MICROCONTROLLERS

Chapter 9

STM32 Peripherals - USART

Dr. Saeed Ebadollahi

References:

- ARM® Cortex® M4 Cookbook – Mark Fischer – Packt publishing – 2016
- The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors – Joseph Yiu – Newnes – 2014
- Discovering the STM32 Microcontroller - Geoffrey Brown - 2012

Introduction

After LEDs and pushbuttons, the most basic method for communication with an embedded processor is asynchronous serial. Asynchronous serial communication in its most primitive form is implemented over a symmetric pair of wires connecting two devices – here I'll refer to them as the host and target, although those terms are arbitrary. Whenever the host has data to send to the target, it does so by sending an encoded bit stream over its transmit (TX) wire; this data is received by the target over its receive (RX) wire. Similarly, when the target has data to send to the host it transmits the encoded bit stream over its TX wire and this data is received by the host over its RX wire. This arrangement is illustrated in Figure 5. This mode of communications is called "asynchronous" because the host and target share no time reference. Instead, temporal properties are encoded in the bit stream by the transmitter and must be decoded by the receiver.

Introduction (Cont.)

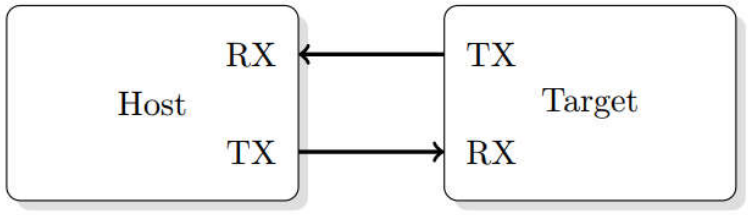

Figure 5.1: Basic Serial Communications Topology

USART

A commonly used device for encoding and decoding such asynchronous bit streams is a Universal Asynchronous Receiver/Transmitter (UART) which converts data bytes provided by software into a sequence of individual bits and, conversely, converts such a sequence of bits into data bytes to be passed off to software. The STM32 processors include (up to) five such devices called USARTs (for universal synchronous/asynchronous receiver/transmitter) because they support additional communication modes beyond basic asynchronous communications. In this Chapter we will explore serial communication between the (target) STM32 USART and a USB/UART bridge connected to a host PC.

USART (Cont.)

UARTs can also be used to interface to a wide variety of other peripherals. For example, widely available GSM/GPRS cell phone modems and Bluetooth modems can be interfaced to a micro-controller UART. Similarly GPS receivers frequently support UART interfaces. As with the other interfaces we'll consider in future chapters, the serial protocol provides access to a wide variety of devices.

USART Frame

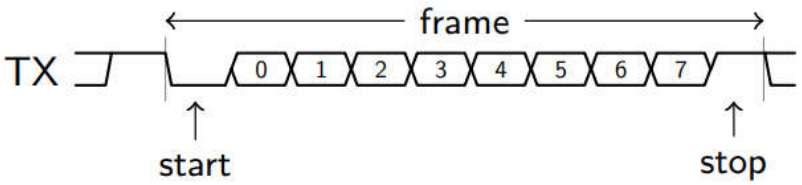

Figure 5.2: Serial Communications Protocol

USART Frame (Cont.)

One of the basic encodings used for asynchronous serial communications is illustrated in Figure 5. Every character is transmitted in a "frame" which begins with a (low) start bit followed by eight data bits and ends with a (high) stop bit. The data bits are encoded as high or low signals for (1) and (0), respectively. Between frames, an idle condition is signaled by transmitting a continuous high signal. Thus, every frame is guaranteed to begin with a high-low transition and to contain at least one low-high transition. Alternatives to this basic frame structure include different numbers of data bits (e.g. 9), a parity bit following the last data bit to enable error detection, and longer stop conditions.

USART Frame (Cont.)

There is no clock directly encoded in the signal (in contrast with signaling protocols such as Manchester encoding) – the start transition provides the only temporal information in the data stream. The transmitter and receiver each independently maintain clocks running at (a multiple of) an agreed frequency – commonly, and inaccurately, called the baud rate. These two clocks are not synchronized and are not guaranteed to be exactly the same frequency, but they must be close enough in frequency (better than 2%) to recover the data.

USART Frame (Cont.)

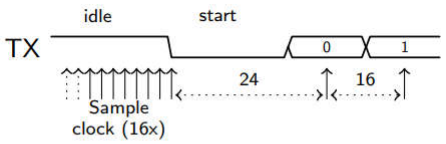

Figure 5.3: UART Signal Decoding

USART Sampling

To understand how the receiver extracts encoded data, assume it has a clock running at a multiple of the baud rate (e.g. 16x) starting from an idle state as illustrated in Figure 3. The receiver "samples" its RX signal until it detects a high-low transition. It then waits 1.5 bit periods (24 clock periods) to sample its RX signal at what it estimates to be the center of data bit 0. The receiver then samples RX at bit-period intervals (16 clock periods) until it has read the remaining 7 data bits and the stop bit. From that point the process repeats. Successful extraction of the data from a frame requires that, over 10.5 bit periods, the drift of the receiver clock relative to the transmitter clock be less that 0.5 periods in order to correctly detect the stop bit.