CP316
Serial Communication-UART

Terry Sturtevant

Wilfrid Laurier University

February 13, 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

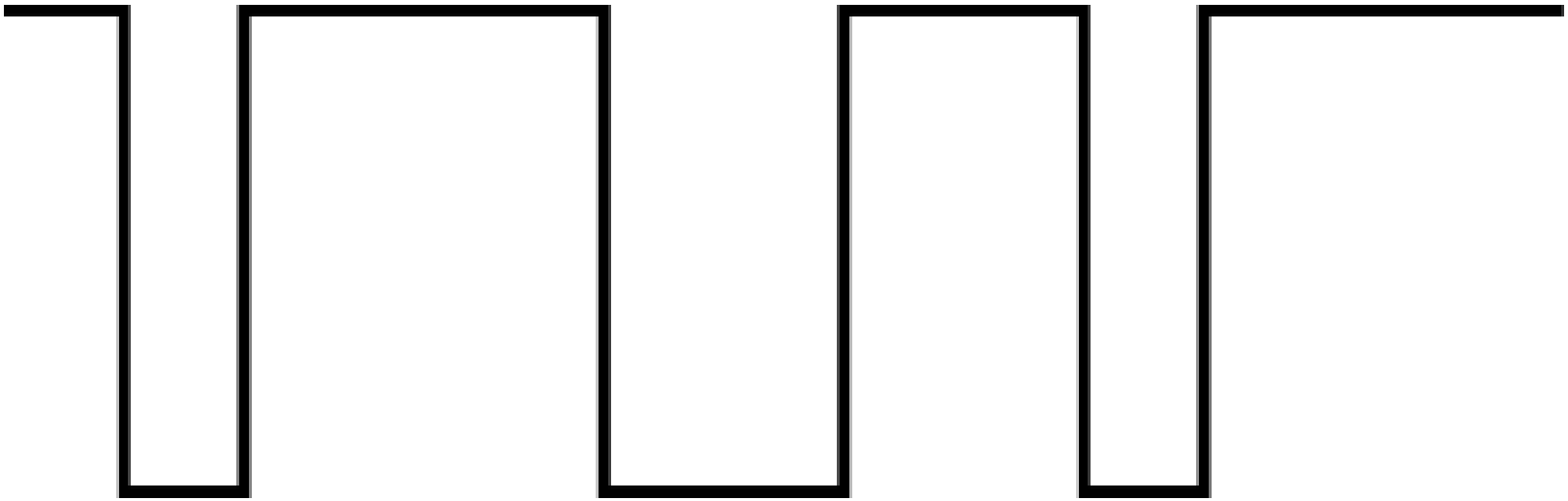
Communication parameters

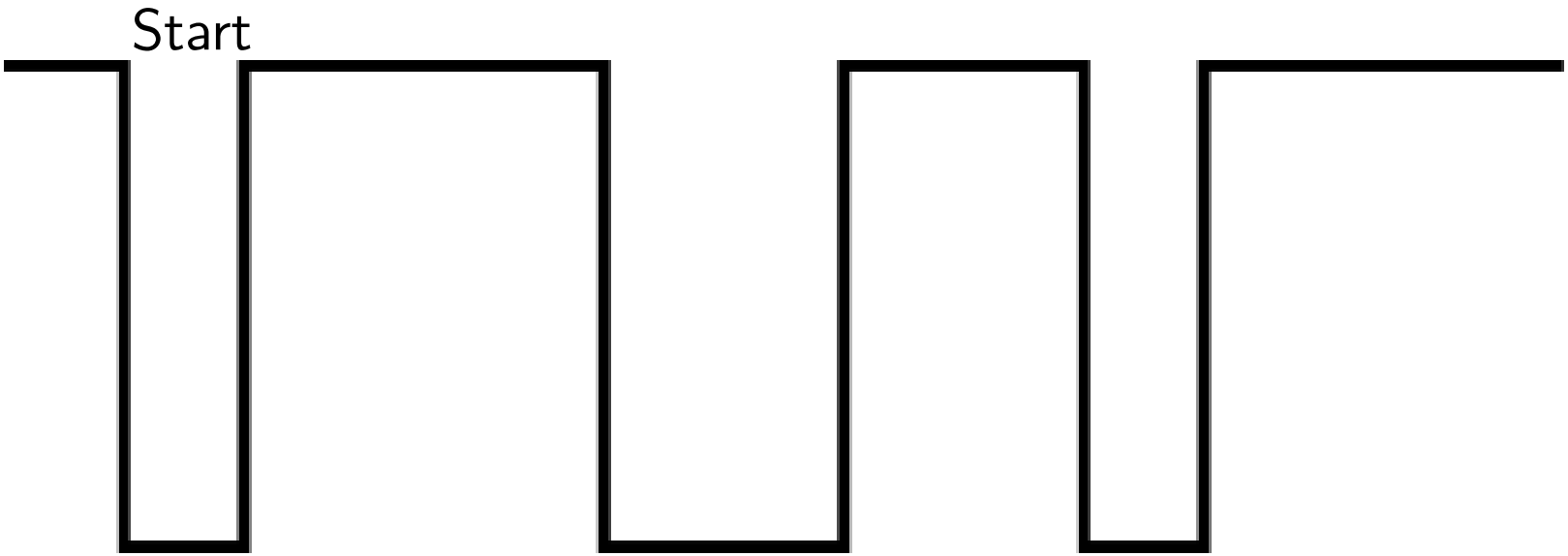
- 1 Start bit at “0” level
- LSB transmitted first
- Can have odd, even, or no parity bit
- 1 or 2 Stop bits at “1” level

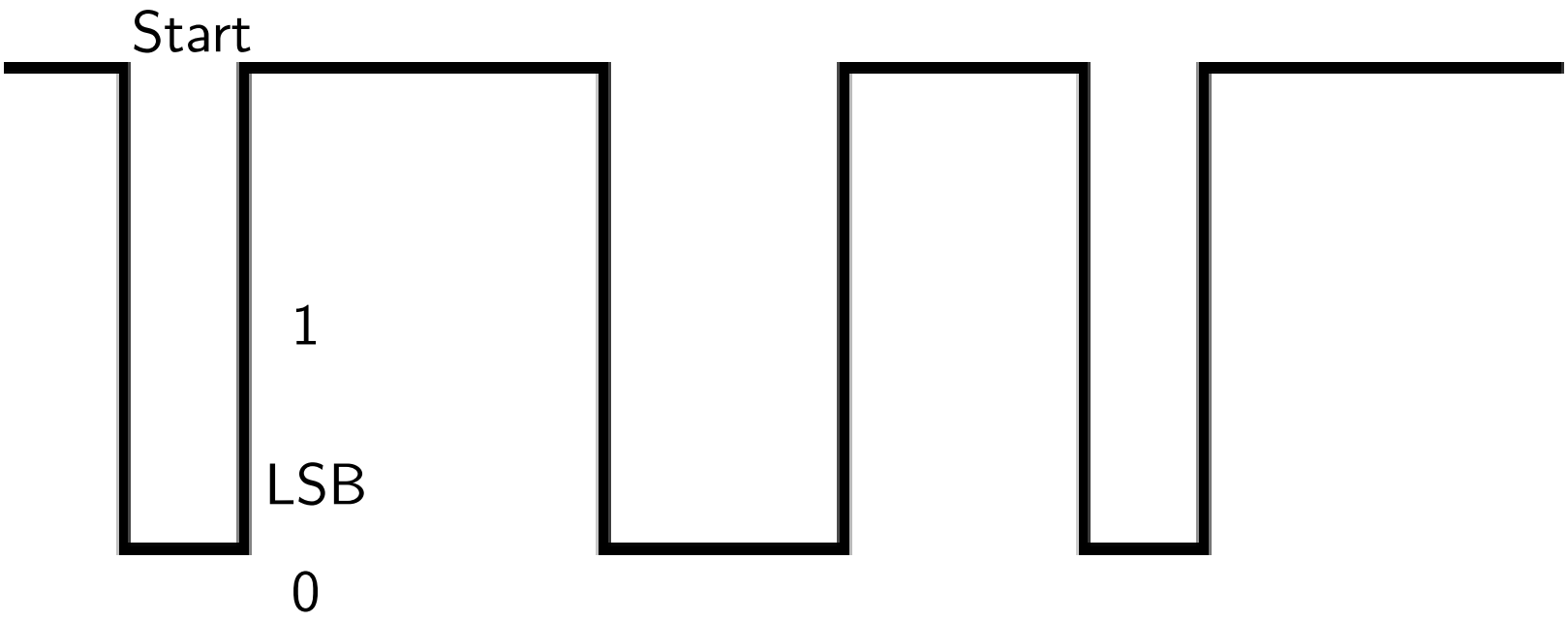
Communication parameters

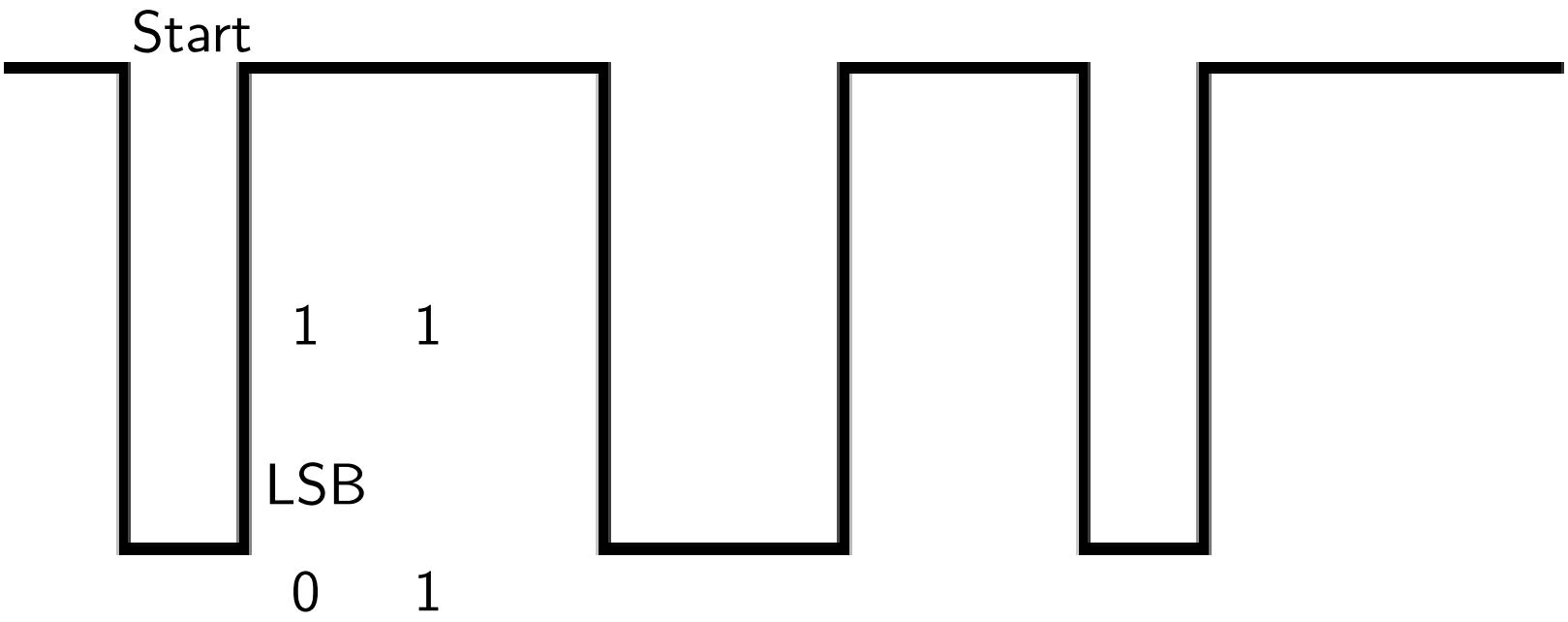
- 1 Start bit at “0” level
- LSB transmitted first
- Can have odd, even, or no parity bit
- 1 or 2 Stop bits at “1” level

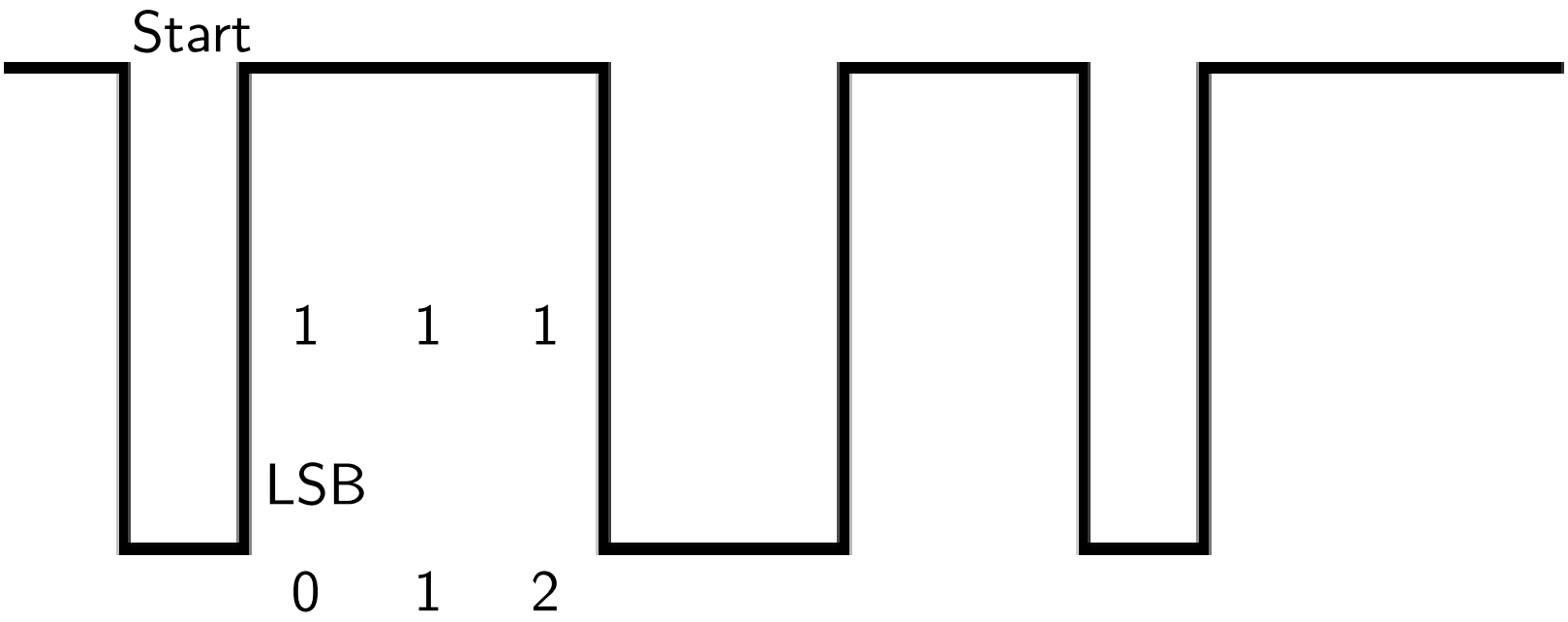
Since start and stop bits are opposite, new characters can always be detected.

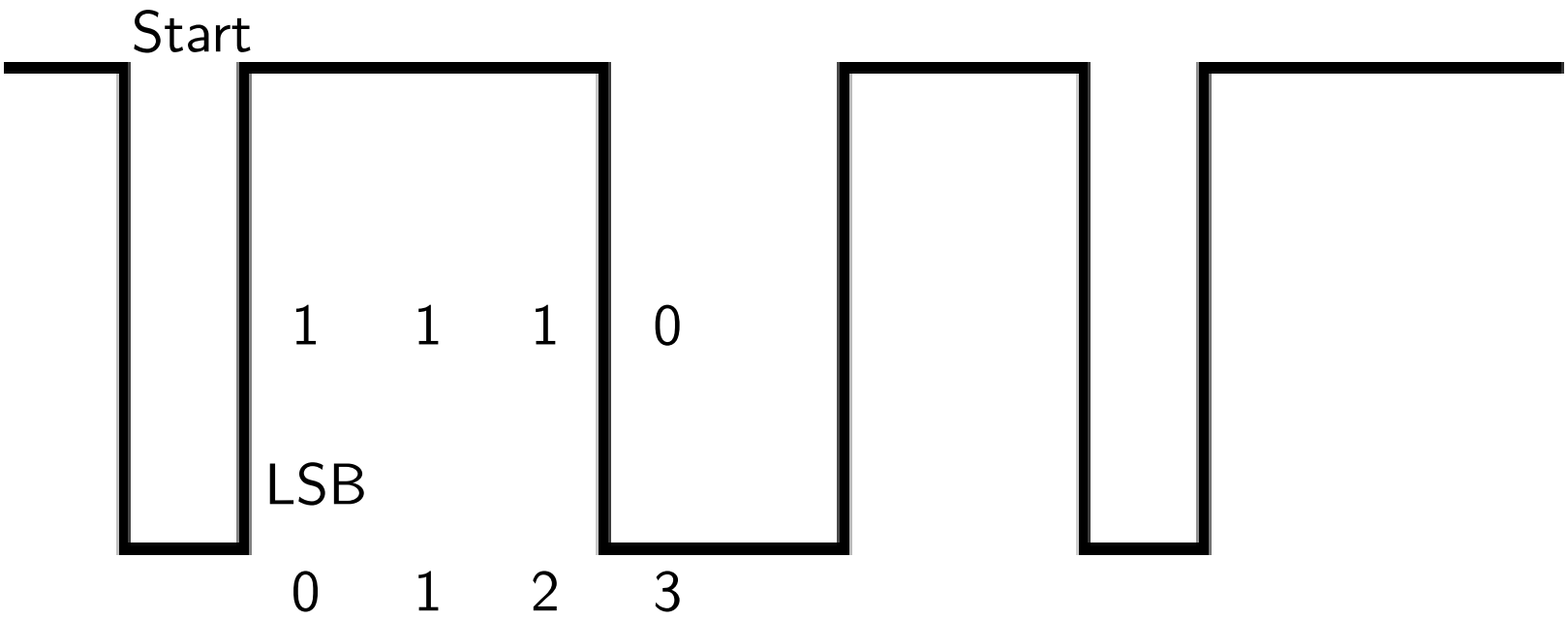


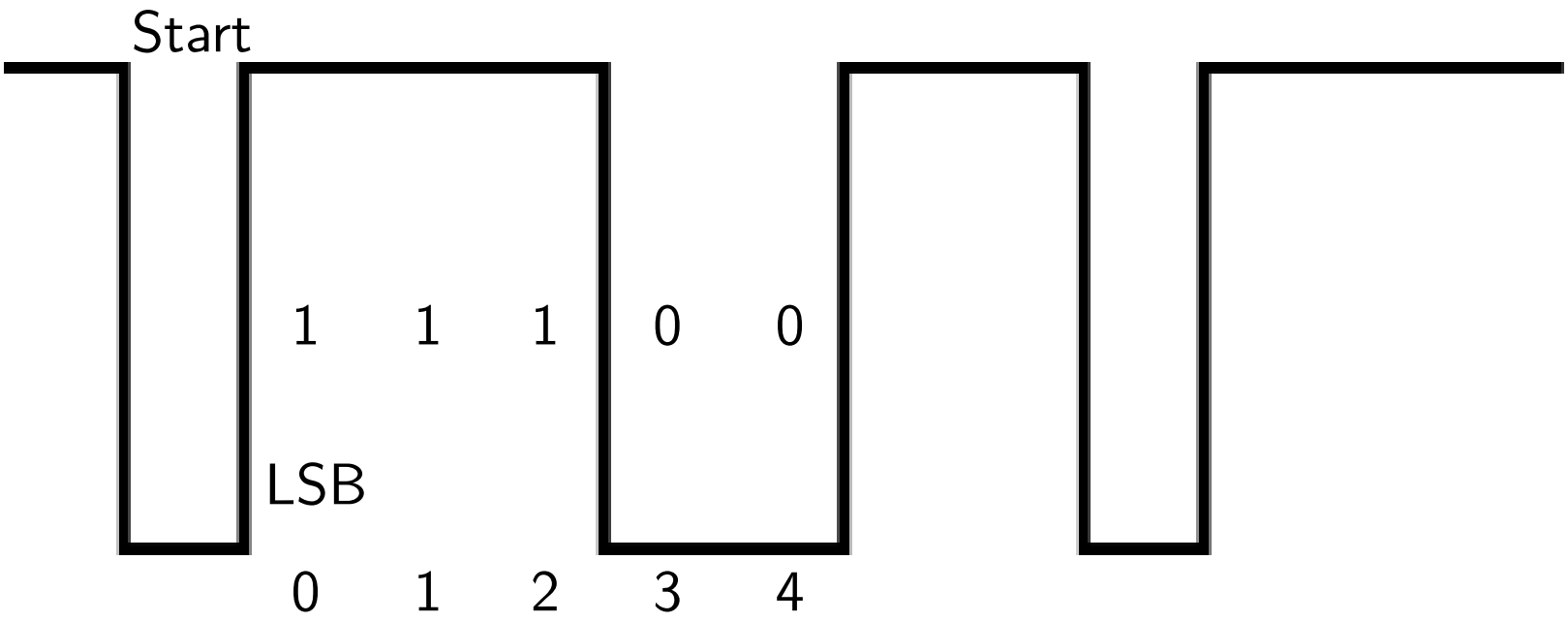

Start

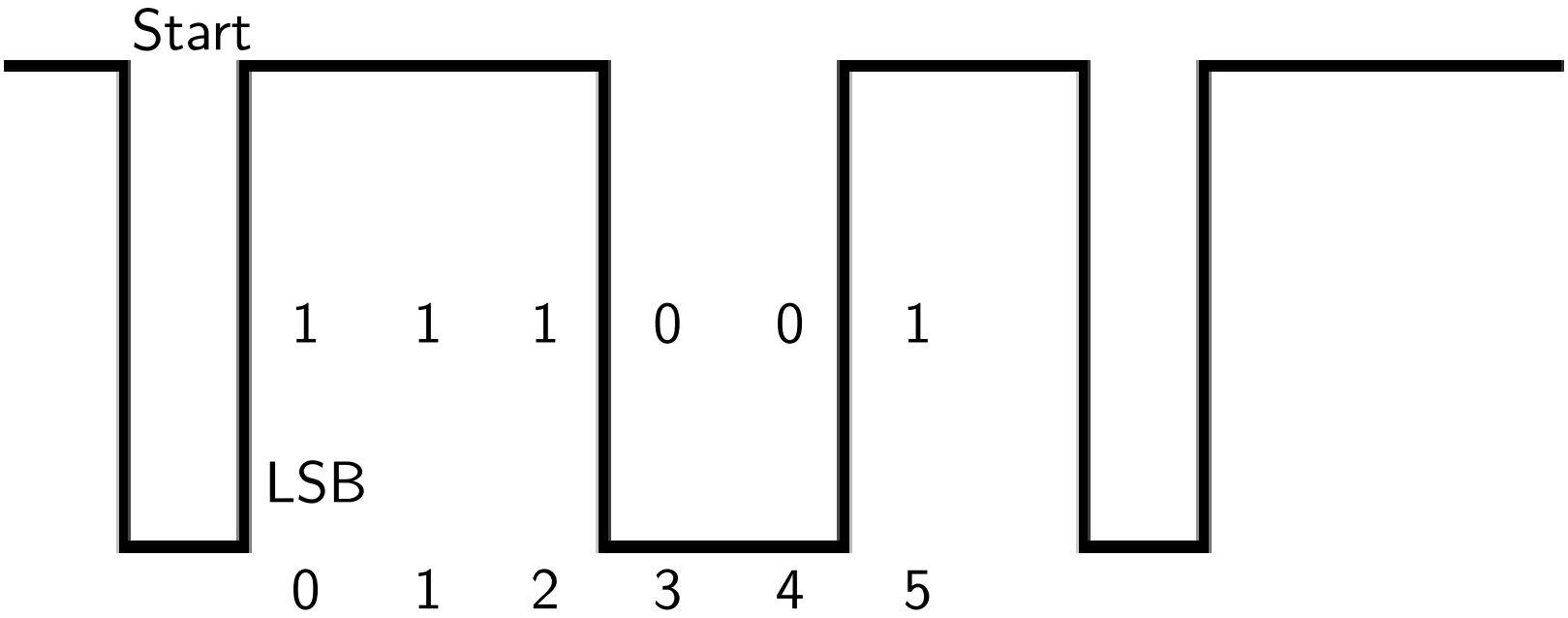

Start
1
LSB
0

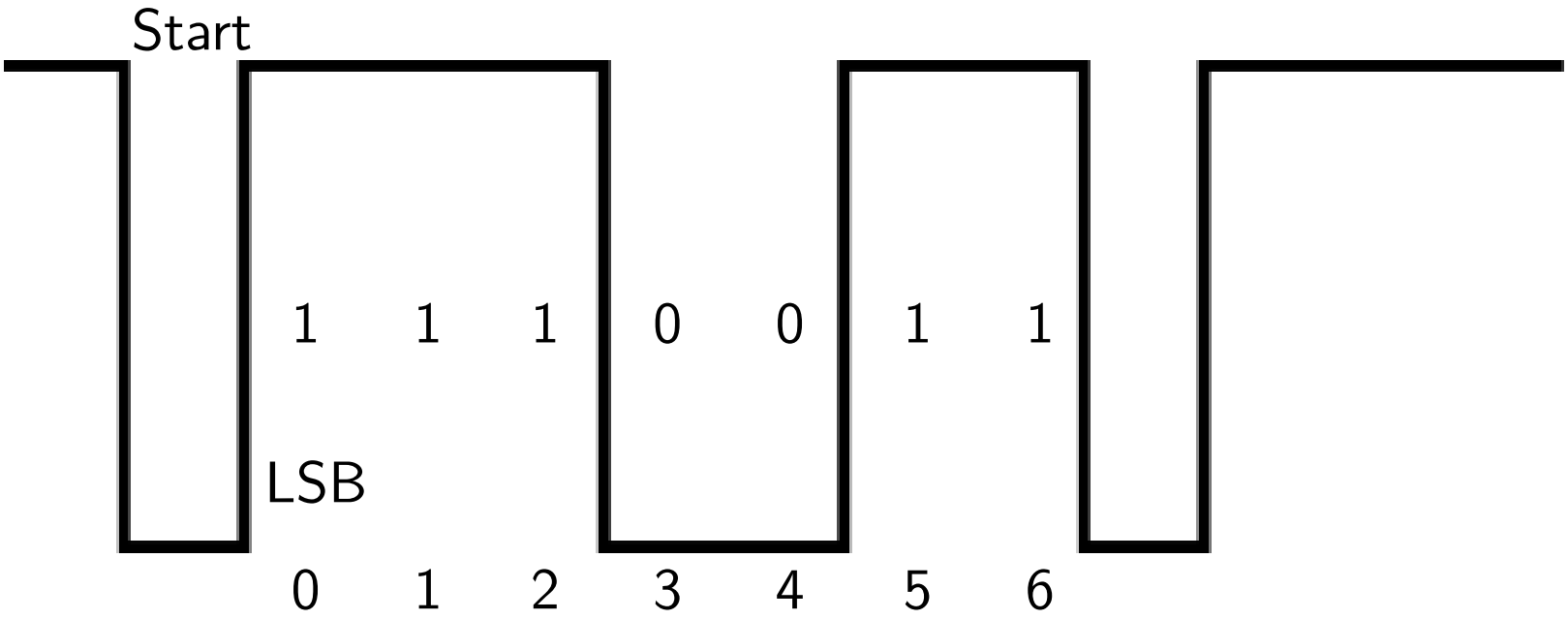

Start
1
1
LSB
0
1

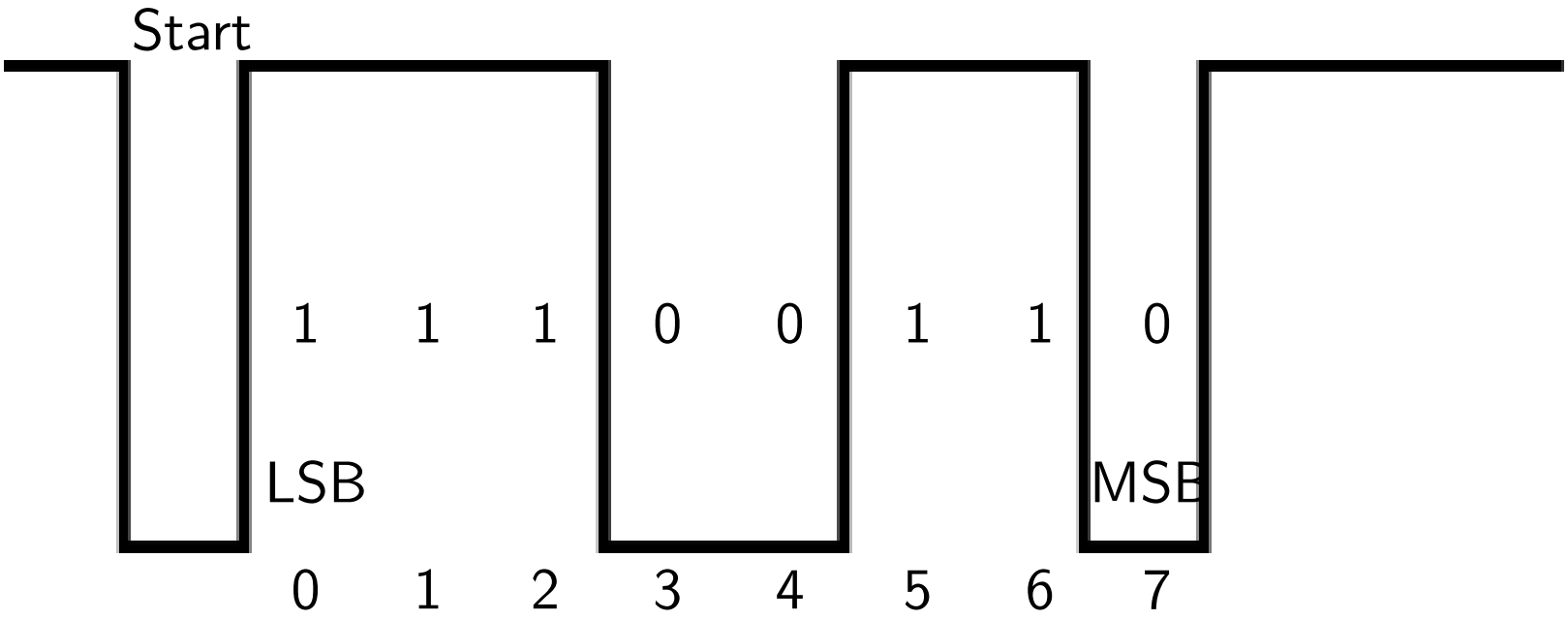

Start
1
1
1
LSB
0
1
2

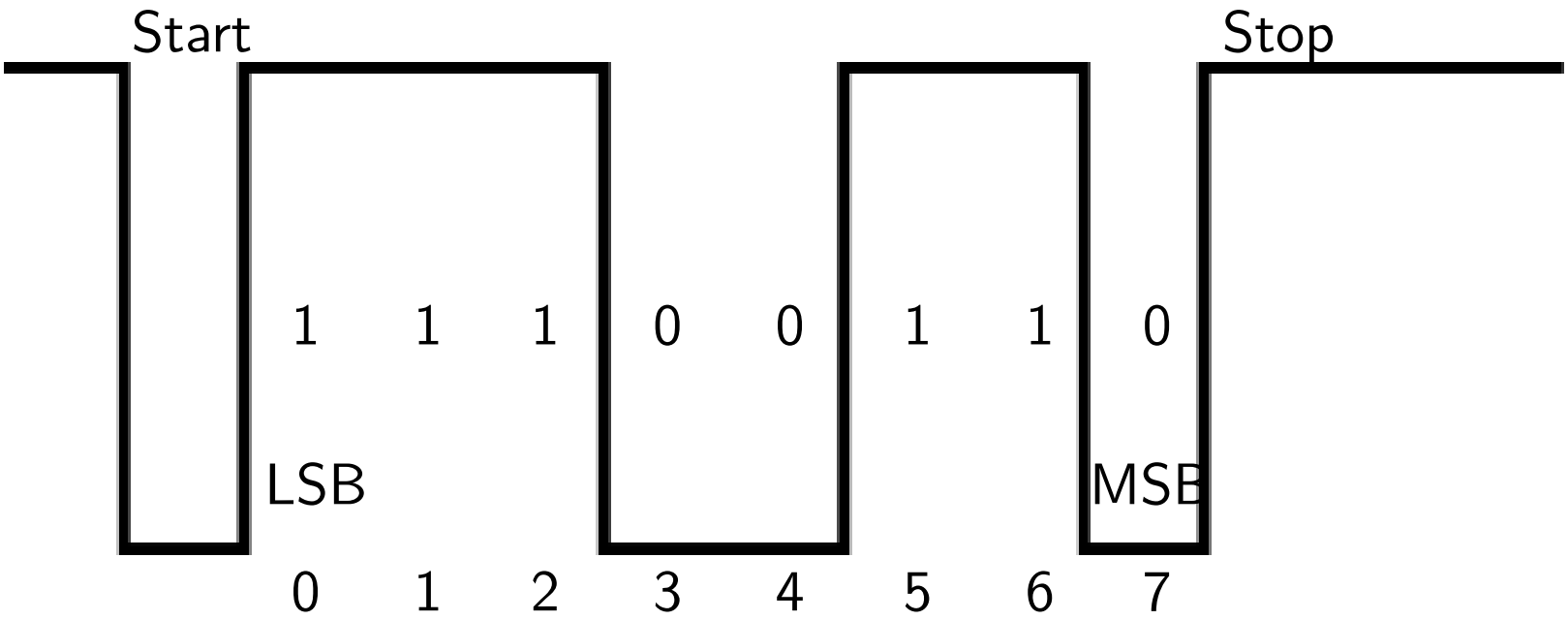

Start
1
1
1
0
LSB
0
1
2
3

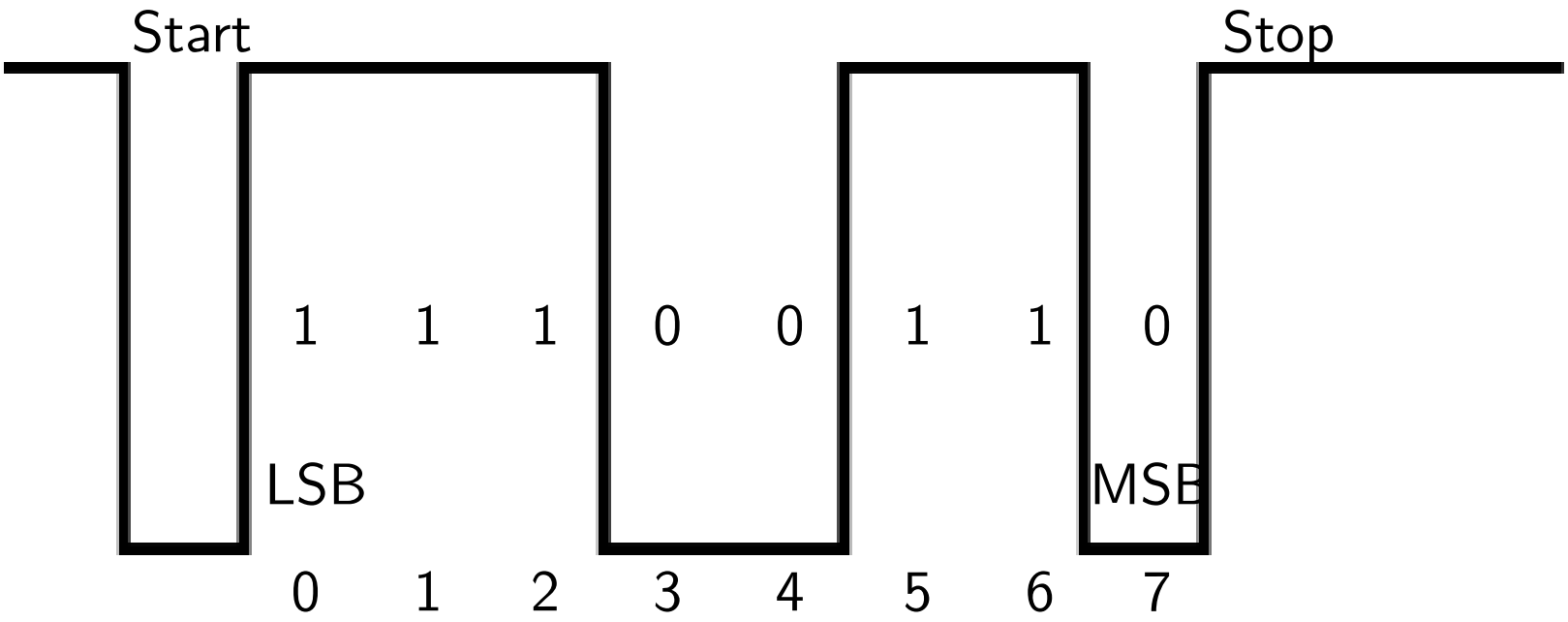

Start
1
1
1
0
0
LSB
0
1
2
3
4


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

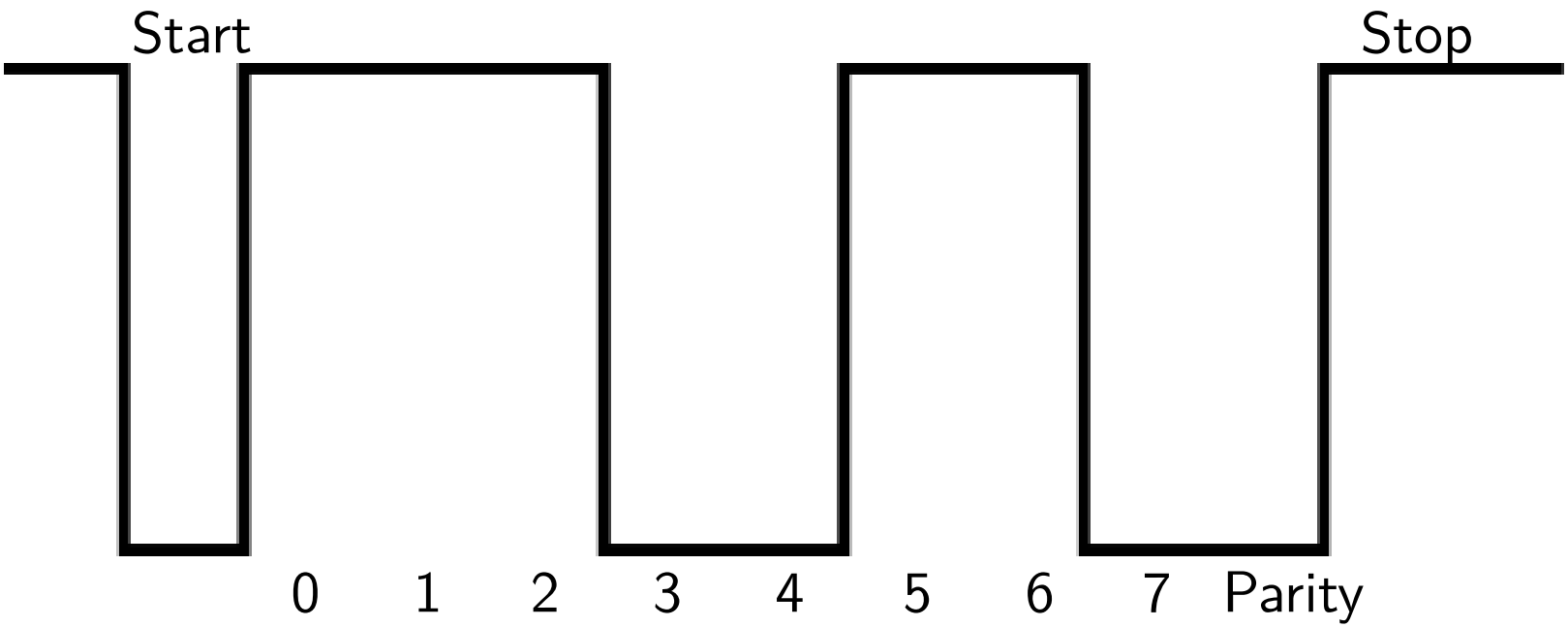

Start
1
1
1
0
0
1
1
LSB
0
1
2
3
4
5
6

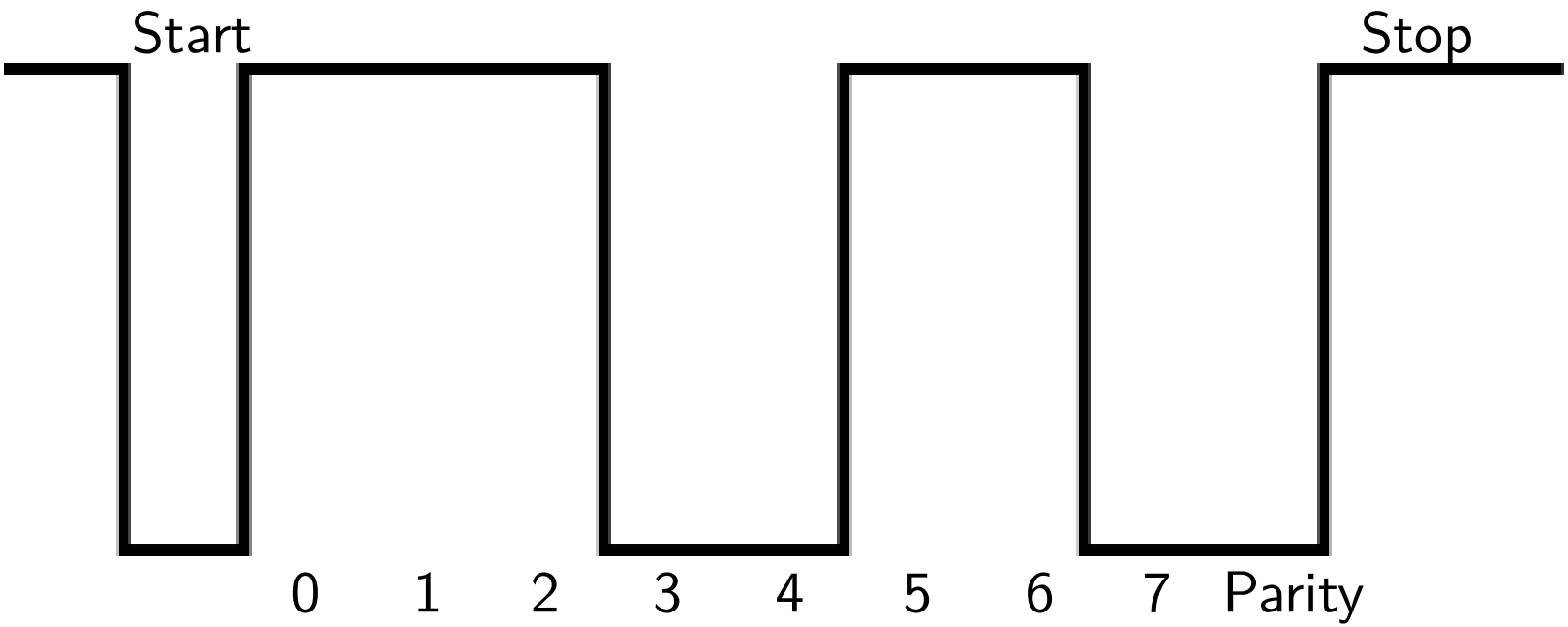

Start
1 1 1 0 0 1 1 0
LSB
MSB
0 1 2 3 4 5 6 7


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

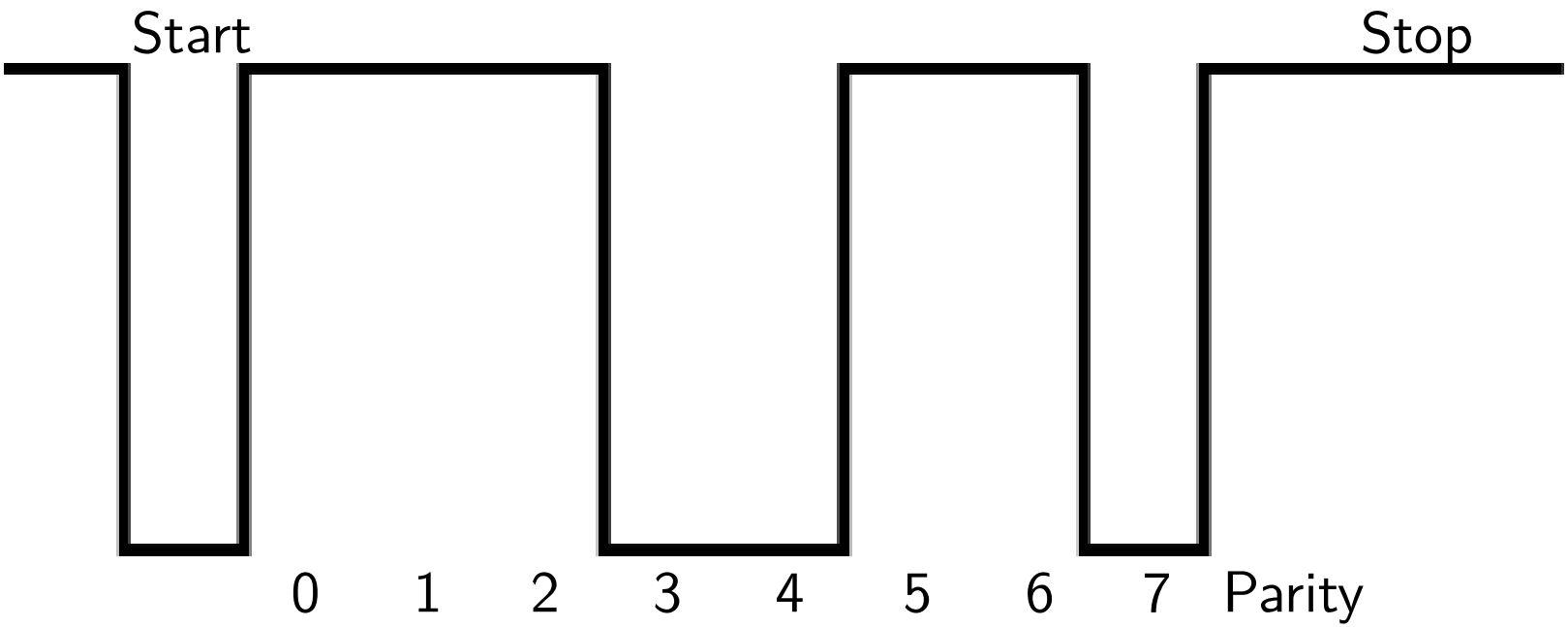


UART no parity - 01100111

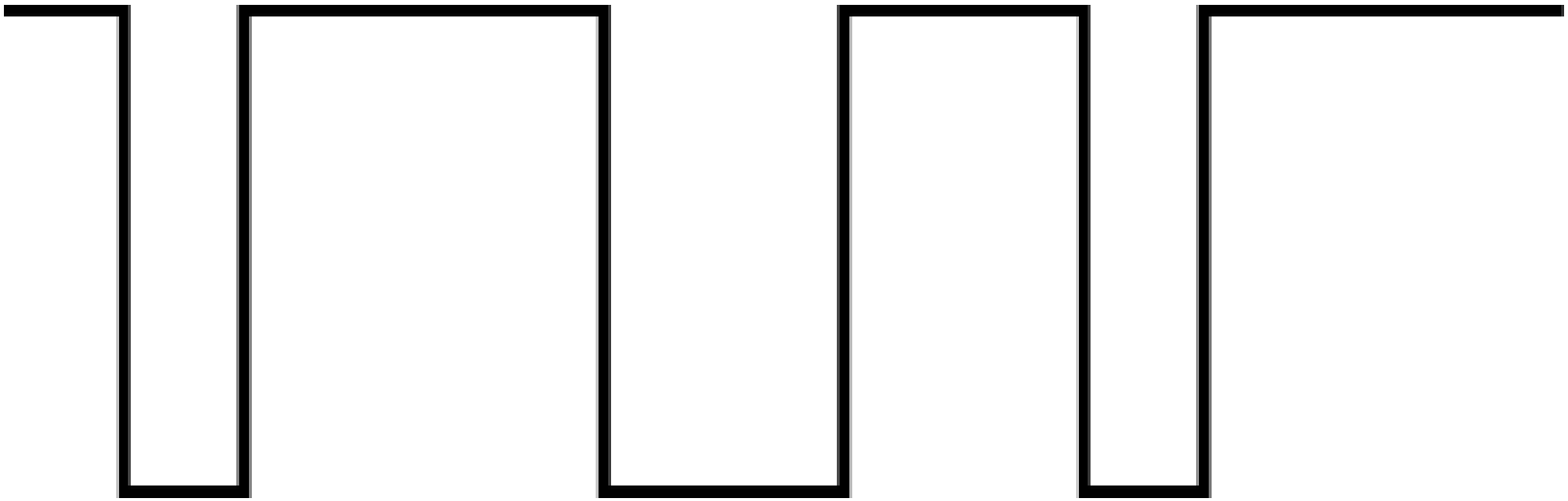

Start
Stop
0
1
2
3
4
5
6
7
Parity

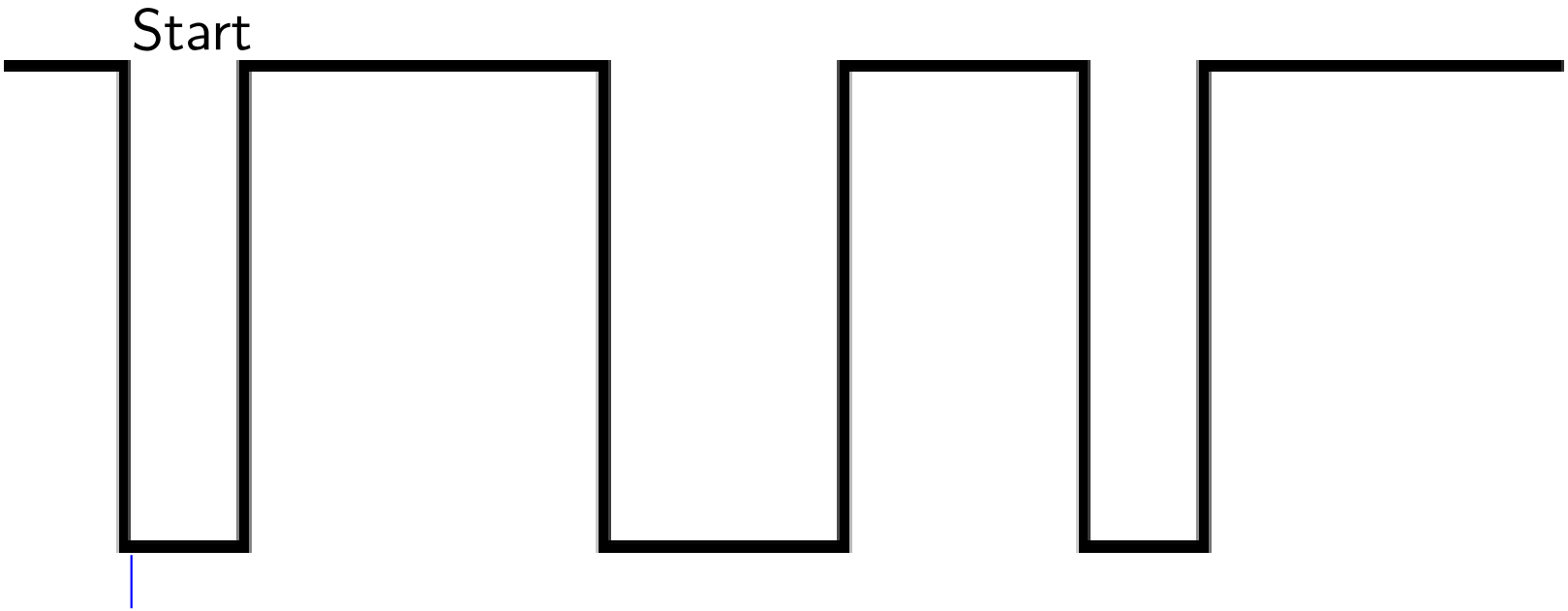


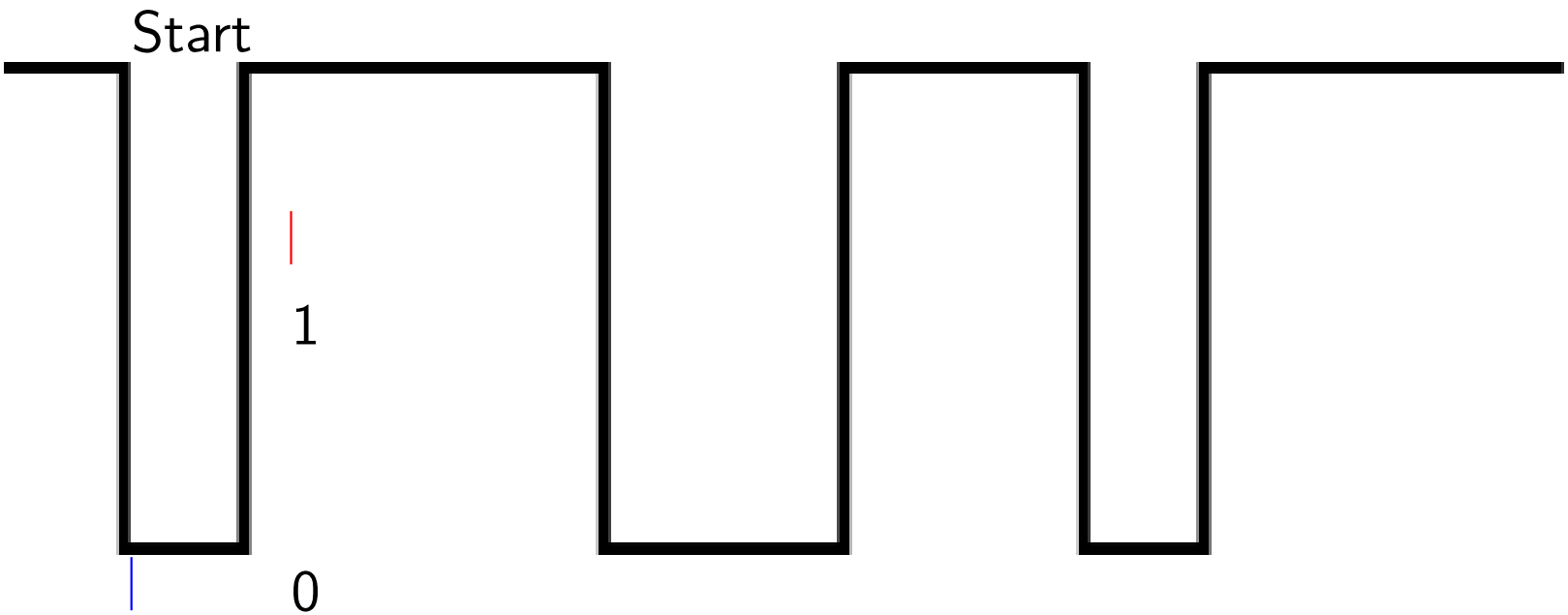
UART even parity

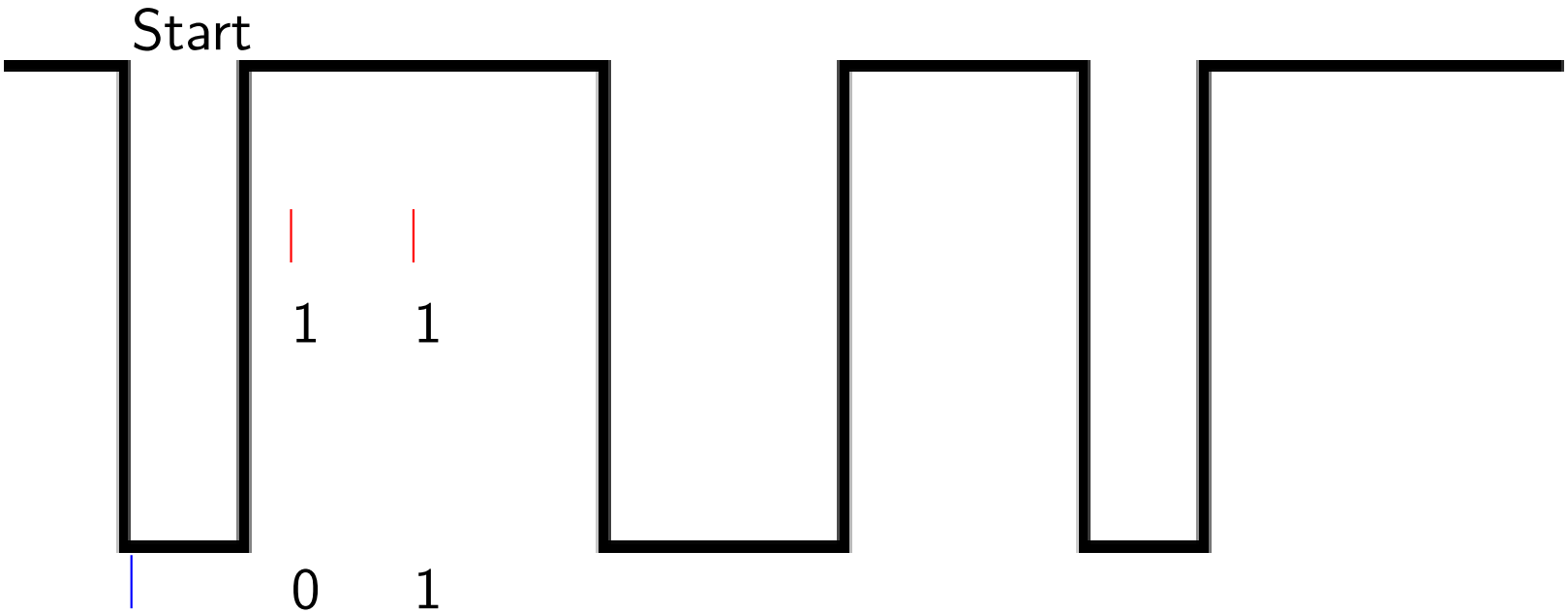


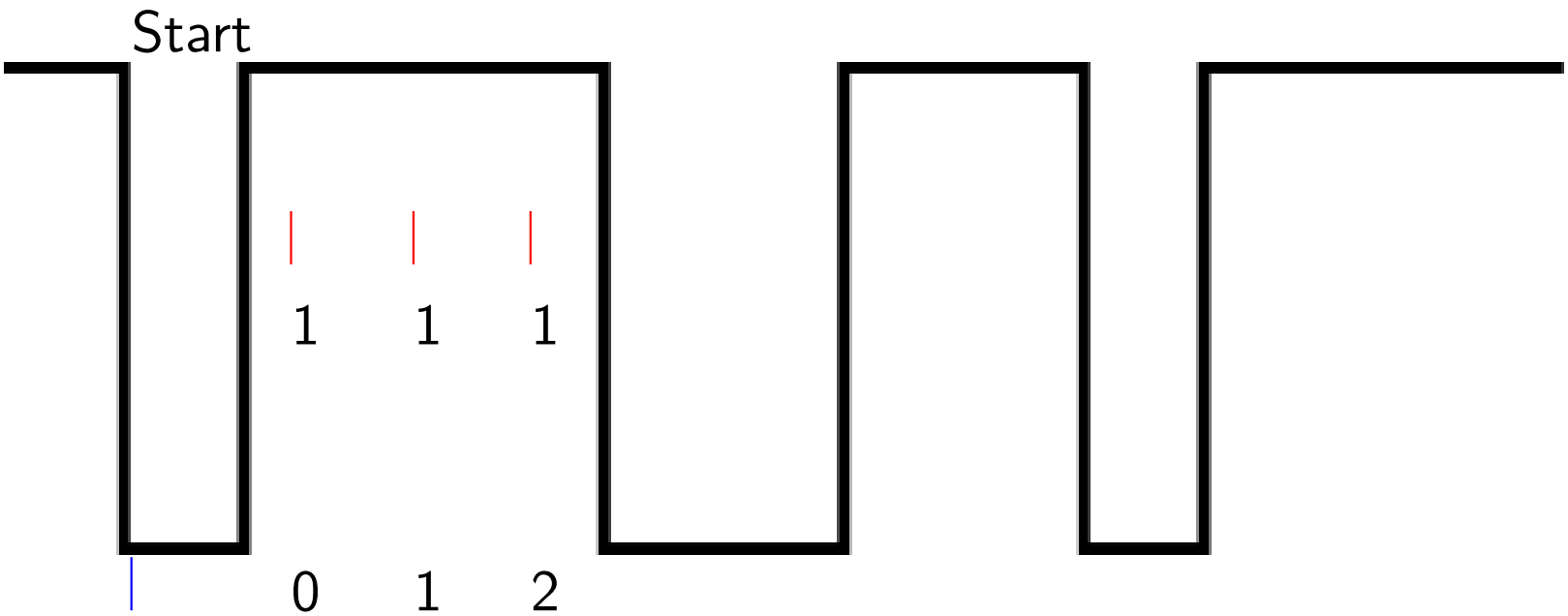
UART odd parity

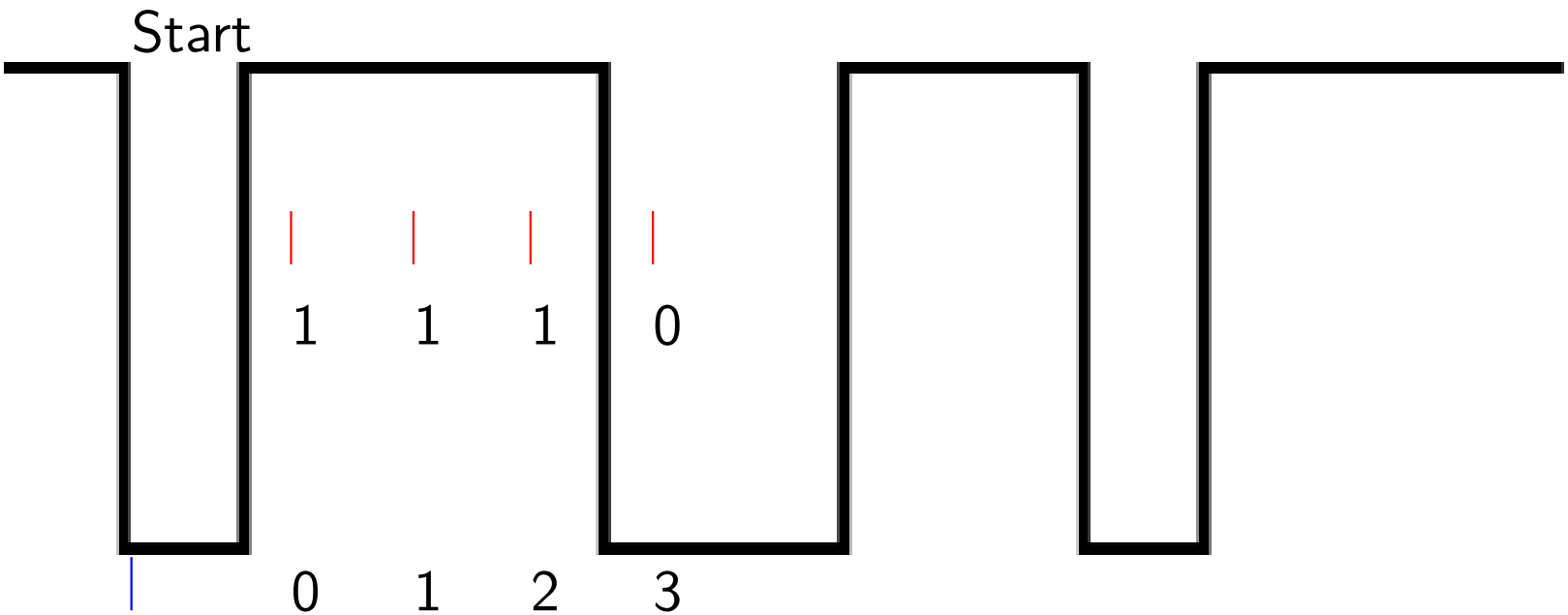


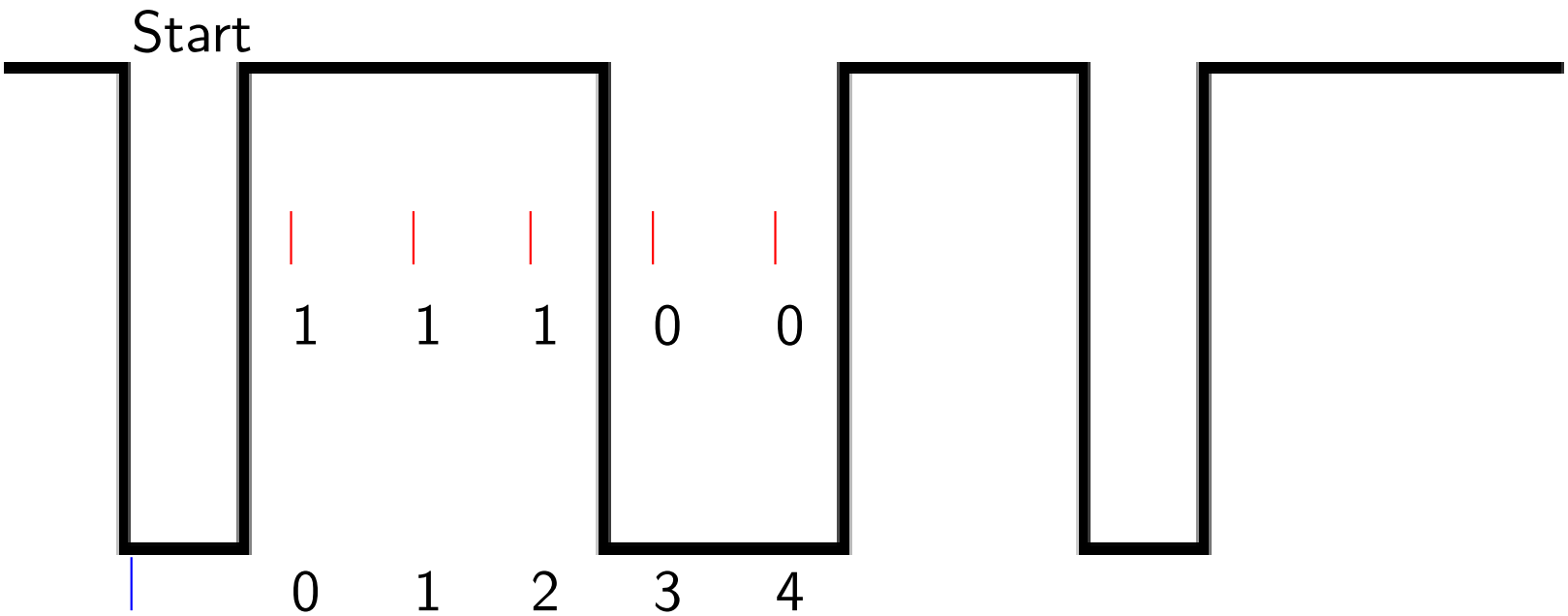

Start

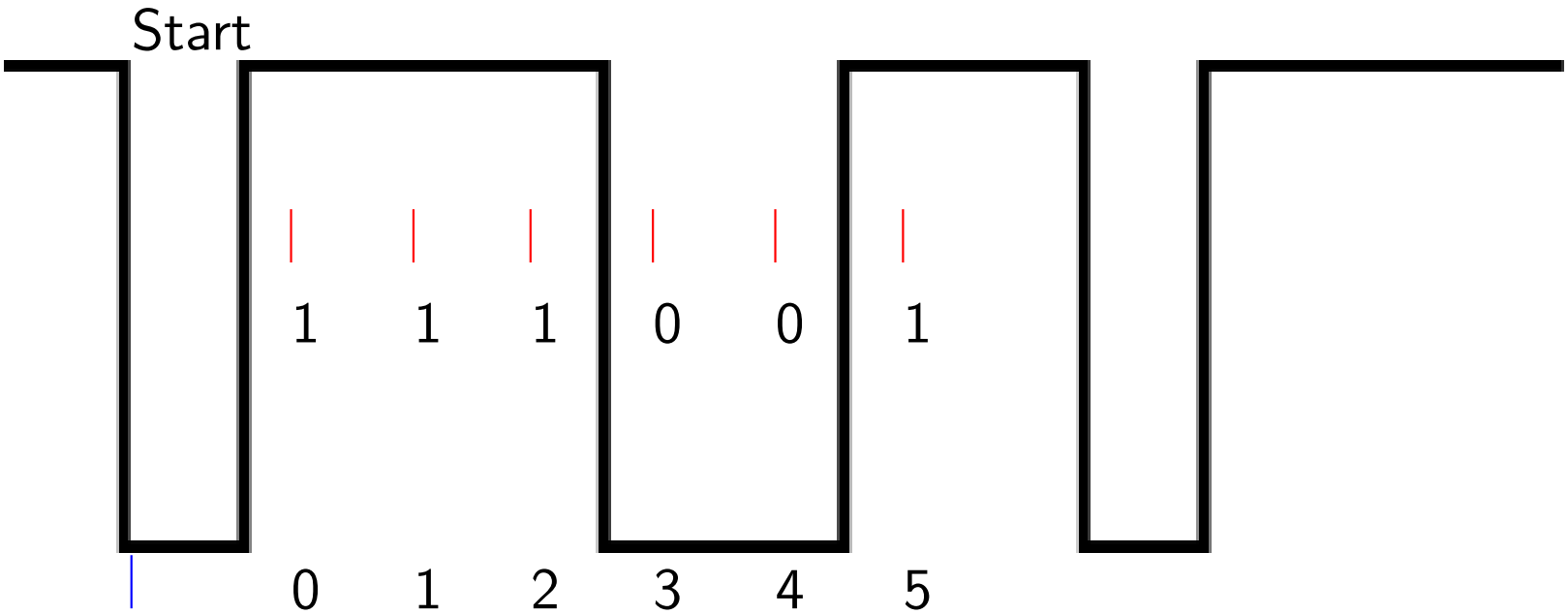

Start
1
0

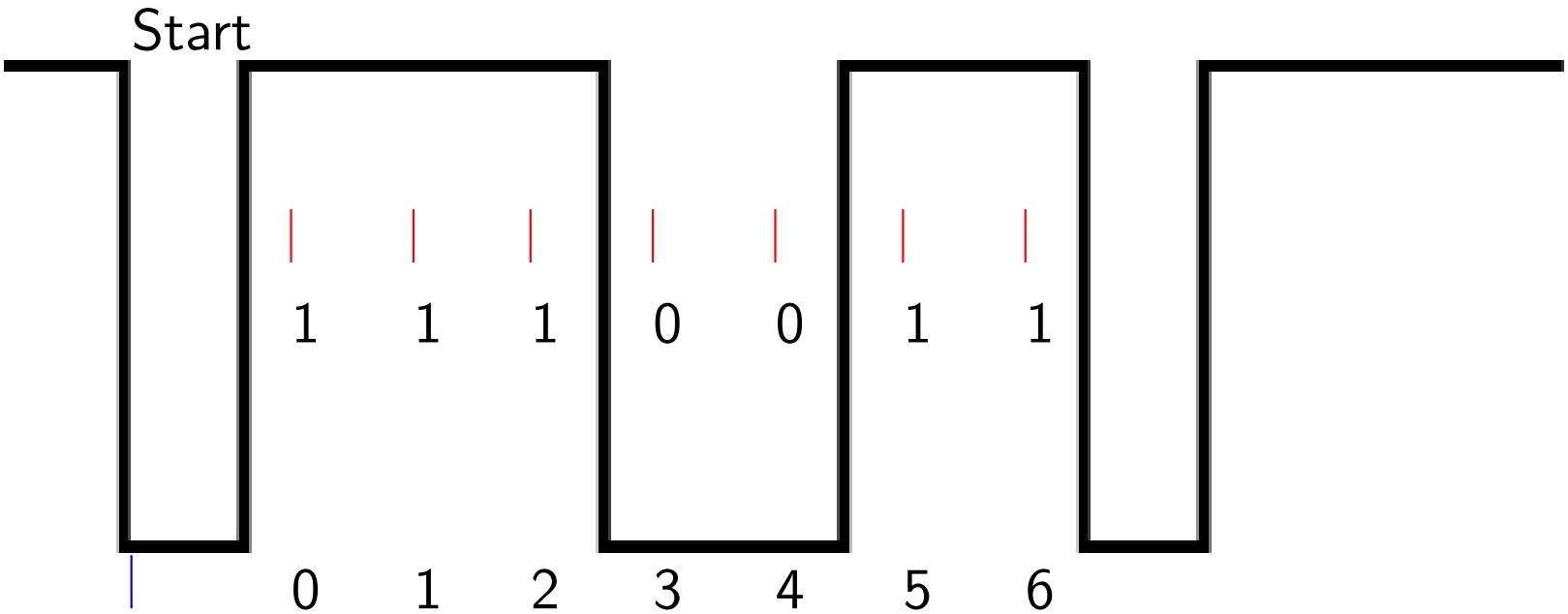

Start
1
1
0
1

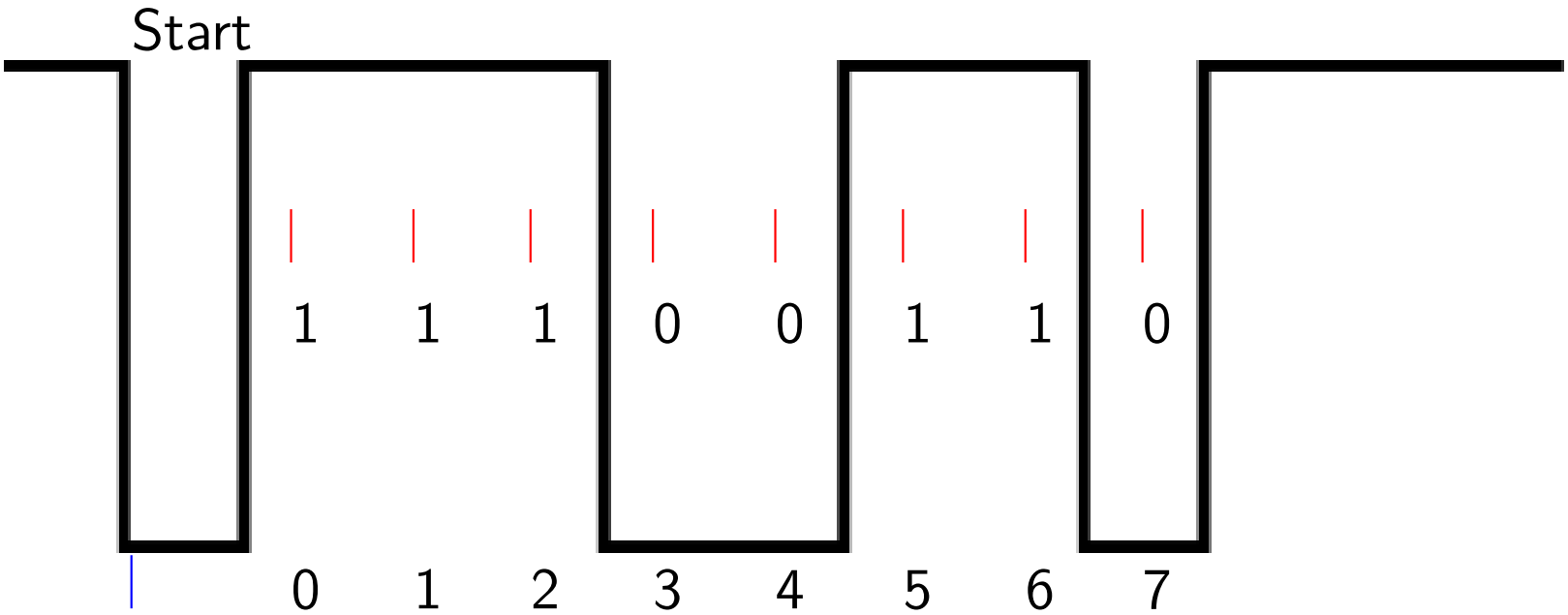

Start
1
1
1
0
1
2

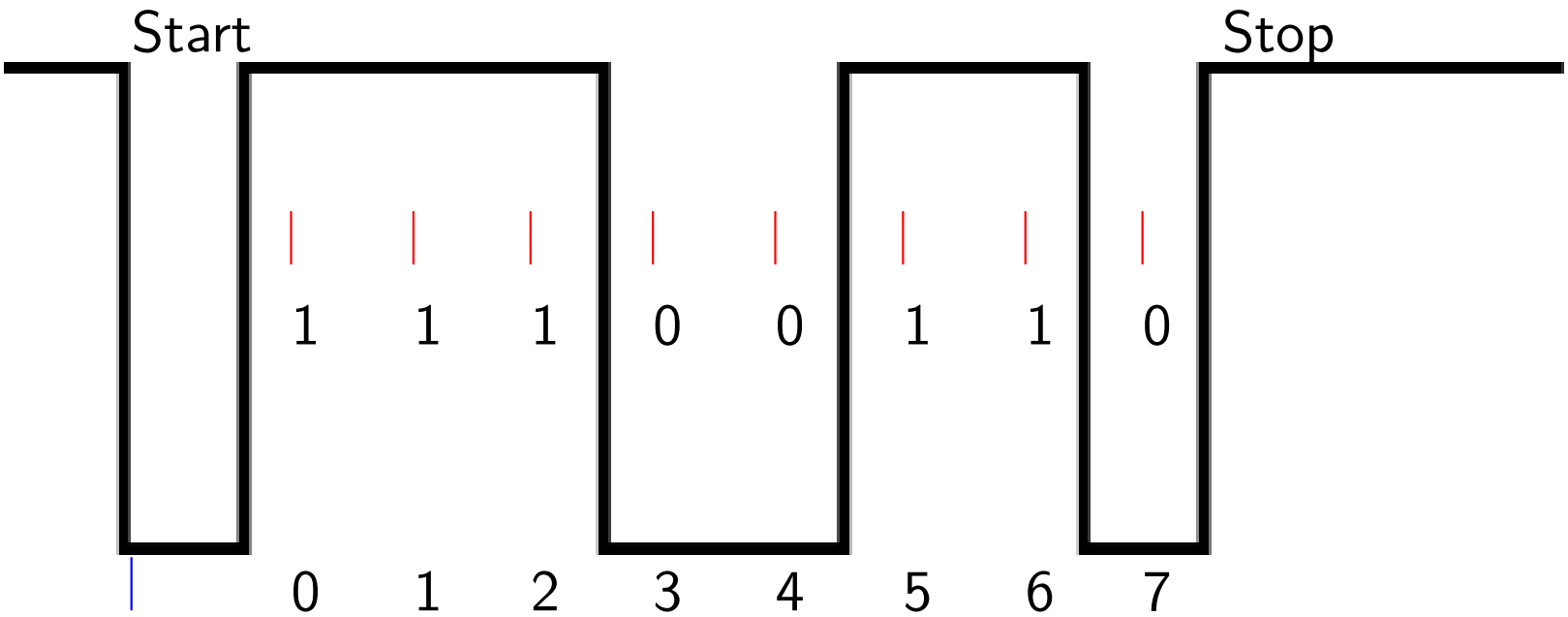

Start
1
1
1
0
0
1
2
3

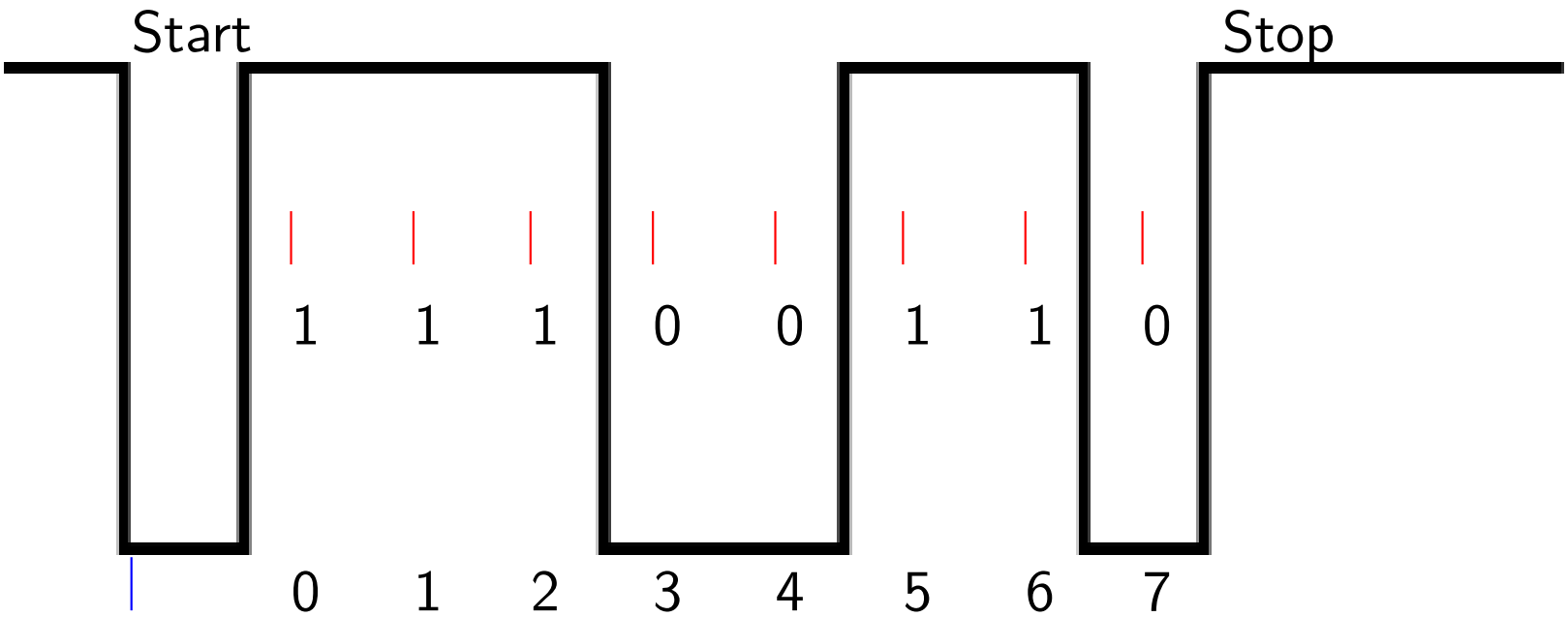

Start
1
1
1
0
0
0
1
2
3
4


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

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

Baud rate selection

Baud rate selection

Baud rate is determined by oscillator frequency,

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

and the state of the **BRGH** bit in **TXSTA**

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

and the state of the **BRGH** bit in **TXSTA**

If **BRGH** $= 0$,

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

and the state of the **BRGH** bit in **TXSTA**

If **BRGH** $= 0$,

$$\textbf{SPBRG} = \frac{F_{OSC}}{64 \times Desired\ \ Baud\ \ Rate} - 1$$

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

and the state of the **BRGH** bit in **TXSTA**

If **BRGH** $= 0$,

$$\textbf{SPBRG} = \frac{F_{OSC}}{64 \times Desired\ \ Baud\ \ Rate} - 1$$

For $F_{OSC} = 10MHz$,

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

and the state of the **BRGH** bit in **TXSTA**

If **BRGH** $= 0$,

$$\textbf{SPBRG} = \frac{F_{OSC}}{64 \times Desired\ \ Baud\ \ Rate} - 1$$

For $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{156250}{Desired\ \ Baud\ \ Rate} - 1$$

Baud rate selection

Baud rate is determined by oscillator frequency,

the value in the **SPBRG** register,

and the state of the **BRGH** bit in **TXSTA**

If **BRGH** $= 0$,

$$\textbf{SPBRG} = \frac{F_{OSC}}{64 \times Desired\ \ Baud\ \ Rate} - 1$$

For $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{156250}{Desired\ \ Baud\ \ Rate} - 1$$

The higher the value in **SPBRG**, the more precise the baud rate will be.

Baud rate selection (continued)

Baud rate selection (continued)

If **BRGH** =1,

Baud rate selection (continued)

If **BRGH** =1,

$$\textbf{SPBRG} = \frac{F_{OSC}}{16 \times Desired\ \ Baud\ \ Rate} - 1$$

Baud rate selection (continued)

If **BRGH** =1,

$$\textbf{SPBRG} = \frac{F_{OSC}}{16 \times Desired\ \ Baud\ \ Rate} - 1$$

For $F_{OSC} = 10MHz$,

Baud rate selection (continued)

If **BRGH** =1,

$$\textbf{SPBRG} = \frac{F_{OSC}}{16 \times Desired\ \ Baud\ \ Rate} - 1$$

For $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{625000}{Desired\ \ Baud\ \ Rate} - 1$$

Higher baud rates can be achieved with **BRGH** = 1

Baud rate selection (continued)

If **BRGH** =1,

$$\textbf{SPBRG} = \frac{F_{OSC}}{16 \times Desired\ \ Baud\ \ Rate} - 1$$

For $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{625000}{Desired\ \ Baud\ \ Rate} - 1$$

Higher baud rates can be achieved with **BRGH** = 1

If a given baud rate can be achieved with *either* value of **BRGH**, the one with **BRGH** =1 will give a more precise baud rate.

Sample baud rate calculations

Sample baud rate calculations

If **BRGH** $= 0$,

Sample baud rate calculations

If **BRGH** $= 0$,

and $F_{OSC} = 10MHz$,

Sample baud rate calculations

If **BRGH** $= 0$,

and $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{156250}{Desired\ \ Baud\ \ Rate} - 1$$

Sample baud rate calculations

If **BRGH** $= 0$,

and $F_{OSC} = 10MHz$,

SPBRG $= \frac{156250}{Desired\ \ Baud\ \ Rate} - 1$

so if $Desired\ \ Baud\ \ Rate = 9600$

Sample baud rate calculations

If **BRGH** $= 0$,

and $F_{OSC} = 10MHz$,

SPBRG $= \frac{156250}{Desired\ \ Baud\ \ Rate} - 1$

so if $Desired\ \ Baud\ \ Rate = 9600$

then **SPBRG** $= \frac{156250}{9600} - 1 = 16.27 - 1 \approx 15 = X'0F$

Sample baud rate calculations

Sample baud rate calculations

If **BRGH** $= 1$,

Sample baud rate calculations

If **BRGH** $= 1$,

and $F_{OSC} = 10MHz$,

Sample baud rate calculations

If **BRGH** $= 1$,

and $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{625000}{Desired\ \ Baud\ \ Rate} - 1$$

Sample baud rate calculations

If **BRGH** $= 1$,

and $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{625000}{Desired\ \ Baud\ \ Rate} - 1$$

so if $Desired\ \ Baud\ \ Rate = 9600$

Sample baud rate calculations

If **BRGH** $= 1$,

and $F_{OSC} = 10MHz$,

$$\textbf{SPBRG} = \frac{625000}{Desired\ \ Baud\ \ Rate} - 1$$

so if $Desired\ \ Baud\ \ Rate = 9600$

then $\textbf{SPBRG} = \frac{625000}{9600} - 1 = 65.1 - 1 \approx 64 = X'40$

RS232 communication

RS232 communication

- Voltages are inverted

RS232 communication

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

RS232 communication

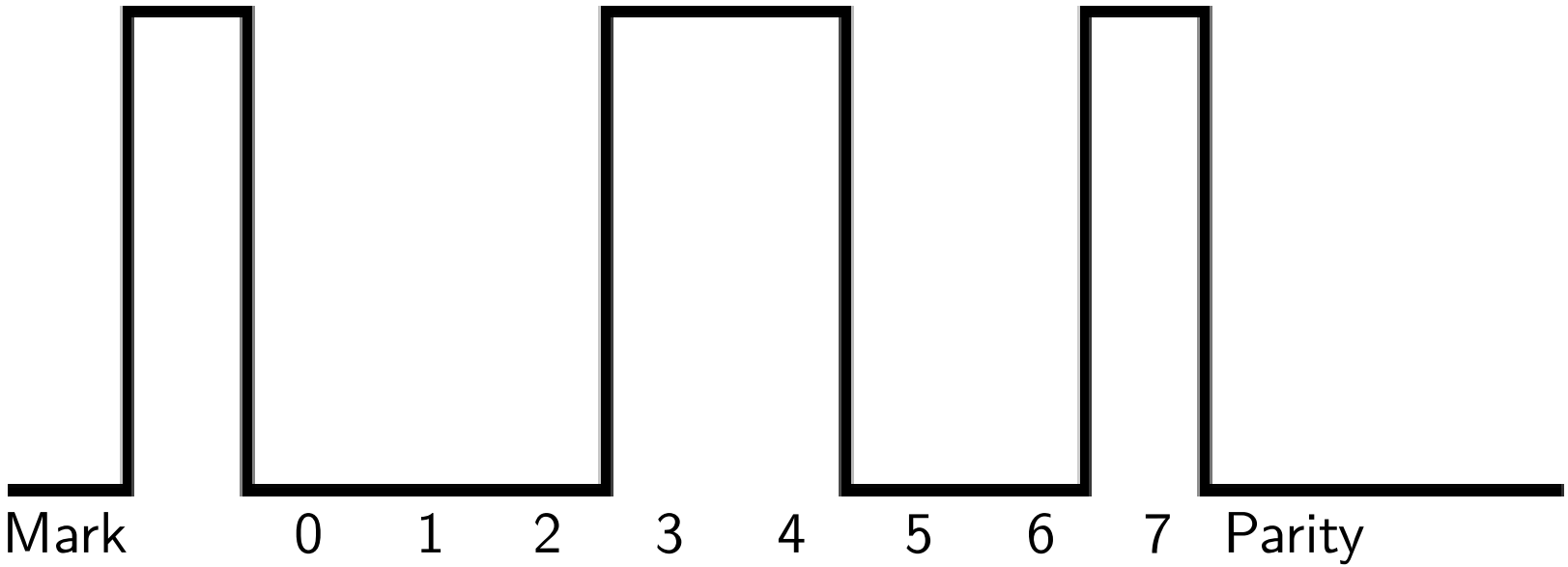
- Voltages are inverted
- $\pm 3 \rightarrow \pm 12$
- Zero is not a valid voltage

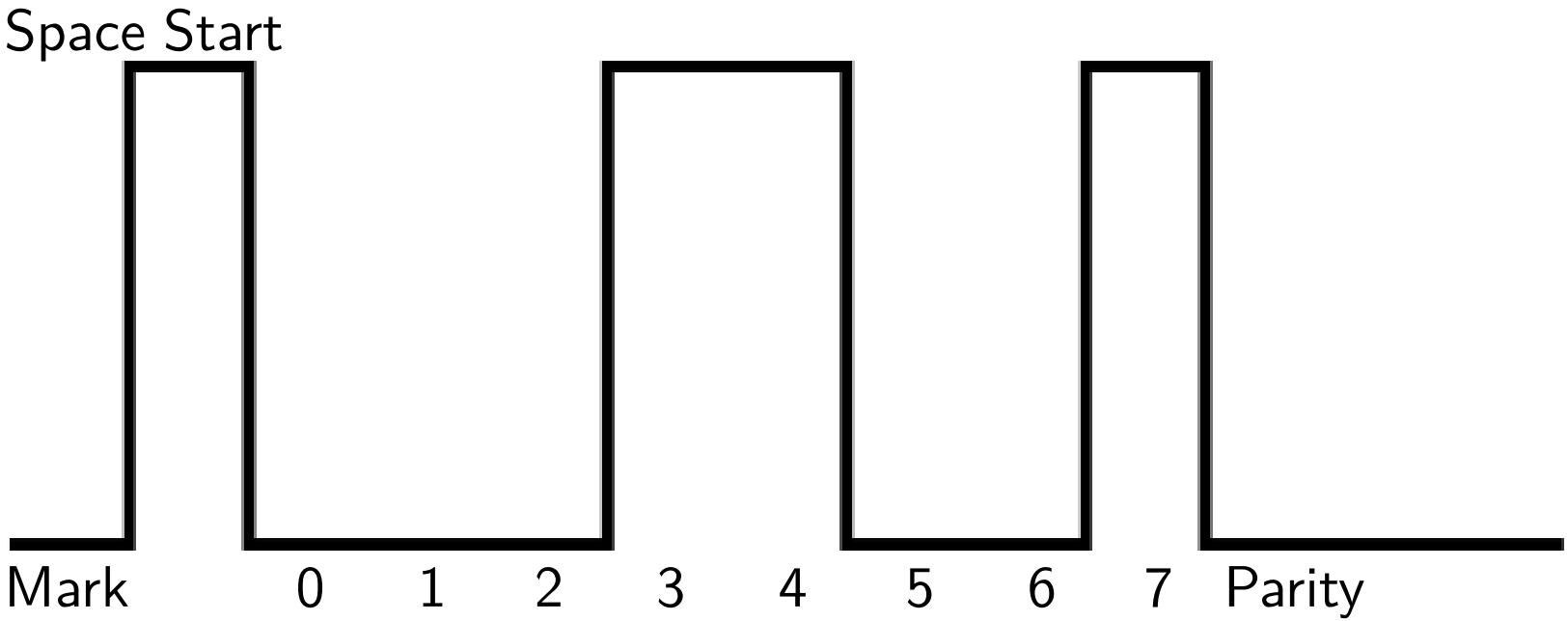
RS232 communication

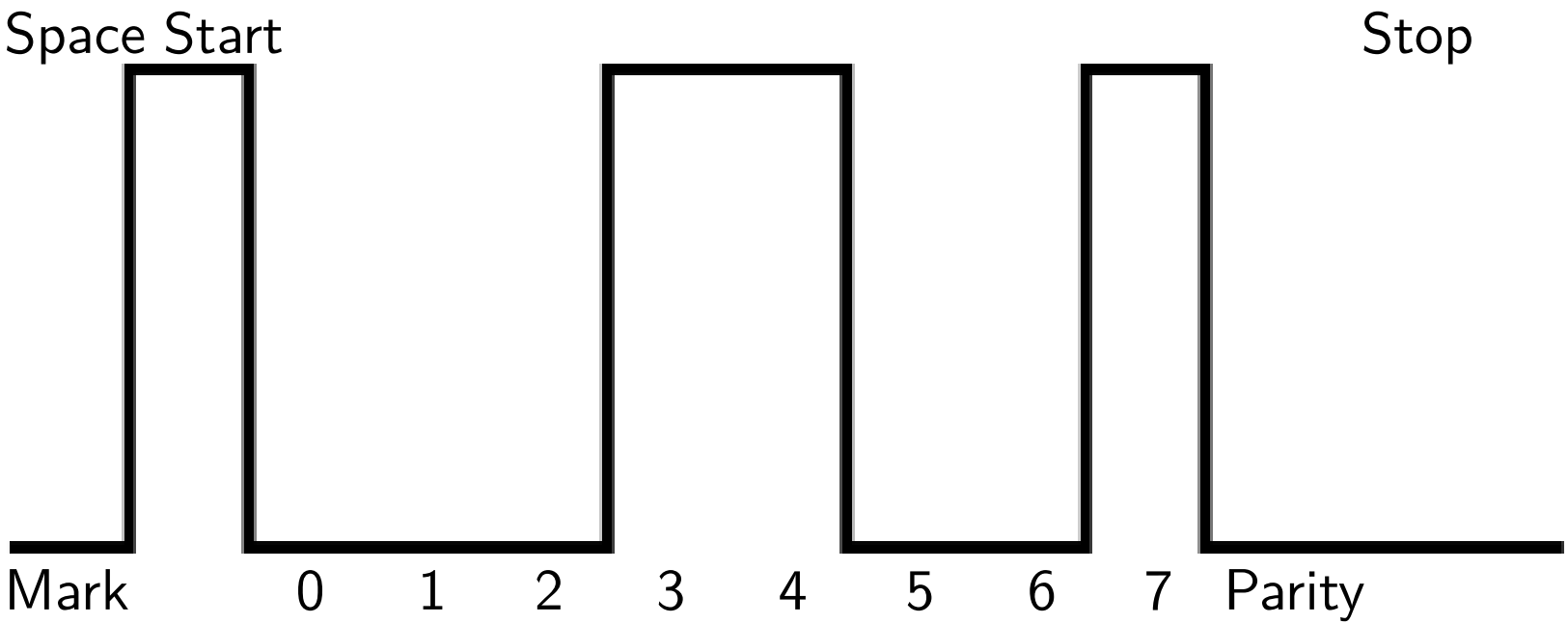
- Voltages are inverted
- $\pm 3 \rightarrow \pm 12$
- Zero is not a valid voltage
- Mark level (inactive/1) is a negative voltage

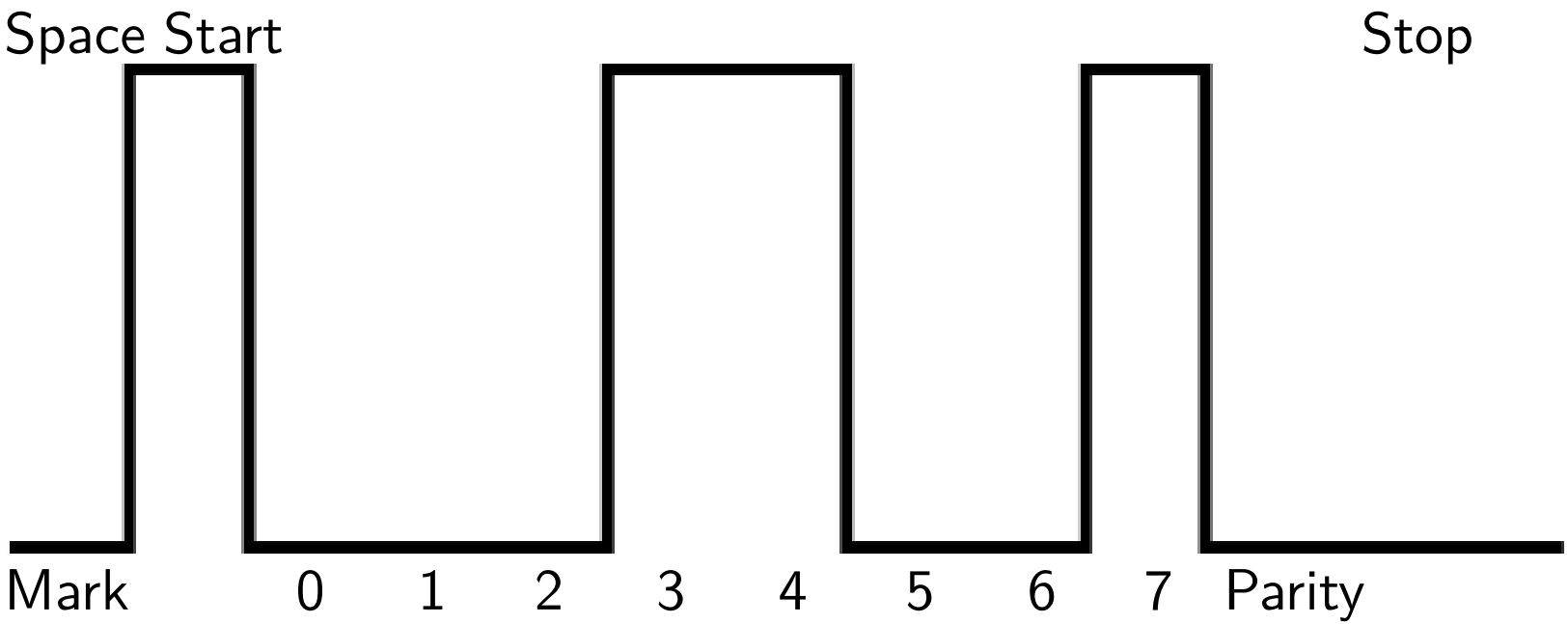
RS232 communication

- 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

PIC USART

PIC USART

asynchronous *and* synchronous modes

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

→ Section 9.4.1

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

→ Section 9.4.1

USART asynchronous mode

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

→ Section 9.4.1

USART asynchronous mode

→ Section 9.4.2

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

→ Section 9.4.1

USART asynchronous mode

→ Section 9.4.2

USART asynchronous mode to EIA232

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

→ Section 9.4.1

USART asynchronous mode

→ Section 9.4.2

USART asynchronous mode to EIA232

→ Section 9.4.5

PIC USART

asynchronous *and* synchronous modes

→ Section 9.3

USART registers

→ Section 9.4.1

USART asynchronous mode

→ Section 9.4.2

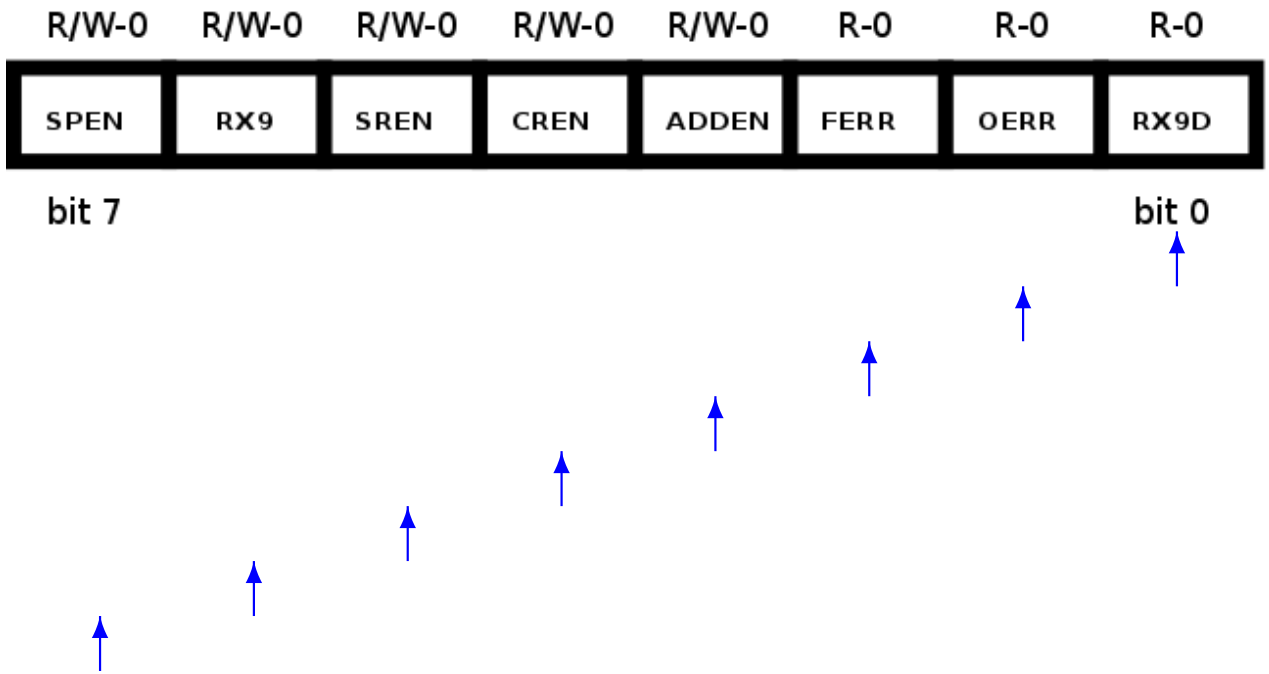
USART asynchronous mode to EIA232

→ Section 9.4.5

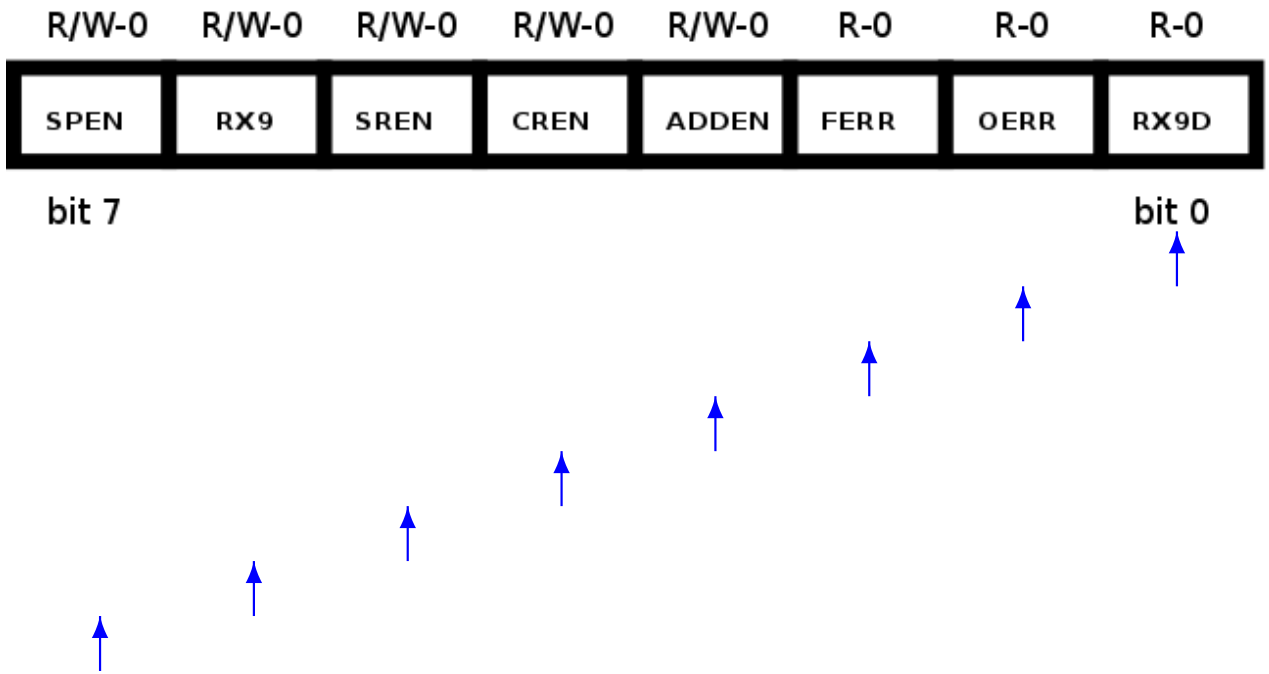
→ Section 16.0

RCSTA

RCSTA


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

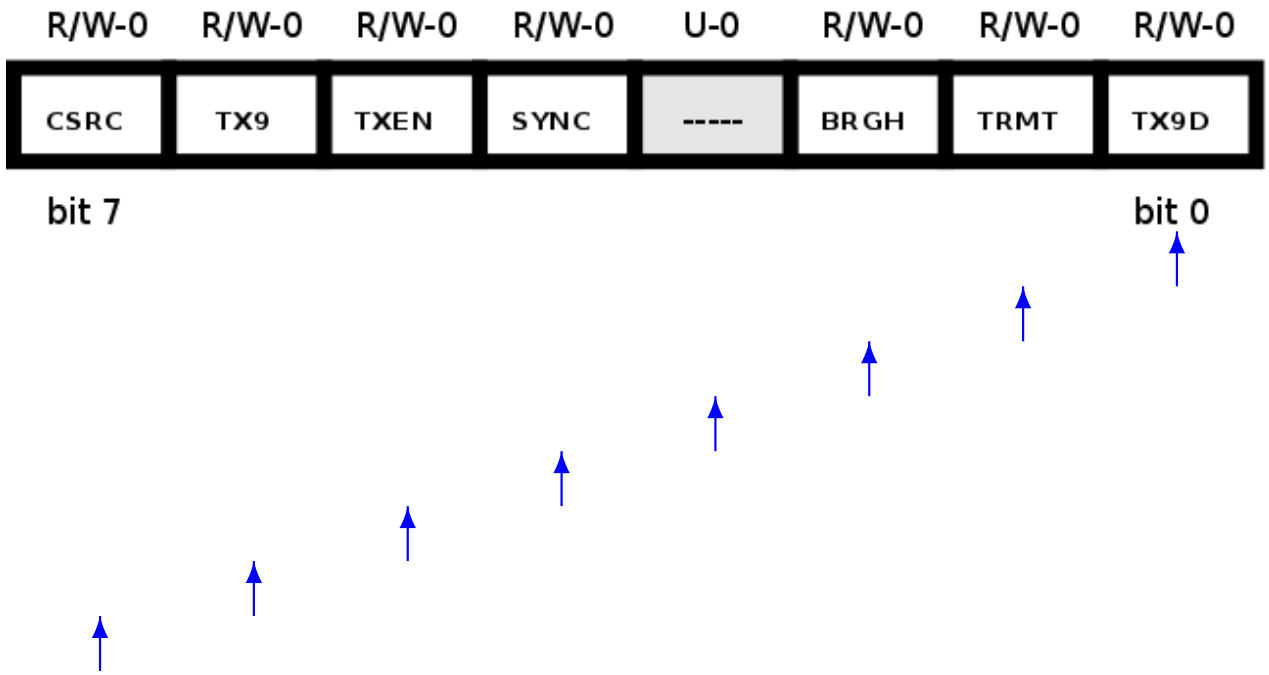
RCSTA



Bits in RCSTA register

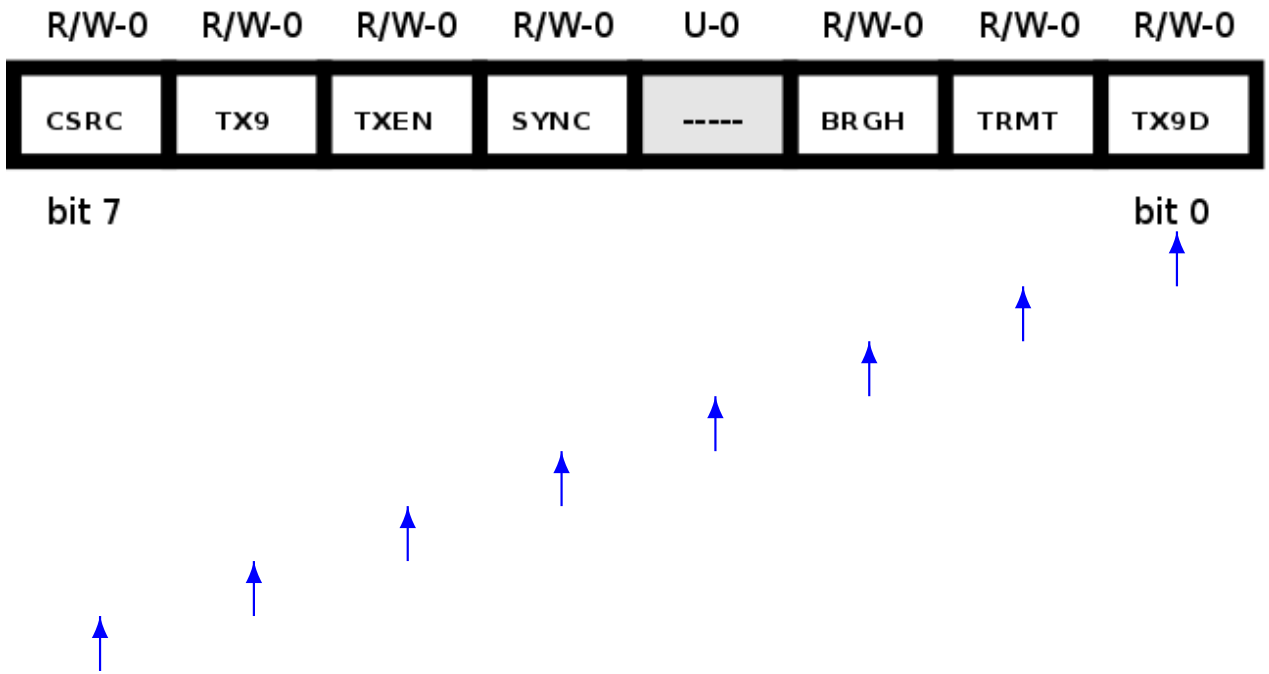
TXSTA

TXSTA


R/W-0
R/W-0
R/W-0
R/W-0
U-0
R/W-0
R/W-0
R/W-0
CSRC
TX9
TXEN
SYNC

BRGH
TRMT
TX9D
bit 7
bit 0

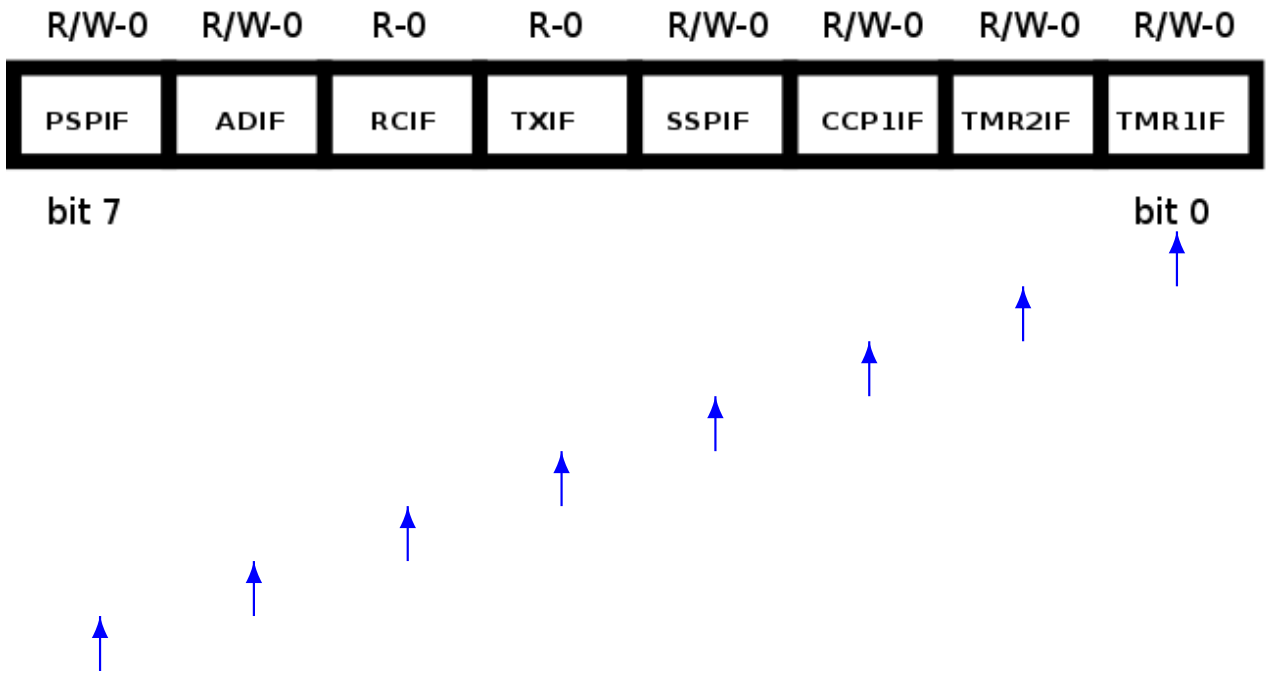
TXSTA



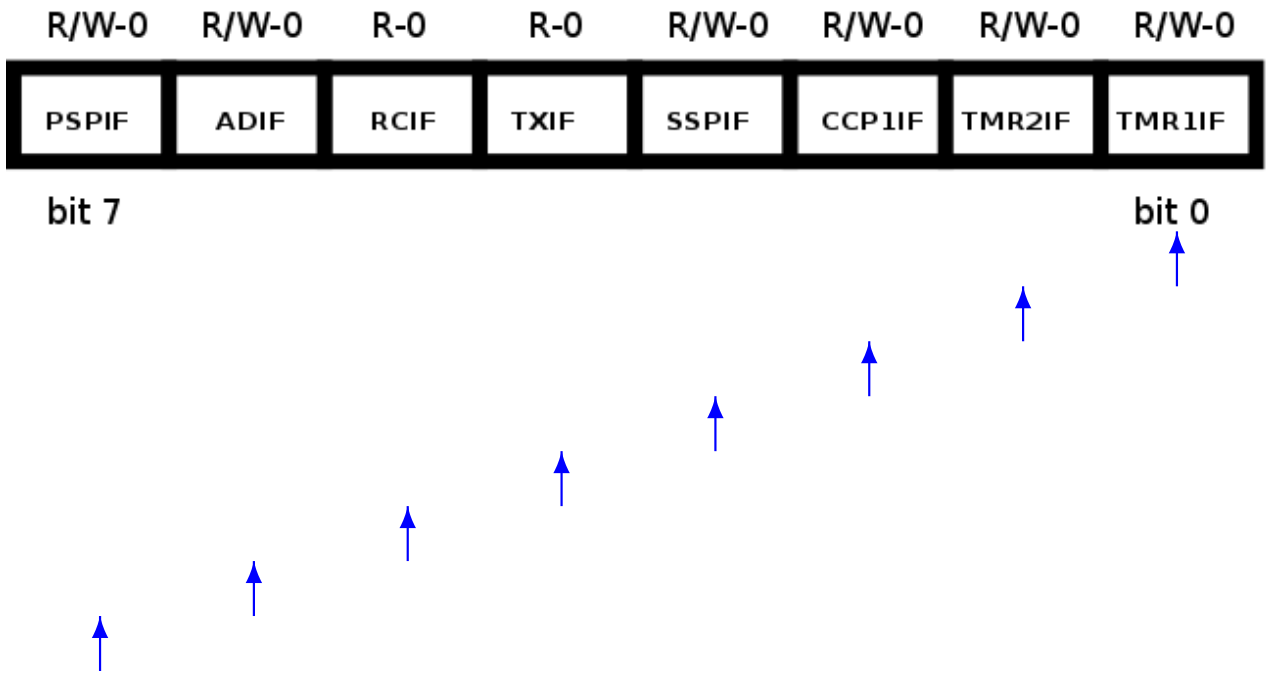
Bits in TXSTA register

PIR1

PIR1

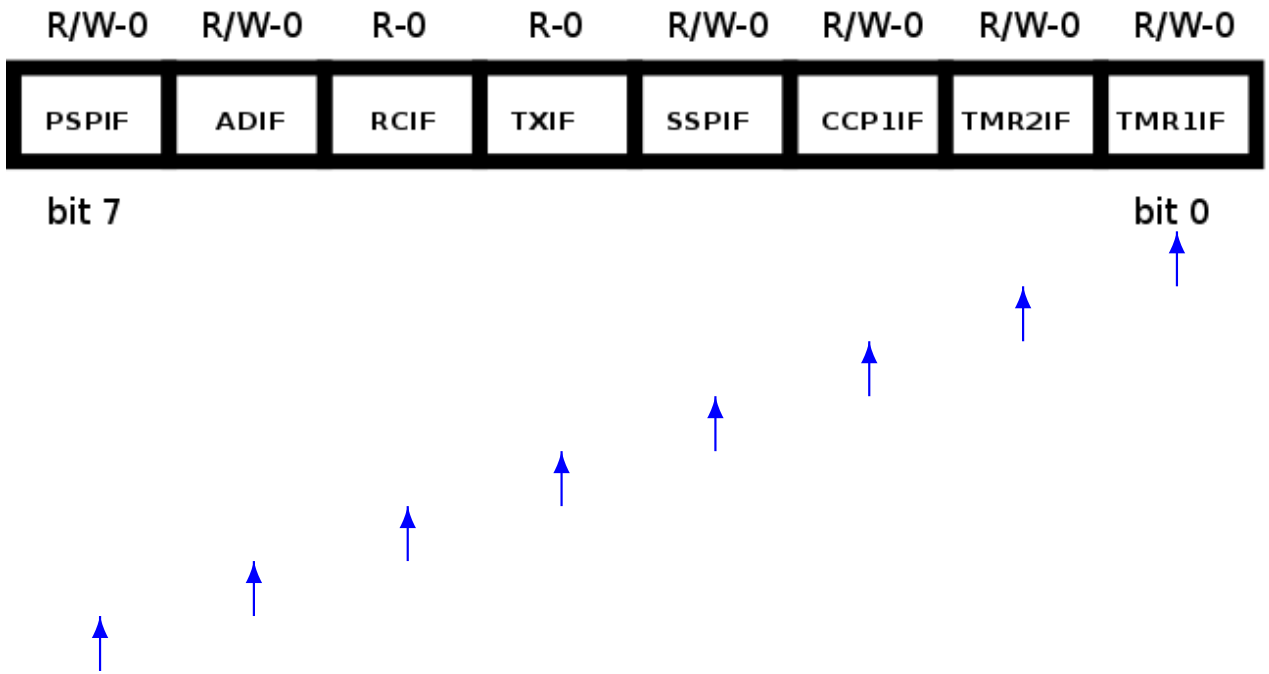

R/W-0
R/W-0
R-0
R-0
R/W-0
R/W-0
R/W-0
R/W-0
PSPIF
ADIF
RCIF
TXIF
SSPIF
CCP1IF
TMR2IF
TMR1IF
bit 7
bit 0

PIR1



Bits in PIR1 register

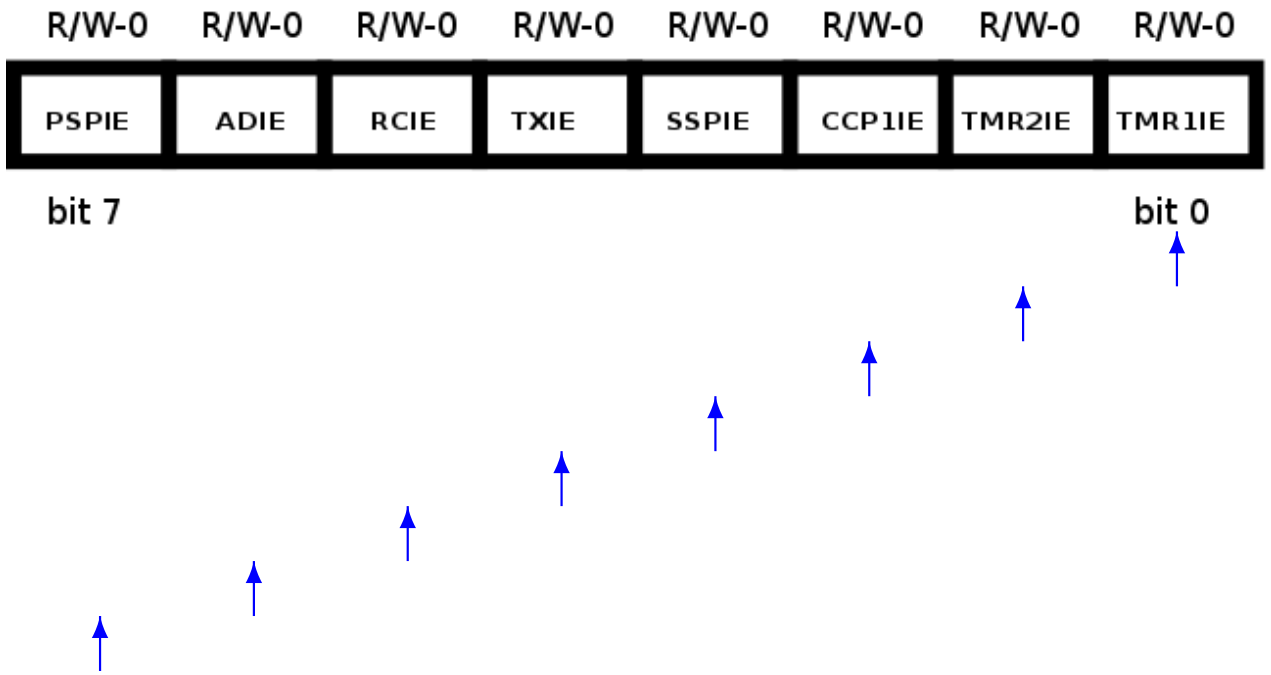
PIR1



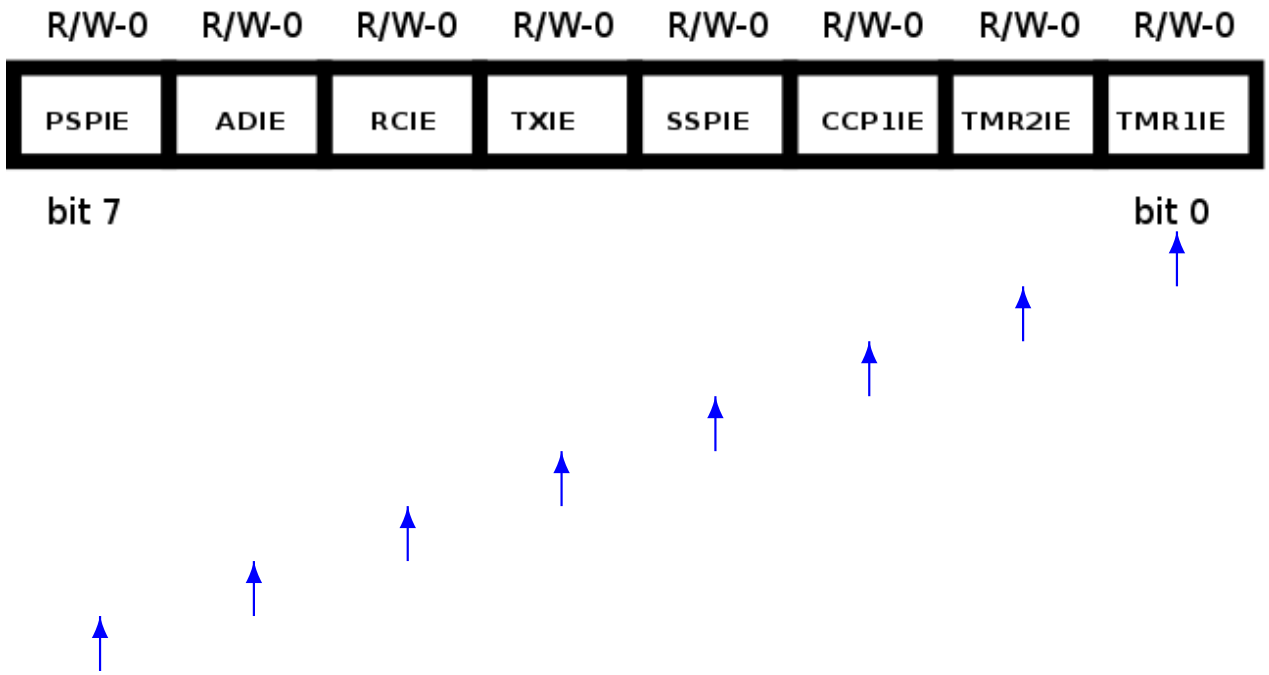
Bits in PIR1 register - Note RCIF, TXIF

PIE1

PIE1

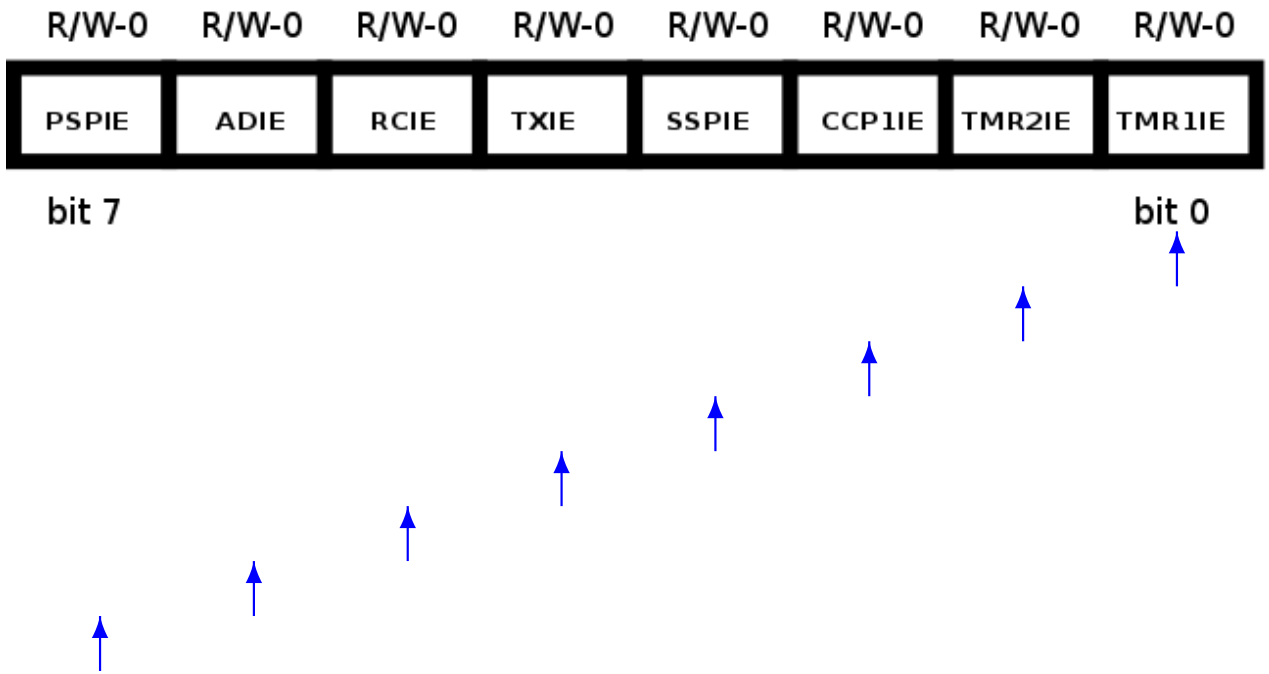

R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
R/W-0
PSPIE
ADIE
RCIE
TXIE
SSPIE
CCP1IE
TMR2IE
TMR1IE
bit 7
bit 0

PIE1



Bits in PIE1 register

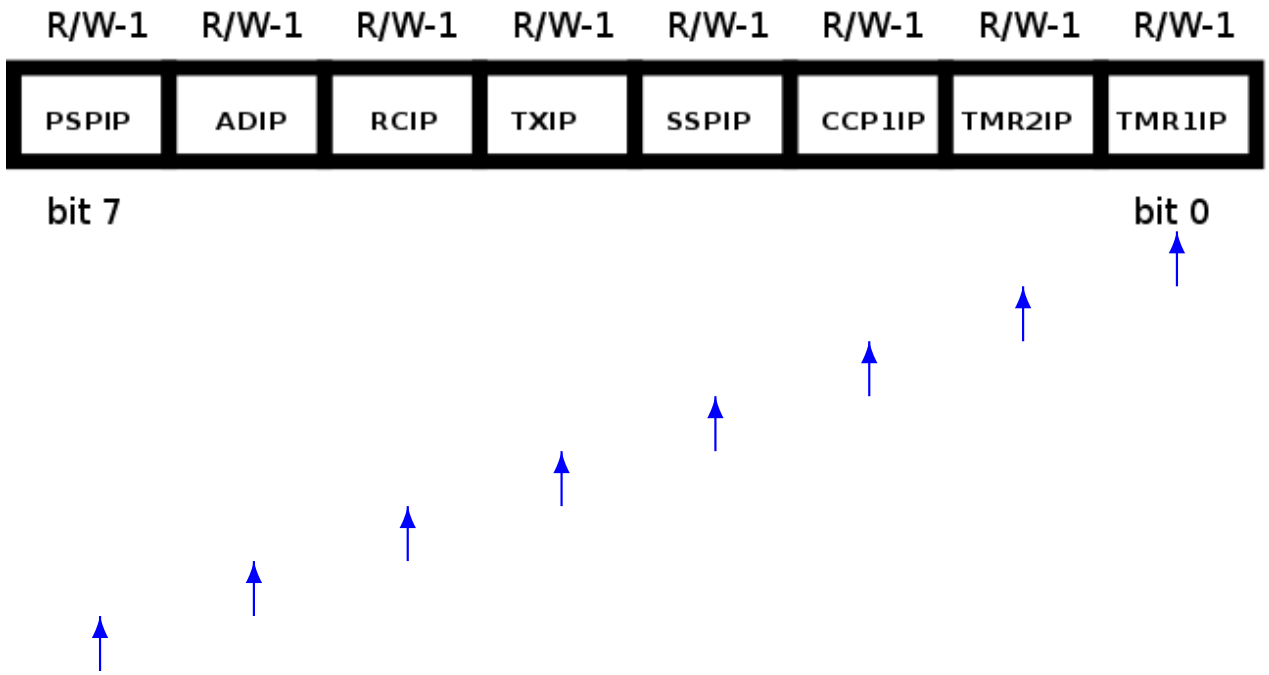
PIE1



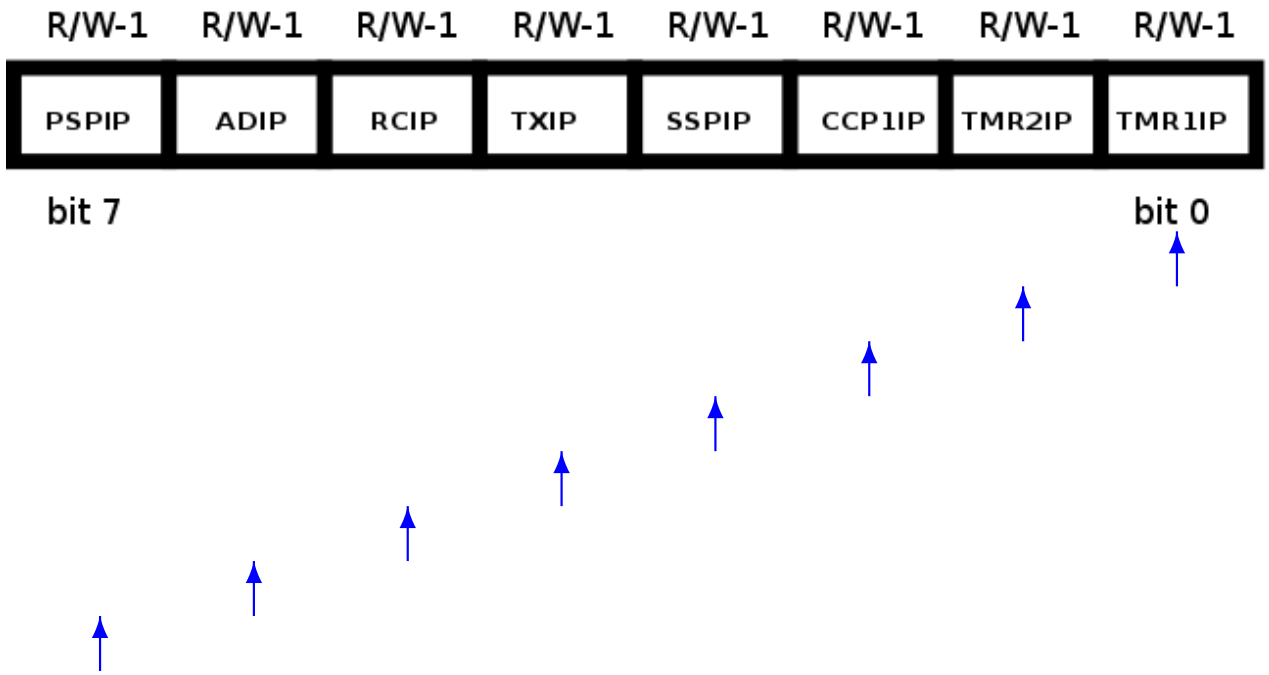
Bits in PIE1 register - Note RCIE, TXIE

IPR1

IPR1

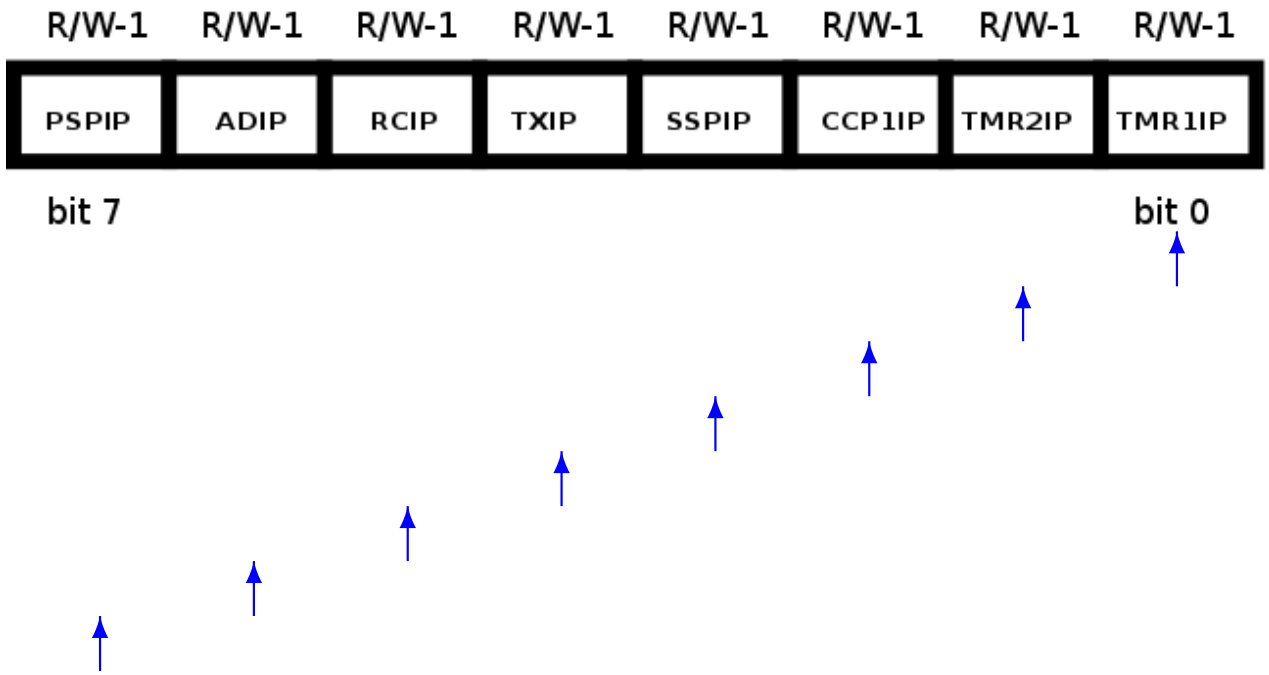

R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
R/W-1
PSPIP
ADIP
RCIP
TXIP
SSPIP
CCP1IP
TMR2IP
TMR1IP
bit 7
bit 0

IPR1



Bits in IPR1 register

IPR1



Bits in IPR1 register - Note RCIP, TXIP

QwikFlash connections

QwikFlash connections

QwikFlash modules

QwikFlash connections

QwikFlash modules

ramifications???

QwikFlash connections

QwikFlash modules

ramifications???

interrupts; transmit and receive

QwikFlash connections

QwikFlash modules

ramifications???

interrupts; transmit and receive

$\rightarrow$ Sections 6.4.5 to 6.4.7

QwikFlash connections

QwikFlash modules

ramifications???

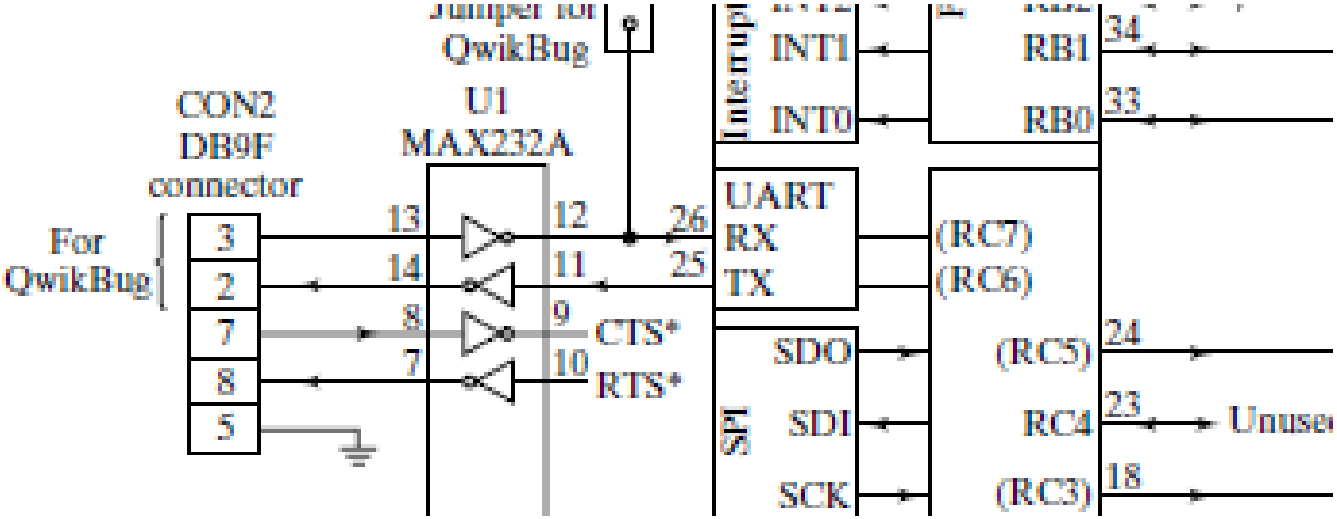
interrupts; transmit and receive

→ Sections 6.4.5 to 6.4.7

→ Section 8.2

QwikFlash UART connections

QwikFlash UART connections



EIA232 summary

EIA232 summary

2 wires (+ ground), one-to-one

EIA232 summary

2 wires (+ ground), one-to-one

TX

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

Fixed baud rate, common to both devices

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

Fixed baud rate, common to both devices

At least one *start* bit

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

Fixed baud rate, common to both devices

At least one *start* bit

At least one *stop* bit

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

Fixed baud rate, common to both devices

At least one *start* bit

At least one *stop* bit

voltage levels not TTL; inverted (normally)

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

Fixed baud rate, common to both devices

At least one *start* bit

At least one *stop* bit

voltage levels not TTL; inverted (normally)

(except "TTL serial" devices)

EIA232 summary

2 wires (+ ground), one-to-one

TX

RX

Fixed baud rate, common to both devices

At least one *start* bit

At least one *stop* bit

voltage levels not TTL; inverted (normally)

(except "TTL serial" devices)

packets are single characters

Bit-bashing

Bit-bashing

overview

Bit-bashing

overview

reasons

Bit-bashing

overview

reasons

NIB

Code

Code

PORT configuration

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

$\rightarrow$ macro or subroutine?

Code

PORT configuration

$\rightarrow$ macro or subroutine?

Initialization

$\rightarrow$ macro or subroutine?

Write to device

Code

PORT configuration

→ macro or subroutine?

Initialization

→ macro or subroutine?

Write to device

→ macro or subroutine?

Code

PORT configuration

→ macro or subroutine?

Initialization

→ macro or subroutine?

Write to device

→ macro or subroutine?

Read from device

Code

PORT configuration

→ macro or subroutine?

Initialization

→ macro or subroutine?

Write to device

→ macro or subroutine?

Read from device

→ macro or subroutine?

Initialization

Initialization

1. configure port (**TRISC**)

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits
2. set baud rate (**TXSTA** , **SPBRG**)

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits
2. set baud rate (**TXSTA** , **SPBRG**)

   high speed or low speed?

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits
2. set baud rate (**TXSTA** , **SPBRG**)

   high speed or low speed?
3. set other parameters (**TXSTA** , **RCSTA**)

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits
2. set baud rate (**TXSTA** , **SPBRG**)

   high speed or low speed?
3. set other parameters (**TXSTA** , **RCSTA**)

   data bits, asynchronous, etc.

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits
2. set baud rate (**TXSTA** , **SPBRG**)

   high speed or low speed?
3. set other parameters (**TXSTA** , **RCSTA**)

   data bits, asynchronous, etc.
4. configure interrupts (if desired) (**IPR1**)

Initialization

1. configure port (**TRISC**)

   don't inadvertently alter other bits
2. set baud rate (**TXSTA** , **SPBRG**)

   high speed or low speed?
3. set other parameters (**TXSTA** , **RCSTA**)

   data bits, asynchronous, etc.
4. configure interrupts (if desired) (**IPR1**)

   for each individual source required

Sending

Sending

Sending is simple

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)
3. enable transmit interrupt (if using) (**PIE1**)

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)
3. enable transmit interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)
3. enable transmit interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
4. wait for flag to see that buffer is empty (**TXSTA**)

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)
3. enable transmit interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
4. wait for flag to see that buffer is empty (**TXSTA**)

   (if not using interrupts)

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)

   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)
3. enable transmit interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
4. wait for flag to see that buffer is empty (**TXSTA**)

   (if not using interrupts)
5. when no more characters to send,

Sending

Sending is simple

1. check flag to see that buffer is empty (**TXSTA**)
   otherwise there is a character *being* transmitted already
2. place value in buffer (**TXREG**)
3. enable transmit interrupt (if using) (**PIE1**)
   assuming interrupts are enabled globally
4. wait for flag to see that buffer is empty (**TXSTA**)
   (if not using interrupts)
5. when no more characters to send,
   disable transmit interrupt (if using) (**PIE1**)

Receiving

Receiving

Receiving is also simple

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)

   otherwise there is no character yet

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)

   otherwise there is no character yet
2. enable receive interrupt (if using) (**PIE1**)

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)
   otherwise there is no character yet
2. enable receive interrupt (if using) (**PIE1**)
   assuming interrupts are enabled globally

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)

   otherwise there is no character yet
2. enable receive interrupt (if using) (**PIE1**)

   assuming interrupts are enabled globally
3. wait for flag to see that buffer is full (**RCSTA**)

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)
   otherwise there is no character yet
2. enable receive interrupt (if using) (**PIE1**)
   assuming interrupts are enabled globally
3. wait for flag to see that buffer is full (**RCSTA**)
   (if not using interrupts)

Receiving

Receiving is also simple

1. check flag to see that buffer is full (**RCSTA**)
   otherwise there is no character yet
2. enable receive interrupt (if using) (**PIE1**)
   assuming interrupts are enabled globally
3. wait for flag to see that buffer is full (**RCSTA**)
   (if not using interrupts)
4. get value from buffer (**RCREG**)