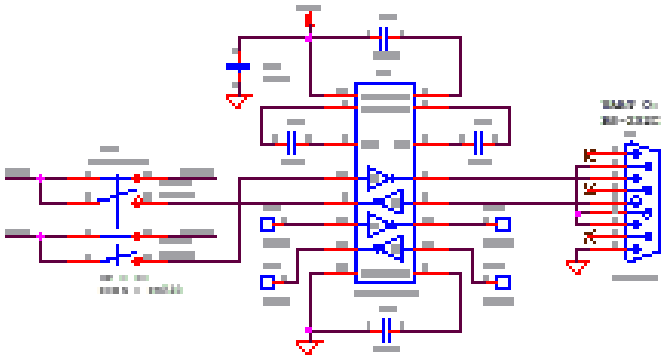
Serial Communications

These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

In these notes . . .

- Why serial?
- Serial Communications
  - RS232 standard
  - UART operation
  - Polled Code
  - Interrupt Driven Code



- For details see **MSP430FR57xx Family User's Guide**
  - *Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode*

Why Communicate Serially?

- Native word size is multi-bit (8, 16, 32, etc.)
- Often it's not feasible to support sending all the word's bits at the same time
  - Cost and weight: more wires needed, larger connectors needed
  - Mechanical reliability: more wires => more connector contacts to fail
  - Timing Complexity: some bits may arrive later than others due to variations in capacitance and resistance across conductors
  - Circuit complexity and power: may not want to have 16 different radio transmitters + receivers in the system

How can we communicate serially?

- Need clocking information
  - When does a word start?
  - When is a bit being sent?
  - When does a word end?
- Options
  - Explicit clock – synchronous
    - Separate signal
      - SPI has 1 clock and 1 data line
    - Modify signal to provide clocking
      - Example: short pulse = 0, long pulse = 1
  - Implicit clock – asynchronous
    - Transmitter and receiver follow same rules (protocol)
    - Protocol specifies how to know when word starts, when to sample bits, when word is done
      - Maybe also error detection and other information

Asynchronous Serial Communication Basics

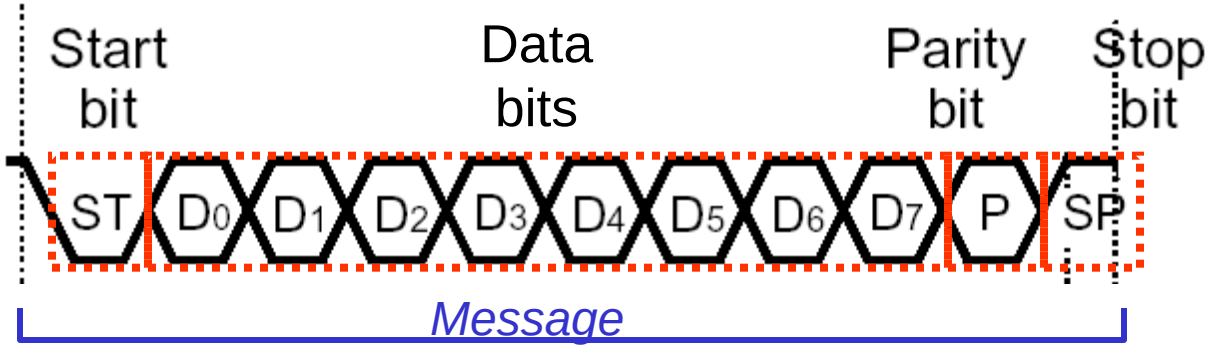
- Transmitter
  - If no data to send, keep sending 1 (stop bit)
  - When there is a data word to send
    - Send a 0 (start bit) to indicate the start of a word
    - Send each data bit in the word (use a shift register for the *transmit buffer*)
    - Send a 1 (stop bit) to indicate the end of the word (keep sending it until more data to send)

- Receiver
  - Wait for a falling edge (beginning of a Start bit)
    - Then wait ½ bit time
    - Do the following for as many data bits in the word
      - Wait 1 bit time
      - Read the data bit and shift it into a *receive buffer* (shift register)
    - Wait 1 bit time
    - Read the bit
      - if 1 (Stop bit), then OK
      - if 0, there's a problem!

For this to work…

- Transmitter and receiver must agree on several things (protocol)
  - Order of data bits
  - Number of data bits
  - What a start bit is (1 or 0)
  - What a stop bit is (1 or 0)
  - How long a bit lasts
    - Transmitter and receiver clocks must be pretty close, since the only timing reference is the start of the start bit

Serial Communication Specifics



- Message fields
  - Start bit (one bit)
  - Data (LSB first or MSB, and size – 7, 8, 9 bits)
  - Optional parity bit is used to make total number of ones in data even or odd
  - Stop bit (one or two bits)
- All devices must use the same communications parameters
  - E.g. communication speed (300 baud, 600, 1200, 2400, 9600, 14400, 19200, etc.)

- More sophisticated network protocols have more information in each message
  - Medium access control – when multiple nodes are on bus, they must arbitrate for permission to transmit
  - Addressing information – for which node is this message intended?
  - Larger data payload
  - Stronger error detection or error correction information
  - Request for immediate response (“in-frame”)

UART Concepts

- UART
  - Universal – configurable to fit protocol requirements (for the whole universe)
  - Asynchronous – no clock line needed to de-serialize bits
  - Receiver/Transmitter

UART Concepts

UART subsystems

- –Two fancy shift registers
  - Parallel to serial for transmit
  - Serial to parallel for receive
- –Programmable clock source
  - Clock must run at 16x desired bit rate
- –Error detection
  - Detect bad stop or parity bits
  - Detect receive buffer overwrite
- –Interrupt generators
  - Character received
  - Character transmitted, ready to send another

eUSCI_A Introduction – UART Mode

- In asynchronous mode, the eUSCI_Ax modules connect the device to an external system via two external pins, **UCAxRXD** and **UCAxTXD**. UART mode is selected when the UCSYNC bit is cleared.
- UART mode features include:
  - 7-bit or 8-bit data with odd, even, or non-parity
  - Independent transmit and receive shift registers
  - Separate transmit and receive buffer registers
  - LSB-first or MSB-first data transmit and receive
  - Built-in idle-line and address-bit communication protocols for multiprocessor systems
  - Receiver start-edge detection for auto wake up from LPMx modes
  - Programmable baud rate with modulation for fractional baud-rate support
  - Status flags for error detection and suppression
  - Status flags for address detection
  - Independent interrupt capability for receive, transmit, start bit received, and transmit complete.

eUSCI_A Operation – UART Mode

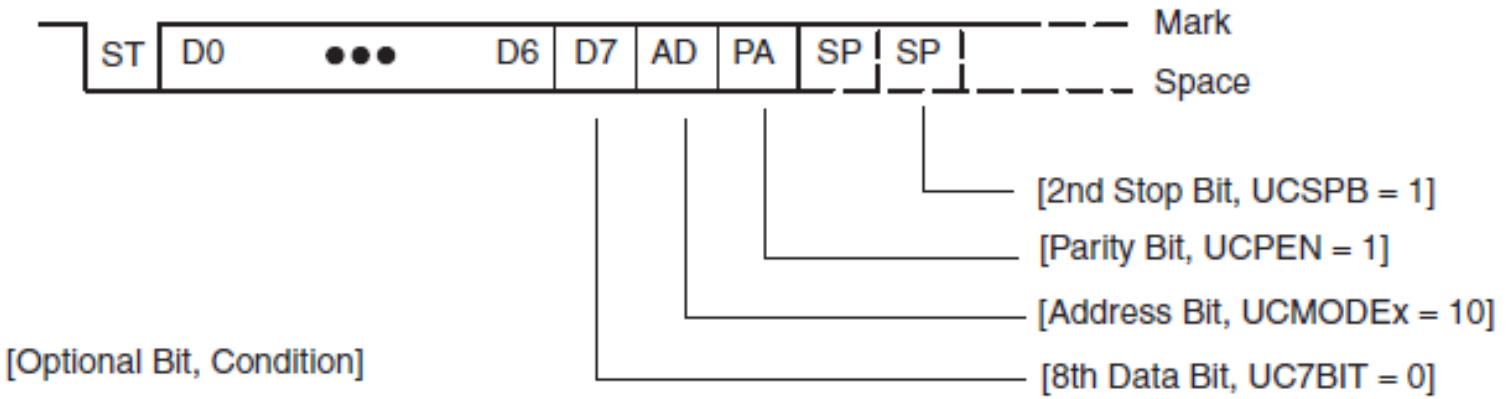
- In UART mode, the eUSCI_A transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected **baud rate** of the eUSCI_A.
- The transmit and receive functions use the same baud-rate frequency.

Initializing the eUSCI_A module

- The recommended eUSCI_A initialization / reconfiguration process is:
  - 1. Set UCSWRST
  - 2. Initialize all eUSCI_A registers with UCSWRST = 1 (including UCAxCTL1).
  - **3. Re - Configure ports.**
  - 4. Clear UCSWRST via software
  - 5. Enable interrupts (optional) via UCRXIE or UCTXIE.

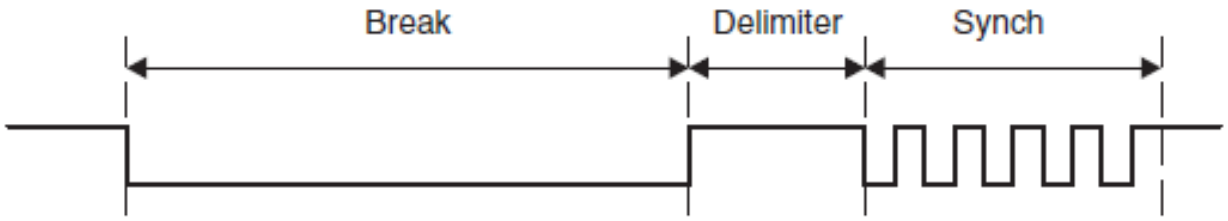
Character Format

- The UART character format consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB first is typically required for UART communication.



Automatic Baud-Rate Detection

- When **UCMODEx = 11**, UART mode with automatic baud-rate detection is selected.
- For automatic baudrate detection, a data frame is preceded by a synchronization sequence that consists of a break and a synch field. A break is detected when 11 or more continuous zeros (spaces) are received. If the length of the break exceeds 21 bit times, the break timeout error flag UCBTOE is set. The eUSCI_A cannot transmit data while receiving the break/sync field. The synch field follows the break



eUSCI_A Receive Enable

- The eUSCI_A module is enabled by clearing the **UCSWRST** bit and the receiver is ready and in an idle state. The receive baud rate generator is in a ready state but is not clocked nor producing any clocks.

- The **falling edge of the start bit enables** the baud rate generator and the UART state machine **checks for a valid start bit**. If no valid start bit is detected the UART state machine returns to its idle state and the baud rate generator is turned off again. If a valid start bit is detected, a character is received.

eUSCI_A Transmit Enable

- The eUSCI_A module is enabled by clearing the **UCSWRST** bit and the transmitter is ready and in an idle state. The transmit baud-rate generator is ready but is not clocked nor producing any clocks.

- A transmission is initiated by writing data to **UCAxTXBUF**. When this occurs, the baud-rate generator is enabled, and the data in **UCAxTXBUF** is moved to the transmit shift register on the next **BITCLK** after the transmit shift register is empty. **UCTXIFG** is set when new data can be written into **UCAxTXBUF**.

- Transmission continues as long as new data is available in **UCAxTXBUF** at the end of the previous byte transmission. If new data is not in **UCAxTXBUF** when the previous byte has transmitted, the transmitter returns to its idle state and the baud-rate generator is turned off.

UART Baud-Rate Generation

- The eUSCI_A baud-rate generator is capable of producing standard baud rates from nonstandard source frequencies. It provides two modes of operation selected by the UCOS16 bit.

- **Low-Frequency Baud-Rate Generation**

- **Oversampling Baud-Rate Generation**

Setting a Baud Rate

- For a given BRCLK clock source, the baud rate used determines the required division factor N:

$$\mathbf{N = f_{BRCLK} / Baudrate}$$

Typically N = SMCLK / Baud Rate

- The division factor N is often a non-integer value, thus, at least one divider and one modulator stage is used to meet the factor as closely as possible.

- If N is equal or greater than 16, it is recommended to use the oversampling baud-rate generation mode by setting UCOS16.

Baudrate settings quick set up

To calculate the correct the correct settings for the baudrate generation, perform these steps:

1. Calculate N = f_{BRCLK}/Baudrate

   [if N > 16 continue with step 3, otherwise with step 2]

2. OS16 = 0, UCBRx = INT(N) [continue with step 4]

3. OS16 = 1, UCBRx = INT(N/16),

   UCBRFx = INT([(N/16) – INT(N/16)] × 16)

4. UCBRSx can be found by looking up the fractional part of N

   (= N - INT(N)) in the table

5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed

eUSCI_A Interrupts

- The eUSCI_A has only one interrupt vector that is shared for transmission and for reception.

```
//-------------------------------------------------------------------------------
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void){
  unsigned int temp;
  switch(__even_in_range(UCA0IV,0x08)){
    case 0:                                 // Vector 0 - no interrupt

      break;
    case 2:                                 // Vector 2 - RXIFG
// code for Receive
      break;
    case 4:                                 // Vector 4 - TXIFG
// Code for Transmit
      break;
  default: break;
  }
}
//-------------------------------------------------------------------------------
```

eUSCI_A Transmit Interrupt Operation

- The **UCTXIFG** interrupt flag is set by the transmitter to indicate that **UCAxTXBUF** is ready to accept another character. An interrupt request is generated if **UCTXIE** and **GIE** are also set.
- **UCTXIFG** is automatically reset if a character is written to **UCAxTXBUF**.

- **UCTXIFG** is set after a PUC or when **UCSWRST** = 1.
- **UCTXIE** is reset after a PUC or when **UCSWRST** = 1.

eUSCI_A Receive Interrupt Operation

- The **UCRXIFG** interrupt flag is set each time a character is received and loaded into **UCAxRXBUF**. An interrupt request is generated if **UCRXIE** and **GIE** are also set.
- **UCRXIFG** and **UCRXIE** are reset by a system reset PUC signal or when **UCSWRST** = 1.
- **UCRXIFG** is automatically reset when **UCAxRXBUF** is read.

- Additional interrupt control features include:
  - When UCAxRXEIE = 0, erroneous characters do not set UCRXIFG.
  - When UCDORM = 1, non-address characters do not set UCRXIFG in multiprocessor modes. In plain UART mode, no characters are set UCRXIFG.
  - When UCBRKIE = 1, a break condition sets the UCBRK bit and the UCRXIFG flag.

UCAxIV, Interrupt Vector Generator

- The eUSCI_A interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register **UCAxIV** is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the **UCAxIV** register that can be evaluated or added to the program counter to automatically enter the appropriate software routine.

- Disabled interrupts do not affect the **UCAxIV** value.

- Read access of the **UCAxIV** register automatically resets the highest-pending Interrupt condition and flag.

- Write access of the **UCAxIV** register clears all pending Interrupt conditions and flags. If another interrupt flag is set, another interrupt is generated immediately after servicing the initial interrupt.

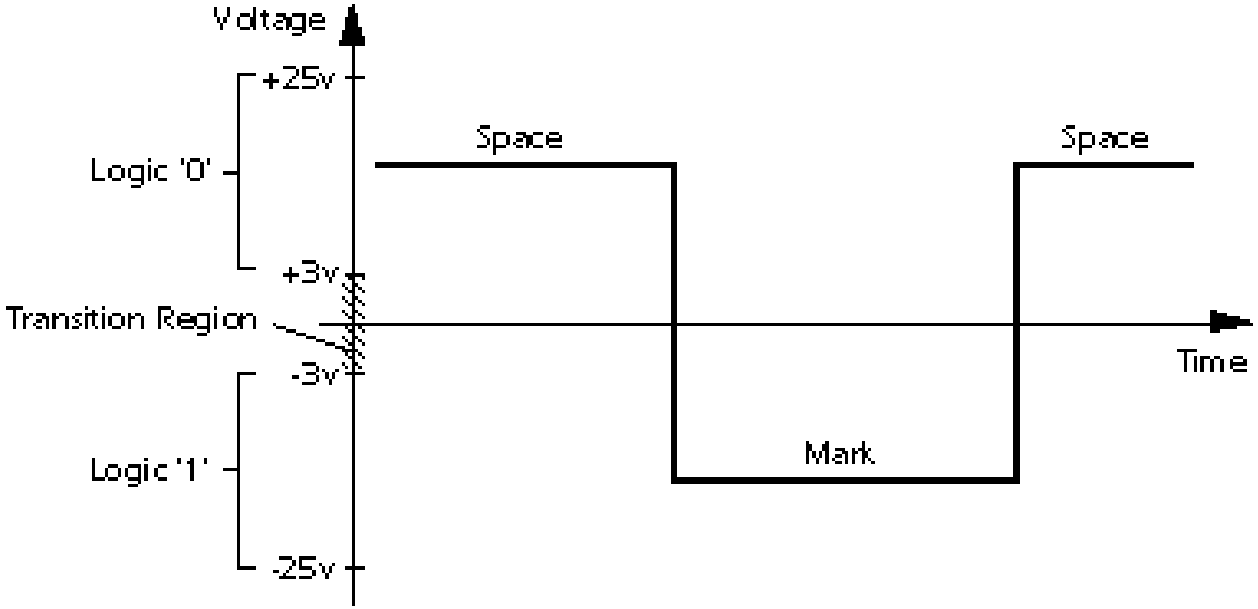
Bit Rate vs. Baud Rate

- Bit Rate: how many ***data bits*** are transmitted per second?
- Baud Rate: how many ***symbols*** are transmitted per second?
  - How many times does the communication channel change state per second?
  - A symbol may be represented by a voltage level, a sine wave's frequency or phase, etc.

- These will be different
  - Extra symbols (channel changes) may be inserted for framing, Start bits, stop bits, error detection, acknowledgment, etc. These ***reduce*** the bit rate

RS232 Information

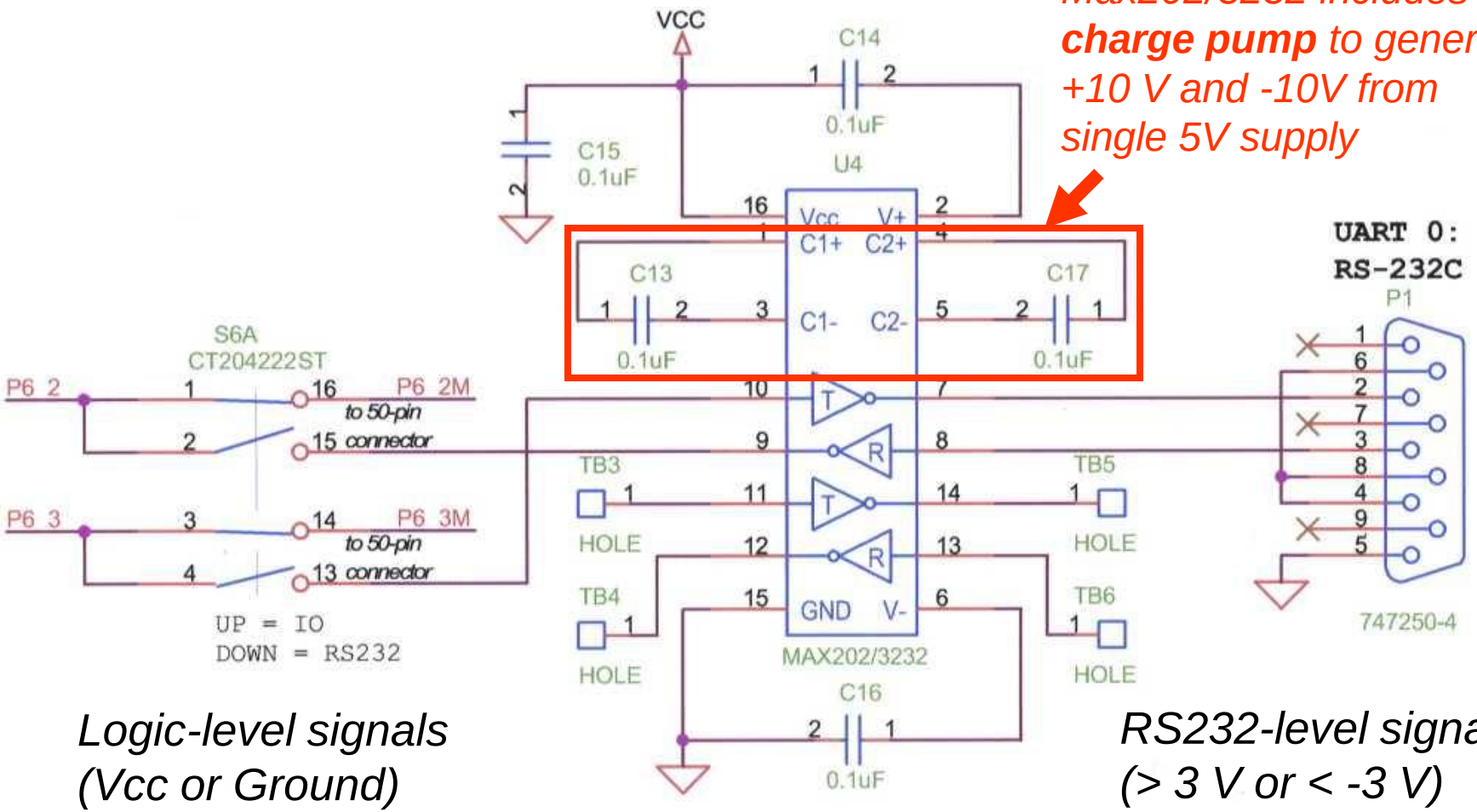
- RS232: rules on connector, signals/pins, voltage levels, handshaking, etc.
- RS232: Fulfilling All Your Communication Needs, Robert Ashby
- Quick Reference for RS485, RS422, RS232 and RS423
- Not so quick reference:
  The RS232 Standard: A Tutorial with Signal Names and Definitions,
  Christopher E. Strangio
- Bit vs Baud rates:
  http://www.totse.com/en/technology/telecommunications/bits.html



RS232 Communications Circuit

- Example RS-232 buffer (level-shifting) circuit
  - Not included on MSP430

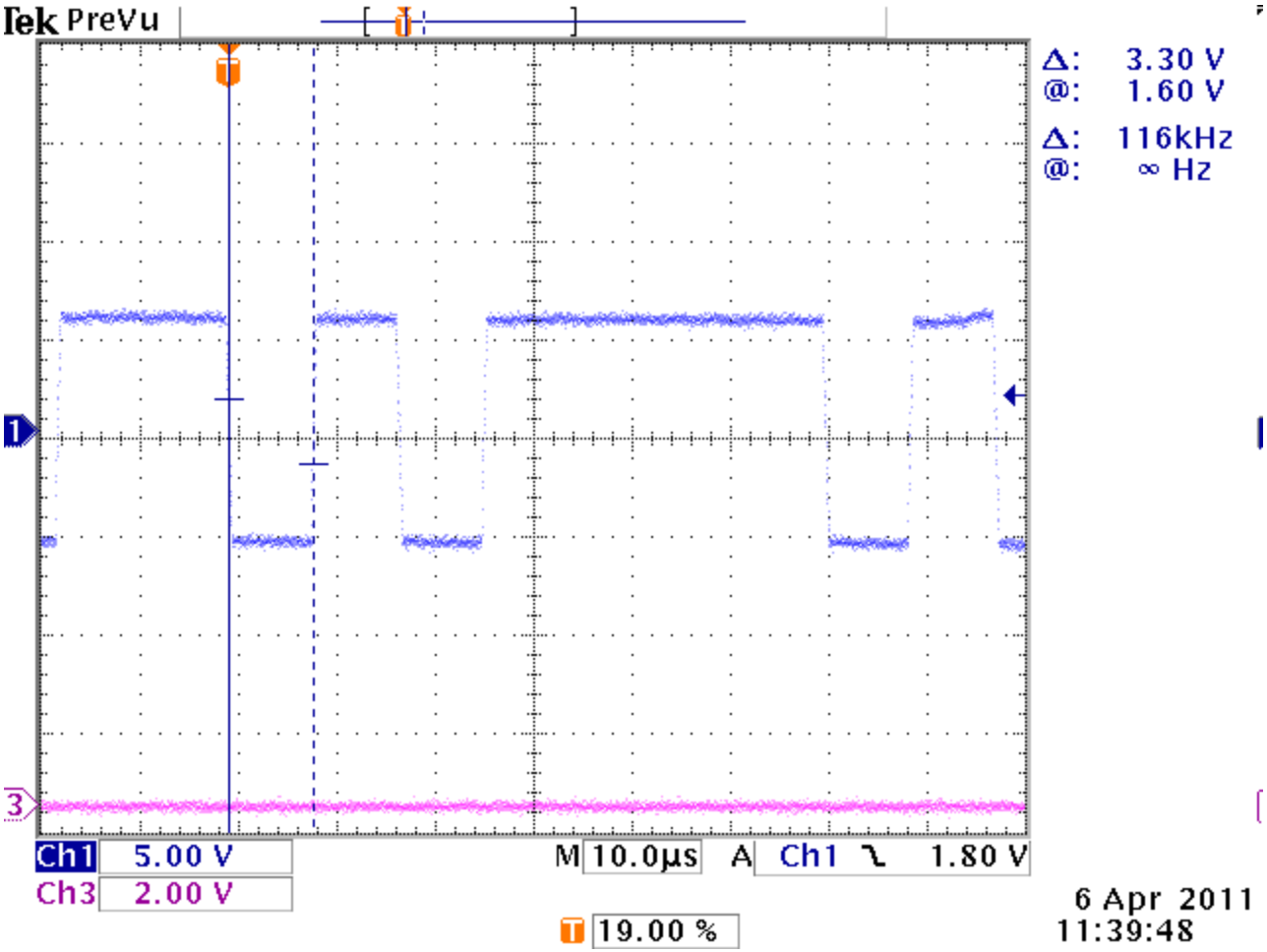
*Max202/3232 includes **charge pump** to generate +10 V and -10V from single 5V supply*



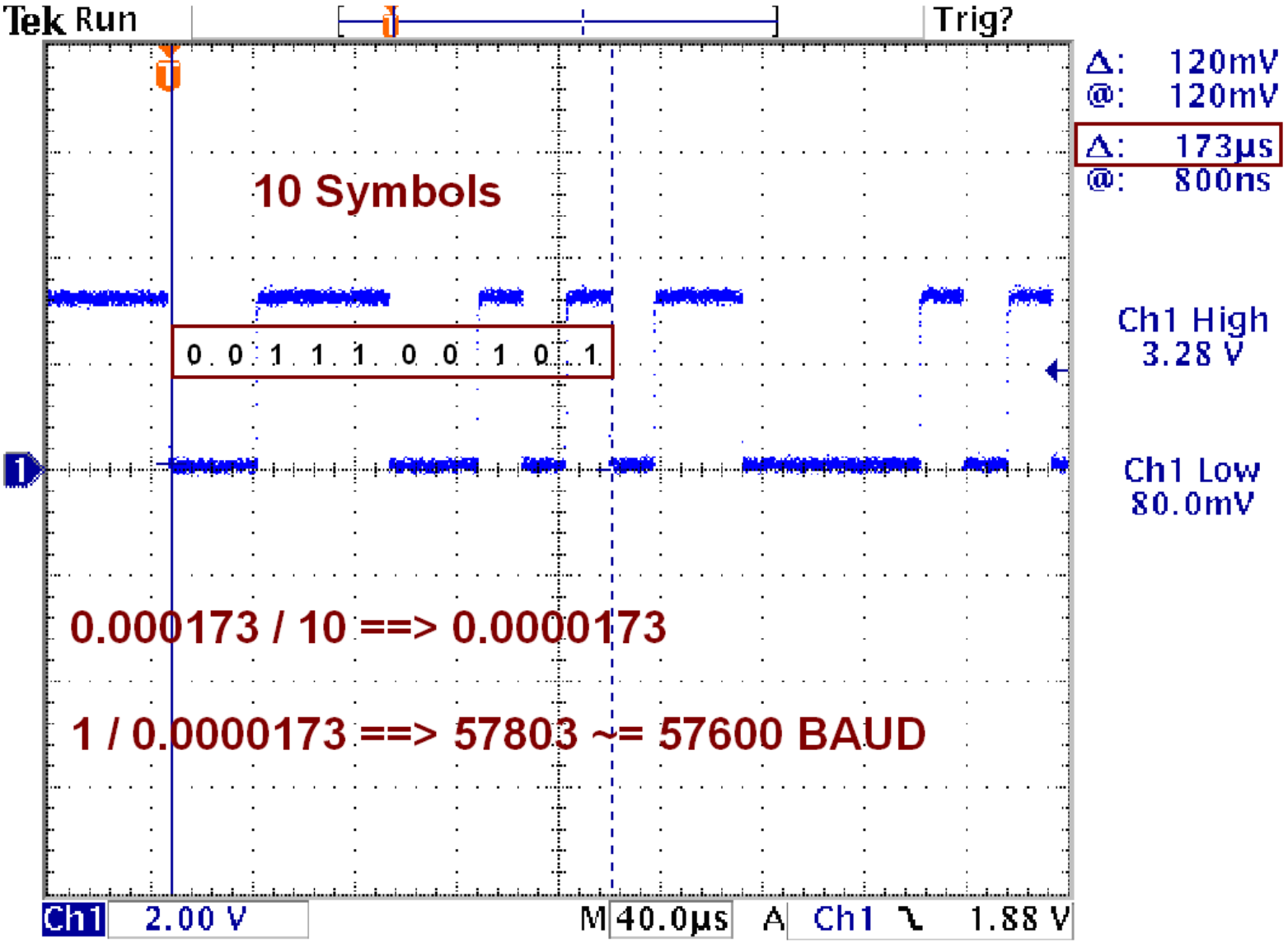
Logic-level signals (Vcc or Ground)

RS232-level signals (> 3 V or < -3 V)

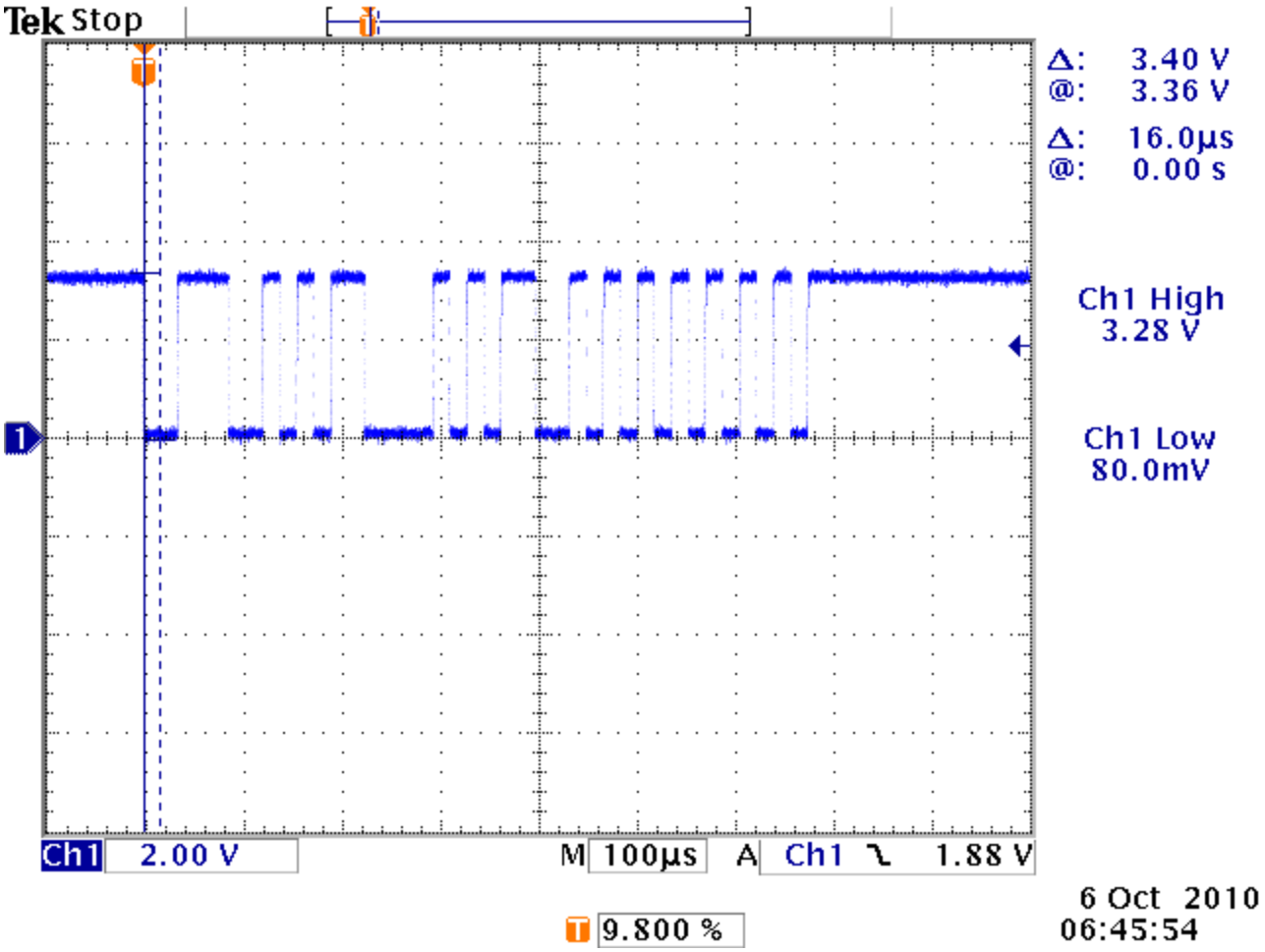
RS-232 Snapshot timing of a Symbol



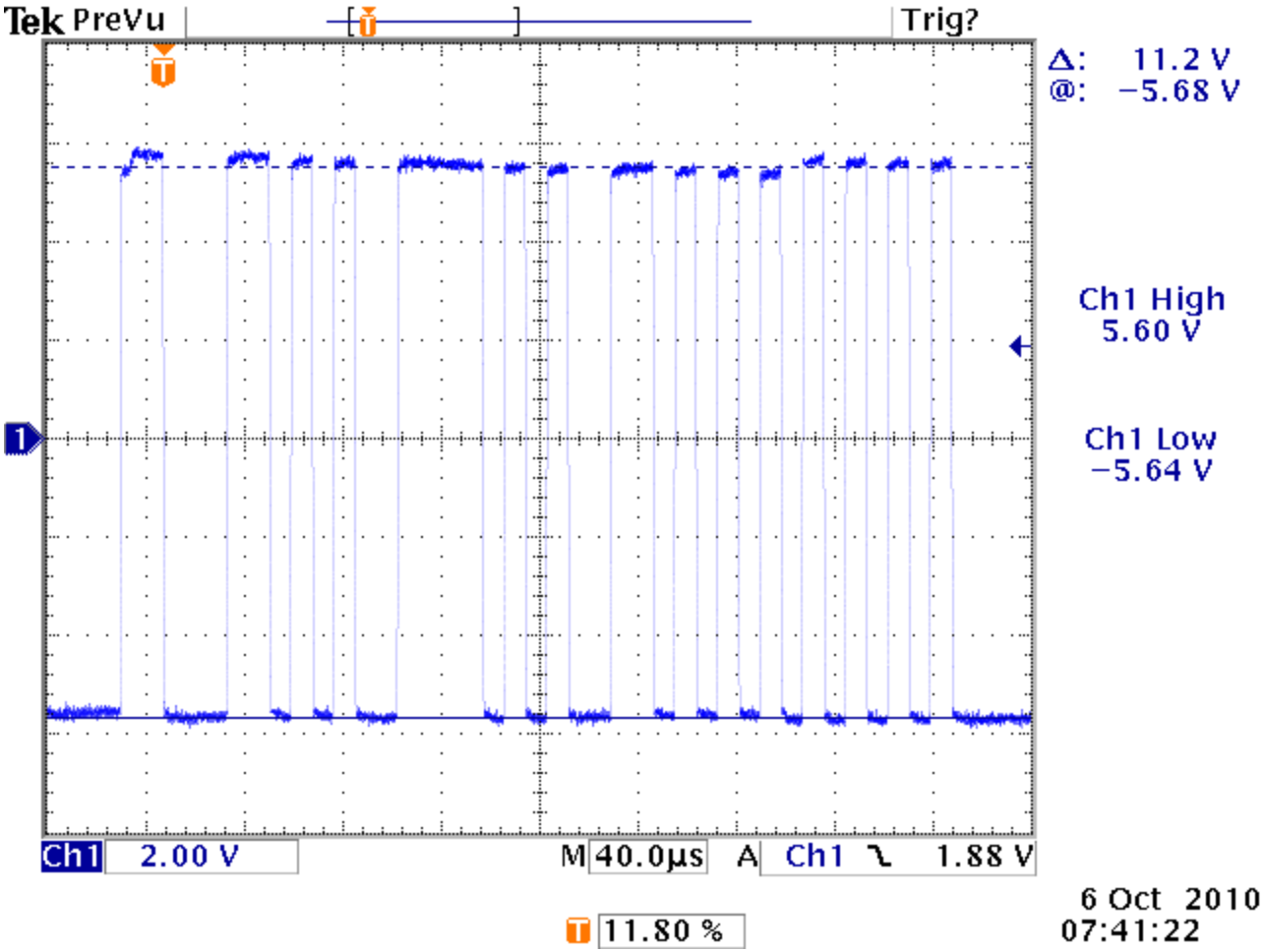
Baud Rate Evaluation



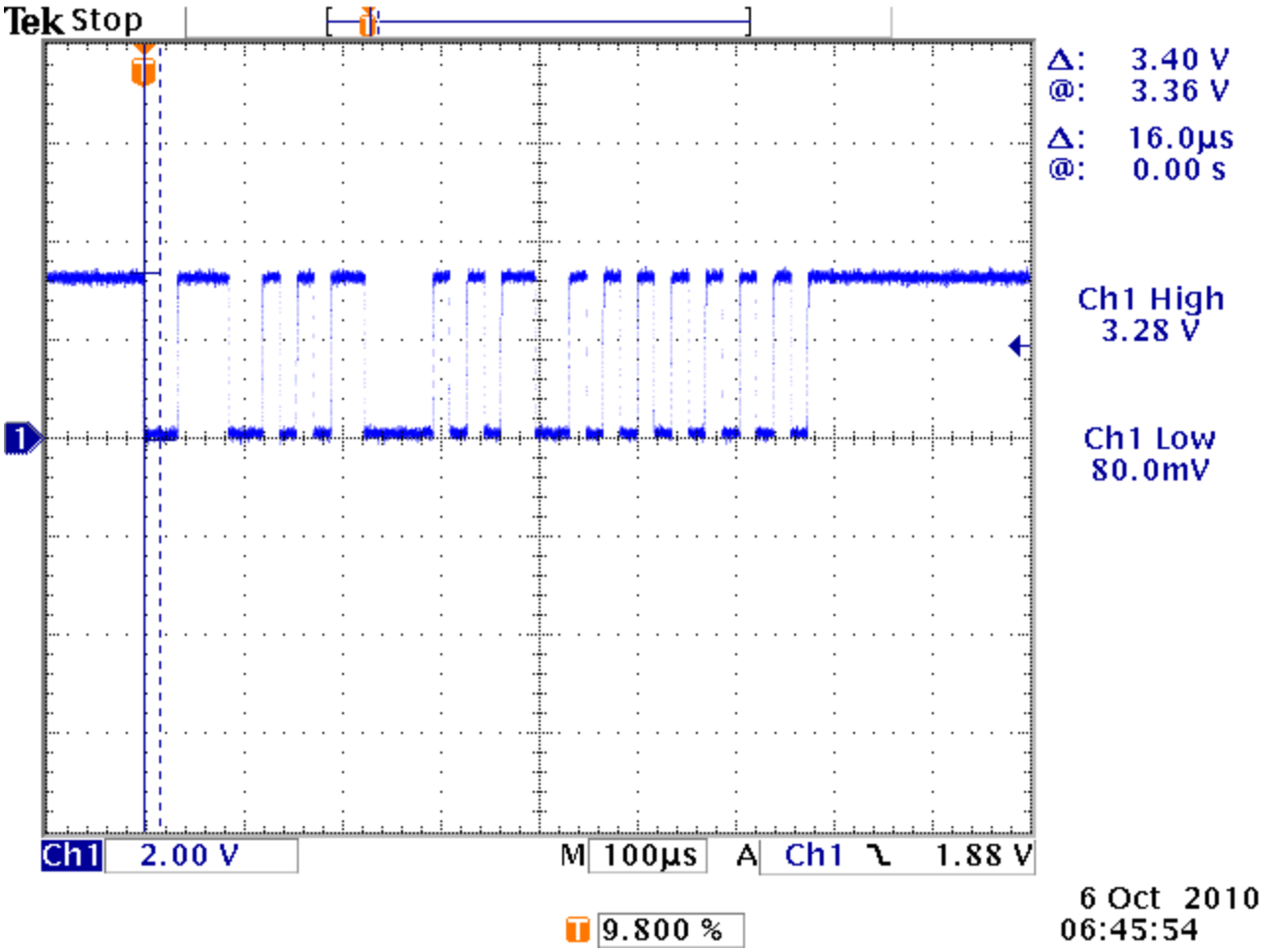
Digital Domain



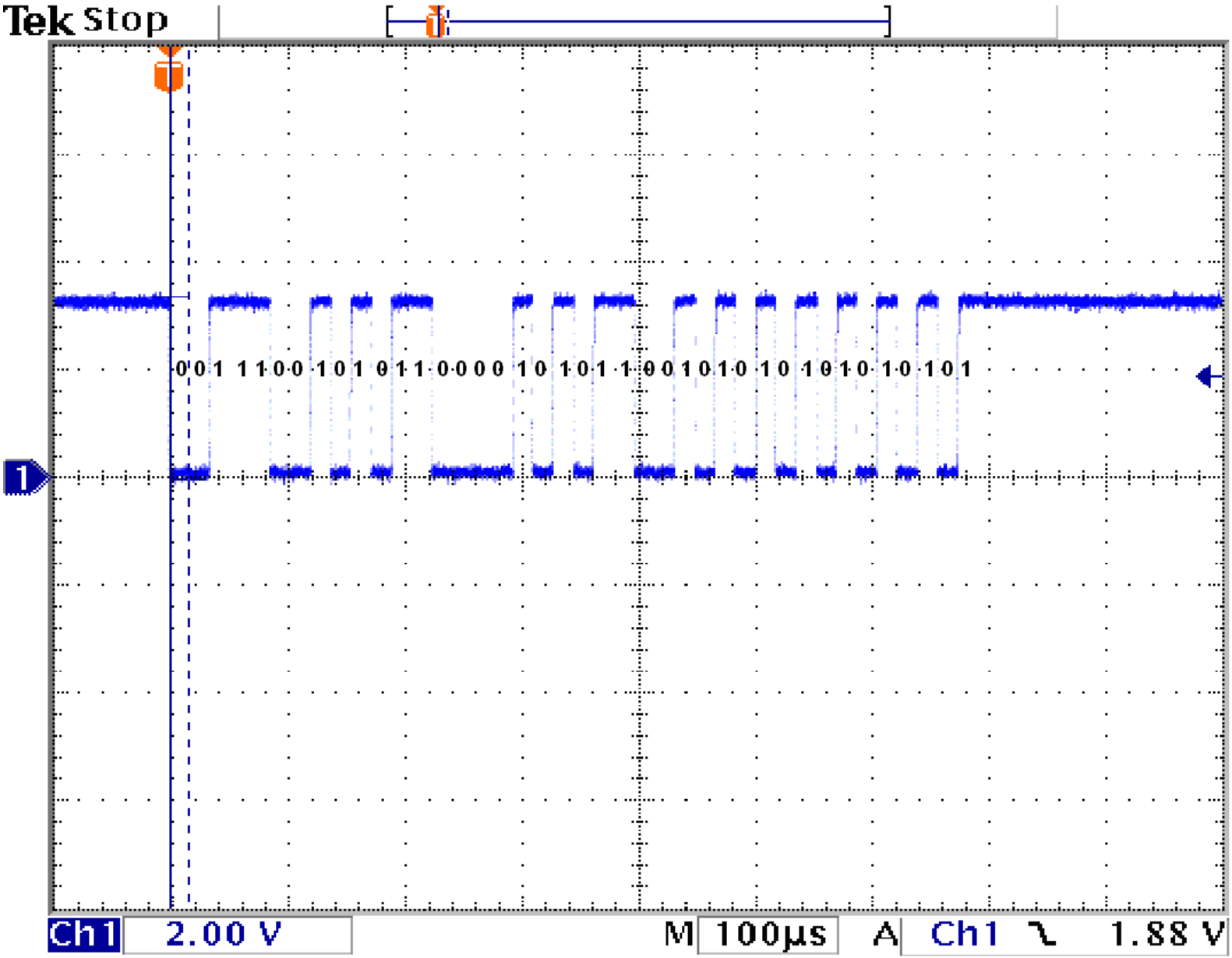
Analog Domain [DB-9 RS-232]



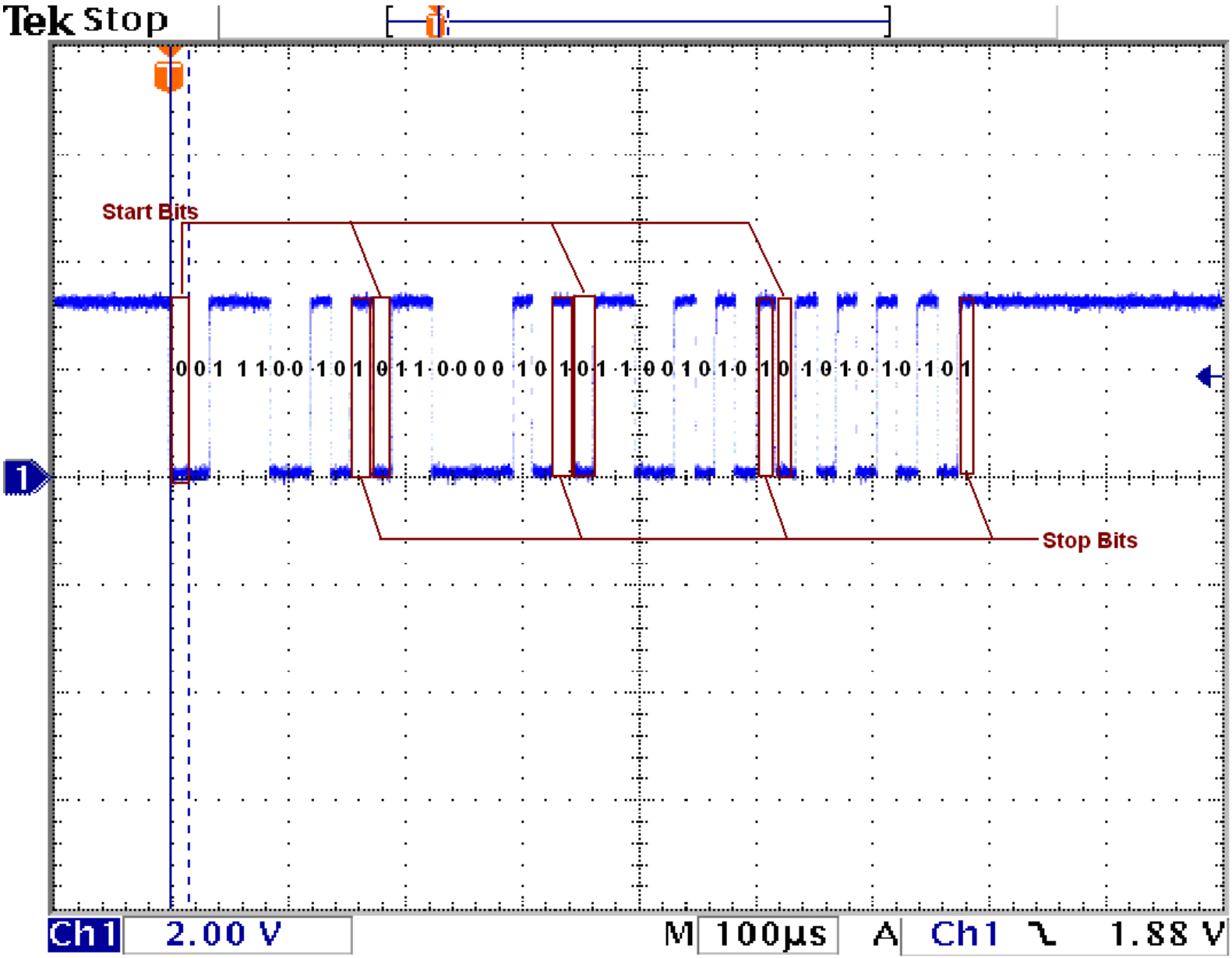
What was Transmitted



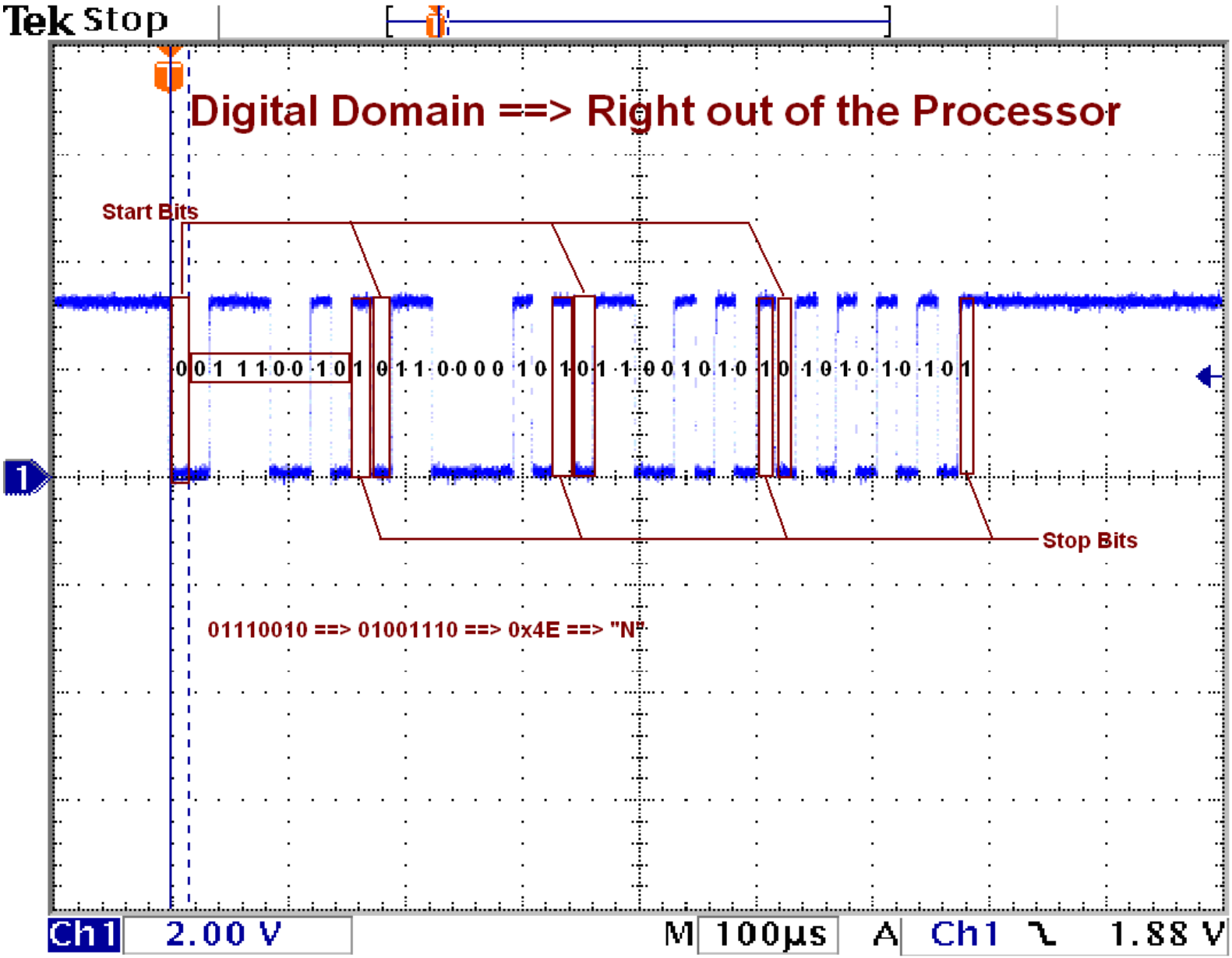
Determine Bits



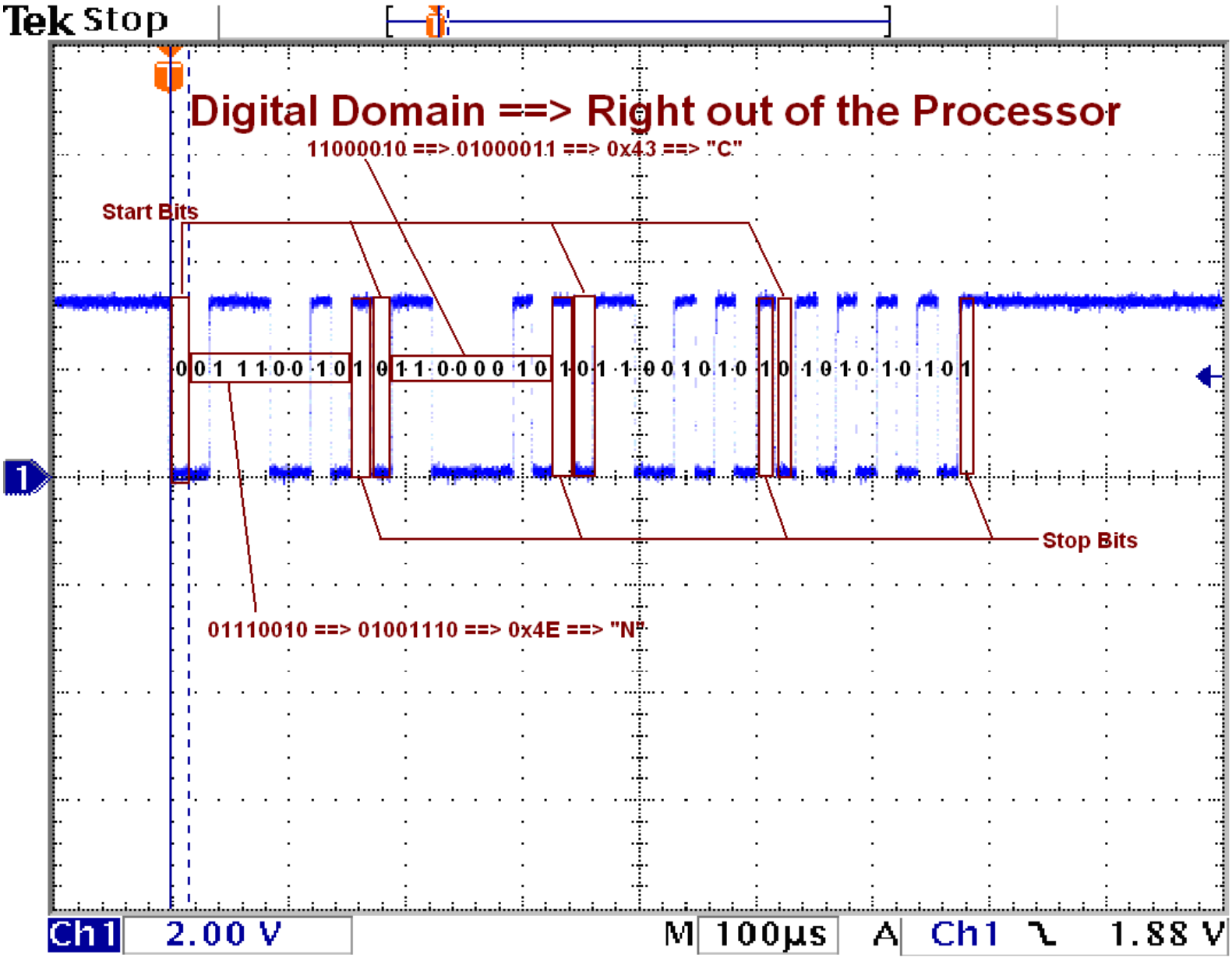
Identify Start / Stop Bits



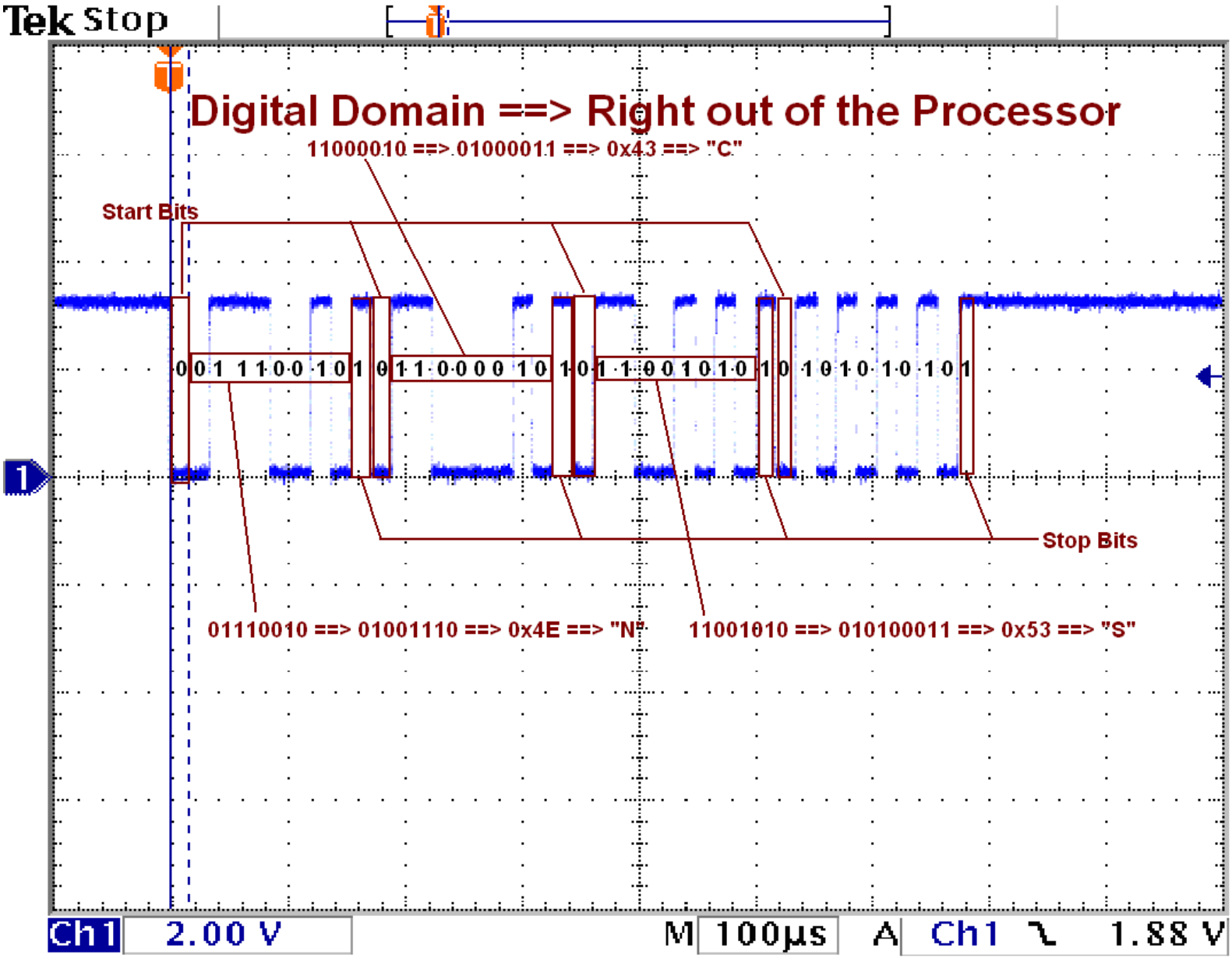
Determine Data Convert to ASCI



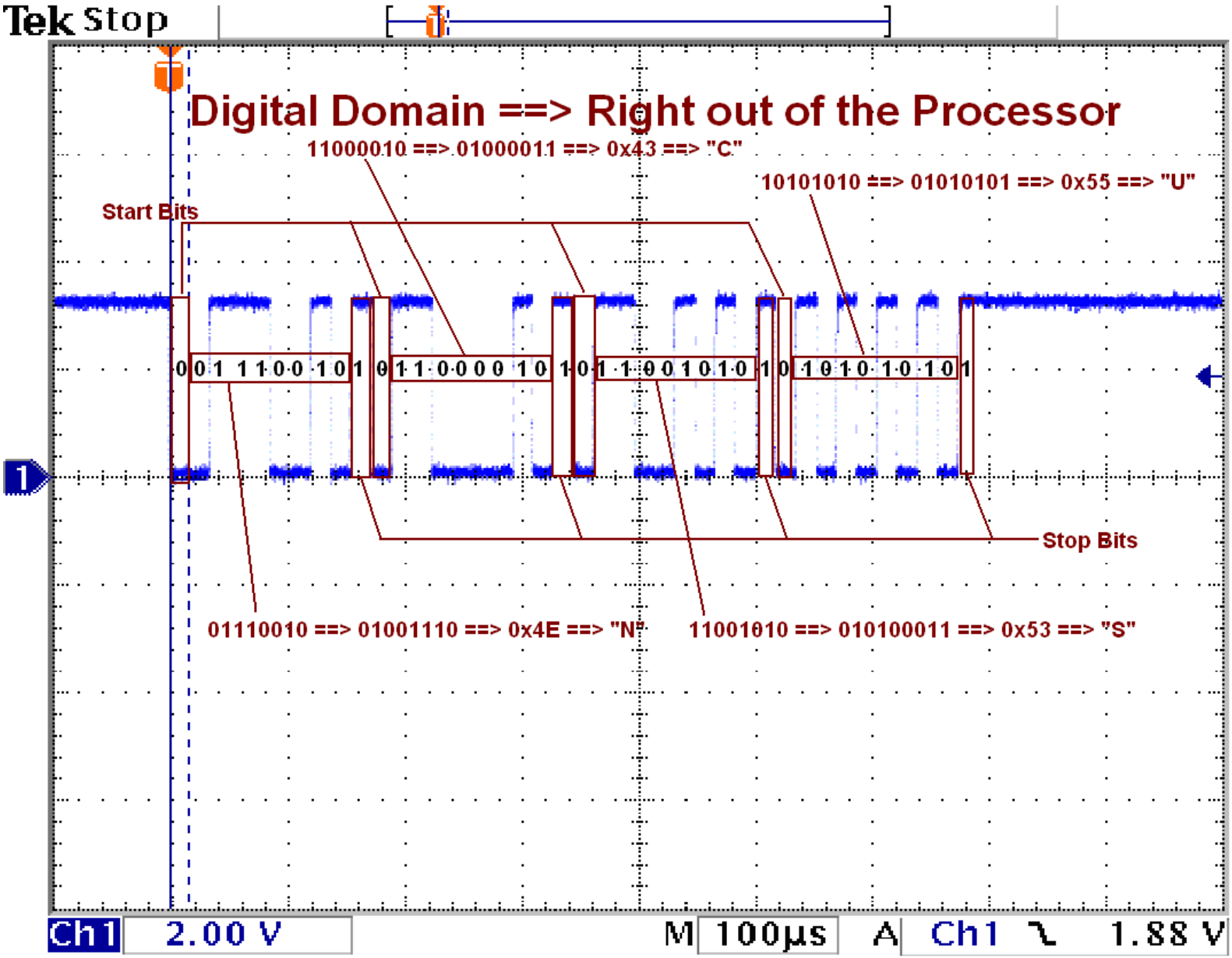
Determine Data Convert to ASCI



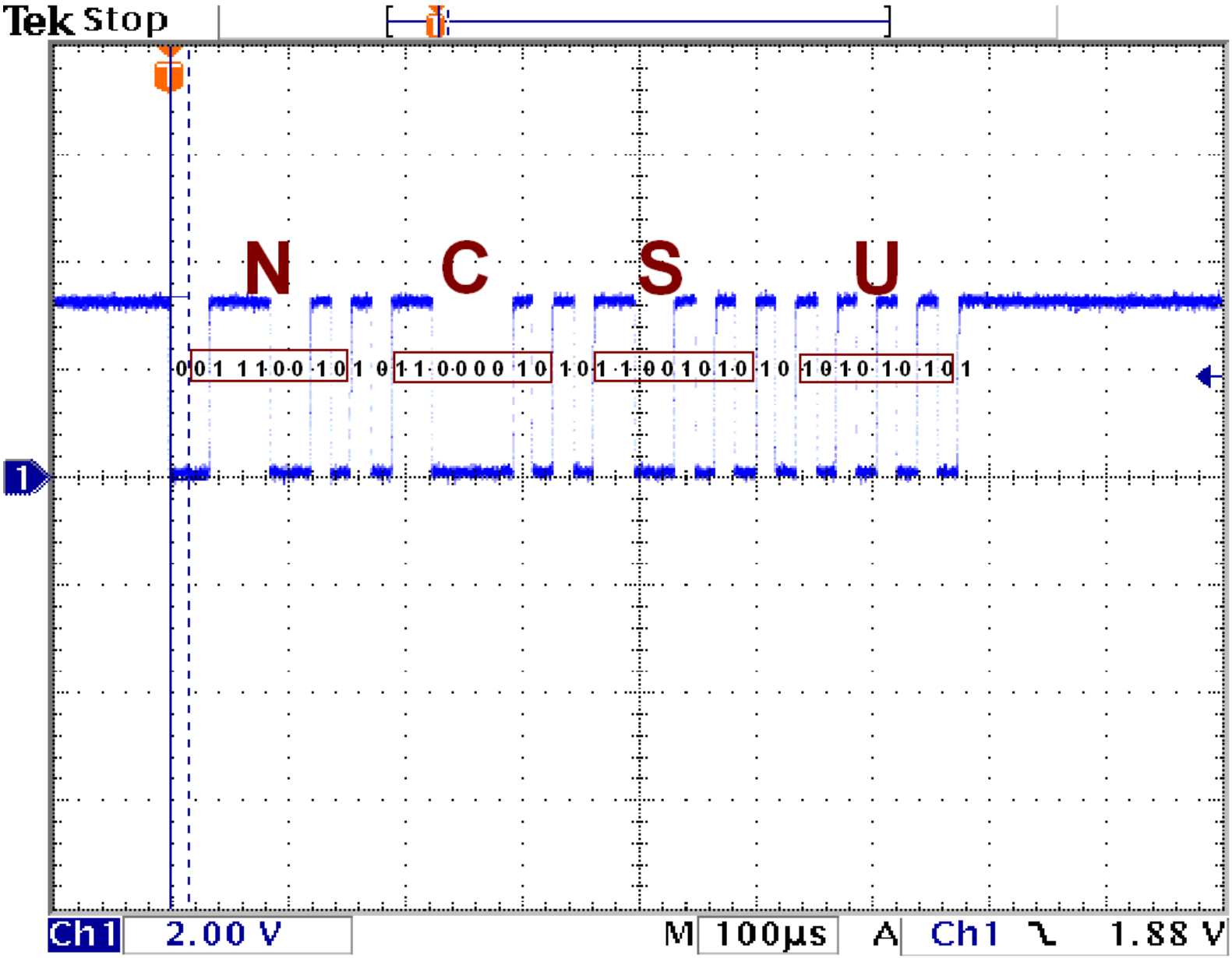
Determine Data Convert to ASCI



Determine Data Convert to ASCI



Process Data



Software Example

In Main -

```
 Init_Serial_UCA0();       // Initialize Serial Port for USB
// #defines
#define BEGINNING            (0)
#define SMALL_RING_SIZE     (16)
//---------------------------------------------------------------------------
void Init_Serial_UCA0(void){
  int i;
  for(i=0; i<SMALL_RING_SIZE; i++){
    USB_Char_Rx[i] = 0x00;                  // USB Rx Buffer
  }
  usb_rx_ring_wr = BEGINNING;
  usb_rx_ring_rd = BEGINNING;

  for(i=0; i<LARGE_RING_SIZE; i++){         // May not use this
    USB_Char_Tx[i] = 0x00;                  // USB Tx Buffer
  }
  usb_tx_ring_wr = BEGINNING;
  usb_tx_ring_rd = BEGINNING;

  // Configure UART 0
  UCA0CTLW0 = 0;                            // Use word register
  UCA0CTLW0 |= UCSSEL__SMCLK;               // Set SMCLK as fBRCLK
  UCA0CTLW0 |= UCSWRST;                     // Set Software reset enable
//***** Continued on next slide
```

Software Example

```
// ***** Continuation from previous slide
// 9,600 Baud Rate
// 1. Calculate N = fBRCLK / Baudrate
//     N = SMCLK / 9,600 => 8,000,000 / 9,600 = 833.3333333
//  if N > 16 continue with step 3, otherwise with step 2
// 2. OS16 = 0, UCBRx = INT(N)
//  continue with step 4
// 3. OS16 = 1
//        UCx       = INT(N/16),
//                  = INT(N/16) = 833.333/16 => 52
//        UCFx      = INT([(N/16) – INT(N/16)] × 16)
//                  = ([833.333/16-INT(833.333/16)]*16)
//                  = (52.08333333-52)*16
//                  = 0.083*16 = 1
// 4. UCSx is found by looking up the fractional part of N (= N-INT(N)) in table Table 18-4
//     Decimal of SMCLK / 8,000,000 / 9,600 = 833.3333333 => 0.333 yields 0x49 [Table]
// 5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed
//                                                  TX error (%) RX error (%)
//  BRCLK   Baudrate UCOS16  UCBRx  UCFx    UCSx     neg   pos  neg  pos
//  8000000    9600     1     52      1     0x49   -0.08 0.04 -0.10 0.14
 UCA0BRW = 52;                            // 9,600 Baud
//  UCA0MCTLW = UCSx concatenate UCFx concatenate UCOS16;
//  UCA0MCTLW = 0x49 concatenate 1 concatenate 1;
  UCA0MCTLW = 0x4911 ;
  UCA0CTL1 &= ~UCSWRST;                    // Release from reset
  UCA0IE |= UCRXIE;                        // Enable RX interrupt
}
```

Table 18-4. UCBRSx Settings for Fractional Portion of N = fBRCLK/Baudrate

Fractional Portion of N	UCBRSx(1)	Fractional Portion of N	UCBRSx(1)
0.0000	0x00	0.5002	0xAA
0.0529	0x01	0.5715	0x6B
0.0715	0x02	0.6003	0xAD
0.0835	0x04	0.6254	0xB5
0.1001	0x08	0.6432	0xB6
0.1252	0x10	0.6667	0xD6
0.1430	0x20	0.7001	0xB7
0.1670	0x11	0.7147	0xBB
0.2147	0x21	0.7503	0xDD
0.2224	0x22	0.7861	0xED
0.2503	0x44	0.8004	0xEE
0.3000	0x25	0.8333	0xBF
0.3335	**0x49**	0.8464	0xDF
0.3575	0x4A	0.8572	0xEF
0.3753	0x52	0.8751	0xF7
0.4003	0x92	0.9004	0xFB
0.4286	0x53	0.9170	0xFD
0.4378	0x55	0.9288	0xFE

(1) The UCBRSx setting in one row is valid from the fractional portion given in that row until the one in the next row

Software Example

```
// #defines
#define BEGINNING            (0)
#define SMALL_RING_SIZE     (16)

// global variables
volatile unsigned int usb_rx_ring_wr;
volatile char USB_Char_Rx[SMALL_RING_SIZE] ;
//-------------------------------------------------------------------------
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void){
  unsigned int temp;
  switch(__even_in_range(UCA0IV,0x08)){
    case 0:                                   // Vector 0 - no interrupt
      break;
    case 2:                                   // Vector 2 - RXIFG
      temp = usb_rx_ring_wr;
      USB_Char_Rx[temp] = UCA0RXBUF;          // RX -> USB_Char_Rx character
      if (++usb_rx_ring_wr >= (SMALL_RING_SIZE)){
        usb_rx_ring_wr = BEGINNING;           // Circular buffer back to beginning
      }
      break;
    case 4:                                   // Vector 4 - TXIFG
      break;
  default: break;
  }
}
//-------------------------------------------------------------------------
```