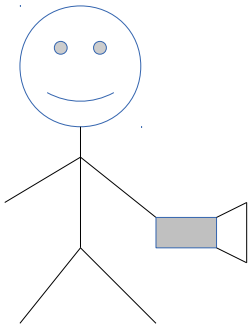
Serial communications

Serial communications

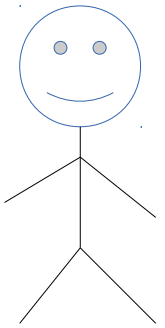
- Sending data between two computer systems
  - Options:
    - 1 wire (channel) per bit = Parallel data transmission
    - Sequential transmission of each bit on a single wire (channel) = Serial data transmission.

Serial communications

Example: Serial communications using a torch.



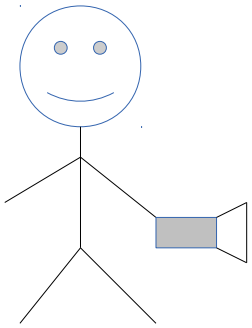
Transmitter



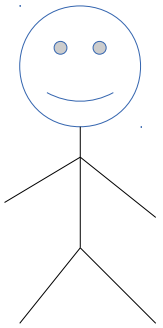
Receiver

Serial communications

Transmitter wants to send the value 244



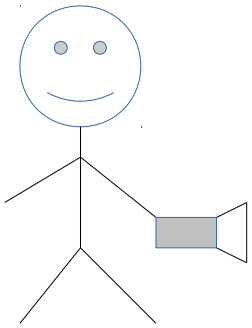
Transmitter



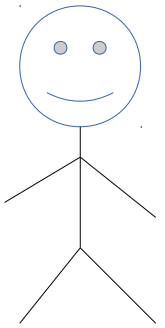
Receiver

Serial communications

Encode in 8 bit binary: 11110100



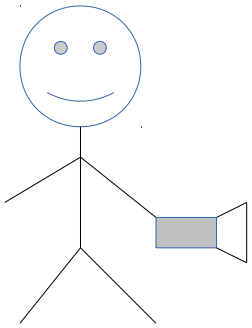
Transmitter



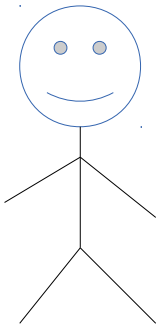
Receiver

Serial communications

Encode in 8 bit binary: 11110100



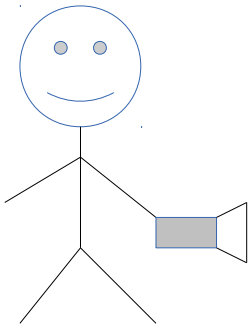
Transmitter



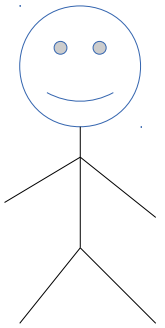
Receiver

Serial communications

How can receiver decode this?



Transmitter



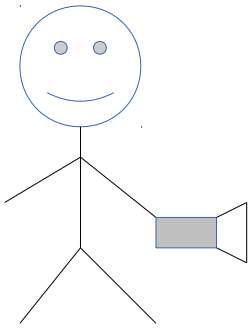
Receiver

Serial communications

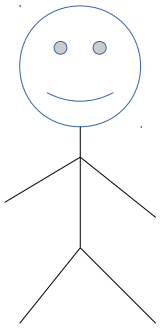
- When does the transmission start?
- When does each bit start?
- How many bits should be expected?
- LSB or MSB first?
- Was it transmitted without error?

Serial communications

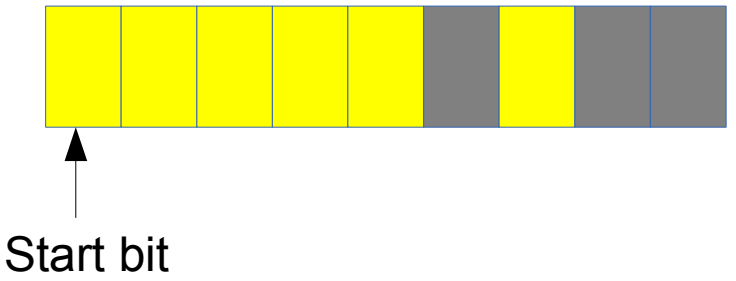
Detecting start : Send an extra “Start bit”



Transmitter

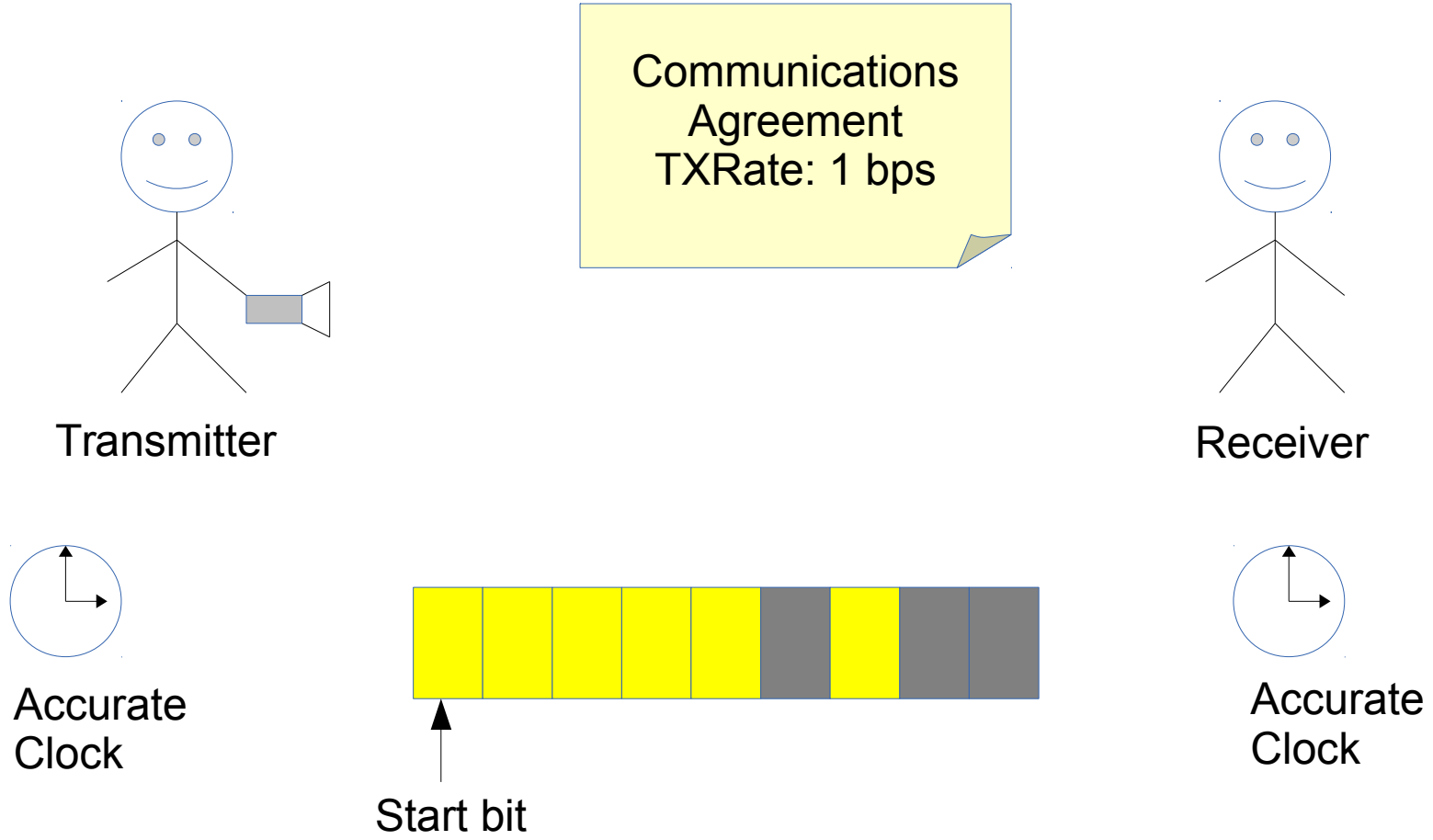


Receiver



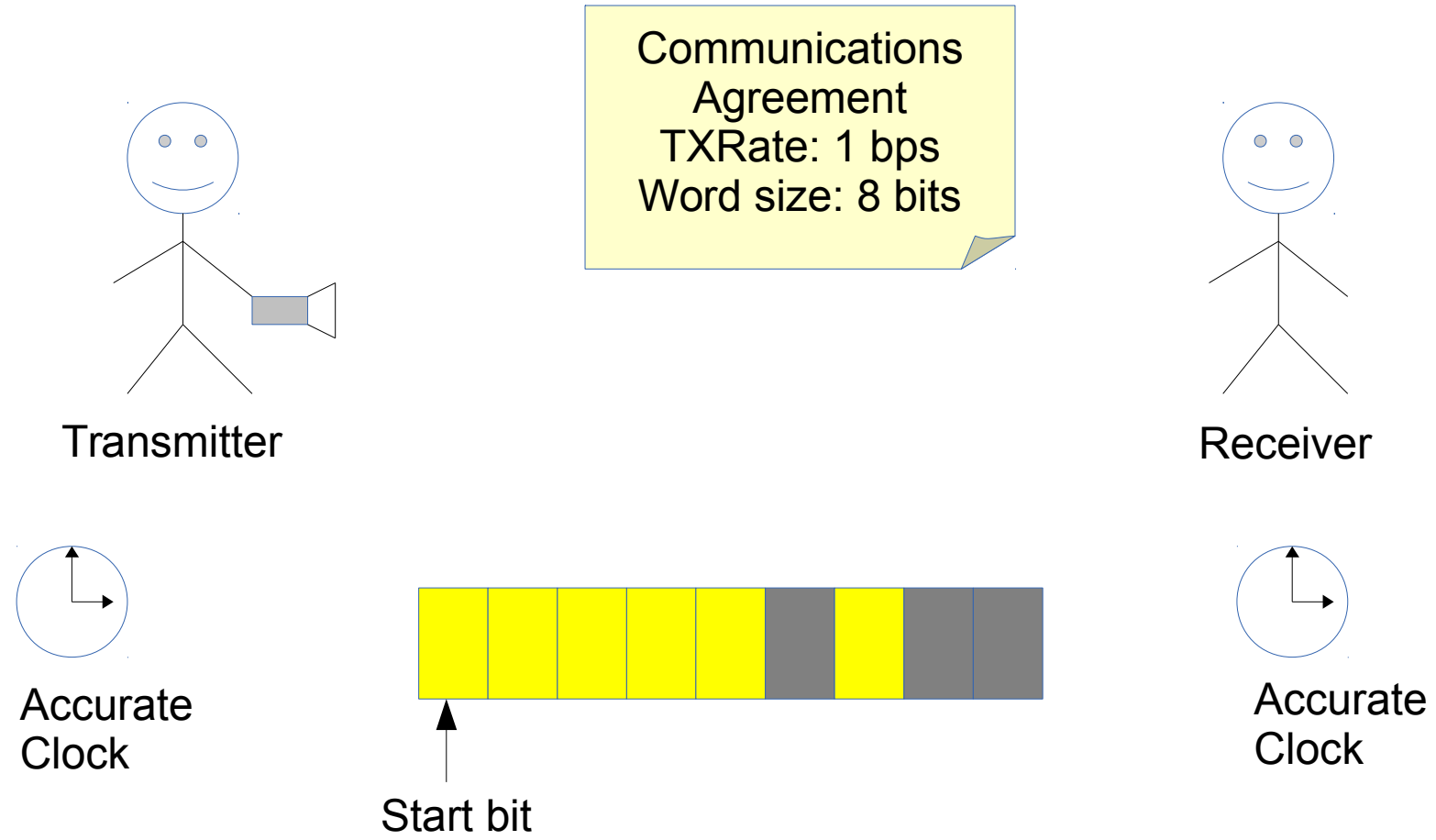
Serial communications

How long is each bit?



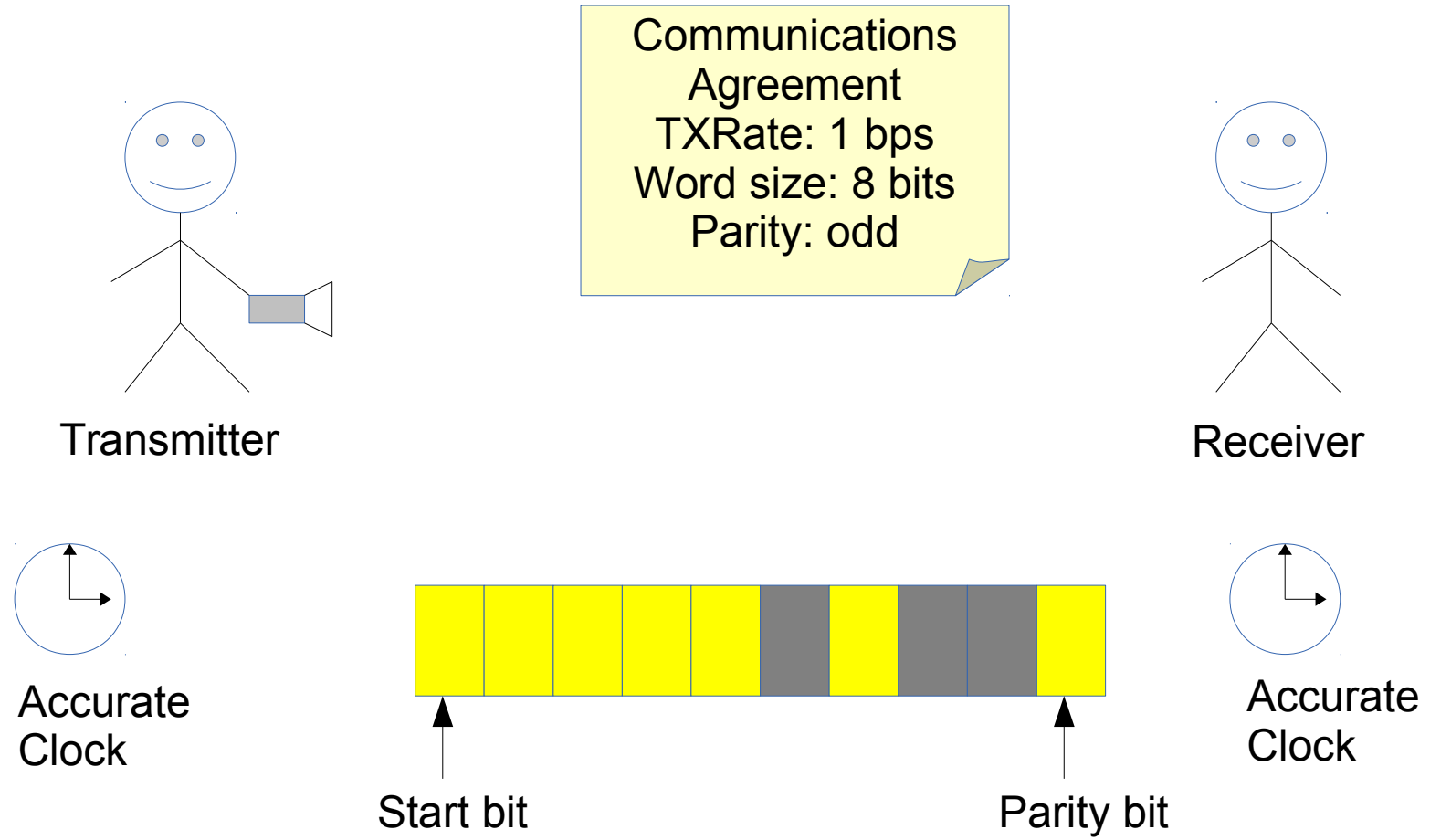
Serial communications

How many bits per transmission?



Serial communications

Accurate transmission?

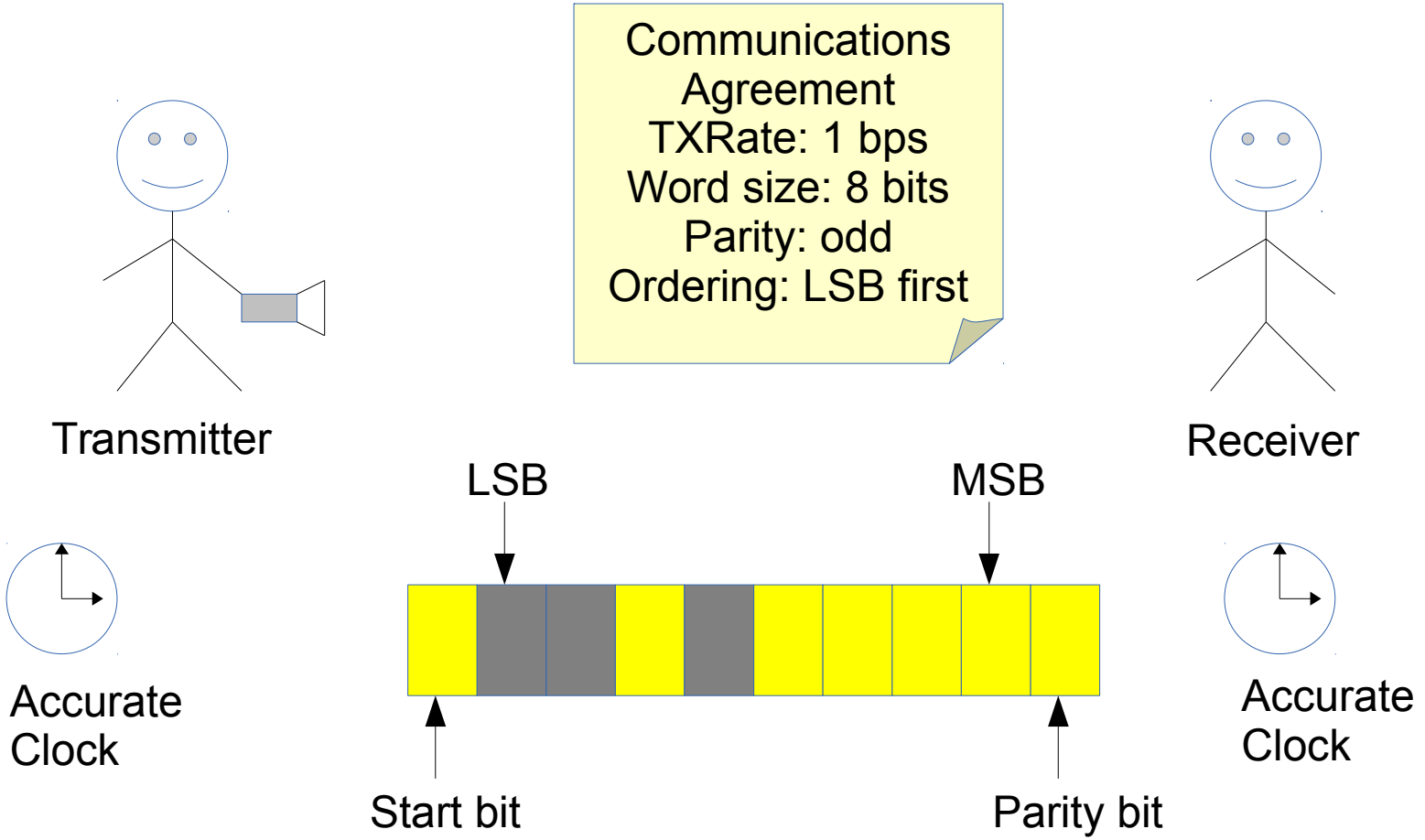


Serial communications

- Parity checking: Count the number of 1's in transmitted data (not the start bit) N
- If agreement is for odd parity:
  - If N odd, send parity bit of 0
  - If N even, send parity bit of 1
- If agreement is for even parity:
  - If N odd, send parity bit of 1
  - If N even, send parity bit of 0

Serial communications

LSB or MSB first?



Serial communications

- Encoding transmission
- Software approach:
  - (demo)

Serial communications

```
#include <stdio.h>
void Transmit(char TX)
{
    int i;
    for (i=0;i<8;i++)
    {
        if (TX & 1)
            printf(" 1 ");
        else
            printf(" 0 ");
        TX = TX >> 1;
    }
    printf("\n");
}
int main()
{
    char TXValue = 192;
    Transmit(TXValue);
}
```

Serial communications

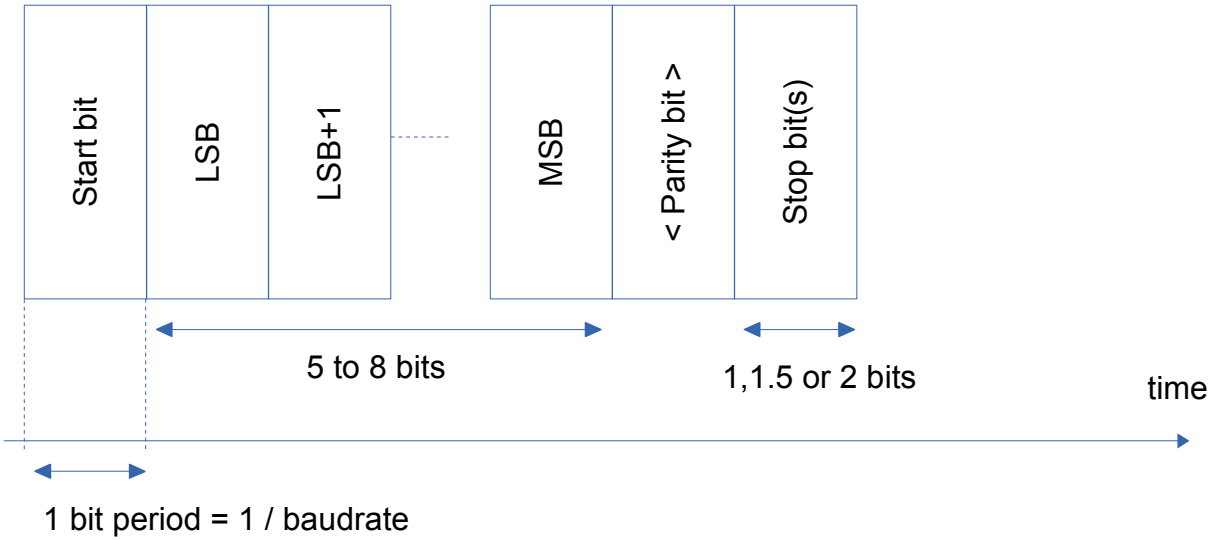
- Hardware serial communications
  - UART = Universal Asynchronous Receiver Transmitter
  - USART = Universal Synchronous/Asycnchronous Receiver/Transmitter
- Commonly found in microcontrollers

Serial communications

- Hardware protocols:
  - Asynchronous
    - RS232
    - RS422
    - RS485
  - Synchronous (shared clock)
    - SPI
    - I2C
    - USB
    - Ethernet

Serial communications

- We will focus on RS232



Serial communications

RS232 is an asynchronous, point to point (usually), digital protocol. Transmitter and receiver must be preconfigured to have matching settings for:

Baud rate. Typical values include: 300,600,1200,2400,4800,9600,19200,38400,57600,115200 bits per second.

Number of **data bits per frame** : Between 5 and 8. 8 is common of character (ASCII) based communications.

Parity : Values can be **None**, **Odd**, **Even**, **None**, **Mark**, and **Space**

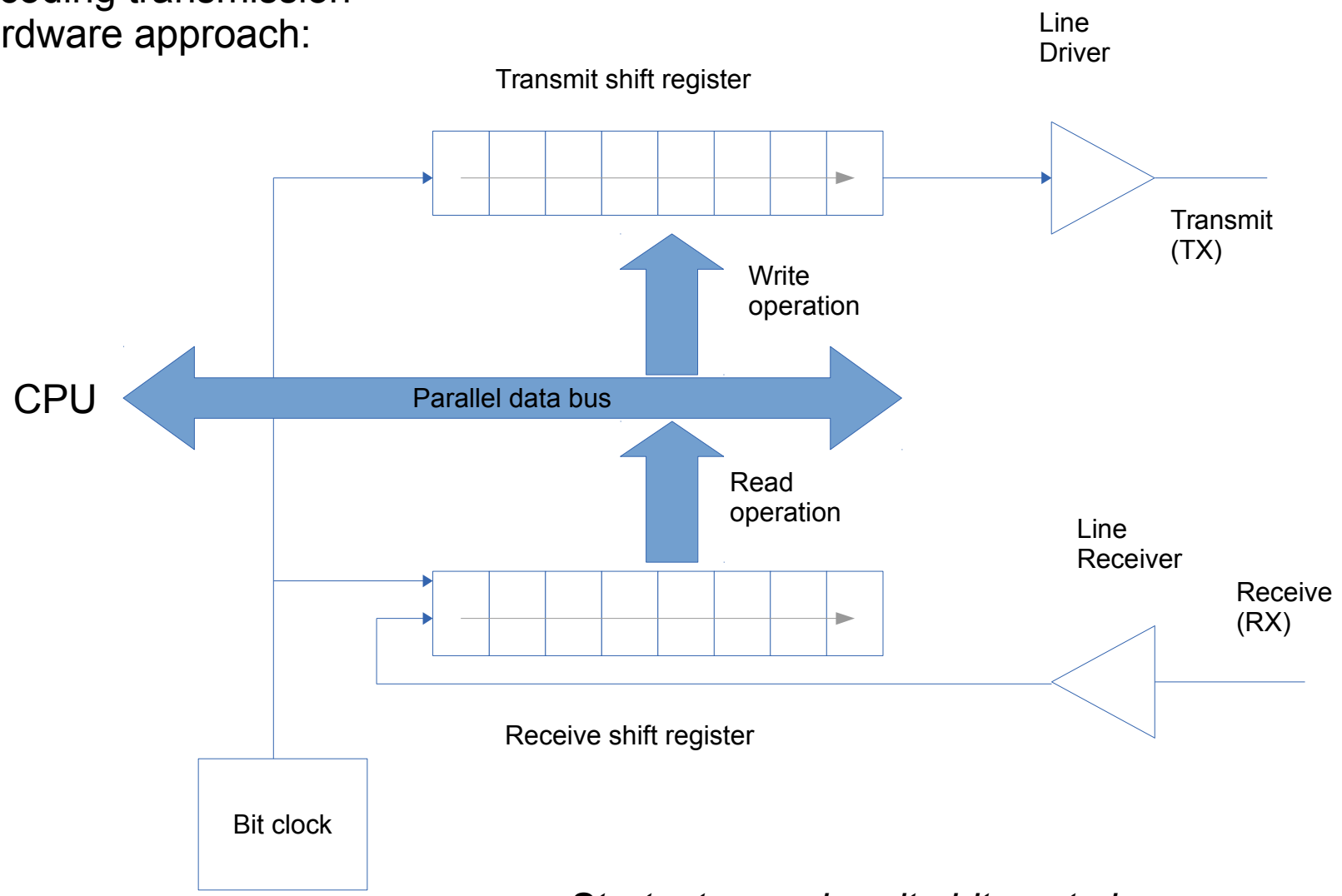
Number of **stop bits per frame**: 1, 1.5 or 2 bits.

Serial communications

- RS232 in the field uses negative logic:
  - Logic 1 is represented by a negative voltage
  - Logic 0 is represented by a positive voltage
- “Logic level” RS232 operates using voltages as found on a PC (We are using this)
  - Logic 1 = 3V
  - Logic 0 = 0V

Serial communications

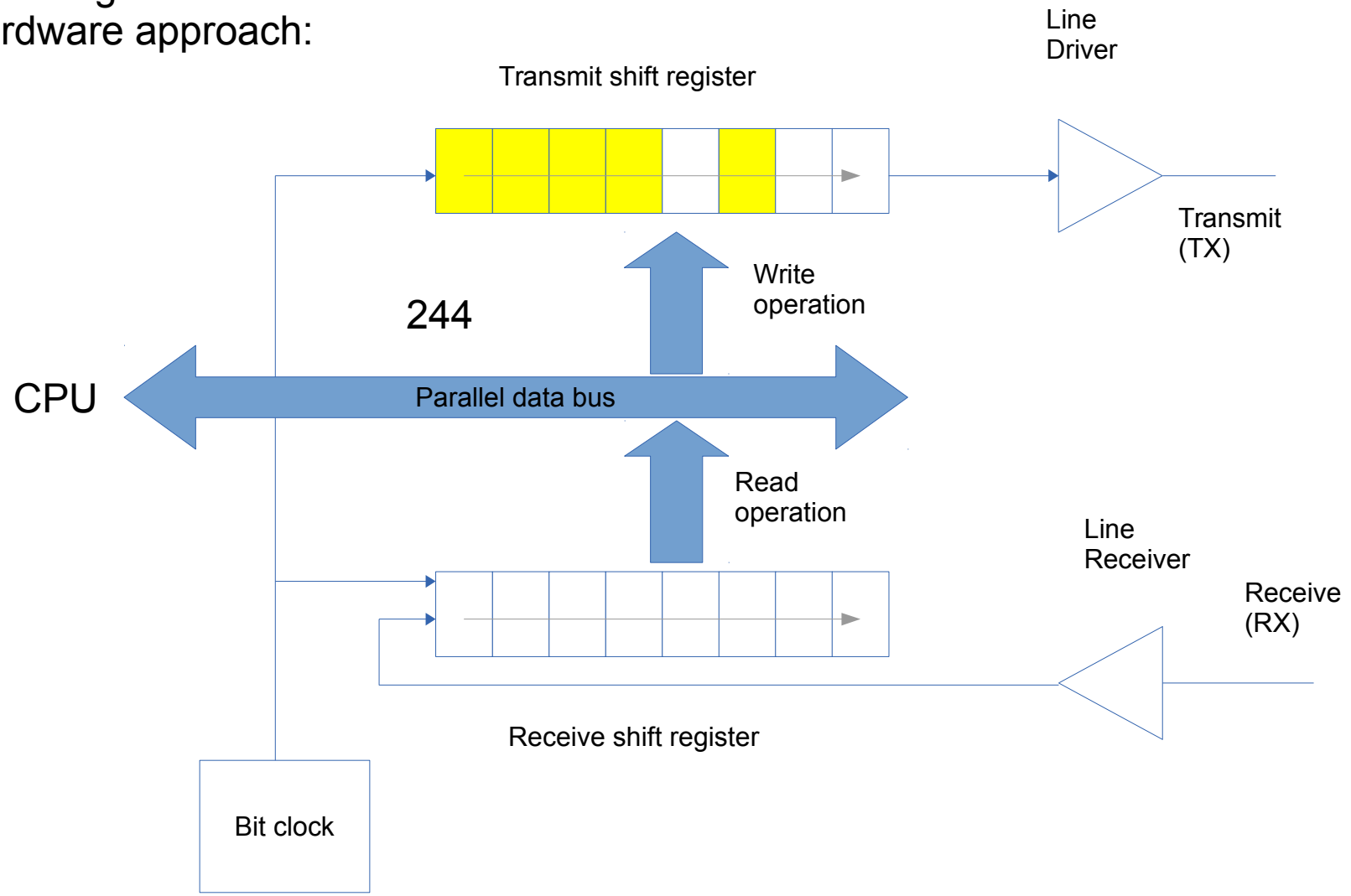
- Encoding transmission
- Hardware approach:



Start, stop and parity bits not shown

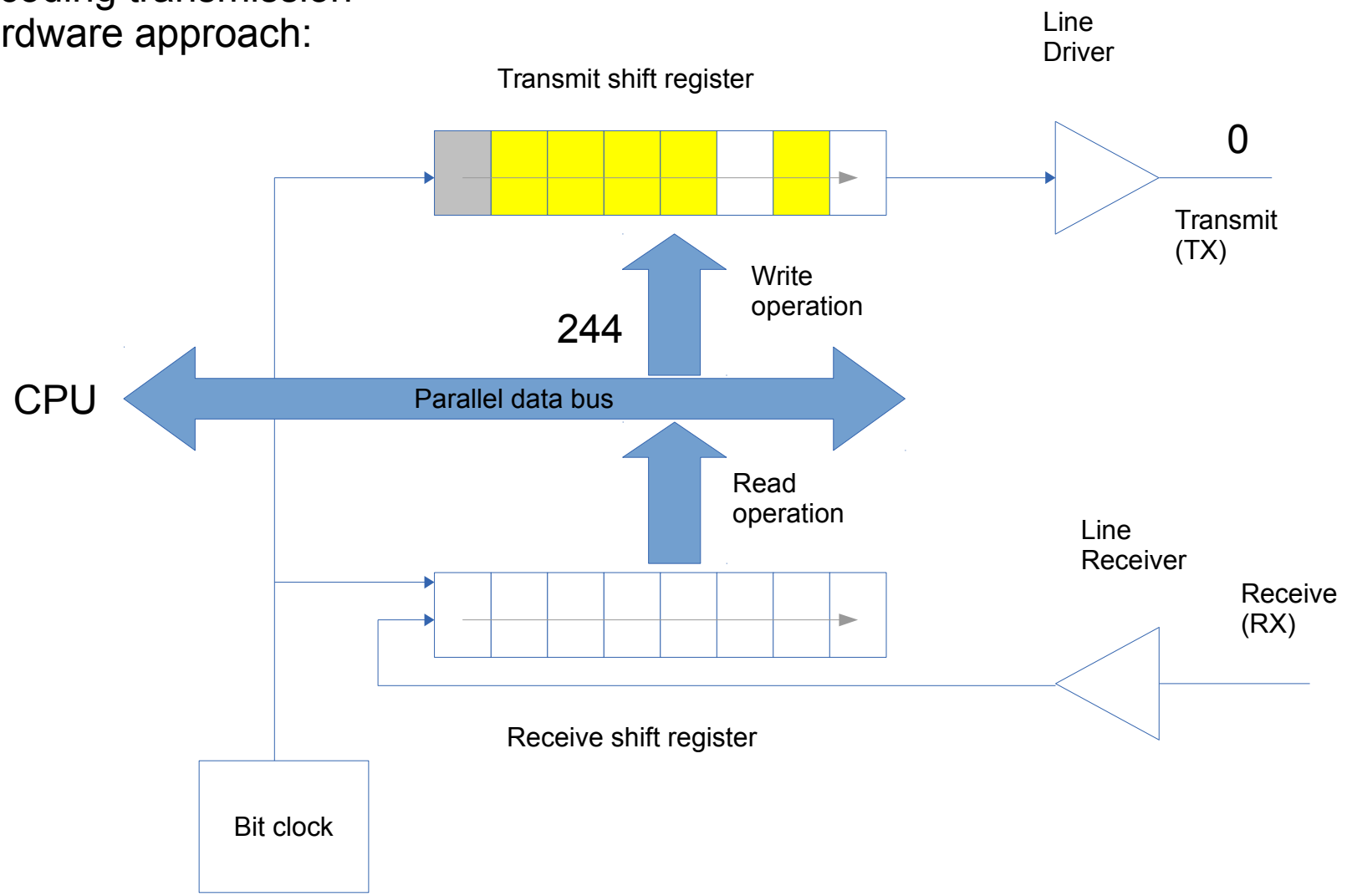
Serial communications

- Encoding transmission
- Hardware approach:



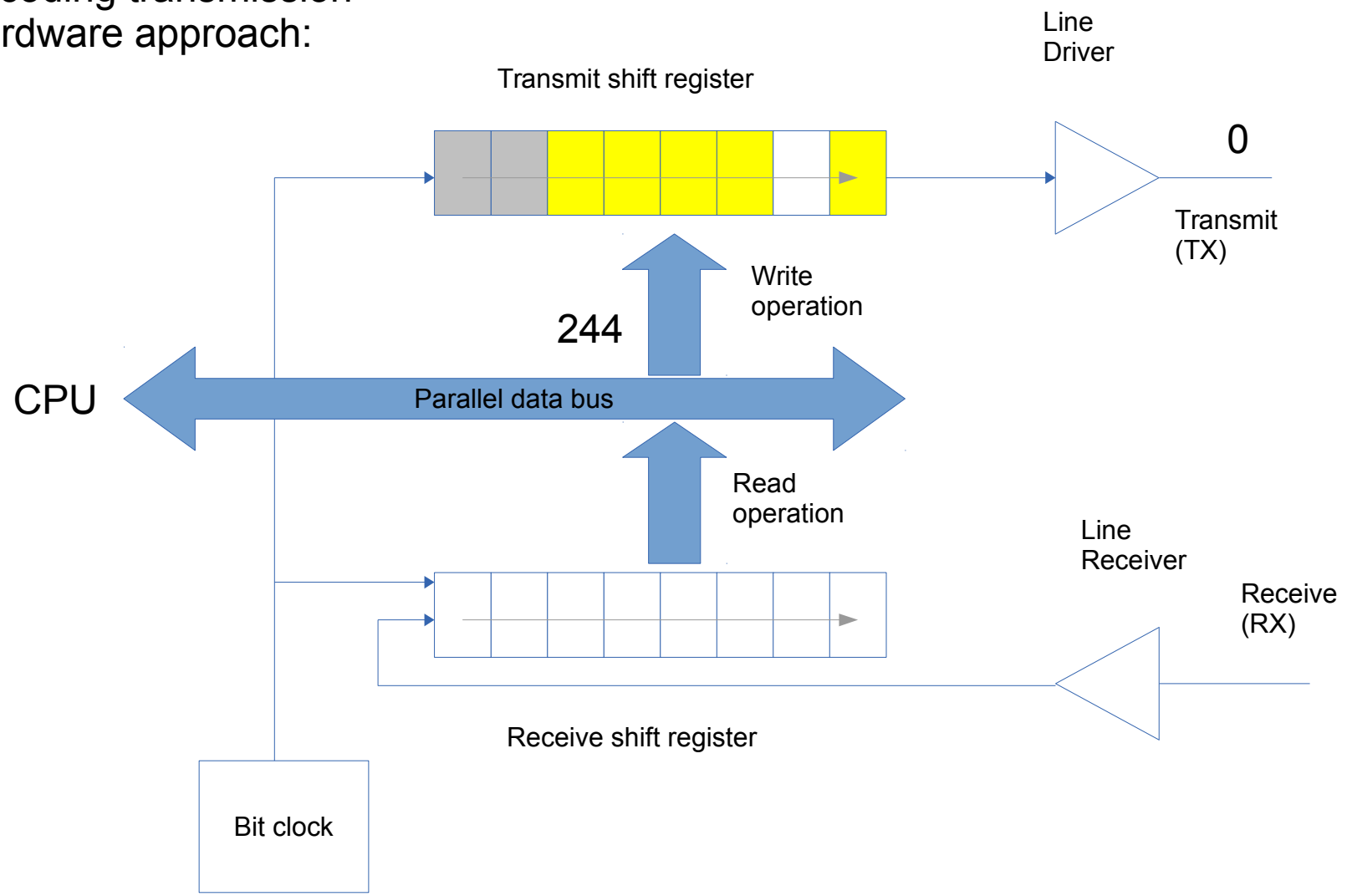
Serial communications

- Encoding transmission
- Hardware approach:



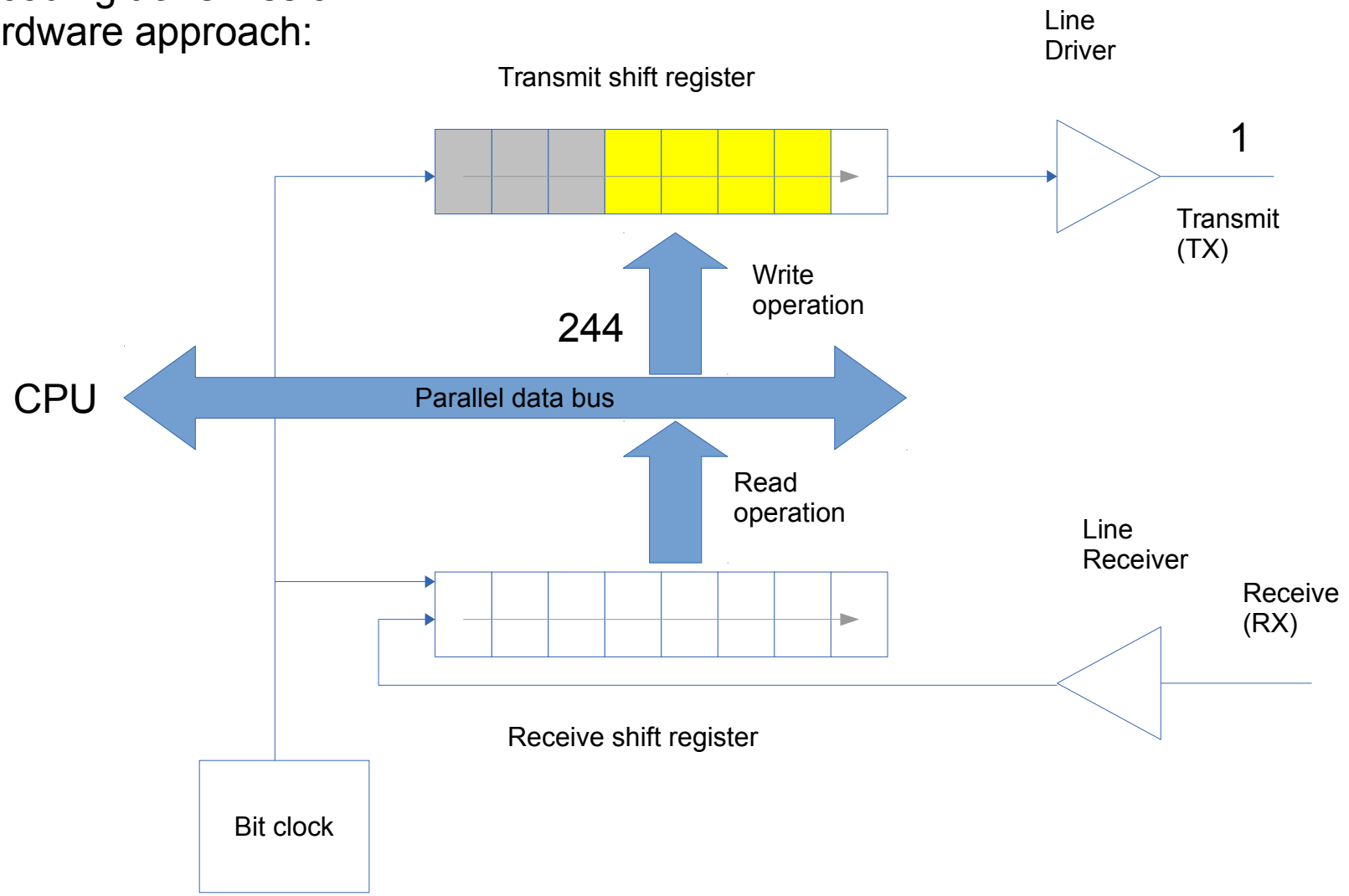
Serial communications

- Encoding transmission
- Hardware approach:



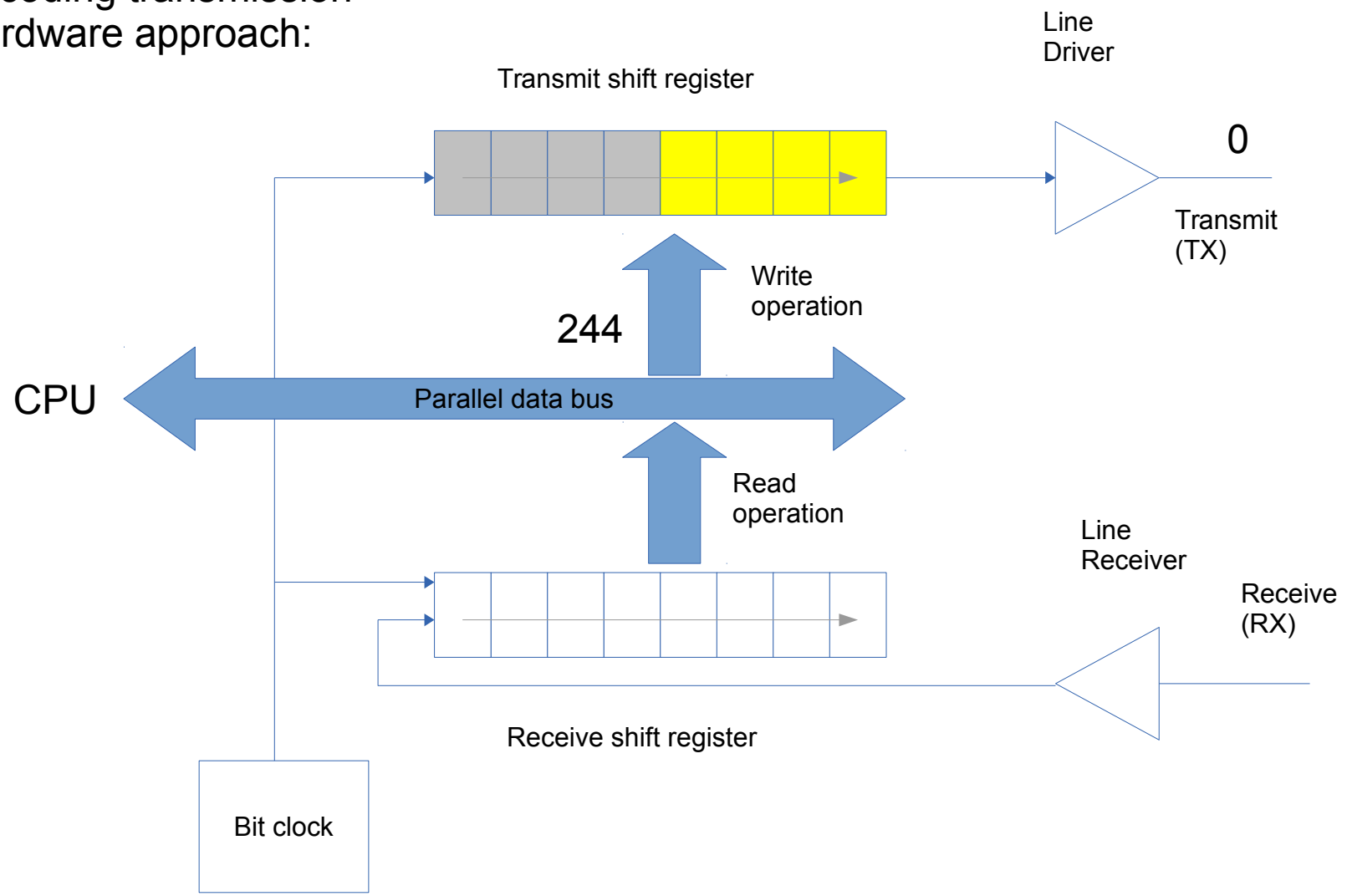
Serial communications

- Encoding transmission
- Hardware approach:



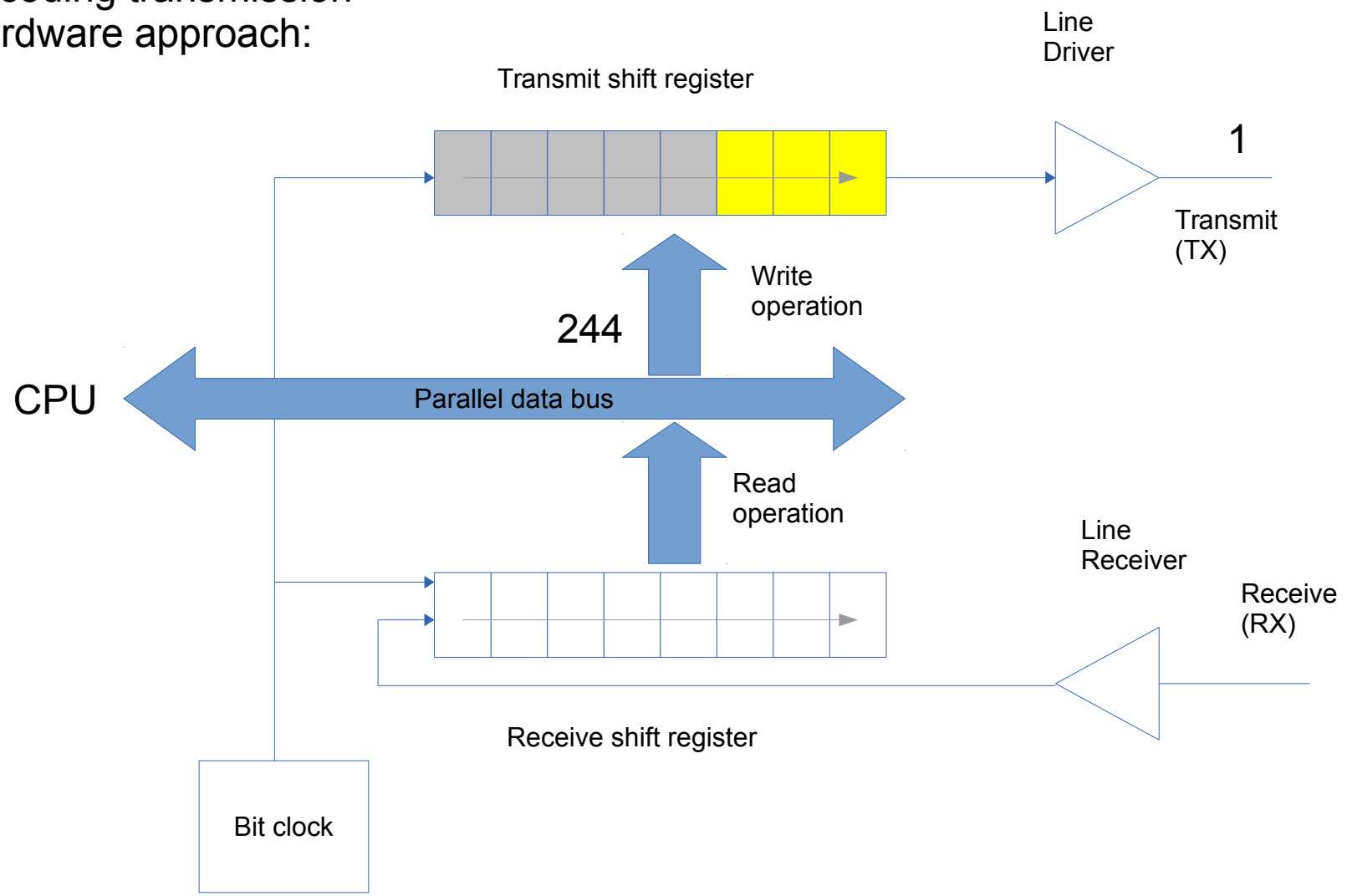
Serial communications

- Encoding transmission
- Hardware approach:



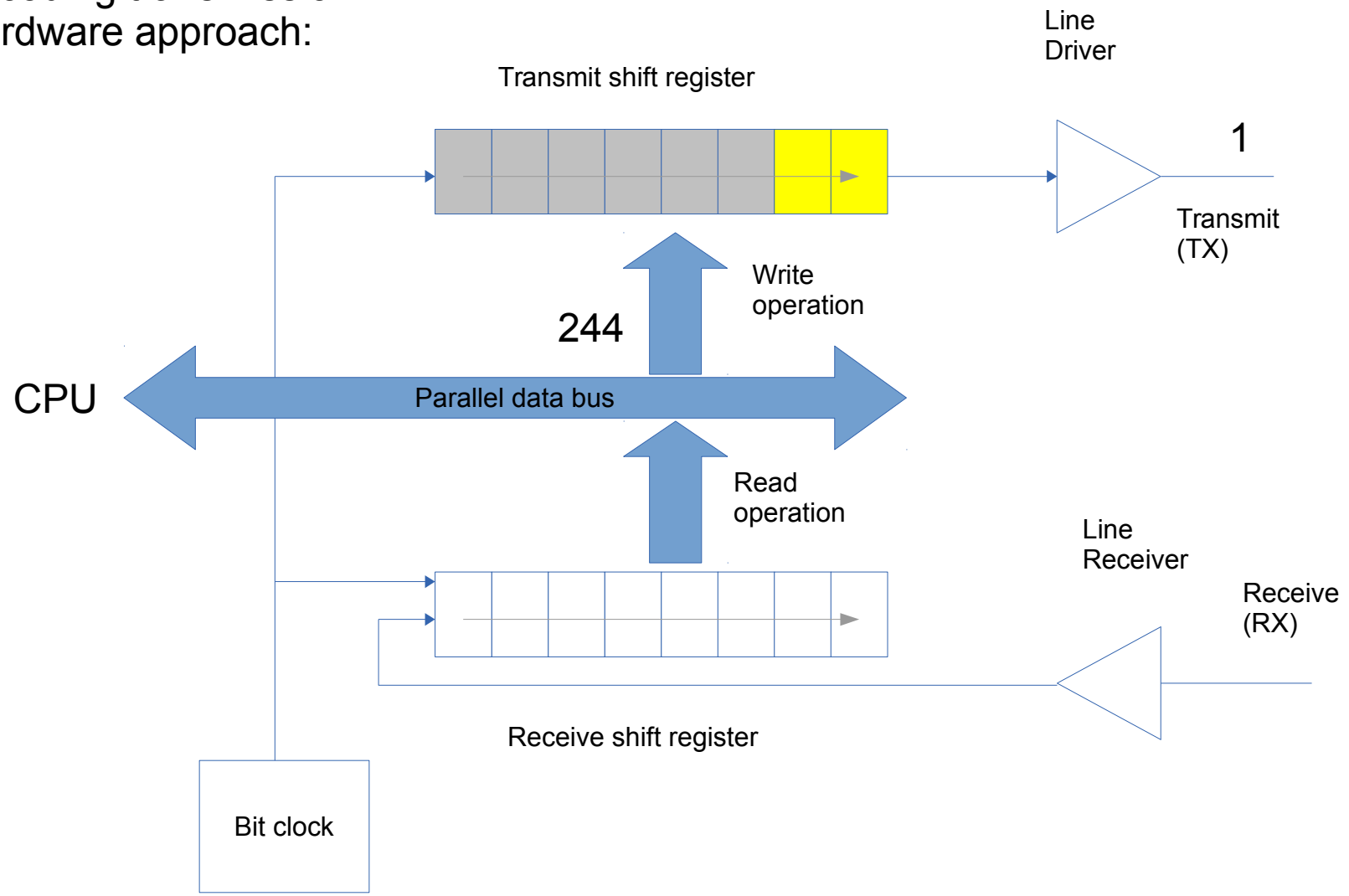
Serial communications

- Encoding transmission
- Hardware approach:



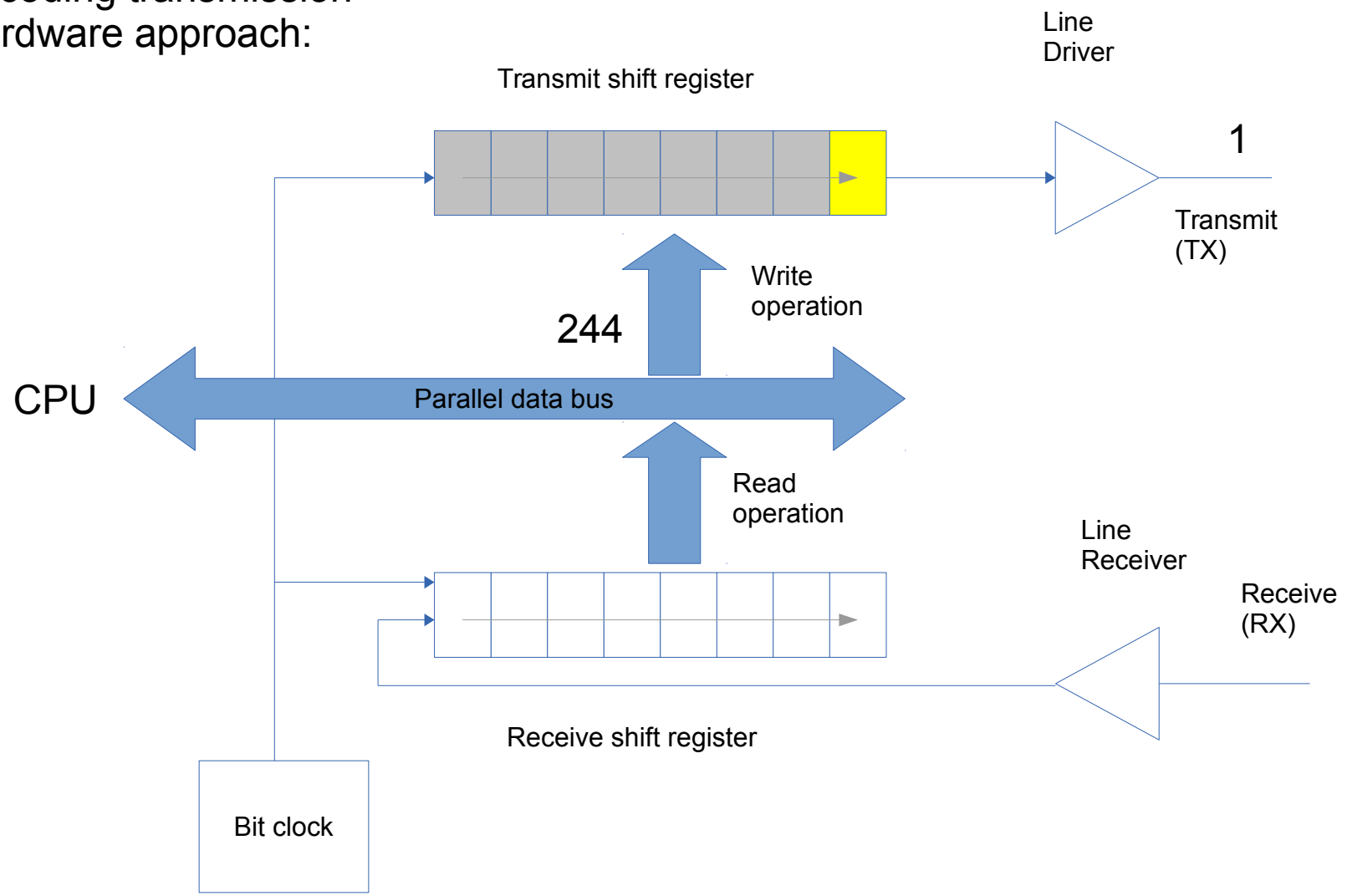
Serial communications

- Encoding transmission
- Hardware approach:



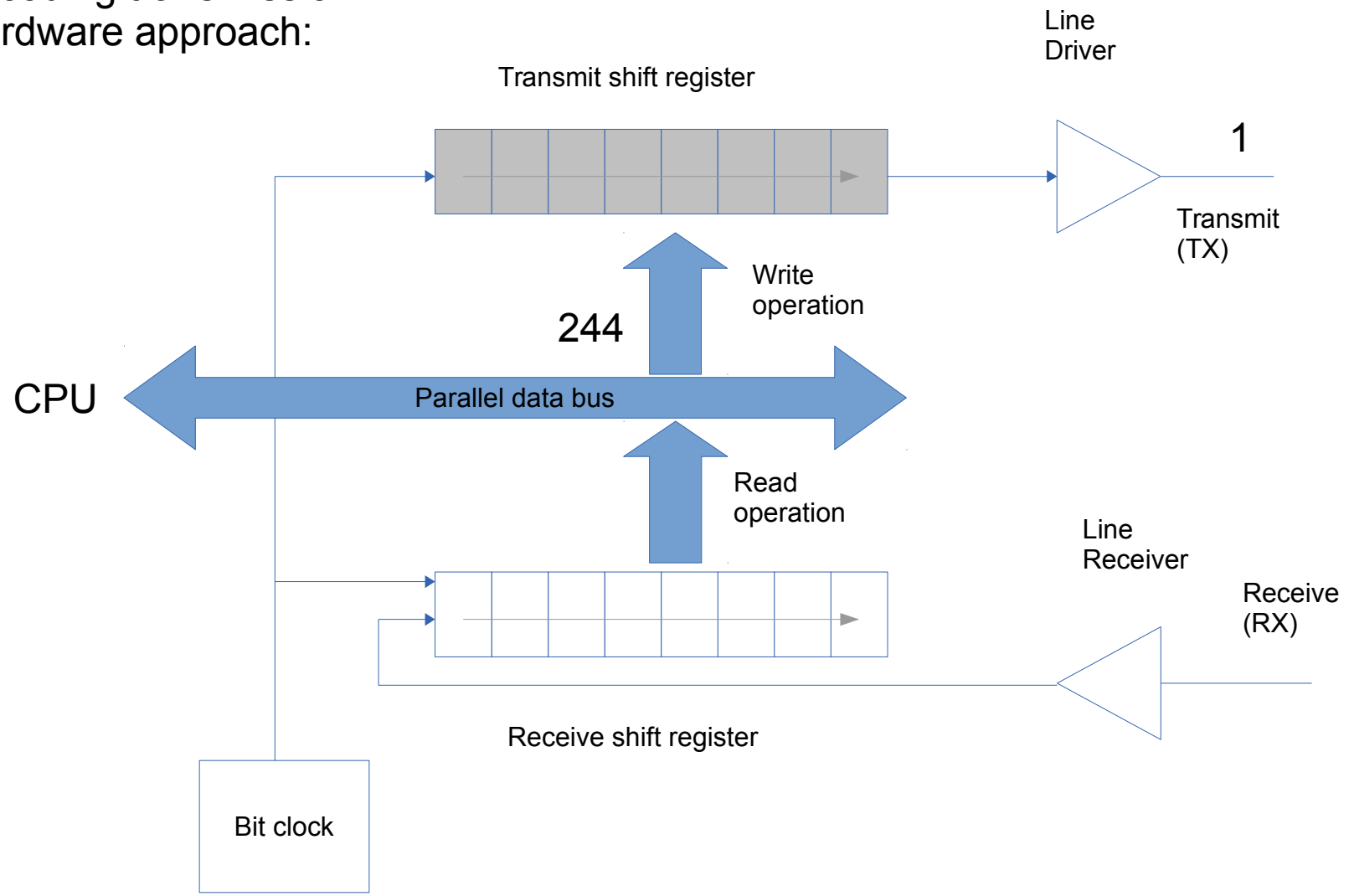
Serial communications

- Encoding transmission
- Hardware approach:



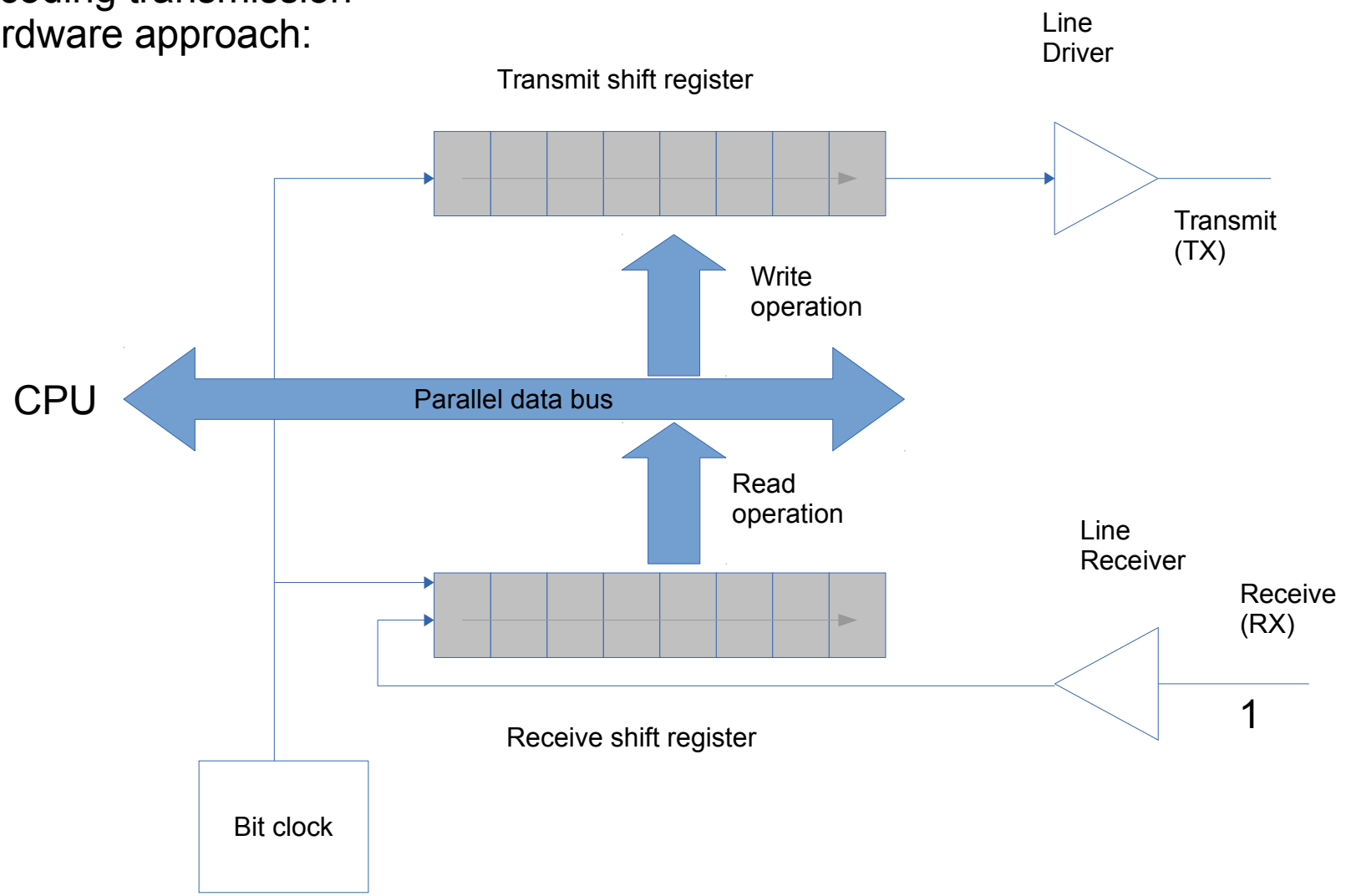
Serial communications

- Encoding transmission
- Hardware approach:



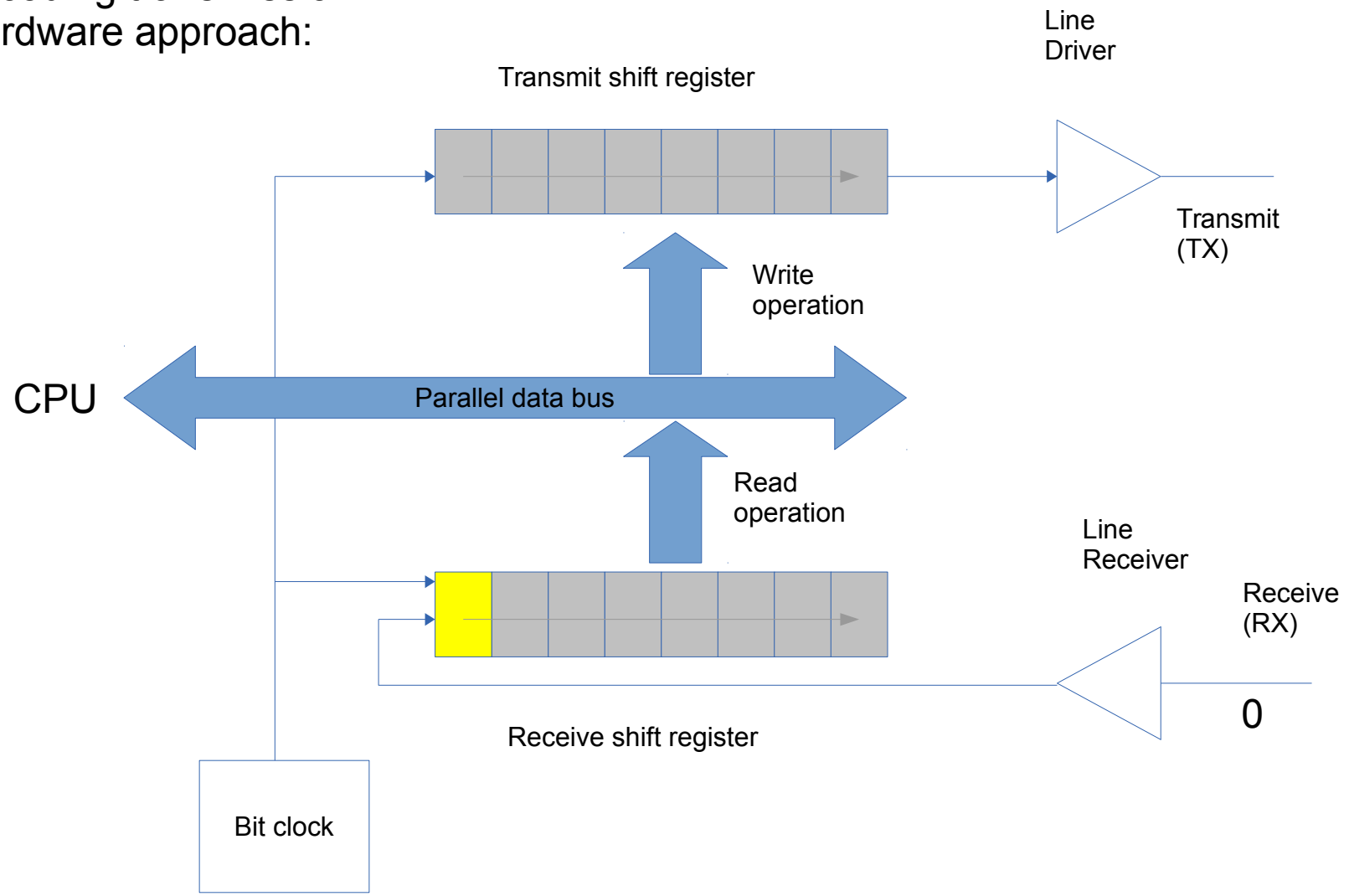
Serial communications

- Encoding transmission
- Hardware approach:



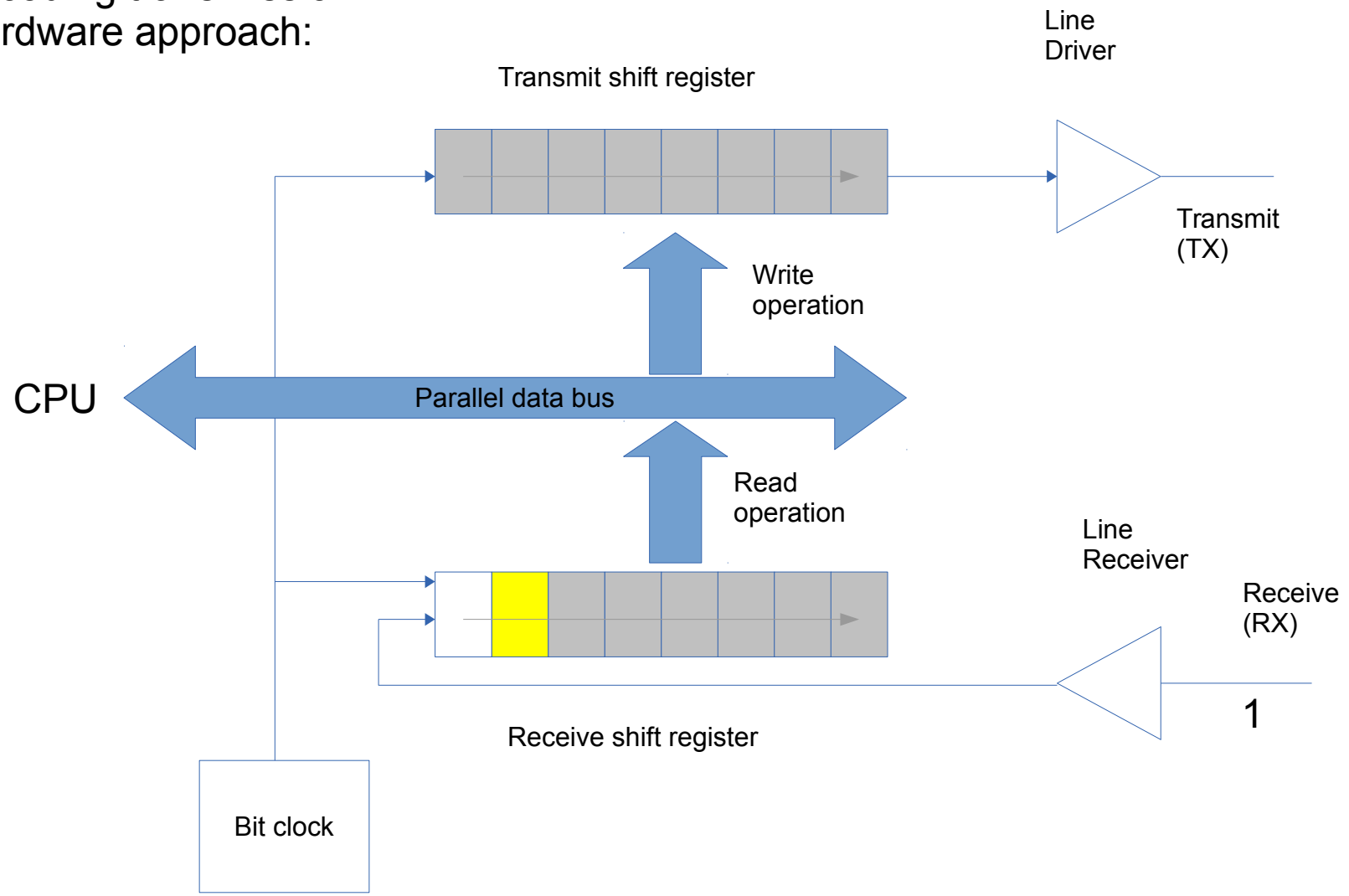
Serial communications

- Encoding transmission
- Hardware approach:



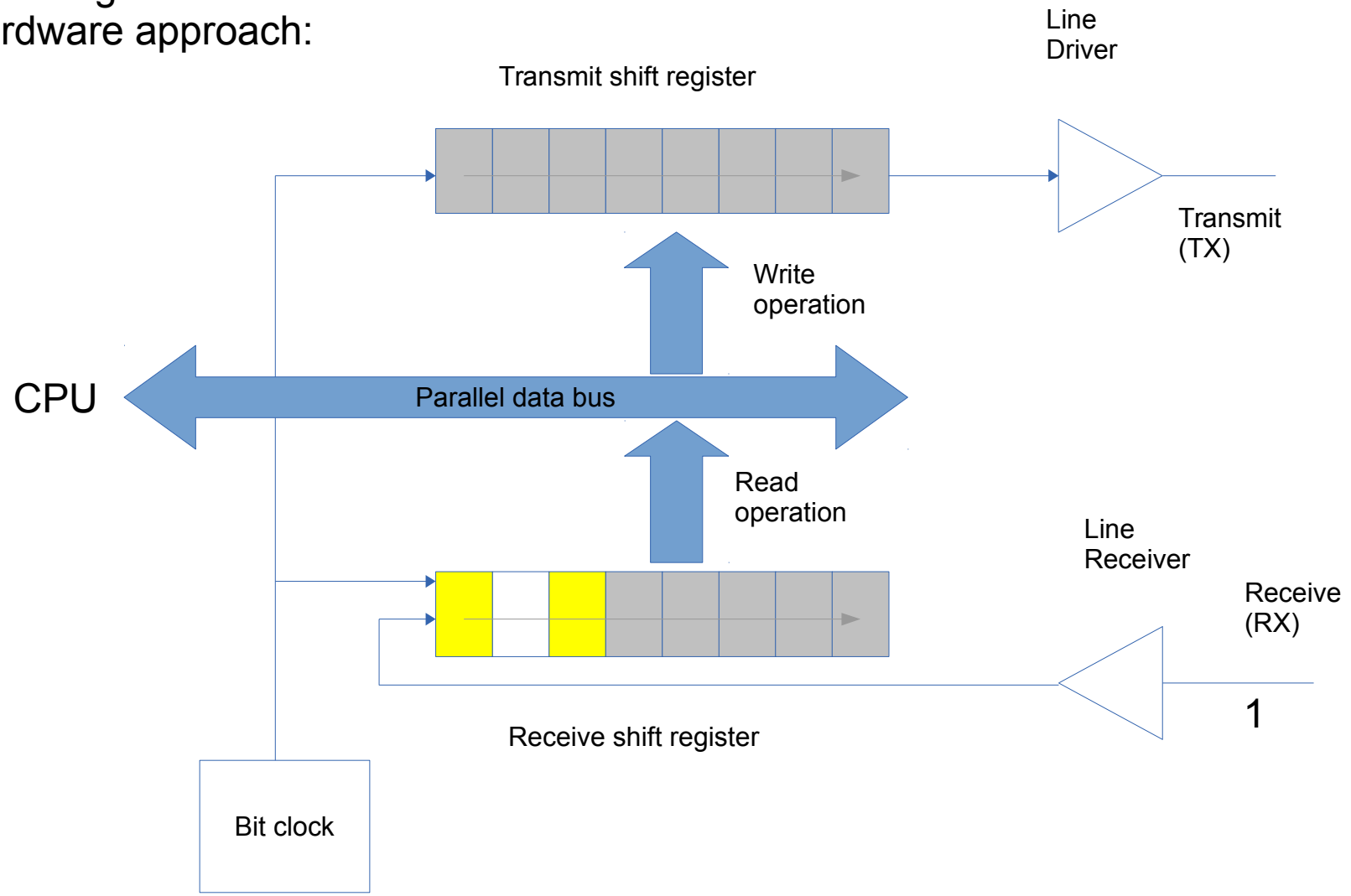
Serial communications

- Encoding transmission
- Hardware approach:



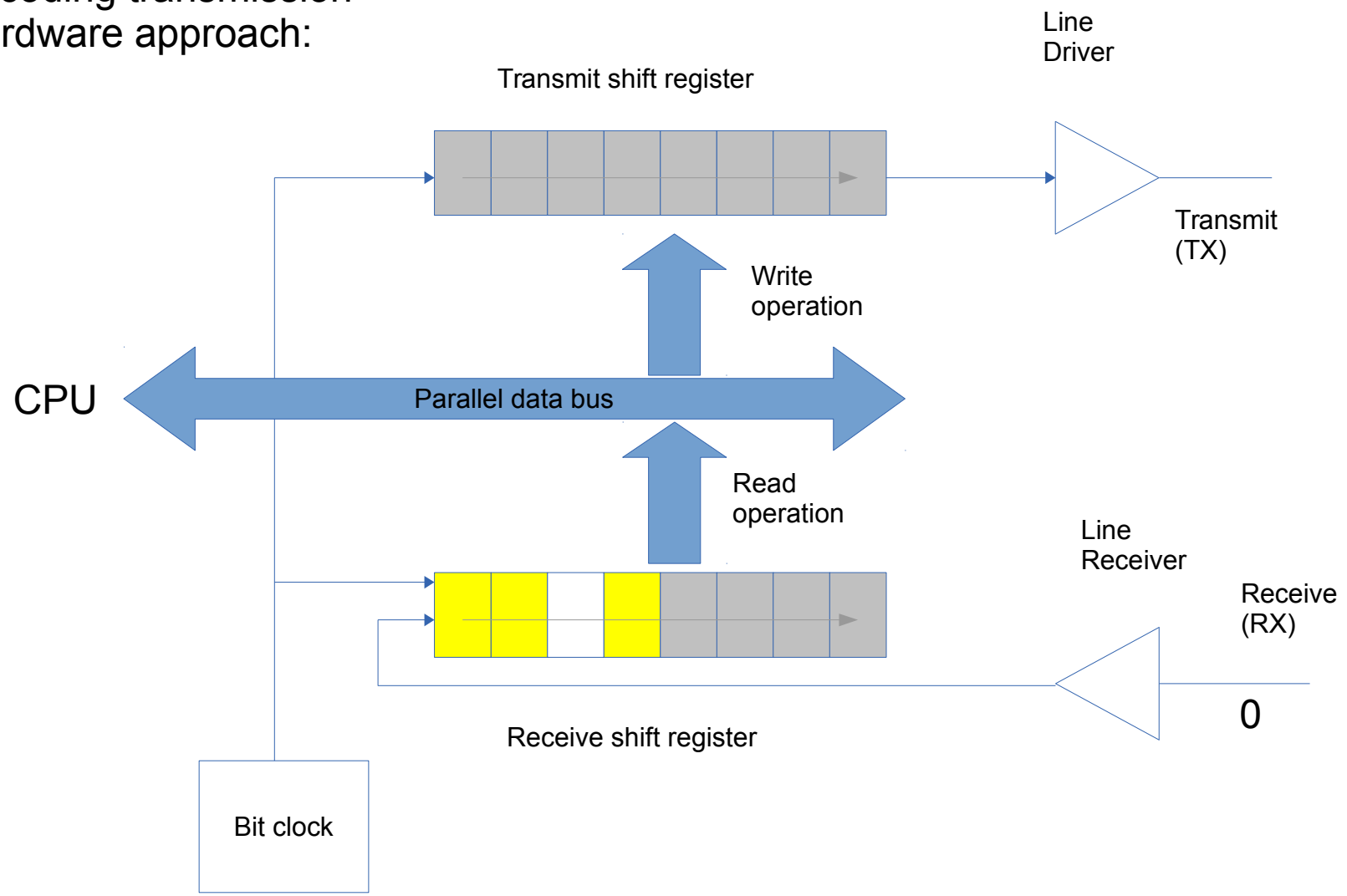
Serial communications

- Encoding transmission
- Hardware approach:



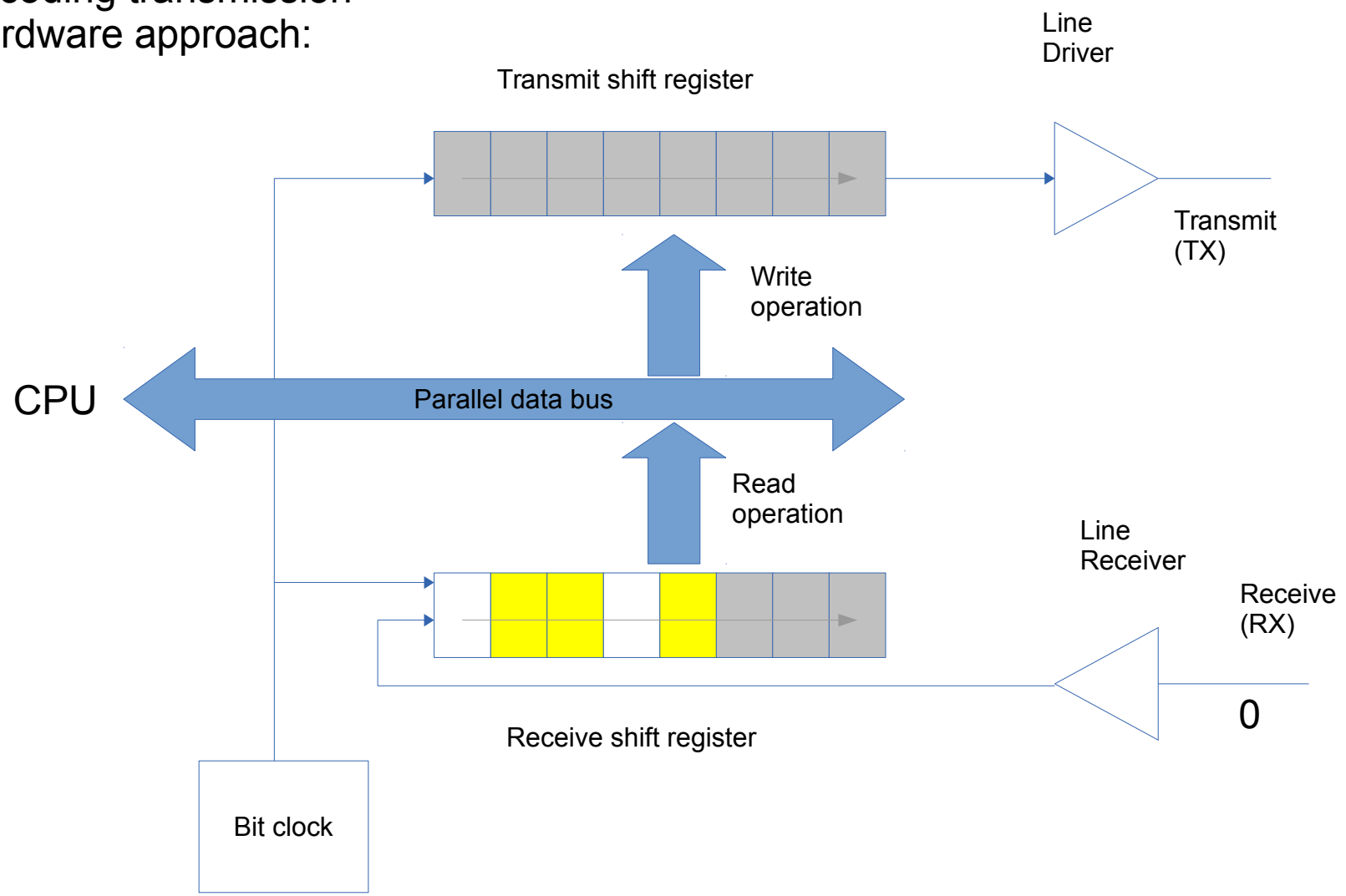
Serial communications

- Encoding transmission
- Hardware approach:



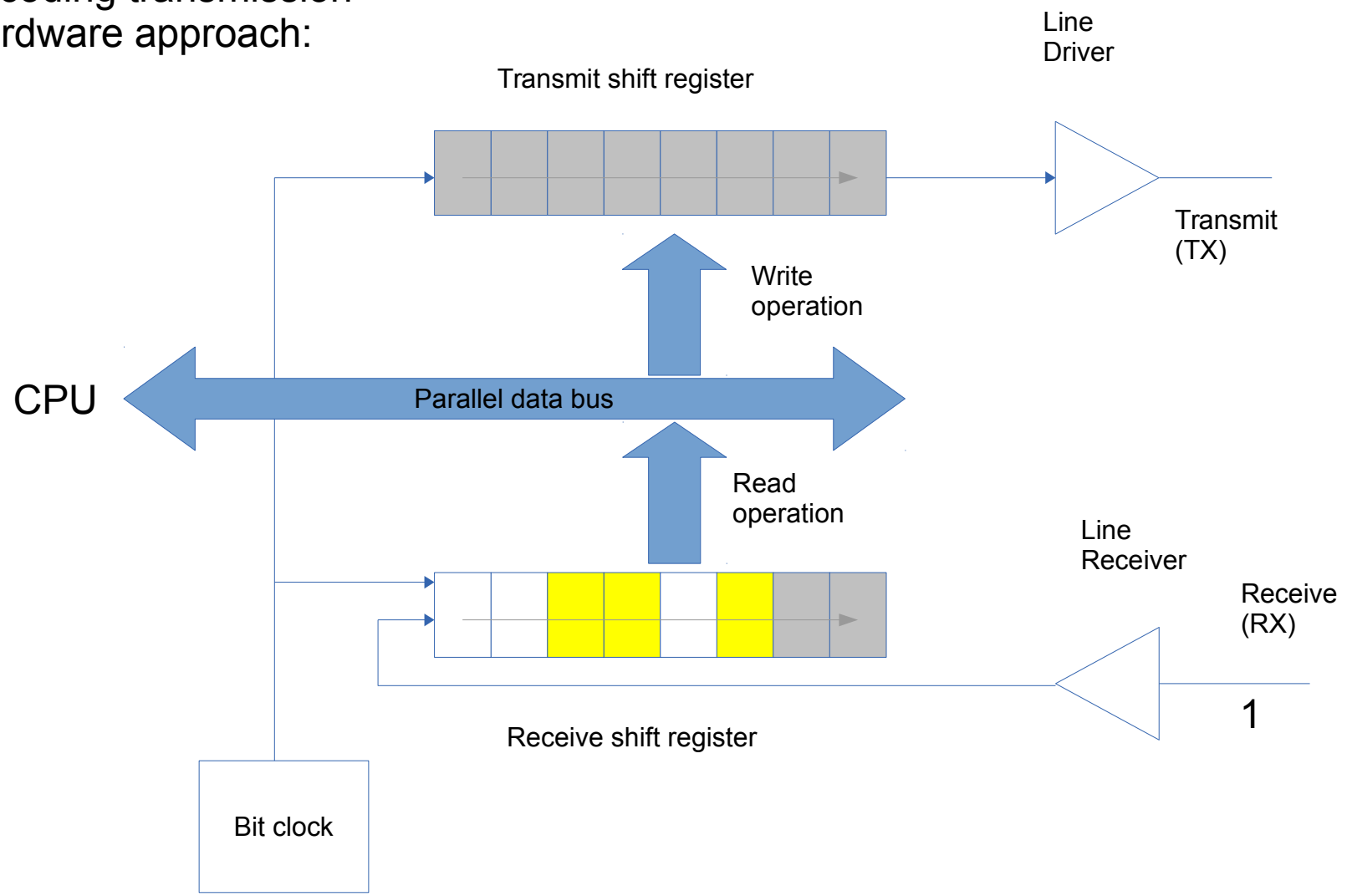
Serial communications

- Encoding transmission
- Hardware approach:



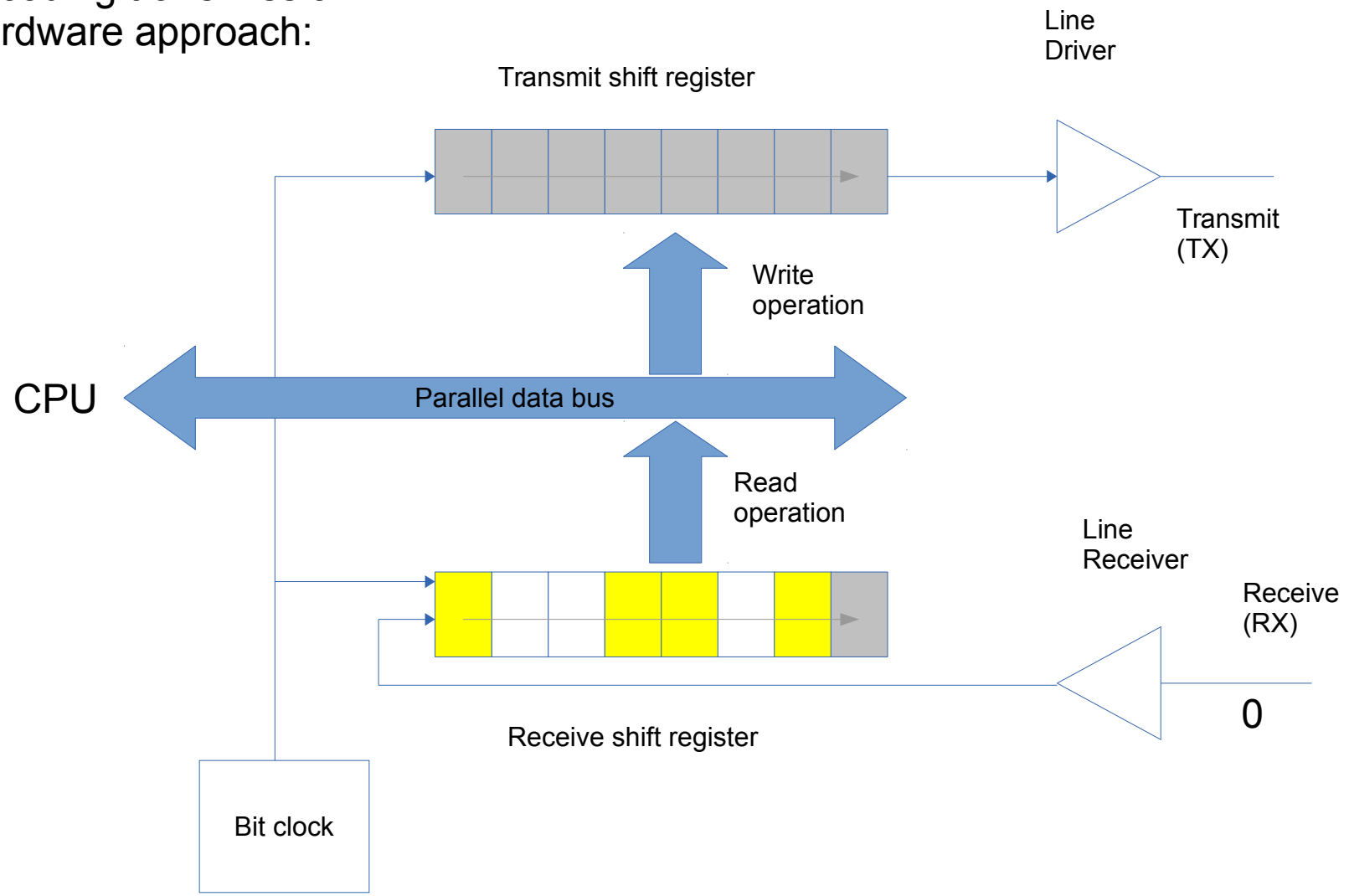
Serial communications

- Encoding transmission
- Hardware approach:



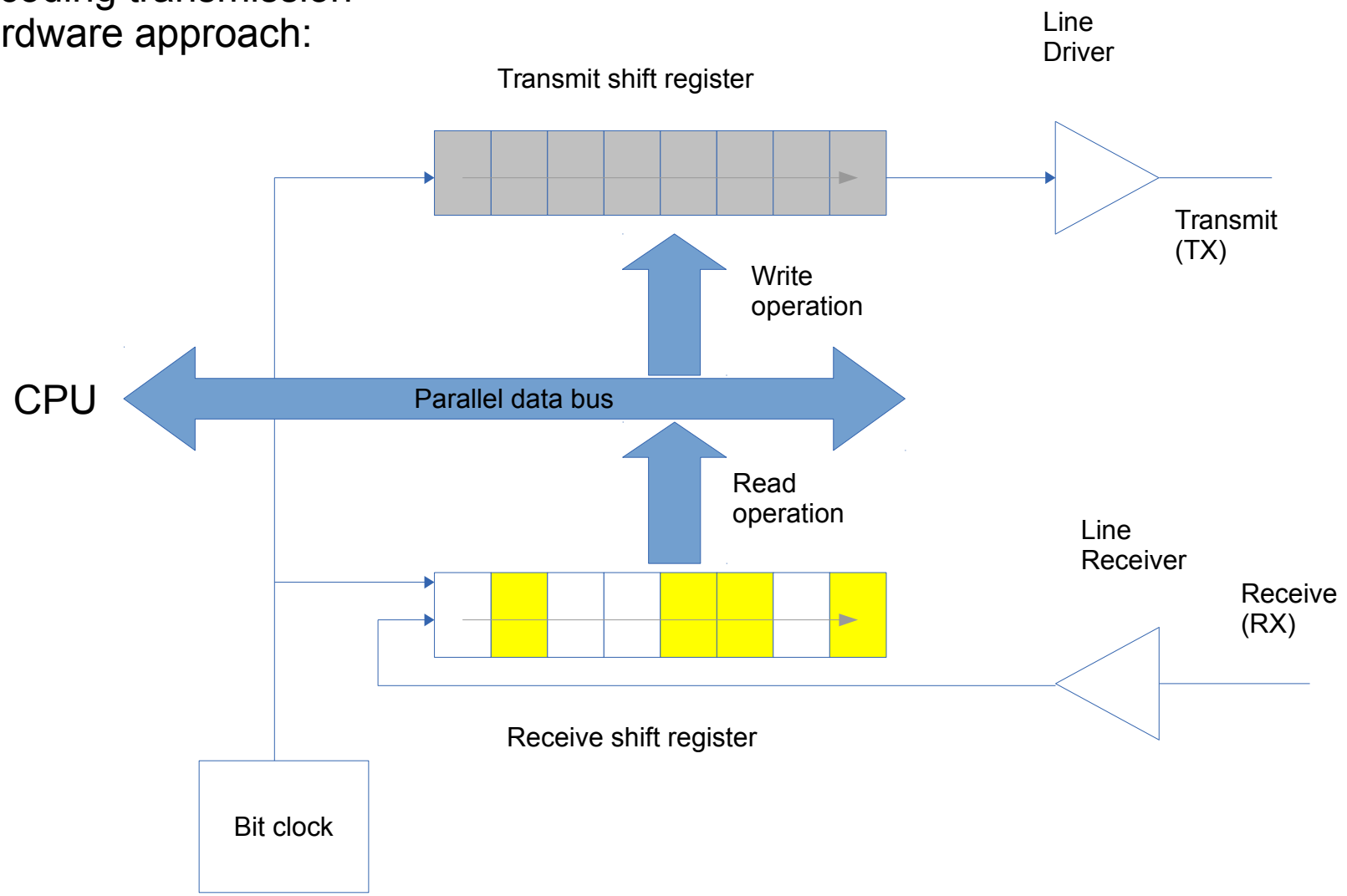
Serial communications

- Encoding transmission
- Hardware approach:



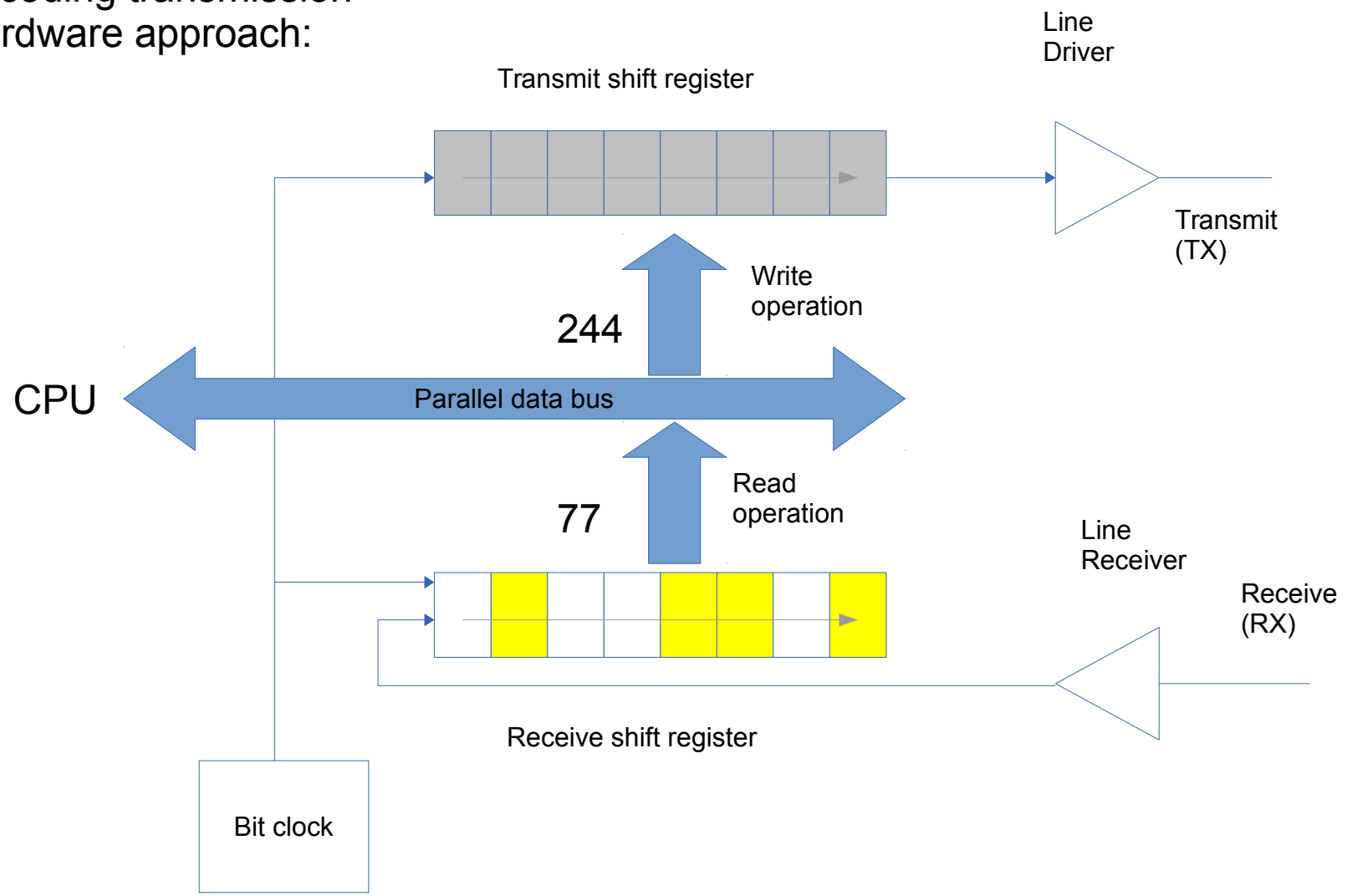
Serial communications

- Encoding transmission
- Hardware approach:

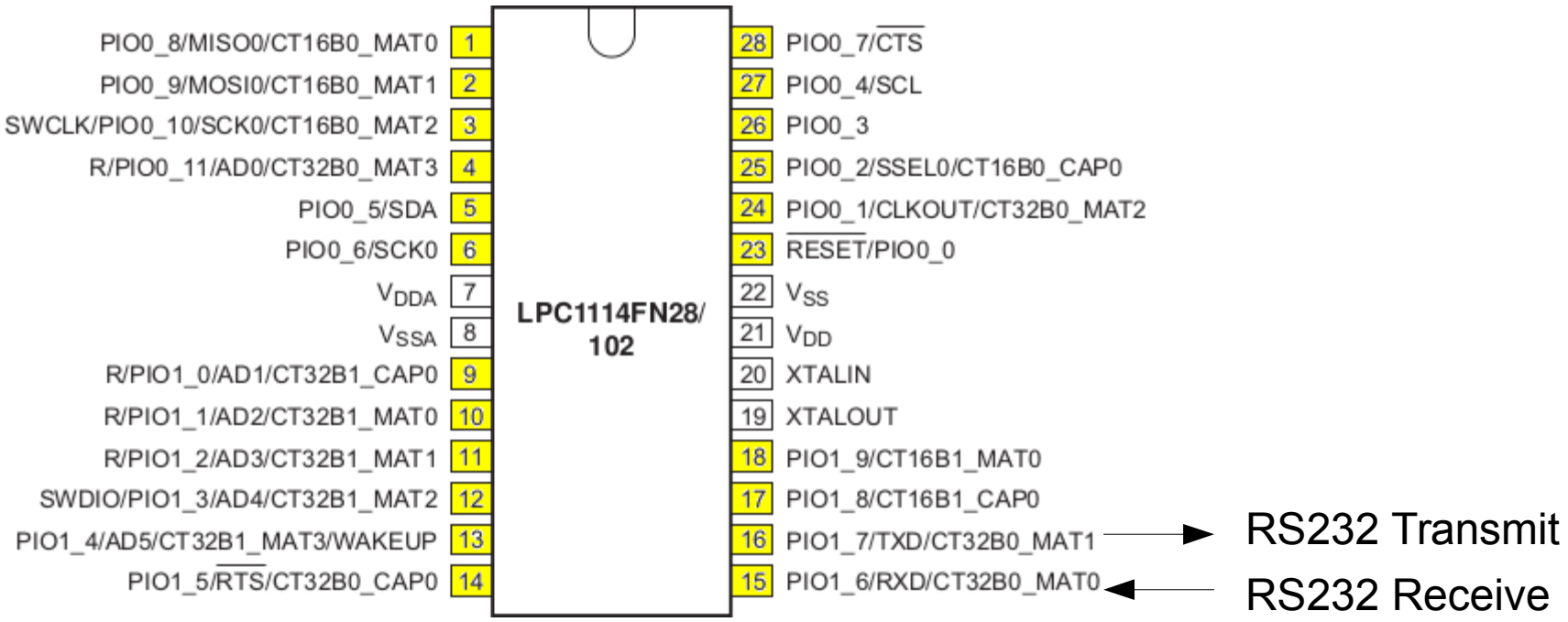


Serial communications

- Encoding transmission
- Hardware approach:



Serial communications



Serial communications

```
void initUART()
{
    SYSAHBCLKCTRL |= BIT6 + BIT16; // Turn on clock for GPIO and
IOCON
    // Enable UART RX function on PIO1_6
    IOCON_PIO1_6 |= BIT0;
    IOCON_PIO1_6 &= ~(BIT1+BIT2);
    // Enable UART TX function on PIO1_7
    IOCON_PIO1_7 |= BIT0;
    IOCON_PIO1_7 &= ~(BIT1+BIT2);
    // Turn on clock for UART
    SYSAHBCLKCTRL |= BIT12;
    UARTCLKDIV = 1;
    // PCLK = 48Mhz. Desired Baud rate = 9600
    // See table 199
    // 9600=48MHz/(16* (256*U0DLM + U0DLL)*(1+DivAddVal/MulVal))
    // 312.5 = (256*U0DLM+U0DLL)*(1+DivAddVal/MulVal)
    // let U0DLM=1, DivAddVal=0,MulVal =1
    // 312.5=256+U0DLL
    // U0DLL=56.5.
    // Choose U0DLL=56.
    // Actual baud rate achieved = 9615 - close enough.
```

Serial communications

```
    U0LCR |= BIT7; // Enable divisor latch access
    U0FDR = (1<<4)+0; // Set DivAddVal = 0; MulVal = 1
    U0DLL = 56;
    U0DLM = 1;
    U0LCR |= (BIT1+BIT0); // set word length to 8 bits.
    // Turn on FIFO, reset, set trigger to 1 byte
    U0FCR = (BIT0 | BIT1 | BIT2);
    U0LCR &= ~BIT7; // Disable divisor latch access
}// end of initUART
```

Serial communications

```
void eputc(char c)
{
    U0THR = c; // put char in UART0 Transmit Holding Register
    while((U0LSR & BIT5) == 0); // Wait for tx to finish
}
char egetc()
{
    return U0RBR;  // return contents of UART0 Receive Buffer
                   // register
}
void printString(char *String)
{
    while(*String)
    {
        eputc(*String);
        String++;
    }
}
```

Serial communications

```
char HexDigit(int Value)
{
    if ((Value >=0) && (Value < 10))
        return Value+'0';
    else if ((Value >9) && (Value < 16))
        return Value-10 + 'A';
    else
        return 'z';
}
```

Serial communications

```
void printHex(unsigned int Number)
{
    // Output the number over the serial port as
    // as hexadecimal string.
    char TxString[9];
    int Index=8;
    TxString[Index]=0; // terminate the string
    Index--;
    while(Index >=0)
    {
        TxString[Index]=HexDigit(Number & 0x0f);
        Number = Number >> 4;
        Index--;
    }
    printString(TxString);
}
```

Serial communications

- How would you implement a printDecimal?
- What about printf ?