Electronics
Serial Communication-UART

Terry Sturtevant

Wilfrid Laurier University

November 7, 2018

Serial Communication -UART

Serial Communication -UART

- Universal Asynchronous Receiver Transmitter

Serial Communication -UART

- Universal Asynchronous Receiver Transmitter
- Simplest form of serial communication

Serial Communication -UART

- Universal Asynchronous Receiver Transmitter
- Simplest form of serial communication
- Between 2 devices

Serial Communication -UART

- Universal Asynchronous Receiver Transmitter
- Simplest form of serial communication
- Between 2 devices
- Uses 2 signals (and Ground), Rx and Tx

Serial Communication -UART

- Universal Asynchronous Receiver Transmitter
- Simplest form of serial communication
- Between 2 devices
- Uses 2 signals (and Ground), Rx and Tx
- Asynchronous, so both must agree on baud rate

Communication parameters

Communication parameters

- 1 Start bit at “0” level

Communication parameters

- 1 Start bit at “0” level
- LSB transmitted first

Communication parameters

- 1 Start bit at “0” level
- LSB transmitted first
- Can have odd, even, or no parity bit

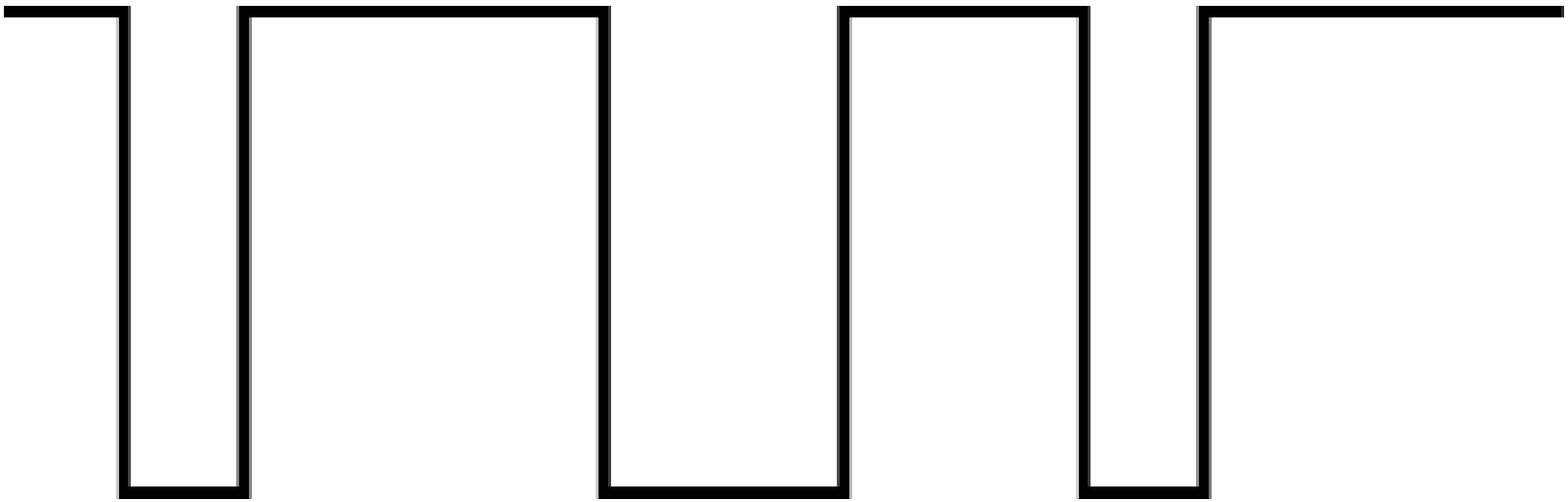
Communication parameters

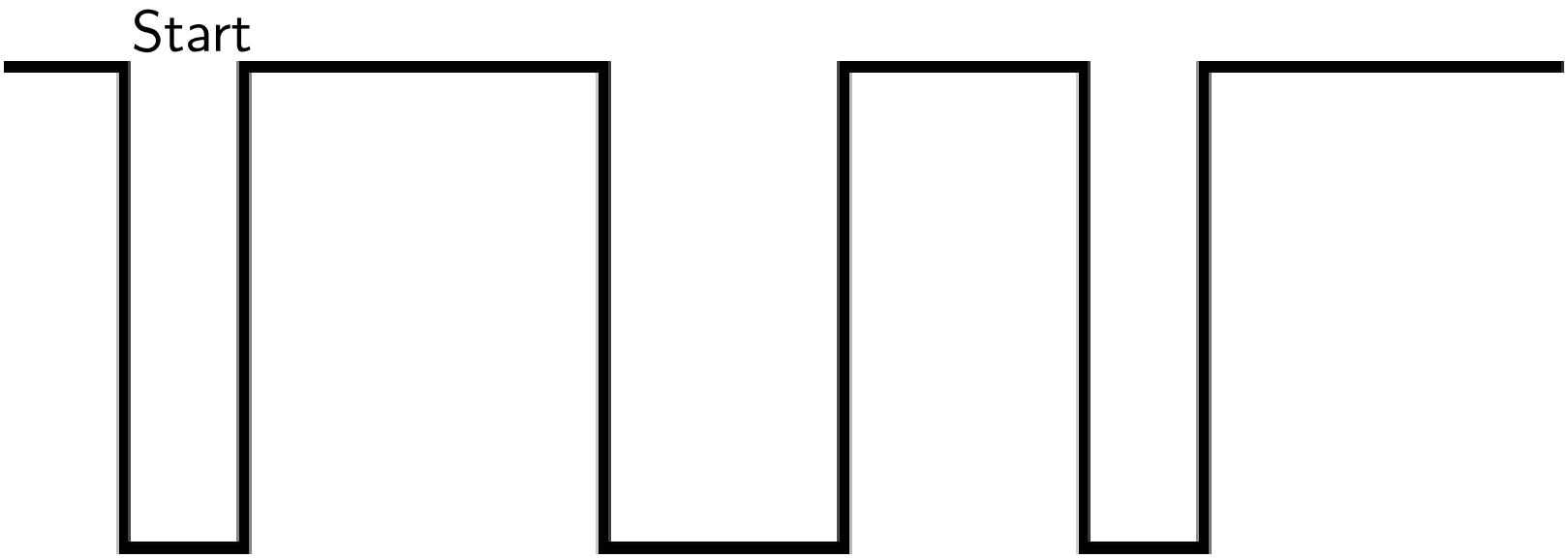
- 1 Start bit at “0” level
- LSB transmitted first
- Can have odd, even, or no parity bit
- 1 or 2 Stop bits at “1” level

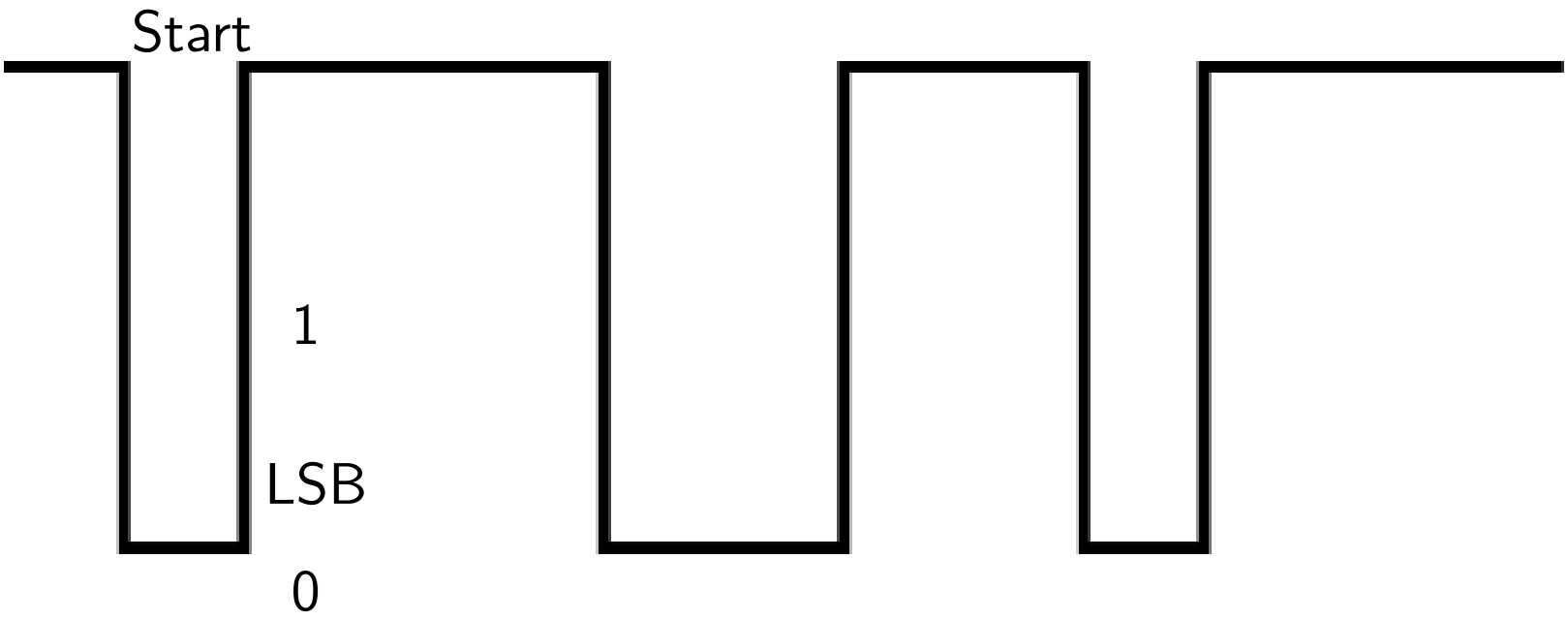
Communication parameters

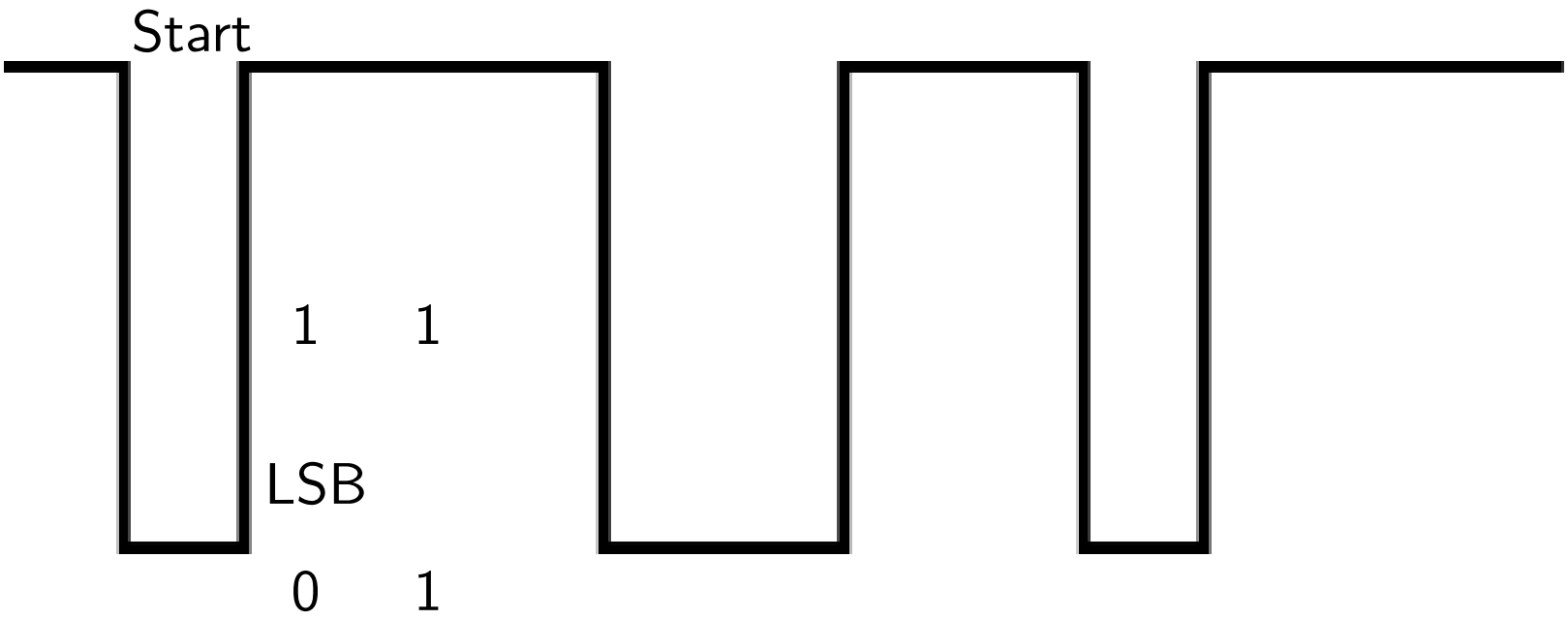
- 1 Start bit at “0” level
- LSB transmitted first
- Can have odd, even, or no parity bit
- 1 or 2 Stop bits at “1” level

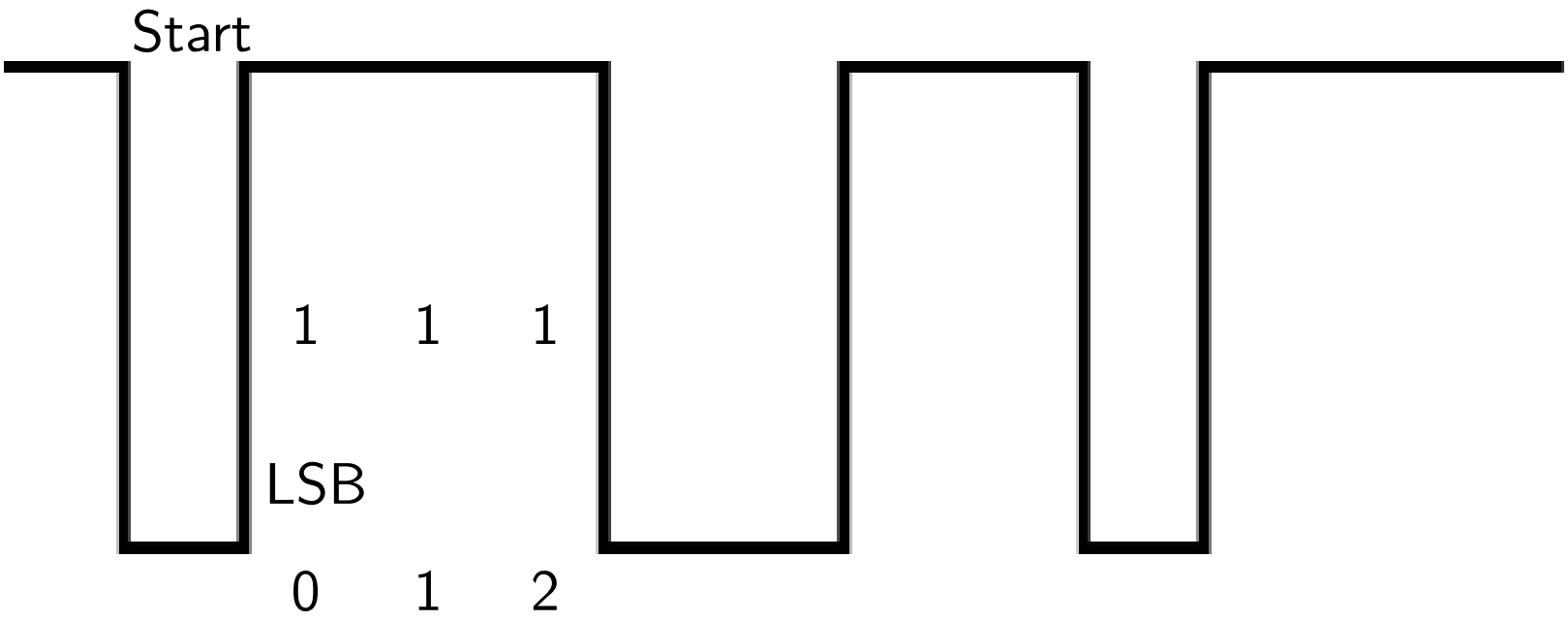
Since start and stop bits are opposite, new characters can always be detected.

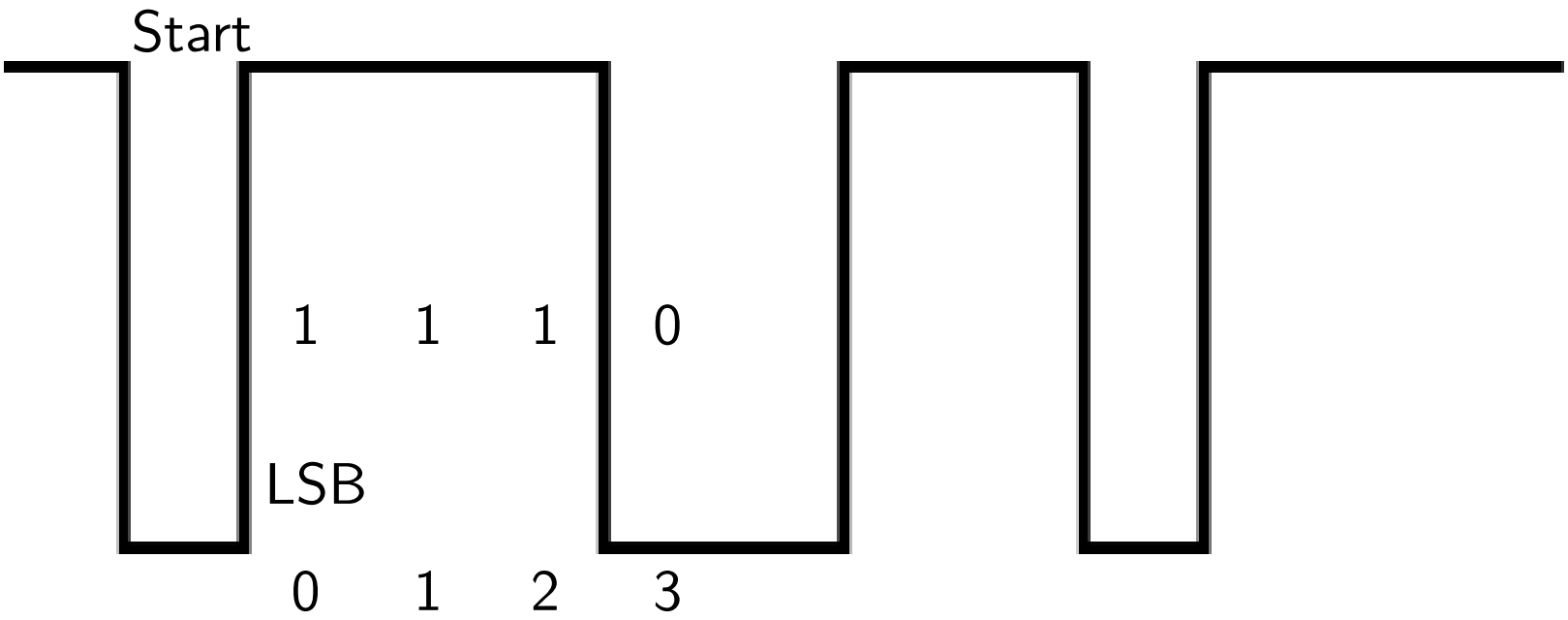


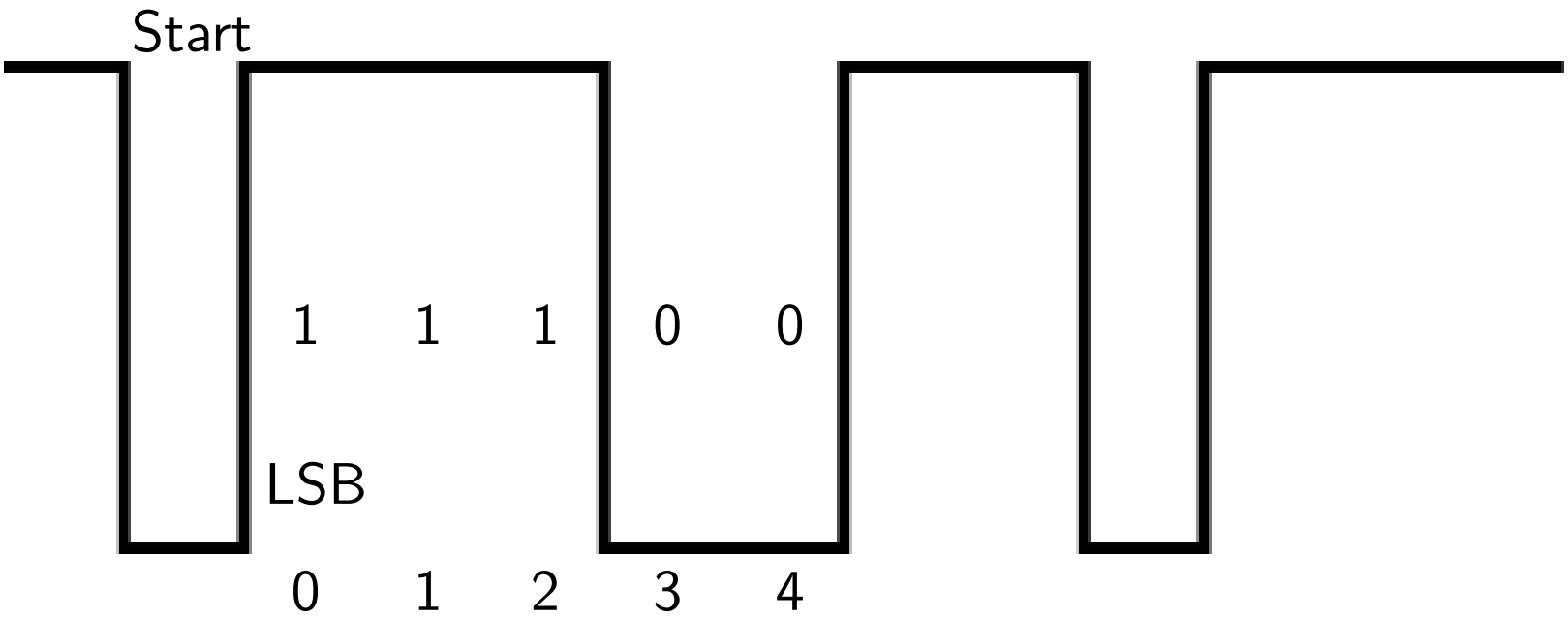

Start

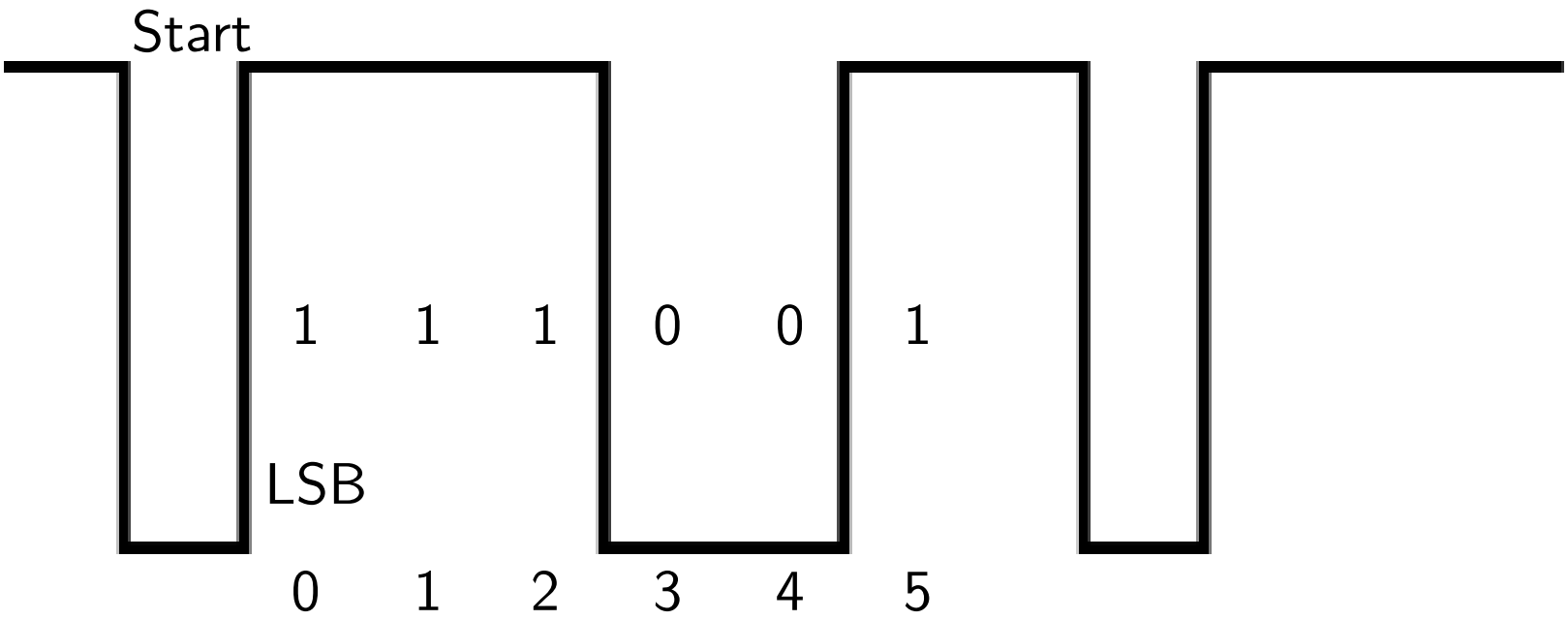

Start
1
LSB
0

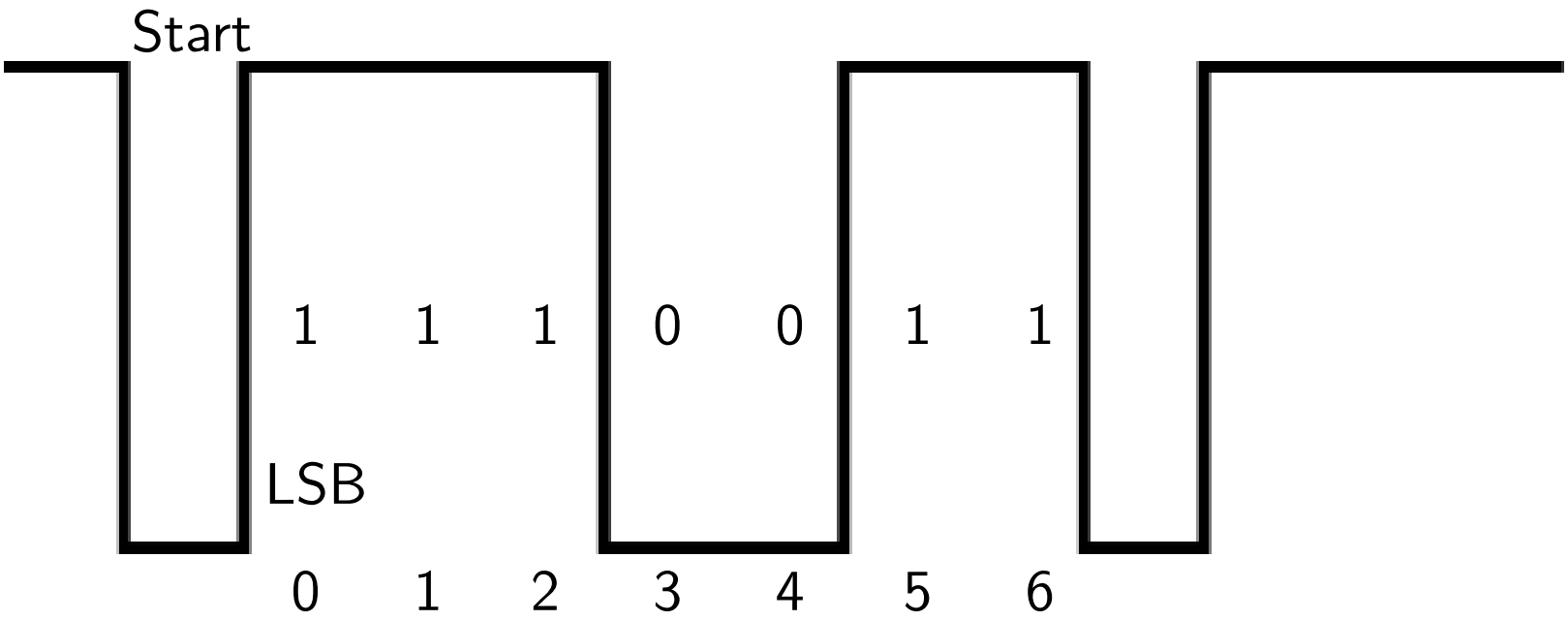

Start
1
1
LSB
0
1

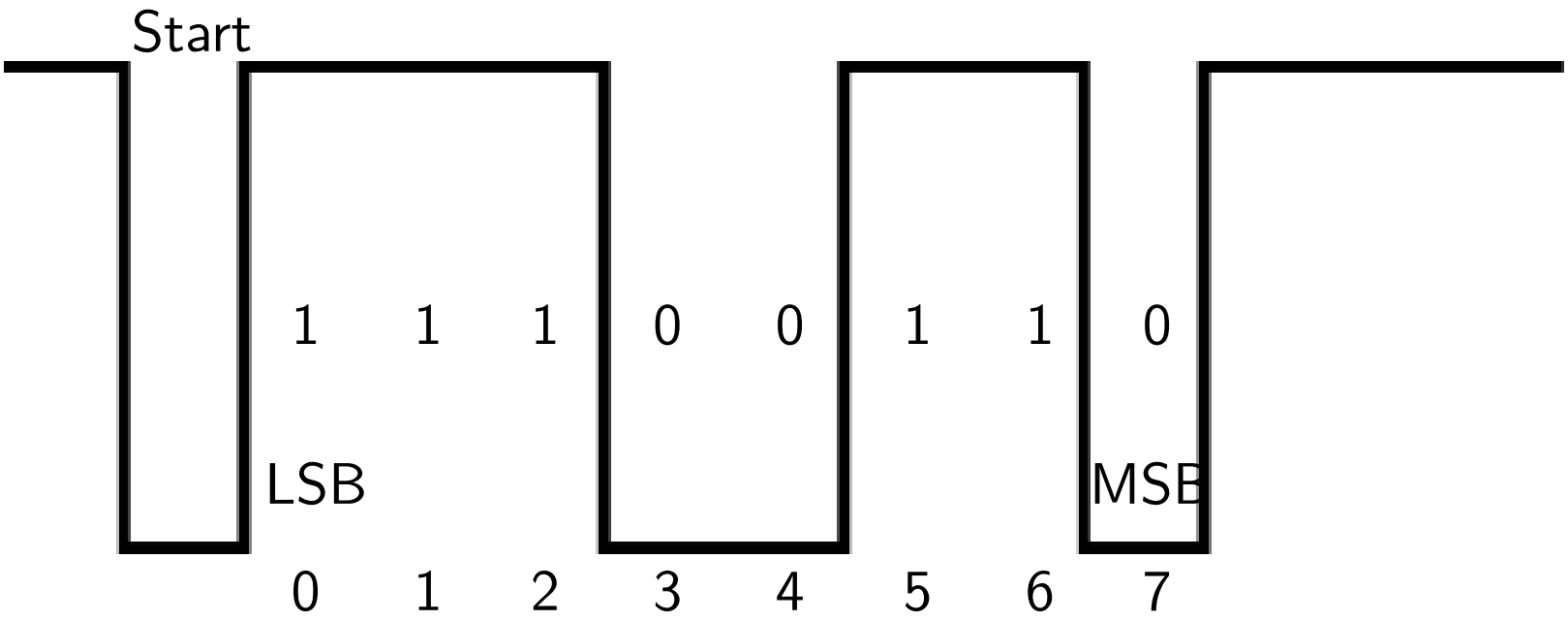

Start
1
1
1
LSB
0
1
2

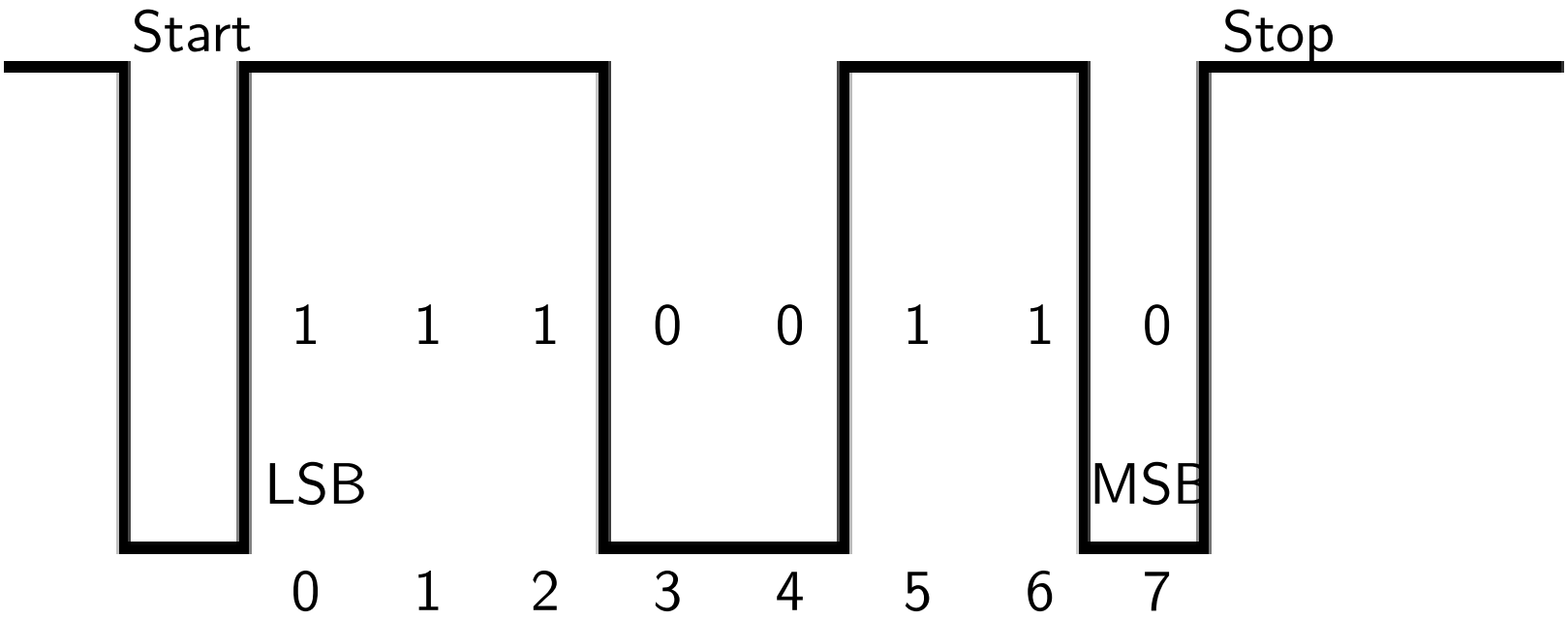

Start
1
1
1
0
LSB
0
1
2
3

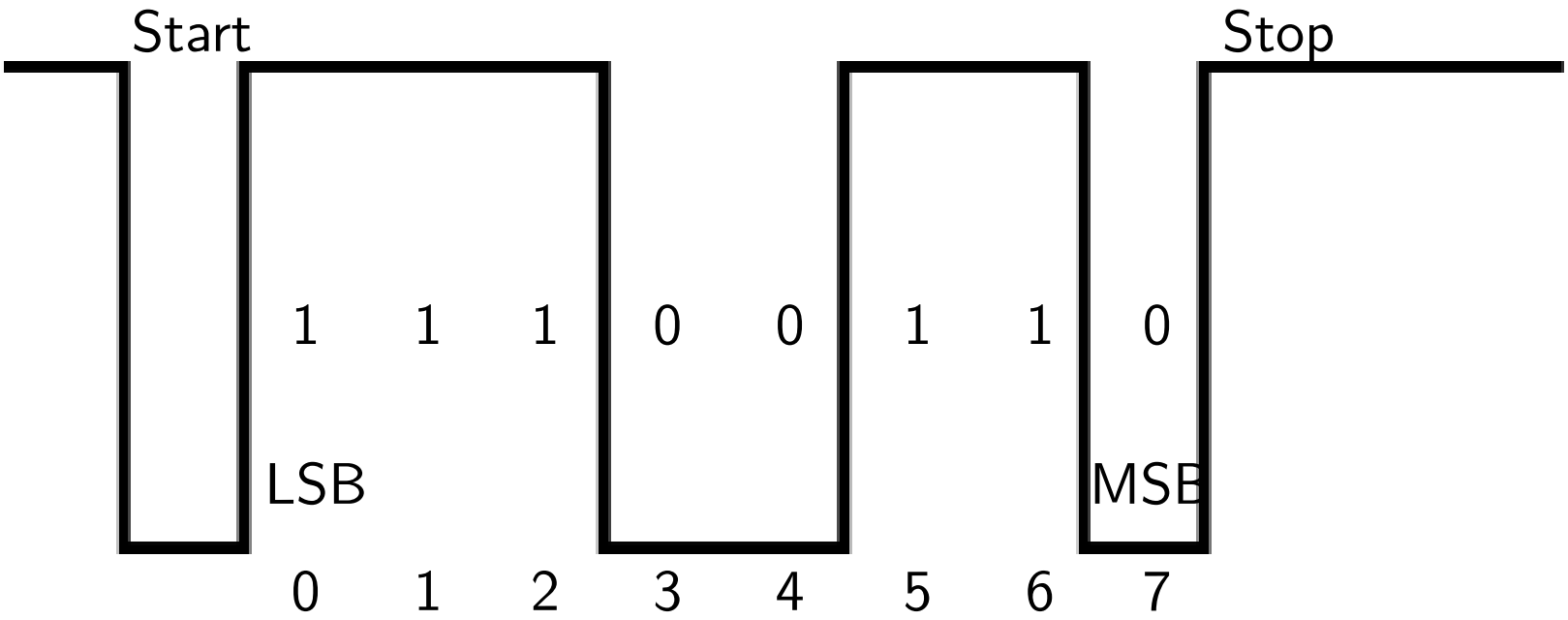

Start
1
1
1
0
0
LSB
0
1
2
3
4


Start
1
1
1
0
0
1
LSB
0
1
2
3
4
5

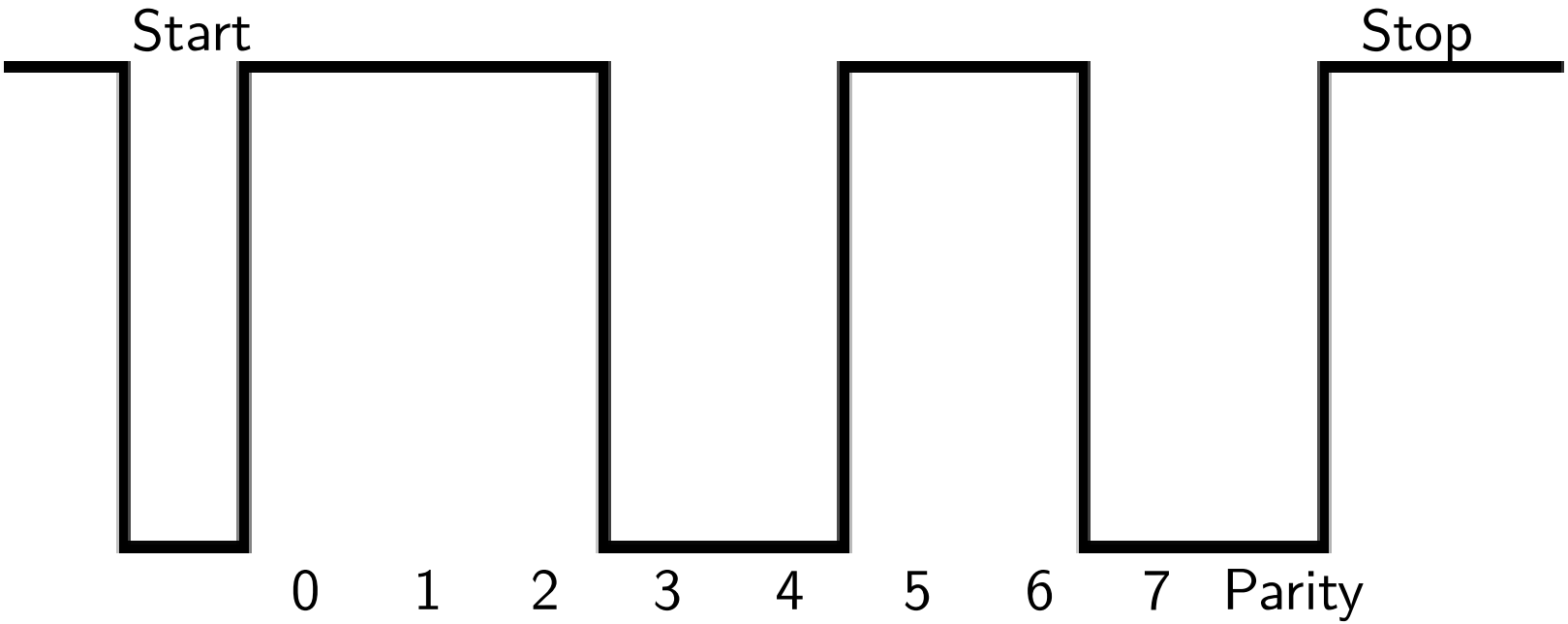

Start
1
1
1
0
0
1
1
LSB
0
1
2
3
4
5
6

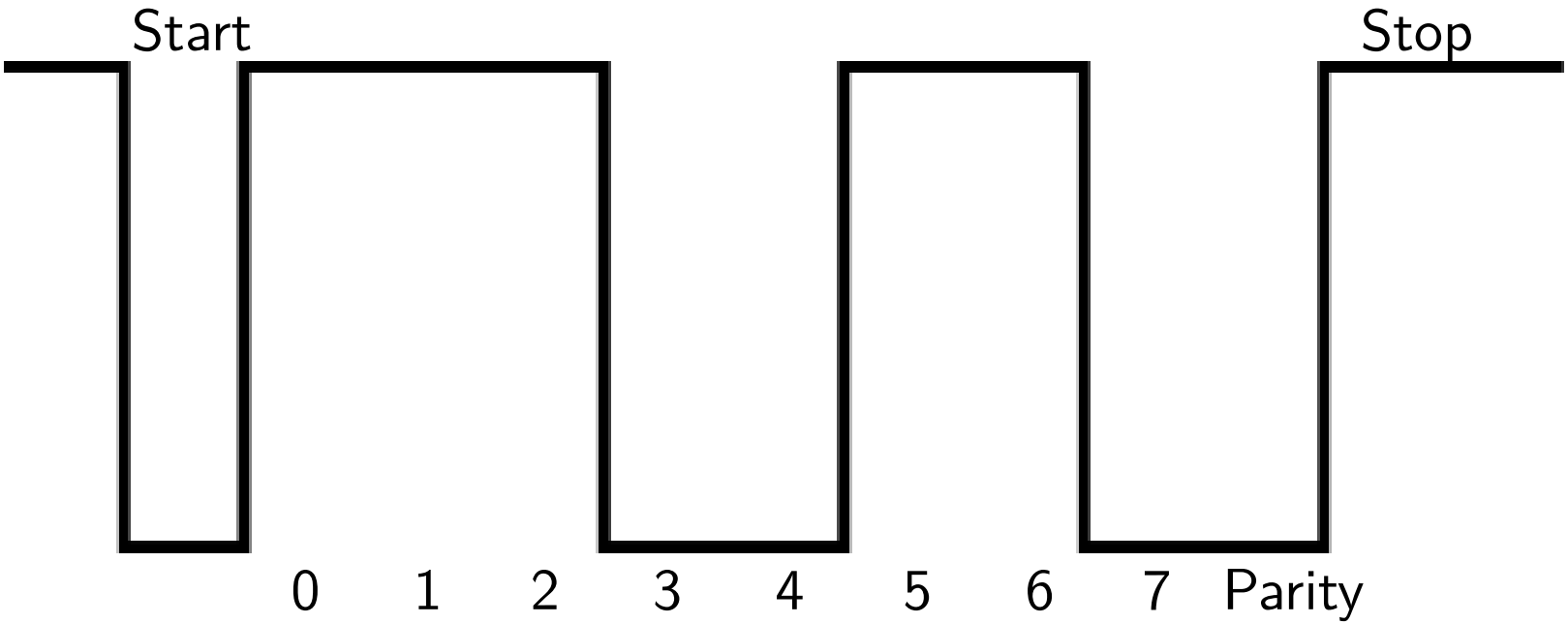

Start
1 1 1 0 0 1 1 0
LSB
MSB
0 1 2 3 4 5 6 7


Start
Stop
1 1 1 0 0 1 1 0
LSB
MSB
0 1 2 3 4 5 6 7

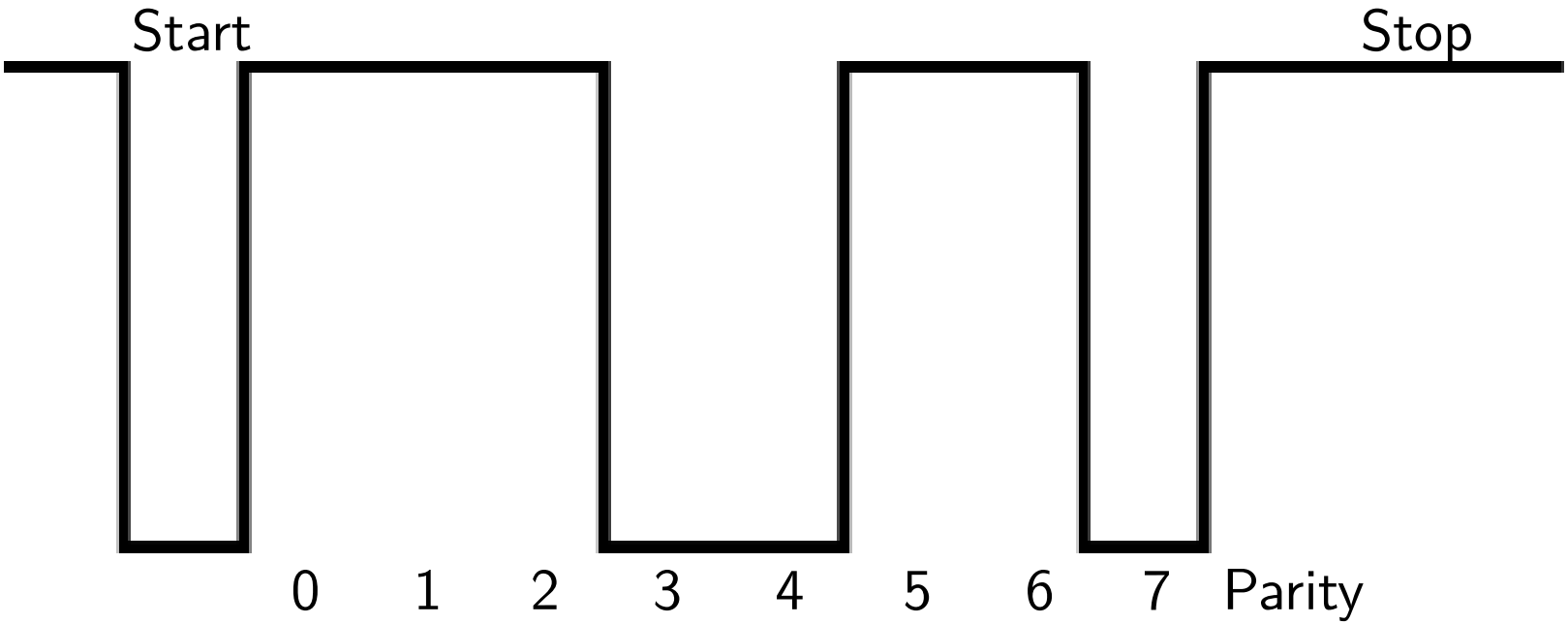


UART no parity - 01100111


Start
Stop
0
1
2
3
4
5
6
7
Parity



UART even parity



UART odd parity

Baud rate calculation

Baud rate calculation

- Baud rate is the number of bits possible in a second

Baud rate calculation

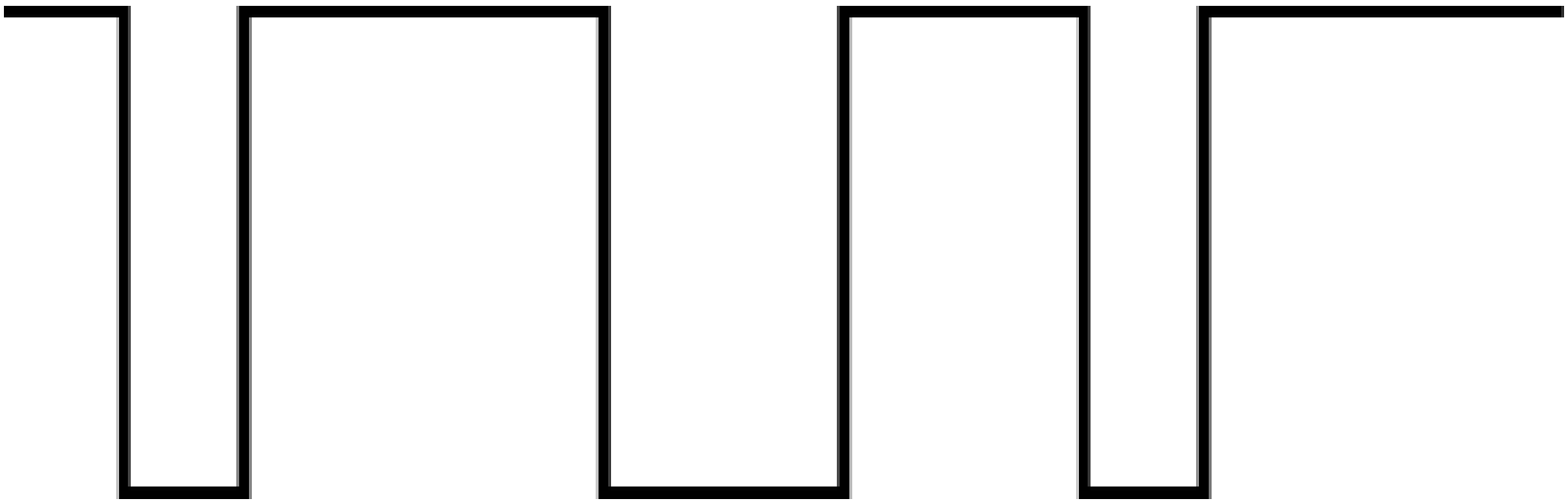
- Baud rate is the number of bits possible in a second
- e.g. 9600 baud $\rightarrow$ 1 bit takes $\frac{1}{9600}$ second

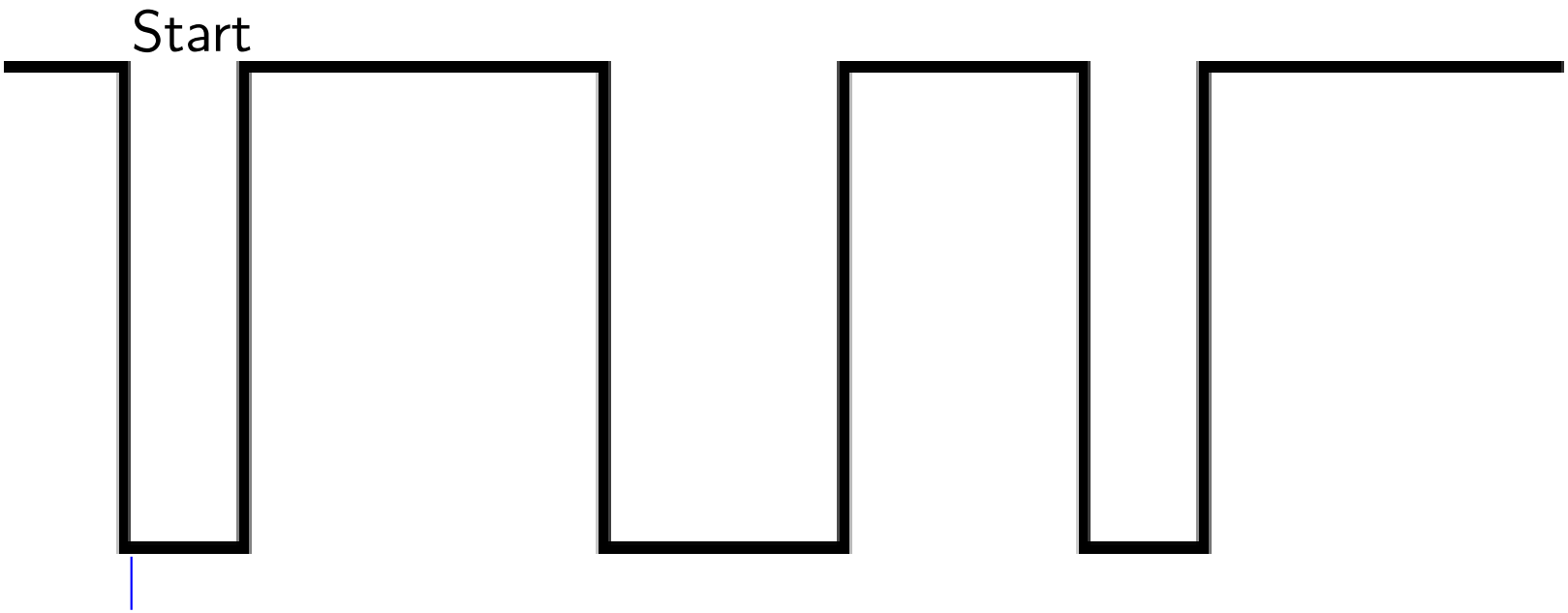
Baud rate calculation

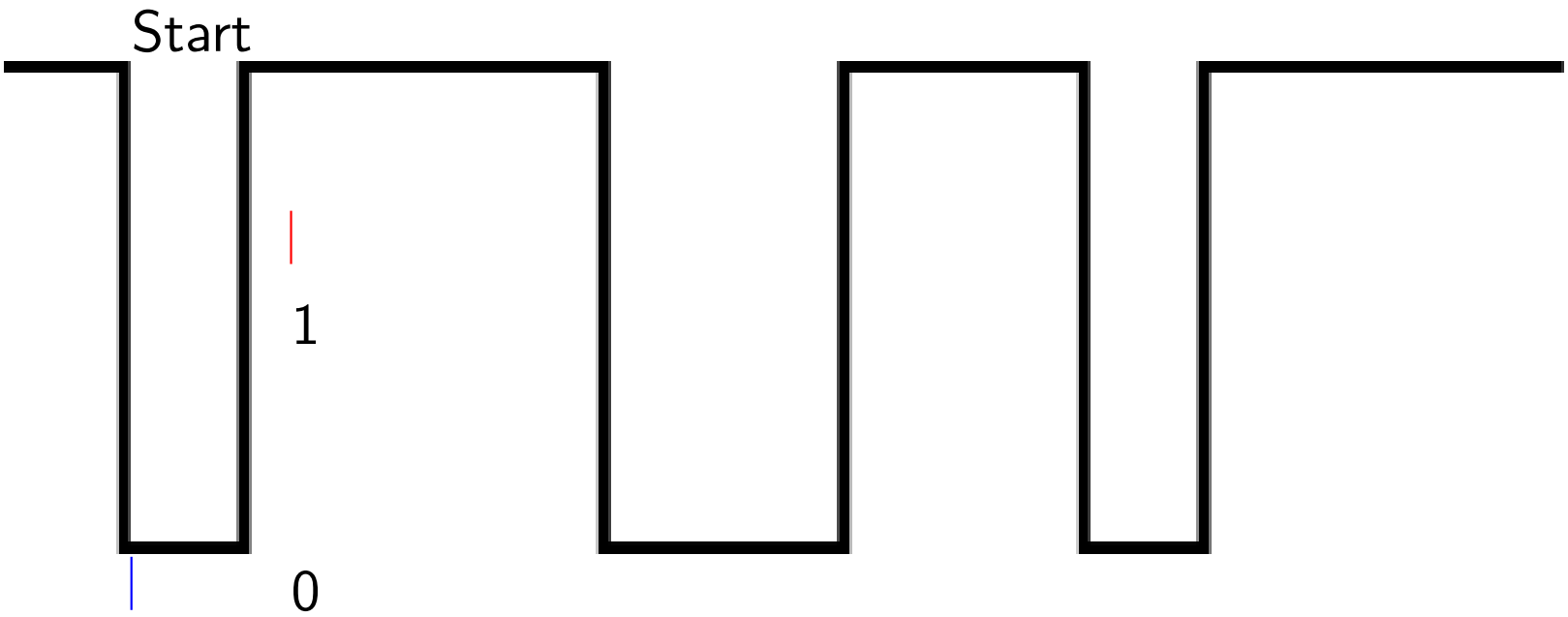
- Baud rate is the number of bits possible in a second
- e.g. 9600 baud $\rightarrow$ 1 bit takes $\frac{1}{9600}$ second
- After start bit is detected, wait time for $1\frac{1}{2}$ bit to test for first data bit and then after every 1 bit interval

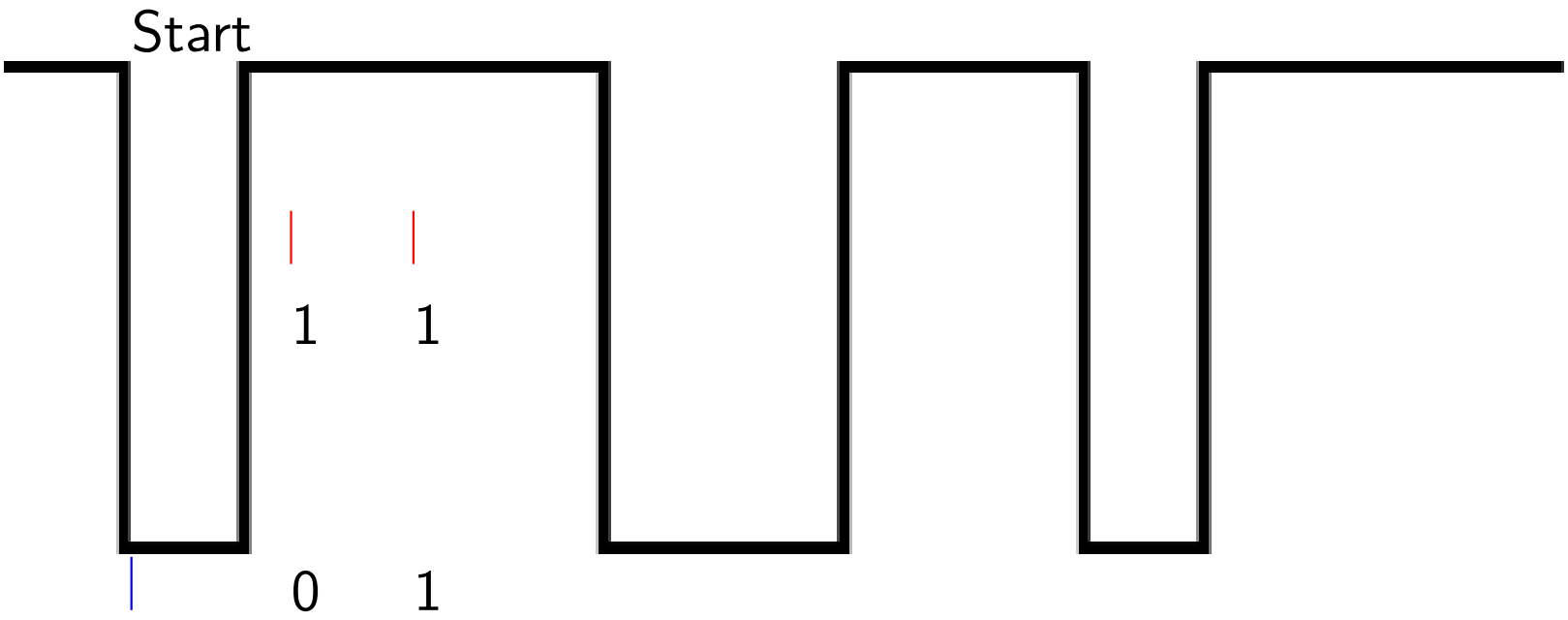
Baud rate calculation

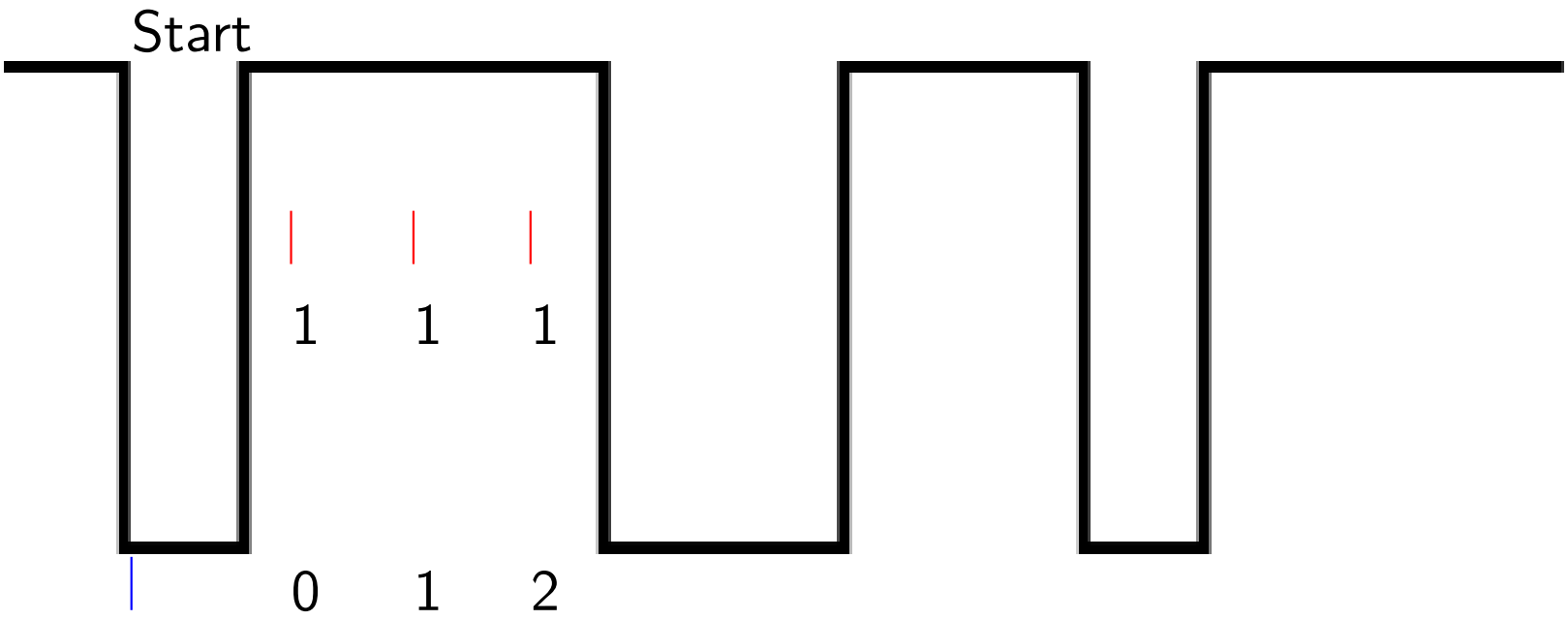
- Baud rate is the number of bits possible in a second
- e.g. 9600 baud $\rightarrow$ 1 bit takes $\frac{1}{9600}$ second
- After start bit is detected, wait time for $1\frac{1}{2}$ bit to test for first data bit and then after every 1 bit interval
- Resetting at the start bit allows some clock variation

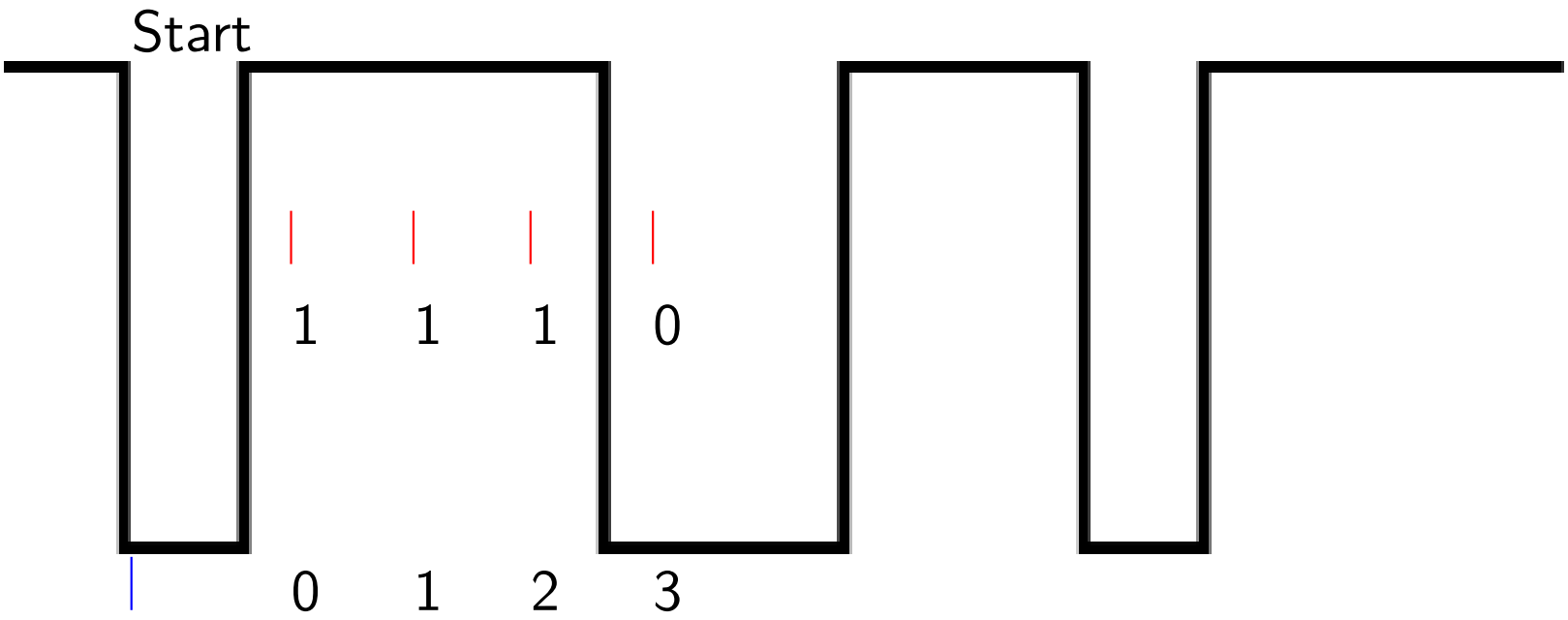


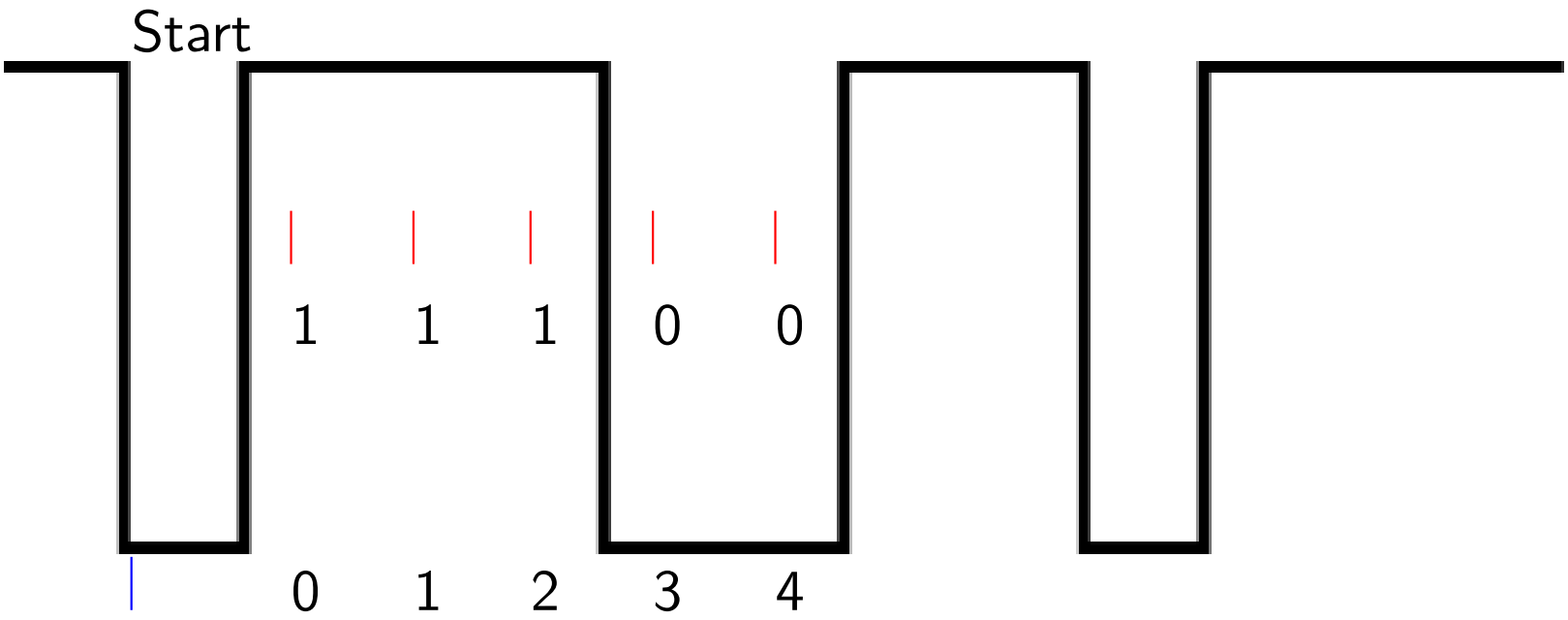

Start

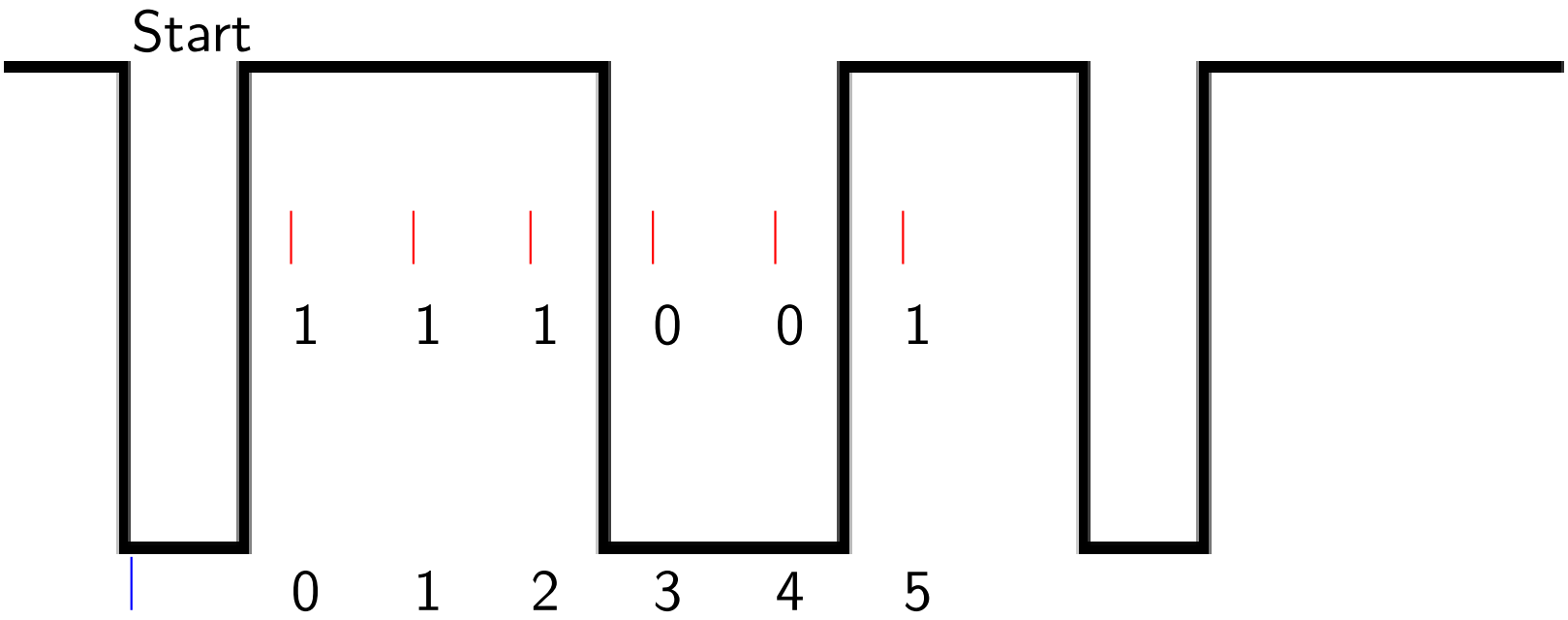

Start
1
0

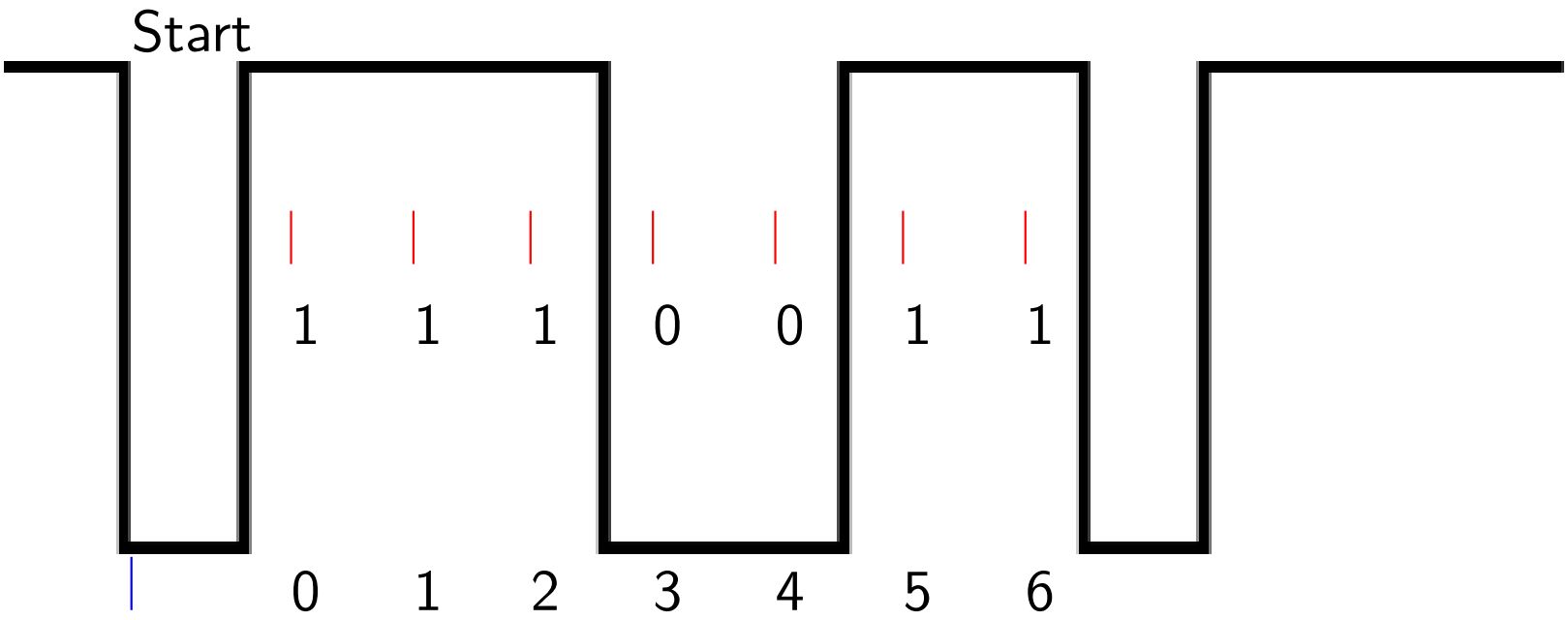

Start
1
1
0
1

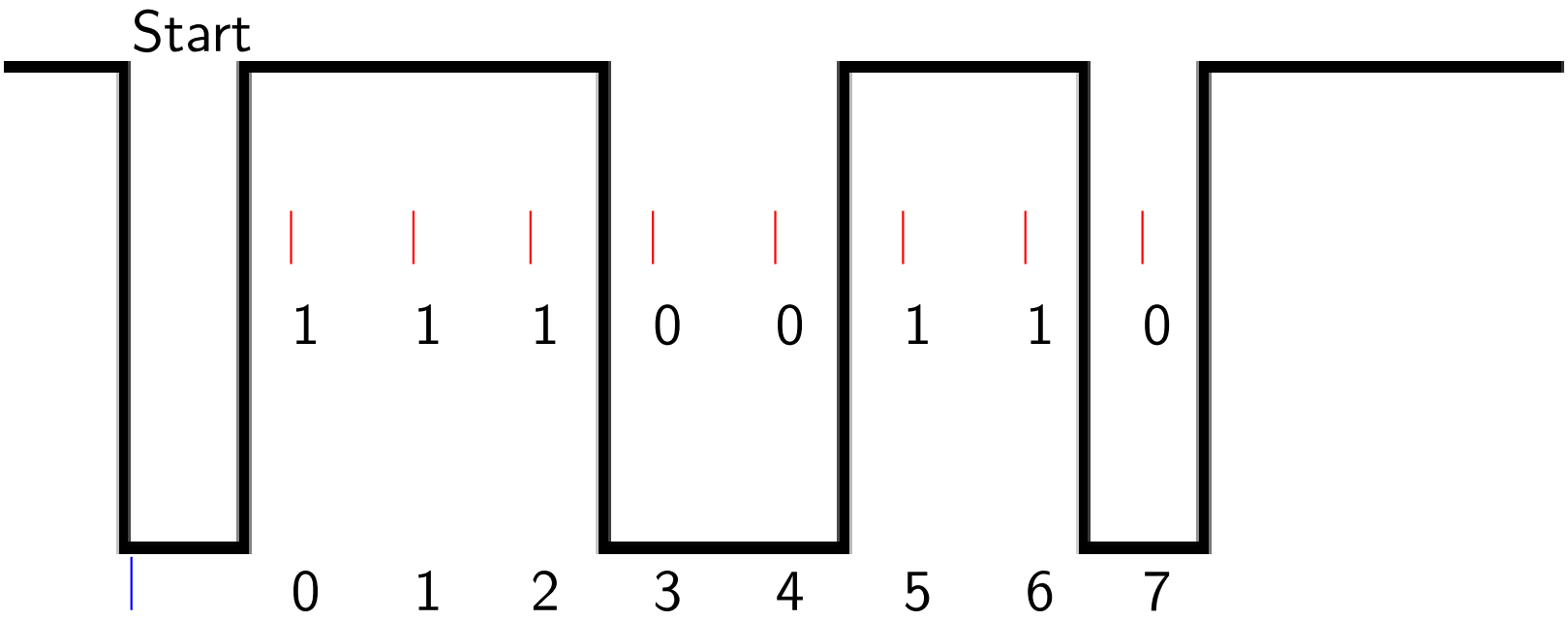

Start
1
1
1
0
1
2

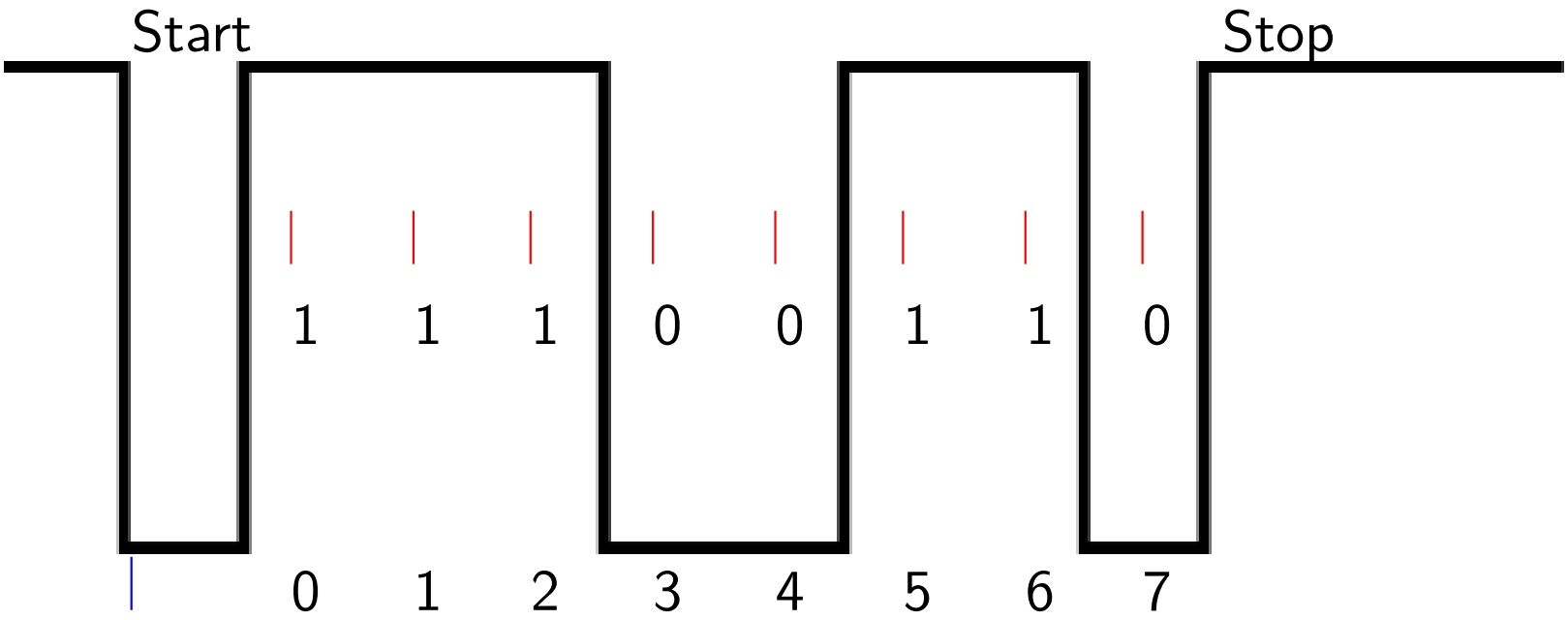

Start
1
1
1
0
0
1
2
3

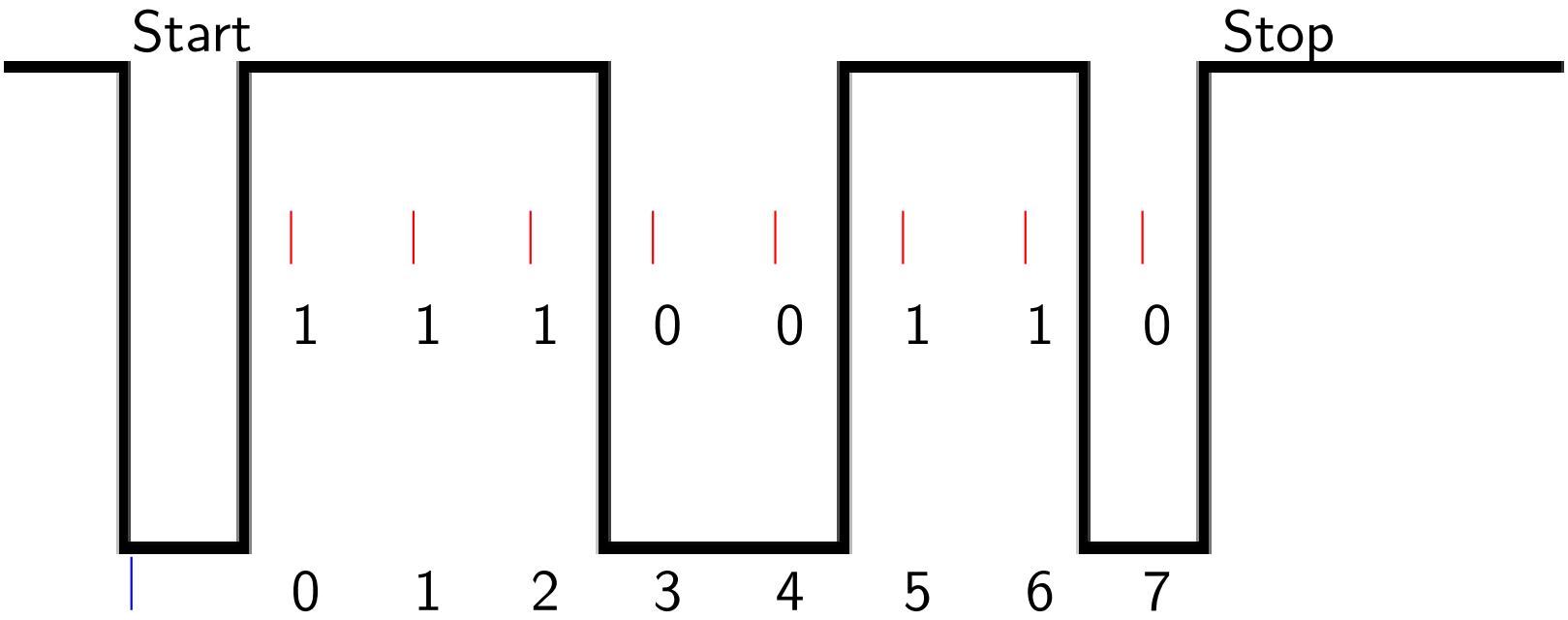

Start
1
1
1
0
0
0
1
2
3
4


Start
1
1
1
0
0
1
0
1
2
3
4
5


Start
1
1
1
0
0
1
1
0
1
2
3
4
5
6


Start
1
1
1
0
0
1
1
0
0
1
2
3
4
5
6
7


Start
Stop
1
1
1
0
0
1
1
0
0
1
2
3
4
5
6
7



Bit timing

RS232 communication

RS232 communication

- Voltages are inverted

RS232 communication

- Voltages are inverted
- $\pm 3 \rightarrow \pm 12$

RS232 communication

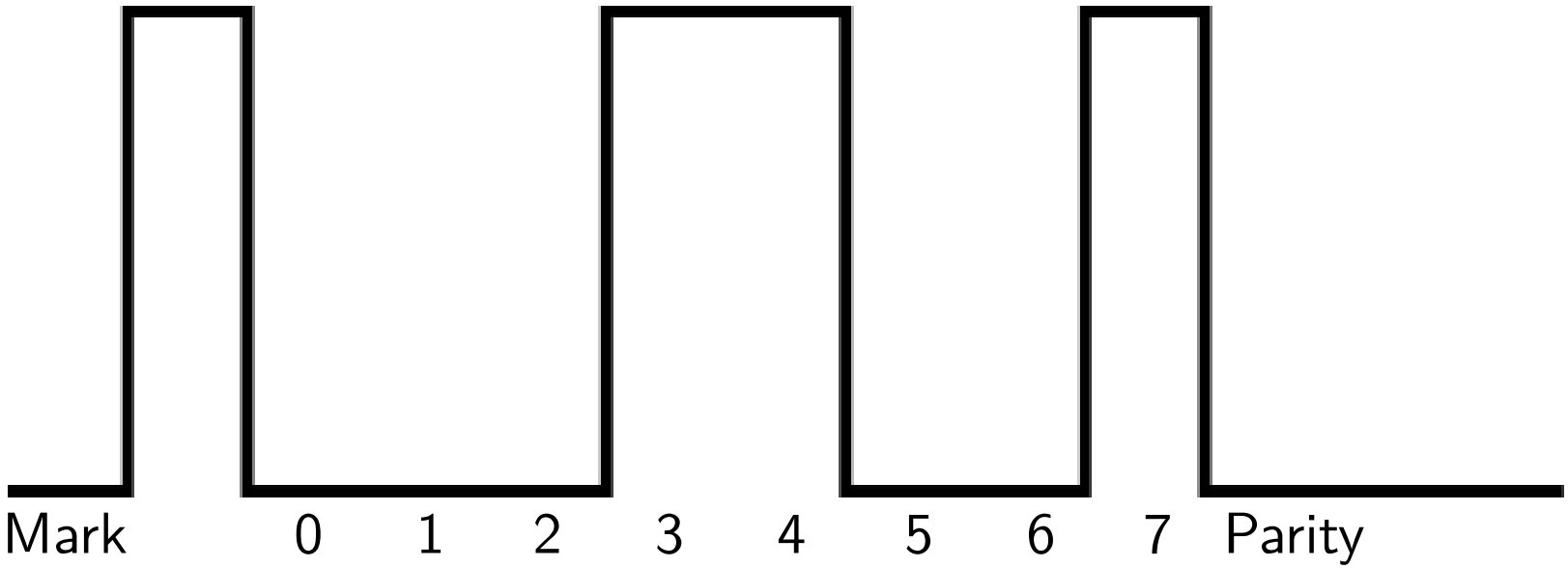
- Voltages are inverted
- $\pm 3 \rightarrow \pm 12$
- Zero is not a valid voltage

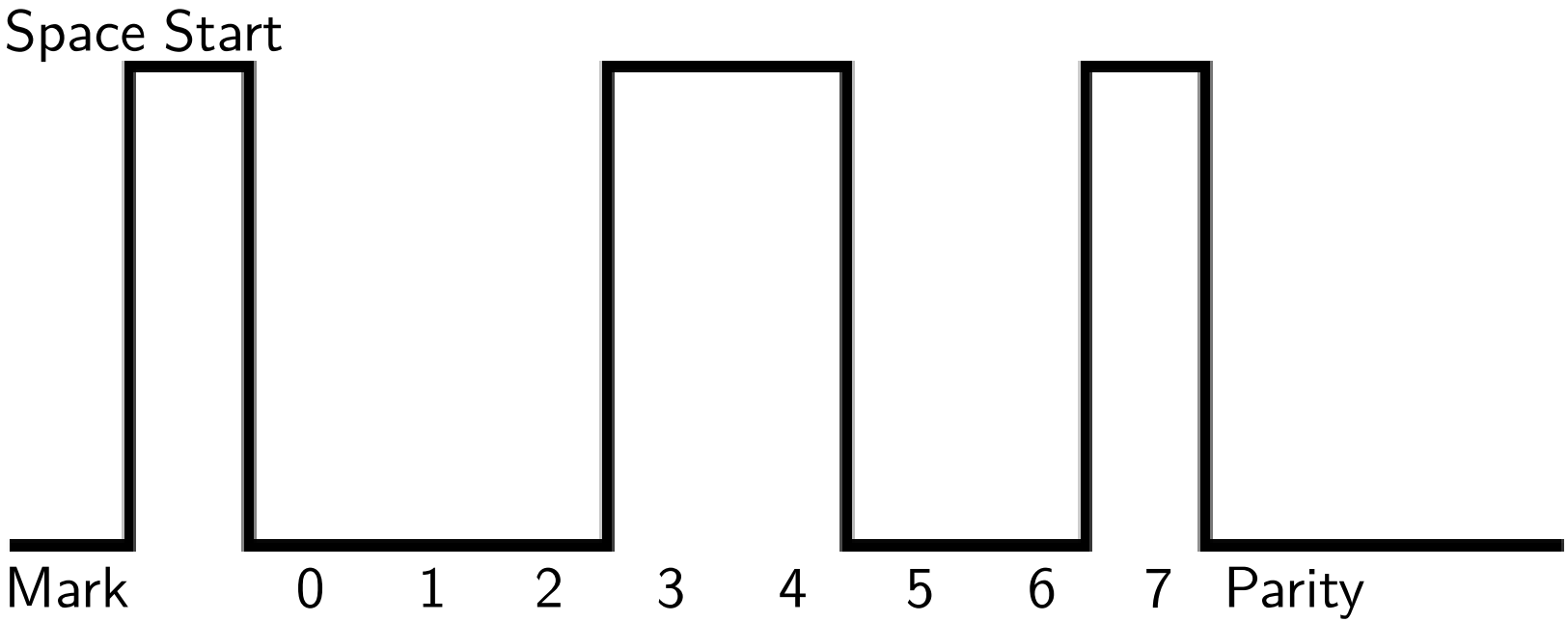
RS232 communication

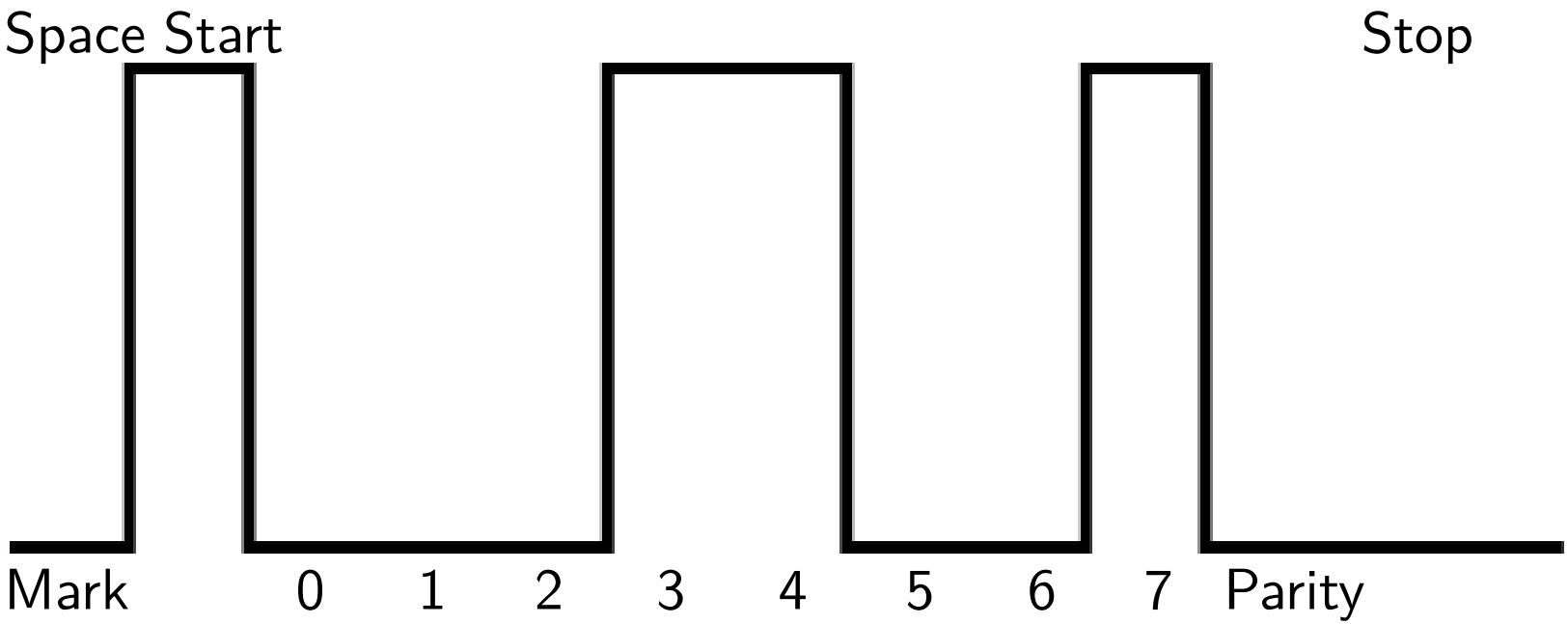
- Voltages are inverted
- $\pm 3 \rightarrow \pm 12$
- Zero is not a valid voltage
- Mark level (inactive/1) is a negative voltage

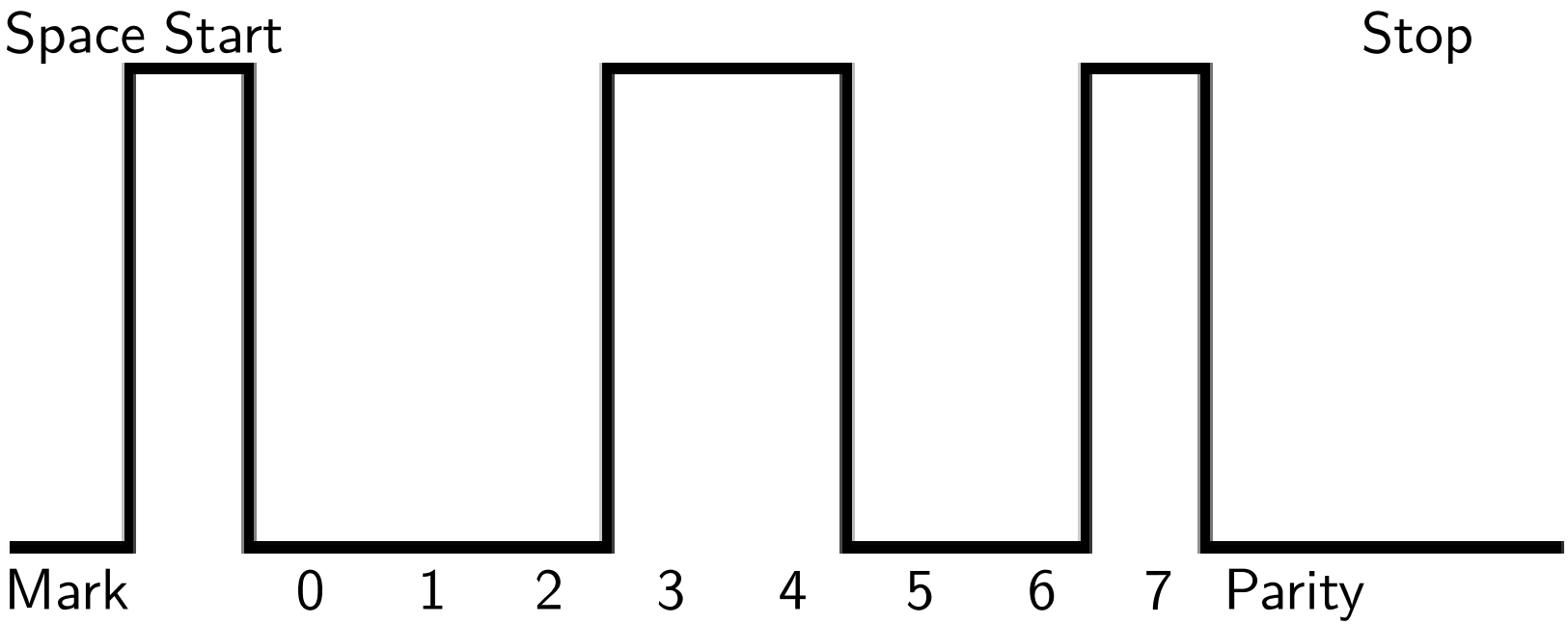
RS232 communication

- Voltages are inverted
- $\pm 3 \rightarrow \pm 12$
- Zero is not a valid voltage
- Mark level (inactive/1) is a negative voltage
- Space level (active/0) is a positive voltage


Mark
0
1
2
3
4
5
6
7
Parity


Space Start
Mark
0
1
2
3
4
5
6
7
Parity


Space Start
Stop
Mark
0
1
2
3
4
5
6
7
Parity



RS232 levels

UART or Serial Console

UART or Serial Console

- On the Raspberry Pi, the serial console uses the UART

UART or Serial Console

- On the Raspberry Pi, the serial console uses the UART
  $\rightarrow$ the UART isn't available for other devices

UART or Serial Console

- On the Raspberry Pi, the serial console uses the UART
  $\rightarrow$ the UART isn't available for other devices
- The serial console can be disabled

UART or Serial Console

- On the Raspberry Pi, the serial console uses the UART

  $\rightarrow$ the UART isn't available for other devices
- The serial console can be disabled

  This means you need to connect to monitor, keyboard, etc.

There is a script to enable and disable the serial console

There is a script to enable and disable the serial console

- **sudo rpi-serial-console disable**

  will disable the console

There is a script to enable and disable the serial console

- **sudo rpi-serial-console disable**

  will disable the console
- **sudo rpi-serial-console enable**

  will enable the console

There is a script to enable and disable the serial console

- **sudo rpi-serial-console disable**

  will disable the console

- **sudo rpi-serial-console enable**

  will enable the console

Remember you probably don't want to be using this command from *within* the serial console.

PySerial

PySerial

- **ser = serial.Serial()**

  open port

PySerial

- **ser = serial.Serial()**

  open port
- **ser.baudrate = 19200**

  set baudrate

PySerial

- **ser = serial.Serial()**

  open port
- **ser.baudrate = 19200**

  set baudrate
- **ser.port ='//dev//ttyAMA0'**

  set port

PySerial

- **ser = serial.Serial()**

  open port
- **ser.baudrate = 19200**

  set baudrate
- **ser.port ='//dev//ttyAMA0'**

  set port
- **ser.is_open**

  returns *True* if open, *False* if not

PySerial (continued)

PySerial (continued)

- **ser.write(b'string')**

  write string

PySerial (continued)

- **ser.write(b'string')**

  write string
- **ser.read(10)**

  read 10 bytes (or until timeout)

PySerial (continued)

- **ser.write(b'string')**

  write string
- **ser.read(10)**

  read 10 bytes (or until timeout)
- **ser.readline()**

  read until '\n' received

PySerial (continued)

- **ser.write(b'string')**

  write string
- **ser.read(10)**

  read 10 bytes (or until timeout)
- **ser.readline()**

  read until '\n' received
- **ser.close()**

  close port

PySerial sample code

PySerial sample code

```
import serial
ser = serial.Serial ("/dev/ttyAMA0")
ser.baudrate = 9600
data = ser.read(10)
#data = ser.readline()
ser.write(data)
ser.close()
```