CC2511 Week 4: Lecture 2

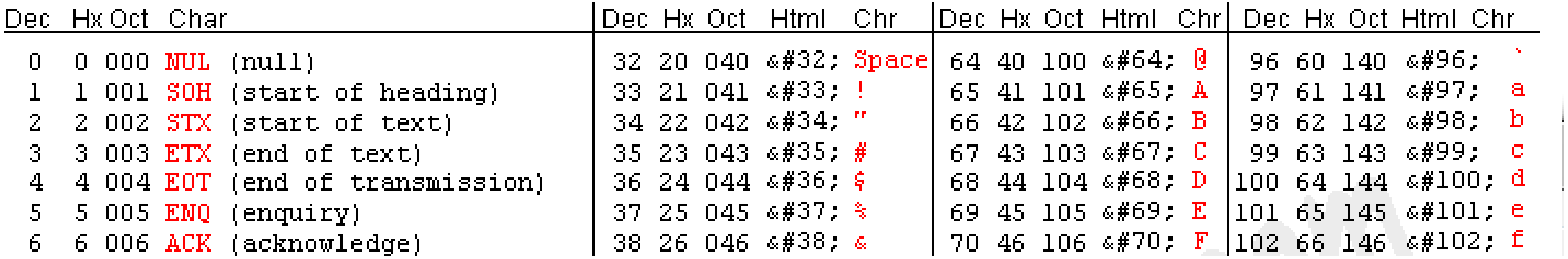
Today: Digital communications and the serial port

This lecture:
Introduction to digital communication

- Character encodings (ASCII, Unicode)
- The serial port standard
- Using the serial port in your programs

Transmitting text

- The traditional method of encoding text uses the ASCII code.
  - ASCII stands for American Standard Code for Information Interchange.
  - One byte per character.



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f

ASCII code pages

- ASCII uses one byte per character, therefore there are $2^8 = 256$ unique characters.
- The standard ASCII table covers English letters and symbols in the first 128 codes.
- Therefore 128 codes remain for other languages. This is not enough!

ASCII code pages

- Different encodings were developed for different languages.
- The “code pages” tells which language to use.
- Example: value 170 could be interpreted as:
  - ς in the Greek code page.
  - ت in the Arabic code page.
  - Ąin the Thai code page.
  - ｪ in the Japanese code page.

Code pages

- The code page system had many problems:
  - No real way to mix languages in a single document.
  - The correct code page must be known in advance.
  - Some scripts needed more than 128 symbols.
- The solution: a globalised encoding called Unicode.

Unicode

- Unicode defines all the symbols in all the world’s languages with numeric identifiers called “code points”.
- Code points are abstract numbers that represent symbols.
- The code points are then encoded using a specific format to produce a sequence of bytes.
- Most encodings have a variable width, e.g. some symbols consume one byte and other symbols consume two bytes.

Unicode encodings

- UTF8: Backwards-compatible with ASCII.
    - English text is the same as ASCII.
    - Other languages use multi-byte characters.
    - Used for most communication systems.
- UTF16: All characters are a minimum of two bytes.
    - Some uncommon characters may be longer than two bytes.
    - Used internally by Windows.

Unicode for the programmer

- Unless specified otherwise, it’s probably safe to assume UTF-8 encoding.
- However, you cannot assume that one byte == one character.
- If your system needs to handle foreign text, you will need to learn more about Unicode.
  - Especially for Asian languages.

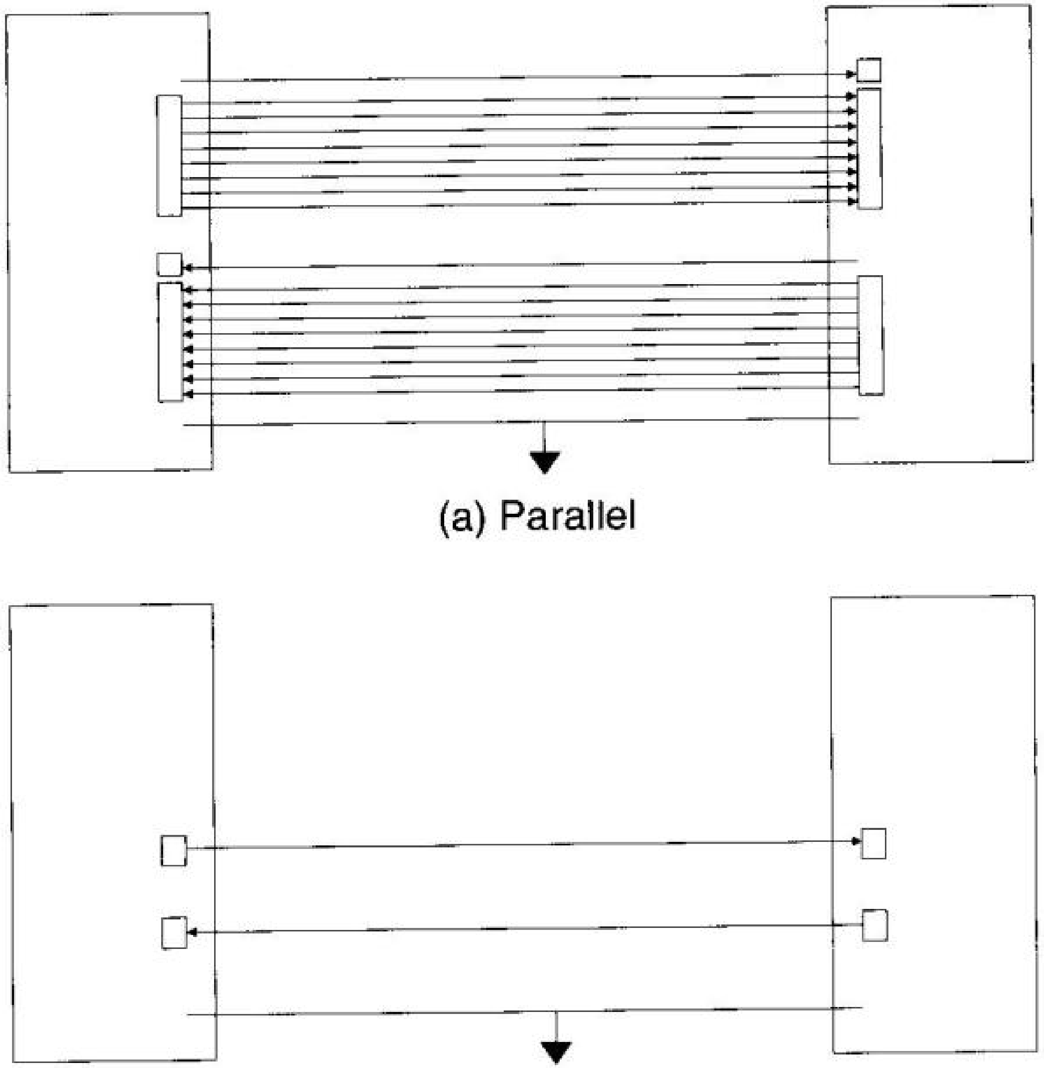
Communication

- How to send our UTF-8 encoded text from one place to another?

- Let’s look at some terminology.

Serial vs parallel

- Parallel: many bits sent simultaneously over many wires.
- Serial: each bit sent one at a time.

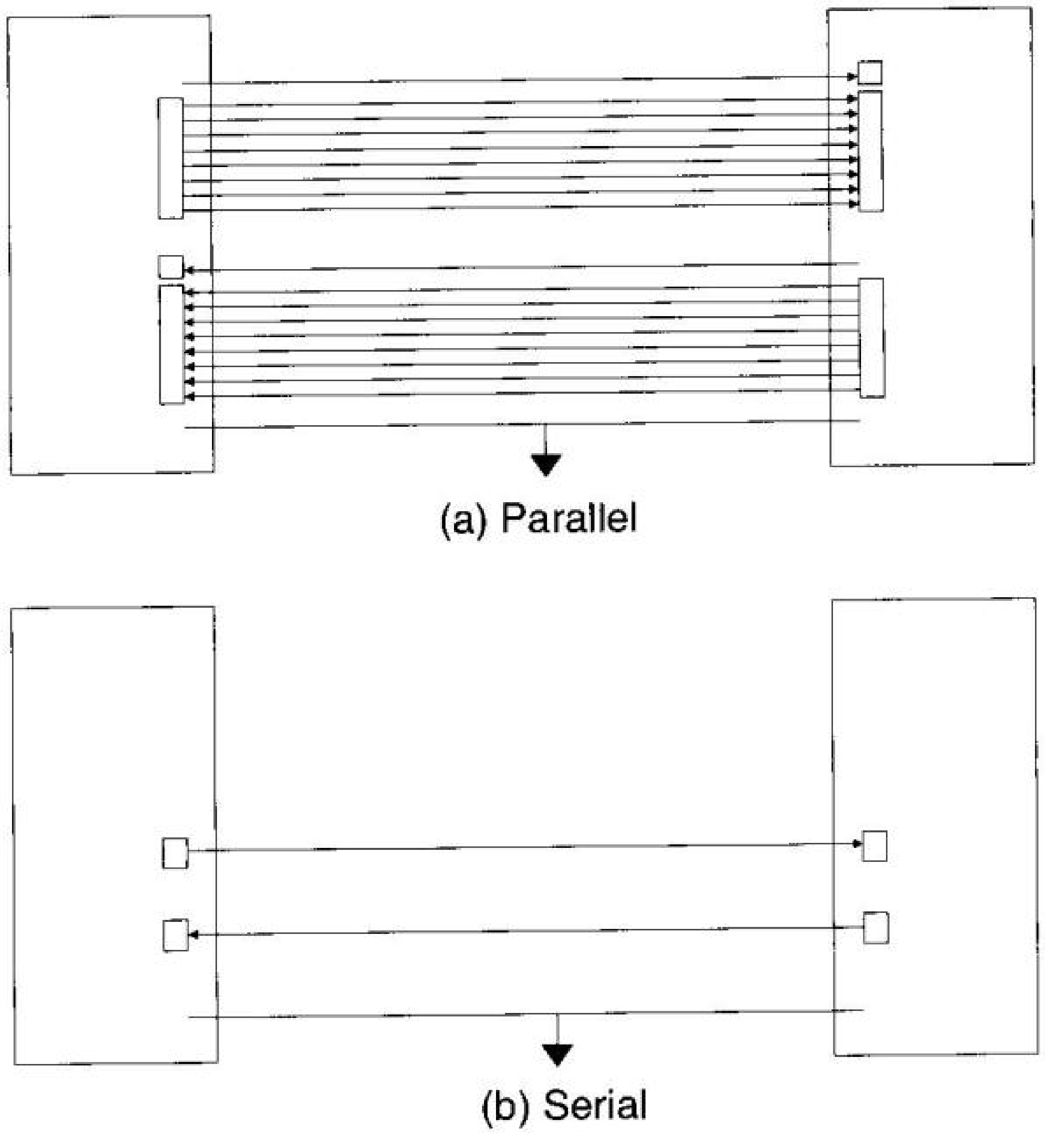


(a) Parallel

(b) Serial

Serial vs parallel

- At a given clock rate, parallel transmission is faster.
- At high speed, parallel systems must be carefully designed to minimise inter-wire interference.
- Many modern interfaces such as USB and Ethernet are series because of simpler and more reliable cabling.


(a) Parallel

(b) Serial

Simplex vs duplex

- Simplex: data is sent in one direction only.
  - e.g. television or radio broadcasting.
- Duplex: data is sent in both directions.
  - e.g. most computer or microprocessor interfaces.

Half vs full duplex

- Half duplex: only one device transmits at a time.
  - Devices take turns transmitting.
  - They must agree on a protocol to decide who transmits when.
  - If both transmit at the same time then a **collision** occurs.
  - Advantage: fewer wires are needed.
- Full-duplex: both devices may transmit simultaneously.
  - Usually separate data lines are used for each direction.
  - Advantage: higher throughput.
  - Disadvantage: more wires are needed.

The serial port

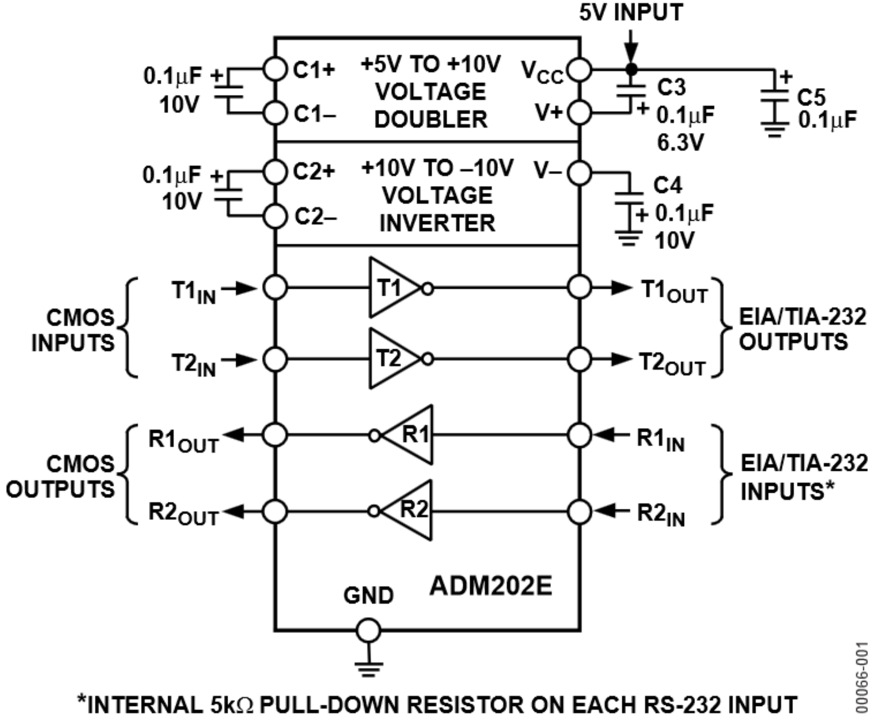
- The **serial port**, also called **COM port** is a simple digital communication system.
- In the past computers often had serial ports built-in using a DB9 connector (see image).
  - Now a USB-to-serial adaptor is needed.
- Very common in microprocessor systems.
- Full duplex.

U.S.A. PATENT 5199906 GENDER CHANGER
U.S.A. PATENT 5199906 GENDER CHANGER

RS232

- DB9 serial ports usually implement the **RS232** standard.
- RS232 defines:
  - Logic 0 as a voltage between +3 V and +25 V.
  - Logic 1 as a voltage between -3 and -25 V.
- Usually a separate RS232 chip is needed to interface with a microprocessor, because the microprocessor won't tolerate 25 V inputs.

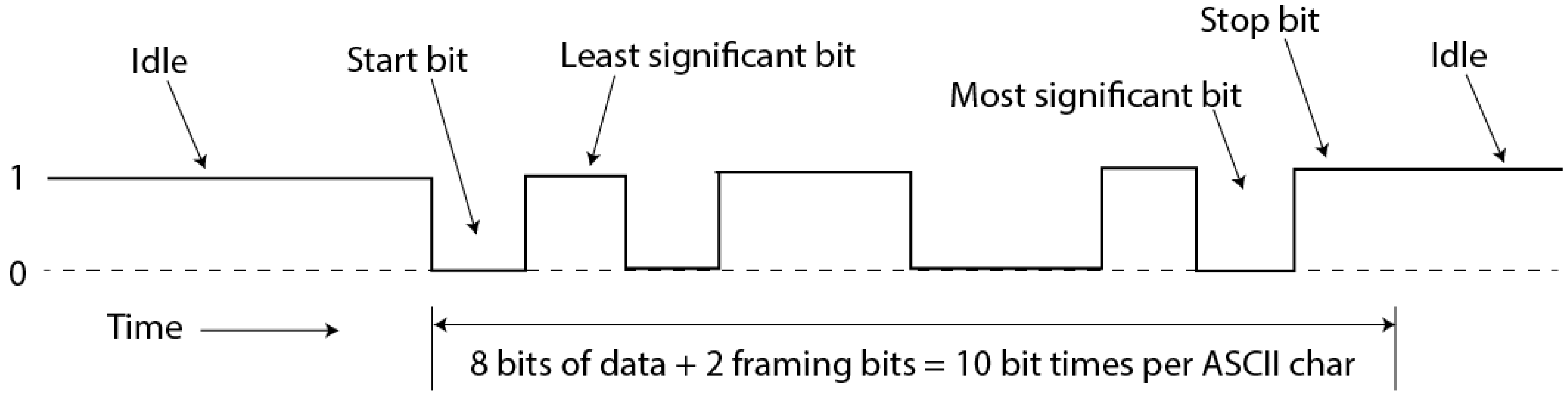
FUNCTIONAL BLOCK DIAGRAMS



*INTERNAL 5kΩ PULL-DOWN RESISTOR ON EACH RS-232 INPUT

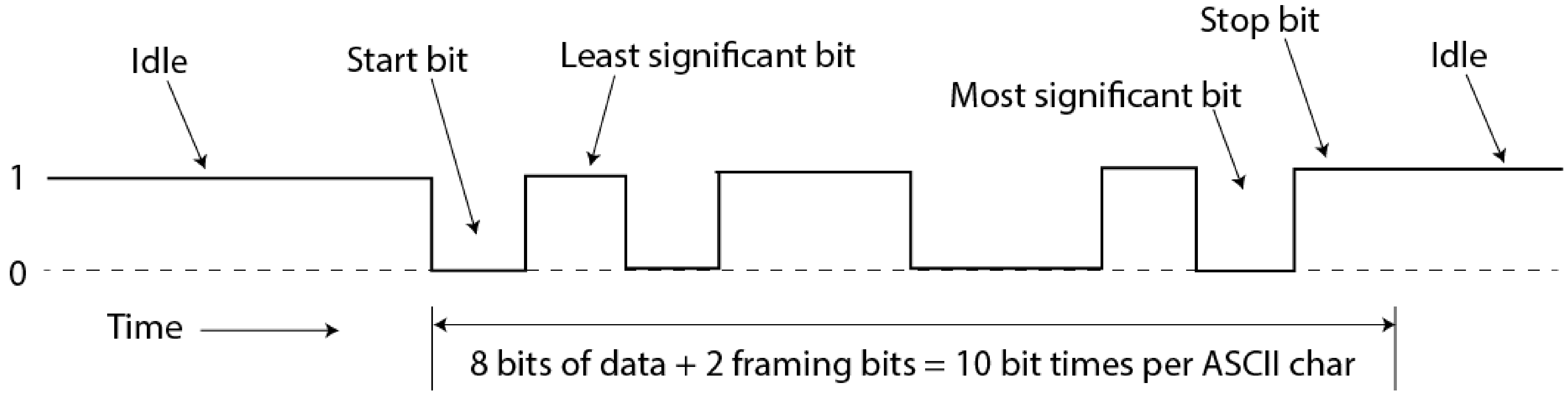
Asynchronous Serial Communications

- Data is only transmitted when available.
- At other times the line is held in an idle state.
- Example of transmitting the binary number 01001101:
  - Notice the least significant bit is transmitted first



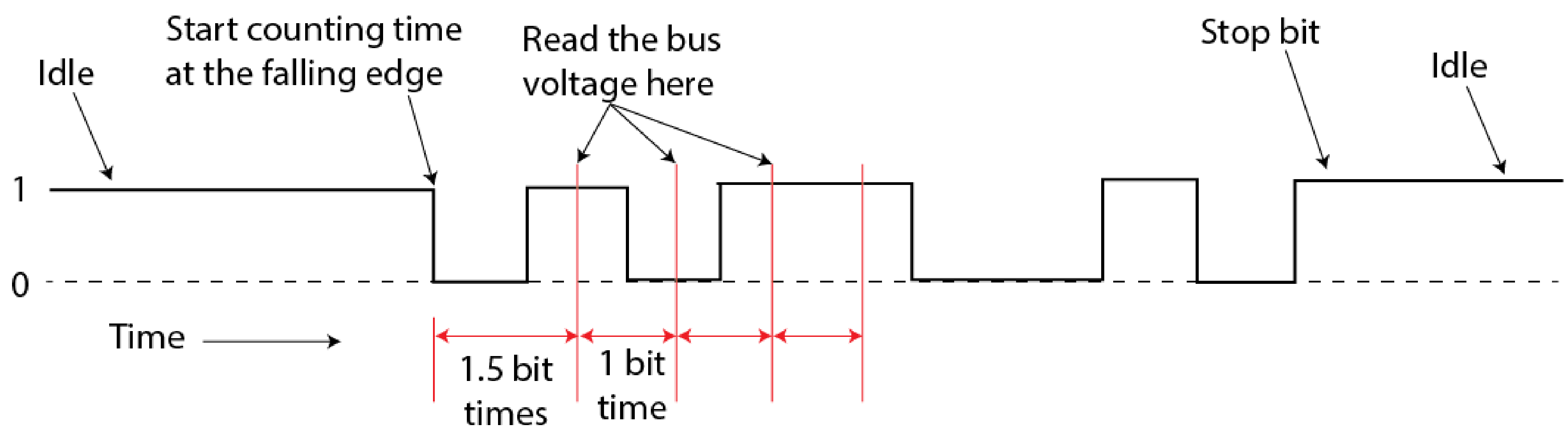
Asynchronous Serial Communications

- At idle the line is held high (logic 1).
- The start of transmission is signalled by a low voltage (logic 0).
- The baud rate (number of bits per second) is agreed in advance.



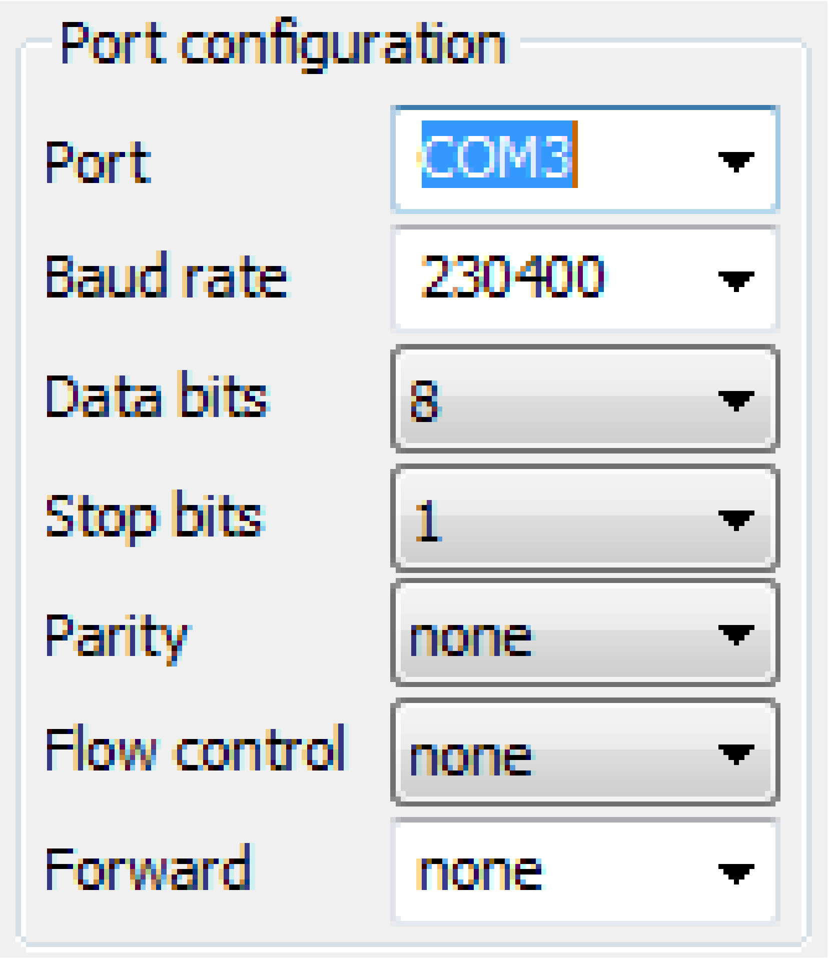
Asynchronous Serial Communications

- The receiver waits for a falling edge on the data line. This indicates transmission is about to begin.
- It then counts half a bit frame and reads a logic 0 (the start bit)
- Next, it counts full bit frames and samples in the middle of the waveform.


Idle
Start counting time at the falling edge
Read the bus voltage here
Stop bit
Idle
1
0
Time
1.5 bit times
1 bit time

Serial port settings

- Both ends of the serial link must be configured in advance with the same settings.
- The **baud rate** is the number of symbols per second (including the stop bit and start bit).
- With default settings there are 10 symbols transmitted per byte, so divide the baud rate by 10 to get the speed in bytes/sec.



Port configuration

Port: COM3

Baud rate: 230400

Data bits: 8

Stop bits: 1

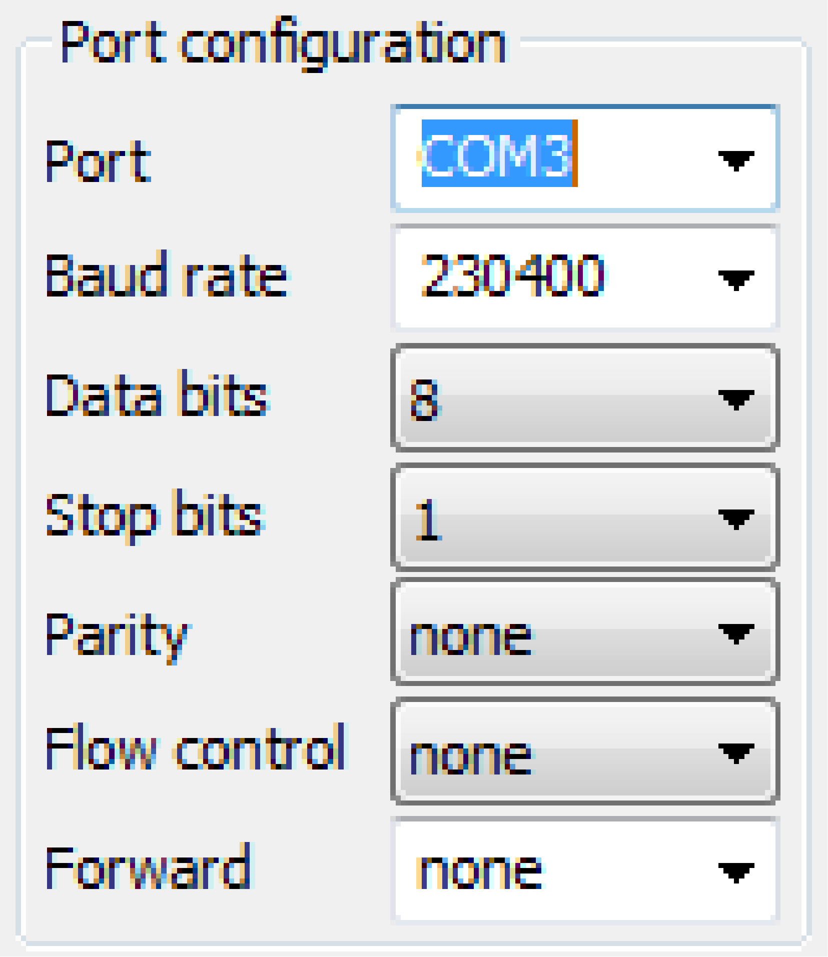
Parity: none

Flow control: none

Forward: none

Serial port settings

- The number of data bits per frame and the number of stop bits can also be configured.
- Almost always each frame has 8 data bits (i.e. one byte) and 1 stop bit.

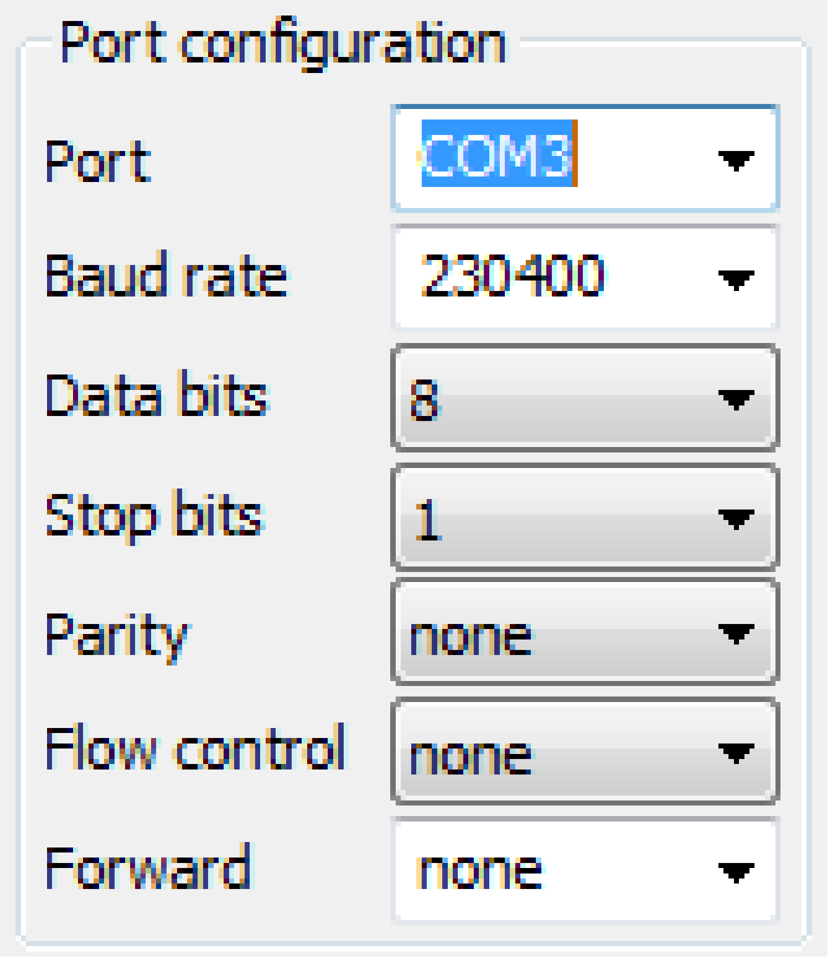


Port configuration

Port	COM3
Baud rate	230400
Data bits	8
Stop bits	1
Parity	none
Flow control	none
Forward	none

Serial port settings

- **Parity** is a simple error checking scheme where an extra bit is added to the frame.
- For **even parity** the extra bit is set so that the total number of 1s is even.
- For **odd parity** the extra bit is set so that the total number of 1s is odd.
- The receiver counts the number of 1s and can detect that an error occurred

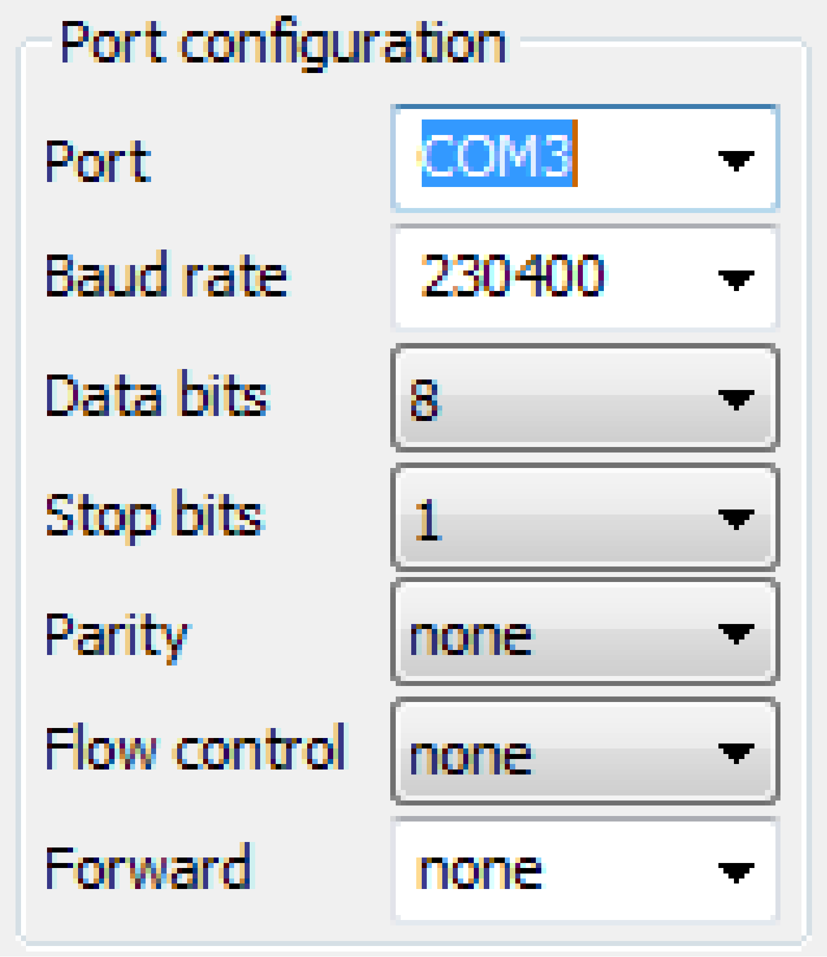


Port configuration

Port	COM3
Baud rate	230400
Data bits	8
Stop bits	1
Parity	none
Flow control	none
Forward	none

Flow control

- **Flow control** allows the receiver to signal whether it is ready to process another message yet.
- The transmitter can wait for the receiver to become ready.
- Rarely used nowadays because modern devices are fast enough to handle full throughput, and have enough memory to queue up received data as it arrives.



Port configuration

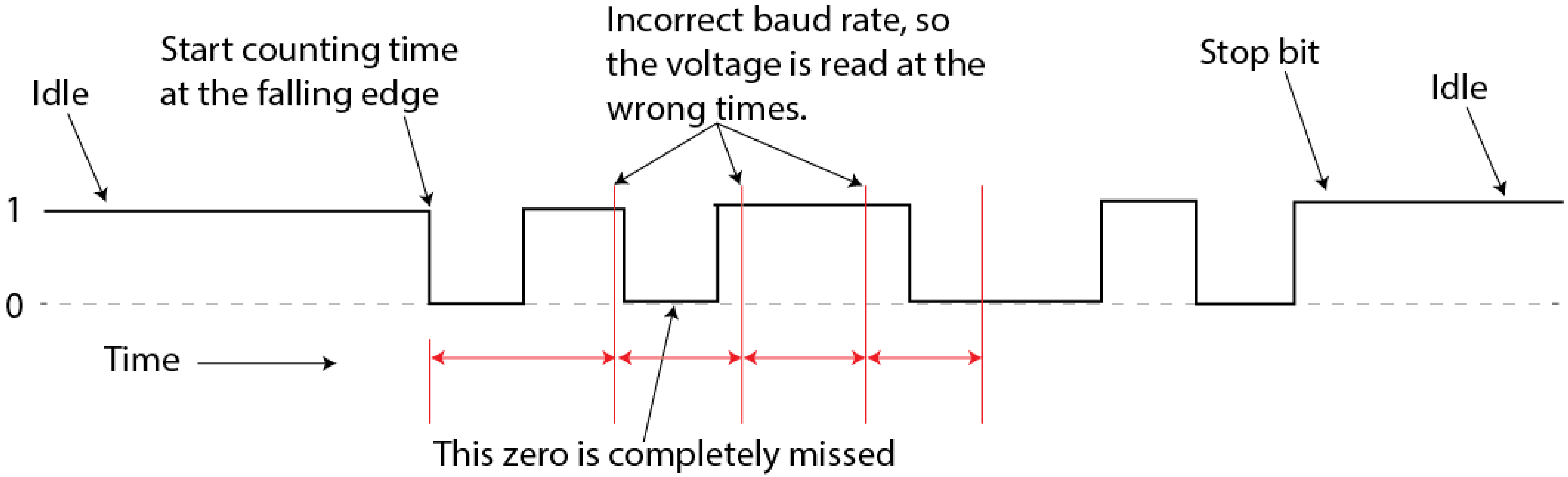
Port	COM3
Baud rate	230400
Data bits	8
Stop bits	1
Parity	none
Flow control	none
Forward	none

Question

- In a serial connection, what will happen if the transmitter and receiver baud rates are not equal?

Answer

- In a serial connection, what will happen if the transmitter and receiver baud rates are not equal?
- Incorrect data will be received.


Idle
Start counting time at the falling edge
Incorrect baud rate, so the voltage is read at the wrong times.
Stop bit
Idle
1
0
Time
This zero is completely missed

UARTs

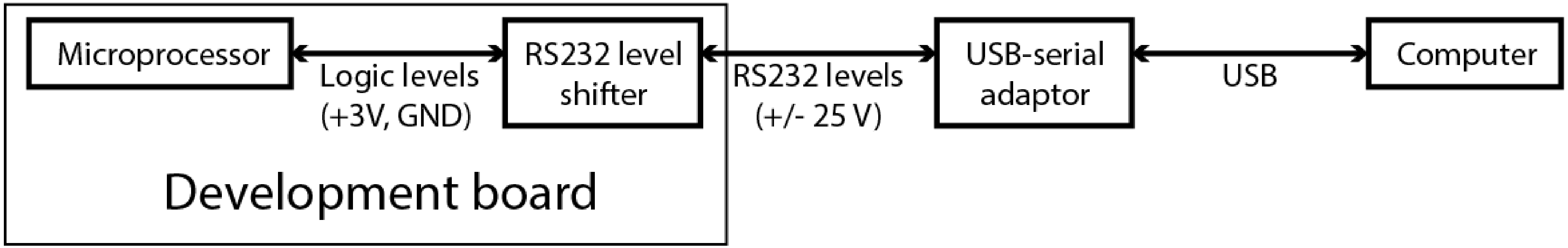
- Most microprocessors include a **universal asynchronous receiver/transmitter (UART)** that encodes and decodes serial communication.
- The baud rate, parity, etc are configurable in software.
- The UART outputs signals at Vcc (+3.3V on the FRDM board) and GND.
  - +5 V is also common.
- An RS232 level shifter is needed to achieve RS232 voltage levels.

USB-to-Serial

- USB-to-serial devices allow a serial port to be added to computers without them.
- The DB9 port implements RS232 levels.
- The USB port appears to the computer as a serial port.

Microprocessor-computer serial interface

Boards with a serial port:



Microprocessor

Logic levels (+3V, GND)

RS232 level shifter

RS232 levels (+/- 25 V)

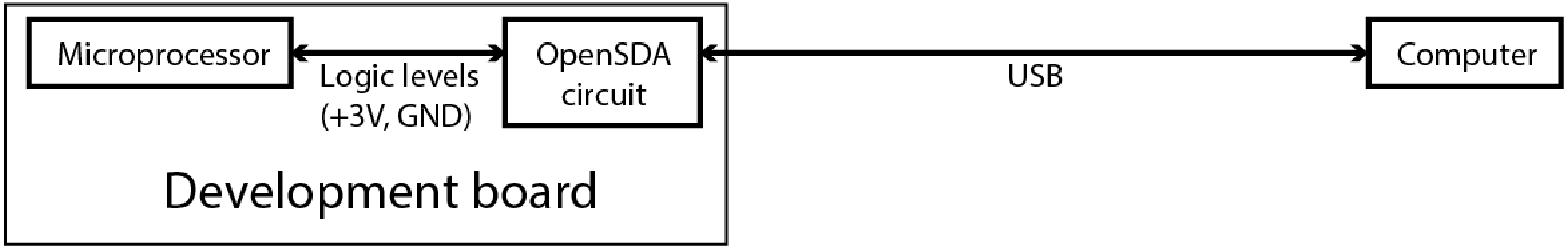
USB-serial adaptor

USB

Computer

Development board

FRDM board: (no serial port; OpenSDA emulates one)



Microprocessor

Logic levels (+3V, GND)

OpenSDA circuit

USB

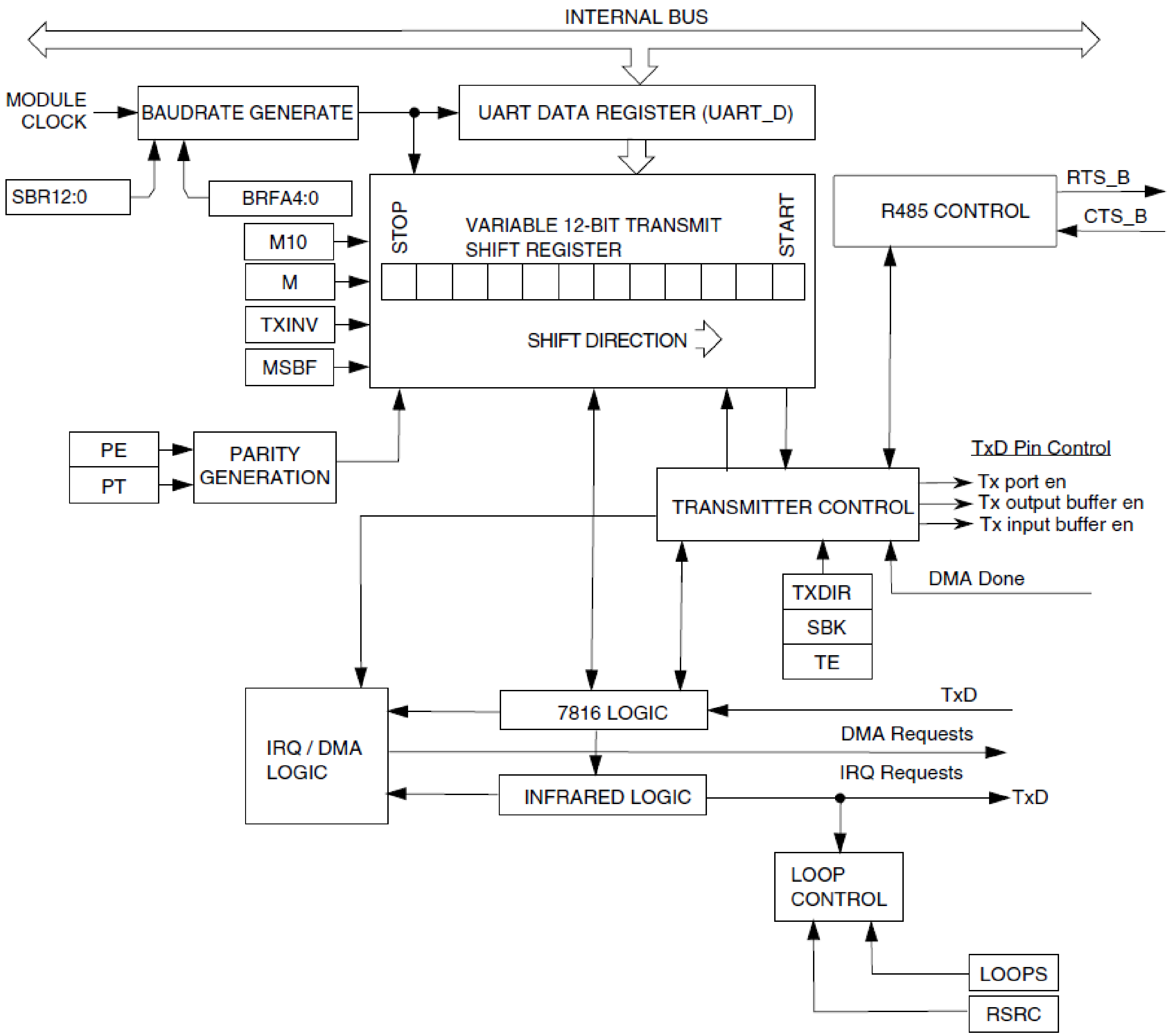
Computer

Development board

The UART in the K20 family

- Reference Manual (Ch. 45 for K20 and Ch. 47 for K22).
- Supports standard serial comms, ISO-7816 (for SIM cards and smartcards), and IrDA (for infrared transmit and receive).
- There are three independent UART modules called UART0, UART1 and UART2.
  - Each can communicate independently on different pins.
- The K22 also has a separate low power UART (LPUART) that can operate while the CPU is stopped. It can wake the CPU upon message receipt.

45.4.2 Transmitter



45.4.3 Receiver

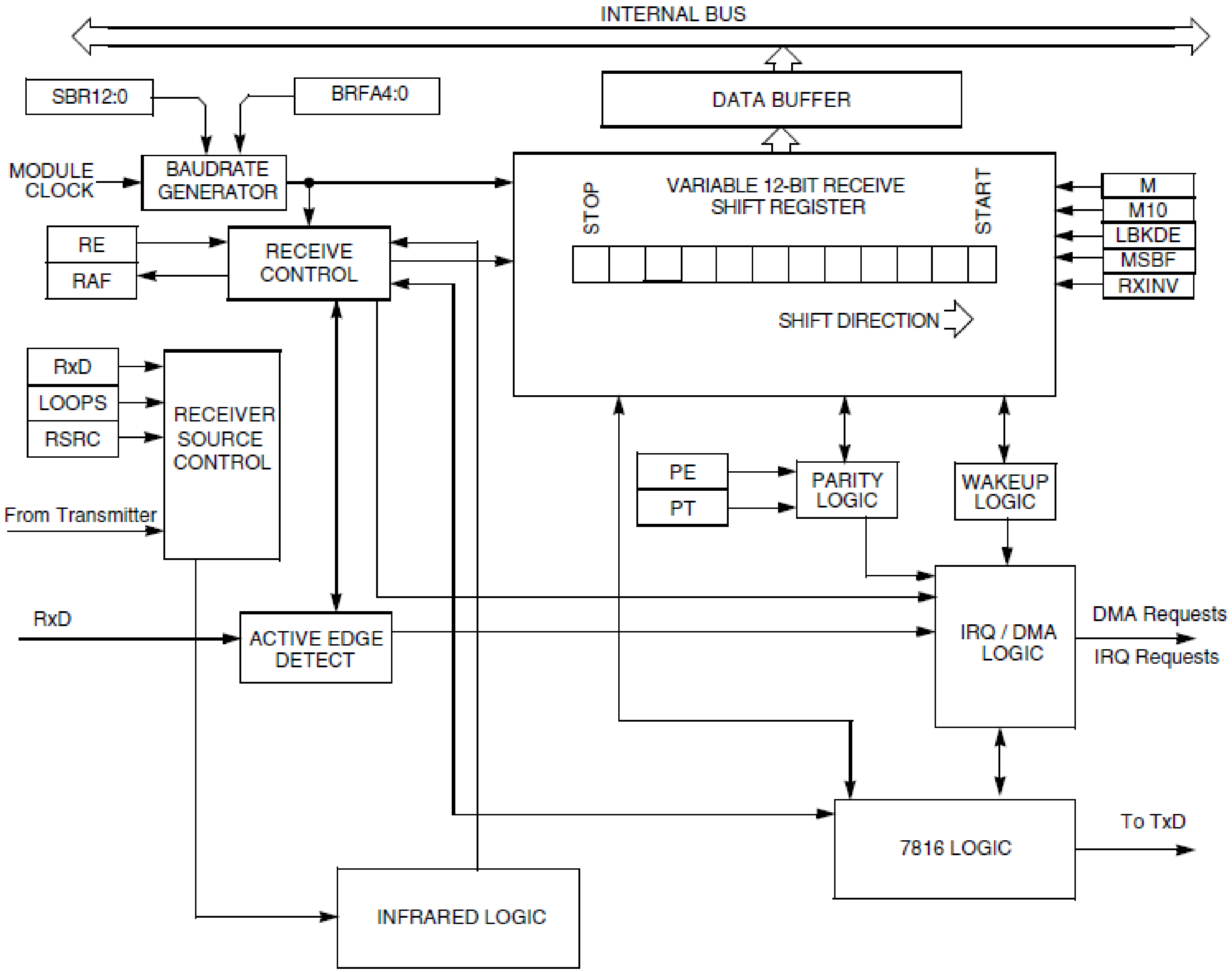
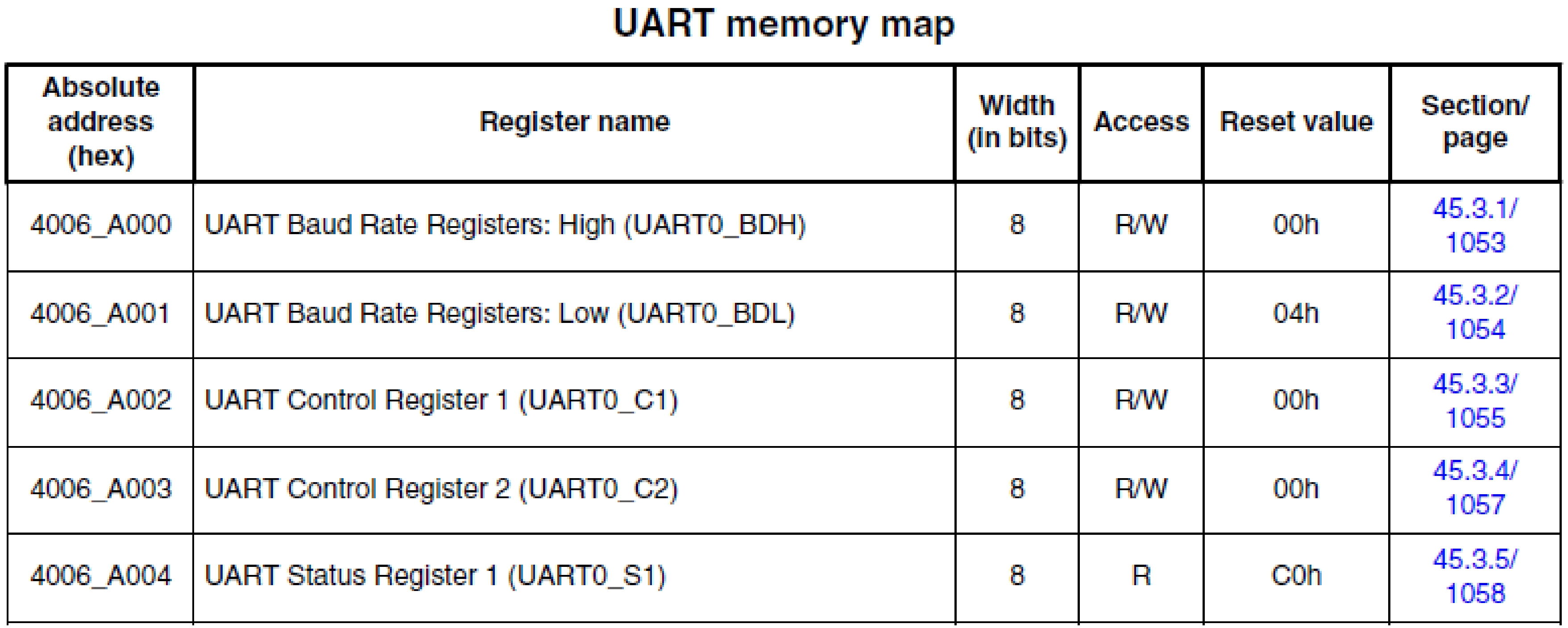


Figure 45-200. UART receiver block diagram

Configuring the UART: Manually

- You can configure the UART by setting registers as described in the Reference Manual



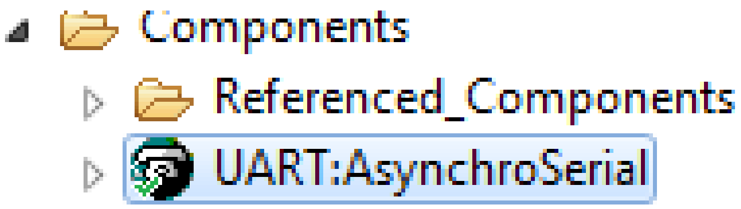
UART memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
4006_A000	UART Baud Rate Registers: High (UART0_BDH)	8	R/W	00h	45.3.1/ 1053
4006_A001	UART Baud Rate Registers: Low (UART0_BDL)	8	R/W	04h	45.3.2/ 1054
4006_A002	UART Control Register 1 (UART0_C1)	8	R/W	00h	45.3.3/ 1055
4006_A003	UART Control Register 2 (UART0_C2)	8	R/W	00h	45.3.4/ 1057
4006_A004	UART Status Register 1 (UART0_S1)	8	R	C0h	45.3.5/ 1058

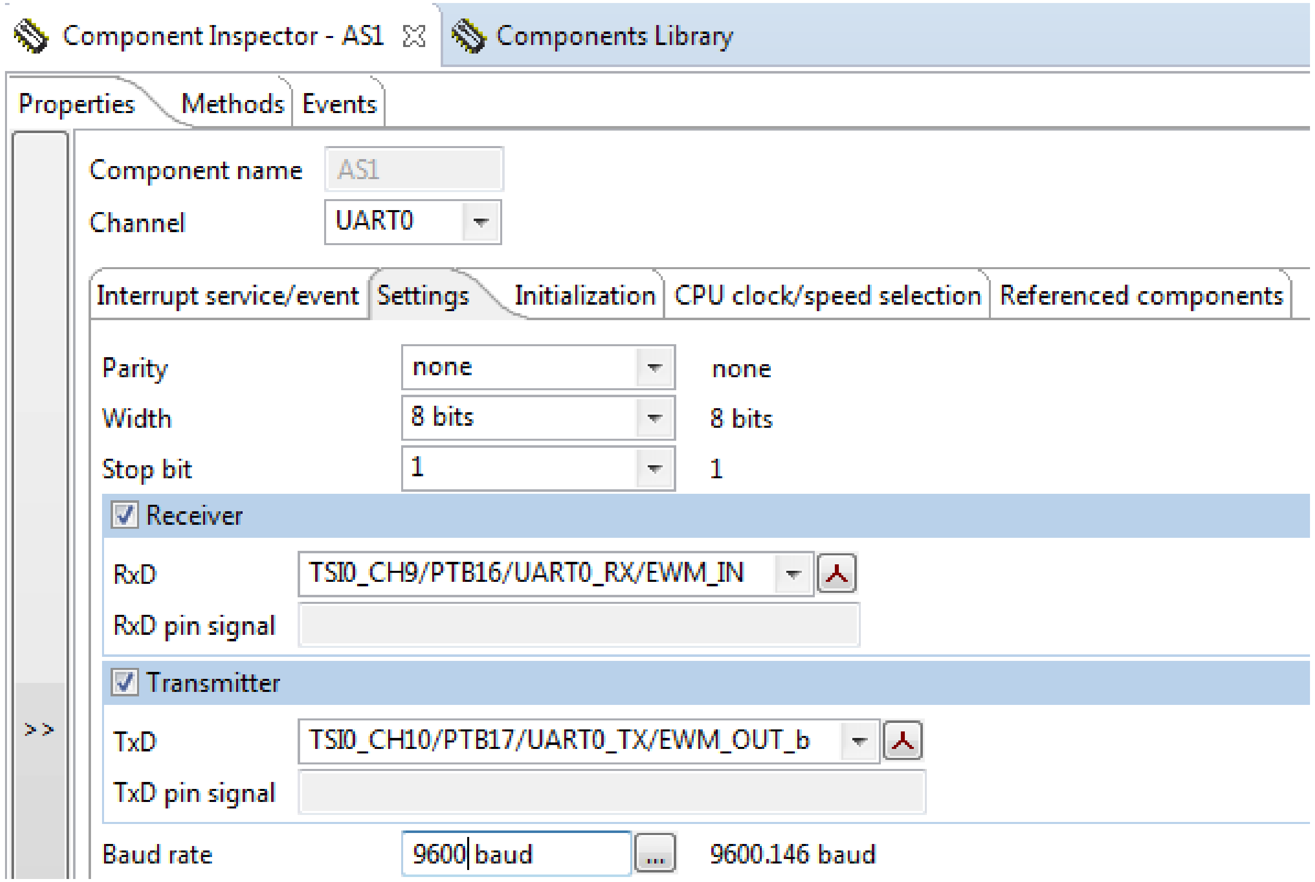
Configuring the UART: Processor Expert

- You can also use Processor Expert to set up the UART

- Recommended component: **AsynchroSerial**



Components
- Referenced_Components
- UART:AsynchroSerial



Component name: AS1

Channel: UART0

Parity	none	none
Width	8 bits	8 bits
Stop bit	1	1

☑ Receiver

RxD: TSI0_CH9/PTB16/UART0_RX/EWM_IN

RxD pin signal:

☑ Transmitter

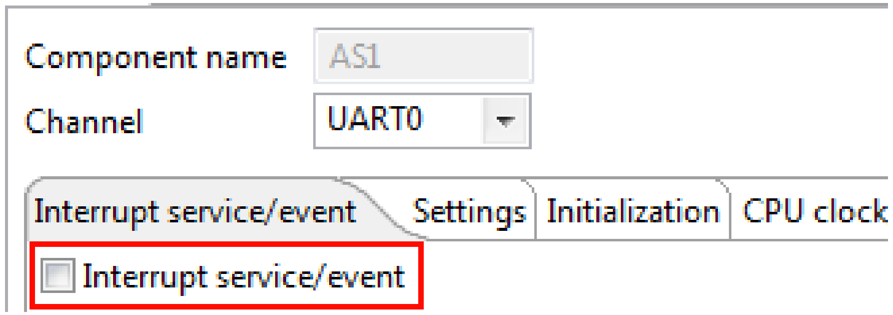
TxD: TSI0_CH10/PTB17/UART0_TX/EWM_OUT_b

TxD pin signal:

Baud rate: 9600 baud … 9600.146 baud

AsynchroSerial

- AsynchroSerial has two modes: **interrupts enabled** or **interrupts disabled.**
- Usage with interrupts is more complex but also more powerful.
- We’ll consider interrupts later in the subject.
- For now, disable interrupts.



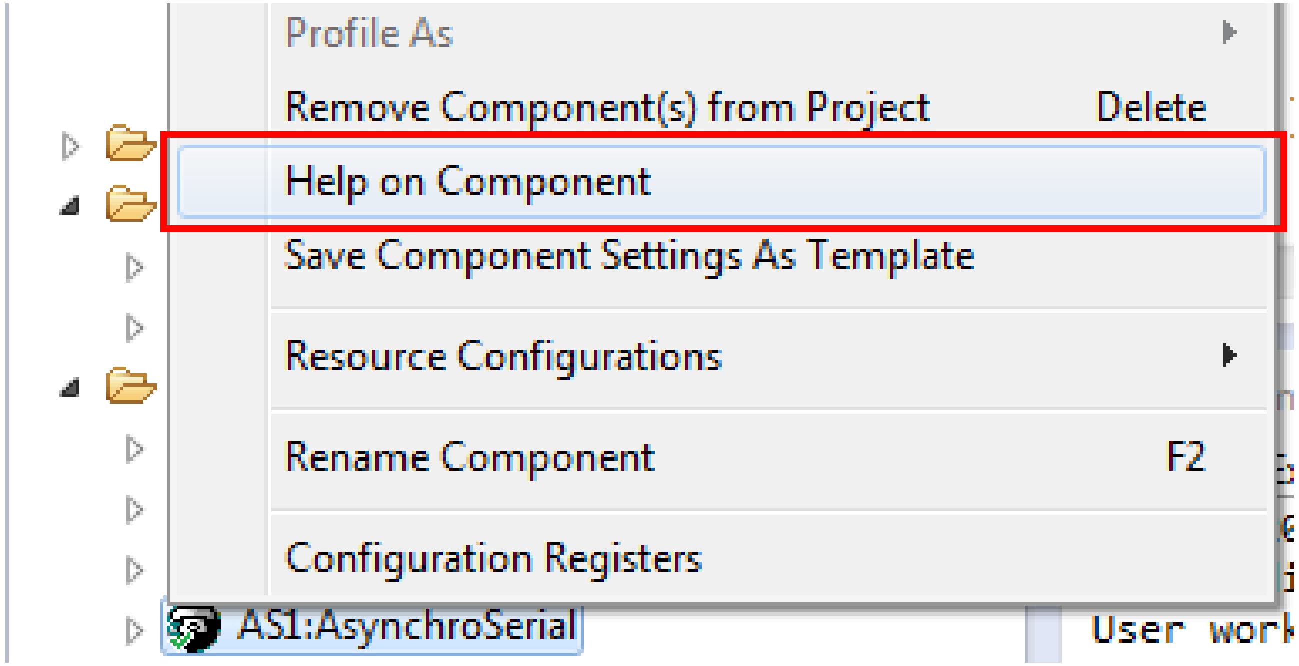
Component name AS1

Channel UART0

□ Interrupt service/event

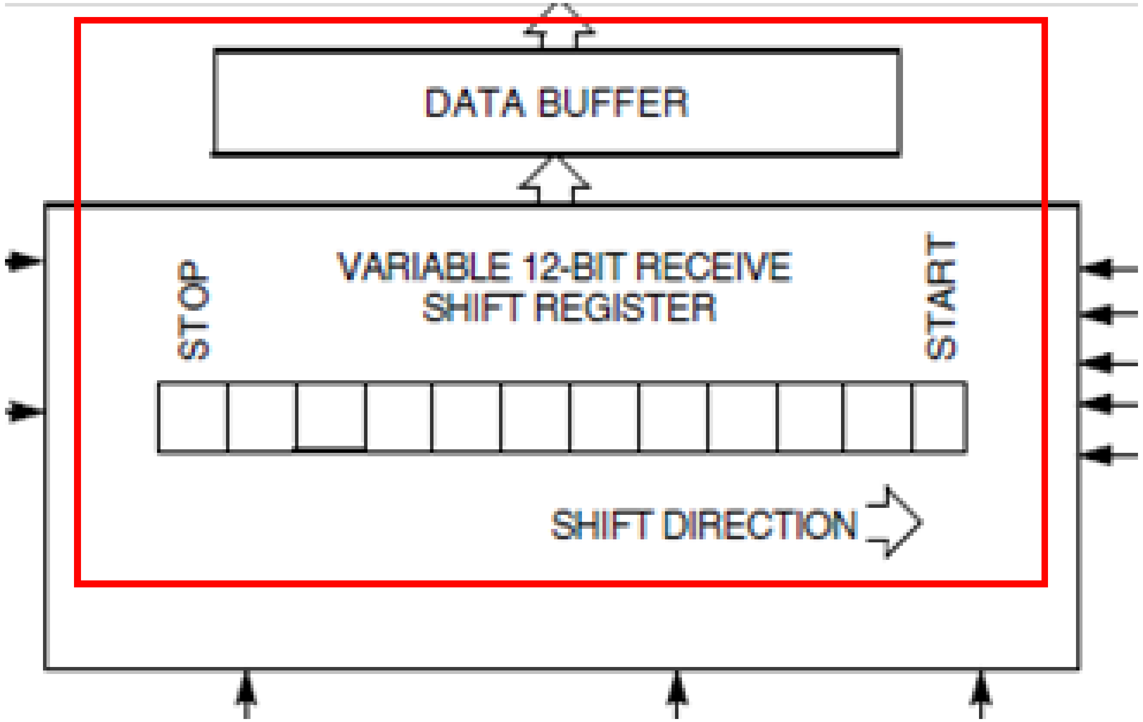
Read the documentation for AsynchroSerial

- Right click on the component to view its documentation.



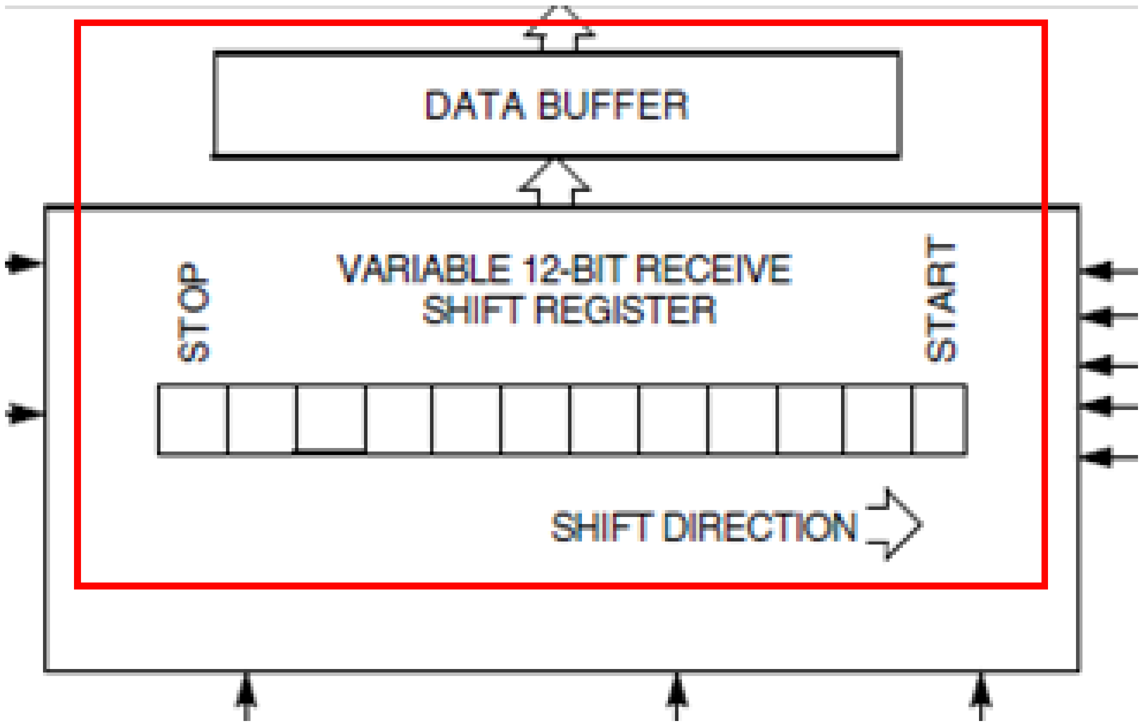
Receiving characters

- As each bit arrives it is placed into the receive shift register.
- Once the frame is ended, the complete byte is copied to the data buffer.
- The byte must be read from the data buffer before the next frame arrives.



Receiving characters

- Software needs to repeatedly “poll” the UART module to see if a byte has arrived.



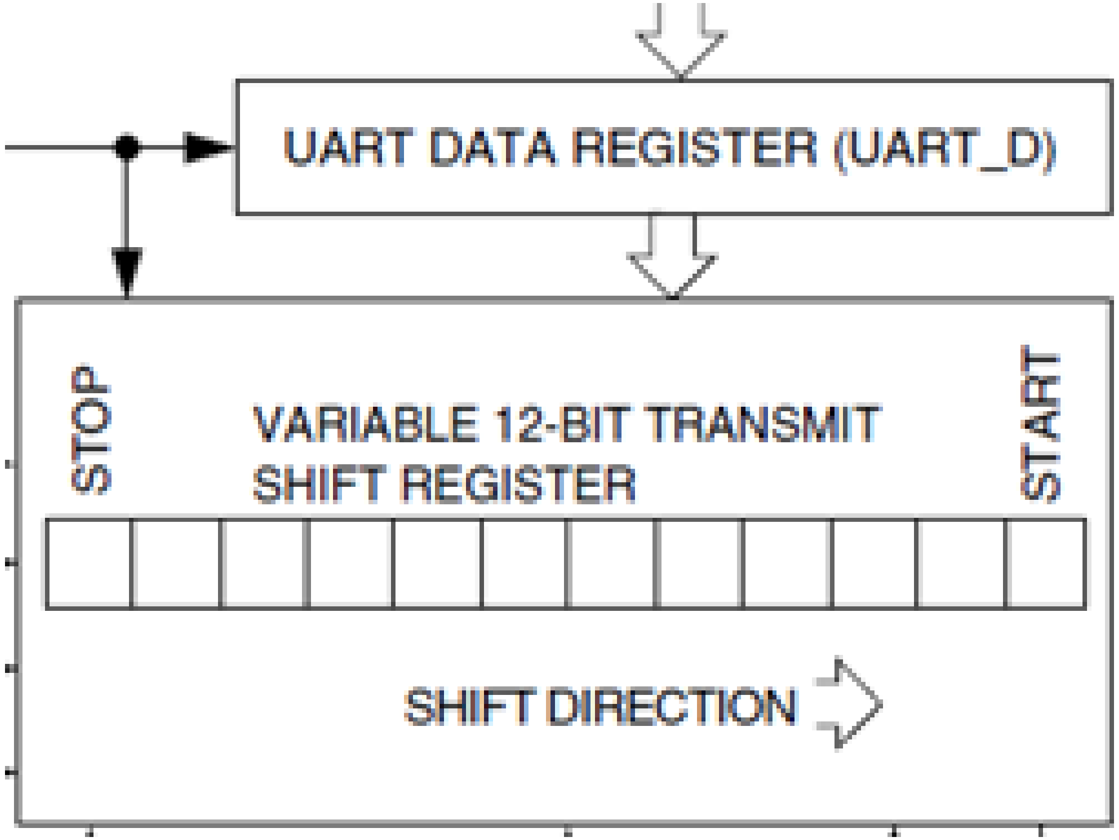
Receiving with AsynchroSerial

- AsynchroSerial generates a method “**RecvChar**” to receive a single character.
- It returns ERR_RXEMPTY if the receive data buffer is empty.
- It returns ERR_OK when a byte was received.

```
byte err;
char c;
do {
     err = AS1_RecvChar(&c); // component name = AS1
} while (err != ERR_OK);
// the received byte is in the variable c
```

Transmitting characters

- A byte cannot be loaded into the shift register until it's empty!
- Software must spin in a loop over and over until the last character was transmitted.


UART DATA REGISTER (UART_D)
STOP
VARIABLE 12-BIT TRANSMIT SHIFT REGISTER
START
SHIFT DIRECTION

Transmitting with AsynchroSerial

- AsynchroSerial generates a method “**SendChar**” to transmit a single character.
- It returns ERR_TXFULL if the transmit data buffer is full.
- It returns ERR_OK when the byte was successfully transmitted.

```
byte err;
char c = 'a'; // transmit the letter a
do {
    err = AS1_SendChar(c); // component name = AS1
} while (err != ERR_OK);
```

Example code: Transmitting a string

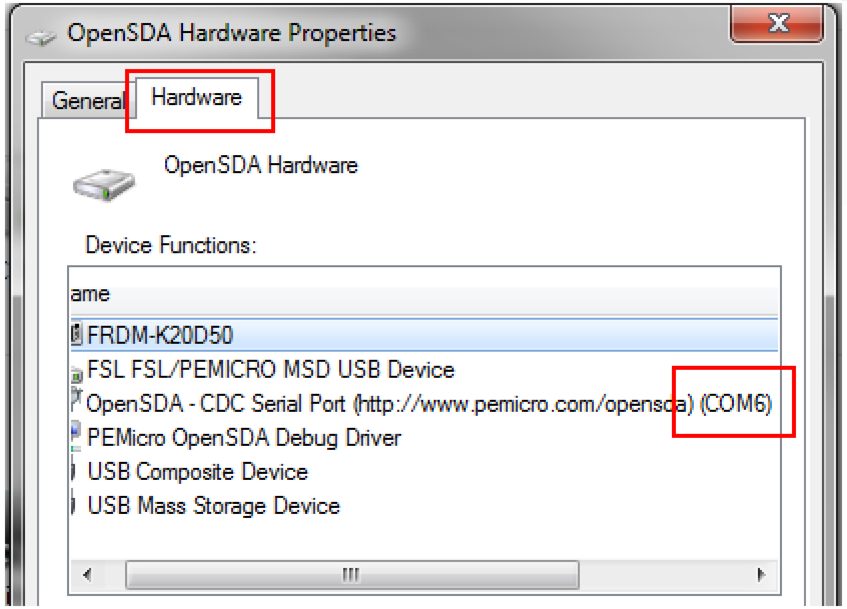
```
void send_string(const char *str)
{
      size_t len, i; // a size_t is an unsigned integer
      len = strlen(str); // returns the number of chars in str
      byte err;
      for (i = 0; i < len; i++) {
            // send this character
            do {
                  err = UART_SendChar(str[i]);
            } while (err != ERR_OK);
      }
}
```

Serial ports on the computer side

- Microsoft Windows calls serial ports “COM” ports.
- They are numbered, e.g. COM1, COM2, COM3, …
- Remember that OpenSDA emulates a serial port.
- Windows will assign the OpenSDA port to a particular number.

Identifying the OpenSDA serial port number

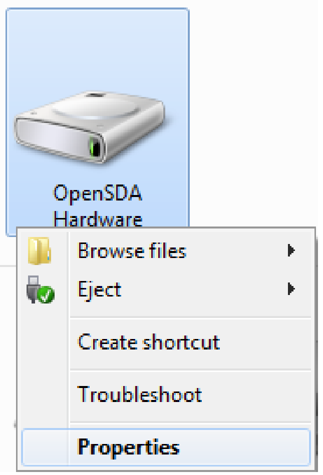
- Click the **Start** button -> **Devices and Printers**
- **Right click** on “OpenSDA Hardware” and choose **Properties**
- Click the **Hardware** tab
- Read off the COM port number



OpenSDA Hardware

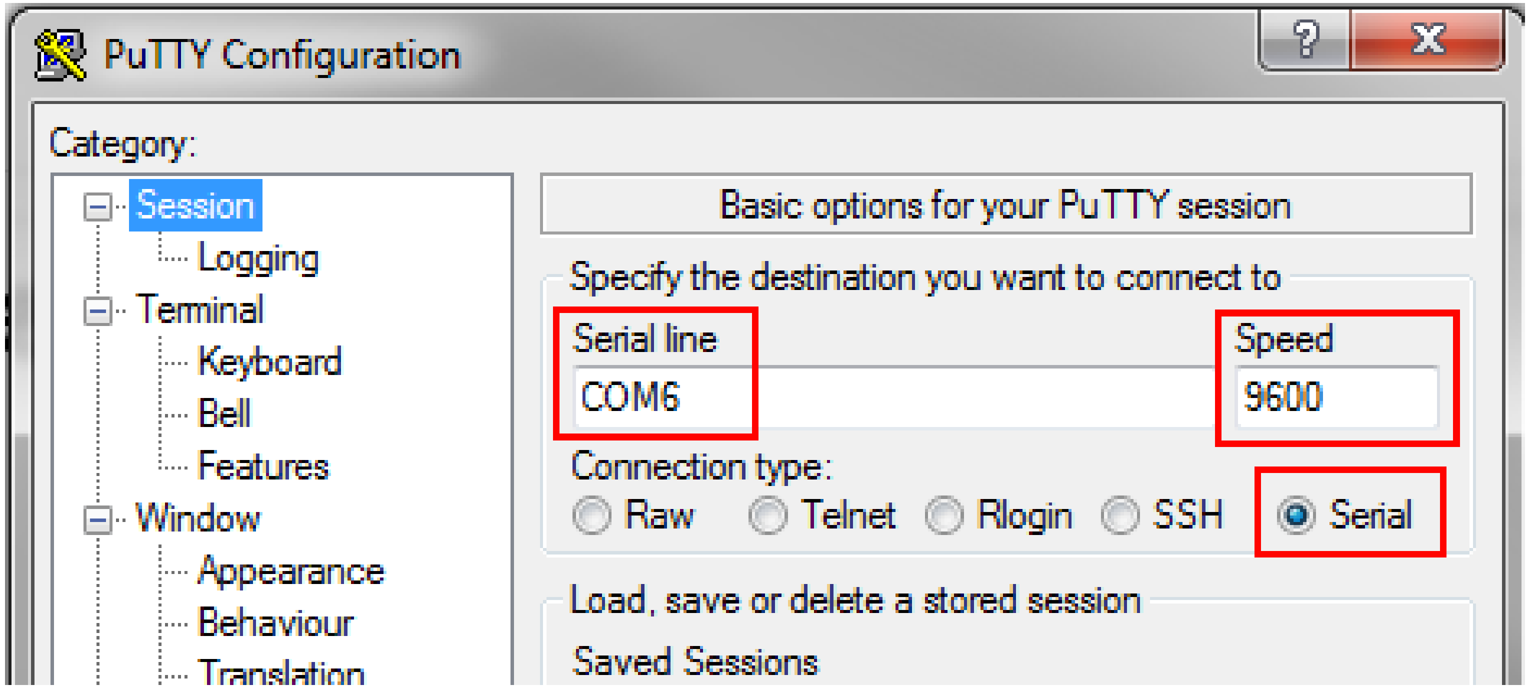
Device Functions:

ame
FRDM-K20D50
FSL FSL/PEMICRO MSD USB Device
OpenSDA - CDC Serial Port (http://www.pemicro.com/opensda) (COM6)
PEMicro OpenSDA Debug Driver
USB Composite Device
USB Mass Storage Device



Opening a serial port on the computer side

- Open a terminal emulator, e.g. **PuTTY**
- Choose the relevant COM port, the baud rate (aka speed).



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM6

Speed: 9600

Connection type:
○ Raw ○ Telnet ○ Rlogin ○ SSH ● Serial

Load, save or delete a stored session

Saved Sessions

Summary

- A UART component sends and receives logic-level signals for serial communications.
- Inter-board serial comms usually uses RS232 voltages (up to +/- 25 V).
- Software to send and receive must repeatedly poll the UART module because the UART only handles one byte at a time.

This week’s lab

- You’ll implement serial communications between your dev board and the computer