Semiconductors

AN10353

Application of UART in GPS navigation system

Rev. 01 — 1 March 2005

Application note

Document information

Info	Content
Keywords	UART, GPS
Abstract	This application note shows how to combine a GPS module into a navigation system by using a Philips UART.

PHILIPS

Revision history

Rev	Date	Description
01	20050301	Application note (9397 750 14702); initial version.

Contact information

For additional information, please visit: **http://www.semiconductors.philips.com**

For sales office addresses, please send an email to: **sales.addresses@www.semiconductors.philips.com**

1. Introduction

With the rapid development of GPS (Global Positioning System) techniques, GPS gets wider application in many fields. GPS has features such as high precision, global coverage, convenience, high quality and low cost. Recently, the use of GPS extends speedily from military to civilian applications such as automobile navigation systems which combine the GPS system, e-map, and wireless network. GPS is getting popular, and the market for GPS techniques is extending continuously. Figure 1 is a typical diagram of an automobile navigation system.

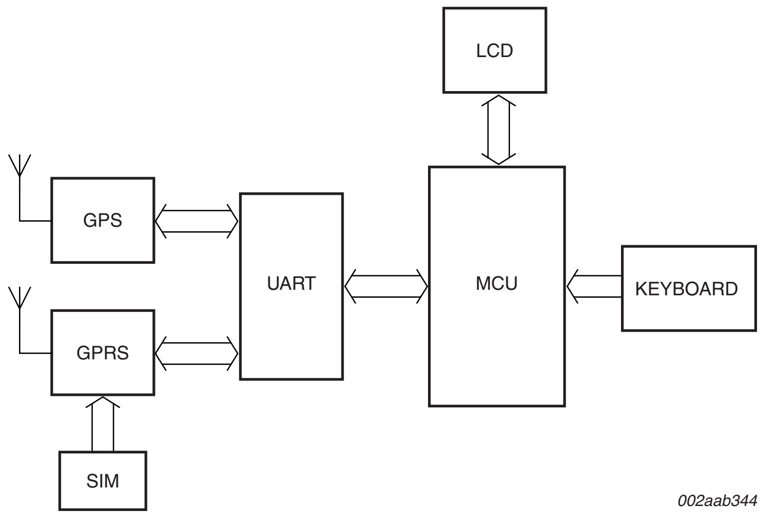

Fig 1. Typical automobile navigation system

2. GPS

GPS mainly includes: antenna interface, DSP, microprocessor, memory, data interface, and a power interface. Figure 2 is a block diagram of a current GPS receiver.

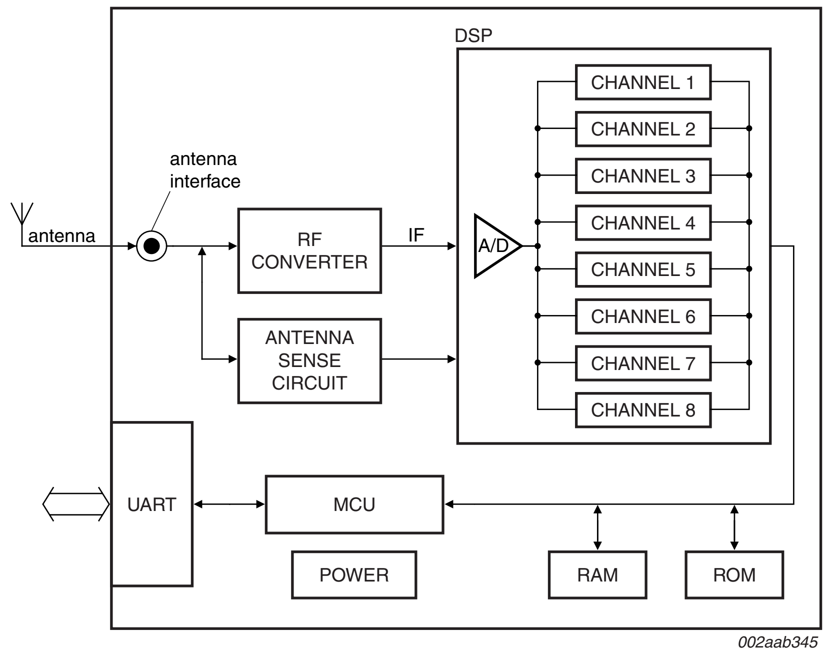

Fig 2. GPS receiver

The data interface of a current GPS can be UART, USB, or CAN. The simpler transport protocol of the UART results in much less software overhead, more cost-effective hardware solution, and with a high performance UART (such as a UART from Philips Semiconductors), the data thoughput on the UART interface is almost comparable to the USB interface.

3. UART

UARTs provide serial asynchronous receive data synchronization, parallel-to-serial and serial-to-parallel data conversion for both the transmitter and receiver sections. These functions are necessary for converting the serial data stream into parallel data that is required with digital systems. Synchronization for the serial data stream is accomplished by adding start and stop bits to the transmit data to form a data character. Data integrity is ensured by attaching a parity bit to the data character. The parity bit is checked by the receiver for any transmission bit errors.

To a host system, the UART appears as an 8-bit input and output port that it can read from and write to. Whenever the host has data to be sent, it just sends these data to the UART in byte format (8-bit wide), whenever the UART receives data from another serial device it will buffer these data in its FIFO (again, 8-bit wide), then it will indicate the availability of these data to the host through an internal register bit, or through a hardware interrupt signal.

In addition to the transmitter and the receiver, UARTs from Philips Semiconductors also incorporate other features that significantly reduce software overhead and increase system efficiency. Some of these features are:

- Wide range of supply voltage: 2.5 V, 3.3 V, 5.0 V
- Hardware and software autoflow control
- Large FIFOs (up to 256 bytes)
- Fast baud rate (5 Mbit/s max.)
- Industrial temperature range: –40 °C to +85 °C
- Fast bus access time (43 ns)
- Sleep mode, where the device current consumption is reduced to about 50 µA
- Small footprint (HVQFN32)

Hardware and software autoflow control prevent FIFO overflow conditions automatically. Without automatic flow control, the host software needs to empty the receive FIFO immediately when it is about to be filled up.

Large FIFOs reduce the host processor service time to the UART; this allows the processor more time to do other tasks. Faster baud rate and faster bus access improve the overall system performance; the system can send/receive more data in less time.

Figure 3 shows an internal block diagram of the SC16C2552B UART.

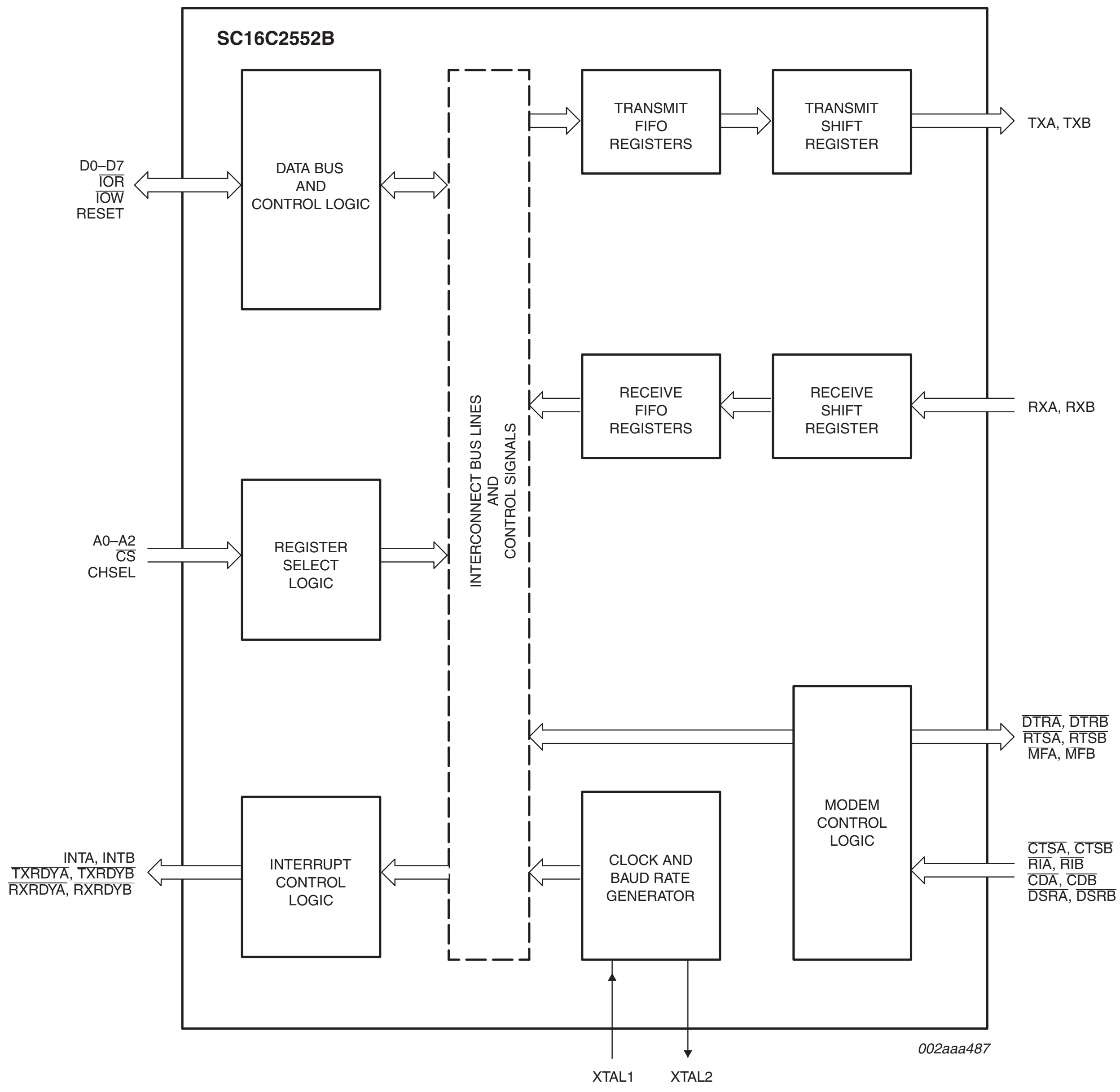

Fig 3. SC16C2552B block diagram

3.1 Data bus and control logic block

The host controller sends and receives data to and from the UART through this block. The control logic block within this block generates various control signals for the internal interconnect bus.

3.2 Register select logic

The register select logic block decodes the addresses from the host to select which UART's internal register the host wants to access.

3.3 Internal registers

The host and the UART communicate through a set of registers. These registers function as data holding registers (THR/RHR), interrupt status and control registers (IER/ISR), a FIFO control register (FCR), line status and control registers (LCR/LSR), modem status and control registers (MCR/MSR), programmable data rate (clock) control registers (DLL/DLH), and a user-accessible scratch pad register (SPR).

3.3.1 Transmit Holding Register (THR) and Receive Holding Register (RHR)

These registers are used to store transmitting and receiving data. The host writes data to THR for transmission, and reads RHR for data received by the UART.

3.3.2 Interrupt Enable Register (IER)

The Interrupt Enable Register is used to enable/disable different types of interrupts supported by the UART. Some of these interrupts are Receive Data Ready, Transmit Empty, Line Status REgister, and Modem Status Register.

3.3.3 FIFO Control Register (FCR)

FCR is used for enabling the FIFOs, clearing the FIFOs, setting transmitter and receiver trigger levels.

3.3.4 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) provides the user with four interrupt status bits. Performing a read cycle on the ISR will provide the user with the highest pending interrupt level to be serviced. No other interrupts are acknowledged until the pending interrupt is serviced.

3.3.5 Line Control Register (LCR) and Line Status Register (LSR)

LCR is used to set the data communication format. The word length, number of stop bits, and parity type are selected by writing the appropriate bits to the LCR. Line Status Register (LSR) provides the status of the data transfer between the UART and a remote UART. This register reports framing error, parity error, overrun error, and other FIFOs status.

3.3.6 Divisor Latch Low (DLL) and Divisor Latch High (DLH)

DLL and DLH store the 16-bit divisor for generation of the baud clock in the baud rate generator. DLH stores the most significant part of the divisor; DLL stores the least significant part of the divisor.

4. UART to GPS interface

This application note describes how to combine GPS into a navigation system by using a Philips 2-channel UART, the SC16C2552B. Figure 4 is a block diagram of the interface; Figure 5 is a hardware schematic diagram.

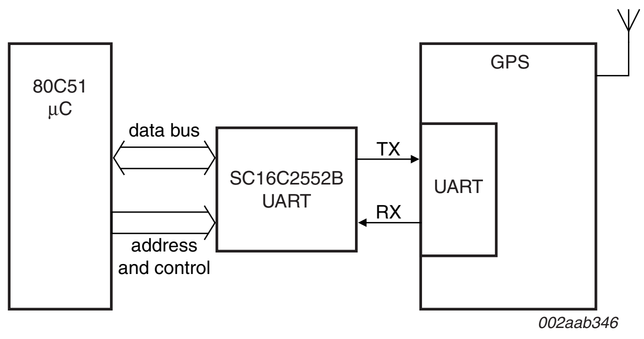

Fig 4. GPS navigation system interface

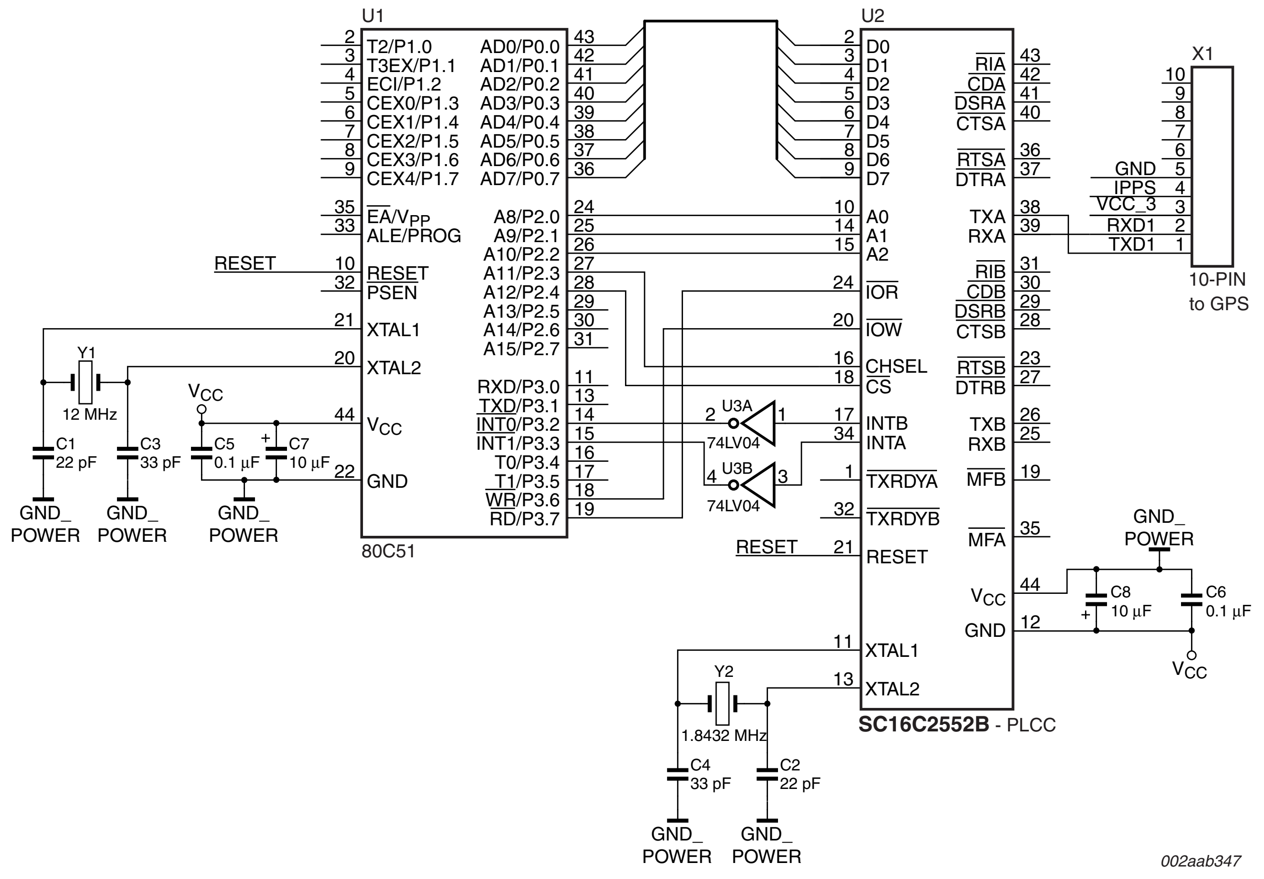

Fig 5. Hardware schematic diagram

5. Software code

The software flowchart is shown in Figure 6.

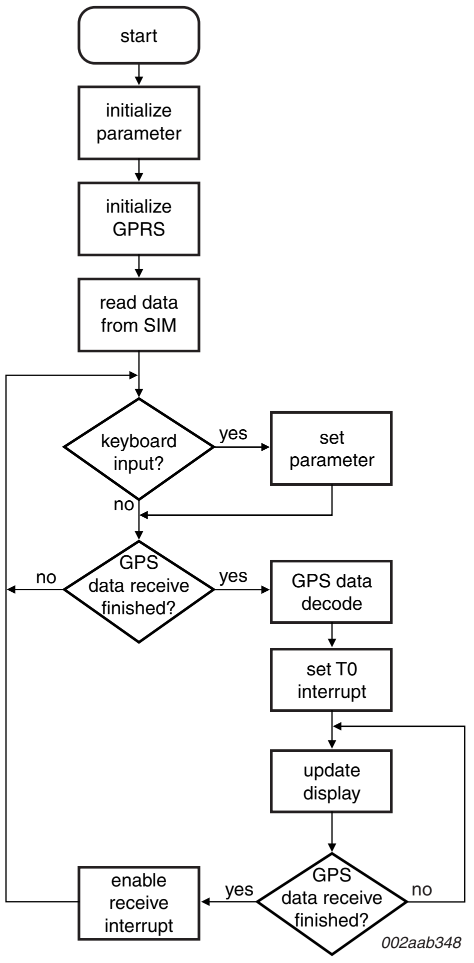

Fig 6. Software flowchart

5.1 UART initialized code

```
#define PORT 0x800          /* defined base address COM1: 0x800 */

Initialize uart:
/*============================================================*/
void INIT_UART(Uint PORT)
{
 XBYTE[PORT+LCR]=0x80;     /* set LCR.7 before set baud-rate   */
 XBYTE[PORT+DLL]=0x0C;     /* crystal oscillation frequency 1.8432 MHz,
                              set baud rate: 9600              */
 XBYTE[PORT+DLM]=0x00;
 XBYTE[PORT+LCR]=0x03;     /* data format, data bit :8; stop bit :1; no parity */
 XBYTE[PORT+IER]=0x05;     /* receive interrupt enable         */
 XBYTE[PORT+MCR]=0x00;
 XBYTE[PORT+FCR]=0x4F;     /* receive interrupt trigger levels :FIFO=4 bytes   */

}

/*Interrupt for receive data from GPS : */
/*============================================================*/
void service_int1() interrupt 2 using 2
{
   ISR_REG=XBYTE[PORT+ISR]
   if((ISR_REG==0xC4)||(ISR_REG==0xCC))     // receive interrupt
     {
      while((inportb(LSR) & 0x01)==0x01)    // if receiver-holding register has data
     { buffer_rx[rx_count++]=XBYTE[PORT];   // read data into buffer
      .
      .
      .
     }
   else if (ISR_REG==0xC2)                  // transmit interrupt
     {
      .
      .
      .
     }
   else if (ISR_REG==0xC0)                  // MODEM interrupt
     {
      .
      .
      .
     }
   else if (ISR_REG==0xC6)                  // receive line status interrupt
     {
      .
      .
      .
     }
}
```

6. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status ‘Production’), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

7. Contents

Date of release: 1 March 2005
Document number: 9397 750 14702

Published in The Netherlands