INTEGRATED TECHNICAL EDUCATION CLUSTER
AT ALAMEERIA

SPRING 2015

E-626-A
Real-Time Embedded Systems (RTES)

Lecture #6
UART & Time Analysis for RTES

Instructor:
Dr. Ahmad El-Banna

Agenda

- Need for Timing Analysis
- Software Behavior
- Hardware Timing
- UART Module

TIME ANALYSIS

Intro.

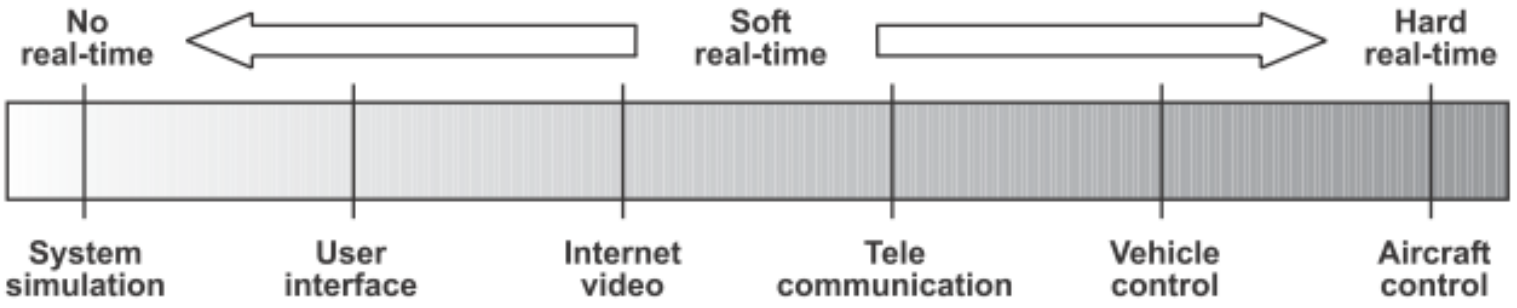
- A **real-time** system must react within precise **time constraints**, related to events in its environment and the system it controls.
- This means that the **correct behavior** of a real-time system depends not only on the **result** of the computation but **also** on the **time** at which the result is produced.
- The **size** and the **complexity** of the **software** in real-time system are **increasing**.
  - This makes it **hard**, or even impossible, to perform exhaustive **testing** of the **execution time**.
- The **hardware** used in real-time systems is also becoming more **complex**, including advanced computer **architecture** features such as **caches**, **pipelines**, **branch prediction**, and **out-of-order execution**.
  - These features **increase** the **speed** of execution on **average**,
  - **but** also makes the timing behavior much **harder to predict**, since the variation in execution time between fortuitous and worst cases increase.

Execution time analysis

- **Variations** in the execution time occur **due to** variations in
  - input data,
  - the characteristics of the software, the processor and
  - the computer system in which the program is executed.
- Execution Times:
  - The worst-case execution time (**WCET**) of a program is defined as the **longest** execution time that will ever be observed when the program is run on its target hardware.
  - The best-case execution time (**BCET**) is defined as the **shortest** time ever observed.
  - The average-case execution time (**ACET**) lies somewhere **in-between** the WCET and the BCET, and depends on the execution time distribution of the program.

Need for timing analysis

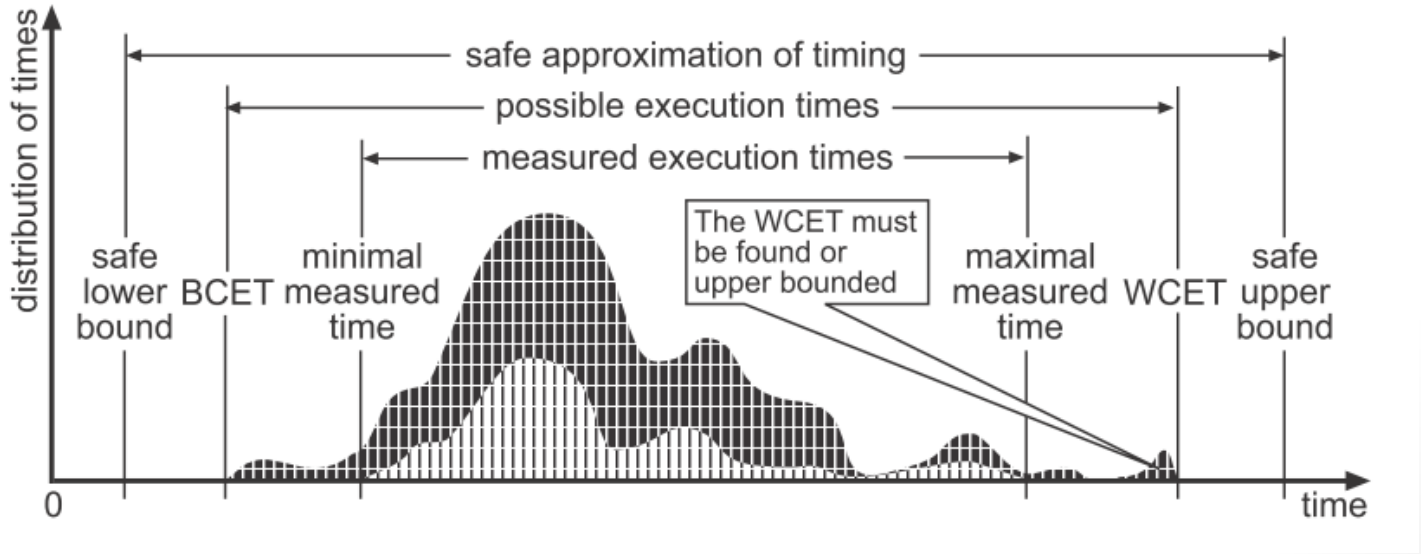
- Reliable timing estimates are important when designing and verifying many type of embedded systems and real-time systems.
- This is especially true, **when the system is used to control safe critical products** such as vehicles, aircraft, military equipment and industrial plants.



timing criticality

Example distribution of execution time

- A WCET analysis derives an estimate of the WCET for a program or part of the program.
- To guarantee that no deadline are missed, a WCET estimate must be safe (or conservative), i.e., a value greater than or equal to the WCET.



Software Behavior

- Embedded **software** comes in many different flavors, using many **different languages**.
- Most **timing-critical** real-time software is written in **C** or **Ada**, with some assembly language.
- The **software behavior contributes a large part of the execution time** variability of a program, often dominating the effect of local hardware timing variability.
- **even small codes** might **exhibit variable** and interesting **behavior**.

```
1.   // The main function
2.   void task N(void) {
3.        // Read values from sensors
4.        int val1 = SENSOR1;
5.        int val2 = SENSOR2;
6.        // To hold calculated values
7.        int res1 = 0;
8.        int res2 = 0;
9.        // Call twice with different values
10.       res1 = convert(val1);
11.       res2 = convert(val2);
12.       // Set actuator to calculated sum
13.       ACTUATOR = res1 + res2;
14.  }
```

```
15.  // Convert read value
16.  int convert(int val) {
17.       int i = 0;
18.       int j = 0;
19.       total = 0;
20.       while(i <= val) {
21.            if(j < 5)
22.                 j++;
23.            if(j > val) break;
24.            total = total + j - 2;
25.            i++;
26.       }
27.  return total;
28.  }
```

one branch takes longer than the other branch to execute !

Hardware Timing

- The **main complexity** in hardware timing analysis is the behavior of the **processor** itself, along with its **memory** system.
- **Other** components of a computer system like I/O, networks, sensors, and actuators have **less impact** on the program timing.
- Traditional **8-bit and 16-bit processors** typically feature **simple architectures** where **instructions** have **fixed execution times**, and each instruction has minimal effect on the timing of other instructions.
- Somewhat more **complex 32-bit** processors are designed for **cost-sensitive** applications.

Issues affect the time in Hardware

- Memory Access Times
  - e.g. RAM technology
- Long Timing Effects
  - Number of pipelining, one instruction waits the other to finish
- Caches
  - Cache Structures/Levels
- Branch Prediction
  - Predict which branch will be taken before being resolved in the processor pipelining
- Multicore and Multiprocessor Systems
  - can both benefit and hinder timing analysis!
  - Interface between tasks and shared resources or memory
- Custom Accelerator Hardware
  - ASIC, SoC or FPGA

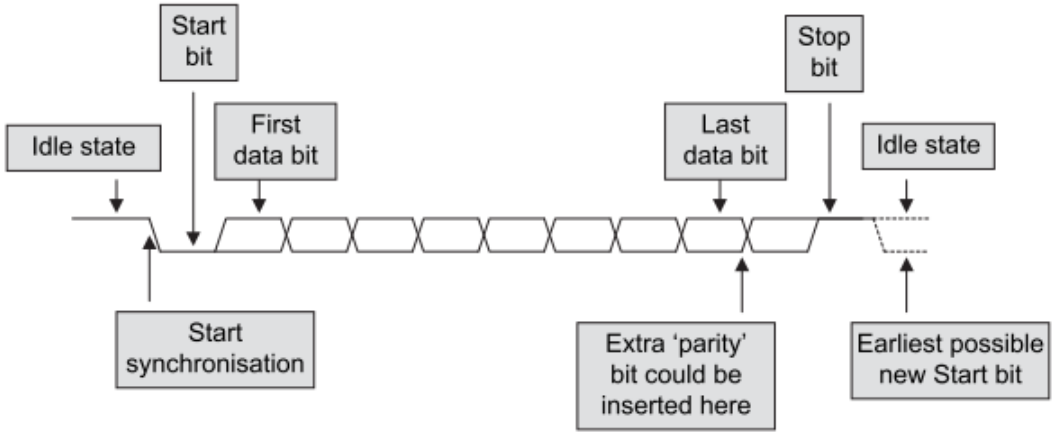
UART

Serial Communication

- Asynchronous
  - No need for clock
  - Low speed
  - Covered in this lecture
- Synchronous
  - Need a clock
  - High speed
  - Covered later

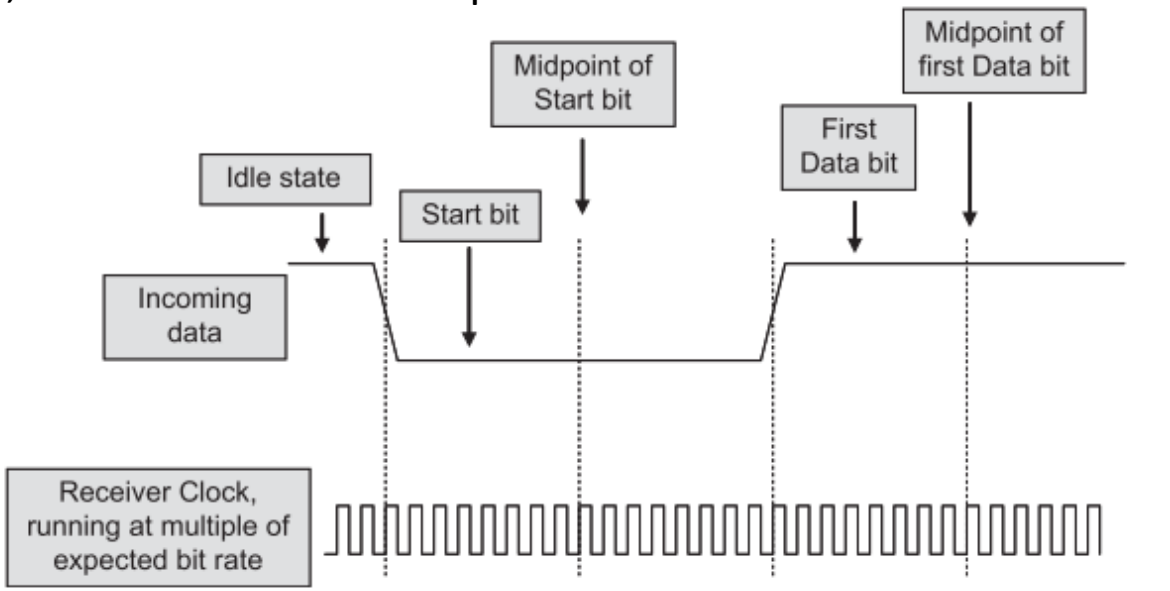
Asynchronous principles

- Solves the disadvantages of the synchronous comm. which are:
  - An extra line is needed to go to every data node.
  - The bandwidth needed for the clock is always twice the bandwidth needed for the data; therefore, it is demands of the clock which limit the overall data rate.
  - Over long distances, clock and data themselves could lose synchronisation.



Synchronizing serial data – without an incoming clock

- The receiver runs an internal clock whose frequency is an exact multiple of the expected bit rate.
- The receiver monitors the state of the incoming data on the serial receive line.
- When a Start bit is detected, a counter begins to count clock cycles e.g. 16 cycles until the midpoint of the anticipated Start bit is reached.
- The clock counter counts a further 16 cycles, to the middle of the first data bit, and so on until the Stop bit.



The 16F87XA Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART)

- The USART can be **configured** as synchronous **master**, synchronous **slave** or in **asynchronous** mode.
- In the **asynchronous** mode, it is **full duplex** – that is, it can transmit and receive at the same time.
- Thus, it has both a **receive** shift **register** and a **transmit** shift **register**, which can operate **simultaneously**.
- Both sections **share** the same **baud rate generator** and have the same **data format**.
- **Operation** of the USART is controlled by **two registers**,
  - **TXSTA** and
  - **RCSTA**
- The port is enabled by the **SPEN** bit of **RCSTA**, and selection of synchronous or asynchronous modes is by the **SYNC** bit of the **TXSTA** register.

The transmit status and control register, TXSTA (address 98 H)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care.

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit

1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit

1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit

1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit

1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

The RCSTA register (address 18H), receive status and control register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

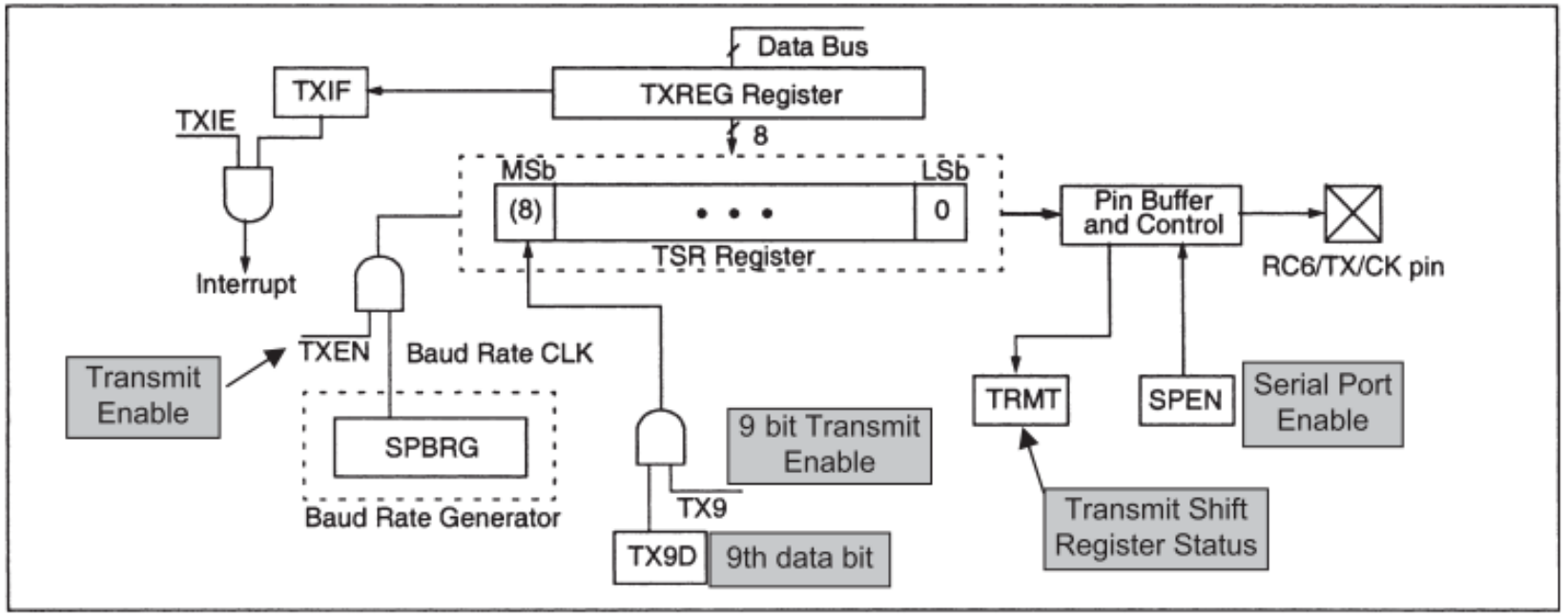
bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

The USART baud rate generator



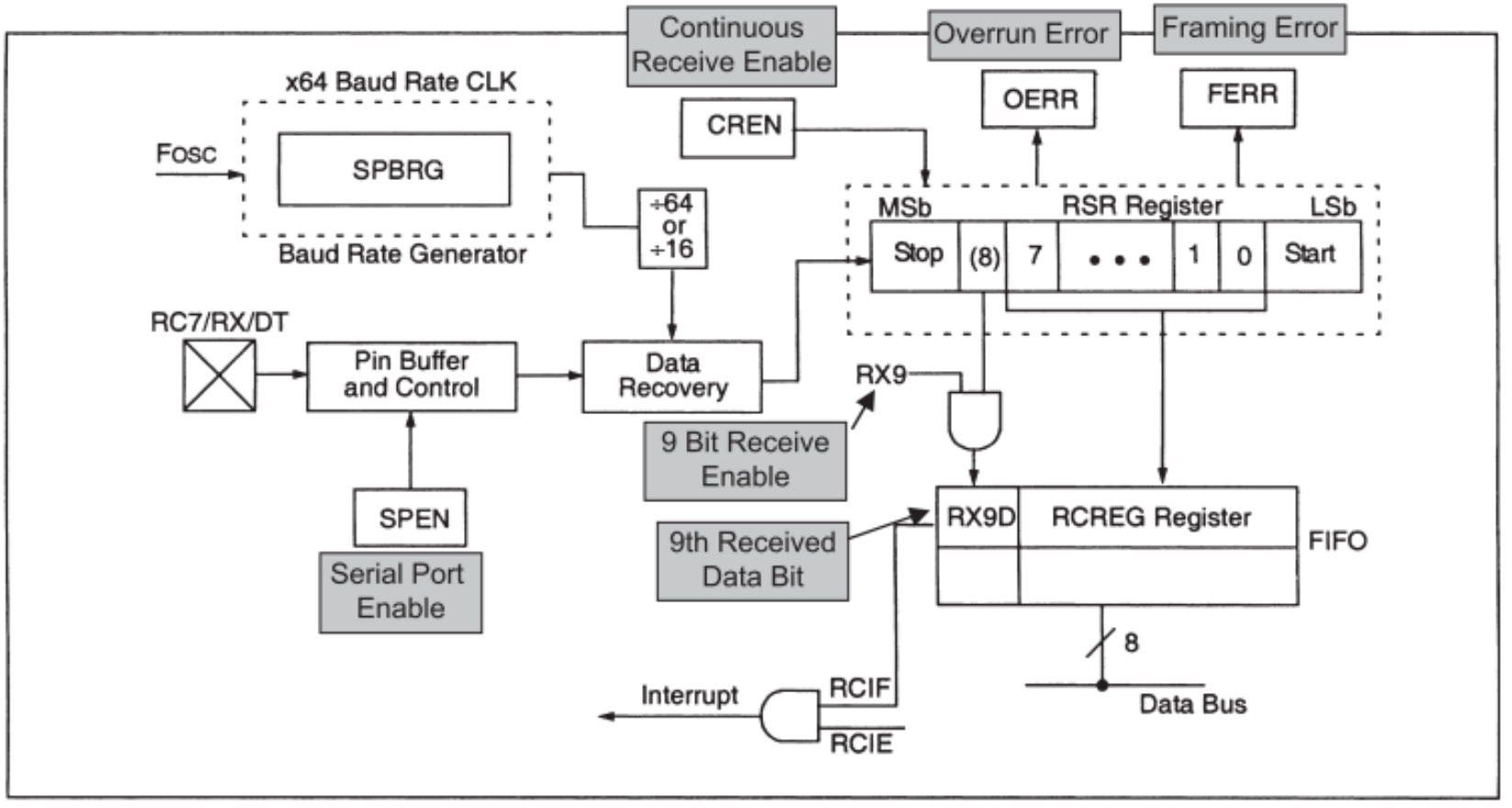
For **BRGH** = 0 Baud rate = $\frac{f_{osc}}{64([\mathbf{SPBRG}]+1)}$

For **BRGH** = 1 Baud rate = $\frac{f_{osc}}{16([\mathbf{SPBRG}]+1)}$

Synchronous

BRGH = *don't care* Baud rate = $\frac{f_{osc}}{4([\mathbf{SPBRG}]+1)}$

The USART asynchronous receiver



Sample Project

- Send the word " Good Morning ! " to the PC
- Receive its reply by flashing a green LED if "OK" is received
- Otherwise, flash a red LED.

- For more details, refer to:
  - Chapter 10, T. Wilmishurst, **Designing Embedded Systems with PIC Microcontrollers**, 2010.
  - A. Ermedahl, **Execution Time Analysis for Embedded Real-Time Systems**.
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12134
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg