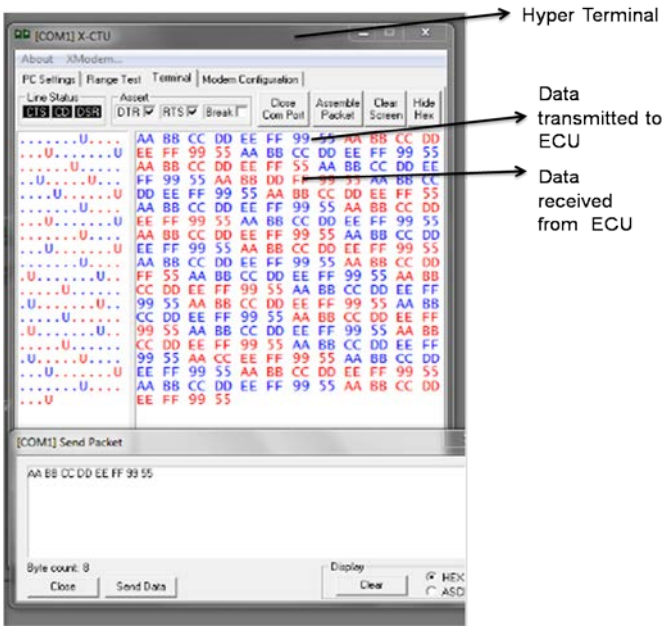
4.4 Calling sequence

After Powering ECU, initialization of ECU and OS is done. The operating system enables the task of highest priority which calls the Runnable entity related to testing serial communication. The sequence is carried out via the following modules: OS, RTE, Communication Hardware Abstraction, Communication driver and Microcontroller Port pins connected to HyperTerminal via Transceiver IC.

The calling sequence to transmit and receive data to and from HyperTerminal is as follows:

- OS enables Reception complete and Error detection interrupt and maps the receive ISR to interrupt priority level vector.
- OS Task is scheduled to call the TestCommunication() function.
- ASW TestCommunication() calls the Port_init of Port driver for port pin configuration.
- It also calls UART_init() for uart initialization and enables communication.
- Once communication is enabled, Data_transmission() of UART interface is called to transmit multiple data bytes.
- Data_transmission() calls UART_transmit() of UART driver which puts data onto transmit buffer and microcontroller sends the data byte out over port pin configured as transmit pin.
- Transmitted data is visualized on Hyper Terminal. Thus verifying Transmission. Data is also sent from Hyper Terminal to ECU. LIN controller detects start bit and buffers the data into receive buffer. On detection of stop bit, Receive complete interrupt is generated.
- Receive ISR is executed on reception complete, in which data byte is read out of receive buffer. Received data byte is transmitted back to Hyper Terminal for verification.

Fig. 6. Result of serial communication test visualized in Hyper Terminal.



In Fig. 6. data transmitted from Hyper Terminal (shown in blue) is received at ECU and is sent back to Hyper Terminal (shown in red). This shows UART Communication stack is successfully implemented and tested.

5 Conclusion

With increase in the complexity of vehicle electronics and automotive software, development processes need to be simplified. The shorter development time and software portability are required for standard core solutions. AUTOSAR based software design forms the most comprehensive and promising solutions. In this paper, AUTOSAR Communication stack with simple UART is implemented according to available specification. As shown, AUTOSAR provides an easy deployment of software components with interfaces between the modules. Here UART driver and UART interface is implemented and tested over Hyper Terminal.

References

[1] "AUTOSAR Technical overview, 2014", AUTOSAR Specification Release 4.2.1, retrieved on 14/08/2014.

[2] "AUTOSAR Layered Software Architecture, 2014", AUTOSAR Specification Release 4.2.1, Retrieved on 12/10/2014. "AUTOSAR Basic Software Module, 2014", AUTOSAR Specification Release 4.2.1, Retrieved on 20/12/2014.

[3] Johan Elgered and Jesper Jansson ,"AUTOSAR Communication Stack Implementation With FlexRay" , March 2012.

[4] "AUTOSAR Specification of PORT Driver, 2014", AUTOSAR Specification Release 4.2.1, Retrieved on 12/11/2014

[5] "AUTOSAR SWS LIN Driver, 2014", AUTOSAR Specification Release 4.2.1, retrieved on 28/02/2014