UART

RUDRA PRATAP SUMAN
UART: Universal Asynchronous Receiver Transmitter

- UART is a simple half-duplex, asynchronous, serial protocol.
- Simple communication between two equivalent nodes.
- Any node can initiate communication.
- Since connection is half-duplex, the two lanes of communication are completely independent.
UART: Universal Asynchronous Receiver Transmitter

- What makes it ‘universal’?
  - Its parameters (format, speed ..) are configurable.
- Why ‘asynchronous’?
  - It doesn’t have a clock
UART Basics

- **Baud Rate:**
  - No. of bits transmitted/received per second = ______bits/sec.

- **Format of Communication**

---

Figure 17: Basic UART packet format: 1 start bit, 8 data bits, 1 parity bit, 1 stop bit.
UART Basics

Connections for UART
UART Basics

Connections for UART
UART Basics

Connections for UART

Device 1

Rx
Tx
Gnd

Device 2

Tx
Rx
Gnd
UART Basics

Connections for UART

- Device 1: Rx, Tx, Gnd
- Device 2: Tx, Rx, Gnd
UART Characteristics

- The speed of communication (measured in bauds) is predetermined on both ends.
- A general rule of thumb is to use 9600 bauds for wired communication.
- UART implements error-detection in the form of parity bit.
• Parity bit is HIGH when number of 1’s in the Data is odd.

• Respectively, it is LOW when number of 1’s in the Data is even.
UART in AtMega16

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(XCK/T0)</td>
<td>PB0</td>
</tr>
<tr>
<td>(T1)</td>
<td>PB1</td>
</tr>
<tr>
<td>(INT2/AIN0)</td>
<td>PB2</td>
</tr>
<tr>
<td>(OC0/AIN1)</td>
<td>PB3</td>
</tr>
<tr>
<td>(SS)</td>
<td>PB4</td>
</tr>
<tr>
<td>(MOSI)</td>
<td>PB5</td>
</tr>
<tr>
<td>(MISO)</td>
<td>PB6</td>
</tr>
<tr>
<td>(SCK)</td>
<td>PB7</td>
</tr>
<tr>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td></td>
</tr>
<tr>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>XTAL2</td>
<td></td>
</tr>
<tr>
<td>XTAL1</td>
<td></td>
</tr>
<tr>
<td>(RXD)</td>
<td>PD0</td>
</tr>
<tr>
<td>(TXD)</td>
<td>PD1</td>
</tr>
<tr>
<td>(INT0)</td>
<td>PD2</td>
</tr>
<tr>
<td>(INT1)</td>
<td>PD3</td>
</tr>
<tr>
<td>(OC1B)</td>
<td>PD4</td>
</tr>
<tr>
<td>(OC1A)</td>
<td>PD5</td>
</tr>
<tr>
<td>(ICP1)</td>
<td>PD6</td>
</tr>
<tr>
<td>PA0 (ADC0)</td>
<td>40</td>
</tr>
<tr>
<td>PA1 (ADC1)</td>
<td>39</td>
</tr>
<tr>
<td>PA2 (ADC2)</td>
<td>38</td>
</tr>
<tr>
<td>PA3 (ADC3)</td>
<td>37</td>
</tr>
<tr>
<td>PA4 (ADC4)</td>
<td>36</td>
</tr>
<tr>
<td>PA5 (ADC5)</td>
<td>35</td>
</tr>
<tr>
<td>PA6 (ADC6)</td>
<td>34</td>
</tr>
<tr>
<td>PA7 (ADC7)</td>
<td>33</td>
</tr>
<tr>
<td>AREF</td>
<td>32</td>
</tr>
<tr>
<td>GND</td>
<td>31</td>
</tr>
<tr>
<td>AVCC</td>
<td>30</td>
</tr>
<tr>
<td>PC7 (TOSC2)</td>
<td>29</td>
</tr>
<tr>
<td>PC6 (TOSC1)</td>
<td>28</td>
</tr>
<tr>
<td>PC5 (TDI)</td>
<td>27</td>
</tr>
<tr>
<td>PC4 (TDO)</td>
<td>26</td>
</tr>
<tr>
<td>PC3 (TMS)</td>
<td>25</td>
</tr>
<tr>
<td>PC2 (TCK)</td>
<td>24</td>
</tr>
<tr>
<td>PC1 (SDA)</td>
<td>23</td>
</tr>
<tr>
<td>PC0 (SCL)</td>
<td>22</td>
</tr>
<tr>
<td>PD7 (OC2)</td>
<td>21</td>
</tr>
</tbody>
</table>
Connecting AtMega16’s with UART

Device 1

Device 2
MAX-232 and USB-Serial

RS-232 Level Converter Circuit
UART Communication

Tx  Rx
Connecting AtMega16 with Computer

Latest Direct Way:
Coding with UART

- Three simple commands:
  - `putchar(char);`
  - Sends 8-bit characters through UART
  - `getchar();`
  - Receives 8-bit characters via UART
  - `puts(string);`
  - Sends a constant string
Where do we code..?
Where do we code.. ?
Where do we code..?
Sample Code for UART

Input MCU
// a is a char variable
a = inputFromUser();
putchar(a); // Data transmitted, now print

LCD MCU
a = getchar();
// Program will wait for data
printChar(a);
Coding for Arduino
• Serial.begin(speed)
  ○ Sets the data rate in bits per second (baud) for serial data transmission.
Coding for Arduino

- **Serial.begin(speed)**
  - Sets the data rate in bits per second (baud) for serial data transmission.

- **Serial.end()**
  - Disables serial communication, allowing the RX and TX pins to be used for general input and output.
  - To re-enable serial communication, call `Serial.begin()`.
Coding for Arduino

- **Serial.begin(speed)**
  - Sets the data rate in bits per second (baud) for serial data transmission.

- **Serial.end()**
  - Disables serial communication, allowing the RX and TX pins to be used for general input and output.
  - To re-enable serial communication, call `Serial.begin()`.

- **Serial.read()**
  - Reads incoming serial data
Coding for Arduino

- **Serial.begin(speed)**
  - Sets the data rate in bits per second (baud) for serial data transmission.

- **Serial.end()**
  - Disables serial communication, allowing the RX and TX pins to be used for general input and output.
  - To re-enable serial communication, call `Serial.begin()`.

- **Serial.read()**
  - Reads incoming serial data

- **Serial.println(val)**
  - Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n')
  - `Serial.println(val, format)` provides more control over how the string is formatted.
• Serial.print(val)
  Serial.print(val, format)
  ○ Prints data to the serial port as human-readable ASCII text.
Coding for Arduino

- Serial.print(val)
  Serial.print(val, format)
  - Prints data to the serial port as human-readable ASCII text.

- Serial.flush()
  - Waits for the transmission of outgoing serial data to complete.
    (Prior to Arduino 1.0, this instead removed any buffered incoming serial data.)
Coding for Arduino

- **Serial.print(val)**
  - **Serial.print(val, format)**
    - Prints data to the serial port as human-readable ASCII text.

- **Serial.flush()**
  - Waits for the transmission of outgoing serial data to complete. (Prior to Arduino 1.0, this instead removed any buffered incoming serial data.)

- **Serial.available()**
  - Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes).
**Sample Code for Arduino**

- `int incomingByte = 0; // for incoming serial data`

```c
void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```
Coding in DevCPP

link for downloading DevC++

Coding in DevCPP
Coding in DevCPP
Coding in DevCPP
Coding in DevCPP
Coding in DevCPP
Coding in DevCPP

```cpp
#include <conio.h>
#include <iostream>

int main()
{
    Tserial *com;
    char ch;
    com = new Tserial();
    com->connect("COM3", 4800, spNONE);
    com->sendChar('Y');
    com->disconnect();
}
```
Coding in DevCPP

- #ifdef __BORLANDC__
- #pragma hdrstop    // borland specific
- #include <condefs.h>
- #pragma argsused
- USEUNIT("Tserial.cpp");
- #endif
- #include "conio.h"
- #include "Tserial.cpp"

- int main(){
  Tserial *com;
  com = new Tserial();
  com->connect("COM3", 4800, spNONE);
  com->sendChar('F');
  com->disconnect();
  }

\textbf{Coding in DevCPP} \\

- \texttt{#ifdef \_\_BORLANDC\_\_}
- \texttt{#pragma hdrstop // borland specific}
- \texttt{#include <condefs.h>}
- \texttt{#pragma argsused}
- \texttt{USEUNIT("Tserial.cpp");}
- \texttt{#endif}
- \texttt{#include "conio.h"}
- \texttt{#include "Tserial.cpp"}

For Including “Tserial.cpp” library.

place “Tserial.Cpp” with your code just place it in same folder where your code is presented.

\begin{itemize}
\item \texttt{int main()}
\item \texttt{Tserial *com;}
\item \texttt{com = new Tserial();}
\item \texttt{com->connect("COM3", 4800, spNONE);}
\item \texttt{com->sendChar('F');}
\item \texttt{com->disconnect();}
\end{itemize}
Coding in DevCPP

- `#ifdef __BORLANDC__`
- `#pragma hdrstop`    // borland specific
- `#include <condefs.h>`
- `#pragma argsused`
- `USEUNIT("Tserial.cpp");`
- `#endif`
- `#include "conio.h"
- `#include "Tserial.cpp"

- `int main()
- `Tserial *com;
- `com = new Tserial();
- `com->connect("COM3", 4800, spNONE);
- `com->sendChar('F');
- `com->disconnect();`
Coding in DevCPP

- #ifdef __BORLANDC__
- #pragma hdrstop    // borland specific
- #include <condefs.h>
- #pragma argsused
- USEUNIT("Tserial.cpp");
- #endif
- #include "conio.h"
- #include "Tserial.cpp"

- int main(){
  Tserial *com;
  com = new Tserial();
  com->connect("COM3", 4800, spNONE);
  com->sendChar('F');
  com->disconnect();
  }

Object Creation
Coding in DevCPP

- #ifdef __BORLANDC__
- #pragma hdrstop  // borland specific
- #include <condefs.h>
- #pragma argsused
- USEUNIT("Tserial.cpp");
- #endif
- #include "conio.h"
- #include "Tserial.cpp"

- int main(){
-    Tserial *com;
-    com = new Tserial();
-    com->connect("COM3", 4800, spNONE);
-    com->sendChar('F');
-    com->disconnect();
- }

Connecting to a serial port
#ifdef __BORLANDC__
#pragma hdrstop // borland specific
#include <condefs.h>
#pragma argsused
USEUNIT("Tserial.cpp");
#endif
#include "conio.h"
#include "Tserial.cpp"

int main(){
  Tserial *com;
  com = new Tserial();
  com->connect("COM3", 4800, spNONE);
  com->sendChar('F');
  com->disconnect();
}

Send Character on Com port
Coding in DevCPP

- #ifdef __BORLANDC__
- #pragma hdrstop    // borland specific
- #include <condefs.h>
- #pragma argsused
- USEUNIT("Tserial.cpp");
- #endif
- #include "conio.h"
- #include "Tserial.cpp"

- int main(){
  Tserial *com;
  com = new Tserial();
  com->connect("COM3", 4800, spNONE);
  com->sendChar('F');
  com->disconnect();
  }

  Don’t forget to disconnect Com port
Coding in DevCPP

- `#ifdef __BORLANDC__`
- `#pragma hdrstop`  // borland specific
- `#include <condefs.h>`
- `#pragma argsused`
- `USEUNIT("Tserial.cpp");`
- `#endif`
- `#include "conio.h"`
- `#include "Tserial.cpp"`

```
int main()
{
    Tserial *com;
    com = new Tserial();
    com->connect("COM3", 4800, spNONE);
    com->sendChar('F');
    com->disconnect();
}
```
Coding in DevCPP

For More Details

Opening Com Port

Python
Matlab
JAVA
C Lang
Thank You Question??